Efficient Learning of Mesh-Based Physical Simulation with Bi-Stride Multi-Scale Graph Neural Network

Yadi Cao [1 †] **Menglei Chai** [2 †] **Minchen Li** [3] **Chenfanfu Jiang** [3]

Abstract

Learning the physical simulation on large-scale meshes with flat Graph Neural Networks (GNNs) and stacking Message Passings (MPs) is challenging due to the scaling complexity w.r.t. the number of nodes and over-smoothing. There has been growing interest in the community to introduce *multi-scale* structures to GNNs for physical simulation. However, current state-of-the-art methods are limited by their reliance on the labor-intensive drawing of coarser meshes or building coarser levels based on spatial proximity, which can introduce wrong edges across geometry boundaries. Inspired by the bipartite graph determination, we propose a novel pooling strategy, *bi-stride* to tackle the aforementioned limitations. Bi-stride pools nodes on every other frontier of the breadth-first search (BFS), without the need for the manual drawing of coarser meshes and avoiding the wrong edges by spatial proximity. Additionally, it enables a one-MP scheme per level and non-parametrized pooling and unpooling by interpolations, resembling U-Nets, which significantly reduces computational costs. Experiments show that the proposed framework, *BSMS-GNN*, significantly outperforms existing methods in terms of both accuracy and computational efficiency in representative physical simulations.

1. Introduction

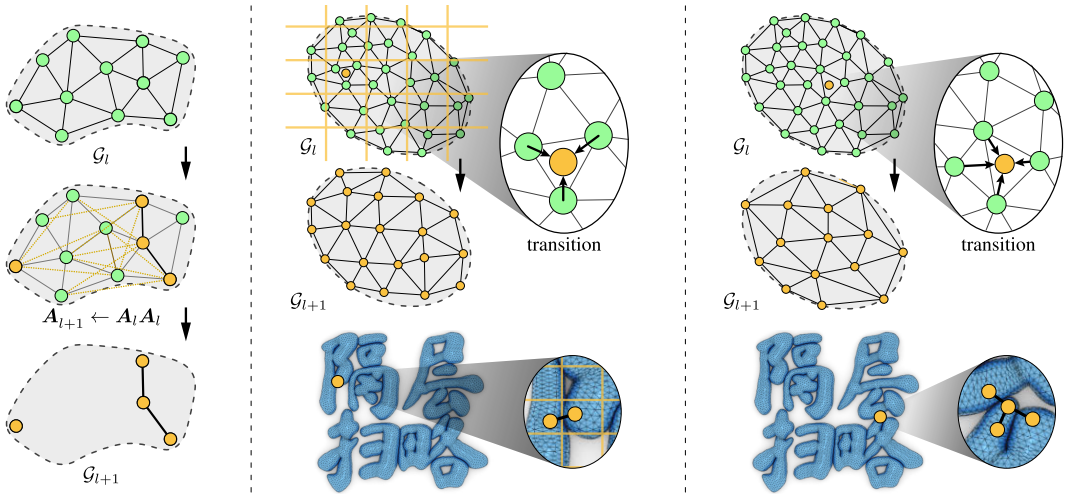
Simulating physical systems through numerically solving partial differential equations (PDEs) is a key area in science and engineering, with applications ranging from solid mechanics (Jiang et al., 2016; Li et al., 2020a), to fluid (Bridson, 2015; Cao & Li, 2018), aerodynamics (Cao et al., 2022), and heat transfer (Cao et al., 2019). However, traditional numerical solvers are often computationally expensive, particularly for time-sensitive applications such as iterative design optimization, where fast online inference is desired. In recent years, the machine learning community has shown great interest in improving efficiency or replacing traditional solvers with learned models. These works include both end-to-end frameworks (Grzeszczuk et al., 1998; Obiols-Sales et al., 2020) and those utilizing physics-informed losses (Raissi et al., 2019; Karniadakis et al., 2021; Sun et al., 2020). Many existing works apply convolutional neural networks (CNNs) (Fukushima & Miyake, 1982) to physical systems residing on two- or three-dimensional structured grids (Guo et al., 2016; Tompson et al., 2017; Kim et al., 2019; Fotiadis et al., 2020). However, the strict dependency on regular domain shapes makes it non-trivial to be applied on unstructured meshes. While it is possible to deform simple irregular domains into rectangular shapes to apply CNNs (Gao et al., 2021; Li et al., 2022a), the challenge remains for domains with complex topologies, which are common in practice.

As a result, the use of graph neural networks (GNNs) in physics-based simulations on unstructured meshes has gained significant attention in recent years (Battaglia et al., 2018; Sanchez-Gonzalez et al., 2018; Belbute-Peres et al., 2020; Pfaff et al., 2020; Sanchez-Gonzalez et al., 2020; Harsch & Riedelbauch, 2021; Gao et al., 2022). The naive GNN approach stacks multiple MPs to model information propagation throughout space. However, as the graph size increases, this approach faces two major challenges: (1) **Complexity**: as both the number of nodes to be processed and the MP iterations increase linearly, a quadratic complexity becomes inevitable for both the running time and memory usage of the computational graph (Fortunato et al., 2022). (2) **Oversmoothing**: the graph convolution can be seen as a low-pass filter that suppresses the higher-frequency signals (Chen et al., 2020; Li et al., 2020c). Then the stacked MPs iteratively project the information onto the eigenspace of the graph while all higher-frequency signals are smoothed out, making the training more difficult.

To address these limitations, researchers have begun intro-

[1]Department of Computer Science, UCLA, Los Angeles, USA [2]AR Perception, Google, Los Angeles, USA [3]Department of Mathematics, UCLA, Los Angeles, USA. Correspondence to: Chenfanfu Jiang <cffjiangmath.ucla.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.



(a) learnable pooling (b) pooling by rasterization (c) pooling by spatial proximity

Figure 1. **Issues of existing multi-level GNNs.** (a) A learnable pooling (Gao & Ji, 2019) may lead to loss of connectivity even after 2^{nd}-order enhancement. (b) A pooling by rasterization (Lino et al., 2021; 2022a;b) and (c) by spatial proximity (Liu et al., 2021; Fortunato et al., 2022) can lead to wrong connections across the boundaries at the coarser level.

ducing multi-scale GNNs (MS-GNNs) for physics-based simulation (Li et al., 2020c; Liu et al., 2021; Lino et al., 2021; Fortunato et al., 2022; Lino et al., 2022b;a). The multi-scale approach mitigates over-smoothing by building sub-level graphs at coarser resolutions, resulting in longer-range interactions and fewer MP iterations. The existing approaches for constructing the multi-scale structure include: utilizing spatial proximity to generate sub-level graphs at coarser levels (Lino et al., 2021; Liu et al., 2021; Lino et al., 2022a); applying Guillard's coarsening algorithm (Guillard, 1993) (Lino et al., 2022b); manually drawing coarser meshes for the original geometry (Liu et al., 2021; Fortunato et al., 2022); or randomly pooling nodes and applying factorization to the adjacency matrix (Li et al., 2020c). However, these solutions all have their limitations. For example, learnable or random pooling can introduce artificial partitions in the sub-level graphs (Fig. 1. (a)), even with adjacency enhancement, which impedes information exchange across partitions, spatial proximity can lead to wrong edges across the boundaries at coarser levels (Fig. 1. (b) and (c)); Guillard's algorithm only applies to 2D triangle meshes; and manually drawing tens of thousands of meshes is too labor-intensive. We observe that all these limitations originate from the unmatured operations: *pooling* and *building graph connections at coarser levels*. We desire to design operations that: 1) conserve the correct connections at coarser levels, 2) do not introduce edges that blur the boundaries, 3) are general for any mesh type, and 4) are automatic.

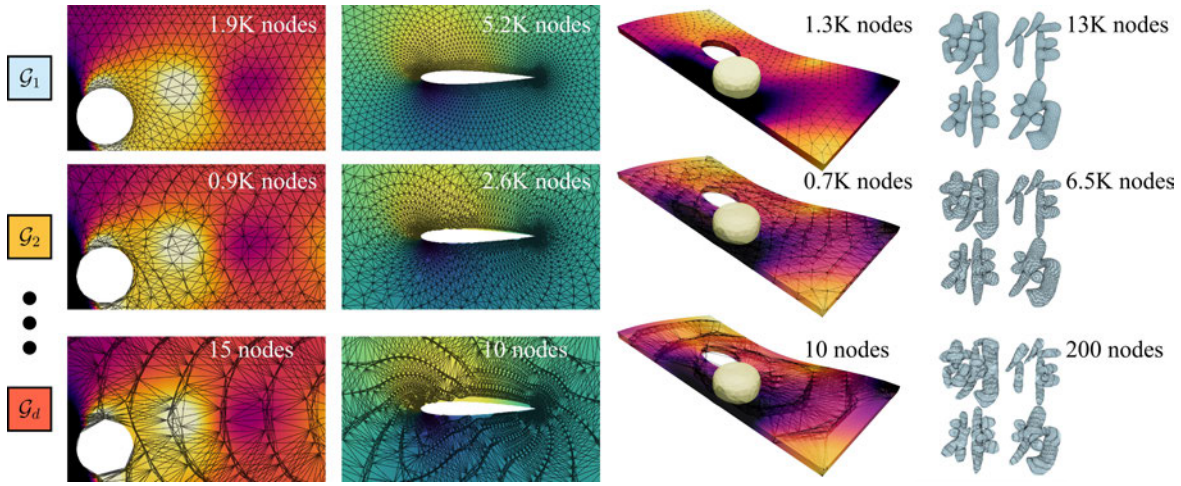
We tackle these challenges with two progressive contributions:

- **First**, we introduce a novel yet simple pooling strategy, *bi-stride*. Bi-stride is inspired by the bi-partition determination in DAG (directed acyclic graph). It pools all nodes on every other BFS (breadth-first-search) frontier, such that a 2^{nd}-powered adjacency enhancement ($\boldsymbol{A} \leftarrow \boldsymbol{A}^2$, where $\boldsymbol{A}$ is the adjacency matrix of the graph) conserves all the correct connectivity. Bi-stride solely uses the input mesh without the need for spatial proximity, is general for any mesh type, and is fully automatic.
- **Second**, bi-stride pooling conserves direct connections between pooled/un-pooled nodes; Utilizing this advantage, one MP suffices to exchange the information between pooled and unpooled nodes before moving into the adjacent level; We also design a non-parameterized aggregating and returning method, resembling the interpolation in U-Net, to handle the transition between adjacent levels. These simplifications significantly reduce the computational requirements compared to SOTAs.

Together, these two contributions give birth to our *Bi-Stride Multi-Scale GNN* (*BSMS-GNN*), a novel framework representing a significant advancement in the field of learned mesh-based simulations, particularly for the deployment in real industrial applications where meshes are often complex in geometry and large in size.

2. Multi-Scale Building as Preprocessing

We first introduce the bi-stride pooling strategy, multi-scale building, and the corresponding data preprocessing. Note that all algorithms and preprocessing steps in this section are deterministic and done in one pass.



(a) CylinderFlow (b) Airfoil (c) DeformingPlate (d) InflatingFont

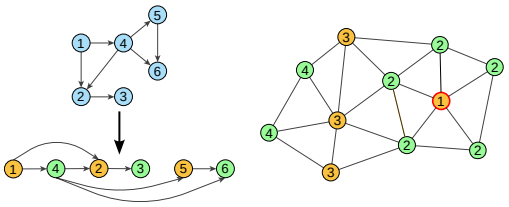
Figure 2. **Example multi-level graphs produced by bi-stride pooling.** Our datasets contain both Eulerian and Lagrangian systems. Many meshes are highly irregular and contain massive self-contact, challenging to build coarser-level connections by spatial proximity. Our bi-stride strategy only relies on topological information and is proven to be robust and reliable on arbitrary geometry.

2.1. Motivations

As summarized in Fig. 1. The pooling strategy can be categorized into two groups by either utilizing spatial proximity (Lino et al., 2021; 2022a) or purely relying on graph information (Gao & Ji, 2019; Li et al., 2020c). For complex geometry, it's preferred to avoid spatial proximity unless desired by the simulation cases (such as contact or interface interactions). The only benchmark that uses only graph information is GraphUNets (Gao & Ji, 2019) with an obvious drawback: it's easy to lose the connectivity and artificially introduce partitions, even with adjacent matrix enhancement (Fig. 1 (a)).

For a more clear illustration, we first define the adjacency enhancement by the K^{th}-order matrix power as $\boldsymbol{A} \leftarrow \boldsymbol{A}^K$, where $\boldsymbol{A}$ is the adjacency matrix of the graph. Geometrically, $\boldsymbol{A}(i,j) = 1$ means the edge (i,j) exists, and $\boldsymbol{A}^K(i,j) = 1$ means that node j is connected to node i via at most K hops. Given a pooling strategy P and the pooled nodes' indices $\mathcal{I}$, we define a Kth-order outlier set as $\mathcal{O}_K$, where the nodes in $\mathcal{O}_K$ are not connected to any pooled nodes even after K^{th}-order adjacency enhancement: $\boldsymbol{A}^K(i,j) = 0, \forall i \in \mathcal{I}, \forall j \in \mathcal{O}_K$.

We finally define that a pooling strategy P is K^{th}-order connection conservative (K-CC) if $\mathcal{O}_K$ is empty. Empirically, the larger K in K^{th}-order adjacency enhancement is harmful to distinguish the node features: as K increases, $\boldsymbol{A}^K(i,j)$ (converted to boolean) approaches a matrix with all its entries equal to 1, representing a fully connected graph; then a single convolution will average all node features and make them indistinguishable. The most preferred, i.e. the smallest possible K we seek is naturally 2. With such reason, Gao & Ji (2019) uses the smallest 2^{nd} order enhancement to help conserve the connectivity. Nonetheless, there is no theoretical guarantee that a learnable pooling module is consistently 2-CC for any graph. These limitations motivate us to create a consistent 2-CC pooling strategy, as followed in Sec. 2.2.



(a) Bi-partition of a DAG (b) Bi-stride of a mesh

Figure 3. **The similarity between Bi-partition and Bi-stride.** The number in the circle means the depth of the frontiers of either topological sorting or BFS. The red-bounded circle (number 1) means the starting node, i.e. the seed.

2.2. Bi-Stride Pooling and Adjacency Enhancement

We draw the initial inspiration from the bi-partition determination algorithm (Asratian et al., 1998) in a directed acyclic graph (DAG). As shown in Fig. 3(a), after topological sorting, pooling on every other depth (yellow and green) generates a bi-partition where all edges reside between two partitions, and pooling either partition is obviously 2-CC. To resemble bi-partition determination on a mesh, which is not bi-partite due to cycles, we can conduct a breadth-first search (BFS) to compute the geodesic distances from a seed to all other nodes, and then stride and pool all nodes at every other BFS frontier (*bi-stride*). A bi-stride example is shown in Fig. 3(b), where the number in each vertex represents the geodesic distance to the seed (node 1 in the red circle) by BFS. This pooling is 2-CC by construction and conserves direct connections between pooled and unpooled

nodes. As a result, we avoid building edges by using spatial proximity or handling cumbersome corner cases such as cross-boundary connections while also using the smallest adjacency enhancement.

Seeding Heuristics We claim that there should exist some freedom as long as the seeding is balanced to a certain degree. For training datasets, we choose two deterministic seeding heuristics: 1) closest to the center of a cluster (CloseCenter) for INFLATINGFONT, and 2) the minimum average distance (MinAve) for all other cases, and we preprocess the multi-level building in one pass. One can consider the cheaper heuristic CloseCenter during the online inferring phase if an unseen geometry is encountered. The details of the algorithms can be found in Sec.A.7. The sensitivity study on choosing a different seeding heuristic in the test phase is presented in Sec.A.4.2.

Contact Edges For problems involving contacts, such as DEFORMINGPLATE (Fig. 2(c)) and INFLATINGFONT (Fig. 2(d)), the finest-level contact edges $\boldsymbol{A}^C$ are built dynamically by spatial proximity between nodes. Note the edge-building by the spatial proximity does not apply to the internal elastic mechanics, whose edge is defined by the mesh only. The enhancement of the contact edges should be handled properly for multi-scale GNN, which, to the best of our knowledge, has not been addressed in prior works. At any level l, given adjacent matrices $\boldsymbol{A}_l$ obtained by the input mesh and the enhancement rule, and the contact edge $\boldsymbol{A}_l^C$ at this level, we first apply bi-stride pooling to select nodes $\mathcal{I}$, then enhance $\boldsymbol{A}_{l+1}$ and $\boldsymbol{A}_{l+1}^C$ using the following rule, where $[\mathcal{I},\mathcal{I}]$ means striding on the matrix rows and columns:

$$\begin{aligned} \boldsymbol{A}'_{l+1} &\leftarrow \boldsymbol{A}_l\boldsymbol{A}_l, \quad \boldsymbol{A}_{l+1} \leftarrow \boldsymbol{A}'_{l+1}[\mathcal{I},\mathcal{I}], \\ \boldsymbol{A}'^C_{l+1} &\leftarrow \boldsymbol{A}_l\boldsymbol{A}_l^C\boldsymbol{A}_l, \quad \boldsymbol{A}_{l+1}^C \leftarrow \boldsymbol{A}'^C_{l+1}[\mathcal{I},\mathcal{I}]. \end{aligned} \tag{1}$$

The enhancement of contact edges can be geometrically interpreted as contact edge (i,j) should exist if j is reachable from i in 2 hops and at least one of them is a contact edge at the finer level. We prove in Sec. A.6 that bi-stride pooling with the enhancement in Eq. 1 also conserves all contact edges.

3. Bi-Stride Multi-Scale (BSMS)-GNN

Here we formally introduce BSMS-GNN, a hierarchical GNN where the multi-level structure has been determined by the input mesh and the preprocessing in Sec. 2.

3.1. Definitions

Figure. 4 presents the overall structure of BSMS-GNN. We consider the evolution of a physics-based system discretized on a mesh, which is converted to an bi-directed graph $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{E}_1)$. Here, with subscript 1, $\mathcal{V}_1$, and $\mathcal{E}_1$ label the nodal fields and the edges at the finest level (the input mesh), respectively. Specifically for edges, we define $\mathcal{E}_1 = \{\mathcal{E}_1^1, \cdots, \mathcal{E}_1^S\}$, where $\mathcal{E}_1^1$ is the edge set directly copied from the input mesh and $\{\mathcal{E}_1^k|_{k=2}^S\}$ are the optional problem-dependent edge sets. For example, both DEFORMINGPLATE (Fig. 2(c)) and INFLATINGFONT (Fig. 2(d)) benchmarks have a contact edge set $\mathcal{E}_1^2$ for the colliding vertices. We use $\{p,q\}$, stacked vectors of $\{p_i,q_i\}$ of all nodes $i \in \mathcal{V}_1$, to denote the input and output nodal fields, respectively. Given an input field p^j at a time t_j, one pass of BSMS-GNN returns the output field q^{j+1} at time $t_{j+1} = t_j + \Delta t$, where Δt is a fixed time step size. The output q can contain more physical fields than the input p and must be able to derive the input for the next pass. The rollout refers to iteratively conducting BSMS-GNN from the initial state $p^0 \to q^1 \to p^1 \to \cdots \to q^n$ and producing the temporal sequence output $\{q^1, q^2, \cdots, q^n\}$ within the time range of $(t_0, t_0 + n\Delta t]$, where n is the total number of evaluations.

Message Passing In general, we follow the *encode-process-decode* fashion in MESHGRAPHNETS, where encoder and decoder only appear at the top level $\mathcal{G}_1$, mapping the nodal input p and output q to/from the latent feature $\boldsymbol{v}$, respectively (see Table A.1 for the domain-specific information). As for the processor, unlike (Fortunato et al., 2022), we observe that a single MP per level is sufficient for all experiments. We do not separately encode the edge offsets $\Delta\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$, instead, simply prepend this to the stacked sender/receiver latent as the input to calculate edge flow. For a problem involving S edge sets, an MP pass at level l is formulated as:

$$\begin{aligned} \boldsymbol{e}^s_{l,ij} &\leftarrow \mathrm{f}^s_l\left(\Delta\boldsymbol{x}_{l,ij}, \boldsymbol{v}_{l,i}, \boldsymbol{v}_{l,j}\right), \quad s = 1, \cdots, S, \\ \boldsymbol{v}'_{l,i} &\leftarrow \boldsymbol{v}_{l,i} + \mathrm{f}^V_l\left(\boldsymbol{v}_{l,i}, \sum_j \boldsymbol{e}^1_{l,ij}, \cdots, \sum_j \boldsymbol{e}^S_{l,ij}\right), \end{aligned} \tag{2}$$

where f is a MLP function, $\boldsymbol{e}$ is the latent information flow through an edge, and $\boldsymbol{v}$ is the latent node feature. Please refer to Sec. A.2 for the detailed architecture of the model.

3.2. Transition Between Levels

We handle information transition between two adjacent levels with non-parametrized *downsampling* and *upsampling* modules to reduce the overhead of the learnable transition modules between every pair of adjacent levels. Here we define downsampling as the sequence of pooling nodes and then aggregating the information from the neighbors to the coarser level, and upsampling as the sequence of unpooling and then returning the information of the pooled nodes to their neighbors at the finer level.

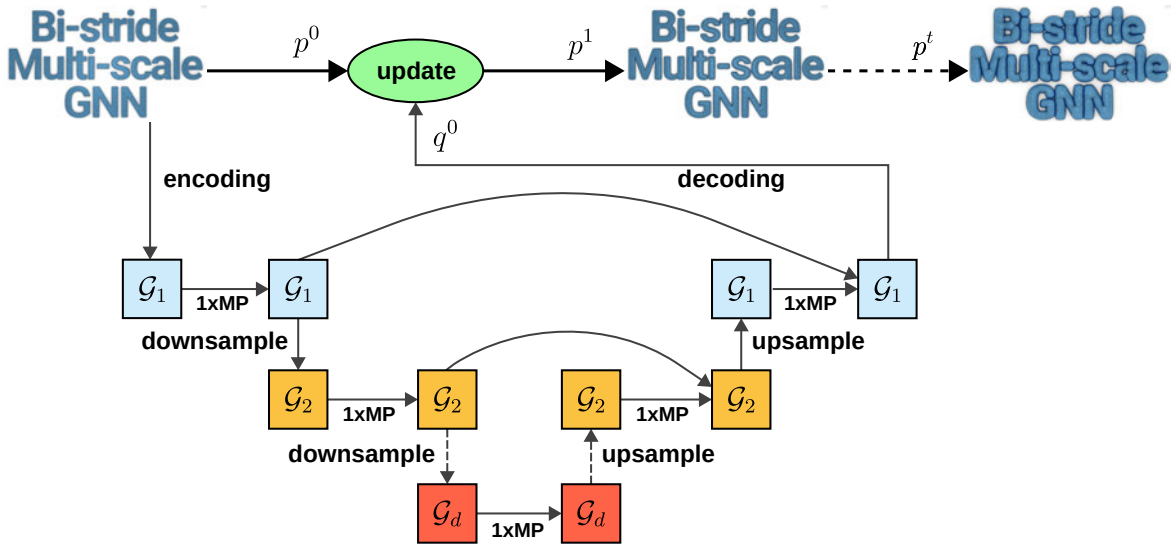


Figure 4. **BSMS-GNN pipeline** is trained with one-step supervision. $\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_d$ are graphs at different levels (fine to coarse). Encoder/decoder only connects the input/output fields with the latent fields at $\mathcal{G}_1$. Latent nodal fields are updated by one MP at each level. The bi-stride pooling selects the pooled nodes for the adjacent coarser level, and the transition is conducted in a non-parameterized way.

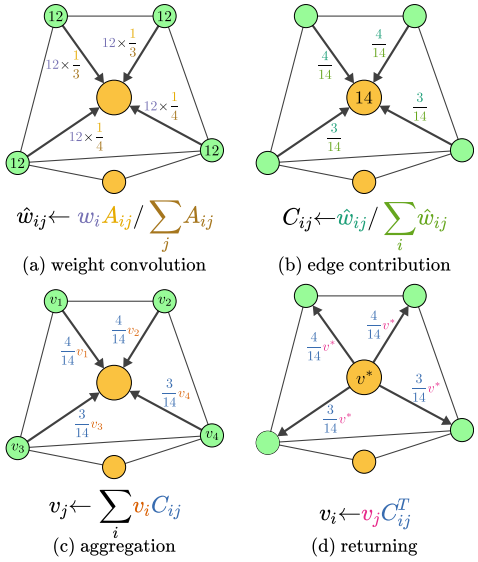


Figure 5. **Schematic plot of transition between adjacent levels.**

Downsampling Let $\boldsymbol{A}$ be the unweighted, boolean adjacency matrix. We initiate a nodal weight field w, which is one on the finest level and will be aggregated during downsampling. With the receiver j and its sender i, the formal procedure of downsampling is formulated as (Fig. 5):

- Normalize by row as in a standard graph convolution $\hat{\boldsymbol{A}}_{ij} \leftarrow \boldsymbol{A}_{ij} / \sum_j \boldsymbol{A}_{ij}$, and then convolve the weight once $\hat{w}_{ij} \leftarrow w_i \hat{\boldsymbol{A}}_{ij}$ (Fig. 5(a));
- Calculate edge weights $\boldsymbol{C}_{ij} \leftarrow \hat{w}_{ij} / \sum_i \hat{w}_{ij}$, where $\boldsymbol{C}$ can be viewed as a contribution table with $\boldsymbol{C}_{ij}$ as the share of weights in receiver j contributed by sender i (Fig. 5(b));
- Convolve the latent information by contribution table $\boldsymbol{v}_j \leftarrow \sum_i \boldsymbol{v}_i \boldsymbol{C}_{ij}$, which is equivalent to equally splitting and sending the weighted information out from senders, and then conducting the weighted average on receivers (Fig. 5(c)).

Upsampling After unpooling, all nodes except pooled ones have zero information. A returning process, resembling transposed convolution in U-Net, can help distinguish the information between receivers. With the contribution table $\boldsymbol{C}$ recording edge weights, a natural choice is $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_j \boldsymbol{C}_{ij}^T$ (Fig. 5(d)).

4. Experiments

4.1. Experiment Setup

Datasets We adopt three representative public datasets from GraphMeshNets (Pfaff et al., 2020): 1) CylinderFlow: incompressible fluid around a cylinder; 2) Airfoil: compressible flow around an airfoil; and 3) DeformingPlate: deforming an elastic plate with an actuator. In addition, we create a new dataset, InflatingFont, featuring the inflation of enclosed elastic surfaces (Fang et al., 2021). The example plots of them are plotted in Fig. 2. Compared to existing datasets, InflatingFont has more complex geometric shapes, 2 to 8 times the number of nodes, and 70 times the number of contact edges. Hence it is very suitable for testing the capability of competing GNNs in terms of scalability and compatibility with complex geometries. More details are included in Sec. A.1.

Baselines On all datasets, we compare the computational complexity, training/inference time, and memory footprint of BSMS-GNN to baselines: 1) MESHGRAPHNETS (Pfaff et al., 2020): the single-level GNN architecture of GraphMeshNets; 2) MS-GNN-GRID (Lino et al., 2021; 2022a;b): a representative work for those building the hierarchy with spatial proximity; and 3) GRAPHUNETS (Gao & Ji, 2019): a representative work for those using learnable modules for pooling. The reimplementation details can be found in Sec. A.2. We note again that methods such as (Liu et al., 2021; Fortunato et al., 2022) are not practical because they require manually drawing coarser meshes for every trajectory instance with CAE software. For all cases combined, this means manually drawing about $20,000$ meshes.

Implementation We implement our framework with PyTorch (Paszke et al., 2019) and PyG (PyTorch Geometric) (Fey & Lenssen, 2019). We train the entire model by supervising the single-step L_2 loss between the ground truth and the nodal field output of the decoding module. More details, such as the network structures and hyperparameters are included in A.2. Our datasets and code are publicly available at *https://github.com/Eydcao/BSMS-GNN*.

MISCs The ablation study is conducted for the specific choice of our transition method in Sec. A.4. The ablation study concerning whether or not to use learnable pooling modules is not standalone listed but covered by comparing to GRAPHUNETS in full-scale experiments (details in Sec. 4.2).

4.2. Results and Discussions

By evaluating BSMS-GNN and other competitors on all the baselines (Sec. 4.1), we observe the following conclusions:

- We experimentally show the learnable pooling method is not applicable for the deployment of large-scale, complex geometries;
- We design a small-scale experiment where pooling by spatial proximity leads to wrong edge and inference;
- BSMS-GNN shows dominant advantages in significantly less memory footprint, training time to reach the desired accuracy, and the inference time;
- BSMS-GNN also reaches the highest accuracy, reducing the rollout RMSE approximately by half on INFLATINGFONT with the largest mesh size and the most complex geometries; we also zero-shoot the trained model on a teaser with approximately 7x more nodes with the same level of accuracy.

For conciseness, we plot the detailed results in Table. 1 and Table. 2 in Sec. A.3.

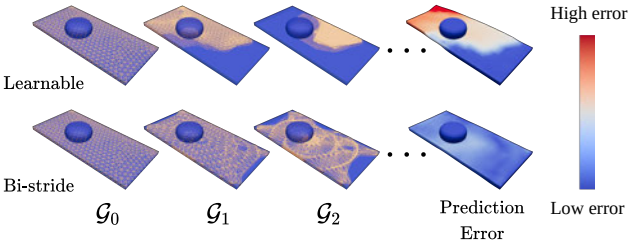


Figure 6. **BSMS-GNN Comparison between pooling methods**. Unlike bi-stride pooling, which generalizes to any input graph, a learnable module leads to unfair pooling on unseen geometries, impeding the information exchange for unselected nodes. The inferred results show larger errors than bi-stride pooling.

Disadvantages of Learnable Pooling Compared to other competitors, GRAPHUNETS shows a significantly higher RMSE in both 1-step and global rollouts on all datasets, except for the AIRFOIL dataset, where the mesh is consistent across trajectories. To confirm varying mesh leads to poor inference with learnable pooling, we apply the trained GRAPHUNETS model to an unseen test trajectory of the DEFORMINGPLATE dataset. Fig 6 clearly reveals the unfair pooling distribution by the learnable module, which impedes information passing on coarser levels and results in poor inference. In comparison, bi-stride generates uniform pooling and accurate inference.

Additionally, GRAPHUNETS has to conduct the adjacency matrix multiplication in the forward pass, which results in a 2-40x increase in the training and inference times, particularly in larger datasets. In the largest INFLATINGFONT, one training epoch requires nearly 50 hours to complete, making the convergence of the model infeasible. Given its poor performance in both accuracy and efficiency, we conclude that GRAPHUNETS is not suitable for simulation cases with large-scale, complex geometries. By default, we will not specifically make comparisons to GRAPHUNETS in the following discussions.

Failure Cases for Spatial Proximity To illustrate the adversarial impact of wrongly constructed edges by spatial proximity, we design a simple 1-D steady-state heat transfer simulation on sticks (Figure 7 left), on which BSMS-GNN and MS-GNN-GRID are trained and evaluated. The training set consists of two mirrored instances, where one end of the stick is fixed at a certain temperature and the other end has a fixed heat flux, resulting in a linear temperature distribution. In the test set, we simply align two sticks in a head-to-tail configuration with a tiny space between them to prevent heat diffusion across the boundary. MS-GNN-GRID, utilizing spatial proximity, incorrectly constructs an edge between the two sticks. As a result, in half of the test cases, MS-GNN-GRID shows wrong results at the boundary due to the erroneous edge(Figure 7 right, leading two

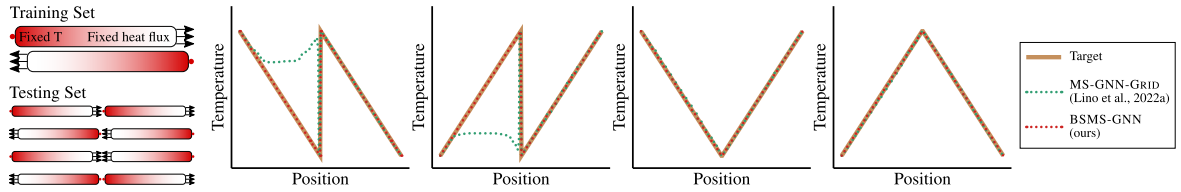
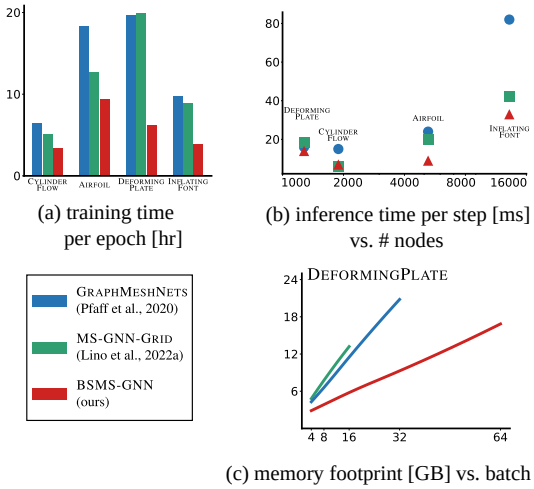


Figure 7. **Failure cases for MS-GNN-GRID.** Left: the configuration of the simplest failure case for multi-level GNNs by spatial proximity: steady-state 1-D heat transfer. Right, leading two columns: two tests showing that even if trained to convergence, the erroneous edge across the boundary can still result in the wrong inference. Right, last two columns: the erroneous edge coincidentally does not affect the results due to the symmetry of the solution, and no heat will diffuse between two nodes with the same temperature.



(a) training time per epoch [hr]

(b) inference time per step [ms] vs. # nodes

(c) memory footprint [GB] vs. batch size

Figure 8. **Performance comparison between BSMS-GNN, MS-GNN-GRID, and MESHGRAPHNETS.** GRAPHUNETS is not included because of poor performance. Full results are plotted in Table. 1 and Table. 2 in Sec. A.3.

columns). Although only in simple cases, one can alleviate this issue by marking the two sticks as separate clusters and making inferences independently; a similar fix is unfeasible for connected, complex geometries. For instance, in a long, thin U-shaped tunnel, two nodes located on the parallel sides of the "U" are spatially close but geodesically distant, and hence should not be connected by an edge.

Accuracy and Generalization In terms of accuracy, our method has the smallest rollout RMSE for all cases except for the DEFORMINGPLATE dataset, where all three competitors have similar results. We assume the main reason is that the mesh size for DEFORMINGPLATE is too small, so the flat architecture does not show any disadvantage. In the largest and most complex dataset, INFLATINGFONT, our method achieves the highest accuracy, cutting down 40% of the rollout RMSE compared to the competitors (Fig. 9. Left.). Additionally, our model demonstrates the ability to perform zero-shot inference on larger meshes than those in the training set, and still produce accurate global rollouts, even when the characters are written in a different language.

Performance Advantages of BSMS-GNN Our method has a simplified and lightweight model architecture, characterized by a reduced number of MPs at each level and the absence of learnable transition modules. This results in a significant reduction in memory footprint during training (in Fig.8. (c) and Table.2); BSMS-GNN consumes $43\% \sim 87\%$ memory in training as MS-GNN-GRID, $48\% \sim 53\%$ as MESHGRAPHNETS, and only 10% as GRAPHUNETS. our method also uses the least memory during inference, except for the DEFORMINGPLATE dataset where the consumption is slightly higher (20MBs) than MESHGRAPHNETS.

Memory reduction also contributes to improved training efficiency, as it allows for larger batch sizes, more random sampling, and fewer times of data swapping between CPU and GPU. Combined, our method has the fastest unit training speed (per epoch) among all competitors (Fig.8. (a) and Table.1), where it consumes only $26\% \sim 58\%$ unit training time as that of MS-GNN-GRID and MESHGRAPHNETS.

In terms of inference time, our method exhibits similar performance to MS-GNN-GRID on smaller mesh size datasets (CYLINDERFLOW and DEFORMINGPLATE), both outperforming MESHGRAPHNETS. However, as the mesh size increases, BSMS-GNN surpasses MS-GNN-GRID, showing better scalability. In large mesh size cases (AIRFOIL and INFLATINGFONT), our method shows a $1.5\times$ and $1.9\times$ improvement over MS-GNN-GRID and MESHGRAPHNETS, respectively (Fig.8. (b) and Table.1).

Concerning the total training cost to reach a desired global rollout RMSE, we observe that all methods reach the target with a similar number of epochs. This is because that all these methods learn to resist rollout noise by seeing different, random noises at each epoch; enough noise patterns (proportional to epoch number) is the key for accurate rollout; as such, the total wall time is roughly proportional to the unit training time, given similar epoch numbers, making our method the most efficient.

Scaling Analysis By training and evaluating competing methods on INFLATINGFONT with varying resolutions (5K,15K,30K, and 45K), we observe that both BSMS-GNN and MS-GNN-GRID scale up with a near-linear

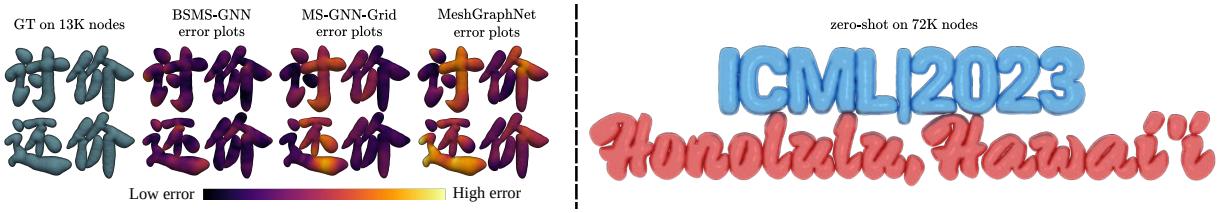


Figure 9. **Left:** Comparing the rollout RMSE between BSMS-GNN and existing SOTAs on benchmarks INFLATINGFONT. Our framework reaches the highest accuracy, showing a stronger capability for complex, large geometries with massive contacts. **Right:** Our trained model can zero-shot infer on never-seen fonts, with about 7x size in mesh size with the same accuracy.

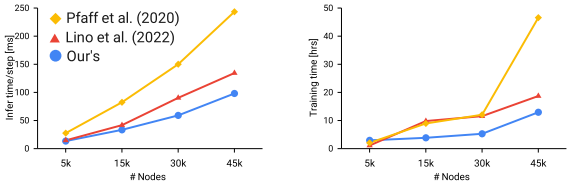


Figure 10. **Scaling analysis**. With the growing size of INFLATINGFONT, BSMS-GNN shows a progressive advantage over MESHGRAPHNETS and MS-GNN-GRID.

growing rate; still, our method is more efficient than MS-GNN-GRID as illustrated earlier(Fig. 10). We leave the details of scaling analysis in A.5.

5. Related Works

GNNs for Physics-Based Simulation The application of GNNs to physics-based simulation has first been applied to deformable solids and fluids (both represented by particles) (Sanchez-Gonzalez et al., 2018). A notable milestone in this field is MESHGRAPHNETS (Pfaff et al., 2020), which enables the general scheme for learning mesh-based simulations. Subsequently, several variants of MESHGRAPHNETS have been proposed: such as combining GNNs with Physics-Informed Neural Networks (PINNs) (Gao et al., 2022), making long-term predictions by combining the GraphAutoEncoder (GAE) and Transformer (Han et al., 2022), directly predicting the steady-states through the multi-layer readouts (Harsch & Riedelbauch, 2021), and accelerating the finer-level simulation by feeding the up-sampled coarser results inferred by GNN (Belbute-Peres et al., 2020).

Multi-Scale GNNs Multi-scale GNNs (MS-GNNs) have been widely used in general graph-related tasks other than physics (Wu et al., 2020; Mesquita et al., 2020; Zhang et al., 2019). The GraphUNet(Gao & Ji, 2019) introduces the UNet structure into GNNs with a trainable scoring module for pooling, and a 2^{nd}-powered adjacency enhancement to help conserve the connectivity. Several works have investigated the use of MS-GNNs for physics-based simulations, including the two- and multi-level GNNs(Fortunato et al., 2022; Liu et al., 2021), which rely on manually drawing coarse meshes. Works such as MS-GNN-GRID (Lino et al., 2021; 2022a) rely on spatial proximity to generate multi-level structures. Li et al. (2020c) adopts multi-level matrix factorization to generate the kernels at coarser levels. Lino et al. (2022b) utilizes Guillard's coarsening algorithm to build the coarse-level meshes, but only for 2-D triangle elements. Additionally, there are representative works that abandon the original mesh and build connections and hierarchies on point clouds, such as GNS (Sanchez-Gonzalez et al., 2020), PointNet (Qi et al., 2017a), PointNet++(Qi et al., 2017b), and GeodesicConv(Masci et al., 2015).

6. Conclusion, Limitations, and Future Work

The Bi-Stride Multi-Scale Graph Neural Network (BSMS-GNN) utilizes a novel pooling strategy that allows for the creation of an arbitrary-depth, multi-level graph neural network using the original mesh as the sole input. This approach eliminates the need for manually drawing coarser meshes and reduces the potential for wrong edges introduced by spatial proximity. Additionally, Bi-stride pooling enables a one-MP scheme and a non-parametric transition, resulting in a significant reduction in computational costs. Overall, BSMS-GNN improves the capability and generality of applying GNNs to large-scale physical simulations with complex geometries.

Further research following our path may include handling huge graphs through the combination of multi-level GNNs with batched and distributed training (Strönisch et al., 2023). Combining the neural operators (Li et al., 2020b;c), i.e. the ability to handle a wide range of PDE parameters, is also appealing. It would be interesting to combine BSMS-GNNs with Transformer (Han et al., 2022; Geneva & Zabaras, 2022; Li et al., 2022b) for stable roll-outs. Regarding more precise contact modeling, the point-point approach can be improved by face pairs (Allen et al., 2023). Finally, it is worth investigating strategies to score and prunes edges (Ding et al., 2006; Yu et al., 2014) at coarser levels.

ACKNOWLEDGMENTS

Yadi Cao would like to thank Na Li for her remote support and companionship throughout his research and study. We thank Neil Shah, Tong Zhao, and Sergey Tulyakov for their invaluable discussions on general GNNs. Yadi Cao would like to acknowledge Dr. Shaowu Pan for the initial inspiration of this work. We extend our appreciation to our reviewers for their valuable feedback on this work and manuscript. This work has been supported in part by NSF CAREER 2153851, CCF-2153863, ECCS-2023780.

References

Allen, K. R., Rubanova, Y., Lopez-Guevara, T., Whitney, W. F., Sanchez-Gonzalez, A., Battaglia, P., and Pfaff, T. Learning rigid dynamics with face interaction graph networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=J7Uh781A05p.

Asratian, A. S., Denley, T. M., and Häggkvist, R. *Bipartite graphs and their applications*, volume 131. Cambridge university press, 1998.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Belbute-Peres, F. D. A., Economon, T., and Kolter, Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *international conference on machine learning*, pp. 2402–2411. PMLR, 2020.

Bridson, R. *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015.

Cao, Y. and Li, R. A liquid plug moving in an annular pipe—flow analysis. *Physics of Fluids*, 30(9):093605, 2018.

Cao, Y., Gao, X., and Li, R. A liquid plug moving in an annular pipe–heat transfer analysis. *International Journal of Heat and Mass Transfer*, 139:1065–1076, 2019.

Cao, Y., Chen, Y., Li, M., Yang, Y., Zhang, X., Aanjaneya, M., and Jiang, C. An efficient b-spline lagrangian/eulerian method for compressible flow, shock waves, and fracturing solids. *ACM Transactions on Graphics (TOG)*, 41(5):1–13, 2022.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3438–3445, 2020.

Ding, C., He, X., Xiong, H., Peng, H., and Holbrook, S. R. Transitive closure and metric inequality of weighted graphs: detecting protein interaction modules using cliques. *International journal of data mining and bioinformatics*, 1(2):162–177, 2006.

Fang, Y., Li, M., Jiang, C., and Kaufman, D. M. Guaranteed globally injective 3d deformation processing. *ACM Trans. Graph.*, 40(4):75–1, 2021.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A., and Battaglia, P. Multiscale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop*, 2022.

Fotiadis, S., Pignatelli, E., Valencia, M. L., Cantwell, C., Storkey, A., and Bharath, A. A. Comparing recurrent and convolutional neural networks for predicting wave propagation. *arXiv preprint arXiv:2002.08981*, 2020.

Fukushima, K. and Miyake, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Gao, H. and Ji, S. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Gao, H., Sun, L., and Wang, J.-X. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428: 110079, 2021.

Gao, H., Zahr, M. J., and Wang, J.-X. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

Geneva, N. and Zabaras, N. Transformers for modeling physical systems. *Neural Networks*, 146:272–289, 2022.

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20, 1998.

Guillard, H. *Node-nested multi-grid method with Delaunay coarsening*. PhD thesis, INRIA, 1993.

Guo, X., Li, W., and Iorio, F. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 481–490, 2016.

Han, X., Gao, H., Pffaf, T., Wang, J.-X., and Liu, L.-P. Predicting physics in mesh-reduced space with temporal attention. *arXiv preprint arXiv:2201.09113*, 2022.

Harsch, L. and Riedelbauch, S. Direct prediction of steady-state flow fields in meshed domain with graph networks. *arXiv preprint arXiv:2105.02575*, 2021.

Jiang, C., Schroeder, C., Teran, J., Stomakhin, A., and Selle, A. The material point method for simulating continuum materials. In *ACM SIGGRAPH 2016 Courses*, pp. 1–52. 2016.

Karniadakis, G. E., Kevrekidis, I. G., Lu, L., Perdikaris, P., Wang, S., and Yang, L. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Kim, B., Azevedo, V. C., Thuerey, N., Kim, T., Gross, M., and Solenthaler, B. Deep fluids: A generative network for parameterized fluid simulations. In *Computer graphics forum*, volume 38, pp. 59–70. Wiley Online Library, 2019.

Li, M., Ferguson, Z., Schneider, T., Langlois, T. R., Zorin, D., Panozzo, D., Jiang, C., and Kaufman, D. M. Incremental potential contact: intersection-and inversion-free, large-deformation dynamics. *ACM Trans. Graph.*, 39(4): 49, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., and Anandkumar, A. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:6755–6766, 2020c.

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022a.

Li, Z., Meidani, K., and Farimani, A. B. Transformer for partial differential equations' operator learning. *arXiv preprint arXiv:2205.13671*, 2022b.

Lino, M., Cantwell, C., Bharath, A. A., and Fotiadis, S. Simulating continuum mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2106.04900*, 2021.

Lino, M., Fotiadis, S., Bharath, A. A., and Cantwell, C. Towards fast simulation of environmental fluid mechanics with multi-scale graph neural networks. *arXiv preprint arXiv:2205.02637*, 2022a.

Lino, M., Fotiadis, S., Bharath, A. A., and Cantwell, C. D. Multi-scale rotation-equivariant graph neural networks for unsteady eulerian fluid dynamics. *Physics of Fluids*, 34(8):087110, 2022b.

Liu, W., Yagoubi, M., and Schoenauer, M. Multi-resolution graph neural networks for pde approximation. In *International Conference on Artificial Neural Networks*, pp. 151–163. Springer, 2021.

Masci, J., Boscaini, D., Bronstein, M., and Vandergheynst, P. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pp. 37–45, 2015.

Mesquita, D., Souza, A., and Kaski, S. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.

Obiols-Sales, O., Vishnu, A., Malaya, N., and Chandramowliswharan, A. Cfdnet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM international conference on supercomputing*, pp. 1–12, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. W. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017a.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *Advances in neural information processing systems*, 30, 2017b.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P.

Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pp. 4470–4479. PMLR, 2018.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Strönisch, S., Sander, M., Meyer, M., and Knüpfer, A. Multi-gpu approach for training of graph ml models on large cfd meshes. In *AIAA SCITECH 2023 Forum*, pp. 1203, 2023.

Sun, L., Gao, H., Pan, S., and Wang, J.-X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Computer Methods in Applied Mechanics and Engineering*, 361:112732, 2020.

Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. Accelerating eulerian fluid simulation with convolutional networks. In *International Conference on Machine Learning*, pp. 3424–3433. PMLR, 2017.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Yu, Z., Xu, C., Meng, D., Hui, Z., Xiao, F., Liu, W., and Liu, J. Transitive distance clustering with k-means duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 987–994, 2014.

Zhang, Z., Bu, J., Ester, M., Zhang, J., Yao, C., Yu, Z., and Wang, C. Hierarchical graph pooling with structure learning. *arXiv preprint arXiv:1911.05954*, 2019.

A. Appendix

A.1. Dataset details

We adopt three existing test cases: Cylinder (Flow), Airfoil, and (Deforming) Plate from MESHGRAPHNETS. The Cylinder includes the transient incompressible flow field around a fixed cylinder at varying locations. The Airfoil includes the transient compressible flow field at varying Mach numbers around the airfoil with varying angles of attack (AOA). The Plate includes hyperelastic plates squeezed by moving obstacles. In addition to these three cases, our Font(INFLATINGFONT) case involves the quasi-static inflation of enclosed elastic surfaces (3D surface mesh) possibly with self-contact. We create the INFLATINGFONT cases using the open-source simulator (Fang et al., 2021), with the same material properties and inflation speed. The input geometries for INFLATINGFONT are $1,400$ 2×2-character matrices in Chinese. All the datasets are split into 1000 training, 200 validation, and 200 testing instances. In the following table, the second entries with superscript* in the average edge number column are for the contact edges:

Case	Ave # nodes	Ave # edges	Mesh type	Seed method	# Levels	# Steps
Cylinder	1886	5424	triangle, 2D	MinAve	7	600
Airfoil	5233	15449	triangle, 2D	MinAve	9	600
Plate	1271	$4611, 94^*$	tetrahedron, 3D	MinAve	6	400
Font	13177	$39481, 6716^*$	triangle, 3D	CloseCenter	6	100

Below we list the model configurations: 1) the offset inputs to prepend before the material edge processor e_{ij}^M, and e_{ij}^W, and 2) nodes p_i, as well as the nodal outputs q_i from the decoder for each experiment cases, where $\boldsymbol{X}$ and $\boldsymbol{x}$ stand for the material-space and world-space positions, $\boldsymbol{v}$ is the velocity, ρ is the density, P is the absolute pressure, and the dot $\dot{a} = a_{t+1} - a_t$ stands for temporal change for a variable a. All the variables involved are normalized to zero-mean and unit variance via pre-processing.

Case	Type	Offset inputs e_{ij}^M	Offset inputs e_{ij}^W	Inputs p_i	Outputs q_i
Cylinder	Eulerian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert$	NA	$\boldsymbol{v}_i, n_i$	$\dot{\boldsymbol{v}}_i$
Airfoil	Eulerian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert$	NA	$\rho_i, \boldsymbol{v}_i, n_i$	$\dot{\boldsymbol{v}}_i, \dot{\rho}_i, P_i$
Plate	Lagrangian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert, \boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\dot{\boldsymbol{x}}_i, n_i$	$\dot{\boldsymbol{x}}_i$
Font	Lagrangian	$\boldsymbol{X}_{ij}, \lvert\boldsymbol{X}_{ij}\rvert, \boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	$\boldsymbol{x}_{ij}, \lvert\boldsymbol{x}_{ij}\rvert$	n_i	$\dot{\boldsymbol{x}}_i$

As for time integration, Cylinder, Airfoil, and Plate inherited the first-order integration from MESHGRAPHNETS. For INFLATINGFONT, the first-order quasi-static integration (Fang et al., 2021) is used in the solver. Hence, we also adopt the first-order integration for INFLATINGFONT.

A.2. Additional Model details

A.2.1. BASIC MODULES AND ARCHITECTURES

The MLPs for the nodal encoder, the processor, and the nodal decoder are ReLU-activated two-hidden-layer MLPs with the hidden-layer and output size at 128, except for the nodal decoder whose output size matches the prediction $\boldsymbol{q}$. All MLPs have a residual connection. A LayerNorm normalizes all MLP outputs except for the nodal decoder.

A.2.2. BASELINE: MESHGRAPHNETS

Our MESHGRAPHNETS implementation uses the same MLPs as above but with an additional module: the edge encoder. Also, the edge latent is updated and carried over throughout the end of multiple MPs. We use 15 times MP for all cases to keep it consistent with MESHGRAPHNETS.

A.2.3. BASELINE: MS-GNN-GRID

Our re-implementation of MS-GNN-GRID uses the same MLPs as above but with four additional modules: the edge encoder at the finest level, the aggregation modules for nodes and edges at every level for the transitions, and the returning modules for nodes at every level. This method also requires assigning the regular grid nodes for each level. We assign these grid nodes by defining an initial grid resolution and an inflation rate between levels. As for the MP times at each level, we

follow Lino et al. (2022a) to use four at the top and bottom levels and two for the others.

Case	# Levels	Initial grid dx	dx inflation	Level-wise # MPs
Cylinder	4	[5e-2, 5e-2]	2	[4, 2, 2, 4]
Airfoil	4	[4.5, 4.5]	2	[4, 2, 2, 4]
Plate	4	[4e-3, 4e-3, 4e-3]	2	[4, 2, 2, 4]
Font	4	[1.5e-2, 1.5e-2, 1e-3]	2	[4, 2, 2, 4]

A.2.4. BASELINE: GRAPHUNETS

Our re-implementation of GRAPHUNETS uses the same number of levels as those of BSMS-GNN. Likewise, we make the following modifications to the original GRAPHUNETS : (1) We change the information passing from GCN to our message passing module for consistency and translational invariance. (2) GraphUNet was intended for tiny graphs (100 nodes) and used dense matrix multiplications. This design is not scalable as it can break the memory limit and slow down the training to take more than 30 days per epoch in our graph size (1500 to 15000 nodes). We thus optimize the operations such as matrix multiplication and aggregation with sparse implementations.

A.2.5. NOISE AND BATCH NUMBER

For each of these benchmarks, we generated Gaussian noise with a specific scale and added it to the original trajectory at the beginning of every epoch. The purpose of noise injection was to mimic the effects of noise generated by the model, thereby improving the model's ability to correct its output when fed with noisy inputs. Furthermore, we carefully tuned the batch size under smaller subset experiments for each method to achieve optimal convergence rates.

Case	Batch size				Noise scale
	BSMS-GNN	MS-GNN-GRID	MESHGRAPHNETS	GRAPHUNETS	
Cylinder	32	16	16	2	velocity: 2e-2
Airfoil	8	4	8	1	velocity: 2e-2, density: 1e1
Plate	8	2	2	1	pos: 3e-3
Font	2	1	1	1	pos: [5e-3, 5e-3, 3.33e-4]

A.3. Detailed results

We plot the detailed measurements in Table. 1 and Table. 2. All experiments are conducted using a single Nvidia RTX 3090.

A.4. Ablation study

A.4.1. TRANSITION METHOD

While exploring the non-parametric transition solutions, we started with no transition because our method is adopted directly from GUN (Gao & Ji, 2019). The no-transition strategy produces low enough 1-step RMSE and visually correct rollouts for INFLATINGFONT. However, in the global rollouts of CYLINDERFLOW and AIRFOIL cases, we observed stripe patterns (Figure. 11 (c), column **None**) where the stripes are aligned with the edges at the coarser levels (Figure. 11 (d)). We suspect that this error results from the fact that the unpooled nodes all have zero information before MP, making them indistinguishable from the processor modules and exaggerating the difference between pooled and unpooled nodes over rollouts.

The no-transition strategy resembles no interpolation during the super-resolution phase of CNN+UNet. Naturally, we then tried a single step of graph convolution (without activation) to resemble the interpolation in regular grids. However, this turns out to over-smooth the features (Figure. 11 (e), column **Graph Conv**), and the information propagation was smeared out except for the area near the generator (in this case, near the cylinder).

We believe the over-smoothing issue arises from the ignorance of the irregularity of the mesh. Unlike CNN, where the fine nodes regularly lie at the center of coarser grids, irregular meshes have varying topology and element sizes. The element sizes are almost always smaller near the interface for higher precision in simulations; hence an unweighted graph convolution can smear the finer information near the cylinder and their adjacent neighbors during returning. The natural choice to account for the irregularity is to include reasonable nodal weights (such as the size). In the end, we arrive at the

Table 1. **Detailed measurements** of our method, MS-GNN-GRID, MESHGRAPHNETS, and GRAPHUNETS. BSMS-GNN consistently generates stable and competitive global rollouts with the smallest training cost. BSMS-GNN is also lightweight and has the fastest inference time. It is also free from the large RMSE due to poor pooling on unseen geometries where the learnable pooling module of GRAPHUNETS suffers. The 2nd column of entries in Infer time is the speed up compared to the numerical solver. The 2nd column of entries in Training cost is the epoch for achieving the converged model.

Measurements	Case	Our's	(Lino et al., 2021)	(Pfaff et al., 2020)	(Gao & Ji, 2019)
Training time/step [ms]	Cylinder	**10.14**	15.36	19.29	16.20
	Airfoil	**18.82**	25.26	36.72	55.08
	Plate	**15.58**	49.65	49.15	31.88
	INFLATINGFONT	**45.96**	107.16	117.48	1,833.37
Infer time/step [ms]	Cylinder	6.75, 121x	**6.18, 133x**	14.50, 57x	24.30, 34x
	Airfoil	**8.64, 1275x**	20.40, 540x	24.20, 455x	33.60, 328x
	Plate	**14.01, 207x**	18.12, 160x	15.70, 184x	16.20, 179x
	INFLATINGFONT	**33.33, 210x**	41.66, 168x	82.35, 85x	629.33, 11x
Training cost [hrs], Final epoch	Cylinder	**21.41, 19**	35.84, 21	64.30, 30	76.15, 39
	Airfoil	**122.33, 39**	176.82, 42	275.40, 45	206.55, 37
	Plate	**56.07, 27**	125.78, 19	176.94, 27	127.50, 30
	INFLATINGFONT	**2.68E+01, 21**	5.66E+01, 19	6.20E+01, 19	NA
RMSE-1 [1e-2]	Cylinder	**2.04E-01**	2.20E-01	2.26E-01	8.09E-01
	Airfoil	2.88E+01	**2.68E+01**	4.35E+01	2.93E+01
	Plate	2.87E-02	2.20E-02	**1.98E-02**	2.03E-02
	INFLATINGFONT	**1.77E-02**	1.87E-02	1.95E-02	NA
RMSE-50 [1e-2]	Cylinder	**2.42**	2.74	4.39	1.87E+01
	Airfoil	**1.10E+03**	1.22E+03	1.66E+03	1.17E+03
	Plate	3.18E-02	**2.78E-02**	2.88E-02	5.19E-02
	INFLATINGFONT	**1.08E-01**	3.24E-01	1.78E-01	NA
RMSE-all [1e-2]	Cylinder	**8.37**	8.49	1.07E+01	1.65E+02
	Airfoil	**4.21E+03**	5.56E+03	6.95E+03	6.11E+03
	Plate	1.60E-01	**1.48E-01**	1.51E-01	5.46E-01
	INFLATINGFONT	**2.20E-01**	3.78E-01	3.65E-01	NA

solution proposed in Sec. 3.2 by utilizing the nodal weights during aggregation and recording the shares of contribution for later returning. Our transition method works consistently for all experiment cases and produces the lowest RMSE for global rollouts (Figure. 11 (b)).

Comparing to alternative transition methods Additionally, we compare our transition methods to two alternatives extracted from previous works: (1) calculating the edge weights (kernel) for the information flow using the inverse of its length (node position offset), which we refer to as **Pos-Kernel** (Liu et al., 2021); and (2) the level-wise learnable transition modules implemented by additional MP, which we refer to as **Learnable** (Fortunato et al., 2022).

In addition to the high RMSE of **None** and **Graph-Conv** shown in Figure. 11, we can also observe that: (1) the training/infer time and RAM consumption for all non-parametric transitions (including **None**) are similar, which supports the statement that our transition method is light-weighted. (2) **Learnable** transition can reach slightly higher accuracy but at the price of $\sim 70\%$ more training/infer time and RAM. As mentioned in Sec. 4.2, higher training RAM can limit the batch number and increase the frequency of data communication between CPU and GPU, slowing down the training process even further when the scale goes up. (3) **Pos-Kernel** results in a slightly higher RMSE compared to our method, making it a competitive alternative in production.

A.4.2. SENSITIVITY ANALYSIS ON SEEDING HEURISTICS

In this section, we investigated the impact of using different seeding heuristics during the training and testing phases on the sensitivity of a converged model. We deliberately altered the heuristics used on each benchmark and evaluated the RMSEs. The results showed that the inconsistency in seeding heuristics led to higher roll-outs compared to the results obtained when using consistent seeding, as shown in Table 4. Specifically, the roll-outs were 1.01x to 2.04x and 1.13x to 2.21x for random seeding and inverse seeding, respectively, with respect to consistent seeding. However, the RMSEs remained in the same magnitude, indicating that our method is not sensitive to the initial seeding. It should be also noted that this is not a practical issue, as the seeding can easily be kept consistent during different phases.

Table 2. **Memory footprint under multi-batches**, BSMS-GNN consistently cuts RAM consumption by approximately half in all cases in the training stage, and also has the smallest (except for DEFORMINGPLATE) inference RAM.

Case	Method	Training RAM (GBs) with different Batch #						Infer RAM (GBs)
		2	4	8	16	32	64	
CYLINDERFLOW	**Our's**	**2.41**	**2.92**	**4.37**	**6.06**	**11.4**	**22.27**	**1.92**
	(Lino et al., 2021)	2.79	3.60	5.31	8.56	15.10	-	1.97
	(Pfaff et al., 2020)	3.25	4.46	6.91	11.84	21.60	-	1.94
	(Gao & Ji, 2019)	23.33	-	-	-	-	-	2.18
AIRFOIL	**Our's**	**3.66**	**5.46**	**8.88**	**15.70**	-	-	**2.02**
	(Lino et al., 2021)	4.18	6.25	10.65	19.25	-	-	**2.02**
	(Pfaff et al., 2020)	5.53	8.90	16.08	-	-	-	2.06
	(Gao & Ji, 2019)	-	-	-	-	-	-	2.67
DEFORMINGPLATE	**Our's**	**2.36**	**2.87**	**3.85**	**5.78**	**9.28**	**16.85**	1.95
	(Lino et al., 2021)	3.41	4.81	7.75	13.20	-	-	2.00
	(Pfaff et al., 2020)	3.10	4.29	6.59	11.49	20.80	-	**1.93**
	(Gao & Ji, 2019)	-	-	-	-	-	-	2.18
INFLATINGFONT	**Our's**	**6.28**	**10.80**	-	-	-	-	**2.23**
	(Lino et al., 2021)	10.87	19.79	-	-	-	-	2.45
	(Pfaff et al., 2020)	12.48	23.39	-	-	-	-	2.28
	(Gao & Ji, 2019)	-	-	-	-	-	-	4.51

Table 3. **Detailed measurements** of different transition methods. Ours and **Pos-Kernel** are the only two non-parametric transitions that are light-weighted and produce reliable rollouts compared to the expensive **Learnable** transition.

Measurements	**Ours**	**None**	**Graph-Conv**	**Pos-Kernel**	**Learnable**
Training time/step [ms]	10.14	**9.30**	10.07	10.06	17.75
Infer time/step [ms]	6.75	**5.70**	6.46	6.90	11.28
Training RAM [GBs]	**11.041**	**11.041**	**11.041**	**11.041**	18.033
Infer RAM [GBs]	**1.923**	**1.923**	**1.923**	**1.923**	1.931
RMSE-1 [1e-2]	2.85E-01	**1.49E-01**	3.41E-01	6.38E-01	4.70E-01
RMSE-50 [1e-2]	1.43E+01	2.05E+02	2.40E+02	1.77E+01	**1.35E+01**
RMSE-all [1e-2]	1.68E+01	2.59E+02	5.51E+02	2.01E+01	**1.57E+01**

A.5. Details for scaling analysis on INFLATINGFONT

We generate the downscale and the upscale version of INFLATINGFONT with different average node numbers for the initial geometry, and then use the same settings to simulate the sequence. As reported in Fortunato et al. (2022), the low-resolution model suffers from converging to very small RMSE; hence we loosen the termination criteria by enlarging the target RMSE relative to the average edge length to prevent convergence failures. Similarly, the noise injection is also adjusted to be relative to the average edge length. Moreover, with a smaller number of nodes, the number of levels required to achieve the same bottom resolution also reduces. We make the corresponding adjustments to the levels of our model d_1 and that of the MS-GNN-GRID d_2. The adjustments are plotted below.

A.6. The Proof of conservation of contact edges

With Bi-stride pooling, our pooling conserves all the contact edges under the enhancement in Eq. 1. We assume the graph is undirected and unweighted, such that the adjacent matrix is a boolean matrix.

Formally speaking, given any contact edge (i, j) at level l (i.e. $\boldsymbol{A}_l^C[i, j] = 1$) and a Bi-stride pooling P which pools nodes $\mathcal{I}$, there exists a contact edge (i', j') that remains in the coarser level (i.e. $\boldsymbol{A'}_{l+1}^C[i', j'] = 1, i', j' \in \mathcal{I}$) and $i/i', j/j'$ are connected (i.e. $\boldsymbol{A}_l[i, i'] = \boldsymbol{A}_l[j, j'] = 1$). There are only four scenarios concerning the pooling nodes $\mathcal{I}$ and the contact edge nodes i, j, under which the assertion always holds:

1. Both i, j are pooled, i.e. $i, j \in \mathcal{I}$. Obviously $\boldsymbol{A'}_{l+1}^C[i', j'] = 1$ by letting $i' = i, j' = j$.
2. Only i is pooled, $i \in \mathcal{I}, j \notin \mathcal{I}$. Since we use Bi-stride pooling, j can either be the seed at level 0 (Bi-stride can select either even or odd levels) that directly connects to all nodes at level 1, or must have at least one direct connection from the previous level. I.e, at least one neighbor of j in the adjacent level is pooled, we let it be j': $\boldsymbol{A}_l[j, j'] = 1, j' \in \mathcal{I}$.

Table 4. **Sensitivity analysis of seeding heuristics on model performance.** The "Random" heuristic refers to choosing a random seed for every cluster, while "Inverse w.r.t. Train" refers to choosing the inverse seeding heuristic compared to that used during the training phase. For example, if the MinAve heuristic was used during training, the CloseCenter heuristic was chosen during the testing phase.

Seeding @ Test	Ratio in RMSE	Cylinder	Airfoil	Plate	Font
Random	1-step	2.06	14.04	2.95	1.15
	50-step	1.06	2.16	5.21	1.04
	Rollout	1.01	2.04	1.53	1.04
Inverse w.r.t. Train	1-step	1.96	12.77	3.14	1.15
	50-step	1.07	1.76	7.52	1.02
	Rollout	1.13	2.21	1.46	1.06

Table 5. **The adjustment for multi-scale parameters, the target RMSE and noise injection for scaling analysis.**

# Nodes	d_1	d_2	Initial grid dx	Target RMSE	Noise in pos
5k	4	2	[6e-2 6e-2 4e-3]	1.73e-4	[8.5e-3, 8.5e-3, 5.7e-4]
15k	6	4	[1.5e-2 1.5e-2 1e-3]	1e-4	[5e-3, 5e-3, 3.33e-4]
30k	7	5	[7.5e-3 7.5e-3 5e-4]	1e-4	[3.5e-3, 3.5e-3, 2.4e-4]
45k	7	5	[7.5e-3 7.5e-3 5e-4]	1e-4	[2.9e-3, 2.9e-3, 1.9e-4]

Then $\boldsymbol{A}_l^C \boldsymbol{A}_l[i,j'] \geq \boldsymbol{A}_l^C[i,j] * \boldsymbol{A}_l[j,j'] = 1$, and $\boldsymbol{A}_l(\boldsymbol{A}_l^C \boldsymbol{A}_l)[i,j'] \geq \boldsymbol{A}_l[i,i] * (\boldsymbol{A}_l^C \boldsymbol{A}_l)[i,j'] = 1$. Let $i' = i$, then $\boldsymbol{A'}_{l+1}^C[i',j'] = 1$.

3. Only j is pooled, $i \notin \mathcal{I}, j \in \mathcal{I}$. Similarly we have at least one i' such that: $\boldsymbol{A}_l[i',i] = 1, i' \in \mathcal{I}$. Then $\boldsymbol{A}_l \boldsymbol{A}_l^C[i',j] \geq \boldsymbol{A}_l[i',i] * \boldsymbol{A}_l^C[i,j] = 1$, and $(\boldsymbol{A}_l \boldsymbol{A}_l^C)\boldsymbol{A}_l[i',j] \geq (\boldsymbol{A}_l \boldsymbol{A}_l^C)[i',j] * \boldsymbol{A}_l[j,j] = 1$. Let $j' = j$, then $\boldsymbol{A'}_{l+1}^C[i',j'] = 1$.
4. None of i, j is pooled, $i, j \notin \mathcal{I}$. We select one direct pooled neighbor for i, j, respectively, that $\boldsymbol{A}_l[i',i] = \boldsymbol{A}_l[j,j'] = 1, i', j' \in \mathcal{I}$. Then $\boldsymbol{A}_l \boldsymbol{A}_l^C[i',j] \geq \boldsymbol{A}_l[i',i] * \boldsymbol{A}_l^C[i,j] = 1$, and $(\boldsymbol{A}_l \boldsymbol{A}_l^C)\boldsymbol{A}_l[i',j'] \geq (\boldsymbol{A}_l \boldsymbol{A}_l^C)[i',j] * \boldsymbol{A}_l[j,j'] = 1$.

A.7. Algorithms for the seeding heuristics

Here we elaborate our two seeding heuristics for the bi-stride pooling at every level: picking the seed that 1) is closest to the center of a cluster (CloseCenter), and 2) with the minimum average geodesic distance to its neighbors (MinAve). The complexity for MinAve is $\boldsymbol{O}(N^2)$ as we need to conduct BFS for every node to find the one with the minimum average distance to neighbors. In our experiments, the quadratic cost of MinAve is tolerable for all cases but INFLATINGFONT.

Algorithm 1 MinAve: seeding by minimum average geodesic distance to neighbors

Input: Unweighted, Bi-directional graph, $\boldsymbol{G} = (N, E)$
{List of seeds in each clusters L_s}
$L_c \leftarrow$ DetermineCluster($\boldsymbol{G}$)
$L_s \leftarrow \emptyset$
{BFS(s) returns the list of distances to all other neighbors from s}
{if unreachable, the distance is set to infinity}
$D \leftarrow$ {BFS(s) for s in N}
for idx in L_c **do**
  $D_c \leftarrow D$[idx, idx]
  $\bar{D}_c \leftarrow$ average(D_c, dim $= 1$)
  $s \leftarrow$ idx[argmin($\bar{D}_c$)]
  L_s.append(s)
end for
output L_s

For INFLATINGFONT, the largest mesh has around 47K nodes, and the time for pre-processing with MinAve becomes intolerable. We switch to CloseCenter with the linear complexity.

For both heuristics, we search seeds in a per-cluster fashion to avoid the information from other clusters that could pollute

Algorithm 2 CloseCenter: seeding by minimum distance to the center of cluster

Input: Unweighted, Bi-directional graph, $\boldsymbol{G} = (N, E)$; Positions of the nodes, X
{List of seeds in each clusters L_s}
$L_c \leftarrow$ DetermineCluster($\boldsymbol{G}$)
$L_s \leftarrow \emptyset$
for idx **in** L_c **do**
  $\bar{X} \leftarrow$ average(X[idx], dim $= 0$)
  $\Delta X \leftarrow X - \bar{X}$
  $D \leftarrow ||\Delta X||_2$
  $s \leftarrow$ idx[argmin(D)]
  L_s.append(s)
end for
output L_s

the search result. For example, when determining the center of an isolated part of the input geometry, the positions of nodes from other clusters could pollute this process. The determination of clusters given in a graph is elaborated below.

Algorithm 3 DetermineCluster

Input: Unweighted, Bi-directional graph, $\boldsymbol{G} = (N, E)$
{R stands for remaining nodes that are not inside any cluster}
$R \leftarrow N$
$L_c \leftarrow \emptyset$
while $R \neq \emptyset$ **do**
  $s \leftarrow R$.pop( )
  if $|R| = 0$ **then**
    L_C.append($\{c\}$)
  else
    $D \leftarrow$ BFS(s)
    $C \leftarrow \emptyset$
    $R^* \leftarrow \emptyset$
    for n in R **do**
      if $D[n] = \infty$ **then**
        R^*.append(n)
      else
        C.append(n)
      end if
    end for
    L_C.append(C)
    $R \leftarrow R^*$
  end if
end while
output L_c

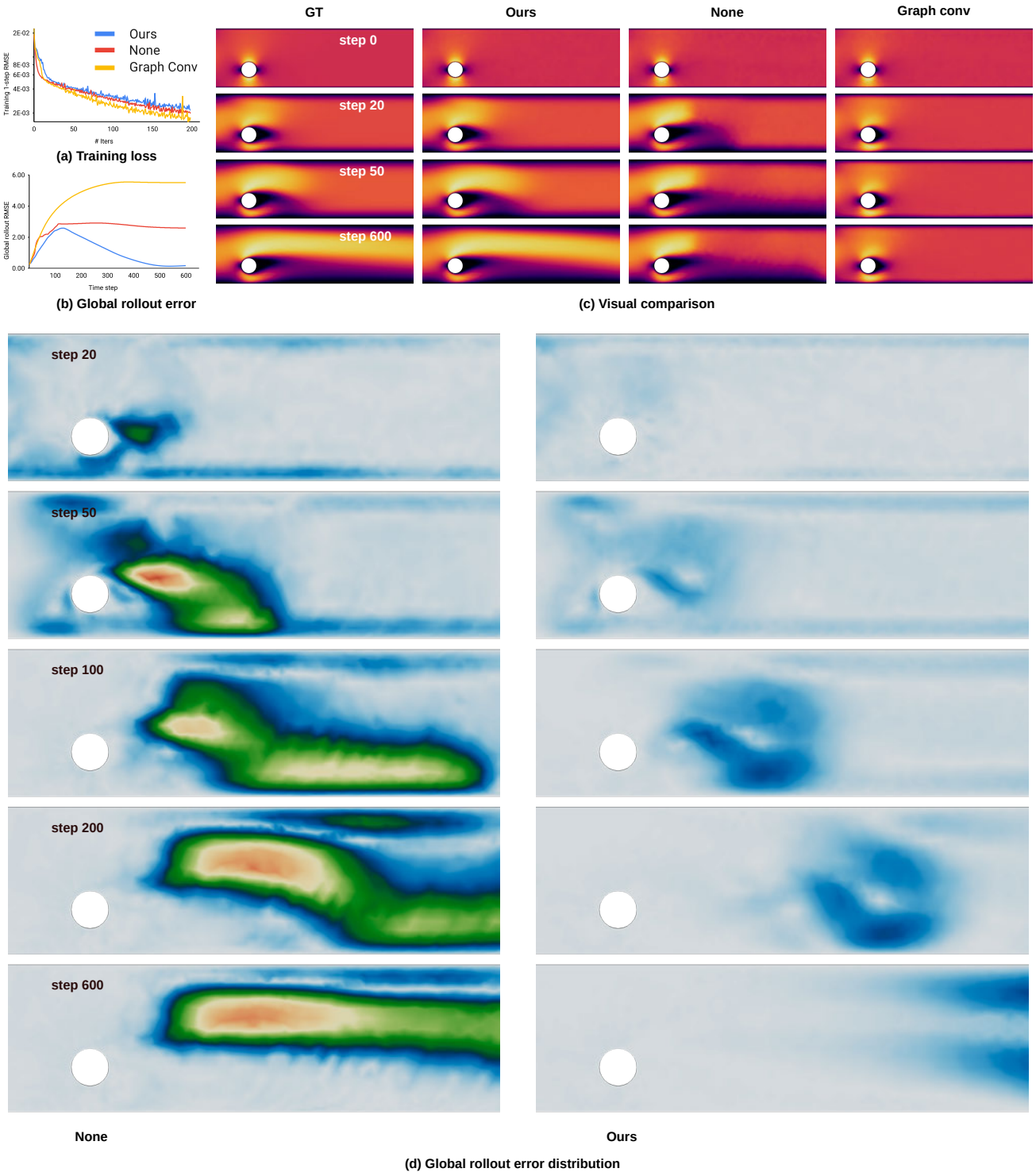


Figure 11. (a) All three transition methods can reach the target training RMSE given 200 iterations. (b) However, our weighted graph aggregration+returning has the strongest resistance to the noise during the rollout. (c) The visual comparisons show that no transition produces mosaic-like patterns, while the graph convolution transition smeared out the information and ceased propagating downstream. (d) The global rollout error distribution of no transition (**Left**) shows the edge of the mosaic patterns look similar to the simulation mesh; The error of our transition (**Right**) travels with the generated vortices downstream and leaves the domain after step 200, which explains the RMSE drop in (b).