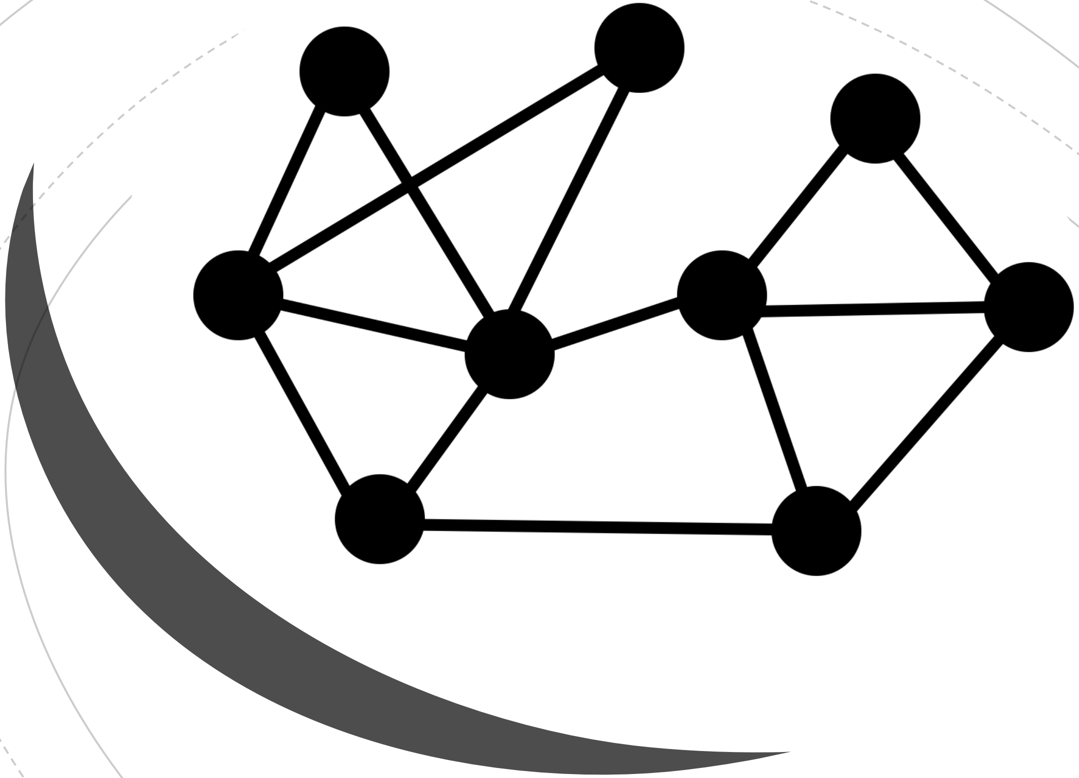


Graph Neural Network for Physics-based Simulation

Qingqing Zhao
Stanford University
05/28/2023@集智斑图

Learned simulation using GNN

- **Classical simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free

Learned simulation using GNN

- **Classical simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free

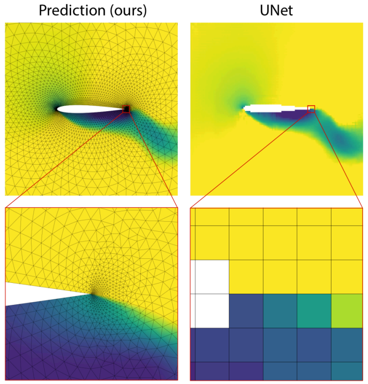
- **With GNN**

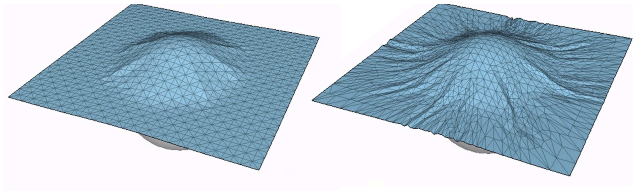
Learned simulation using GNN

- **Classical simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

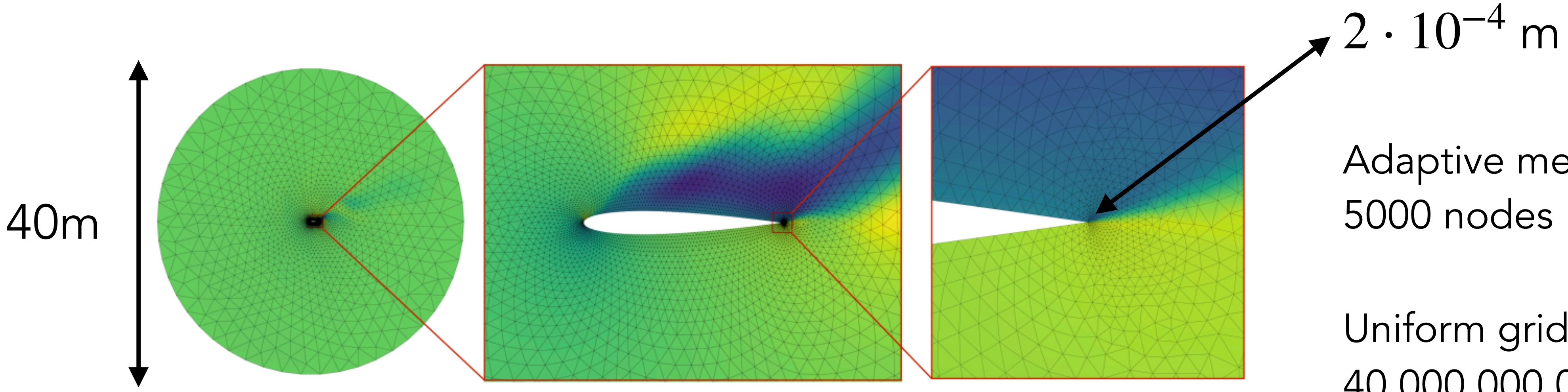
- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free

- **With GNN**
  - Spatial-Temporal Adaptivity





Graphs (Meshes) in simulation - Adaptivity



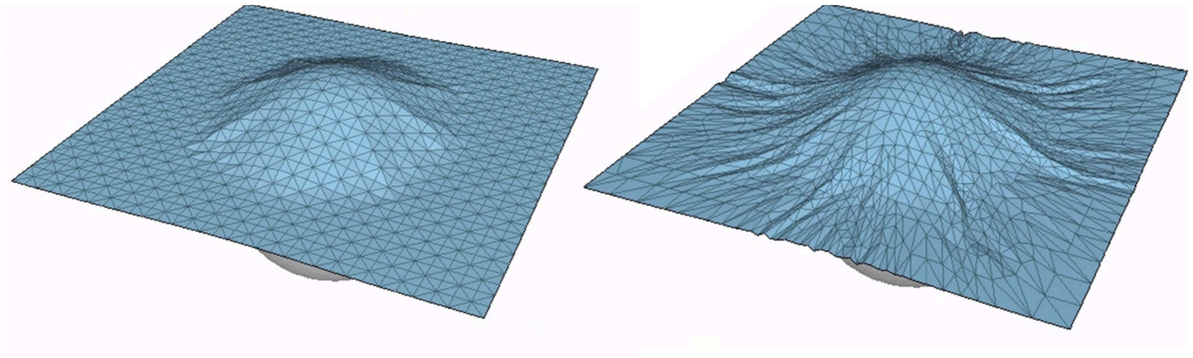
Adaptive meshes:
5000 nodes

Uniform grid at $2 \cdot 10^{-4}$ m:
40,000,000,000 nodes

Graphs (Meshes) in simulation - Adaptivity

Adaptive meshes:
5000 nodes

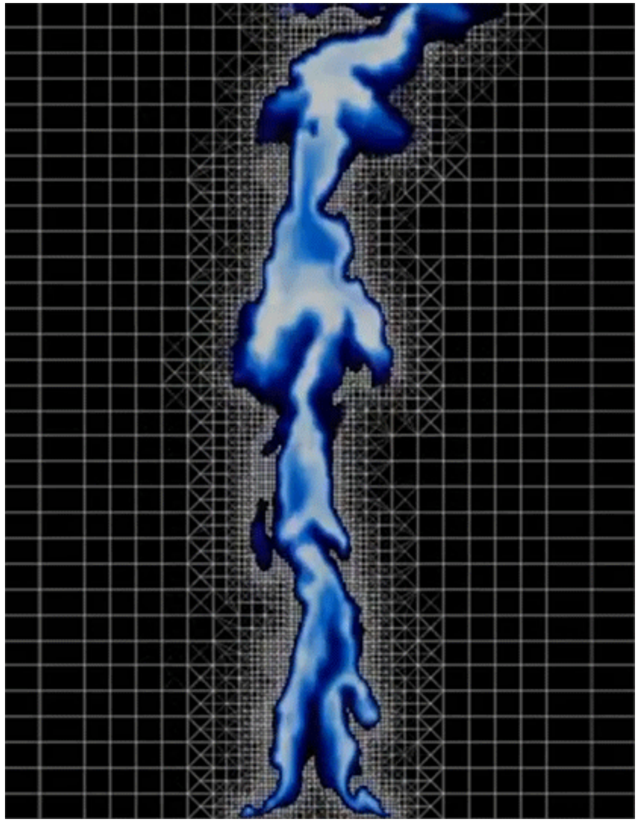
Uniform grid at $2 \cdot 10^{-4}$ m:
40,000,000,000 nodes

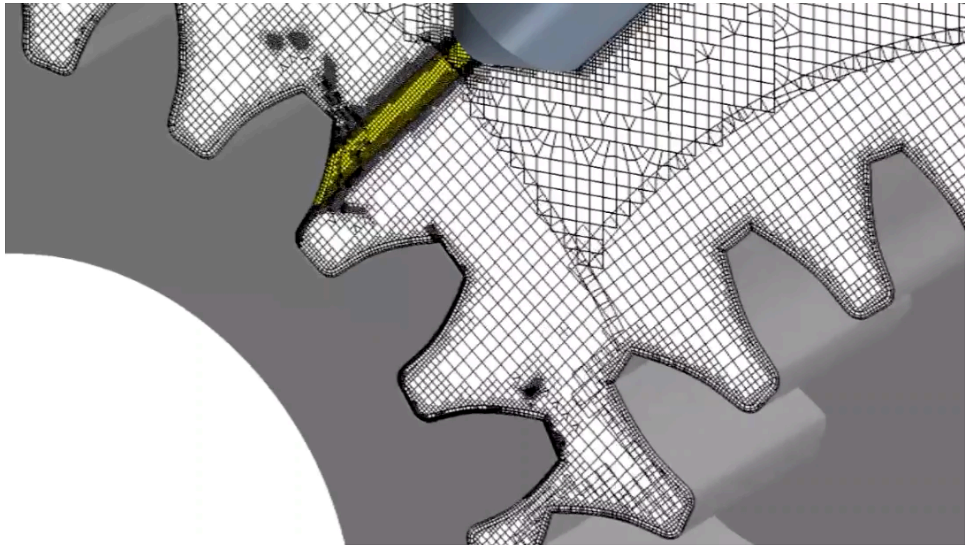


Regular mesh: 1k nodes

Adaptive mesh: 1k nodes

Graphs (Meshes) in simulation - Adaptivity



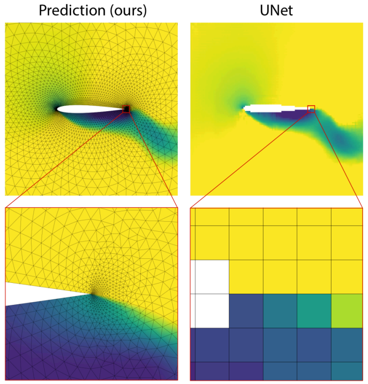


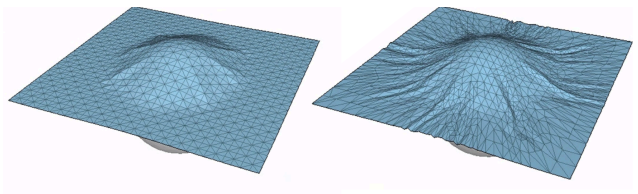
https://fyfluiddynamics.com/2019/12/adapting-to-the-flow/

Learned simulation using GNN

- **Engineered simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free



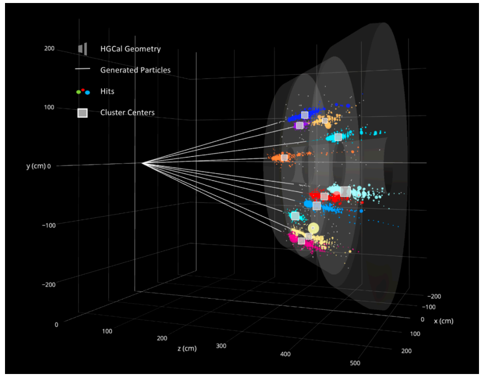


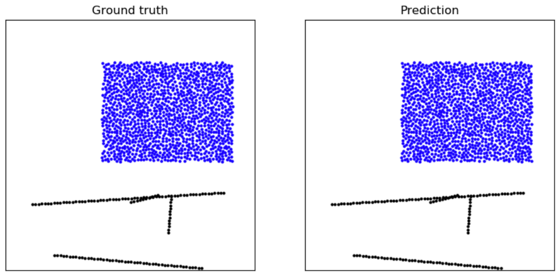
- **With GNN**
  - Spatial-Temporal Adaptivity

Learned simulation using GNN

- **Engineered simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free



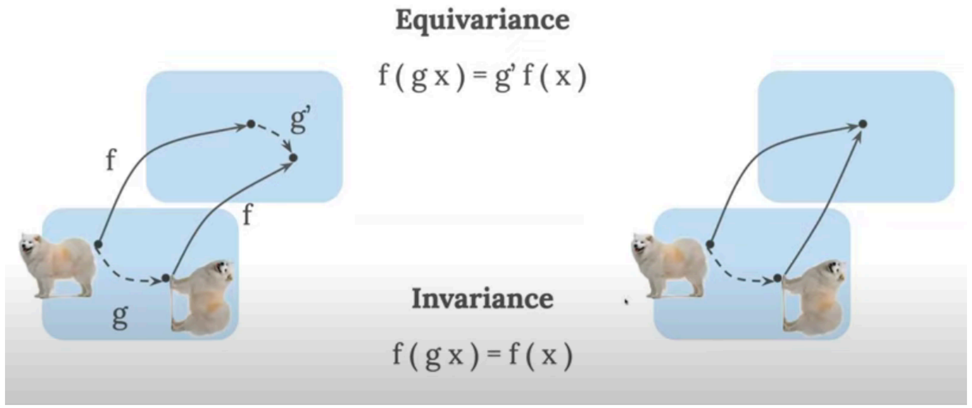


- **With GNN**
  - Spatial-Temporal Adaptivity
  - Dynamic graph

Learned simulation using GNN

- **Engineered simulation**
  - Time consuming to build
  - Resources consuming to run
  - Only as accurate as model
    - model may be over-simplified
  - Not good for solving inverse problem

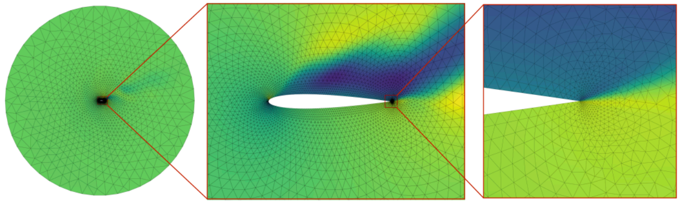
- **Learned simulation**
  - General framework
  - Fast to run
  - As accurate as data
  - Fast forward model, get gradient for free



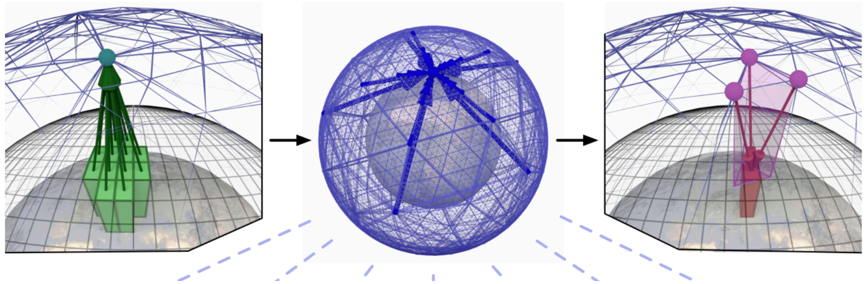
- **With GNN**
  - Spatial-Temporal Adaptivity
  - Dynamic graph
  - Inductive biases
    - e.g. invariance/equivariance

Outline

- Framework:
  - Learning mesh-based simulation with Graph Networks (MeshGraphNet, Tobias Pfaff, etl. 2021)



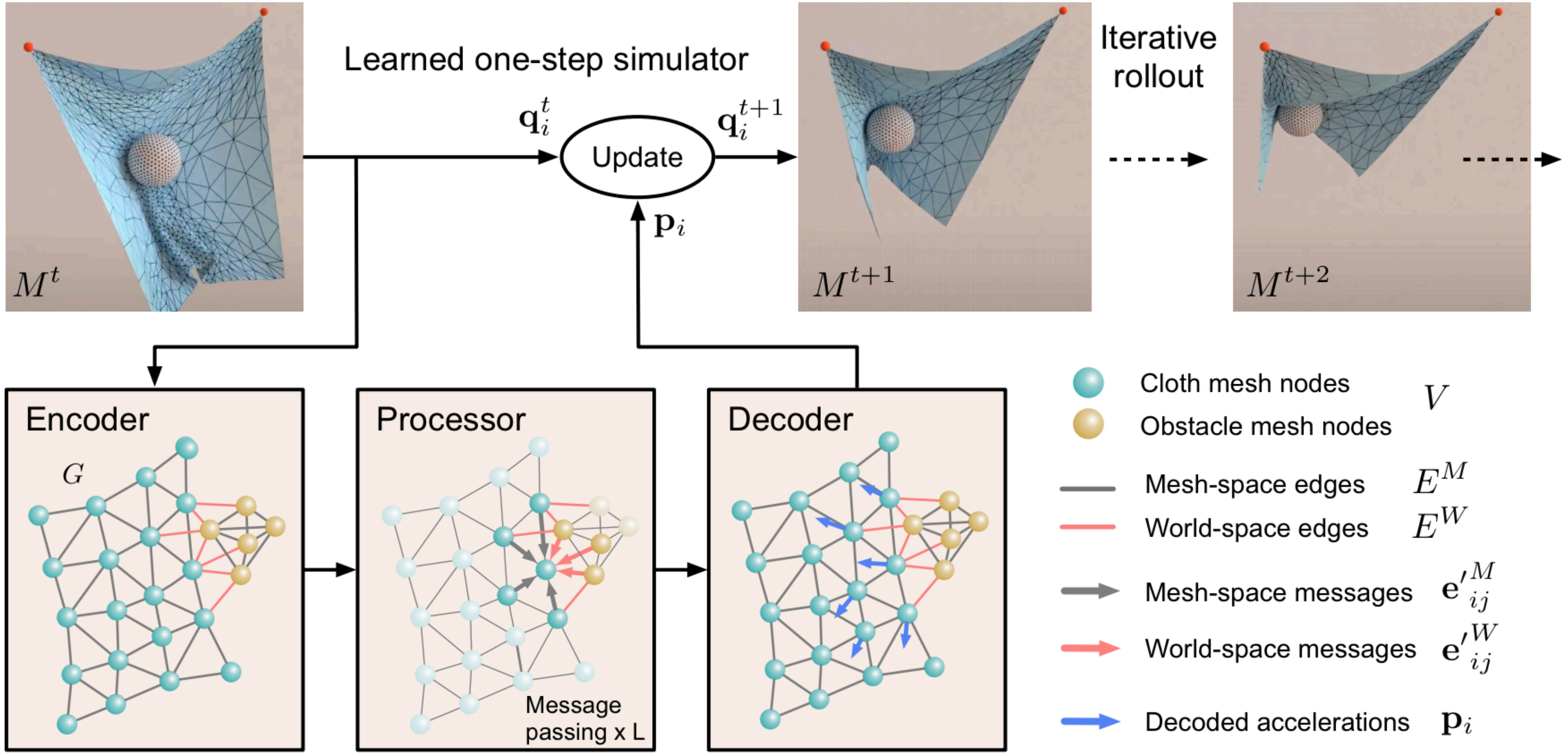
- Application:
  - GraphCast: Learning skillful medium-range global weather forecasting (Remi Lam, etl, 2022)



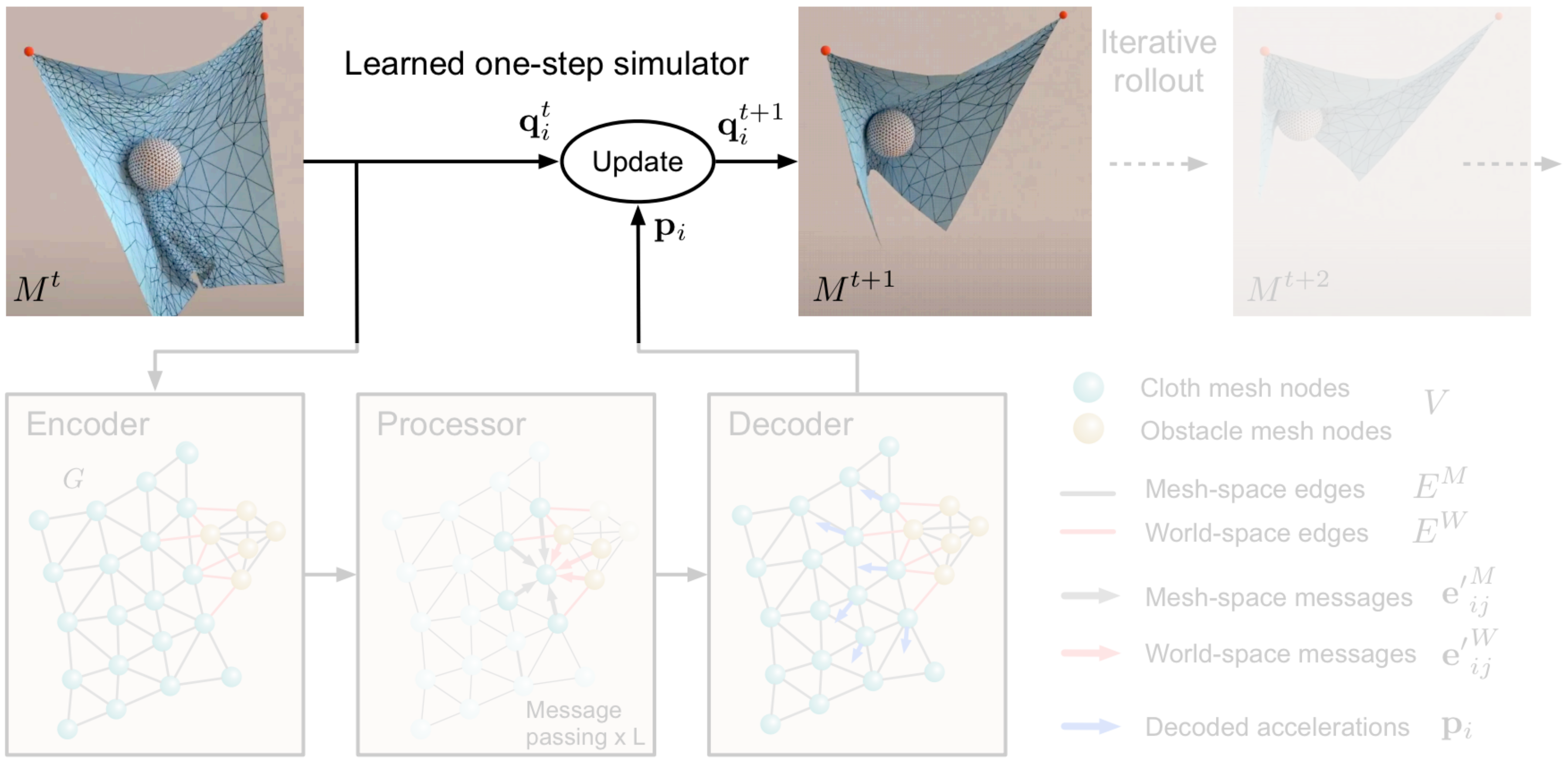
https://arxiv.org/abs/2212.12794

https://arxiv.org/pdf/2010.03409.pdf

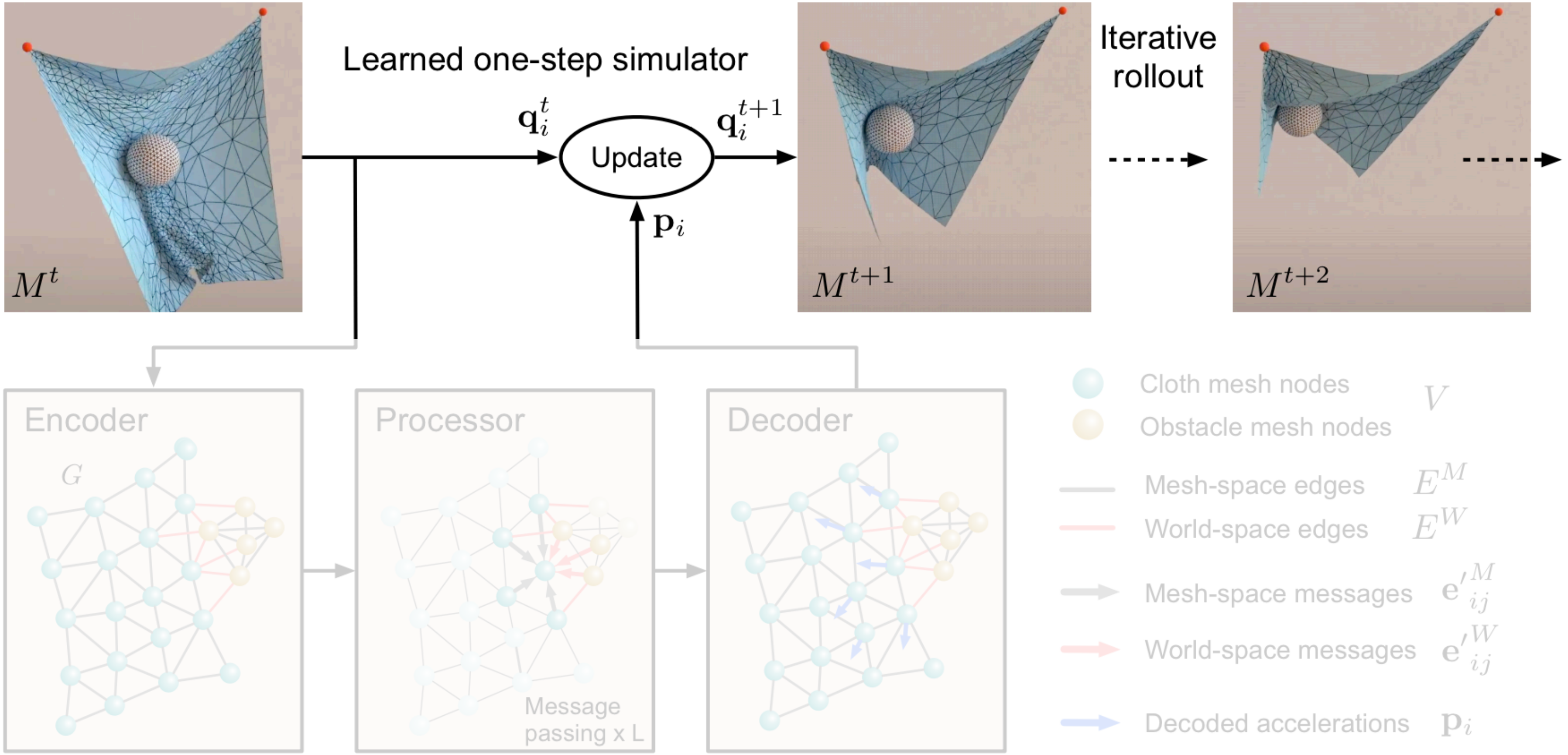
Learning Mesh-Based Simulation with Graph Networks



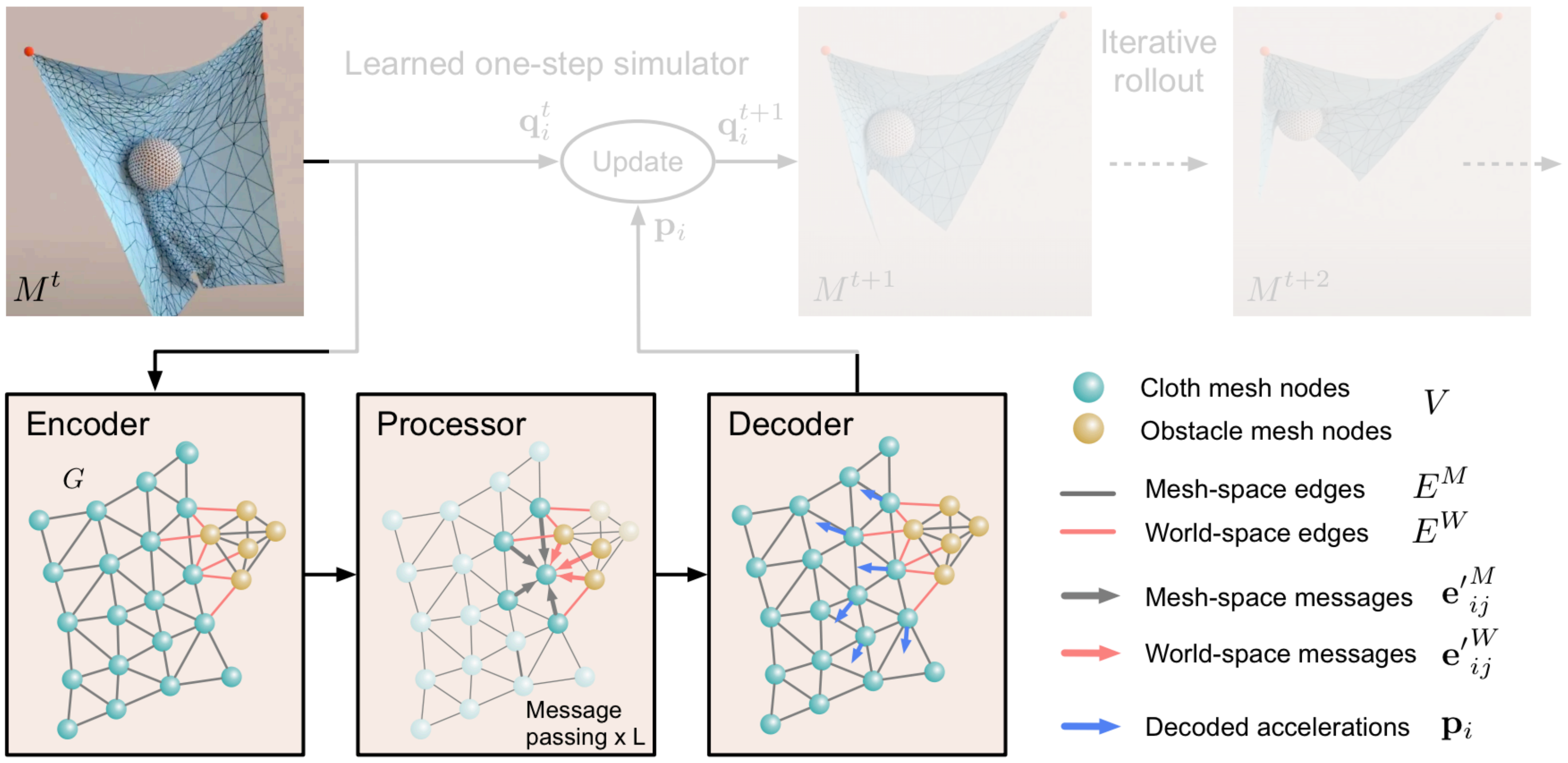
Learning Mesh-Based Simulation with Graph Networks



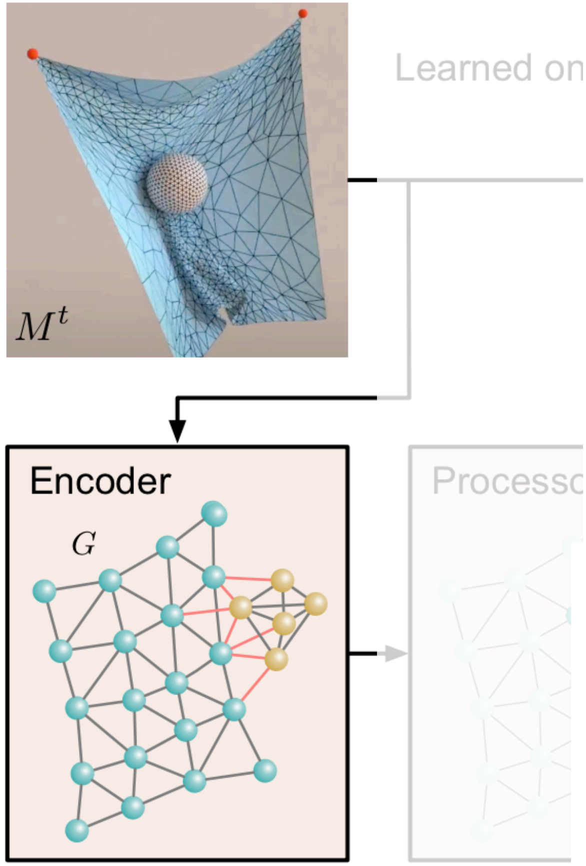
Learning Mesh-Based Simulation with Graph Networks

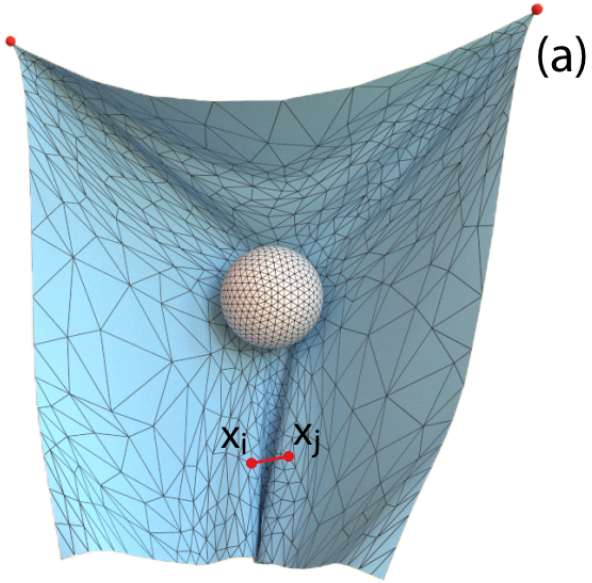


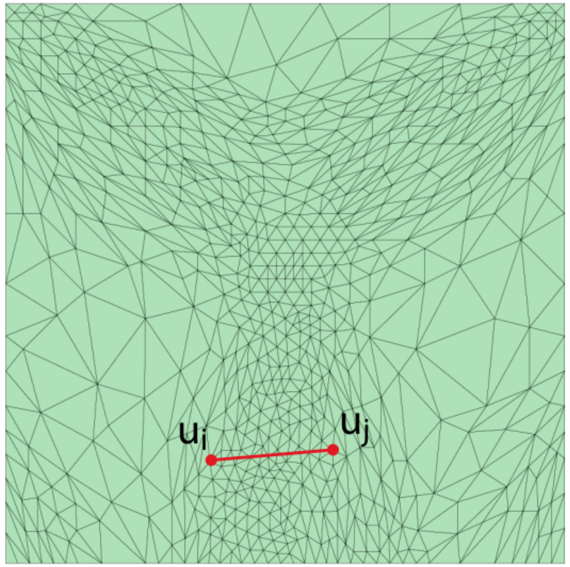
Learning Mesh-Based Simulation with Graph Networks



Learning Mesh-Based Simulation with Graph Networks



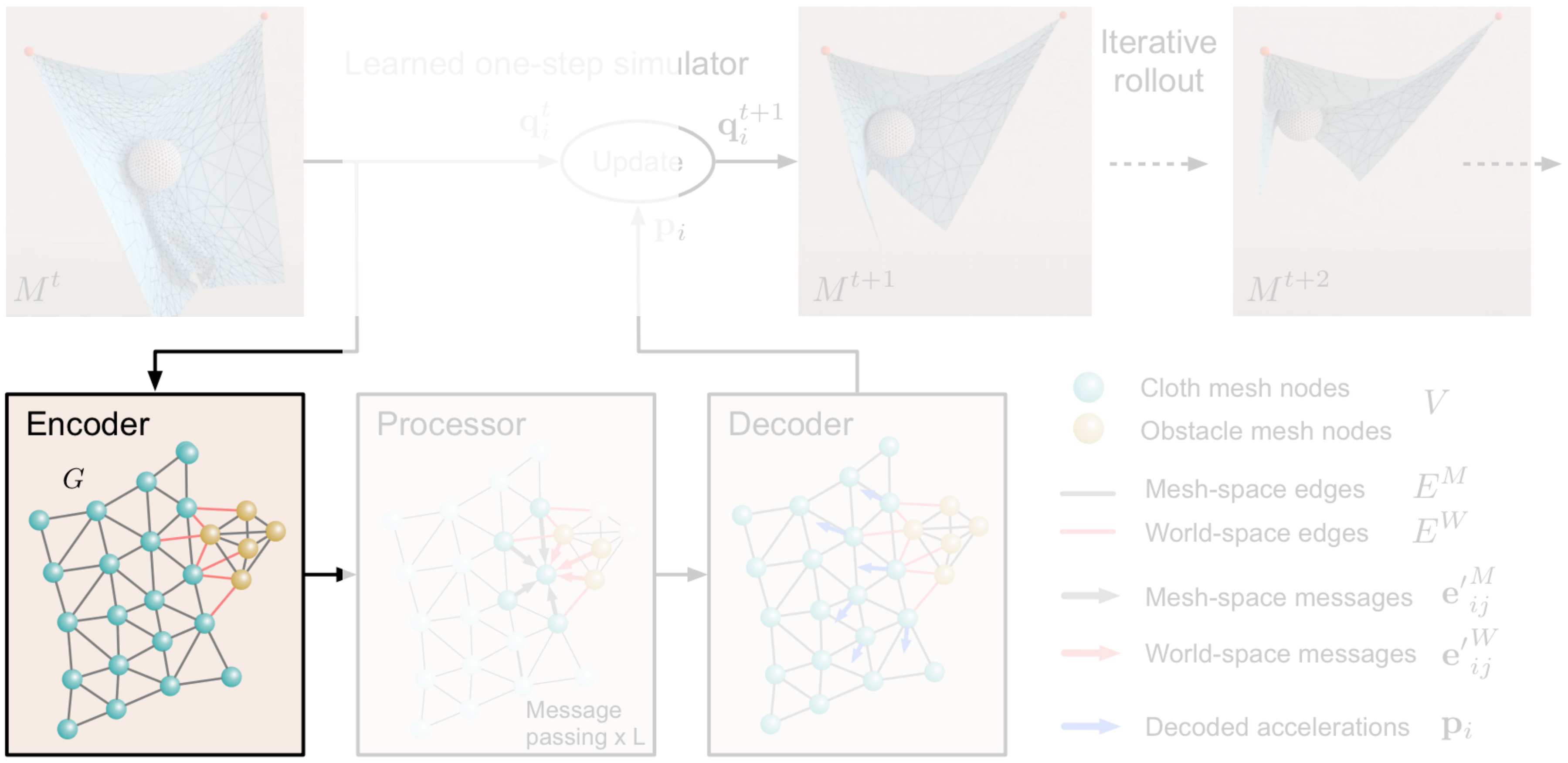




External dynamics:
Computing e.g. collision and contact

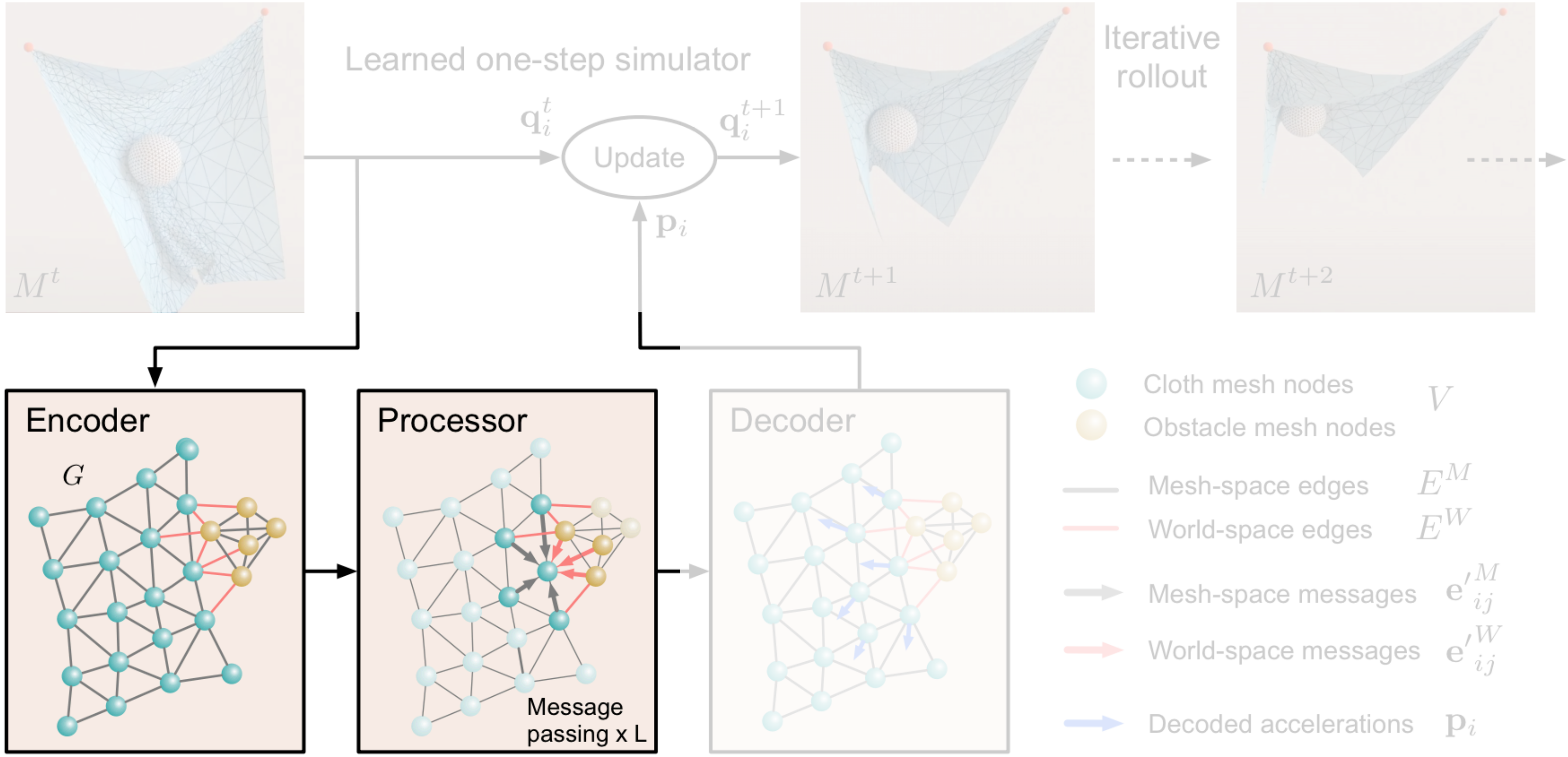
Internal dynamics:
Estimating differential operators on the simulation manifold

Learning Mesh-Based Simulation with Graph Networks



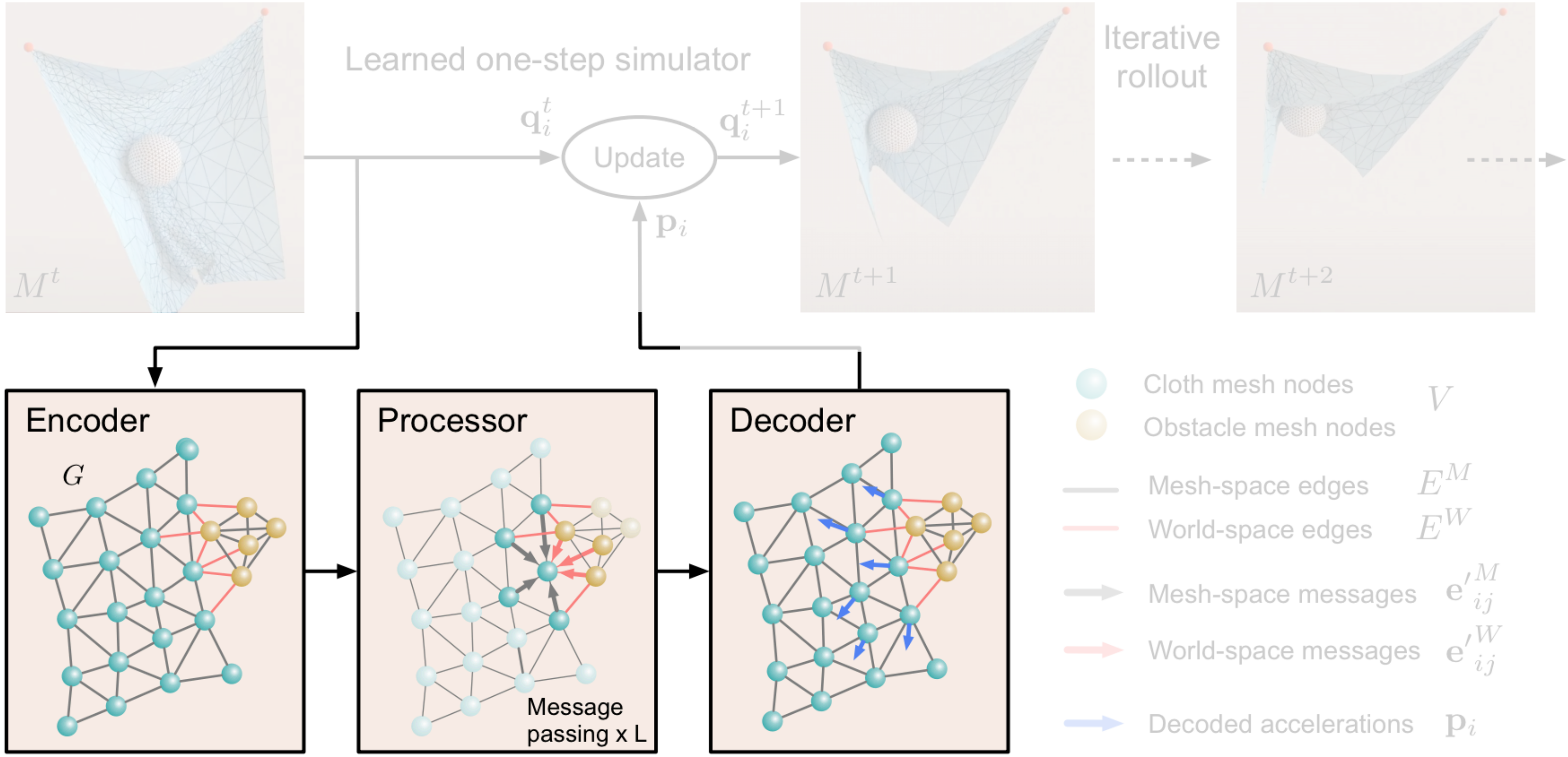
$$
{\mathbf{e}'}_{ij}^M \leftarrow f^M(\mathbf{e}_{ij}^M, \mathbf{v}_i, \mathbf{v}_j), \quad {\mathbf{e}'}_{ij}^W \leftarrow f^W(\mathbf{e}_{ij}^W, \mathbf{v}_i, \mathbf{v}_j), \quad {\mathbf{v}'}_i \leftarrow f^V(\mathbf{v}_i, \sum_j {\mathbf{e}'}_{ij}^M, \sum_j {\mathbf{e}'}_{ij}^W)
$$

Learning Mesh-Based Simulation with Graph Networks

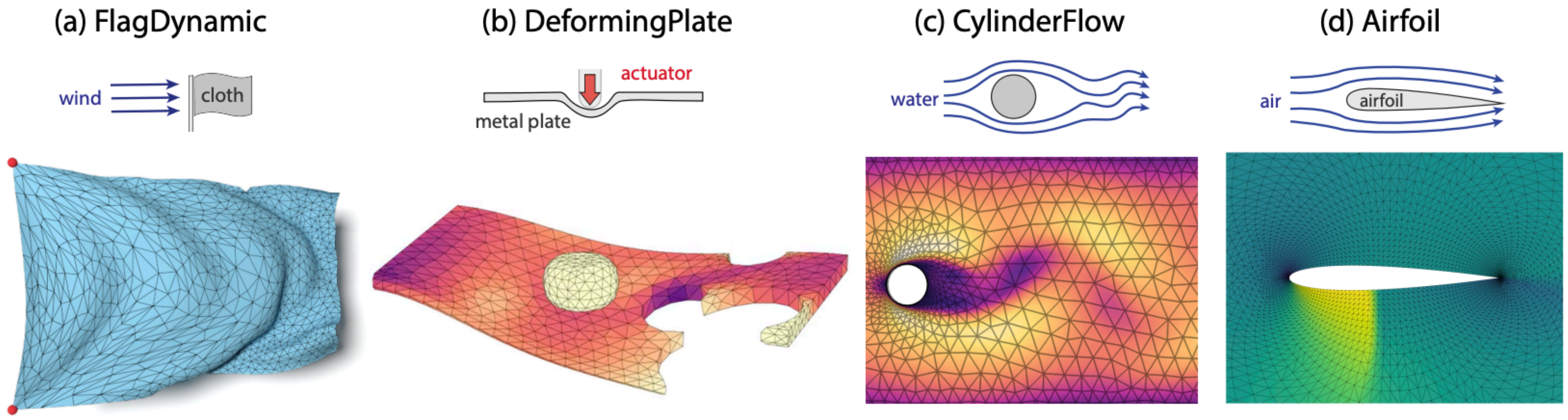


$$\mathbf{e}'^M_{ij} \leftarrow f^M(\mathbf{e}^M_{ij}, \mathbf{v}_i, \mathbf{v}_j), \quad \mathbf{e}'^W_{ij} \leftarrow f^W(\mathbf{e}^W_{ij}, \mathbf{v}_i, \mathbf{v}_j), \quad \mathbf{v}'_i \leftarrow f^V(\mathbf{v}_i, \sum_j \mathbf{e}'^M_{ij}, \sum_j \mathbf{e}'^W_{ij})$$

Learning Mesh-Based Simulation with Graph Networks

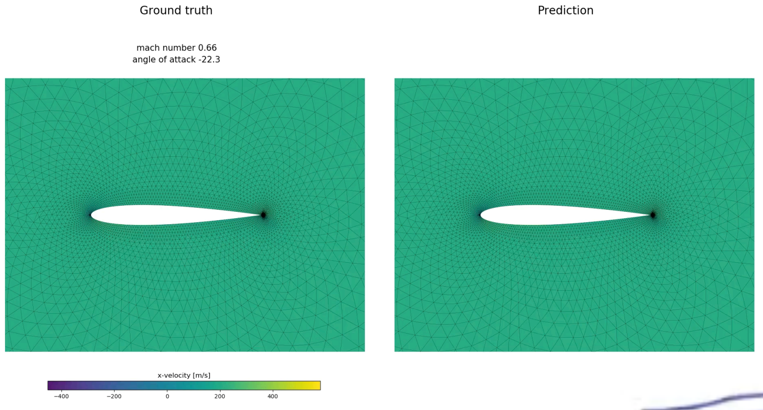


MeshGraphNet: Learning mesh-based simulation with Graph Networks

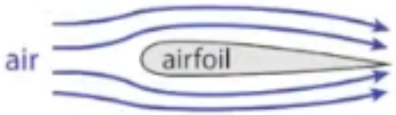


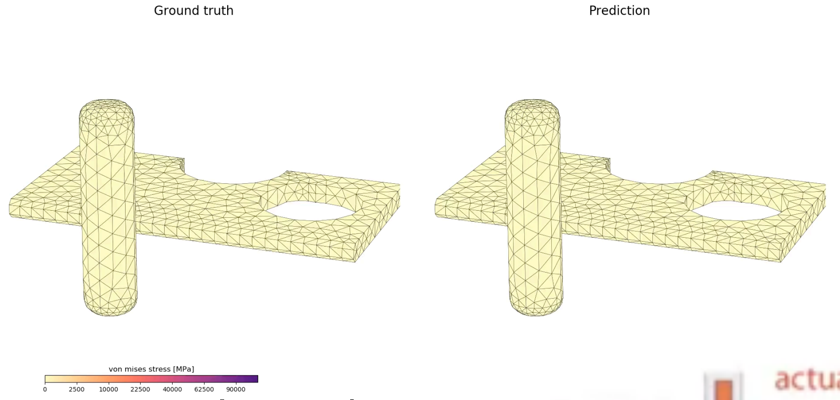
- Versatile

Evaluations: MeshGraphNet is versatile

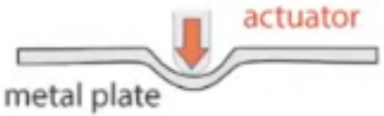


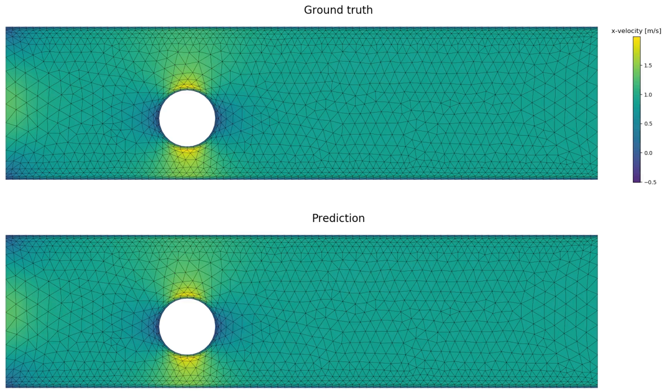
Compressible Aerodynamics



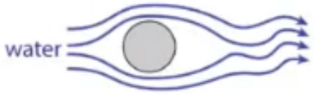


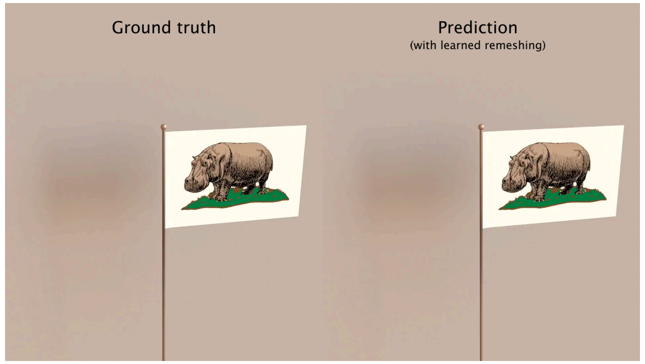
Structural Mechanics





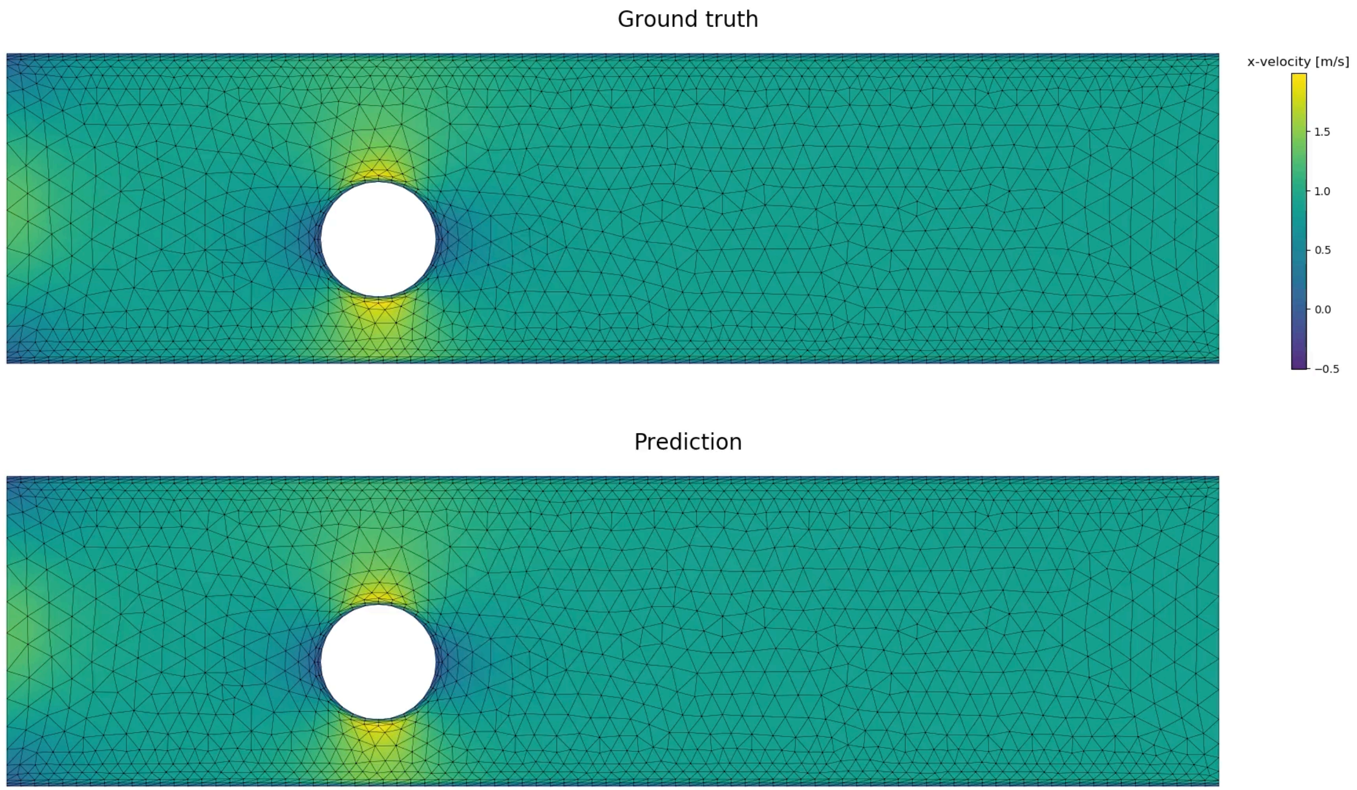
Incompressible Fluid Simulation





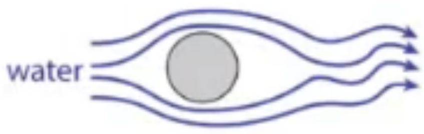
Cloth Simulation

Results: Incompressible Fluids

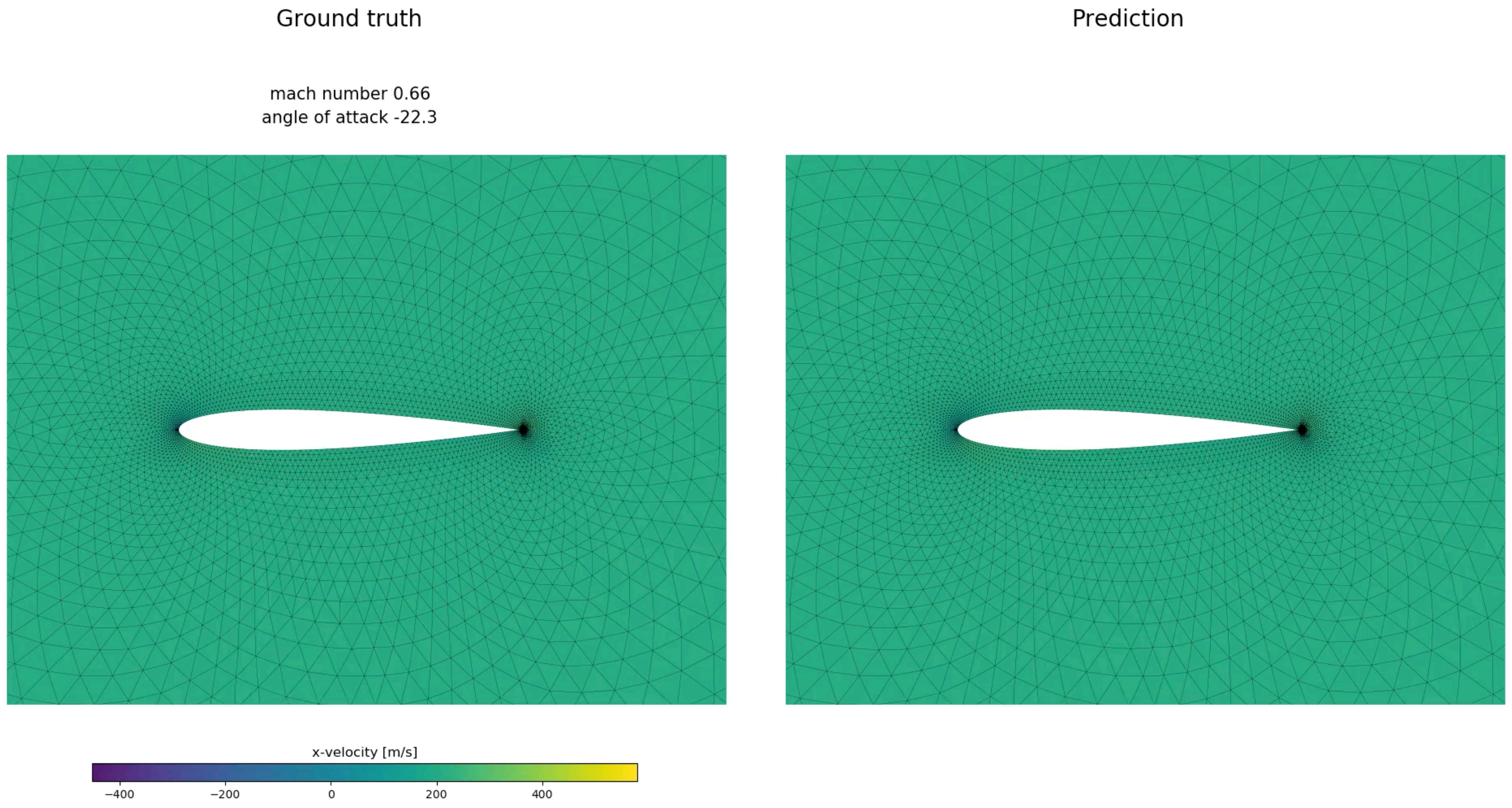


Training data:
COMSL

Network output:
Velocity Field
Pressure Field

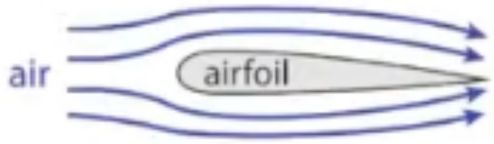


Results: Aerodynamics

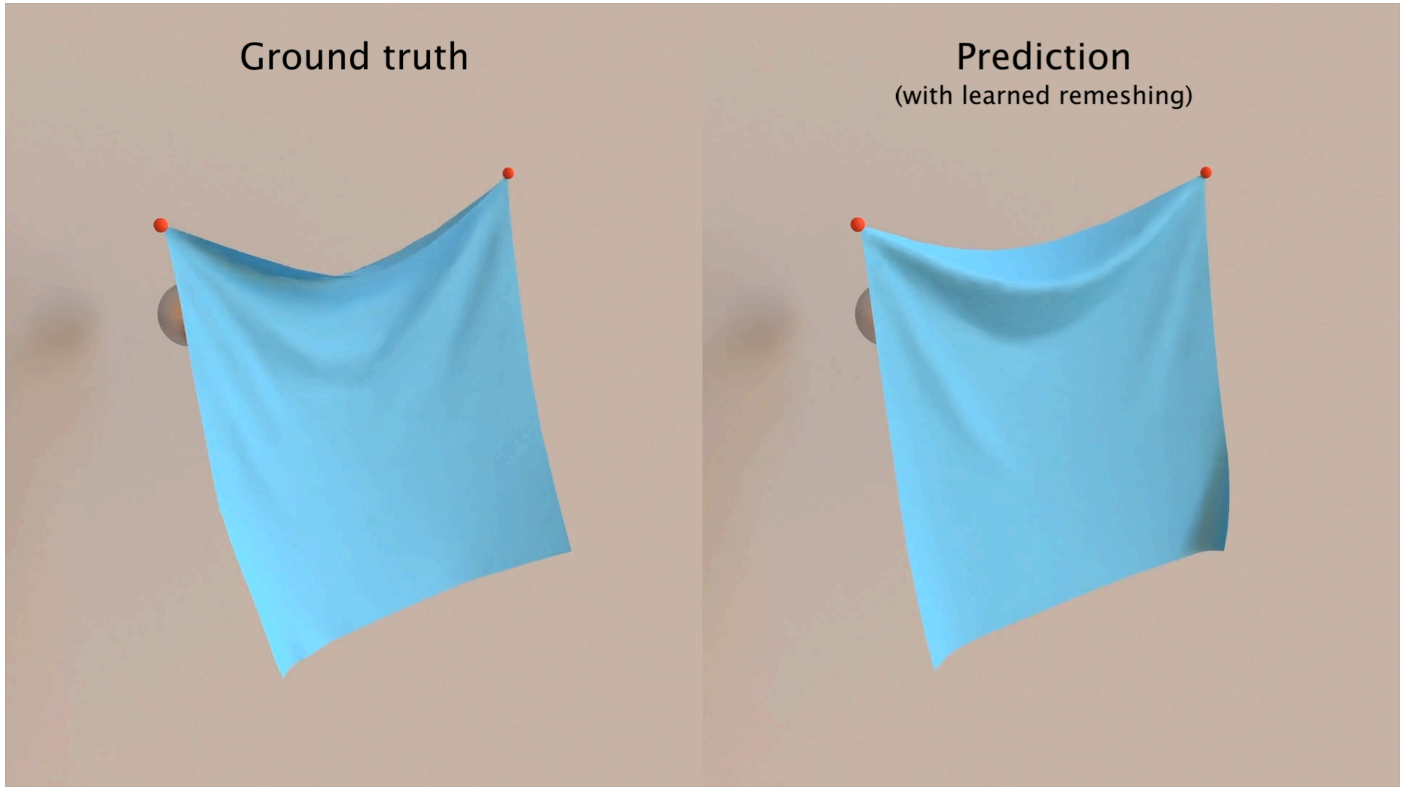


Training data:
SU2

Network output:
Velocity Field
Density Field
Pressure Field

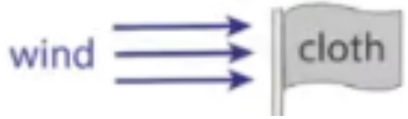


Result: Cloth Dynamics

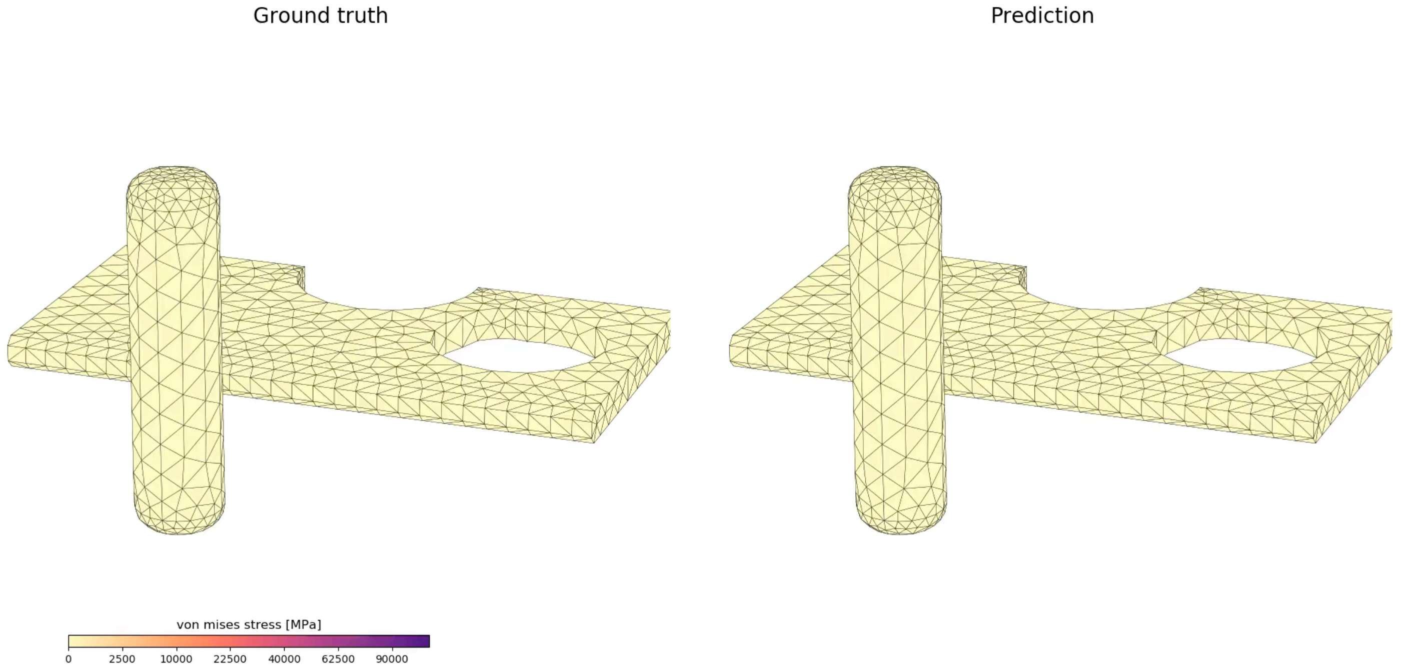


Training data:
Arcsim

Network output:
Per-node
acceleration

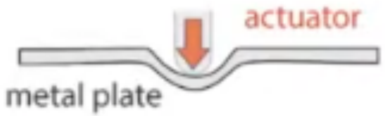


Result: Structural Mechanics

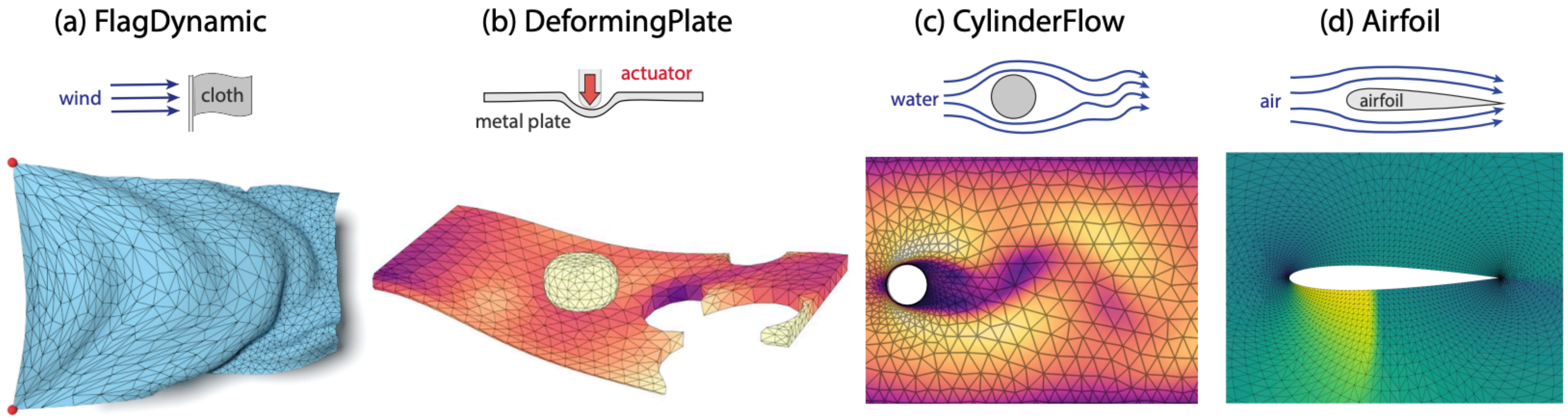


Training data:
COMSOL

Network output:
Per-node
position change



MeshGraphNet: Learning mesh-based simulation with Graph Networks



- Versatile
- Better and stabler rollout compare to prior works

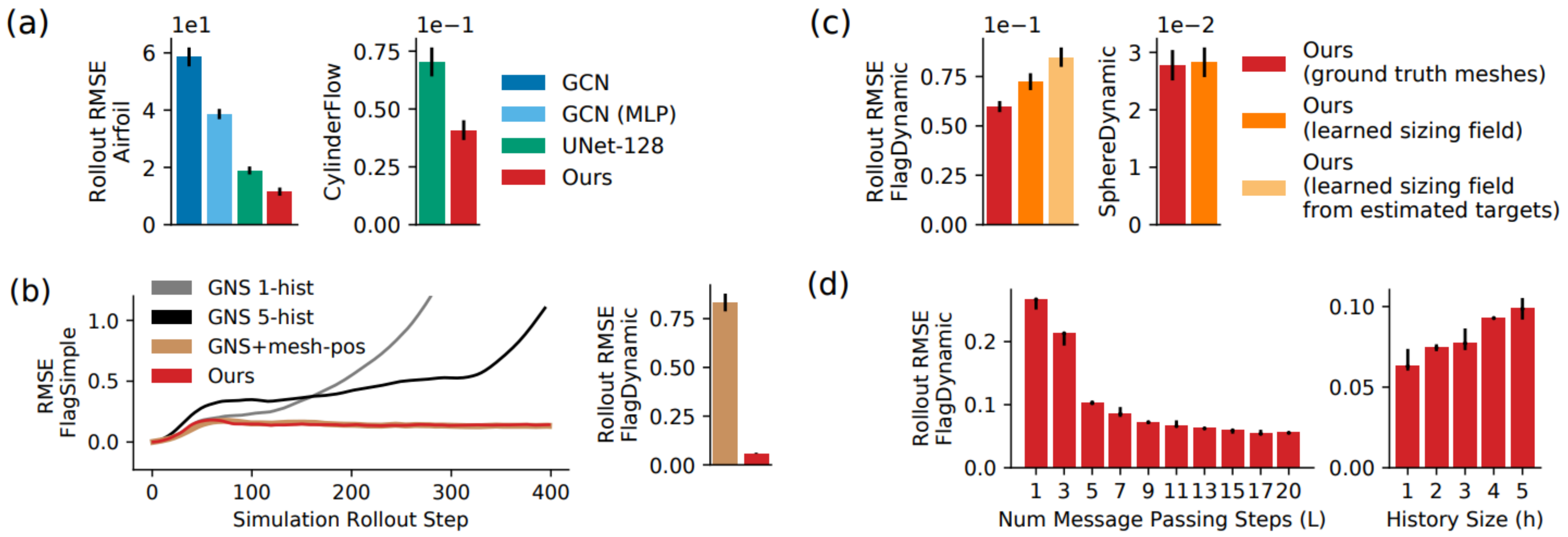
Stable Rollout - noise

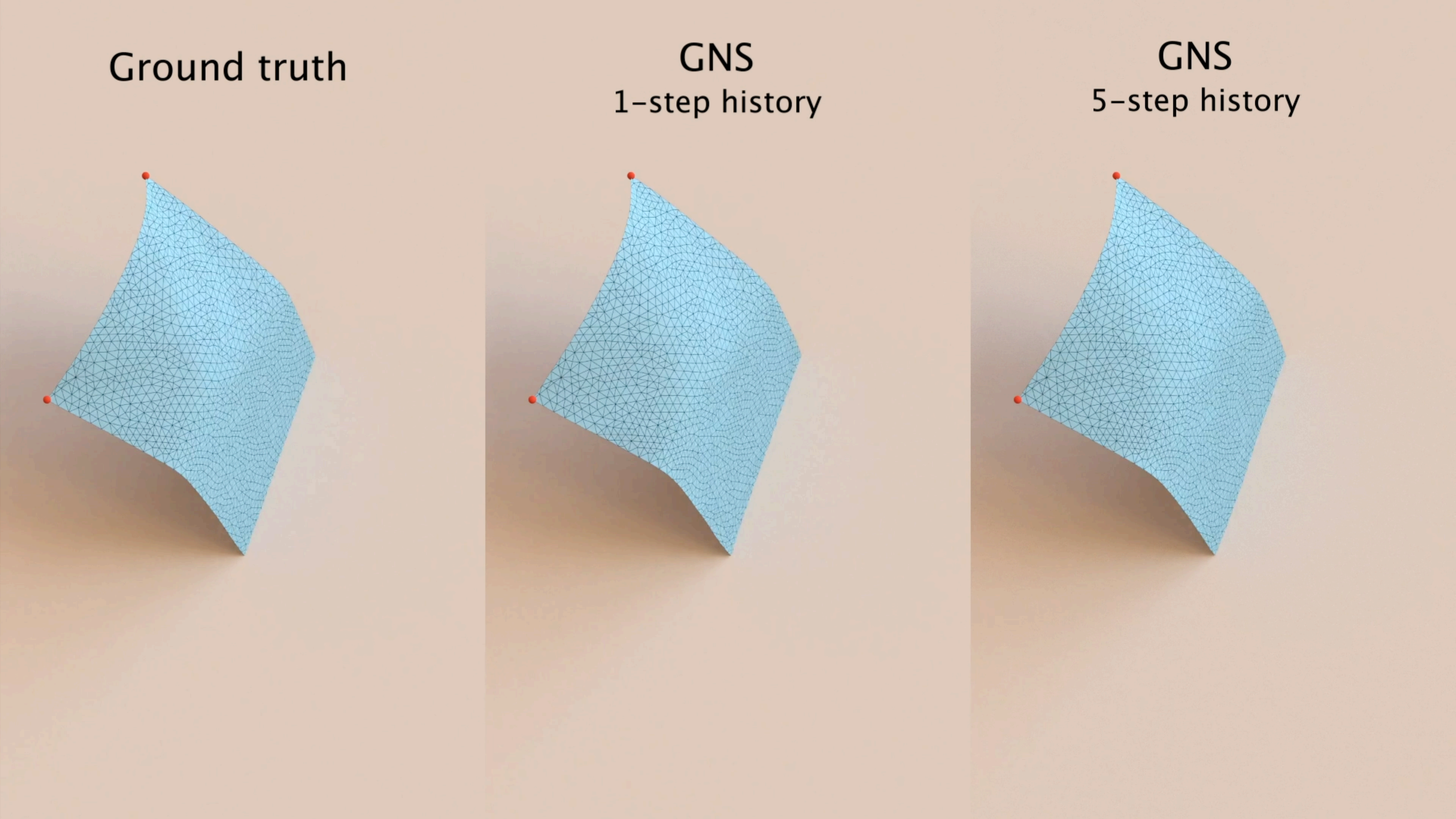
- For stable roll-out
  - Add noise to training dataset

    -> model see inputs that are corrupted by noise

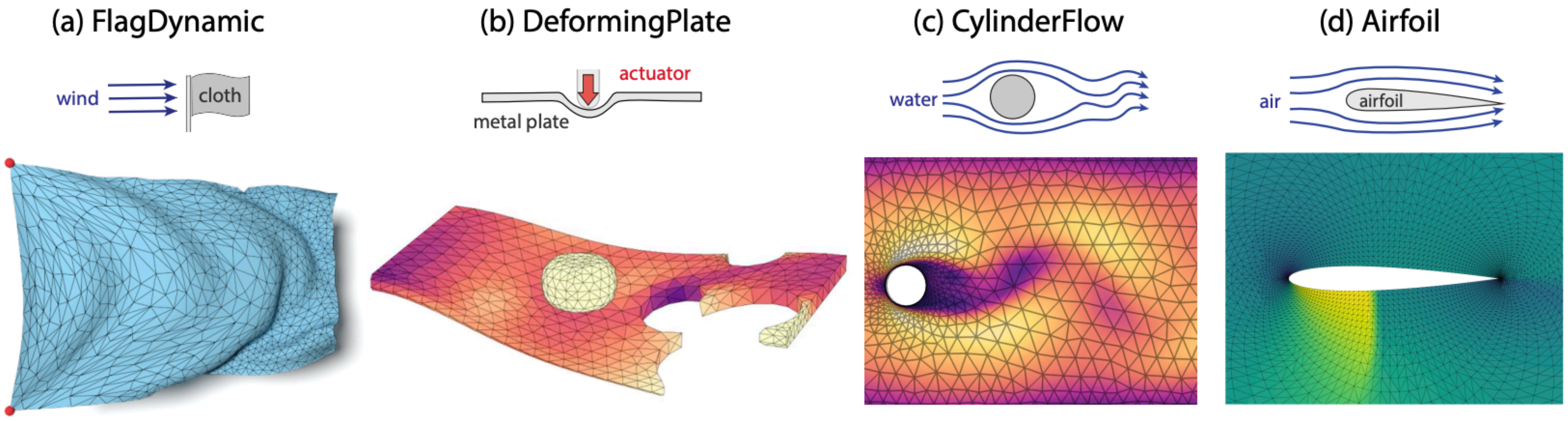
Dataset	Batch size	Noise scale
FLAGSIMPLE	1	pos: 1e-3
FLAGDYNAMIC	1	pos: 3e-3
SPHEREDYNAMIC	1	pos: 1e-3
DEFORMINGPLATE	2	pos: 3e-3
CYLINDERFLOW	2	momentum: 2e-2
AIRFOIL	2	momentum: 1e1, density: 1e-2

Stable Rollout - evaluation




Ground truth
GNS
1-step history
GNS
5-step history

MeshGraphNet: Learning mesh-based simulation with Graph Networks



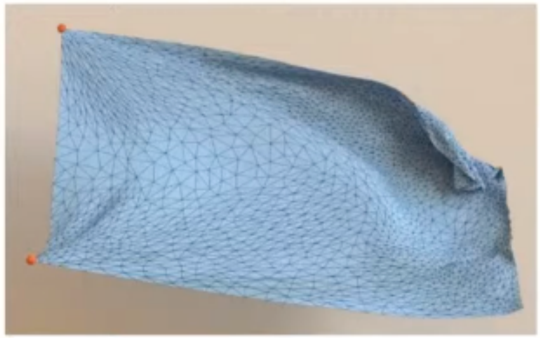
- Versatile
- Better and stabler rollout compare to prior works
- Spatial-temporal Adaptive resolution (adaptive mesh refinement)

Adaptive Remeshing

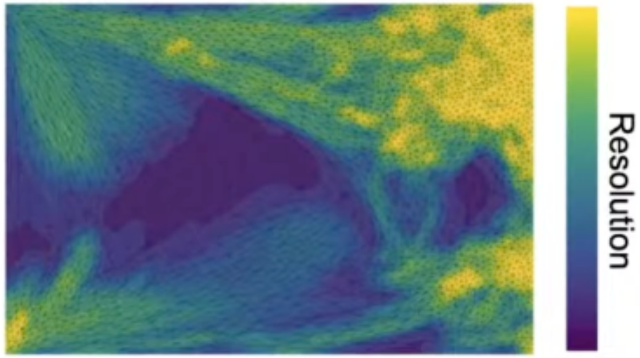
- Step1: Decide target resolution at each point in space (domain specific heuristics, sizing field tensor for cloth simulation)
- Step2: Adjust the mesh

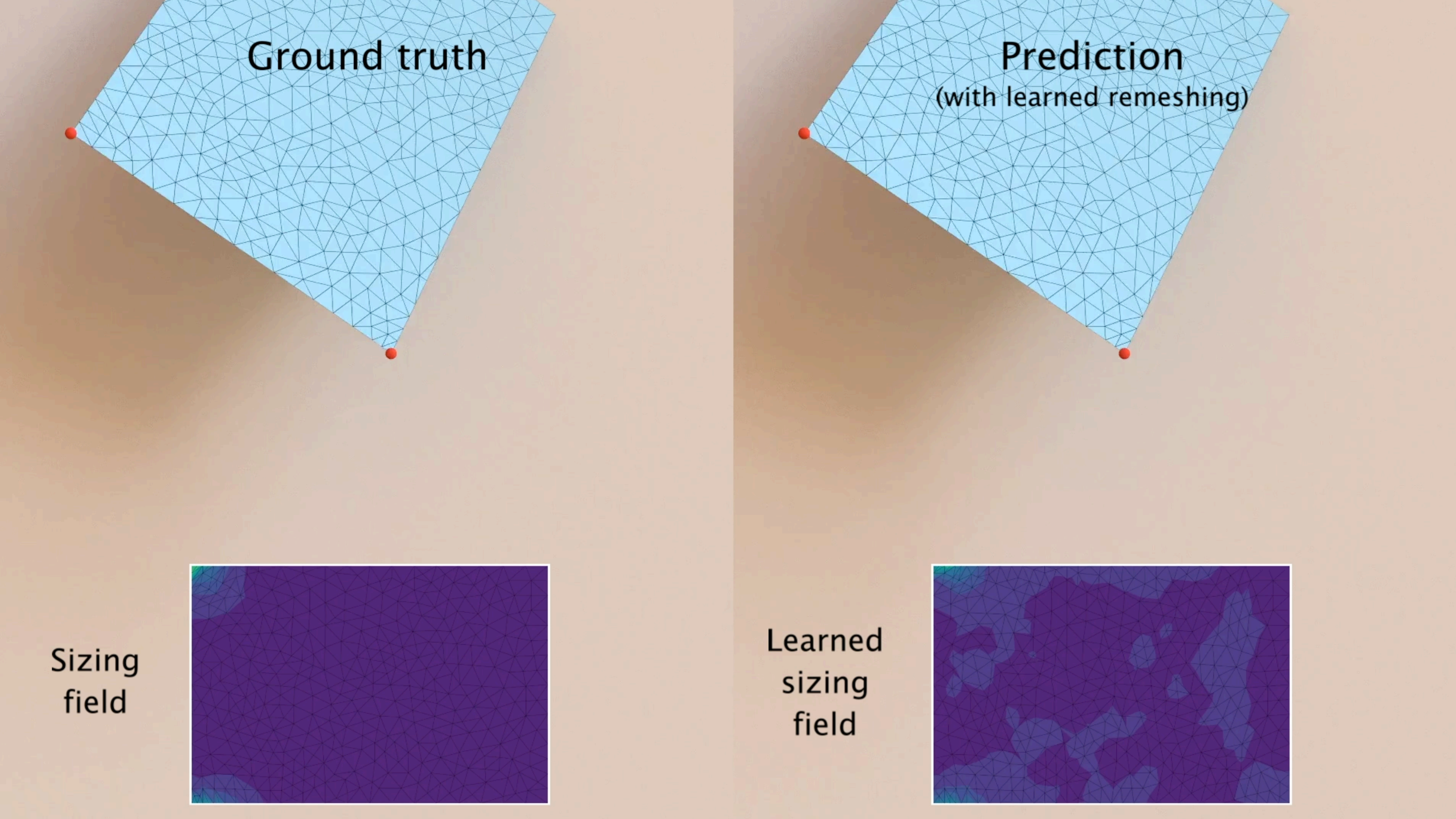
- MeshGraphNet:
  - Predict sizing field directly using graph neural network
    - Given GT supervision from classical approach mentioned above

Flag in world space

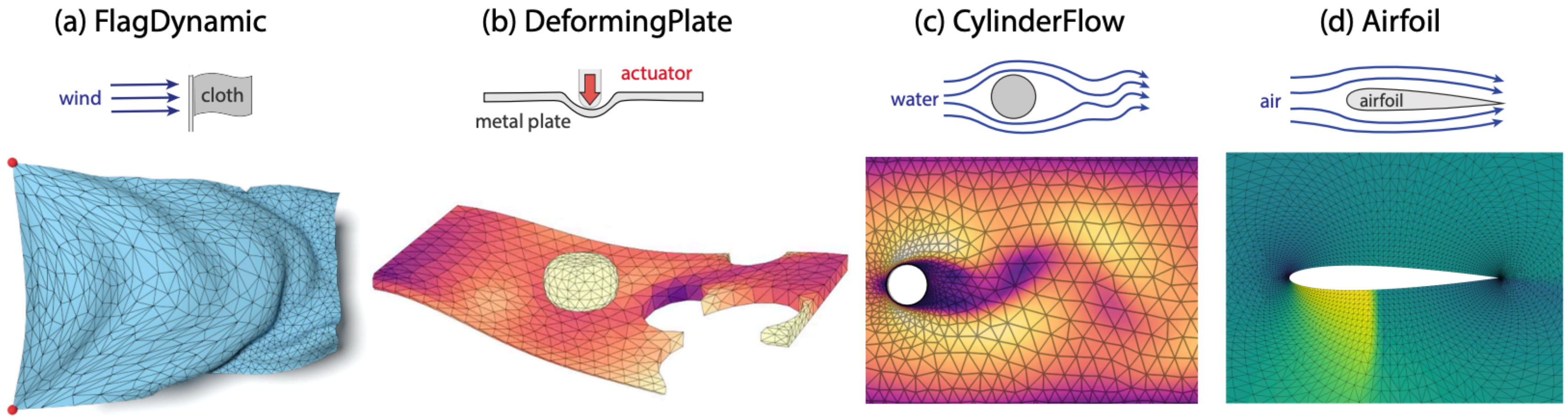


Sizing field in mesh space




Ground truth
Prediction
(with learned remeshing)
Sizing field
Learned sizing field

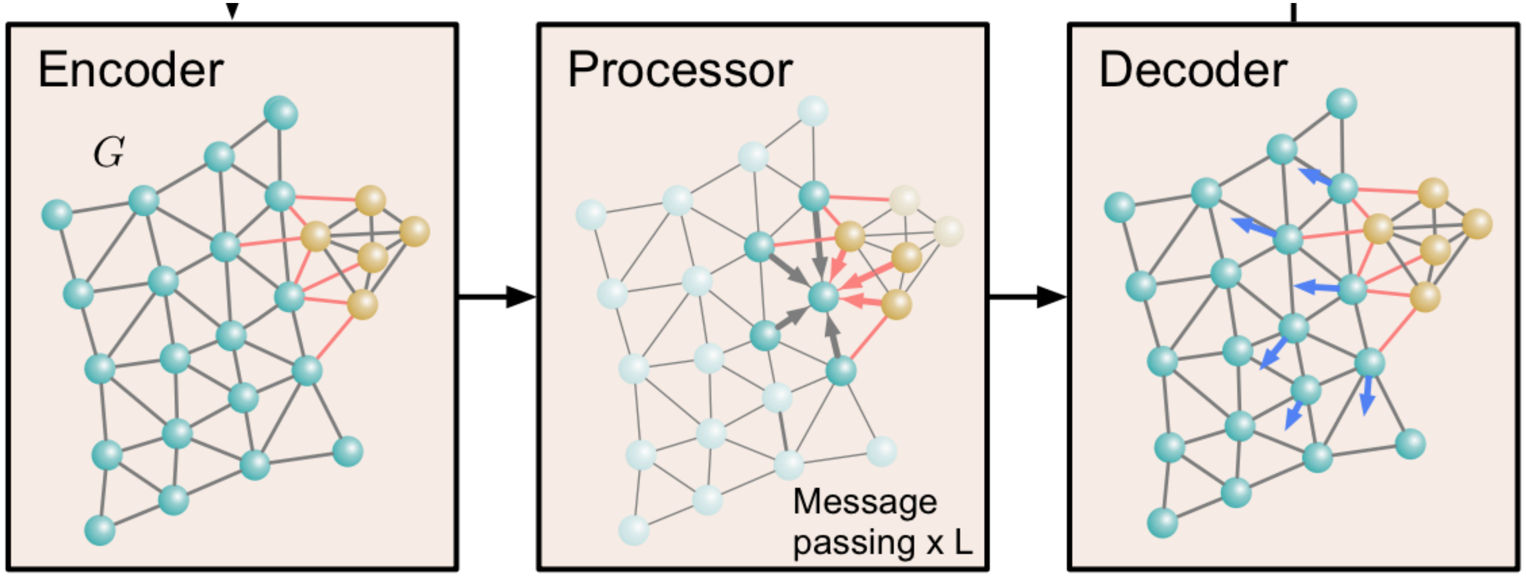
MeshGraphNet: Learning mesh-based simulation with Graph Networks

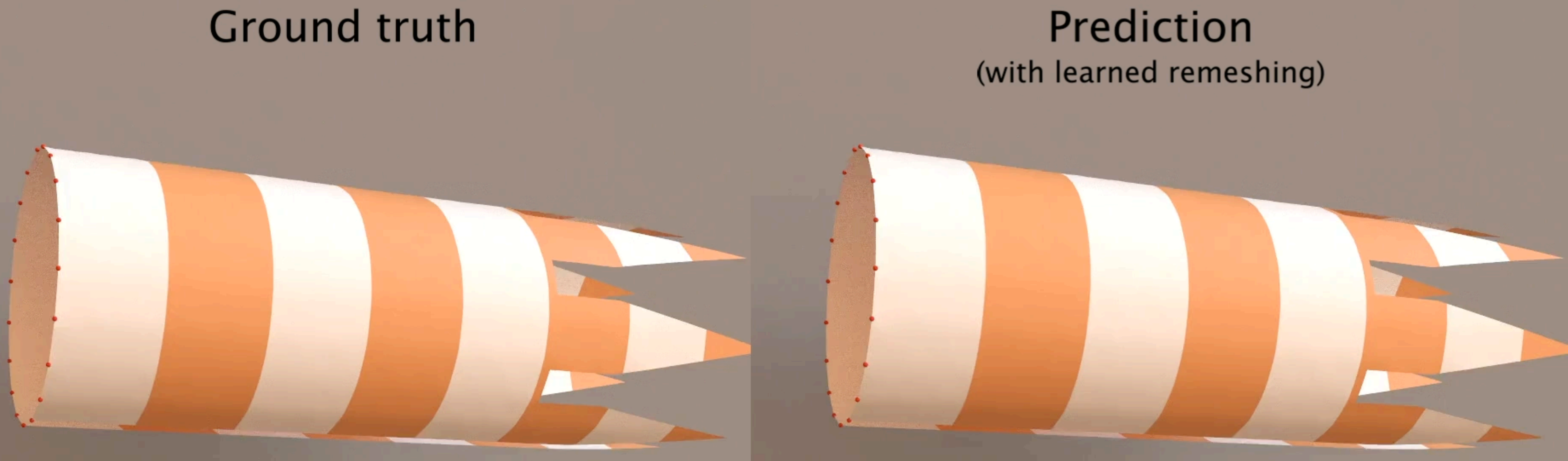


- Versatile
- Better and stabler rollout compare to prior works
- Spatial-temporal Adaptive resolution (adaptive mesh refinement)
- Generalize to large domain

Generalize to more nodes

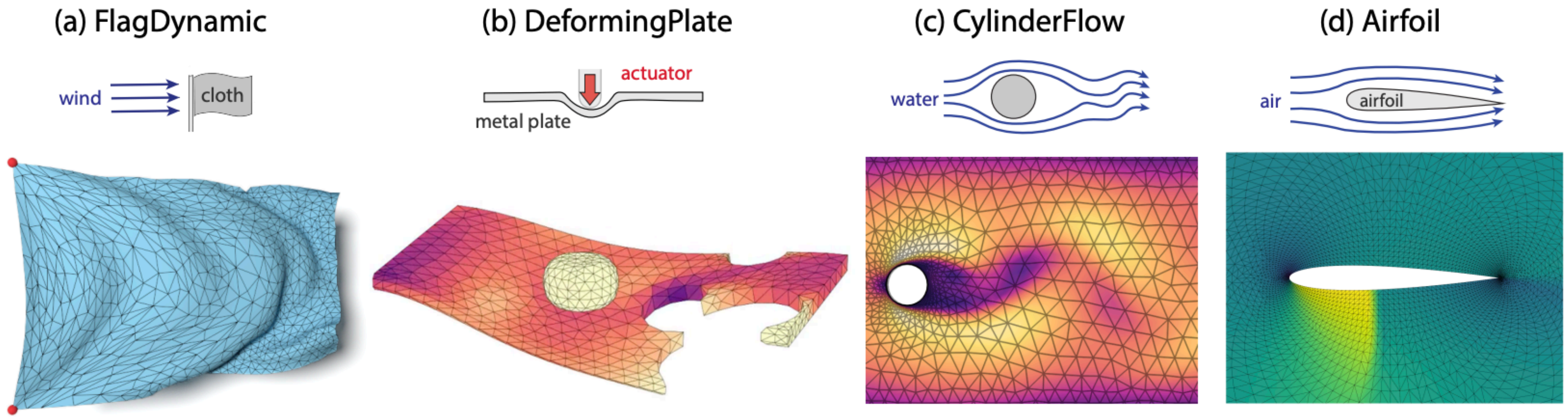
- Learned local Interaction
- Train on 2k nodes, generalize to >20k nodes





Generalize to more Nodes

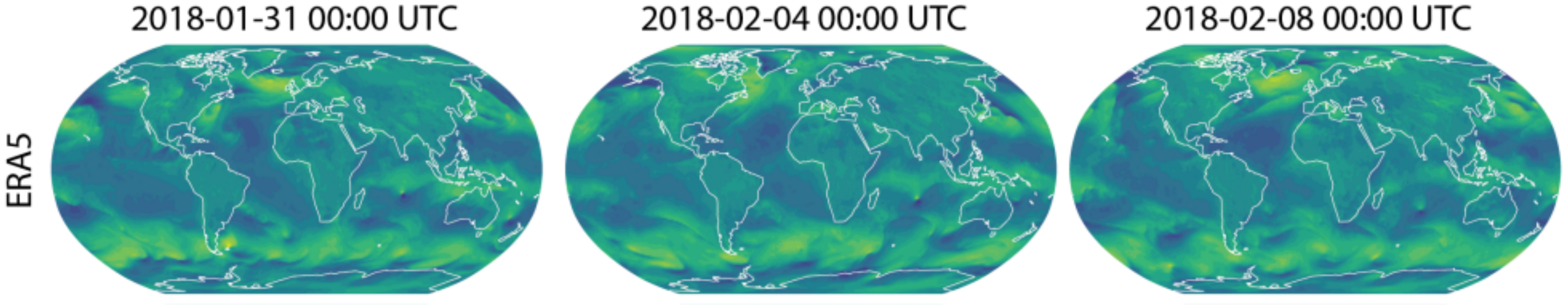
MeshGraphNet: Learning mesh-based simulation with Graph Networks



- Versatile
- Better and stabler rollout compare to prior works
- Spatial-temporal Adaptive resolution (adaptive mesh refinement)
- Generalize to large domain

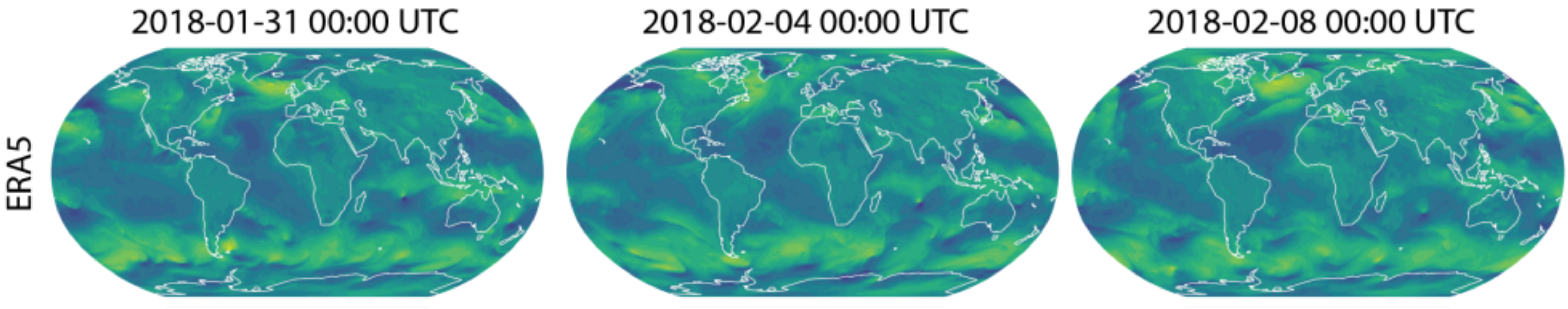
GraphCast: Learning skillful medium-range global weather forecasting

Why Learning?
- numerical weather prediction do not scale well with data
- there vast archives of weather and climatological observations available



GraphCast: Learning skillful medium-range global weather forecasting

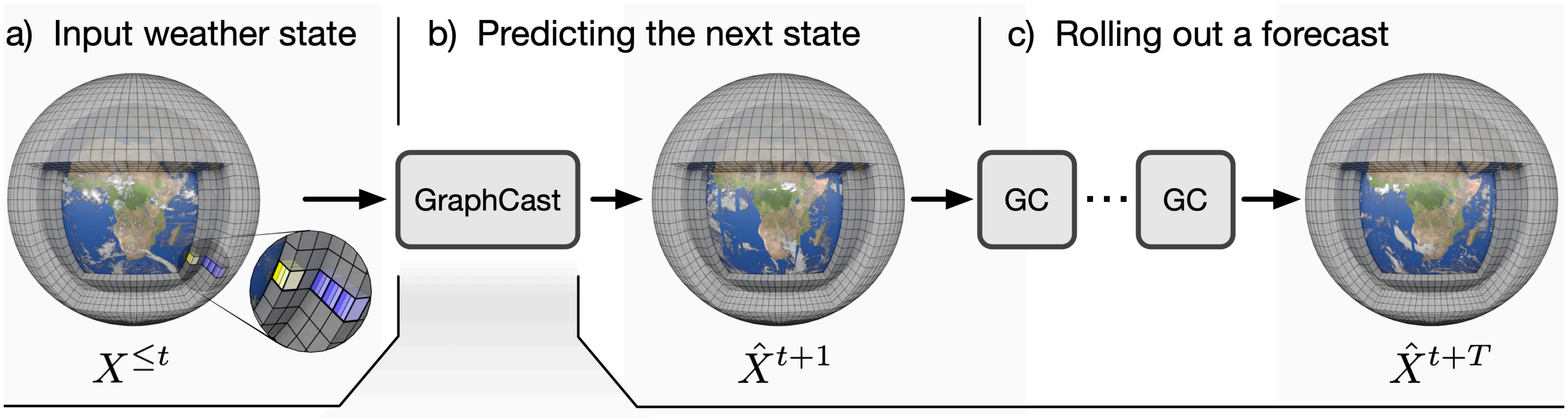
1. Accuracy
   - more accurate than ECMWF's deterministic operational forecasting system
   - outperforms the most accurate previous ML-based weather forecasting model
2. Speed
   - generate a 10-day forecast (35 gigabytes of data) in under 60 seconds on Cloud TPU v4 hardware.



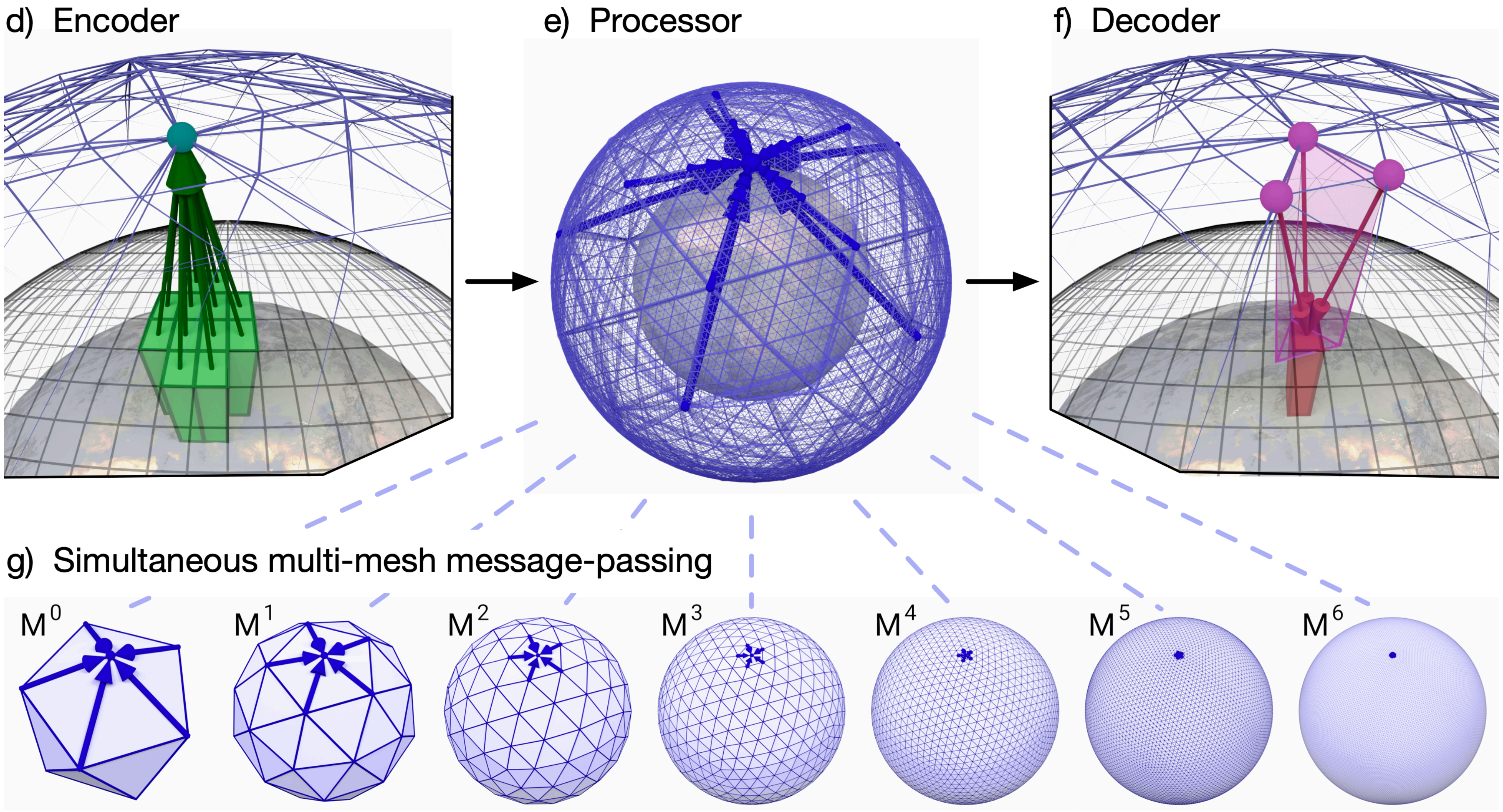
Architecture

$$\hat{X}^{t+1} = \text{GraphCast}(X^t, X^{t-1})$$

$$\hat{X}^{t+1:t+T} = \underbrace{(\text{GraphCast}(X^t, X^{t-1}), \text{GraphCast}(\hat{X}^{t+1}, X^t), \ldots, \text{GraphCast}(\hat{X}^{t+T-1}, \hat{X}^{t+T-2}))}_{1 \ldots T \text{ autoregressive iterations}}$$



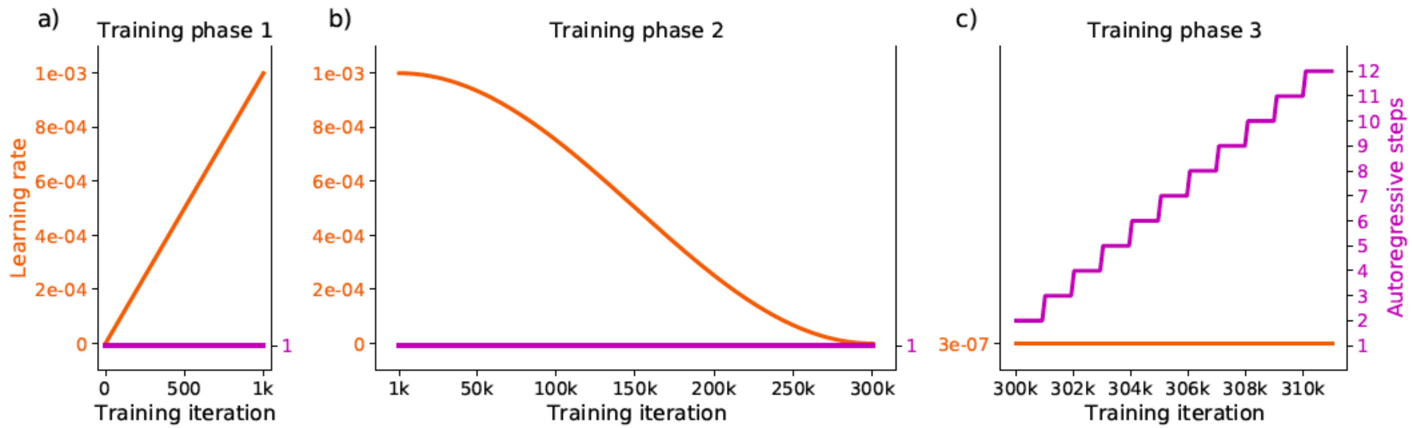
Architecture



Training

$$\mathcal{L}_{\mathrm{MSE}} = \underbrace{\frac{1}{|D_{\mathrm{batch}}|}\sum_{d_0 \in D_{\mathrm{batch}}}}_{\text{forecast date-time}} \underbrace{\frac{1}{T}\sum_{\tau \in 1:T_{\mathrm{train}}}}_{\text{lead time}} \underbrace{\frac{1}{|G_{0.25^\circ}|}\sum_{i \in G_{0.25^\circ}}}_{\text{spatial location}} \underbrace{\sum_{j \in J}}_{\text{variable-level}} s_j w_j a_i \underbrace{\left(\hat{x}_{i,j}^{d_0+\tau} - x_{i,j}^{d_0+\tau}\right)^2}_{\text{squared error}} \quad (2)$$

- autoregressive, multi-step loss
  - help minimize error accumulation over long forecasts



Training schedule.

Evaluation

- Compare with the HRES 10-day forecast and other baseline ML models.

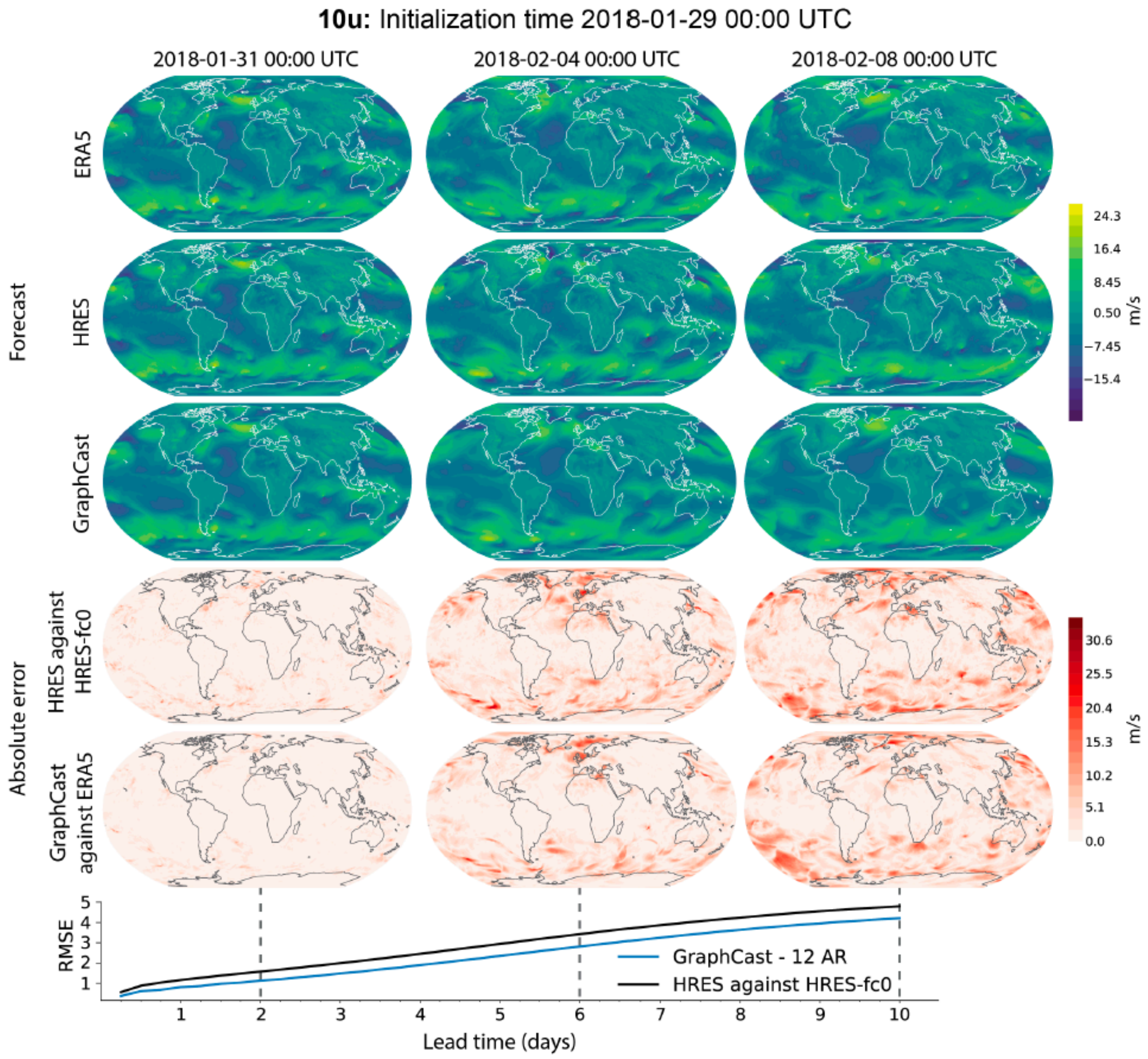
$$\mathcal{L}_{\text{RMSE}}^{j,\tau} = \frac{1}{|D_{\text{eval}}|} \sum_{d_0 \in D_{\text{eval}}} \sqrt{\frac{1}{|G_{0.25^\circ}|} \sum_{i \in G_{0.25^\circ}} a_i \left(\hat{x}_{j,i}^{d_0+\tau} - x_{j,i}^{d_0+\tau} \right)^2} \tag{A.20}$$

where

- $d_0 \in D_{\text{eval}}$ represent forecast initialization date-times in the evaluation dataset,
- $j \in J$ indexes the variable and level, e.g., $J = \{\text{z}1000, \text{z}850, \ldots, 2\text{T}, \text{MSL}\}$,
- $i \in G_{0.25^\circ}$ are the location (latitude and longitude) coordinates in the grid,
- $\hat{x}_{j,i}^{d_0+\tau}$ and $x_{j,i}^{d_0+\tau}$ are predicted and target values for some variable-level, location, and lead time,
- a_i is the area of the latitude-longitude grid cell (normalized to unit mean over the grid) which varies with latitude.

Evaluation

results show GraphCast comprehensively outperforms HRES's weather forecasting skill across10-day forecasts, at 0.25° horizontal resolution.



Open Questions

- How to incorporate appropriate symmetry and conservation properties into the architecture of a Graph Neural Network (GNN)?
- How to simulate multiscale systems? Here, multiscale can include spatial, temporal, etc
- How to achieve more accurate simulations with reduced computational cost for graphs with a large number of nodes (such as millions or billions)?
- How to perform complex and diverse inverse design on graphs with a large number of nodes?

GraphCast was trained to minimize an objective function over 12-step forecasts (3 days) against ERA5 targets, using gradient descent. The objective function was,

$$\mathcal{L}_{\text{MSE}} = \underbrace{\frac{1}{|D_{\text{batch}}|}\sum_{d_0\in D_{\text{batch}}}}_{\text{forecast date-time}} \underbrace{\frac{1}{T}\sum_{\tau\in 1:T_{\text{train}}}}_{\text{lead time}} \underbrace{\frac{1}{|G_{0.25^\circ}|}\sum_{i\in G_{0.25^\circ}}}_{\text{spatial location}} \underbrace{\sum_{j\in J}}_{\text{variable-level}} s_j w_j a_i \underbrace{(\hat{x}^{d_0+\tau}_{i,j} - x^{d_0+\tau}_{i,j})^2}_{\text{squared error}} \tag{2}$$

which averages the squared errors over forecast date-times, lead times, spatial locations, variables and levels, where

- $d_0 \in D_{\text{batch}}$ represent forecast initialization date-times in a batch of forecasts in the training set,
- $\tau \in 1 : T_{\text{train}}$ are the lead times that correspond to the T_{train} autoregressive steps during training,
- $i \in G_{0.25^\circ}$ are the spatial latitude and longitude coordinates in the grid,
- $j \in J$ indexes the variable and level, e.g., $J = \{\text{z1000}, \text{z850}, \ldots, \text{2T}, \text{MSL}\}$,
- $\hat{x}^{d_0+\tau}_{j,i}$ and $x^{d_0+\tau}_{j,i}$ are predicted and target values for some variable-level, location, and lead time,
- s_j is the per-variable-level inverse variance of single-timestep differences,
- w_j is the per-variable-level loss weight,
- a_i is the normalized area of the latitude-longitude grid cell, which varies with latitude.

The quantities $s_j = \mathbb{V}_{i,t}\left[x^{t+1}_{i,j} - x^{t}_{i,j}\right]^{-1}$ are per-variable-level inverse variance estimates of the time differences. The w_j are per-variable-level loss weights we specified in a simple way, to control how heavily different target variables are weighed during optimization. The a_i weight depends on latitude, and weights the errors proportionally to the area of their corresponding grid cells. See the Appendix A.3 for full details of these symbols and indices.

input

Type	Variable name	Short name	ECMWF Parameter ID	Role (accumulation period, if applicable)
Atmospheric	Geopotential	z	129	Input/Predicted
Atmospheric	Specific humidity	q	133	Input/Predicted
Atmospheric	Temperature	t	130	Input/Predicted
Atmospheric	U component of wind	u	131	Input/Predicted
Atmospheric	V component of wind	v	132	Input/Predicted
Atmospheric	Vertical velocity	w	135	Input/Predicted
Single	2 metre temperature	2t	167	Input/Predicted
Single	10 metre u wind component	10u	165	Input/Predicted
Single	10 metre v wind component	10v	166	Input/Predicted
Single	Mean sea level pressure	msl	151	Input/Predicted
Single	Total precipitation	tp	28	Input/Predicted (6h)
Single	TOA incident solar radiation	tisr	212	Input (1h)
Static	Geopotential at surface	z	129	Input
Static	Land-sea mask	lsm	172	Input
Static	Latitude	n/a	n/a	Input
Static	Longitude	n/a	n/a	Input
Clock	Local time of day	n/a	n/a	Input
Clock	Elapsed year progress	n/a	n/a	Input

Table A.1 | **ECMWF variables used in our datasets.** The "Type" column indicates whether the variable represents a *static* property, a time-varying *single*-level property (e.g., surface variables are included), or a time-varying *atmospheric* property. The "Variable name" and "Short name" columns are ECMWF's labels. The "ECMWF Parameter ID" column is a ECMWF's numeric label, and can be used to construct the URL for ECMWF's description of the variable, by appending it as suffix to the following prefix, replacing "ID" with the numeric code: `https://apps.ecmwf.int/codes/grib/param-db/?id=ID`. The "Role" column indicates whether the variable is something our model takes as input and predicts, or only uses as input context (the double horizontal line separates predicted from input-only variables, to make the partitioning more visible.

grid nodes

Grid nodes $\mathcal{V}^{\mathrm{G}}$ represents the set containing each of the grid nodes v_i^{G}. Each grid node represents a vertical slice of the atmosphere at a given latitude-longitude point, i. The features associated with each grid node v_i^{G} are $\mathbf{v}_i^{\mathrm{G,features}} = [\mathbf{x}_i^{t-1}, \mathbf{x}_i^{t}, \mathbf{f}_i^{t-1}, \mathbf{f}_i^{t}, \mathbf{f}_i^{t+1}, \mathbf{c}_i]$, where $\mathbf{x}_i^t$ is the time-dependent weather state X^t corresponding to grid node v_i^{G} and includes all the predicted data variables for all 37 atmospheric levels as well as surface variables. The forcing terms $\mathbf{f}^t$ consist of time-dependent features that can be computed analytically, and do not require to be predicted by GraphCast. They include the total incident solar radiation at the top of the atmosphere, accumulated over 1 hour, the sine and cosine of the local time of day (normalized to [0, 1)), and the sine and cosine of the of year progress (normalized to [0, 1)). The constants $\mathbf{c}_i$ are static features: the binary land-sea mask, the geopotential at the surface, the cosine of the latitude, and the sine and cosine of the longitude. At 0.25° resolution, there is a total of $721 \times 1440 = 1{,}038{,}240$ grid nodes, each with (5 *surface variables* + 6 *atmospheric variables* × 37 *levels*) × 2 *steps* + 5 *forcings* × 3 *steps* + 5 *constant* = 474 input features.

Mesh nodes

Mesh nodes $\mathcal{V}^{\mathrm{M}}$ represents the set containing each of the mesh nodes v_i^{M}. Mesh nodes are placed uniformly around the globe in a R-refined icosahedral mesh M^R. M^0 corresponds to a unit-radius icosahedron (12 nodes and 20 triangular faces) with faces parallel to the poles (see Figure 1g). The mesh is iteratively refined $M^r \to M^{r+1}$ by splitting each triangular face into 4 smaller equilateral faces, resulting in an extra node in the middle of each edge, and re-projecting the new nodes back onto the unit sphere.[16] Features $\mathbf{v}_i^{\mathrm{M,features}}$ associated with each mesh node v_i^{M} include the cosine of the latitude, and the sine and cosine of the longitude. GraphCast works with a mesh that has been refined $R = 6$ times, M^6, resulting in 40,962 mesh nodes (see Supplementary Appendix Table A.2), each with 3 input features.

Refinement	0	1	2	3	4	5	6
Num Nodes	12	42	162	642	2,562	10,242	40,962
Num Faces	20	80	320	1,280	5,120	20,480	81,920
Num Edges	60	240	960	3,840	15,360	61,440	245,760
Num Multilevel Edges	60	300	1,260	5,100	2,0460	81,900	327,660

Table A.2 | **Multi-mesh statistics.** Statistics of the multilevel refined icosahedral mesh as function of the refinement level R. Edges are considered to be bi-directional and therefore we count each edge in the mesh twice (once for each direction).

Mesh edges

Mesh edges $\mathcal{E}^{\mathrm{M}}$ are bidirectional edges added between mesh nodes that are connected in the mesh. Crucially, mesh edges are added to $\mathcal{E}^{\mathrm{M}}$ for all levels of refinement, i.e., for the finest mesh, M^6, as well as for M^5, M^4, M^3, M^2, M^1 and M^0. This is straightforward because of how the refinement process works: the nodes of M^{r-1} are always a subset of the nodes in M^r. Therefore, nodes introduced at lower refinement levels serve as hubs for longer range communication, independent of the maximum level of refinement. The resulting graph that contains the joint set of edges from all of the levels of refinement is what we refer to as the "multi-mesh". See Figure 1e,g for a depiction of all individual meshes in the refinement hierarchy, as well as the full multi-mesh.

For each edge $e^{\mathrm{M}}_{v^{\mathrm{M}}_{\mathrm{s}} \to v^{\mathrm{M}}_{\mathrm{r}}}$ connecting a sender mesh node $v^{\mathrm{M}}_{\mathrm{s}}$ to a receiver mesh node $v^{\mathrm{M}}_{\mathrm{r}}$, we build edge features $\mathbf{e}^{\mathrm{M,features}}_{v^{\mathrm{M}}_{\mathrm{s}} \to v^{\mathrm{M}}_{\mathrm{r}}}$ using the position on the unit sphere of the mesh nodes. This includes the length of the edge, and the vector difference between the 3d positions of the sender node and the receiver node computed in a local coordinate system of the receiver. The local coordinate system of the receiver is computed by applying a rotation that changes the azimuthal angle until that receiver node lies at longitude 0, followed by a rotation that changes the polar angle until the receiver also lies at latitude 0. This results in a total of 327,660 mesh edges (See Appendix Table A.2), each with 4 input features.

Grid2Mesh edges

Grid2Mesh edges $\mathcal{E}^{\mathrm{G2M}}$ are unidirectional edges that connect sender grid nodes to receiver mesh nodes. An edge $e^{\mathrm{G2M}}_{v^{\mathrm{G}}_{\mathrm{s}} \to v^{\mathrm{M}}_{\mathrm{r}}}$ is added if the distance between the mesh node and the grid node is smaller or equal than 0.6 times[17] the length of the edges in mesh M^6 (see Figure 1) which ensures every grid node is connected to at least one mesh node. Features $\mathbf{e}^{\mathrm{G2M,features}}_{v^{\mathrm{G}}_{\mathrm{s}} \to v^{\mathrm{M}}_{\mathrm{r}}}$ are built the same way as those for the mesh edges. This results on a total of 1,618,746 Grid2Mesh edges, each with 4 input features.

Mesh2Grid edges $\mathcal{E}^{\mathrm{M2G}}$ are unidirectional edges that connect sender mesh nodes to receiver grid nodes. For each grid point, we find the triangular face in the mesh M^6 that contains it and add three Mesh2Grid edges of the form $e^{\mathrm{M2G}}_{v^{\mathrm{M}}_{\mathrm{s}} \to v^{\mathrm{G}}_{\mathrm{r}}}$, to connect the grid node to the three mesh nodes adjacent to that face (see Figure 1). Features $\mathbf{e}^{\mathrm{M2G,features}}_{v^{\mathrm{M}}_{\mathrm{s}} \to v^{\mathrm{G}}_{\mathrm{r}}}$ are built on the same way as those for the mesh edges. This results on a total of 3,114,720 Mesh2Grid edges (3 mesh nodes connected to each of the 721×1440 latitude-longitude grid points), each with four input features.