SCIENTIFIC COMPUTING ALGORITHMS TO LEARN ENHANCED SCALABLE SURROGATES FOR MESH PHYSICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data-driven modeling approaches can produce fast surrogates to study large-scale physics problems. Among them, graph neural networks (GNNs) that operate on mesh-based data are desirable because they possess inductive biases that promote physical faithfulness, but hardware limitations have precluded their application to large computational domains. Our work shows that it is *possible* to train a class of physically-faithful GNN surrogates on 3D meshes. We show that MeshGraphNets (MGN), a subclass of GNNs for mesh-based physics modeling, can be scaled via our domain decomposition approach to facilitate training that is mathematically equivalent to training on the whole domain. With this, we were able to train MGN on meshes with *millions* of nodes to generate computational fluid dynamics (CFD) simulations. Furthermore, we show how to enhance MGN learning via higher-order numerical integration schemes, which can significantly reduce MGN's error. We validated our methods using a $\sim$3M-node dataset of 3D CO_2 capture CFD simulations (to be released as an open-source benchmark dataset). This work presents a practical path to scaling MGN for real-world applications.

1 INTRODUCTION

Understanding physical systems and engineering processes often requires extensive numerical simulations of their underlying models. However, these simulations are typically computationally expensive to generate, which can hinder their applicability to large-scale problems and downstream tasks such as design optimization. In such cases, surrogate models can be used to emulate the original models and produce more computationally efficient simulations.

Surrogate approaches can suggest efficiency-fidelity tradeoffs (Willard et al., 2020; Wang, 2021). For example, while physically faithful and generalizable to unseen domains, mesh-based GNN simulations offer smaller speedups (11x-290x) (Pfaff et al., 2020) than those offered by less generalizable CNN approaches (125x-716x) (Kim et al., 2019). Moreover, hardware limitations and memory requirements of mesh-based GNNs prevent their application to large graphs. To the best of our knowledge, mesh-based GNNs have not been applied to graphs with more than 20K nodes (Fortunato et al., 2022).

Despite this apparent tradeoff, many practical applications of surrogates require *both* generalization to unseen domains and efficient processing on large datasets. Thus, we investigate how to improve the applicability of GNNs without harming the inductive biases critical to their accuracy, leveraging and building on scientific computing algorithms in the process. Focusing on MeshGraphNets (MGN) surrogates (Pfaff et al., 2020), we enhance MGN learning via domain decomposition and higher-order numerical integration – methods that fall into a framework we call **Scientific Computing Algorithms to Learn Enhanced Scalable Surrogates (SCALES2)**. Our contributions are the following:

1. We train GNN surrogates on CFD meshes with millions of nodes, roughly 100 times larger than the largest meshes used in prior work (Pfaff et al., 2020; Fortunato et al., 2022).
2. We present a domain decomposition method for training on large meshes, with conditions for its mathematical equivalence to training on the large mesh directly.
3. We show that MGN accuracy and efficiency can be greatly improved by using higher-order numerical integration schemes that can augment surrogate faithfulness.
4. We release C3FD, a large-scale dataset of CO_2 capture CFD simulations on 3D meshes containing 3.1 million nodes, to facilitate benchmarking of graph-based surrogates.

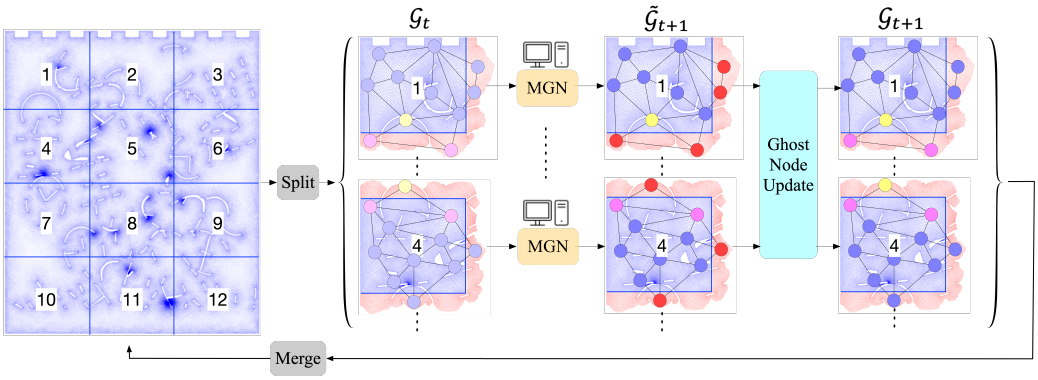

Figure 1: **Scaling MGN via domain decomposition.** We partition graph $\mathcal{G}_t$ into P subdomains (squares), augment the subdomains with ghost nodes to get overlapping patches, then run MGN on the patches $\mathcal{G}_t^{(p)}$ to obtain $\tilde{\mathcal{G}}_{t+1}^{(p)}$. (E.g., Patch1 and Patch4 have ghost nodes from each other, shown as pink and yellow nodes.) For inference, nodes with incorrect states (red) – due to "incomplete" message-passing from a subset of their neighbors – are updated by communicating across patches to obtain $\mathcal{G}_{t+1}^{(p)}$. When training, ghost node updates are not needed as we predict only the next timestep.

2 Approach

2.1 MeshGraphNets

MGN uses message-passing (MP) GNNs to learn from mesh-based simulations. In each MP step, MGN updates each edge's features by aggregating the features of its (two) incident nodes, then updates each node's features by aggregating the features of its incident edges. After k iterations, each node and edge will be influenced by its k-hop neighborhood. The final features of each node are then used to predict the node's next state. The original MGN uses a forward Euler (FE) update rule: $y_{t+1} = y_t + \text{MGN}(y_t)$, where y_t is the system's estimated state at time t.

2.2 Domain decomposition for scaling

We develop a domain decomposition framework that allows mesh-based GNN surrogates to be trained on large domains (see Figure 1). Domain decomposition is useful when application of a surrogate to a domain produces out-of-memory errors on the hardware used for inference or learning. Broadly, it divides a simulation problem defined on a domain into a jointly equivalent (or similar) set of problems, each of which is defined on its own subdomain (Tang et al., 2021).

For our datasets, we create evenly-sized subdomains by dividing each dimension of the domain into equal parts. Inspired by overlapping domain decomposition approaches (Tang et al., 2021), we augment each subdomain with its k-hop neighborhood, which we call the ghost zone with width k. We call the union of the subdomain and its ghost zone a patch. Because each subdomain node can access its k-hop neighborhood in its patch, an MGN with m message-passing (MP) steps, where $k \geq m$, will update each subdomain node as if the entire domain had been available. Thus, training on patches is equivalent to training on the full domain, under three conditions: (1) $k \geq m$; (2) we only compute the loss on the subdomain nodes (excluding the ghost nodes); and (3) we accumulate the gradients over multiple batches until all patches in the domain have been visited before updating the weights. In practice, we violate the third condition by updating the surrogate's weights after a random subset of a domain's patches are seen – we call the resulting procedure "patch training".

2.3 Higher-order integration for enhanced learning

Denoting the state of a system at time t as $y(t)$ and y_t as its numerical estimate, the quantity y_t is evolved by MGN using forward Euler (FE): $y_{t+1} = y_t + hf(y_t)$. FE is a first-order (global error) method that introduces a second-order local truncation error by ignoring quadratic and higher order terms in the following Taylor series: $y(t+1) = y(t) + hf(y(t)) + \mathcal{O}(h^2)$.

Thus, MGN's prediction error $|y(t+1) - y_{t+1}|$ can be decomposed into an error in the time derivative $f(y)$ (approximated by MGN(y_t)) and a $\mathcal{O}(h^2)$ local truncation error. A surrogate trained with FE must learn to improve its prediction of $f(y)$ while simultaneously correcting for the $\mathcal{O}(h^2)$ error. As there is no guarantee that the surrogate will learn to offset the $\mathcal{O}(h^2)$ errors, we explore the use of higher-order integrators – Heun's second- and third-order methods (Heun et al., 1900; J.C.Butcher, 1996) – which result in higher-order (smaller) truncation errors. When the truncation error correction is difficult to learn from the data (in a physically faithful manner), these higher-order integrators should reduce the truncation error from $O(h^2)$ to $O(h^3)$ (or even $O(h^4)$), freeing the surrogate (e.g., MGN) to more faithfully learn $f(y)$. See Appendix A for further discussion.

3 DATASETS

CylinderFlow CylinderFlow contains 2D simulations of inviscid, incompressible flow around a cylinder, generated by numerical solvers with Navier-Stokes equations on an Eulerian mesh (Pfaff et al., 2020). We train surrogates to predict the fluid momenta at each node of the mesh.

C3FD Our CO_2 Capture CFD (C3FD) dataset contains STAR-CCM+ simulations of a CO_2-capturing solvent flowing across packing structures in a column. This dataset contains 50 2D and 16 3D simulations, each of which spans 500 timesteps. The 2D and 3D simulation meshes have 150,073 and 3,129,815 nodes, respectively. Each node contains information about its physical location, liquid volume fraction (a quantity related to capture efficiency), momentum per unit volume, contact angle, and node type. We train surrogates to predict the liquid volume fraction and momenta at each node. Additional dataset details and visualizations are available in Appendix B.

4 EXPERIMENTS

As a sanity check, we first replicate experiments from Pfaff et al. (2020) on the full 2D C3FD domain. In Figures 3 and 5, we show that we can train MGNs to low error and generate 500-timestep rollouts that are similar to the ground truth. Further, we show that MGNs trained on the 2D C3FD domain generalize to a distinct 2D domain (i.e., a new slice of the 3D domain).

These initial experiments on 2D C3FD required roughly 50 GB of memory (we met this requirement via an NVIDIA A100 GPU with 80 GB of memory). Access to such resources is costly, and applying MGN to larger datasets, like the 3D C3FD dataset (requiring roughly 1 TB of accelerator memory), is not possible with commonly available hardware. Thus, we set out to make mesh-based GNN surrogates more efficient, focusing on algorithms to scale and enhance their learning. Ultimately, our patch training allows MGN to scale sufficiently to learn from the 3D C3FD data. We also demonstrate that MGN learning can be enhanced by using higher-order integration.

4.1 DOMAIN DECOMPOSITION IS KEY TO SCALING SURROGATES

In Section 2.2, we outlined patch training. However, it remains unclear how mesh surrogates (like MGN) perform on very large domains. Training on a full domain (through domain decomposition) may be wasteful when the domain is very large, because each patch in the domain must be visited before the weights are updated. We explore if it may be just as effective to use patch training, which updates the weights after visiting a small sample of a domain's patches.

Unlike training on a smaller domain and then generalizing to the large domain at test time (illustrated in Figure 6), we find that patch training induces no performance drop relative to training on the large domain directly. This holds for CylinderFlow (Figure 8) and 2D C3FD (Figure 7). Based on this, we use patch training to efficiently train an MGN on a 3D mesh with millions of nodes (3D C3FD) for the first time (Figure 2). To optimize this model for practical deployment, further work (e.g., hyperparameter tuning) remains. We provide hyperparameter and other experiment details in Appendix C and additional results in Appendix D.

4.2 HIGHER-ORDER INTEGRATION ENHANCES LEARNING

Following discussion from Section 2.3, we want to investigate: How significant are the $\mathcal{O}(h^2)$ truncation errors from FE? If significant, how effective is MGN in learning to correct these errors?

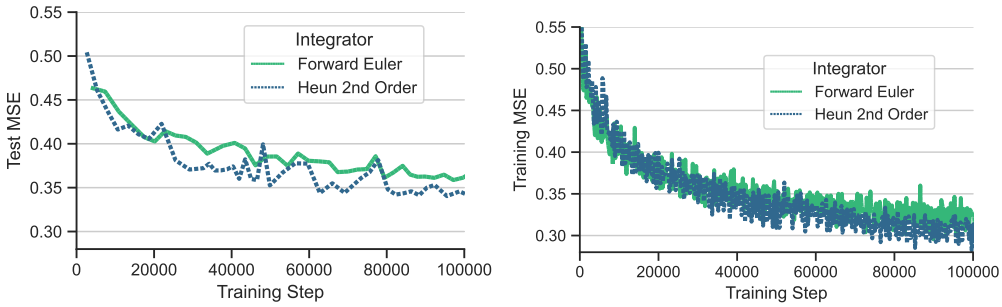

Figure 2: **Patched training on our 3M node dataset**. Experiments run on 3D C3FD simulations with 3,129,815 nodes per timestep.

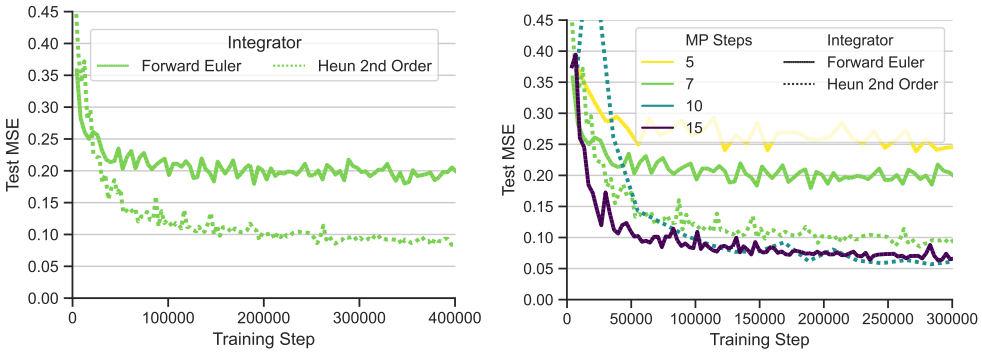

Figure 3: **Higher-order integration enhances learning**. Experiments run on 2D C3FD data with 150,073 nodes per timestep. (*Left*) An MGN using Heun's 2nd-order method (H2) with 7 message-passing (MP) steps is able to halve the error – relative to forward Euler (FE) – on both test (as shown) and training (see Figure 9) data. (*Right*) H2 with 7 MP steps is comparable to FE with 15 MP steps, despite the benefits associated with more MP steps. H2 with 10 MP steps can outperform FE with 15 MP steps. In Figure 10, we show higher-order integration's benefits are stable across training runs.

While it is clear that higher-order integration methods should reduce the truncation errors, these higher-order integrators may not provide benefits if MGN can easily correct for these errors.

Using the 2D C3FD data, we train MGNs using three different integrators: FE, Heun's second-order method (H2), and Heun's third-order method (H3). Comparing FE and H2, we see that test error can be roughly halved with H2 (Figure 3 *left*). Using more MP steps generally improves surrogate performance, so we also compare the integrators across a range of MP steps (Figure 3 *right*). An H2-surrogate using 7 MP steps achieves roughly the same error as an FE-surrogate using 15 steps. Furthermore, H2 with 10 MP steps reaches even lower error than FE with 15 MP steps. This suggests that higher-order integration is an important tool for improving MGN performance, with the added bonus of requiring fewer MP steps to achieve the same level of accuracy. We provide more discussion and results in Appendix D and review related work in Appendix E.

5 Discussion

We introduced scientific computing algorithms (SCALES2) to enhance and scale training of GNN surrogates on mesh physics. Our approach enabled MGN to train on 3D meshes with millions of nodes. We also showed that higher-order integration can halve the error in our 2D experiments. To stimulate further research on scaling and improving surrogate simulation, we release C3FD, a benchmark dataset with CO_2 capture CFD simulations on both 2D and 3D meshes (with 150,073 nodes and 3,129,815 nodes, respectively). While MGN training focuses on next-step predictions, it remains to be seen the overall improvement that SCALES2 may provide to longer rollouts on large domains. We discuss additional limitations and extensions to our work in Appendix F.

REFERENCES

Brian Bartoldson, Rui Wang, Yucheng Fu, David Widemann, Sam Nguyen, Jie Bao, Zhijie Xu, and Brenda Ng. Latent space simulation for carbon capture design optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11):12447–12453, Jun. 2022a.

Brian R Bartoldson, Bhavya Kailkhura, and Davis Blalock. Compute-efficient deep learning: Algorithmic trends and opportunities. *arXiv preprint arXiv:2210.06640*, 2022b.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. *arXiv preprint arXiv:2210.00612*, 2022.

Yucheng Fu, Jie Bao, Rajesh Singh, Chao Wang, and Zhijie Xu. Investigation of countercurrent flow profile and liquid holdup in random packed column with local cfd data. *Chemical Engineering Science*, 221:115693, 2020.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Karl Heun et al. Neue methoden zur approximativen integration der differentialgleichungen einer unabhängigen veränderlichen. *Z. Math. Phys*, 45:23–38, 1900.

Steeven Janny, Aurélien Bénéteau, Madiha Nadri, Julie Digne, Nicolas Thome, and Christian Wolf. Eagle: Large-scale learning of turbulent fluid dynamics with mesh transformers. In *International Conference on Learning Representations*, 2023.

J.C.Butcher. *Applied Numerical Mathematics*, volume 20. Elsevier B.V., 1996.

Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. Improving the accuracy, scalability, and performance of graph neural networks with ROC. In I. Dhillon, D. Papailiopoulos, and V. Sze (eds.), *Proceedings of Machine Learning and Systems*, volume 2, pp. 187–198, 2020. URL https://proceedings.mlsys.org/paper/2020/file/fe9fc289c3ff0af142b6d3bead98a923-Paper.pdf.

Byungsoo Kim, Vinicius C. Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep Fluids: a generative network for parameterized fluid simulations. *Computer Graphics Forum*, 38(2):59–70, 2019.

Marten Lienen and Stephan Günnemann. Learning the dynamics of physical systems from sparse observations with finite element networks. *arXiv preprint arXiv:2203.08852*, 2022.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2020.

Anthony Ralston. Runge-kutta methods with minimum error bounds. *Mathematics of computation*, 16(80):431–437, 1962.

HS Tang, RD Haynes, and G Houzeaux. A review of domain decomposition methods for simulation of fluid flows: Concepts, algorithms, and applications. *Archives of Computational Methods in Engineering*, 28:841–873, 2021.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Rui Wang. Physics-guided deep learning for dynamical systems: A survey. *arXiv preprint arXiv:2107.01272*, 2021.

Jared Willard, Xiaowei Jia, Shaoming Xu, Michael S. Steinbach, and Vipin Kumar. Integrating physics-based modeling with machine learning: A survey. *CoRR*, abs/2003.04919, 2020. URL `https://arxiv.org/abs/2003.04919`.

Ao Zhou, Jianlei Yang, Yeqi Gao, Tong Qiao, Yingjie Qi, Xiaoyi Wang, Yunli Chen, Pengcheng Dai, Weisheng Zhao, and Chunming Hu. Brief industry paper: Optimizing memory efficiency of graph neural networks on edge computing platforms. In *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pp. 445–448. IEEE, 2021.

A HIGHER-ORDER INTEGRATION ANALYSIS

Integrators of sufficiently high order can practically eliminate the local truncation error. At that point, the neural network's error is entirely attributable to its misestimation of the time derivative $f(y)$. Accordingly, we explore the use of higher-order integration as a potential path toward enhanced learning of surrogate simulators. In Section 4.2 and Appendix D.4, we train surrogates that use FE (a first-order method), and Heun's second and third-order methods.

Heun's second-order method is as follows: $y_{t+1} = y_t + \frac{h}{2}(f(y_t) + f(y_t + hf(y_t)))$. Comparing this method to FE (see Equation 1), we find that it requires that the MGN learns $f(y)$ as FE does, but Heun's method incurs lower truncation error. When only 1st-order derivative information is learnable in a physically consistent manner, higher order integrators may make MGNs more physically faithful by reducing the scale of the truncation error relative to the scale of the error in the 1st-order derivative.

$$\begin{aligned} y(t+1) - y(t) &= \underbrace{hf(y(t))}_{\text{FE method update: requires 1st-order derivative}} + \underbrace{\mathcal{O}(h^2)}_{\text{truncation error}} \\ &= \underbrace{\frac{h}{2}\Big(f\big(y(t)\big) + f\big(y(t) + hf(y(t))\big)\Big)}_{\text{Heun's method update: requires 1st-order derivative}} + \underbrace{\mathcal{O}(h^3)}_{\text{truncation error}} . \end{aligned} \tag{1}$$

Therefore, higher-order integration may provide *faster learning* by focusing learning on $f(y)$ (i.e., by providing a better signal-to-noise ratio). Further, higher-order integration may provide *more accurate learning* (lower final error) by explicitly eliminating $\mathcal{O}(h^2)$ truncation error terms that are not removable by FE-based training, which is not guaranteed to learn how to offset the varying $\mathcal{O}(h^2)$ error terms. Whether these corrections can be learned depends on properties of the data and the surrogate (e.g., if the second derivative is learnable then such corrections may be achievable).

B DATASET DETAILS

B.1 C3FD

Figure 4 shows the 3D C3FD domain, which is a representative section of a bench-scale column of a CO_2 capture system. The 2D domain (see Figure 1 for an illustration) has 150,073 polygonal mesh grid points, while the 3D domain has 3,129,815 mesh grid points. There are 16 3D simulations and 50 2D simulations. The different simulations correspond to different velocities for the liquid inlet at the top of the domain. Training and validation splits were created to allow validation to test extrapolation and interpolation. For 2D, validation simulations covered the four slowest and four fastest inlet velocities, in addition to four intermediate velocity simulations. For 3D, validation simulations covered the fastest and slowest inlet velocities, in addition to one simulation with an intermediate velocity.

The C3FD simulations follow the setup described in Fu et al. (2020): Eulerian simulations of the counter-current solvent and gas flow are generated by solving the continuity and momentum equations and using the volume-of-fluid method to model multiphase fluid separation. In particular, we use STAR-CCM+ to solve:

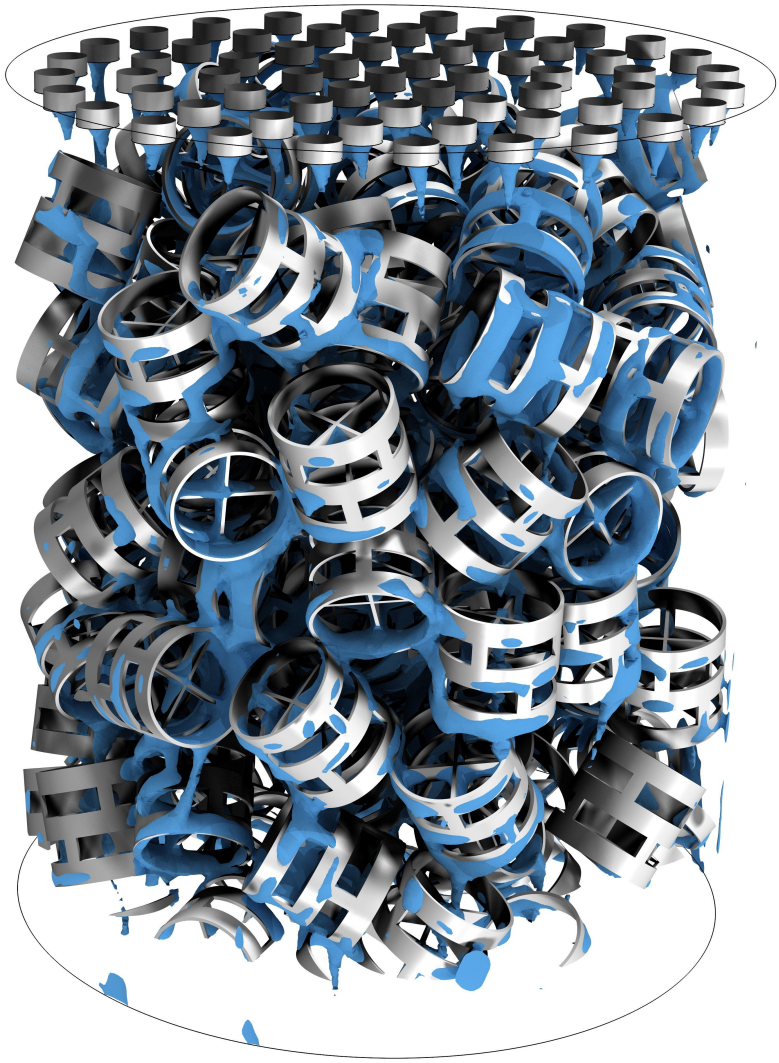

Figure 4: Visualization of the 3D C3FD domain, which is a 100mm wide and 200mm tall cylinder packed with pall rings. Gas enters from below and exits above, while liquid does the opposite.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0, \tag{2}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g} + \mathbf{F}_\sigma, \tag{3}$$

where $\mathbf{u}$ is velocity, ρ is density, p is pressure, $\mathbf{g}$ is gravity, μ is viscosity, and $\mathbf{F}_\sigma$ is the surface tension at the gas-liquid interface. Density and viscosity are calculated via the volume fraction average of the liquid (α) and gas phase (1-α). The following transport equation governs the evolution of α:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{u}\alpha) = 0. \tag{4}$$

The 2D simulation domain is a cross-section of the 3D domain, but it is solved independently. For 2D, simulations are solved with $\Delta t = 0.001s$, and data is output at every tenth timestep. For 3D data,

the simulator uses $\Delta t = 0.0002s$, and data is saved at every fiftieth timestep. The data used for both 2D and 3D surrogates therefore has $\Delta t = 0.01s$.

As a preprocessing step, we use the SciPy's implementation of Delaunay triangulation to create mesh edges (Virtanen et al., 2020). Notably, we noticed some splinter-like cells after this procedure – future work could alleviate this through, e.g., the procedure given in Lienen & Günnemann (2022).

Available quantities in the data are the following: position, velocity (a vector), pressure, momentum per unit volume (a vector), liquid volume fraction, contact angle, and node type. Liquid volume fraction is related to capture efficiency and is thus the most important quantity to predict for this dataset from a CO_2-capture point of view.

C EXPERIMENT DETAILS

C.1 C3FD EXPERIMENT DETAILS

For C3FD, the surrogate inputs are: the momenta, liquid volume fraction, contact angle, and node type. The surrogate must predict changes in the momenta and liquid volume fraction, while the other variables are constants. We observed better prediction performance when excluding pressure as a model input and output. We use the same edge features as Pfaff et al. (2020).

For MGN input noise, we used 0.02 for horizontal momenta, 0.03 for vertical momentum, and 0.05 for liquid volume fraction. These were not heavily optimized values, and preliminary experiments suggest performance benefits from using values larger than these.

As in Pfaff et al. (2020), we trained models with the Adam optimizer and an exponentially decaying learning rate, decaying from 1×10^{-3} to 1×10^{-7} over 4,000,000 steps. We typically stopped training at step 400,000 or earlier, when performance plateaued, as shown in our plots.

When using patch training, we randomly sampled patches from across the training simulations and across the timesteps. For 2D training, we used 12 patches (3 along the 100mm width and 4 along the 200mm height). For 3D training, partitioning created 2,160 patches (12 along each horizontal axis and 15 along the vertical axis), 1,980 of which were non-empty. Each 2D and 3D patch contained on the order of 1×10^4 nodes, on average. Future work may achieve better 3D load balancing by using a more sophisticated decomposition scheme than uniform space partitioning.

On 3D experiments, we used 32 V100 16 GB GPUs, each with a batch size of 1 patch (total batch size of 32). On 2D experiments, we used one A100 80 GB GPU with a batch size of 1 full domain, except for patching experiments where we used 12 V100s, each with a batch size of 1 patch (i.e., with 2D training the updates used a roughly equal number of nodes for patch and non-patch training).

Validation errors are based on 1,000 randomly selected full-domain examples, except on 2D and 3D patch training experiments where we compute errors based on 1,000 patches.

Pfaff et al. (2020) found 15 MP steps provides good performance, and this is a baseline we use except in our 3D experiments (where we use 3 MP steps) and in our higher-order integrator experiments where we use the number indicated in the legend. Consistent with Pfaff et al. (2020), 2D MGNs used 128 dimensional latent states for edges and nodes. 3D MGNs used 150 dimensional latent states.

C.2 CYLINDERFLOW EXPERIMENT DETAILS

The CylinderFlow (Pfaff et al., 2020) experiments were run on 8 V100 16 GB GPUs. We use batch size 8 for each GPU (reaching an effective batch size of 64). Following Pfaff et al. (2020), we use 15 MP steps, 128 dimensional latent states, and node type and momenta as inputs with 0.02 input noise. We use the same edge features as Pfaff et al. (2020). We predict momenta, and we use the same learning rate schedule approach we described for C3FD data, stopping training when loss plateaus. For patch training, we partition the grid into four equal size subdomains.

D SUPPORTING RESULTS

D.1 ROLLOUTS

In Figure 5, we plot rollouts created with the 15 MP MGN that had its training dynamics plotted in Figure 3. After training on the 2D C3FD dataset, we evaluate this model at a test simulation with an intermediate velocity (testing velocity interpolation). Additionally, we evaluate this model on a new 2D domain created by taking a new cross-section of the 3D C3FD domain.

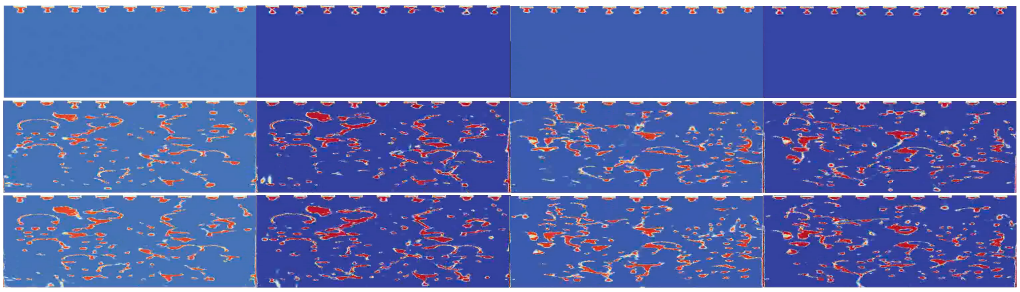

Figure 5: **MGN has stable and visually accurate performance across long time horizon on domains with more than 100,000 nodes, even when generalizing to new domain.** The top, middle, and bottom rows correspond to timesteps 30, 250, and 500. From left to right, the columns represent: the MGN's rollout on the 2D C3FD domain, the ground truth CFD simulation for this domain, the MGN's rollout on a new 2D domain (the column's packings are differently configured), and the ground truth CFD simulation on this latter domain. Liquid volume fraction is the visualized variable.

D.2 TRAINING ON A LARGE DOMAIN IS BETTER THAN GENERALIZING TO IT

To motivate the need for patch training, Figure 6 shows that training on the left third of the 2D C3FD dataset's domain does not facilitate test performance as high as training on the full 2D C3FD dataset. Patch training can address this by making it possible to train on the entire domain, even when it does not fit on an accelerator.

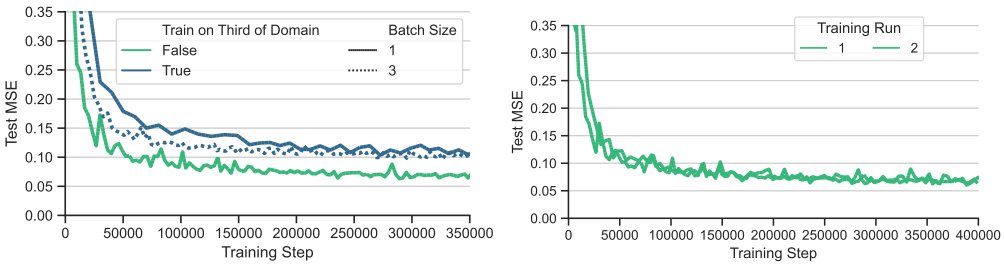

Figure 6: **Generalizing from a smaller mesh is inferior to training on the full domain**. Training only on the left third of our 150,073 node 2D C3FD domain then generalizing at test time leads to worse test performance (left), regardless of whether or not the batch size is adjusted to account for training on a subdomain. The baseline model's training has low variance across runs (right) – the standard deviation of the minimum test RMSEs reached by the two baseline models is 0.003. Relative to this standard deviation, the gap (left) between the minimum test RMSEs of generalizing from a smaller mesh and the baseline approach is large at 0.036.

D.3 PATCH TRAINING IS AN EFFECTIVE WAY TO TRAIN ON A LARGE DOMAIN

We show that our patch training approach that updates weights before the entire domain is seen (based on a random sample of patches from different simulations and timesteps) produces training comparable to updating weights based on the entire domain. This is true for C3FD (Figure 7) and CylinderFlow (Figure 8).

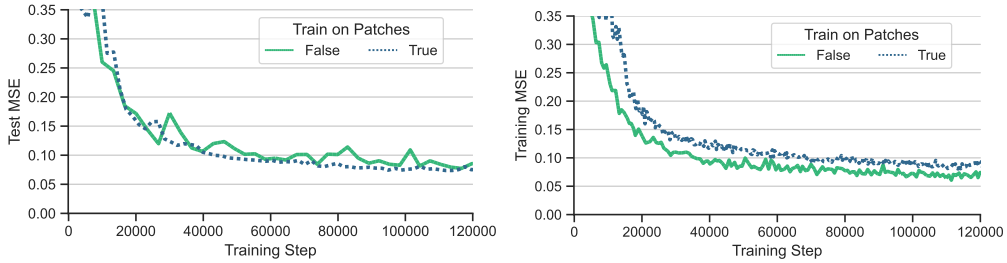

Figure 7: **Patch training on the full-domain can replace regular training on the full-domain**. Training on patches produced by domain decomposition leads to similar test (left) and training (right) performance, even though patch training involves randomly sampling patches from different simulations and timesteps. This illustrates that domain decomposition can facilitate training on domains that would normally cause out-of-memory errors. Experiments run on 2D C3FD simulations with 150,073 nodes per timestep.

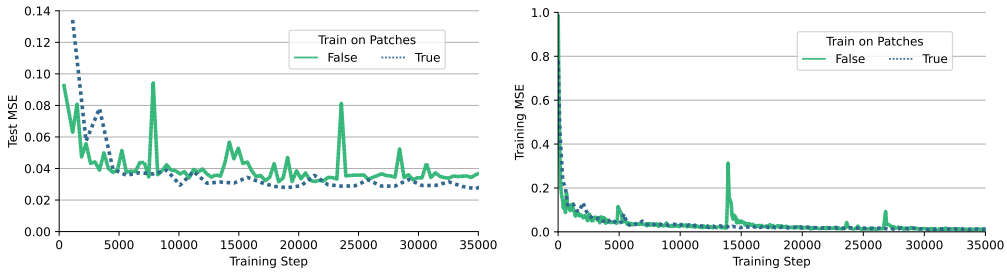

Figure 8: **Patch training is effective on CylinderFlow**. Patch training on the CylinderFlow simulations improves the loss on the test (left) and training (right) datasets.

D.4 Additional results on higher-order integration

We observed that Heun's second order method benefits test errors in Figure 3: Figure 10 shows this benefit is stable across training runs. In Figure 9, we show that this benefit extends to training. Notably, with Heun's third order method, training error decreases even more quickly (see Figure 11). This is consistent with the hypothesis that mesh-based GNNs like MGN have difficulty removing not just $\mathcal{O}(h^2)$ but also $\mathcal{O}(h^3)$ local truncation errors. However, we also observe test error is not best served by a third order integrator, which may suggest that use of higher order integrators would benefit from additional regularization. In other words, irremovable/unlearnable local truncation errors may provide regularization during training, and their removal by higher order integrators may create a need for more regularization for good test performance.

E Related work

Neural network surrogates for similar large-scale CFD simulations have been studied previously (Bartoldson et al., 2022a), but we are unaware of prior use of physically faithful GNN surrogates that generalize across domains in such contexts. In particular, Pfaff et al. (2020) demonstrated that MeshGraphNets (MGN) could learn generalizable rules for updating physics fields from meshes, and MultiScale MGNs attain state-of-the-art performance on many tasks (Fortunato et al., 2022), but their memory inefficiency had prevented the use of any MGN on large, application-relevant graphs.

More broadly, scaling GNNs to large graphs is an important topic that has been approached in a variety of ways including parallelism and off-accelerator memory use (Jia et al., 2020) and more efficient learning algorithms. This prior algorithmic work has improved the efficiency of operating on large graphs by sampling (Hamilton et al., 2017; Chiang et al., 2019) or otherwise modifying the function computed by the baseline, inefficient GNN (Zhou et al., 2021) – such approaches may be critical when average node degree is high. In contrast, we are focused on domains that demand

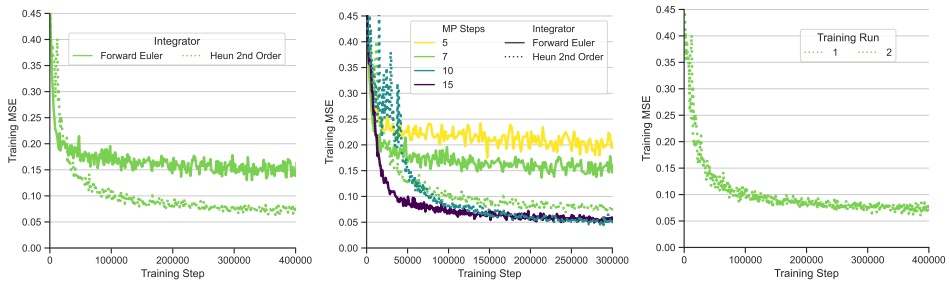

Figure 9: **Higher-order integration enhances learning on training data**. (*Left*) Using Heun's 2nd-order method on an MGN with 7 MP steps halves the error – relative to the baseline forward Euler (FE) integrator – on training data. (*Middle*) Heun's 2nd-order method on a 7 MP step MGN is competitive with FE on a 15 MP step MGN, despite the visible (and previously documented) benefits of more MP steps. (*Right*) We show benefits of higher-order integration are stable across two training runs. Experiments run on 2D CO_2 capture simulations with 150,073 nodes per timestep.

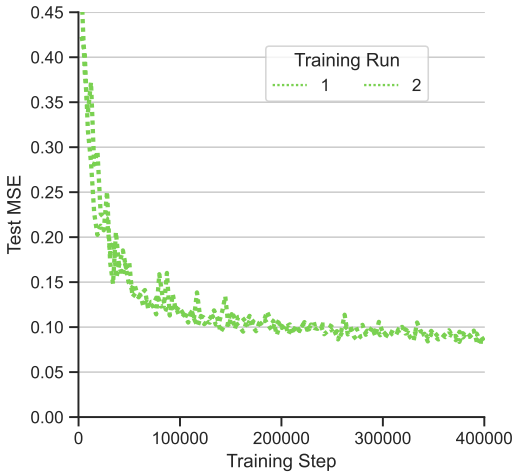

Figure 10: **Higher-order integration's benefit is stable across training runs.**

physical faithfulness, making approximations/proxies less attractive.[1] Accordingly, we take advantage of the manageable node degree associated with physics meshes to introduce a GNN physics surrogate scaling approach that can provide training mathematically equivalent to training on a mesh that would normally cause out-of-memory errors.

Also providing a benchmark dataset for physics surrogate learning, Janny et al. (2023) introduce EAGLE, which has over one thousand 2D simulations. EAGLE has meshes with 3,388 nodes per timestep on average, while our 3D C3FD dataset uses a mesh with 3,129,815 nodes at each timestep ($\approx$ 1000x larger). Further differences include the following: our datasets use the same mesh as the CFD solver while EAGLE downsamples the CFD mesh greatly; we consider a static mesh rather than a dynamic mesh; our CFD simulations are of CO_2 capture columns relevant to understanding capture performance as a function of simulation parameters; and our geometries include a variety of complex obstacles in the form of column packings.

Separately, Janny et al. (2023) show that the number of message-passing iterations can be reduced through use of their GNN surrogate that employs attention and graph pooling. In contrast, we show that the number of message-passing iterations can be reduced by using a higher-order integrator.

[1] If the state of node u affects the state of node v according to physical laws, then a surrogate should seemingly not learn an update rule for v based on a sampling algorithm that may exclude u. Indeed, while compatible with other efficiency methods, we do not integrate sampling-based approaches into our study to avoid harming the inductive biases of the surrogates we consider.

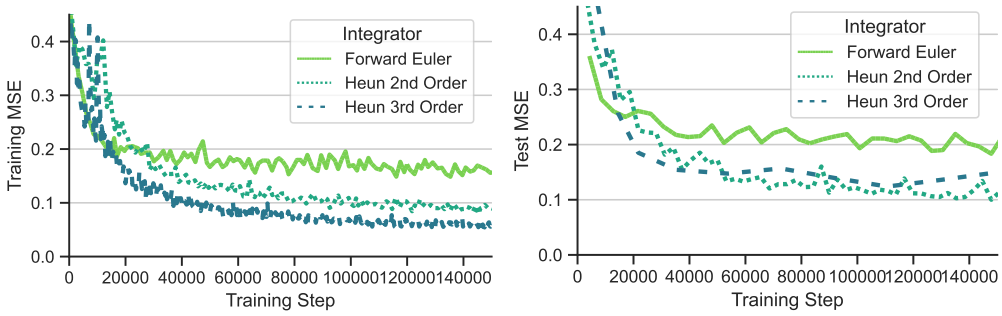

Figure 11: **Third order integrators improve training but not test performance**. All MGN models use 7 MP iterations.

Mesh-based graph surrogates have previously been used to produce time derivative estimates for use in a higher-order integration scheme (Lienen & Günnemann, 2022). However, this prior work has used GNNs that are embedded within a differential equation solving framework (i.e., a finite element method), which grants structure to the rules learned by the GNN that is not available when using the more general mesh surrogate GNNs that are our focus. Further, this prior work uses higher-order integration as a byproduct of inserting a GNN into a differential equation solver, which runs for several timesteps. In contrast, we motivate consideration of higher-order integration through analysis of its effect on the learning task and inductive biases of any network that predicts time derivatives, and we test related hypotheses by comparing higher-order integration to a baseline first-order method.

F Limitations and Future Work

Our work only scratches the surface of opportunities for SCALES2 to improve machine learning of GNN surrogates that operate on meshes. For example, it may be possible to obtain a higher-order integrator's benefits without incurring its memory and compute overhead – the Predict–Evaluate–Correct (PEC) framework may facilitate this at inference time. Also, better performance may be seen with other integrators, such as Ralston's methods, which are second-, third-, and fourth-order methods designed to minimize the local truncation error (Ralston, 1962). Relatedly, our results indicate that training details (like regularization strength) may need to be adjusted to match the integrator's order.

Importantly, given a number of MP steps, a higher-order integrator leads to slower training steps. However, it also reduces the number of training steps required to reach a target accuracy. Thus, while we did not document any speedups associated with higher-order integration, there may be settings where use of fewer steps more than offsets added training time per step, allowing higher-order integrators to provide a speedup. Further, if the accelerator idles during training, then the per-step cost of the higher-order integrator can be hidden, facilitating a speedup (Bartoldson et al., 2022b). Relatedly, in our experiments, we noticed that 3D models trained with Heun's method reached target errors notably faster than FE models. Separately, the memory and compute overhead of higher-order integration may be addressed by lowering the MP steps: indeed, FE and 15 MP steps has roughly the same memory cost and performance as Heun's second-order method and 7 MP steps.

Separately, more sophisticated domain decomposition approaches are likely to provide benefits. Our approach constructs patches, each of which contains a subdomain with a ghost zone border to facilitate mathematical equivalence to directly training on the full domain. However, training on a patch is more compute-intensive than training on just the subdomain. Additionally, on our 3D meshes, average node degree is higher than on 2D meshes. This makes the k-hop neighborhood larger and one condition for mathematically-equivalent training – see condition (1) in Section 2.2 – more difficult to satisfy. We find accelerator out-of-memory errors (on a 16GB GPU) on 3D data can be avoided when using $k \leq 5$ with edges created by Delaunay triangulation. Yet, a larger k-hop neighborhood may be desired, and use of a nearest neighbors algorithm to create edges may help tame node degree in this case. Alternatively, a partitioning approach that balances the node counts of the resulting patches rather than their spatial extents may allow larger k. Addressing these issues may lead to better scaling and performing GNN surrogate simulators of mesh physics.