Multi-Scale Rotation-Equivariant Graph Neural Networks for Unsteady Eulerian Fluid Dynamics

Mario Lino[1,2] Chris Cantwell[2] Stathi Fotiadis[2] Anil Bharath[2]

[1]Technical University of Munich [2]Imperial College London

Imperial College London

Workshop on Machine Learning for Fluid Dynamics, Sorbonne University

8th March 2024

Contents

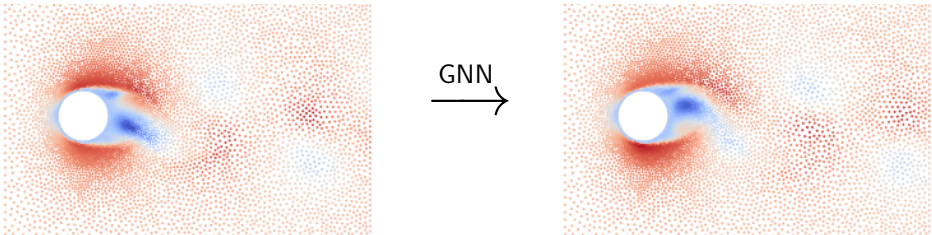
Graph Neural Network based time-steppers

Common problem in science and engineering: Given a system of PDEs

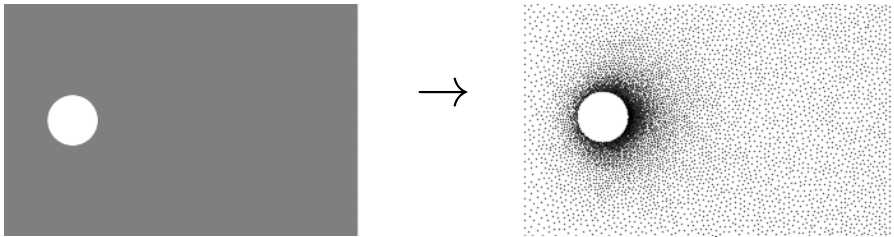
$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{F}(\boldsymbol{u}, \zeta_1, \zeta_2, \ldots, \zeta_n), \;\; \boldsymbol{x} \in \Omega, \;\; t > t_0,$$

infer the temporal evolution of the solution field $\boldsymbol{u}(t, \boldsymbol{x})$

- DL-based simulators are 10 to 10^4 times **faster than CFD solvers**
- Use a **GNN to infer the solution at the next time-point**



Graph representation of the fluid



- Attributes of node i:

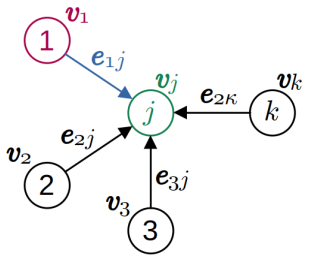
$$v_i = [\boldsymbol{u}(t_0, \boldsymbol{x}_i), \zeta_1(t_0, \boldsymbol{x}_i), \zeta_2(t_0, \boldsymbol{x}_i), \ldots, \zeta_n(t_0, \boldsymbol{x}_i), d_i]$$

- Attributes of edge (i, j):

$$\boldsymbol{e}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$$

Message passing on graphs

- **Message passing** (MP) from Battaglia et al. (2018):

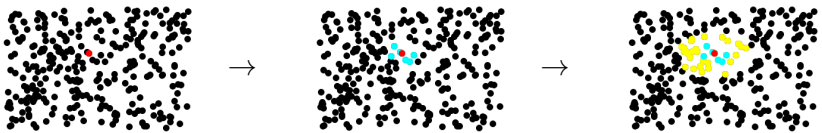


1st Edge update: $\boldsymbol{e}_{ij} \leftarrow \text{MLP}^e(\boldsymbol{e}_{ij}, \boldsymbol{v}_i, \boldsymbol{v}_j)$

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

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

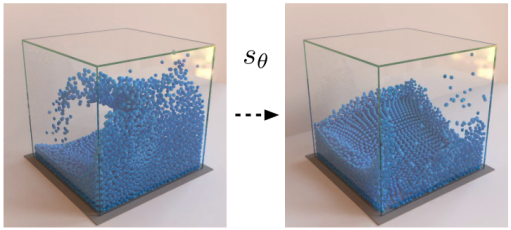
- Repeat MP to process and propagate the node features:



Battaglia et al. *Relational inductive biases, deep learning, and graph networks.* 2018.

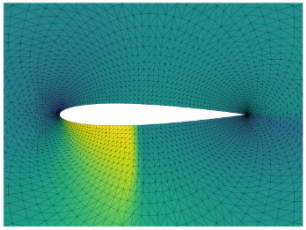
Previous Work: MP for fluid simulation

- Previous work has used MP on graphs for particle- and mesh-based simulations



Sanchez-Gonzalez et al. 2018

Learning to Simulate Complex Physics with Graph Networks

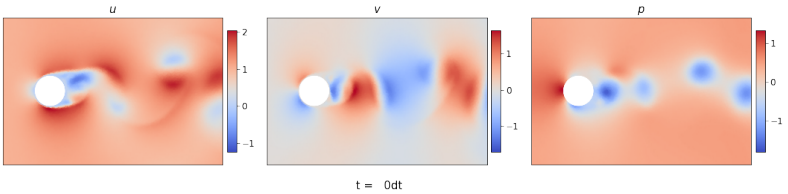


Pfaff et al. 2020

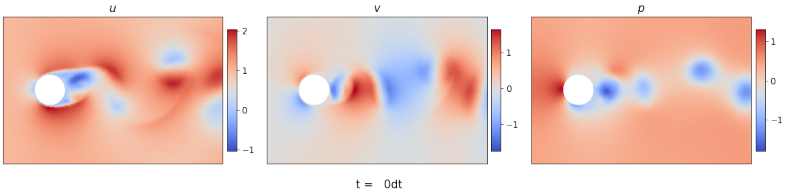
Learning Mesh-Based Simulation with Graph Networks

Previous Work: Multi-scale MP

- Single-scale MP (16 MP layers):



- Three-scale MP (16 MP layers):



Lino et al. *Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics.* 2022.

The generalisation challenge

- DL-based simulators offer a **poor generalisation** to new geometries and parameters
- Data augmentation (translations, **rotations**, etc.) is commonly employed to mitigate this
- With rotation-augmented data, the model learns to satisfy

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

We propose a GNN which by design guarantees the exact satisfaction of

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

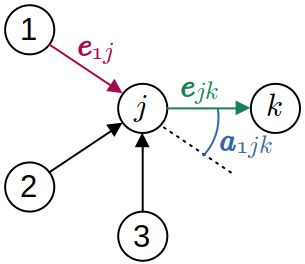
1st No need to learn the rotation equivariance $\rightarrow$ **the capacity of the network can be used to better learn other features**

2nd **Better accuracy, generalisation and long-term stability**

Contents

How is rotation equivariance enforced in REMuS-GNN?

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



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

- Extended graph data-structure:

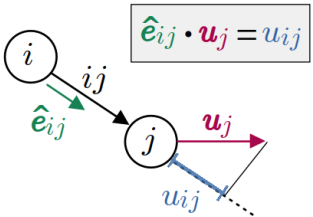
$$\boxed{H^1 := (V^1, E^1, \boldsymbol{A^1})}$$

- A^1 is a set of directed angles between pairs of adjacent edges

$$A^1 := \{(i, j, k) \mid (i, j), (j, k) \in E^1\}$$

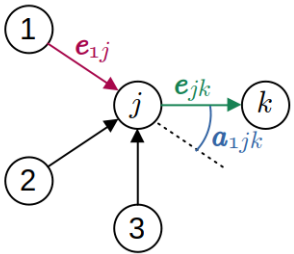
- H^1 is assigned input edge- and angle-attributes (not node attributes)

Input attributes of REMuS-GNN



For the incompressible N-S equations:

- Input edge-attributes: $e^1_{ij} := [u_{ij}, Re_j, d_j]$
- u_{ij} is the projection of $u(t_0, x_j)$ along the direction of edge (i, j)



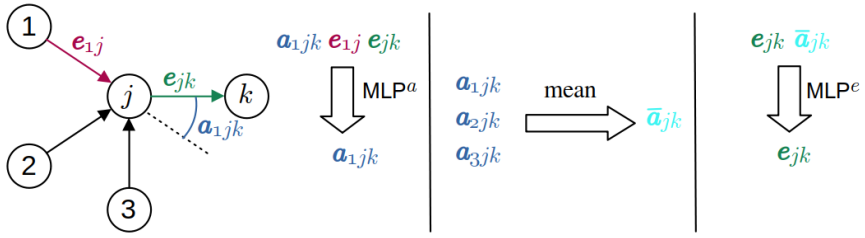
- Input angle-attributes:

$$a^1_{ijk} := [|x_j - x_i|_2, |x_k - x_j|_2, \cos(\alpha_{ijk}), \sin(\alpha_{ijk})]$$

All the inputs are independent of the location and orientation of the coordinate system

Message passing on H

Modified MP layer based on the directional MP algorithm by Klicpera et al. (2020) and the MP layer by Battaglia et al. (2018)



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

→

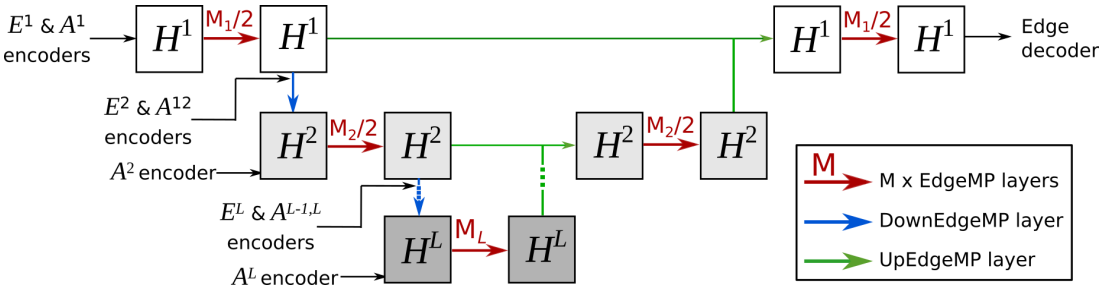
1st - Angle update: $a_{ijk}^{\ell} \leftarrow \text{MLP}^a(a_{ijk}^{\ell}, e_{ij}^{\ell}, e_{jk}^{\ell})$

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

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

REMuS-GNN architecture

- REMuS-GNN follows a U-Net-like architecture

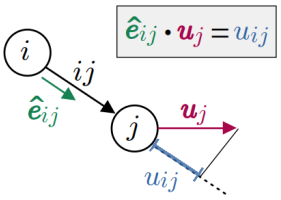


- New downsampling and upsampling layers are also required to work with angle and edge attributes

How to recover a vector field as output?

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

Remember...



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

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

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

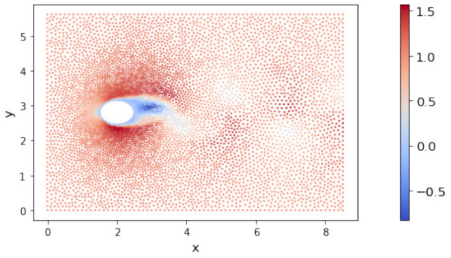
Contents

Training REMuS-GNN for flow inference

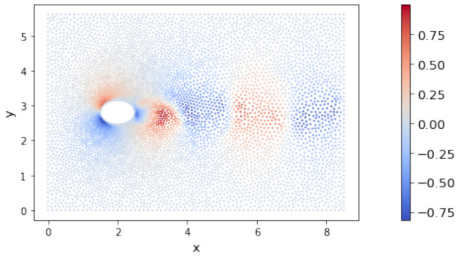
- Three-scale GNN ($\sim$2M parameters) with MP layers distributed according to:

Scale 1	4×MP		4×MP
Scale 2	2×MP	2×MP	
Scale 3	4×MP		

- Train to simulate incompressible flow around an elliptical cylinder



(a) $u(t_0, x_{V^1}, y_{V^1})$



(b) $v(t_0, x_{V^1}, y_{V^1})$

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

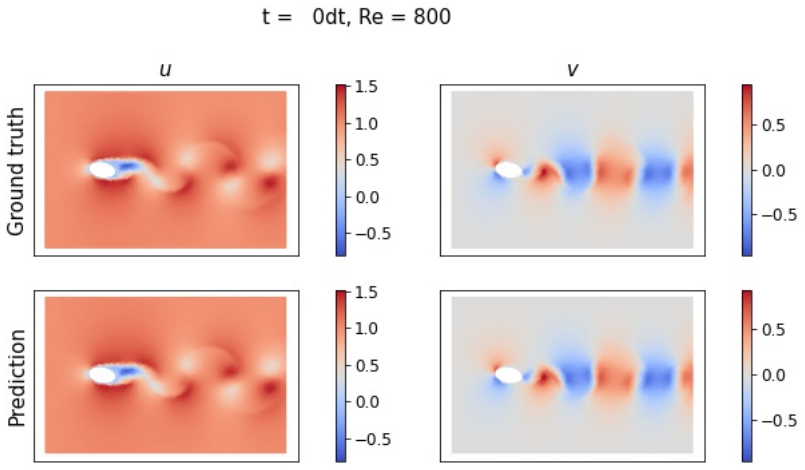
Improved generalisation

Coefficient of determination of u and v on testing datasets (averaged over 100 time-steps)

Model → / Dataset ↓	REMuS-GNN R_u^2	REMuS-GNN R_v^2	MuS-GNN R_u^2	MuS-GNN R_v^2
Validation	**0.9621**	**0.8960**	0.9436	0.8552
Low Re	**0.9322**	**0.7306**	0.8566	0.4499
High Re	**0.7082**	**0.1039**	-0.0008	-0.8769
High aspect ratio	**0.8883**	**0.2329**	0.8388	0.0389
Low aspect ratio	**0.9316**	**0.8999**	0.9168	0.8849
Narrow channel	**0.9557**	0.8678	0.9486	**0.8742**
Wide channel	**0.9437**	**0.8478**	0.9341	0.8395
Angle of attack	**0.9519**	**0.8622**	0.9327	0.8164

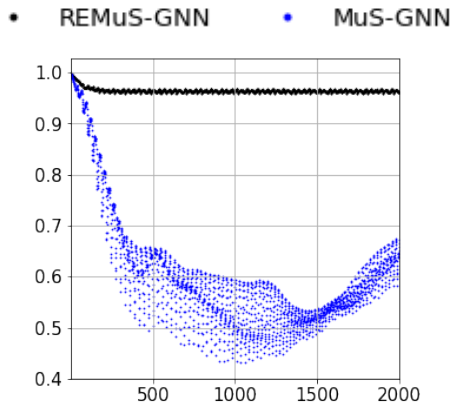
Improved generalisation

Simulation with AoA = 10 deg



Improved long-term stability

The R^2 remains indefinitely above 0.95



Contents

Conclusions

- Rotation equivariance was **enforced by working with invariant features**: scalar magnitudes, projected vector magnitudes, the distance between nodes and the angle between edges.
- This has helped to learn the underlying physics better and consequently **improve the generalisation and long-term stability**.
- It can applied to other tasks (e.g., turbulence modelling)

Corresponding publication

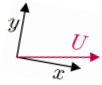
Lino, M., Fotiadis, S., Bharath, A. A., & Cantwell, C. D. (2022). *Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics*. Physics of Fluids, 34(8).

GitHub repo

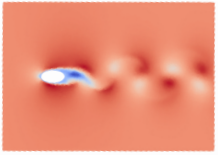
https://github.com/mario-linov/graphs4cfd

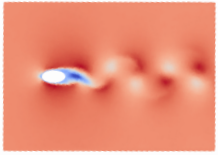
Rotation equivariance

Let's rotate the system of coordinates...



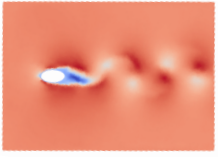
Ground truth

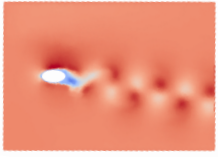




REMuS-GNN

MuS-GNN with
data augmentation

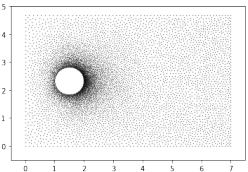




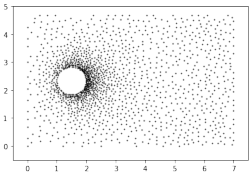
MuS-GNN without
data augmentation

DownEdgeMP: Edge-attributes downsampling layer

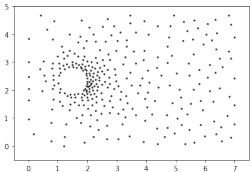
- Let's first define the low-resolution $H^\ell := (V^\ell, E^\ell, A^\ell)$
- Low-resolution nodes obtained with a modified Guillard's coarsening (Guillard et al. 1993): $V^4 \subset V^3 \subset V^2 \subset V^1$



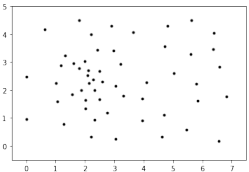
$|V^1| = 6945$



$|V^2| = 3068$



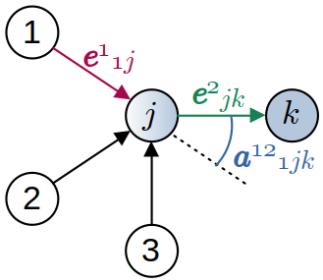
$|V^3| = 610$



$|V^4| = 108$

- Low-resolution edges created by k-nearest neighbours: E^2, E^3 and E^4
- Angles in A^2, A^3 and A^4 are created between pairs of adjacent edges in E^2, E^3 and E^4 respectively
- These edges and angles are also assigned input attributes

DownEdgeMP: Edge-attributes downsampling layer



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

High-resolution edges in E^1 pass messages to low-resolution edges in E^2 via the angles in A^{12}

Similar to EdgeMP:

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

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

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

This is possible because each low-resolution edge has at least one incoming edge – in this case there are κ

UpEdgeMP: Edge-attributes upsampling layer

1st - Transform edge attributes $\boldsymbol{e}_{ij}^2 \in \mathbb{R}^F$ into node attributes $Q_j^2 \in \mathbb{R}^{2\times F}$ using the projection-aggregation function ρ_j^2 of each node $j \in V^2$:

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

2nd - Interpolate Q^2 from V^2 to $V^1 \rightarrow Q^1 \in \mathbb{R}^{2\times F}$

3rd - Project the columns of Q_k^1 along the direction of the κ incoming edges at each node k:

$$\boldsymbol{q}_{lk}^1 = \left(\hat{\boldsymbol{e}}_{lk}^2\right)^T Q_k^1 \qquad \in \mathbb{R}^F$$

4th - Update the edge features:

$$\boldsymbol{e}_{lk}^1 \leftarrow \mathsf{MLP}_{\mathsf{skip}}^{21}(\boldsymbol{e}_{lk}^1, \boldsymbol{q}_{lk}^1)$$