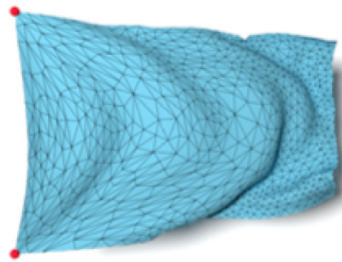
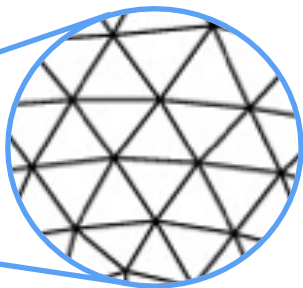
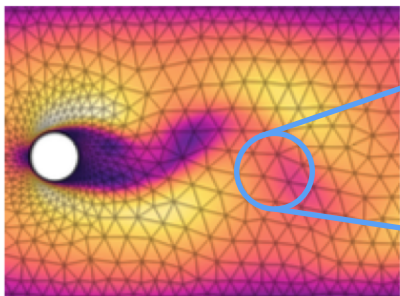


GNNs for Particle- and Mesh-Based Simulation (III)



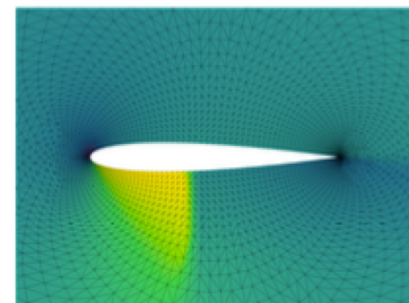
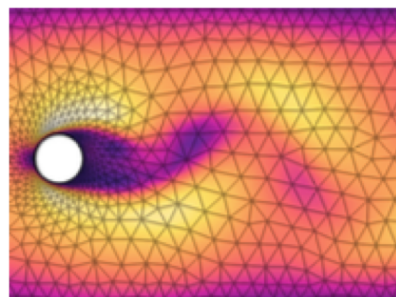
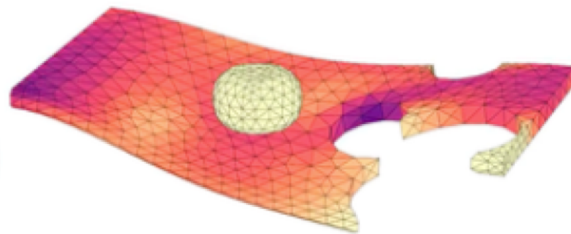
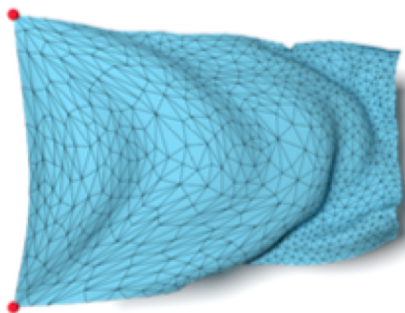
4. Mesh-based simulation with GNNs

Mesh-based simulation with GNNs

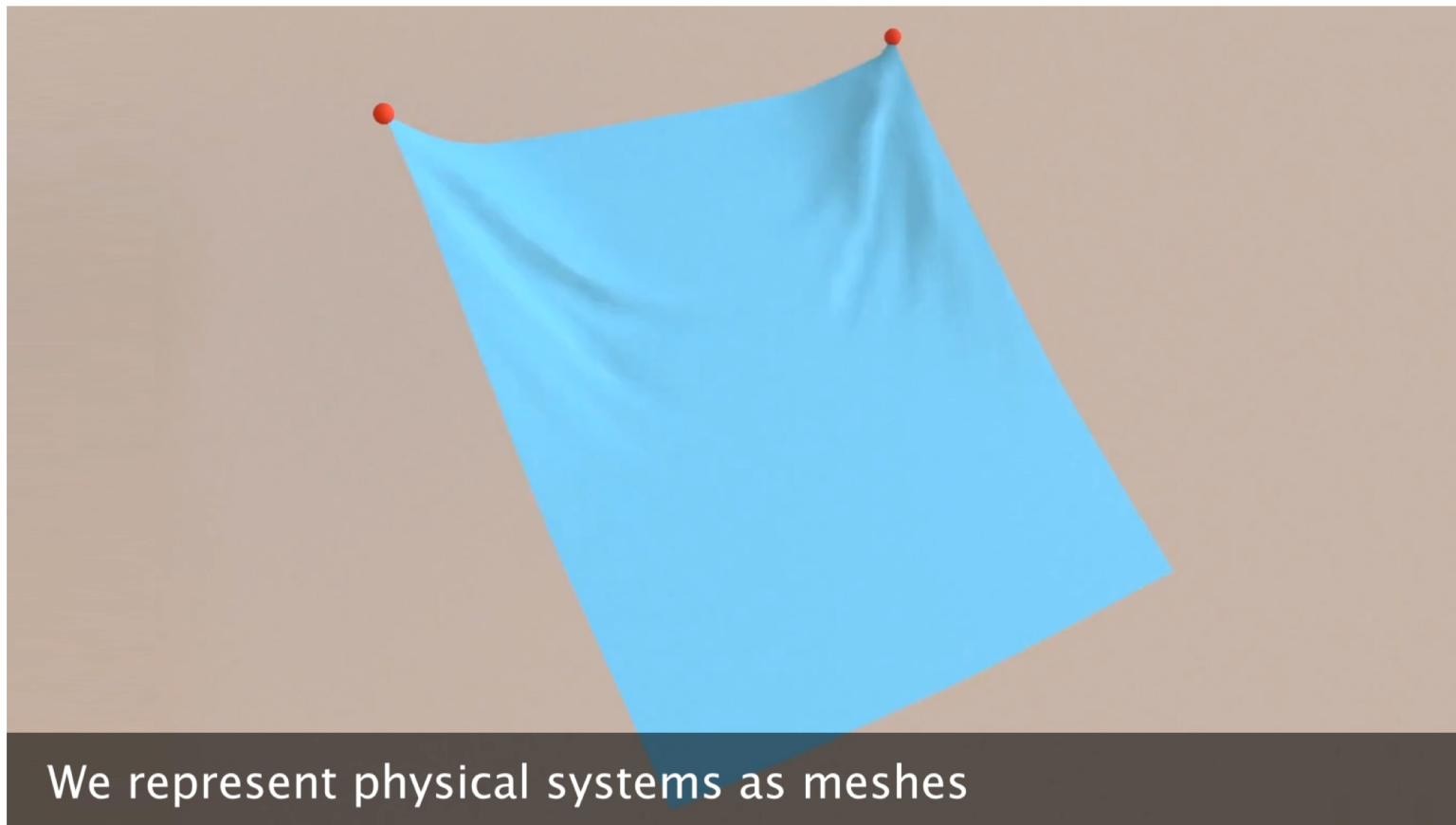
Topics that we will cover:

1. Learning Mesh-Based Simulation with Graph Networks
2. Graph Pooling and Unpooling
3. Multi-Scale Rotation-Equivariant Graph Neural Networks for Unsteady Eulerian Fluid Dynamics

Learning Mesh-Based Simulation with Graph Networks



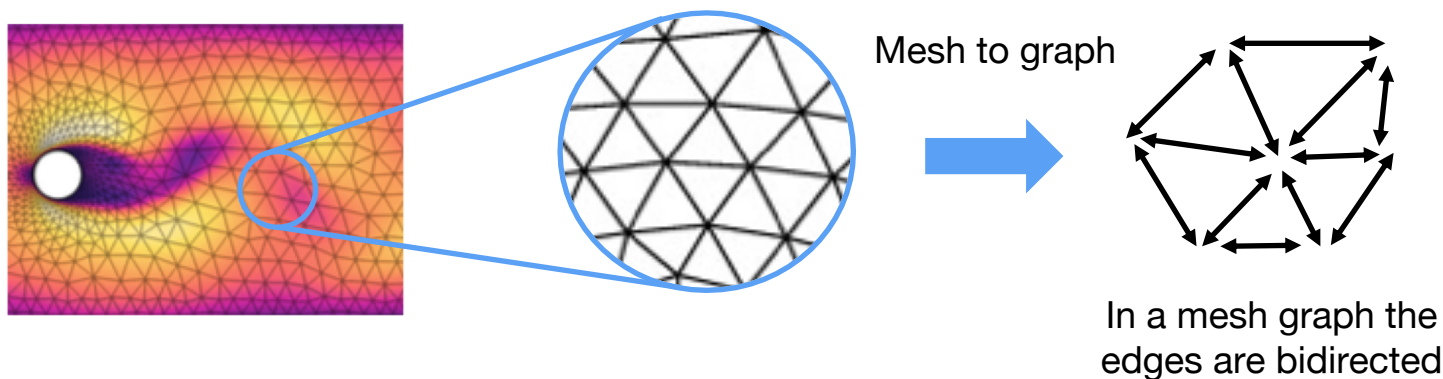
Learning Mesh-Based Simulation with GNs



We represent physical systems as meshes

Learning Mesh-Based Simulation with GNNs

- Same GNN architecture as in *Learning to Simulate Complex Physics with Graph Networks* by Sanchez-Gonzalez et al. – Encoder + $L \times \text{MP}$ + Decoder.
- MP applied to mesh graphs.
- Applied to infer the **(i) evolution of continuous fields (fluid dynamics)** and the **(ii) movement and deformation of solid surfaces (cloth dynamics) and volumes (structural dynamics)**.



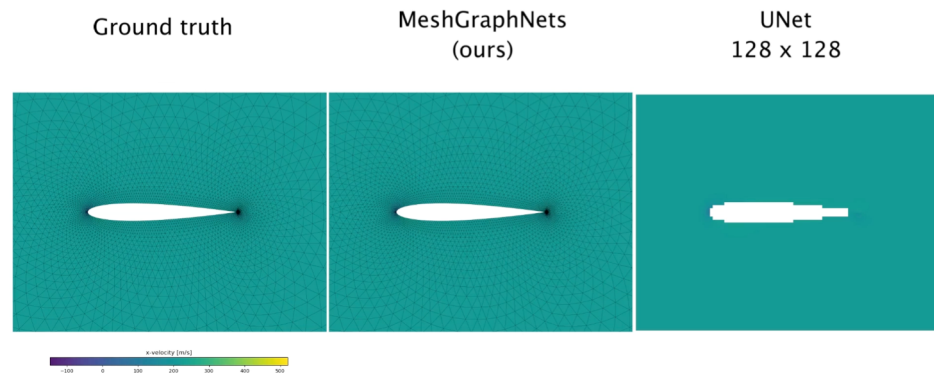
Learning Mesh-Based Simulation with GNNs

Fluid dynamics

- The state-of-the-art models for simulating fluid dynamics were CNNs applied to regular grids.
- Meshes allow to represent accurately complex geometries and adapt the resolution over space.

The GNN model employs 4x less nodes and still resolves better smaller scales.

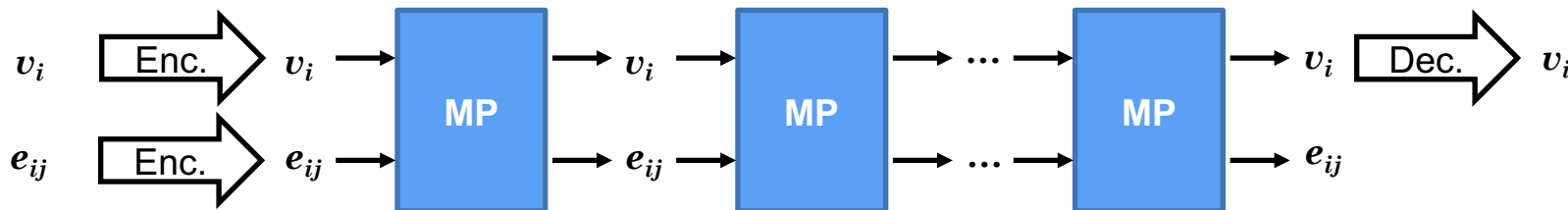
The GNN model is **(approx)** **discretization invariant** thanks to being trained with edges whose length ranges from $2 \cdot 10^{-4}$ m to 3.5 m.



Learning Mesh-Based Simulation with GNNs

Fluid dynamics

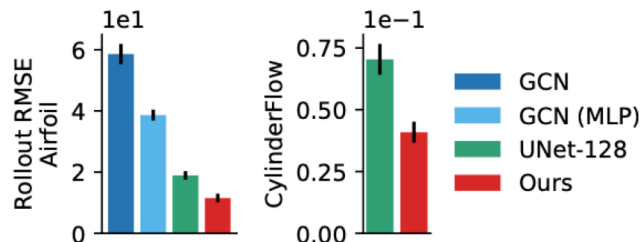
- Input node-features: (density), momentum and a hot-vector encoding the node type (fluid, inlet, outlet or wall).
- Input edge-features: relative position, $\mathbf{x}_i - \mathbf{x}_j$, and distance, $\|\mathbf{x}_i - \mathbf{x}_j\|$.
- Output node features: change in momentum (and density) and pressure.
- GNN architecture: Same as in *Learning to Simulate Complex Physics with Graph Networks* by Sanchez-Gonzalez et al.



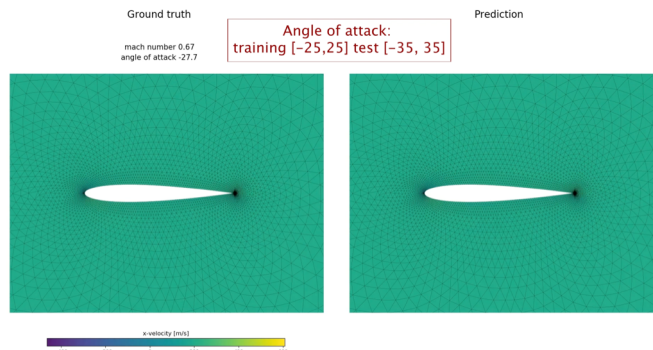
Learning Mesh-Based Simulation with GNs

Fluid dynamics

- Comparison with baseline models:



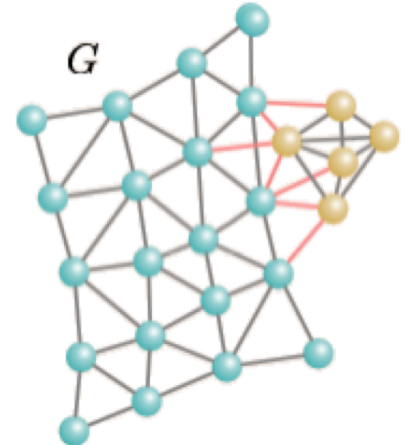
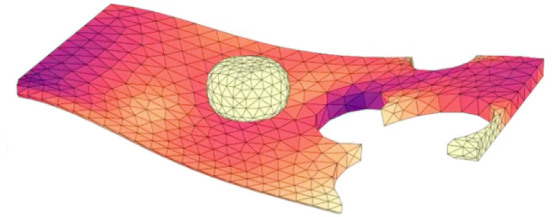
- Generalisation: steeper angles (-35 to 35 deg vs. -25 to 25 in training) and higher inflow speeds (Mach number 0.7 to 0.9 vs. 0.2 to 0.7 in training). The RMSE raises only from 11.5 at training to 12.4 for steeper angles and 13.1 for higher inflow speeds.



Learning Mesh-Based Simulation with GNNs

Structural dynamics: A rigid actuator deforming a hyper-elastic plate.

- There are external nodes in the actuator.
- The distance between the nodes on the plate changes over time.
- Solution:
 - **Mesh-space edges**, E^M (in black in the figure), between nodes in the same solid. They are given by the mesh and model internal dynamics.
 - **World-space edges**, E^W (in red in the figure), between nodes in different solids closer than a distance R . They model external dynamics, such as contact and collision.



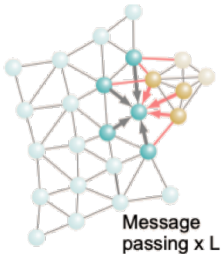
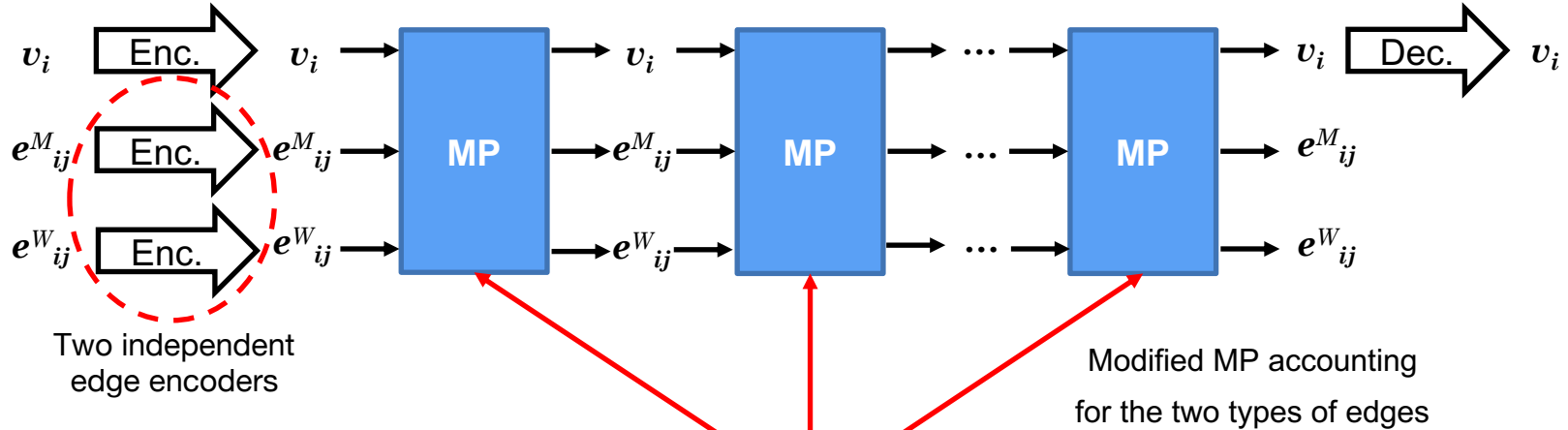
Learning Mesh-Based Simulation with GNs

Structural dynamics

- The coordinates of each node in the original mesh, before being deformed, is denoted by ξ_i , and in the world-space, after applying the deformation, is denoted by x_i .
- Input node-features: a one-hot vector encoding for the node type (actuator or plate).
- Input mesh-space edge-features (e_{ij}^M): $x_i - x_j$, $\|x_i - x_j\|$, $\xi_i - \xi_j$, and $\|\xi_i - \xi_j\|$.
- Input world-space edge-features (e_{ij}^W): $x_i - x_j$, and $\|x_i - x_j\|$.
- Output node features: change in x_i and the von-Misses stress.
- An MLP encoder is also needed for the input world-space edge-features.
- MP is modified to account for two type of edges.

Learning Mesh-Based Simulation with GNs

Structural dynamics



Update mesh-space edge-attributes:

$$e_{ij}^M \leftarrow f^M(e_{ij}^M, v_i, v_j), \quad \forall (i, j) \in E^M,$$

Update world-space edge-attributes:

$$e_{ij}^W \leftarrow f^W(e_{ij}^W, v_i, v_j), \quad \forall (i, j) \in E^W,$$

Update node attributes:

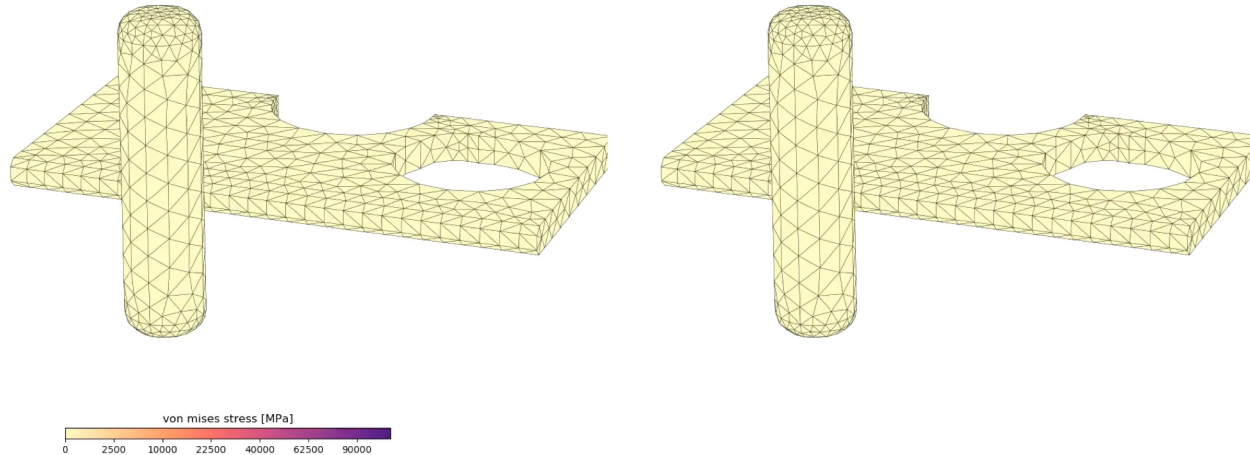
$$v_j \leftarrow f^V\left(v_j, \sum_{i \in \mathcal{N}_j^{-,M}} e_{ij}^M, \sum_{i \in \mathcal{N}_j^{-,W}} e_{ij}^W\right), \quad \forall j \in V,$$

Learning Mesh-Based Simulation with GNs

Structural dynamics

Ground truth

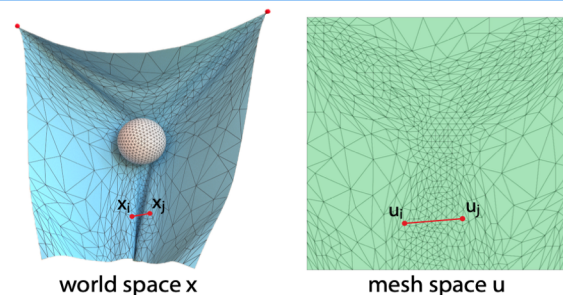
Prediction



Learning Mesh-Based Simulation with GNNs

Cloth dynamics:

1. A flag blowing in the wind.
2. A piece of cloth interacting with a moving sphere.



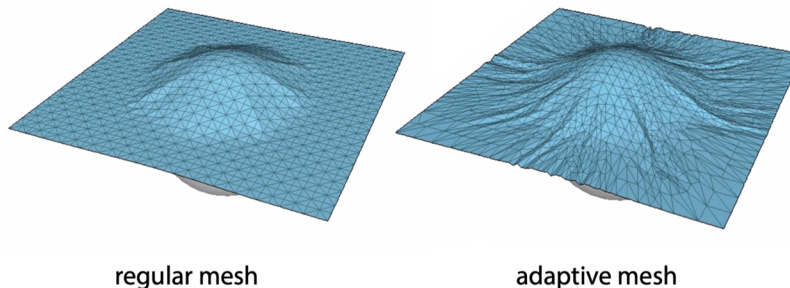
- Input node-features: a one-hot vector encoding for the node type (cloth or rigid sphere) and velocity, $\dot{\mathbf{x}}_i$, at the current and previous time point.
- Input mesh-space edge-features (\mathbf{e}_{ij}^M): $\mathbf{x}_i - \mathbf{x}_j$, $\|\mathbf{x}_i - \mathbf{x}_j\|$, $\xi_i - \xi_j$, and $\|\xi_i - \xi_j\|$.
- Input world-space edge-features (\mathbf{e}_{ij}^W): $\mathbf{x}_i - \mathbf{x}_j$, and $\|\mathbf{x}_i - \mathbf{x}_j\|$.
- Output node features: acceleration $\ddot{\mathbf{x}}_i$.
- Same GNN model as for structural dynamics.

between closer than a distance R
and not connected by mesh edges

Learning Mesh-Based Simulation with GNs

Cloth dynamics: Adaptive Remeshing

- After **each time-step** the number of nodes and their connections are adapted to efficiently **optimise the local resolution of the mesh graph**.
- In cloth dynamics, high-curvature areas are refined to ensure smooth bedding dynamics and low curvature areas are coarsened.



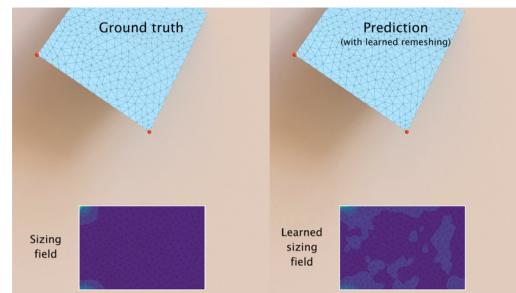
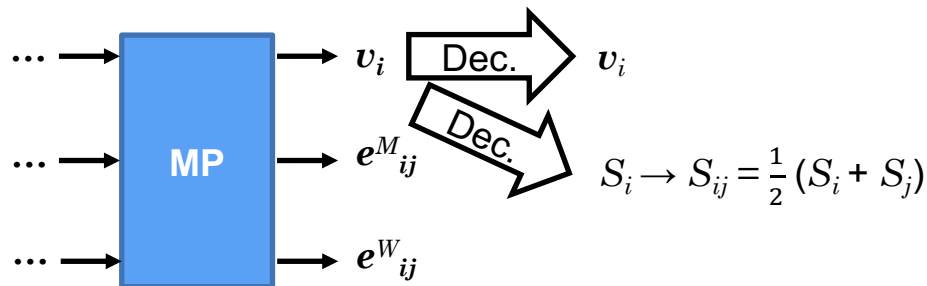
- Adaptive remeshing has two steps:
 1. Identify which regions need to be refined or coarsened.
 2. Adapt the nodes and edges to satisfy the conditions in point 1.

Learning Mesh-Based Simulation with GNs

Cloth dynamics: Adaptive Remeshing

1. Identify which regions need to be refined or coarsened

- The **sizing field methodology** is followed.
- The size field quadratic form, $S(\xi)$, indicates the maximally allowed edge length at each point and along each direction.
- An edge (i,j) is valid if and only if $(\xi_j - \xi_i)^T S_{ij} (\xi_j - \xi_i) \leq 1$
- To reduce the computational cost, the 4 components of the sizing field tensor S_i were predicted by applying an MLP decoder to the output of the last MP layer.



Learning Mesh-Based Simulation with GNs

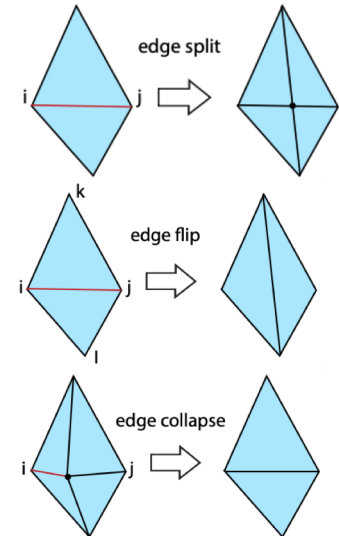
Cloth dynamics: Adaptive Remeshing

2. Adapt the nodes and edges to satisfy the previous condition

(Not machine learning)

1. **Split** all edges with $(\xi_j - \xi_i)^T S_{ij} (\xi_j - \xi_i) > 1$ in descending order of this metric.
2. **Flip** all edges with

$$(\xi_{jk} \times \xi_{ik}) \xi_{il}^T S_A \xi_{jl} < \xi_{jk}^T S_A \xi_{ik} (\xi_{il} \times \xi_{jl}), \quad S_A = 1/4(S_i + S_j + S_k + S_l)$$
 To optimise the direction of the edges.
3. **Collapse** all edges that do not generate invalid edges in ascending order of $(\xi_j - \xi_i)^T S_{ij} (\xi_j - \xi_i)$.
4. **Flip** edges again to optimise the direction of the edges (Repeat 2).

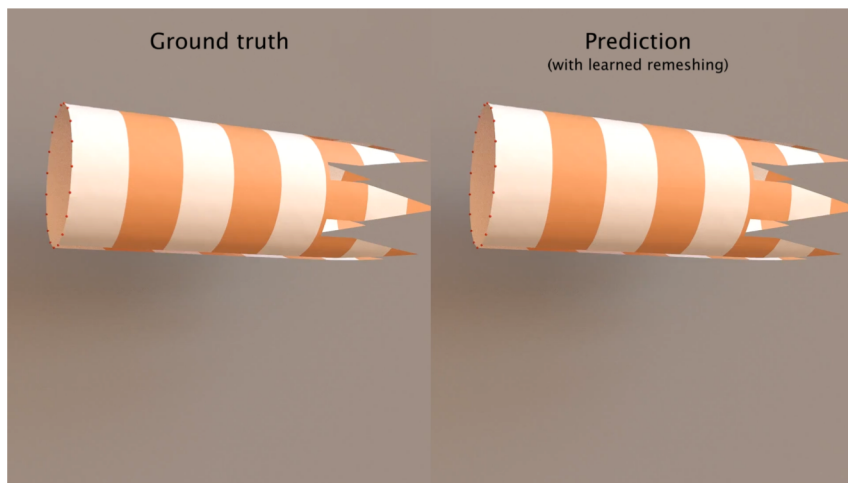
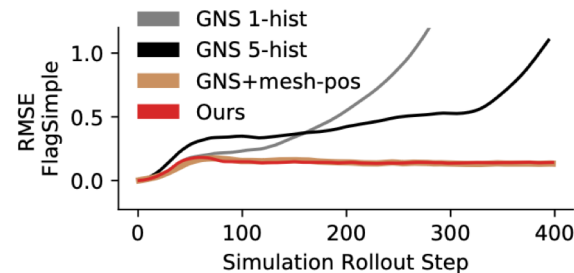


Learning Mesh-Based Simulation with GNs

- Comparison to baseline models:

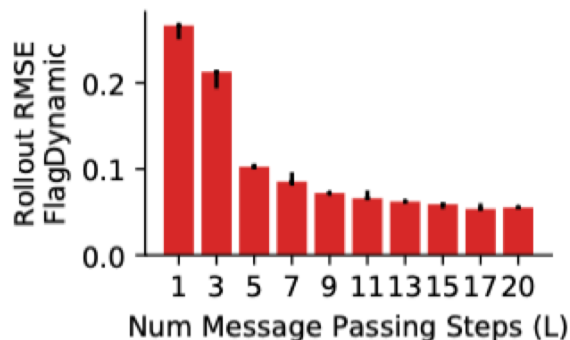
Nevertheless GNS+mesh-pos develops entangled triangles, which indicate a lack of understanding of the mesh surface.

- Generalisation to larger flags, three flags instead of one and different shapes.



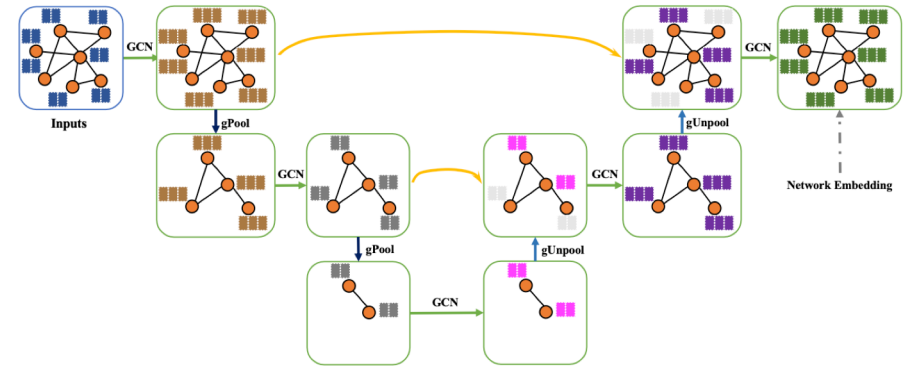
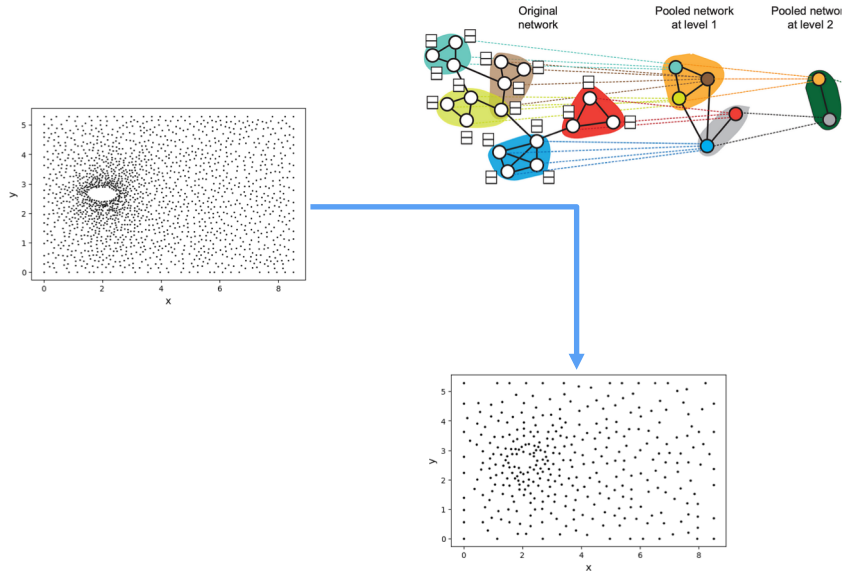
Learning Mesh-Based Simulation with GNs

- Increasing the number of MP layers generally improves the accuracy.



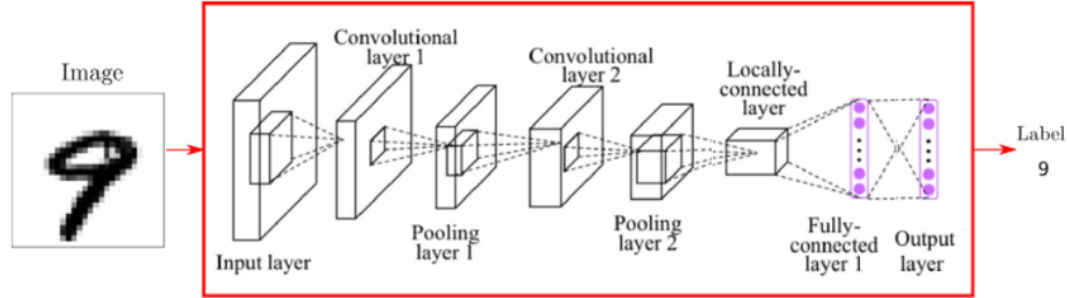
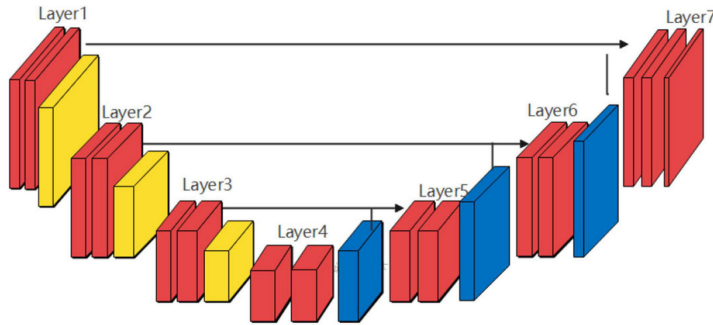
- Why?
- What are the drawbacks of increasing the number of ML layers?

Graph Pooling and Unpooling



Graph Pooling and Unpooling

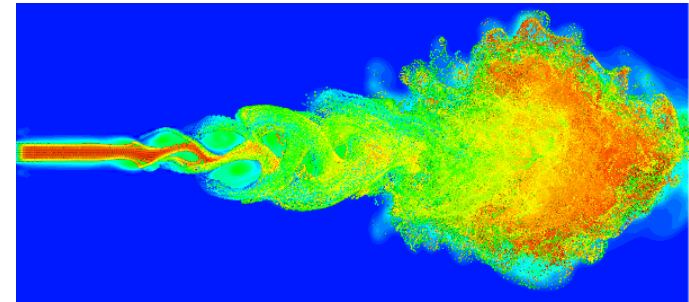
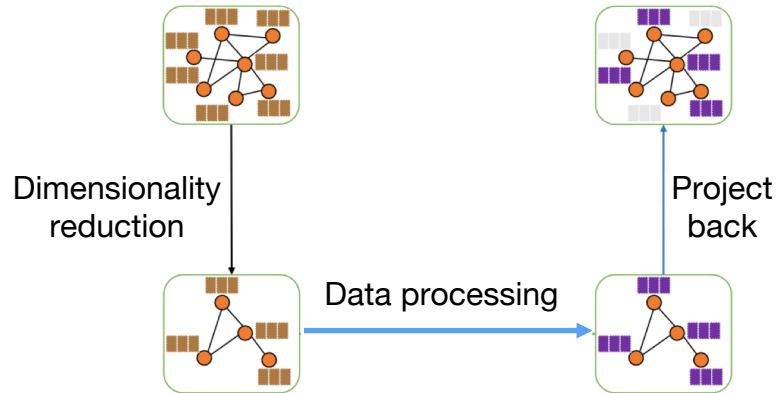
- Pooling and unpooling are fundamental operation in CNNs for pixel-wise predictions and for image-level predictions.



- Graph pooling is more complex because the nodes do not form a grid and they do not always have spatial locality (e.g., social networks).
- Several graph pooling and unpooling algorithms have been propose.
- The selection of one or another is highly domain dependent.

Graph Pooling

- In physics applications, graph pooling is needed for **dimensionality reduction**, to **increase the receptive field**, to **process graph-level information** and **extract features across different levels of resolution**.



Flow patterns can be identified at multiple scales

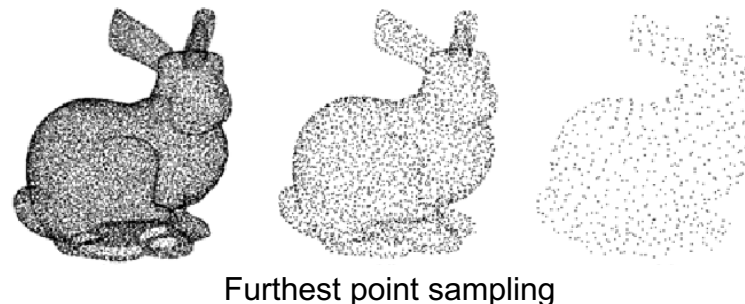
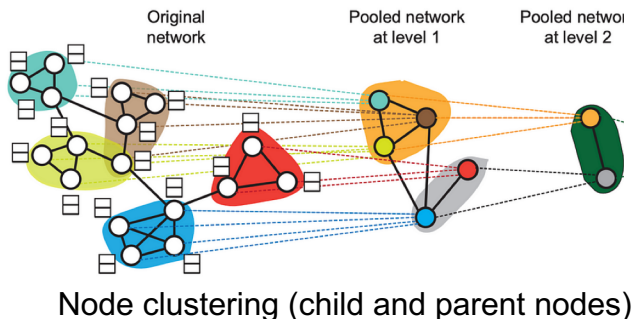
- The dynamics of many physical systems are global: a modification of a physical magnitude at any point in space can affect every other point.
- Some physical phenomena (e.g. turbulent phenomena) span a range of length-scales. 22

Graph Pooling

Graph pooling from a graph $G^1 = (V^1, E^1)$ to a graph $G^2 = (V^2, E^2)$ has 4 steps:

1. Coarsen V^1 to obtain V^2

- This can be **learnable** (e.g., Top-k pooling) or **non-learnable** (e.g., k-means clustering, voxel-grid clustering, Guillard coarsening).
- Coarsening can be based on **node-clustering** or **node-dropping** (e.g., Top-K pooling, FPS).



Graph Pooling

2. Define E^2

- The new edges can preserve the connectivity: there is an edge from parent node i to parent node j if there is an edge from any of i 's child nodes to any of j 's child nodes – **see the left diagram on the previous slide.**
- The new edges can be defined based on the nodes' spatial proximity (e.g., k -nearest neighbours, neighbourhood of radius R).

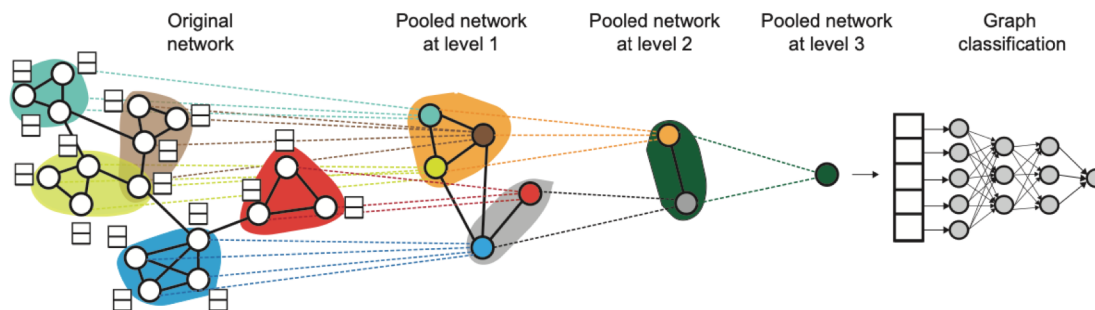
3. Assign features to the nodes in V^2 (based on the features of V^1)

- If V^2 was obtained by node clustering we can do child to parent interpolation or message passing. We can also do mean or max pooling.
- If V^2 was obtained by node dropping we can preserve the features they had in V^1 or add a node-wise projection (e.g., Top-k pooling).

4. Assign features to edges in E^2 . We can assign the relative position of the coarse nodes.

Graph Pooling: DiffPooling (Additional Material)

- Introduced in *Hierarchical Graph Representation Learning with Differentiable Pooling* by Ying et al in 2019.
- Aim: Learnable node clustering, the new edges preserve the connectivity and the node features are node-wise projected and cluster-wise aggregated.



- In practice: Every node belongs, in a lower or higher percentage, to every cluster and the new graph is fully connected.
- In order to achieve learnable clustering they use a learnable and **differentiable** *cluster assignment matrix*, S .

Graph Pooling: DiffPooling (Additional Material)

- The inputs of a DiffPooling layer are the higher-resolution adjacency matrix $A^1 \in \mathbb{R}^{|V^1| \times |V^1|}$ and the node-feature matrix $X^1 \in \mathbb{R}^{|V^1| \times F_{\text{in}}}$, and it returns as output the lower-resolution adjacency matrix $A^2 \in \mathbb{R}^{|V^2| \times |V^2|}$ and the node-feature matrix $X^2 \in \mathbb{R}^{|V^2| \times F_{\text{out}}}$
- It has two separate GNNs:
 - The embedding GNN: It generates new features $Z^1 \in \mathbb{R}^{|V^1| \times F_{\text{out}}}$ for the clustered-nodes V^2

$$Z^1 \leftarrow \text{GNN}_{\text{emb}}(A^1, X^1)$$

- The pooling GNN: It returns the cluster assignment matrix $S^1 \in \mathbb{R}^{|V^1| \times |V^2|}$

$$S^1 \leftarrow \text{softmax}(\text{GNN}_{\text{pool}}(A^1, X^1))$$

The softmax is applied row-wise (i.e., node-wise), so it indicates the probability of each node belonging to each cluster V^2 .

Graph Pooling: DiffPooling (Additional Material)

- Aggregate the node features according to their probability of belonging to each cluster:

$$X^2 = (S^1)^T Z^1$$

- Compute the new adjacency matrix:

$$A^2 = (S^1)^T A^1 S^1$$

- This adjacency matrix is dense, where the higher-resolution one could be very sparse. This makes DiffPooling not desirable for large sparse matrices.
- We could assign each node to only to highest probability cluster:

$$s_i = [0.1, 0.3, 0.2, \mathbf{0.4}] \rightarrow s_i = [0, 0, 0, 1]$$

But this leads to unstable training.

Graph Pooling: DiffPooling (Additional Material)

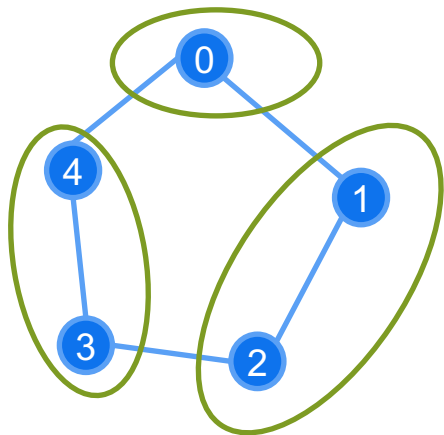
- DiffPooling layers also may have slow convergence and be very sensitive to the initial training weights.
- To mitigate this, two extra terms are added to the loss function as a form of regularisation:
 1. To reward the assignment of a unique cluster to each node, we want to minimise the entropy of each row of S and so we add the following term:

$$-\sum_{i \in V^1} s_i^1 \log(s_i^1)$$

2. To reward that each node belongs to the same cluster as its neighbours (to avoid clustering node located very far away from each other), we add the following term:

$$\|A^1 - (S^1)^T S^1\|_2$$

Graph Pooling: DiffPooling (Additional Material)



$$\|A^1 - (S^1)^T S^1\|_2$$

This rewards that each node belongs to the same cluster as its neighbours

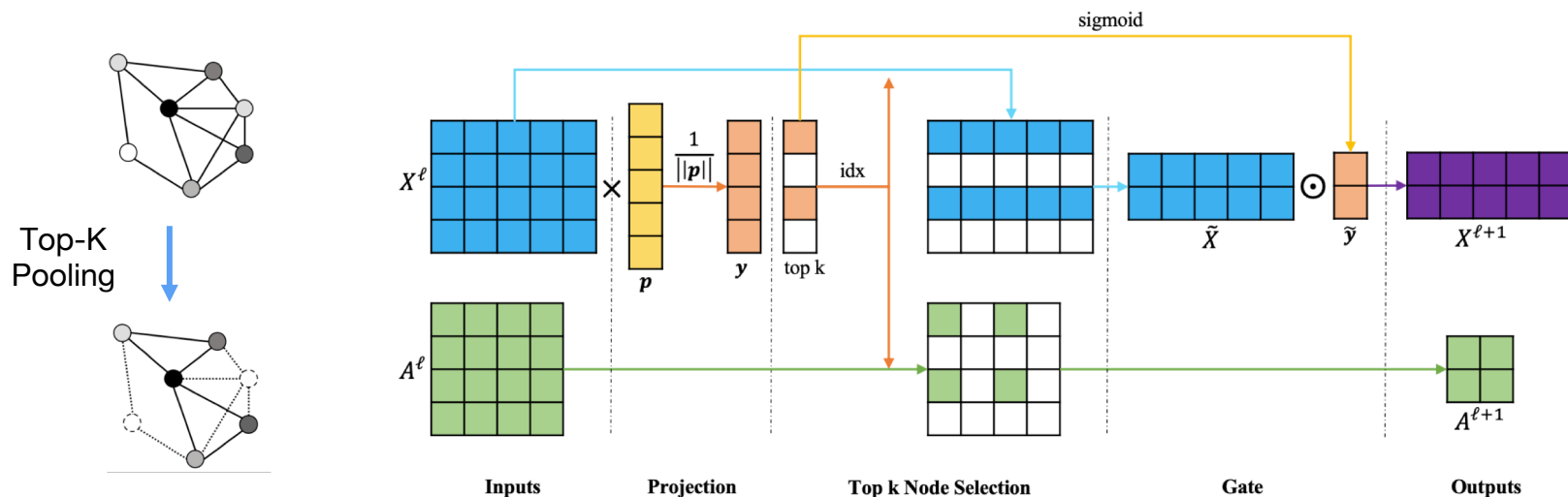
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	0	0
2	0	1	0	1	0
3	0	0	1	0	1
4	1	0	0	1	0

	0	1	2	3	4
0	1	0	0	0	0
1	0	1	1	0	0
2	0	1	1	0	0
3	0	0	0	1	1
4	0	0	0	1	1

The i -th row indicates which nodes belong to the same cluster as node i

Graph Pooling: Top-k pooling

- Introduced in *Graph U-Nets* by Gao and Ji in 2019.
- Learnable node-dropping**, the new **edges preserve the connectivity** and the **node features are node-wise scaled** (they did not consider edge features).



- They introduce a learnable projection vector, p .

Graph Pooling: Top-k pooling

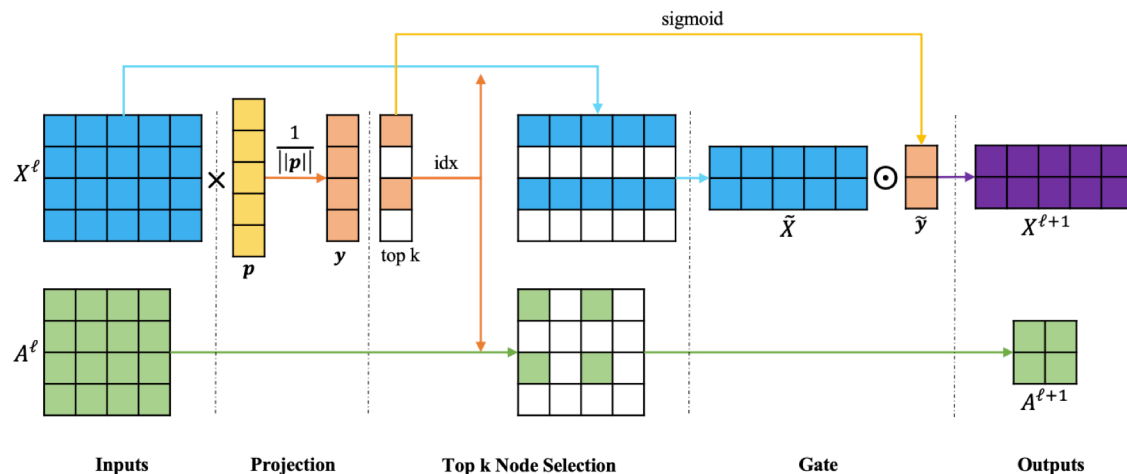
$$\mathbf{y} = \frac{\mathbf{X}^1 \mathbf{p}}{\|\mathbf{p}\|}$$

$$\text{idx} = \text{top}_k(\mathbf{y})$$

$$\tilde{\mathbf{y}} = \sigma(\mathbf{y}[\text{idx}])$$

$$\mathbf{X}^2 = (\mathbf{X}^1[\text{idx}] \odot \tilde{\mathbf{y}})$$

$$\mathbf{A}^2 = \mathbf{A}^1[\text{idx}, \text{idx}]$$



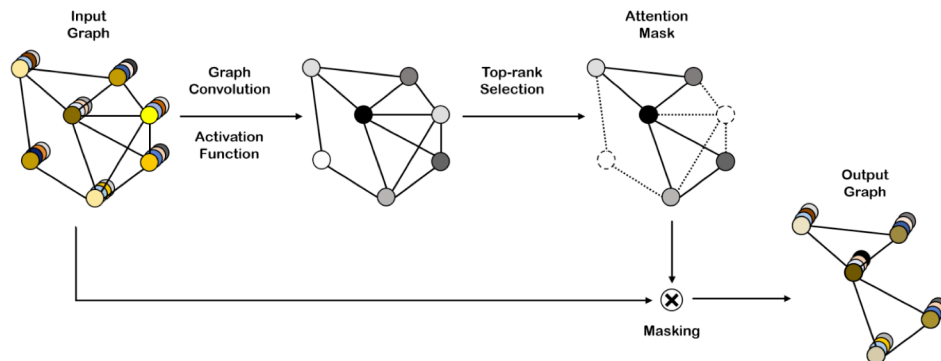
- The projection of the node features along p (score) measures how much information of each node can be retained after coarsening.
- We keep nodes with the largest projection/score to preserve as much information as possible.
- The gate operation makes vector p **trainable by back-propagation**.

Graph Pooling: Top-k pooling

- To avoid isolated nodes and low indegrees, it is possible to increase connectivity among nodes by adding edges between nodes whose distances are at most k hops:

$$\mathbf{A}^2 = (\mathbf{A}^1)^k [\mathbf{idx}, \mathbf{idx}]$$

- Self-Attention Graph Pooling (SAGPooling)** is a more general form of Top-k pooling which pays more careful attention to the graph topology by applying an additional GNN, diverging from the main GNN branch, instead of using the learnable vector \mathbf{p} :



$$\mathbf{y} = \sigma(\text{GNN}(G))$$

$$\mathbf{idx} = \text{top}_k(\mathbf{y})$$

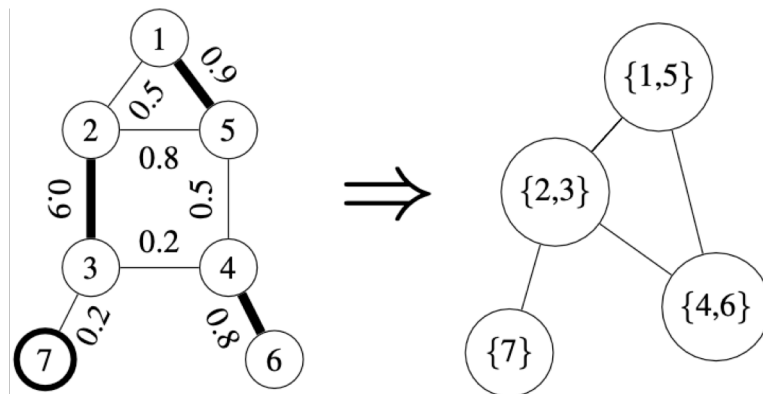
$$\tilde{\mathbf{y}} = \mathbf{y}[\mathbf{idx}]$$

$$\mathbf{X}^2 = (\mathbf{X}^1[\mathbf{idx}] \odot \tilde{\mathbf{y}})$$

$$\mathbf{A}^2 = \mathbf{A}^1[\mathbf{idx}, \mathbf{idx}]$$

Graph Pooling: Edge pooling

- Top-K pooling improves on DiffPooling since it is **sparse and variable in graph size**.
- But, whole areas of a graph might see no node chosen, **losing information**.
- To avoid this **edge pooling relies on edge contraction**.
- Introduced in *Towards Graph Pooling by Edge Contraction* by Diehl et al. In 2019.
- Learnable node clustering (clusters of size 2 or 1), the new edges preserve the connectivity and the node features are cluster-wise aggregated.



Graph Pooling: Edge pooling

- A score, s_{ij} , is computed for each edge:

$$s_{ij} = \sigma \left(\mathbf{w}^T [\mathbf{v}_i || \mathbf{v}_j] + b \right)$$

- The activation function can be:

$$\sigma(e_{ij}) = \tanh(e_{ij}) \quad \text{or} \quad \sigma(e_{ij}) = \text{softmax}_{k \in \mathcal{N}_j^-}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_j^-} \exp(e_{kj})}$$

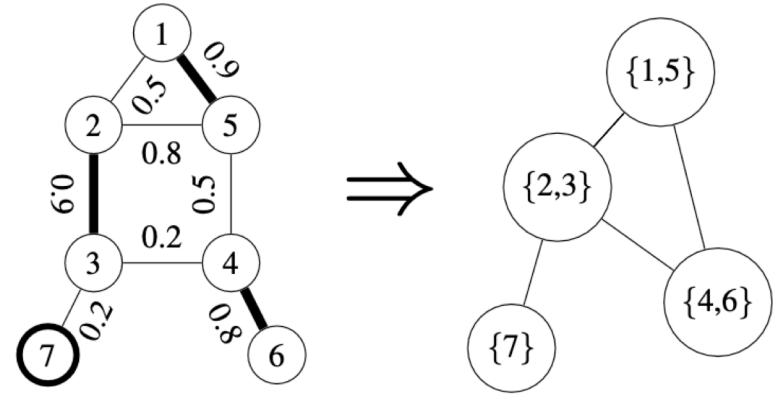
It does not take
neighbouring edges into
account

By normalising over all edges which end in the same node
we give a higher score to edges with an unnormalised
score higher than their neighbours

Graph Pooling: Edge pooling

1. The edges are contracted in order of their score (edges {1, 5}, {4, 6}, {2, 3}), with nodes that have already been part of a pooled edge ignored (e.g., edge {2, 5}).
2. The new edges preserve the connectivity of the clusters in the old graph.
3. To enable the gradients to flow into the scores, the aggregated node features are multiplied by the edge score:

$$\mathbf{v}_{ij} \leftarrow s_{ij}(\mathbf{v}_i + \mathbf{v}_j)$$

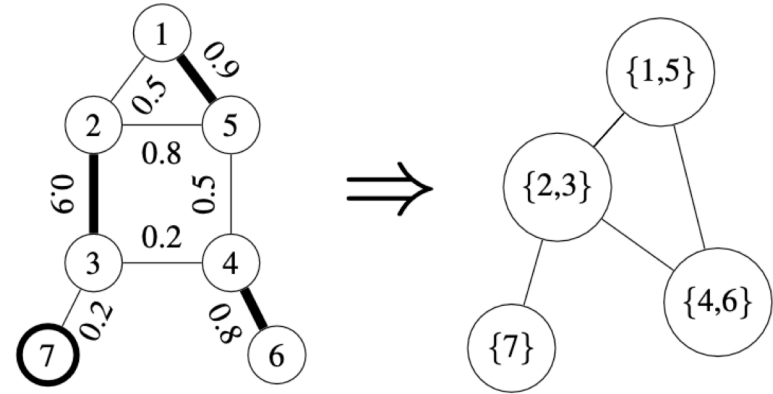


- Advantage: The features of every node are encoded into the pooled graph.
- Disadvantage: Each step roughly halves the number of nodes in the graph. This ratio is fixed and cannot be chosen.

Graph Pooling: Edge pooling

1. The edges are contracted in order of their score (edges {1, 5}, {4, 6}, {2, 3}), with nodes that have already been part of a pooled edge ignored (e.g., edge {2, 5}).
2. The new edges preserve the connectivity of the clusters in the old graph.
3. To enable the gradients to flow into the scores, the aggregated node features are multiplied by the edge score:

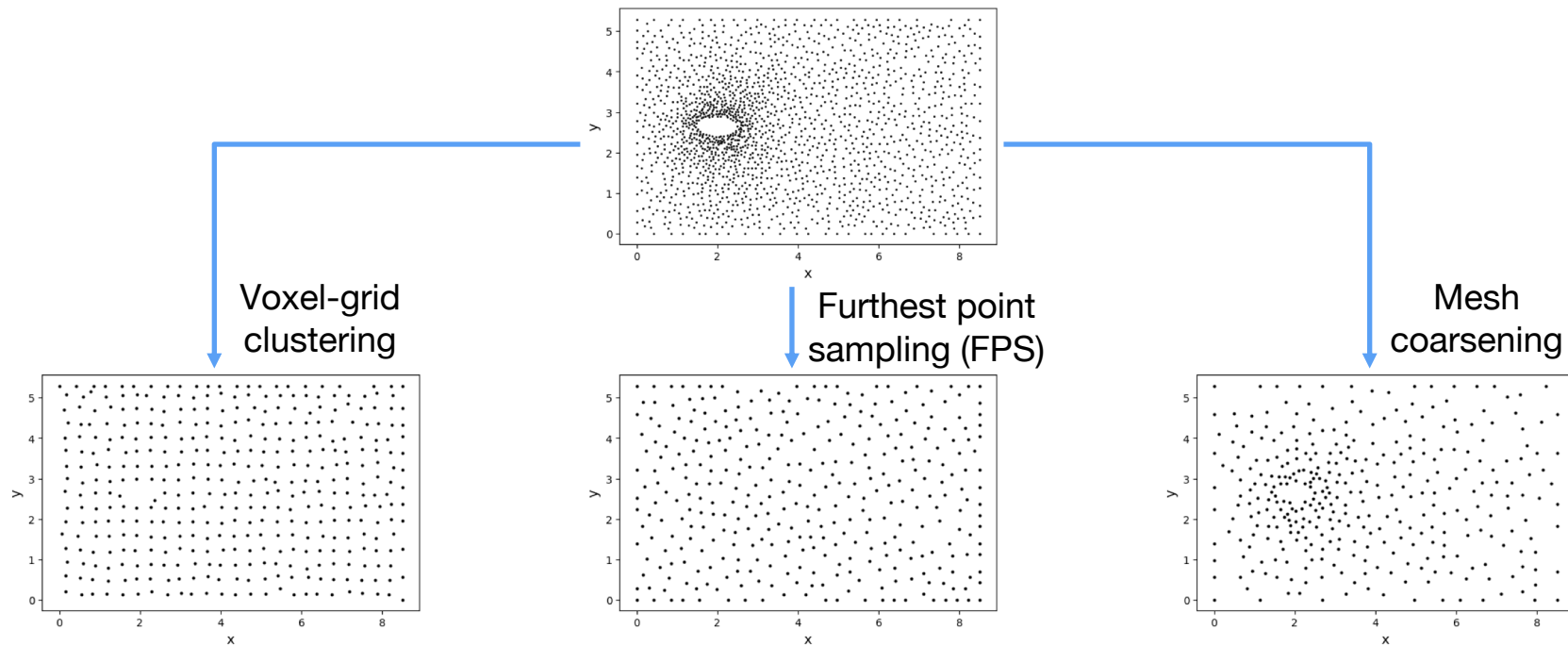
$$\mathbf{v}_{ij} \leftarrow s_{ij}(\mathbf{v}_i + \mathbf{v}_j)$$



- Advantage: The features of every node are encoded into the pooled graph.
- Disadvantage: Each step roughly halves the number of nodes in the graph. This ratio is fixed and cannot be chosen.

Graph Pooling: Non-learnable coarsening

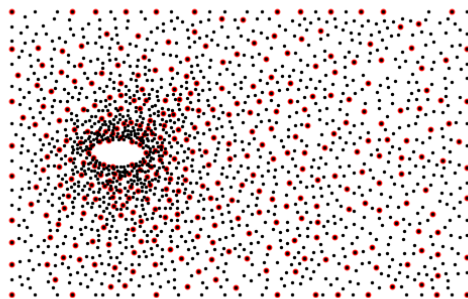
The coarsening does not depend on latent node-features, it depends only on the nodes position and/or connections. There are many options available. For instance:



Graph Pooling: Non-learnable coarsening

Assignment of new features for the nodes in V^2

- If the coarsening was of the node-dropping type we simply keep the features of the nodes that are not removed.



$$X^2 = X^1[\mathbf{idx}]$$

- If the coarsening was of the node-clustering type we can perform message passing from the child nodes to the parent node.
- This message passing can be learnable or non-learnable.

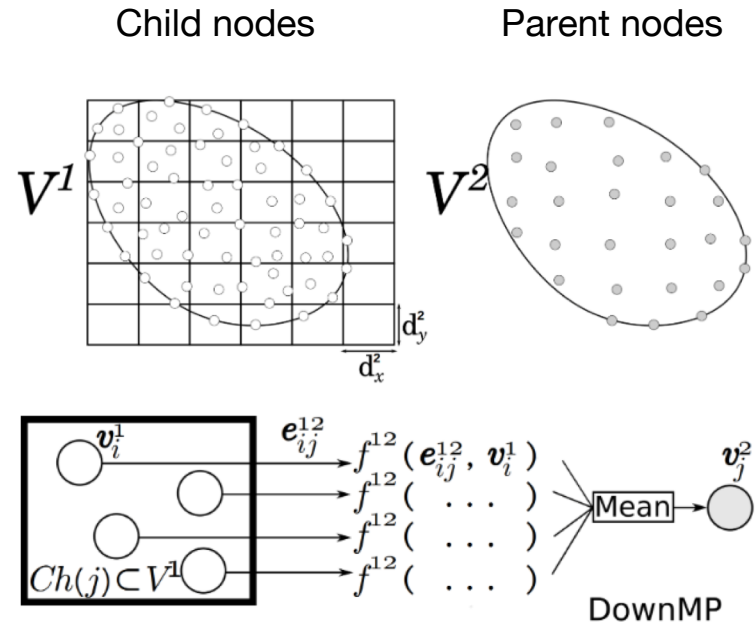
Graph Pooling: Non-learnable coarsening

Message passing from child nodes to parent nodes

- Each cluster has a set of child nodes in V^1 and it is assigned a parent node in V^2 , with the mean coordinates of its child nodes.
- Messages are sent from child nodes to parent node and aggregated:

$$v_j^2 \leftarrow \frac{1}{|Ch(j)|} \sum_{i \in Ch(j)} f^{12}(e_{ij}^{1,2}, v_i^1)$$

- The message function, f^{12} , can be any function, commonly an MLP.
- The attributes e^{12} are commonly the relative position of the parent and child nodes.



Graph Pooling: Non-learnable coarsening

- This form of graph pooling was proposed in *Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics* by Lino et al.
- The employed coarsening was voxel-grid clustering due to its efficiency.
- The new edges were defined by preserving the connectivity of the graph.
- The *new* edge-attributes e_{ij}^2 are the edge-wise mean of the attributes of the *old* edges between $Ch(i)$ to $Ch(j)$.
- It is also possible to use a non-learnable message passing from child nodes to parent node. For instance, the k -NN interpolation by Qi et al.:

$$\mathbf{v}_j^2 \leftarrow \frac{\sum_{i \in Ch(j)} c_i \mathbf{v}_i^1}{\sum_{i \in Ch(j)} c_i} \quad \text{with} \quad c_i = 1 / \|\mathbf{x}_j^2 - \mathbf{x}_i^1\|_2^2$$

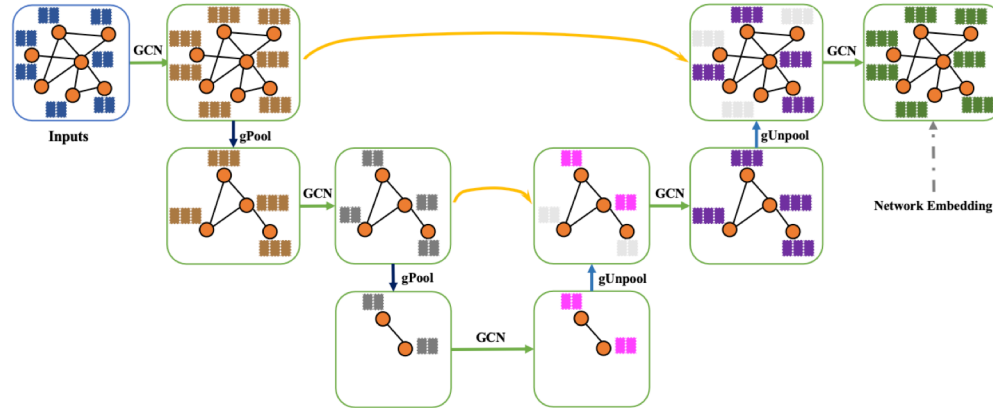
Graph Pooling: Non-learnable coarsening

- This form of graph pooling was proposed in *Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics* by Lino et al.
- The employed coarsening was voxel-grid clustering due to its efficiency.
- The new edges were defined by preserving the connectivity of the graph.
- The *new* edge-attributes e_{ij}^2 are the edge-wise mean of the attributes of the *old* edges between $Ch(i)$ to $Ch(j)$.
- It is also possible to use a non-learnable message passing from child nodes to parent node. For instance, the k -NN interpolation by Qi et al.:

$$\mathbf{v}_j^2 \leftarrow \frac{\sum_{i \in Ch(j)} c_i \mathbf{v}_i^1}{\sum_{i \in Ch(j)} c_i} \quad \text{with} \quad c_i = 1 / \|\mathbf{x}_j^2 - \mathbf{x}_i^1\|_2^2$$

Graph Unpooling

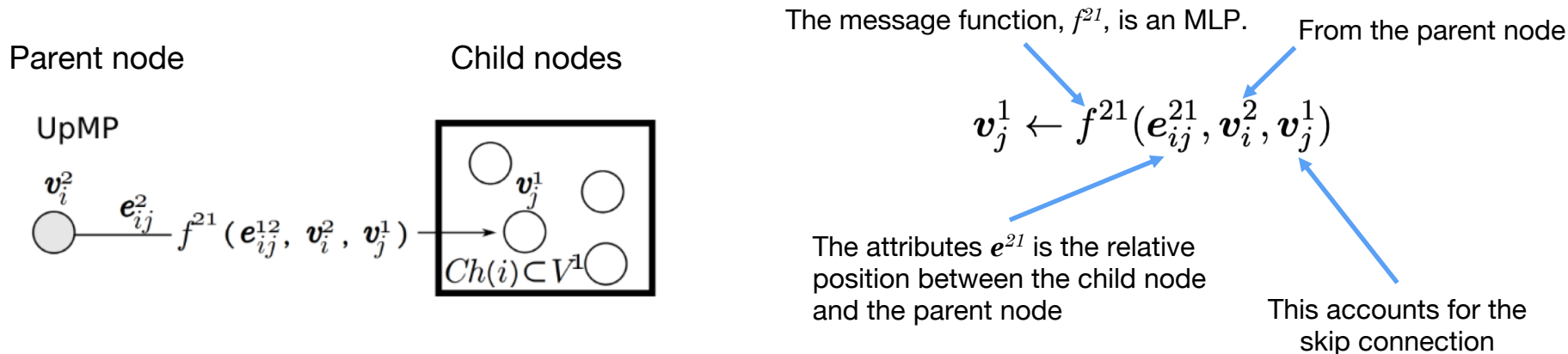
- For graph unpooling we restore the the nodes and edges that we had before.



- The edge features are also restored typically (from the skip connection).
- If pooling was done by node-dropping, we can unpool the features by adding or concatenating the low-resolution features with the high-resolution features from the skip connection (zeros are used for the low-resolution features of dropped nodes).

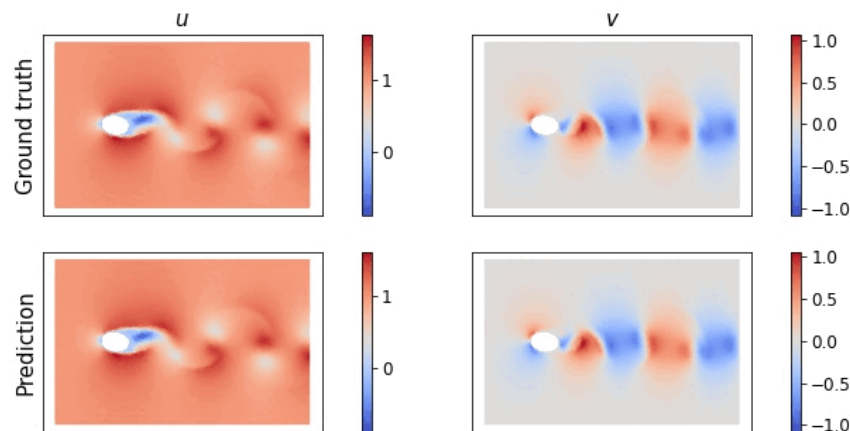
Graph Unpooling

- If pooling was done by node-clustering, we can perform the node-feature unpooling by parent node to child node message passing as proposed in Lino et al.



- Non-learnable options are also possible. E.g., directly assign to the child nodes the features of their parent (this is nearest neighbour interpolation).

Multi-scale rotation-equivariant GNNs for unsteady Eulerian fluid dynamics



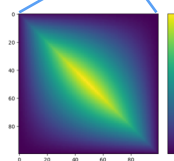
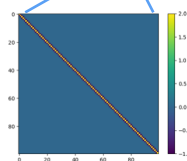
Multi-scale rotation-equivariant GNNs

- Modelling certain physical phenomena requires the use of message passing across multiples scales or levels of resolution.
- For instance, in elliptic PDEs (Poisson equation) the information is propagated at an infinite speed. Thus, the network must be able to communicate any two nodes, no matter their distance.

Poisson equation

Discretised equation

$$\nabla^2 p = f(x, y) \rightarrow L\mathbf{p} = \mathbf{f} \rightarrow \mathbf{p} = L^{-1}\mathbf{f}$$

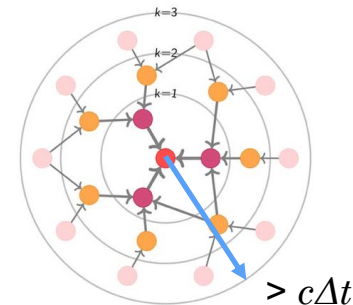


When computing the inverse of L we get a dense matrix

- To simulate fluid and structural dynamics we need to solve a Poisson equation.

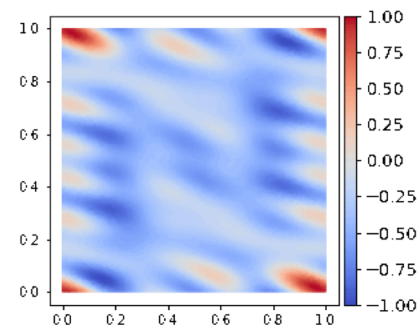
Multi-scale rotation-equivariant GNNs

- In advection problems, a *flat* GNN is enough **provided that the receptive field of every node is larger than the distance travelled by the information** ($c\Delta t$) – this is analogous to the Courant–Friedrichs–Lewy (CFL) condition.
- Lino et al. trained a GNN with 4 MP sequential layers against advection on a 1x1 periodic domain. This has a simple analytical solution.



$$\frac{\partial u}{\partial t} + c \frac{\partial u}{\partial x} = 0 \longrightarrow u(\mathbf{x}, t) = u_0(\mathbf{x} - c\mathbf{t})$$

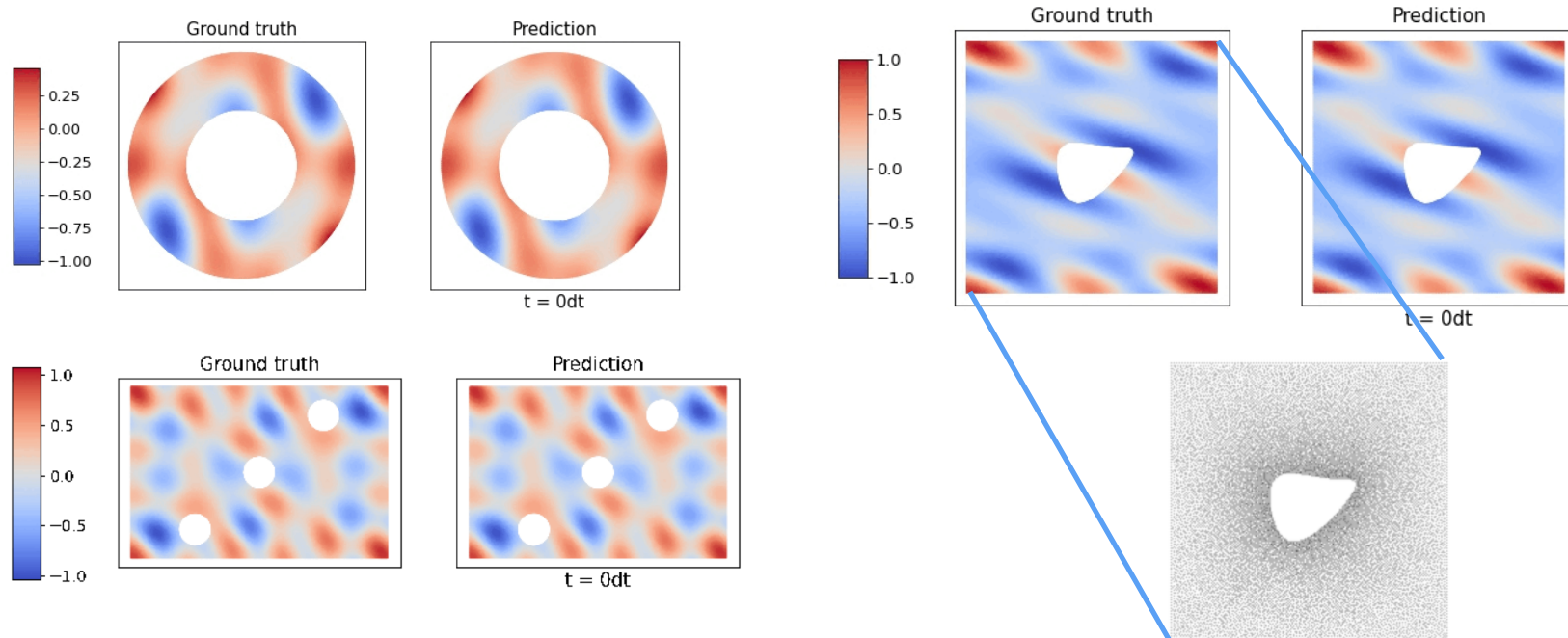
- Node inputs: $[u_i^n, c_i, \omega]$, Edge inputs: $\mathbf{x}_j - \mathbf{x}_i$
- Node output: u_i^{n+1}



The edges were built by k -NN

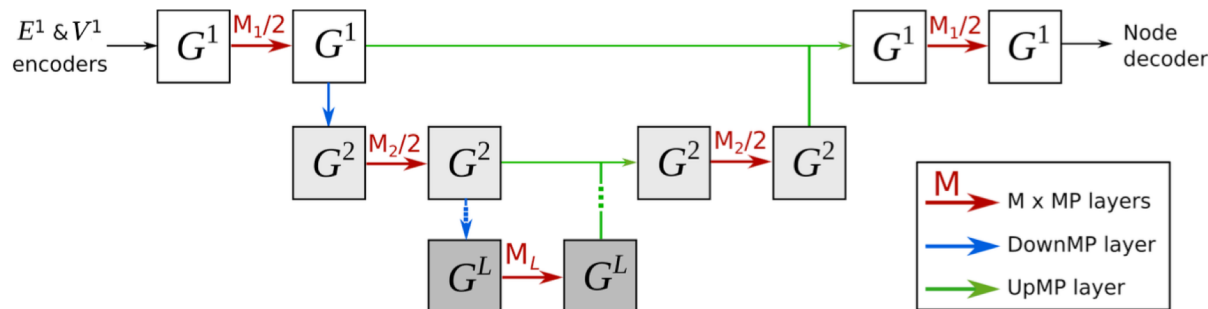
Multi-scale rotation-equivariant GNNs

- This simple GNN trained against trivial simulations generalised to more complex cases:



Multi-scale rotation-equivariant GNNs

- They tested other three models with $L = 2, 3$, and 4 scales and $M_l = 4$.



- Pooling: voxel-grid clustering and MP from child nodes to parent node. The pooled edge-features are the edge-wise mean of the features of the *old* edges between $Ch(i)$ to $Ch(j)$ (Slide 119).
- Unpooling: MP from parent node to child node (Slide 123).

- Adding MP layers at lower-resolution scales did not help to improve the accuracy. It even became worse due to over-fitting.
- This illustrates that a multi-scale architecture is not required for advection problems.

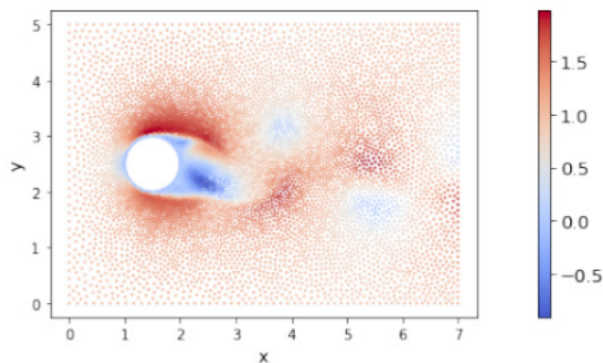
	$L = 1$	$L = 2$	$L = 3$	$L = 4$
AdvTaylor	0.9588	0.9472	0.9441	0.9350
AdvCircle	0.9880	0.9885	0.9877	0.9874
AdvCircleAng	0.9887	0.9851	0.9849	0.9839
AdvSquare	0.9872	0.9857	0.9844	0.9853
AdvEllipseH	0.9864	0.9850	0.9841	0.9842
AdvEllipseV	0.9860	0.9849	0.9840	0.9844
AdvSpline	0.9847	0.9828	0.9817	0.9818
AdvInCir	0.8720	0.4136	0.1605	0.2288

Multi-scale rotation-equivariant GNNs

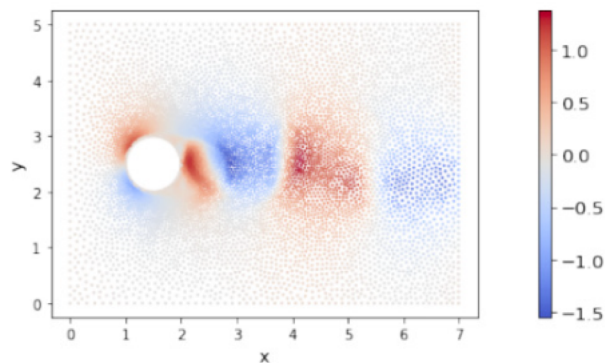
- They trained a GNN with 8 MP sequential layers to simulate incompressible flow around a circular cylinder.
- Node inputs: $[u_i^n, v_i^n, p_i^n, Re, \omega]$, Edge inputs: $\mathbf{x}_j - \mathbf{x}_i$
- Node output: $[u_i^{n+1}, v_i^{n+1}, p_i^{n+1}]$

Training: 500 -1000

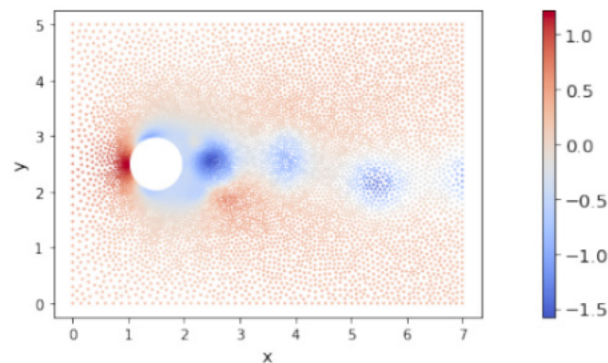
Testing: 300 - 500 and 1000 -1500



(a) $u(t_0, x_{V1}, y_{V1})$



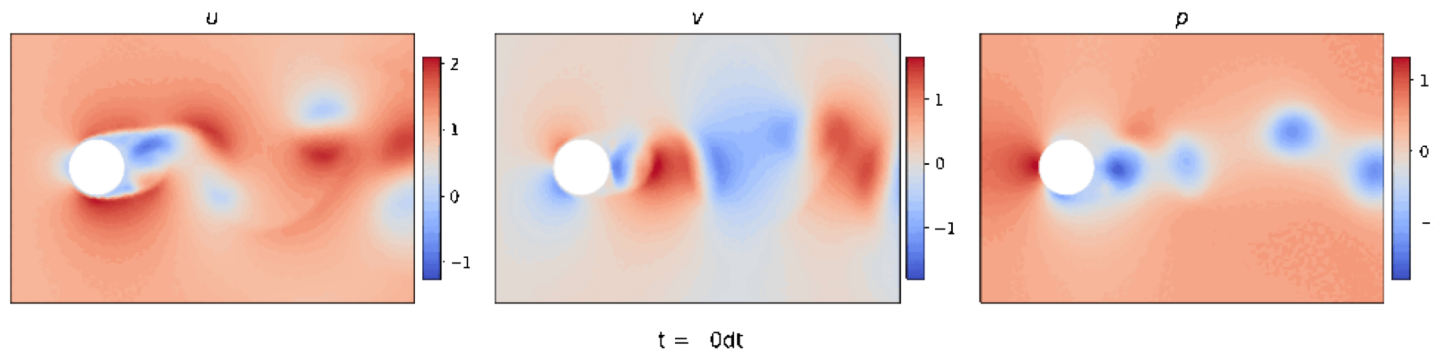
(b) $v(t_0, x_{V1}, y_{V1})$



(c) $p(t_0, x_{V1}, y_{V1})$

Multi-scale rotation-equivariant GNNs

- After 100 time-steps the velocity field is very inaccurate:



- Adding more MP layers could improve the accuracy by increasing the receptive field (Pfaff et al. 2021 – slide 99).
- Theoretically, we need the communication between any pair of nodes (the Navier-Stokes equations are non-local), which requires many MP layers.
- In larger domains the number of MP layers needed would be prohibitive.

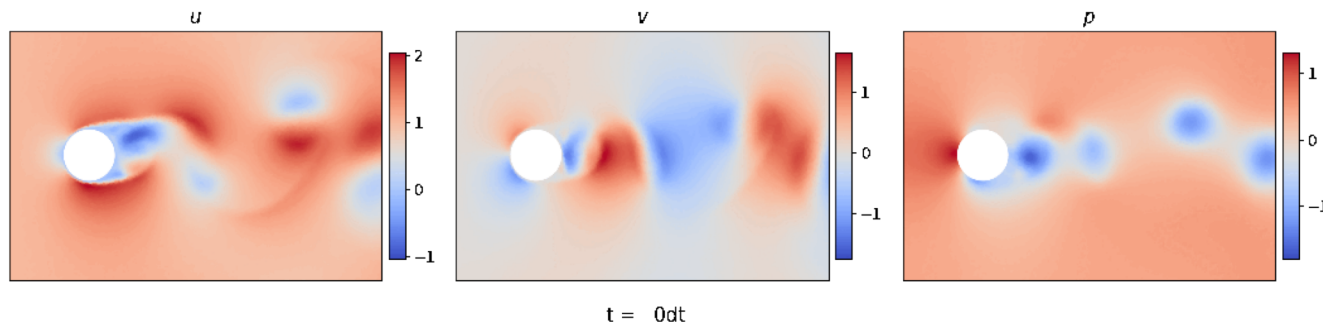
Multi-scale rotation-equivariant GNNs

- This was solved by using a U-Net-like multi-scale GNN.
- MP on lower resolution graphs allows to propagate the node features to farther regions.
- Besides, the time cost of MP in lower-resolution graphs is smaller than in the original graph (scales linearly with the number of edges).
- Comparison:

Number of scales →	$L = 1$			$L = 2$			$L = 3$			$L = 4$		
Dataset ↓	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2	R_u^2	R_v^2	R_p^2
NsCircleMidRe	0.7641	0.6544	0.7985	0.9320	0.9052	0.9314	0.9517	0.9315	0.9446	0.9569	0.9443	0.9530
NsCircleLowRe	0.7080	0.5178	0.7522	0.8711	0.7716	0.8744	0.8869	0.7830	0.9069	0.9589	0.9328	0.9533
NsCircleHighRe	0.4204	0.2065	0.5655	0.6432	0.3061	0.6546	0.7382	0.6388	0.7217	0.8269	0.7755	0.7900

Multi-scale rotation-equivariant GNNs

- A GNN with 4-2-4-2-4 MP layers was 40% faster than a flat GNN with 16 MP layers, and the accuracy was higher:



- Conclusion: Pay attention to the speed of information propagation in the physics you want to simulate.

Multi-scale rotation-equivariant GNNs (Additional Material)

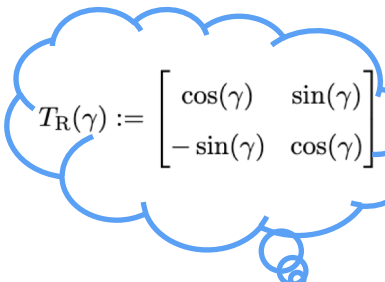
- DL-based simulators offer a poor generalisation to new geometries and parameters.
- **Data augmentation** (translation, rotation, etc.) is commonly employed to mitigate this.
- Rotation augmented data: During training we apply a randomly chosen rotation to all vector fields (inputs and targets).
- In the case of the Navier-Stokes equations:

Input edge-features:

$$\mathbf{e}_{ij}^1 \leftarrow T_R(\gamma) \mathbf{e}_{ij}^1$$

Input and output node-features:

$$\begin{bmatrix} u(t, x_i, y_i) \\ v(t, x_i, y_i) \end{bmatrix} \leftarrow T_R(\gamma) \begin{bmatrix} u(t, x_i, y_i) \\ v(t, x_i, y_i) \end{bmatrix}$$


$$T_R(\gamma) := \begin{bmatrix} \cos(\gamma) & \sin(\gamma) \\ -\sin(\gamma) & \cos(\gamma) \end{bmatrix}$$

- Data augmentation exposes the model to more diverse inputs, improving its generalisation.

Multi-scale rotation-equivariant GNNs (Additional Material)

- Besides, the model learns to **approximately** satisfy

$$\text{GNN}(\mathcal{R}(G)) \approx \mathcal{R}(\text{GNN}(G))$$

- The graph transformation \mathcal{R} applies a rotation to every vector magnitude provided as node or edge features.
- This condition, known as rotation equivariance, means that if the input vector fields are rotated an angle γ , then the output vector fields are rotated that same angle γ .
- Many PDEs satisfy this condition.
- Learning an approximately rotation-equivariant model could help to better learn the underlying the physics, which also explains for the better accuracy of these models.
- However, data augmentation is a **soft constraint**.

Multi-scale rotation-equivariant GNNs (Additional Material)

- A **hard constraint** ensures that the condition $\text{GNN}(\mathcal{R}(G)) = \mathcal{R}(\text{GNN}(G))$ is exactly satisfied.
- A model which, by design, guarantees rotation equivariance without relying on data augmentation or other training techniques could better learn the underlying physics, since the capacity of the network can be used to learn other properties.
- This is similar to the translation invariance of CNNs.
- Lino et al. designed a rotation-equivariant model by selecting **inputs that are independent of the orientation of the coordinate system** and which, at the same time, completely inform about the current fluid state.
- If all the inputs to a function (in this case a GNN) are independent of the location and orientation of the coordinate system the output is translation and rotation invariant:

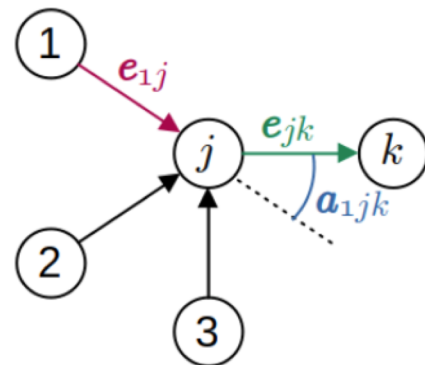
$$f(\mathcal{T} \circ \mathcal{R} \circ G) = f(G)$$

Multi-scale rotation-equivariant GNNs (Additional Material)

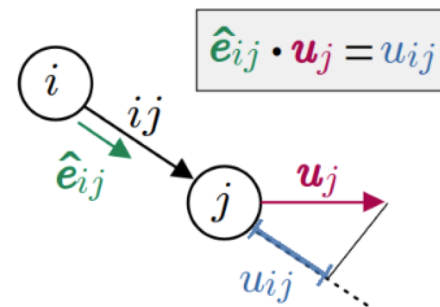
- They extend the definition of graph: $H^1 := (V^1, E^1, A^1)$
- $A^1 := \{(i, j, k) \mid (i, j), (j, k) \in E^1\}$ is a set of **directed angles** between pairs of adjacent incoming-outgoing edges.
- Input angle-features:

$$\mathbf{a}_{ijk}^1 := [|\mathbf{x}_j - \mathbf{x}_i|_2, |\mathbf{x}_k - \mathbf{x}_j|_2, \cos(\alpha_{ijk}), \sin(\alpha_{ijk})]$$
- Input edge-features: $\mathbf{e}_{ij}^1 := [\mathbf{u}_{ij}, Re_j, \omega_j]$

Projection of the input velocity along edge (i, j)
- The angle features inform about the location of the nodes and the node features about scalar and vector physical magnitudes. There are not node features.



$(1, j, k)$ is an angle directed from edge $(1, j)$ to edge (j, k) and vector \mathbf{a}_{1jk} contains its attributes



Multi-scale rotation-equivariant GNNs (Additional Material)

- They employed a modified definition of message passing to account for the angles.
- Instead of node to node messages now we have edge to edge messages:

1st Edge update: $e_{ij} \leftarrow \text{MLP}^e(e_{ij}, v_i, v_j)$

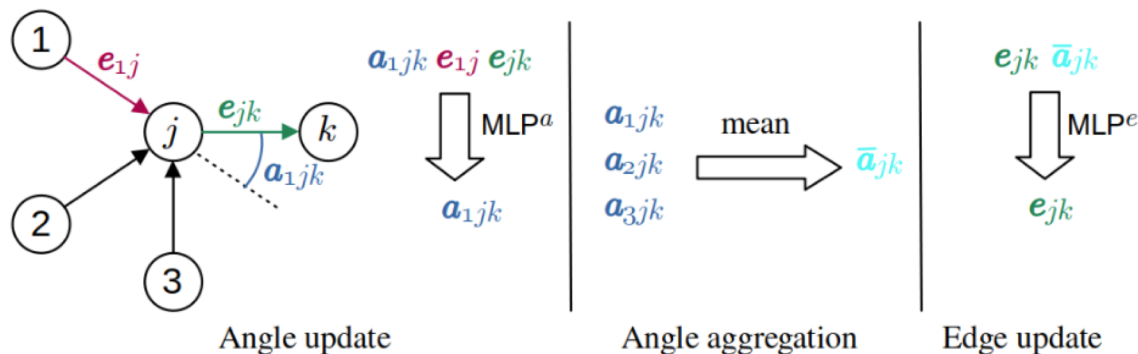
2nd Edge aggregation: $\bar{e}_j \leftarrow \frac{1}{|\mathcal{N}_j^-|} \sum_{i \in \mathcal{N}_j^-} e_{ij}$

3rd Node update: $v_j \leftarrow \text{MLP}^v(\bar{e}_j, v_j)$

1st - Angle update: $a_{ijk}^l \leftarrow \text{MLP}^a(a_{ijk}^l, e_{ij}^l, e_{jk}^l)$

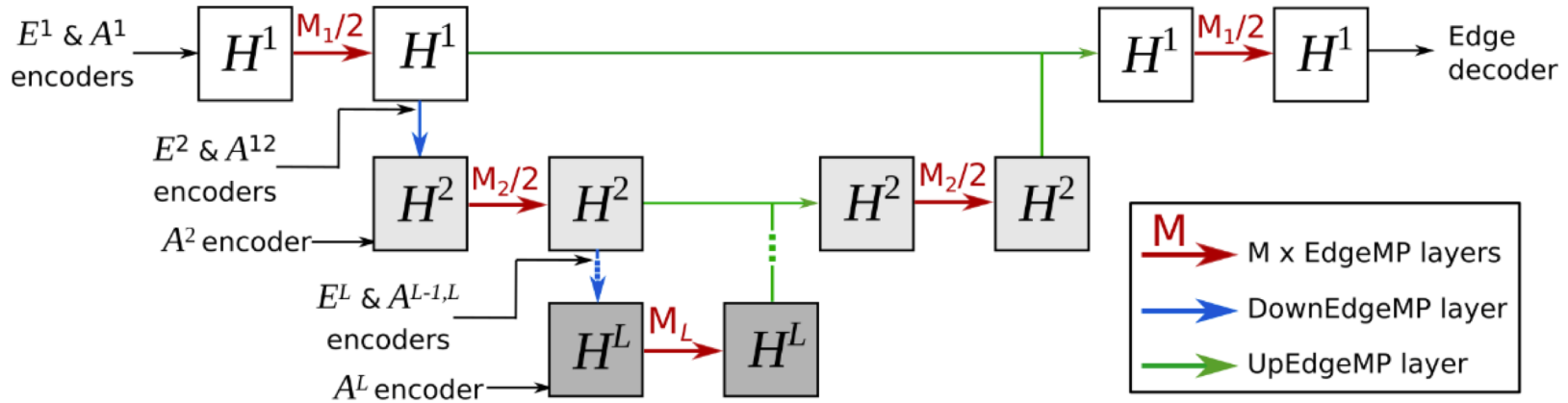
2nd - Angle aggregation: $\bar{a}_{jk}^l \leftarrow \frac{1}{j} \sum_{i \in \mathcal{N}_j^-} a_{ijk}^l$

3rd - Edge update: $e_{jk}^l \leftarrow \text{MLP}^e(e_{jk}^l, \bar{a}_{jk}^l)$



Multi-scale rotation-equivariant GNNs (Additional Material)

- This rotation-equivariant GNN also follows a U-Net-like architecture:

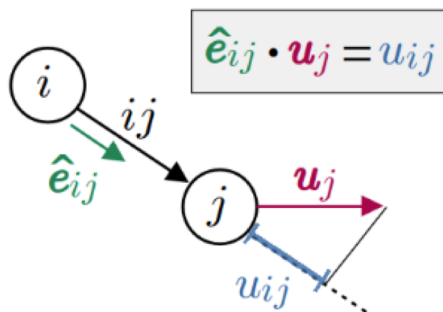


- New pooling and unpooling layers are also required to work with angle and edge features.
- At each edge (i, j) , the output is $u_{ij}(t_0 + \Delta t)$, which represents $\hat{e}_{ij} \cdot u_j(t_0 + \Delta t)$.

Multi-scale rotation-equivariant GNNs (Additional Material)

- At each edge (i, j) , the output is $u_{ij}(t_0 + \Delta t)$, which represents $\hat{e}_{ij} \cdot \mathbf{u}_j(t_0 + \Delta t)$.

Remember...



- Let's consider a node j with 3 incoming edges: $(1, j)$, $(2, j)$ and $(3, j)$.
- Then, we have 3 projection equations:

$$\begin{bmatrix} \text{---} & \left(\hat{e}_{1j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{2j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{3j}^1 \right)^T & \text{---} \end{bmatrix} \begin{bmatrix} u_j(t_0 + \Delta t) \\ v_j(t_0 + \Delta t) \end{bmatrix} = \begin{bmatrix} u_{1j}(t_0 + \Delta t) \\ u_{2j}(t_0 + \Delta t) \\ u_{3j}(t_0 + \Delta t) \end{bmatrix}$$

Dimensions: $[3 \times 2]$ $[2 \times 1]$ $[3 \times 1]$

- Solve this **overdetermined** system of equations to get the **predicted velocity** from the **edge outputs**.

Multi-scale rotation-equivariant GNNs (Additional Material)

- This overdetermined system is solved by the method of least squares:

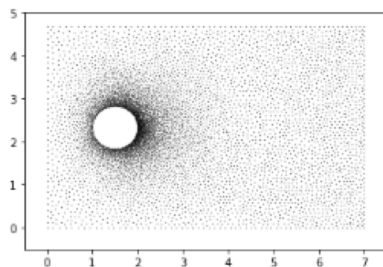
$$\begin{bmatrix} u_j(t_0 + \Delta t) \\ v_j(t_0 + \Delta t) \end{bmatrix} = \begin{bmatrix} \text{---} & \left(\hat{e}_{1j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{2j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{3j}^1 \right)^T & \text{---} \end{bmatrix}^+ \begin{bmatrix} u_{1j}(t_0 + \Delta t) \\ u_{2j}(t_0 + \Delta t) \\ u_{3j}(t_0 + \Delta t) \end{bmatrix}$$

- The Moore-Penrose pseudoinverse matrices of each node can be computed in a pre-processing step.
- It will come in handy to define the *projection-aggregation function*, ρ_j^1 , for each node j :

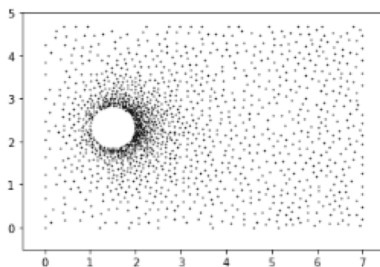
$$\rho_j^1(\mathbf{x}) = \begin{bmatrix} \text{---} & \left(\hat{e}_{1j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{2j}^1 \right)^T & \text{---} \\ \text{---} & \left(\hat{e}_{3j}^1 \right)^T & \text{---} \end{bmatrix}^+ \mathbf{x} \xrightarrow{\text{SO}} \begin{bmatrix} u_j(t_0 + \Delta t) \\ v_j(t_0 + \Delta t) \end{bmatrix} = \rho_j^1 \left(\begin{bmatrix} u_{1j}(t_0 + \Delta t) \\ u_{2j}(t_0 + \Delta t) \\ u_{3j}(t_0 + \Delta t) \end{bmatrix} \right)$$

Multi-scale rotation-equivariant GNNs (Additional Material)

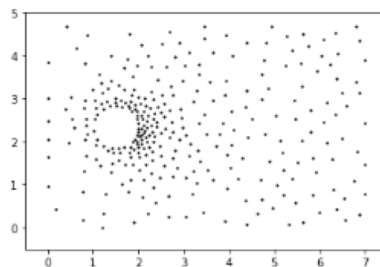
- Let's have a look to the pooling and unpooling operations.
- Low-resolution nodes obtained by Guillard's coarsening (non-learnable node-dropping):



$$|V^1| = 6945$$



$$|V^2| = 3068$$



$$|V^3| = 610$$

- Low-resolution edges (E^2 and E^3) are created by k -nearest neighbours.
- Angles in A^2 and A^3 are created between pairs of adjacent incoming-outgoing edges in E^2 and E^3 , respectively:

$$A^l := \{(i, j, k) \mid (i, j), (j, k) \in E^l\}$$

Multi-scale rotation-equivariant GNNs (Additional Material)

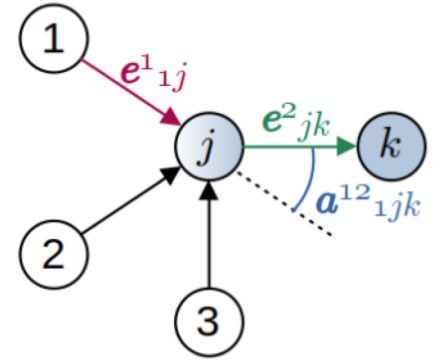
- The low-resolution edges and angles are assigned input features defined in the same way as for the high-resolution graph (see [Slide 136](#)).
- We also define a set of angles, A^{12} , connecting incoming edges in E^1 and outgoing edges in E^2 .
- **Pooling:** The edges in E^1 send messages to the edges in E^2 through the angles in A^{12} :

1st - Angle update: $\mathbf{a}_{ijk}^{12} \leftarrow \text{MLP}_a^{12}(\mathbf{a}_{ijk}^{12}, \mathbf{e}_{ij}^1, \mathbf{e}_{jk}^2)$

2nd - Angle aggregation: $\bar{\mathbf{a}}_{jk}^{12} \leftarrow \frac{1}{\kappa} \sum_{i \in \mathcal{N}_j^-} \mathbf{a}_{ijk}^{12}$

3rd - Edge update (with skip connection): $\mathbf{e}_{jk}^2 \leftarrow \text{MLP}_e^{12}(\mathbf{e}_{jk}^2, \bar{\mathbf{a}}_{jk}^{12})$

- This is possible because each low-resolution edge has at least one high-resolution incoming edge (k to be exact).



Nodes 1, 2, 3, $j, k \in V^1$

Nodes $j, k \in V^2$

Edges $(1, j), (2, j), (3, j) \in E^1$

Edge $(j, k) \in E^2$

Angle $(i, j, k) \in A^{12}$

Multi-scale rotation-equivariant GNNs (Additional Material)

- Not every high-resolution edge has at least one adjacent low-resolution edge.
 - Thus for **unpooling** we express the low-resolution edge-features as low-resolution node-features, interpolate to the high-resolution nodes, and express the interpolated features as edge features again:
1. Transform the (scalar) edge features into (vector) node features using the projection-aggregation function:

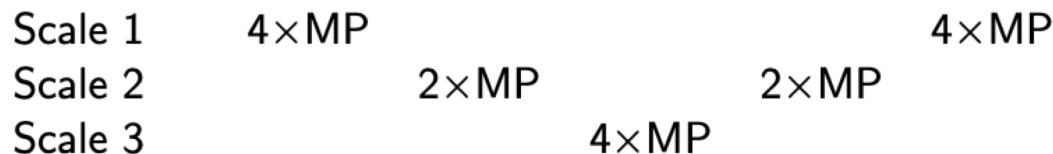
$$Q_j^2 = \left[\rho_j^2 \left(\begin{bmatrix} e_{1,j}^2[1] \\ \vdots \\ e_{\kappa,j}^2[1] \end{bmatrix} \right) \quad \rho_j^2 \left(\begin{bmatrix} e_{1,j}^2[2] \\ \vdots \\ e_{\kappa,j}^2[2] \end{bmatrix} \right) \quad \cdots \quad \rho_j^2 \left(\begin{bmatrix} e_{1,j}^2[F] \\ \vdots \\ e_{\kappa,j}^2[F] \end{bmatrix} \right) \right]$$

2. Interpolate Q^2 from V^2 to $V^1 \rightarrow Q^1$
3. Project the columns of Q_k^1 along the direction of the incoming edges: $\mathbf{q}_{lk}^1 = (\hat{e}_{lk}^2)^T Q_k^1$
4. Update the edge features: $e_{lk}^1 \leftarrow \text{MLP}_{\text{skip}}^{21}(e_{lk}^1, \mathbf{q}_{lk}^1)$

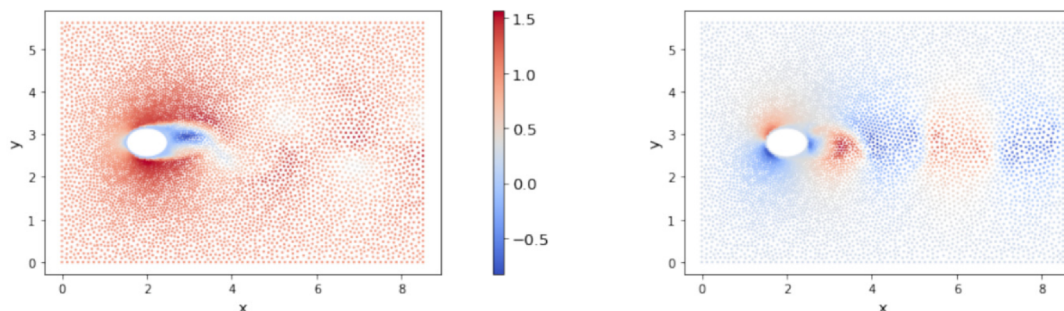
Multi-scale rotation-equivariant GNNs (Additional Material)

Experiments

- Rotation-equivariant multi-scale GNN with 3 scales:

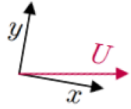


- Train to simulate incompressible flow around an elliptical cylinder.

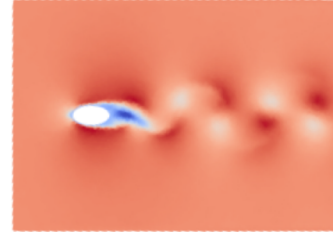
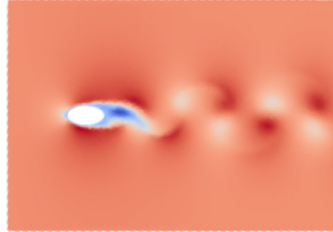


- Dataset parameters: $Re \in [500, 1000]$, $b/\alpha \in [0.5, 0.8]$ and $W \in [5, 6]$

Multi-scale rotation-equivariant GNNs (Additional Material)

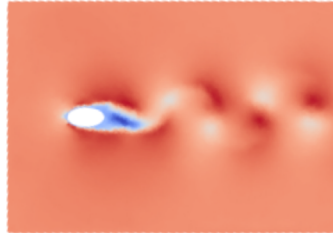


Ground truth

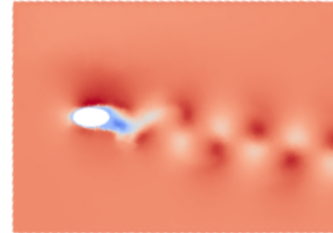


REMuS-GNN

MuS-GNN with
data augmentation



MuS-GNN without
data augmentation

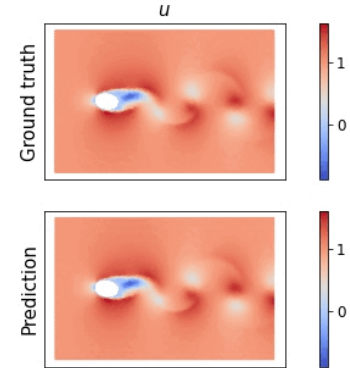


- If we rotate the system of coordinates, a non-rotation-equivariant model trained without rotated data keeps predicting a vortex street parallel to the x -axis instead of parallel to the free stream.

Multi-scale rotation-equivariant GNNs (Additional Material)

- Generalisation:

Model →	REMuS-GNN			MuS-GNN		
Dataset ↓	R_u^2	R_v^2	$MAE_{x_{sp}}(\%)$	R_u^2	R_v^2	$MAE_{x_{sp}}(\%)$
NsEllipseVal	0.9621	0.8960	2.75	0.9436	0.8552	3.97
NsEllipseLowRe	0.9322	0.7306	3.08	0.8566	0.4499	4.42
NsEllipseHighRe	0.7082	0.1039	6.46	-0.0008	-0.8769	7.52
NsEllipseThin	0.8883	0.2329	2.60	0.8388	0.0389	2.98
NsEllipseThick	0.9316	0.8999	4.49	0.9168	0.8849	4.87
NsEllipseNarrow	0.9557	0.8678	2.93	0.9486	0.8742	3.83
NsEllipseWide	0.9437	0.8478	2.96	0.9341	0.8395	3.96
NsEllipseAoA	0.9519	0.8622	3.22	0.9327	0.8164	4.39



- Long-term stability:

