SCIENTIFIC COMPUTING ALGORITHMS TO LEARN ENHANCED SCALABLE SURROGATES FOR MESH PHYSICS

Brian R. Bartoldson[†], **Yeping Hu**[†,*], **Amar Saini**[†,*], **Jose Cadena**[†,*], **Yucheng Fu**[‡], **Jie Bao**[‡], **Zhijie Xu**[‡], **Brenda Ng**[†], & **Phan Nguyen**[†]
[†]Lawrence Livermore National Laboratory [‡]Pacific Northwest National Laboratory

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. We show that it is *possible* to train a class of GNN surrogates on 3D meshes. We scale MeshGraphNets (MGN), a subclass of GNNs for mesh-based physics modeling, via our domain decomposition approach to facilitate training that is mathematically equivalent to training on the whole domain under certain conditions. 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 via higher-order numerical integration, which can reduce MGN's error and training time. We validated our methods on an accompanying dataset of 3D CO_2-capture CFD simulations on a 3.1M-node mesh. 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. In such cases, surrogates can be used to emulate the original models and produce 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 GNNs offer smaller speedups (11–290x) (Pfaff et al., 2020) than those of less generalizable CNNs (125–716x) (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, they have not been applied to graphs with more than 20K nodes (Fortunato et al., 2022). Despite this tradeoff, many applications of surrogates require *both* generalization to unseen domains and efficient processing on large datasets.

Thus, we seek to improve the applicability of GNNs without harming the inductive biases critical to their accuracy by building on scientific computing algorithms. Focusing on MeshGraphNets (MGN) (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 patch training, a domain decomposition method for training on large meshes, and conditions for its mathematical equivalence to training on a 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 Carbon Capture CFD simulations on 3D meshes containing 3.1M nodes, to facilitate benchmarking of graph-based surrogates.

[*]Equal contribution. Corresponding authors: `bartoldson@llnl.gov`, `nguyen97@llnl.gov`.

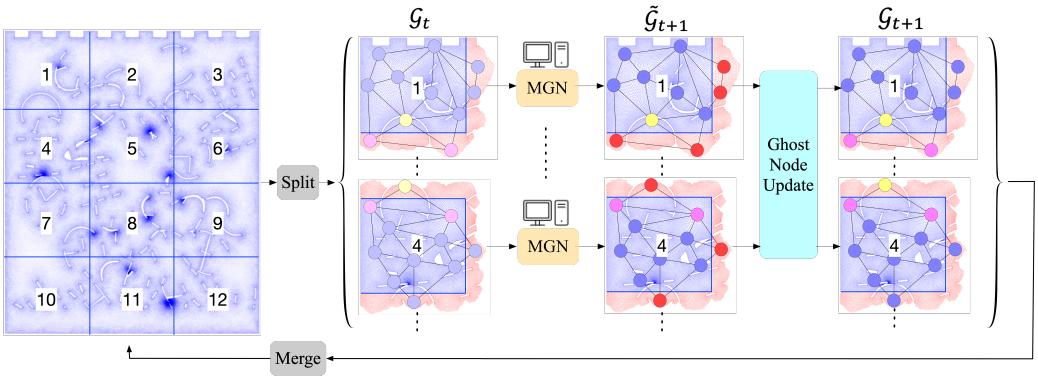

Figure 1: **Scaling MGN via domain decomposition.** We partition graph $\mathcal{G}_t$ into P subdomains (squares), augment the subdomains with ghost nodes to get overlapping patches (e.g., Patch1 and Patch4 have ghost nodes from each other, shown as pink and yellow nodes), then run MGN on the patches $\mathcal{G}_t^{(p)}$ to obtain $\tilde{\mathcal{G}}_{t+1}^{(p)}$. For inference, nodes with incorrect states (red)—due to "incomplete" message-passing with 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 incident nodes, and 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 using the following update rule: $y_{t+h} = y_t + \text{MGN}(y_t)$, where y_t is the system's numerically estimated state at time t and h is the timestep size.

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 MP steps, where $m \leq k$, 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) $m \leq k$; (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 weights after a random subset of a domain's patches is seen—we call the resulting procedure "patch training".

2.3 Higher-order integration for enhanced learning

MGN's update rule is based on the forward Euler (FE) method, which introduces first-order global error and second-order local truncation error by ignoring quadratic and higher-order terms in the following Taylor series: $y(t+h) = y(t) + hf(y(t)) + \mathcal{O}(h^2)$, where $y(t)$ is the true state of a system at time t. Accordingly, MGN's prediction error is $|y(t+h) - y_{t+h}| = \left|[hf(y(t)) - \text{MGN}(y(t))] + \mathcal{O}(h^2)\right|$,

which can be decomposed into a prediction error of the (scaled) time derivative, $[hf(y) - \text{MGN}(y)]$, and a second-order local error, $\mathcal{O}(h^2)$.[1]

While MGN may learn to offset the $\mathcal{O}(h^2)$ term when learning to estimate the time derivative, local error can be reduced by changing how MGN updates y_t to y_{t+h}, increasing the benefit of correctly predicting the time derivative. Specifically, we replace FE with Heun's second- and third-order methods (Heun et al., 1900; J.C.Butcher, 1996), which have smaller local errors ($\mathcal{O}(h^3)$ and $\mathcal{O}(h^4)$, respectively). In Appendix A, we hypothesize that this can speed up and enhance learning.

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 Carbon Capture CFD (C3FD) dataset contains STAR-CCM+ simulations of a CO_2-capturing solvent flowing across packing structures in a column. The dataset contains 50 2D and 50 3D simulations, each spanning 500 timesteps. The 2D and 3D simulation meshes have 150K and 3.1M nodes, respectively. Each node contains information about its physical location, type, liquid volume fraction (a quantity related to capture efficiency), momentum per unit volume, and contact angle. We train surrogates to predict liquid volume fraction and momentum at each node. Additional dataset details and visualizations are available in Appendix B. The C3FD dataset is available at `https://doi.org/10.25584/1963908`.

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 accelerator memory to apply the original MGN. Applying MGN to larger datasets like the 3D C3FD dataset, which requires roughly 1 TB of accelerator memory, is not possible with commonly available hardware. Thus, aiming to make mesh-based GNN surrogates more efficient, we test algorithms to scale and enhance their learning. Ultimately, our patch training successfully scales MGN training to learn from the 3D C3FD data. We also show that MGN training can be enhanced by using higher-order integration.

4.1 PATCH TRAINING 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 with 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 smaller domains and then generalizing to a large test domain (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 MGNs 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 additional experiment details and results in Appendices C and D, respectively.

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?

[1] In practice, this analysis is not exact because MGN error uses CFD data that only approximates $y(t+h)$.

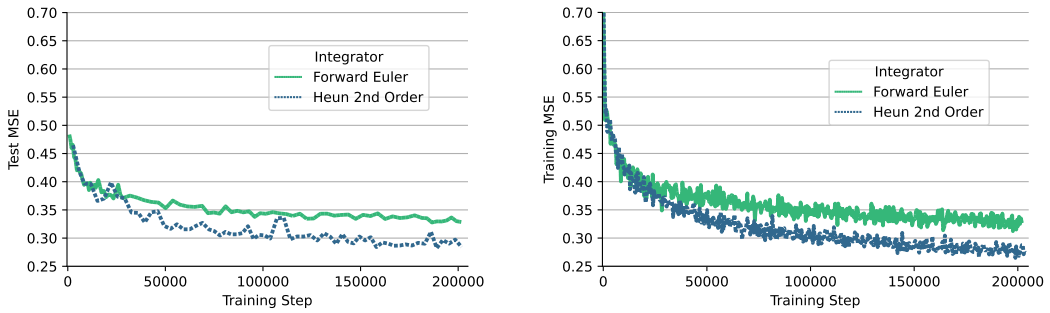

Figure 2: **Patch training on 3D C3FD**. Experiments run on data with 3.1M nodes per timestep.

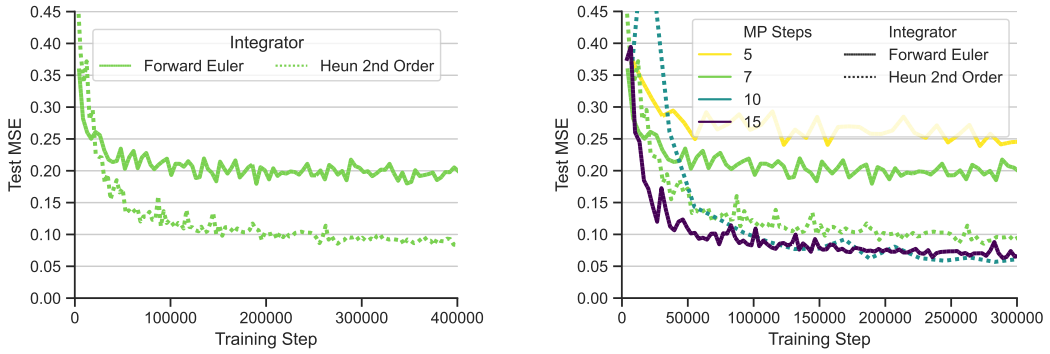

Figure 3: **Higher-order integration enhances learning**. Experiments run on 2D C3FD data with 150K nodes per timestep. (*Left*) An MGN using Heun's second-order method (H2) with 7 message-passing (MP) steps is able to halve the error—relative to forward Euler (FE)—on both test (as shown) and training (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*). The H2-surrogate using 7 MP steps achieves roughly the same error as the 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 additional 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 released C3FD, a benchmark dataset with CO_2 capture CFD simulations on both 2D and 3D meshes (with 150K nodes and 3.1M nodes, respectively). As this study 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.

ACKNOWLEDGMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 using NERSC award BES-ERCAP0024437.

LLNL Release Number (Paper): LLNL-CONF-845156.
PNNL Release Number (Dataset): PNNL-SA-182276.

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. 2022.

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.

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.

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, suggesting that MGN updates based on such integrators can produce extremely high accuracy solely through learning to compute the time derivative $f(y)$. Accordingly, we explore their use as a path toward improving surrogate performance. In Section 4.2 and Appendix D.4, we train surrogates that use forward Euler (FE) (a first-order method), and Heun's second- (H2) and third-order (H3) methods.

The H2 method is as follows: $y_{t+h} = y_t + \frac{h}{2}(f(y_t) + f(y_t + hf(y_t)))$. Comparing this to FE (Equation 1), we find that H2 requires the MGN to also learn $f(y)$ as FE does, but H2 incurs lower truncation error. When only first-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 first-order derivative.

$$\begin{aligned} y(t+h) - y(t) &= \underbrace{hf(y(t))}_{\text{FE update}} + \underbrace{\mathcal{O}(h^2)}_{\text{truncation error}} \\ &= \underbrace{\frac{h}{2}\Big(f\big(y(t)\big) + f\big(y(t) + hf(y(t))\big)\Big)}_{\text{H2 update}} + \underbrace{\mathcal{O}(h^3)}_{\text{truncation error}} . \end{aligned} \tag{1}$$

Using a state update based on H2, MGN's error can again (see Section 2.3) be decomposed into two terms: an error based on MGN's ability to predict the (scaled) time derivative,

$$\left[\frac{h}{2}\Big(f\big(y(t)\big) + f\big(y(t) + hf(y(t))\big)\Big) - \frac{1}{2}\Big(\text{MGN}(y(t)) + \text{MGN}\big(y(t) + \text{MGN}(y(t))\big)\Big)\right], \tag{2}$$

and the $\mathcal{O}(h^3)$ term, a *third-order* local truncation error. Note that while MGN still only needs to learn to predict the scaled time derivative, the benefit of $\text{MGN}(y(t))$ mimicking $hf(y)$ increases with an H2-based updated because the truncation error term is now smaller.

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

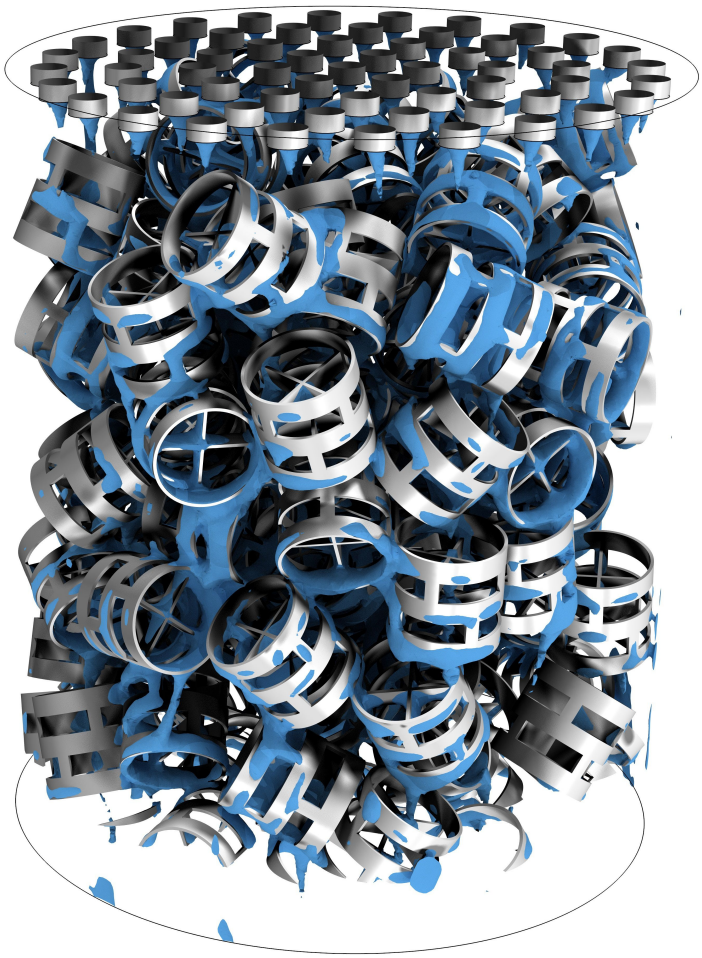

Figure 4: Visualization of the 3D C3FD domain, which is a 100mm wide and 200mm tall cylinder packed with pall rings. Gas enters from below and exits above, while liquid does the opposite.

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 150K polygonal mesh grid points, while the 3D domain has 3.1M points (the exact number varies, as each simulation used one of three unique meshes). There are 50 3D simulations and 50 2D simulations. The different simulations correspond to different velocities for the liquid entering at the inlets at the top of the domain; other variables, such as the geometry of the domain, are held constant across the 50 simulations. Training and validation splits were created to allow validation to test extrapolation and interpolation. Validation simulations covered the four slowest, four fastest, and four intermediate inlet velocity simulations.

C3FD followed the setup described in Fu et al. (2020): Eulerian simulations of the counter-current solvent and gas flow were generated by solving the continuity and momentum equations and using the volume-of-fluid method to model multiphase fluid separation. STAR-CCM+ was used to solve

Node Type	2D Data Boundary Index	3D Data Boundary Index	Contact Angle
fluid	1.79769313486232e+308	1.79769313486232e+308	N/A
liquid_inlet	34	47	N/A
outlet	31	46	N/A
gas_inlet	30	45	N/A
inlet_to_outlet_wall	29	44	180
side_wall	28	43	33.5
packing	27	42	33.5

Table 1: The C3FD data's mapping from boundary index to node type and contact angle.

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \rho \mathbf{u} = 0, \tag{3}$$

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) = -\nabla p + \mu \nabla^2 \mathbf{u} + \rho \mathbf{g} + \mathbf{F}_\sigma, \tag{4}$$

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:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot (\mathbf{u}\alpha) = 0. \tag{5}$$

The 2D simulation domain is a cross-section of the 3D domain, but it was solved on independently. For 2D, the solver used timestep size $h = 0.001s$, and data was output at every tenth timestep. For 3D, the solver used $h = 0.0002s$, and data was saved at every fiftieth timestep. The 2D and 3D data provided in C3FD was therefore saved using $h = 0.01s$.

To create edges, we used SciPy's implementation of Delaunay triangulation (Virtanen et al., 2020). Notably, we noticed some splinter-like cells after this procedure; further mesh refinement or filtering may be applied to resolve this, e.g., the procedure described by Lienen & Günnemann (2022).

The following quantities are provided at each node of the computational domain: position, velocity, pressure, and liquid volume fraction, and boundary index. For velocity, a value is provided for each spatial dimension. Rather than include the momentum vector in the dataset explicitly, we compute it in the following way: $\mathbf{momentum}_i = [\alpha\rho_L + (1-\alpha)\rho_G]\mathbf{u}_i$, where $\rho_L = 1010$ and $\rho_G = 1.18415$ are the liquid and gas densities, respectively. 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 perspective.

Node type and contact angle information are not explicitly provided in the data, but may be derived from boundary index information as indicated in Table 1.

C Experiment Details

C.1 C3FD experiment details

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

For MGN input noise, we used 0.02 for horizontal momentum (x-direction in 2D; x-, y-directions in 3D), 0.03 for vertical momentum (y-direction in 2D; z-direction in 3D), and 0.05 for liquid volume fraction. These values were not heavily optimized, and preliminary experiments suggest performance benefits from using larger values.

Pfaff et al. (2020) found that 15 MP steps provided good performance. We also used 15 MP steps in our study, except in our 3D experiments (3 MP steps) and in our higher-order integrator experiments

(indicated in the plot legends). As with Pfaff et al. (2020), we used 128 latent dimensions to encode nodes and edges for the 2D dataset. We used 150 latent dimensions for the 3D dataset.

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 different simulations and timesteps. For 2D training, we created 12 patches (3 evenly-spaced partitions along the 100mm width, 4 along the 200mm height). For 3D training, we created 2,160 patches (12 evenly-spaced partitions along each horizontal axis, 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 with patch training, we used 32 V100 16 GB GPUs, each with a batch size of 1 patch. On 2D experiments with patch training, we used 12 V100 16 GB GPUs, each with a batch size of 1 patch. Since 2D C3FD's computational domain was small enough to fit on certain GPUs, we also used one A100 80 GB GPU with a batch size of 1 full domain to compare. We note that with 2D C3FD, MGN weight updates used a roughly equal number of nodes for patch and non-patch training.

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

C.2 CylinderFlow experiment details

We used 8 V100 16 GB GPUs to run experiments on the CylinderFlow dataset (Pfaff et al., 2020). We used a batch size of 8 for each GPU (for an effective batch size of 64). For patch training, we also partitioned the grid into four equal-size subdomains. Following Pfaff et al. (2020), we use 15 MP steps, 128 latent dimensions to encode nodes and edges, and node type and momentum as inputs with 0.02 input noise. We also used the same edge features. We predicted momentum and used the same learning rate schedule approach we described for C3FD data, stopping training when loss plateaus.

D Supporting Results

D.1 Rollouts

Figure 5 shows several frames of the rollouts predicted with the MGN that used 15 MP steps and FE updates (discussed in Section 4.2 and Figure 3). After training on 2D C3FD, we evaluated this model on a test simulation with an intermediate velocity (testing velocity interpolation). Additionally, we evaluated it on a new 2D domain created by taking a new cross-section of the 3D C3FD domain.

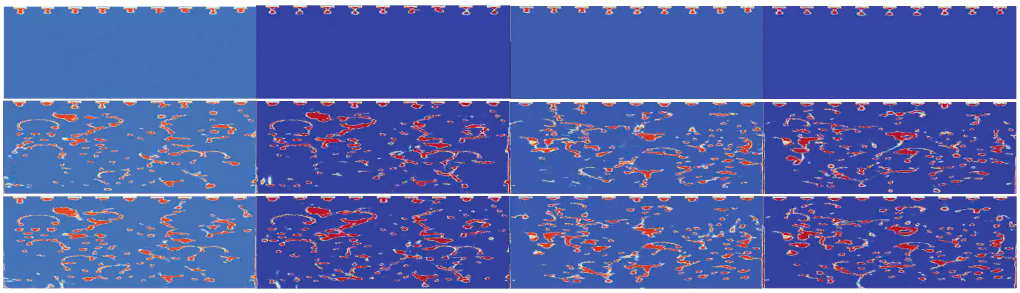

Figure 5: **MGN has stable and visually accurate performance across long time horizons on domains with more than 100K nodes, even when generalizing to a new domain.** The top, middle, and bottom rows correspond to timesteps 30, 250, and 500. From left to right, the columns represent: the MGN's rollout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

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

An alternative to training on a large domain via domain decomposition is simply training on a smaller domain. For instance, Pfaff et al. (2020) showed that MGNs can generalize from training meshes with roughly 2×10^3 nodes to testing meshes with roughly 2×10^4 nodes. However, it is unclear if relying on such generalization is sufficient, or if we need to train on domains with scales and shapes similar to those of the test simulations. Preliminary results on this subject suggest that changing the shape of the domain can cause a small but notable amount of harm to performance (Pfaff et al., 2020).

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

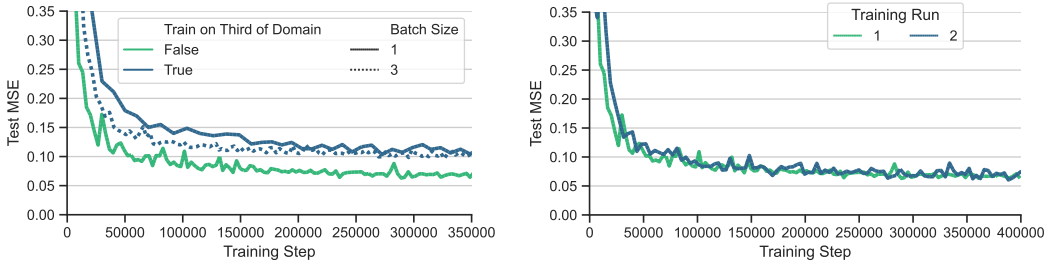

Figure 6: **Generalizing from a smaller mesh is inferior to training on the full domain**. Training only on the left third of our 150K node 2D C3FD domain and then generalizing at test time leads to worse test performance (left), regardless of whether or not the batch size is adjusted to account for 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 patch training, which updates weights using a random sample of patches from different simulations and timesteps, produces error that is comparable to regular training by updating weights using the entire domain. This is true for both C3FD (Figure 7) and CylinderFlow (Figure 8).

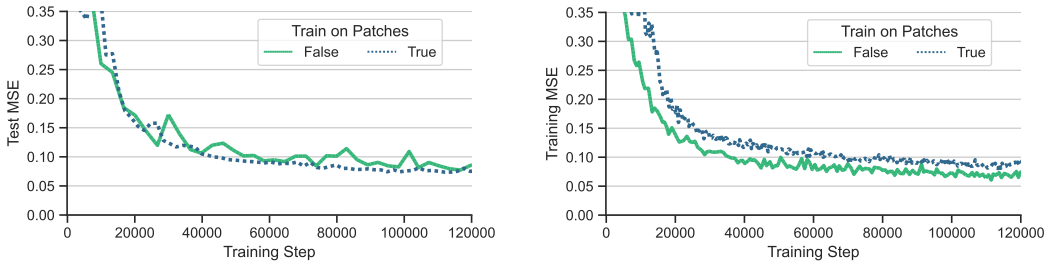

Figure 7: **Patch training can replace regular training on the full domain**. Training on patches produced by domain decomposition leads to similar test (left) and training (right) performance, even though patch training uses randomly sampled patches from different simulations and timesteps. This illustrates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

D.4 ADDITIONAL RESULTS ON HIGHER-ORDER INTEGRATION

In Figure 3, we observed that Heun's second-order method benefits test errors; Figure 10 shows that this benefit is stable across training runs. In Figure 9, we also show that this benefit also extends to training. Notably, with Heun's third-order method, training error decreases even faster (see Figure 11).

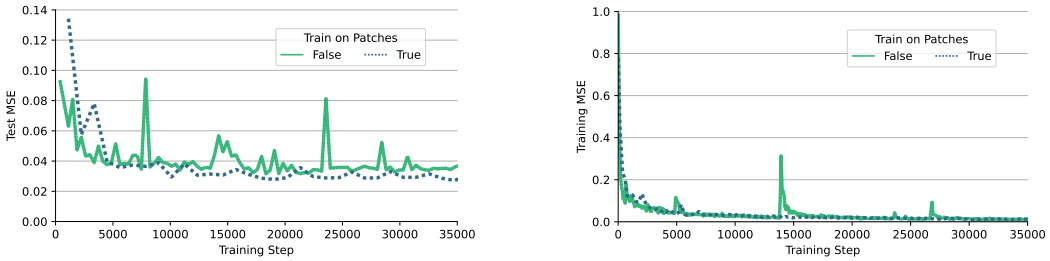

Figure 8: **Patch training is effective on CylinderFlow**. Patch training on the CylinderFlow simulations improves the loss on the test (left) and training (right) datasets.

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 suggests that 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.

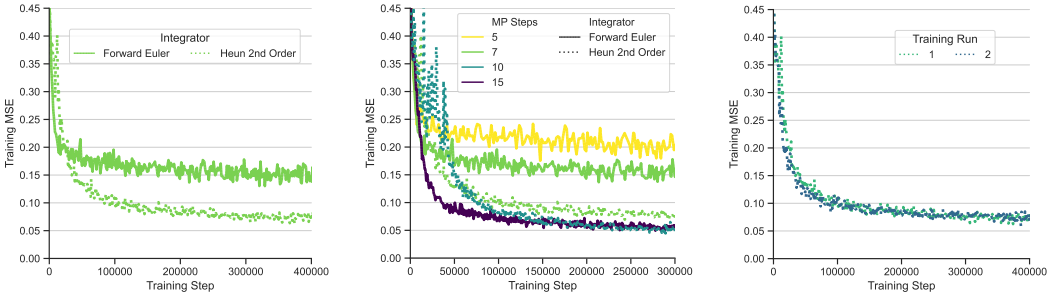

Figure 9: **Higher-order integration enhances learning on training data**. (*Left*) Using Heun's second-order (H2) method on an MGN with 7 MP steps halves the training error, relative to the baseline forward Euler (FE) method. (*Middle*) H2 and 7 MP steps is competitive with FE and 15 MP steps, despite the visible (and previously documented) benefits of more MP steps. (*Right*) The benefits of higher-order integration are stable across two training runs. Experiments run on 2D C3FD.

E Related work

Neural network surrogates for similar large-scale CFD simulations have been studied previously (Bartoldson et al., 2022), 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) can learn generalizable rules for updating physics fields from meshes, and MultiScale MGNs attained state-of-the-art performance on many tasks (Fortunato et al., 2022), but their memory inefficiencies prevented their use on large, application-relevant graphs.

More broadly, scaling GNNs to large graphs is an important topic that has been approached in various ways, including parallelism and off-accelerator memory use (Jia et al., 2020) and more efficient learning algorithms. Prior algorithmic work has improved operating efficiency on large graphs by sampling (Hamilton et al., 2017; Chiang et al., 2019) or otherwise modifying the function computed by an inefficient GNN (Zhou et al., 2021)—such approaches may be critical when average node degree is high. In contrast, we focused on domains that demand physical faithfulness, making approximations less attractive.[2] Accordingly, we take advantage of the manageable node degree associated with

[2]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.

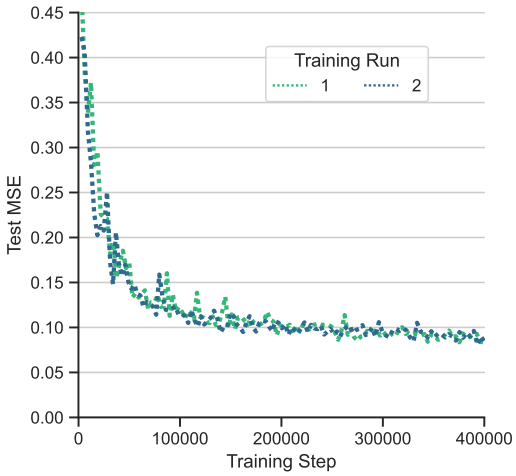

Figure 10: **Higher-order integration's benefit is stable across training runs.**

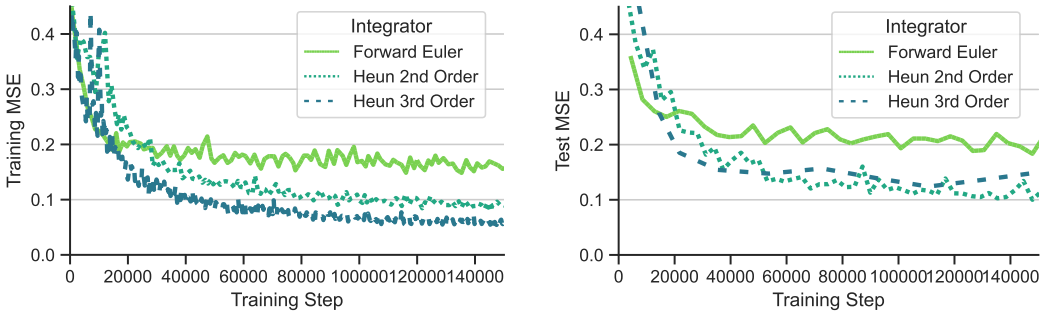

Figure 11: **Third-order integration improves training but not test performance**. All MGN models use 7 MP steps.

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.

Prior to our work, Janny et al. (2023) introduced EAGLE, a benchmark dataset for physics surrogate learning with over 1,000 2D simulations. EAGLE contains meshes with 3,388 nodes per timestep on average, while our 3D C3FD dataset uses a mesh with 3.1M nodes at each timestep ($\sim$ 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 involve 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) showed that the number of message-passing iterations can be reduced by using attention and graph pooling. In contrast, we showed that the number of message-passing iterations can be reduced by using a higher-order integrator.

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 used GNNs that were embedded within a differential equation solving framework (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 used higher-order integration as a byproduct of inserting a GNN into a differential equation solver, which runs for several timesteps. In contrast, we motivated 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 tested 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.[3] Also, better performance may be seen with other integrators, such as Ralston's methods, to minimize the local truncation error (Ralston, 1962). Relatedly, our results indicate that training details (such as 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 more computation on the accelerator and (if computation on the accelerator is a bottleneck) longer training steps. However, it can also reduce the number of training steps needed to reach a target accuracy. Indeed, while we do not thoroughly characterize speedups associated with higher-order integration, there were parameter settings where training speedups resulted from its use. In particular, our 3D models trained with H2 reached target error levels in strikingly fewer steps than FE models (see Figure 2) while having the same time per training step (≈ 1.05 seconds): e.g., models beat 0.33 test MSE in about 12 hours ($\approx 40K$ steps) with H2 compared to about 54 hours ($\approx 185K$ steps) with FE.

Another limitation of our work is that we used 32 accelerators to train on the 3D C3FD dataset quickly with a large effective batch size (32 patches). Future work may consider using smaller batch sizes (perhaps with correspondingly smaller learning rates) to reduce the relevance of accelerator count to training MGN-like models on very large computational domains such as those in C3FD.

Relatedly, 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 training that is mathematically equivalent to training on the full domain directly. Training on a patch is more compute-intensive than training on just the subdomain. Moreover, on our 3D meshes, average node degree is higher than on 2D meshes (using edges created via Delaunay triangulation). This leads to faster growth in subdomains' k-hop neighborhoods and, using 2,160 3D patches, out-of-memory errors on a 16GB GPU when going beyond $k = 7$ (see Figure 12 for a depiction of this growth). Since larger k-hop neighborhoods may be desired without resorting to use of more accelerator memory or creation of more but smaller patches, future work may consider creating edges with a nearest neighbors algorithm to tame node degree. Alternatively, a partitioning approach that balances the node counts of the resulting patches rather than subdomain spatial extents may help. Finally, this issue may also be addressed by violating condition 1 (i.e., by setting k to be less than the MP steps m): preliminary experiments we ran on CylinderFlow suggest that performance is robust to using smaller k, provided that a majority of nodes in the subdomain have access to their true m-hop neighbors. We expect that further work on these issues will lead to better scaling and performing GNN surrogate simulators for mesh physics.

[3]We note that the memory and compute overhead of higher-order integration may be addressed by lowering the MP steps: for example, FE with 15 MP steps has roughly the same memory cost and performance as Heun's second-order method with 7 MP steps.

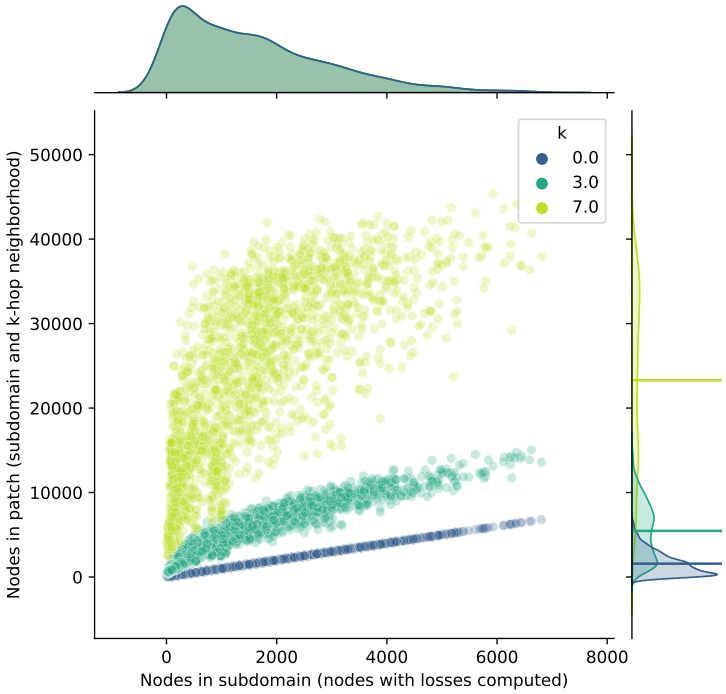

Figure 12: **With 3D C3FD, increasing k leads to increasingly larger patches.** Each patch contains its subdomain's k-hop neighborhood. For each k, horizontal bars on the right denote the average number of nodes in a patch, which increases nonlinearly with k. Thus, as the number of MP steps m increases, satisfying condition 1 from Section 2.2 ($m \leq k$) requires increasingly more resources.