# Sampling-based Distributed Training with Message Passing Neural Network

**Anonymous Authors**[1]

## Abstract

In this study, we introduce a domain-decomposition-based distributed training and inference approach for message-passing neural networks (MPNN). Our objective is to address the challenge of scaling edge-based graph neural networks as the number of nodes increases. Through our distributed training approach, coupled with Nyström-approximation sampling techniques, we present a scalable graph neural network, referred to as DS-MPNN (D and S standing for distributed and sampled, respectively), capable of scaling up to $O(10^5)$ nodes. We validate our sampling and distributed training approach on two cases: (a) a Darcy flow dataset and (b) steady RANS simulations of 2-D airfoils, providing comparisons with both single-GPU implementation and node-based graph convolution networks (GCNs). The DS-MPNN model demonstrates comparable accuracy to single-GPU implementation, can accommodate a significantly larger number of nodes compared to the single-GPU variant (S-MPNN), and significantly outperforms the node-based GCN.

## 1. Introduction and Related Work

The application of machine learning algorithms to build surrogates for partial differential equations (PDEs) has seen a significant push in recent years (Li et al., 2020b; Raissi et al., 2019; Lu et al., 2021; Pfaff et al., 2020). While the majority of these ML models for PDEs surrogate modeling are CNN-based (Zhu et al., 2019; Ranade et al., 2022; Ren et al., 2022), the introduction of graph neural networks (Kipf & Welling, 2017) has made graph-based modeling increasingly common. Traditional numerical methods for real-world applications predominantly rely on mesh-based unstructured representations of the computational domains (Mavriplis, 1997) owing to the ease of (a) discretization of complex domains and (b) extension to adaptive mesh refinement strategies to capture multi-scale physical phenomenon like turbulence, boundary layers in fluid dynamics, etc. These mesh-based unstructured formulations can be seamlessly formulated on graphs. Furthermore, a graph-based representation eliminates the need for interpolating data onto a structured Euclidean grid – a step inherent in CNNs, further reducing cost and error. In recent years, these graph-based models have also been applied to crucial domains like weather modeling (Lam et al., 2022), biomedical science (Fout et al., 2017), computational chemistry (Gilmer et al., 2017; Duvenaud et al., 2015), etc.

From the perspective of physical system dynamics modeling, we can divide the GNN-based methods into two broad categories: (a) node-based approaches and (b) edge-based approaches. The node-based approaches, including graph convolution networks (GCNs) (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), and graph attention networks (GATs) (Veličković et al., 2018), are easier to scale with an increasing number of elements in an unstructured grid setting. However, they lack the mechanism of message-passing among nodes which is crucial to the modeling of spatial relationships between the nodes in complex PDEs. Hence, the edge-based graph methods dominate the paradigm of graph-based modeling for physical systems, such as Graph Network-based Simulators (Sanchez-Gonzalez et al., 2020), and its evolution MeshGraphNets (MGN) (Pfaff et al., 2020). Pfaff et al. (2020) indicate that these edge-based graph methods outperform node-based GCNs in accuracy and stability, and can be applied to a wide variety of simulations. Another approach to edge-based methods for PDE modeling is message-passing neural networks (Gilmer et al., 2017; Li et al., 2020a) where the input and the output solution space are invariant to mesh grids and independent of discretization (Li et al., 2020a). Message-passing neural networks (MPNN) (Simonovsky & Komodakis, 2017) condition weights around the vertices based on edge attributes and have properties to learn the PDE characteristics from a sparsely sampled field.

While edge-based methods are accurate in modeling physical systems, their memory requirement scales with the number of edges in a mesh. Hence these methods do not

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

scale for modeling realistic physical problems where the number of nodes can be substantially higher ($O(10^5 - 10^6)$). Bonnet et al. (2022b) attempted to solve this issue by randomly sampling a set of nodes and edges based on Nyström approximation (Li et al., 2020a) and memory constraints of the GPUs available, thus creating a sparse domain. The sampling method yields similar errors as modeling complete graphs. Despite many attempts, edge-based methods, especially MPNNs, remain memory-intensive, constrained by the current limits of single GPU memory. Strönisch et al. (2023) proposed an approach implementing multi-GPU utilization for MeshGraphNets (Pfaff et al., 2020). In their methodology, the domain is partitioned across various GPUs, enabling inter-node communication. However, this communication is limited to node-based features within the latent space, excluding interaction among edge-based features. The errors observed in the multi-GPU setup proposed by (Strönisch et al., 2023) were notably higher compared to their single GPU training. This highlights the imperative need to devise methodologies for training edge-based graph models on multi-GPU setups without compromising accuracy.

We present a sampling-based distributed MPNN (DS-MPNN) that involves partitioning the computational domain (or graph) across multiple GPUs, facilitating the scalability of edge-based MPNN to a large number of nodes. Here, 'distributed' implies partitioned spatial domains that are put on different GPUs which can be on same or different machines. This scalability holds significant practical value for the scientific community. We combine the idea of domain decomposition and message-passing among GPUs to enable the training of message-passing neural networks for physical systems with no or minimal loss in accuracy.

Our two key contributions are:

1. We devise a method of training and inference on multiple GPUs for MPNN with no or minimal loss in accuracy.

2. We demonstrate the scaling and acceleration of MPNN training for graphs with DS-MPNN to $O(10^5)$ nodes through the combination of multi-GPU parallelization and node-sampling techniques

## 2. Model Description

The message-passing graph model used in this work uses a convolutional graph neural network with edge conditioning (Simonovsky & Komodakis, 2017) together with random sampling (Li et al., 2020a; Bonnet et al., 2022b).

### 2.1. Graph construction

Consider a set of nodes $V \in \mathbb{R}^d$ on the domain $\Omega \in \mathbb{R}^d$. A subset $\Omega_s \subset \Omega$ is formed by randomly sampling nodes, with $|V_s| = s$. Graph kernels $\mathcal{G}_i = (V_i, \mathcal{E}_i)$ are constructed using these sampled nodes $v_i \in V_s$ as centers within a radius $\rho$. Edge connections $e_{ij} \in \mathcal{E}_i$ are established between node $j$ within a specified radius and the respective central nodes $i$. For edges $|\mathcal{E}_i| > n_e$, $n_e$ edges are further randomly sampled from $|\mathcal{E}_i|$. The representation of this graph kernel construction is shown in figure 1. Each node $v_i$ and the connecting edges $e_{ij}$ are assigned attributes or labels, $v_i^l$ and $e_{ij}^l$, respectively. In PDE modeling, the nodal and edge attributes can represent the initial functional space $\mathcal{F}_{inp}$ or the initial conditions. Edge attributes $e_{ij}^l$ in this work are derived by calculating the relative difference between node coordinates and attributes $(v^l)$ of nodes $i$ and $j$.

At the core of this model is the concept of using edge-conditioned convolution (Simonovsky & Komodakis, 2017) to calculate the node attribute $v_i^l$ in one message-passing step by summing the product of weights $\mathcal{K}_\phi$, based on the edge attributes $e_{ij}$ and neighboring node attributes $v_j$ in $\mathcal{G}_i$, giving

$$v_i{}^l = \frac{1}{|\mathcal{E}_i|} \sum_{j=1}^{j \leq n_e} \mathcal{K}_\phi(e_{ij}^{l-1}; \theta) v_j{}^{l-1} + b. \tag{1}$$

Here, $\theta, b$ are neural network parameters. $l$ corresponds to the message-passing step among 'radius hops' $h$. $e_{ij}^l$ is updated based on new values of $v_i^l$. Edge conditioning in the convolutional process renders the model adept at handling non-uniform grid points or graph structures, typical in physics simulations involving boundary layers and shocks, where grid density varies significantly across the field.

### 2.2. Model algorithm

The current model, adapted from(Bonnet et al., 2022b), is composed of an encoder ($\mathcal{N}_e$), decoder ($\mathcal{N}_d$) and message-passing network ($\mathcal{K}_\phi$). This encoder transforms the initial node attributes, $v_i^{l=0}$, where $l = 0$ indicates the initial or lower-level state of these attributes, into a latent representation. The latent space representation of node attributes at the $l$-th message-passing step is represented as $v_{L,i}^l$, with $L$ signifying the latent representation of the node attribute $v_i^l$. The transformation process is governed by the following equations:

$$v_{L,i}^{l=0} = \mathcal{N}_e(v_i^{l=0}, \theta), \tag{i}$$

$$v_{L,i}^{l+1} = \frac{1}{|\mathcal{E}_i|} \sum \mathcal{K}_\phi(e_{ij}^l; \theta) v_{L,j}^l + b, \tag{ii}$$

$$v_i^{l+1} = \mathcal{N}_d(v_{L,i}^{l+1}, \theta), \quad \text{(iii)}$$

$$e_{ij}^{(l+1)} \leftarrow v_i^{l+1} - v_j^{l+1}. \quad \text{(iv)}$$

The process iterates through equations (ii) to (iv) for a total of $l = 0, h-1$ message-passing steps, representing $h$ radius hops in the graph, to update the node and edge attributes. The objective of the neural network denoted as $\mathcal{N}(\mathcal{N}_e, \mathcal{N}_d, \mathcal{K}_\phi)$, is to learn the mapping from the initial function space, $\mathcal{F}_{\text{inp}}$, to the final function space $\mathcal{F}_{\text{out}}$, giving:

$$\mathcal{N} : \mathcal{F}_{\text{inp}} \rightarrow \mathcal{F}_{\text{out}}. \quad (2)$$

This framework facilitates the application of graph neural networks in solving PDEs by iteratively updating and transforming node and edge attributes within the graph structure.

## 3. Methodology

We now turn to DS-MPNN, our framework devised to train edge-based graphs on many GPU systems. DS-MPNN incorporates communication strategies tailored for MPNN-based graph methodologies across multiple GPUs. In this framework, prior to the formulation of a graph kernel $\mathcal{G} = (V_s, \mathcal{E})$, the computational domain $\Omega$ is partitioned into distinct subdomains and each subdomain is allocated to a GPU. Each GPU is allocated a distinct subdomain $\Omega_r \subset \Omega$. This arrangement entails dividing $\Omega$ into $n_{\text{procs}}$ (total number of GPUs available for parallelization) subdomains $\Omega_r$, with each subdomain featuring an extended overlap of length $l$. In all our runs, unless explicitly mentioned, we set $l = r$ to ensure that nodes at the edges of any given spatial partition have complete kernels This extended overlap is pivotal for ensuring comprehensive kernel construction at the interior edges of the domain, thereby circumventing the issues of incomplete kernel formation that can result in discontinuities in the predicted solutions. This approach is depicted in figure 1

In the framework described in section 2.2, each domain, accompanied by its kernels, is allocated to distinct GPUs for a series of $h$ hops. The methodology facilitates inter-GPU communication of the latent space node attributes $v_{i,L}^l$ through overlapping regions. As depicted in figure 2, the overlap area of a given domain is updated from the neighboring domains' interiors. Concurrently, the decoder's output in the physical space, denoted as $v_i^l$, updates the edge attributes correspondingly.

The computational graph accumulates gradients during the $h$ radius hops that are computed in relation to the total loss function $\mathcal{L}$ across the entire computational domain. This computation encompasses a summation of the domains' interior points across all GPUs. Following this, an aggregation of the gradients from each GPU is performed, leading to a synchronous update of the neural network parameters across all the GPUs - , utilizing the aggregated gradient. For interested readers, the detailed DS-MPNN training mechanism is provided in the Appendix.

During testing, the algorithm divides the domains into smaller, randomly selected sub-domains $\Omega_s$ along with its overlapping regions. These are sequentially fed into the trained model. They are reassembled in post-processing to form the original output dimension $\Omega \in \mathbb{R}^d$. The inference time of these surrogate models is orders of magnitude smaller than the time required to solve the PDEs, as we will show in the subsequent sections.

We investigate a range of PDEs for both structured and unstructured meshes, incorporating a diverse array of mesh sizes. The core of our study involves a detailed comparative analysis of two distinct computational implementations: a single-GPU implementation S-MPNN and the multi-GPU DS-MPNN. The multi-GPU implementation leverages up to 4 GPUs DS-MPNN4. This scalability is not limited to the four-GPU configuration tested; our methodology is designed to be flexible with respect to GPU count, allowing for expansion beyond the tested range. Such adaptability is essential for handling a variety of computational demands and hardware configurations.

In the multi-GPU setup, domains are equally partitioned based on their coordinates, and a distributed communication package from PyTorch facilitates inter-node communication (Paszke et al., 2019). The neural network architecture employed in this study consists of a 3-layer encoder and a 3-layer decoder, augmented by a convolution kernel. The network, with approximately 700k parameters, is utilized consistently across all experiments involving DS-MPNN and the baseline Graph Convolution Networks (GCN) This uniformity ensures parameter consistency across different models, and all models are implemented using PyTorch geometric package (Fey & Lenssen, 2019). GCN here uses 6 hidden layers with a size of 378. Post-encoding, the latent space representation for all experiments apart from parametric studies maintains a consistent dimensionality of 32 attributes. Optimization is conducted using the Adam optimizer, complemented by the OneCycleLR scheduler (Smith & Topin, 2019), to enhance learning efficiency and effectiveness.

## 4. Experiments

### 4.1. Darcy Flow (Structured Data)

We explore an example utilizing a structured grid, specifically focusing on 2-D Darcy flow that describes the fluid flow through a porous medium. 2-D Darcy flow equation

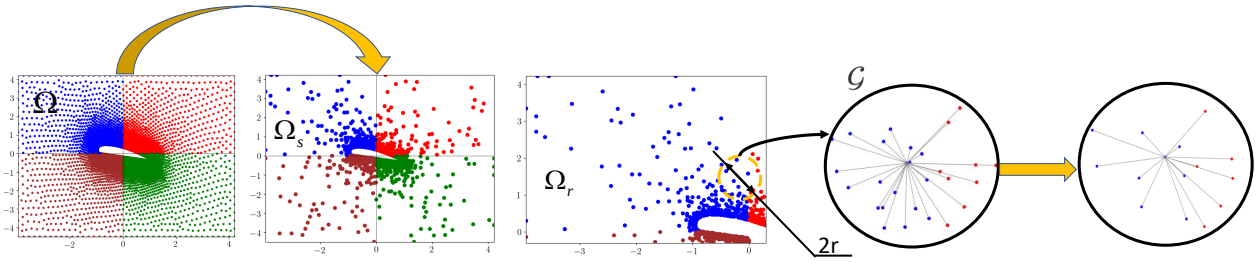



*Figure 1.* Graph kernel $\mathcal{G}$ construction for an individual node in the low-fidelity AirfRANS dataset. Yellow arrows indicate node and edge sampling; various colors denote distinct computational domains on separate GPUs; and $\Omega_r$ represents the distributed domain, encompassing an overlap from neighboring domains with a length of kernel radius $r$.

on the unit box is a second order elliptic PDE given as:

$$-\nabla\dot{(}a(x)\nabla u(x)) = f(x) \quad \forall x \in (0,1)^2 \tag{3}$$

$$u(x) = 0 \quad \forall x \in \partial(0,1)^2. \tag{4}$$

Here, $a$ and $f$ are the spatially varying diffusion coefficients and the forcing field, respectively. $u$ is the solution field on the 2-D domain. This example parallels the approach used in the Graph Neural Operator (GNO) as discussed by (Li et al., 2020a), adapted for a single GPU setup with a notable distinction: we update the edge attributes following each radius hop or message-passing step, and both edges and nodes are subjected to sampling. The experiment investigates the mapping $a \rightarrow u$.

The experimental setup comprises 1024 training samples, each initialized differently, and 30 test samples. Each sample is structured on a grid of $421 \times 421$, with sampled nodes $|V_s| = 421$. The attributes of each node consist of two-dimensional grid coordinates ($x_i$ and $y_i$) and the field values of $a_i$, resulting in node and edge attributes with dimensions $\mathbb{R}^3$. The graph kernel employed has a radius of 0.2, and the maximum number of edges allowed per node is set at $n_e = 64$. The model is trained over 200 epochs, with each domain in the test case being randomly sampled five times. Evaluations were performed using both a single GPU framework and a multi-GPU configuration, wherein the computational domain was partitioned into four equivalent segments. The number of radius hops employed in this context, denoted by $h$, was set to eight. The test error was quantified using root mean squared error (RMSE), as defined in equation 5:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5}$$

*Table 1.* Comparison of DS-MPNN (on 4 GPUs) with single-GPU implementation (S-MPNN) and GCN.

| Method | GCN | S-MPNN | DS-MPNN4 |
|---|---|---|---|
| Test Error | 3.3E-4 | 7.44E-6 | 7.60E-6 |

Table 1 compares the performance of GCN and MPNN models on the Darcy flow dataset, using both single and 4-GPU setups. GCN under-performs relative to MPNN, highlighting the superiority of edge-based methods in PDE surrogate modeling, even for simpler datasets. This discrepancy is largely due to GCN's omission of edge representation, a key feature in node-based models. Furthermore, the DS-MPNN model shows comparable results on both single and 4-GPU configurations. The minor variance in test error between these two setups might result from the segmentation of the computational graph across different domains. Figure 3 corroborates this observation, illustrating similar predictions from both DS-MPNN and S-MPNN configurations, while GCN lags behind as it excessively smooths the solution and fails to predict the roughness at the boundaries.

For the Darcy flow dataset, similar to the work by Sanchez-Gonzalez et al. (2020) and Pfaff et al. (2020), we study the effect of changing crucial hyperparameters for a graph-based model on the performance of our model evaluated using $L_1$ loss. These key hyperparameters include (a) total number of hops ($h$), (b) sampling radius ($r$), (c) number of edges ($n_e$), and (d) number of nodes ($s$). Table 2 shows that increasing these four hyperparameters helps improve the model's accuracy, albeit showing saturation in accuracy beyond a certain threshold. This observation is in agreement with previous works on edge-based techniques by Pfaff et al. (2020) and Sanchez-Gonzalez et al. (2020). The key point to note here is that for problems with higher complexity, increasing any of these four hyperparameters will result in an increased GPU memory consumption, underlying the need for a parallelized training paradigm for graph-based

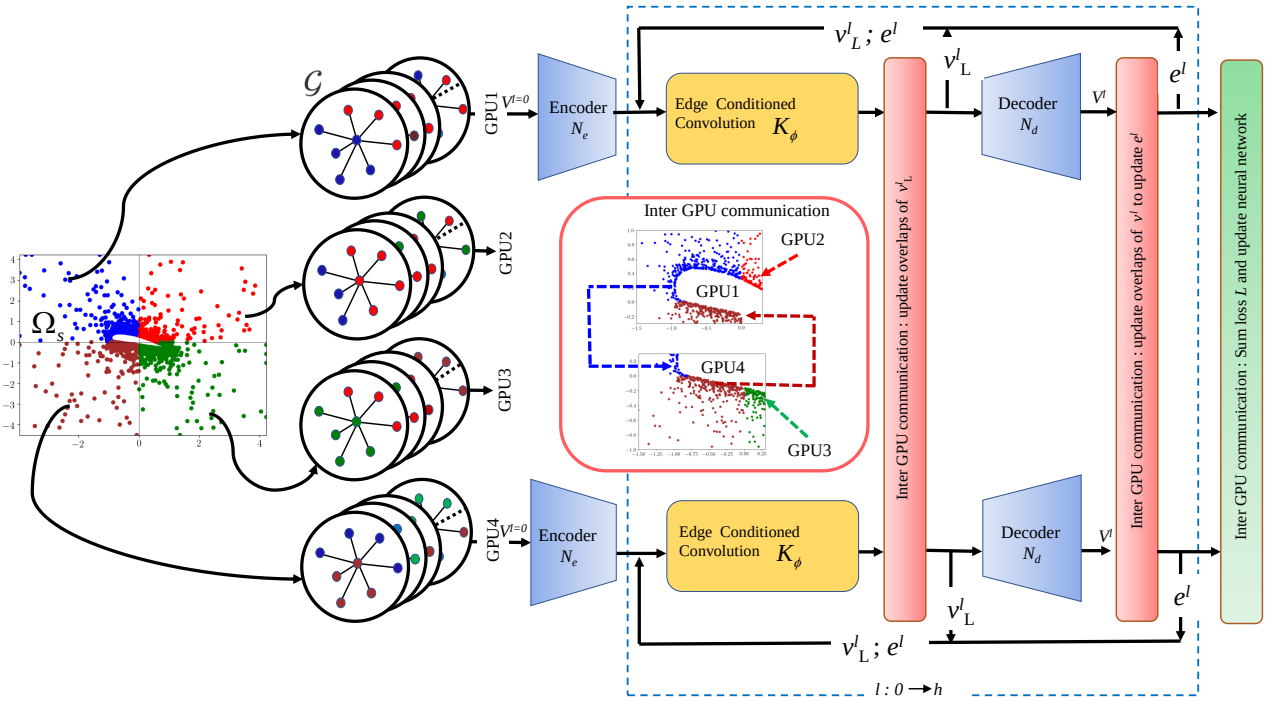



*Figure 2.* Methodology representing graph kernels $\mathcal{G}$ from separate distributed domains of an AirfRANS dataset being processed on four individual GPUs. Inter GPU communication represents the exchange of information between the neighboring domains through overlap regions during each hop. After $h$ radius hops, loss $L$ is calculated on the interior points and aggregated over all GPUs to update the neural network parameters.

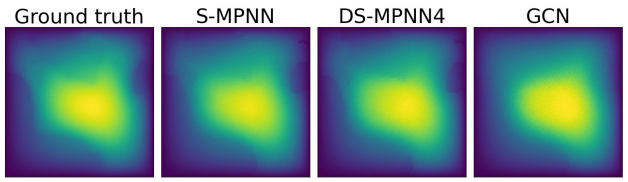



*Figure 3.* Comparison of model performance between single GPU (S-MPNN) – RMSE=$5.3 \times 10^{-6}$, four-GPU (DS-MPNN4) – RMSE=$4.71 \times 10^{-6}$, and GCN – RMSE=$4 \times 10^{-3}$, for a test sample of Darcy flow dataset.

models like ours.

### 4.2. AirfRANS (Unstructured Data)

The AirfRANS dataset corresponds to the solution of incompressible steady-state Reynolds Averaged Navier Stokes (RANS) equations in 2-D, given by

$$\mathbf{U}.\nabla\mathbf{U} = -\frac{1}{\rho}\nabla P + (\nu + \nu_t)\nabla^2\mathbf{U}, \tag{6}$$

$$\nabla.\mathbf{U} = 0. \tag{7}$$

Here, $\mathbf{U} = [U_x, U_y]$ represents the mean 2-D velocity components, and $P$ denotes the mean pressure. The variable $\nu_t$ signifies the kinematic turbulent viscosity, which is spatially varying, while $\nu$ denotes the constant kinematic viscosity. In this dataset, the learning process involves mapping the airfoil shape and angle of attack to $[U_x, U_y, P, \nu_t]$, as further elaborated in the subsequent sections.

The AirfRANS datasets comprise two distinct unstructured sets, each varying in complexity and scale. The first dataset, as described by (Bonnet et al., 2022b), contains approximately 15,000 nodes per sample, while the second, a higher-fidelity version detailed in (Bonnet et al., 2022a) under "AirfRANS," has approximately 175,000 nodes per sample. The DS-MPNN model efficiently processes both datasets, under various Reynolds numbers and attack angles, highlighting its versatility in diverse and complex aerodynamic simulations.

*Table 2.* Test error based on $L_1$ loss demonstrating the impact of varying parameters: hops ($h$), sampling radius ($r$), maximum number of edges ($n_e$), and nodes sampled ($n_s$).

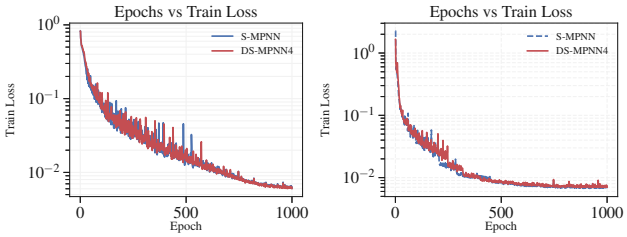
| Radius hops ($h$) | $h = 2$ | $h = 4$ | $h = 12$ |
|---|---|---|---|
| Test Error | 0.029 | 0.024 | 0.023 |
| Radius ($r$) | $r = 0.05$ | $r = 0.12$ | $r = 0.2$ |
| Test Error | 0.1 | 0.032 | 0.024 |
| Sampled edges ($n_e$) | $n_e = 32$ | $n_e = 64$ | $n_e = 128$ |
| Test Error | 0.029 | 0.024 | 0.024 |
| Sampled nodes ($s$) | $s = 420$ | $s = 840$ | $s = 1260$ |
| Test Error | 0.0244 | 0.0244 | 0.0235 |

#### 4.2.1. Low-fidelity AirfRANS dataset

The low-fidelity dataset includes a diverse array of airfoils subjected to varying angles of attack, ranging between $(-0.3°,0.3°)$. This dataset also incorporates a spectrum of Reynolds numbers, specifically from $10^6$ to $5 \times 10^6$. For this investigation, the test error is quantified RMSE in equation 5, while mean square error is used as the criterion for optimization during training. The current experimental run comprises 180 training samples and 29 test samples. Each sample in the dataset is characterized by input node attributes, which include grid coordinates, inlet velocity, pressure, and the distance function between the surface and the node, denoted as $v_i^{l=0} \in \mathbb{R}^6$. The edge attributes, denoted as $e_{ij}^{(l)} \in \mathbb{R}^8$, include velocity and pressure attributes. These attributes are updated after each radius hop, reflecting changes based on the model output $y_i \in \mathbb{R}^4$, which comprises the $x - y$ velocity components, pressure, and turbulent viscosity. For this specific case, the number of nodes sampled is $s = 1600$ and the number of edges $n_e = 64$. The training process spanned over 1000 epochs, with each test dataset being sampled 10 times.

Figure 4(a) illustrates the training loss trajectories of S-MPNN and DS-MPNN4. The convergence patterns, indicated by the overlapping trend in training loss, suggest that DS-MPNN4 achieves a level of training convergence comparable to that of S-MPNN. Furthermore, upon adopting an alternative scheduler strategy—specifically, reducing the learning rate once the training loss plateaus, a similar convergence pattern is observed in figure 4(b). This consistency underscores the training robustness of the DS-MPNN model.

Table 3 shows the test RMSE for GCN, single-GPU MPNN (S-MPNN), and DS-MPNN implemented on 2 and 4 GPUs. The losses are very similar between the single GPU and DS-MPNN runs, underscoring the validity of our algorithm and its effectiveness in handling unstructured grid representation distributed over multiple GPUs. Similar to the Darcy flow results 4.1, GCN performs worse than MPNNs. However, the accuracy of GCN deteriorates further for the AirfRANS dataset compared to the structured Darcy dataset, highlighting the challenges of using node-based methods for complex PDE modeling.




**(a)** Scheduler: One cycle learning rate

**(b)** Scheduler: Dropping learning rate on plateau

*Figure 4.* Training loss vs. epochs for different schedulers.

*Table 3.* Comparison of DS-MPNN (on 2 and 4 GPUs) with single-GPU implementation (S-MPNN) and GCN for low-fidelity AirfRANS dataset.

| Method | GCN | S-MPNN | DS-MPNN2 | DS-MPNN4 |
|---|---|---|---|---|
| RMSE | 0.347 | 0.094 | 0.097 | 0.096 |

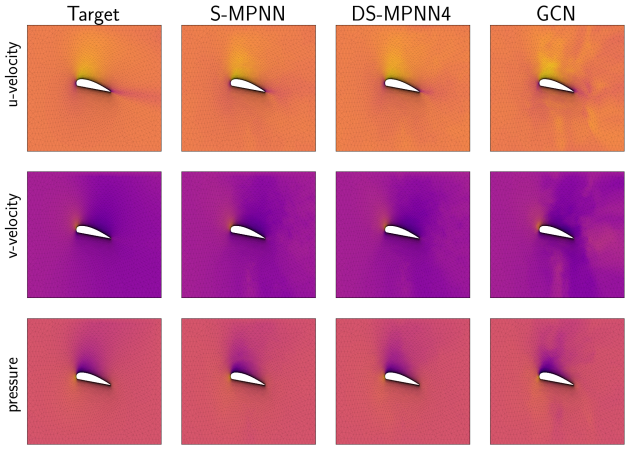



*Figure 5.* Comparison among GCN, S-MPNN, DS-MPNN2 and DS-MPNN4 for a test sample from low-fidelity AirfRANS dataset.

Figure 5 presents a comparative analysis of various models concerning the two velocity components and pressure. It is observed that the performance of the models utilizing a single GPU (S-MPNN) and four GPUs (DS-MPNN4) is comparable. However, the GCN model exhibits inferior predictions, characterized by incorrect flow features around the airfoil across all fields.

*Table 4.* Variation of test RMSE as a function of overlap length ($l$). Kernel radius is set fixed at $r = 0.3$.

| | $l = 0$ | $l = 0.15$ | $l = 0.3$ | $l = 0.6$ |
|---|---|---|---|---|
| Test RMSE | 0.124 | 0.096 | 0.097 | 0.097 |

Table 4 presents the test RMSE errors for various extended overlap lengths ($l$) for a fixed kernel radius $r = 0.3$. As the overlap radius increases from 0 to 0.15, the error decreases significantly. However, beyond 0.15, the variations are minimal. As previously mentioned in section 3, we set $l = r$ in all our experiments unless explicitly mentioned. This is done to ensure that nodes at the edges of any given spatial partition have complete kernels. It should be noted that the kernel radius $r$ is an use-case specific hyperparameter.

#### 4.2.2. High-fidelity AirfRANS dataset

For the high-fidelity mesh analysis, we employ the standardized AirfRANS dataset as detailed by Bonnet et al. (Bonnet et al., 2022a). Our focus in this study is to show the effectiveness of DS-MPNN across various network hyperparameters rather than to achieve the best accuracy. Hence, we use a specific subset of the original AirfRANS dataset, termed as the 'scarce' dataset by the authors, that consists of 180 training samples and 20 test samples. The dataset encompasses a Reynolds number variation between 2 million and 6 million, and the angle of attack ranges from $-5°$ to $15°$. The input node attributes bear resemblance to the low-fidelity dataset 4.2.1 but with an expanded dimension of unit surface outward-pointing normal for the node $v_i^{l=0} \in \mathbb{R}^7$, and the edge attributes are characterized by $e_{ij}^{(l)} \in \mathbb{R}^{10}$. The model output is denoted as $y_i \in \mathbb{R}^4$. Table 5 presents the test error, quantified as the Root Mean Square Error (RMSE) loss, where the dataset is sampled once. These results indicate that the DS-MPNN framework maintains its efficacy even with a substantial increase in the node count in the dataset, consistently performing well across different GPU configurations. In contrast, the GCN exhibits poorer performance under similar conditions. Additionally, an experiment involving a network configuration without inter-GPU communication, but with distinct domains distributed across GPUs, resulted in divergent training outcomes which are different from (Strönisch et al., 2023) where no communication model worked better. This underscores the critical role of communication between GPUs for maintaining model stability and performance. Table 5 shows that using the DS-MPNN framework with more GPUs increases training and inference speeds. This is because each GPU deals with fewer edges and nodes than it would if the entire domain was on a single GPU, which speeds up model execution. The exploration of scalability and the associated communication overhead for DS-MPNN is comprehensively addressed in Section 5.

*Table 5.* Comparison among GCN, S-MPNN and DS-MPNN4 for AirfRANS dataset.

| Method | GCN | S-MPNN | DS-MPNN4 |
|---|---|---|---|
| Test Error | 0.32 | 0.17 | 0.18 |
| Train Time (s/epoch) | 68 | 197 | 113 |
| Inference Time (s) | 7.05 | 17.7 | 11.7 |

Training cases with 3.5 million parameter models and different hyperparameters are showcased when trained on 4 GPUs for 400 epochs. On 3.5 million parameters, a single GPU has a memory overflow on Nvidia RTX6000, when trained on $s = 6000$ nodes, $n_e = 64$ sampled edges, and $h = 4$ radius hops, but as we increase the number of nodes, which is possible only through DS-MPNN, we see that the RMSE test accuracy increases in table 6. Similarly, table 7, demonstrates the need for DS-MPNN to enable the sampling of a higher number of edges.

*Table 6.* Effect of increasing nodes on the performance for DS-MPNN4.

| Nodes | Hops | Edges | RMSE Loss |
|---|---|---|---|
| 3000 | 4 | 64 | 0.27 |
| 6000 | 4 | 64 | 0.21 |
| 10000 | 4 | 64 | 0.20 |

*Table 7.* Effect of increasing edges on the performance for DS-MPNN4.

| Nodes | Hops | Edges | RMSE Loss |
|---|---|---|---|
| 6000 | 4 | 16 | 0.26 |
| 6000 | 4 | 32 | 0.24 |
| 6000 | 4 | 64 | 0.21 |

## 5. Ablation Studies on Scalability and Communication Overhead

The subsequent ablation experiments were conducted to evaluate the scalability and communication burden of DS-MPNN4 in comparison with the standard single GPU framework, S-MPNN. These studies were carried out using the 'scarce' high-fidelity AirfRANS dataset, which consists of 180 training samples. For these experiments, the kernel radius and the number of nodes were predetermined at $r = 0.05$ and 6000, respectively. Unless specifically mentioned, the baseline overlap length is set to the kernel radius.

Figure 6, clearly demonstrate the associated cost of communication for DS-MPNN4. The training and inference costs for communication are higher compared to scenarios with no communication across various non-zero lengths of the overlap region in the 4 sub-domains. Nevertheless, despite the communication overhead, DS-MPNN4 is consistently better than S-MPNN in terms of training and inference duration. We also increase the sampled node count $s$ to ascertain

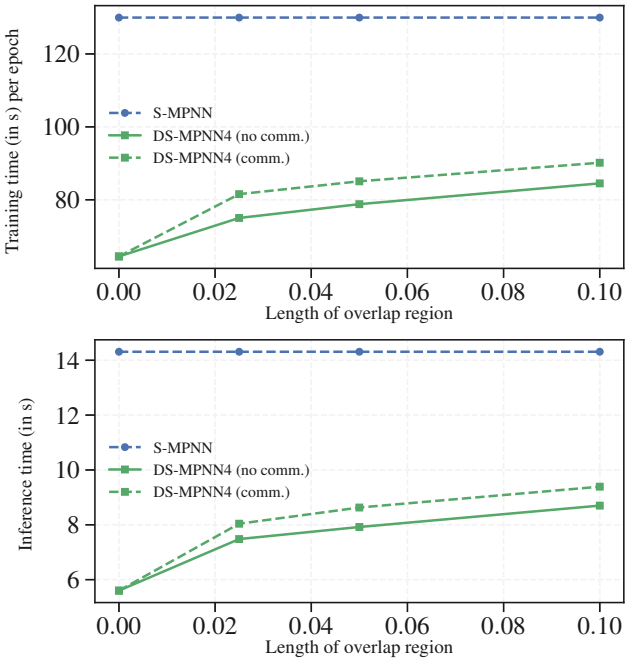



*Figure 6.* Training time per epoch and inference time as a function of the length of the overlap region ($l$).

the impact of scalability and communication overhead. This increase in node samples becomes imperative for larger numerical grids. Figure 7 shows the increase in the training speed of DS-MPNN4 compared to S-MPNN. The increase in speed is attributed to the reduction in the time required for graph kernel $\mathcal{G}$ generation within DS-MPNN4, as a consequence of decreased number of edge formations required across each GPU domain. Correspondingly, the percentage of training time spent on communication decreases as the node count increases, because, while the aggregate data for communication grows with the node count, the substantial bandwidth of GPUs maintains a constant communication time.

Moreover, scalability appears in figure 8 that shows a decrease in training and inference time with increasing number of domains from one (S-MPNN) to four (DS-MPNN4) GPU setup. This is consistent with our previous observations.

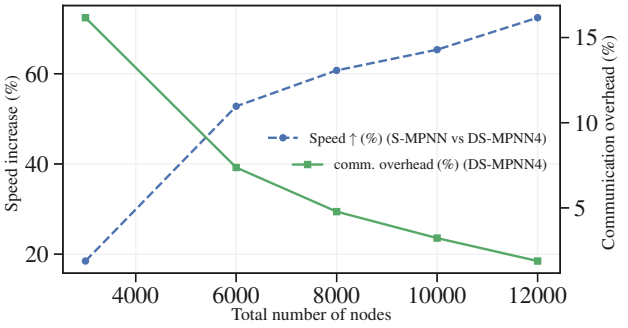



*Figure 7.* Performance of DS-MPNN4 against S-MPNN with node increase.

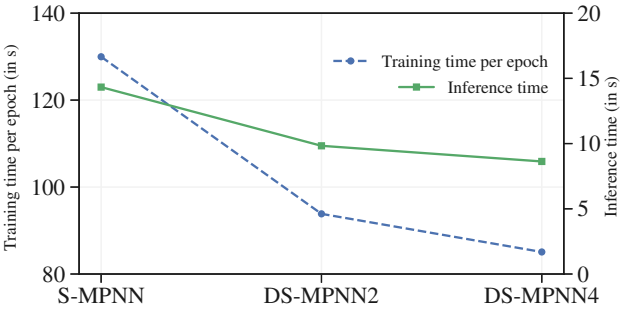



*Figure 8.* Training and inference time as a function of number of distributed domains.

## 6. Conclusions

We address the issue of scaling edge-based graph methods to larger numbers of nodes by introducing distributed training for message-passing neural networks (MPNNs). When combined with node sampling techniques, this distributed approach allows us to scale a larger number of nodes with losing no or minimal loss in accuracy as compared to the single-GPU implementation while achieving a considerable decrease in training and inference times. We also show comparisons to the graph convolution networks (GCNs) and establish that edge-based methods like MPNNs outperform node-based methods like GCNs - This work opens up new avenues for the use of edge-based graph neural networks in problems of practical interest where the number of nodes can be impractically large, $\sim O(10^5 - 10^6)$, for single-GPU memory limits. The deployment of DS-MPNN is complex, notably in managing unstructured grids and their adjacent elements. However, the domain of distributed graph training is advancing swiftly. Notably, libraries tailored for these challenges have been introduced into the latest iteration of PyTorch Geometric (Fey & Lenssen, 2019), version 2.5.0.

## 7. Impact statements

This paper presents work whose goal is to advance the field of graph neural networks applied to physical systems. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

## References


Bonnet, F., Mazari, J. A., Cinnella, P., and Gallinari, P. AirfRANS: High fidelity computational fluid dynamics dataset for approximating reynolds-averaged navier–stokes solutions. In Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2022a. URL https://arxiv.org/abs/2212.07564.

Bonnet, F., Mazari, J. A., Munzer, T., Yser, P., and Gallinari, P. An extensible benchmarking graph-mesh dataset for studying steady-state incompressible navier-stokes equations. In ICLR 2022 Workshop on Geometrical and Topological Representation Learning, 2022b. URL https://openreview.net/forum?id=rqUUi4-kpeq.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. Advances in neural information processing systems, 28, 2015.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428, 2019.

Fout, A., Byrd, J., Shariat, B., and Ben-Hur, A. Protein interface prediction using graph convolutional networks. Advances in neural information processing systems, 30, 2017.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In International conference on machine learning, pp. 1263–1272. PMLR, 2017.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. 2017.

Lam, R., Sanchez-Gonzalez, A., Willson, M., Wirnsberger, P., Fortunato, M., Alet, F., Ravuri, S., Ewalds, T., Eaton-Rosen, Z., Hu, W., et al. Graphcast: Learning skillful medium-range global weather forecasting. arXiv preprint arXiv:2212.12794, 2022.

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, 2020a.

Li, Z., Kovachki, N. B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., Anandkumar, A., et al. Fourier neural operator for parametric partial differential equations. In International Conference on Learning Representations, 2020b.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature machine intelligence, 3(3):218–229, 2021.

Mavriplis, D. Unstructured grid techniques. Annual Review of Fluid Mechanics, 29(1):473–514, 1997.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 32, 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.

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.

Ranade, R., Hill, C., Ghule, L., and Pathak, J. A composable machine-learning approach for steady-state simulations on high-resolution grids. Advances in Neural Information Processing Systems, 35:17386–17401, 2022.

Ren, P., Rao, C., Liu, Y., Wang, J.-X., and Sun, H. Phycrnet: Physics-informed convolutional-recurrent network for solving spatiotemporal pdes. Computer Methods in Applied Mechanics and Engineering, 389:114399, 2022.

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.

Simonovsky, M. and Komodakis, N. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 3693–3702, 2017.

Smith, L. N. and Topin, N. Super-convergence: Very fast training of neural networks using large learning rates. In Artificial intelligence and machine learning for multi-domain operations applications, volume 11006, pp. 369–386. SPIE, 2019.

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.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In International Conference on Learning Representations, 2018.

Zhu, Y., Zabaras, N., Koutsourelakis, P.-S., and Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. Journal of Computational Physics, 394: 56–81, 2019.

# A. Training Algorithm

**Algorithm 1:** Training of DS-MPNN.

**Input:** Node encoder: $\mathcal{N}_e$, Node decoder: $\mathcal{N}_d$, Graph kernel: $\mathcal{K}_\phi$; Domain: $\Omega$, Sampled-domain: $\Omega_s$, Sub-domain: $\Omega_r$; grid coordinates and initial conditions: $v^{l=0}$., Target values: $Y$, No of training samples: $N_{train}$, Number of epochs: $E$, Number of distributed domains: $n_{proc}$.

**for** $j = 1$ **to** $N_{train}$ **do**
- $\Omega_s \leftarrow rand(\Omega)$ ▷ Sample nodes from domain $\Omega$
- $\Omega_r \leftarrow \Omega_s$ ▷ Decompose domain based on available gpus
- $v^{l=0} \leftarrow \Omega_r$ ▷ Grid coordinates and initial values to create nodes
- $Y \leftarrow \Omega_r$ ▷ Get targets in the sub-domain
- $edge_I \leftarrow edgeindexcreator(\Omega_r, r)$ ▷ Create edge index for kernel radius r
- $edge_i \leftarrow rand(edge_I)$ ▷ Sample edge index
- $e^l \leftarrow (dv^{l=0})$ ▷ Create edge attributes ($dv = v_i - v_j$)
- $trainloader \leftarrow tuple(v^{l=0}, Y, edge_i, e^l)$

**for** epoch $= 1$ **to** $E$ **do**

▷ This loop run across $n_{proc}$ GPUs

- $v^{l=0}, Y, edge_i, e^l \leftarrow trainloader$
- $i_b \leftarrow getindex(v^{l=0})$ ▷ Tracking index $i_b$ for unstructured grids
- **for** $k = 1$ **to** $h$ **do**
  - **if** k = 1 **then**
    - $v_L^l \leftarrow \mathcal{N}_e\left(v^{l=0}, w\right)$ ▷ Node encoding
  - $v_{res} \leftarrow v_L^l$ $v_L^l \leftarrow \mathcal{K}_\phi\left(v_L^l, edge_i, e^l, w\right)$ ▷ Graph convolution
  - $v_L^l \leftarrow v_L^l + v_{res}$ $v_i^l \leftarrow \mathcal{N}_d\left(v_L^l, w\right)$ ▷ Node decoding
  - $v_L^l \leftarrow Comm(i_b, \Omega, v_L^l)$ ▷ Communicate node values in latent space
  - $v^l \leftarrow Comm(i_b, \Omega, v^l)$ ▷ Communicate output values
  - $e^l \leftarrow dv^l$ ▷ Update edge attributes with new values
- $\mathcal{L}_{local}$ = MSE $\left(v^l{}_{i_{(int)}}, Y_{i_{(int)}}\right)$ ▷ MSE on interior points
- $\mathcal{L} = \sum_{i=1}^{n_{proc}} \mathcal{L}_{local}$ ▷ Getting loss from all procs
- $\nabla w_{local} \leftarrow$ Backprop($\mathcal{L}$) ▷ Get local gradients
- $\nabla w = \sum_{i=1}^{n_{proc}} \nabla w_{local}$ ▷ Sum local gradients
- $w \leftarrow w - \eta \nabla w$ ▷ Update parameters

**Output:** Trained DS-MPNN model $\mathcal{N}(\mathcal{N}e, \mathcal{N}d, \mathcal{K}\phi)$