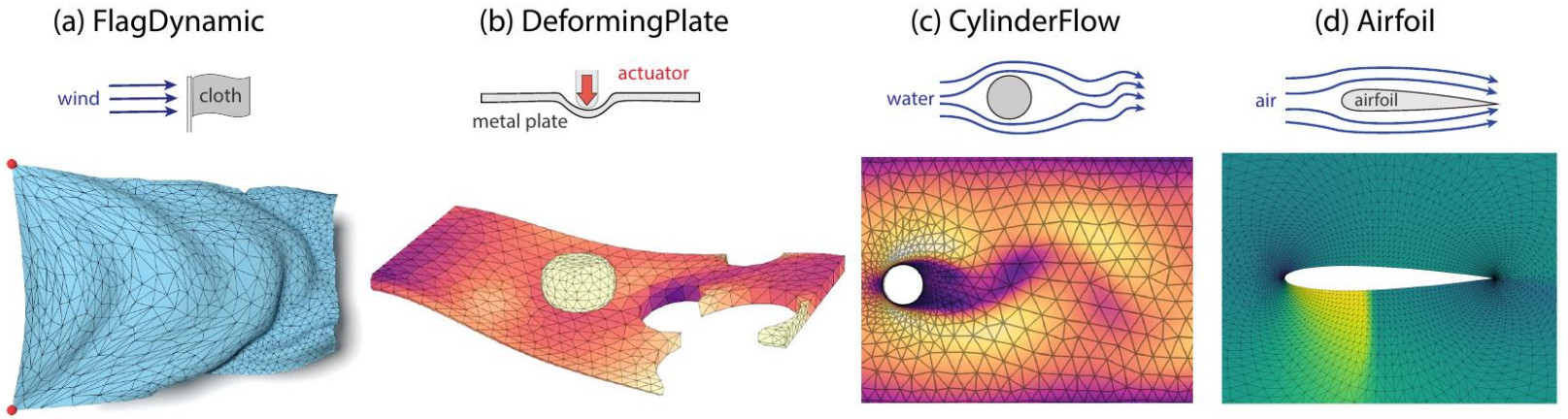


ML FOR SCIENCE · LECTURE 13

GNNs and Mesh-Based Learning

Aparna Chandramowlishwaran · EECS 195/221 · Fall 2026

Unstructured meshes in simulation



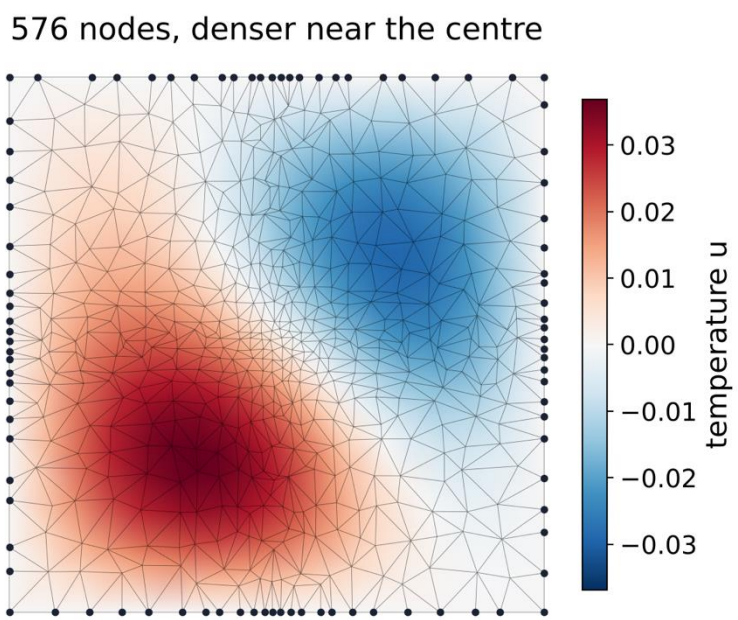
- Lecture 12: the FFT in an FNO requires a **uniform grid**
- Finite-element and finite-volume solvers use **unstructured meshes**: **small cells** where the solution changes quickly (walls, wakes, fronts), **large cells** elsewhere
- The mesh **changes from one simulation to the next**: a new airfoil shape, a cloth that moves, particles that flow
- Interpolating to a uniform grid wastes cells on smooth regions and **loses the thin layers near walls**

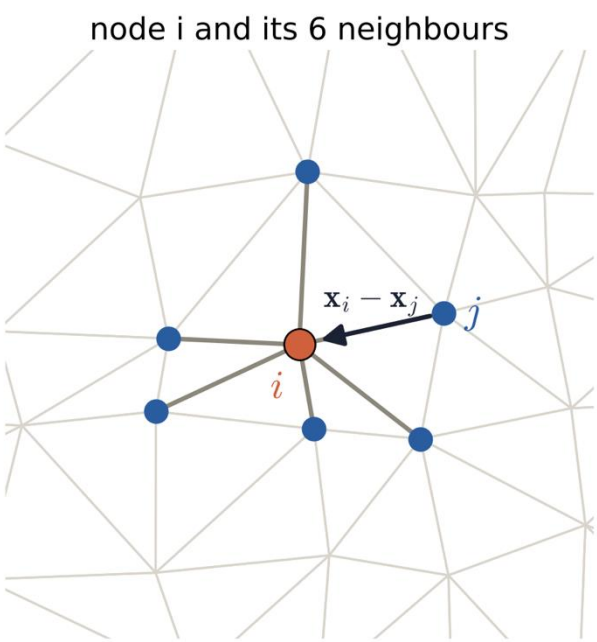
Today: models that take the mesh as input

Pfaff, Fortunato, Sanchez-Gonzalez & Battaglia, Learning mesh-based simulation with graph networks, ICLR 2021

A mesh is a graph

- A **graph** $G = (V, E)$: nodes V and directed edges E. $N(i)$: the neighbors of node i
- **Mesh:** one node per mesh vertex, *two edges* (one each way) for every side of a triangle
- **Particles:** an edge between particles i and j when $\|\mathbf{x}_i - \mathbf{x}_j\| < r$
- **Node features:** field values, e.g. temperature u_i, and a node type (interior, wall, inflow)
- **Edge features:** the relative position $\mathbf{x}_i - \mathbf{x}_j$ and its length
- **Absolute positions are left out**, so the model is the same everywhere in space (**translation invariance**)



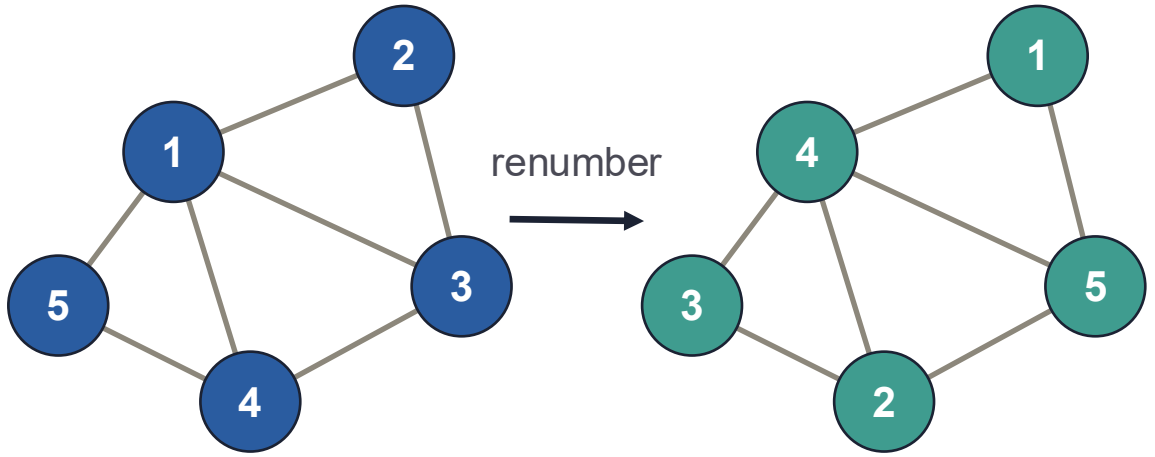


Permutation equivariance on graphs

- Renumbering the nodes *permutes the rows* of the node-feature matrix **H** (*N* x *d*) and relabels the edges
- The model should return the same field, renumbered. This is **permutation equivariance**:

$$f(P\,\mathbf{H}) = P\,f(\mathbf{H})$$

- *P*: any permutation matrix
- An MLP on the vector of all nodes **fails this**, and needs a fixed *N*
- A sum $\Sigma_j \mathbf{h}_j$, a mean or a max **does not depend on the order**
- Message passing applies **the same function** to every node and every edge and combines neighbors with **a sum**



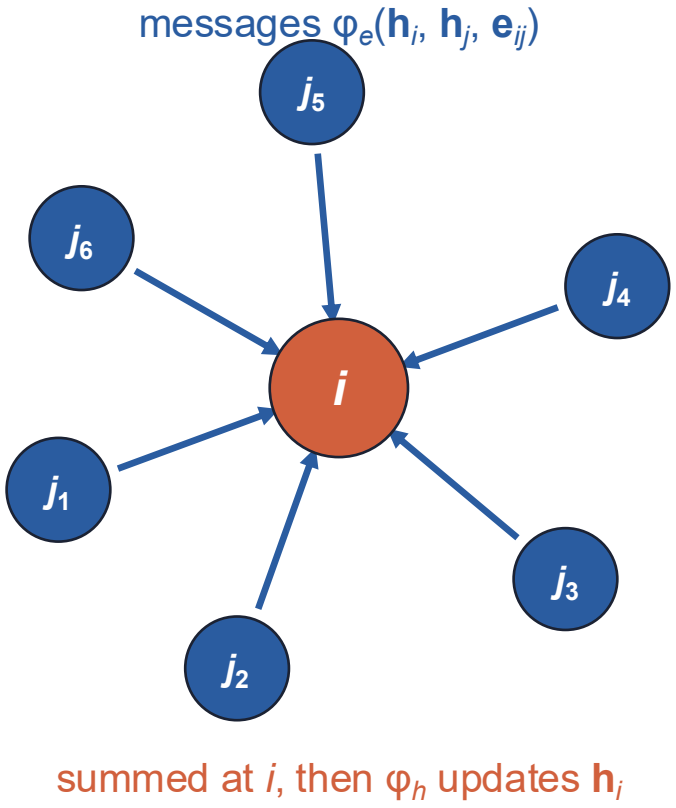
the same mesh, two numberings

Shared node and edge functions + a sum over neighbors result in permutation equivariance

Message passing

$$\mathbf{m}_i = \sum_{j \in \mathcal{N}(i)} \phi_e\left(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij}\right), \qquad \mathbf{h}_i \leftarrow \phi_h\left(\mathbf{h}_i, \mathbf{m}_i\right)$$

- $\mathbf{h}_i$: the hidden vector of node *i*. $\mathbf{e}_{ij}$: the features of the edge $j \to i$
- φ_e, φ_h: small MLPs. **The same weights on every edge and every node**, like a convolution kernel shared across pixels
- $\mathbf{m}_i$: the sum of the messages arriving at node *i*
- One step moves information along **one edge**. After *K* steps, node *i* depends on nodes up to *K* edges away
- Applies to **any number of nodes** and **any number of neighbors**

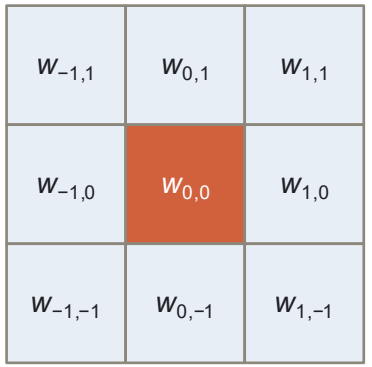


Gilmer, Schoenholz, Riley, Vinyals & Dahl, Neural message passing for quantum chemistry, ICML 2017
Battaglia et al., Relational inductive biases, deep learning, and graph networks, arXiv 2018

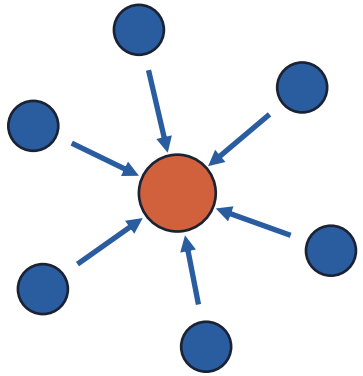
Stencils and convolutions are message passing

$$(\nabla^2 u)_i \approx \sum_{j \in \mathcal{N}(i)} w_{ij} \left(u_j - u_i\right)$$

- On a uniform 1D grid, $w_{ij} = 1/h^2$ results in the stencil $[1, -2, 1]/h^2$. On a mesh, finite elements compute w_{ij} from *the shape of the triangles* around each edge
- A 3 x 3 convolution has **one weight per offset $\mathbf{x}_j - \mathbf{x}_i$**. A grid has *only 8 offsets*
- On a mesh every edge has **its own offset**, so φ_e takes the offset as an input. The message becomes a **learned function of the geometry**
- Graph convolutional network (GCN): $\mathbf{h}_i \leftarrow \sigma(W \Sigma_j \mathbf{h}_j / \sqrt{(d_i\, d_j)})$, with d_i the number of neighbors. **One weight matrix for all neighbors** and **no edge features**: it cannot tell a near neighbor from a far one



grid: one weight per offset

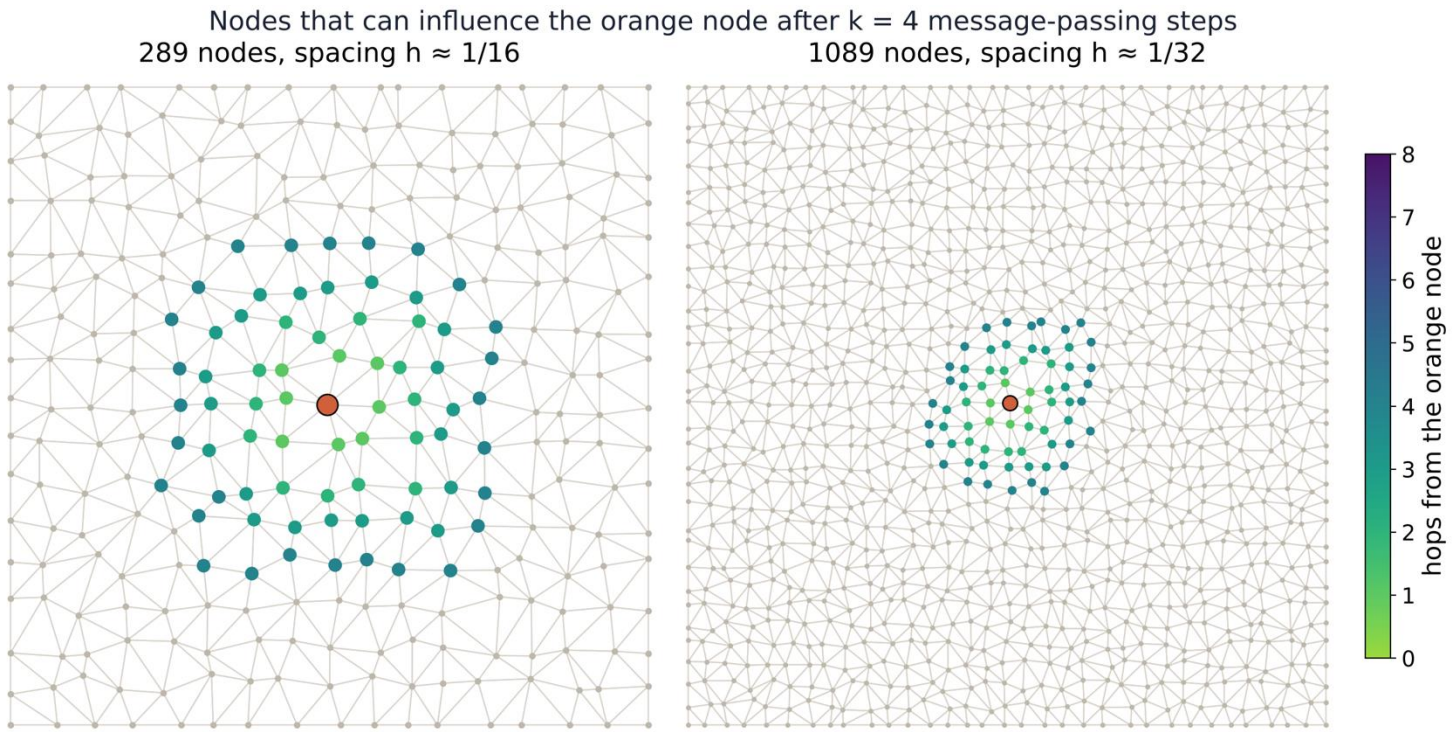


mesh: weight from $\varphi_e(\ldots, \mathbf{x}_i - \mathbf{x}_j)$

The message function plays the role of the stencil and is learned from data

Kipf & Welling, Semi-supervised classification with graph convolutional networks, ICLR 2017

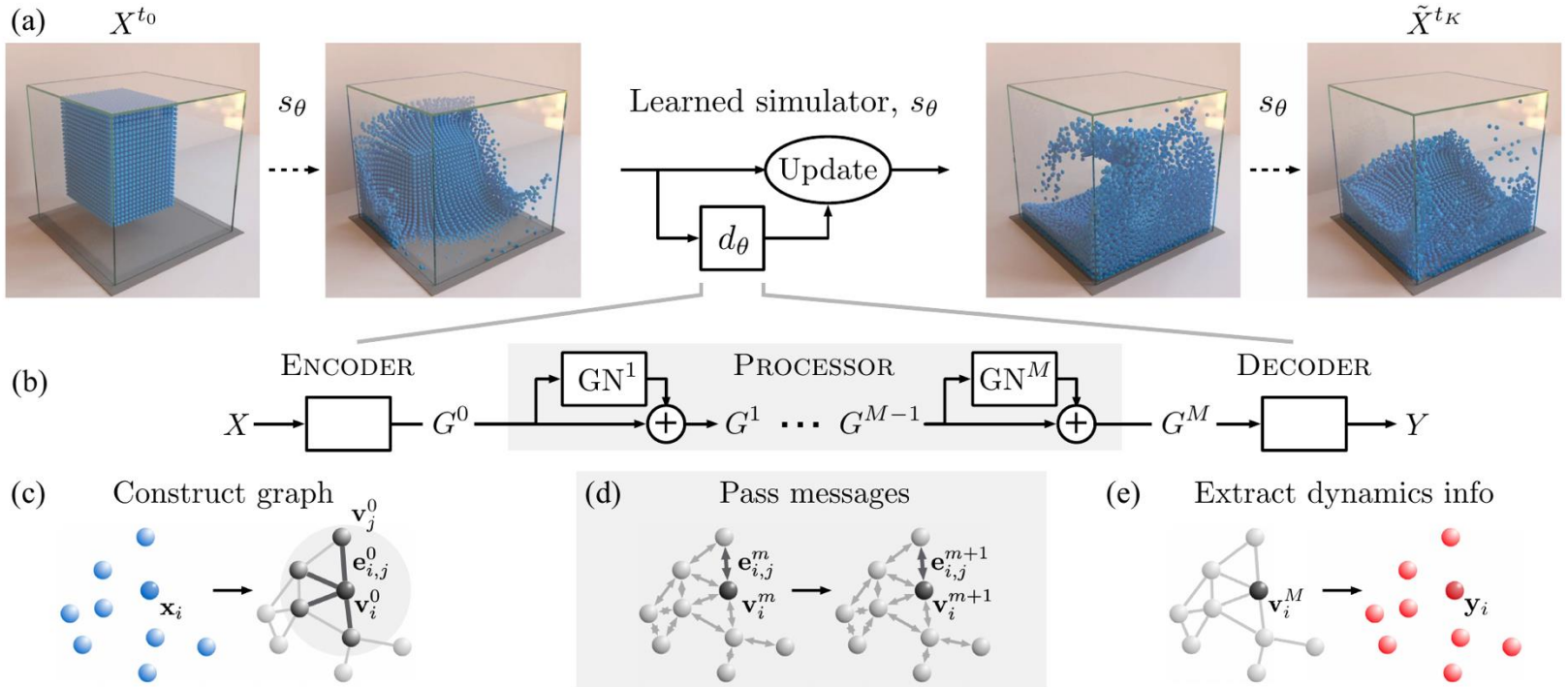
Information travels one edge per step



- In one time step Δt, the solution at a point depends on **a neighborhood set by the physics**: about $\sqrt{(\alpha \Delta t)}$ for diffusion, $c\Delta t$ for a wave with speed c
- K message-passing steps on edges of length h reach about Kh. The model needs ***Kh* ≳ the physical reach**
- On a mesh twice as fine, the same K steps cover **half the distance**

Finer meshes need more message-passing steps

Encode, process, decode

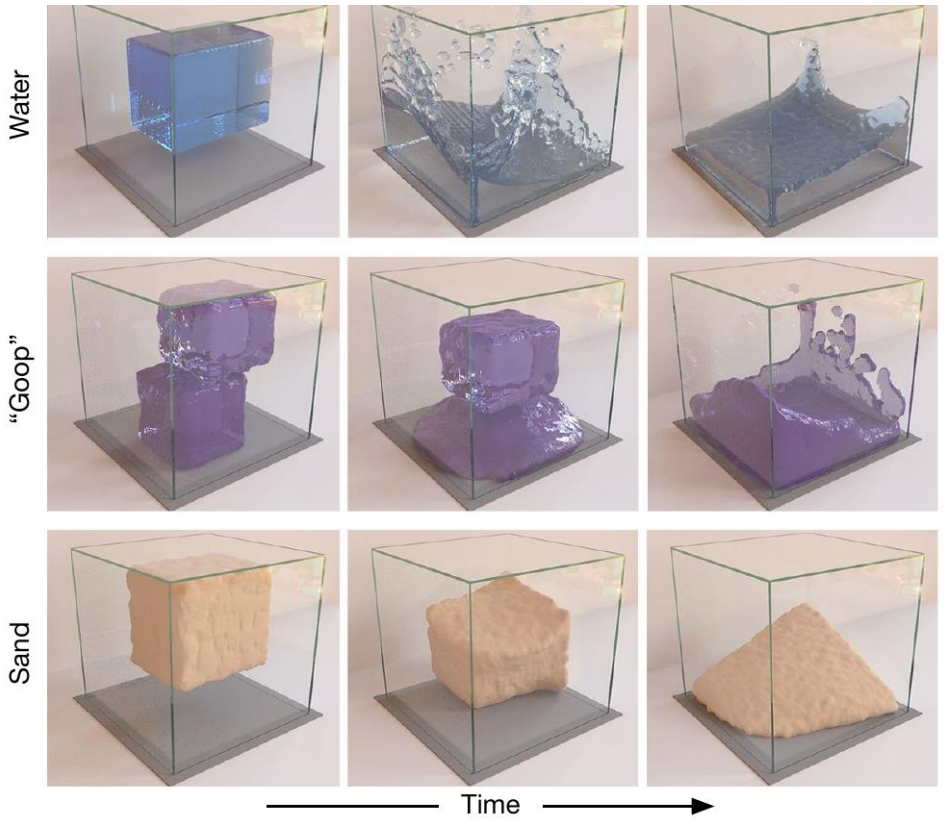


- **Encoder**: MLPs map each node's and each edge's inputs to latent vectors
- **Processor**: *M* message-passing steps with *residual updates.* Each step has *its own weights*
- **Decoder**: an MLP maps each node's latent vector to the output, e.g. an acceleration **a**. An Euler step gives the next state: $\mathbf{v} \leftarrow \mathbf{v} + \Delta t\, \mathbf{a}$, then $\mathbf{x} \leftarrow \mathbf{x} + \Delta t\, \mathbf{v}$

Sanchez-Gonzalez, Godwin, Pfaff, Ying, Leskovec & Battaglia, Learning to simulate complex physics with graph networks, ICML 2020

Learning to simulate particles (GNS)

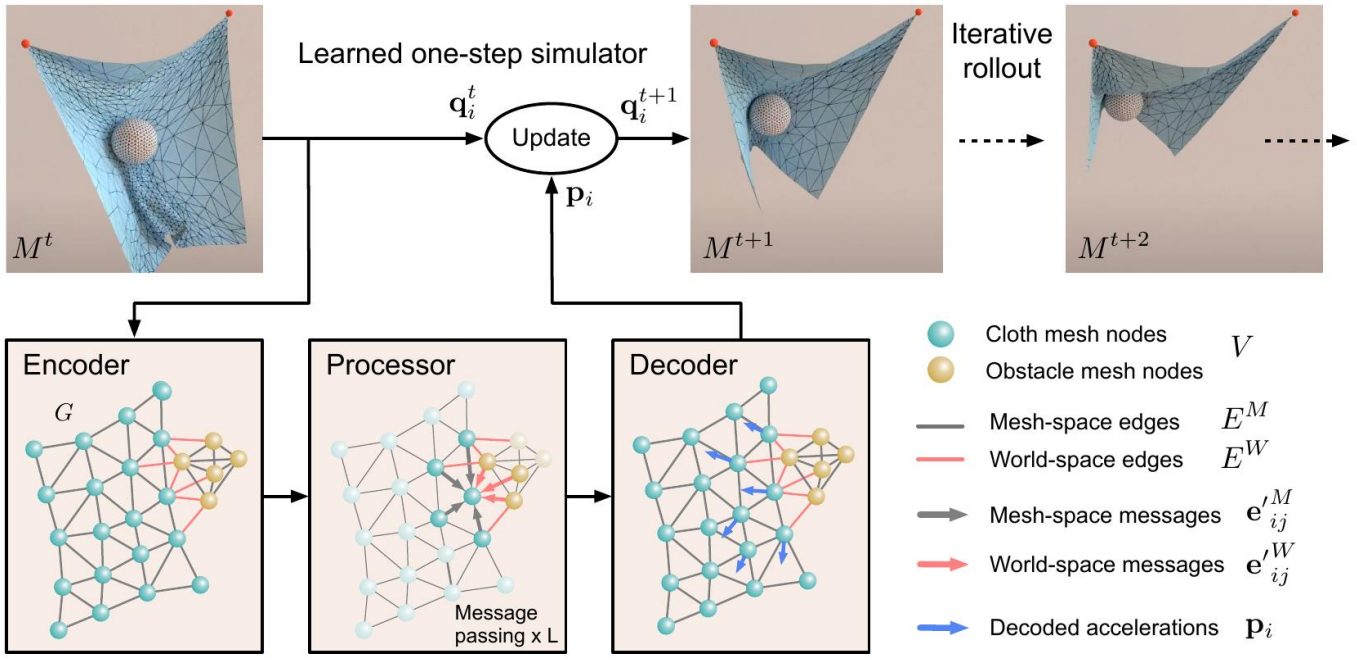
- Water, sand and goop as **thousands of particles**. Edges join particles within a radius *r.* The graph is *rebuilt every step*
- **Input:** positions, the last 5 velocities and the material type. **Output:** each particle's **acceleration**
- Trained one step ahead. Random-walk **noise** is added to the input positions, so the model learns to *correct its own errors* in rollout
- Rollouts of **thousands of steps** stay plausible
- The same model runs on **more particles and larger domains** than in training



Sanchez-Gonzalez et al., Learning to simulate complex physics with graph networks, ICML 2020

MeshGraphNets

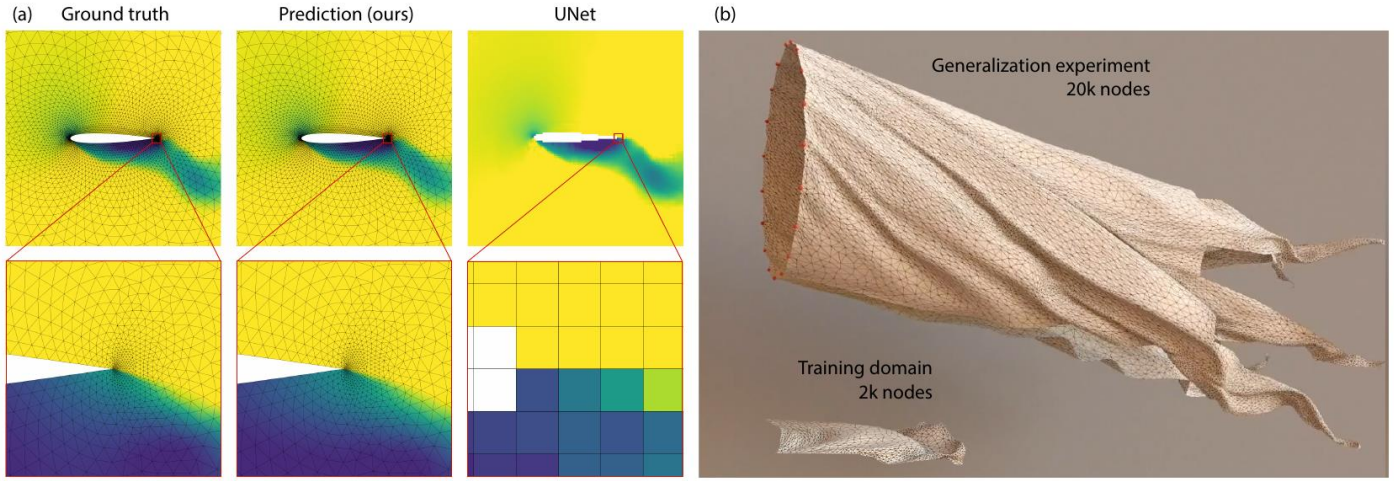
$$\mathbf{e}_{ij} \leftarrow \mathbf{e}_{ij} + \phi_e\left(\mathbf{e}_{ij}, \mathbf{h}_i, \mathbf{h}_j\right), \qquad \mathbf{h}_i \leftarrow \mathbf{h}_i + \phi_h\left(\mathbf{h}_i, \sum_j \mathbf{e}_{ij}\right)$$

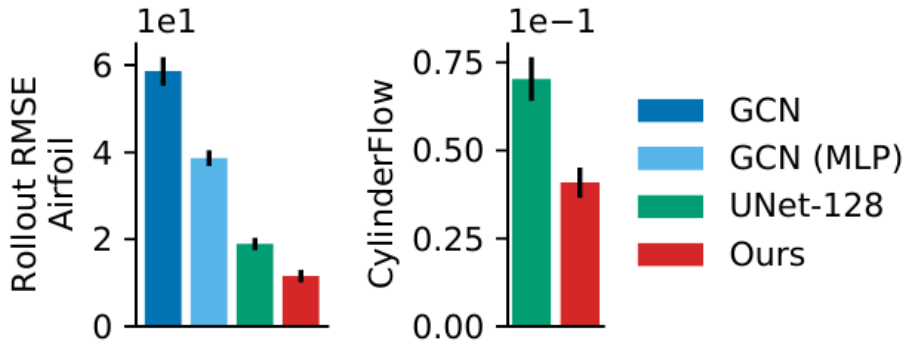


- Encode-process-decode on the simulation mesh, with **residual** edge and node updates
- Mesh edges carry the **relative position** in mesh space and its length
- Cloth: **world edges** join nodes that are *close in 3D* but *far apart on the mesh*, so collisions are seen
- **Node type:** fluid, wall, inflow, outflow, obstacle
- **15 message-passing steps**, latent size 128. **Output:** change in velocity or acceleration at each node

Pfaff, Fortunato, Sanchez-Gonzalez & Battaglia, Learning mesh-based simulation with graph networks, ICLR 2021

MeshGraphNets compared with grid models

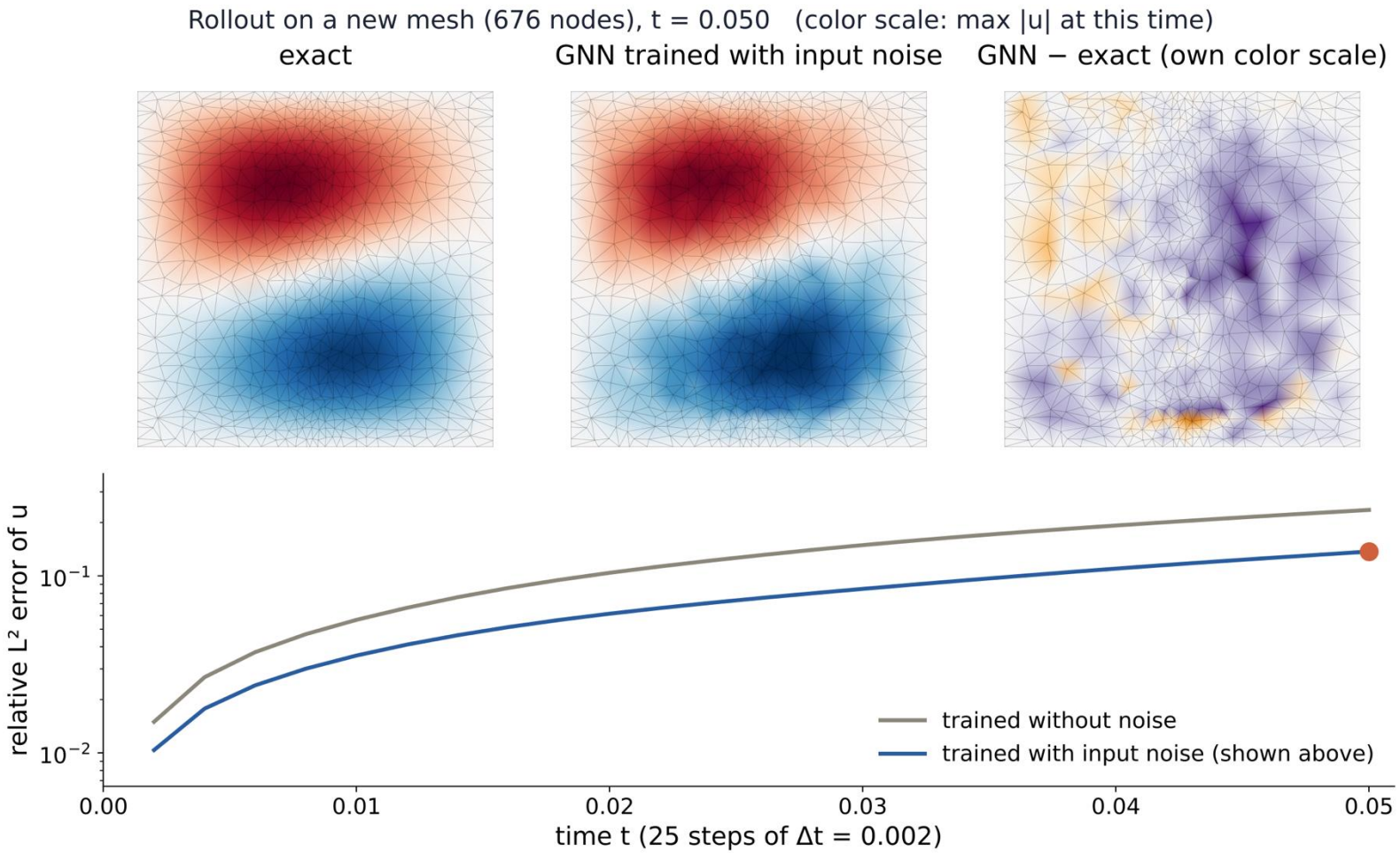




- Airfoil (left): the U-Net works on a uniform grid that **cannot resolve the thin layer** near the wing. MeshGraphNets **resolves it on the solver's fine mesh**
- **Lower rollout error** than a GCN and a grid U-Net on the fluid datasets (fig, right). The GCN has **no edge features**
- A cloth model trained on meshes with about **2000** nodes runs on **20,000** nodes (middle)
- **Tens to hundreds of times faster** than the solvers that produced the training data

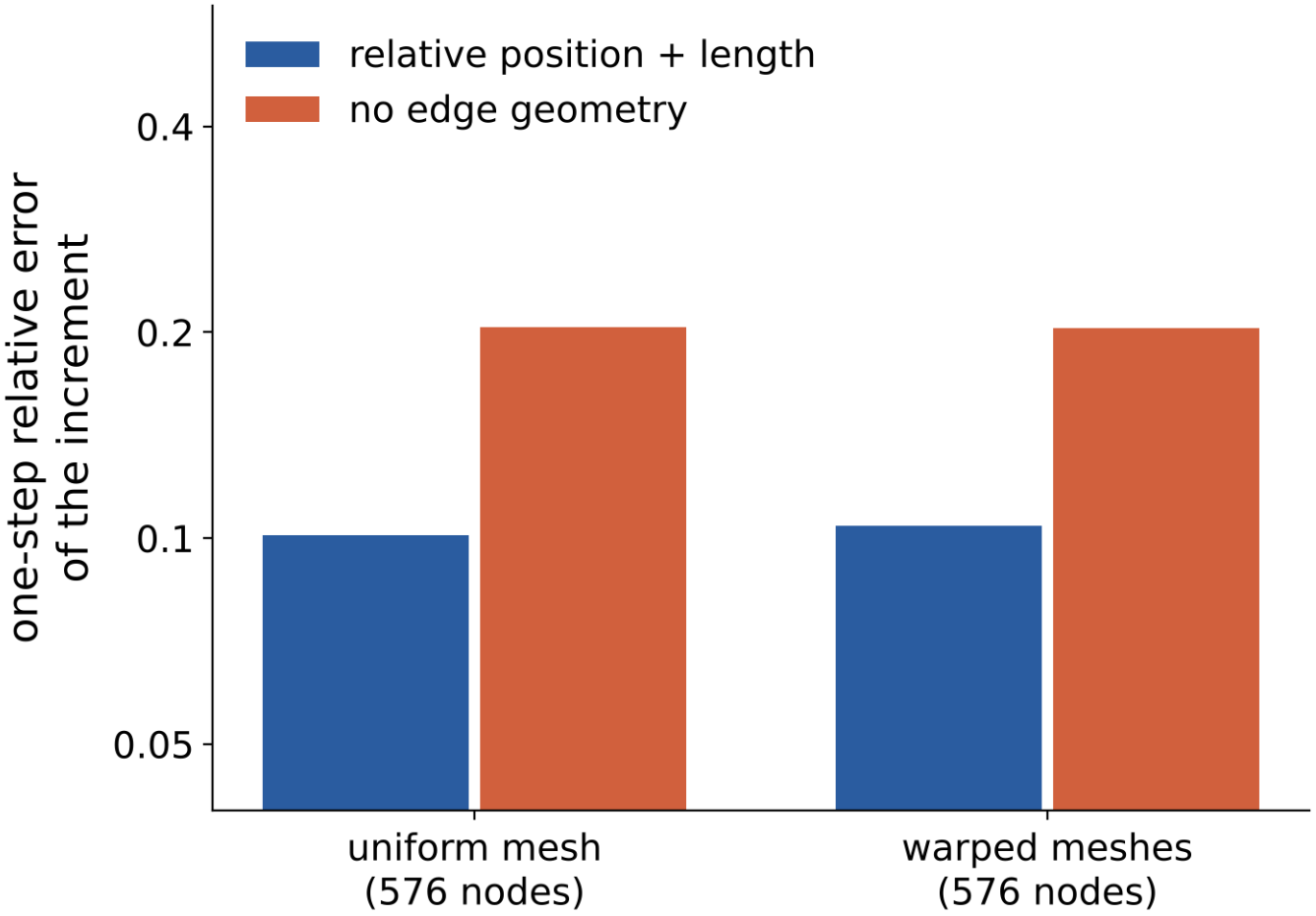
Pfaff et al., Learning mesh-based simulation with graph networks, ICLR 2021

Rollout on a new mesh



- Heat equation $u_t = \alpha \nabla^2 u$ on the unit square, $u = 0$ on the boundary. The exact solution is a sine series, so the reference **does not depend on the mesh**
- Trained one step ahead on 200 meshes (400 to 1024 nodes, each warped differently). **The test mesh is new**
- Trained without noise (gray): the error **keeps growing** with rollout
- Fine-tuned with noise added to the input (GNS, slide 9): the error grows **more slowly** (blue). Over 15 rollouts on 5 new meshes the gap is larger

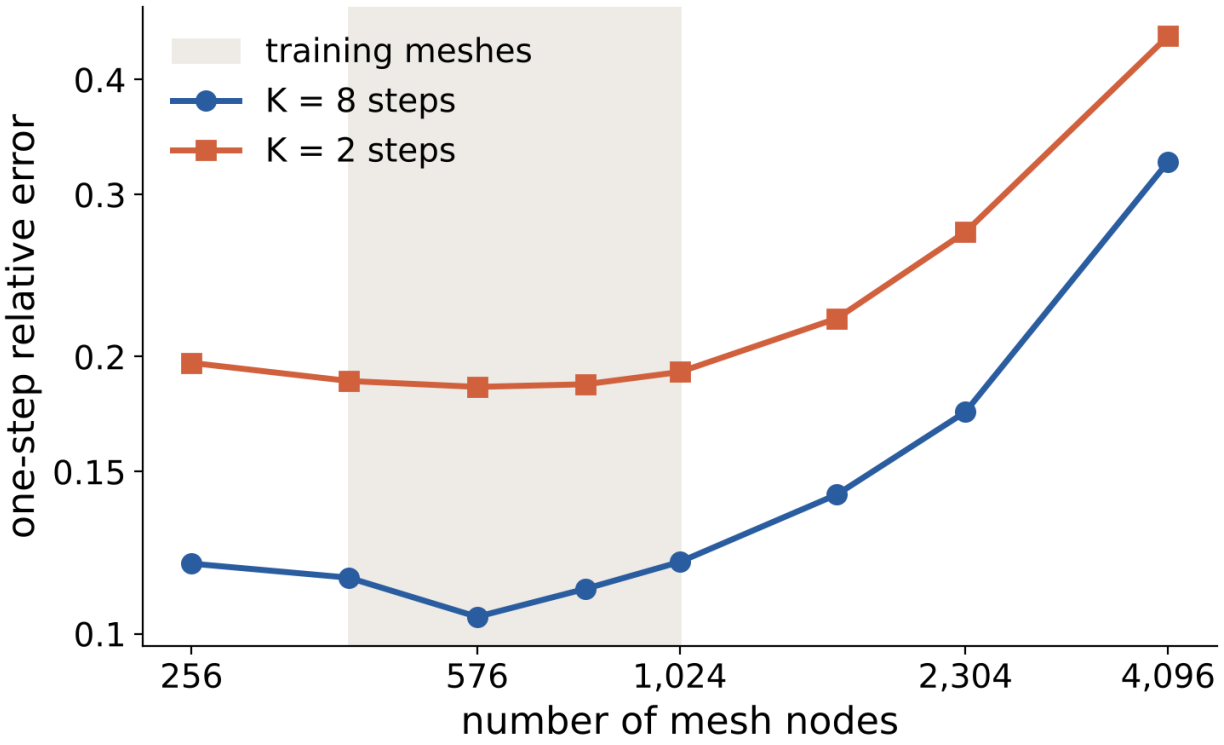
Relative positions on edges reduce the error

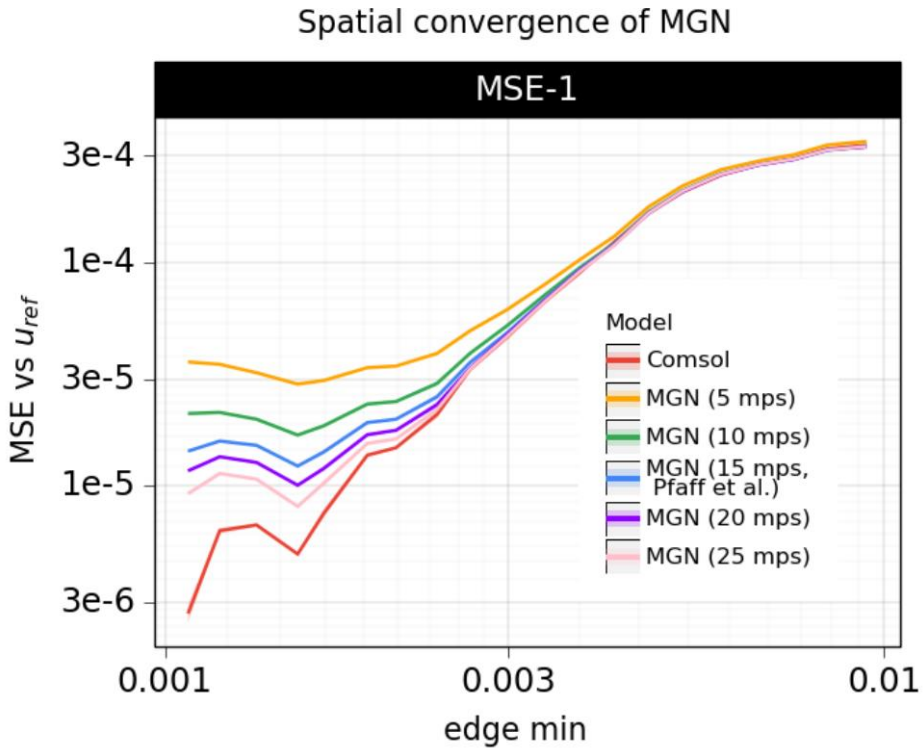


- **Same network, data and training**. Both take $u_j - u_i$ as an edge input; the second replaces the relative position and length with **a constant**
- Without geometry the one-step error is **clearly higher**, on the uniform mesh and on warped meshes
- With geometry, a message can weight each neighbor by its **distance and direction**, like a finite-element stencil
- MeshGraphNets: the GCN baseline, which has no edge features, has the **highest error**

Include the geometry in the messages: relative positions and lengths

Finer meshes need more message-passing steps

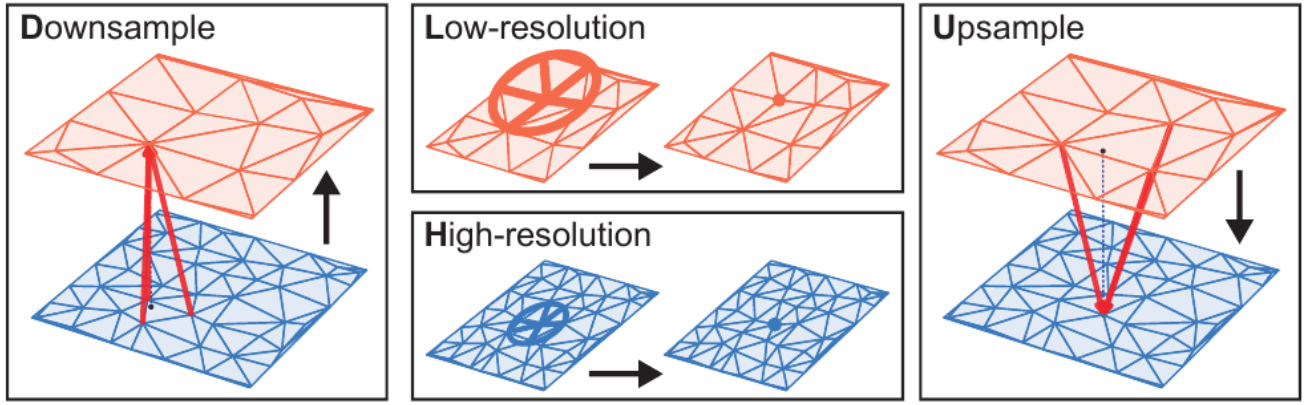


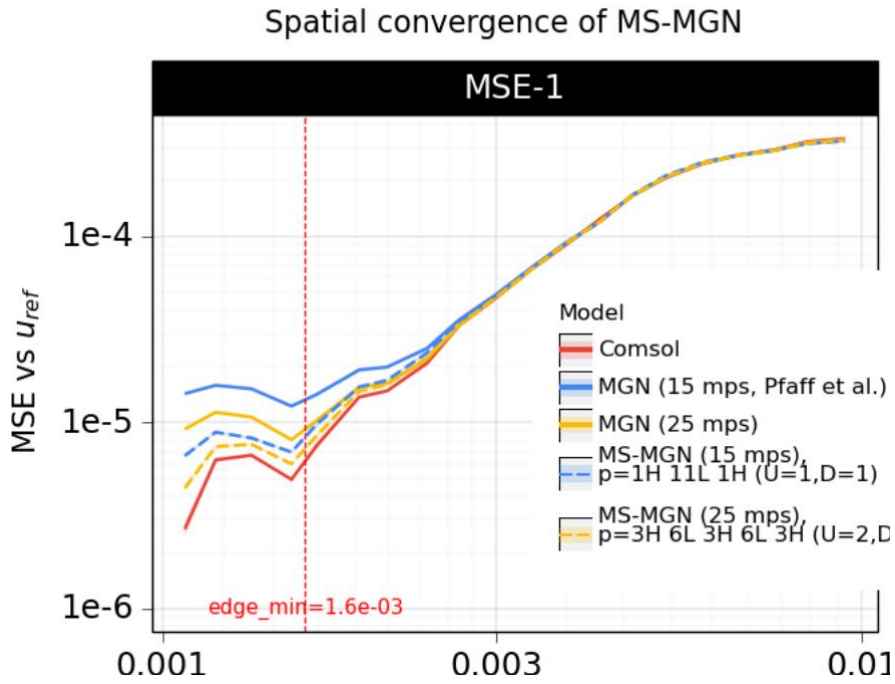


- Left (notebook): with $K = 8$ the error is **lowest on meshes like the training meshes** and **grows on finer meshes**. With $K = 2$ it is **higher everywhere**
- Right (Fortunato et al.): MeshGraphNets error **stops falling** as the mesh is refined, while the solver's **keeps falling**. More steps (mps) help, *but slowly*

Fortunato, Pfaff et al., MultiScale MeshGraphNets, ICML 2022 AI4Science Workshop

Multiscale graphs

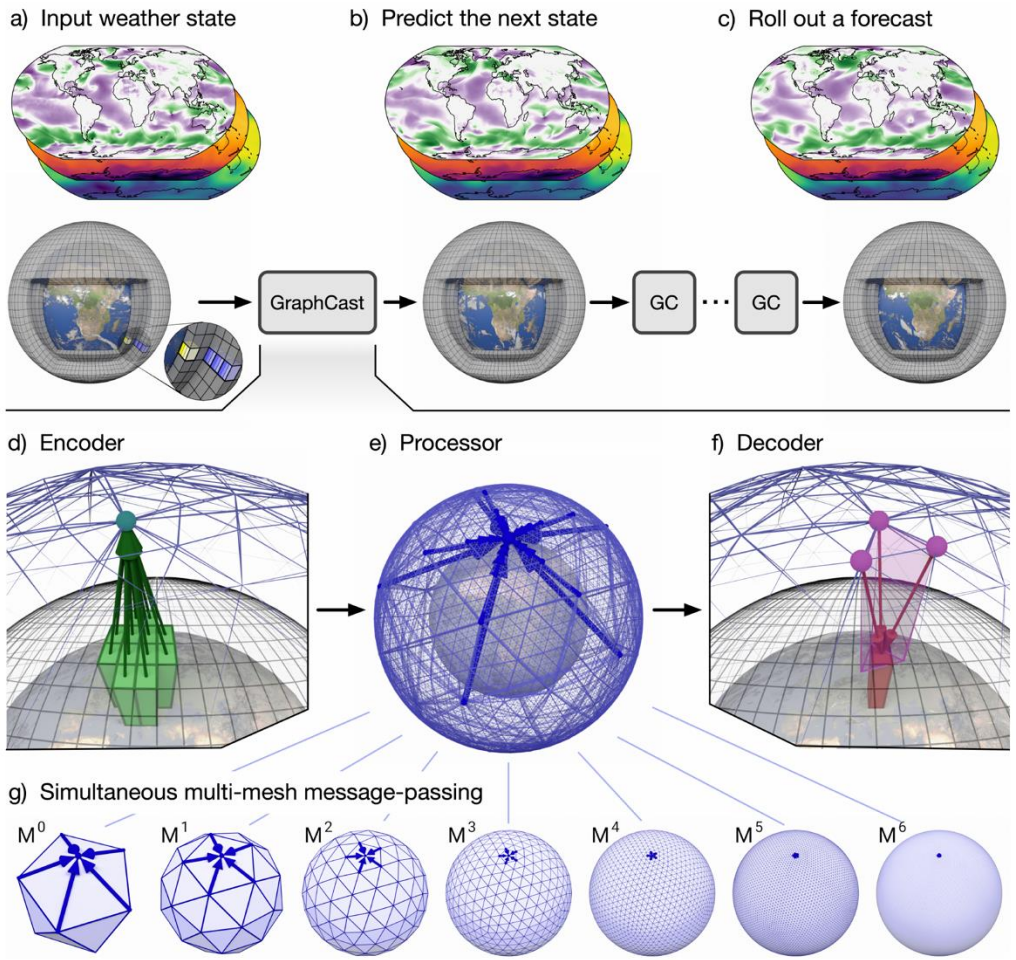




- Two meshes: the **fine mesh** for detail, a **coarse mesh** for long-range messages
- Downsample to the coarse mesh, pass messages there, upsample back. The same idea as the levels of a **U-Net** and **multigrid** solvers
- With the same number of steps, MS-MGN **tracks the solver's error** to finer meshes (right)
- **Over-squashing**: the *K*-hop neighborhood grows quickly with *K*, and all of it must fit in **one fixed-size vector**

Fortunato, Pfaff et al., MultiScale MeshGraphNets, ICML 2022 AI4Science Workshop. Alon & Yahav, On the bottleneck of graph neural networks, ICLR 2021

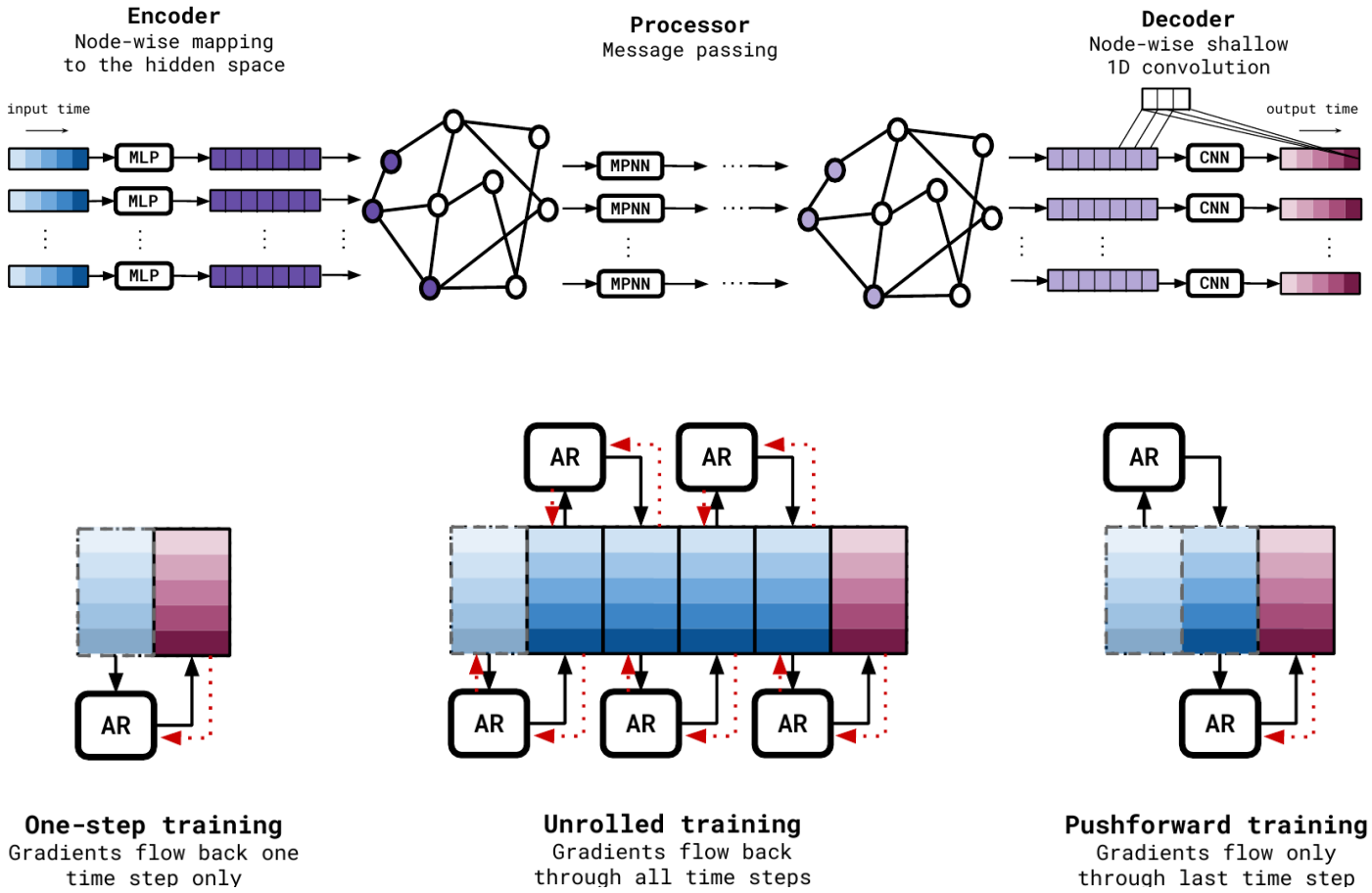
GraphCast: message passing on a multimesh



- **Input:** the weather at two times on a 0.25° latitude–longitude grid, about 1 million points x 227 variables
- **Encoder:** grid → icosahedral mesh. **Processor:** 16 message-passing steps on the **multimesh**, the edges of all refinement levels M^0 to M^6 together
- **Long edges** from coarse levels carry information around the globe in a few steps; **short edges** resolve local detail
- **Decoder:** mesh → grid. One step is *6 hours*; rolled out to *10 days*
- More accurate than ECMWF's HRES forecast on **about 90%** of 1380 verification targets (WeatherBench 2)

Lam et al., Learning skillful medium-range global weather forecasting, Science 2023

Message passing neural PDE solvers



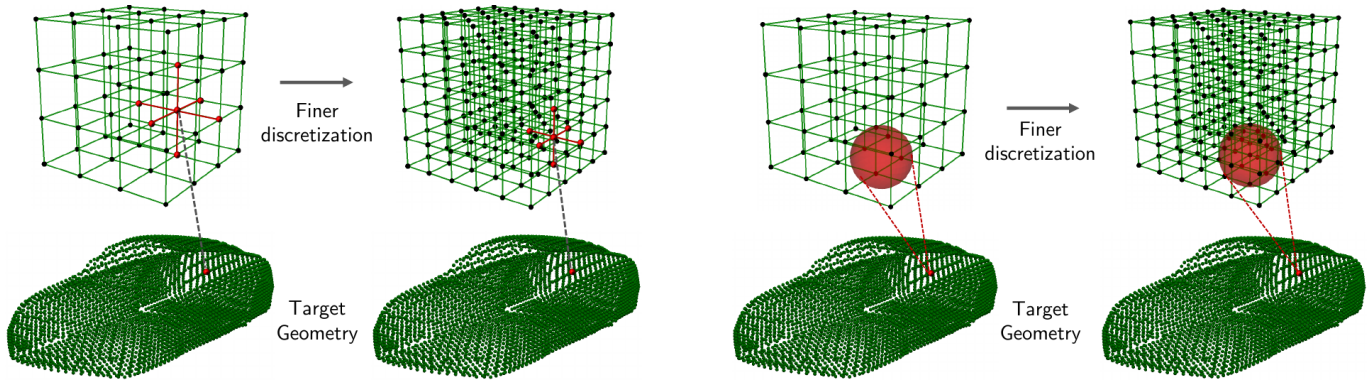
- **Node inputs:** the last K values of u, position, time, and the PDE's parameters θ_{PDE} (e.g. viscosity), so **one model covers a family of equations**
- Messages use $u_i - u_j$ and $\mathbf{x}_i - \mathbf{x}_j$, *like a finite-difference stencil*
- **Temporal bundling**: predict K future steps at once, so a rollout needs ***K* times fewer model calls**
- **Pushforward trick** (bottom right): unroll two steps, **backpropagate through the last only**, so training sees the model's own errors (Lecture 7)

Brandstetter, Worrall & Welling, Message passing neural PDE solvers, ICLR 2022

Message passing as a kernel integral

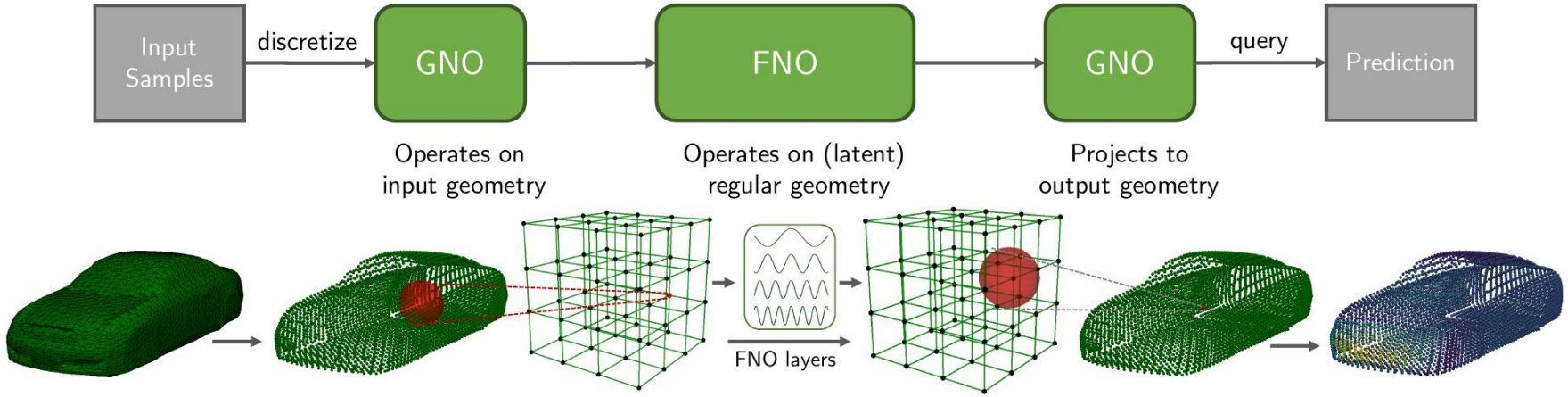
$$(\mathcal{K}v)(x_i) \approx \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \kappa_\theta\big(x_i, x_j, a(x_i), a(x_j)\big)\, v(x_j)$$

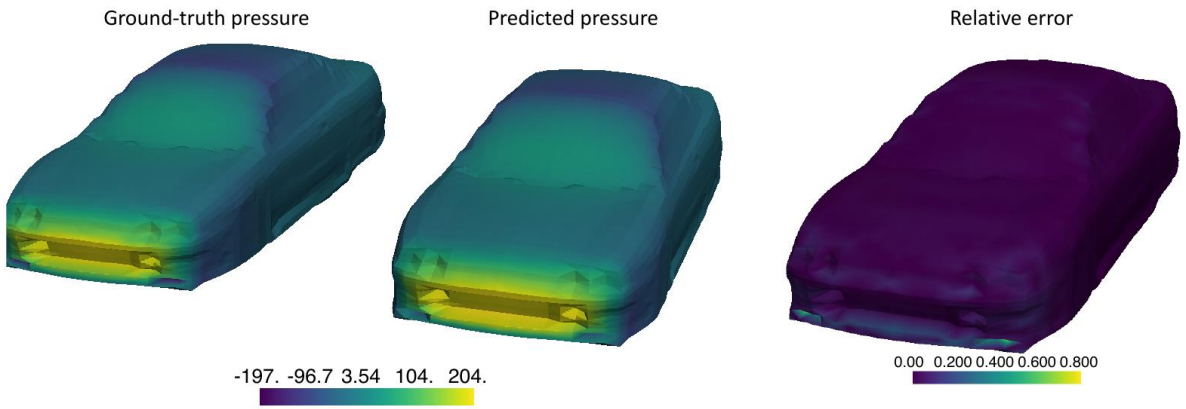
- Lecture 12: a neural operator layer computes $\int \kappa_\theta(x, y)\, v(y)\, dy$. On a graph whose edges join points within a radius r, the integral becomes an **average of messages**
- As the points get denser, the average **converges to the integral**: the layer is **the same operator on every mesh**
- A GNN with a **fixed number of nearest neighbors** sees a *smaller region* on a finer mesh (bottom left), and converges to a **pointwise** map



Li et al., Neural operator: graph kernel network for partial differential equations, 2020
Li et al., Geometry-informed neural operator for large-scale 3D PDEs, NeurIPS 2023

GINO for 3D shapes





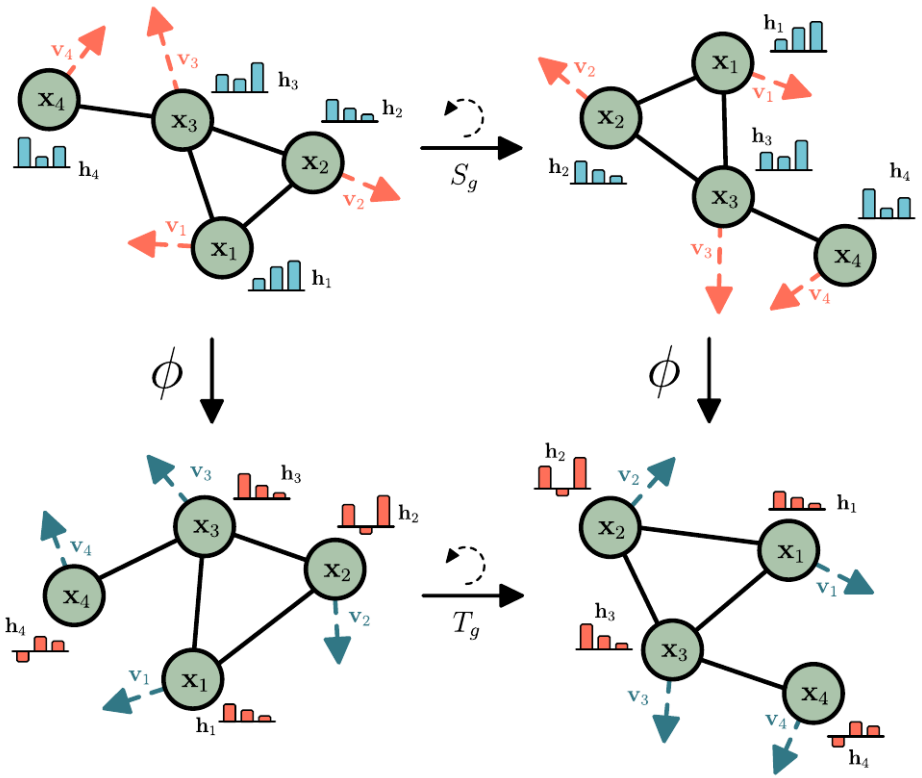
- **Input:** a car surface as a **point cloud**, different for every car
- GNO: points → a uniform latent grid. **FNO** on the grid. GNO: grid → query points on the surface
- **Output: surface pressure** and from it the **drag**
- Trained on Reynolds-averaged Navier-Stokes (RANS) simulations; **orders of magnitude faster** than the CFD solver

Li et al., Geometry-informed neural operator for large-scale 3D PDEs, NeurIPS 2023

E(n)-equivariant GNNs

$$\mathbf{m}_{ij} = \phi_e\left(\mathbf{h}_i, \mathbf{h}_j, \|\mathbf{x}_i - \mathbf{x}_j\|^2\right), \quad \mathbf{x}_i \leftarrow \mathbf{x}_i + \frac{1}{|\mathcal{N}(i)|}\sum_j (\mathbf{x}_i - \mathbf{x}_j)\,\phi_x(\mathbf{m}_{ij}), \quad \mathbf{h}_i \leftarrow \phi_h\left(\mathbf{h}_i, \sum_j \mathbf{m}_{ij}\right)$$

- Rotating a system of particles should **rotate the predicted velocities and forces**
- Relative positions give **translation invariance**. A generic MLP on $\mathbf{x}_i - \mathbf{x}_j$ is **not rotation-equivariant**
- **EGNN:** messages use **only the distance** $\|\mathbf{x}_i - \mathbf{x}_j\|^2$, which rotations do not change
- Position updates are sums of the vectors $\mathbf{x}_i - \mathbf{x}_j$ times learned scalars, so they **rotate with the input**
- **Exact equivariance** *without data augmentation*



Satorras, Hoogeboom & Welling, E(n) equivariant graph neural networks, ICML 2021

Grids, operators and graphs compared

	U-Net (Lecture 6)	FNO (Lecture 12)	GNN (today)
Domain	uniform grid	uniform grid (Geo-FNO, GINO for others)	any mesh or point cloud
New resolution	kernels are stencils: retrain	same weights	same weights; finer meshes need more steps
Long-range information	downsampling levels	global Fourier modes in every layer	one edge per step; multiscale meshes
Cost per layer	$O(N)$	$O(N \log N)$	O(number of edges), irregular memory access
Built-in symmetry	translation	translation (periodic)	permutation, translation; rotation with EGNN

Uniform grid: start with a U-Net or FNO. Changing mesh or particles: use a GNN

GNNs in your project

- Use a GNN when the data live on a **mesh or point cloud that changes between samples**: airfoils, particles, molecules, interfaces tracked as points
- **Libraries:** PyTorch Geometric, DGL. NVIDIA PhysicsNeMo includes MeshGraphNets; the MeshGraphNets datasets (CylinderFlow, Airfoil) are public
- **Baseline:** interpolate to a grid and train a U-Net. Report *the error the interpolation introduces*
- Set the number of steps $K \gtrsim \ell / h$: ℓ is the distance the physics travels in one Δt, h the edge length
- Normalize node and edge inputs; add **input noise** for stable rollouts (GNS); report **rollout error**, not only one-step error

pyg.org, dgl.ai, github.com/NVIDIA/physicsnemo, github.com/google-deepmind/deepmind-research

Summary

- Meshes and particles are **graphs**: node features (values, type), edge features (relative position, length)
- **Message passing:** shared MLPs on edges and nodes, a sum over neighbors. *Equivariant to renumbering the nodes*
- Finite-difference stencils and convolutions are **special cases**; on a mesh the message **must see the edge geometry**
- Information travels **one edge per step**. **Finer meshes need more steps**; multiscale meshes (MS-MGN, GraphCast) shorten the paths
- **Encode-process-decode** simulators: GNS for particles, MeshGraphNets for meshes, MP-PDE for PDE families
- Mean over a fixed radius approximates the **kernel integral** of a neural operator (GNO, GINO)
- EGNN adds **exact rotation equivariance** with distances and relative vectors

After class

- Pfaff et al., Learning mesh-based simulation with graph networks, ICLR 2021
- Sanchez-Gonzalez et al., Learning to simulate complex physics with graph networks, ICML 2020
- Battaglia et al., Relational inductive biases, deep learning, and graph networks, arXiv 2018: Sections 3-4
- Brandstetter, Worrall & Welling, Message passing neural PDE solvers, ICLR 2022
- Lam et al., Learning skillful medium-range global weather forecasting, Science 2023
- Bronstein, Bruna, Cohen & Veličković, Geometric deep learning, 2021: Chapter 5 (graphs)

Next, Lecture 14 (Tue, Nov 10): physics-informed ML

NEXT: HANDS-ON

From concepts to code

→ **Open the Lecture 13 notebook**

1. Build a graph from a triangular mesh; check permutation equivariance
2. How far 1, 4 and 8 message-passing steps reach on a coarse and a fine mesh
3. Pretrained GNNs on new meshes: with and without edge geometry
4. Rollout on a new mesh, and the error on finer meshes for K = 2 and K = 8