Multi-resolution Graph Neural Networks for PDE Approximation

Wenzhuo Liu[1,2(✉)], Mouadh Yagoubi[1], and Marc Schoenauer[2]

[1] IRT SystemX, Orsay, France
{wenzhuo.liu,mouadh.yagoubi}@irt-systemx.fr
[2] INRIA TAU, LISN CNRS and U. Paris-Saclay, ORsay, France

Abstract. Deep Learning algorithms have recently received a growing interest to learn from examples of existing solutions and some accurate approximations of the solution of complex physical problems, in particular relying on Graph Neural Networks applied on a mesh of the domain at hand. On the other hand, state-of-the-art deep approaches of image processing use different resolutions to better handle the different scales of the images, thanks to pooling and up-scaling operations. But no such operators can be easily defined for Graph Convolutional Neural Networks (GCNN). This paper defines such operators based on meshes of different granularities. Multi-resolution GCNNs can then be defined. We propose the MGMI approach, as well as an architecture based on the famed U-Net. These approaches are experimentally validated on a diffusion problem, compared with projected CNN approach and the experiments witness their efficiency, as well as their generalization capabilities.

Keywords: Graph Neural Networks · PDEs · Multi-resolution GNNs

1 Introduction

Numerical simulation techniques are widely used to predict the behavior of complex systems in the real world. Partial differential equations (PDEs) are typically used to model the physical problem, and their solutions are approximated by numerical techniques such as the finite element (FEM) and finite volume (FVM) methods. Such typical approaches discretize the domain into *meshes* and compute the approximated values of the quantities of interest on each node of the mesh. These approaches can predict the physical behavior of the systems accurately, however, at a high computational cost for complex systems.

In recent years, there has been a rapid rise in the use of machine learning algorithms to solve problems from different domains where the conventional mathematical models are hard to build or expensive to compute. Deep neural networks have become the most popular approach due to the ability to solve complex tasks such as computer vision and natural language processing, outperforming existing algorithms when large-scale data are available. The great success of deep learning has naturally encouraged researchers to investigate its

I. Farkaš et al. (Eds.): ICANN 2021, LNCS 12893, pp. 151–163, 2021.
https://doi.org/10.1007/978-3-030-86365-4_13

use of learning numerical solutions of PDEs for the purpose of reducing simulation time. Indeed, with the wide applications of numerical simulations, a high volume of data has been accumulated. Many scientists have proposed data-driven methods, making full use of such data [3,5,11,19]. The first approaches [5,19] applied convolutional neural networks(CNNs) to construct deep learning models, due to their tremendous successes for image analysis, thanks to their capacity to capture spatial features. However, when it comes to PDEs simulations, complex geometries and the construction of unstructured meshes are inevitable. In such cases, data generated from numerical simulations is not similar to an image with regular pixels. A visible solution is to embed the complex domain into a regular rectangle domain and use interpolation, so that CNNs can be applied directly. However, in the real world, physical problems have complex geometric domains, and such *CNN approach* can lead to a significant interpolation error, in particular on the boundary of an actual domain. Furthermore, such an approach can hardly take into account the necessary mesh refinements that are frequently needed for a good approximation of the solution of the PDE at hand.

Graph Neural Networks (GNNs) are ML methods that handle data that live on graphs. Convolutional GNNs aim at reproducing the locality properties of CNNs, but one drawback is the lack of a recognized approach to down-sampling (aka pooling) and up-sampling that allow GNNs to extract local features.

In such context, the overall goal of the present work is to learn from examples a model for the numerical solution of a PDE using GNNs and to preserve locality, such that solving the same PDE with different source or on different domains can be done at a much lower computational cost than with the classical numerical approaches (e.g., FEM) while giving a good approximation of the solution. To this end, the contributions of this work are twofold. First, we propose generic up- and down-sampling procedures for Convolutional GNNs taking advantage of meshes of different granularities; from thereon, inspired by the multi-grid methods in the numerical field [6], we introduce two such architectures for GNNs in the context of PDE simulations: the Multi-Grid U-Net (MG-UNet), based on the famous U-Net [17] proposed for image segmentation, and the novel Multi-Grid Multi-Input (MGMI). Second, we validate experimentally these multi-grid approaches against the *CNN approaches* (based on projections on a regular grid).

The paper is organized as follows: Sect. 2 briefly surveys the state-of-the-art in terms of Machine Learning approaches to PDE solving, as well as the use of GNNs in such context. Section 3 introduces, based on a hierarchy of meshes on the domain of the PDE, the multi-grid architectures MGMI and MG-UNet. Section 4 describes the experiments (solving a nonlinear diffusion equation on different types of domains), while Sect. 5 discusses the results that validate the proposed approach in the case of irregular domains.

2 Related Works

2.1 Machine Learning for PDEs

Using machine learning algorithms to solve PDEs has received special attention in the last few years. Current numerical solutions on PDEs are inefficient

for problems with high dimensions or on complex geometry. A major difficulty for such problems is meshing. On the one hand, forming a mesh is costly for complex geometric problems; on the other hand, it becomes infeasible in high dimensional space. Some scientists proposed mesh-free methods based on unsupervised learning to avoid mesh construction. These algorithms train a deep neural network to approximate PDE solution by satisfying the differential operator, initial conditions and boundary conditions for a specific PDE. By entering a variable x_0, the network will predict the value $u(x_0)$. [4] used fully connected layers to approximate the solution on complex geometry. [16,18] discussed the possibility to solve high dimensional problems by neural networks. The authors of [18] proved that the neural network converges to the PDE as the number of hidden units increases. However, the proposed algorithms has been applied on only one physical problem, and every slightly change on the PDE may require to re-investigate the proposed architecture.

Other approaches are based on data-driven methods. Similar to our work, the model learned from mesh data simulated by numerical solvers (e.g.: FEM, FVM) predicts solutions for unseen inputs. [19] construct a convolutional model to approximate electromagnetic problems by solving Poisson's equation on a squared domain. Training data is generated by finite differential solver, making it in Euclidean space, which ensures the feasibility of applying CNN layers. A U-Net model is proposed by [5] to solve Reynolds Averaged Navier-Stokes (RANS) flow problems on airfoil shapes. The generated data on the unstructured mesh is first projected on structured grids as images before training. Comparing with traditional methods, the CNN models can reduce computational time. Since these models apply CNN layers on structured grids, they are less adaptable for problems with complex domains, which we will discuss in this paper.

In recent years, many attempts have been made to construct a deep learning model that can be applied directly on mesh data instead of projection into structured grids. [11] discussed fluid flow fields problems on different irregular geometries. It considers CFD data as a set of points (called point clouds) and applies the PointNet [14] architecture specially designed for such data type. [3] combines graph neural networks with traditional CFD solver (run on a coarse mesh) to accelerate fluid flow prediction on a much finer mesh. Although this combination allows improving prediction on new situations from the same family with the studied problems (i.e., without any change in the geometry), the generalization to new problems with different geometries/meshes is not confirmed. Unlike these methods, that apply GNNs directly on a fin mesh, we proposed a hierarchical structure aiming at extracting both global and local features from mesh data.

2.2 Graph Neural Networks

An increasing number of studies have focused on data from non-Euclidean space such as graphs, meshes, and manifolds where CNN is no longer suited. The graph neural networks (GNNs) generalize deep learning models on non-Euclidean space. Inspired by the great success of CNN on images, there are plenty of studies searching for a method to define convolutional operators on non-Euclidean data.

The first spectral-based convolution operator is proposed by [7]. It defines the operator on graph Laplacian spectrum space. After graph Fourier transformation, the convolution can be defined as the multiplier on Fourier space which enables the construction of a graphical convolutional layer. Since then, several works [9,12] improved spectral-based graph networks. In the meantime, some scientists attempt to construct the convolution on the spatial space. These approaches, such as [2,20], aggregate the feature information from its neighbors. The MoNet layer [13], used in this paper, is also a spatial based method where edge features are trained to decide aggregation weights.

There are several kinds of research made to solve problems with a set of points(point clouds). The PointNet [14] is the first neural network for point clouds. The basic idea is that each point feature is encoded and then aggregated to a global vector by a symmetric function. PointNet++ [15] improved the PointNet by introducing hierarchical structures to capture local features.

3 Multi-grid Graph Neural Networks

3.1 Graph Convolutional Neural Networks

Rationale. The Finite Element Method (FEM) is based on a variational formulation of the PDE and an approximation space for the solution described by basis vectors attached to the mesh at hand (e.g., a given basis vector is 1 at a given node, and 0 on all other nodes). Applying the variational formulation to each basis vector, in turn, gives a system of equations whose unknowns are the coordinates of the solution $(x_1, \ldots, x_N)$ on the chosen basis, and in which the equation for unknown x_i only involves neighbors x_j of node i in the mesh. If the PDE is linear (e.g., the linear Poisson equation), this system is a linear system that is usually solved numerically using some iterative methods due to its size. For instance, the Jacobi iterative method computes a sequence of vectors that ultimately converges to the solution of the linear system, by

$$x_i^{k+1} = \frac{1}{A_{ii}}(b_i - \sum_j A_{ij} x_j^k)$$

If the PDE is not linear, the system to be solved to compute $(x_1, \ldots, x_N)$ is nonlinear, and it is usually approximated by a sequence of linear systems that are solved sequentially. The whole process can be considered as using Jacobi iterative method multiple times with different A_{ij} and b_i. And this is exactly what Graph Convolutional Neural Networks like MoNet [13] do.

The MoNet Architecture. Consider a weighted graph $G = (N, E, \mathcal{V}, \mathcal{E})$, where N is the number of nodes, E the set of edges, $\mathcal{V} \in \mathbb{R}^{N \times F}$ are the node features (to each of the N nodes, an F-dimensional feature vector is attached), and $\mathcal{E} \in \mathbb{R}^{E \times D}$ the edge features (D-dimensional vectors). Let $x_i \in \mathbb{R}^F$ be the feature vector of node i, and $e_{ij} \in \mathbb{R}^D$ be the feature of the edge $i \to j$ defining

the set of neighbours $N(i)$ of node i. The basic idea of MoNet is to define a trainable function w that computes an edge weight w_{ij} from the edge feature e_{ij}. MoNet then defines the convolutional operator on node i as:

$$\mathbf{x}'_i = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} \frac{1}{K} \sum_{k=1}^{K} \mathbf{w}_k(\mathbf{e}_{ij}) \odot \mathbf{\Theta}_k \mathbf{x}_j$$

where $\mathbf{x_j} \in \mathbb{R}^N$ represents the feature on node j, $\mathbf{K}$ is the user-defined kernel size, $\mathbf{\Theta_k} \in \mathbb{R}^{M \times N}$ stands for the trainable matrix applying a linear transformation on the input data, $\odot$ is the element-wise product, and $w_k, k = 1, \ldots, K$ are trainable edge weights. Following [13], we chose Gaussian kernels defined as:

$$w_k(e_{ij}) = exp(-\frac{1}{2}(e_{ij} - \mu_k)^T \Sigma_k^{-1}(e_{ij} - \mu_k))$$

Both μ_k and Σ_k are trainable variables representing the mean vector and covariance matrix of the Gaussian kernel.

MoNets for PDEs: Node and Edge Features. Given a PDE defined on some domain Ω (see Sect. 4) and a mesh of its domain of definition, this work is concerned with training a MoNet-like Neural Network to approximate the numerical solution of the PDE on the mesh.

A first node feature is the right-hand side of the PDE, defined on domain Ω, that will be represented by its values on the nodes. Another node feature g is used to describe whether the node is on the boundary ($g = 1$) or not ($g = 0$).

Finally, the 2-dimensional edge features are defined as $e_{ij} = p_j - p_i$, where p_k is the coordinates of node k. This ensures the translation-invariance of the approach.

3.2 Multi-grid Approaches

Many variants of multi-grid algorithms have been proposed in the context of numerical simulations (see, e.g., Chap. 3 in [6]): the main idea is to compute approximate solutions to the problem at hand on meshes of different resolution. The steps on coarse meshes are fast and help to unveil global features of the solution, while fine meshes refine the solutions, removing unwanted spatial oscillations. In order to extend this notion to the GNN framework, we assume in the following having a hierarchy of meshes $\mathcal{M}_i$ of the same domain, of increasing complexity (# nodes).

As a matter of fact, the idea of multi-grid has already been used in the neural network framework, in the context of image analysis and is based on pooling and upsampling layers that merge or expand rectangle patches of the image. Among well-known examples are the Autoencoders, and the famed U-Net architecture [17] which adds to the reduction/reconstruction structure some "horizontal" connection between downstream and upstream layers of the same dimension.

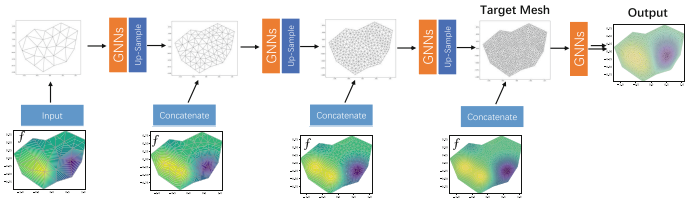

Fig. 1. The MGMI architecture: Coarse-to-fine meshes are linked with up-sampling operators, and the right-hand side is input to the NN at all different resolutions.

However, when it comes to Graph Neural Networks, no obvious down-sampling (pooling) and up-sampling operators exist. Some operators have been proposed in the context of graph or node classification on graphs [10], but they inevitably destroy the graph structure and lose node connections. Moreover, to the best of our knowledge, no explicit pooling operators for graphs with edge features have yet been proposed. The situation here is different, thanks to the hierarchy of meshes of increasing complexity over the same 2D-domain. It easily defines operators that transform the features from one mesh to the next, up- or downward.

Sampling Operators. The sampling operators from mesh $\mathcal{M}_1$ to mesh $\mathcal{M}_2$ use the k-nearest interpolation proposed in PointNet++ [15]. Let y be a node from $\mathcal{M}_1$, and assume its k nearest neighbors on $\mathcal{M}_2$ are $(x_1, \ldots, x_k)$. The interpolated feature for y is defined from those of the x_i's as:

$$\mathbf{f}(\mathbf{y}) = \frac{\sum_{i=1}^{k} w(x_i)\mathbf{f}(x_i)}{\sum_{i=1}^{k} w(x_i)}, \text{ where } w(x_i) = \frac{1}{||y - x_i||_2}$$

Based on these operators, it is straightforward to define multi-grid architectures in the context of PDE simulation.

The Multi-grid U-Net Architecture. First, we propose a simple adaptation of the U-Net architecture proposed in [17], where each block is a MoNet block (Sect. 3.1), followed by one sampling operator as described above. The "horizontal" connections that characterize U-Net are added, the input f is provided to the first layer, and the solution u of the PDE is the output of the last layer (details in Sect. 5).

The Multi-grid Multi-input Architecture. Figure 1 displays the proposed architecture (MGMI). Here, only upsampling operators are used: the model starts from the coarsest mesh, and the input is the projection of f on this mesh. After a GNN block, an upsampling operator adapts the output to the next mesh, and a projection of f on the current mesh concatenated with the previous output is again fed to the next GNN block. The process repeats until reaching the finest mesh (4 different levels will be used throughout this paper). This should allow the features of different granularities to be discovered gradually, from global to local features.

This architecture takes advantage of the hierarchy of meshes from the input perspective, feeding the different dimensions with ad hoc samples of the input f as well. Note that a similar strategy with the U-Net architecture did not make any significant difference.

4 Experimental Conditions

The Partial Differential Equation. The case study in this paper is a numerical simulation of the following nonlinear Poisson equation with constant Dirichlet boundary condition, an elliptic PDE defined on some domain $\Omega \in \mathbb{R}^2$:

$$-\nabla((1 - u(x) + u(x)^2) \cdot \nabla u(x)) = f(x) \quad in\ \Omega \text{ with } u(x)|_{\partial\Omega} = 1$$

The goal is to compute a numerical approximation of $u(x)$, solution of problem (1), for any continuous function $f(x)$ defined Ω. Note that any boundary condition can be handled by the proposed approach (not shown here).

The Finite Element Library. There are numerous FEM packages that can solve problem (4): *FEniCS* [1] was used throughout this work. *FEniCS* includes a mesh generator that generates a mesh from a user-defined criterion and discretization of the boundary of Ω, balancing the mesh so that all triangles are as close as possible from the equilateral (to make a long story short). Hence, the coarseness of the mesh can be roughly controlled by the user, allowing them to reach a target number of nodes approximately.

The Input Functions. The source terms f of Eq. (4) are randomly generated as a linear combination of eight isotropic Gaussian functions sharing the same standard deviation, resulting in 25 control parameters: the coordinates of the means of the Gaussian functions, their weights in the linear combination and the standard deviation. They are chosen uniformly in domain-dependent intervals.

Ground Truth and Loss Function. *FEniCS* also includes a FEM solver that computes an approximated solution of the target PDE based on the decomposition of the solution on a basis defined from a given mesh [8]. In all cases, the solution provided by *FEniCS* on the finest mesh will be considered as the ground truth. The loss function is defined by the **Mean Absolute Error** (MAE) between the network output and this ground truth. Note, however, that we will report the **Relative MAE(%)** in Sect. 5, to allow a meaningful comparison between the different experiments.

Algorithms. Our ultimate goal is to validate the proposed multi-grid approaches that use Graph NNs, MG-UNet and MGMI, i.e., to investigate the accuracy of their results when predicting the solution of some unknown test case w.r.t the ground truth given by *FEniCS*. The baseline algorithm here, called

Direct, is a simple MoNet-like GNN that works only on the finest mesh and directly predicts the solution from the input. But another goal is to compare the mesh-based GNN approach with the straightforward CNN-based more common approach. Hence each of the three algorithms described above will be transposed in the CNN framework, i.e., applied to a structured mesh and using the standard up- and down-sampling operators of deep image processing. In this framework, the Direct algorithm is simply a plain CNN. In case the mesh is not on regular grid, the data from the unstructured mesh is interpolated on the structured mesh where the CNN model is applied, and the data is projected back on the unstructured mesh, giving the approximate solution we are looking for.

Neural Networks Topology. The multi-grid architectures (Fig. 1) alternate Graph Convolutional blocks with up- or down-sampling operators. Each block is defined by 3 hyper-parameters: the number of layers, the kernel size, and the number of channels. The Direct architecture only contains GCNN blocks with the same parameters; the number of blocks is adjusted so that the order of magnitude of the number of weights is similar to the Multi-Grid ones. For the CNN architecture, the number of CNN layers is similarly adjusted.

After some preliminary experiments, the number of nearest neighbors in all sampling operators was set to 6; The kernel size was set to 5 (see 3.1) for Graph-based architectures; And the filter of CNN models is set to 3×3. Other hyper-parameters were adjusted using the validation set[1].

Implementation is done in PyTorch using the PyTorch-Geometric for the MoNet layers and the Adam optimizer, with batch size 100, with an exponentially decaying learning rate and early stopping condition, based on the loss on the validation set.

Domains. Three different types of 2D domains will be considered: First, a simple square domain that should favor the CNN approach; then a 'donut-like' domain to somehow penalize the CNN approach. In both cases, the domain and the meshes will be fixed; only the right-hand side f of the PDE will be subject to learning. Finally, the last experiment will also learn the domain itself: all samples will have different domain shapes and different function f.

Methodology and Result Presentation. For each type of domain, a training set of 42 000 examples is generated. A 12-folds cross-validation is used to assess the robustness of the approach, and we report the averages and standard deviations of the relative MAE errors on the 12 hold-out test sets (labeled "Val"). For CNN algorithms, we also report the relative MAE errors computed on the grid mesh (before projecting back onto the unstructured mesh), labeled "CNN Error". Furthermore, as some of these 40000 samples have been used for hyper-parameter tuning, we also report the errors on test sets that have never

[1] Due to space limitation, all details cannot be included here. An INRIA Technical Report will soon be available with details and many more experimental results.

Table 1. Relative MAE (%) results on Squared domain (see text for row details)

Models	Graph			CNN		
	MGMI	MG-UNet	Direct	MGMI	U-Net	Direct
GPU cost(s)	63.98	61.58	138.12	23.58	23.61	44.72
Val	0.61 ± 0.08	**0.40 ± 0.04**	4.00 ± 0.31	0.93 ± 0.01	1.00 ± 0.03	1.07 ± 0.05
CNN error	\	\	\	0.25 ± 0.02	0.40 ± 0.05	0.53 ± 0.04
Test	0.61 ± 0.08	**0.40 ± 0.04**	4.02 ± 0.30	0.93 ± 0.01	1.01 ± 0.03	1.08 ± 0.05
OoD 1	1.46 ± 0.13	**0.71 ± 0.07**	6.76 ± 0.35	1.09 ± 0.01	1.31 ± 0.11	1.76 ± 0.07
OoD 2	1.50 ± 0.31	**1.21 ± 0.31**	4.46 ± 0.44	1.83 ± 0.92	2.73 ± 0.52	1.63 ± 0.19

been used during the tuning phase (row "Test"). Finally, in order to check the generalization capabilities of all models, we compute some out-of-distribution errors (i.e., errors on test cases that do not belong to the same distribution as the training samples), labeled "OoD". All reported error values are expressed in percentage terms. We also report the GPU cost of training one epoch (inference CPU costs are discussed in Sect. 5.3). Another interesting quantity is the average *interpolation error*, that is computed by projecting on the regular grid a solution u defined on the unstructured mesh, and projecting it back onto the unstructured mesh. For each training set, we compute its relative MAE *interpolation error* to better compare graph and CNN models.

Statistical Significance. For all pair-wise comparisons between test errors, we performed a Wilcoxon signed-rank statistical test with 95% confidence on the results of the 12 models obtained through the 12-fold procedure. As most differences are statistically significant, and because of space limitation, we will only signal the non-statistically significant differences.

5 Experimental Results

5.1 Fixed Domains

A first experiment deals with examples in the same **square domain** $\Omega = [0, 1] \times [0, 1]$, that is meshed in 31×31 squares, or into unstructured meshes with respectively 961, 249, 70, 23 nodes. In this case, the boundary, in particular, is exactly discretized by both the structured (regular grid) and unstructured mesh. The average interpolation error is $0.91\% \pm 0.25$.

Table 1 shows the results of all algorithms. Multi-Grid models outperform Direct models, especially in the context of Graph NNs. Furthermore, in the Graph context, the training time of the Direct model is also larger. Hence we will not consider these Direct models any further in this work.

Graph-based models clearly outperforms CNN-based ones. However, CNN models have an excellent performance on the regular grid, and the discrepancy is only due to the interpolation error. Final remark for the square domain: whereas

MG-UNet slightly outperform MGMI in the graph-based approaches, the reverse is true for the CNN-based approaches.

Another fixed domain was also experimented with, some **donut** shape, for which the boundary is ill-represented on the regular grid. As could be expected, the performances of CNN-based approaches are much worse than on the square domain, while those of the graph-based ones remain very similar.

Out-of-Distribution Generalization. All source terms up to now had been drawn using the same distribution (see Sect. 5). The out-of-distribution generalisation is assessed by two test datasets. "OoD 1" in Table 1 uses a larger standard deviation (in $\mathbb{U}(40, 50)$ instead of $\mathbb{U}(10, 40)$), while "OoD 2" uses larger weights ($\mathbb{U}(20, 35)$ instead of $\mathbb{U}(-20, 20)$). The results are very good compared to the "Test" ones . And MG-UNet slightly, but significantly, outperforms MGMI.

5.2 Variable Domains

This case aims at checking the ability of different models to also generalize w.r.t. the domain itself, as well as the right-hand side f. Each training sample is defined on a different polygonal domain (and hence a different mesh), also involving a different f. Domains are defined by n vertices lying at a normally-distributed distance (with mean 1 and tunable variance) from the origin (0, 0), at angular $\frac{2\pi}{n} \pm \epsilon$ for some uniformly distributed ϵ ($n = 30$ here). This process ensures that all domains have similar areas – see on Fig. 2 a sample domain, and the solutions of FEM (left) and MGMI (right). As before, 4 different meshes are created for each domain, with the largest mesh size in $[1000, 1200]$ (average in the dataset: 1037). The regular meshes are defined on the $[-2, 2]^2$ square with 73×73 discretization, and the average number of grid points inside the polygon domains is 1023. The average interpolation error is $1.63\% \pm 0.57$.

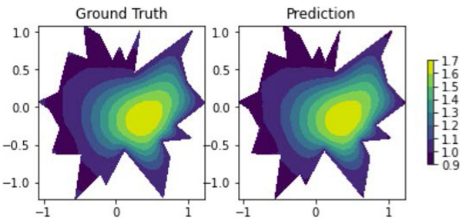

Fig. 2. Comparison between ground truth and prediction from MGMI.

The top part of Table 3 shows the overall errors are greater than that in the previous test cases – which was to be expected. However, graph models largely outperform CNN models on unfixed meshes. Also, MGMI is slightly but significantly better than MG-UNet on these problems with very different graph structures.

Out-of-Distribution Generalization. Two out-of-distribution experiments are presented here. 5000 examples are generated in each case, with random domains and source terms f.

Mesh Complexity. The first experiment (second set of rows in Table 3) investigates the influence of the number of nodes of the meshes. In the training set, the finest meshes had from 1 000 to 1 200 nodes. Two test sets were generated, with #nodes in $[1200, 1600]$ and $[1600, 2000]$ respectively. All performances nicely degrade when departing from the training distribution. And the graph-based approaches still perform much better than the CNN ones, with a slight but significant advantage to MGMI over MG-UNet.

Domain Shape. Whereas the training set was made of polygons with 30 vertices, this second OoD experiment concerned shapes made respectively with 5, 10, and 20 vertices, keeping the number of mesh nodes approximately the same. The bottom set of rows of Table 3 gives the results. Here again, the graph-based approaches outperform the CNN ones, and MGMI slightly outperforms MG-UNet (though not significantly in the case of 5 and 10 vertices). However, more severely than expected, the performance degrades when going from 30 vertices (Test) down to 5 vertices: the learned models seem more sensitive to the shape of the meshes than to their complexity. Further experiments (not shown for space reasons) used a variable number of vertices during training. The training accuracy remained good with a relative error of $2.37\% \pm 0.34$, while the test on 20, 10 and 5 vertices were greatly improved (to 2.20, 2.44 and 2.94% respectively). But of course, these test sets are not Out-of-Distribution anymore.

5.3 Computational Costs

We use a batch size 100 to solve 5 000 unknown PDEs on the different domain types, and compare the forward computational cost of the graph-based approaches on a single GPU GTX 1080Ti with that of *FEniCS* solver on Intel(R) Xeon(R) Silver 4108 CPU (there is no GPU version of *FEniCS*) (Table 2).

Table 2. Inference CPU cost

Time (s)	Square	Donut	Polyg.
MGMI	0.912	0.643	7.336
MG-UNet	1.075	0.782	11.312
FEM	173.87	164.35	163.72

For problems on fixed mesh, the computation time of neural networks is about 100 times faster than *FEniCS*, which cannot take advantage of the fixed mesh on this nonlinear problem. When considering problems on variable polygon domains, the sampling operators slow down the graph-based approaches, making the computation time only one order of magnitude faster than *FEniCS*. Also, MGMI has fewer down-sampling operators than MG-UNet and hence allows slightly faster prediction. Finally, note that this work studied a simple problem, though nonlinear, for which FEM solvers are relatively fast. The advantage of graph-based inference for complex PDEs (e.g., 3D CFD) would be even more significant.

Table 3. Relative MAE (%) Results on Polygonal Domains (see text for details)

Models	Graph		CNN	
	MGMI	MG-UNet	MGMI	MG-UNet
GPU cost(s)	169.52	170.46	71.54	71.23
Val	2.19 ± 0.15	2.40 ± 0.20	6.09 ± 0.29	5.24 ± 0.19
CNN error	\	\	1.62 ± 0.15	1.38 ± 0.07
Test	**2.20 ± 0.16**	2.41 ± 0.18	6.14 ± 0.26	5.24 ± 0.15
1200–1600	**2.53 ± 0.17**	2.73 ± 0.21	6.55 ± 0.32	5.90 ± 0.20
1600–2000	**3.71 ± 0.25**	4.05 ± 0.27	7.21 ± 0.41	6.52 ± 0.26
Test 5 vertices	**5.29 ± 0.41**	**5.71 ± 0.60**	6.40 ± 0.72	4.73 ± 0.46
Test 10 vertices	**3.56 ± 0.31**	**3.73 ± 0.22**	6.11 ± 0.54	4.13 ± 0.42
Test 20 vertices	**2.39 ± 0.18**	2.61 ± 0.13	5.72 ± 0.39	5.08 ± 1.26

6 Conclusion

This paper introduced multi-resolution graph-based approaches to learn PDE solutions on unstructured meshes, addressing the up- and down-sampling issues of GNNs spatially, based on a hierarchy of meshes of increasing complexity. Bypassing the projection on a regular mesh and the use of standard CNNs, these approaches thus avoid the resulting interpolation errors. Experiments have shown that these hierarchical models can improve the prediction accuracy on test sets of different source terms and domain shapes, as well as the inference time compared to the classical FEM computation. Furthermore, whereas Out-of-Distribution generalization is satisfactory w.r.t. the source characteristics and the mesh complexity, further work is needed to decrease the dependency w.r.t. the domain shape outside the strict bounds of the training distribution. A short-term perspective is to consider transfer learning approaches to deal with meshes of different resolution on different domains. A longer-term perspective is to port the proposed approaches on 3D problems, for which an unstructured mesh is far more expressive than pixel-like grids.

References

1. Alnæs, M.S., Blechta, J., et al.: The FEniCS project version 1.5. Arch. Numer. Softw. **3**(100) (2015)
2. Atwood, J., Towsley, D.: Diffusion-convolutional neural networks (2016)
3. de Avila Belbute-Peres, F., Economon, T.D., Kolter, J.Z.: Combining differentiable PDE solvers and GNNs for fluid flow prediction. In: 37th ICML (2020)
4. Berg, J., Nyström, K.: A unified deep ANN approach to PDEs in complex geometries. Neurocomputing **317**, 28–41 (2018)
5. Bhatnagar, S., Afshar, Y., et al.: Prediction of aerodynamic flow fields using CNNs. Comput. Mech. **64**(2), 525–545 (2019)

6. Briggs, W., Henson, V., et al.: A Multigrid Tutorial, 2nd edn. SIAM, Philadelphia (2000)
7. Bruna, J., Zaremba, W., Szlam, A., LeCun, Y.: Spectral networks and locally connected networks on graphs (2014)
8. Ciarlet, P.G.: Finite Element Method for Elliptic Problems. Society for Industrial and Applied Mathematics, USA (2002)
9. Defferrard, M., Bresson, X., Vandergheynst, P.: CNNs on graphs with fast localized spectral filtering. In: NeurIPS (2017)
10. Gao, H., Ji, S.: Graph U-nets. In: 36th ICML. PMLR (2019)
11. Kashefi, A., Rempe, D., Guibas, L.J.: A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. Phys. Fluids **33**(2), 027104 (2021)
12. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th ICLR (2017)
13. Monti, F., Boscaini, D., et al.: Geometric deep learning on graphs and manifolds using mixture model CNNs. In: CVPR (2016)
14. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation (2017)
15. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space (2017)
16. Raissi, M.: Deep hidden physics models: deep learning of nonlinear partial differential equations. JMLR **19**(1), 932–955 (2018)
17. Ronneberger, O., Fischer, P., Brox, T.: U-net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24574-4_28
18. Sirignano, J., Spiliopoulos, K.: DGM: a deep learning algorithm for solving PDEs. J. Comput. Phys. **375**, 1339–1364 (2018)
19. Tang, W., Shan, T., et al.: Study on a Poisson's equation solver based on deep learning technique. In: IEEE EDAPS, pp. 1–3 (2017)
20. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: ICLR (2018)