Learning Mesh-Based Simulation with Graph Networks

Tobias Pfaff[*], Meire Fortunato[*], Alvaro Sanchez-Gonzalez[*], Peter W. Battaglia
Deepmind, London, UK
{tpfaff,meirefortunato,alvarosg,peterbattaglia}@google.com

Abstract

Mesh-based simulations are central to modeling complex physical systems in many disciplines across science and engineering, as they support powerful numerical integration methods and their resolution can be adapted to strike favorable trade-offs between accuracy and efficiency. Here we introduce MeshGraphNets, a graph neural network-based method for learning simulations, which leverages mesh representations. Our model can be trained to pass messages on a mesh graph and to adapt the mesh discretization during forward simulation. We show that our method can accurately predict the dynamics of a wide range of physical systems, including aerodynamics, structural mechanics, and cloth– and do so efficiently, running 1-2 orders of magnitude faster than the simulation on which it is trained. Our approach broadens the range of problems on which neural network simulators can operate and promises to improve the efficiency of complex, scientific modeling tasks.

1 Introduction

State-of-the art modeling of complex physical systems often employs mesh representations, and mesh-based finite element methods are ubiquitous in structural mechanics [23, 38], aerodynamics [10, 26], electromagnetics [24], geophysics [27, 30], and acoustics [19] and many other domains. Meshes enables optimal use of the resource budget, by allocating greater resolution to regions of the simulation domain where strong gradients are expected or more accuracy is required, such as the tip of an airfoil in an aerodynamics simulation. But despite their advantages, meshes have received relatively little attention in machine learning for physical predictions; most methods focus on grids, owing to the popularity and hardware support for CNN architectures [13].

We introduce MeshGraphNets, a learnable simulation framework which capitalizes on the advantages of adaptive mesh representations. Our method works by encoding the simulation state into a graph, and performing computations in two separate spaces: the mesh-space, spanned by the simulation mesh, and the Euclidean world-space in which the simulation manifold is embedded (see Figure 3a). By passing messages in mesh-space, we can approximate differential operators that underpin the internal dynamics of most physical systems. Message-passing in world-space can estimate external dynamics, not captured by the mesh-space interactions, such as contact and collision. Unstructured irregular meshes support learning dynamics which are independent of resolution, allowing variable resolution and scale at runtime. This allows us learn the dynamics of vastly different physical systems, from cloth simulation over structural mechanics to fluid dynamics directly from data, providing only very general biases such as spatial equivariance. We demonstrate that we can reliably model materials with a rest state such as elastics (which are challenging for mesh-free models [28]), outperform particle- and grid-based baselines, and generalize to more complex dynamics than seen in training.

[*]equal contribution
Videos of all our experiments can be found at https://sites.google.com/view/meshgraphnets

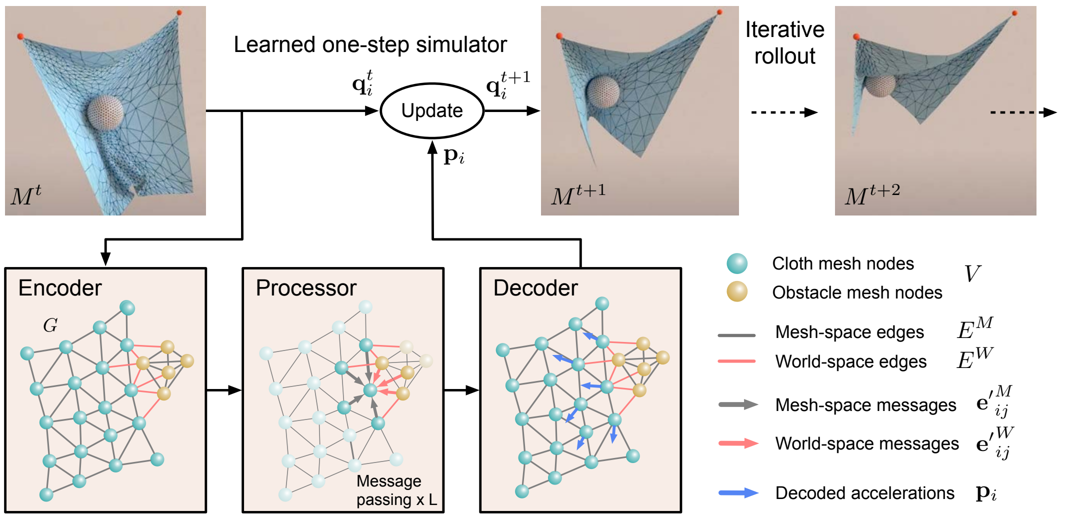


Figure 1: Diagram of MESHGRAPHNETS operating on our SPHEREDYNAMIC domain (video). The model uses an Encode-Process-Decode architecture trained with one-step supervision, and can be applied iteratively to generate long trajectories at inference time. The encoder transforms the input mesh M^t into a graph, adding extra world-space edges. The processor performs several rounds of message passing along mesh edges and world edges, updating all node and edge embeddings. The decoder extracts the acceleration for each node, which is used to update the mesh to produce M^{t+1}.

2 Related Work

Simulations of high-dimensional physical systems are often slow, and there is an increased interest in using machine learning to provide faster predictions, particularly in engineering [1, 5, 14, 11, 39] and graphics [37, 32, 36]. Learned simulations can be useful for real-world predictions where the physical model, parameters or boundary conditions are not fully known [9]. Conversely, the accuracy of predictions can be increased by including specialized knowledge about the system modelled in the form of loss terms [34, 17], or by physics-informed feature normalization [31]. All methods mentioned so far are based on convolutional architectures on regular grids. Although this is by far the most widespread architecture for learning physical prediction models, particle representations on random forests [16] or graph neural network (GNN) architectures [18, 33, 28] have been used with success for modeling liquids and granular materials.

There is increased attention in using meshes for learned geometry and shape processing [7, 22, 12], but despite the widespread use in classical simulators, adaptive mesh representations have not seen much use in learnable prediction model, with a few notable exceptions. Belbute-Peres et al. [4] embed a differentiable aerodynamics solver in a graph convolution (GCN) [15] prediction pipeline for super-resolution in aerodynamics predictions. Our method has similarities, but without a solver in the loop, which potentially makes it easier to use and adapt to new systems. In Section 4 we show that MESHGRAPHNETS are better suited for dynamical prediction than GCN-based architectures. Finally, Graph Element Networks [2] uses meshes over 2D grid domains to more efficiently compute predictions and scene representations. Notably they use small planar systems (< 50 nodes), while we show how to scale mesh-based predictions to complex 3D systems with thousands of nodes.

3 Learning the dynamics model

We describe the state of the system at time t using a simulation mesh $M^t = (V, E^M)$ with nodes V connected by mesh edges E^M. Each node $i \in V$ is associated with a reference mesh-space coordinate $\mathbf{u}_i$ which spans the simulation mesh, and additional dynamical quantities $\mathbf{q}_i$ that we want to model. *Eulerian* systems (Figure 2c,d) model the evolution of continuous fields such as velocity over a fixed mesh, and $\mathbf{q}_i$ sample these fields at the mesh nodes. In *Lagrangian* systems, the mesh represents a moving and deforming surface or volume (e.g. Figure 2a,b), and contains an extra

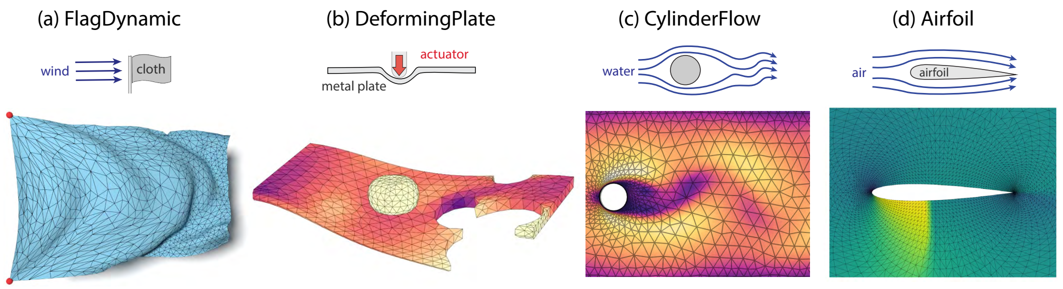


Figure 2: Our model can predict dynamics of vastly different physical systems, from structural mechanics over cloth to fluid dynamics. We demonstrate this by simulating (a) a flag waving in the wind, (b) a deforming plate, (c) flow of water around a cylinder obstacle, and (d) the dynamics of air around the cross-section of an aircraft wing (videos). The color map shows the von-Mises stress in (b), and the x-component of the velocity field in (c),(d).

world-space coordinate $\mathbf{x}_i$ describing the dynamic state of the mesh in 3D space, in addition to the fixed mesh-space coordinate $\mathbf{u}_i$ (Figure 3a).

The task is to learn a forward model of the dynamic quantities of the mesh at time $t{+}1$ given the current mesh M^t and (optionally) a history of previous meshes $\{M^{t-1}, ..., M^{t-h}\}$. We propose MeshGraphNets, a graph neural network model with an Encode-Process-Decode architecture [3, 28], followed by an integrator. Figure 1 shows a visual scheme of the architecture. As an extension of this method, we can also learn to adapt the mesh resolution during model rollouts (adaptive remeshing). This extension is described in Section A.3.

Encoder The encoder encodes the current mesh M^t into a multigraph $G = (V, E^M, E^W)$. We assign mesh nodes to graph nodes V, and mesh edges to mesh-space edges E^M in the graph. This subgraph serves to compute the internal dynamics of the mesh. For Lagrangian systems, we add world edges E^W to the graph, to enable learning external dynamics such as (self-) collision and contact, which are non-local in mesh-space.[2] World-space edges are created by spatial proximity: that is, given a fixed-radius r_W on the order of the smallest mesh edge lengths, we add a world edge between nodes i and j if $|\mathbf{x}_i - \mathbf{x}_j| < r_W$, excluding node pairs already connected in the mesh. This encourages using world edges to pass information between nodes that are spatially close, but distant in mesh space (Figure 3a).

To achieve spatial equivariance, positional features are provided as relative edge features. We encode the relative displacement vector in mesh space $\mathbf{u}_{ij} = \mathbf{u}_i - \mathbf{u}_j$ and its norm $|\mathbf{u}_{ij}|$ into the mesh edges $\mathbf{e}^M_{ij} \in E^M$. Then, we encode the relative world-space displacement vector $\mathbf{x}_{ij}$ and its norm $|\mathbf{x}_{ij}|$ into both mesh edges $\mathbf{e}^M_{ij} \in E^M$ and world edges $\mathbf{e}^W_{ij} \in E^W$. All remaining dynamical features $\mathbf{q}_i$, as well as a one-hot vector indicating node type, are provided as node features in $\mathbf{v}_i$. The encoder uses MLPs $\epsilon^M, \epsilon^W, \epsilon^V$ to encode the respective concatenated features into mesh edge $\mathbf{e}^M_{ij}$, world edge $\mathbf{e}^W_{ij}$, and node $\mathbf{v}_i$ embeddings. See section A.1 for more details on input encoding.

Processor The processor consists of L identical message passing blocks, which generalize Graph-Net blocks [29] to multiple edge sets. Each block P_i contains a separate set of network parameters, and is applied in sequence to the output of the previous block, updating the mesh edge $\mathbf{e}^M_{ij}$, world edge $\mathbf{e}^W_{ij}$, and node $\mathbf{v}_i(l)$ embeddings to $\mathbf{e'}^M_{ij}$, $\mathbf{e'}^W_{ij}$, $\mathbf{v'}_i$ respectively by

$$\mathbf{e'}^M_{ij} \leftarrow f^M(\mathbf{e}^M_{ij}, \mathbf{v}_i, \mathbf{v}_j), \quad \mathbf{e'}^W_{ij} \leftarrow f^W(\mathbf{e}^W_{ij}, \mathbf{v}_i, \mathbf{v}_j), \quad \mathbf{v'}_i \leftarrow f^V(\mathbf{v}_i, \sum_j \mathbf{e'}^M_{ij}, \sum_j \mathbf{e'}^W_{ij}) \tag{1}$$

where f^M, f^W, f^V are implemented using MLPs with a residual connection.

[2]From here on, any mention of world edges and world coordinates applies only to Lagrangian systems; they are omitted for Eulerian systems.

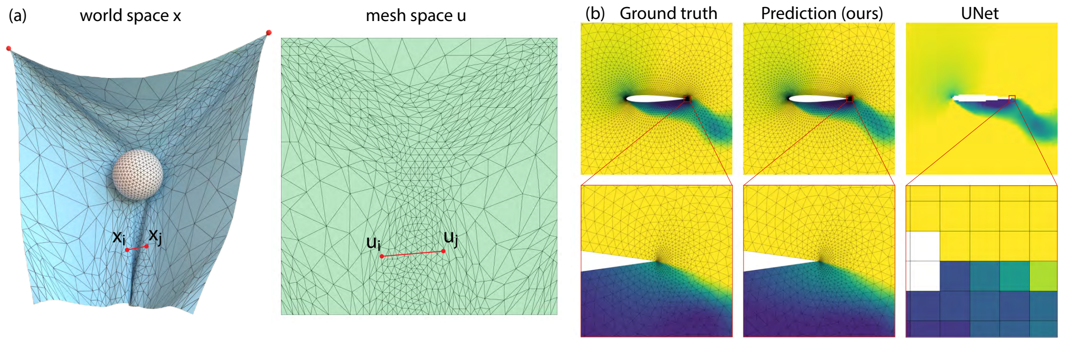


Figure 3: (a) Simulation of a cloth interacting with a sphere. In red, we highlight two nodes which are close in world-space but far in mesh-space, between which a world edge may be created. (b) Rollout of our model versus ground truth on dataset AIRFOIL. Adaptive meshing allows us to accurately predict dynamics at large and small scales. The grid-based U-Net baseline is capable of making good predictions at large scales, but it cannot resolve the smaller scales, despite using four times more cells than our model (video).

Decoder and State Updater For predicting the time $t+1$ state from the time t input, the decoder uses an MLP δ^V to transform the latent node features $\mathbf{v}_i$ after the final processing step into one or more output features $\mathbf{p}_i$.

Each output feature is then processed by a forward-Euler integrator with $\Delta t = 1$ to update the dynamical quantity $\mathbf{q}_i^t \rightarrow \mathbf{q}_i^{t+1}$. For first-order systems the output $\mathbf{p}_i$ is integrated once to update $\mathbf{q}_i^{t+1} = \mathbf{p}_i + \mathbf{q}_i^t$, while for second-order integration happens twice: $\mathbf{q}_i^{t+1} = \mathbf{p}_i + 2\mathbf{q}_i^t - \mathbf{q}^{t-1}$. Additional output features $\mathbf{p}_i$ are also used to make direct predictions of auxiliary quantities such as pressure or stress. For domain-specific details on decoding, see Section A.1. Finally, the output mesh nodes V are updated using $\mathbf{q}_i^{t+1}$ to produce M^{t+1}.

Domains We evaluated our method on a variety of systems with different underlying PDEs (Figure 2). FLAGDYNAMIC and SPHEREDYNAMIC show a cloth, represented as an adaptive triangular mesh, interacting with wind and an obstacle, respectively. DEFORMINGPLATE is a tetrahedral, quasi-static simulation of a plate being bent by an actuator. We also simulated incompressible flow around an obstacle CYLINDERFLOW, as well as the compressible aerodynamics around an airfoil wing AIRFOIL. The simulation meshes vary from regular to highly irregular: the edge lengths of dataset AIRFOIL range between $2 \cdot 10^{-4}$m to 3.5m, and we also simulate meshes which dynamically change resolution over the course of a trajectory. For the Lagrangian systems (cloth and DEFORMINGPLATE) we model the world-space position as a dynamical quantity, while for the Eulerian fluid dynamics domains we model the momentum field and, for AIRFOIL, the density field. Full details on the datasets and the model's input and target features for each domain can be found in Section A.1.

Model training We trained our dynamics model by supervising on the per-node output features $\mathbf{p}_i$ produced by the decoder using a L_2 loss between $\mathbf{p}_i$ and the corresponding ground truth values $\bar{\mathbf{p}}_i$. Additional training details are provided in Section A.2.

4 Results

We tested our MESHGRAPHNETS model on our four experimental domains, and compared it to three different baseline models. Our main findings are that MESHGRAPHNETS are able to produce high-quality rollouts on all domains, outperforming particle- and grid-based baselines, while being significantly faster than the ground truth simulator, and generalizing to much larger and more complex settings at test time. Videos of rollouts, as well as comparisons, can be found at https://sites.google.com/view/meshgraphnets. Even though our model was trained on next-step predictions, model rollouts remain stable over hundreds of time steps, and visually the dynamics remain plausible and faithful to the ground truth. Table 1 shows 1-step prediction and rollout errors

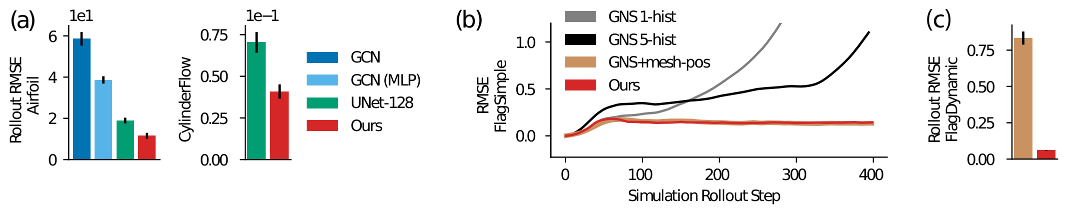


Figure 4: Our model outperforms GCN and CNN-based baselines (a). (b) GNS diverges on cloth datasets (b); providing mesh-space positions (GNS+mesh-pos) helps, but still fails on dynamic meshes (c).

in all of our datasets, while qualitative and quantitative comparisons are provided in Figure 3 and Figure 4.

Computational efficiency Our approach is consistently faster than ground truth solvers by one to two orders of magnitude on all domains (see Table 1 for inference and rollout times). We believe this is due to both being able to take much larger timesteps than classical solvers, and neural networks' natural propensity for parallelization and running on fast accelerators. While highly-parallel solvers exist for specific problems, complex domains or interfacing (e.g. cloth collision resolution) often constrain parallelism. In practice, general-purpose engineering simulators often do not parallelize well, and GPU-accelerated versions are rarely available. Our model's strong efficiency advantage means it may be applicable in situations where computing costs are otherwise prohibitive.

Generalization Our MESHGRAPHNETS model generalizes well outside of the training distribution, with respect to underlying system parameters, mesh shapes, and mesh size. This is because the architectural choice of using relative encoding on graphs has shown to be very conducive to generalization [28]. Also, by forcing the network to make predictions on very irregularly-shaped and dynamically changing meshes, we encourage learning resolution-independent physics.

In AIRFOIL, we evaluate the model on steeper angles ($-35°...35°$ vs $-25°...25°$ in training) and higher inflow speeds (Mach number 0.7...0.9 vs 0.2...0.7 in training). In both cases, the behavior remains plausible (video) and RMSE raises only slightly from 11.5 at training to 12.4 for steeper angles and 13.1 for higher inflow speeds. We also trained a model on a FLAGDYNAMIC variant with wind speed and directions varying between trajectories, but constant within each trajectory. At inference time, we can then vary wind speed and direction freely (video). This shows that the local physical laws our models learns can extrapolate to untrained parameter ranges.

We also trained a model in the FLAGDYNAMIC domain containing only simple rectangular cloth, and tested its performance on three disconnected fish-shaped flags (video). The learned dynamics model generalized to the new shape, and the predicted dynamics were visually similar to the ground truth sequence. In a more extreme version of this experiment, we test that same model on a windsock with tassels (video). Not only has the model never seen a non-flat starting state during training, but the dimensions are also much larger — the mesh averages at 20k nodes, an order of magnitude more than

Dataset	**# nodes** (avg.)	**# steps**	$\mathbf{t_{model}}$ ms/step	$\mathbf{t_{full}}$ ms/step	$\mathbf{t_{GT}}$ ms/step	**RMSE 1-step** $\times 10^{-3}$	**RMSE rollout-50** $\times 10^{-3}$	**RMSE rollout-all** $\times 10^{-3}$
FLAGSIMPLE	1579	400	18.8	19.4	4165.9	1.08 ± 0.02	92.6 ± 5.0	139.0 ± 2.7
FLAGDYNAMIC	2767	250	43.2	836.5	26199.1	1.57 ± 0.02	72.4 ± 4.3	151.1 ± 5.3
SPHEREDYNAMIC	1373	500	31.5	139.5	1610.4	0.292 ± 0.005	11.5 ± 0.9	28.3 ± 2.6
DEFORMINGPLATE	1271	400	24.3	32.5	2892.9	0.25 ± 0.05	1.8 ± 0.5	15.1 ± 4.0
CYLINDERFLOW	1885	600	20.7	23.2	819.6	2.34 ± 0.12	6.3 ± 0.7	40.88 ± 7.2
AIRFOIL	5233	600	37.1	38.1	11014.7	314 ± 36	582 ± 37	11529 ± 1203

Table 1: Left: Inference timings of our model per step on a single GPU, for pure neural network inference ($\mathbf{t_{model}}$) and including remeshing and graph recomputation ($\mathbf{t_{full}}$). Our model has a significantly lower running cost compared to the ground truth simulation ($\mathbf{t_{GT}}$). Right: Errors of our methods for a single prediction step (1-step), 50-step rollouts, and rollout of the whole trajectory.

seen in training. This result shows the strength of learning resolution and scale-independent models: we do not necessarily need to train on costly high-resolution simulation data; we may be able to learn to simulate large systems that would be too slow on conventional simulators, by training on smaller examples and scaling up at inference time.

Comparison to mesh-free GNS model We compared our method to the particle-based method GNS [28] on the fixed-mesh dataset FLAGSIMPLE to study the importance of mesh-space embedding and message-passing. As in GNS, the encoder builds a graph with fixed radius connectivity (10-20 neighbors per node), and relative world-space position embedded as edge features. As GNS lacks the notion of cloth's resting state, error accumulates dramatically and the simulation becomes unstable, with slight improvements if providing 5 steps of history (Figure 4b).

We also explored a hybrid method (GNS+mesh-pos) which adds a mesh-space relative position feature $\mathbf{u}_{ij}$ to the GNS edges. This yields rollout errors on par with our method (flattening after 50 steps due to decoherence in both cases), however, it tends to develop artifacts such as entangled triangles, which indicate a lack of reliable understanding of the mesh surface (video). On irregularly spaced meshes (FLAGSIMPLE), GNS+mesh-pos was not able to produce stable rollouts at all. A fixed connectivity radius will always oversample high-res regions, and undersample low-res regions of the mesh, leading to instabilities and high rollout errors (Figure 4c). We conclude that both having access to mesh-space positions as well as passing messages along the mesh edges are crucial for making predictions on irregularly spaced meshes.

Comparison to GCNs We implemented the GCN architecture from Belbute-Peres et al. [4] (without the super-resolution component) originally designed for a simpler aerodynamic steady-state prediction task (see Section A.4.2). On the much richer AIRFOIL task, however, GCN was unable to obtain stable rollouts. This is not simply a question of capacity; we created a hybrid (GCN-MLP) with our model (linear layers replaced by 2-hidden-layer MLPs + LayerNorm; 15 GCN blocks instead of 6), but the rollout quality was still poor (Figure 4a, video). We believe the key reason GCN performs worse is the lack of relative positional encoding, which makes the GCN less likely to learn local physical laws and more prone to overfitting.

Comparison to grid-based methods (CNNs) Arguably the most popular methods for predicting physical systems are grid-based convolutional architectures. It is fundamentally hard to simulate Lagrangian deforming meshes with such methods, but we can compare to grid-based methods on the Eulerian 2D domains CYLINDERFLOW and AIRFOIL, by interpolating the ROI onto a 128×128 grid. We implemented the UNet architecture from Thürey et al. [31], and found that on both datasets, MESHGRAPHNETS outperforms the UNet in terms of RMSE (Figure 4a). While the UNet was able to make reasonable predictions on larger scales on AIRFOIL, it undersampled the important wake region around the wingtip (Figure 3a), even while using four times more cells to span a region 16 times smaller than our method (Figure A.2). We observe similar behavior around the obstacle in CYLINDERFLOW. Additionally, as seen in the video, the UNet tends to develop fluctuations during rollout. This indicates that predictions over meshes presents advantages even in flat 2D domains.

5 Conclusion

MESHGRAPHNETS are a general-purpose mesh-based method which can accurately and efficiently model a wide range of physical systems, generalizes well, and can be scaled up at inference time. Our method may allow more efficient simulations than traditional simulators, and because it is differentiable, it may be useful for design optimization or optimal control tasks. Variants tailored to specific physical domains, with physics-based auxiliary loss terms, or energy-conserving integration schemes have the potential to increase the performance further. This work represents an important step forward in learnable simulation, and offers key advantages for modeling complex systems in science and engineering.

Acknowledgments

We would like to thank Danilo Rezende, Jonathan Godwin, Charlie Nash, Oriol Vinyals, Matt Hoffman, Kimberly Stachenfeld and Jessica Hamrick for valuable discussions and feedback on the work and manuscript.

References

[1] MS Albergo, G Kanwar, and PE Shanahan. Flow-based generative models for markov chain monte carlo in lattice field theory. *Physical Review D*, 100(3):034515, 2019.

[2] Ferran Alet, Adarsh Keshav Jeewajee, Maria Bauza Villalonga, Alberto Rodriguez, Tomas Lozano-Perez, and Leslie Kaelbling. Graph element networks: adaptive, structured computation and memory. In *International Conference on Machine Learning*, pages 212–222, 2019.

[3] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[4] Filipe de Avila Belbute-Peres, Thomas D. Economon, and J. Zico Kolter. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *Proceedings of the 37th International Conference on Machine Learning ICML 2020*, 2020.

[5] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[6] Frank J Bossen and Paul S Heckbert. A pliant method for anisotropic mesh generation. In *5th Intl. Meshing Roundtable*, pages 63–74. Citeseer, 1996.

[7] Michael M. Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *CoRR*, abs/1611.08097, 2016.

[8] Comsol. Comsol multiphysics® v. 5.4. http://comsol.com, 2020.

[9] Emmanuel de Bezenac, Arthur Pajot, and Patrick Gallinari. Deep learning for physical processes: Incorporating prior scientific knowledge. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124009, 2019.

[10] Thomas D Economon, Francisco Palacios, Sean R Copeland, Trent W Lukaczyk, and Juan J Alonso. Su2: An open-source suite for multiphysics simulation and design. *Aiaa Journal*, 54(3):828–846, 2016.

[11] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

[12] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019.

[13] Sara Hooker. The hardware lottery. *arXiv preprint arXiv:2009.06489*, 2020.

[14] Gurtej Kanwar, Michael S. Albergo, Denis Boyda, Kyle Cranmer, Daniel C. Hackett, Sébastien Racanière, Danilo Jimenez Rezende, and Phiala E. Shanahan. Equivariant flow-based sampling for lattice gauge theory. *Phys. Rev. Lett.*, 125:121601, Sep 2020.

[15] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[16] L'ubor Ladickỳ, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. Data-driven fluid simulations using regression forests. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[17] Sangseung Lee and Donghyun You. Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *Journal of Fluid Mechanics*, 879:217–254, 2019.

[18] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B. Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *International Conference on Learning Representations*, 2019.

[19] Steffen Marburg and Bodo Nolte. *Computational acoustics of noise propagation in fluids: finite and boundary element methods*, volume 578. Springer, 2008.

[20] Rahul Narain, Tobias Pfaff, and James F. O'Brien. Folding and crumpling adaptive sheets. *ACM Trans. Graph.*, 32(4), 2013.

[21] Rahul Narain, Armin Samii, and James F. O'Brien. Adaptive anisotropic remeshing for cloth simulation. *ACM Trans. Graph.*, 31(6), 2012.

[22] Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning*, 2020.

[23] SK Panthi, N Ramakrishnan, KK Pathak, and JS Chouhan. An analysis of springback in sheet metal bending using finite element method (fem). *Journal of Materials Processing Technology*, 186(1-3):120–124, 2007.

[24] D Pardo, L Demkowicz, C Torres-Verdin, and M Paszynski. A self-adaptive goal-oriented hp-finite element method with electromagnetic applications. part ii: Electrodynamics. *Computer methods in applied mechanics and engineering*, 196(37-40):3585–3597, 2007.

[25] Tobias Pfaff, Rahul Narain, Juan Miguel De Joya, and James F O'Brien. Adaptive tearing and cracking of thin sheets. *ACM Transactions on Graphics (TOG)*, 33(4):1–9, 2014.

[26] Ravi Ramamurti and William Sandberg. Simulation of flow about flapping airfoils using finite element incompressible flow solver. *AIAA journal*, 39(2):253–260, 2001.

[27] Zhengyong Ren and Jingtian Tang. 3d direct current resistivity modeling with unstructured mesh by adaptive finite-element method. *Geophysics*, 75(1):H7–H17, 2010.

[28] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning ICML 2020*, 2020.

[29] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 4467–4476, 2018.

[30] Christoph Schwarzbach, Ralph-Uwe Börner, and Klaus Spitzer. Three-dimensional adaptive higher order finite element simulation for geo-electromagnetics—a marine csem example. *Geophysical Journal International*, 187(1):63–74, 2011.

[31] Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020.

[32] Kiwon Um, Xiangyu Hu, and Nils Thuerey. Liquid splash modeling with neural networks. In *Computer Graphics Forum*, volume 37, pages 171–182. Wiley Online Library, 2018.

[33] Benjamin Ummenhofer, Lukas Prantl, Nils Thürey, and Vladlen Koltun. Lagrangian fluid simulation with continuous convolutions. In *International Conference on Learning Representations*, 2020.

[34] Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *ACM SIGKDD international conference on knowledge discovery and data mining*, 2020.

[35] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New results and new trends in computer science*, pages 359–370. Springer, 1991.

[36] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, pages 71–82. Wiley Online Library, 2019.

[37] You Xie, Erik Franz, Mengyu Chu, and Nils Thuerey. Tempogan: A temporally coherent, volumetric gan for super-resolution fluid flow. *ACM Trans. Graph.*, 37(4), July 2018.

[38] Abdelaziz Yazid, Nabbou Abdelkader, and Hamouine Abdelmadjid. A state-of-the-art review of the x-fem for computational fracture mechanics. *Applied Mathematical Modelling*, 33(12):4269–4282, 2009.

[39] Yao Zhang, Woong Je Sung, and Dimitri N Mavris. Application of convolutional neural network to predict airfoil lift coefficient. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 1903, 2018.

A Appendix

A.1 Dataset details

We evaluated our method on a variety of systems with different underlying PDEs, including cloth, structural mechanics, incompressible and compressible fluids. We used ArcSim [21] for simulating the cloth datasets, SU2 [10] for compressible flows, and COMSOL [8] for incompressible flow and hyperelastic simulations. Each dataset consists of 1000 training, 100 validation and 100 test trajectories, each containing 250-600 time steps.

Our structural mechanics experiments involve a hyper-elastic plate, deformed by a kinematic actuator, simulated with a quasi-static simulator (DEFORMINGPLATE). Both actuator and plate are part of the Lagrangian tetrahedral mesh, and are distinguished by a one-hot vector for the corresponding node type $\mathbf{n}_i$. We encode the node quantities $\mathbf{u}_i, \mathbf{x}_i, \mathbf{n}_i$ in the mesh, and predict the Lagrangian velocity $\dot{\mathbf{x}}_i$, which is integrated once to form the next position $\mathbf{x}_i^{t+1}$. As a second output, the model predicts the von-Mises stress σ_i at each node.

Our cloth experiments involve a flag blowing in the wind (FLAGDYNAMIC) and a piece of cloth interacting with a kinematic sphere (SPHEREDYNAMIC) on an adaptive triangular mesh, which changes resolution at each time step. We encode inputs $\mathbf{u}_i, \mathbf{x}_i, \mathbf{n}_i$ as above, but since this is a fully dynamic second order system, we additionally provide $h = 1$ steps of history, by including the velocity estimate $\dot{\mathbf{x}}_i^t = \mathbf{x}_i^t - \mathbf{x}_i^{t-1}$ as a node feature. The decoder outputs acceleration $\ddot{\mathbf{x}}_i$ which is integrated twice.

Our incompressible fluid experiments use the CYLINDERFLOW dataset, which simulates the flow of water around a cylinder on a fixed 2D Eulerian mesh. The mesh contains the node quantities $\mathbf{u}_i, \mathbf{n}_i, \mathbf{w}_i$, where $\mathbf{w}_i$ is a sample of the momentum field at the mesh nodes. The network predicts change in momentum $\dot{\mathbf{w}}_i$, which is integrated once, and a direct prediction of the pressure field p.

Our compressible fluid experiments use the AIRFOIL dataset, which simulates the aerodynamics around the cross-section of an airfoil wing. We model the evolution of momentum[3] $\mathbf{w}$ and density ρ fields, and hence the 2D Eulerian mesh encodes the quantities $\mathbf{u}_i, \mathbf{n}_i, \mathbf{w}_i, \rho_i$. We treat this as a first order system and predict change in momentum $\dot{\mathbf{w}}_i$ and density $\dot{\rho}_i$, as well as pressure p_i.

Below we list details for all of our datasets. “System" describes the underlying PDE: cloth, hyper-elasticity or compressible and incompressible Navier-Stokes flow. Meshing can be either *regular*, i.e. all edges having similar length, *irregular*, i.e. edge lengths vary strongly in different regions of the mesh or *dynamic*, i.e. change at each step of the simulation trajectory. For Lagrangian systems, the world edge radius r_W is provided.

Dataset	**System**	**Mesh type**	**Meshing**	**# nodes** (avg.)	**# steps**	r_W
FLAGSIMPLE	cloth	triangle (3D)	regular	1579	400	—
FLAGDYNAMIC	cloth	triangle (3D)	dynamic	2767	250	0.05
SPHEREDYNAMIC	cloth	triangle (3D)	dynamic	1373	500	0.05
DEFORMINGPLATE	hyper-el.	tetrahedral (3D)	irregular	1271	400	0.03
CYLINDERFLOW	incompr. NS	triangle (2D)	irregular	1885	600	—
AIRFOIL	compr. NS	triangle (2D)	irregular	5233	600	—

Next, we list input encoding for mesh edges $\mathbf{e}_{ij}^M$, world edges $\mathbf{e}_{ij}^W$ and nodes $\mathbf{v}_i$, as well as the predicted output for each system.

[3]In visualizations, we show velocity, calculated as momentum $\mathbf{w}$ divided by density ρ.

System	**Type**	**inputs** $\mathbf{e}_{ij}^M$	**inputs** $\mathbf{e}_{ij}^W$	**inputs** $\mathbf{v}_i$	**outputs** $\mathbf{p}_i$	**history** h
Cloth	Lagrangian	$\mathbf{u}_{ij}, \lvert\mathbf{u}_{ij}\rvert, \mathbf{x}_{ij}, \lvert\mathbf{x}_{ij}\rvert$	$\mathbf{x}_{ij}, \lvert\mathbf{x}_{ij}\rvert$	$\mathbf{n}_i, (\mathbf{x}_i^t - \mathbf{x}_i^{t-1})$	$\ddot{\mathbf{x}}_i$	1
Hyper-El.	Lagrangian	$\mathbf{u}_{ij}, \lvert\mathbf{u}_{ij}\rvert, \mathbf{x}_{ij}, \lvert\mathbf{x}_{ij}\rvert$	$\mathbf{x}_{ij}, \lvert\mathbf{x}_{ij}\rvert$	$\mathbf{n}_i$	$\dot{\mathbf{x}}_i, \sigma_i$	0
Incomp. NS	Eulerian	$\mathbf{u}_{ij}, \lvert\mathbf{u}_{ij}\rvert$	—	$\mathbf{n}_i, \mathbf{w}_i$	$\dot{\mathbf{w}}_i, p_i$	0
Compr. NS	Eulerian	$\mathbf{u}_{ij}, \lvert\mathbf{u}_{ij}\rvert$	—	$\mathbf{n}_i, \mathbf{w}_i, \rho_i$	$\dot{\mathbf{w}}_i, \dot{\rho}_i, p_i$	0

All second-derivative output quantities ($\ddot{\square}$) are integrated twice, while first derivative outputs ($\dot{\square}$) are integrated once as described in Section 3; all other outputs are direct predictions, and are not integrated. The one-hot node type vector $\mathbf{n}_i$ allows the model to distinguish between normal and kinematic nodes. Normal nodes are simulated, while kinematic either remain fixed in space (such as the two nodes which keep the cloth from falling), or follow scripted motion (as the actuator in DEFORMINGPLATE). For scripted kinematic nodes, we additionally provide the *next-step* world-space velocity $\mathbf{x}_i^{t+1} - \mathbf{x}_i^t$ as input; this allows the model to predict next-step positions which are consistent with the movement of the actuator. In the variant of FLAGDYNAMIC with varying wind speeds (generalization experiment in Section 4), the wind speed vector is appended too the node features.

In the dynamically meshed datasets (FLAGDYNAMIC, SPHEREDYANMIC), the mesh changes between steps, and there is no 1:1 correspondence between nodes. In this case, we interpolate dynamical quantities from previous meshes $M^{t-1}, ..., M^{t-h}$ as well as M^{t+1} into the current mesh M^t using barycentric interpolation in mesh-space, in order to provide history and targets for each node.

A.2 Additional model details

A.2.1 Architecture and training

The MLPs of the Encoder ϵ^M, ϵ^W, ϵ^V, the Processor f^M, f^W, f^V, and Decoder δ^V are ReLU-activated two-hidden-layer MLPs with layer and output size of 128, except for δ^V whose output size matches the prediction $\mathbf{p}_i$. All MLPs outputs except δ^V are normalized by a LayerNorm. All input and target features are normalized to zero-mean, unit variance, using dataset statistics.

For training, we only supervise on the next step in sequence; to make our model robust to rollouts of hundreds of steps we use training noise (see Section A.2.2). Models are trained on a single v100 GPU with the Adam optimizer for 10M training steps, using an exponential learning rate decay from 10^{-4} to 10^{-6} over 5M steps.

A.2.2 Training noise

We used the same training noise strategy as in GNS [28] to make our model robust to rollouts of hundreds of steps. We add random normal noise of zero mean and fixed variance to the most recent value of the corresponding dynamical variable (Section A.2.3). When choosing how much noise to add, we looked at the one-step model error (usually related to the standard deviation of the targets in the dataset) and scanned the noise magnitude around that value on a logarithmic scale using two values for each factor of 10. For the exact numbers for each dataset, see Table A.2.3.

In the cases where the dataset is modelled as a first-order system (all, except cloth domains); we adjust the targets according to the noise, so that the model decoder produces an output that after integration would have corrected the noise at the inputs. For example, in DEFORMINGPLATE, assume the current position of a node is $x_i^t = 2$, and $\tilde{x}_i^t = 2.1$ after adding noise. If the next position is $x_i^{t+1} = 3$, the target velocity for the decoder $\dot{x}_i = 1$ will be adjusted to $\tilde{\dot{x}}_i = 0.9$, so that after integration, the model output $\tilde{x}_i^{t+1}$ matches the next step x_i^{t+1} effectively correcting for the added noise, i.e. : $\tilde{x}_i^{t+1} = \tilde{x}_i^t + \tilde{\dot{x}}_i = 3 \equiv x_i^{t+1}$.

In the second-order domains (cloth), the model decoder outputs acceleration $\ddot{x}_i$ from the input position x_i^t and velocity $\dot{x}_i^t = x_i^t - x_i^{t-1}$ (as in GNS). As with other systems, we add noise to the position x_i^t, which indirectly results on a noisy derivative $\dot{x}_i^t$ estimate. In this case, due to the strong dependency between position and velocity, it is impossible to adjust the targets to simultaneously correct for noise in both values. For instance, assume $x_i^{t-1} = 1.4, x_i^t = 2, x_i^{t+1} = 3$, which implies $\dot{x}_i^t = 0.6, \dot{x}_i^{t+1} = 1$, and ground truth acceleration $\ddot{x}_i = 0.4$. After adding 0.1 of noise the inputs are $\tilde{x}_i^t = 2.1 \Rightarrow \tilde{\dot{x}}_i^t = 0.7$. At this point, we could use a modified acceleration target of $\tilde{\ddot{x}}_i^P = 0.2$, so that after integration, the next velocity is $\tilde{\dot{x}}_i^{t+1} = \tilde{\dot{x}}_i^t + \tilde{\ddot{x}}^P = 0.9$, and the next position $\tilde{x}_i^{t+1} = \tilde{x}_i^t + \tilde{\dot{x}}_i^{t+1} = 3 \equiv x_i^{t+1}$, effectively correcting for the noise added to the position. However, note that in this case the predicted next step velocity $\tilde{\dot{x}}_i^{t+1} = 0.9$ does not match the ground truth $\dot{x}_i^{t+1} = 1$. Similarly, if we chose a modified target acceleration of $\tilde{\ddot{x}}_i^V = 0.3$, the next step velocity $\tilde{\dot{x}}_i^{t+1} = 1$ would match the ground truth, correcting the noise in velocity, but the same would not be true for the next step position $\tilde{x}_i^{t+1} = 3.1$. Empirically, we treated how to correct the noise for cloth simulation as a hyperparameter $\gamma \in [0, 1]$ which parametrizes a weighted average between the two options: $\tilde{\ddot{x}}_i = \gamma \tilde{\ddot{x}}_i^P + (1 - \gamma)\tilde{\ddot{x}}_i^V$. Best performance was achieved with $\gamma = 0.1$.

Finally, when the model takes more than one step of history ($h > 1$) (e.g. in the ablation from Figure 4d on FLAGDYNAMIC), the noise is added in a random walk manner with a per-step variance such as the variance at the last step matches the target variance (in accordance with GNS [28]).

A.2.3 Hyper-parameters

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

Table 2: Training noise parameters and batch size.

A.3 Adaptive remeshing

For many problems, the optimal discretization will change over the course of a simultion run. Instead of operating on a fixed mesh for the whole trajectory, we can adaptively change the mesh during the simulation, to make sure important regions remain well-sampled.

Adaptive remeshing algorithms generally consist of two parts: identifying which regions of the simulation domain need coarse or fine resolution, and adapting the nodes and their connections to this target resolution. Only the first part requires domain knowledge of the type of physical system, which usually comes in the form of heuristics. For instance, in cloth simulation, one common heuristic is the refinement of areas with high curvature to ensure smooth bending dynamics, while in computational fluid dynamics, it is common to refine around wall boundaries where high gradients of the velocity field are expected.

In this work we adopt the *sizing field* methodology [21]. The sizing field tensor $\mathbf{S}(\mathbf{u}) \in \mathbb{R}^{2\times 2}$ specifies the desired local resolution by encoding the maximally allowed oriented, edge lengths in the simulation mesh. An edge $\mathbf{u}_{ij}$ is valid iff $\mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_i \, \mathbf{u}_{ij} \leq 1$, otherwise it is too long, and needs to be split[4]. Given the sizing field, a generic local remeshing algorithm can simply split all invalid edges to refine the mesh, and collapse as many edges as possible, without creating new invalid edges, to coarsen the mesh. We denote this remeshing process as $M' = \mathcal{R}(M, \mathbf{S})$.

A.3.1 Learned remeshing

To leverage the advantages in efficiency and accuracy of dynamic remeshing, we need to be able to adapt the mesh at test time. Since remeshing requires domain knowledge, we would however need to call the specific remesher used to generate the training data at each step during the model rollout, reducing the benefits of learning the model. Instead, we learn a model of the sizing field (the only domain-specific part of remeshing) using the same architecture as in Section 3 and train a decoder

[4]This formulation allows different maximal edge lengths depending on the direction. For e.g. a mesh bend around a cylinder, it allows to specify shorter edge lengths in the bent dimension than along the cylinder.

output $\mathbf{p}_i$ to produce a sizing tensor for each node. At test time, for each time step we predict both the next simulation state and the sizing field, and use a generic, domain-independent remesher $\mathcal{R}$ to compute the adapted next-step mesh as $M^{t+1} = \mathcal{R}(\hat{M}^{t+1}, \hat{\mathbf{S}}^{t+1})$. In Section A.3.3 we describe the simple generic remesher that we use for this purpose.

A.3.2 Model training

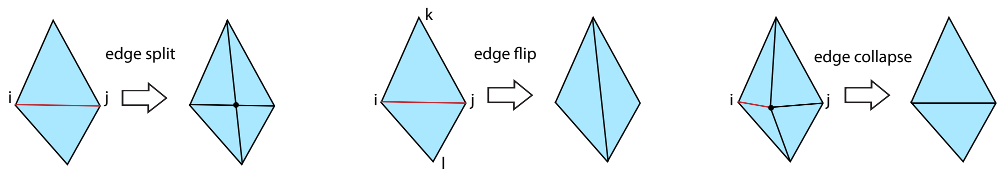
The sizing field model is trained with an L_2 loss on the ground truth sizing field. If no sizing field is available to train the sizing model, we can estimate it from a sequence of meshes. That is, for two consecutive meshes M^t, M^{t+1} we want to find the sizing field $\mathbf{S}$ that would have induced this transition with a local remesher, i.e. $M^{t+1} = \mathcal{R}(M(t), \mathbf{S})$. To do this, we assume that the remesher is near-optimal, that is, all resulting edges are valid, yet maximum-length under the metric $\mathbf{S}$. For each $\mathbf{S}_i$ associated with the node i, this can be expressed as:

$$\mathbf{S}_i = \operatorname{argmax} \sum_{j \in \mathcal{N}_i} \mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_i \, \mathbf{u}_{ij} \,, \quad s.t. \forall j \in \mathcal{N}_i : \mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_i \mathbf{u}_{ij} \leq 1 \qquad (2)$$

This problem corresponds to finding the minimum-area, zero-centred ellipse containing the points $\mathbf{u}_{ij}$, and can be solved efficiently using the MINIDISK algorithm [35].

A.3.3 A domain-invariant local remesher

A local remesher [21, 20, 25] changes the mesh by iteratively applying one of three fundamental operations: *splitting* an edge to refine the mesh, *collapsing* an edge to coarsen it, and *flipping* an edge to change orientation and to preserve a sensible aspect ratio of its elements. Edge splits create a new node whose attributes (position, etc.), as well as the associated sizing tensor, are obtained by averaging values of the two nodes forming the split edge. Collapsing removes a node from the mesh, while edge flips leave nodes unaffected.



Given the sizing field tensor $\mathbf{S}_i$ at each node i, we can define the following conditions for performing edge operations:

- An edge connecting node i and j should be *split* if it is invalid, i.e. $\mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_{ij} \mathbf{u}_{ij} > 1$ with the averaged sizing tensor $\mathbf{S}_{ij} = \frac{1}{2}(\mathbf{S}_i + \mathbf{S}_j)$.
- An edge should be *collapsed*, if the collapsing operation does not create any new invalid edges.
- An edge should be *flipped* if the an-isotropic Delaunay criterion [6]

$$(\mathbf{u}_{jk} \times \mathbf{u}_{ik}) \mathbf{u}_{il}^T \mathbf{S}_A \mathbf{u}_{jl} < \mathbf{u}_{jk}^T \mathbf{S}_A \mathbf{u}_{ik} (\mathbf{u}_{il} \times \mathbf{u}_{jl}) \,, \qquad \mathbf{S}_A = \frac{1}{4}(\mathbf{S}_i + \mathbf{S}_j + \mathbf{S}_k + \mathbf{S}_l)$$

  is satisfied. This optimizes the directional aspect ratio of the mesh elements.

We can now implement a simple local remesher by applying these operations in sequence. First, we split all possible mesh edges to refine the mesh (in descending order of the metric $\mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_{ij} \mathbf{u}_{ij}$), then flip all edges which should be flipped. Next, we collapse all edges we can collapse (in ascending order of the metric $\mathbf{u}_{ij}^{\mathrm{T}} \mathbf{S}_{ij} \mathbf{u}_{ij}$) to coarsen the mesh as much as possible, and finally again flip all possible edges to improve mesh quality.

A.4 Additional results

A.4.1 Ablations

Learned remeshing We trained both a dynamics and a sizing field model to perform learned dynamic remeshing during rollout on FLAGDYNAMIC and SPHEREDYNAMIC. We compare learned

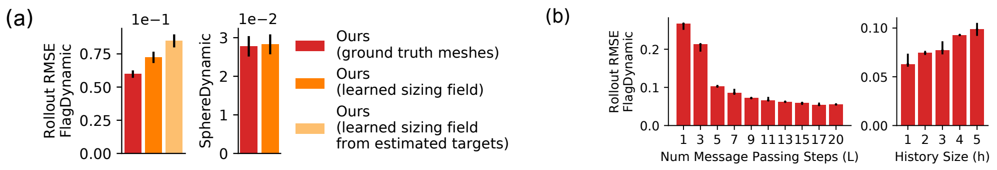


Figure A.1: (a) Remeshing with learned or estimated sizing fields produces accurate rollouts. (b) Taking sufficient message passing steps is crucial for good performance, and limiting history size increases accuracy by preventing overfitting.

remeshing variants with sizing model learned from labeled sizing data, as in Section A.3.1, as well as from estimated targets, as in Section A.3.2. As a baseline, we ran our forward model on the ground truth mesh sequence. As observed in the video, all learned remeshing variants are able to shift the resolution to the new folds as they appear in the cloth, yield equally plausible dynamics, and are on par[5] in terms of quantitative performance (Figure A.1a). Thus, our learned remeshing method provides the advantages of adaptive remeshing without requiring a domain-specific remesher in the loop.

Key hyperparameters We tested several architecture variants and found our method is not very sensitive to many choices, such as latent vector width, number of MLP layers and their sizes. Nonetheless we identified two key parameters which influence performance (Figure A.1b). Increasing the number of graph net blocks (message passing steps) generally improves performance, but it incurs a higher computational cost. We found that a value of 15 provides a good efficiency/accuracy trade-off for all the systems considered. Second, the model performs best given the shortest possible history (h=1 to estimate $\dot{\mathbf{x}}$ in cloth experiments, h=0 otherwise), with any extra history leading to overfitting. This differs from GNS [28], which used $h \in 2...5$ for best performance.

A.4.2 Baseline details

Baseline architectures were trained within our general training framework, sharing the same normalization, noise and state-update strategies. We optimized the training hyper-parameters separately in each case.

GCN baseline We re-implemented the base GCN method (without the super-resolution component) from Belbute-Peres et al. [4]. To replicate the results, and ensure correctness of our implementation of the baseline, we created a dataset AIRFOILSTEADY which matches the dataset studied in their work. It uses the same solver and a similar setup as our dataset AIRFOIL, except that it has a narrower range of angle of attack ($-10°...10°$ vs $-25°...25°$ in AIRFOIL). The biggest difference is that the prediction task studied in their paper is not a dynamical simulation as our experiments, but a steady-state prediction task. That is, instead of unrolling a dynamics model for hundreds of time steps, this task consists of directly predicting the final steady-state momentum, density and pressure fields, given only two scalars (Mach number m, angle of attack α) as well as the target mesh positions $\mathbf{u}_i$— essentially learning a parametrized distribution.

In AIRFOILSTEADY, the GCN predictions are visually indistinguishable to the ground truth, and qualitatively match the results reported in Belbute-Peres et al. [4] for their "interpolation regime" experiments. We also trained our model in AIRFOILSTEADY, as a one-step direct prediction model (without an integrator), with encoding like in AIRFOIL (see Section A.1), but where each node is conditioned on the global Mach number m and angle of attack α), instead of density and momentum. Again, results are visually indistinguishable from the ground truth (video), and our model outperforms GCN in terms of RMSE (ours 0.116 vs GCN 0.159). This is remarkable, as our models' spatial equivariance bias works against this task of directly predicting a global field. This speaks of the flexibility of our architecture, and indicates that it can be used for tasks beyond learning local physical laws for which it was designed.

[5]Note that the comparison to ground truth requires interpolating to the ground truth mesh, incurring a small interpolation penalty for learned remeshing models.

Grid (CNN) baseline We re-implemented the UNet architecture of Thurey et al. [31] to exactly match their open-sourced version of the code. We used a batch size of 10. The noise parameters from Section A.2.3 are absolute noise scale on momentum 6e-2 for CylinderFlow, and 1e1 on momentum and 1.5e-2 on density in the Airfoil dataset.

A.4.3 Error metrics

Rollout RMSE is calculated as the root mean squared error of the position in the Lagrangian systems and of the momentum in the Eulerian systems, taking the mean for all spatial coordinates, all mesh nodes, all steps in each trajectory, and all 100 trajectories in the test dataset. The error bounds in Table 1 and the error bars in Figure 4(a-c) indicate standard error of the RMSE across 100 trajectories. Error bars in Figure 4(d) correspond to min/median/max performance across 3 seeds.

In FlagSimple and FlagDynamic, we observed decoherence after the first 50 steps (Figure 4b), due to the chaotic nature of cloth simulation. Since the dynamics of these domains are stationary, we use the rollout error in the first 50 steps of the trajectory for the comparison shown in the bar plots, as a more discerning metric for result quality. However, the reported trends also hold when measured over the whole trajectory.

In Airfoil, we compute the RMSE in a region of interest around the wing (Figure A.2 middle), which corresponds to the region shown in figures and videos. For comparisons with grid-based methods, we map the predictions on the grid to the ground truth mesh to compute the error.

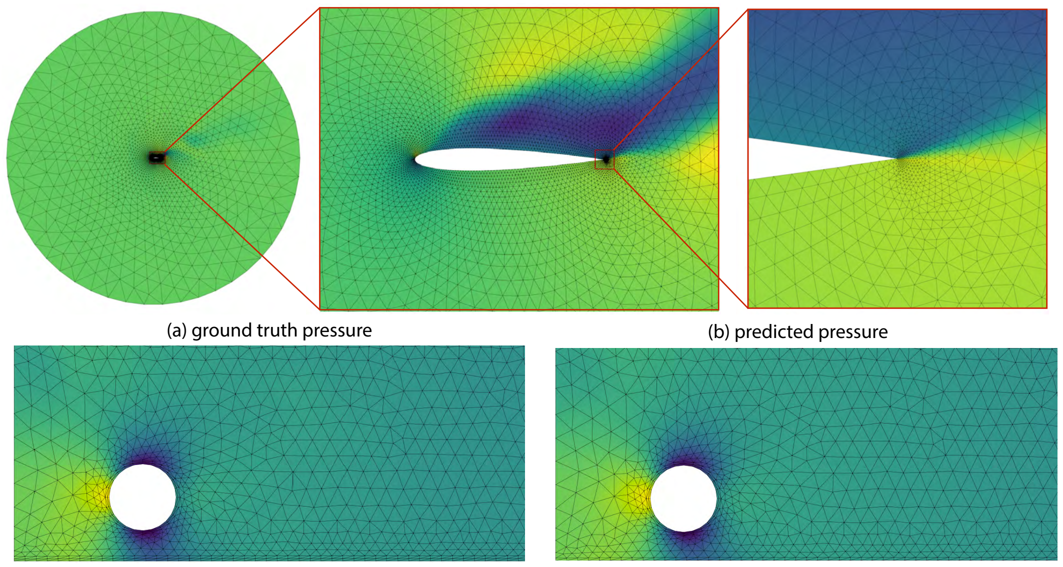


Figure A.2: Top: Many of our datasets have highly irregular meshing, which allows us to predict dynamics at several scales. With only 5k nodes, the dataset Airfoil spans a large region around the wing (left: entire simulation domain), while still providing high resolution around the airfoil (middle: ROI for visual comparison and RMSE computation), down to sub-millimeter details around the wing tip (right). Bottom: Beside output quantities such as position or momentum, which are integrated and fed back into the model as an input during rollout, we can also predict auxiliary output quantities, such as pressure or stress. These quantities can be useful for further analyzing the dynamics of the system. Here, we show a snapshot of auxiliary predictions of the pressure field in CylinderFlow.