MG-GNN: Multigrid Graph Neural Networks for Learning Multilevel Domain Decomposition Methods

Ali Taghibakhshi [1] Nicolas Nytko [2] Tareq Uz Zaman [3] Scott MacLachlan [4] Luke N. Olson [2] Matt West [1]

Abstract

Domain decomposition methods (DDMs) are popular solvers for discretized systems of partial differential equations (PDEs), with one-level and multilevel variants. These solvers rely on several algorithmic and mathematical parameters, prescribing overlap, subdomain boundary conditions, and other properties of the DDM. While some work has been done on optimizing these parameters, it has mostly focused on the one-level setting or special cases such as structured-grid discretizations with regular subdomain construction. In this paper, we propose multigrid graph neural networks (MG-GNN), a novel GNN architecture for learning optimized parameters in two-level DDMs. We train MG-GNN using a new unsupervised loss function, enabling effective training on small problems that yields robust performance on unstructured grids that are orders of magnitude larger than those in the training set. We show that MG-GNN outperforms popular hierarchical graph network architectures for this optimization and that our proposed loss function is critical to achieving this improved performance.

1. Introduction

Differential equations are at the core of many important scientific and engineering problems (Gholizadeh et al., 2021; Han & Lin, 2011), and often, there is no analytical solution available; hence, researchers utilize numerical solvers (Vos et al., 2002; Gholizadeh et al., 2023). Among numerical methods for solving the systems of equations obtained from discretization of partial differential equations (PDEs), domain decomposition methods (DDMs) are a popular approach (Toselli & Widlund, 2005; Quarteroni & Valli, 1999; Dolean et al., 2015). They have been extensively studied and applied to elliptic boundary value problems, but are also considered for time-dependent problems. Schwarz methods are among the simplest and most popular types of DDM, and map well to MPI-style parallelism, with both one-level and multilevel variants. One-level methods decompose the global problem into multiple subproblems (subdomains), which are obtained either by discretizing the same PDE over a physical subdomain or by projection onto a discrete basis, using subproblem solutions to form a preconditioner for the global problem. Classical Schwarz methods generally consider Dirichlet or Neumann boundary conditions between the subdomains, while Optimized Schwarz methods (OSM) (Gander et al., 2000) consider a combination of Dirichlet and Neumann boundary conditions, known as Robin-type boundary conditions, to improve the convergence of the method. Restricted additive Schwarz (RAS) methods (Cai & Sarkis, 1999) are a common form of Schwarz methods, and optimized versions of one-level RAS has been theoretically studied by St-Cyr et al. (2007). Two-level methods extend one-level approaches by adding a (global) coarse-grid correction step to the preconditioner, generally improving performance but at an added cost.

Optimizing PDE solvers has been of significant importance to the engineering and applied mathematic communities, and studies have been using various tools such as genetic programming Schmitt et al. (2021). Nevertheless, in recent years, there has been a growing focus on using machine learning (ML) methods to learn optimized parameters for iterative PDE solvers, including DDM and algebraic multigrid (AMG). In Greenfeld et al. (2019) convolutional neural networks (CNNs) are used to learn the interpolation operator in AMG on structured problems, and in a following study (Luz et al., 2020), graph neural networks (GNNs) are used to extend the results to arbitrary unstructured grids. In a different fashion, reinforcement learning methods along with GNNs are used to learn coarse-gird selection in reduction-based AMG in Taghibakhshi et al. (2021). As mentioned in Heinlein et al. (2021), when combining ML methods with DDM, approaches can be categorized into two main

[1]Department of Mechanical Science and Engineering, University of Illinois at Urbana-Champaign [2]Department of Computer Science, University of Illinois at Urbana-Champaign [3]Scientific Computing Program, Memorial University of Newfoundland [4]Department of Mathematics and Statistics, Memorial University of Newfoundland. Correspondence to: Ali Taghibakhshi <alit2@illinois.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

families, namely using ML within a classical DDM framework to obtain improved convergence and using deep neural networks as the main solver or discretization module for DDMs. In a recent study (Taghibakhshi et al., 2022), GNNs are used to learn interface conditions in optimized Schwarz DDMs that can be applied to many subdomain problems, but their study is limited to one-level solvers. Two-level domain decomposition methods often converge significantly faster than one-level methods since they include coarse-grid correction, but obtaining optimized multilevel DDMs for general unstructured problems with arbitrary subdomains remains an open challenge.

Graph neural networks (GNNs) extend learning based methods and convolution operators to unstructured data. Similar to structured problems, such as computer vision tasks, many graph-based problems require information sharing beyond just a limited local neighborhood in a given graph. However, unlike in CNNs, where often deep CNNs are used with residual skip connections to achieve long range information passing, GNNs dramatically suffer depth limitations. Stacking too many GNN layers results in *oversmoothing*, which is due to close relation of graph convolution operators to Laplacian smoothing (Li et al., 2018; Oono & Suzuki, 2019). Oversmoothing essentially results in indistinguishable node representations after too many GNN layers, due to information aggregation in a large local neighborhood. Inspired by the Unet architecture in CNNs (Ronneberger et al., 2015), graph U-nets (Gao & Ji, 2019) were introduced as a remedy for longe-range information sharing in graphs without using too many GNN layers. Similar to their CNN counterparts, graph-Unet architectures apply down-sampling layers (pooling) to aggregate node information to a coarser representation of the problem with fewer nodes. This is followed by up-sampling layers (unpooling, with the same number of layers as pooling) to reconstruct finer representations of the problem and allow information to flow back to the finer levels from the coarser ones.

As mentioned in Ke et al. (2017), U-net and graph-Unet architectures suffer from a handful of problems and non-optimalities. In these architectures, scale and abstraction are combined, meaning earlier, finer layers cannot access the information of the coarse layers. In other words, initial layers learn deep features only based on a local neighborhood without considering the larger picture of the problem. Moreover, finer levels do not benefit from information updates until the information flow reaches the coarsest level and flows back to the finer levels. That is, the information flow has to complete a full (graph) U-net cycle to update the finest level information, which could potentially require multiple conventional layers, leading to oversmoothing in the case of graph U-nets. More recently, there has been similar hierarchical GNN architectures utilized for solving PDEs, such as those proposed by Fortunato et al. (2022) and Li et al. (2020). In each case, the architecture is similar to a U-net, in terms in terms of information flow (from the finest to coarsest graph and back), and there is no cross-scale information sharing, making them prone to the aforementioned U-net type problems.

To fully unlock the ability of GNNs to learn optimal DDM operators, and to mitigate the shortcomings of graph U-nets mentioned above, we introduce here a novel GNN architecture, multigrid graph neural networks (MG-GNN). MG-GNN information flow is parallel at all scales, meaning every MG-GNN layer processes information from both coarse and fine scale graphs. We employ this MG-GNN architecture to advance DDM-based solvers by developing a learning-based approach for two-level optimized Schwarz methods. Specifically, we learn the Robin-type subdomain boundary conditions needed in OSM as well as the overall coarse-to-fine interpolation operator. We also develop a novel loss function essential for achieving superior performance compared to previous two-level optimized RAS. The summary of contributions of this paper is as follows:

- Introduce a multigrid graph neural network (MG-GNN) architecture that outperforms existing hierarchical GNN architectures in learning DDMs and scales linearly with problem size;
- Improve the loss function with theoretical guarantees essential for training two-level Schwarz methods;
- We demonstrate a general method that can train only on small problems and then generalizes to orders of magnitude larger problems, and
- Outperform classical two-level RAS, both as stationary algorithm and as a preconditioner for the flexible generalized minimum residual (FGMRES) iteration.

2. Background

In this section, we review one and two-level DDMs. Let Ω be an open set in $\mathbb{R}^2$, and consider the Poisson equation:

$$-\Delta\Phi = f, \qquad (1)$$

where Δ is the Laplace operator and $f(x, y)$ and $\Phi(x, y)$ are real-valued functions. Alongside (1), we consider inhomogeneous Dirichlet conditions on the boundary of Ω, $\partial\Omega$, and use a piecewise linear finite-element (FE) discretization on arbitrary triangulations of Ω. In the linear FE discretization, every node in the obtained graph corresponds to a degree of freedom (DoF) in the discretization, and the set of all nodes is denoted by D. The set D is decomposed into S non-overlapping subdomains $\{D^0_1, D^0_2, \ldots, D^0_S\}$ (where the superscript in the notation indicates the amount of overlap; hence, the superscript zero for the non-overlapping decomposition). The union of the subdomains covers the set of all DoFs, $D = \cup D_i^0$, so that each node in D is contained in exactly one D_i^0. Denote the restriction operator for discrete DoFs onto those in D_i^0 by R_i^0 and the corresponding extension from D_i^0 to D by $(R_i^0)^T$. Following the FE discretization of problem, we obtain a linear system to solve, $Ax = b$, where A is the global stiffness matrix. For every D_i^0, we obtain the subdomain stiffness matrix as $A_i^0 = R_i^0 A (R_i^0)^T$. In the OSM setting, alternative definitions to this Galerkin projection for A_i^0 are possible as noted below. To obtain the coarse-level representation of the problem, let $P \in \mathbb{R}^{S\times|D|}$ be the piecewise-constant interpolation operator that assigns every node in D_i^0 to the i-th coarse node. The coarse-level operator is then obtained as $A_C = P^T AP$.

The restricted additive Schwarz method (RAS) (Cai & Sarkis, 1999) is an important extension to the Schwarz methodology for the case of overlapping subdomains, where some nodes in D belong to more than one subdomain. Denoting the overlap amount by $\delta \in \mathbb{N}$, we define the subdomains D_i^δ for $\delta > 0$ by recursion, as $D_i^\delta = D_i^{\delta-1} \cup \{j \,|\, a_{kj} \neq 0 \text{ for } k \in D_i^{\delta-1}\}$. For the coarse-grid interpolation operator, P, each of the overlapping nodes is now associated with multiple columns of P, which is typically chosen as a partition of unity, with rows of P having equal non-zero weights (that can be interpreted as the probability of assigning a fine node to a given subdomain). The conventional two-level RAS preconditioner is then defined by considering the fine-level operator, M_{RAS}, and the coarse-level correction operator, $C_{\text{2-RAS}}$, given by

$$M_{\text{RAS}} = \sum_{i=1}^{S} (\tilde{R}_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta, \tag{2}$$

$$C_{\text{2-RAS}} = P(P^T AP)^{-1} P^T, \tag{3}$$

where $A_i^\delta = (R_i^\delta)^T AR_i^\delta$. The operator R_i^δ denotes restriction for DoFs in D to those in D_i^δ while $\tilde{R}_i^\delta$ is a modified restriction from D to D_i^δ that takes nonzero values only for DoFs in D_i^0. The two-level RAS preconditioner is given as $M_{\text{2-RAS}} = C_{\text{2-RAS}} + M_{\text{RAS}} - C_{\text{2-RAS}} AM_{\text{RAS}}$, with the property that $I - M_{\text{2-RAS}}A = (I - C_{\text{2-RAS}}A)(I - M_{\text{RAS}}A)$.

In the case of optimized Schwarz, the subdomain systems (fine-level A_i^δ) are modified by imposing a Robin boundary condition between subdomains, writing $\tilde{A}_i^\delta = A_i^\delta + L_i$, where L_i is the term resulting from the Robin-type condition:

$$\alpha u + \vec{n} \cdot \nabla u = g(x), \tag{4}$$

where g denotes inhomogeneous data and $\vec{n}$ is the outward unit normal to the boundary. From the finite-element perspective, a Robin boundary condition appears in the weak form as an integral over the exterior edges of the (sub)domain of the form $\alpha \int_\partial \Omega uv$. While it is common to think of this for straight edges of a regular domain, it is defined in the same way for irregular subdomains, and the resulting nonzero pattern in L_i is simply (periodic) tridiagonal, when the vertices are ordered in a cycle around the boundary of the (sub-)domain. Here, α could be variable along the edges of each subdomain, leading to a general symmetric (periodic) tridiagonal structure for L_i when the vertices are ordered in a cycle (and a known pattern when the vertices are in another order). Thus, for the optimization, we identify a cycle or path in the graph corresponding to A_i with the property that every node in the cycle (or path) is on the boundary of D_i^δ, but not the boundary of D, the discretized domain. This defines the (permuted) tridiagonal structure for L_i, and we restrict the nonzero entries in L_i to its diagonal and off-diagonal entries associated with edges on the cycle or path.

The fine-level operator for optimized Schwarz is then given by

$$M_{\text{ORAS}} = \sum_{i=1}^{S} \left(\tilde{R}_i^\delta\right)^T \left(\tilde{A}_i^\delta\right)^{-1} R_i^\delta, \tag{5}$$

where the choice of weight, α, in the subdomain Robin boundary condition is a parameter for optimization. Similarly, the method can be improved by optimizing the choice of coarse-level interpolation operator, P, but this has not been fully explored in the OSM literature. Similarly to with RAS, we define the two-level ORAS preconditioner as $M_{\text{2-ORAS}} = C_{\text{2-RAS}} + M_{\text{ORAS}} - C_{\text{2-RAS}} AM_{\text{ORAS}}$.

The work of Taghibakhshi et al. (2022) suggests a method to learn L_i for one-level ORAS. Here, we learn both L_i and P for two-level methods since, as later shown in Figure 7, the two-level methods are significantly more robust. Furthermore, as we show in Section 5.2, while learning both ingredients improves the performance, learning the interpolation operator, P, is significantly more important than learning L_i's in order to obtain a two-level solver that outperforms classical two-level RAS.

3. Multigrid graph neural network

The multigrid neural architecture (Ke et al., 2017) is an architecture for CNNs that extracts higher level information in an image more efficiently by cross-scale information sharing, in contrast to other CNN architectures, such as U-nets, where abstraction is combined with scale. That is, in one multigrid layer, the information is passed between different scales of the problem, removing the necessity of using deep CNNs or having multilevel U-net architectures. Inspired by Ke et al. (2017), we develop a multigrid architecture for GNNs, enabling cross-scale message (information) passing without making the GNN deeper; we call our architecture

Multigrid GNN, or MG-GNN. Figure 1 shows one layer of the MG-GNN with two levels (a fine and a coarse level).

The input data to one layer of an MG-GNN has L different graphs, from fine to coarse, denoted by $G^{(\ell)} = (X^{(\ell)}, A^{(\ell)})$, where $A^{(\ell)} \in \mathbb{R}^{n_\ell \times n_\ell}$ and $X^{(\ell)} \in \mathbb{R}^{n_\ell \times d}$ are adjacency and node feature matrices, respectively, and n_ℓ and d are the number of nodes and node feature dimension in ℓ-th graph for $\ell \in \{0, 1, \dots, L-1\}$, with $\ell = 0$ denoting the finest level. If the input graph does not have multiple levels, we obtain the coarser levels recursively by considering a node assignment matrix (clustering operator) $R^{(\ell)} \in \mathbb{R}^{n_{\ell+1} \times n_\ell}$, for $\ell \in \{0, 1, \dots, L-2\}$:

$$X^{(\ell+1)} = R^{(\ell)} X^{(\ell)}, \tag{6}$$

$$A^{(\ell+1)} = R^{(\ell)} A^{(\ell)} (R^{(\ell)})^T. \tag{7}$$

We note that, in general, the assignment matrix R^ℓ could be any pooling/clustering operator, such as k-means clustering, learnable pooling, etc. We denote $R^{(\ell \to k)}$ to be the assignment matrix of graph level ℓ to level k (with $\ell < k$), which is constructed through $R^{(\ell \to k)} = \prod_{j=\ell}^{k-1} R^{(j)}$ (down-sampling). To complement this terminology, we also define $R^{(k \to \ell)} = (R^{(\ell \to k)})^T$ for $\ell > k$ (up-sampling), and for the case of $\ell = k$, the assignment matrix is simply the identity matrix of dimension n_ℓ. The mathematical formalism of the m-th layer of the MG-GNN with L levels is as follows: given all graphs feature matrices, $X_m^{(\ell)}$, for $\ell \in \{0, 1, \dots, L-1\}$:

$$\dot{X}^{\ell \to k} = F^{\ell \to k}(X_m^{(\ell)}, X_m^{(k)}, R^{(\ell \to k)}) \tag{8}$$

$$\tilde{X}_m^{(\ell)} = [\dot{X}^{0 \to \ell} \| \dot{X}^{1 \to \ell} \| \dots \| \dot{X}^{k-1 \to \ell}] \tag{9}$$

$$X_{m+1}^{(\ell)} = \text{GNN}^{(\ell)}(\tilde{X}_m^{(\ell)}, A^{(\ell)}) \tag{10}$$

where $\|$ denotes concatenation, and $\text{GNN}^{(\ell)}$ and $F^{\ell \to \ell}$ could be any homogeneous and heterogeneous GNNs, respectively. For the case of $\ell \neq k$, we consider $F^{\ell \to k}$ to be a heterogeneous message passing scheme between levels ℓ and k, which is defined as follows. Consider any node v in $G^{(\ell)}$ and denote the row in $X_m^{(\ell)}$ corresponding to the feature vector of node v by x_v. Then, $F^{\ell \to k}(X_m^{(\ell)}, X_m^{(k)}, R^{(\ell \to k)})$ is given by

$$m_v = g^{\ell \to k}\left(\underset{\omega \in \mathcal{N}(v)}{\square} f^{\ell \to k}(x_v, x_\omega, e_{v\omega}), x_v \right) \tag{11}$$

where $e_{v\omega}$ is the feature vector of the edge (if any) connecting v and ω, $\square$ is any permutation invariant operator such as sum, max, min, etc., and $f^{\ell \to k}$ and $g^{\ell \to k}$ are learnable multilayer perceptrons (MLPs). See Figure 2 for visualization of up-sampling and down-sampling in MG-GNN. In this study, we consider a two-level MG-GNN (see Figure 1) and, for the clustering, we consider a k-means-based clustering algorithm (best known as Lloyd's algorithm) which has $O(n)$ time complexity and guarantees that every node will be assigned to a subdomain (Bell, 2008; Lloyd, 1982)) in a connected graph. As mentioned earlier, the MG-GNN architecture could alternatively use any pooling/clustering method such as DiffPool (Ying et al., 2018), top-K pooling (Gao & Ji, 2019), ASAP (Ranjan et al., 2020), SAGPool (Lee et al., 2019), to name but a few. In other words, MG-GNN architecture does not have any restrictions in terms of the choice of the pooling/clustering algorithm. However, for the case of this paper, since RAS (and therefore, ORAS) necessitates every node in the fine grid be assigned to a subdomain, we do not consider the aforementioned pooling (clustering) methods. We note that other k-means-based clustering algorithms (e.g. Graclus clustering) could alternatively be used instead of Lloyd's algorithm; however, we only consider Lloyd's algorithm in this study.

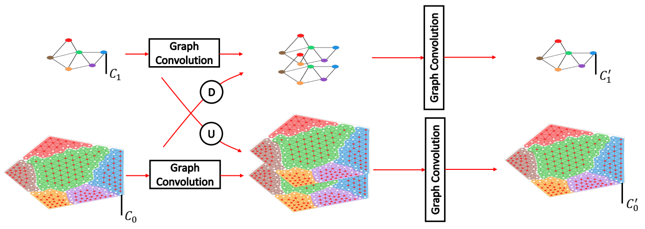


Figure 1. One layer of MG-GNN. c_i and c'_i denote the feature dimensions of different levels before and after passing through an MG-GNN layer, respectively.

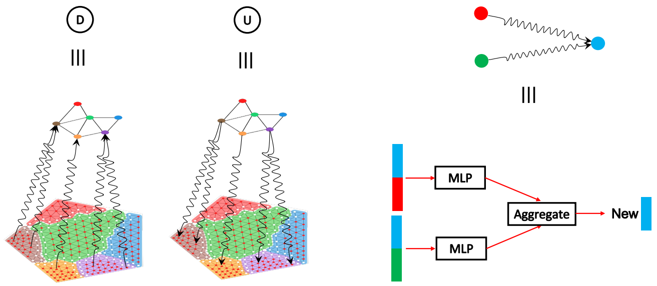


Figure 2. Upsampling and downsampling in MG-GNN.

4. Optimization problem and loss function

In this section, we denote the ℓ^2 norm of a matrix or vector by $\| \cdot \|$ and the spectral radius of matrix T by $\rho(T)$. Our objective is to minimize the asymptotic convergence factor of the two-level ORAS method, defined as minimizing $\rho(T)$, where $T = I - M_{\text{2-ORAS}} A = (I - C_{\text{2-ORAS}} A)(I - M_{\text{ORAS}} A)$ is the error propagation operator of the method. Since T is not necessarily symmetric, $\rho(T)$ is formally defined as the extremal eigenvalue of $T^T T$. As discussed in Wang et al. (2019), numerical unsuitability of backpropagation of an eigendecomposition makes it infeasible to directly minimize $\rho(T)$. To this end, Luz et al. (2020) relax the spectral radius to the Frobenius norm (which is an upper bound for it), and minimize that instead. However, for the case of optimizing one-level DDM methods, the work in Taghibakhshi et al. (2022) highlights that the Frobenius norm is not a "tight" upper bound for $\rho(T)$, and considers minimizing a relaxation of $\rho(T)$ inspired by Gelfand's formula, $\forall_{K\in\mathbb{N}} \quad \rho(T) \leq \|T^K\|^{\frac{1}{K}} = \sup_{x:\|x\|=1}(\|T^K x\|)^{\frac{1}{K}}$. We present a modified version of the loss function introduced by Taghibakhshi et al. (2022) and, in Section 5.3, we show the necessity of this modification for improving the two-level RAS results.

Consider the discretized problem with DoF set D of size n, decomposed into S subdomains, $D_1^\delta, D_2^\delta, \ldots, D_S^\delta$ with overlap δ. The GNN takes D, its decomposition, and a sparsity pattern for the interface values and that of the interpolation operator as inputs and its outputs are the learned interface values and interpolation operator (see Appendix B for more discussion on inputs and outputs of the network):

$$P^{(\theta)}, L_1^{(\theta)}, L_2^{(\theta)}, \ldots, L_S^{(\theta)} \leftarrow \psi^{(\theta)}(D). \tag{12}$$

where ψ^θ denotes the GNN, and θ represents the learnable parameters in the GNN.

We obtain the modified two-level ORAS (Optimized Restricted Additive Schwarz) operator by using the learned coarse grid correction operator, $C^\theta_{\text{2-ORAS}} = P^{(\theta)}\left((P^{(\theta)})^T A P^{(\theta)}\right)^{-1}\left(P^{(\theta)}\right)^T$, and the fine grid operator, $M^{(\theta)}_{\text{ORAS}}$ from (12). The associated 2-level error propagation operator is then given by $T^{(\theta)} = (I - C^{(\theta)}_{\text{2-ORAS}}A)(I - M^{(\theta)}_{\text{ORAS}}A)$.

In order to obtain an approximate measure of $\rho(T^{(\theta)})$ while avoiding eigendecomposition of the error propagation matrix, similar to Taghibakhshi et al. (2022), we use stochastic sampling of $\left\|\left(T^{(\theta)}\right)^K\right\|$, generated by the sample set $X \in \mathbb{R}^{n\times m}$ for some $m \in \mathbb{N}$, given as

$$X = [x_1, x_2, \ldots, x_m], \forall_j \; x_j \sim \mathbb{R}^n \text{ uniformly}, \|x_j\| = 1, \tag{13}$$

where each x_j is sampled uniformly randomly on a unit sphere in $\mathbb{R}^n$ using the method introduced in Box (1958). We then define

$$Y_K^{(\theta)} = \left\{\left\|\left(T^{(\theta)}\right)^K x_1\right\|, \left\|\left(T^{(\theta)}\right)^K x_2\right\|, \ldots, \left\|\left(T^{(\theta)}\right)^K x_m\right\|\right\}. \tag{14}$$

Note that $\left\|\left(T^{(\theta)}\right)^K x_j\right\|$ is a lower bound for $\left\|\left(T^{(\theta)}\right)^K\right\|$. Taghibakhshi et al. (2022) use $\mathcal{L}^{(\theta)} = \max(Y_K^{(\theta)})$ as a practical loss function. However, for large values of K, this loss function suffers from vanishing gradients. Moreover, as we show in Section 5.3, employing this loss function results in inferior performance of the learned method in comparison to two-level RAS. To overcome these issues, we define $Z_k^{(\theta)} = \max((Y_k^{(\theta)})^{\frac{1}{k}})$ for $1 \leq k \leq K$ to arrive at a new loss function,

$$\mathcal{L}^{(\theta)} = \langle \text{softmax}(Z^{(\theta)}),\ Z^{(\theta)}\rangle + \gamma \text{tr}\left((P^{(\theta)})^T A P^{(\theta)}\right), \tag{15}$$

where $Z^{(\theta)} = \left(Z_1^{(\theta)}, Z_2^{(\theta)}, \ldots, Z_K^{(\theta)}\right)$, $0 < \gamma$ is an adjustable constant, and tr(M) is the trace of matrix M. Adding the term $\text{tr}((P^{(\theta)})^T A P^{(\theta)})$ is inspired by energy minimization principles, to obtain optimal interpolation operators in theoretical analysis of multilevel solvers (Xu, 1992; Wan et al., 1999). In Section 5.3, we show the significance of this term in the overall performance of our model. Nevertheless, for the first part of the new loss function (15), we prove that it converges to the spectral radius of the error propagation matrix in a suitable limit. First, we include two lemmas:

Lemma 1. *For $x, y \in \mathbb{R}$, with $0 \leq y \leq x$ and any $K \in \mathbb{N}$, $x^{\frac{1}{K}} - y^{\frac{1}{K}} \leq (x-y)^{\frac{1}{K}}$*

Proof. See Lemma 3 from Taghibakhshi et al. (2022). □

Lemma 2. *For any nonzero square matrix $T \in \mathbb{R}^{n\times n}$, $k \in \mathbb{N}$, $\epsilon, \xi > 0$, and $0 < \delta < 1$, there exists $M \in \mathbb{N}$ such that for any $m \geq M$, if we choose $x_1, x_2, \ldots, x_m$ uniformly random from $\{x \in \mathbb{R}^n \mid \|x\| = 1\}$, and $Z = \max\{\|T^k x_1\|^{\frac{1}{k}}, \|T^k x_2\|^{\frac{1}{k}}, \ldots, \|T^k x_m\|^{\frac{1}{k}}\}$ then, with a probability of at least $(1-\delta)$, the following hold:*

$$0 \leq \|T^k\|^{\frac{1}{k}} - Z \leq \epsilon, \tag{16}$$

$$\rho(T) - \xi \leq Z. \tag{17}$$

Proof. The left side of the first inequality is achieved by considering the definition of matrix norm, i.e. for any $1 \leq i \leq m$, $\|T^k x_i\| \leq \sup_{\|x\|=1} \|T^k x\| = \|T^k\|$, then taking the kth root of both sides. For the right side of the first inequality, consider the point $x^* \in \{x \in \mathbb{R}^n \mid \|x\| = 1\}$ such that $\|T^k x^*\| = \sup_{\|x\|=1} \|T^k x\|$ (such a point exists since $\mathbb{R}^n$ is finite dimensional). Let S be the total volume of the surface of an n dimensional unit sphere around the origin, and denote by $\tilde{S}$ the volume on this surface within distance $\tilde{\epsilon}$ of the point x^* in the ℓ^2 measure, for $\tilde{\epsilon} = \frac{\epsilon}{\|T^k\|^{\frac{1}{k}}}$. Let $m \geq M_1 > \frac{\log(\delta)}{\log\left(1-\frac{\tilde{S}}{S}\right)}$, then, since $0 < \delta < 1$, we have:

$$P(\|x^* - x_i\| > \tilde{\epsilon},\ \forall_i) = \left(1 - \frac{\tilde{S}}{S}\right)^m \leq \delta \tag{18}$$

Therefore, with probability of at least $(1-\delta)$, there is one x_i within the $\tilde{\epsilon}$ neighborhood of x^* on the unit sphere. Without loss of generality, let x_1 be that point. Using Lemma 1 and the reverse triangle inequality, we have

$$\|T^k x^*\|^{\frac{1}{k}} - \|T^k x_1\|^{\frac{1}{k}} \leq \left(\|T^k x^*\| - \|T^k x_1\|\right)^{\frac{1}{k}} \leq \|T^k\|^{\frac{1}{k}} \|x^* - x_1\|^{\frac{1}{k}} \leq \|T^k\|^{\frac{1}{k}} \tilde{\epsilon} = \epsilon \quad (19)$$

which finishes the proof for the right side of the first inequality.

For the second inequality, since $\rho(T) \leq \|T^k\|^{\frac{1}{k}}$, choose M_2 such that, with probability $1-\delta$, (16) holds for $\epsilon = \|T^k\|^{\frac{1}{k}} - \rho(T) + \xi > 0$. Rearranging (16) then yields (17) for any $m \geq M = \max\{M_1, M_2\}$. $\square$

We next state the main result on optimality.

Theorem 3. *For any nonzero matrix T, $\epsilon > 0$, and $\delta < 1$, there exist $M, K \in \mathbb{N}$ such that for any $m > M$, if one chooses m points, x_j, uniformly at random from $\{x \in \mathbb{R}^n \,|\, \|x\| = 1\}$ and defines $Z_k = \max\{\|T^k x_1\|^{\frac{1}{k}}, \|T^k x_2\|^{\frac{1}{k}}, \ldots, \|T^k x_m\|^{\frac{1}{k}}\}$, then $Z = (Z_1, Z_2, \ldots, Z_K)$ satisfies:*

$$P\left(|\langle \textit{softmax}(Z),\ Z\rangle - \rho(T)| \leq \epsilon\right) > 1 - \delta. \quad (20)$$

Proof. Since $\rho(T) \leq \|T^k\|^{\frac{1}{k}}$ for any k and $\lim_{k\to\infty} \|T^k\|^{\frac{1}{k}} = \rho(T)$, for any $0 < \alpha$, there exists $K^* \in \mathbb{N}$ such that for any $k > K^*$, $0 \leq \|T^k\|^{\frac{1}{k}} - \rho(T) < \alpha$. Take $0 < \alpha < \min\{\frac{\epsilon}{2}, \log(\frac{e^{-\epsilon}(\epsilon+\rho(T))}{\frac{\epsilon}{2}+\rho(T)})\}$, let $u = \max\{\max_{1\leq k\leq K^*}\{\|T^k\|^{\frac{1}{k}}\}+\alpha, \rho(T)+2\alpha\}$, $\tilde{\delta} = 1-(1-\delta)^{\frac{1}{K}}$, and take $K > \max\{\frac{K^*(ue^u - (\rho(T)+\alpha)e^{\rho(T)+\alpha})}{e^{\rho(T)-\epsilon}(\epsilon+\rho(T)-(\rho(T)+\alpha)e^{\alpha+\epsilon})}, K^*\}$. Note that, by the choice of α, we have $\rho(T)+\alpha < \rho(T)+\frac{\epsilon}{2}$ and $e^{\alpha+\epsilon} < \frac{\rho(T)+\epsilon}{\rho(T)+\frac{\epsilon}{2}}$, which (along with the choice of u) guarantees a positive K. By Lemma 2, for any $1 \leq i \leq K^*$ and $K^* < j \leq K$, there exists $n_i, n_j \in \mathbb{N}$ such that:

$$P(\rho(T) - \epsilon \leq Z_i \leq u) > 1 - \tilde{\delta} \qquad \text{for } m > n_i, \quad (21)$$

$$P(\rho(T) - \epsilon \leq Z_j \leq \rho(T) + \alpha) > 1 - \tilde{\delta} \quad \text{for } m > n_j. \quad (22)$$

For any $1 \leq k \leq K$, take n_k independent points on unit sphere so that the above inequalities are satisfied for all k. Note that this can be achieved by taking $M = \sum_{k=1}^{K} n_k$. Since the points for satisfying equations (21) and (22) are chosen independently, for any $m > M$, with probability of at least $\prod_{k=1}^{K}(1-\tilde{\delta}) = 1-\delta$, we have $\rho(T) - \epsilon \leq Z_k$ for all $1 \leq k \leq K$. Consequently:

$$-\epsilon = \frac{(\rho(T)-\epsilon)\sum_{i=1}^{K} e^{Z_i}}{\sum_{i=1}^{K} e^{Z_i}} - \rho(T) \leq \frac{\sum_{i=1}^{K} Z_i e^{Z_i}}{\sum_{i=1}^{K} e^{Z_i}} - \rho(T) \quad (23)$$

$$= \langle \text{softmax}(Z),\ Z\rangle - \rho(T) \quad (24)$$

$$\leq \frac{uK^*e^u + (K-K^*)(\rho(T)+\alpha)e^{\rho(T)+\alpha}}{Ke^{\rho(T)-\epsilon}} - \rho(T) \quad (25)$$

$$= \frac{K^*(ue^u - (\rho(T)+\alpha)e^{\rho(T)+\alpha})}{Ke^{\rho(T)-\epsilon}} + (\rho(T)+\alpha)e^{\alpha+\epsilon} - \rho(T) \leq \epsilon, \quad (26)$$

where the last inequality is obtained by the choice of K. $\square$

In addition to these properties of the loss function, we now show that obtaining the learned parameters using our MG-GNN architecture scales linearly with the problem size.

Theorem 4. *The time complexity to obtain the optimized interface values and interpolation operator using our MG-GNN is $O(n)$, where n is the number of nodes in the grid (which is also the dimension of the original linear system).*

Proof. Every in/cross-level graph convolution of the MG-GNN has linear complexity. This must be the case when the graph convolution is a message passing scheme due to the sparsity in finite-element triangulations. For the case that the graph convolution is a TAGConv layer, we have $y = \sum_{\ell=1}^{L} G_\ell x_\ell + b\mathbf{1}_n$, where $x_\ell \in \mathbb{R}^n$ are the node features, L is the node feature dimension, b is a learnable bias, and $G_\ell \in \mathbb{R}^{n\times n}$ is the graph filter. In TAGConv layers, the graph filter is given as $G_\ell = \sum_{j=0}^{J} g_{\ell,j} M^j$, where M is the adjacency matrix, J is a constant, and $g_{\ell,j}$ are the filter polynomial coefficients. In other words, the graph filter it is a polynomial in the adjacency matrix M of the graph. Moreover, the matrix M is sparse, hence obtaining M^j has $O(n)$ computation cost, resulting in full TAGConv $O(n)$ time complexity. Moreover, for both the interface value head and the interpolation head of the network, the cost of calculating edge feature and the feature networks are $O(n)$, resulting in overall $O(n)$ cost of MG-GNN. $\square$

Remark: We not that the above theorem only addresses the forward pass of the MG-GNN, and for the backward pass, due to lack of support for sparse matrix operations in PyTorch, the complexity is not linear anymore. This is software limitation and not an algorithmic one, and moreover, we emphasize that this limitation is no longer binding once the MG-GNN is trained, as one no longer needs back-propagation after the training stage, and are able to test on arbitrarily large grids with $O(n)$ scalability as mentioned in Theorem 4.

5. Experiments

5.1. Training

We train each model on 1000 grids of sizes ranging from 800–1000 nodes, each obtained from the piecewise linear finite-element discretization of the diffusion equation, given as Equation 1 in Section 2 (more PDE problems and results are discussed in Appendix D). The grids are generated randomly as a convex polygon and using PyGMSH (Schlömer, 2021) for meshing its interior. The subdomains are generated using Lloyd clustering on the graph (Bell, 2008), the subdomain overlap is set to one, and the weights of the edges along the boundary determine the interface value operators, $L_i^{(\theta)}$. As shown in the interpolation head of the network in Appendix B Figure 11, the weight of the edges connecting the coarse and fine grids determine the interpolation operator. In our case, the edges between the coarse and fine grids connect every fine node to the coarse node corresponding to its own subdomain and its neighboring subdomains. Alternatively, every fine node could connect only to the coarse node corresponding to its subdomain but, as we discuss in Section 5.3, this significantly impacts the performance of the model. Moreover, each row of the interpolation operator, $P^{(\theta)}$, is scaled to have sum of one, as would be the case for classical interpolation operators. Note that this normalization is done within the forward pass in the network before obtaining the interpolation operator. Figure 3 shows several example training grids.

The model is trained for 20 epochs with batch size of 10 using the ADAM optimizer (Kingma & Ba, 2014) with a fixed learning rate of 5×10^{-4}. For the full discussion on model architecture, see Appendix B and Figure 11. For the loss function parameters introduced in Section 4, we use $K = 10$ iterations and $m = 100$ samples. We developed our code[1] using PyAMG (Bell et al., 2022), NetworkX (Hagberg et al., 2008), and PyTorch Geometric (Fey & Lenssen, 2019). All training is executed on an i9 Macbook Pro CPU with 8 cores. In the training procedure, we aim to minimize the convergence of the stationary algorithm and, as described in Section 4, we develop a loss function to achieve this goal by numerically minimizing the spectral radius of the error propagation matrix. In practice, optimized RAS methods are often used as preconditioners for Krylov methods such as FGMRES; as shown in Appendix A, the trained models using this procedure also outperform other baselines when used as preconditioners for FGMRES. Directly training to minimize FGMRES iterations would require using FGMRES in the training loop and backpropagation through sparse-sparse matrix multiplication (Nytko et al., 2022), which is left for future studies.

[1]All code and data for this paper is at `https://github.com/JRD971000/Code-Multilevel-MLORAS/` (MIT licensed).

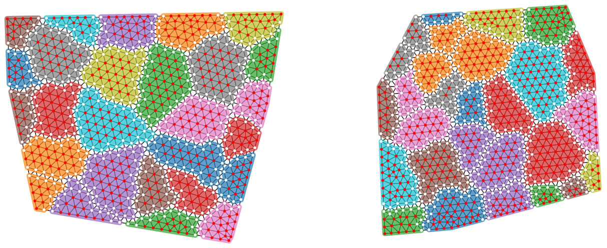


Figure 3. Training grid examples with about 1k nodes.

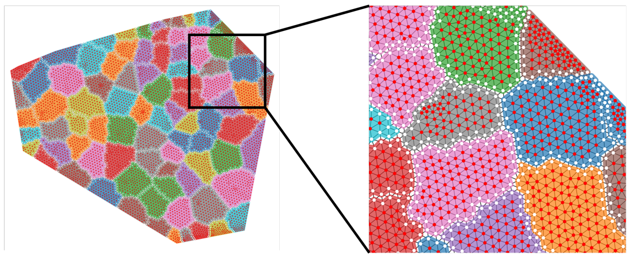


Figure 4. Test grid example with about 7.4k nodes.

We evaluate the model on test grids that are generated in the same fashion as the training grids, but are larger in size, ranging from 800 to 60k DoFs. An example of a test grid is shown in Figure 4. Note that the trained model is applied to test grids with increasing number of subdomains compared to the training set and with arbitrary subdomain shapes.

5.2. Interface values and interpolation operator

As mentioned in the Section 4, to optimize two-level RAS, one could optimize the parameters in the interface conditions (2) and/or the interpolation operator (3). For one-level RAS, on the other hand, there is no interpolation operator (since there is no coarse grid), leaving only the interface values to optimize, as was explored in Taghibakhshi et al. (2022) and St-Cyr et al. (2007). To compare the importance of these two ingredients in the two-level RAS optimization, we compare three different models. Each of these models is trained as described in Section 5.1; however, one of the models (labeled "interface") is trained by only learning the interface values (ignoring the interpolation head of the network), and using classical RAS interpolation to construct $T^{(\theta)}$. Another model, which we label "interpolation", only learns the interpolation operator weights, and uses zeros for interface matrices $L_i^{(\theta)}$ to construct $T^{(\theta)}$. The other model uses both training heads (see Figure 11), learning the interface values and the interpolation operator. We compare the performance of these models with classical RAS in Figure 5 as a stationary algorithm, and in Figure 9 as a preconditioner

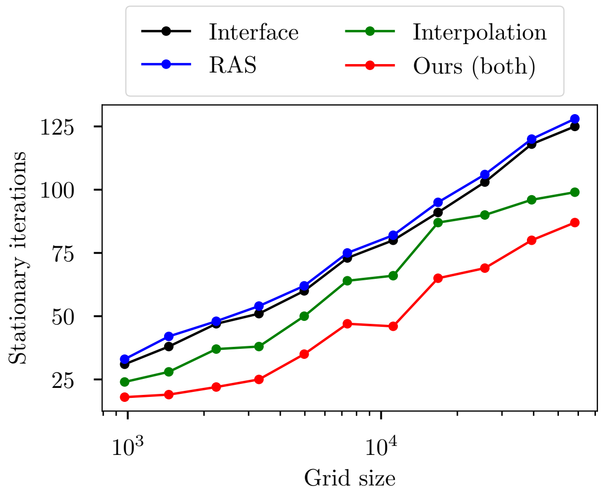


Figure 5. Effect of learning interface values, interpolation operator, or both on stationary iterations.

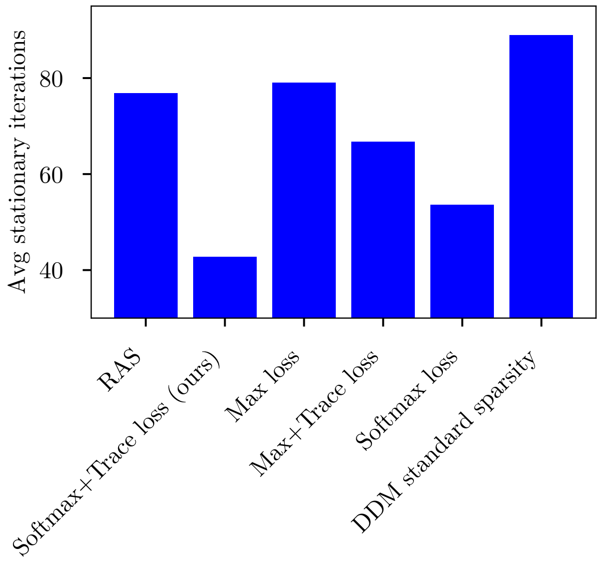


Figure 6. Effect of every ingredient in the model on average stationary iterations. All three variants outperforming the RAS baseline are utilizing modifications introduced in this paper (see (15)) compared to the "Max loss" from Taghibakhshi et al. (2022).

for a Krylov method, FGMRES.

The results show that learning the interpolation operator is more important in optimizing the 2-level RAS. Intuitively, the coarse-grid correction process in (3) plays an important role in scaling performance to large problems, due to its global coupling of the discrete DOFs. The interpolation operator is critical in achieving effective coarse-grid correction. On the other hand, the interface values are *local* modifications to the subdomains (see (2)), that cannot (by themselves) make up for a poor coarse-grid correction process. Learning both operators clearly results in the best performance in Figures 5 and 9, where the interpolation operator can be adapted to best complement the effects of the learned interface values.

5.3. Loss function and sparsity variants

We first compare five variants of our method with the RAS baseline, as shown in Figures 6 and 9. The main model is trained as described in Section 5.1 with the loss function from Section 4. All but one of the variants only differ in their loss function, and share the rest of the details. The variant labeled "Max loss" is trained with the loss function from Taghibakhshi et al. (2022). The variant labeled "Max+Trace loss" is trained with the loss function from Taghibakhshi et al. (2022) plus the $\gamma\text{tr}(P^TAP)$ term. Similarly, the variant labeled "Softmax loss" is trained by removing the $\gamma\text{tr}(P^TAP)$ part from the loss function in (15). For the last variant, we restrict the sparsity of the interpolation operator to that obtained by only connecting every fine node to its corresponding coarse node, labelled "DDM standard sparsity", and trained using the loss function from (15). As shown in Figure 6, the learned operator using this variant achieves worse performance than the baseline RAS. This is partly because, for this variant, the constraint on unit row sums of the interpolation operator effectively removes most of the learned values, since many rows of interpolation have only one nonzero entry in this sparsity pattern.

To show the effectiveness of the coarse-grid correction and the learned operator, we also compare two-level RAS and our two-level learned RAS (MLORAS 2-level) with one-level RAS and one-level optimized RAS from Taghibakhshi et al. (2022) in Figures 7 and 10.

5.4. Comparison to Graph U-net and number of layers

In this section, the performance of Graph U-net and MG-GNN with different numbers of layers is studied. Figures 8 and 10 show the performance of each of the models as stationary iterations and preconditioners for FGMRES, respectively. For a fair comparison, the MG-GNN and graph U-nets that share the same number of layers also have the same number of trainable parameters. As shown here, the best performance is achieved with 4 layers of MG-GNN, and MG-GNN strictly outperforms the graph U-net architecture with the same number of layers.

6. Conclusion

In this study, we proposed a novel graph neural network architecture, which we call multigrid graph neural network (MG-GNN), to learn two-level optimized restricted additive Schwarz (optimized RAS or ORAS) preconditioners. This new MG-GNN ensures cross-scale information sharing at every layer, eliminating the need to use multiple graph convolutions for long range information passing, which was a

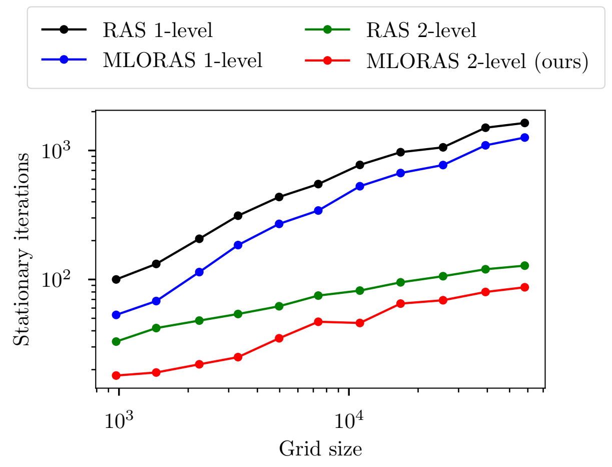


Figure 7. Comparison of stationary iterations of 2-level methods with 1-level methods from Taghibakhshi et al. (2022).

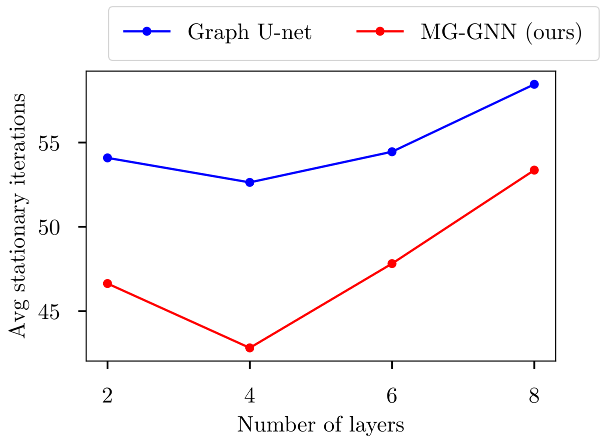


Figure 8. Average stationary iterations of graph U-net and MG-GNN with different number of layers on the test set.

shortcoming of prior graph network architectures. Moreover, MG-GNN scales linearly with problem size, enabling its use for large graph problems. We also introduce a novel unsupervised loss function, which is essential to obtain improved results compared to classical two-level RAS. We train our method using relatively small graphs, but we test it on graphs which are orders of magnitude larger than the training set, and we show our method consistently outperforms the classical approach, both as a stationary algorithm and as an FGMRES preconditioner. Our method can be extended to 3-level ORAS and above; nevertheless, due to lack of backpropagation through sparse matrix multiplication in available software, this is left for future studies. Additionally, the benefits of the MG-GNN structure was only studied for learning ORAS and studying that in other graph problems is left for future research. Moreover, a long-term goal of future research in this direction would be developing an ORAS for higher-order discretizations for which algebraic multigrid (AMG) methods are ineffective.

References

Bell, N., Olson, L. N., and Schroder, J. PyAMG: Algebraic multigrid solvers in python. *Journal of Open Source Software*, 7(72):4142, 2022. doi: 10.21105/joss.04142. URL `https://doi.org/10.21105/joss.04142`. (code is MIT licensed).

Bell, W. N. *Algebraic multigrid for discrete differential forms*. PhD thesis, University of Illinois at Urbana-Champaign, 2008.

Box, G. E. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29:610–611, 1958.

Cai, X.-C. and Sarkis, M. A restricted additive Schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, 1999.

Dolean, V., Jolivet, P., and Nataf, F. *An introduction to domain decomposition methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015. ISBN 978-1-611974-05-8. doi: 10.1137/1.9781611974065.ch1.

Du, J., Zhang, S., Wu, G., Moura, J. M., and Kar, S. Topology adaptive graph convolutional networks. *arXiv preprint arXiv:1710.10370*, 2017.

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

Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A., and Battaglia, P. Multiscale meshgraphnets. *arXiv preprint arXiv:2210.00612*, 2022.

Gander, M., Halpern, L., and Nataf, F. Optimized Schwarz methods. In *Proceedings of the 12th International Conference on Domain Decomposition*, pp. 15–27. ddm.org, 2000.

Gao, H. and Ji, S. Graph u-nets. In *ICML*, pp. 2083–2092. PMLR, 2019.

Gholizadeh, N., Hood, J. M., and Dougal, R. A. Evaluation of linear implicit quantized state system method for analyzing mission performance of power systems. *The Journal of Defense Modeling and Simulation*, pp. 15485129211061702, 2021.

Gholizadeh, N., Hood, J. M., and Dougal, R. A. Suitability of quantized devs lim methods for simulation of power systems. *arXiv preprint arXiv:2302.13437*, 2023.

Greenfeld, D., Galun, M., Basri, R., Yavneh, I., and Kimmel, R. Learning to optimize multigrid PDE solvers. In *International Conference on Machine Learning*, pp. 2415–2423. PMLR, 2019.

Hagberg, A., Swart, P., and S Chult, D. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. (code is BSD licensed).

Han, Q. and Lin, F. *Elliptic partial differential equations*, volume 1. American Mathematical Soc., 2011.

Heinlein, A., Klawonn, A., Lanser, M., and Weber, J. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021.

Ke, T.-W., Maire, M., and Yu, S. X. Multigrid neural architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6665–6673, 2017.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lee, J., Lee, I., and Kang, J. Self-attention graph pooling. In *ICML*, pp. 3734–3743. PMLR, 2019.

Li, Q., Han, Z., and Wu, X.-M. Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI*, 2018.

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, 2020.

Lloyd, S. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pp. 6489–6499. PMLR, 2020.

Nytko, N., Taghibakhshi, A., Zaman, T. U., MacLachlan, S., Olson, L. N., and West, M. Optimized sparse matrix operations for reverse mode automatic differentiation. *arXiv preprint arXiv:2212.05159*, 2022.

Oono, K. and Suzuki, T. Graph neural networks exponentially lose expressive power for node classification. *arXiv preprint arXiv:1905.10947*, 2019.

Quarteroni, A. and Valli, A. *Domain decomposition methods for partial differential equations*. Numerical Mathematics and Scientific Computation. The Clarendon Press, Oxford University Press, New York, 1999. ISBN 0-19-850178-1. Oxford Science Publications.

Ranjan, E., Sanyal, S., and Talukdar, P. Asap: Adaptive structure aware pooling for learning hierarchical graph representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5470–5477, 2020.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer Assisted Intervention*, pp. 234–241. Springer, 2015.

Schlömer, N. pygmsh: A Python frontend for Gmsh, 2021. URL `https://github.com/nschloe/pygmsh`. (GPL-3.0 License).

Schmitt, J., Kuckuk, S., and Köstler, H. Evostencils: a grammar-based genetic programming approach for constructing efficient geometric multigrid methods. *Genetic Programming and Evolvable Machines*, 22:511–537, 2021.

St-Cyr, A., Gander, M. J., and Thomas, S. J. Optimized multiplicative, additive, and restricted additive Schwarz preconditioning. *SIAM Journal on Scientific Computing*, 29(6):2402–2425, 2007.

Taghibakhshi, A., MacLachlan, S., Olson, L., and West, M. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Taghibakhshi, A., Nytko, N., Zaman, T., MacLachlan, S., Olson, L., and West, M. Learning interface conditions in domain decomposition solvers. *arXiv preprint arXiv:2205.09833*, 2022.

Toselli, A. and Widlund, O. *Domain decomposition methods—algorithms and theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005. ISBN 3-540-20696-5. doi: 10.1007/b137868.

Vos, J., Rizzi, A., Darracq, D., and Hirschel, E. Navier–stokes solvers in european aircraft design. *Progress in Aerospace Sciences*, 38(8):601–697, 2002.

Wan, W. L., Chan, T. F., and Smith, B. An energy-minimizing interpolation for robust multigrid methods. *SIAM Journal on Scientific Computing*, 21(4):1632–1649, 1999.

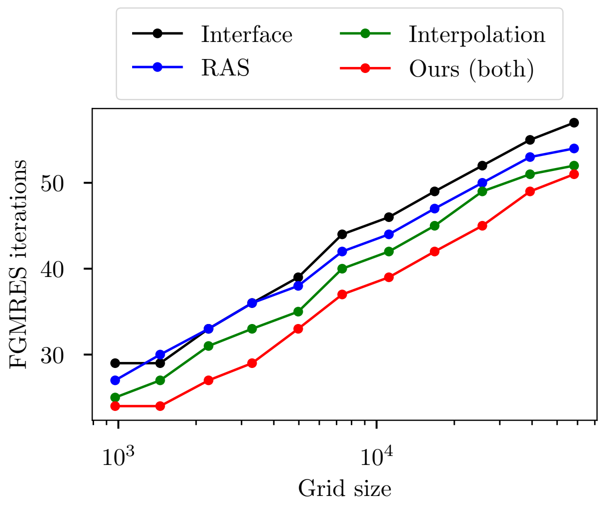
Wang, W., Dang, Z., Hu, Y., Fua, P., and Salzmann, M. Backpropagation-friendly eigendecomposition. *Advances in Neural Information Processing Systems*, 32, 2019.

Xu, J. Iterative methods by space decomposition and subspace correction. *SIAM Review*, 34(4):581–613, 1992.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. *Advances in Neural information processing systems*, 31, 2018.

A. FGMRES plots

In Section 5, in Figures 5 to Figure 8, the performance of the methods was evaluated by considering the convergence of stationary iterations. Here, we present another possible evaluation criterion, assessing the number of iterations to convergence for the preconditioned systems using FGMRES, a standard Krylov method. The following figures are analogous to those provided in the main paper, and demonstrate that our method also achieves superior results compared to other methods, and that the MG-GNN architecture outperforms graph U-nets.



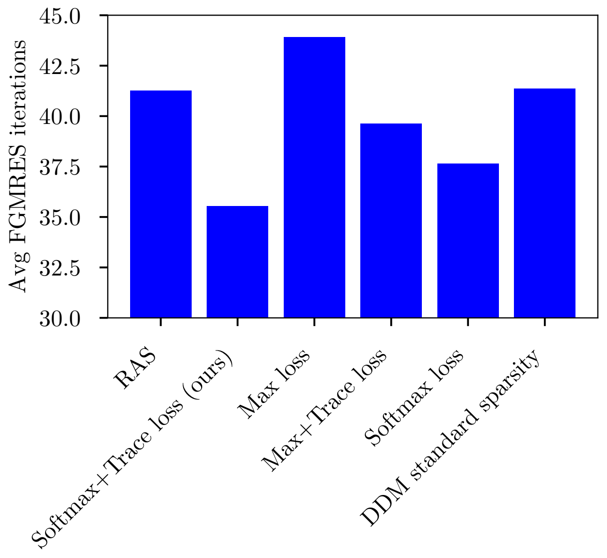
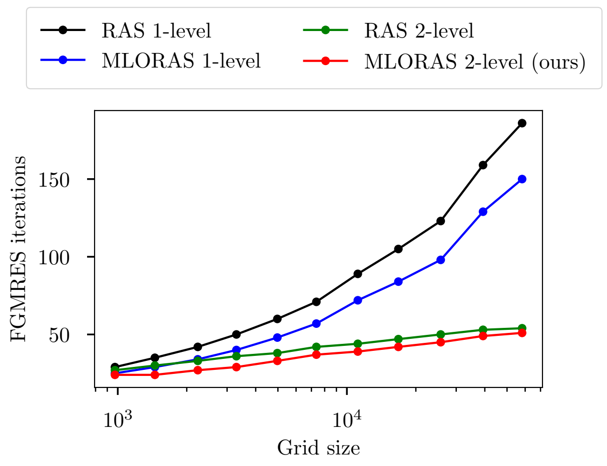


Figure 9. Left: Effect of learning interface values, interpolation operator, or both on FGMRES iterations. All three variants outperforming the RAS baseline are utilizing modifications introduced in this paper (15) compared to the "Max loss" from Taghibakhshi et al. (2022). Right: Effect of every ingredient in the model on average FGMRES iterations.



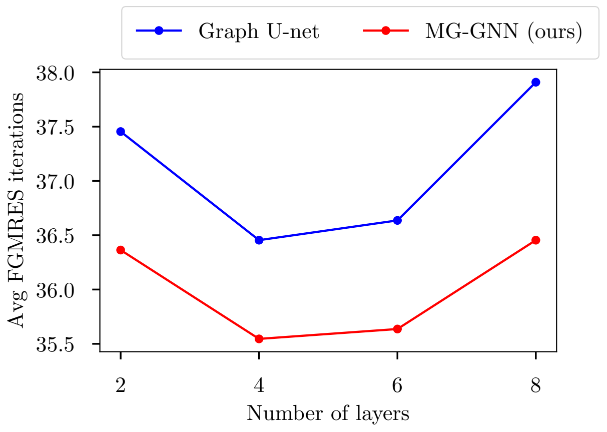


Figure 10. Left: Comparison of FGMRES iterations of 2-level methods with 1-level methods from Taghibakhshi et al. (2022). Right: Average FGMRES iterations of graph U-net and MG-GNN with different number of layers on the test set.

B. Model architecture

Inputs and outputs: The model takes any unstructured grid as its input, which consists of the node features, edge features, and adjacency matrix of both the fine and coarse grids. Every node on the fine level has a binary feature, indicating whether it lies on the boundary of a subdomain. Fine level edge features are obtained from the discretization of the underlying PDE, A, and the adjacency matrix of the fine level is simply the sparsity of A. Similar attributes for the coarse level are obtained as described in Section 3, Equations (6) and (7), and Lloyd aggregation has been used for obtaining subdomains throughout. The outputs of the model are the learned interface values and the interpolation operator.

We use node and edge preprocessing (3 fully connected layers of dimension 128, followed by ReLU activations, in the node and feature space, respectively) followed by 4 layers of MG-GNN. For GNN$^{(\ell)}$ in (10) and $F^{\ell\to\ell}$ in (8), we use a TAGConv layer (Du et al., 2017) and, for $F^{\ell\to k}$ with $\ell \neq k$, we use a heterogeneous message passing GNN as shown in Equation (11). Specifically, we choose summation as the permutation invariant operator in (11) and, for the MLPs, we use two fully connected layers of size 128 with ReLU nonlinearity for $f^{\ell\to k}$ and $g^{\ell\to k}(x,y) = x$.

Following the MG-GNN layers, the network will split in two heads, each having a stack layer (which essentially concatenates the features of nodes on each side of every edge) and an edge feature post-processing (see Figure 12 for details). The edge weights between the coarse and fine level are the learned interpolation operator weights, and the edge values along the subdomains in the fine level are the learned interface values. The upper head of the network has a masking block at the end, which masks the edge values that are not along the boundary, hence only outputting the learned interface values. The overall GNN architecture for learning the interpolation operator and the interface values is shown in Figure 11.

Note: AS stated in the paper, the inter-level transfers and operators ($R^{(\ell)}$) are obtained by the pre-trained Lloyd clustering; in other words, every coarse node is connected to all of its underlying subdomain nodes. Moreover, note that in the two level method, the aggregation in DDM is the same as MG-GNN, and in the general MG-GNN architecture (more than 2-level), still the first level aggregation is the same as DDM, and higher levels are obtained by Lloyd again.

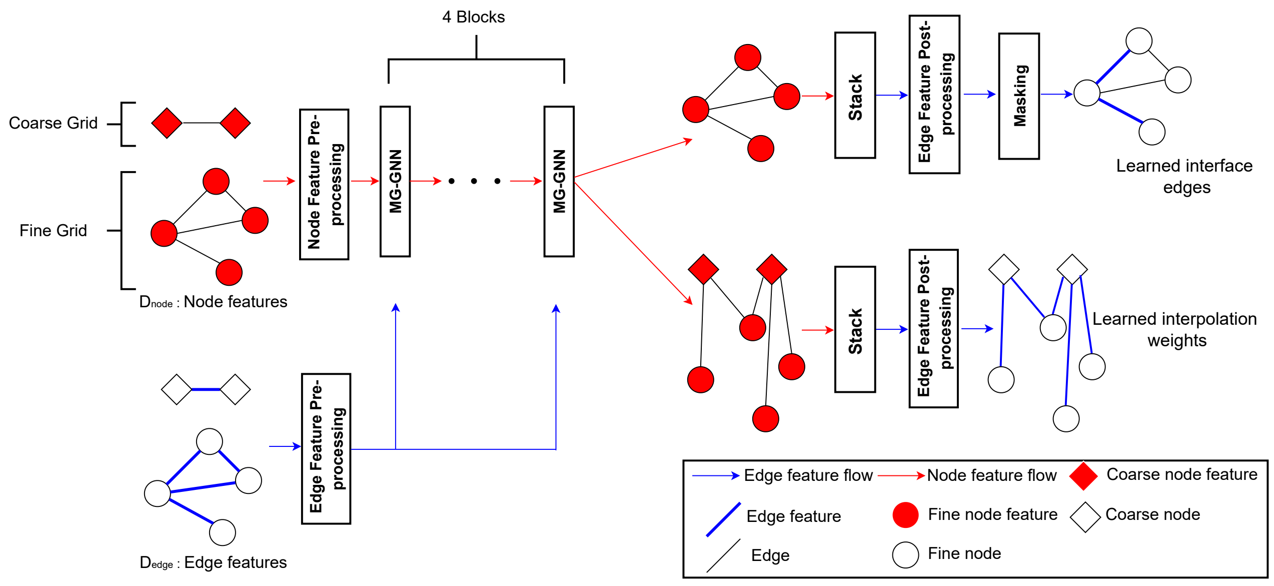


Figure 11. GNN architecture used in this study.

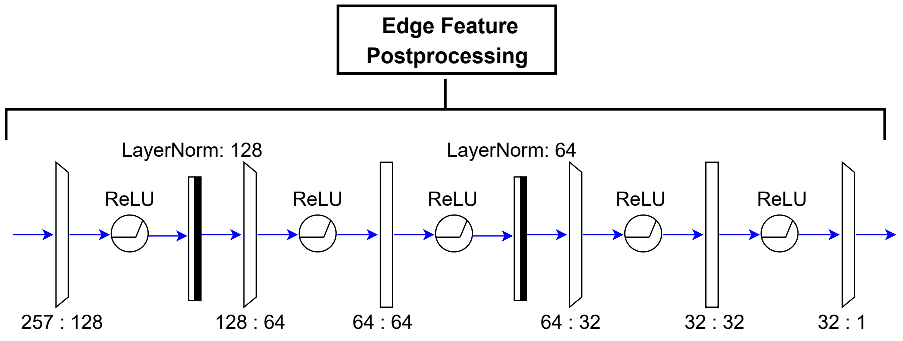


Figure 12. Edge feature post-processing block.

C. Runtime

We find that the (offline) training phase (20 epochs of training on a set of 1000 grids) takes about 7.86 hours. During the (online) testing phase, obtaining the preconditioners (setup of all parts of the preconditioner once the system matrix is given) takes 11.44 seconds for classical RAS and 13.07 seconds for our MLORAS (averaged over 11 tests). However, we find that our method makes up for this extra online time in significantly reduced solve times across a range of grid sizes, both as a stationary iteration and as a preconditioner for FGMRES. We note that the additional online setup time (and even, to some extent, the offline time) can also be amortized when solving problems involving time-stepping or optimal control, where the same linear system needs to be solved with many right-hand sides. The table below shows the stationary solve times on some of the test grids (in seconds):

Test grid size	972	2235	4972	11161	25677	58186
2-level RAS	0.074	0.256	0.839	2.817	10.718	40.089
2-level MLORAS (ours)	0.039	0.116	0.47	1.597	6.986	28.75

D. More problems

We have run tests with more PDEs, including discontinuous and anisotropic diffusion coefficients, as described below, and propose to include these results in the revised manuscript (and supplementary materials, as space permits). We note that in terms of iterations to convergence, we see our approach clearly win over RAS for both stationary iterations and preconditioned FGMRES.

For each of the following problems, we generate training and testing sets with the same parameters for meshes as in the existing results in the main paper, with training grids of size 800-1000 nodes, and testing grids of size 1000-60000 nodes. We average iterations-to-convergence over 11 problems in the testing sets. Our proposed additional problems are:

1- Jump-discontinuous diffusion:

$$-\nabla \cdot \kappa(x,y)\nabla u = f \quad \text{in } \Omega, \quad \kappa(x,y) = \begin{cases} 100 & 0 < x < 0.5 \\ 1 & 0.5 \le x < 1. \end{cases}$$

2- Rotated anisotropic diffusion:

$$-\nabla \cdot (T\nabla u) = f$$

where $T = \begin{bmatrix} \cos^2(\theta) + \xi\sin^2(\theta) & \cos(\theta)\sin(\theta)(1-\xi) \\ \cos(\theta)\sin(\theta)(1-\xi) & \sin^2(\theta) + \xi\cos^2(\theta) \end{bmatrix}$. We use parameters $0.1 < \xi < 10$ and $0 < \theta < \frac{\pi}{4}$ to denote the strength of anisotropy and rotation direction for the problems, with these values chosen uniformly random.

3- Low-diffusion inclusion:

$$-\nabla \cdot \kappa(x,y)\nabla u = f \quad \text{in } \Omega, \quad \kappa(x,y) = \begin{cases} 10^{-8} & \frac{1}{3} < x, y < \frac{2}{3} \\ 1 & \text{else.} \end{cases}$$

Considering stationary iterations to convergence, we see improvements of 20-45%, as summarized in the following table:

Model	Jump discontinuous diffusion	Rotated anisotropic diffusion	Low-diffusion inclusion
2-level RAS	72.81	78.09	77.36
2-level MLORAS (ours)	58.18	54.09	52.36

As is typical, the improvements are somewhat less large for preconditioned FGMRES iterations to convergence, but are still around 10%:

Model	Jump discontinuous diffusion	Rotated anisotropic diffusion	Low-diffusion inclusion
2-level RAS	39.72	42.18	40.45
2-level MLORAS (ours)	37.45	36.72	37.72

E. Spectral radius metric

Spectral radius of the error propagation matrix is often a better criterion to use in comparing methods, particularly for convergence of stationary iterations. However, it is often difficult to measure this directly, particularly for problems larger than a few thousand nodes. To estimate the spectral radius, we run 200 iterations of the stationary iteration on a problem with a zero right-hand side and random initial guess, and measure the effective per-cycle reduction in error, averaged over the final 10 iterations. For suitably small problems, we can compare this against a direct eigenvalue calculation on the preconditioned system matrix, and find that they agree perfectly. Below, we show the measured spectral radii for several test problems (Poisson on grids of differing sizes), showing that our method always achieves a reduction in the spectral radius in comparison to classical RAS:

Test grid size	972	2235	4972	11161	25677	58186
2-level RAS	0.671	0.759	0.809	0.854	0.887	0.903
2-level MLORAS (ours)	0.412	0.574	0.715	0.78	0.844	0.876