Deep Learning Part II: Graph Neural Networks

Maurizio Pierini

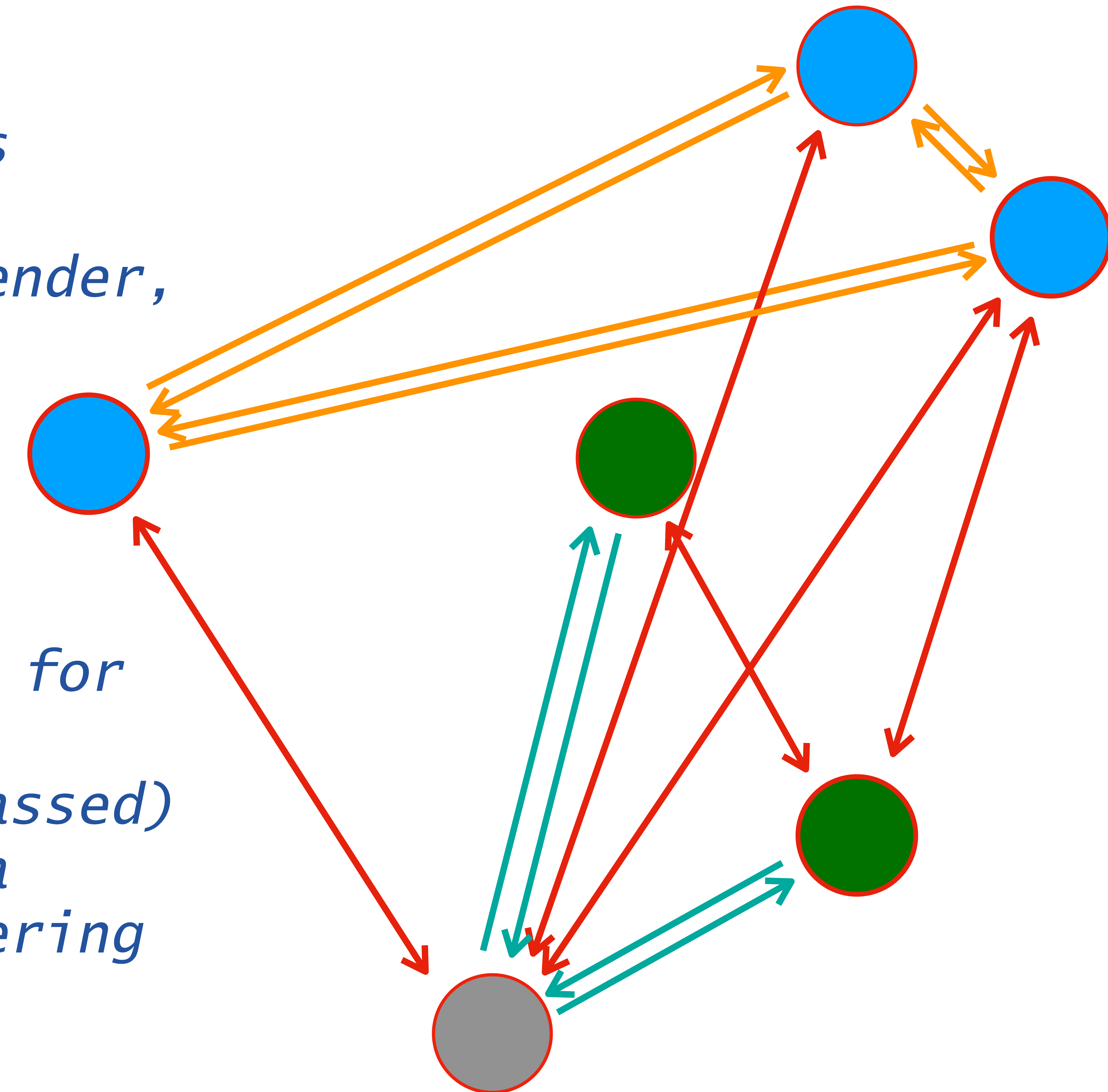
CERN

Message Passing

European Research Council

Learning from Graph: an example

- *Imagine a concrete example: given a social-media user, who will she vote for at the next elections?*
- *The graph here comes from social-media connections*
- *The features are what we know for a given user (gender, age, education, etc.)*
- *We want to gather information on someone from the social network of that person*
  - *we might know who some of her connections voted for*
- *We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features*
- *A final classifier will give us the answer we want*
- *You might become president with this + target pressure (ads, fake news, etc.)*

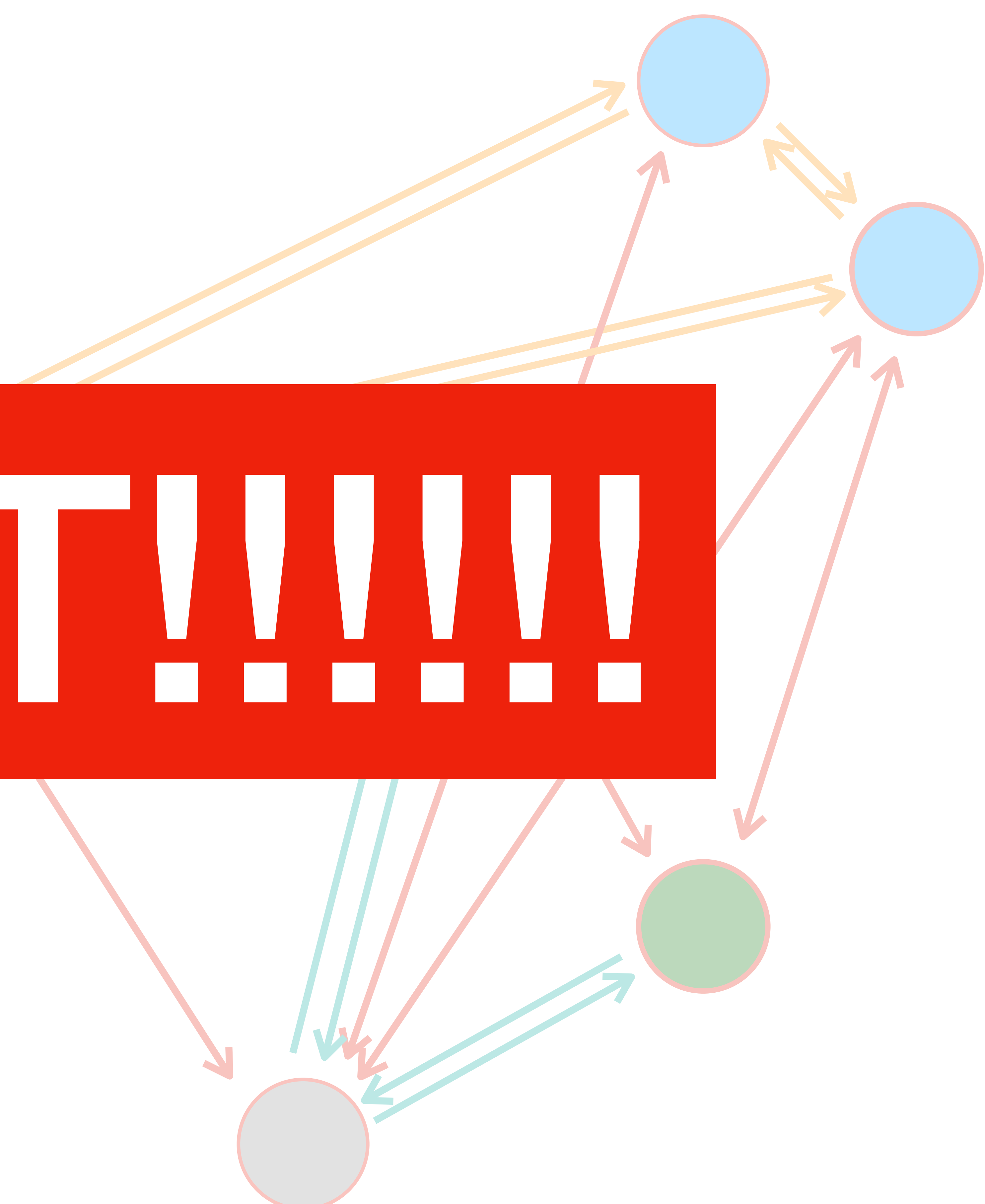


European Research Council

Learning from Graph: an example

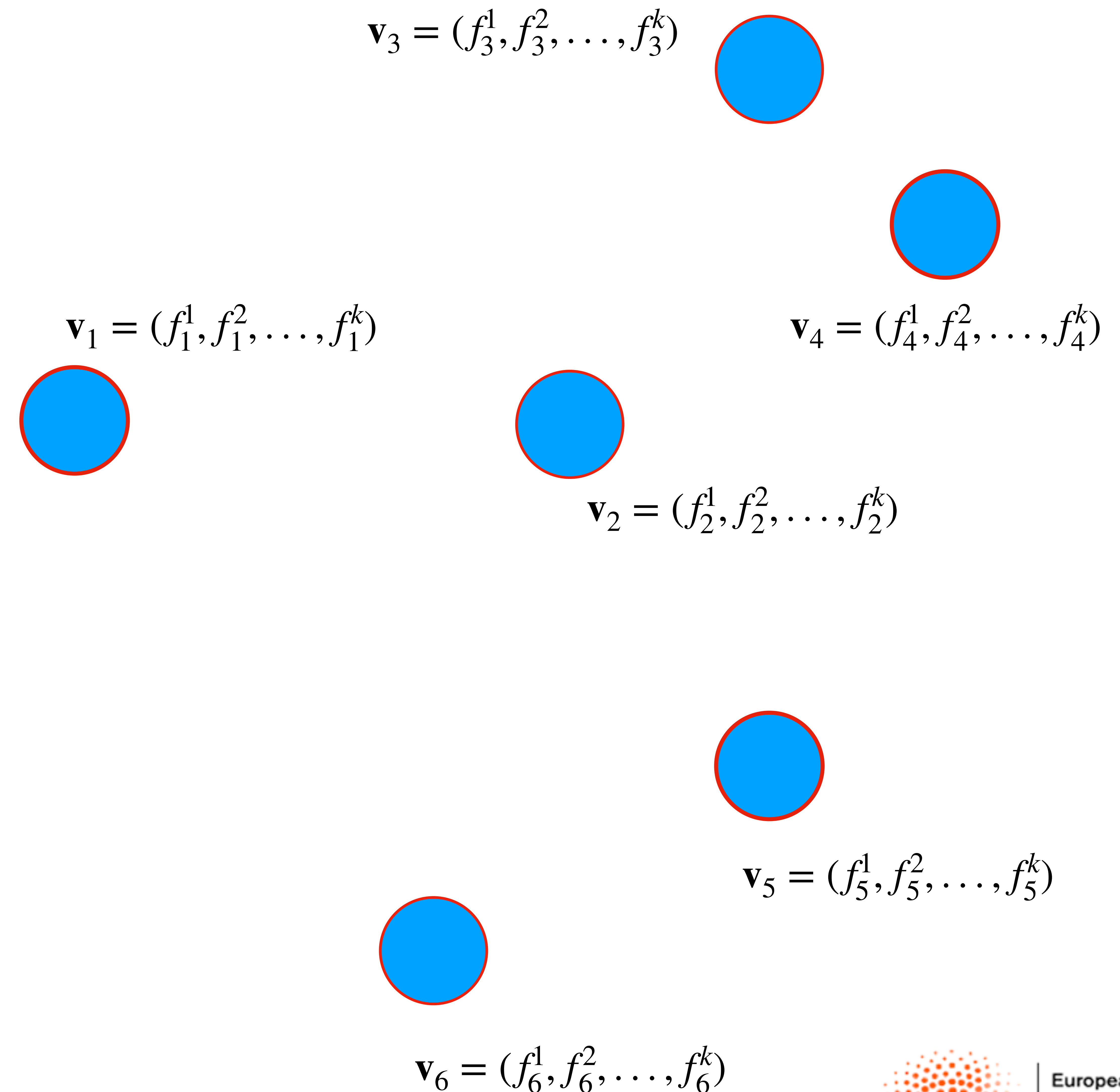
- Imagine a concrete example: given a social-media user, who will she vote for at the next elections?
- The graph here comes from social-media connections
- The features are what we know for a given user (gender, age, education, etc.)
- We w… soci…
  - we…
- We will use NNs to model the influence (message passed) of each user on her connection and learn from data which are the relevant connections. We are engineering features
- A final classifier will give us the answer we want
- You might become president with this + target pressure (ads, fake news, etc.)

DON'T DO IT!!!!!!



Graph Networks

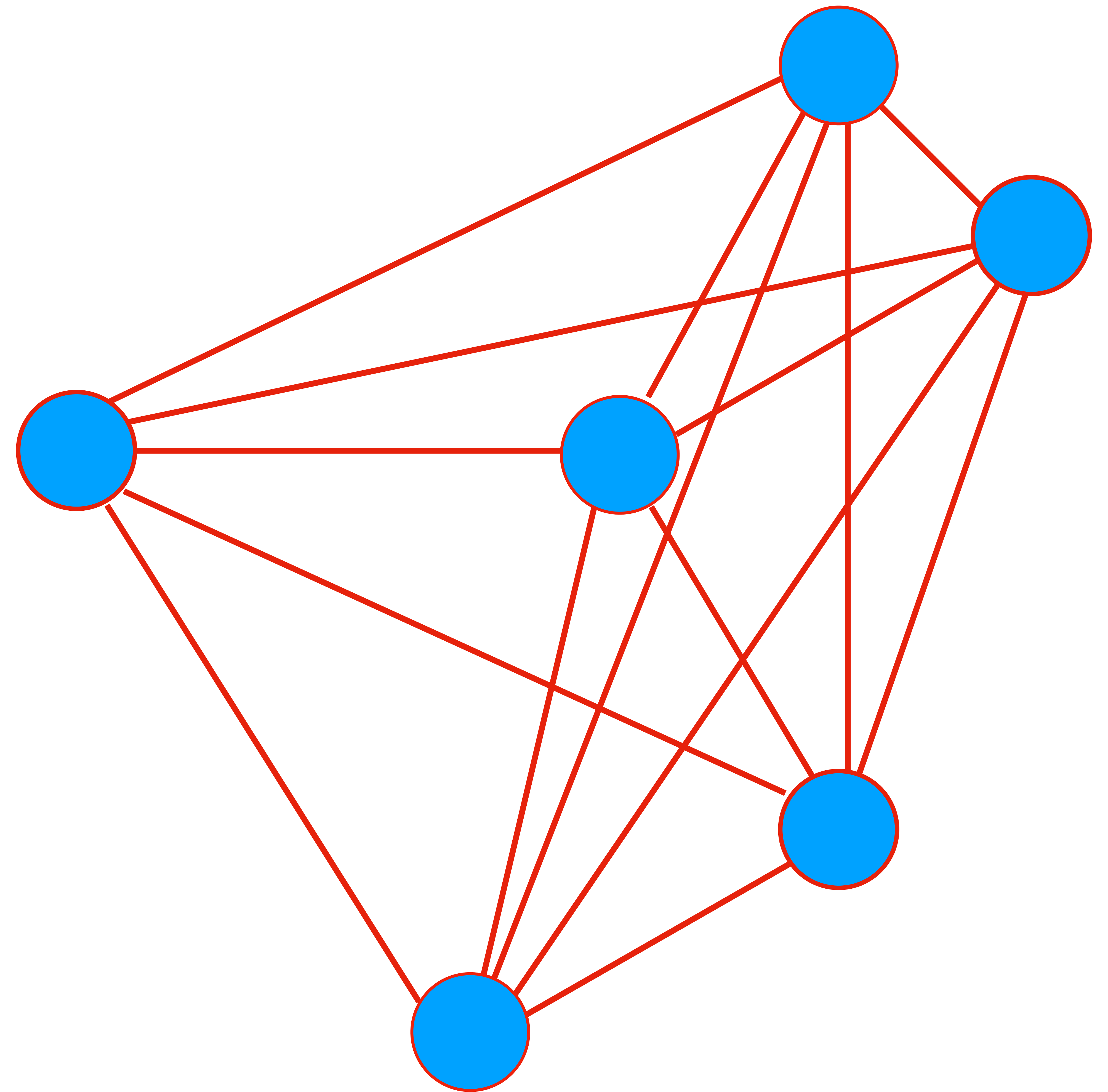
- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - ***Each example in a dataset is represented as a set of vertices***
  - ***Each vertex is embedded in the graph as a vector of features***



https://arxiv.org/pdf/1704.01212.pdf

Graph Networks

- *Graphs Nets are architectures based on an abstract representation of a given dataset*
    - *Each example in a dataset is represented as a set of vertices*
    - *Each vertex is embedded in the graph as a vector of features*
    - ***Vertices are connected through links (edges)***

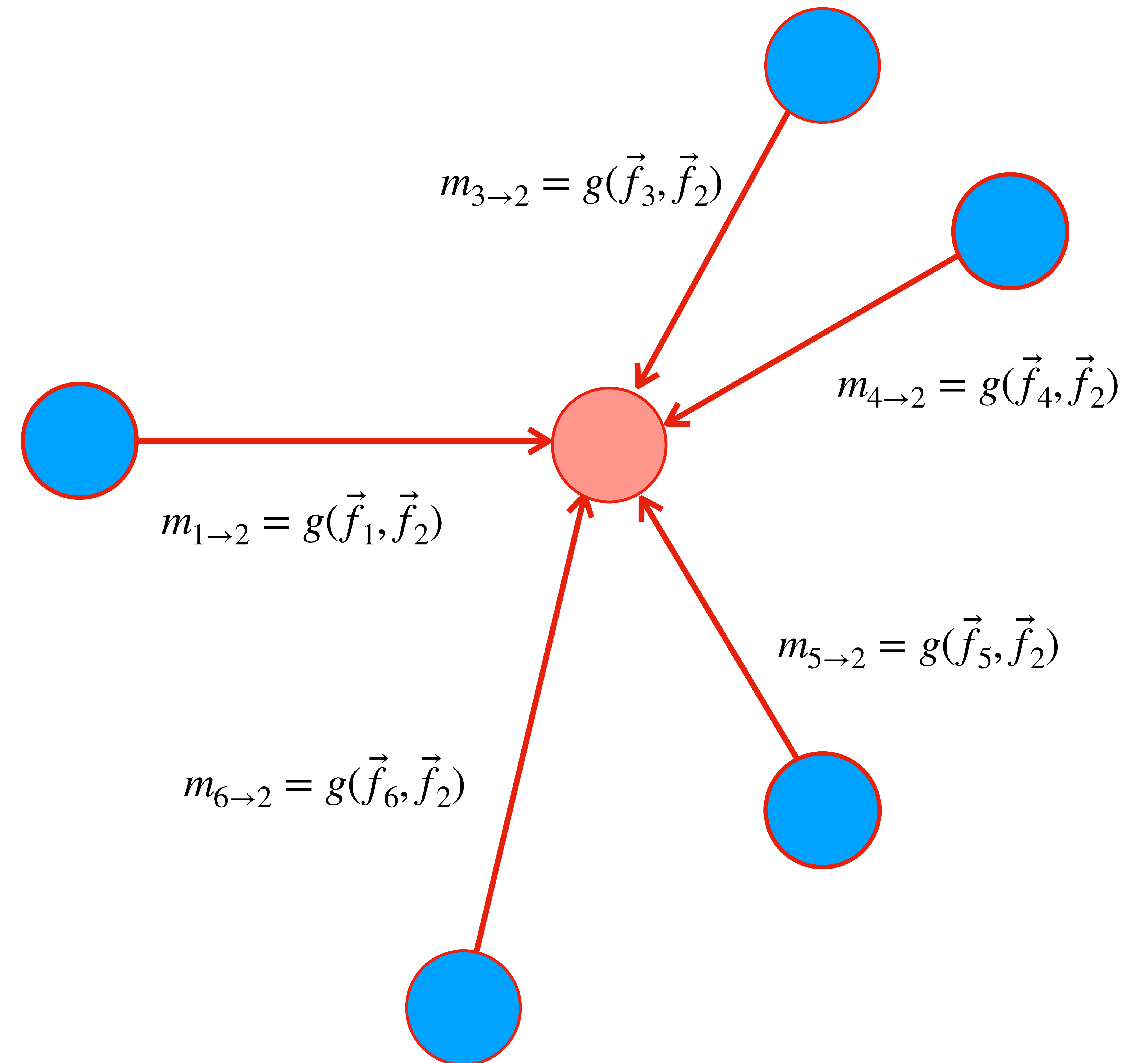


https://arxiv.org/pdf/1704.01212.pdf

European Research Council

Graph Networks

- *Graphs Nets are architectures based on an abstract representation of a given dataset*
  - *Each example in a dataset is represented as a set of vertices*
  - *Each vertex is embedded in the graph as a vector of features*
  - *Vertices are connected through links (edges)*
  - ***Messages are passed through links and aggregated on the vertices***

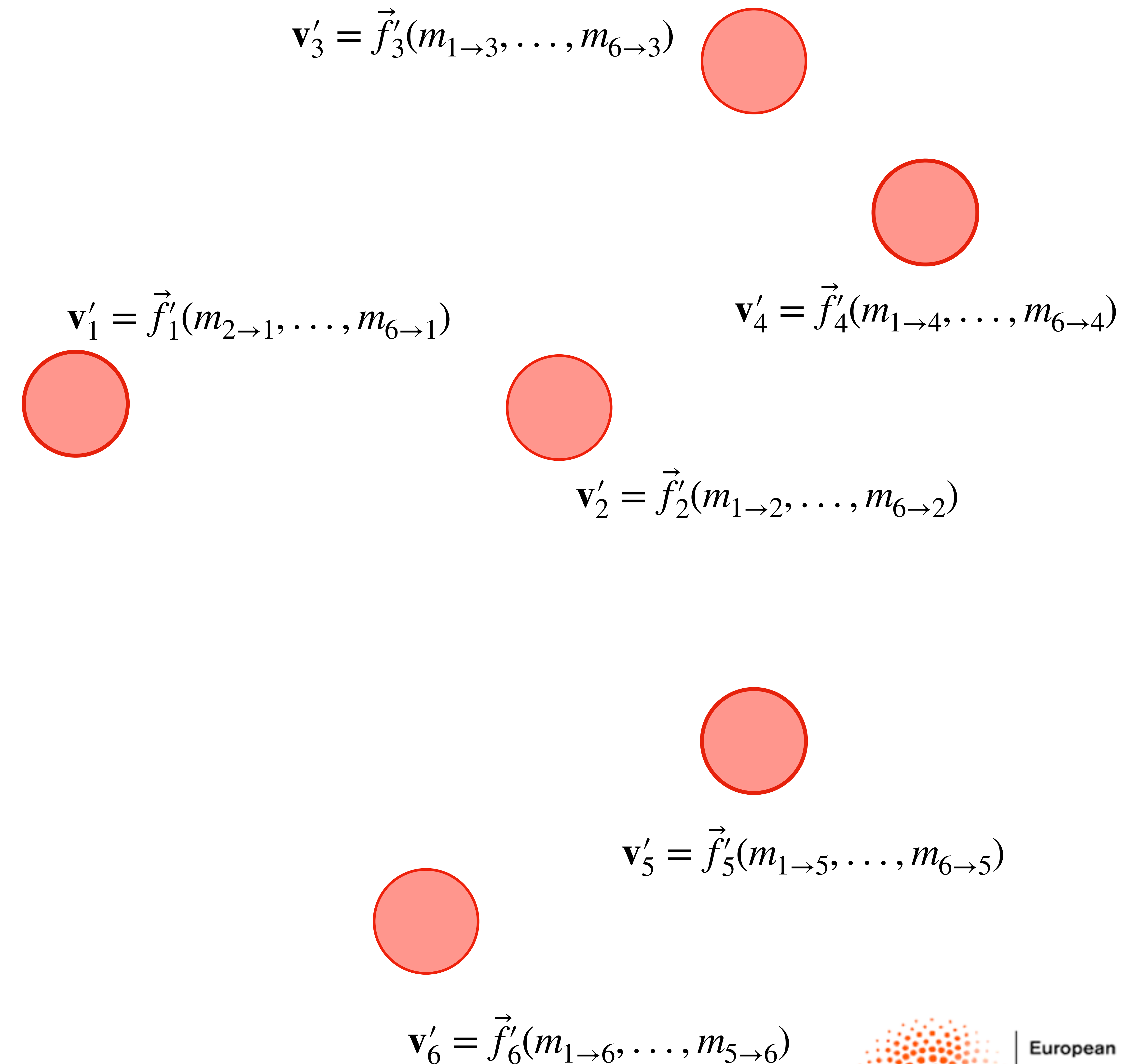


https://arxiv.org/pdf/1704.01212.pdf

European Research Council

Graph Networks

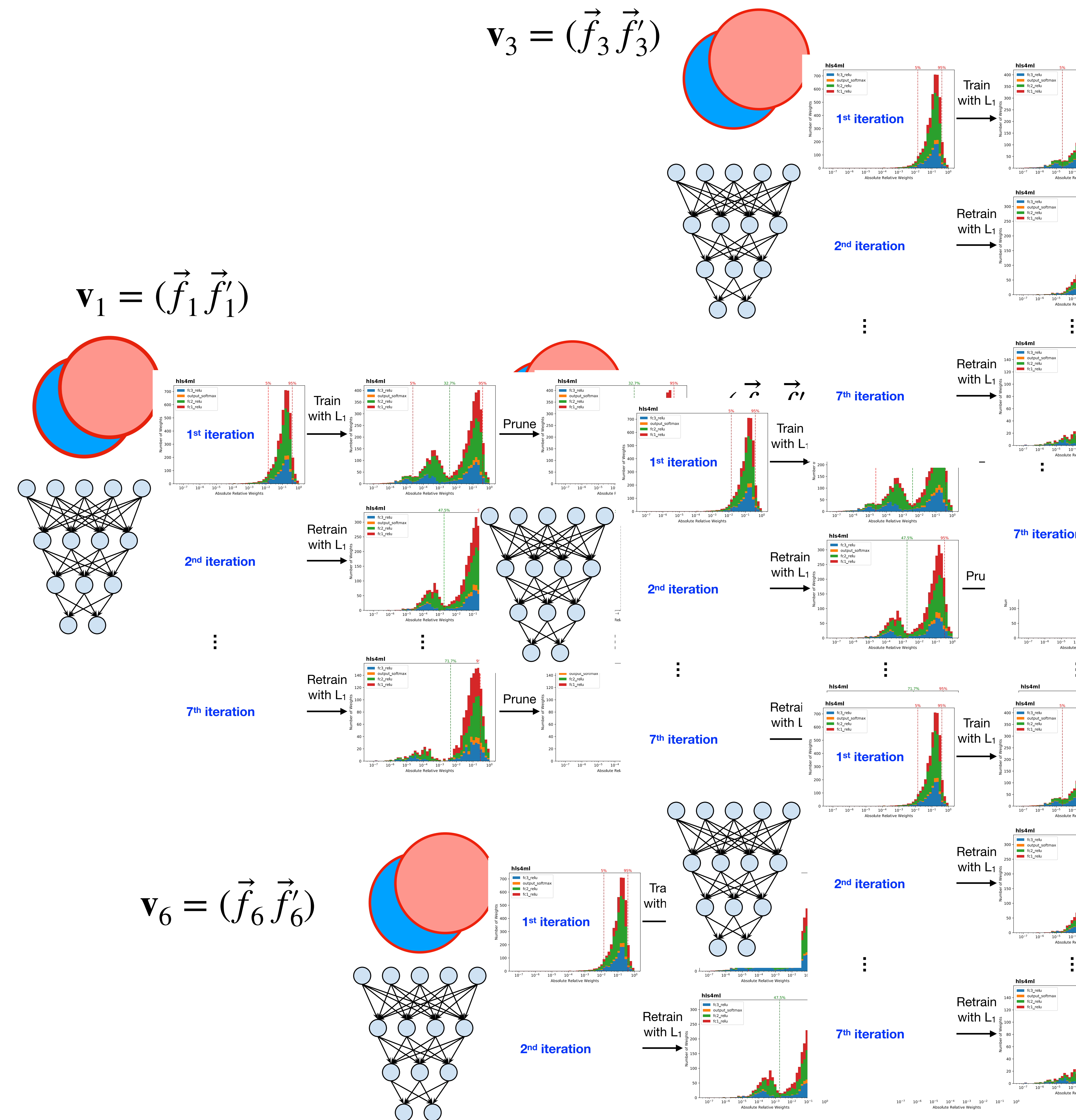
- Graphs Nets are architectures based on an abstract representation of a given dataset
  - Each example in a dataset is represented as a set of vertices
  - Each vertex is embedded in the graph as a vector of features
  - Vertices are connected through links (edges)
  - Messages are passed through links and aggregated on the vertices
  - **A new representation of each node is created, based on the information gathered across the graph**



https://arxiv.org/pdf/1704.01212.pdf

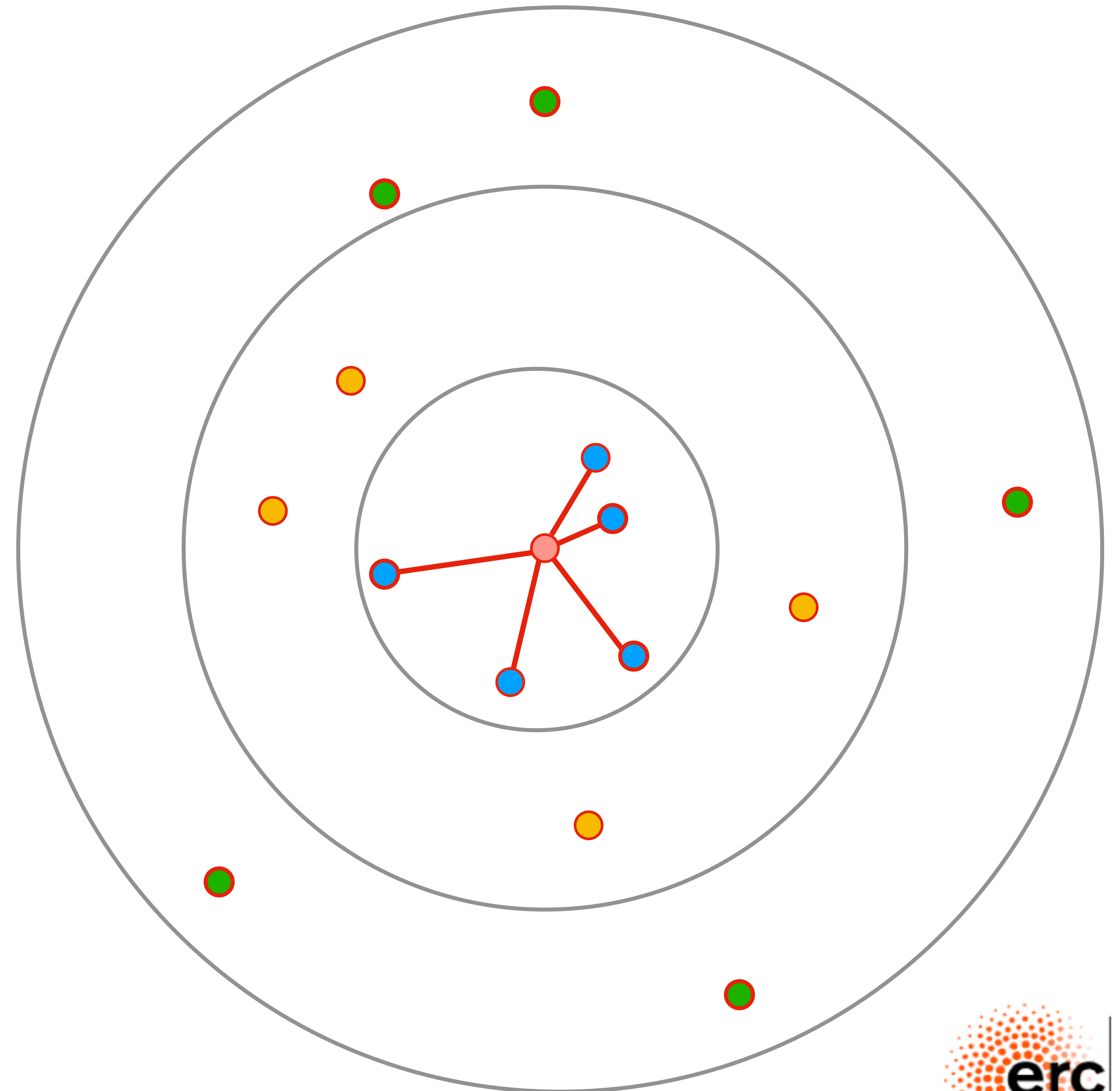
The task-solving step

- *The task-solving step can happen on each vertex (is this a real particle or noise?) or across the graph (is this a b-jet?)*
- *Usually, this is done with a DNN taking*
  - *the initial features f_i*
  - *the learned representation f_i'*
  - *[optional] some ground-truth label (for classifiers)*



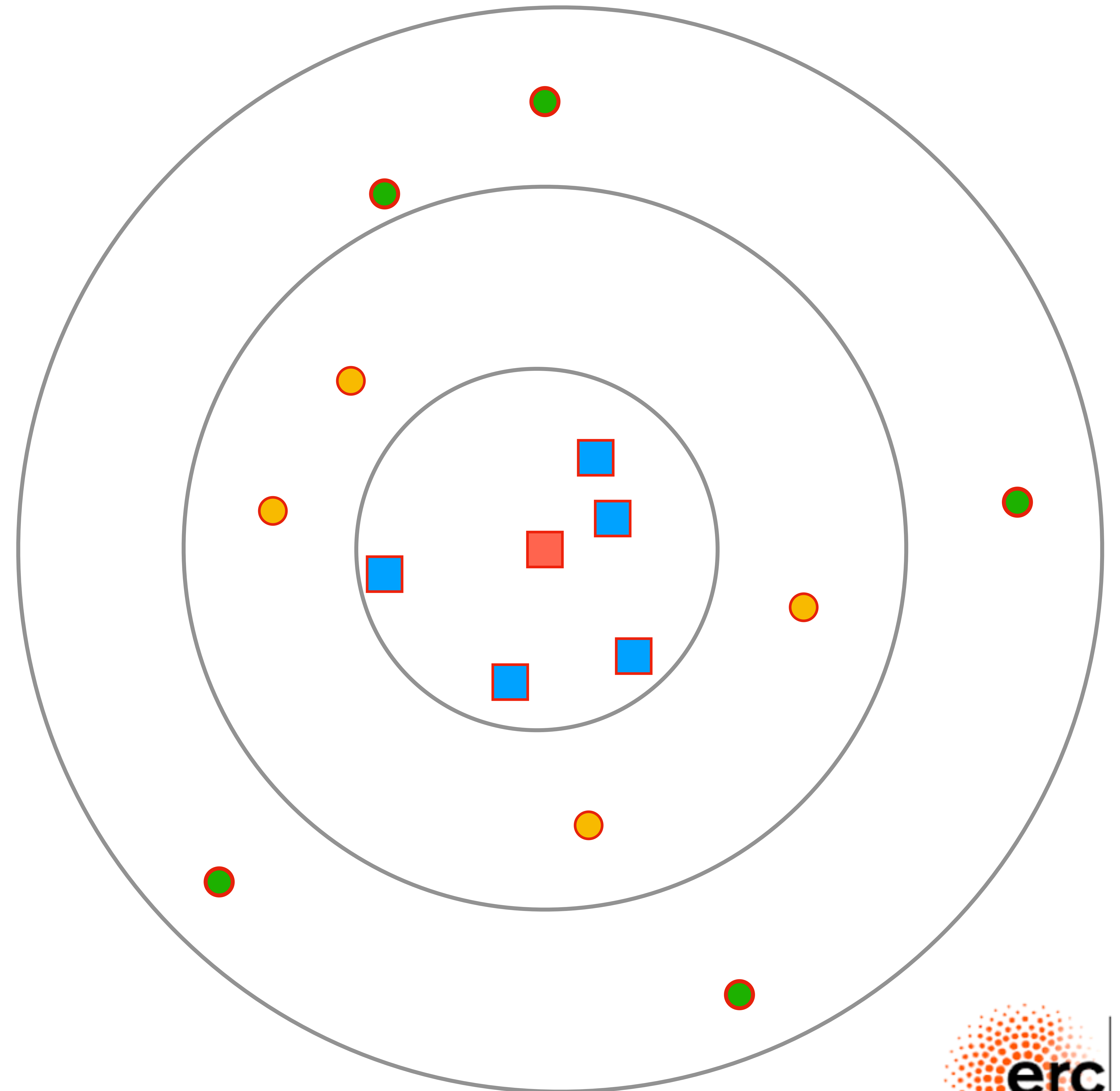
...and repeat

- Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex
  - You could start from coordinates in real space + some feature
  - Build function of them
  - Build functions of functions of them
  - At each step, you improve knowledge on your vertex V



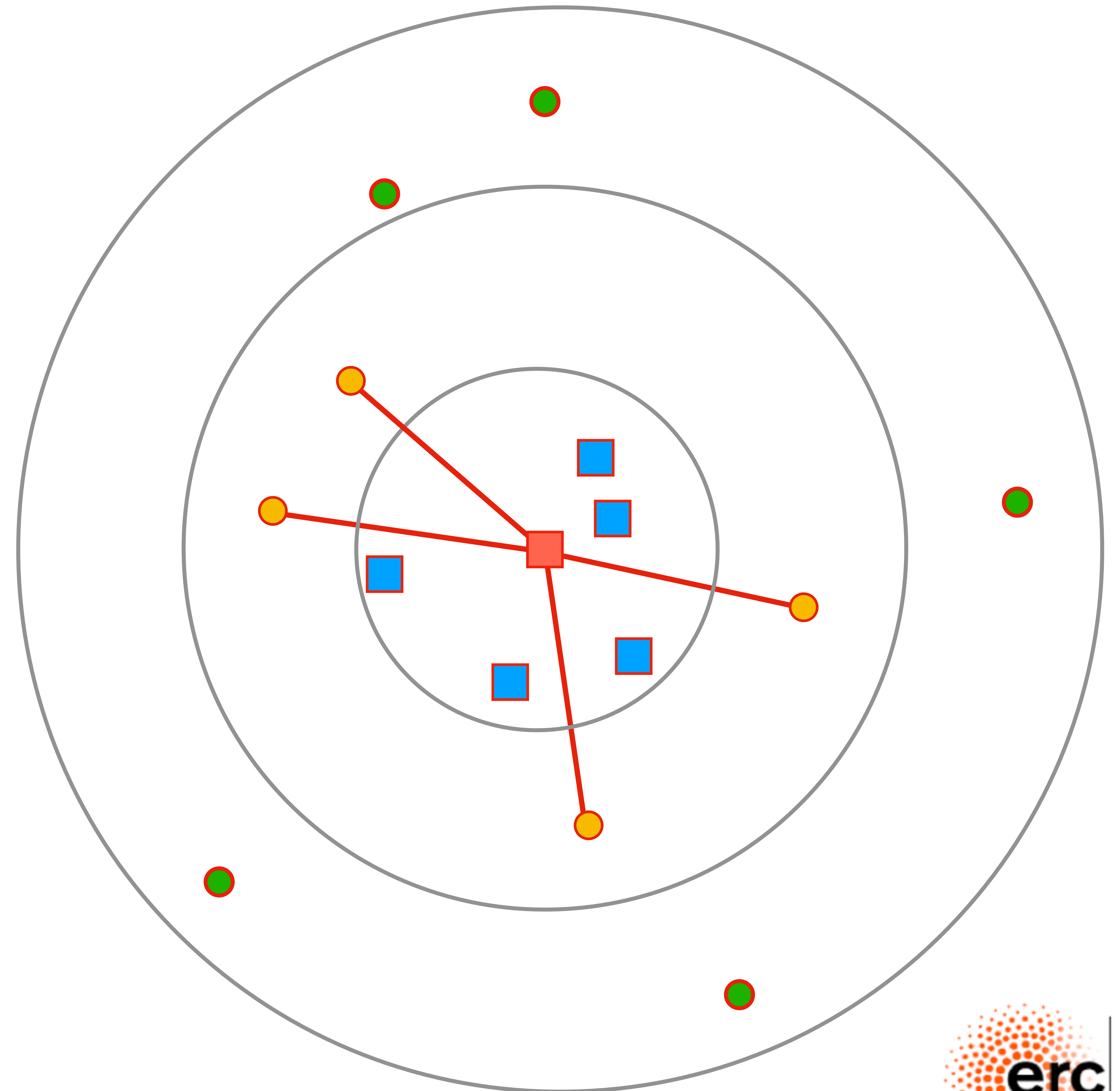
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



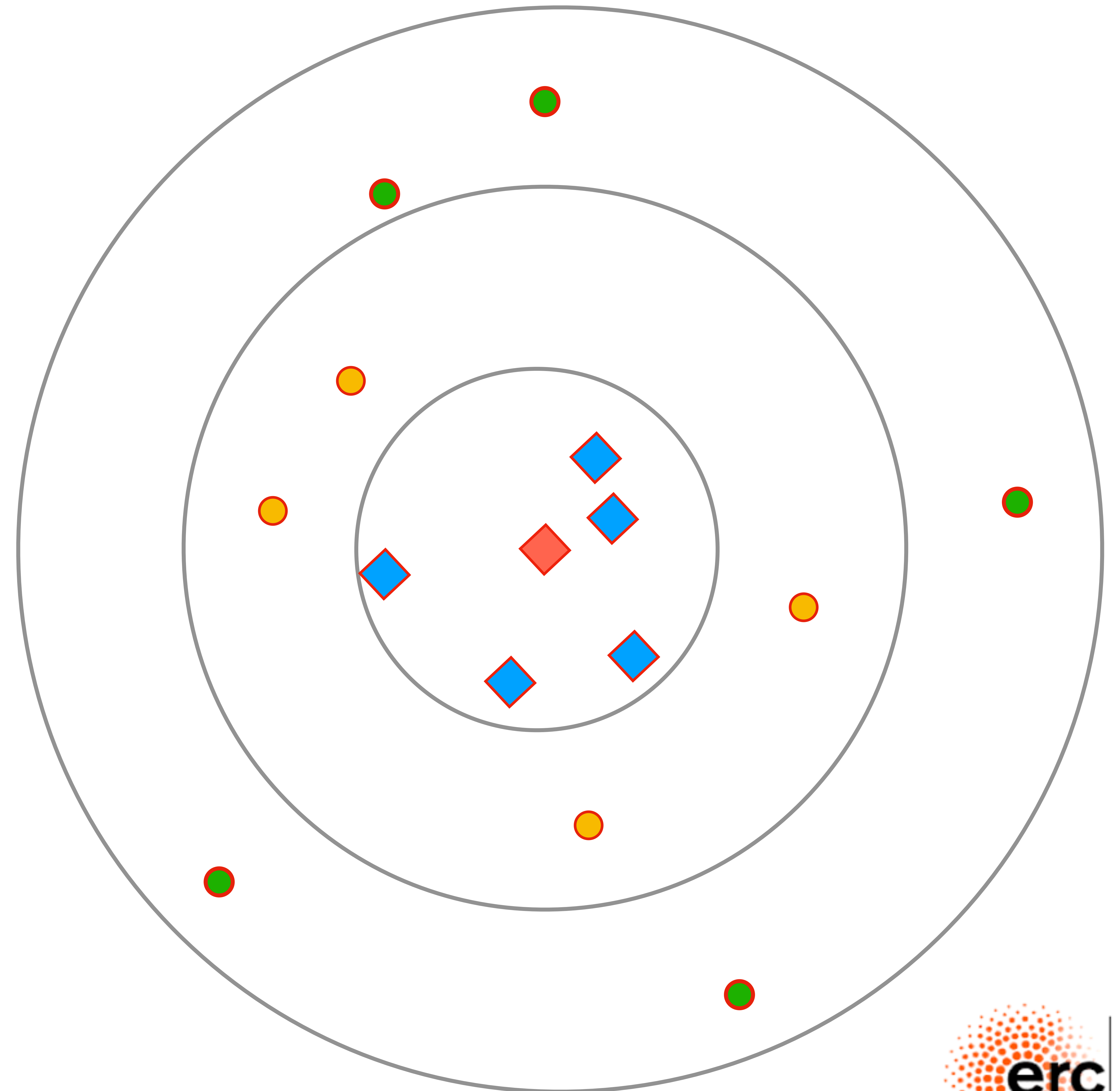
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



...and repeat

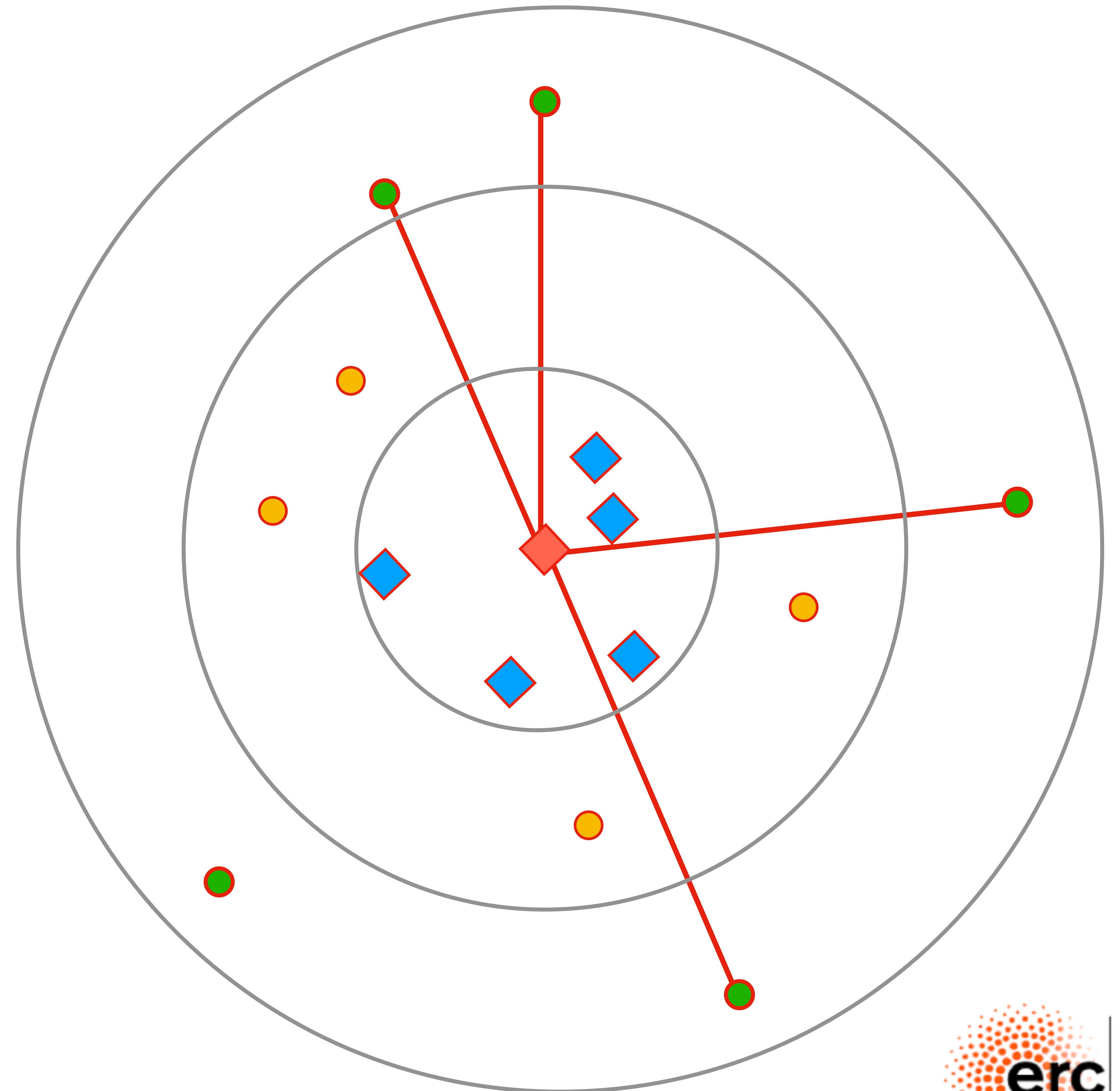
- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



European Research Council

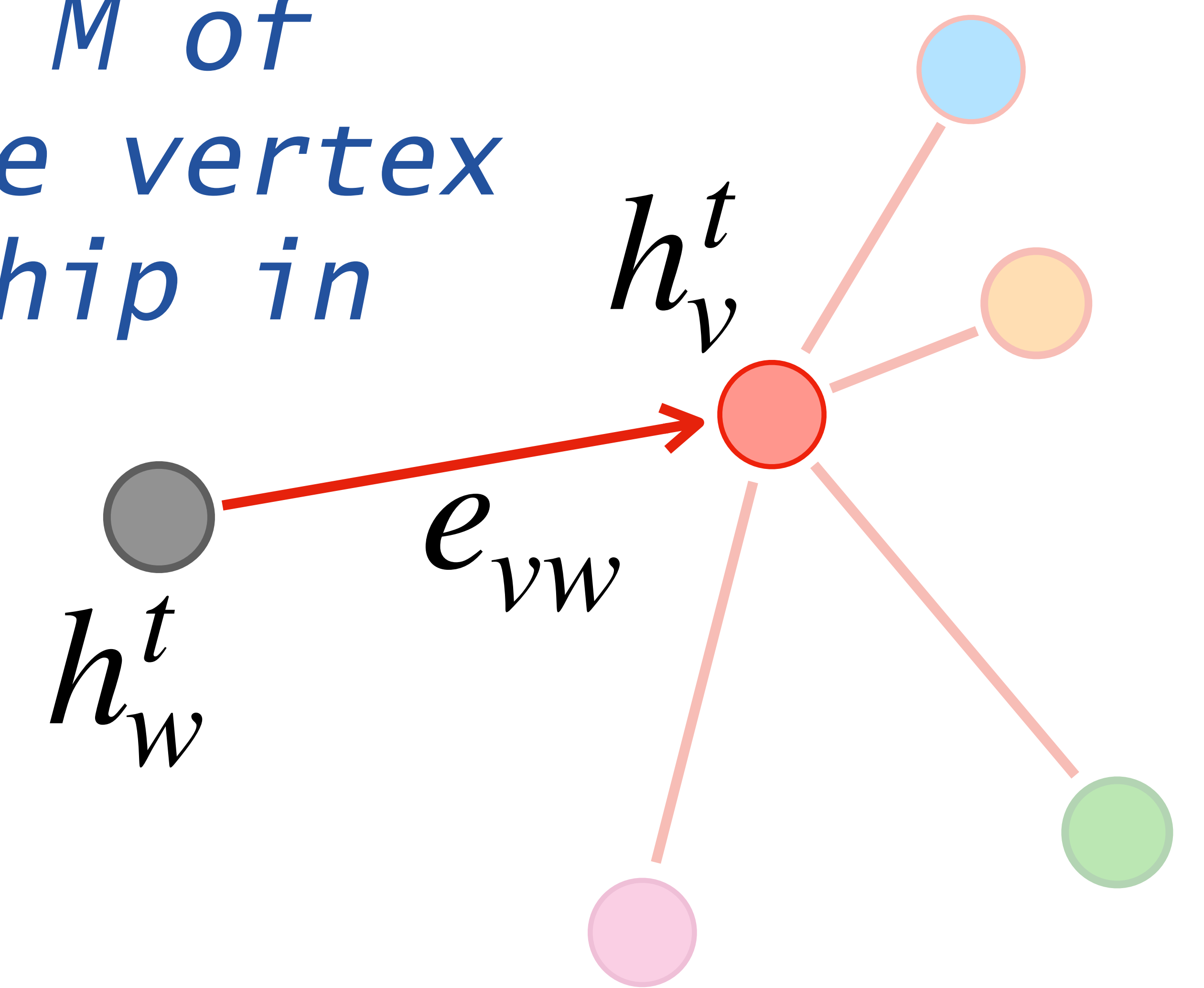
...and repeat

- *Once message is passed, aggregated at each vertex V and processed, it creates a new representation of each vertex*
  - *You could start from coordinates in real space + some feature*
  - *Build function of them*
  - *Build functions of functions of them*
  - *At each step, you improve knowledge on your vertex V*



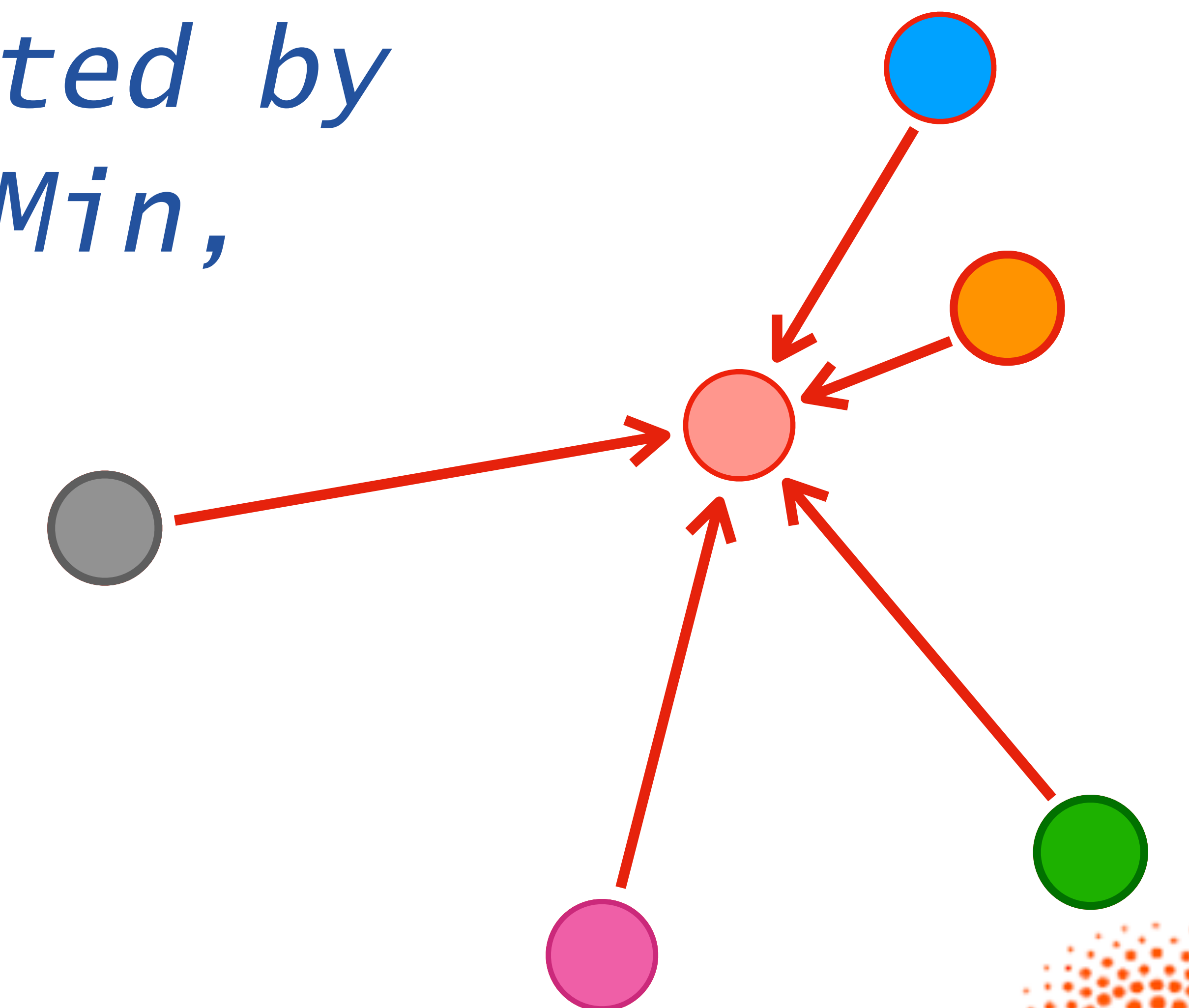
With equations…

- *Your message at iteration t is some function M of the sending and receiving features, plus some vertex features (e.g., business relation vs friendship in social media)*



$$M_t(h_v^t, h_w^t, e_{vw})$$

- *The message carried to a vertex v is aggregated by some function (typically sum, but also Max, Min, etc.)*



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

European Research Council

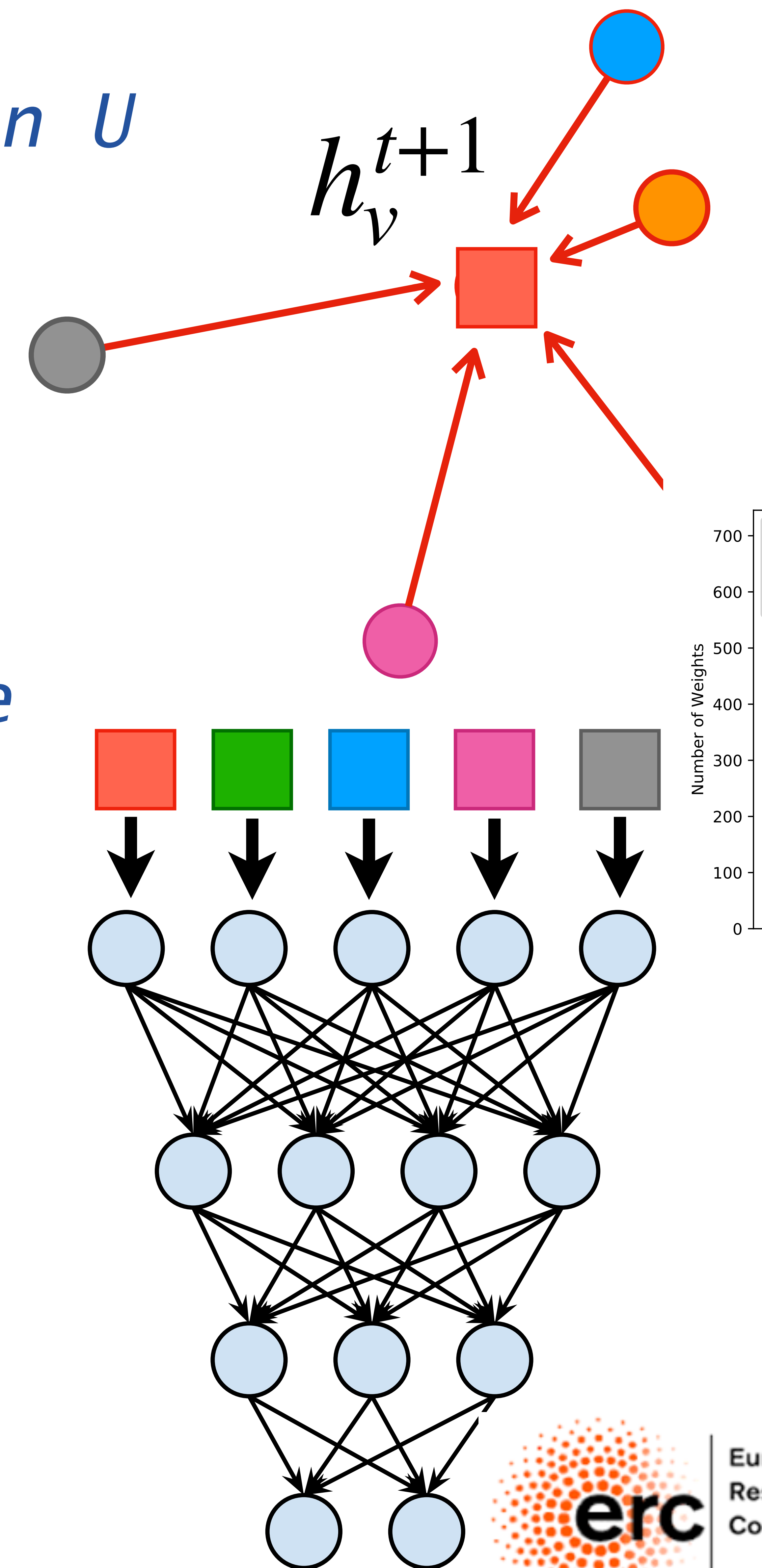
With equations…

- *The state of vertex v is updated by some function U of the current state and the gathered message*

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

- *After T iterations, the last representations of the graph vertices are used to derive the final output answering the question asked (classification, regression, etc.), typically through a NN*

$$\hat{y} = R(h_v^T \,|\, v \in G)$$



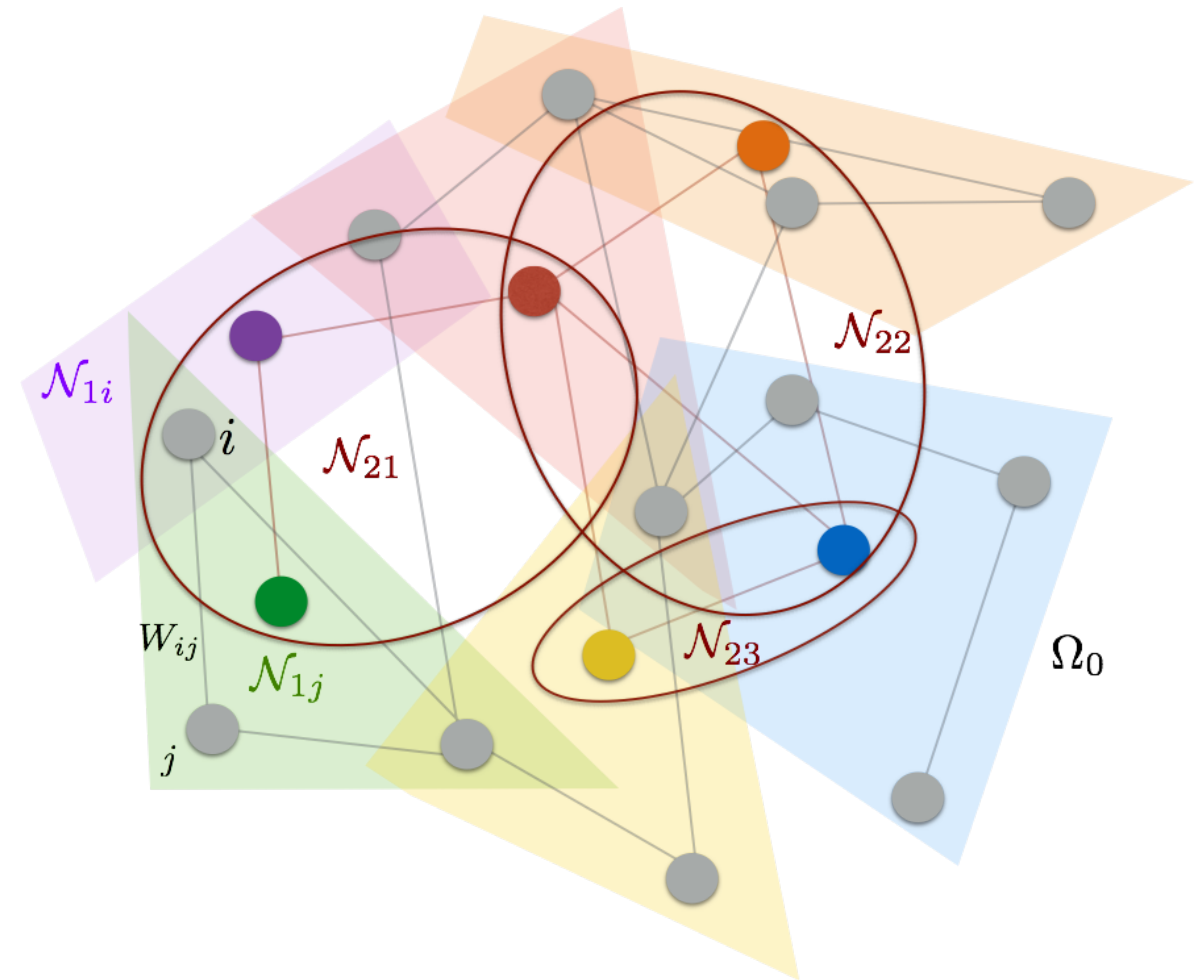
Learning Message

- Typically, the M, U, and R functions are learned from data
  - Expressed as neural networks (fully connected NNs, recurrent NNs, etc.)
  - Which networks to use depends on the specific problem, as much as the graph-building rules
- But you could inject domain knowledge in the game
  - You might know that SOME message is carried by some specific functions (e.,g., Newton's low for N-body system simulation)
  - You could then use analytic functions for some message
  - You could still use a learned function for other messages
- The trick is dealing with differentiable functions not to spoil your back propagation
  - Graph networks become a tool for probabilistic programming

CERN
erc European Research Council

A little bit of History

- *(in this millenium) Graph networks started (as often it is the case) with a Yann LeCun et al. paper*
- *They tried to generalise CNNs beyond the regular-array dataset paradigm*
- *They replaced the translation-invariant kernel structure of CNNs with hierarchical clustering*

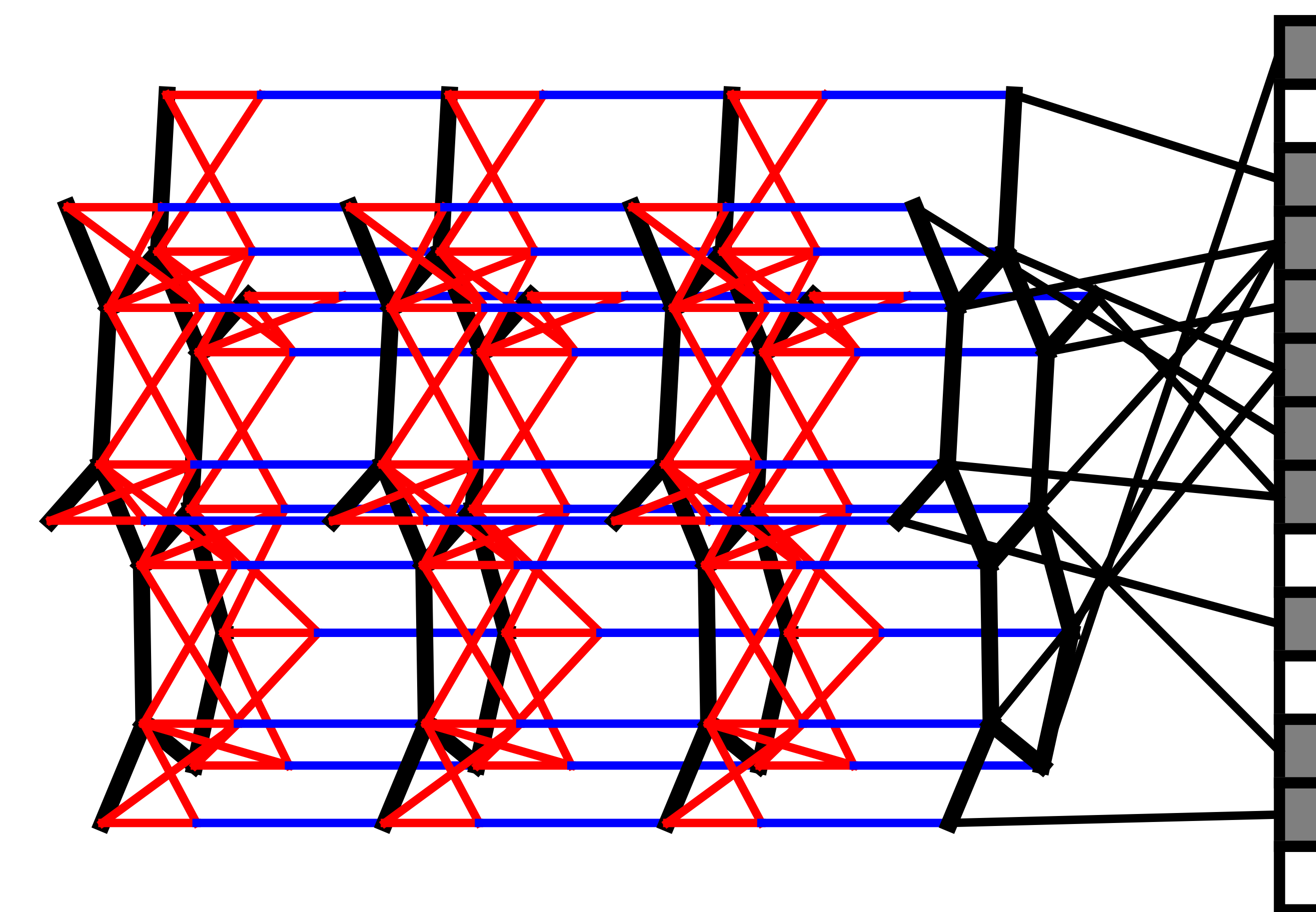


https://arxiv.org/abs/1312.6203

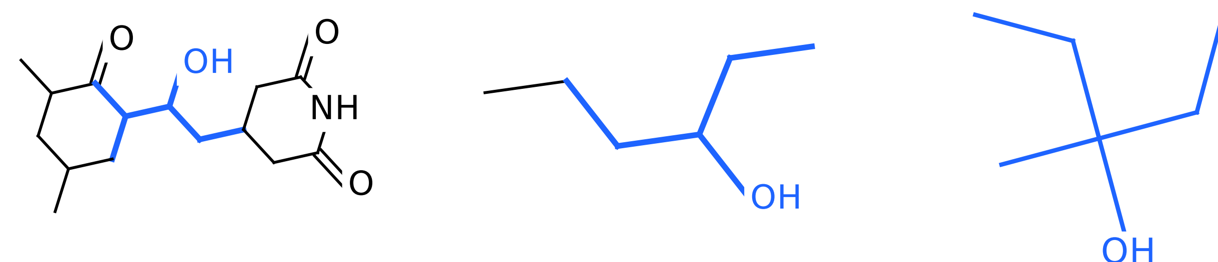
European Research Council

A little bit of History

- *The idea of message passing can be tracked to a ’15 paper by Duvenaud et al.*
- *The paper introduces “a convolutional neural network that operates directly on graphs”*
- *Language is different, but if you look at the algorithm it is pretty much what we discussed (for specific network architecture choices)*



Fragments most activated by pro-solubility feature



Fragments most activated by anti-solubility feature

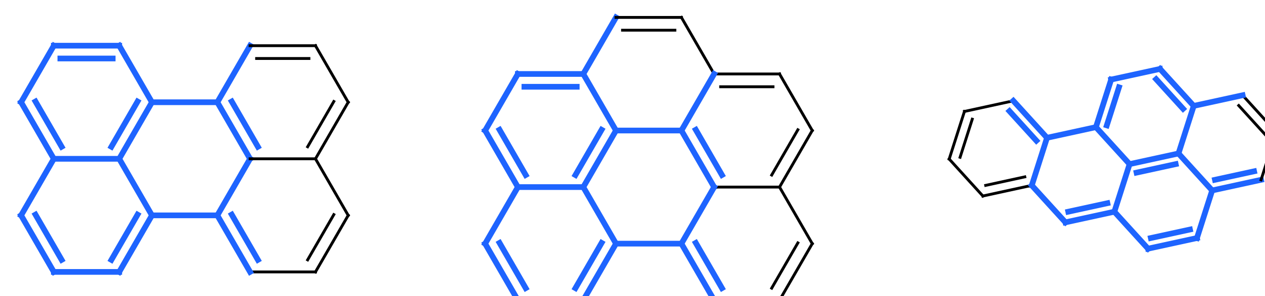


Figure 4: Examining fingerprints optimized for predicting solubility. Shown here are representative examples of molecular fragments (highlighted in blue) which most activate different features of the fingerprint. *Top row:* The feature most predictive of solubility. *Bottom row:* The feature most predictive of insolubility.

Algorithm 2 Neural graph fingerprints

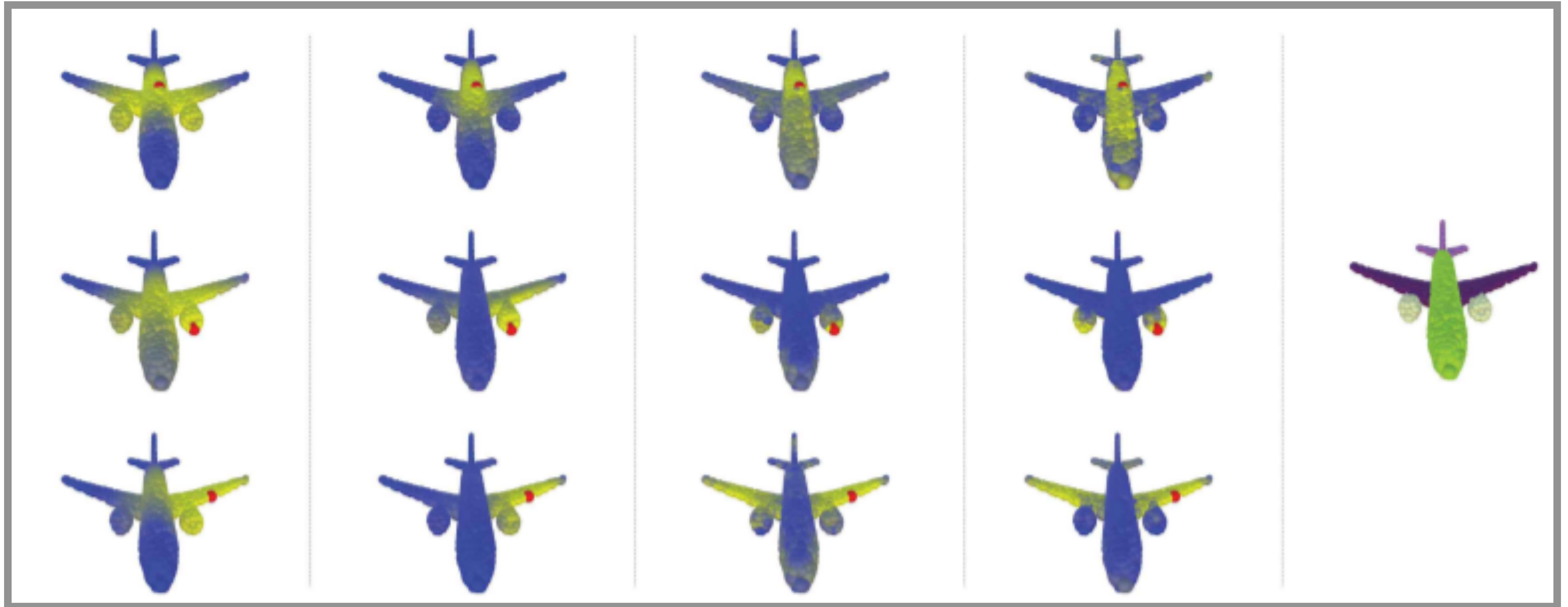
1: **Input:** molecule, radius R, hidden weights $H_1^1 \dots H_R^5$, output weights $W_1 \dots W_R$
2: **Initialize:** fingerprint vector $\mathbf{f} \leftarrow \mathbf{0}_S$
3: **for** each atom a in molecule
4: $\quad \mathbf{r}_a \leftarrow g(a)$ ▷ lookup atom features
5: **for** $L = 1$ to R ▷ for each layer
6: $\quad$ **for** each atom a in molecule
7: $\quad\quad \mathbf{r}_1 \dots \mathbf{r}_N = \text{neighbors}(a)$
8: $\quad\quad \mathbf{v} \leftarrow \mathbf{r}_a + \sum_{i=1}^{N} \mathbf{r}_i$ ▷ sum
9: $\quad\quad \mathbf{r}_a \leftarrow \sigma(\mathbf{v} H_L^N)$ ▷ smooth function
10: $\quad\quad \mathbf{i} \leftarrow \text{softmax}(\mathbf{r}_a W_L)$ ▷ sparsify
11: $\quad\quad \mathbf{f} \leftarrow \mathbf{f} + \mathbf{i}$ ▷ add to fingerprint
12: **Return:** real-valued vector $\mathbf{f}$

https://arxiv.org/pdf/1509.09292.pdf

Further Reading & Coding

- A few recent reviews that could guide you through the many applications and networks
  - A nice BLOG article on GNNs
  - Another nice BLOG article on GNNs
  - A generic review
  - A particle-physics specific one
- A few GitHub entries
  - JEDI-net Interaction Networks for jet tagging on these data
  - PUPPIML: GGNN for pileup subtraction
  - A small GarNet example that fits an FPGA on these data

European Research Council



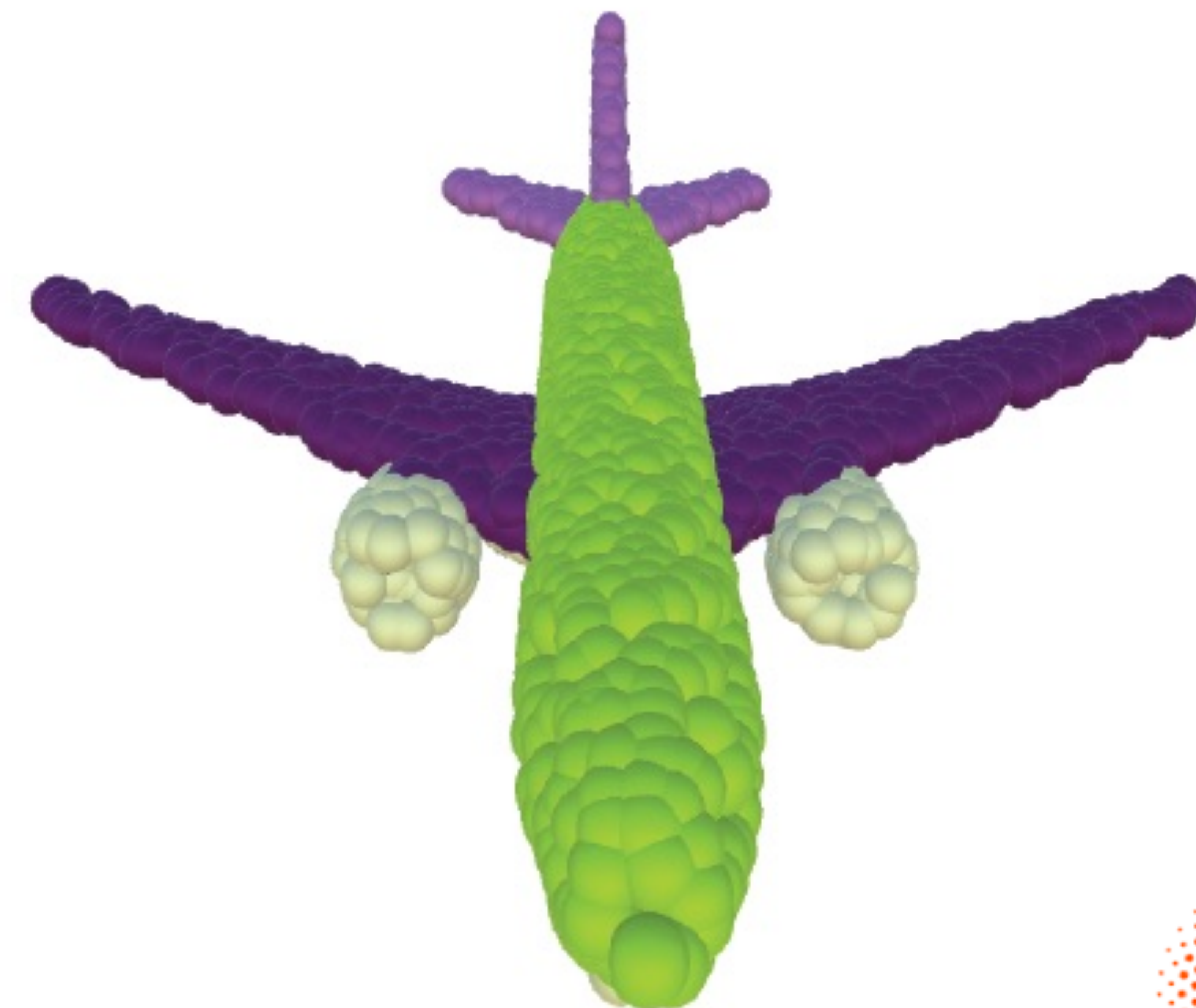
Edge Conv

CERN

European Research Council
erc

EdgeConv

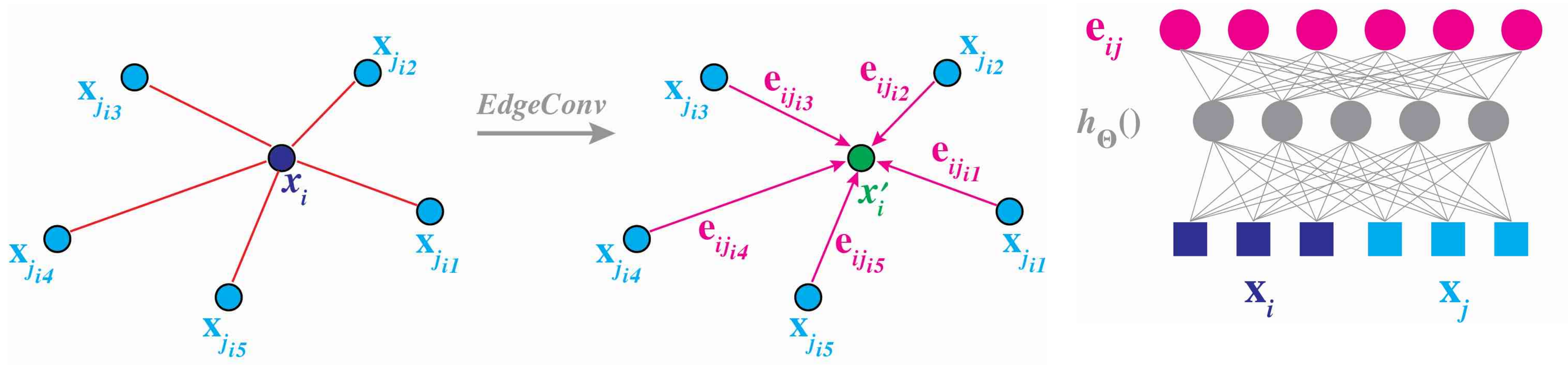
- *Dynamic Graph CNN (DGCNN) is one kind of message-passing neural network*
- *It uses EdgeConv layers to perform point-cloud segmentation*
- *Segmentation is the process of clustering pixels in an image into objects*
- *EdgeConv was capable of extending semantic segmentation beyond nearby-pixel clustering*
  - *the two wings of the airplane are associated to the same cluster, since they are found to be similar*



https://arxiv.org/abs/1801.07829

EdgeConv

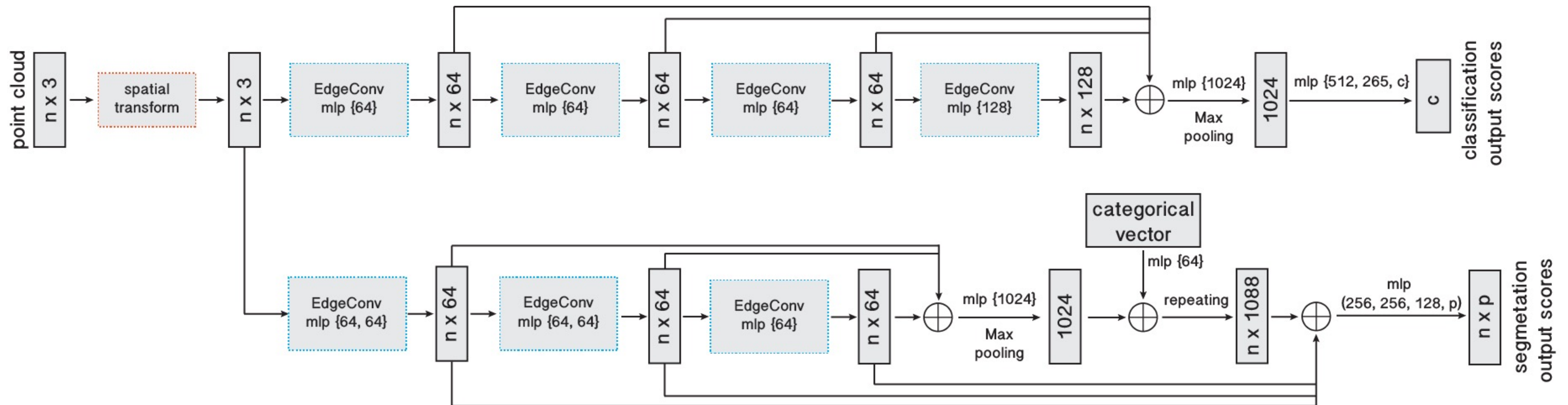
- *EdgeConv is a typical message passing architecture, using fully-connected networks to learn edge representation (the h functions)*



https://arxiv.org/abs/1801.07829

EdgeConv

- But the actual model is much more complicated than that



- Each EdgeConv layer runs a message passing and creates an updated representation of the graph of points

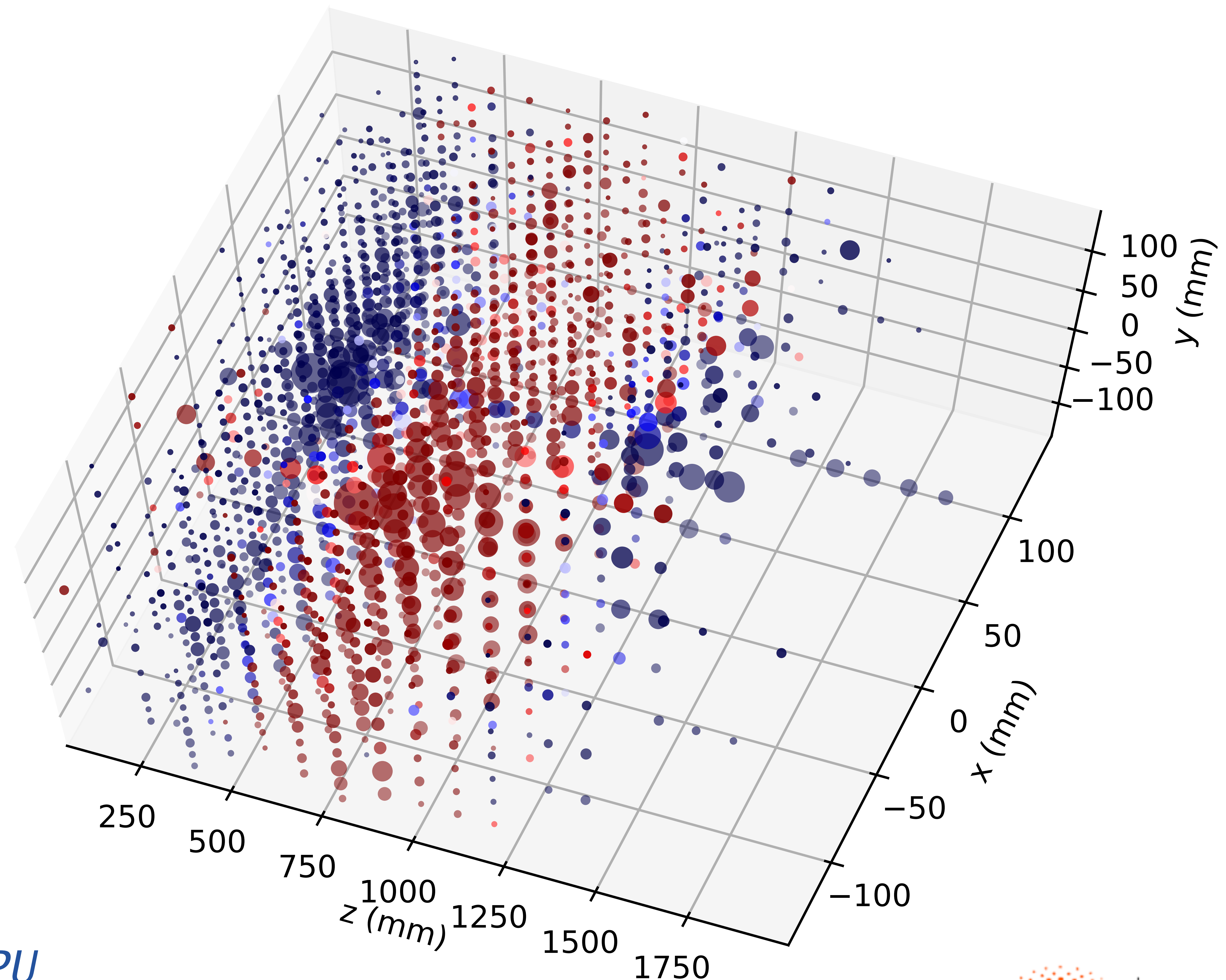
- Similar to a CNN, but capable of processing unordered sets of points

https://arxiv.org/abs/1801.07829

European Research Council

EdgeConv for Particle Physics

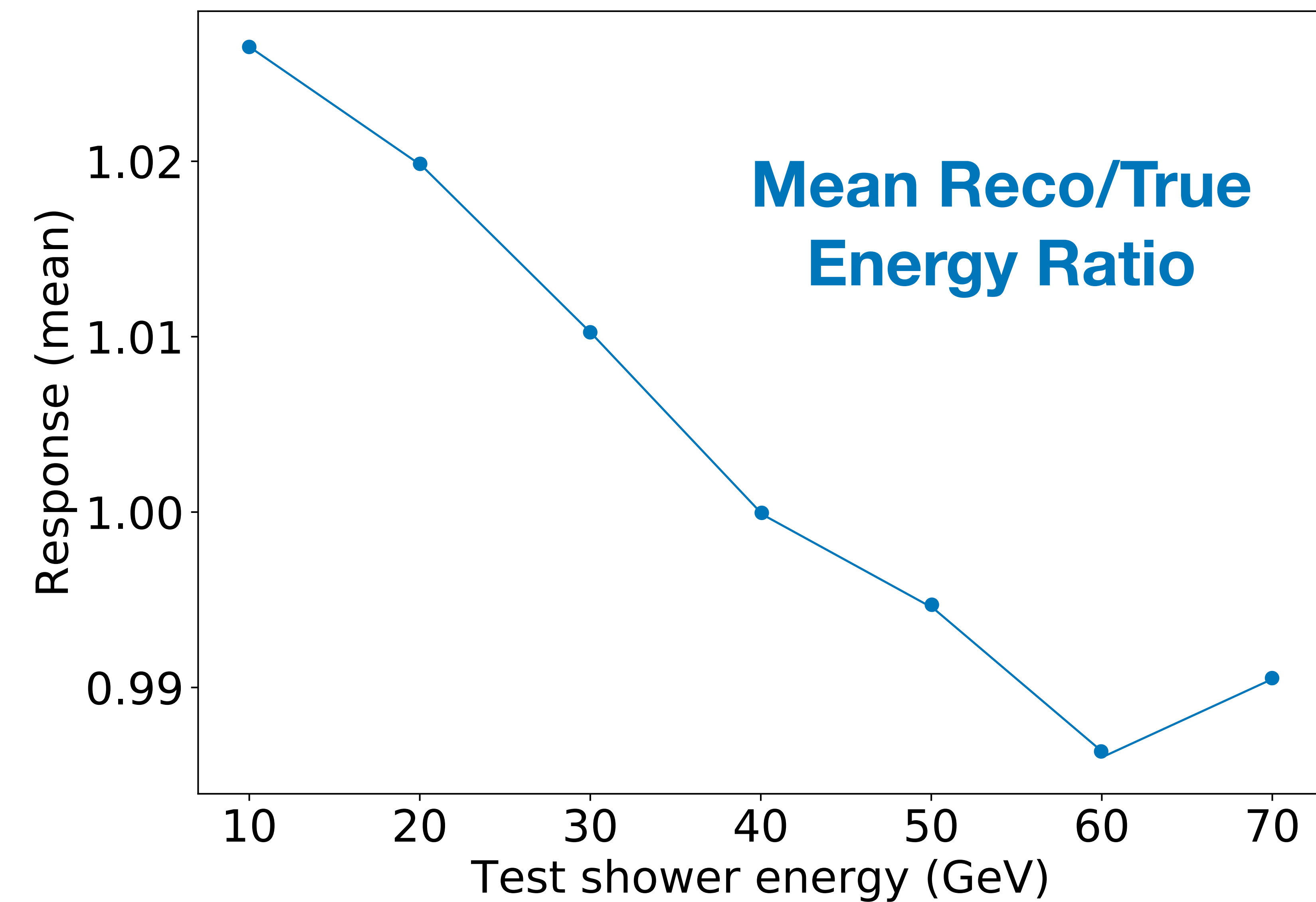
- DGCNN fits very well particle reconstruction in High Energy Physics
  - Particles seen as energy showers in calorimeters
  - DGCNN can be trained to distinguish overlapping showers from different particles
- Success comes at some computational cost:
  - 15 sec/event on a CPU
  - Lowered to 5 sec/event on GPU when using a batch of 100

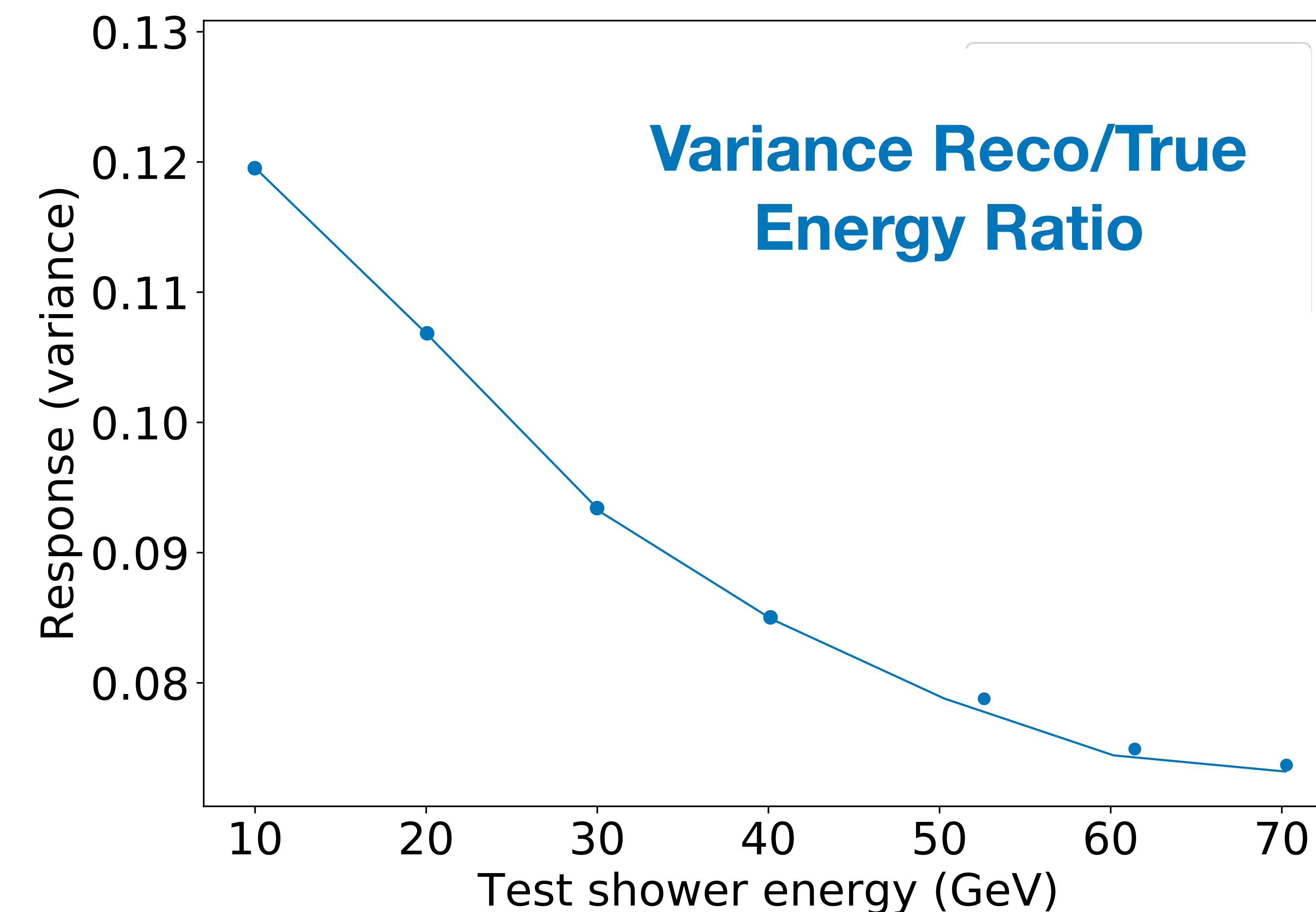


European Research Council

EdgeConv for Particle Physics

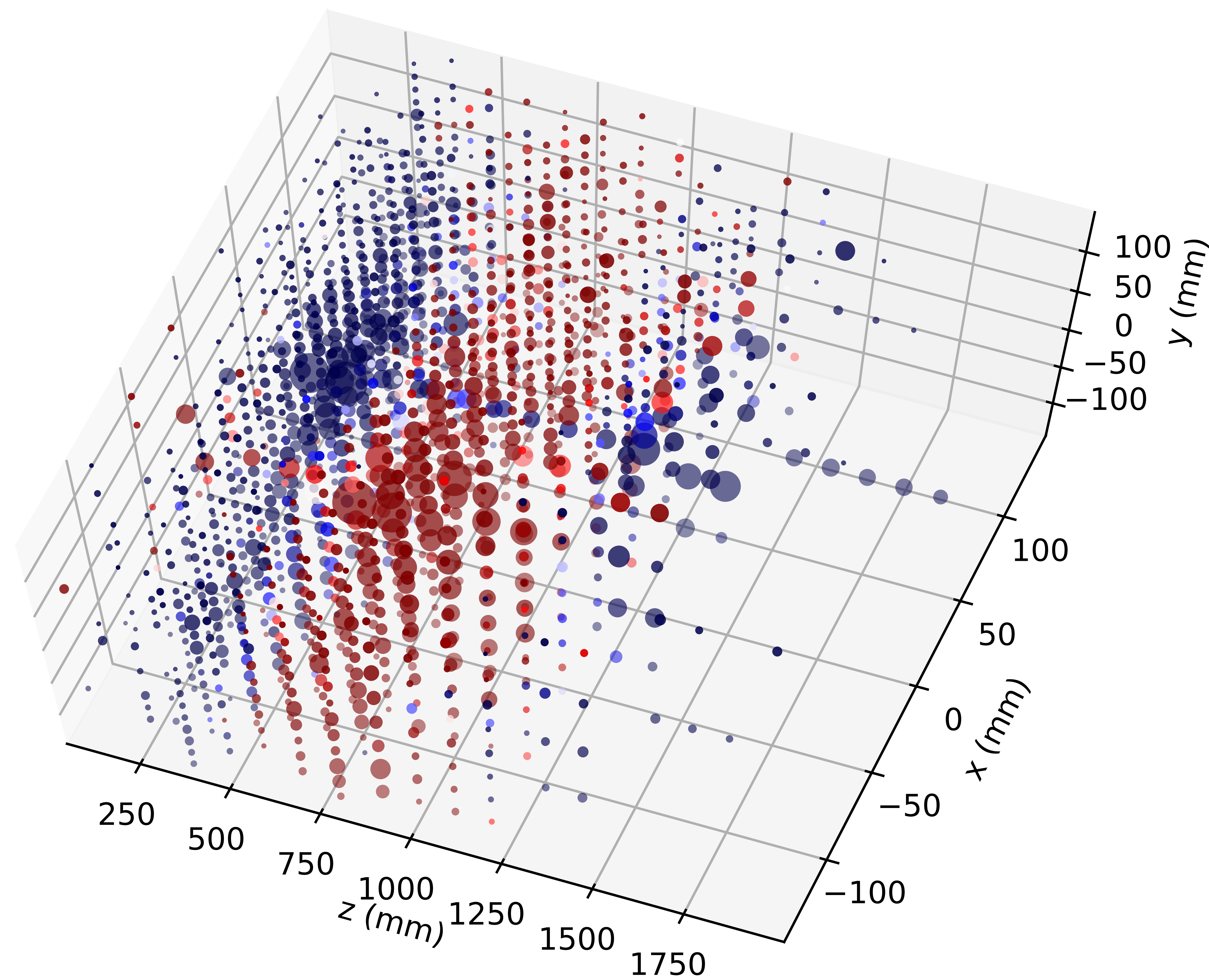
- *DGCNN fits very well particle reconstruction in High Energy Physics*
  - *Particles seen as energy showers in calorimeters*
  - *DGCNN can be trained to distinguish overlapping showers from different particles*
- *Success comes at some computational cost:*
  - *15 sec/event on a CPU*
  - *Lowered to 5 sec/event on GPU when using a batch of 100*



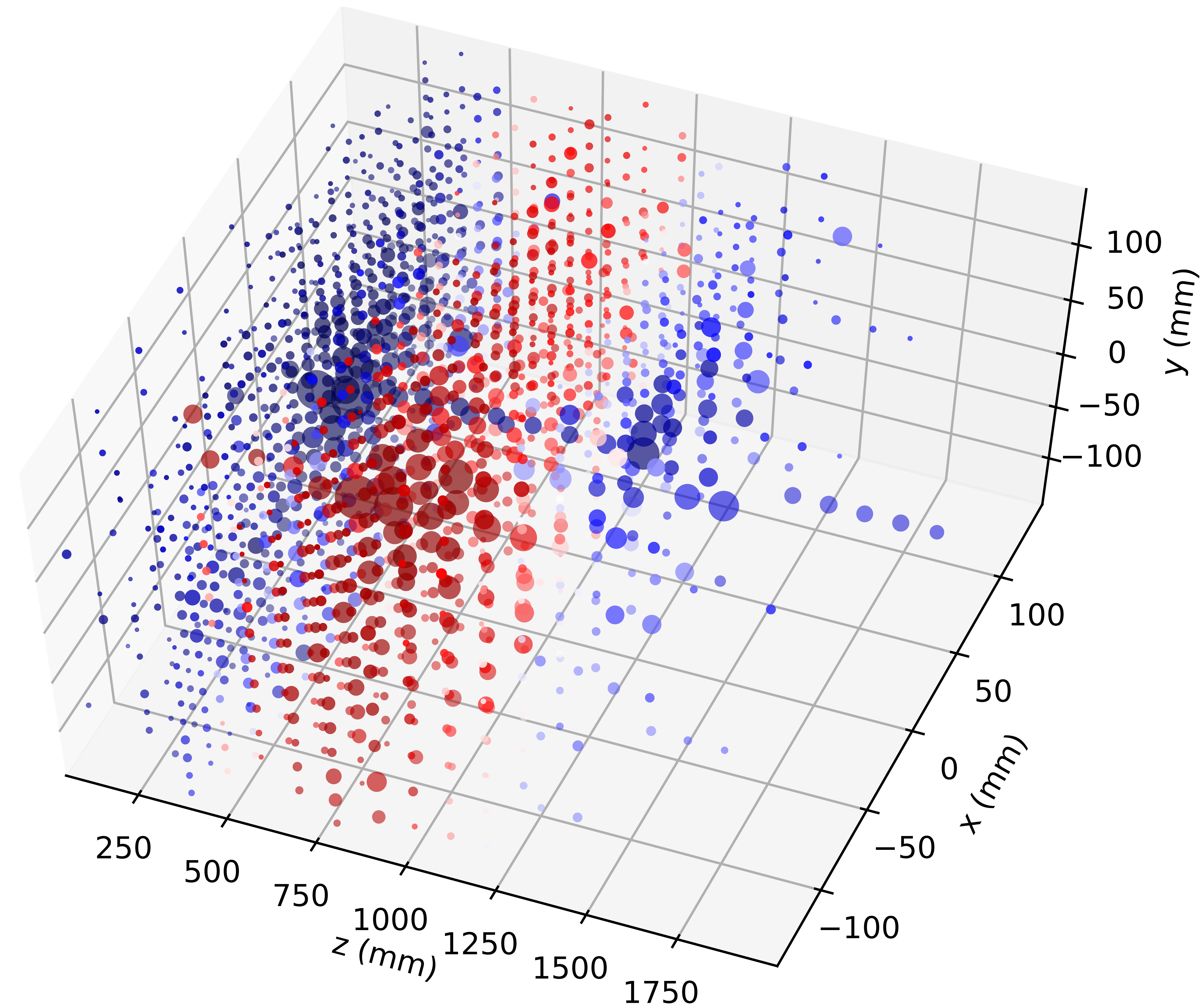


CERN
erc European Research Council

Separating overlapping showers



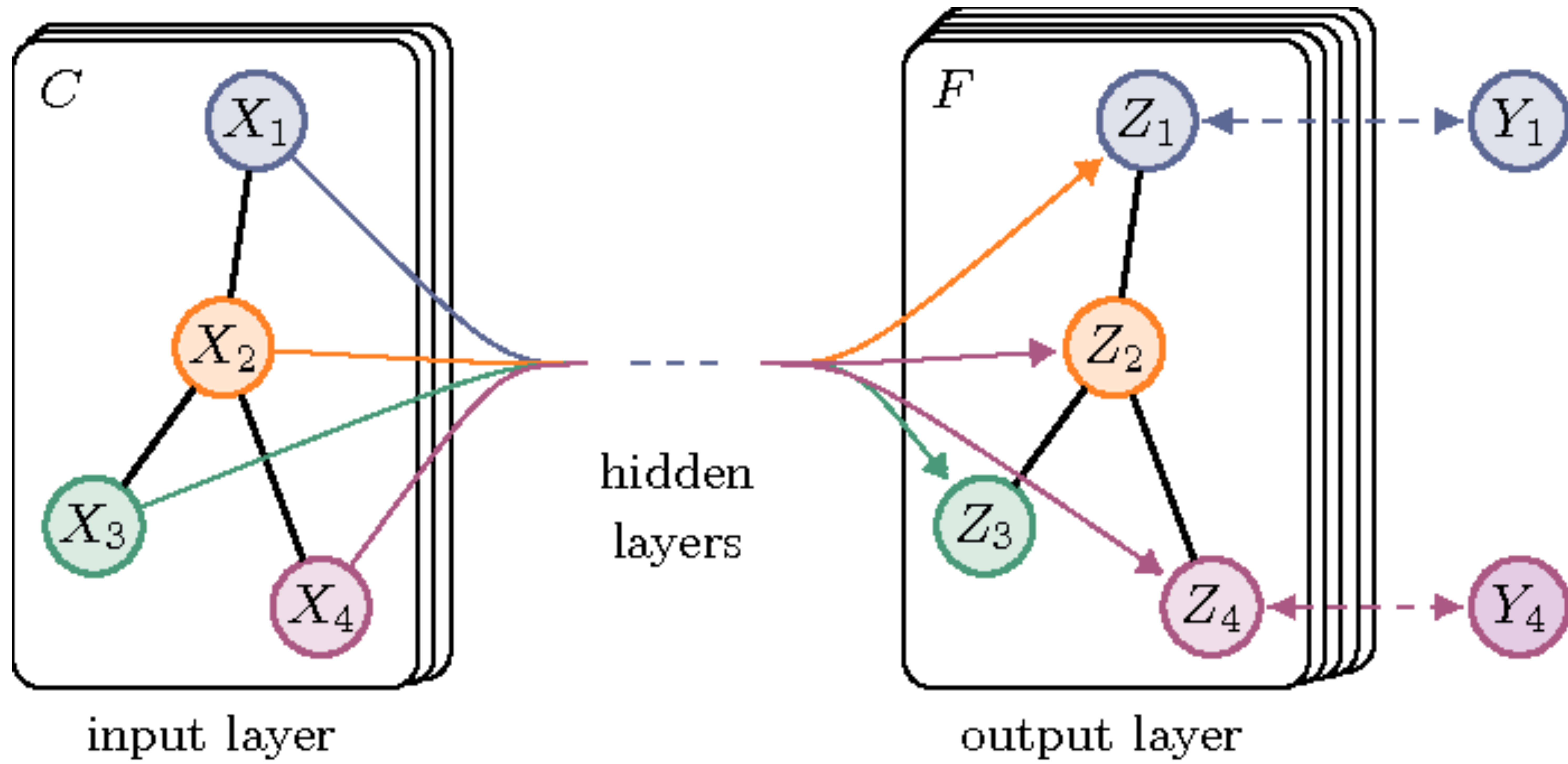
(a) Truth



(b) Reconstructed

European Research Council

CERN



Convolutional Graph Neural Networks

erc European Research Council

Generalising CNN to point clouds

How Graph Convolutions work

CNN on image

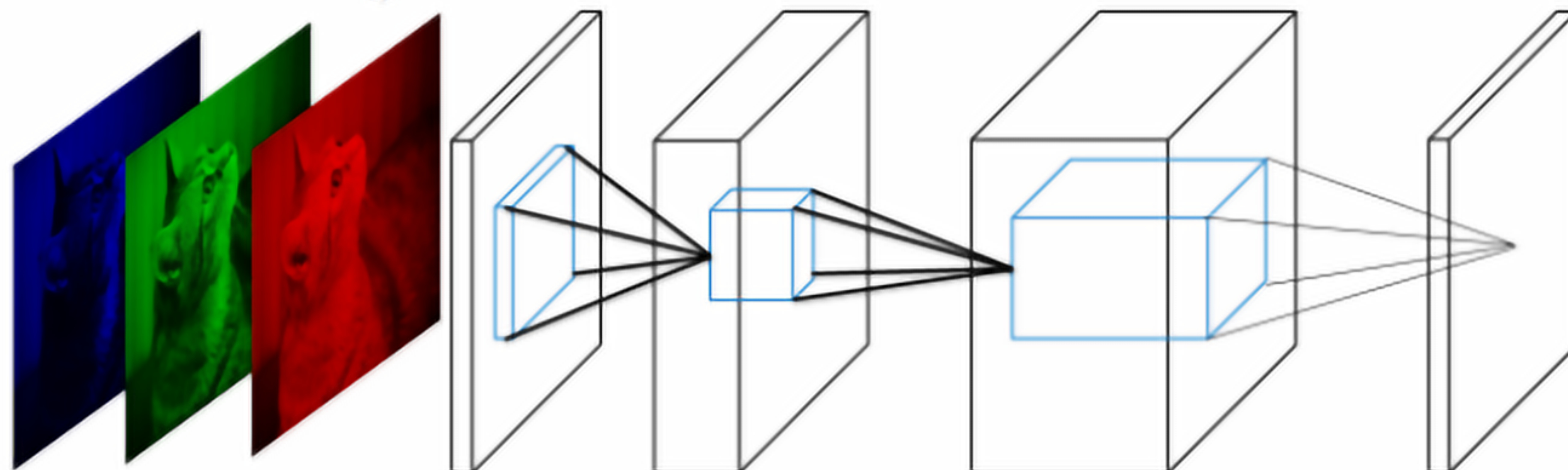
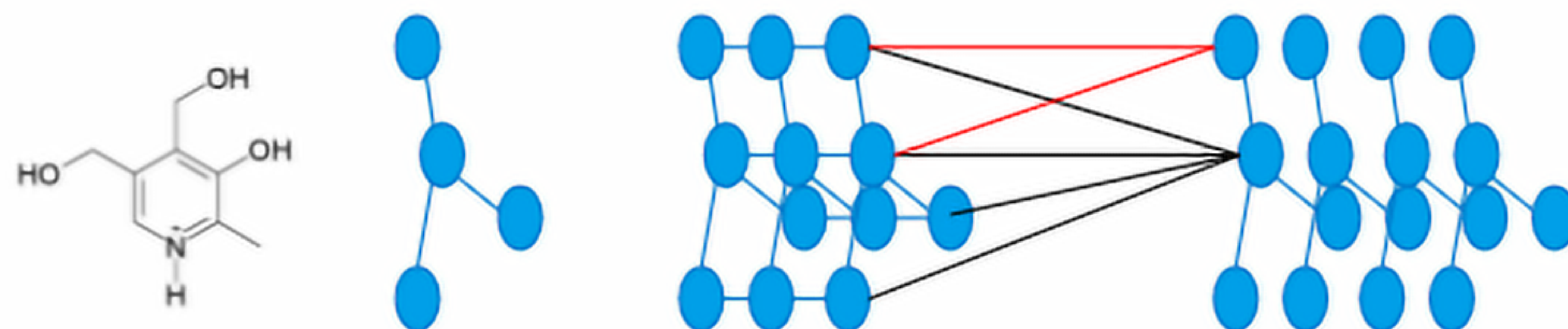


Image class label

Graph convolution

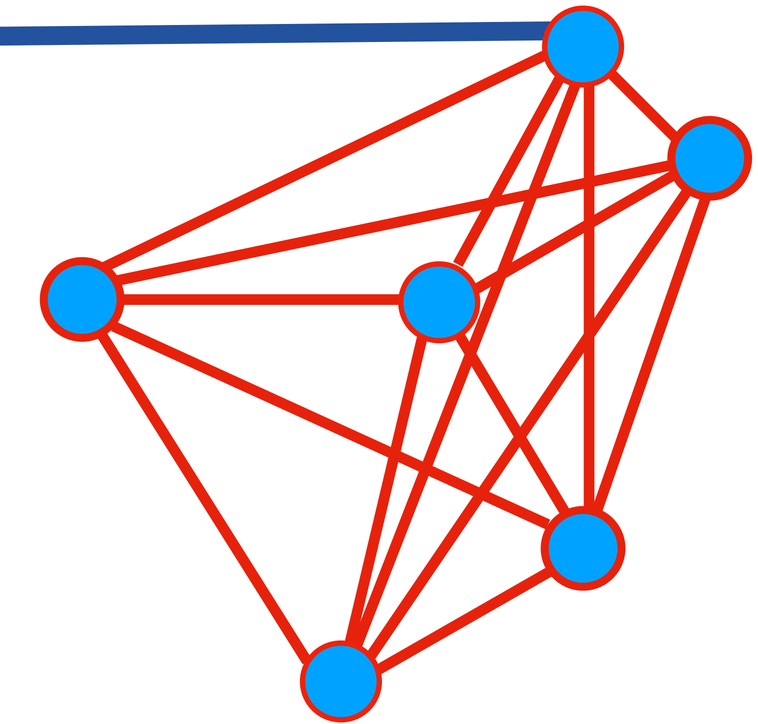


Chemical property

Convolution “kernel” depends on Graph structure

CERN
European Research Council
erc

The math

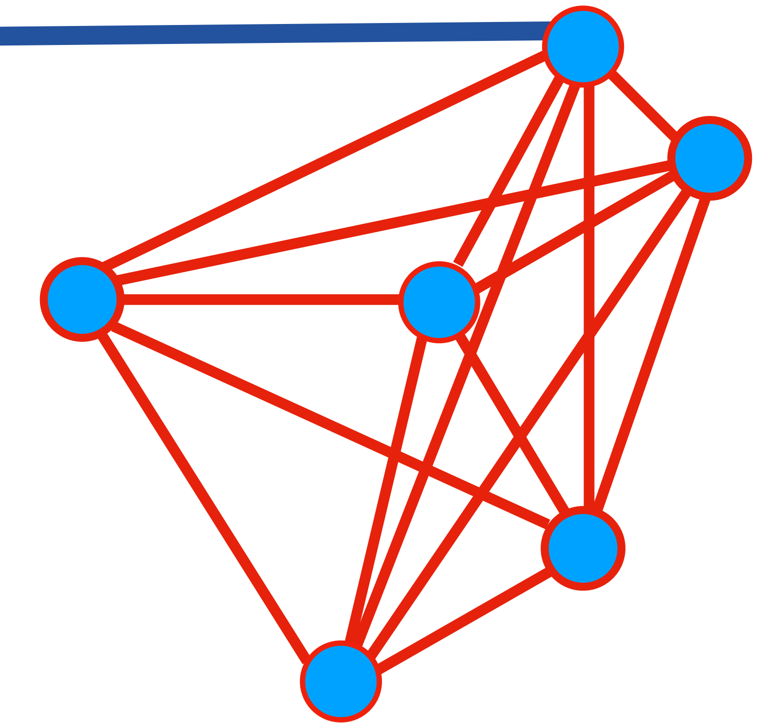


Consider a graph with n nodes, each containing f features.

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- *The inputs X*
- *The weights W*
- *The Adjacency matrix*

The Inputs



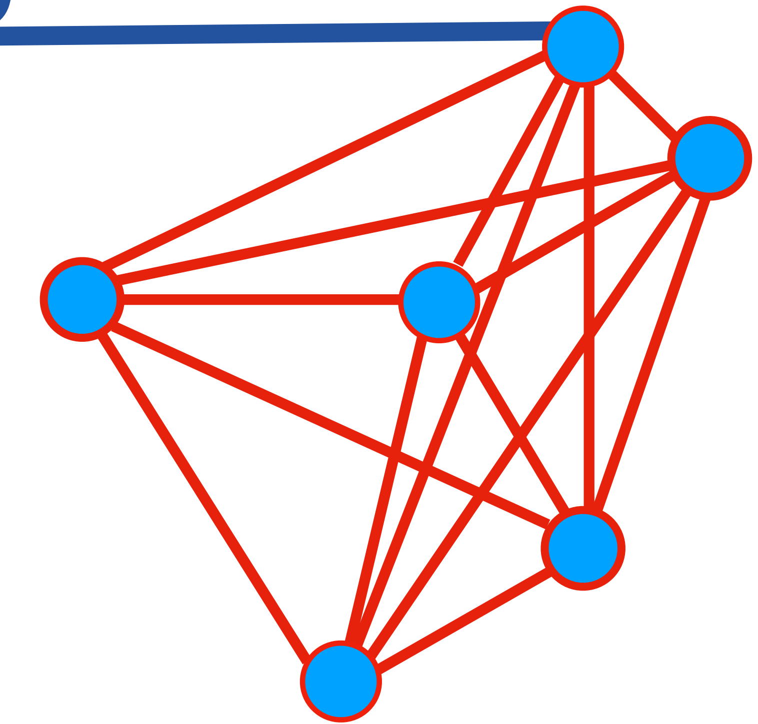
Consider a graph with n nodes, each containing f features.

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n\times n\ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n\times f\ (nodes\times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f\times c\ (feature\ weight\times channels)}$$

- *Same as all other networks*
  - *Each vertex (row) is represented as an array of features (columns)*

The Weights

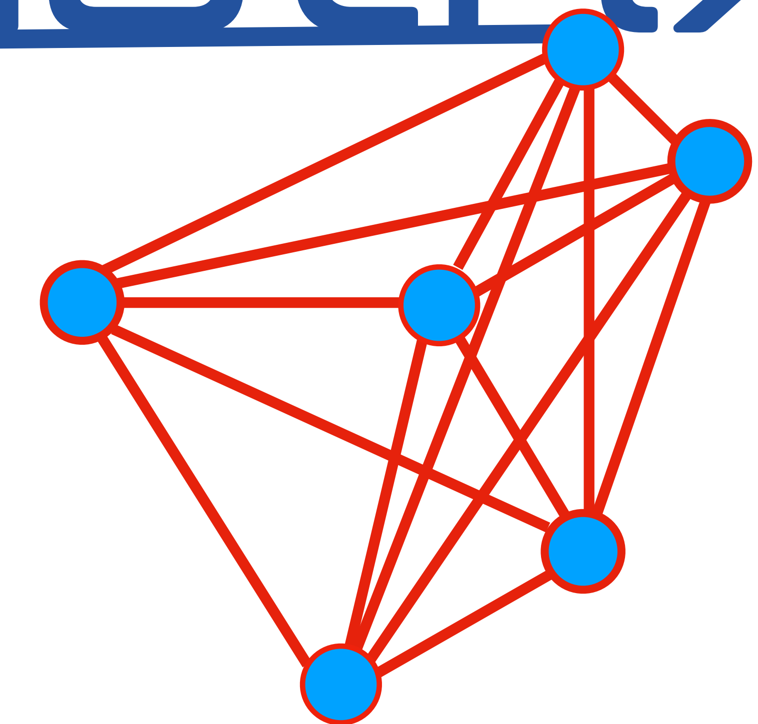
Consider a graph with n nodes, each containing f features.



$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)}, \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- The weight matrix W is used on each vertex to create new function of the inputs x (encoding)
- If wij=1, the input representations is used directly in the message passing

The Adjacency Matrix

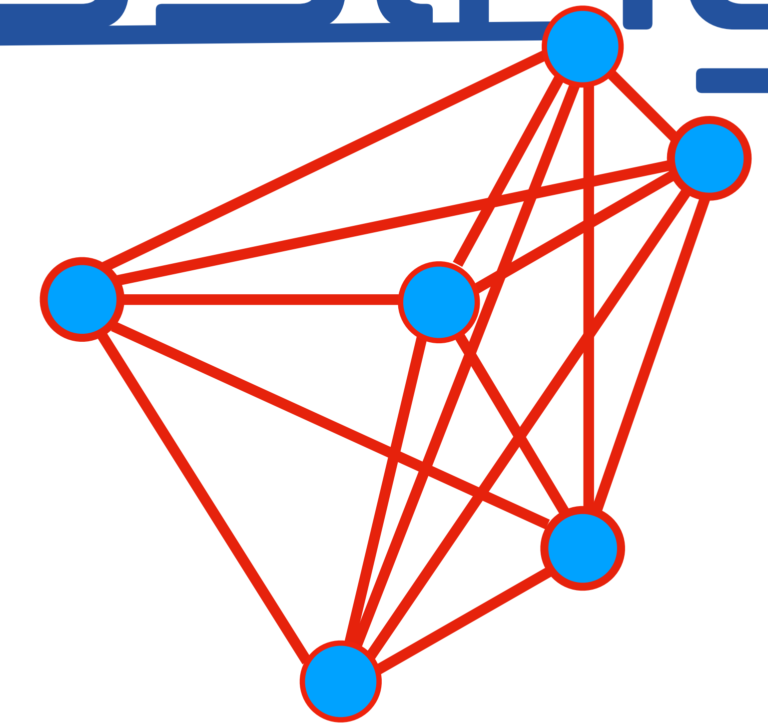


Consider a graph with n nodes, each containing f features.

$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n \times n \ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n \times f \ (nodes \times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f \times c \ (feature \ weight \times channels)}$$

- *Embeds graph structure: says which vertex is connected to which.*
- *The value could be 1 (0 for no connection) or it could be a weight*
- *Could be used with attention mechanism: the fixed weights are replaced by learnable parameters. In training, the graph decides which connections are relevant*

The Message Passing



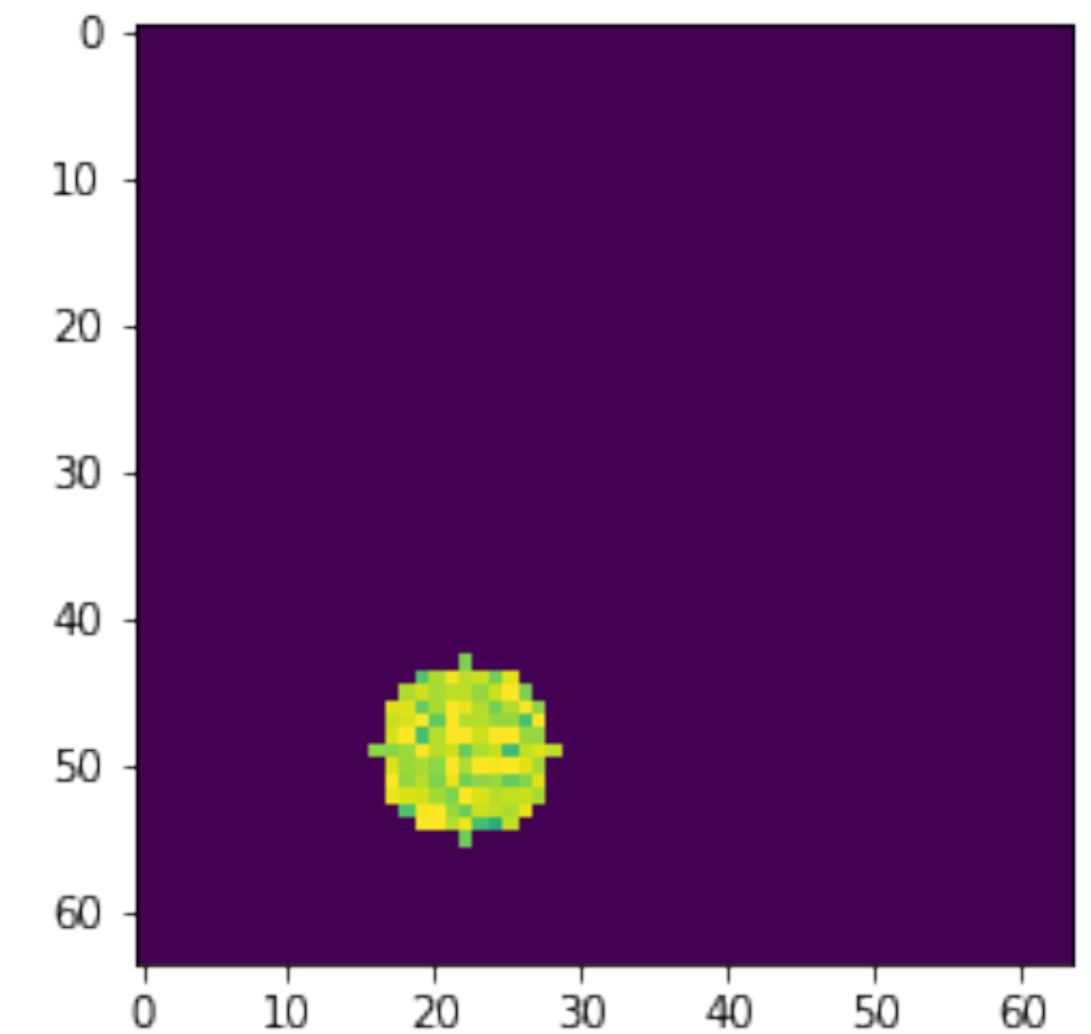
Consider a graph with n nodes, each containing f features.

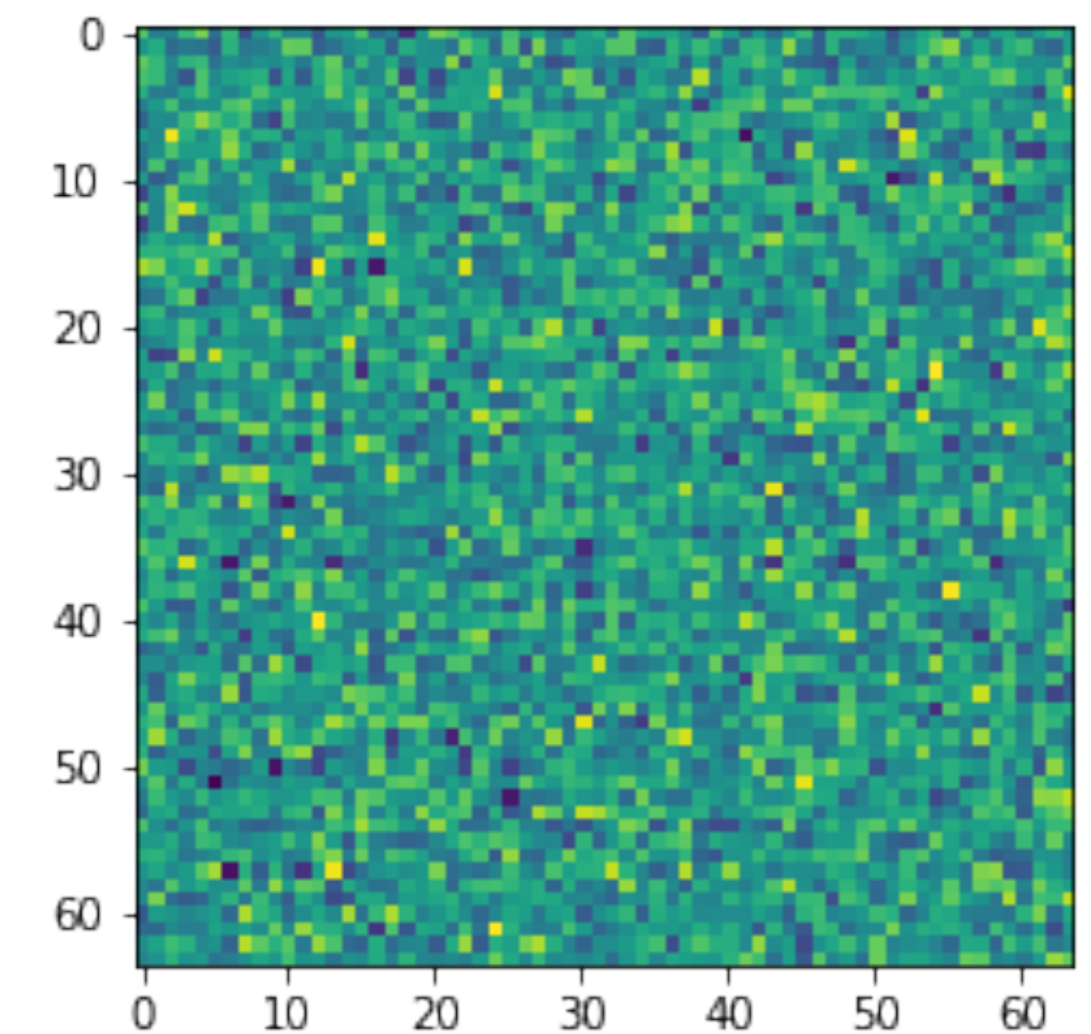
$$A.X.W = \underbrace{\begin{pmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & 0 & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & 0 \end{pmatrix}}_{n\times n\ adjacency} \underbrace{\begin{pmatrix} \vdots & x_{12} & \vdots & \vdots & \vdots \\ x_{21} & x_{22} & x_{23} & \dots & x_{2f} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & x_{n2} & \vdots & \vdots & \vdots \end{pmatrix}}_{n\times f\ (nodes\times features)} \underbrace{\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1c} \\ w_{21} & w_{22} & \dots & w_{2c} \\ \vdots & \vdots & \vdots & \vdots \\ w_{f1} & w_{f2} & \dots & w_{fc} \end{pmatrix}}_{f\times c\ (feature\ weight\times channels)}$$

- *By performing a standard matrix product, one builds the message*
- *This is for one filter. One can have multiple filters, as for CNNs*

We will use GCNs for exercise

- *Images of 64x64x1 pixels*
  - *Class 1: a sample of circles in grey scale (one channel, pixels filled with value in [0,1])*
    - *Fill pixels with Gaus(0.9,0.1)*
  - *Class 0: a noise sample, filled with G(m,0.1) (m =0.3 or 0.7)*
- *Take the 100 pixels closer to 1*
- *Represent the event as a (100,3) list, the three features being (iX, iY, GrayScale)*





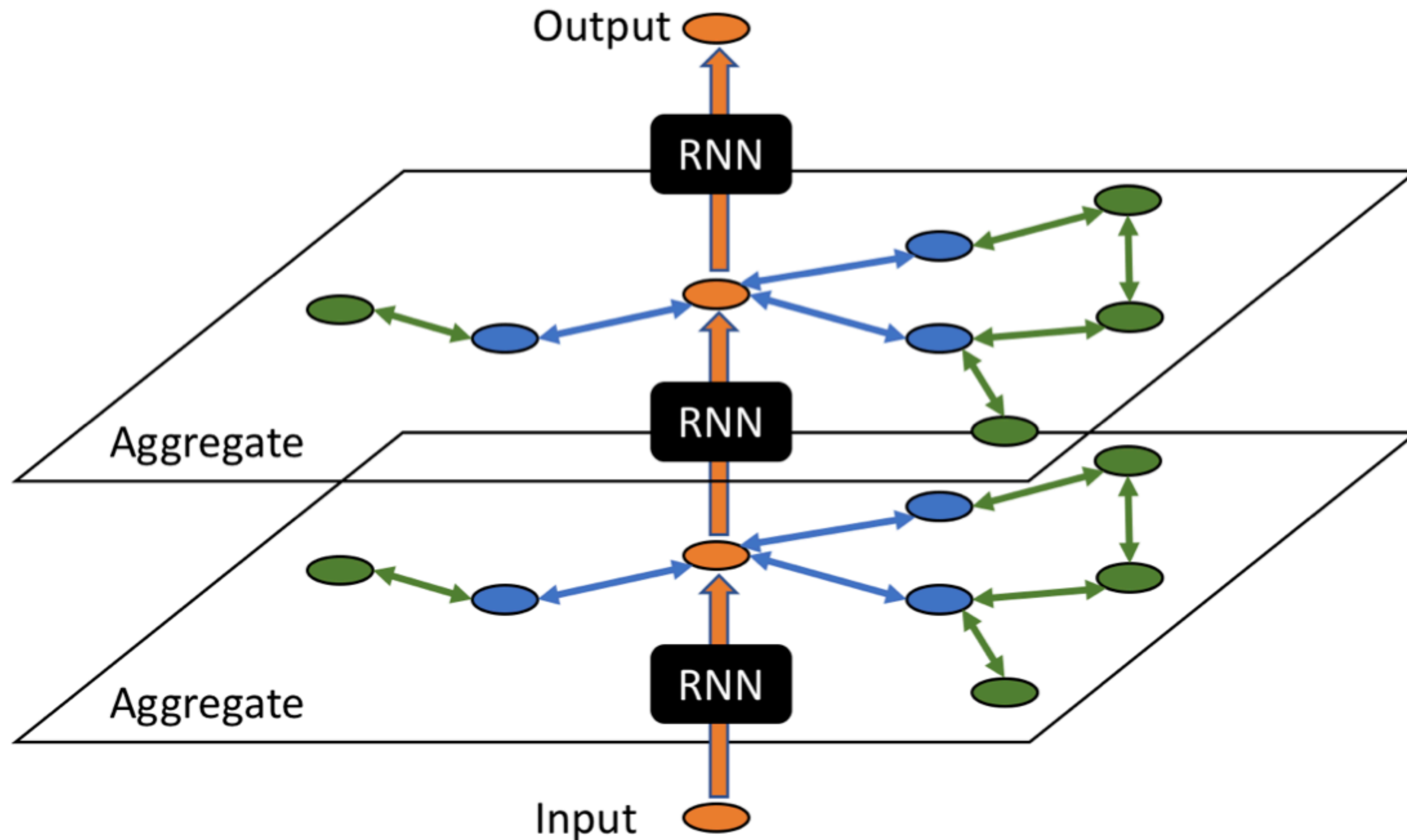
Summary

- *Graph Networks are a powerful tool to learn from sparse data sets*
  - *extend CNN concept beyond the case of geometrical proximity -> learned representation*
  - *allow to abstract from irregular geometry (molecules, particle-physics detectors, stars in a galaxy, …)*
  - *allow to inject domain knowledge in the game (e.g., enforcing physics rules for message-passing functions [Newton's law in N-body simulation]*
  - *But can also be used to learn (how to simulate) physics*

European Research Council

Backup

CERN

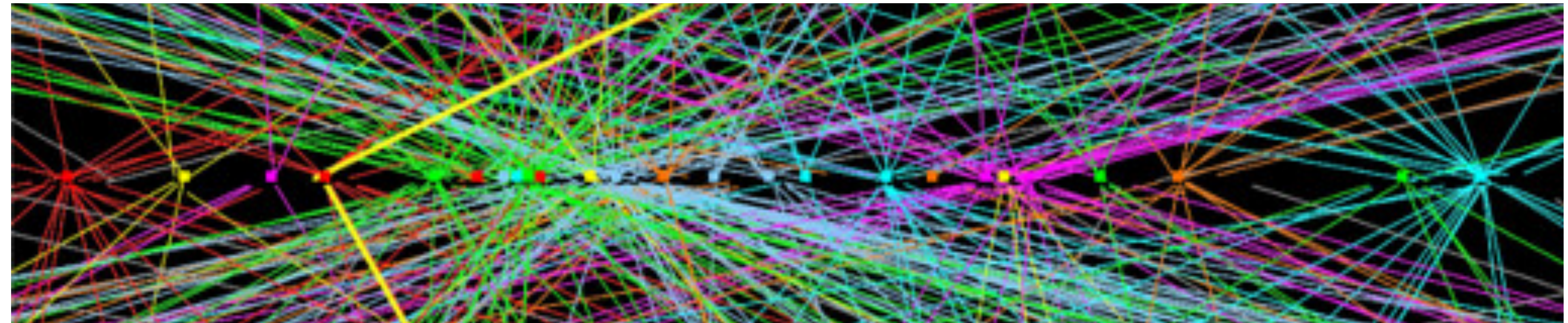


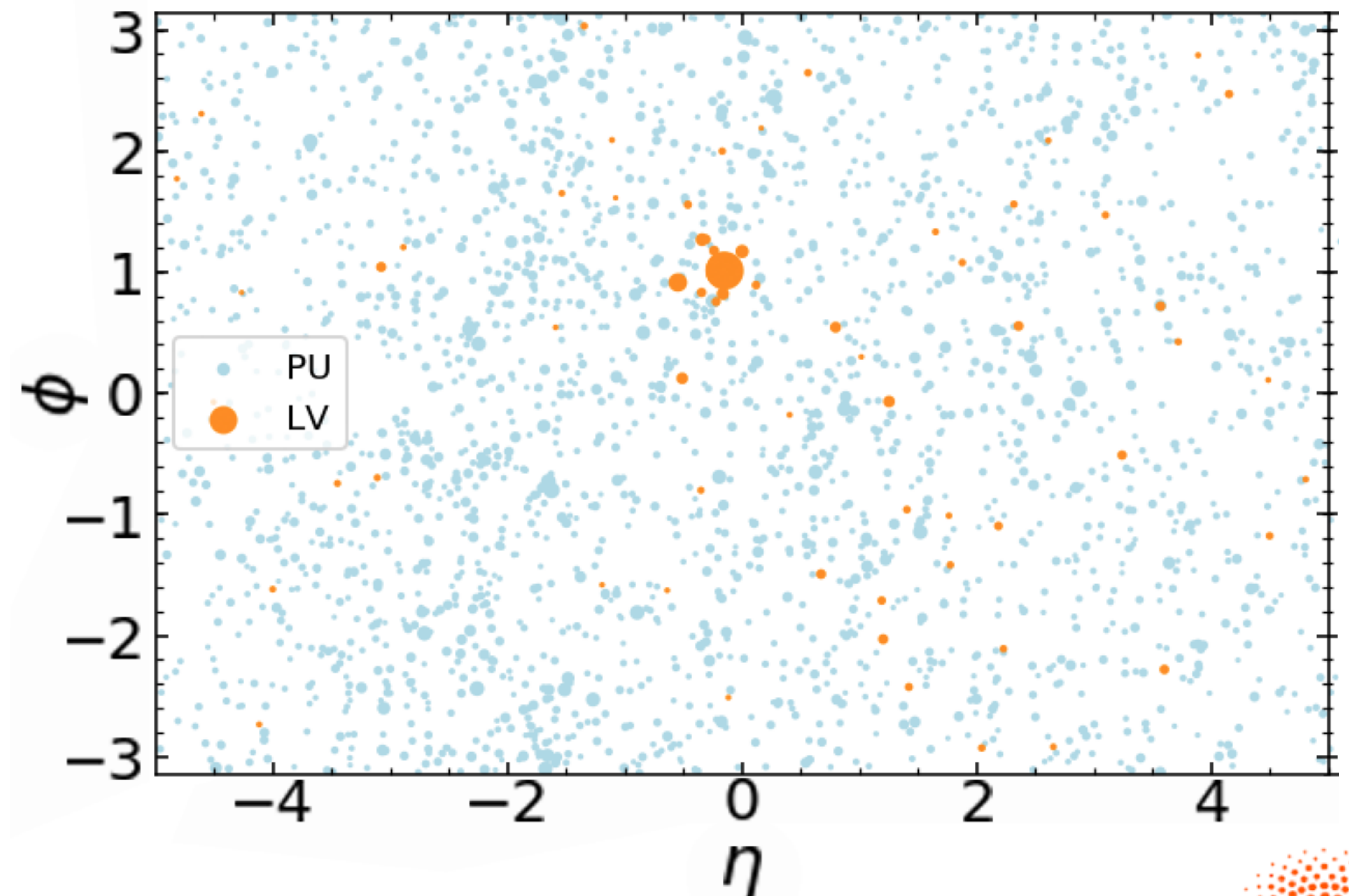
Gated Graph Neural Networks

European Research Council
erc

Example: pileup removal

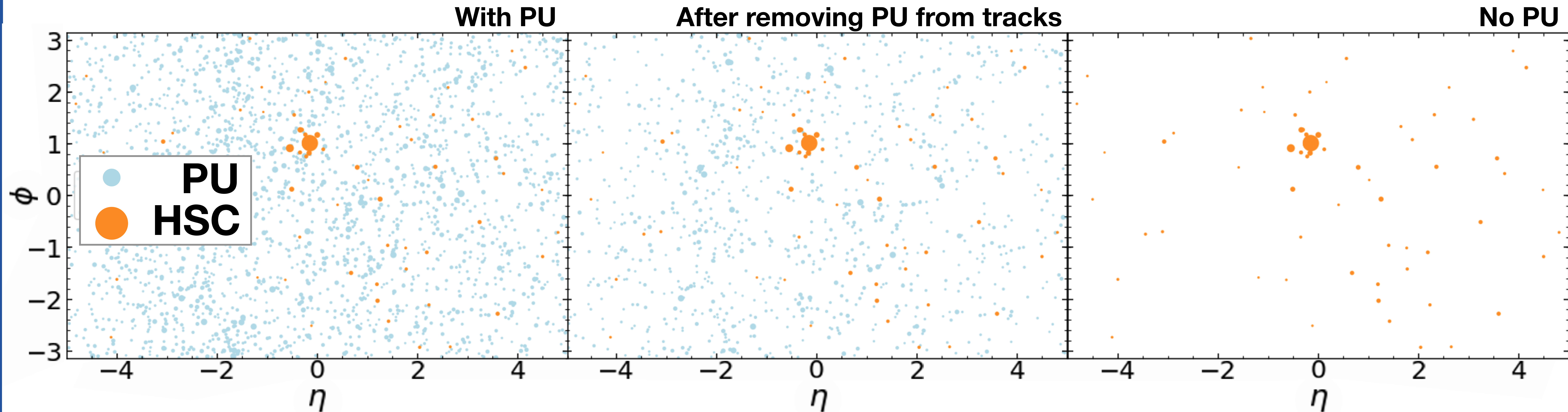
- *At the LHC, parasitic collisions (pileup) happen simultaneously to your interesting one (hard-scattering collision)*
- *They typically happen at ~ same x and y, but at different z*
- *Charged particles are tracked back to their origin and associated to the interesting collision or to a parasitic one*
- *Neutrals cannot be tracked back*





European Research Council

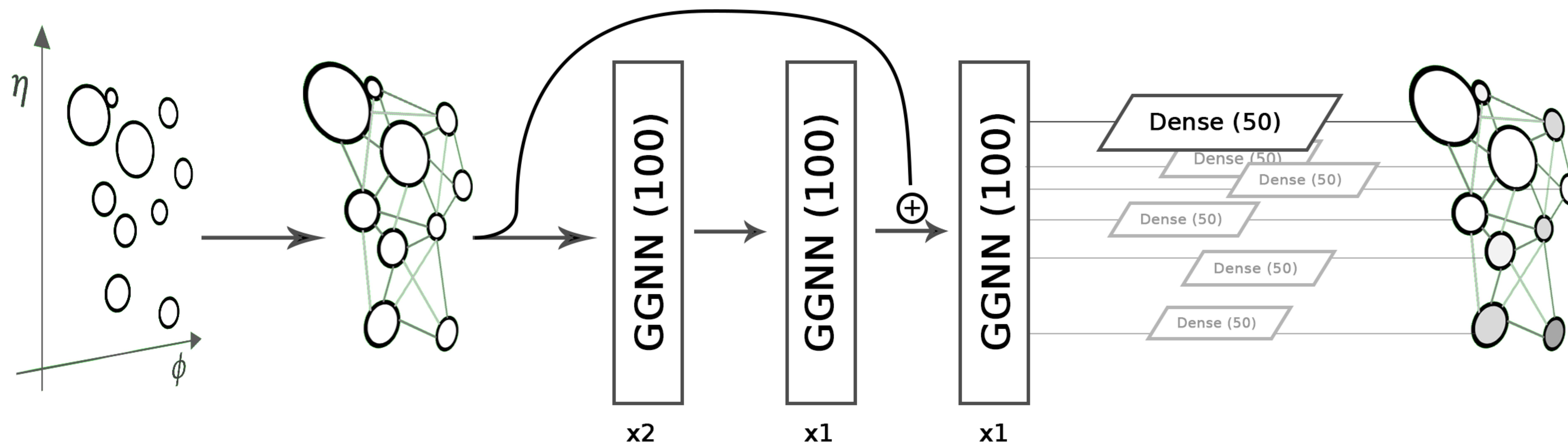
Example: pileup removal



- We want to learn if a given neutral comes from PU of from the hard-scattering collision
- We look at the problem projecting the solid-angle particle distribution in a plane (we unroll the cylindrical detector into a rectangle)
- Each point is a particle (size represents the energy carried by each particle)
- We know the particle charge, plus some extra feature we might want to use (e.g., particle kind [electron, muon, etc.])
- We have labels for charged particles, but not neutrals

European Research Council

Graph Networks for PU removal

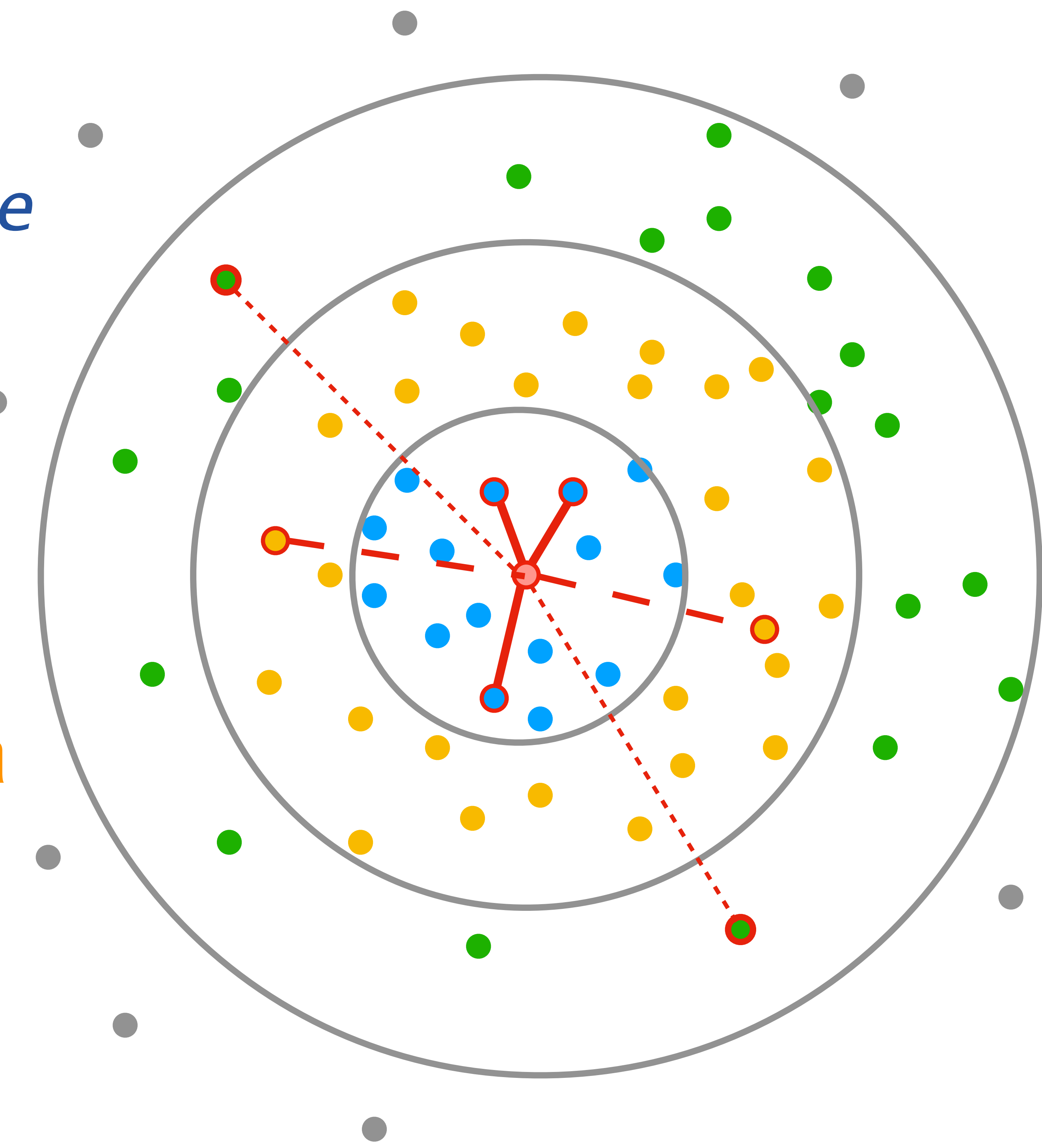


- *We use Gated Graph Neural Network (GGNN), a special kind of message-passing architecture*
  - *Start from a set of particles, each represented as a set of features h*
  - *Build the graph*
  - *Use a recurrent network (GRU) to "pass" messages in sequential steps and evolve the particle representation*
  - *At each iteration, we connect to different neighbours (start with close-by, then go further)*

European Research Council

Building the graph

- *Start with one particle (the red one)*
  - *Connect it to the closest ones (R<R0) with one kind of edge*
  - *Connect it to the next-to-closest ones with a different kind of edge*
  - …
- *Each edge comes with a message*



A = Adjacency matrix, a learnable matrix, different for different kinds of edges

$$m_j = \frac{1}{N}\sum_j^N m_{v,v_j} = \sum_j A \times f_j$$

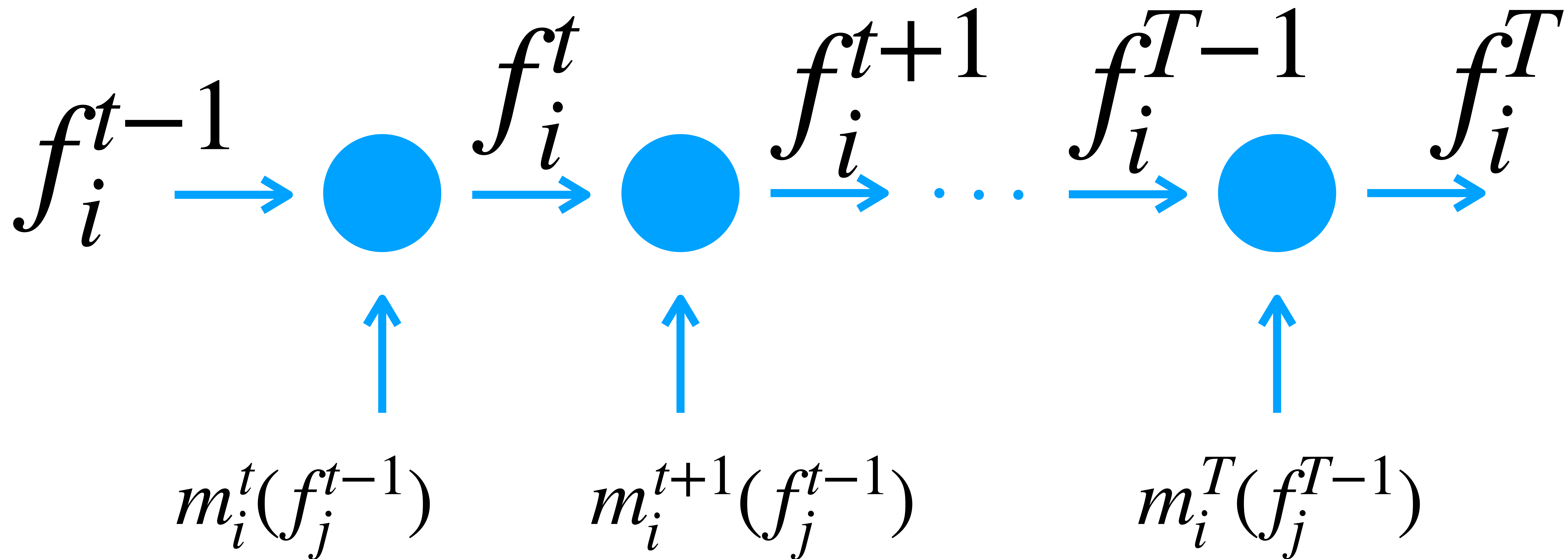
Gathered message (m_j)

Messages (m_{v,v_j})

features of the j-th vertex (f_j)

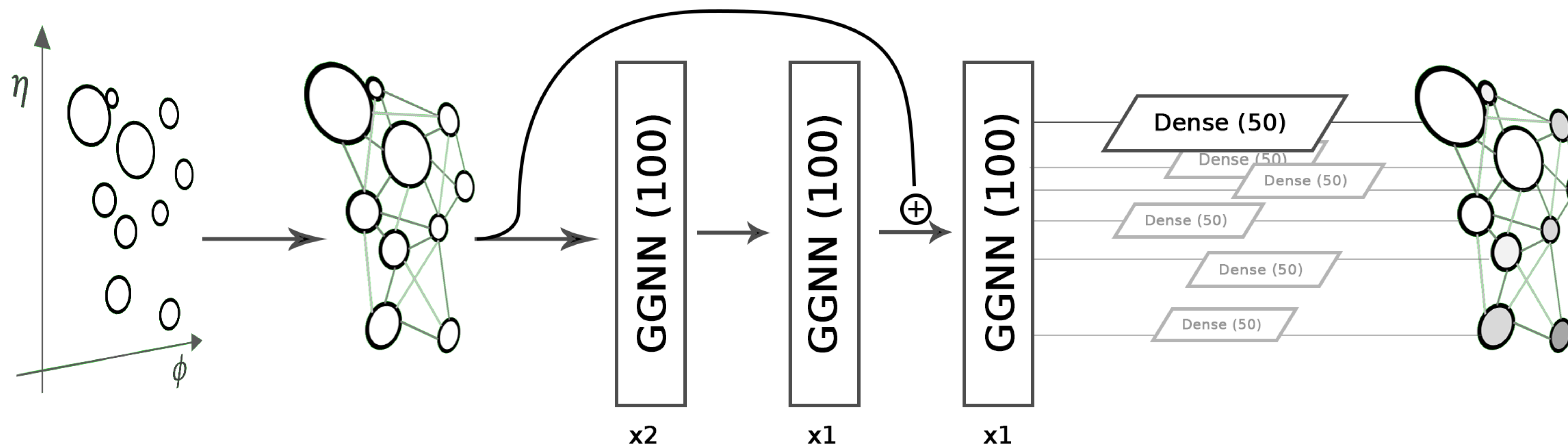
Multiple message passing

- *The procedure is repeated T times using a GRU with n steps*



European Research Council

Done 3 times (for 3 GRU layers)



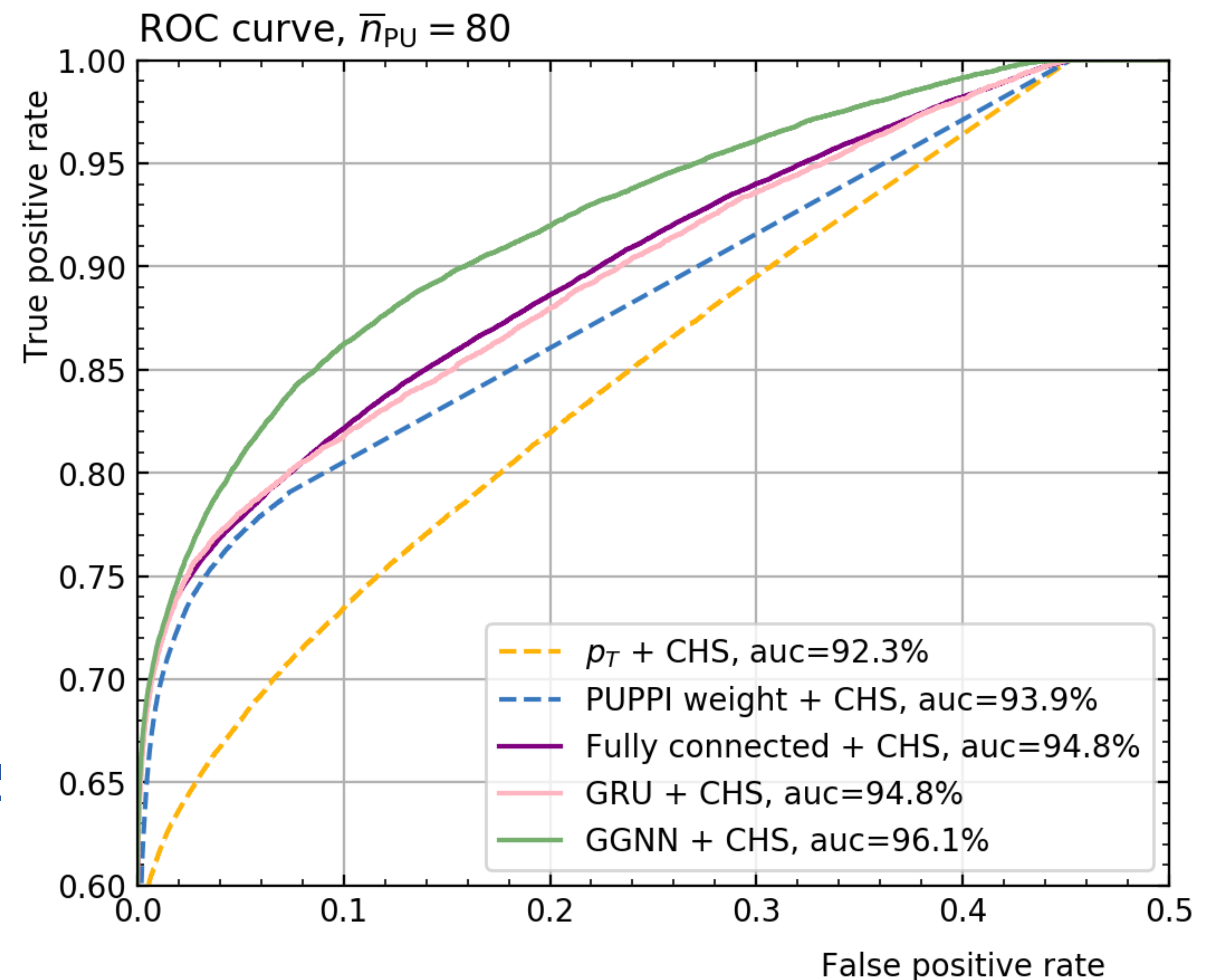
- *The representation created by each layer is passed to the next*
- *A ResNet-like skip connection is implemented (input -> last layer)*
- *The per-particle outcome set of features (function of the features of the connected particles) is used to train a dense classifier: PU vs interesting particle?*

European Research Council

PUPPIML: Graph Nets for PU subtraction

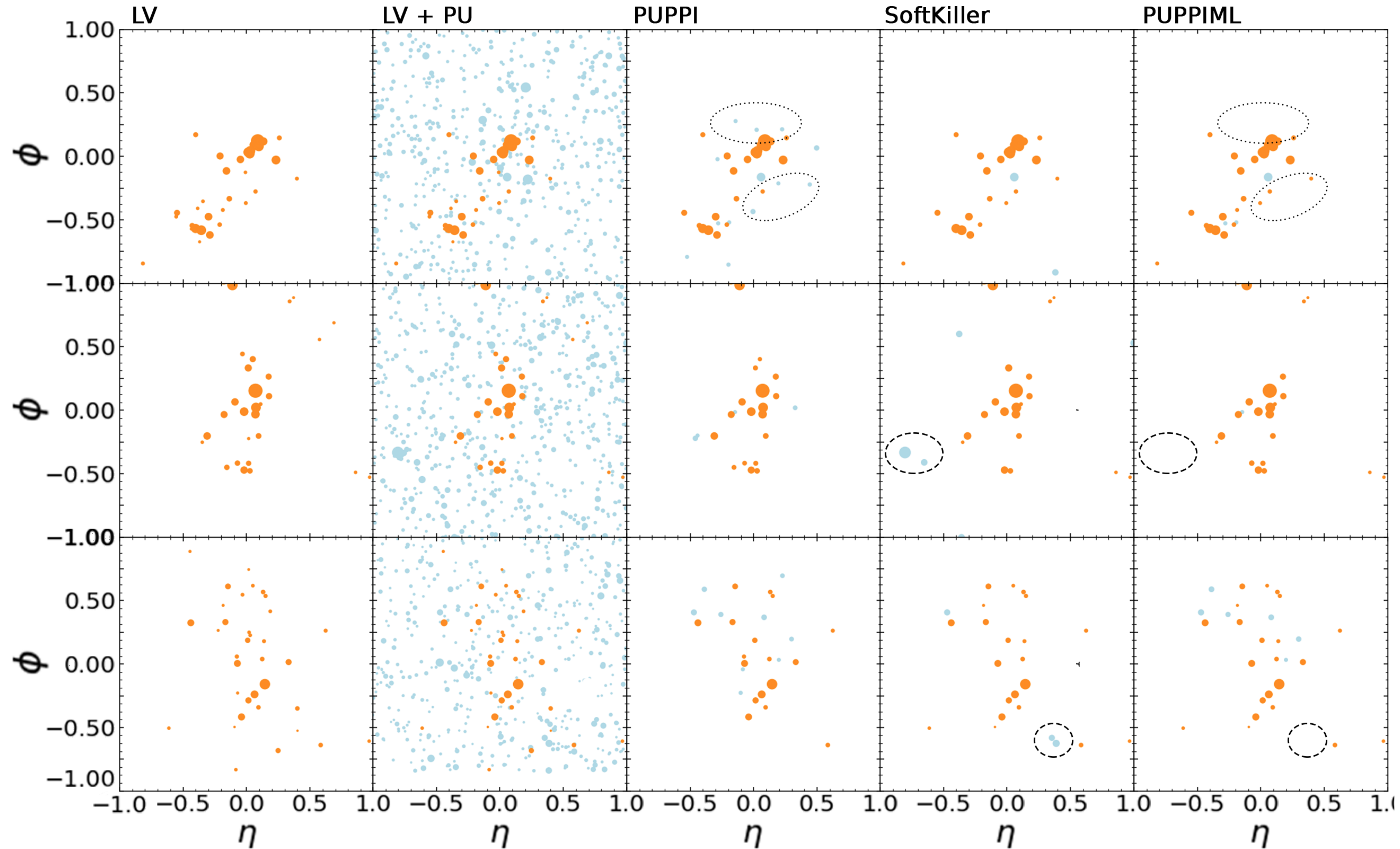
- *Improve state-of-the-art algorithms substantially*
- *Little dependence of algorithm tuning on pileup conditions*
- *Small/No performance loss with average number of PU collisions*
- *Outperforms alternative particle-based architectures (DNNN, simple GRU)*

$\overline{n}_{\text{PU}}$	20 (CHS)	80 (CHS)	140 (CHS)	80 (No CHS)
p_T	92.3%	92.3%	92.5%	64.9%
PUPPI weight	94.1%	93.9%	94.4%	65.1%
Fully-connected	95.0%	94.8%	94.8%	68.5%
GRU	94.8%	94.8%	94.7%	68.8%
GGNN	**96.1%**	**96.1%**	**96.0%**	**70.1%**



https://arxiv.org/pdf/1810.07988.pdf

PUPPIML: Graph Nets for PU subtraction



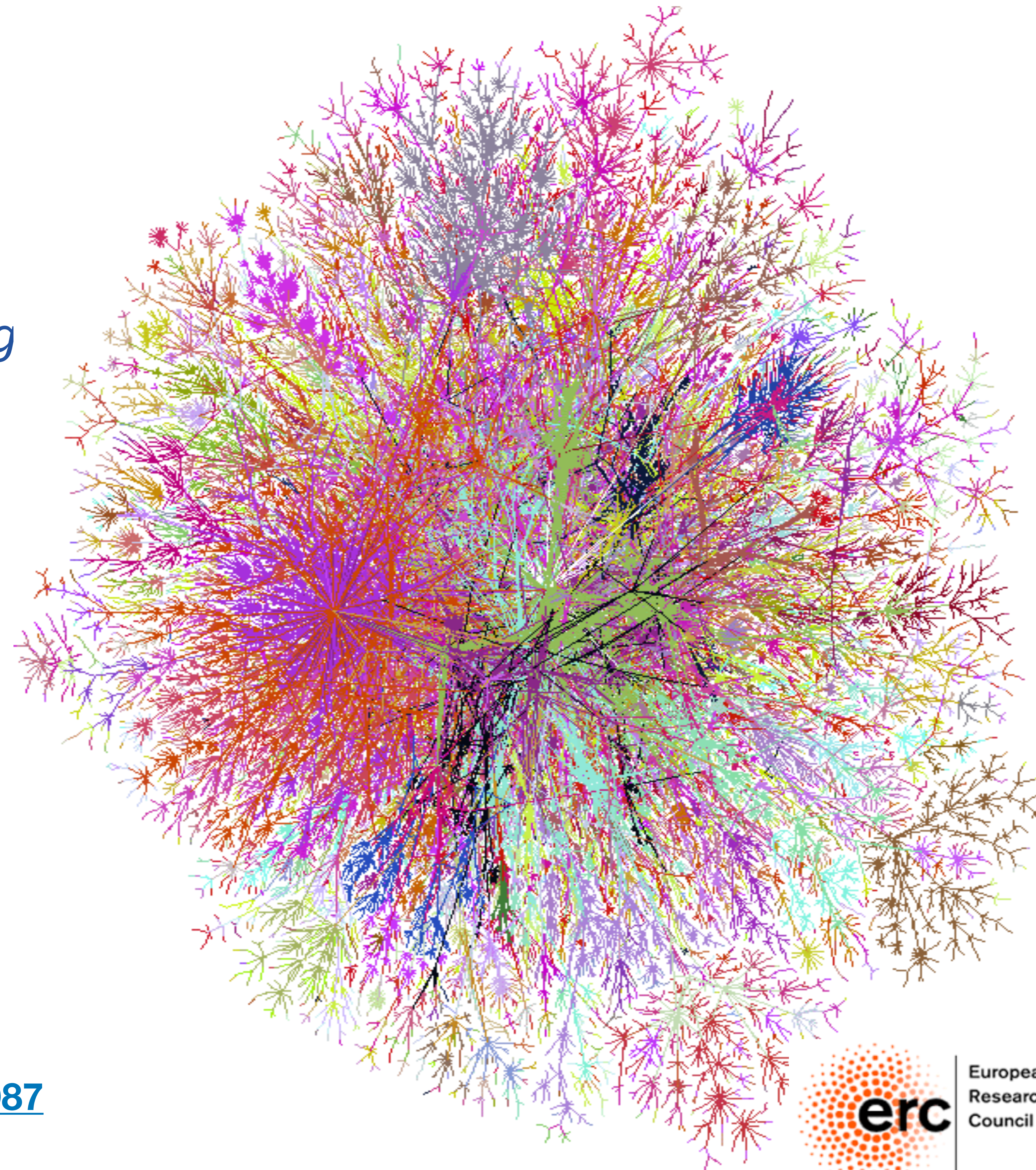
CERN

Distance-Weighted Graph Networks

erc European Research Council

Reducing memory consumption

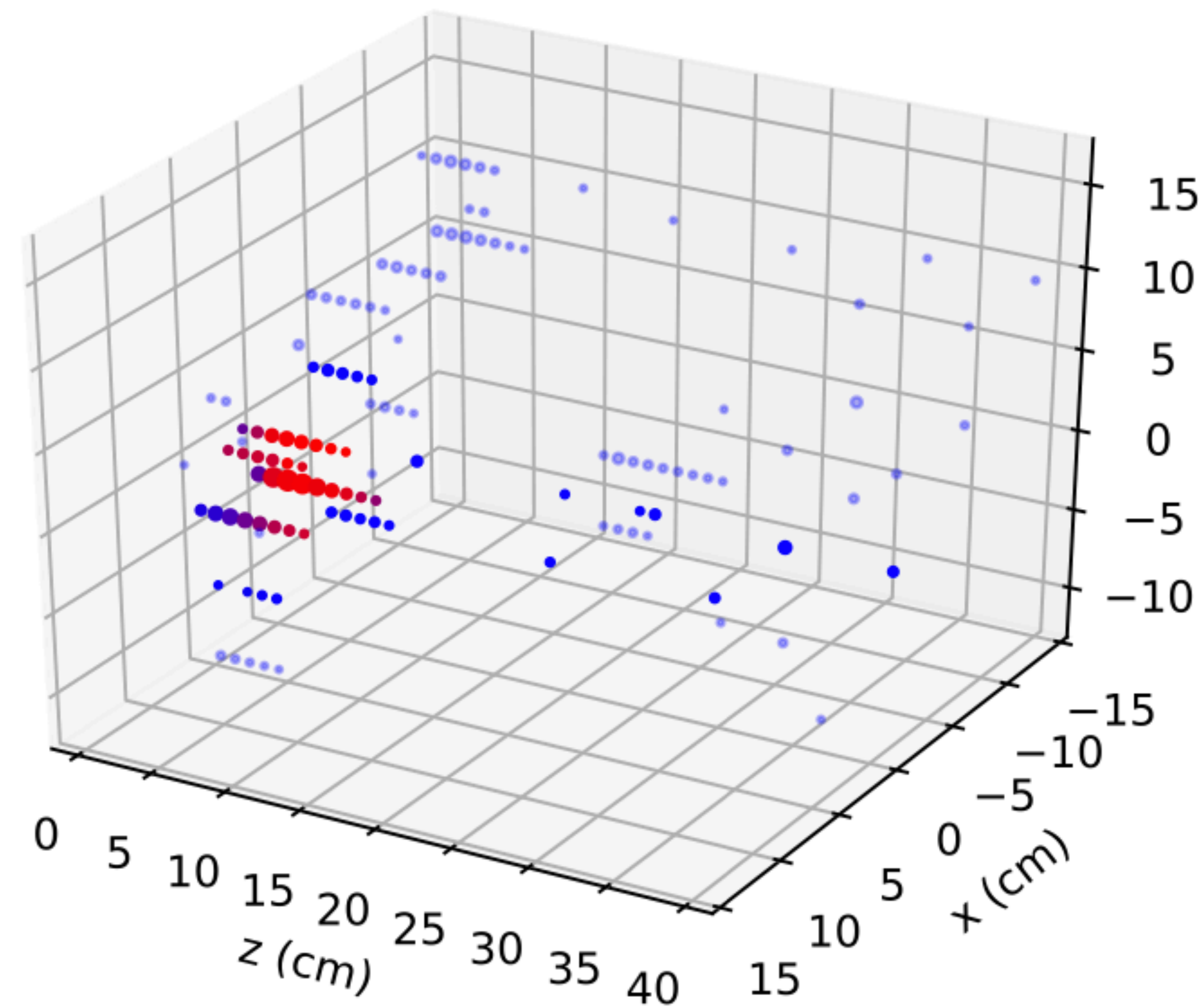
- *When building a graph of N vertices, number of edges (and number of computing operations) scale with N^2*
- *This might clash with computing resource limitations (both for training and inference)*
- *Certainly, this is the case at the LHC*
  - *real-time event selection runs in short time*
  - *most of the selection runs as electronic circuit on electronic board*
- *Gravnet & Garnet: resource friendly graph architectures* https://arxiv.org/abs/1902.07987

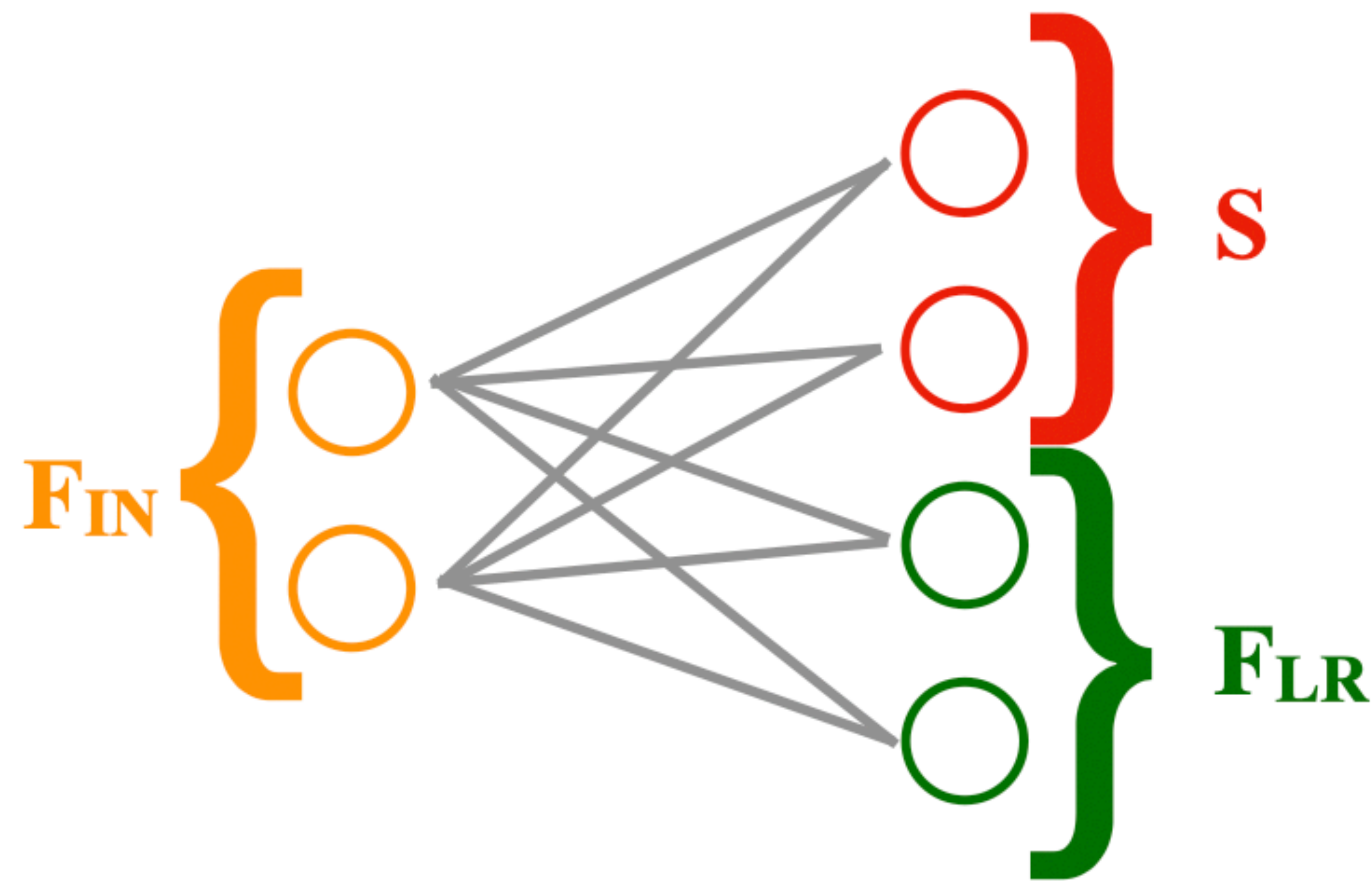


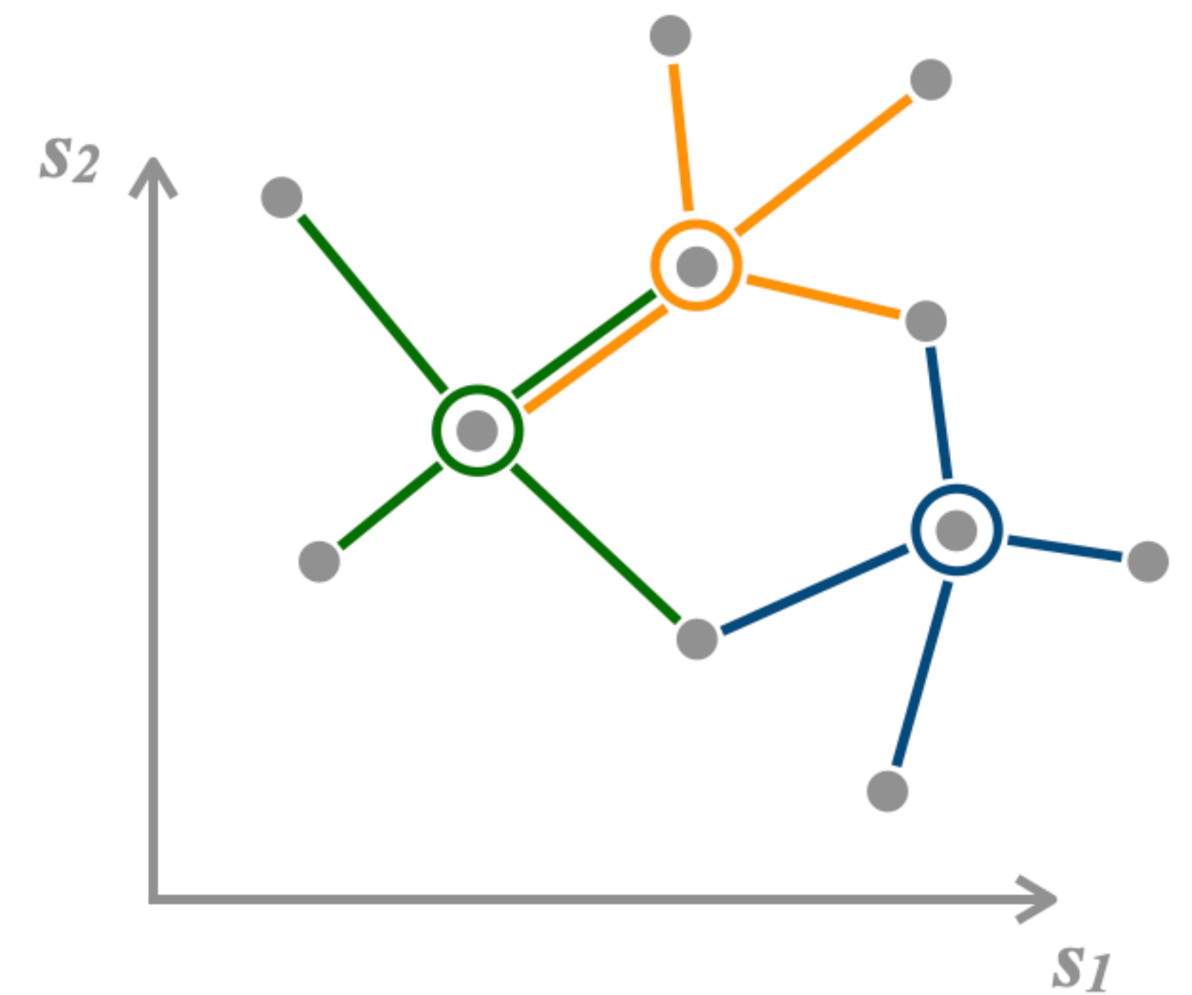
European Research Council

GravNet

https://arxiv.org/abs/1902.07987







1) Start with a graph in geometric space. Each vertex feature vector F_{IN} is characterized by coordinates and features

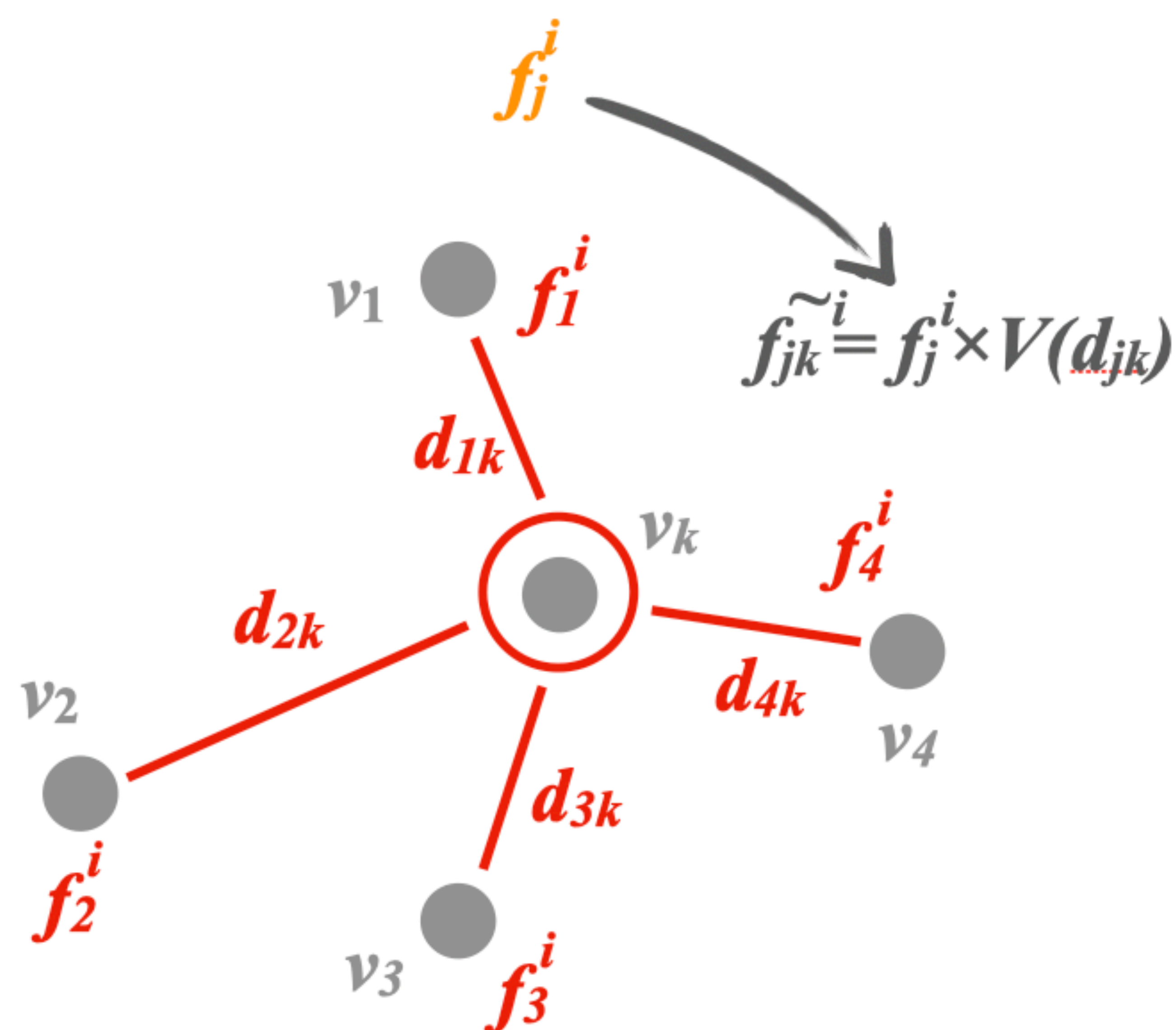
2) Each F_{IN} is processed by a linear network, returning two outputs: a coordinate vector s & a learned representation F_{LR}

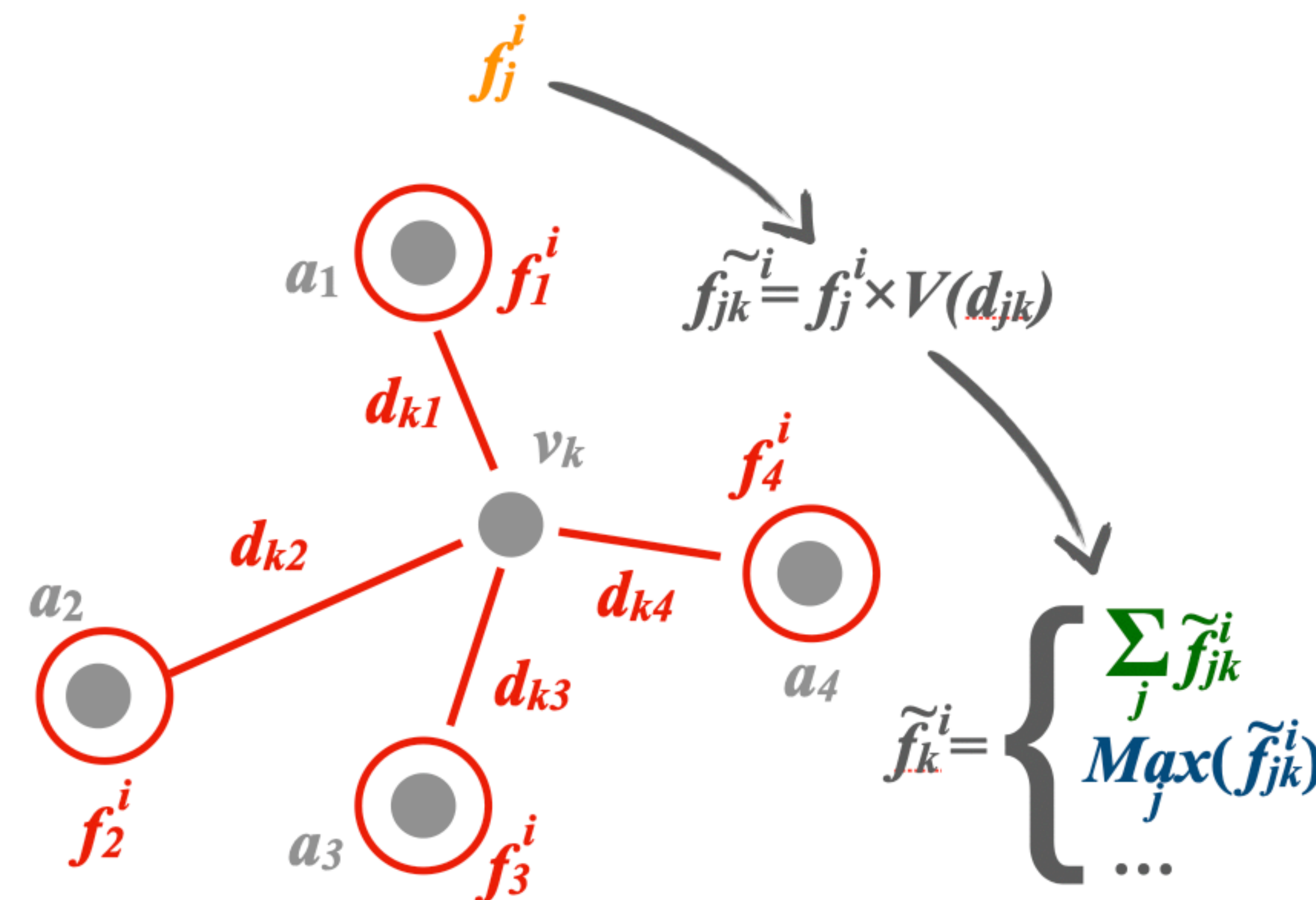
3) With s and F_{LR} we build the new graph in the learned space

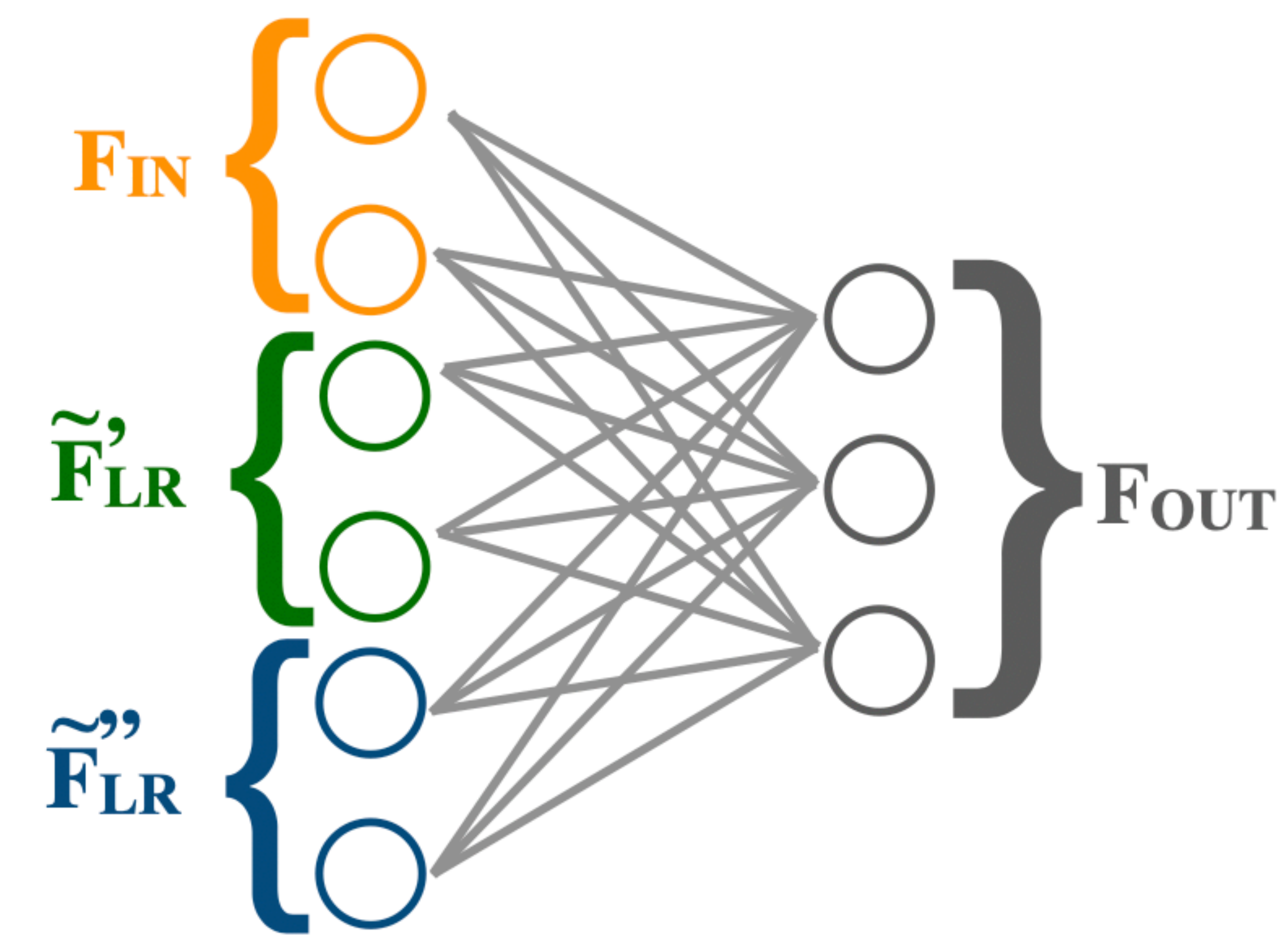
European Research Council

GravNet

https://arxiv.org/abs/1902.07987







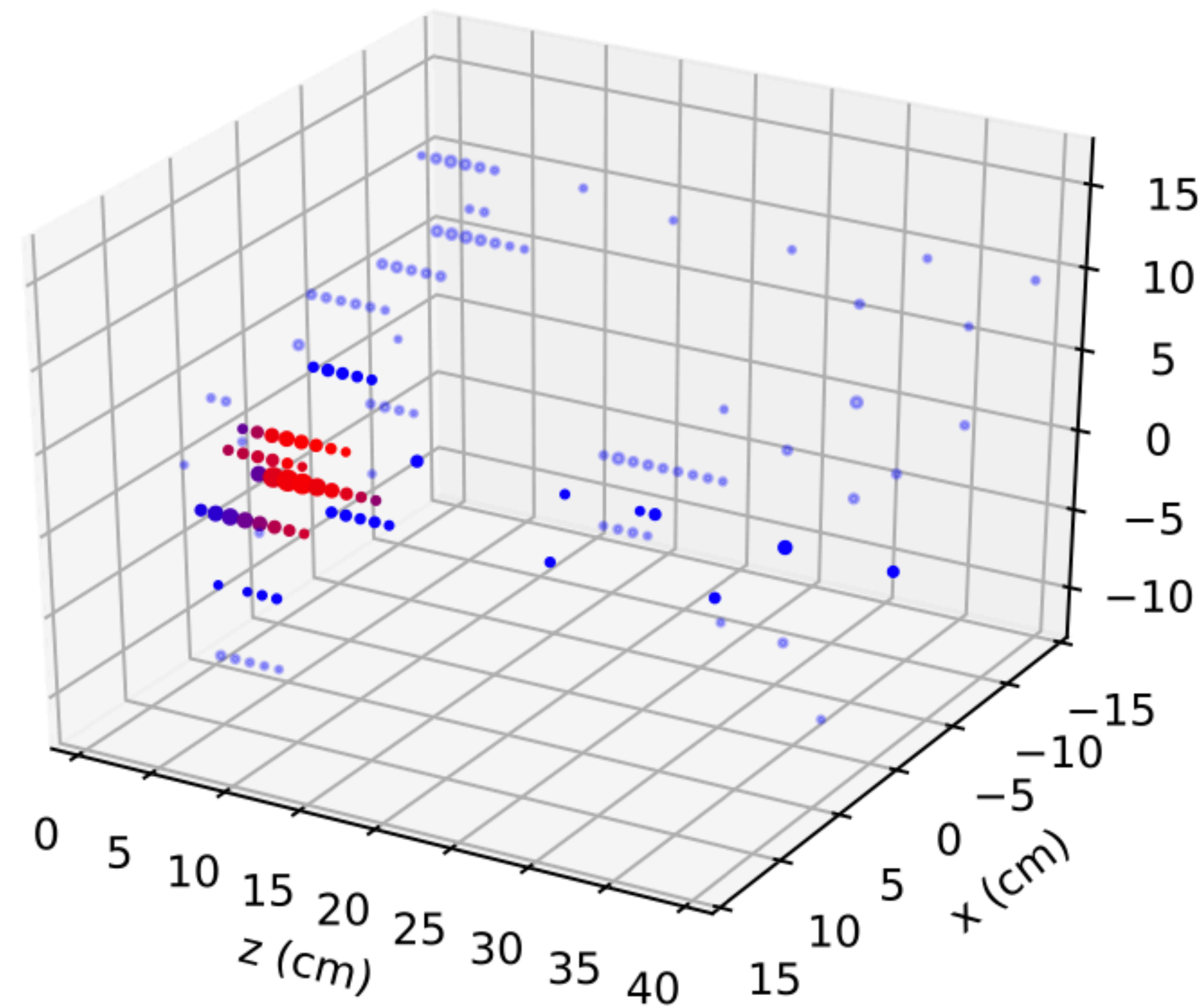
4) Unlike DGCNN, the message function is a potential function (we use e^{-d^2} where d is the Euclidean distance in learned space)

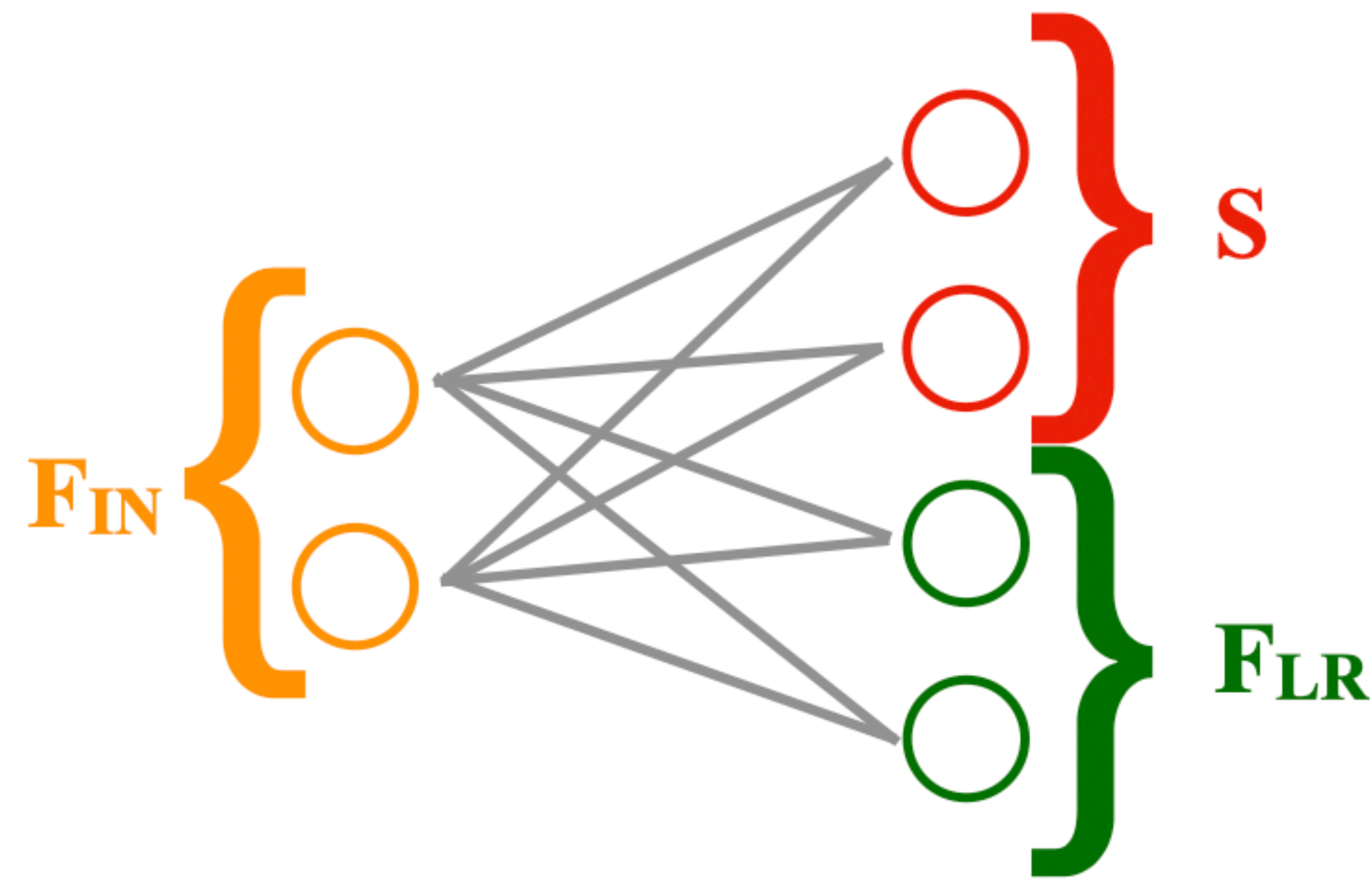
5) Message aggregated with different functions (Max, Average,…)

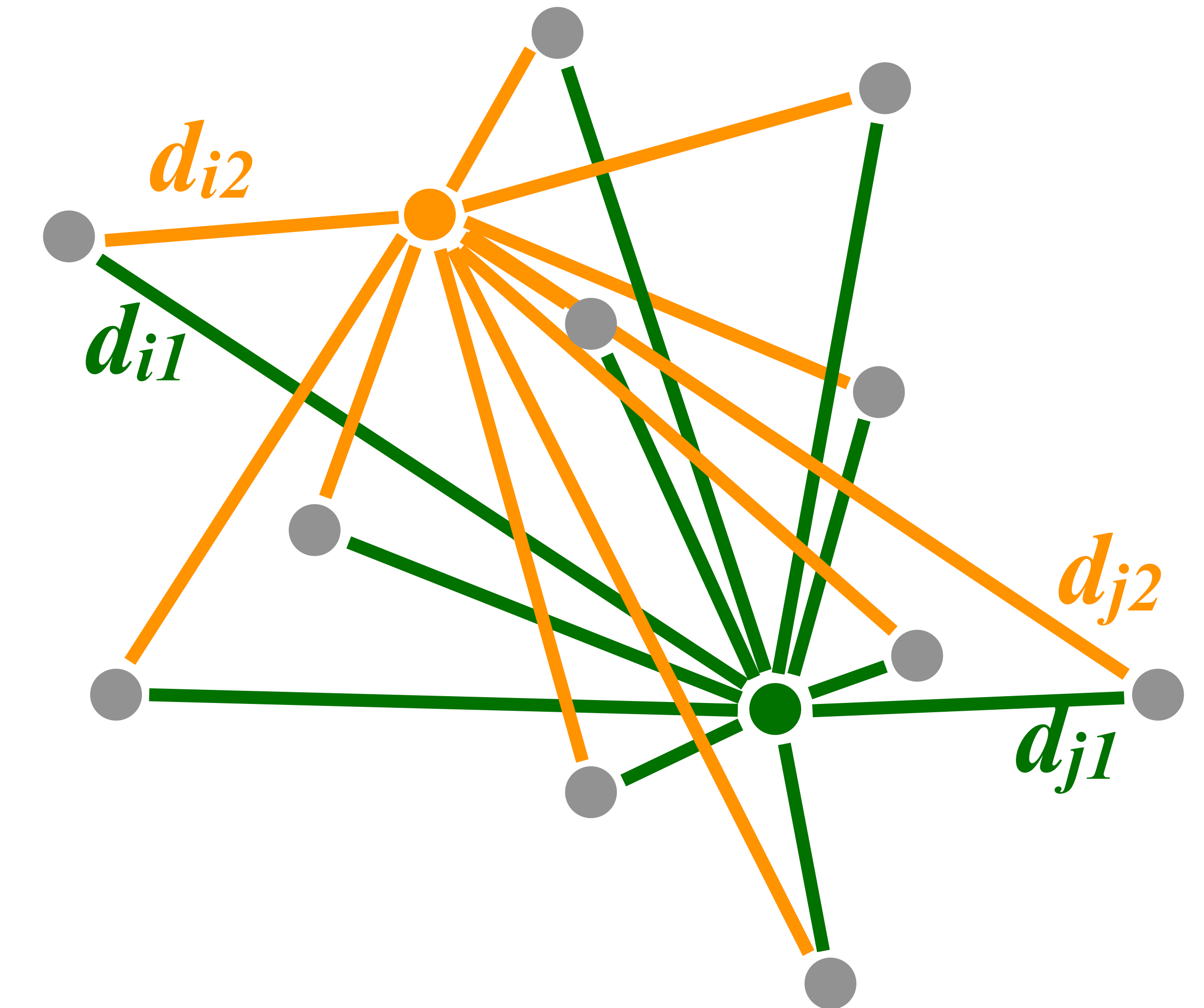
6) Final representation is learned from the engineered features and the original ones

European Research Council

(simplified) GarNet







1) Start with a graph in geometric space. Each vertex feature vector F_{IN} is characterized by coordinates and features

2) Each F_{IN} is processed by a linear network, returning two outputs: a vector of distances s & a learned representation F_{LR}

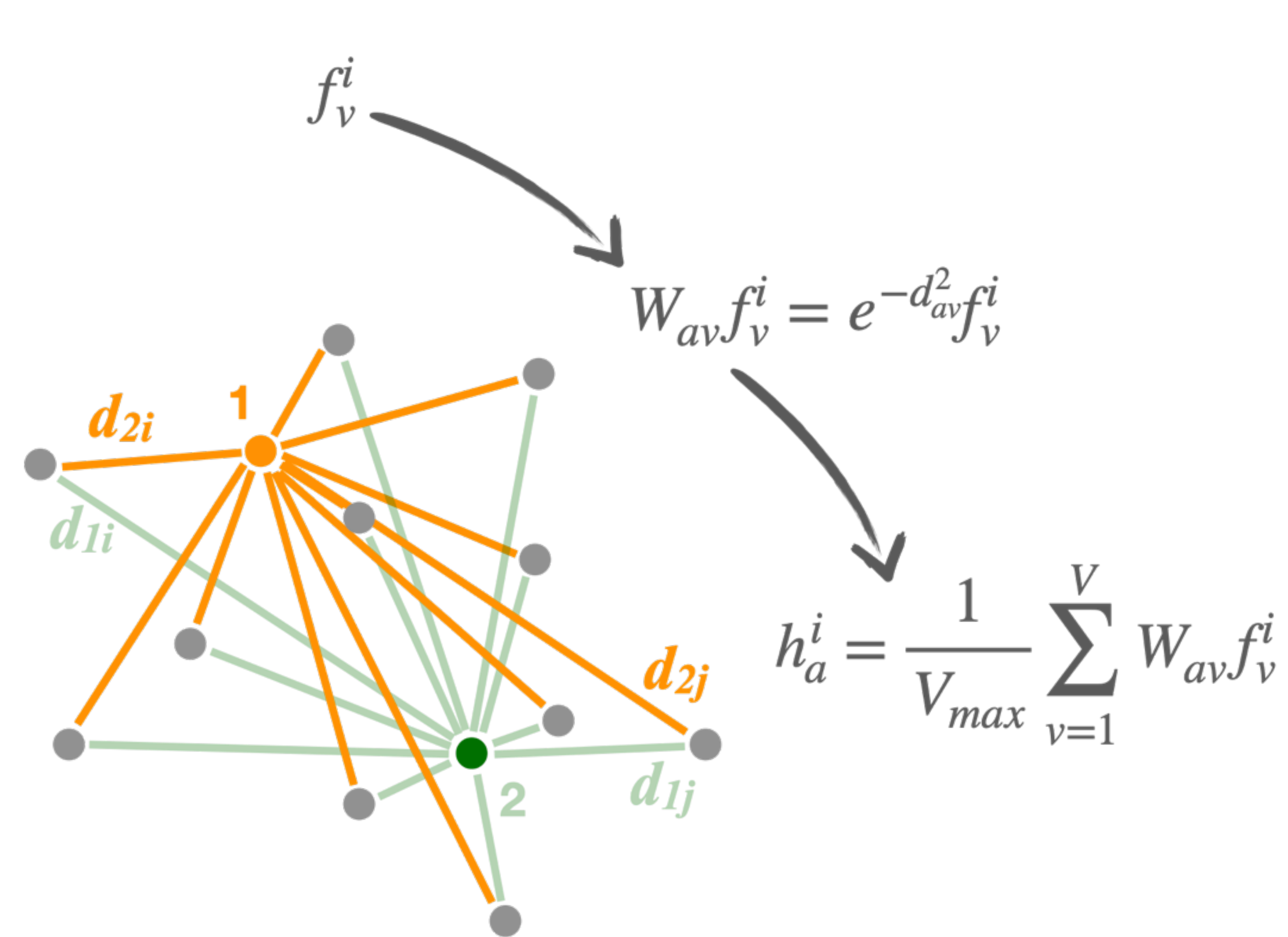
3) s are the distances from Ns aggregators

https://arxiv.org/abs/1902.07987

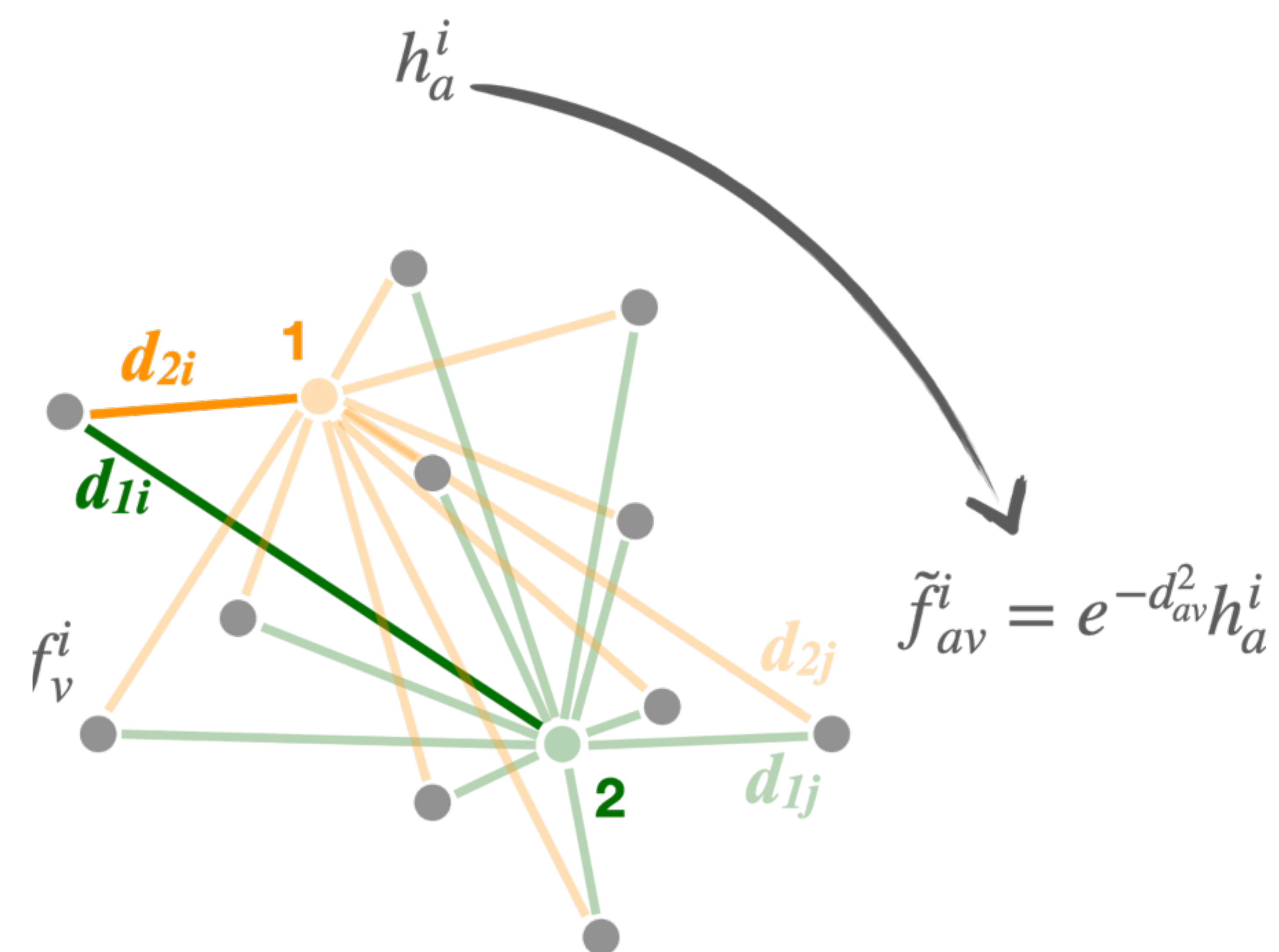
https://arxiv.org/pdf/2008.03601.pdf

CERN
erc European Research Council

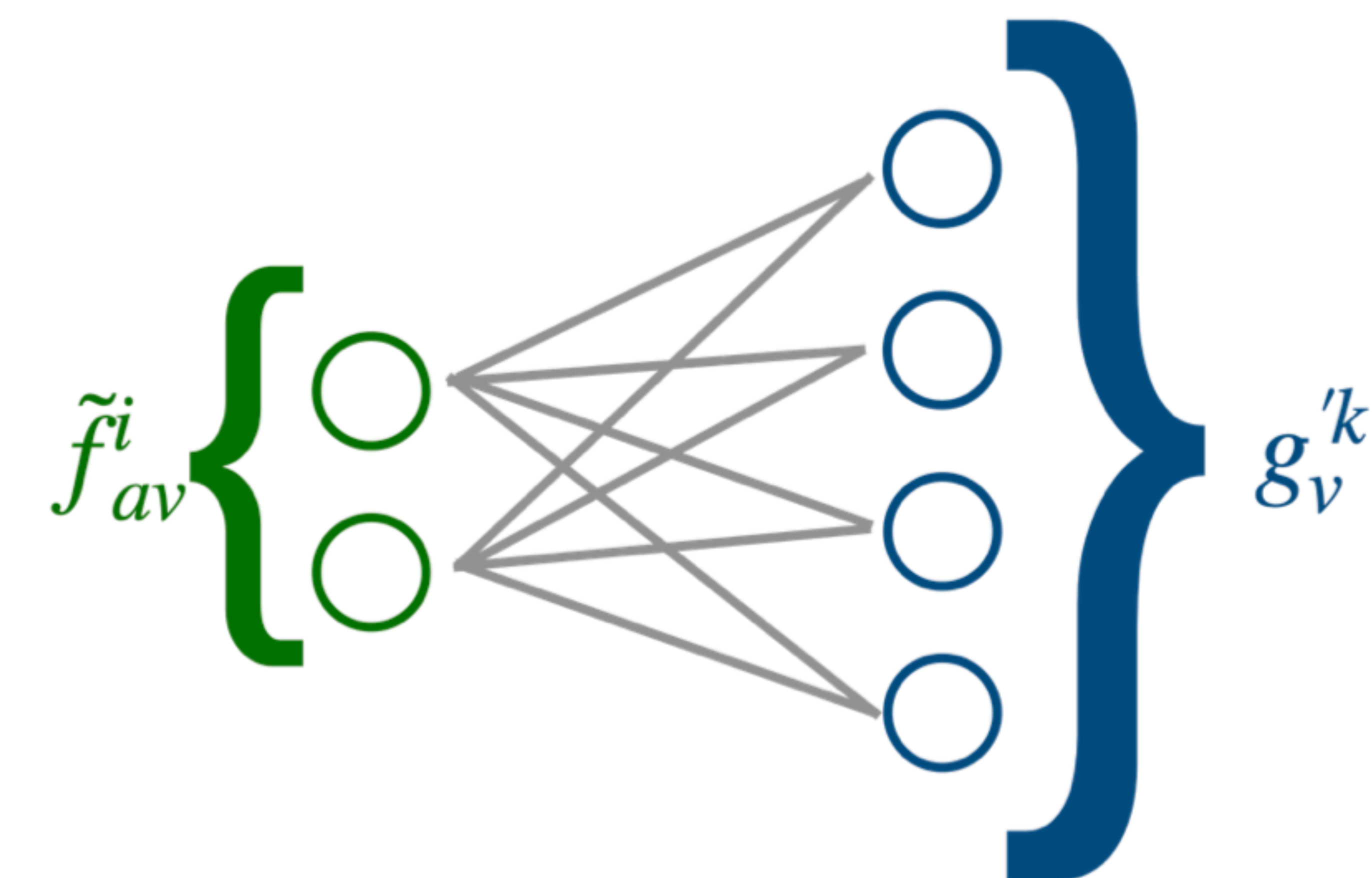
(simplified) GarNet



4) Fwd distance-weighted messages from vertices are gathered at aggregators (weight $W_{ab} = e^{-d_{ab}}$ where d is Euclidean distance in learned space)



5) Bkw distance-weighted messages from aggregators are gathered at vertices (weight $W_{ab} = e^{-d_{ab}}$)



6) Final representation is learned from the engineered features and the original ones

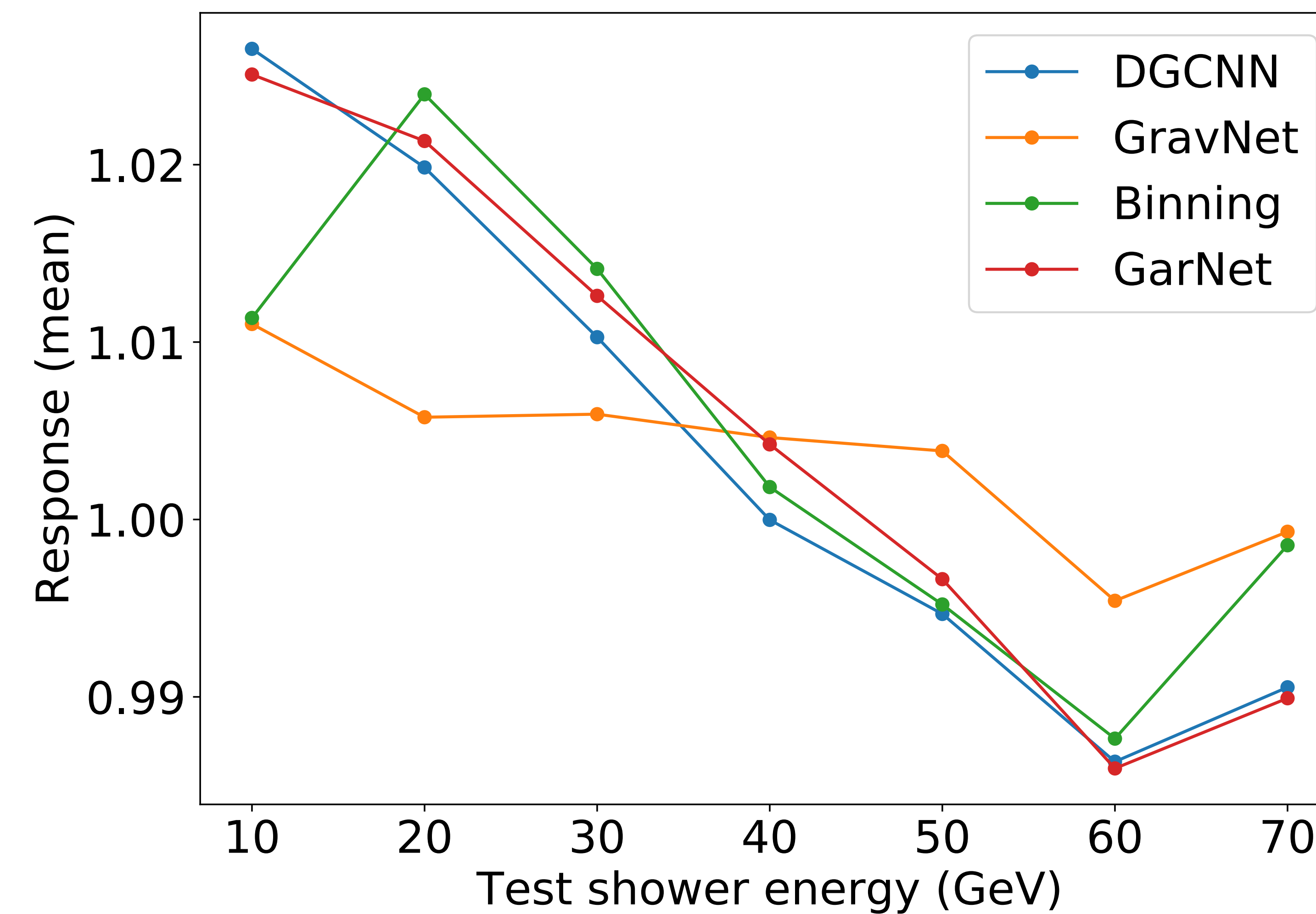
https://arxiv.org/abs/1902.07987

https://arxiv.org/pdf/2008.03601.pdf

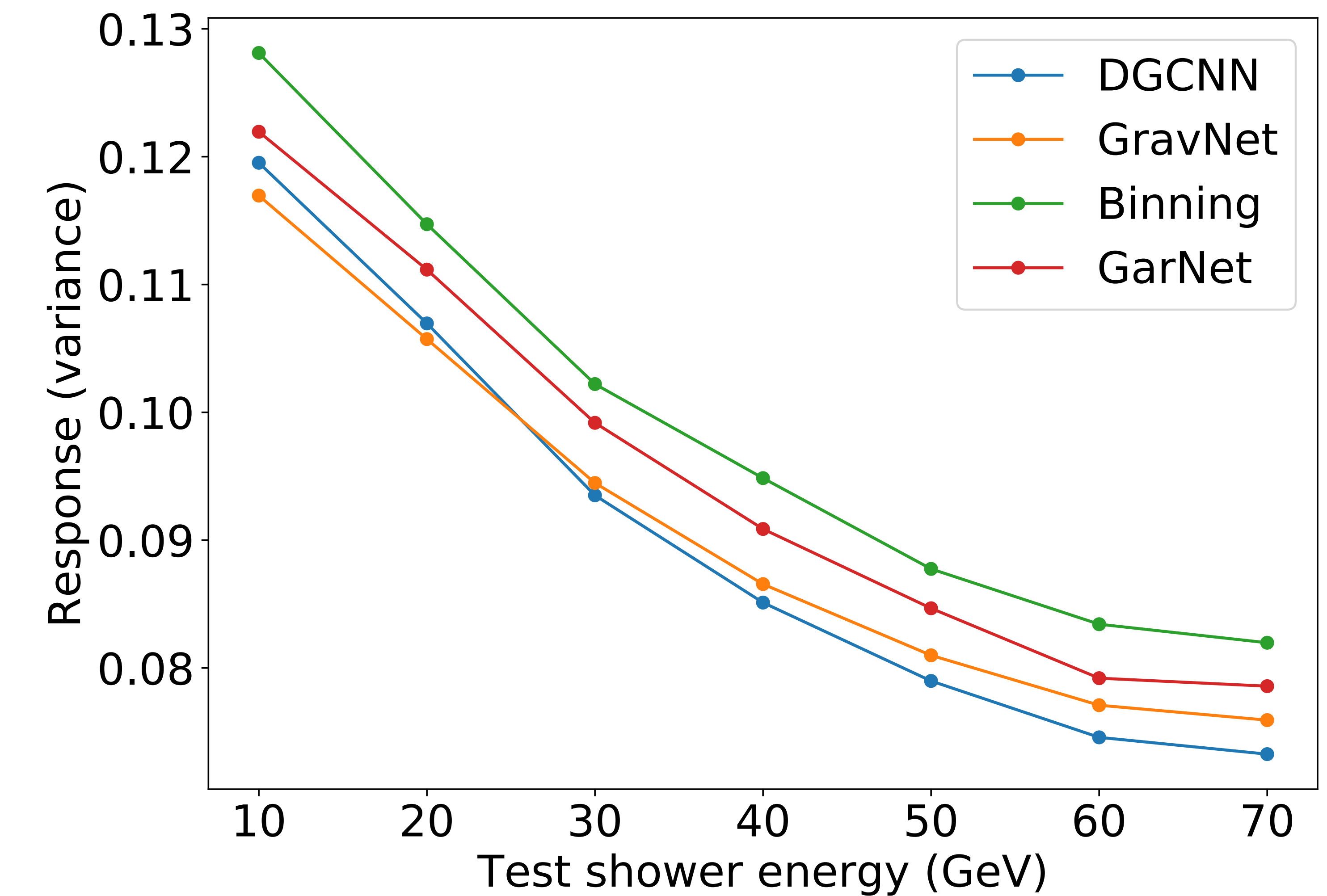
European Research Council

GarNet & GravNet for Calorimetry

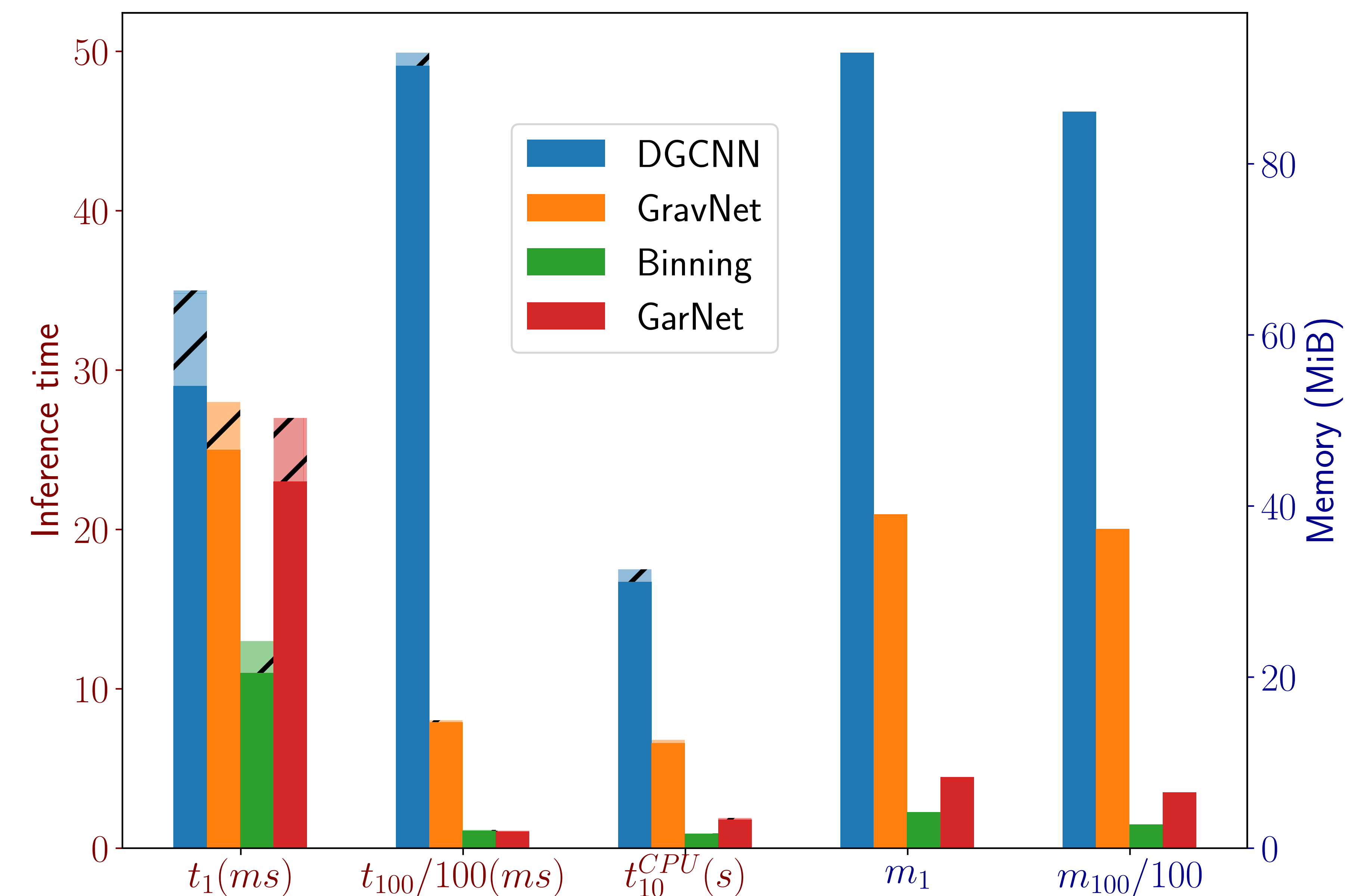
- Good performance achieved, comparable to DGCNN and traditional approaches
- Using a potential (V(d)) to weight up the near neighbours allows to keep memory footprint under control (with respect to other graph approaches)



(c) Mean

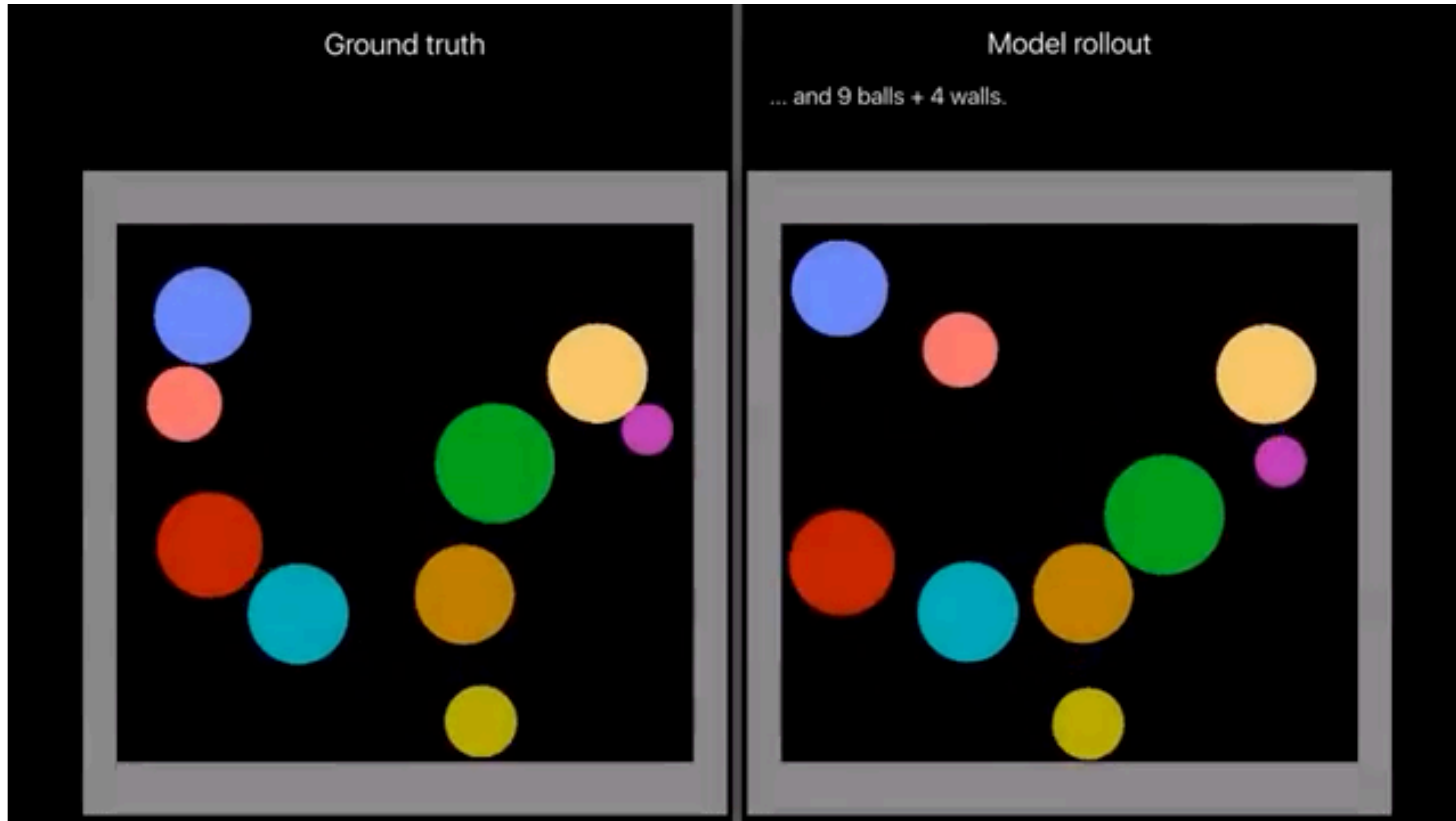


(d) Variance



European Research Council

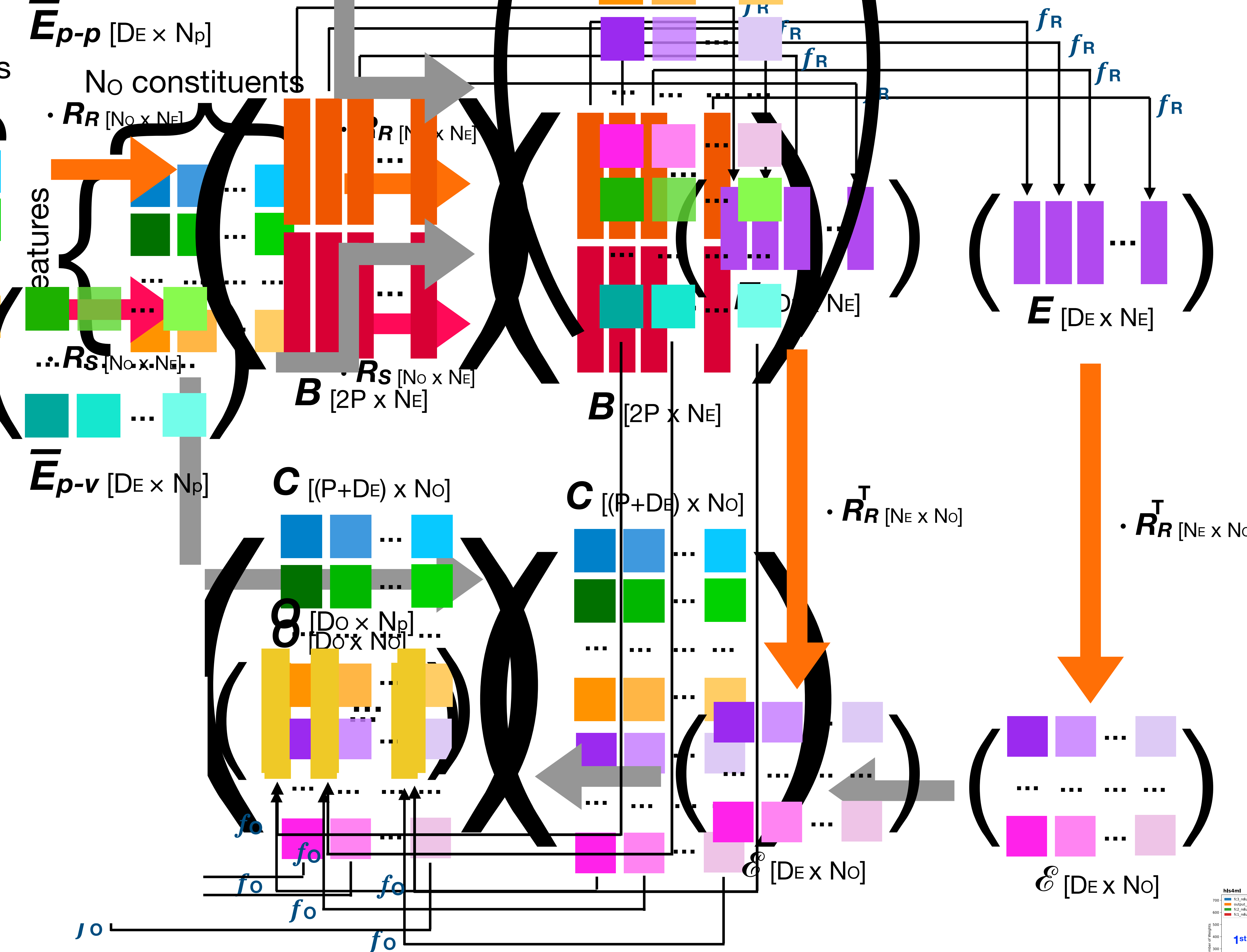
Interaction Networks



https://arxiv.org/abs/1612.00222

Interaction Networks

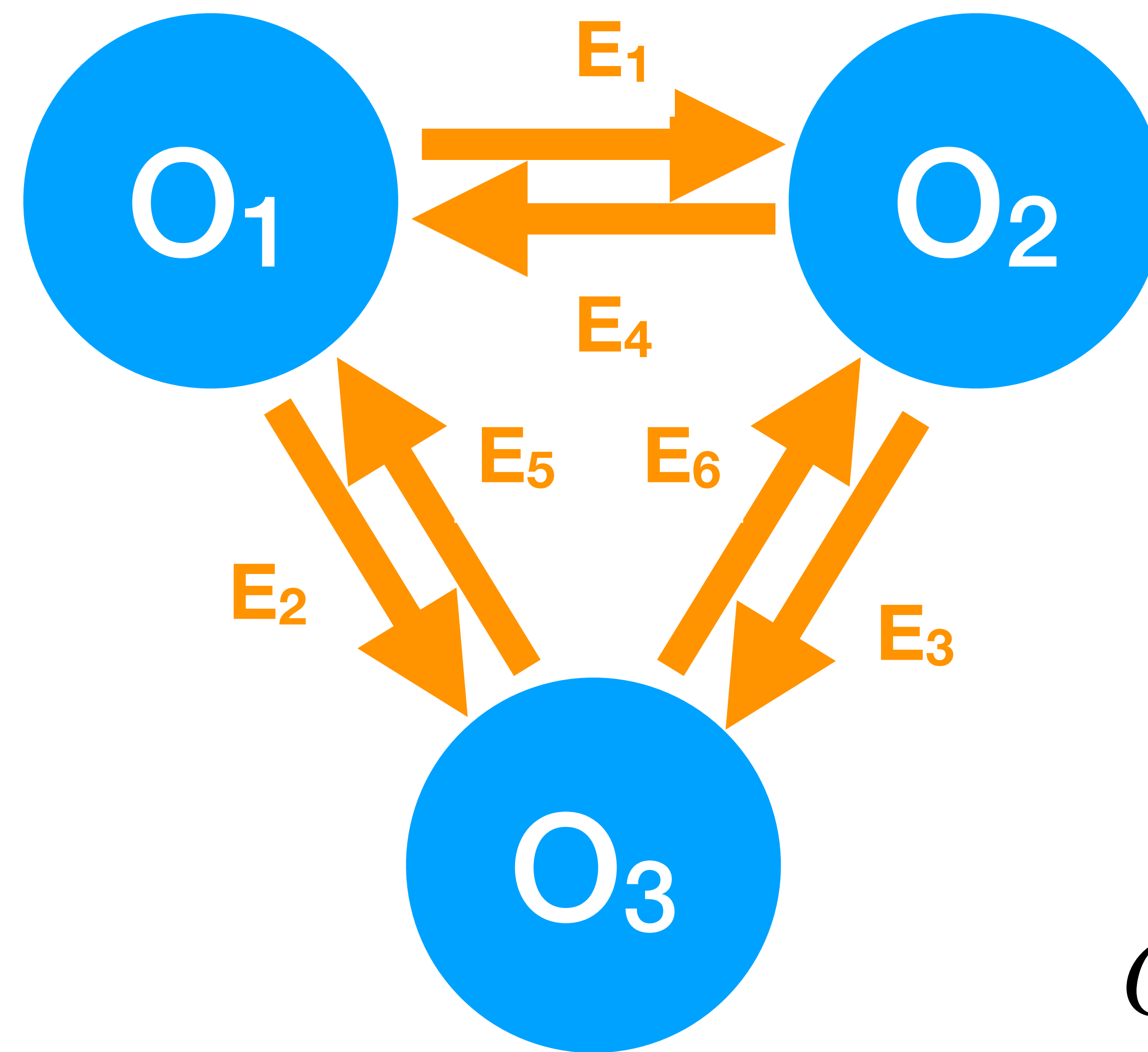
- INs process a list of No x P inputs in pairs, through Receiving and Sending matrices



Inter

25.04.2018

- *INs process a list of No x P inputs in pairs, through Receiving and Sending matrices*
- *The effect of the interaction is learned by fR and combined with the input to learn (through fo) a post-interaction representation*

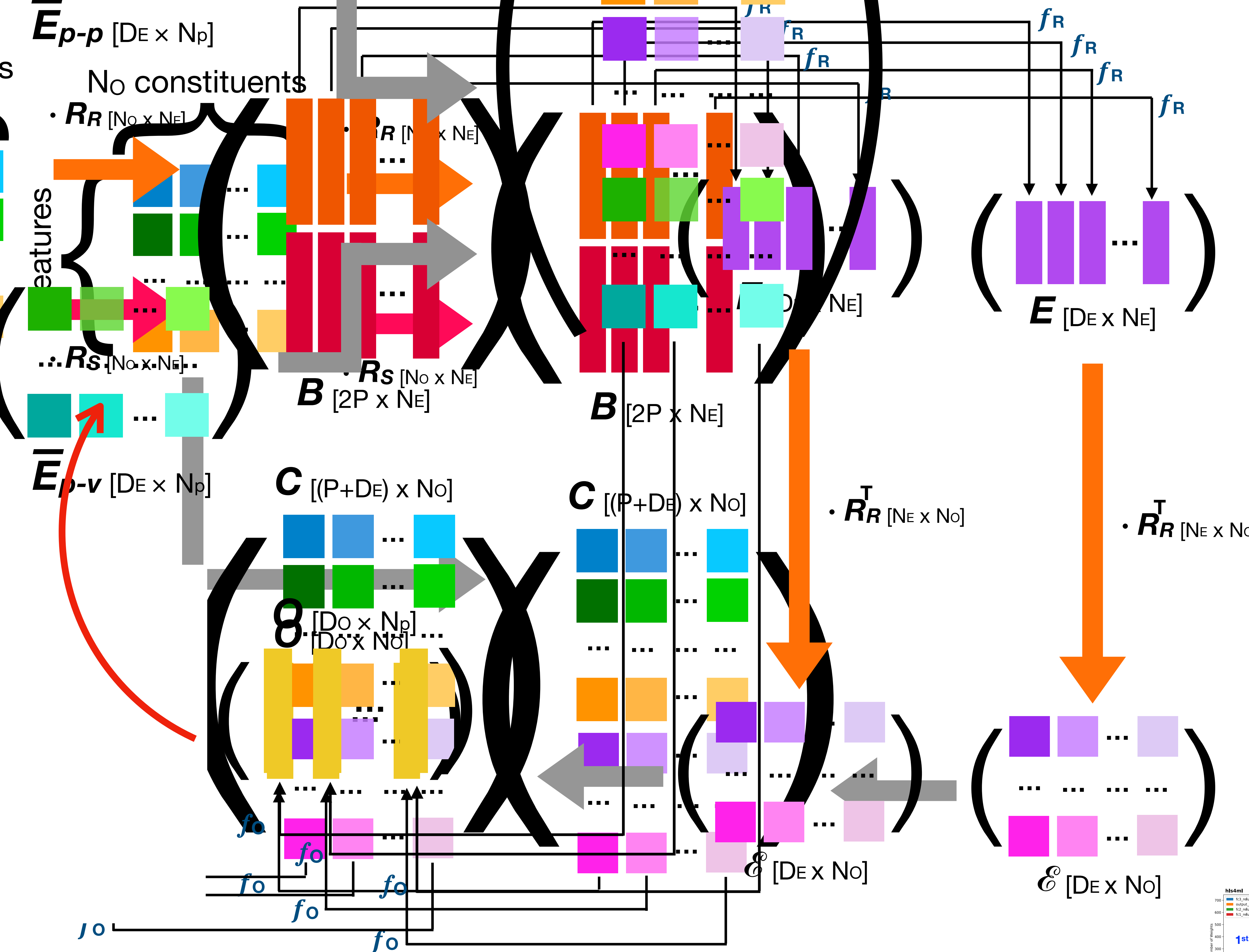


$$R_R = \begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \end{array} \begin{array}{c} \begin{array}{cccccc} E_1 & E_2 & E_3 & E_4 & E_5 & E_6 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$R_S = \begin{array}{c} \\ O_1 \\ O_2 \\ O_3 \end{array} \begin{array}{c} \begin{array}{cccccc} E_1 & E_2 & E_3 & E_4 & E_5 & E_6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}.$$

Interaction Networks

- INs process a list of No x P inputs in pairs, through Receiving and Sending matrices
- The effect of the interaction is learned by fR and combined with the input to learn (through fo) a post-interaction representation
- The procedure can then be iterated to produce further steps i the interactions

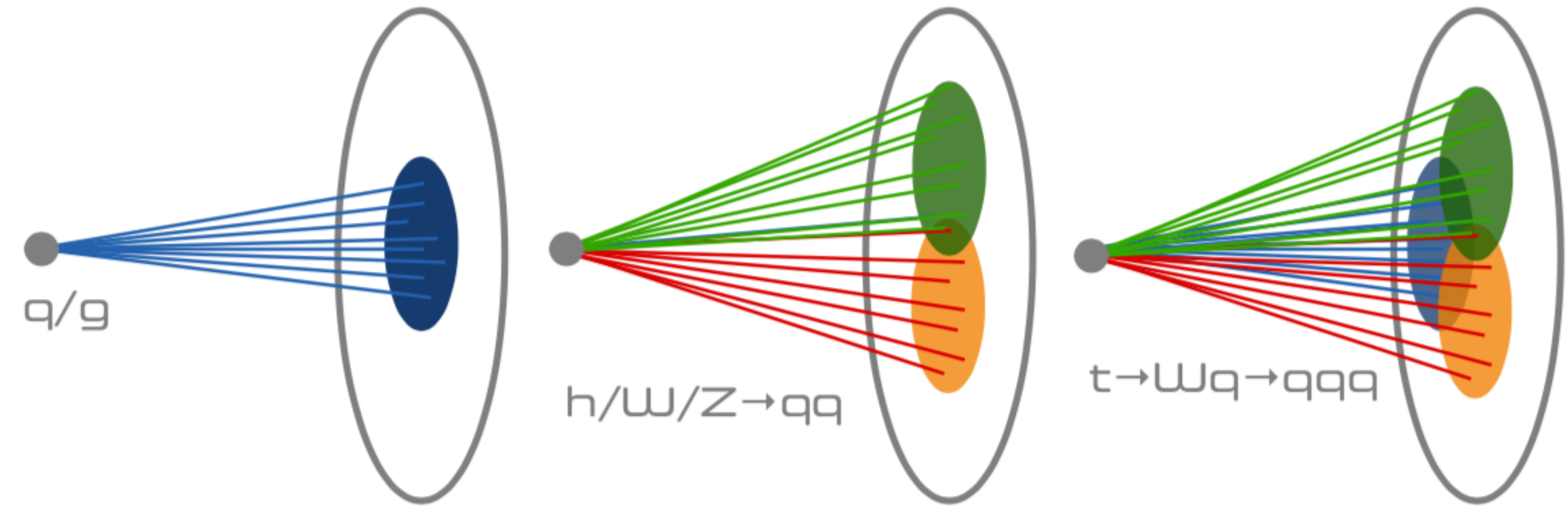


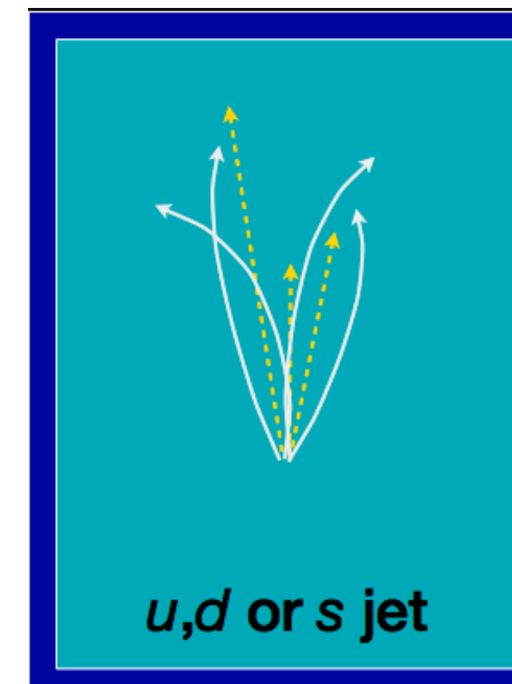
No: # of constituents
P: # of features
NE = No(No-1): # of edges
DE: size of internal representations
Do: size of post-interaction internal representation

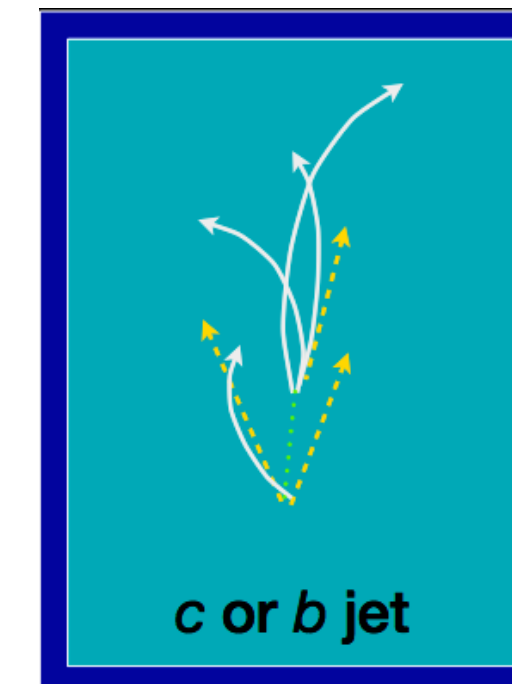
ϕ_C, f_O, f_R parameterized as neural networks

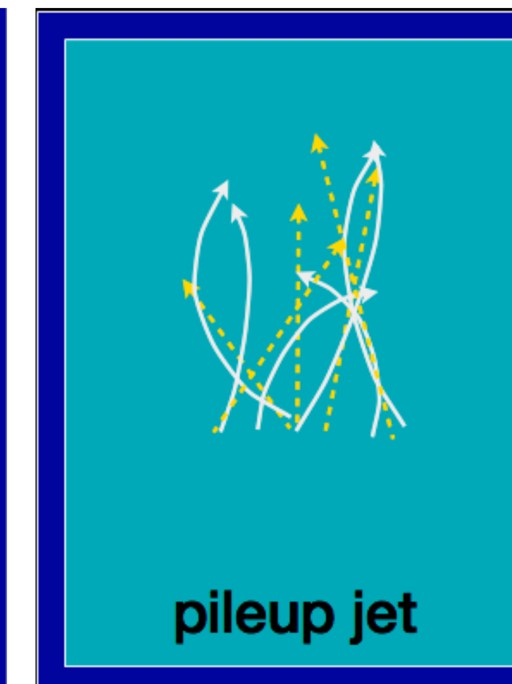
Example: jet tagging

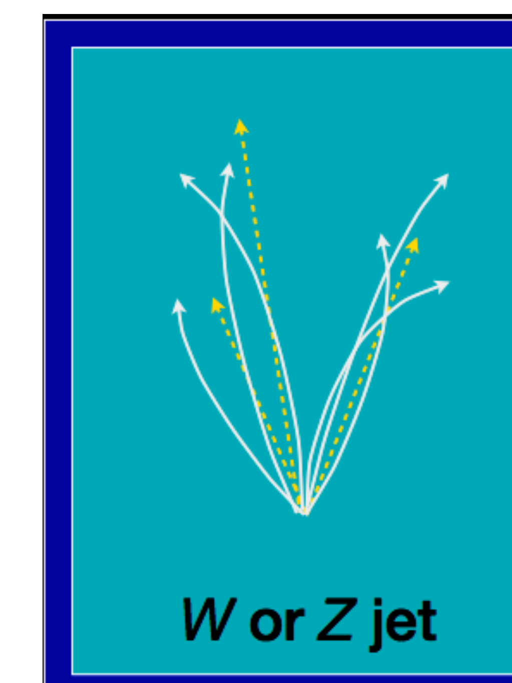
- You have a jet at LHC: spray of hadrons coming from a "shower" initiated by a fundamental particle of some kind (quark, gluon, W/Z/H bosons, top quark)
- You have a set of jet features whose distribution depends on the nature of the initial particle
- You can train a network to start from the values of these quantities and guess the nature of your jet
- To do this you need a sample for which you know the answer

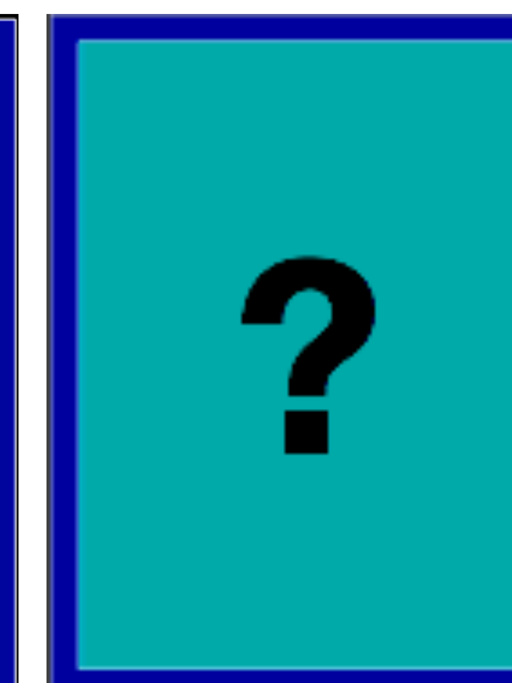






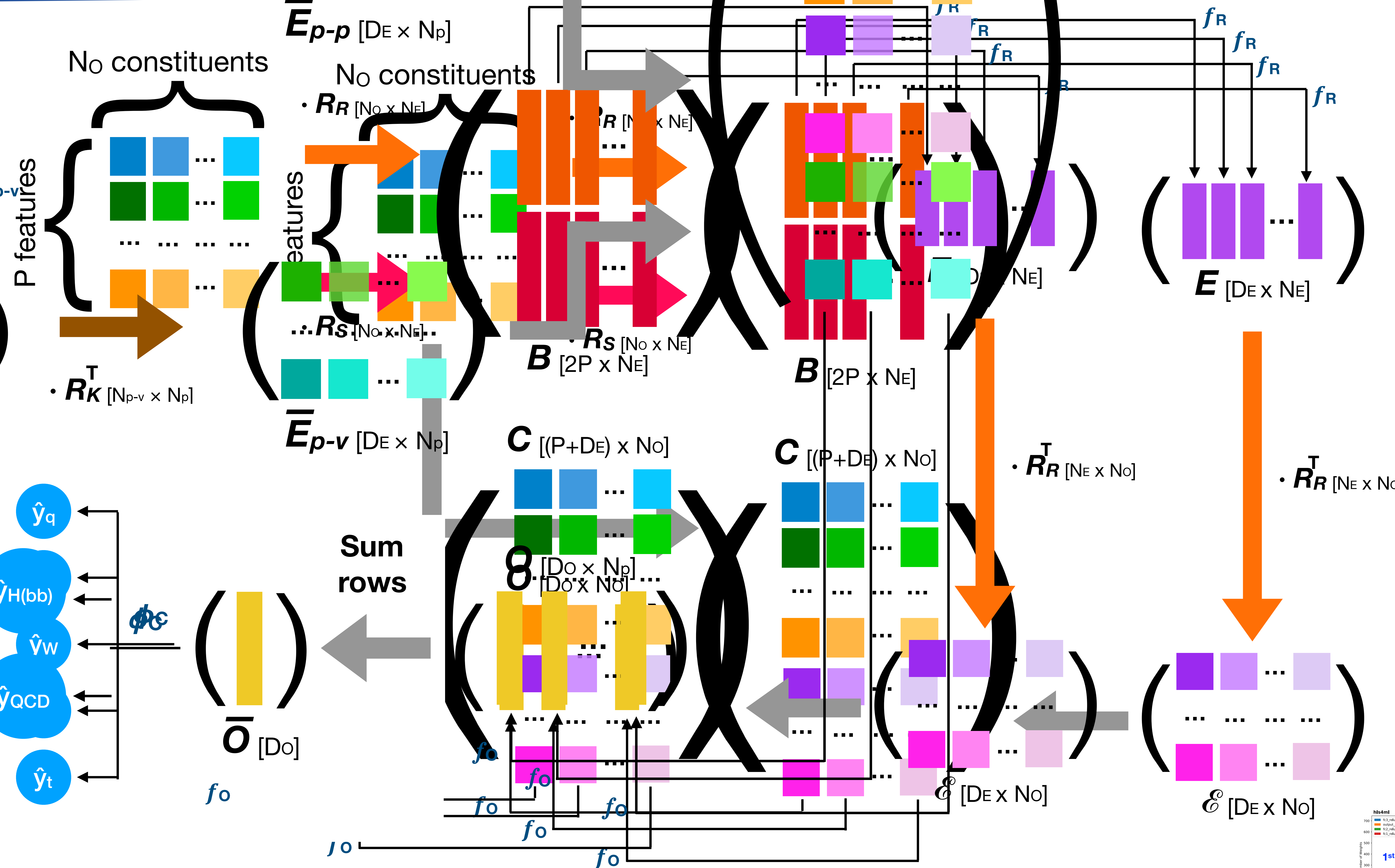






INs for Jet Identification

- *In this case, there is no system update needed (i.e., no cycle)*
- *It is sufficient to use the post-interaction representation as input to a classifier that returns the jet category*
- *The three networks are simultaneously optimized: the learned representation is chosen to help the classification*



A comparison