ADVANCING DOMAIN DECOMPOSITION METHODS AND ENTITY RESOLUTION WITH GRAPH NEURAL NETWORKS

BY

ALI TAGHIBAKHSHI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Doctoral Committee:

Professor Matthew West, Chair
Professor Srinivasa Salapaka, Chair
Professor Luke Olson, Chair
Professor Scott MacLachlan, Memorial University of Newfoundland

Abstract

Domain decomposition methods (DDMs) are popular solvers for discretized systems of PDEs, with one-level and multilevel variants. Yet the *optimal* construction of these methods requires tedious analysis and is often available only in simplified, structured-grid settings, limiting their use for more complex problems. These solvers rely on several algorithmic and mathematical parameters, prescribing overlap, subdomain boundary conditions, and other properties of the DDM. In the first two sections of this study, we develop methods for learning optimal one and two-level DDMs utilizing graph neural networks (GNNs). Key components of our proposed methods include novel loss functions enjoying theoretical guarantees to converge to global optimum in limits.

First, we generalize optimized Schwarz domain decomposition methods to unstructured-grid problems, using Graph Convolutional Neural Networks (GCNNs) and unsupervised learning to learn optimal modifications at subdomain interfaces. A key ingredient in our approach is an improved loss function, enabling effective training on relatively small problems, but robust performance on arbitrarily large problems, with computational cost linear in problem size. The performance of the learned linear solvers is compared with both classical and optimized domain decomposition algorithms, for both structured- and unstructured-grid problems.

Second, 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.

Cross-device user matching is a critical problem in numerous domains, including advertising, recommender systems, and cybersecurity. It involves identifying and linking different devices belonging to the same person, utilizing sequence logs. Previous data mining techniques have struggled to address the long-range dependencies and higher-order connections between the logs. Recently, researchers have modeled this problem as a graph problem and proposed a two-tier graph contextual embedding (TGCE) neural network architecture, which outperforms previous methods. In the last section of this study, we propose a novel hierarchical graph neural network architecture (HGNN), which has a more computationally efficient second level design than TGCE. Furthermore, we introduce a cross-attention (Cross-Att) mechanism in our model, which improves performance by 5% compared to the state-of-the-art TGCE method.

To my Father, Mother, and Wife.

Acknowledgments

This project would not have been possible without the support of many people. First, many thanks to my adviser, Matthew West, and my lead co-authors, Luke Olson and Scott MacLachlan for their invaluable guidance and supervision as well as creating a productive and friendly environment for great research. I would like to also thank Nicolas Nytko and Tareq Zaman for helping me along different projects. Also thanks to my committee member, Srinivasa Salapaka for his guidance and support. And finally, thanks to my wife, parents, and numerous friends who endured this long process with me, always offering unconditional support and love.

Table of contents

List of Abbreviations

GNN	Graph Neural Network.
MGGNN	Multigrid Graph Neural Network.
HGNN	Hierarchical Graph Neural Network.
DDM	Domain Decomposition Method.
RAS	Restricted Additive Schwarz.
ORAS	Optimized Restricted Additive Schwarz.
AMG	Algebraic Multigrid.
ML	Machine Learning.

Chapter 1

Introduction

Domain decomposition methods (DDMs) [1]–[3] are highly effective in solving the linear and nonlinear systems of equations that arise from the numerical approximation of solutions to PDEs. While most effective on elliptic boundary-value problems, DDMs can also be applied to nonlinear problems, either using their nonlinear variants, or successively solving linearizations. Time-dependent problems are normally solved by using a time stepping algorithm in the time domain for implicit methods, which require the solution of a spatial problem for each time step. Of these methods, Schwarz methods are particularly popular given their relative simplicity and ease of parallelization. The common theme is to break the global problem into subproblems, derived either by projection or by discretizing the same PDE over a physical subdomain, and to use solutions on the subdomains as a preconditioner for the global discretization. Classical Schwarz methods generally make use of Dirichlet or Neumann boundary conditions for these subdomain problems, while Optimized Schwarz Methods (OSMs) aim to improve the convergence of the algorithm by using more general interface conditions [4]. Notably, [5] demonstrates that optimal, but non-local, interface conditions exist for more general decompositions.

Much of the OSM literature considers only one-level additive Schwarz methods, although multilevel extensions do exist. For one-level methods (i.e., domain decomposition approaches without a "coarse grid"), restricted additive Schwarz (RAS) approaches [6] are arguably the most common; optimized restricted additive Schwarz (ORAS) methods are considered in [7]. The OSM idea has also been extended to asynchronous Schwarz methods [8], where the computations on each subdomain are done using the newest information available in a parallel computing environment without synchronizing the solves on each subdomain.

With a recent focus on machine learning (ML) techniques for solving PDE systems [9], [10], there is also effort to apply learning-based methods to improve the performance of iterative solvers for PDEs, including DDM and algebraic multigrid (AMG) methods. Within AMG methods, ML techniques have been applied to learning interpolation operators [11], [12] and to coarse-grid selection in reduction-based AMG [13]. Of particular note here is the loss function employed in [11], [12], where they use unsupervised learning to train a graph neural network to minimize the Frobenius norm of the error-propagation operator of their iterative method. Within DDM, significant effort has been invested in combining ML techniques with DDM, as in [14], where two main families of approaches are given: 1) using ML within classical DDM methods to improve convergence, and 2) using deep neural networks, such as Physics Informed Neural Networks (PINNs) [9], as a discretization module and solver for DDM problems. In [15], a fully connected neural network is used to predict the geometric locations of constraints for coarse space enhancements in an adaptive Finite Element

Tearing and Interconnecting-Dual Primal (FETI-DP) method. Using the continuous formulation of DDM, the so-called D3M [16] uses a variational deep learning solver, implementing local neural networks on physical subdomains in a parallel fashion. Likewise, Deep-DDM [17] utilizes PINNs to discretize and solve DDM problems, with coarse space corrections [18] being used to improve scalability.

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 [19], [20]. 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 [21], graph U-nets [22] 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 [23], 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 [24] and [25]. 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.

It is a common occurrence for users to engage in online activities across multiple devices. However, businesses and brands often struggle with having insufficient user identities to work with since users are perceived as different individuals across different devices due to their unique activities. The ability to automatically identify the same user across multiple devices is essential for gaining insights into human behavior patterns, which can aid in applications such as user profiling, online advertising, improving system security. Therefore, in recent years, the has been a flourishing amount of studies focusing on cross-device user matching [26].

In recent years, with the advent of machine learning-based methods for entity resolution, several studies have focused on learning distributed embeddings for the devices based on their URL logs [27]–[29]. The earlier studies focused on utilizing unsupervised feature learning techniques [30], developing handcrafted features for the device logs, or relied on co-occurrence of key attributes of URL logs in pairwise classification [27].

Methods that utilize deep learning have a greater ability to convey dense connections among the sequential device logs. For instance, researchers have utilized a 2D convolutional neural network (CNN) framework to encode sequential log representations to understand the relationship between two devices [31]. However, this model primarily captures local interactions within user sequence logs, limiting its ability to learn the entire

sequence or a higher-level pattern. Recently, there has been further emphasis on the effectiveness of sequential models like recurrent neural networks (RNNs) and attention-based techniques in modeling sequence patterns and achieving promising results in numerous sequence modeling tasks [32]–[34]. Although these methods work well for sequence modeling, they are not specifically designed for user-matching tasks and may not be optimal for learning sequential log embeddings.

Recently, researchers proposed a two-tier graph contextual embedding (TGCE) network for the cross-device user matching [29] task. While previous methods for the task often failed at long-range information passing along the sequence logs, TGCE leverages a two-level structure that can facilitate information passing beyond the immediate neighborhood of a device log. This was specifically achieved by considering a random walk starting from every node in a device log, connecting all of the visited nodes to the original node, and performing a round of message passing using the newly generated shortcut edges.

Although the two-tier structure seems to enable long-range information sharing, we note two major limitations with the existing method. First, in the device graph, the random walk on the URL nodes may randomly connect two URLs that have been visited at two far-away time-stamps. Intuitively, two different URLs browsed by a device with weeks of gap in between share less information than two URLs visited in a shorter time frame. Second, at the end of the TGCE architecture, for the pairwise classification task, the generated graph embeddings for two devices are entry-wise multiplied and sent through a fully connected network to determine if they belong to the same person. However, there could be significant key features in the learned embeddings that may be shared between the devices, which can alternatively get lost if the architecture does not compare them across one another.

In the second chapter, we advance DDM-based solvers by developing a framework for learning optimized Schwarz preconditioners. A key aspect of this is reconsidering the loss function to use a more effective relaxation of the ideal objective function than Frobenius norm minimization [11], [12]. Moreover, the approach introduced here offers an opportunity to reconsider existing limitations of optimized Schwarz methods, where optimal parameter choice is based on Fourier analysis and requires a highly regular subdomain structure, such as in the classical cases of square domains split into two equal subdomains or into regular grids of square subdomains. Our framework learns the optimized Schwarz parameters via training on small problem sizes, in a way that generalizes effectively to large problems, and in a way that allows us to consider both irregular domains and unstructured grids, with no extraordinary limitations on subdomain shape. Furthermore, the evaluation time of our algorithm scales linearly with problem size. This allows significant freedom in defining optimized Schwarz methods, in comparison to classical approaches, allowing us to explore the potential benefits of optimized Schwarz over classical (restricted) additive methods on unstructured grids for the first time.

In the third chapter, 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. Our MG-GNN architecture that outperforms existing hierarchical GNN architectures and scales linearly with problem size.Moreover, we improve the loss function with theoretical guarantees essential for training two-level Schwarz methods. Our

method is also scalable, leading to effective training on small problems and generalization to problems that are orders of magnitude larger. Finally, our method outperforms classical two-level RAS, both as stationary algorithm and as a preconditioner for the flexible generalized minimum residual (FGMRES) iteration.

In the fourth chapter of this study, we propose a new hierarchical graph neural network (HGNN) inspired by the star graph architecture [35]. In the terminology of HGNN, we refer to the URL nodes as *fine* nodes, and in an unraveled sequence of URL logs, HGNN assigns a *coarse* node to every K consecutive fine nodes. The message passing between the coarse and fine nodes enables effective long-range message passing without the need to excessively add edges, as in the random walk method. Moreover, for the pairwise classification task, we utilize a cross-attention mechanism inspired by Li *et. al.* [36], which enables entry-wise cross-encoding of the learned embeddings. The main two-fold contributions of this chapter is that we model a given device log as a hierarchical heterogeneous graph, which is 6x faster than the previous state-of-the-art while keeping a competitive level of accuracy and performance, and that we employ a cross-attention mechanism for pairwise matching of the graphs associated with a device log, which improves the accuracy of the overall method by about 5%.

Chapter 2

Background on Domain Decomposition Method

The domain decomposition method is a numerical method for solving partial differential equations (PDEs) by dividing their computational domain into smaller subdomains and solving the PDEs independently on each subdomain. This approach is particularly useful for solving large-scale problems that can be parallelized, as it allows for distributed computing and can significantly reduce computational costs [37].

Consider a PDE defined on a domain Ω with boundary $\partial\Omega$. The domain decomposition method divides Ω into N non-overlapping subdomains Ω_i, each with its specific boundary $\partial\Omega_i$. The union of all these subdomains embodies the entire domain; in other words $\Omega = \bigcup_{i=1}^{N} \Omega_i$, and the boundaries of the subdomains satisfy $\partial\Omega = \bigcup_{i=1}^{N} \partial\Omega_i$.

The intuition behind domain decomposition methods is to solve the PDE independently on each subdomain and share information at the interfaces of the subdomains to make sure they are compatible and accurate for the overall solve. This passing of information is typically done iteratively until a global solution is obtained [38], [39].

Mathematically, let u_i represent the solution on subdomain Ω_i. The domain decomposition method aims to find $u = (u_1, u_2, \ldots, u_N)$ that satisfies the following conditions:

1. The solution u_i on each subdomain Ω_i solves the PDE locally, i.e., $\mathcal{L}_i(u_i) = 0$ for $i = 1, 2, \ldots, N$, where $\mathcal{L}_i$ is the local PDE operator on subdomain Ω_i.

2. The solution u_i on each subdomain agrees with the neighboring solutions u_j on the interfaces, i.e., $u_i|_{\partial\Omega_i \cap \partial\Omega_j} = u_j|_{\partial\Omega_i \cap \partial\Omega_j}$ for all i, j.

3. The global solution u satisfies the boundary conditions on $\partial\Omega$, i.e., $u|_{\partial\Omega} = g$, where g is the given boundary data.

To achieve these conditions, various domain decomposition methods can be employed, such as the overlapping and non-overlapping methods. These methods are different in terms of how they handle the passing of information along the interfaces and the coupling of the subdomain solutions.

One popular approach is the classical Schwarz method [40], which falls under the non-overlapping category and is explained in the next section.

2.1 Schwarz Domain Decomposition Method

The classical Schwarz method divides the original problem's domain into subdomains and solves the PDE on each subdomain independently. At each iteration, the solutions on the subdomains are updated considering information from its neighboring subdomains [40].

Consider a simple example of a 1D PDE defined on the interval $\Omega = (0, 1)$. We divide Ω into N subdomains $\Omega_i = (\frac{i-1}{N}, \frac{i}{N})$, where $i = 1, 2, \ldots, N$. The PDE is given as $\frac{d^2u}{dx^2} = f(x)$, with the boundary conditions $u(0) = u(1) = 0$.

To solve this problem using the classical Schwarz method, we start with an initial guess for the solution u_i^0 on each of the subdomain. At each iteration k, the solution on each subdomain is updated by considering information from its neighboring subdomains. The update step is written as follows:

$$u_i^{k+1} = \mathcal{L}_i^{-1} \left(f_i - \sum_{j=1, j\neq i}^{N} \mathcal{T}_{ij} u_j^k \right)$$

Here, $\mathcal{L}_i^{-1}$ is the inverse of the local PDE operator on subdomain Ω_i, f_i is the local source term on Ω_i, $\mathcal{T}_{ij}$ represents the transmission operator that transfers information between subdomains i and j, and u_j^k is the solution on subdomain Ω_j at iteration k. The transmission operator is for making sure the solutions at the interfaces are compatible.

2.1.1 Dirichlet-Schwarz Method

The Dirichlet-Schwarz Method considers Dirichlet boundary conditions at the subdomain interfaces [41]. Consider a PDE problem defined on a domain Ω. The goal is to find the solution u that satisfies the equation

$$\mathcal{L}(u) = f \quad \text{in } \Omega, \tag{2.1}$$

where $\mathcal{L}$ is a differential operator and f is a given source term. We impose Dirichlet boundary conditions, $u = g$, on the boundary $\partial\Omega$.

The domain Ω is decomposed into N subdomains, $\Omega_1, \Omega_2, \ldots, \Omega_N$. These subdomains can overlap, and each subdomain has its own local coordinate system. The overlapping regions are called interface regions or overlap regions, denoted by Γ_{ij}, where i and j represent the subdomains sharing the interface.

For applying the Dirichlet-Schwarz method, we consider the decomposition $u = u_1 + u_2 + \cdots + u_N$, where u_i is the solution on the i-th subdomain. By substituting this decomposition into Equation (2.1) and using the linearity of $\mathcal{L}$, we obtain a system of equations for each subdomain:

$$\mathcal{L}_i(u_i) = f \quad \text{in } \Omega_i, \tag{2.2}$$

subject to the interface conditions:

$$u_i = u_j \quad \text{on } \Gamma_{ij}, \tag{2.3}$$

$$\mathcal{L}_i(u_i) \cdot n_i = \mathcal{L}_j(u_j) \cdot n_j \quad \text{on } \Gamma_{ij}, \tag{2.4}$$

where n_i and n_j are the outward unit normals to Γ_{ij} for subdomains Ω_i and Ω_j respectively.

To solve the system of equations (2.2) subject to the interface conditions (2.3) and (2.4), an iterative procedure is employed. The Dirichlet-Schwarz method iterates between solving the local problems on each of the subdomains and updates the interface values until it achieves convergence. The method starts with an initial guess for the solution on each subdomain, $u_i^{(0)}$. At each iteration, the following steps are performed:

1. Solve the local problem on each subdomain Ω_i independently:

$$\mathcal{L}_i(u_i^{(k+1)}) = f \quad \text{in } \Omega_i. \tag{2.5}$$

2. Update the interface values using the updated solutions:

$$u_i^{(k+1)} = u_j^{(k+1)} \quad \text{on } \Gamma_{ij} \quad \text{for all } i, j. \tag{2.6}$$

The iterations continue until convergence, and is is commonly based on the difference between consecutive iterations or the residual norm. The Dirichlet-Schwarz Method is naturally employed for parallel implementation since solutions of local problems on each subdomain are independent.

2.1.2 Neumann-Schwarz Method

The Neumann-Schwarz method is another variant of the Schwarz domain decomposition method that considers Neumann boundary conditions at the interfaces of the subdomains [42]. In this method, we consider a PDE problem as in (2.1), and at the boundary $\partial\Omega$, we consider Neumann boundary conditions of the form:

$$\mathcal{N}(u) \cdot n = g \quad \text{on } \partial\Omega, \tag{2.7}$$

where $\mathcal{N}$ represents the Neumann operator, n is the outward unit normal vector, and g is the given boundary data.

Similar to the Dirichlet-Schwarz method, the Neumann-Schwarz method decomposes the domain Ω into N subdomains, $\Omega_1, \Omega_2, \ldots, \Omega_N$, and the subdomains can be overlapping or not. The interface operators Γ_{ij} denote the overlapping between subdomains i and j.

To apply the Neumann-Schwarz method, we introduce the decomposition $u = u_1 + u_2 + \cdots + u_N$, where u_i is the solution on the i-th subdomain. By applying this decomposition into the PDE, and since the operator $\mathcal{L}$ is linear, we obtain a system of equations for each subdomain:

$$\mathcal{L}_i(u_i) = f \quad \text{in } \Omega_i, \tag{2.8}$$

subject to the interface conditions:

$$\frac{\partial u_i}{\partial n_i} = \frac{\partial u_j}{\partial n_j} \quad \text{on } \Gamma_{ij}, \tag{2.9}$$

where $\frac{\partial u_i}{\partial n_i}$ and $\frac{\partial u_j}{\partial n_j}$ denote the normal derivatives of u_i and u_j with respect to the respective outward unit normal vectors n_i and n_j.

The Neumann-Schwarz method solves the system of equations and applies the interface conditions iteratively. The method starts with an initial guess for the solution on each subdomain, denoted by $u_i^{(0)}$. At each iteration, the following steps are performed:

1. Solve the local problem on each subdomain Ω_i independently:

$$\mathcal{L}_i(u_i^{(k+1)}) = f \quad \text{in } \Omega_i. \tag{2.10}$$

2. Enforce the interface conditions by ensuring the consistency of normal derivatives:

$$\frac{\partial u_i^{(k+1)}}{\partial n_i} = \frac{\partial u_j^{(k+1)}}{\partial n_j} \quad \text{on } \Gamma_{ij} \quad \text{for all } i, j. \tag{2.11}$$

The iterations continue until convergence, typically based on the difference between successive iterates or the residual. The Neumann-Schwarz method can be implemented in a parallel fashion since the solution of local problems on each subdomain are independent.

2.1.3 Robin-Schwarz Method

The Robin-Schwarz method is an extension of the Schwarz domain decomposition method that considers Robin boundary conditions at the subdomain interfaces, and it is used to solve PDEs with Robin boundary conditions in a domain decomposition framework [43]. Consider a PDE problem defined on a domain Ω as in (2.1) but subject to the Robin boundary conditions:

$$\alpha(u) + \beta(u) \cdot n = g \quad \text{on } \partial\Omega, \tag{2.12}$$

where $\mathcal{L}$ is a differential operator, f is a given source term, $\alpha(u)$ and $\beta(u)$ are functions of the solution u, and g is the given boundary data. The term $\alpha(u)$ denotes the coefficient of the normal derivative, and $\beta(u)$ is the coefficient of the solution itself in the Robin condition.

Similar to the Dirichlet-Schwarz and Neumann-Schwarz methods, the Robin-Schwarz method decomposes the domain Ω into N subdomains, $\Omega_1, \Omega_2, \ldots, \Omega_N$, with possible overlap between them. Similar to prior subsections, Γ_{ij} represent the overlapping regions between subdomains i and j, and similarly, to apply the Robin-Schwarz method, we introduce the decomposition $u = u_1 + u_2 + \cdots + u_N$, where u_i is the solution on the i-th subdomain. By substituting this decomposition into the PDE equation, we obtain a system of equations for each subdomain:

$$\mathcal{L}_i(u_i) = f \quad \text{in } \Omega_i, \tag{2.13}$$

subject to the interface conditions:

$$\alpha(u_i) + \beta(u_i) \cdot n_i = \alpha(u_j) + \beta(u_j) \cdot n_j \quad \text{on } \Gamma_{ij}, \tag{2.14}$$

where n_i and n_j are the outward unit normal vectors to Γ_{ij} for subdomains Ω_i and Ω_j, respectively.

The Robin-Schwarz method employs an iterative procedure to solve the system of equations and enforce the interface conditions. The method starts with an initial guess for the solution on each subdomain, denoted as $u_i^{(0)}$. At each iteration, the following steps are performed:

1. Solve the local problem on each subdomain Ω_i independently:

$$\mathcal{L}_i(u_i^{(k+1)}) = f \quad \text{in } \Omega_i. \tag{2.15}$$

2. Enforce the interface conditions by ensuring the compatibility of the Robin conditions:

$$\alpha(u_i^{(k+1)}) + \beta(u_i^{(k+1)}) \cdot n_i = \alpha(u_j^{(k+1)}) + \beta(u_j^{(k+1)}) \cdot n_j \quad \text{on } \Gamma_{ij} \quad \text{for all } i, j. \tag{2.16}$$

The iterations continue until a convergence criterion is met, typically based on the difference between successive iterates or the residual. The Robin-Schwarz method allows for parallel implementation due to the independent solution of local problems on each subdomain, and moreover, various parallel strategies can be employed, such as domain decomposition techniques, message passing interfaces, or hybrid approaches.

2.1.4 Optimized Schwarz Method (OSM)

The Optimized Schwarz Method (OSM) is a more advanced subcategory of the Schwarz domain decomposition method which is developed to improve the convergence rate and efficiency of the classical Schwarz methods [44]–[46]. To achieve this goal, OSM optimizes the transmission conditions along the subdomain interfaces and enhances the information sharing between the subdomains. In the traditional Schwarz method, the transmission operator $\mathcal{T}_{ij}$ is often chosen as a simple restriction or interpolation operator. However, OSM introduces an optimization procedure to determine the transmission conditions that minimize the error and maximize the convergence rate.

To apply the Optimized Schwarz Method, we start with the initial guess u_i^0 for the solution on each subdomain Ω_i. The algorithm proceeds through iterations, where at each iteration k, the following steps are performed:

1. Update the interface values using the previous iteration's solutions:

$$u_i^k = u_j^{k-1} \quad \text{on } \Gamma_{ij} \quad \text{for all } i, j. \tag{2.17}$$

2. Solve the local problem on each subdomain Ω_i independently:

$$u_i^{k+1} = \mathcal{L}_i^{-1} \left(f_i - \sum_{j=1, j\neq i}^{N} \mathcal{T}_{ij} u_j^k \right), \tag{2.18}$$

where $\mathcal{L}_i^{-1}$ is the inverse of the local PDE operator on subdomain Ω_i, f_i is the local source term on Ω_i, $\mathcal{T}_{ij}$ represents the optimized transmission operator between subdomains i and j, and u_j^k is the solution on subdomain Ω_j at iteration k.

The key difference in OSM is the optimized transmission operator $\mathcal{T}_{ij}$. This operator is computed using optimization techniques to minimize the error and improve the overall convergence. By selecting the optimal transmission conditions, OSM improves the information passing between subdomains, leading to faster convergence and improved accuracy.

The optimization procedure for determining $\mathcal{T}_{ij}$ involves solving additional problems and applying appropriate optimization algorithms. The goal is to find the transmission conditions that balance the accuracy and computational efficiency of the method. Several optimization strategies, such as domain decomposition techniques, model reduction, or adaptive algorithms, can be employed to enhance the performance of OSM.

2.1.5 Additive Schwarz Methods

The Additive Schwarz Method (AS) is a domain decomposition technique for solving PDEs that has offers parallelization, and improves convergence features [47]. AS decomposes the computational domain into non-overlapping subdomains and solves the problem independently on each subdomain before combining the solutions to obtain the overall solution. To apply the Additive Schwarz method, we divide the domain Ω into N subdomains, denoted as $\Omega_1, \Omega_2, \ldots, \Omega_N$. Each subdomain covers a specific part of the overall global domain, and the union of all subdomains covers the whole computational domain. The subdomains can be defined based on the geometric decomposition or other criteria appropriate for the considered problem.

The AS algorithm iteratively solves the problem, where at each iteration k, the following steps are performed:

1. Initialize the solution on each subdomain:

$$u_i^{k,0} = u^{k-1} \quad \text{on } \partial\Omega_i \quad \text{for all } i, \tag{2.19}$$

   where u^{k-1} represents the solution from the previous iteration.

2. Solve the local problem on each subdomain Ω_i independently:

$$u_i^{k,m} = \mathcal{L}_i^{-1}\left(f_i - \mathcal{L}_i u_i^{k,m-1} + \sum_{j=1, j\neq i}^{N} \mathcal{B}_{ij} u_j^{k,m-1}\right), \tag{2.20}$$

   where $\mathcal{L}_i^{-1}$ is the inverse of the local PDE operator on subdomain Ω_i, f_i is the local source term on Ω_i, $\mathcal{L}_i$ represents the local PDE operator, $\mathcal{B}_{ij}$ is the restriction operator that restricts the solution from subdomain Ω_j to the interface Γ_{ij} between subdomains i and j, and $u_j^{k,m-1}$ is the solution on subdomain Ω_j at iteration k and level $m-1$.

3. Combine the local solutions to obtain the global solution:

$$u^k = \sum_{i=1}^{N} \mathcal{B}_i u_i^{k,m}, \tag{2.21}$$

   where $\mathcal{B}_i$ is the restriction operator that maps the solution on subdomain Ω_i to the global domain.

By solving the local problems independently, AS is able for parallel computations, which allows each subdomain to be solved at the same time [37], [48]. This parallelization capability makes AS particularly appropriate for high-performance computing architectures. Moreover, the convergence of AS is often faster than that of the classical Schwarz method. This improved convergence can is due to the additive feature of the algorithm. At each iteration, the local solutions are combined additively, which allows error reduction from multiple sources simultaneously.

AS can be applied to a wide range of PDE problems, including elliptic, parabolic, and hyperbolic equations. It is especially useful for problems with heterogeneous coefficients or complex geometries since the subdomain decomposition is able to process local features in an efficient manner.

2.1.6 Multiplicative Schwarz Method (MS)

The Multiplicative Schwarz Method (MS) is another domain decomposition technique used for solving PDEs [48]. Similar to the Additive Schwarz methods (AS), MS divides the computational domain into non-overlapping subdomains and solves the problem iteratively on each subdomain. However, unlike AS, MS considers the interaction between subdomains multiplicatively, which results in different convergence characteristics.

To utilize the Multiplicative Schwarz method, the domain Ω is decomposed into N subdomains, denoted as $\Omega_1, \Omega_2, \ldots, \Omega_N$, where each subdomain covers a distinct portion of the global domain. The subdomain decomposition can be based on geometric considerations or other relevant criteria. The objective of MS is to obtain the solution u by solving the PDE independently on each subdomain and updating the solution iteratively.

The MS algorithm proceeds as follows:

1. Initialize the solution on each subdomain:

$$u_i^{(0)} = u^{k-1} \quad \text{on } \partial\Omega_i \quad \text{for all } i, \tag{2.22}$$

   where u^{k-1} represents the solution from the previous iteration.

2. Iterate until convergence:

   (a) Solve the local problem on each subdomain Ω_i:

$$u_i^{(m)} = \mathcal{L}_i^{-1} \left(f_i - \sum_{j=1, j\neq i}^{N} \mathcal{A}_{ij} u_j^{(m-1)} \right), \tag{2.23}$$

   where $\mathcal{L}_i^{-1}$ is the inverse of the local PDE operator on subdomain Ω_i, f_i is the local source term on Ω_i, $\mathcal{L}_i$ represents the local PDE operator, $\mathcal{A}_{ij}$ is the interaction operator that accounts for the influence of the solution on subdomain Ω_j in the update of $u_i^{(m)}$, and $u_j^{(m-1)}$ is the solution on subdomain Ω_j at iteration $m-1$.

   (b) Update the global solution:

$$u^{(m)} = \prod_{i=1}^{N} \mathcal{B}_i u_i^{(m)}, \tag{2.24}$$

   where $\mathcal{B}_i$ is the restriction operator that maps the solution on subdomain Ω_i to the global domain.

The MS method offers several advantages in the solution of PDEs. By considering the interaction between subdomains in multiplicatively, MS considers the coupling between subdomains more explicitly than AS. This can potentially improve convergence, particularly for problems with highly heterogeneous coefficients or when the solution has strong variations across subdomains.

The MS method can be used for different types of PDE problems, such as elliptic, parabolic, and hyperbolic equations. It is particularly useful for problems with complicated shapes or materials that have varying properties in different regions. By dividing the problem into smaller parts, MS can accurately capture the local features of the problem. However, compared to the AS, MS is not as easily parallelizable. As a result, AS is more commonly used than MS.

Chapter 3

Learning Interface Conditions in Domain Decomposition Solvers

Portions of this chapter appear in the paper "Learning interface conditions in domain decomposition solvers" published in Advances in Neural Information Processing Systems (NeurIPS 2022) [49]

3.1 Introduction

Domain decomposition methods have proven to be highly effective in approximating solutions to partial differential equations. However, achieving the optimal construction of these methods often involves complex analysis and is typically limited to simplified, structured-grid scenarios, thereby restricting their applicability to more intricate problems. In this chapter, we extend the application of optimized Schwarz domain decomposition methods to unstructured-grid problems by leveraging Graph Convolutional Neural Networks (GCNNs) and employing unsupervised learning techniques to acquire optimal modifications at subdomain interfaces. A pivotal element of our approach is the utilization of an enhanced loss function, which facilitates effective training on relatively small problem instances while ensuring robust performance on significantly larger problems. Notably, our approach exhibits a computational cost that scales linearly with the problem size. To evaluate the performance of the learned linear solvers, we conduct comparisons against both classical and optimized domain decomposition algorithms, encompassing both structured- and unstructured-grid problems. The chapter is structured as follow: we fist provide some required background in section 3.2. We then proposed our method in section 3.3, and finally, we discuss the results in section 3.5.

3.2 Background

Let $\Omega \subset \mathbb{R}^2$ be an open set, and consider the positive-definite Helmholtz problem

$$Lu = (\eta - \Delta)u = f \quad \text{in } \Omega, \tag{3.1}$$

with inhomogeneous Dirichlet conditions imposed on the boundary $\partial\Omega$. In (3.1), the parameter $\eta > 0$ represents a *shift* in the Helmholtz problem. In the numerical results below, we consider both finite-difference discretizations of (3.1) on regular grids, as well as piecewise linear finite-element (FE) discretizations on

arbitrary triangulations. In both cases, we denote the set of degrees of freedom as D, and note that these are in a one-to-one correspondence with the nodes of the underlying mesh. Consider a decomposition of D into non-overlapping subdomains $D_i^0, i \in \{1, 2, \ldots, S\}$ such that each node is contained within exactly one subdomain D_i^0, yielding $\cup D_i^0 = D$. In this subdomain notation, the superscript denotes the amount of overlap in the subdomains, which is zero for the non-overlapping subdomains that we first consider. Let R_i^0 be the restriction operator onto the set of degrees of freedom (DoFs) in D_i^0, and let $\left(R_i^0\right)^T$ be the corresponding extension operator from D_i^0 into set D. Then, an FE discretization of the Helmholtz problem leads to a linear system of the form $Ax = b$, where A is the global stiffness matrix and $A_i^0 = R_i^0 A \left(R_i^0\right)^T$ is the subdomain stiffness matrix for D_i^0. We note that alternate definitions to the Galerkin projection for A_i^0 are possible, and are commonly considered in optimized Schwarz settings (as noted below).

In the case of restricted additive Schwarz (RAS) [6], the subdomains are extended to allow for overlap: nodes near the "boundary" of their subdomain are potentially included in two or more subdomains. We denote the amount of overlap by $\delta \in \mathbb{N}$, defining subdomains D_i^δ recursively, by $D_i^\delta = D_i^{\delta-1} \cup \left\{ j \mid a_{kj} \neq 0 \text{ and } k \in D_i^{\delta-1} \right\}$ for $\delta > 0$. The conventional RAS preconditioner is defined as

$$M_{\text{RAS}} = \sum_{i=1}^{S} (\tilde{R}_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta, \tag{3.2}$$

where R_i^δ is the standard restriction operator to subdomain Ω_i^δ, $\tilde{R}_i^\delta$ is a modified restriction operator from D to the DoFs in D_i^δ that takes nonzero values only for DoFs in D_i^0, and $A_i^\delta = \left(R_i^\delta\right)^T A R_i^\delta$. Figure 3.1 shows an example unstructured grid with two subdomains and overlap $\delta = 1$.

In an optimized Schwarz setting, we modify the subdomain systems, A_i^δ. Rather than using a Galerkin projection onto D_i^δ, we rediscretize (3.1) over the subdomain of Ω corresponding to these DoFs, imposing a Robin-type boundary condition on the boundary of the subdomain. We define this matrix to be $\tilde{A}_i^\delta = A_i + L_i$, where A_i is the term resulting from discretization of (3.1) with Neumann boundary conditions, and L_i is additional the term resulting from the Robin-type condition, as in:

$$\text{Dirichlet: } u = g_{\text{D}}(x), \qquad \text{Neumann: } \vec{n} \cdot \nabla u = g_{\text{N}}(x), \qquad \text{Robin: } \alpha u + \vec{n} \cdot \nabla u = g_{\text{R}}(x), \tag{3.3}$$

where $\vec{n}$ is the outward normal to the edge on the boundary and g denotes inhomogeneous data.

The matrix L_i has the same dimensions as A_i, so that $\tilde{A}_i^\delta$ is well-defined. However, it has a significantly different sparsity pattern, with nonzero entries only in rows/columns corresponding to nodes on the boundary of subdomain D_i^δ. In practice, we identify a cycle or path in the graph corresponding to A_i with the property that every node in the cycle is on the boundary of D_i^δ but not the boundary of D (the discretized domain), and then restrict the nonzeros in L_i to the entries corresponding to the edges in this cycle/path (including self-edges, corresponding to entries on the diagonal of L_i).

Using this notation, the ORAS preconditioner can be written as

$$M_{\text{ORAS}} = \sum_{i=1}^{S} \left(\tilde{R}_i^\delta\right)^T \left(\tilde{A}_i^\delta\right)^{-1} R_i^\delta. \tag{3.4}$$

Our aim is to learn the values in the matrices L_i, aiming to outperform the classical choices of these values predicted by Fourier analysis, in the cases where those values are known, and to learn suitable values for cases, such as finite-element discretizations on unstructured grids, where no known optimized Schwarz parameters

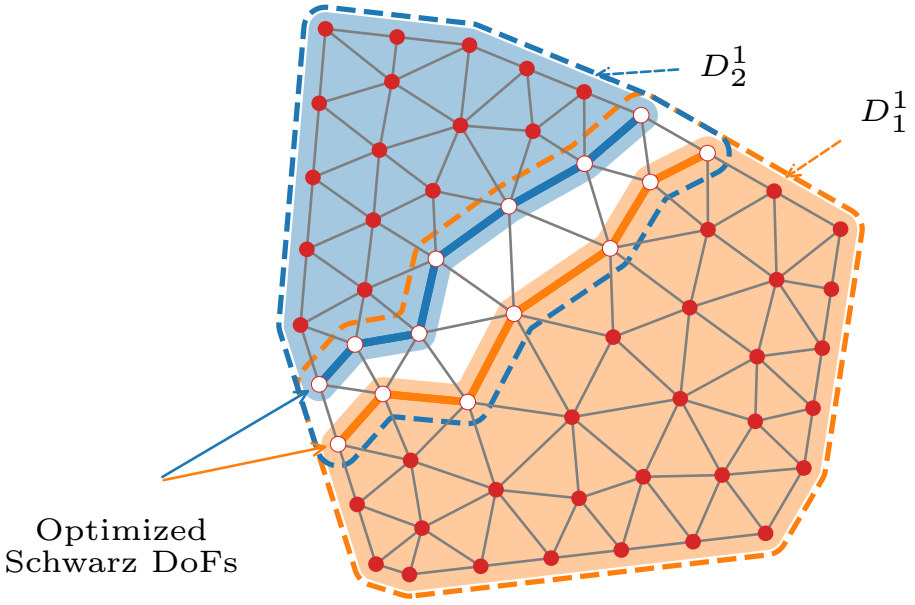

Figure 3.1: Two subdomains with overlap $\delta = 1$ on a 58-node unstructured grid. The blue and orange shading denotes the original non-overlapping partitions (D_1^0 and D_2^0), while the blue and orange dashed outlines show the overlapping subdomains (D_1^1 and D_2^1). Nodes belonging to only one subdomain are marked with a solid circle, while those in white belong to both subdomains. The connections in the boundary matrices L_1 and L_2 are denoted by the edges shaded in blue (for L_1) and orange (for L_2).

exist. We optimize the values for the case of stationary (Richardson) iteration, but evaluate the performance of the resulting methods both as stationary iterations and as preconditioners for FGMRES.

3.2.1 Graph Neural Networks (GNNs):

Applying learning techniques to graph structured data necessitates stepping beyond multilayer perceptron (MLP) and conventional convolutional neural networks (CNN) to a type of network that leverages the underlying graph nature of the problem, namely graph convolutional neural networks (GCNNs). GCNNs are typically divided into two categories: spectral and spatial [50]. Spectral GCNNs, first introduced by Bruna et al. [51], consider a graph Laplacian eigenbasis and define convolution as diagonal operators. As such, spectral GCNN methods suffer from time complexity problems due to the necessity for the eigendecomposition of the Laplacian matrix. Nevertheless, in follow-up work [52], [53], remedies have been proposed to mitigate this. Unlike spectral methods, spatial GCNNs consider local propagation of information in graph domains as a convolution graph. One popular framework is the message passing neural network (MPNN) [54], which is based on sharing information among neighbor nodes in each round of a convolution pass. This has been generalized [55] by introducing a configurable graph network block structure consisting of node and edge convolution modules and a global attribute. In an effort to alleviate computational complexity of GCNNs [56], topology adaptive graph convolution networks (TAGCN) can be constructed by defining learnable filters. This is not only computationally simpler, but also allows for adapting to the topology of the graphs when scanning them for convolution.

3.2.2 Lloyd aggregation

Lloyd's algorithm [57] is a standard approach for partitioning data, closely related to k-means clustering, that can be used (for example) to find close approximations to centroidal Voroni tesselations. Here, we use a modified form of Lloyd's algorithm, known as Lloyd aggregation [58], to partition a given set of degrees of freedom, V, into the non-overlapping subdomains, V_i^0, for $i \in \{1, 2, \ldots, S\}$, needed as input to our algorithm. (Note that, here, we change notation from the main document and use V to denote the vertices in the graph associated with the matrix, rather than D to denote the index set of degrees of freedom.) Consider a 2D planar graph, G, the set of all edges E, the set of all of its nodes V, and $V_c \subseteq V$. The nodes in V_c serve as the centers of "clusters" in the graph that will define the subdomains, V_i^0. These regions are obtained based on the closest center to each graph node, where the distance is measured by the number of edges covered in the shortest path between two nodes (denote distance in the graph between node i and j by g_{ij}). Define the centroid of a region as the farthest node from the boundary, breaking possible ties by random choice. We use a modified version of the Bellman-Ford algorithm, commonly used for obtaining the nearest center to every node in V and its associated distance [59]. Let $\vec{n}$ be a list of graph nodes whose i-th element is the nearest center to the i-th node of the graph, and let g_j be the graph distance from node j to n_j; then, the modified Bellman-Ford algorithm is shown in Algorithm 1.

While this is an iterative computation, it has finite termination when the values in $\vec{g}$ and $\vec{n}$ stop changing. After running this modified Bellman-Ford algorithm, Lloyd's algorithm modifies the clusters by selecting the centroid of every subdomain as its new center, then iterates, using the modified Bellman-Ford algorithm to calculate new distances and nearest centers. Given updated center positions, it forms the new subdomains. The full Lloyd algorithm is shown in Algorithm 2, where we define the set of border nodes, B, as the set of all nodes that are connected by an edge to a node that has a different nearest center node. The key point here is

Algorithm 1 Modified Bellman-Ford

Input E: The set all edges, V: The set of all nodes, V_c: The set of initial center nodes.
$g_i = \infty \ \ \forall_{i \in \{1,2,...,|V|\}}$
$n_i = -1 \ \forall_{i \in \{1,2,...,|V|\}}$
for $c \in V_c$ **do**
    $g_c \leftarrow 0$
    $n_c \leftarrow c$
end for
while True **do**
    Finished $\leftarrow$ True
    for $(i, j) \in E$ **do**
        if $g_i + g_{ij} < g_j$ **then**
            $g_j \leftarrow g_i + g_{ij}$
            $n_j \leftarrow n_i$
            Finished $\leftarrow$ False
        end if
    end for
    if Finished **then**
        return $\vec{g}, \vec{n}$
    end if
end while

that we use Modified Bellman-Ford to assign closest centers, then compute the set of border nodes, then find the new centers as those that are further from the border set within each of the original subdomains (using graph distances from B, but original center assignment in $\vec{n}$).

Algorithm 2 Lloyd Aggregation

Input K: Number of iterations, E: The set of all edges, V: The set of all nodes, V_c: The set of initial center nodes.
for $i = 1, 2, 3, ..., K$ **do**
    $\vec{g}, \vec{n} \leftarrow$ Modified Bellman-Ford(E, V, V_c)
    $B \leftarrow \emptyset$
    for $(i, j) \in E$ **do**
        if $n_i \neq n_j$ **then**
            $B \leftarrow B \cup \{i, j\}$
        end if
    end for
    $\vec{g}, \vec{x} \leftarrow$ Modified Bellman-Ford(E, V, B)
    $V_c \leftarrow \{i \in V : g_i > g_j \ \ \forall n_i = n_j\}$
end for
return $\vec{n}$

Time Complexity: Assuming each node's initial distance to a center node is bounded independently of $|V|$, and also assuming that each node's degree is bounded independently of $|V|$, Algorithm 1 runs a $|V|$-independent number of iterations to determine one nearest center node for every point. Thus, Algorithm 1 is $O(|V|)$ in our case. This is run a $|V|$-independent number of times in Algorithm 2, to generates the subdomains, resulting in an overall algorithmic cost of $O(|V|)$.

3.3 Method

3.3.1 Optimization problem and the loss function

Throughout this study, we use $\|\cdot\|$ to denote the ℓ^2 norm of a matrix or vector, $\|A\|_F$ for the Frobenius norm of A, and $\rho(A)$ as the spectral radius of A. The optimization problem that we seek to solve is to find optimal values for the entries in the matrices L_i, constrained by given sparsity patterns, to minimize $\rho(T)$, where $T = I - M_{\text{ORAS}}A$ is the error-propagation operator for the stationary iteration corresponding to M_{ORAS} defined in (4.5). The spectral radius $\rho(T)$ corresponds to the *asymptotic convergence factor* of the stationary iteration, giving a bound on the asymptotic convergence of the method. Formally defined in terms of the extremal eigenvalue of T^TT (since T is not symmetric), direct minimization of $\rho(T)$ is difficult since backpropagation of an eigendecomposition is numerically unstable [60]. To overcome this, Luz et al. [12] propose to relax the minimization of $\rho(T)$ (for a similar AMG preconditioner) to minimizing the Frobenius norm of T, $\|T\|_F$. In our experiments, however, we find that this is insufficient, leading to preconditioners that do not scale. One reason is that while the Frobenius norm is an upper bound on the spectral radius, it does not appear to be a suitably "tight" bound for use in this context (see Section 3.5.2 and Figure 3.16). Instead, we use a relaxation inspired by Gelfand's formula, that $\rho(T) = \lim_{K\to\infty} \|T^K\|^{\frac{1}{K}}$, and the common bound that

$$\rho(T) \leq \|T^K\|^{\frac{1}{K}} = \sup_{x\neq 0} \left(\frac{\|T^K x\|}{\|x\|} \right)^{\frac{1}{K}} = \sup_{x:\|x\|=1} (\|T^K x\|)^{\frac{1}{K}} \tag{3.5}$$

for some finite $K \in \mathbb{N}$. This results in the optimization problem

$$\min_{\substack{L_i, i=1,2,\ldots,S \\ \text{sparsity of } L_i}} \sup_{x:\|x\|=1} \|T^K x\|. \tag{3.6}$$

3.3.2 Numerical evaluation of the loss function

We denote the action of evaluating the GNN by $f^{(\theta)}$ (where θ represents the network parameters), and consider a discretized problem with DoF set D, of size n. The set D can be decomposed into subdomains either by using fixed geometric choices of the subdomain (e.g., for finite-difference discretizations), using the METIS graph partitioner [61], or a k-means-based clustering algorithm (best known as Lloyd's algorithm which has $O(n)$ time complexity) [57], [58]. For unstructured problems, we use a k-means-based algorithm, decomposing D to subdomains D_i for $i = 1, 2, \ldots, S$, with overlap δ. The GNN then takes D and its decomposition as inputs, as well as sparsity constraints on the matrices L_i for $i = 1, 2, \ldots, S$, and outputs values for these matrices:

$$L_1^{(\theta)}, L_2^{(\theta)}, \ldots, L_S^{(\theta)} \leftarrow f^{(\theta)}(D). \tag{3.7}$$

Using the learned subdomain interface matrices, we then obtain the modified MLORAS (Machine Learning Optimized Restricted Additive Schwarz) operator, $M_{\text{ORAS}}^{(\theta)}$, simply using $\tilde{A}_i^{\delta} = A_i + L_i^{(\theta)}$ in (4.5). We denote the associated error propagation operator by $T^{(\theta)} = I - M_{\text{ORAS}}^{(\theta)}A$.

While Gelfand's formula and the associated upper bound in (3.5) are valid in any norm, it is natural to consider them with respect to the ℓ^2 norm in this setting. However, this raises the same issue as encountered in [12], that it generally requires an eigendecomposition to compute the norm. To avoid this, we use a

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

$$X = [x_1, x_2, \ldots, x_m], \forall_j\, x_j \sim \mathbb{R}^n \text{ uniformly}, \|x_j\| = 1. \tag{3.8}$$

Here, we randomly select m points uniformly on a unit sphere in $\mathbb{R}^n$, which can be done using the method in [62]. We then define

$$Y^{(\theta)} = \left\{\left\|\left(T^{(\theta)}\right)^K x_1\right\|, \left\|\left(T^{(\theta)}\right)^K x_2\right\|, \ldots, \left\|\left(T^{(\theta)}\right)^K x_m\right\|\right\}, \tag{3.9}$$

taking each column of X as the initial guess to the solution of the homogeneous problem $Ax = 0$ and taking K steps of the stationary algorithm to generate $\left(T^{(\theta)}\right)^K x_j$. Since we normalize each column of X to have $\|x_j\| = 1$, the value of $\left\|\left(T^{(\theta)}\right)^K x_j\right\|$ serves as a lower bound for $\left\|\left(T^{(\theta)}\right)^K\right\|$. Thus, taking the maximum of the values in Y provides a practical loss function that we use below, defining

$$\mathcal{L}^{(\theta)} = \max(Y^{(\theta)}). \tag{3.10}$$

We note similarities between this loss function and that used in [63], but that we are able to use the maximum of $Y^{(\theta)}$ (giving a better approximation to the norm) in our context instead of averaging.

Ultimately, the cost of our algorithm depends strongly on the chosen values of m and K. For sufficiently large values of m, we now show that the maximum value in $Y^{(\theta)}$ is an arbitrarily good approximation to $\left\|\left(T^{(\theta)}\right)^K\right\|^{\frac{1}{K}}$ (in the statistical sense). The following lemma is useful in the proof of Theorem 5. While its proof is quite simple, we are not aware of the result in the literature, and include it here for completeness.

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

Proof. Since $y \le x$, the binomial theorem gives that

$$x \le (x-y) + y + \sum_{i=1}^{K-1} \binom{K}{i} y^{\frac{i}{K}} (x-y)^{\frac{K-i}{K}} = (y^{\frac{1}{K}} + (x-y)^{\frac{1}{K}})^K. \tag{3.11}$$

Taking the K^{th} root of both sides and rearranging gives the stated result. □

Theorem 2. *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 $\{x \in \mathbb{R}^n, \|x\| = 1\}$, then*

$$Y = \left\{\left\|T^K x_1\right\|, \left\|T^K x_2\right\|, \ldots, \left\|T^K x_m\right\|\right\} \tag{3.12}$$

satisfies

$$P\left(\left|\rho(T) - \max(Y)^{\frac{1}{K}}\right| < \epsilon\right) > 1 - \delta. \tag{3.13}$$

Proof. According to Gelfand's theorem, there exists $L \in \mathbb{N}$ such that

$$\forall \ell > L, \quad \left|\rho(T) - \sup_{x:\|x\|=1} \|T^\ell x\|^{\frac{1}{\ell}}\right| < \frac{\epsilon}{2}. \tag{3.14}$$

Take any $K \ge L$ and let $\tilde{\epsilon} = \frac{\epsilon}{2\|T^K\|^{\frac{1}{K}}}$. Since $\mathbb{R}^n$ is finite-dimensional, there exists an $x^* \in \mathbb{R}^n$, $\|x^*\| = 1$ such that $\sup_{x:\|x\|=1} \|T^K x\| = \|T^K x^*\|$. Denote the volume of the surface of the n-dimensional sphere of unit

radius around the origin in $\mathbb{R}^n$ by C_{tot}, and the volume of the region on this sphere within radius $\tilde{\epsilon}^K$ of x^* by $C_{\tilde{\epsilon},K}$.

Given $\delta < 1$, let $M \in \mathbb{N}$ satisfy $M \geq \frac{\log(\delta)}{\log\left(1-\frac{C_{\tilde{\epsilon},K}}{C_{\text{tot}}}\right)}$. Then,

$$P\left(\|x^* - x_i\| > \tilde{\epsilon}^K \text{ for all } i\right) < \left(1 - \frac{C_{\tilde{\epsilon},K}}{C_{\text{tot}}}\right)^M \leq \delta. \tag{3.15}$$

Thus, with probability of at least $1-\delta$, we expect at least one point from a selection of $m > M$ points uniformly distributed on the sphere of radius unit radius to be in $C_{\tilde{\epsilon},K}$. Let that point be x_r, giving

$$\left\|T^K x^*\right\|^{\frac{1}{K}} - \left\|T^K x_r\right\|^{\frac{1}{K}} \leq \left(\left\|T^K x^*\right\| - \left\|T^K x_r\right\|\right)^{\frac{1}{K}} \tag{3.16}$$

$$\leq \left(\left\|T^K (x^* - x_r)\right\|\right)^{\frac{1}{K}} \tag{3.17}$$

$$\leq \left\|T^K\right\|^{\frac{1}{K}} \|x^* - x_r\|^{\frac{1}{K}} \leq \|T^K\|^{\frac{1}{K}} \tilde{\epsilon} = \frac{\epsilon}{2} \tag{3.18}$$

using Lemma 1 and the reverse triangle inequality. Finally, by the triangle inequality, with probability of at least $1-\delta$, we have:

$$\left|\rho(T) - \max(Y)^{\frac{1}{K}}\right| \leq \left|\rho(T) - \sup_{x:\|x\|=1} \|T^K x\|^{\frac{1}{K}}\right| + \left|\sup_{x:\|x\|=1} \|T^K x\|^{\frac{1}{K}} - \max(Y)^{\frac{1}{K}}\right| \leq \epsilon. \tag{3.19}$$

□

Remark. *According to [64], since the optimal interface values are Turing-computable, if the depth of the GNN is at least the diameter of the graph, and a TAGConv layer followed by a feature encoder is Turing-complete, the optimal interface values can be learned. In our setting, for a problem on a structured grid of size $N \times N$ with two identical rectangular subdomains, this implies that the GNN will be able to learn the optimal interface values given, if and only the GNN has depth at least $2N$, has deep enough feature encoders, and the width of the layers is unbounded.*

Remark. *Theorem 5 guarantees convergence of the loss function to the spectral radius, in the limits of many samples and many stationary iterations. To the best of our knowledge, such a guarantee is not known for the previous loss functions used in the area [11], [12]. Moreover, there are substantial improvements in the numerical results using the new loss function in comparison to that of [12], as shown in Figure 3.16.*

Theorem 3. *Assuming bounded subdomain size, the time complexity to evaluate the optimal Schwarz parameters using our method is $O(n)$, where n is the number of nodes in the grid.*

Proof. Given bounded Lloyd subdomain size and fixed number of Lloyd aggregation cycles, subdomain generation has $O(n)$ time complexity [58] (see section 3.4). To evaluate each TAGConv layer, one computes $y = \sum_{\ell=1}^{L} G_\ell x_\ell + b\mathbf{1}_n$, where L is the number of node features, $G_\ell \in \mathbb{R}^{n\times n}$ is the graph filter, b is a learnable bias, and $x_\ell \in \mathbb{R}^n$ are the node features. Moreover, the graph filter is a polynomial in the adjacency matrix M of the graph: $G_\ell = \sum_{j=0}^{J} g_{\ell,j} M^j$ where J is a constant and $g_{\ell,j}$ are the coefficients of filter polynomial. Since the graph has bounded node degrees, it implies that M is sparse and the action of M^j has $O(n)$ cost, and therefore, the full TAGConv evaluation also has $O(n)$ cost. Moreover, the cost of edge feature and the feature networks are $O(n)$, resulting in overall $O(n)$ cost. □

3.4 Neural network details

3.4.1 Inputs and Output

Inputs: The GNN takes the grid G as its input which has three main components, namely D_{node} (node feature matrix), D_{edge} (edge feature matrix), and A (the graph adjacency matrix). Every node has a binary feature, and its value is one if it is on a boundary of a subdomain and zero otherwise. Therefore, for every node, the corresponding element in D_{node} indicates whether that point is on a boundary or not. In other words, the binary node feature determines the grid decomposition. D_{edge} consists of the edge values obtained from discretization of the underlying PDE.

Outputs: After passing the input to the GNN architecture which consists of node and edge convolution blocks and is fully described in the following subsection, the learned edge weights are obtained for every edge in the grid. However, only the edges between the nodes on the subdomain boundaries (considering self-loops) are of our interest so we mask the rest of the edges. Therefore, the output of the GNN is the learned values for edges connecting nodes on the boundary of a subdomain which are referred to as interface values in the main document. For the i-th subdomain, the interface value matrix is referred to as L_i^θ, where θ represents the GNN learnable parameters (see Equation 4.12). Figure 3.8 shows an example of the sparsity pattern of an L matrix for each of the identical subdomains of the 10×10 structured grid in Figure 3.9. Also note that the interface values are the nonzero elements of the corresponding L matrix.

3.4.2 Architecture

The overall architecture of the GNN is shown in Figure 3.2. The GNN takes a graph as its input and sends node and edge features to the node convolution and edge feature preprocessing blocks, respectively, both of which are shown in Figure 3.3. Each node convolution block consists of a TAGConv layer with 2-size filters and 128 hidden units, followed by a ReLU activation, an instance norm layer, and a feature network block. The feature network block, as shown in Figure 3.4, consists of 8 blocks with residual connections between each; furthermore, each of the blocks consists of a layer norm followed by a fully connected layer of size 128, followed by a ReLU activation, and finally another fully connected layer of size 128. The edge feature preprocessing block takes the edge features and applies fully connected layers, ReLU nonlinearities, and instance normalization, before the graph convolution pass.

After passing through all of the node convolution blocks, the edge and node features are sent to a stack block. This block simply stacks node features onto the edges adjacent to each node. For every edge, (u, v), where u and v are the nodes on that edge, denote the node and edge features by $E_u, E_v, E_{(u,v)}$, which are the inputs to the stack block. The block then stacks these features and outputs them as the new edge features for the edge (u, v). Following the stack layer is the edge convolution block, depicted in Figure 3.5, which takes the stacked edge and node features and passes them through a series of fully connected layers, ReLU activation functions, and layer norms. It is noteworthy that the size of the input to the edge convolution block is 257, since following the description of the stack block, the two node features, each of size 128, and the edge feature of size 1 are stacked together. The output from the edge convolution block is, finally, passed through a masking block that outputs the interface values. This masks the interior edge values, i.e., takes the output of the GNN (one value per each edge in the graph) and multiplies those values that are not on the boundary of a subdomain by zero, to restrict the output from the GNN to the desired edges in the graph.

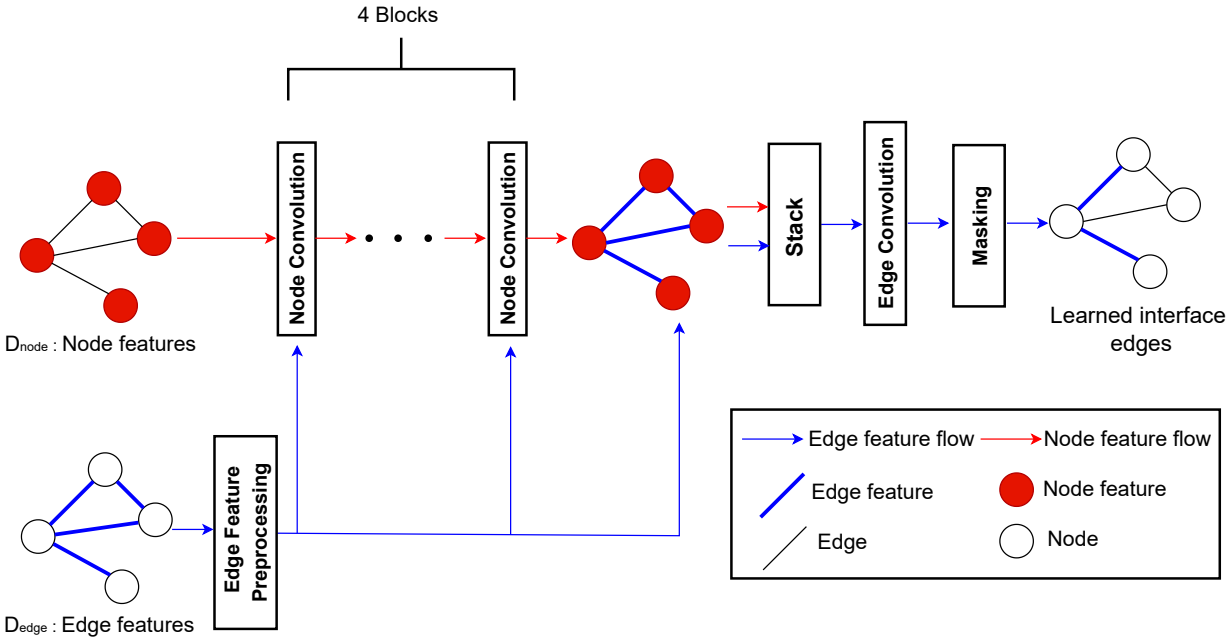

Figure 3.2: Overall GNN architecture.

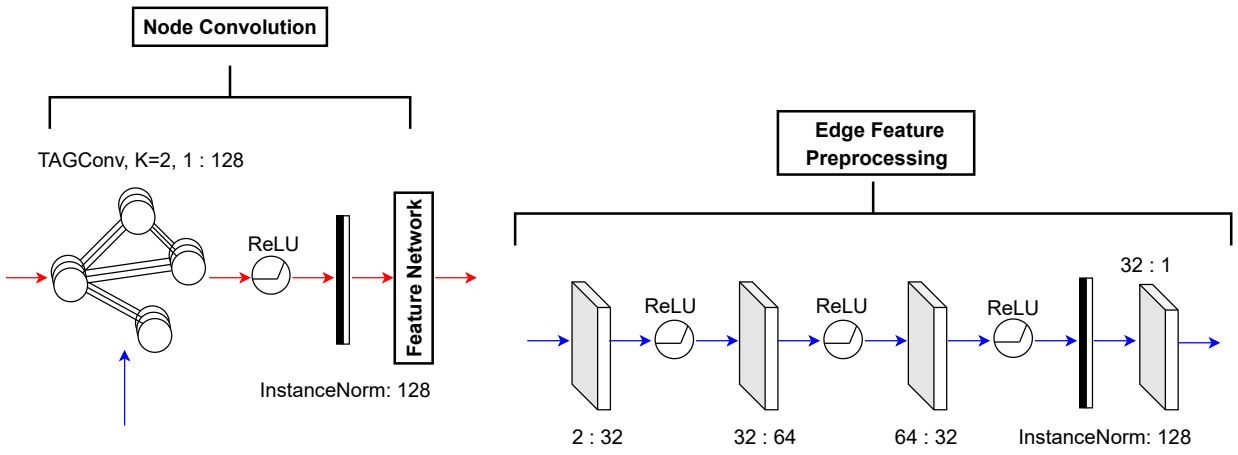

Figure 3.3: Left: Node convolution block. Right: Edge feature preprocessing block.

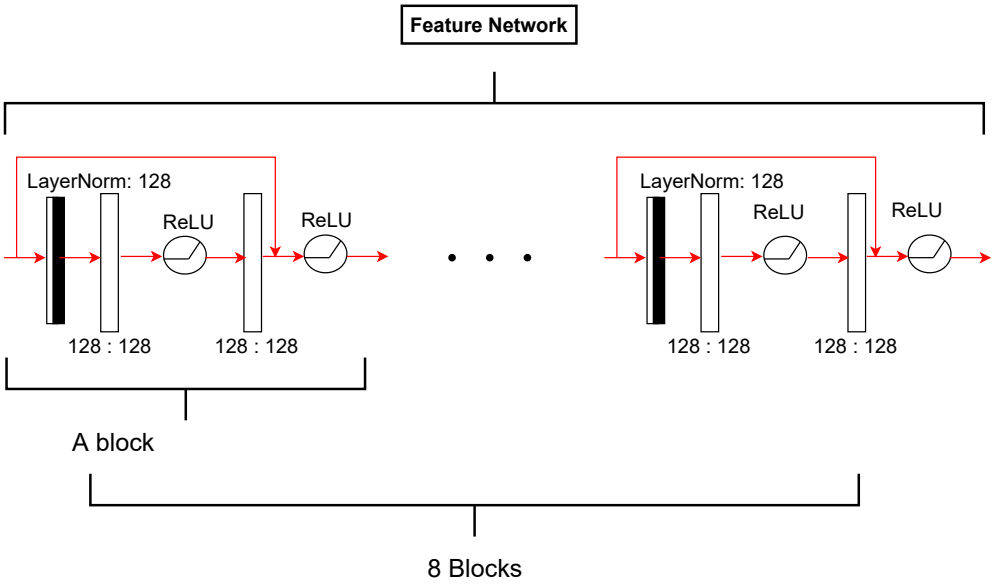

Figure 3.4: Feature network blocks.

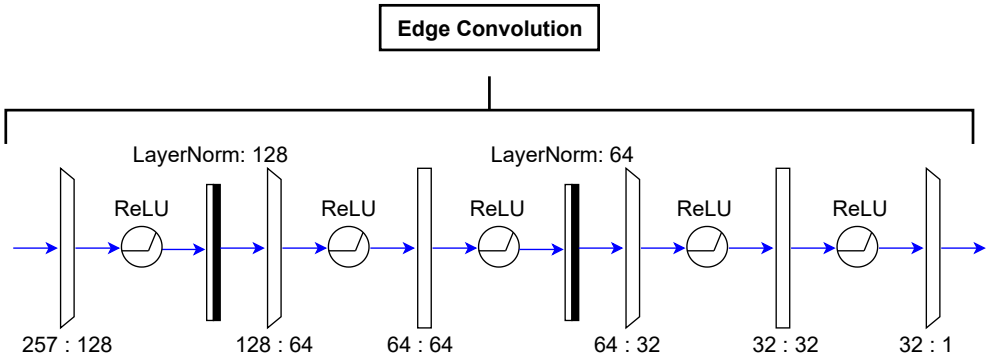

Figure 3.5: Edge convolution block.

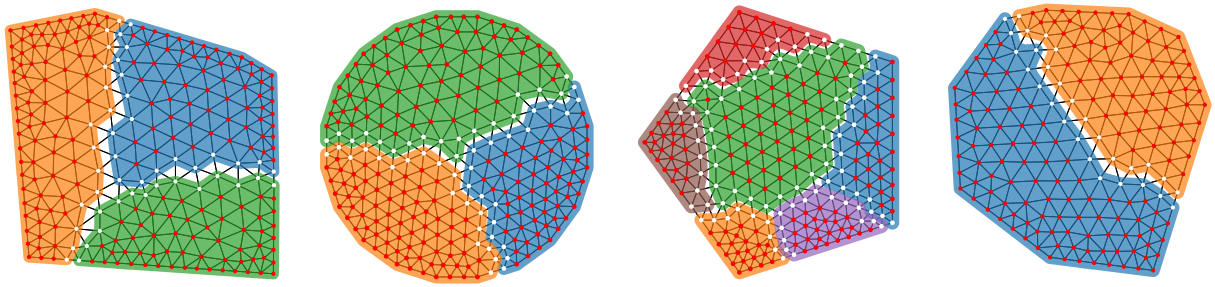

Figure 3.6: Example grids from the training set.

3.4.3 Training Details

We use a graph neural network based on four TAGConv layers and ResNet node feature encoders consisting of eight blocks after each TAGConv layer; see section 3.4 for more details on the structure of the GNN, and Section 3.5.4 for an ablation study. The training set in our study consists of 1000 unstructured grids with piecewise linear finite elements and with grids ranging from 90–850 nodes (and an average of 310 nodes). The grids are generated by choosing either a regular grid (randomly selected 60% of the time) or a randomly generated convex polygon; pygmsh [65] is used to generate the mesh on the polygon interior. A sample of the training grids is shown in Figure 3.6. We train the GNN for four epochs with a mini batch size of 25 using the ADAM optimizer [66] with a fixed learning rate of 10^{-4}. For the numerical evaluation of the loss function (3.10) we use $K = 4$ iterations and $m = 500$ samples. The code[1] was implemented using PyTorch Geometric [67], PyAMG [68], and NetworkX [69]. All training was performed on an 8-core i9 Macbook Pro using CPU only.

Moreover, we train two special networks for use in Section 3.5.1. The first is a "Brute Force" network, which is a single-layer neural network without an activation function, trained only on a structured 10×10 grid with two identical subdomains using the ADAM optimizer. The purpose of training this is to obtain the optimal interface values as a benchmark for comparison, including against our method. The second is the same network used for our method, MLORAS, but overtrained on only the problem in Section 3.5.1, to understand the learning capabilities of the method and choice of GNN architecture.

3.5 Results

3.5.1 Structured grids

We first consider rectangular structured grids with two subdomains. Although restrictive, these problems allow us to directly compare to existing OSM parameters, which are only available for structured grids and exactly two subdomains. We follow [7] and consider the Helmholtz problem (3.1) on the unit square with $\eta = 1$ and homogeneous Dirichlet boundary conditions. We discretize the problem on an $N \times N$ rectangular grid with $h = 1/(N + 1)$ grid spacing, using the standard five-point finite difference stencil.

Figure 3.7 demonstrates a working example of how to build the RAS preconditioner (and similarly MLORAS, by replacing the interface values with the learned values from the network). In this example, a

[1] All code and data for this study is at `https://github.com/compdyn/learning-oras` (MIT licensed).

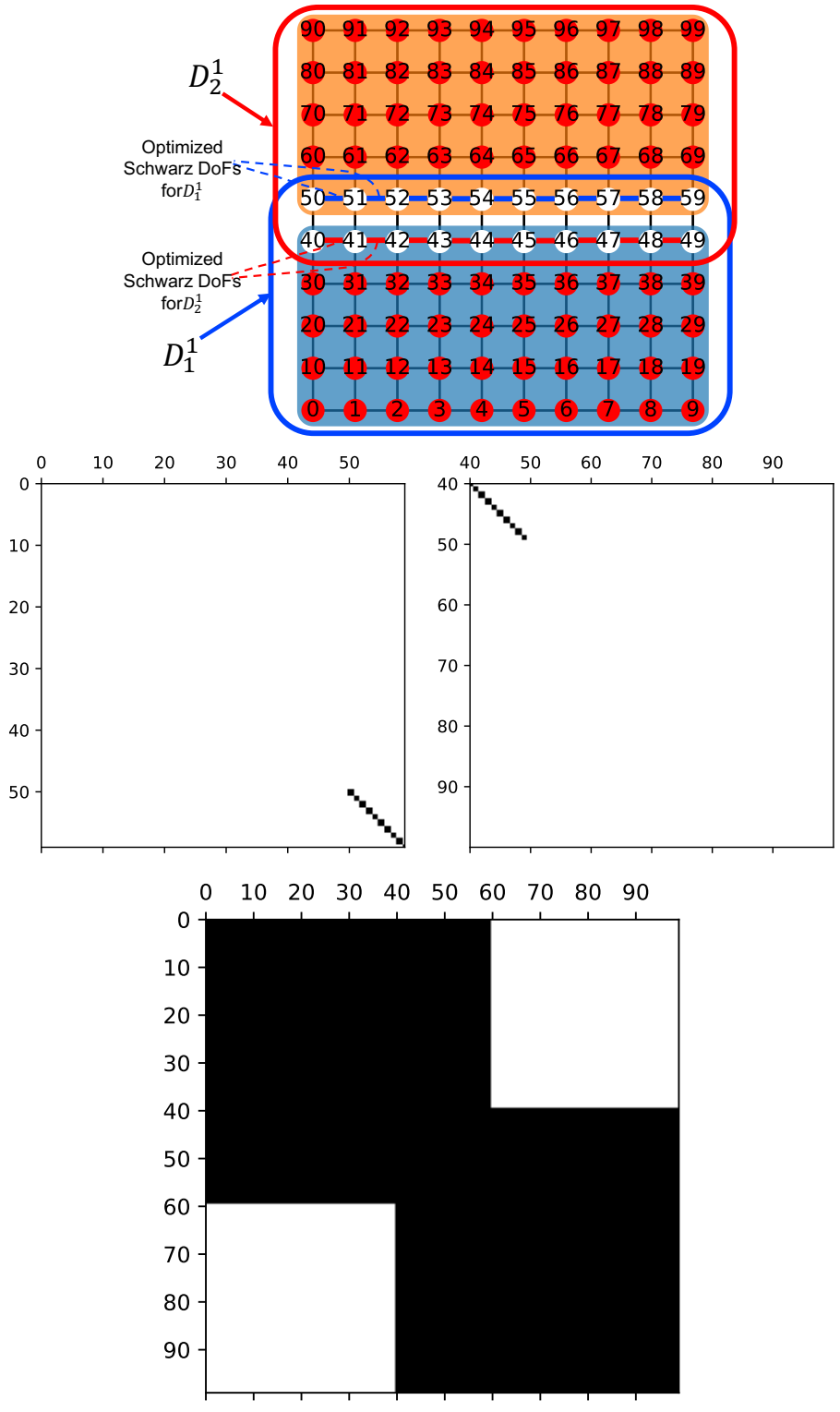

Figure 3.7: Top: A 10×10 structured grid with two identical subdomains. Middle: Sparsity pattern of the interface matrices. Bottom: The sparsity pattern of the RAS, M_{RAS} (and similarly for MLORAS) preconditioner. The interface sparsity plots axis is based on the node indexing, and the axis limit difference between sparsity plots of the interface matrices and M_{RAS} is due to subdomains and the global domain size.

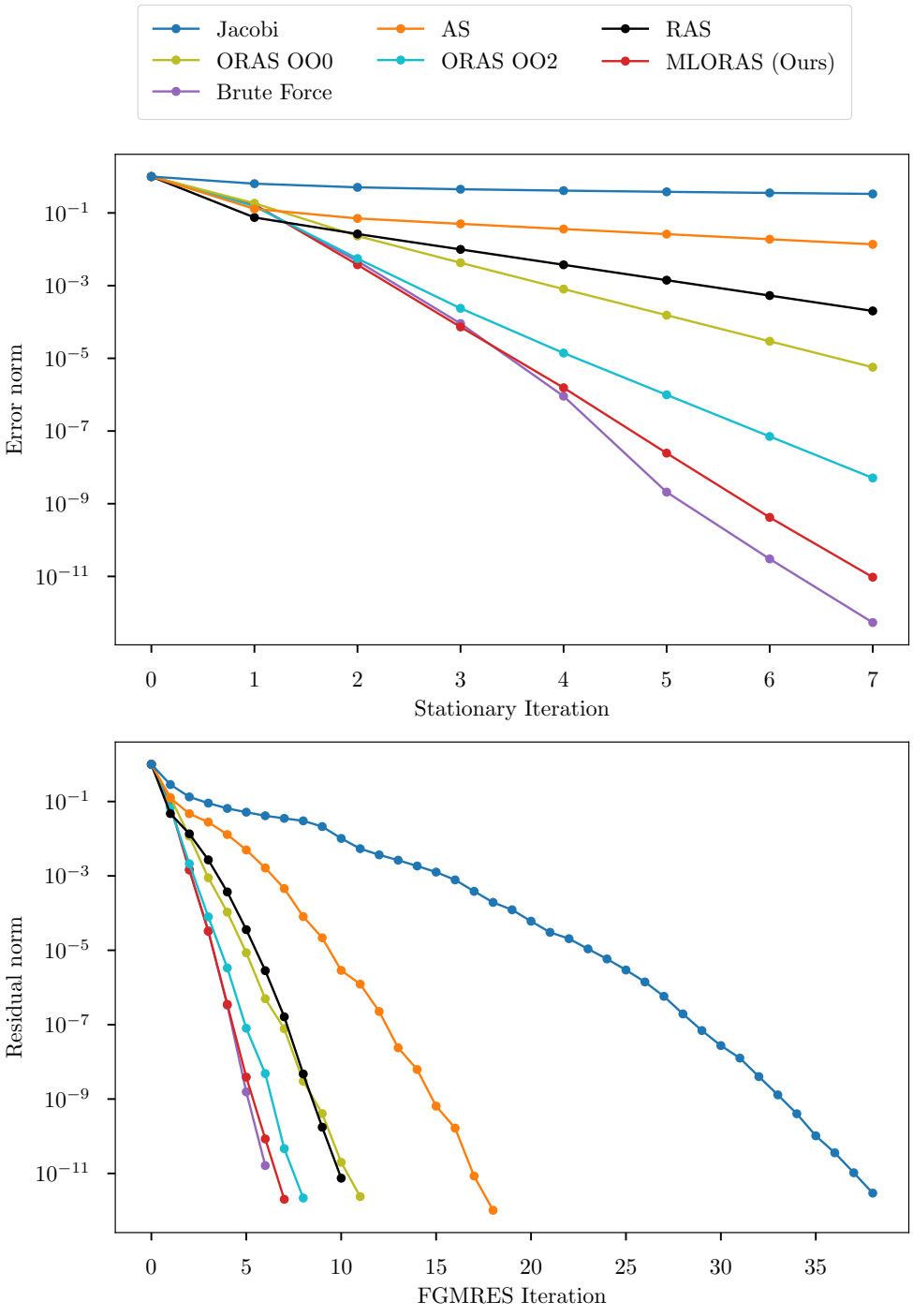

Figure 3.8: Results for the Helmholtz problem on a 10×10 structured grid with two identical subdomains, with convergence plots for the methods used as stationary algorithms (top) and as preconditioners for FGMRES (bottom).

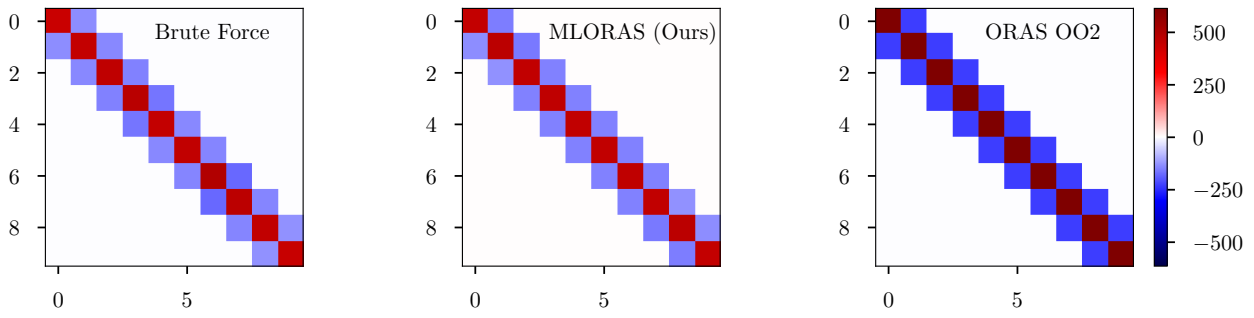

Figure 3.9: Interface values for the 10×10 structured grid with two identical subdomains. From left to right: brute force optimization of the interface values, MLORAS, and second-order ORAS (OO2) [7].

10×10 structured grid is decomposed into two similar subdomains with overlap of 1. The sparsity pattern of each of the interface matrix is shown in the middle and that of the RAS (and similarly MLORAS) preconditioner, which is obtained from 3.2, is demonstrated at the bottom.

Figure 3.8 shows the results of different methods on a 10×10 structured grid with two identical subdomains. We see that the overtrained MLORAS network learns interface parameters that outperform the Restrictive Additive Schwarz (RAS) [6] method; more significantly, MLORAS also outperforms the existing Optimized-RAS (ORAS) methods that were analytically derived for this specific problem (zeroth-order OO0 and second-order OO2 from [7]). Moreover, in these results, the MLORAS network almost reaches the performance of the "Brute Force" network (obtained by directly optimizing all interface values for this single structured grid), indicating that using a GNN to encode the optimal interface values does not significantly restrict the search space.

To understand the performance of the methods in more detail, Figure 3.9 plots the interface values output by the brute force optimization, the MLORAS network, and the OO2 ORAS algorithm. This shows that the MLORAS network is choosing interface values very close to those directly optimized by the brute force method, unlike those selected by the OO2 method. Another illustration of the learned values for the aforementioned three models at each entry is shown in Figure 3.10.

We also study the effect of scaling on structured grids with identical subdomains with larger sizes. We train the model on structured grid of size 10 and evaluate it on structured grids of size $20, 60, 40$, and 80. The evaluation is done for the convergence of stationary algorithm and FGMRES preconditioner, and is compared to classical RAS, Jacobi, and AS. The results are shown in Figure 3.11.

We have compared the solution plot of our method with RAS for the Holmholtz problem. We consider a 100×100 structured grid on the $(0, 1) \times (0, 1)$ domain, and consider the following true solution for the problem: $u^* = \sin(8\pi x) + \sin(8\pi y)$. We then start with an initial random guess with L_2 norm of 1. We apply 10 iterations of RAS and MLORAS (ours) as stationary algorithm and obtain solution for each method. Moreover, we run 10 iterations of FGMRES with MLORAS and RAS preconditioners on the initial guess and obtain predictions for both methods. The results are shown in Figure 4.12 for MLORAS and Figure 4.13 for RAS.

3.5.2 Unstructured grids

To evaluate the performance of the MLORAS network on unstructured grids, we consider 16 unstructured triangular grids in 2D with sizes ranging from about 90 to 40 000 nodes. These grids are defined on convex

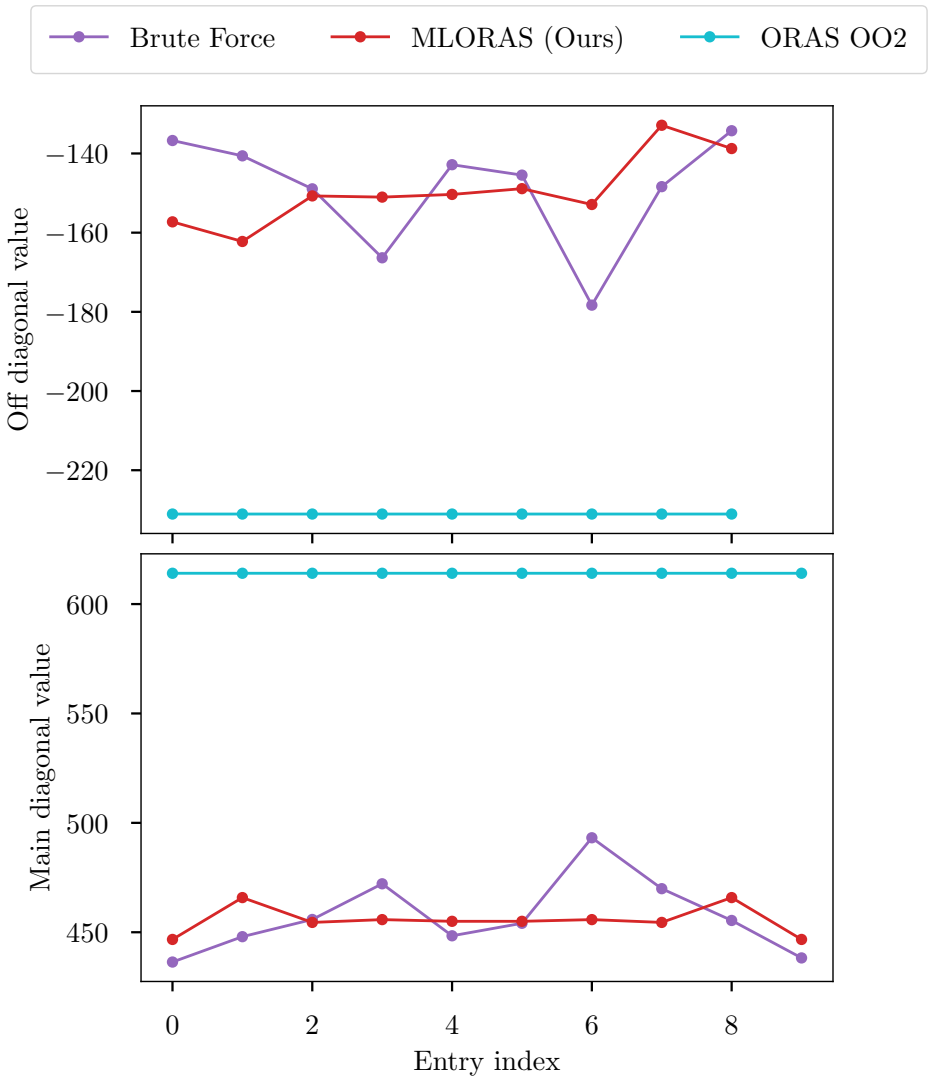

Figure 3.10: Interface values for the 10×10 structured grid with two identical subdomains. Comparison of the learned interface values at each entry for brute force method, MLORAS, and second-order ORAS (OO2) [7].

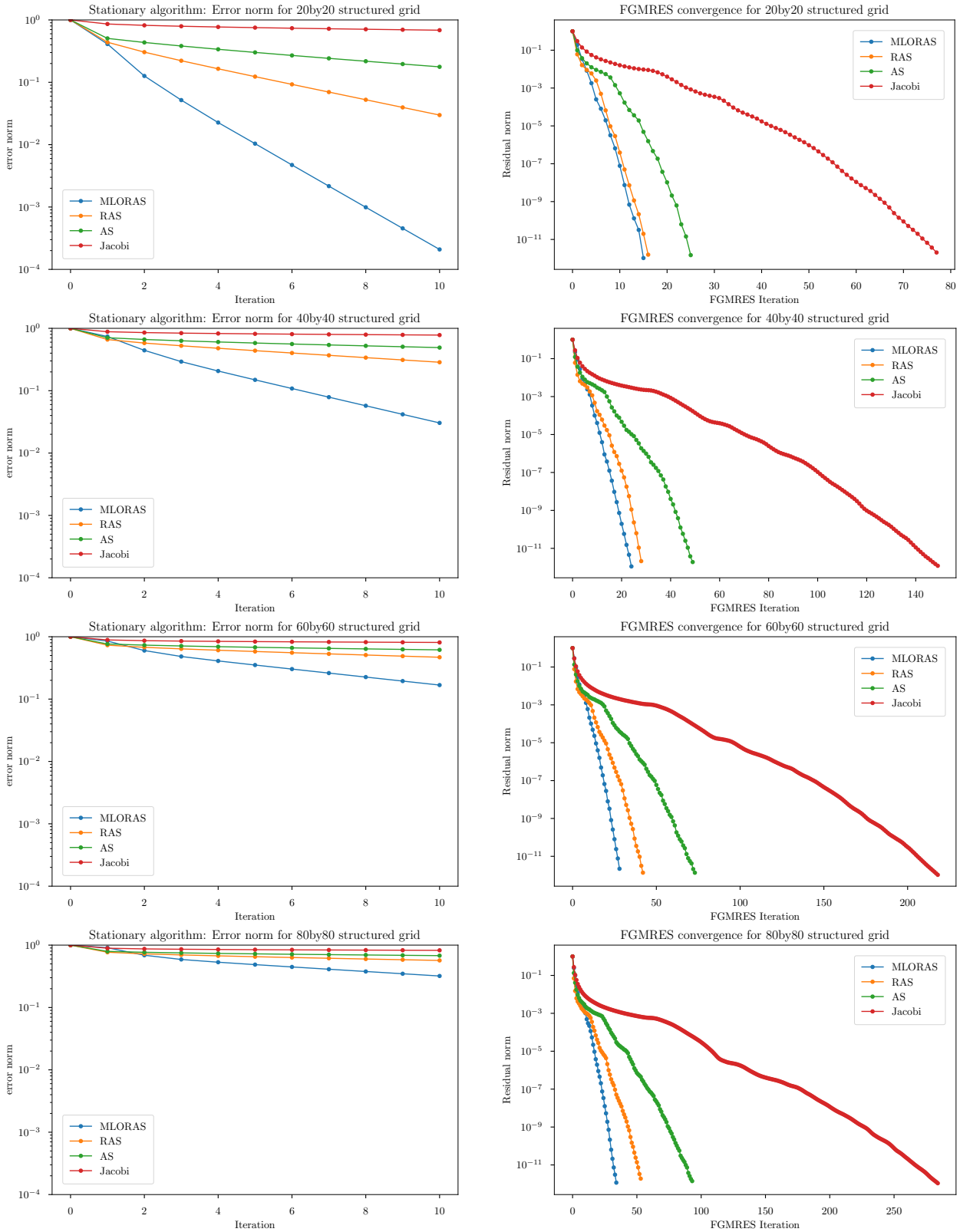

Figure 3.11: Evaluating the trained model on a 10×10 structured grid on larger grids. Comparing perfomance as stationary algorithm (left) and FGMRES preconditioner (right) against Jacobi, AS, and RAS.

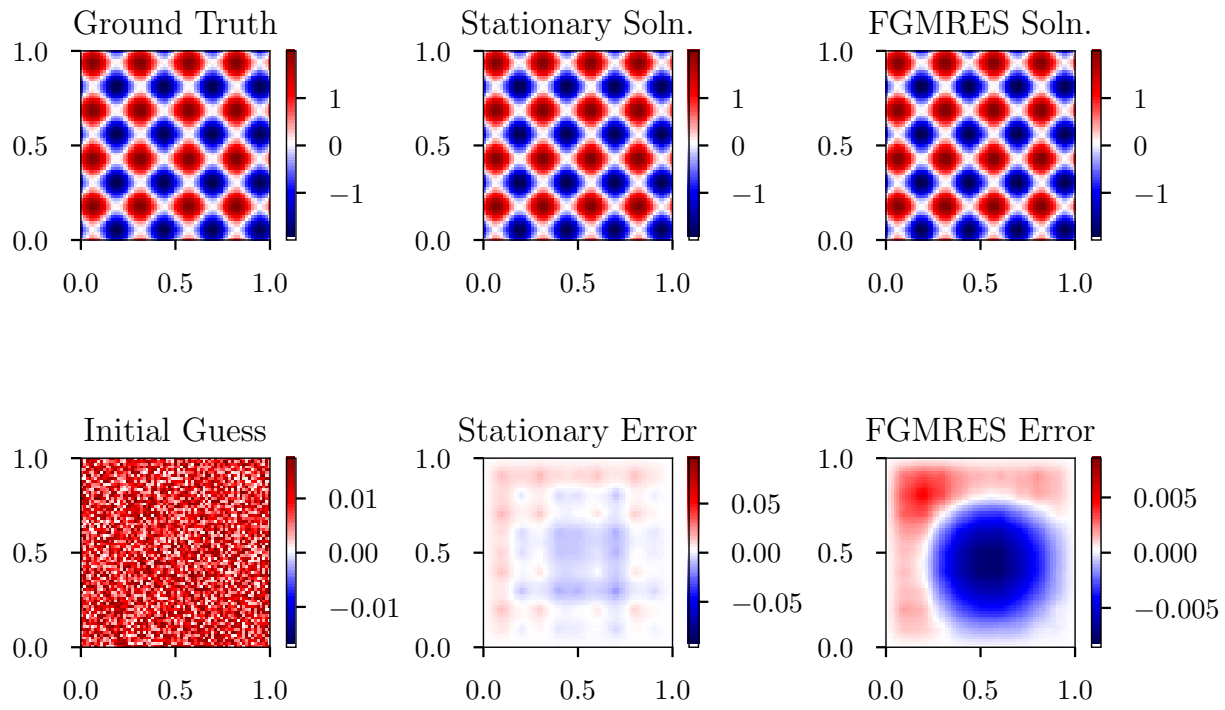

Figure 3.12: MLORAS (ours) solution plots. Top left: ground truth, top middle: MLORAS stationary solution after 10 iterations, top right: FGMRES solution with MLORAS preconditioner after 10 steps, bottom left: initial guess, bottom middle: error of MLORAS stationary solution (L_2 norm of error = 0.231), bottom right: error of FGMRES with MLORAS preconditioner solution (L_2 norm of error = 0.084).

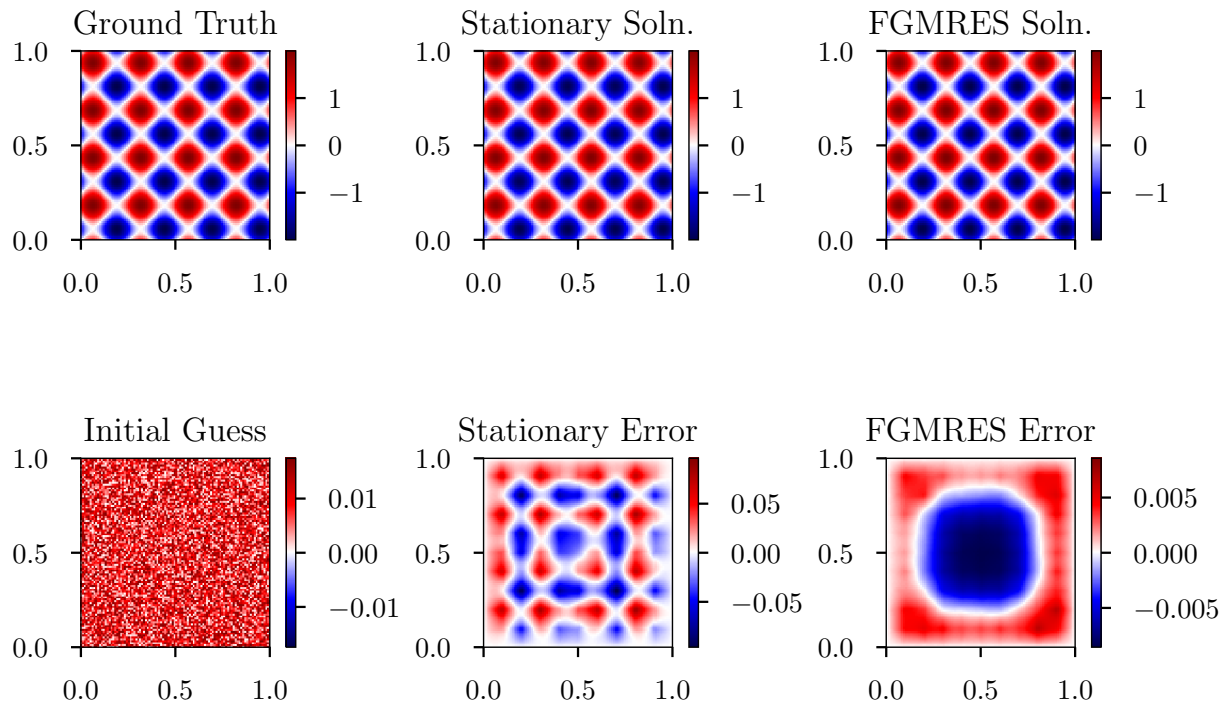

Figure 3.13: RAS solution plots. Top left: ground truth, top middle: RAS stationary solution after 10 iterations, top right: FGMRES solution with RAS preconditioner after 10 steps, bottom left: initial guess, bottom middle: error of RAS stationary solution (L_2 norm of error = 6.526), bottom right: error of FGMRES with RAS preconditioner solution (L_2 norm of error = 0.146).

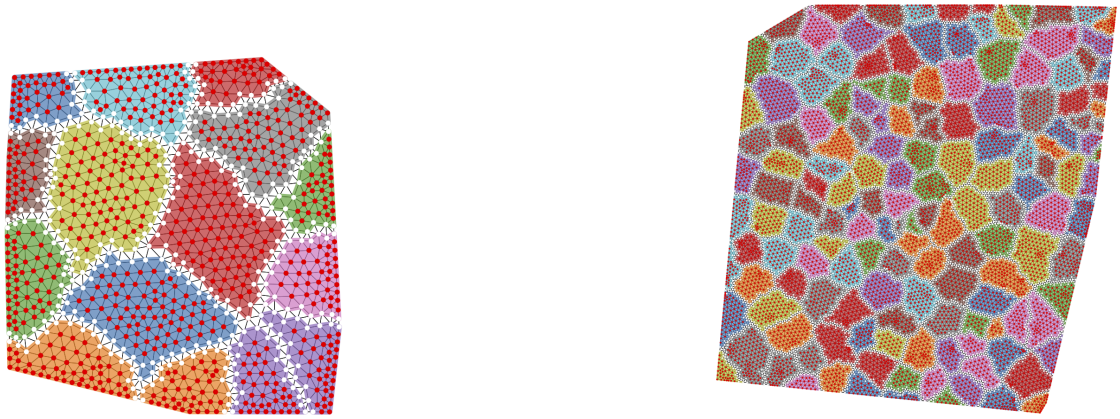

Figure 3.14: Example two unstructured test grids with 1018 nodes (left) and 10260 nodes (right).

subsets of $(0,1) \times (0,1)$; we solve the Helmholtz problem (3.1) on these domains with $\eta = 1$ and homogeneous Dirichlet boundary conditions, discretized with piecewise-linear finite elements.

Example convergence plots are shown in Figure 3.15, where our method (MLORAS) is compared to RAS (Restricted Additive Schwarz [6]), AS (Additive Schwarz), and Jacobi methods as a preconditioner for FGMRES. Here we see that the MLORAS network is able to learn optimized interface parameters for unstructured grids that outperform RAS, and that MLORAS can scale to problems that are much larger than those in the training set, which are all below $n = 1000$ nodes. Importantly, MLORAS retains an advantage over RAS even as the grid size increases.

Figure 3.16 shows the performance of three different methods across all unstructured test grids. The three methods are: (1) "F-norm optimized": the same network as MLORAS, but trained to directly optimize the Frobenius norm of the error propagation matrix, $\|A\|_F$, (2) the RAS [6] method, and (3) the MLORAS method. We do not show ORAS results here, as it cannot be applied to unstructured grids. Figure 3.16 reveals two important facts. First, MLORAS consistently outperforms RAS over the entire testing set, both as an FGMRES preconditioner and as a stationary algorithm, and this advantage is maintained even for large grids. Second, we see that the Frobenius norm is indeed a worse choice for the loss function than our new loss in (3.6). We see this from the fact that MLORAS has a worse Frobenius norm than either of the other two methods, but it has the best convergence rate. In addition, when we explicitly optimize the Frobenius norm (the "F-norm optimized" method), we see that we do obtain the lowest Frobenius norm, but this translates to the worst convergence rate.

3.5.3 Extra test cases

In this section, we test our method for more PDEs, including discontinuous and anisotropic diffusion coefficients, as described below, and we demonstrate our approach clearly win over RAS for both stationary iterations and preconditioned FGMRES. For each of the following problems, we generate training sets with the same parameters for meshes as in the existing results in the previous chapter. For testing on these problems, we consider nine domains with unstructured triangular grids, with sizes ranging from about 100 to over 30k nodes. The subdomains are generated using Lloyd aggregation (fully explained in section 3.2.2), with a fixed ratio of 0.015. The results of our method compared with RAS baseline for both the stationary algorithm and the FGMRES preconditioner are shown in Figures in the following subsections, and show little qualitative difference with the earlier results for Helmholtz.

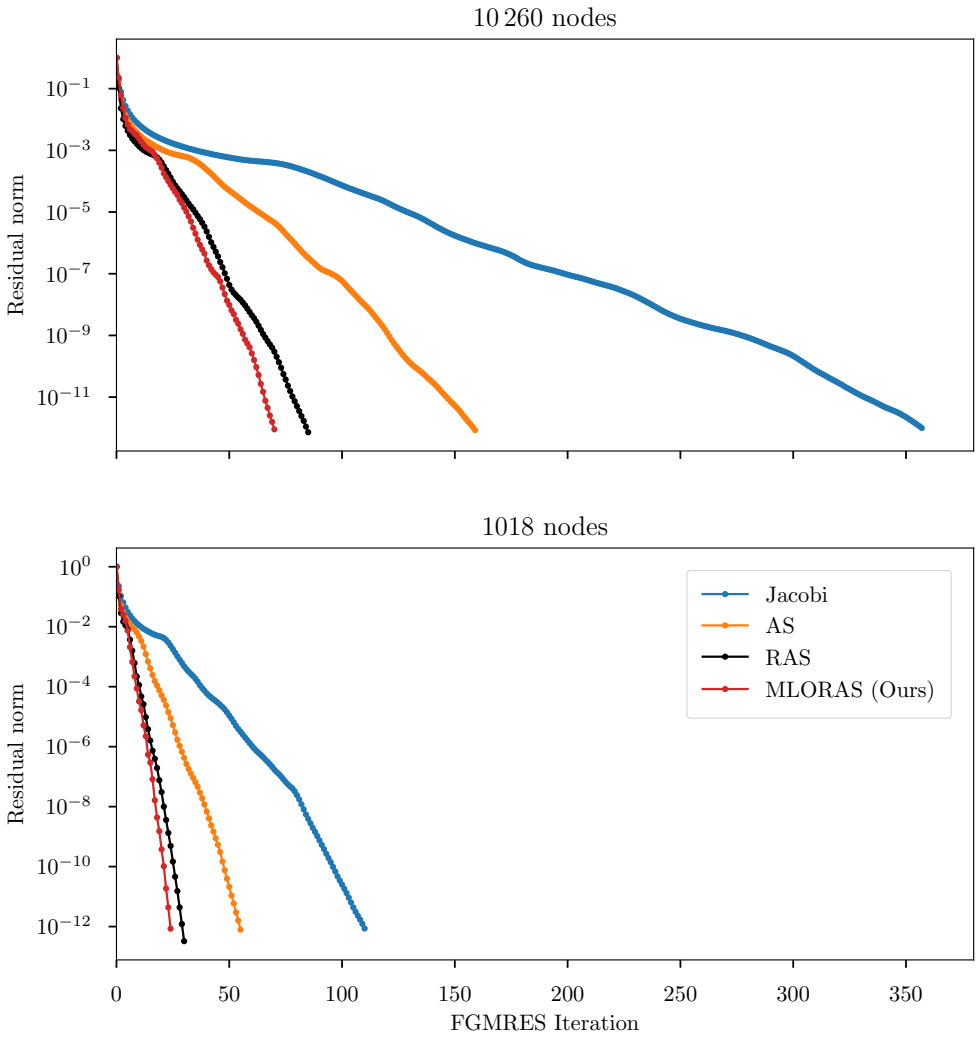

Figure 3.15: Example convergence on the test girds from Figure 3.14: 10260 nodes (top) and 1018 nodes: (bottom).

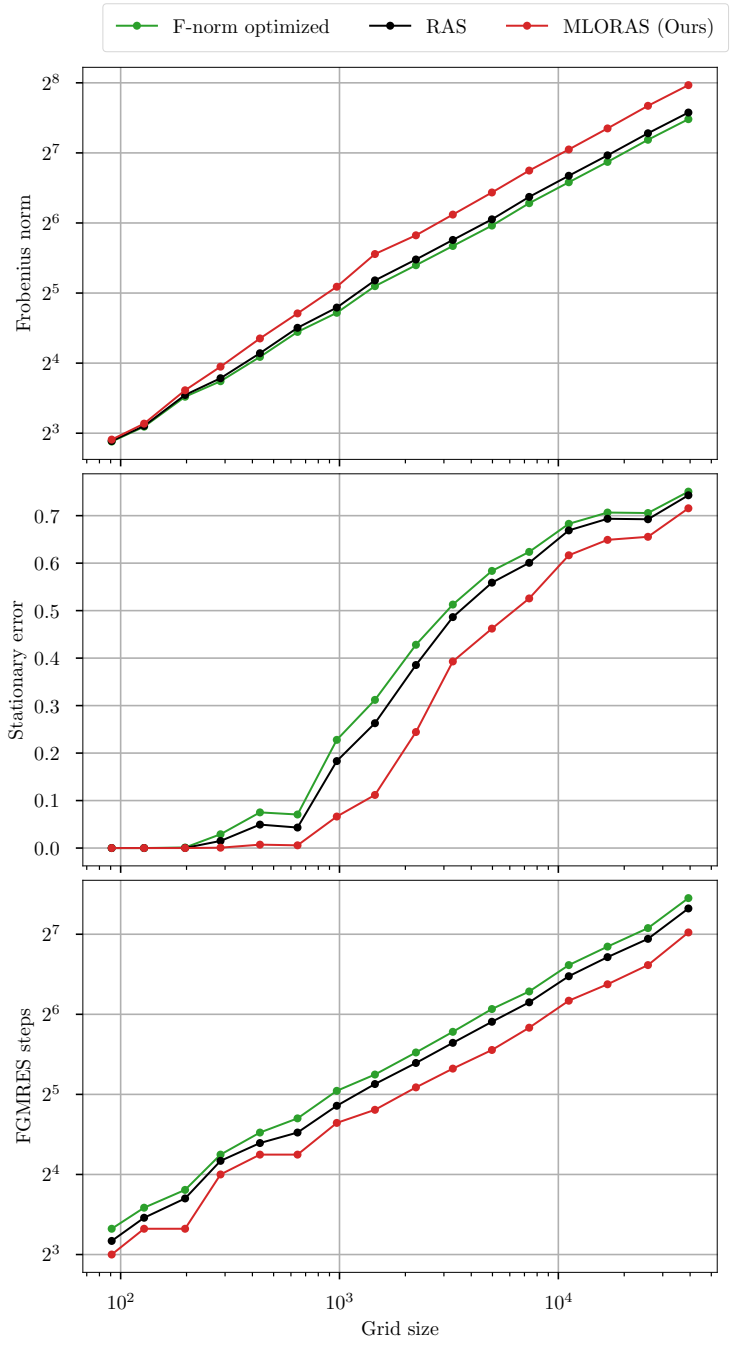

Figure 3.16: Convergence on all unstructured grids in the testing set. Top: Frobenius norm for each method on each test problem. Center: error reduction by the stationary iteration after 10 iterations. Bottom: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

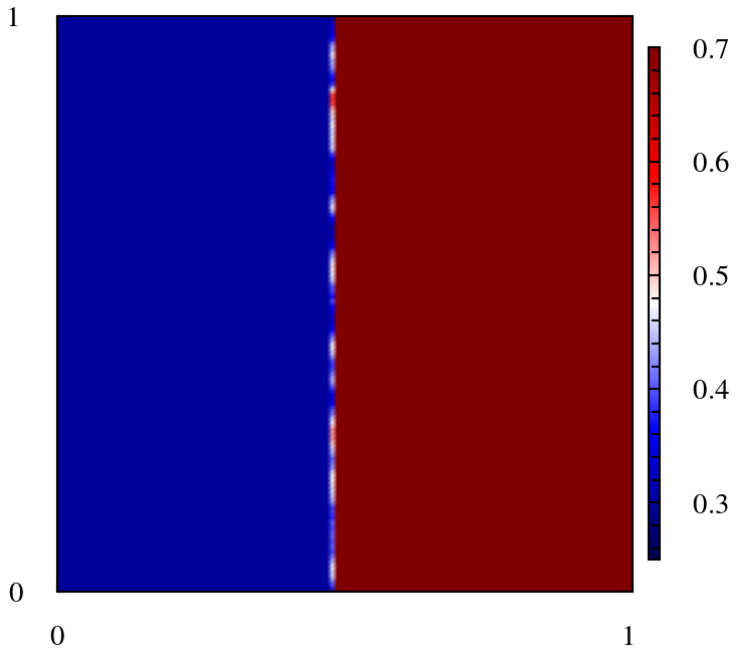

Figure 3.17: Visualization of the strong connections in the Poisson problem with discontinuous diffusion coefficient

Poisson problem with discontinuous diffusion coefficient

We also consider the 2D Poisson problem with discontinuous diffusion coefficient which is formulated as follows:

$$-\boldsymbol{\nabla}\cdot\kappa(x,y)\nabla u = f \quad \text{in } \Omega, \quad \kappa(x,y) = \begin{cases} 1000 & 0 < x < 0.5 \\ 1 & 0.5 \leq x < 1. \end{cases} \tag{3.20}$$

where Ω is, as before, defined as a convex subset of $(0,1)\times(0,1)$ and $\kappa(x,y)$ is the discontinuous diffusion coefficient.

Figure 3.17 shows the illustration (strong connections) of this problem on a unit square domain. Figure 3.18 shows the performance of our method against RAS, both for the stationary iterations and FGMRES.

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.

We illustrate the scaled strength of connection of rotated anisotropic poisson on a unit 2D square domain in Figure 3.19. Figure 3.20 shows the performance of our method against RAS, both for the stationary iterations and FGMRES.

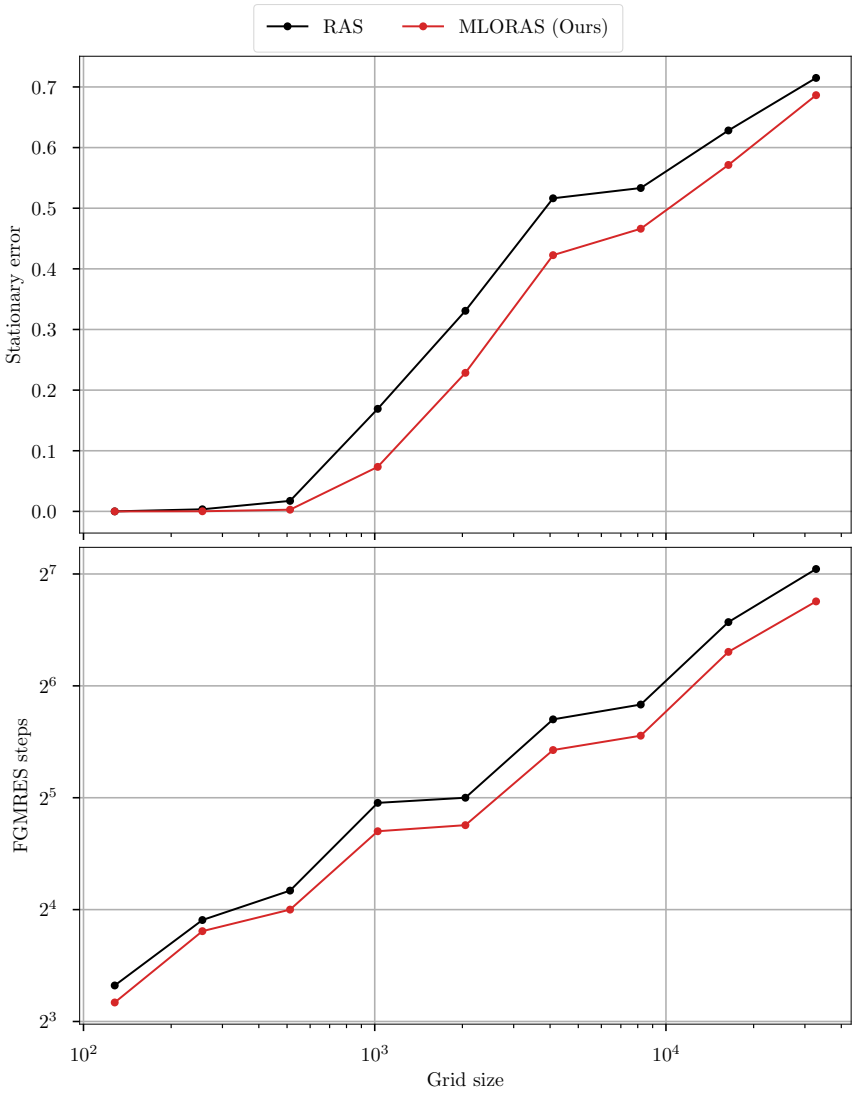

Figure 3.18: Discontinuous diffusion coefficient for Poisson problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

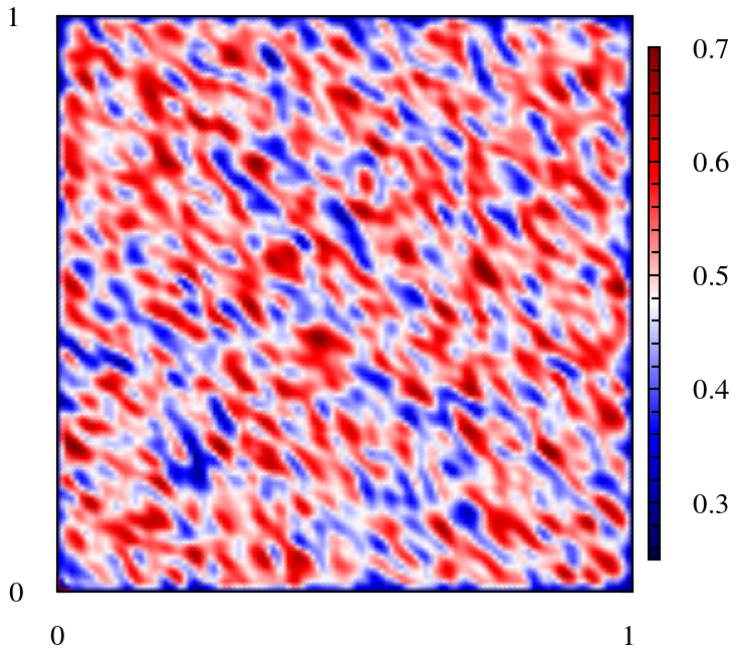

Figure 3.19: Visualization of the strong connections in the rotated anisotropic diffusion problem.

Low-diffusion inclusion

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

Figure 3.21 illustrates the problem visualization for the low-diffusion inclusion, and Figure 3.22 shows the performance of our method against RAS, both for the stationary iterations and FGMRES.

3.5.4 Ablation study

To understand the impact of the network structure on performance, we conduct an ablation study by varying the ResNet block length and the number of TAGConv layers. In each case, we train a network on five different training sets, each consisting of 1000 grids generated as described in Section 3.4.3. The trained networks are tested on 50 unstructured grids, each with 2400 to 2600 nodes. For each architecture, the mean performance is computed, together with error bars from the fivefold repetition, with results shown in Figure 3.23. We see that a higher residual block length is always better, but that the optimal number of TAGConv layers is 4, and using more layers actually decreases performance. This phenomenon has been observed in other contexts (e.g., [70] and [71]) and can be attributed to over-smoothing in GNNs. The use of residual blocks in our architecture is, thus, important to allow us to increase network depth without needing to increase the number of GNN layers.

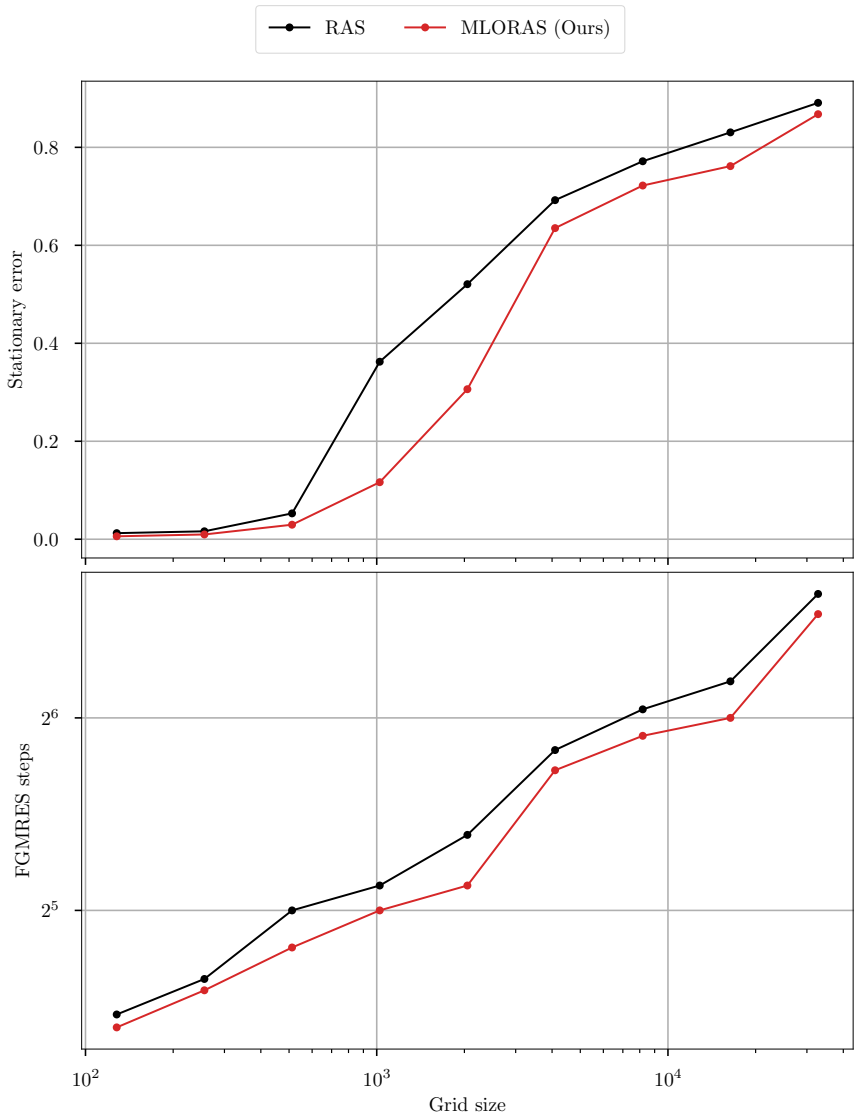

Figure 3.20: Rotated anisotropic diffusion problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

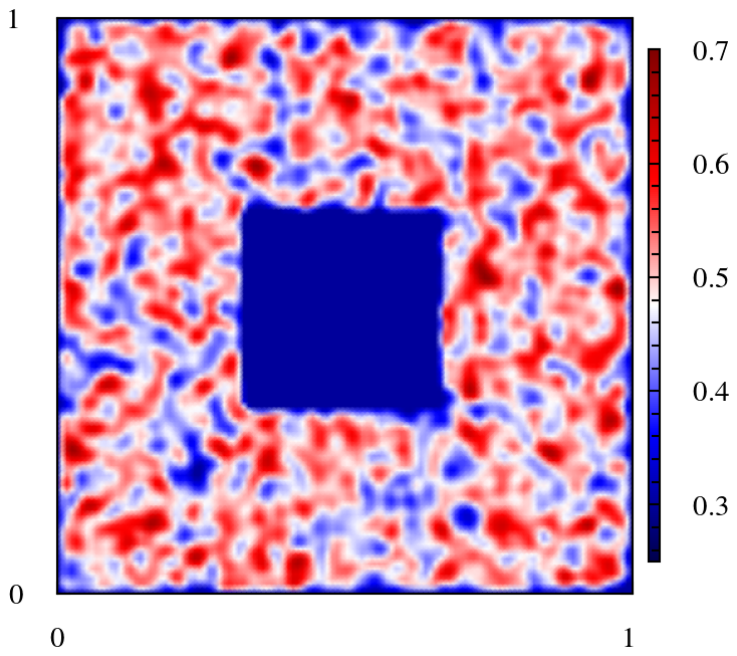

Figure 3.21: Visualization of the strong connections in the low diffusion problem.

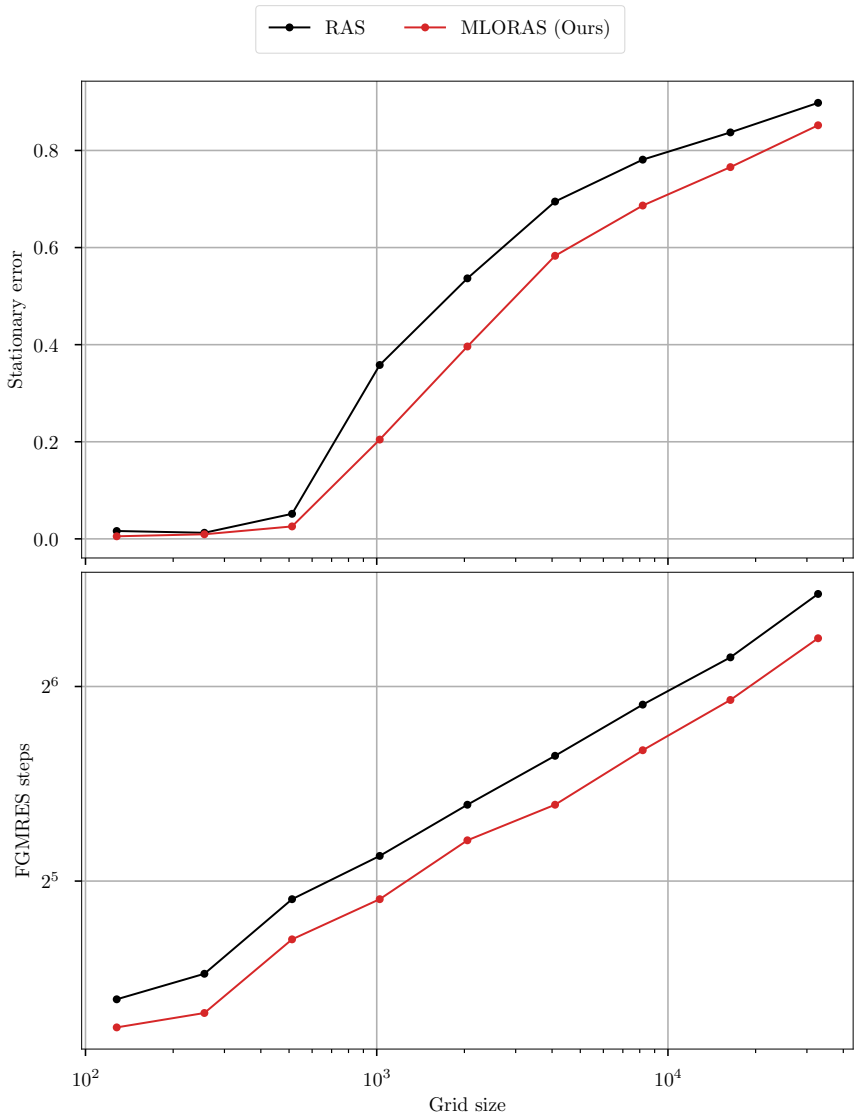

Figure 3.22: Low-diffusion inclusion problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

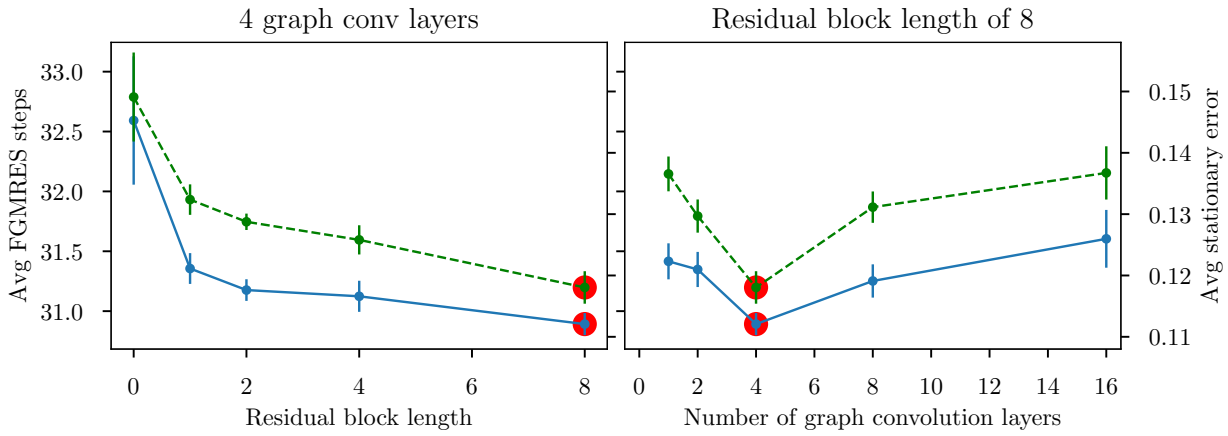

Figure 3.23: Ablation study results. The left panel varies the residual block length while keeping the number of TAGConv layers fixed at 4. The right panel varies the number of TAGConv layers while keeping the residual block length fixed at 8. The solid blue lines (left axis) show the average number of FGMRES steps needed to reduce the relative error below 10^{-12}, while the dashed green lines (right axis) show the average stationary algorithm error reduction after 10 iterations. The red circles mark the results for the network with the best performance (residual block length of 8 and 4 TAGConv layers), which was used for all other studies in this study. Error bars show one standard error of the mean.

Chapter 4

Multigrid Graph Neural Networks for Learning Multilevel Domain Decomposition Method

Portions of this chapter appear in the paper "MGGNN: Multigrid Graph Neural Networks for Learning Multilevel Domain Decomposition Method" accepted at International Conference on Machine Learning (ICML 2023) [72]

4.1 Introduction

Domain decomposition methods (DDMs) are widely used to solve discretized systems of PDEs. These methods come in one-level and multilevel variants, and they depend on various algorithmic and mathematical parameters. These parameters dictate the overlap, subdomain boundary conditions, and other characteristics of the DDM. While some efforts have been made to optimize these parameters, the focus has primarily been on the one-level scenario or specific cases such as structured-grid discretizations with regular subdomain construction. In this chapter, we introduce a new GNN architecture called multigrid graph neural networks (MG-GNN) to learn optimized parameters in two-level DDMs. MG-GNN is trained using a novel unsupervised loss function, allowing effective training even on small problems while achieving robust performance on unstructured grids that are significantly larger than those encountered during training. Our experimental results demonstrate that MG-GNN surpasses other popular hierarchical graph network architectures in terms of parameter optimization. This chapter is organized as follows: we fist provide background knowledge regarding multilevel DDMs in section 4.2. We then introduce our GNN architecture in section 4.3, and the unsupervised loss function we develop for this study in section 4.4. We finally provide the results and experiments in section 4.6.

4.2 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:

$$-\Delta\Phi = f, \tag{4.1}$$

where Δ is the Laplace operator and $f(x, y)$ and $\Phi(x, y)$ are real-valued functions. Alongside (4.1), 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 $\{D_1^0, D_2^0, \ldots, D_S^0\}$ (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 $A_C = P^T A P$.

The restricted additive Schwarz method (RAS) [6] 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

$$M_{\text{RAS}} = \sum_{i=1}^{S} (\tilde{R}_i^\delta)^T (A_i^\delta)^{-1} R_i^\delta, \tag{4.2}$$

$$C_{\text{2-RAS}} = P(P^T A P)^{-1} P^T, \tag{4.3}$$

where $A_i^\delta = (R_i^\delta)^T A R_i^\delta$. 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 $M_{\text{2-RAS}} = C_{\text{2-RAS}} + M_{\text{RAS}} - C_{\text{2-RAS}} A M_{\text{RAS}}$, 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:

$$\alpha u + \vec{n} \cdot \nabla u = g(x), \tag{4.4}$$

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

$$M_{\text{ORAS}} = \sum_{i=1}^{S} \left(\tilde{R}_i^\delta\right)^T \left(\tilde{A}_i^\delta\right)^{-1} R_i^\delta, \tag{4.5}$$

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 $M_{2\text{-ORAS}} = C_{2\text{-RAS}} + M_{\text{ORAS}} - C_{2\text{-RAS}} A M_{\text{ORAS}}$.

The work of [49] 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 4.9, the two-level methods are significantly more robust. Furthermore, as we show in Section 4.6.2, 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.

4.3 Multigrid graph neural network

The multigrid neural architecture [23] 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 [23], 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 4.2 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 $\ell \in \{0, 1, \ldots, L-1\}$, 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 $\ell \in \{0, 1, \ldots, L-2\}$:

$$X^{(\ell+1)} = R^{(\ell)} X^{(\ell)}, \tag{4.6}$$

$$A^{(\ell+1)} = R^{(\ell)} A^{(\ell)} (R^{(\ell)})^T. \tag{4.7}$$

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 $\ell \in \{0, 1, \ldots, L-1\}$:

$$\dot{X}^{\ell\to k} = F^{\ell\to k}(X_m^{(\ell)}, X_m^{(k)}, R^{(\ell\to k)}) \tag{4.8}$$

$$\tilde{X}_m^{(\ell)} = [\dot{X}^{0\to\ell} \| \dot{X}^{1\to\ell} \| \ldots \| \dot{X}^{k-1\to\ell}] \tag{4.9}$$

$$X_{m+1}^{(\ell)} = \mathrm{GNN}^{(\ell)}(\tilde{X}_m^{(\ell)}, A^{(\ell)}) \tag{4.10}$$

where $\|$ denotes concatenation, and $\mathrm{GNN}^{(\ell)}$ 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

$$m_v = g^{\ell\to k}\left(\square_{\omega\in\mathcal{N}(v)} f^{\ell\to k}(x_v, x_\omega, e_{v\omega}), x_v \right) \tag{4.11}$$

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 4.1 for visualization of up-sampling and down-sampling in MG-GNN.

In this study, we consider a two-level MG-GNN (see Figure 4.2) 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 [57], [58]) in a connected graph. As mentioned earlier, the MG-GNN architecture could alternatively use any pooling/clustering method such as DiffPool [73], top-K pooling [22], ASAP [74], SAGPool [75], to name but a few. However, for the case of this study, 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.

4.4 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 $T = I - M_{2\text{-ORAS}}A = (I - C_{2\text{-ORAS}}A)(I - M_{\text{ORAS}}A)$ 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 [60], numerical unsuitability of backpropagation of an eigendecomposition makes it infeasible

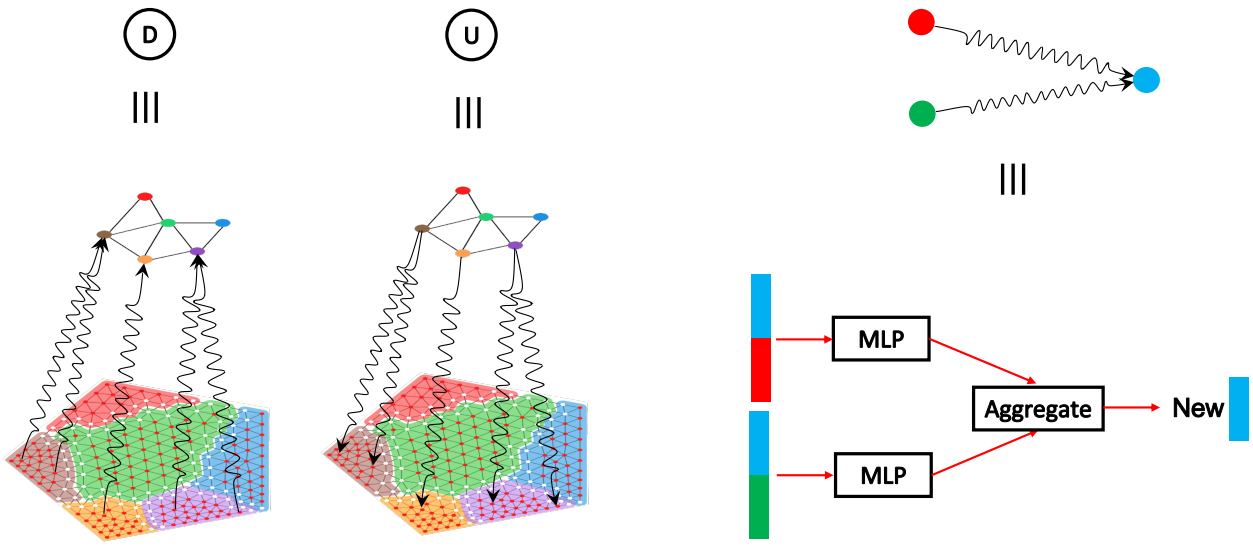

Figure 4.1: Upsampling and downsampling in MG-GNN.

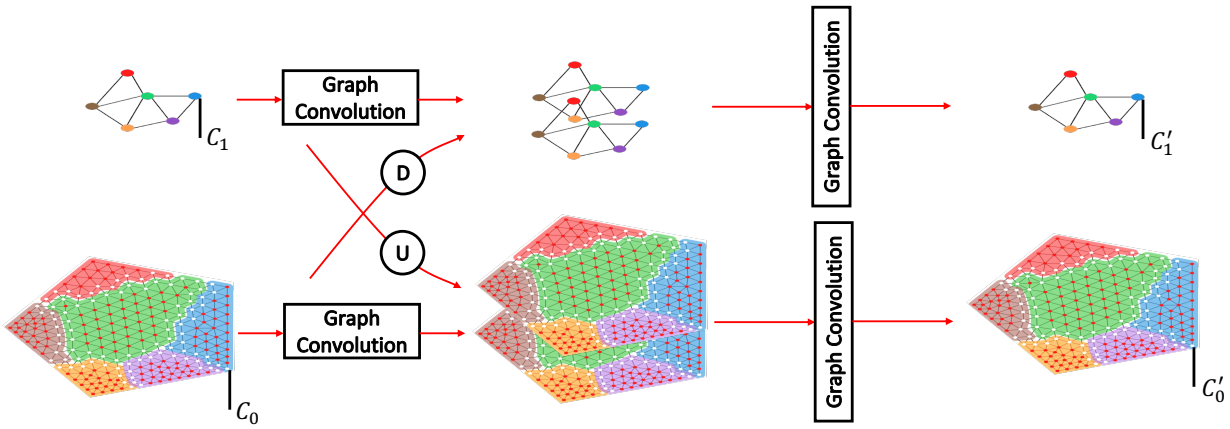

Figure 4.2: One layer of MG-GNN. c_i and c_i' denote the feature dimensions of different levels before and after passing through an MG-GNN layer, respectively.

to directly minimize $\rho(T)$. To this end, [12] 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 [49] 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 [49] and, in Section 5.3.2, 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 section 4.5 for more discussion on inputs and outputs of the network):

$$P^{(\theta)}, L_1^{(\theta)}, L_2^{(\theta)}, \ldots, L_S^{(\theta)} \leftarrow \psi^{(\theta)}(D). \tag{4.12}$$

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 (4.12). 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 [49], 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

$$X = [x_1, x_2, \ldots, x_m], \forall_j\ x_j \sim \mathbb{R}^n \text{ uniformly}, \|x_j\| = 1, \tag{4.13}$$

where each x_j is sampled uniformly randomly on a unit sphere in $\mathbb{R}^n$ using the method introduced in [62]. We then define

$$Y_K^{(\theta)} = \left\{\left\|\left(T^{(\theta)}\right)^K x_1\right\|, \left\|\left(T^{(\theta)}\right)^K x_2\right\|, \ldots, \left\|\left(T^{(\theta)}\right)^K x_m\right\|\right\}. \tag{4.14}$$

Note that $\left\|\left(T^{(\theta)}\right)^K x_j\right\|$ is a lower bound for $\left\|\left(T^{(\theta)}\right)^K\right\|$. [49] 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 5.3.2, 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,

$$\mathcal{L}^{(\theta)} = \langle \text{softmax}(Z^{(\theta)}),\ Z^{(\theta)}\rangle + \gamma \text{tr}\left((P^{(\theta)})^T A P^{(\theta)}\right), \tag{4.15}$$

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 [76], [77]. In Section 5.3.2, we show the significance of this term in the overall performance of our model. Nevertheless, for the first part of the new loss function (4.15), we prove that it convergence to the spectral radius of the error propagation matrix in a suitable limit. First, we include two lemmas:

Lemma 4. *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 $\{x \in \mathbb{R}^n \,|\, \|x\| = 1\}$, and*

$Z = \max\{\|T^k x_1\|^{\frac{1}{k}}, \|T^k x_2\|^{\frac{1}{k}}, \dots, \|T^k x_m\|^{\frac{1}{k}}\}$ *then, with a probability of at least* $(1-\delta)$*, the following hold:*

$$0 \le \|T^k\|^{\frac{1}{k}} - Z \le \epsilon, \tag{4.16}$$

$$\rho(T) - \xi \le Z. \tag{4.17}$$

Proof. The left side of the first inequality is achieved by considering the definition of matrix norm, i.e. for any $1 \le i \le m$, $\|T^k x_i\| \le \sup_{\|x\|=1} \|T^k x\| = \|T^k\|$, then taking the k^{th} root of both sides. For the right side of the first inequality, consider the point $x^* \in \{x \in \mathbb{R}^n \,|\, ||x|| = 1\}$ 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 $m \ge M_1 > \frac{\log(\delta)}{\log\left(1-\frac{\tilde{S}}{S}\right)}$, then, since $0 < \delta < 1$, we have:

$$P(\|x^* - x_i\| > \tilde{\epsilon}, \ \ \forall_i) = \left(1 - \frac{\tilde{S}}{S}\right)^m \le \delta \tag{4.18}$$

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

$$\left\|T^k x^*\right\|^{\frac{1}{k}} - \left\|T^k x_1\right\|^{\frac{1}{k}} \le \left(\left\|T^k x^*\right\| - \left\|T^k x_1\right\|\right)^{\frac{1}{k}} \le \left\|T^k\right\|^{\frac{1}{k}} \|x^* - x_1\|^{\frac{1}{k}} \le \|T^k\|^{\frac{1}{k}} \tilde{\epsilon} = \epsilon \tag{4.19}$$

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

For the second inequality, since $\rho(T) \le \|T^k\|^{\frac{1}{k}}$, choose M_2 such that, with probability $1-\delta$, (4.16) holds for $\epsilon = \|T^k\|^{\frac{1}{k}} - \rho(T) + \xi > 0$. Rearranging (4.16) then yields (4.17) for any $m \ge M = \max\{M_1, M_2\}$. □

We next state the main result on optimality.

Theorem 5. *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* $Z_k = \max\{\|T^k x_1\|^{\frac{1}{k}}, \|T^k x_2\|^{\frac{1}{k}}, \dots, \|T^k x_m\|^{\frac{1}{k}}\}$*, then* $Z = (Z_1, Z_2, \dots, Z_K)$ *satisfies:*

$$P\left(|\langle softmax(Z), \ Z\rangle - \rho(T)| \le \epsilon\right) > 1 - \delta. \tag{4.20}$$

Proof. Since $\rho(T) \le \|T^k\|^{\frac{1}{k}}$ 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 $k > K^*$, $0 \le \|T^k\|^{\frac{1}{k}} - \rho(T) < \alpha$. Take $0 < \alpha < \min\{\frac{\epsilon}{2}, \log\left(\frac{e^{-\epsilon}(\epsilon+\rho(T))}{\frac{\epsilon}{2}+\rho(T)}\right)\}$, let

$$u = \max\{\max_{1\le k\le K^*}\{\|T^k\|^{\frac{1}{k}}\} + \alpha, \rho(T) + 2\alpha\}, \ \ \tilde{\delta} = 1 - (1-\delta)^{\frac{1}{K}}, \tag{4.21}$$

and take

$$K > \max\{\frac{K^*(ue^u - (\rho(T)+\alpha)e^{\rho(T)+\alpha})}{e^{\rho(T)-\epsilon}(\epsilon + \rho(T) - (\rho(T)+\alpha)e^{\alpha+\epsilon})}, K^*\}. \tag{4.22}$$

. 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 4, for any $1 \le i \le K^*$ and $K^* < j \le K$, there exists

$n_i, n_j \in \mathbb{N}$ such that:

$$P(\rho(T) - \epsilon \leq Z_i \leq u) > 1 - \tilde{\delta} \qquad \text{for } m > n_i, \tag{4.23}$$

$$P(\rho(T) - \epsilon \leq Z_j \leq \rho(T) + \alpha) > 1 - \tilde{\delta} \qquad \text{for } m > n_j. \tag{4.24}$$

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 (4.23) and (4.24) 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:

$$-\epsilon = \frac{(\rho(T)-\epsilon)\sum_{i=1}^{K} e^{Z_i}}{\sum_{i=1}^{K} e^{Z_i}} - \rho(T) \leq \frac{\sum_{i=1}^{K} Z_i e^{Z_i}}{\sum_{i=1}^{K} e^{Z_i}} - \rho(T) \tag{4.25}$$

$$= \langle \text{softmax}(Z),\ Z\rangle - \rho(T) \leq \frac{uK^* e^u + (K-K^*)(\rho(T)+\alpha)e^{\rho(T)+\alpha}}{K e^{\rho(T)-\epsilon}} - \rho(T) \tag{4.26}$$

$$= \frac{K^*(ue^u - (\rho(T)+\alpha)e^{\rho(T)+\alpha})}{K e^{\rho(T)-\epsilon}} + (\rho(T)+\alpha)e^{\alpha+\epsilon} - \rho(T) \leq \epsilon, \tag{4.27}$$

where the last inequality is obtained by the choice of K. □

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 6. *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.*

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. □

4.5 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 4.3,

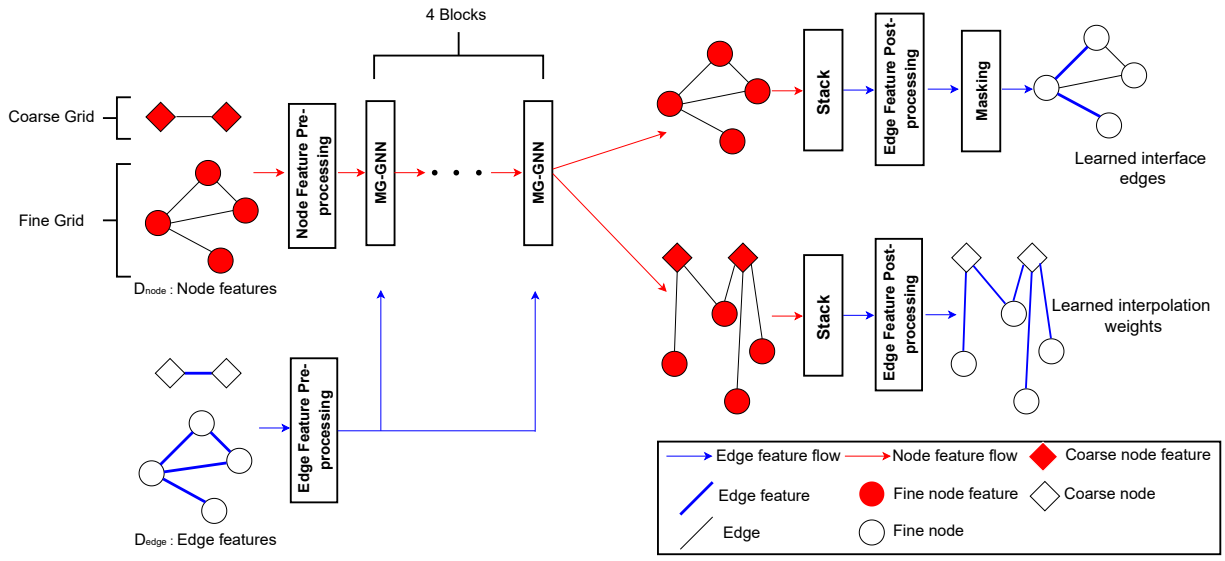

Figure 4.3: GNN architecture used in this study.

Equations (4.6) and (4.7), 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 $\text{GNN}^{(\ell)}$ in (4.10) and $F^{\ell\to\ell}$ in (4.8), we use a TAGConv layer [56] and, for $F^{\ell\to k}$ with $\ell \neq k$, we use a heterogeneous message passing GNN as shown in Equation (4.11). Specifically, we choose summation as the permutation invariant operator in (4.11) 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 4.4 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 4.3.

4.6 Experiments

4.6.1 Training

We train each model on 1000 grids of sizes ranging from 800–1000 nodes. The grids are generated randomly as a convex polygon and using PyGMSH [65] for meshing its interior. The subdomains are generated using Lloyd clustering on the graph [58], 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 section 4.5 Figure 4.3, the weight of the edges connecting the coarse and fine grids determine the interpolation

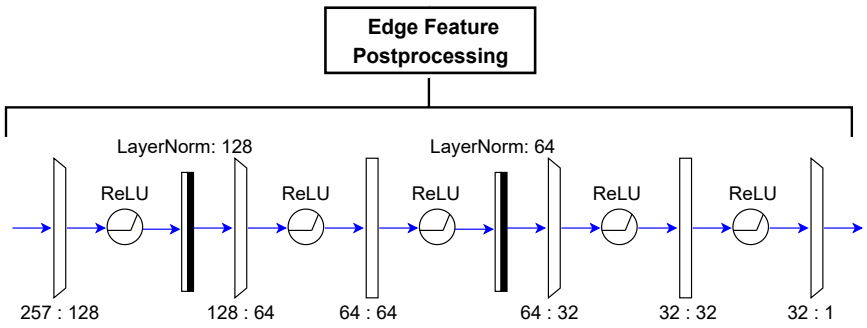

Figure 4.4: Edge feature post-processing block.

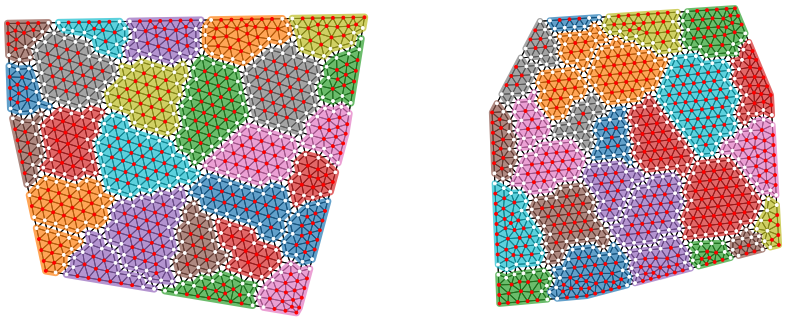

Figure 4.5: Training grid examples with about 1k nodes.

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 5.3.2, 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. Figure 4.5 shows several example training grids.

The model is trained for 20 epochs with batch size of 10 using the ADAM optimizer [66] with a fixed learning rate of 5×10^{-4}. For the full discussion on model architecture, refer back to section 4.5 and Figure 4.3. For the loss function parameters introduced in Section 4.4, we use $K = 10$ iterations and $m = 100$ samples. We developed our code[1] using PyAMG [68], NetworkX [69], and PyTorch Geometric [67]. 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 4.4, 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 later, 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 [78], which is left for future studies.

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 4.6.

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

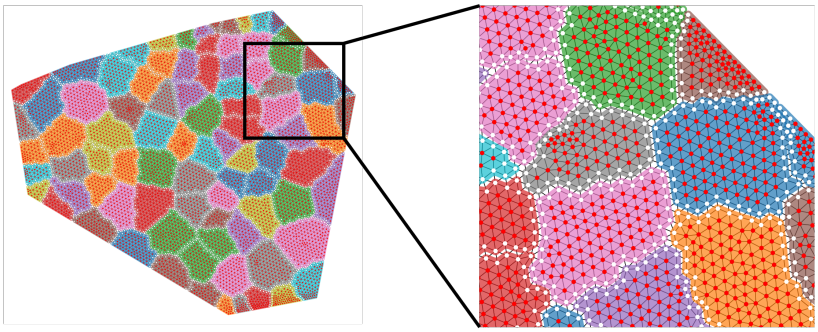

Figure 4.6: Test grid example with about 7.4k nodes.

4.6.2 Interface values and interpolation operator

As mentioned in the Section 4.4, to optimize two-level RAS, one could optimize the parameters in the interface conditions (4.2) and/or the interpolation operator (4.3). 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 [49] and [7]. 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 4.6.1; 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 4.3), learning the interface values and the interpolation operator. We compare the performance of these models with classical RAS in Figure 4.7 as a stationary algorithm a preconditioner 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 (4.3) 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 (4.2)), that cannot (by themselves) make up for a poor coarse-grid correction process. Learning both operators clearly results in the best performance in Figure 4.7, where the interpolation operator can be adapted to best complement the effects of the learned interface values.

4.6.3 Loss function and sparsity variants

We first compare five variants of our method with the RAS baseline, as shown in Figure 4.8. The main model is trained as described in Section 4.6.1 with the loss function from Section 4.4. 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 [49]. The variant labeled "Max+Trace loss" is trained with the loss function from [49] plus the $\gamma \mathrm{tr}(P^T A P)$ term. Similarly, the variant labeled "Softmax loss" is trained by removing the $\gamma \mathrm{tr}(P^T A P)$ part from the loss function in (4.15). 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 (4.15). As shown in Figure 4.8, 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

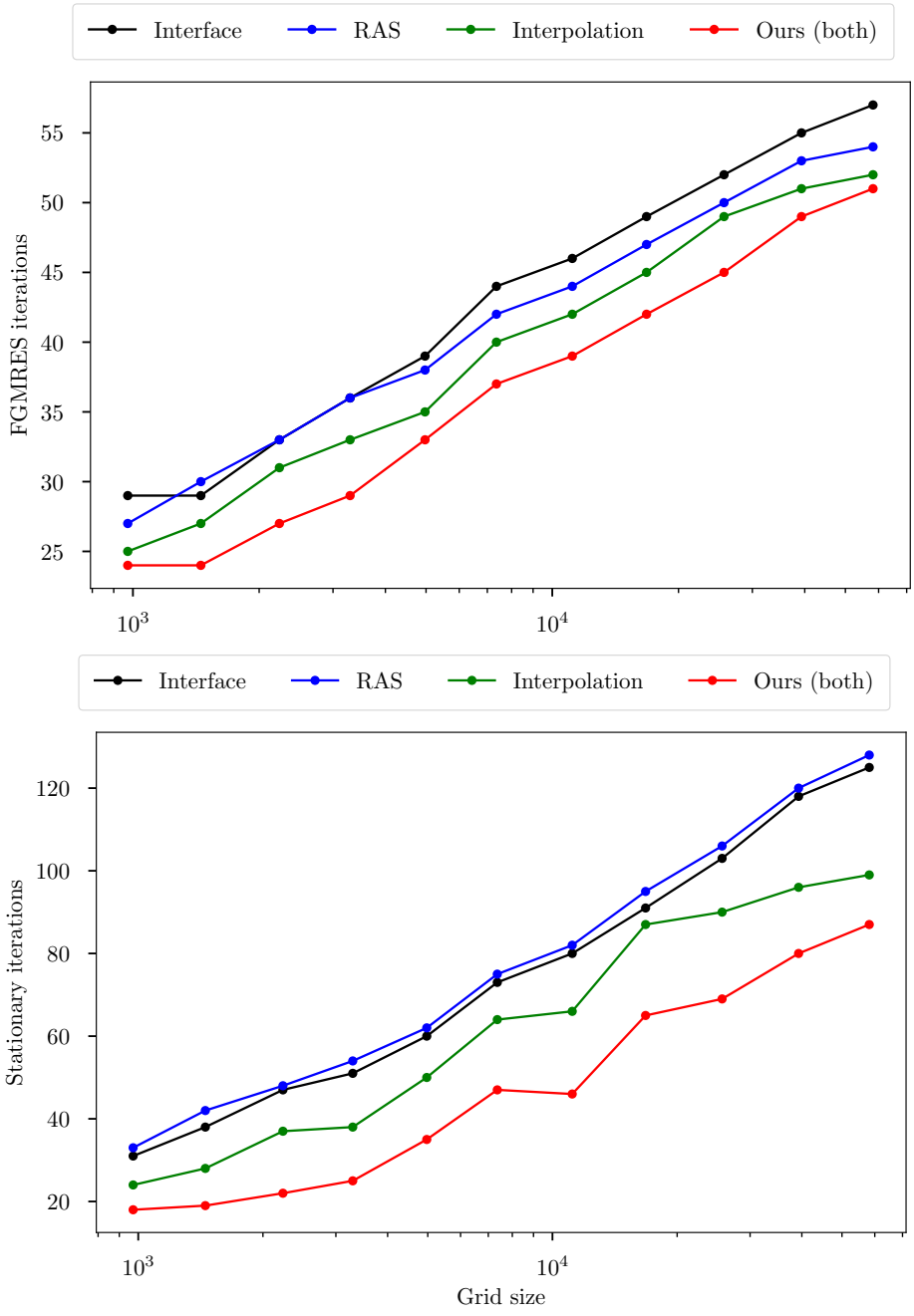

Figure 4.7: Effect of learning interface values, interpolation operator, or both on stationary and FGMRES iterations.

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 [49] in Figure 4.9.

4.6.4 Comparing the 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. Figure 4.10 shows the stationary solve times on some of the test grids (in seconds).

4.6.5 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. In Figure 4.11, 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:

4.6.6 Solution plots

We have compared the solution plots obtained using our method with the RAS method for the Poisson problem. To conduct the comparison, we considered a structured grid of size 100×100 on the domain $(0,1) \times (0,1)$. For this problem, we used the true solution given by $u^* = \sin(8\pi x) + \sin(8\pi y)$.

To initiate the iterative process, we started with an initial random guess having an L_2 norm of 1. We then performed 10 iterations of both the RAS and MLORAS methods as stationary algorithms, obtaining solutions for each method. Furthermore, we ran 10 iterations of the FGMRES method using MLORAS and RAS preconditioners on the initial guess, yielding predictions for both methods.

The solution plots for our MLORAS method are depicted in Figure 4.12, while those for the RAS method are shown in Figure 4.13.

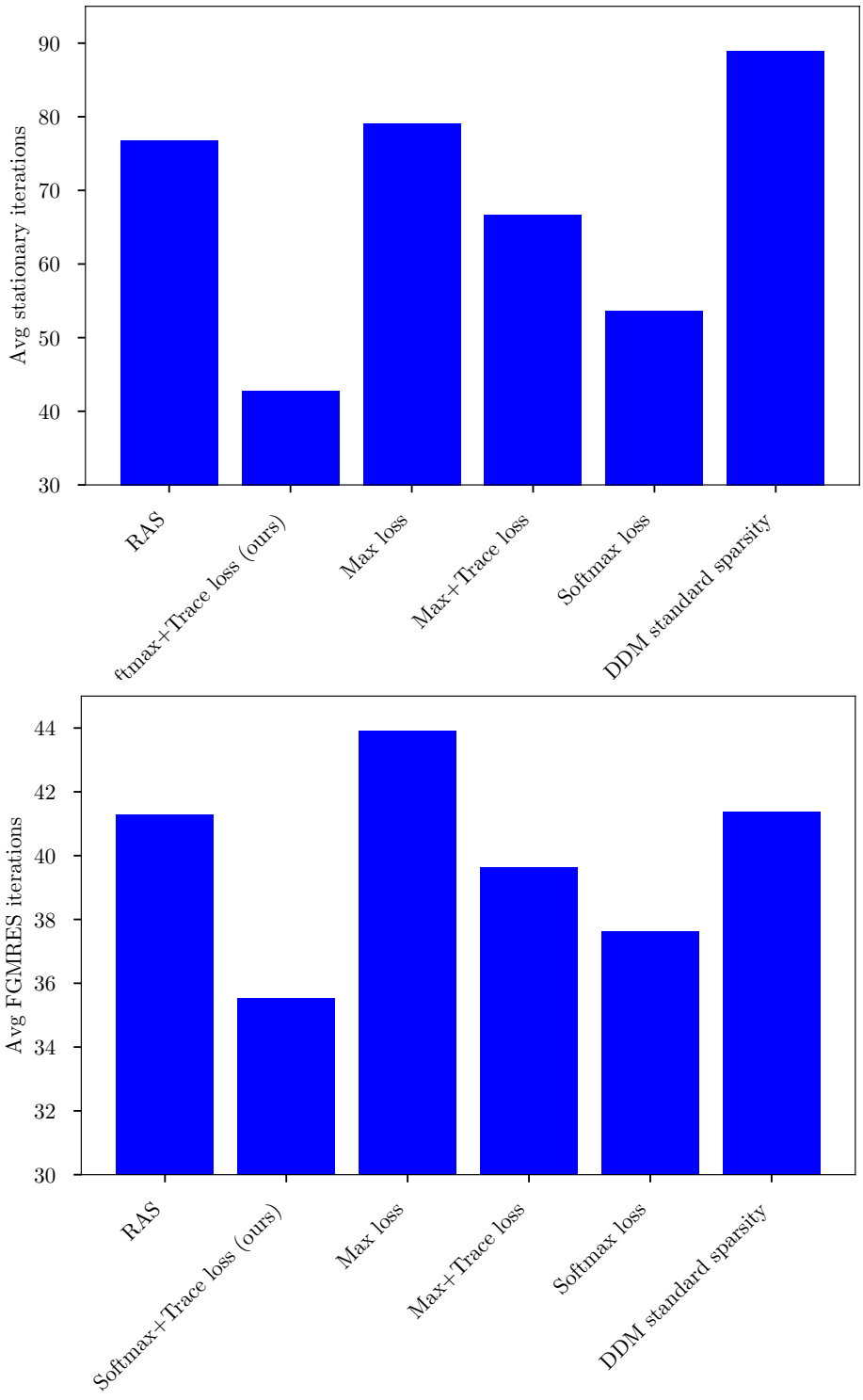

Figure 4.8: Effect of every ingredient in the model on average stationary and FGMRES iterations. All three variants outperforming the RAS baseline are utilizing modifications introduced in this study (see (4.15)) compared to the "Max loss" from [49].

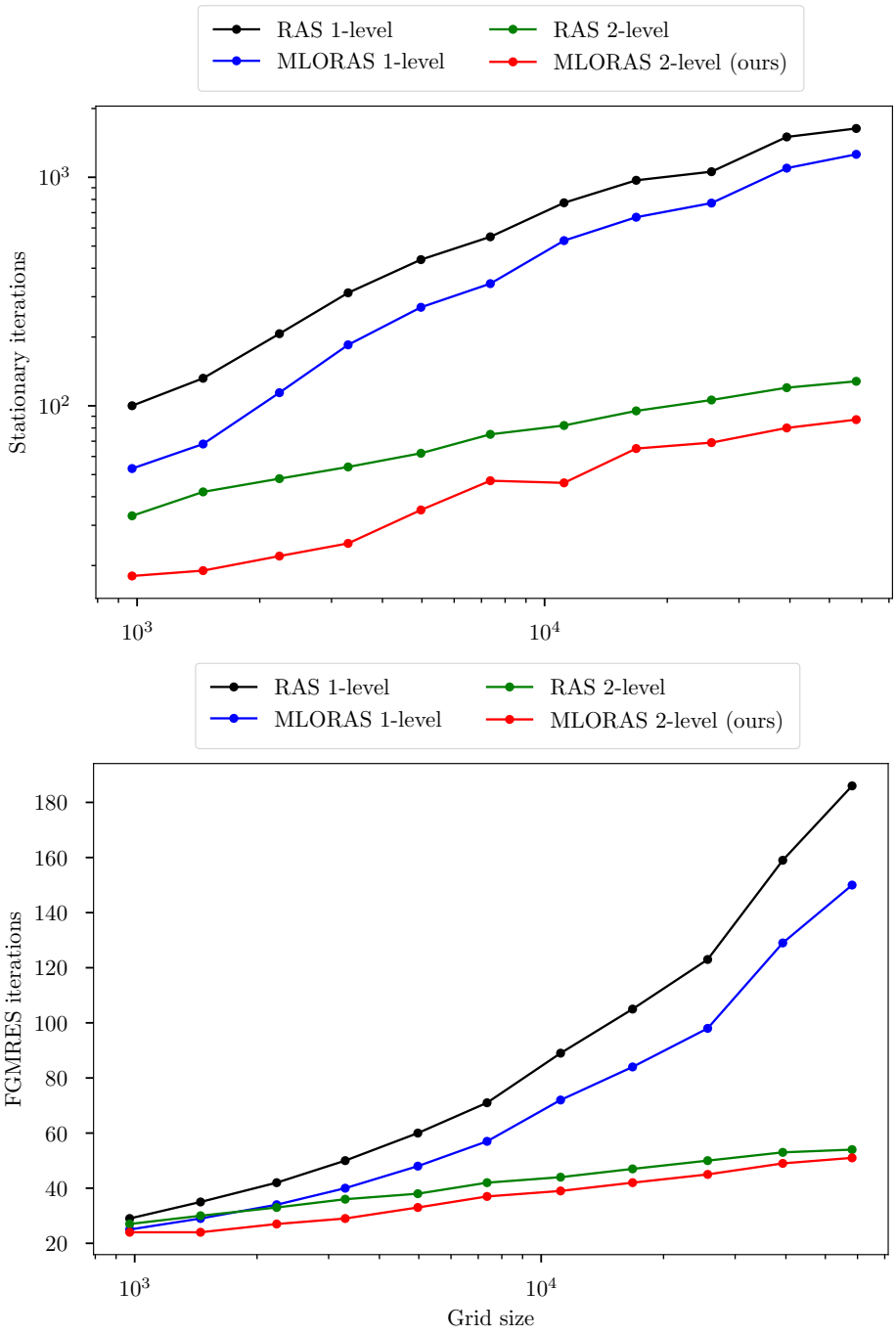

Figure 4.9: Comparison of stationary and FGMRES iterations of 2-level methods with 1-level methods from [49].

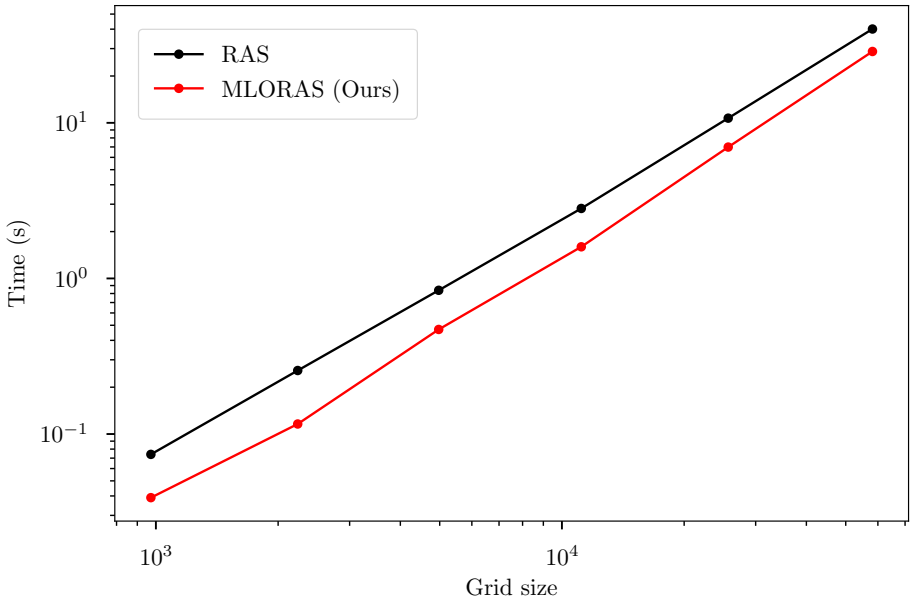

Figure 4.10: Comparison of stationary solver time of RAS and MLORAS (ours) against grid size.

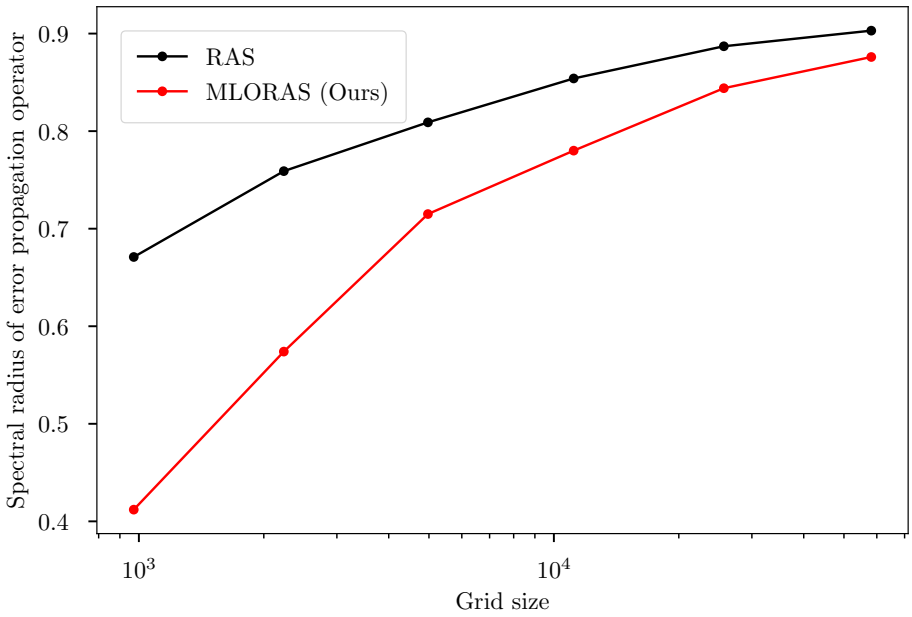

Figure 4.11: Comparison of the spectral radius of the error propagation operator obtained from RAS and MLORAS (ours) against grid size.

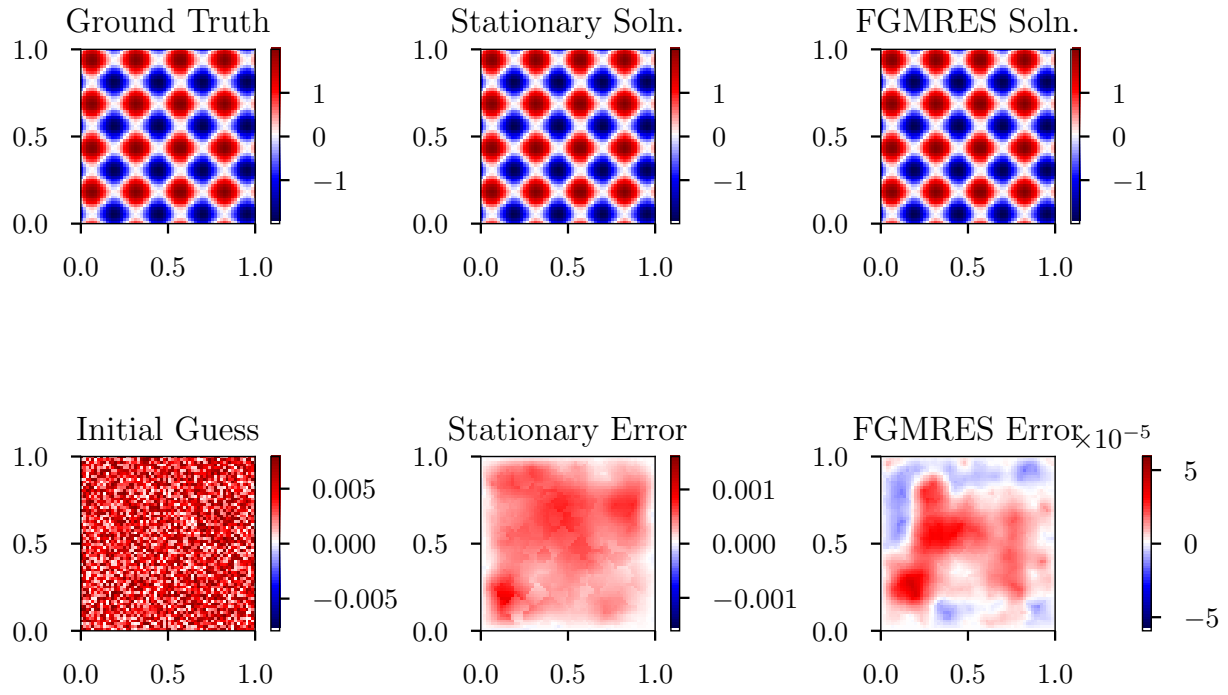

Figure 4.12: MLORAS (ours) solution plots. Top left: ground truth, top middle: MLORAS stationary solution after 10 iterations, top right: FGMRES solution with MLORAS preconditioner after 10 steps, bottom left: initial guess, bottom middle: error of MLORAS stationary solution (L_2 norm of error = 0.03671), bottom right: error of FGMRES with MLORAS preconditioner solution (L_2 norm of error = 0.0012).

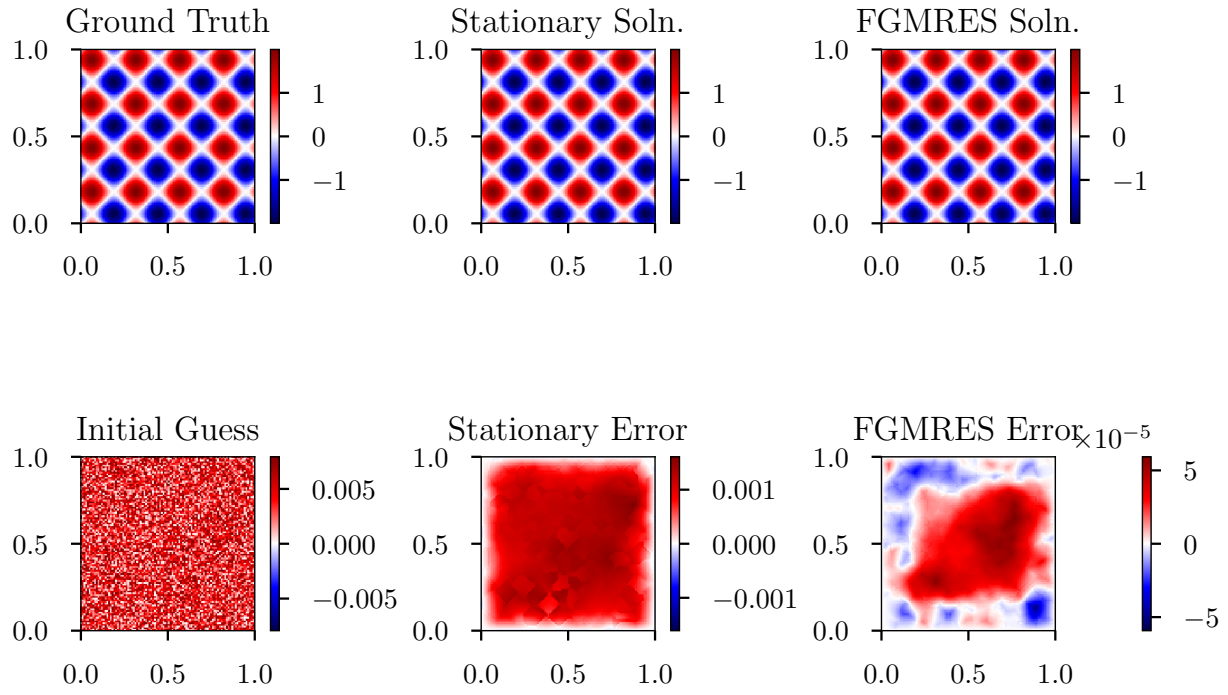

Figure 4.13: RAS solution plots. Top left: ground truth, top middle: RAS stationary solution after 10 iterations, top right: FGMRES solution with RAS preconditioner after 10 steps, bottom left: initial guess, bottom middle: error of RAS stationary solution (L_2 norm of error = 0.0992), bottom right: error of FGMRES with RAS preconditioner solution (L_2 norm of error = 0.0024).

4.6.7 Extra test problems

We have run tests with more PDEs, including discontinuous and anisotropic diffusion coefficients, as described below. 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 previous section, 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. The mathematical formalism of the extra test problem we explore are as below (for visualization of these problems, please refer to Chapter 3 section 3.5.3):

Jump-discontinuous diffusion:

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

Figure 4.14 shows the performance of our method against RAS, both for the stationary iterations and FGMRES.

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. Figure 4.15 shows the performance of our method against RAS, both for the stationary iterations and FGMRES.

Low-diffusion inclusion:

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

Figure 4.16 shows the performance of our method against RAS, both for the stationary iterations and FGMRES. Considering stationary iterations to convergence, we see improvements of 20-45% on average, and as is typical, the improvements are somewhat less large for preconditioned FGMRES iterations to convergence, but are still around 10% on average.

4.6.8 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. Figure 4.17 shows 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.

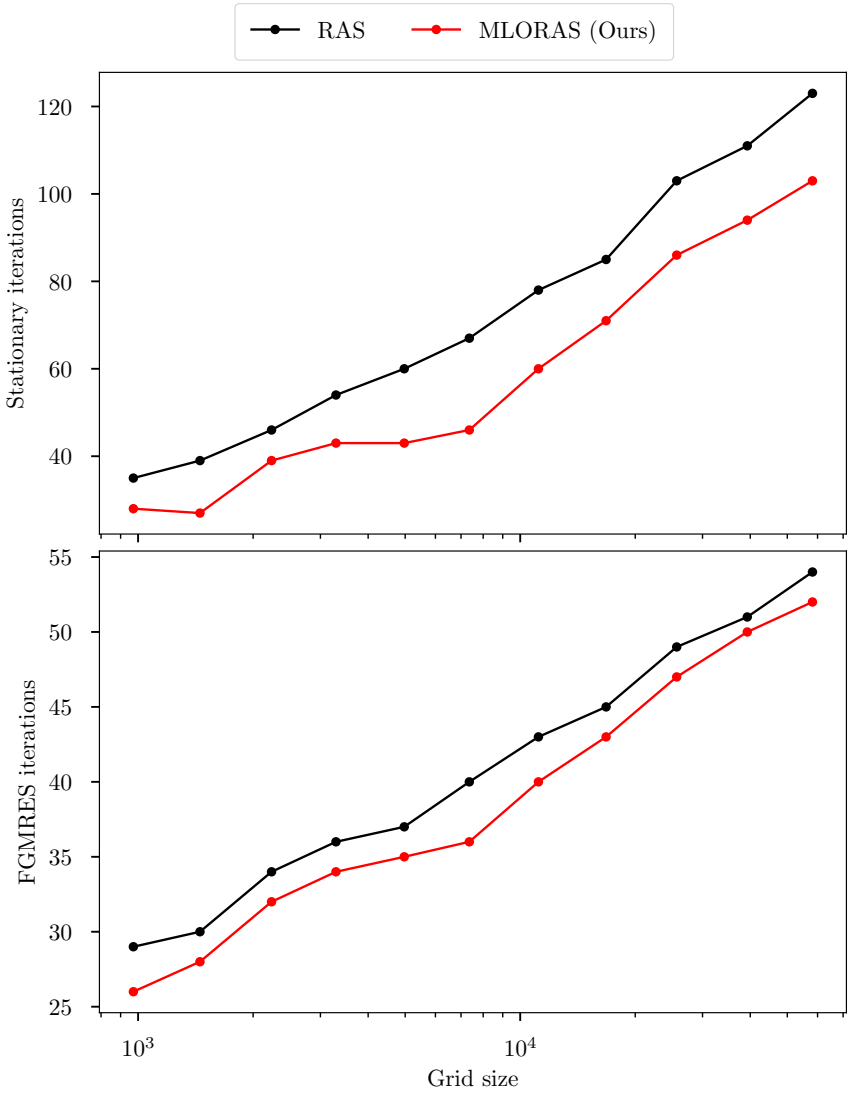

Figure 4.14: Jump-discontinuous diffusion coefficient for Poisson problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

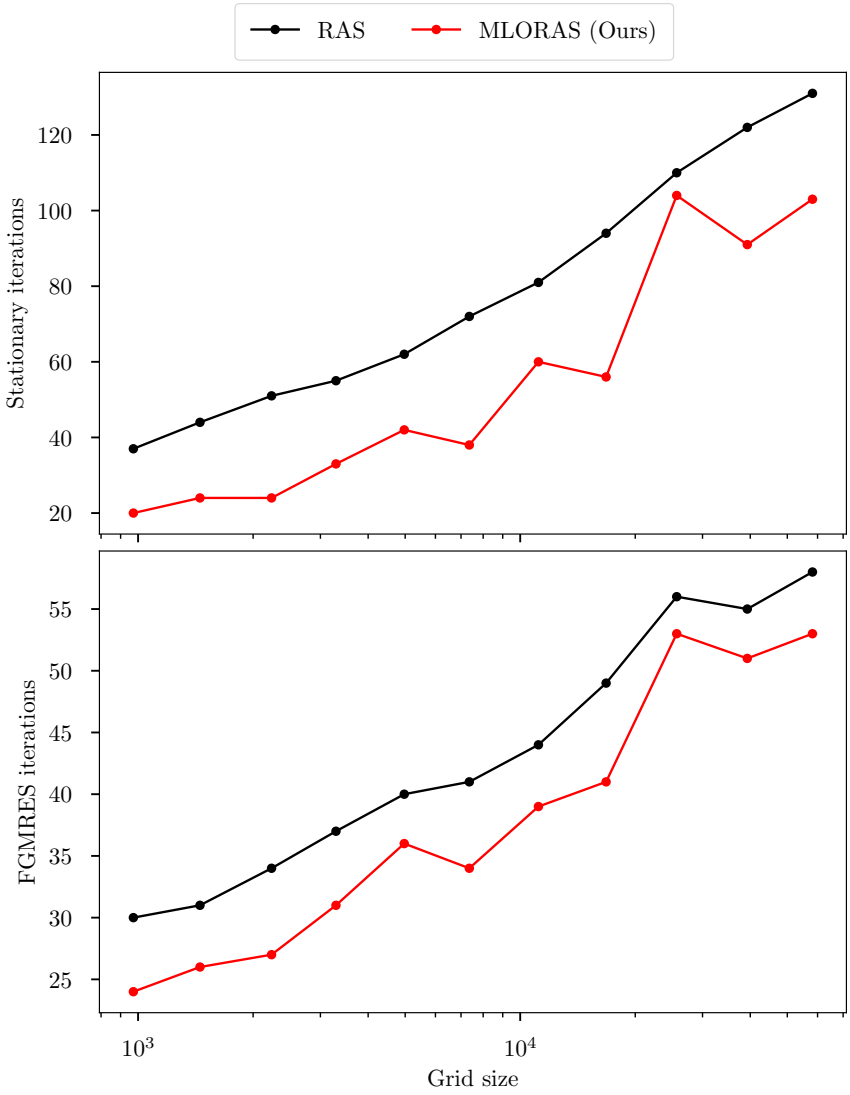

Figure 4.15: Rotated anisotropic diffusion problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

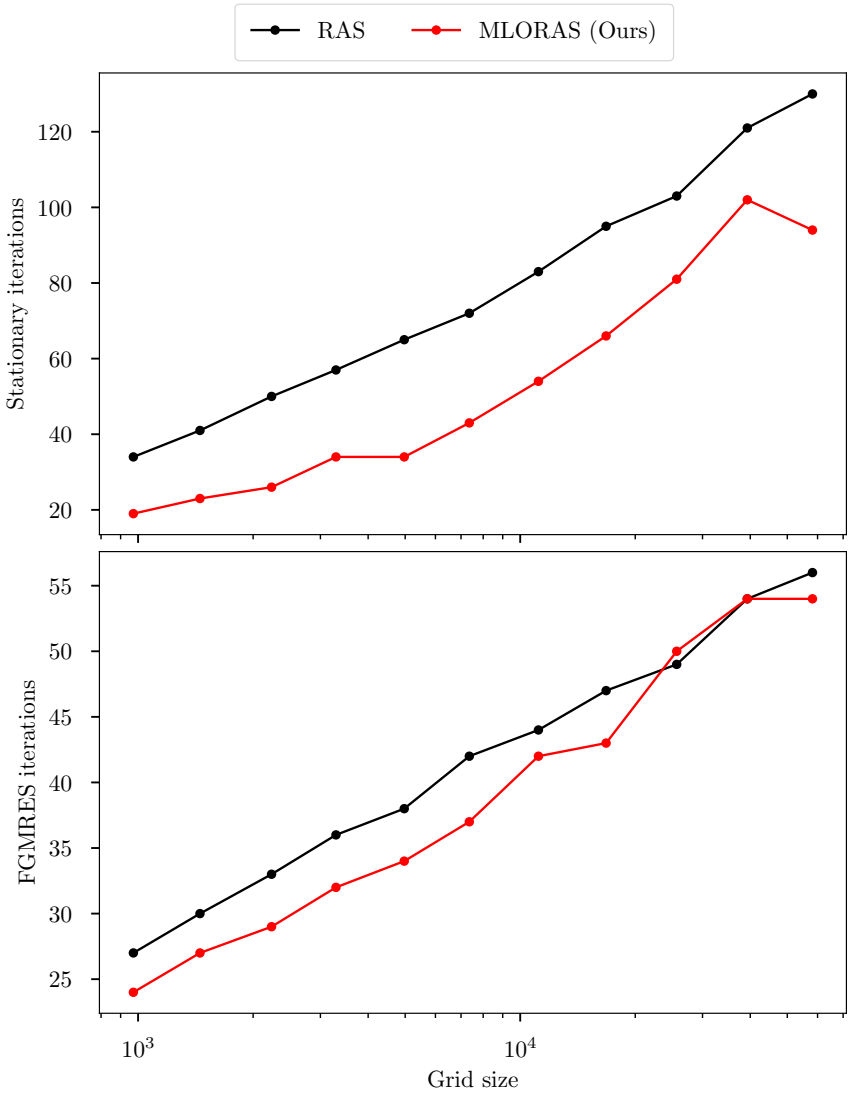

Figure 4.16: Low diffusion problem on various size grids. Up: error reduction by the stationary iteration after 10 iteration. Down: the number of preconditioned FGMRES steps required to solve the problem to within a relative error of 10^{-12}.

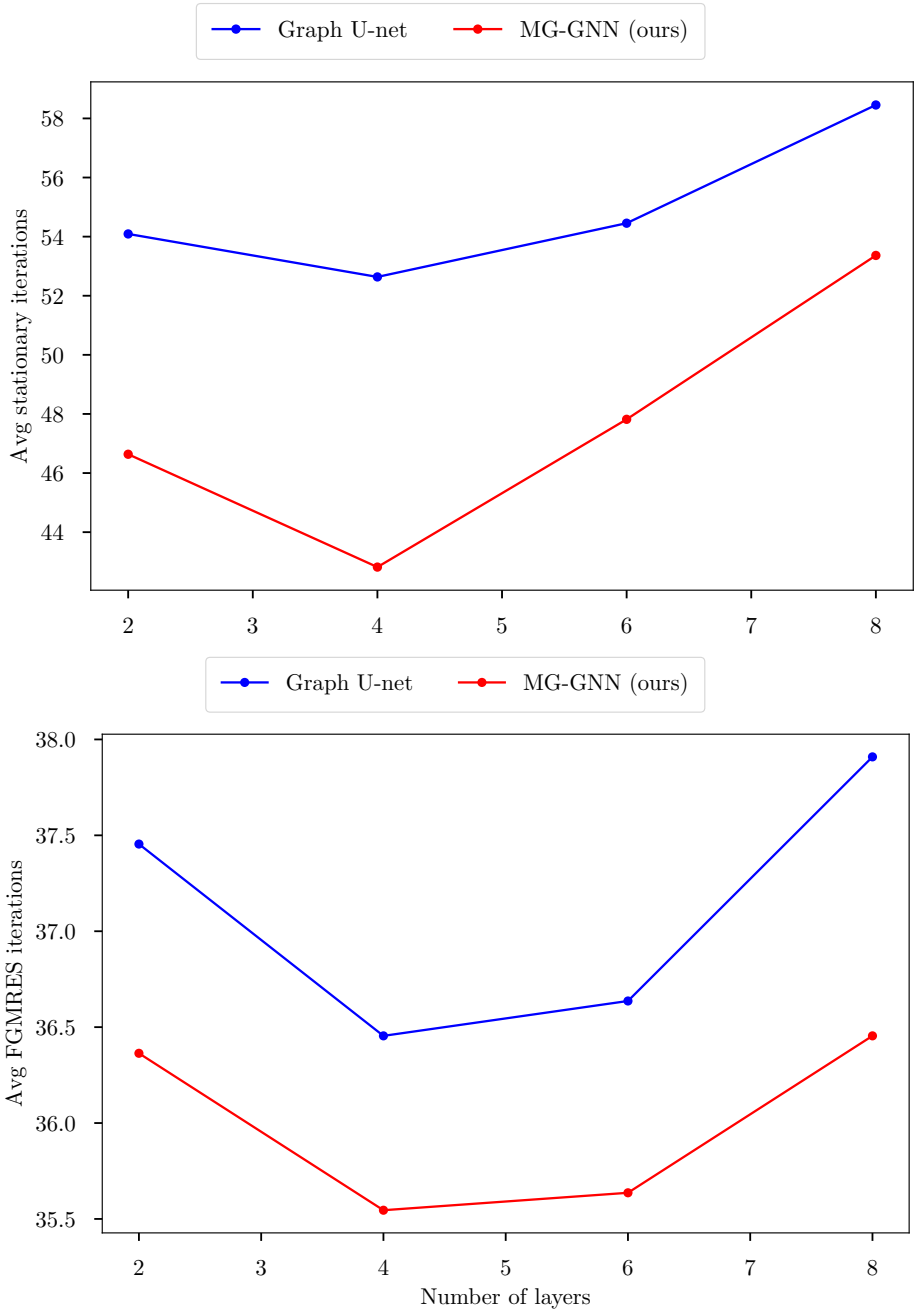

Figure 4.17: Average stationary and FGMRES iterations of graph U-net and MG-GNN with different number of layers on the test set.

Chapter 5

Hierarchical Graph Neural Network with Cross-Attention for Cross-Device User Matching

Portions of this chapter appear in the paper "Hierarchical Graph Neural Network with Cross-Attention for Cross-Device User Matching" accepted at International Conference on Big Data Analytics and Knowledge Discovery (DaWaK 2023) [79]

5.1 Introduction

Matching users across multiple devices is a crucial challenge in various domains such as advertising, recommender systems, and cybersecurity. The task involves the identification and linking of different devices belonging to the same individual by analyzing their sequential logs. Previous approaches in data mining have encountered difficulties in capturing long-range dependencies and higher-order connections present in these logs. Recently, researchers have approached this problem from a graph perspective and proposed a two-tier graph contextual embedding (TGCE) neural network architecture, which has demonstrated superior performance compared to earlier methods. In this study, we present a novel hierarchical graph neural network architecture (HGNN) that offers a more computationally efficient second-level design as compared to TGCE. Moreover, we introduce a cross-attention (Cross-Att) mechanism into our model, leading to a 5% performance improvement compared to the state-of-the-art TGCE approach.

The cross-device user matching task was first introduced in the CIKM Cup 2016[1], and the first proposed methods for the task mainly considered hand-crafted features. For instance, the runner-up solution [26] produces sub-categories based on the most significant URLs to generate detailed features. Furthermore, the competition winner solution proposed by Tay *et. al.* [80] utilizes "term frequency inverse document frequency" (TD-IDF) features of URLs and other related URL visit time features. However, their manually designed features did not fully investigate more intricate semantic details, such as the order of behavior sequences, which restricted their effectiveness. Aside from the hand-crafted features, the features that are developed from the structural information of the device URL visit data are also crucial for accomplishing the task of

[1] http://cikm2016.cs.iupui.edu/cikm-cup/

user matching. To further process sequential log information, studies have applied LSTM, 2D-CNN, and Doc2vec to generate semantic features for a sequence visited by a device [27], [28], [31].

Sequence-based machine learning models have also been employed for different entity resolution tasks; for instance, recurrent neural networks (RNN) have been utilized to encode behavior item sequential information [32]. Nevertheless, long-range dependencies and more advanced sequence features are not well obtained using sequence models [29]. With the advent of graph neural networks (GNN), researchers have modeled device logs as individual graphs where nodes and edges represent visited URLs and transitions between URLs. Each node and/or edge has an initial feature vector obtained from the underlying problem, and the layers of GNNs are then employed to update these features based on information passing in the local neighborhood of every node, such as the SR-GNN paper [81]. Another example is the LESSR [82] method for recommendation systems where the method is capable of long-range information capturing using an edge-order preserving architecture. However, these methods are specifically designed for the recommendation task and do not necessarily achieve desirable results on the cross-device user matching task.

Recently, researchers have proposed TGCE [29], a two-tier GNN for the cross-device user matching task. In the first tier, for every device log, each URL is considered as a node, and directional edges denote transitions between URLs. In the second tier, shortcut edges are formed by starting a random walk from every node and connecting all of the visited nodes to it. After a round of message passing in the first tier, the second tier is supposed to facilitate long-range information sharing in the device log. After the second tier, a position-aware graph attention layer is applied, followed by an attention pooling, which outputs the learned embedding for the whole graph. For the final pairwise classification, these learned embeddings for each of the devices are multiplied in an entry-wise manner and are sent to a fully connected deep neural network to determine whether they belong to the same user.

5.2 Hierarchical graph neural network

In this section, we discuss how we employ a two-level heterogeneous graph neural network for the cross-device user matching problem.

5.2.1 Problem definition

The aim of the cross-device user matching problem is to determine whether two devices belong to the same user, given only the URL visits of each device. Denote a sequence of visited URLs by a device v by $\mathcal{S}_v = \{s_1, s_2, ..., s_n\}$, where s_i denotes the i'th URL visit by the device (note that s_i's are not necessarily different). We build a hierarchical heterogeneous graph, G_v, based on the sequence $\mathcal{S}_v$ as follows: for a visited URL, s_i, consider a *fine* node in G_v and denote it by f_i. Note that if multiple s_i's correspond to the same URL, we only consider one node for it in G_v. Then, we connect nodes corresponding to consecutively visited URLs by directed edges in the graph; we connect f_i and f_{i+1} by a directional edge (if f_i and f_{i+1} correspond to the same URL, the edge becomes a self-loop). Up to this point, we have defined the fine-level graph, and we are ready to construct the second level, which we call the *coarse* level.

To construct the second level, we partition the sequence $\mathcal{S}_v$ into non-overlapping subgroups of K URLs, where each subgroup consists of consecutively visited URLs (the last subgroup may have less than K URLs). For every subgroup j, we consider a coarse node, c_j, and connect it to all of the fine nodes corresponding to the URLs in subgroup j via undirected edges.

5.2.2 Fine level

In the fine level of the graph G_v, for every node f_i, we order the nodes corresponding to the URLs that have an incoming edge to f_i according to their position in $\mathcal{S}_v$. We denote this ordered sequence of nodes by $N_i = \{f_{j_1}, f_{j_2}, ..., f_{j_\kappa}\}$. Also, we denote the feature vector of the fine node f_i by x_i. The l-th round of message passing in the fine-level graph updates the node features according to the following update methods:

$$M_i^{(l)} = \Phi^{(l)}([x_{j_1}, x_{j_2}, ..., x_{j_\kappa}, x_i]), \tag{5.1}$$

$$x_i^{(l+1)} = \Psi^{(l)}(x_i^{(l)}, M_i^{(l)}), \tag{5.2}$$

where $\Phi^{(l)}$ is a sequence aggregation function (such as sum, max, GRU, LSTM, etc.), for which we use GRU [83], and $\Psi^{(l)}$ is a function for updating the feature vector (e.g., a neural network), for which we use a simple mean.

5.2.3 Coarse level

In every round of heterogeneous message passing between fine and coarse level nodes, we update both the fine and coarse node features. Consider the coarse node c_j, and denote its feature by $\tilde{x}_j$. Also, denote the fine neighbor nodes of c_j by $\tilde{\mathcal{N}}(c_j)$. In the l-th layer of heterogeneous message passing, the coarse node feature update is as follows:

$$\tilde{x}_j^{(l+1)} = \underset{i \in \tilde{\mathcal{N}}(c_j)}{\square} (W_1^{(i)} x_i), \tag{5.3}$$

where $W_1^{(l)}$ is a learnable matrix and $\square$ is an aggregation function (such as mean, max, sum, etc.), for which we use mean. Denote by $\mathcal{N}(f_i)$ the set of coarse nodes connected to the fine node f_i. We first learn attention weights for the heterogeneous edges, and then we update fine nodes accordingly. In the l-th round of heterogeneous message passing, the fine node features are updated as follows:

$$e_{i,j}^{(l)} = \phi(W_2^{(l)} x_i^{(l)}, W_3^{(l)} \tilde{x}_j^{(l)}), \tag{5.4}$$

$$\alpha_{i,j}^{(l)} = \frac{\exp(e_{i,j}^{(l)})}{\sum_{j \in \mathcal{N}(f_i)} \exp(e_{i,j}^{(l)})}, \tag{5.5}$$

$$x_i^{(l+1)} = \xi(x_i^{(l)}, \sum_{j \in \mathcal{N}(f_i)} \alpha_{i,j}^{(l)} \tilde{x}_j^{(l)}), \tag{5.6}$$

where $W_2^{(l)}$ and $W_3^{(l)}$ are learnable matrices, and ξ and ϕ are update functions (such as a fully connected network). Figure 5.1 shows the overall architecture of fine and coarse level message passing.

5.2.4 Cross attention

After the message passing rounds in the fine level and long-range information sharing between fine and coarse nodes, we extract the learned fine node embeddings and proceed to cross encoding and feature filtering, inspired by the GraphER architecture [36]. We consider two different device logs v and w, and their learned fine node embeddings as a sequence, ignoring the underlying graph structure. We denote the learned fine node embeddings for device logs v and w as $X_v \in \mathbb{R}^{m_v \times d}$ and $X_w \in \mathbb{R}^{m_w \times d}$, where m_v and m_w are the number of nodes in the fine level of G_v and G_w, respectively. We learn two matrices for cross-encoding X_v into X_w and vice-versa. Consider the i-th and j-th rows of X_v and X_w, respectively, and denote them by $x_{v,i}$

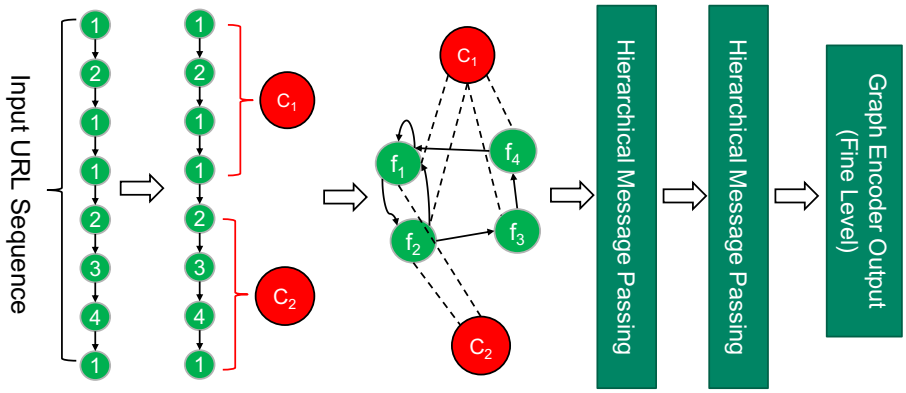

Figure 5.1: From left to right: heterogeneous (fine and coarse) graph modeling from a given URL sequence. The hierarchical message passing blocks consist of message passing on the fine nodes with a GRU aggregation function. Next, the coarse node features are updated using a mean aggregation function. Finally, the fine node features are updated using their previous feature vector as well as an aggregated message from their associated coarse nodes obtained via an attention mechanism between coarse and fine level nodes.

and $x_{w,j}$. The entries $\hat{\alpha}_{i,j}$ of the matrix $A_{v,w}$ for cross-encoding X_v into X_w are obtained using an attention mechanism (and similarly for $A_{w,v}$):

$$\hat{e}_{i,j} = \zeta(W_3 x_{v,i}, W_3 x_{w,j}), \tag{5.7}$$

$$\hat{\alpha}_{i,j} = \frac{\exp(\hat{e}_{i,j})}{\sum_{k=1}^{m_w} \exp(\hat{e}_{i,k})}, \tag{5.8}$$

where ζ is an update function (such as a neural network), for which we use a simple mean. After obtaining the cross-encoding weights, we apply feature filtering, a self-attention mechanism that filters important features. The filtering vector is obtained as $\beta_v = \text{sigmoid}(W_4 \text{tanh}(W_5 X_v^T))$, where W_4 and W_5 are learnable weights (β_w is obtained similarly). We apply the feature-filtering vector to the cross-encoding matrix as follows:

$$L_{v,w} = [\text{diag}(\beta_v)(A_{v,w}X_w - X_v)] \odot [\text{diag}(\beta_v)(A_{v,w}X_w - X_v)], \tag{5.9}$$

where $\odot$ denotes the Hadamard product ($L_{w,v}$ is also obtained similarly). The $L_{v,w} \in \mathbb{R}^{m_v \times d}$ and $L_{w,v} \in \mathbb{R}^{m_w \times d}$ matrices come from the Euclidean distance between the cross-encoding of X_v into X_w and X_w, and therefore are a measure of the closeness of the original sequence logs of v and w.

To obtain a size-independent comparison metric, we apply a multi-layer perceptron (MLP) along the feature dimension of L matrices (the second dimension, d), followed by a max-pooling operation along the first dimension. Finally, we apply a dropout and a ReLU nonlinearity. This yields vectors $r_{v,w}$ and $r_{w,v}$ that have a fixed size for any pair of v and w. For the final pairwise classification task, we concatenate $r_{v,w}$ and $r_{w,v}$ and pass it through an MLP followed by a sigmoid activation to determine if the two devices belong to the same user or not:

$$\hat{y} = \text{sigmoid}(\text{MLP}(r_{v,w} || r_{w,v})). \tag{5.10}$$

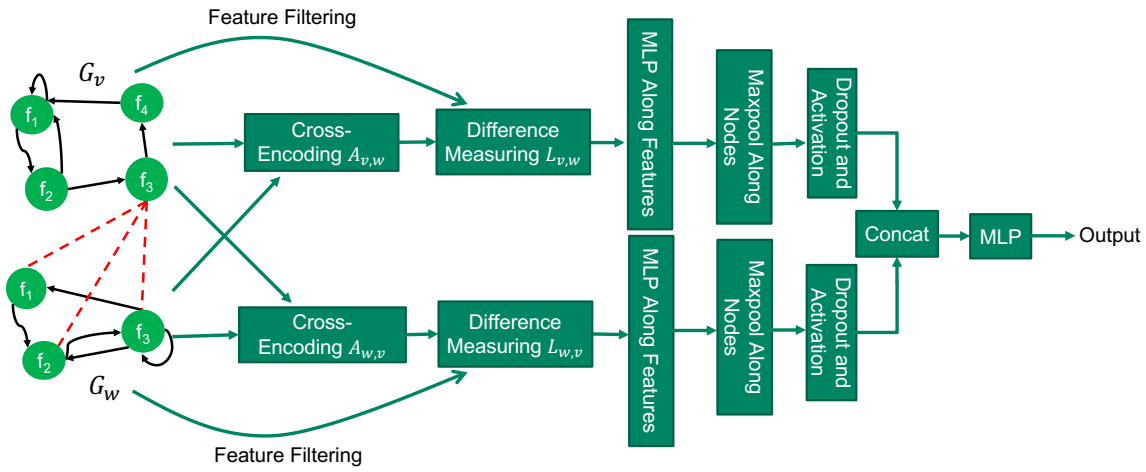

Figure 5.2: Pairwise device graph matching: After the message passing, the two device graphs are cross-encoded via an attention mechanism followed by an attention-based feature filtering. The resulting matrix for each graph is then passed through an MLP layer, acting along the feature, followed by a maxpool operator along the nodes. Next, the obtained vectors pass through a dropout layer followed by an activation function. Finally, the resulting vectors of the two graphs are concatenated and passed through an MLP to obtain the final output.

5.3 Experiment

In this section, we will describe the dataset, training details, and discuss how our method outperforms all other baselines, including TGCE [29], the previous state-of-the-art.

5.3.1 Training details

We studied the cross-device user matching dataset made publicly available by the Data Centric Alliance[2] for the CIKM Cup 2016 competition. The dataset consists of 14,148,535 anonymized URL logs of different devices with an average of 197 logs per device. The dataset is split into 50,146 and 48,122 training and test device logs, respectively. To obtain the initial embeddings of each URL, we applied the same data preprocessing methods as in [28], [29]. We used a coarse-to-fine node ratio of $k = 6$, a batch size of 800 pairs of device logs, a learning rate of 10^{-3}, and trained the model for 20 epochs. We used the binary cross-entropy (BCE) loss function for training our model. The training, evaluation, and test were all executed on an A100 NVIDIA GPU. The BCE loss during training as well as the validation F1 score are shown in Figures 5.3 and 5.4, respectively.

5.3.2 Results

In this section, we evaluate the precision, recall, and F1 score of our method on the test set and compare it to available baselines. All of the baselines have been obtained similarly as described in [29]. We present two variants of our method; the first one, which we label "HGNN", only differs from TGCE in the design of the second tier, i.e., we use the hierarchical structure presented in subsections 5.2.2 and 5.2.3, followed by the rest of the TGCE architecture. The second variant, which we label "HGNN+Cross-Att", uses the

[2]https://competitions.codalab.org/competitions/11171

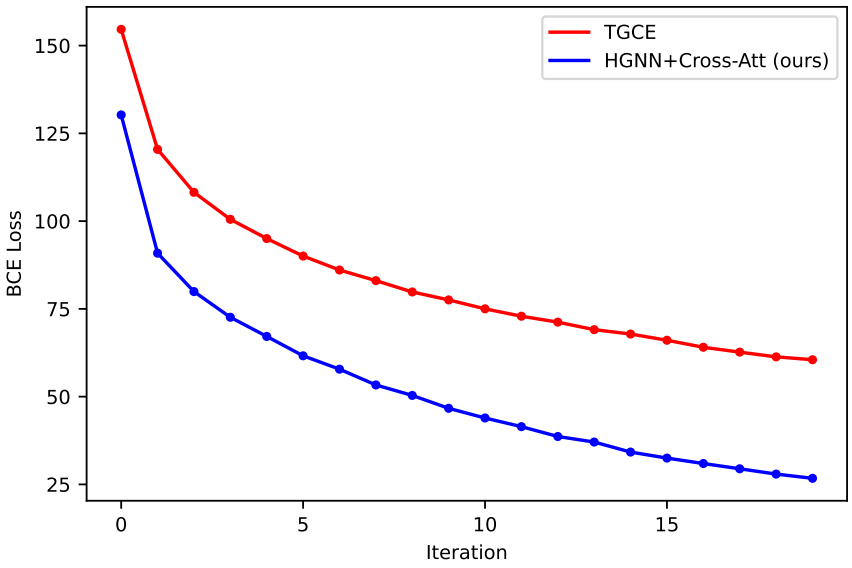

Figure 5.3: Binary cross-entropy loss of our proposed method against that of TGCE. During training, our method obtains strictly better loss values.

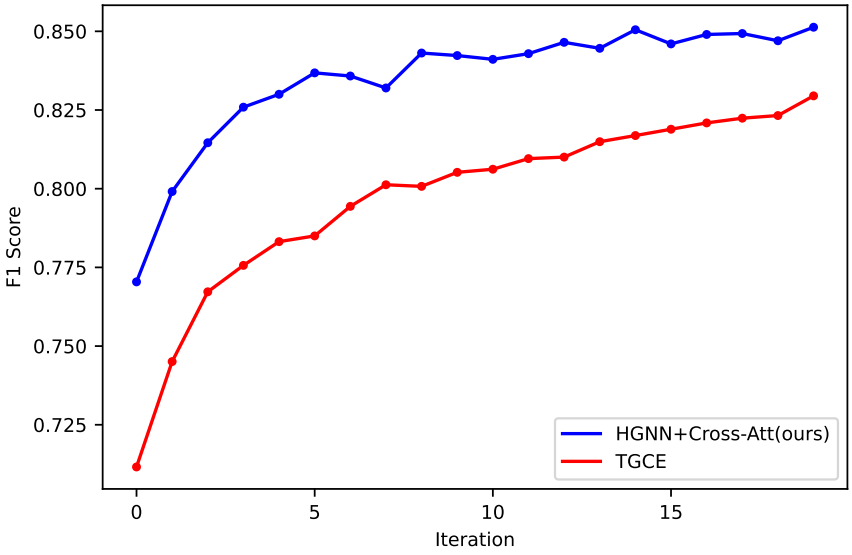

Figure 5.4: Validation F1 score during training. Throughout the training, our method achieves strictly better F1 scores for the validation set compared to that of TGCE.

Table 5.1: Precision, Recall, and F1 score of different methods for cross-device user matching on DCA dataset.

	Precision at Best F1 Score	**Recall at Best F1 Score**	**Best F1 Score**
TF-IDF	0.33	0.27	0.26
Doc2vec	0.29	0.21	0.24
SCEmNet	0.38	0.44	0.41
GRU	0.37	**0.49**	0.42
Transformer	0.39	0.47	0.43
SR-GNN	0.35	0.34	0.34
LESSER	0.41	**0.48**	0.44
TGCE	0.49	0.44	0.46
HGNN (ours)	0.48	0.43	0.45
HGNN+Cross-Att (ours)	**0.57**	**0.48**	**0.51**

Table 5.2: Best F1 score and end-to-end training time of HGNN (without Cross-Att), HGNN+Cross-Att, and TGCE. The HGNN model is 6x faster than TGCE with a slight trade-off (about 1%) on the accuracy side. The HGNN+Cross-Att model has the same training time as TGCE while achieving 5% better F1 score.

	Best F1 Score	**End-to-end Training Time**	**Number of Epochs**
TGCE	0.46	60h	20
HGNN (ours)	0.45	**10h**	20
HGNN+Cross-Att (ours)	**0.51**	60h	**6**

hierarchical structure in subsections 5.2.2 and 5.2.3, and also utilizes the cross-attention mechanism presented in subsection 5.2.4 after the hierarchical structure. As shown in Table 5.1, the "HGNN+Cross-Att" variant outperforms all of the baselines on the F1 score metric, including the second-best method (TGCE) by 5% on the test data.

We also compare the training time of the two variants of our method with that of TGCE. As shown in Table 5.2, our hierarchical structure is significantly more efficient than that of TGCE while keeping a competitive F1 score. Table 5.2 essentially indicates that by simply replacing the second-tier design of TGCE with our hierarchical structure (presented in subsections 5.2.2 and 5.2.3), the method becomes 6x faster while almost keeping the same performance. This is due to the large number of artificial edges generated in the random walk passes in the creation of the second tier of TGCE. Moreover, although including cross-attention slows down the model, we can still obtain the same training time as TGCE and achieve 5% better overall F1 score.

Figure 5.5 shows the precision-recall curve of our method (the HGNN+Cross-Att variant, trained for 6 epochs) with that of TGCE (trained for 20 epochs). As shown in the figure, the precision-recall curve of our method is strictly better than that of TGCE. In other words, for every recall score, our method has a better precision. Additionally, we further trained the HGNN+Cross-Att variant for 20 epochs (the same number of epochs TGCE was trained for) to study if any further improvement is achieved on the test set. We also plot the F1 score with different thresholds (from 0 to 1 incremented by 0.01) for our model trained for 6 and 20

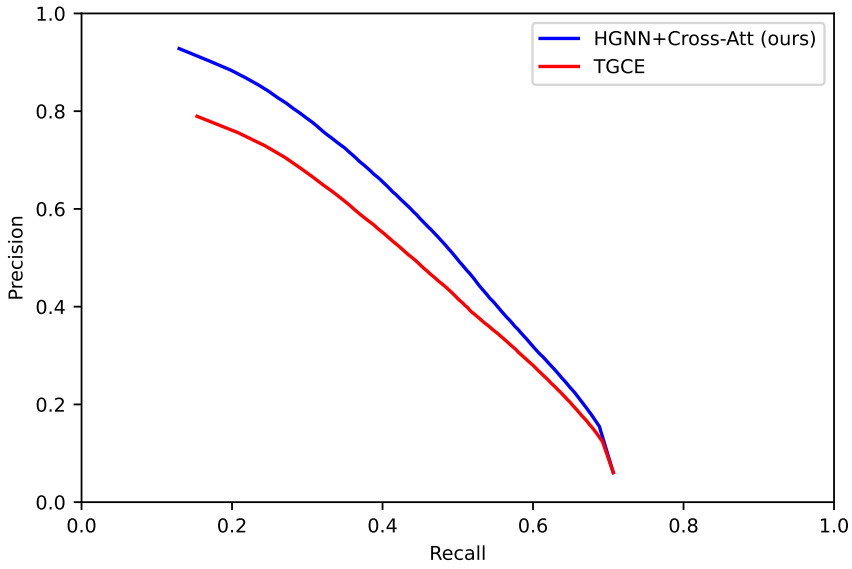

Figure 5.5: Precision-Recall curve of the proposed method and that of TGCE on the test data.

epochs and compare it to that of TGCE. As shown in Figure 5.6, our model trained for 20 epochs strictly outperforms TGCE (also trained for 20 epochs) for every threshold for obtaining the F1 score. However, our model trained for 6 epochs achieves the best overall F1 score, which is 5% higher than TGCE. This is significant since as shown in Table 5.2, the model takes the same time as TGCE to train.

Acknowledgements

The research in this chapter was supported by NVIDIA Corporation. A U.S. patent is to be published following this work.

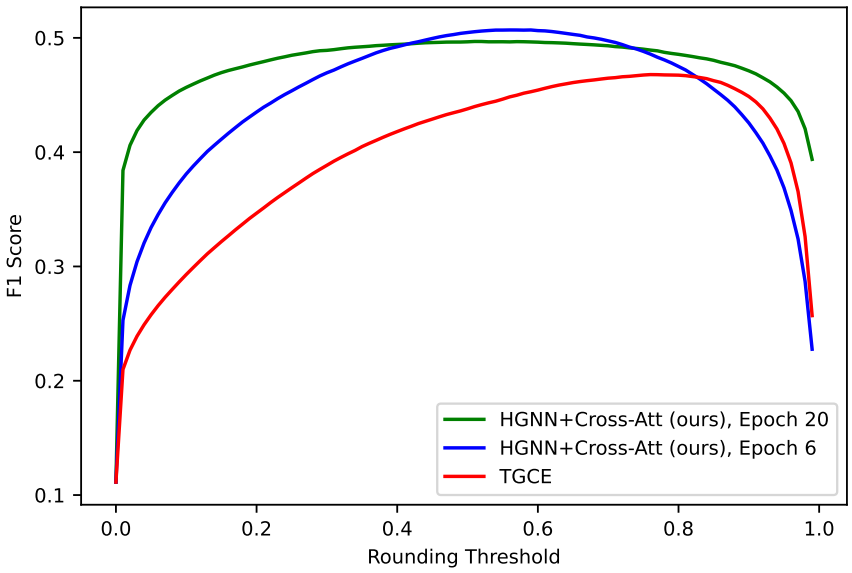

Figure 5.6: F1 score against rounding threshold for our method (both for networks trained for 6 and 20 epochs) compared to that of TGCE (trained for 20 epochs). When our model is trained for 6 epochs, it achieves the highest overall F1 score, which is about 5% better than that of TGCE. However, if our model is trained for 20 epochs, it achieves higher F1 scores for a wider range of rounding thresholds, outperforming TGCE for every rounding threshold.

Chapter 6

Conclusions

In the second chapter of this study, we propose an unsupervised learning method based on Graph Convolutional Neural Networks (GCNNs) for extending optimized restricted additive Schwarz (ORAS) methods to multiple subdomains and unstructured grid cases. Our method is trained with a novel loss function, stochastically minimizing the spectral radius of the error propagation operator obtained using the learned interface values. The time complexity of evaluating the loss function, as well as obtaining the interface values using our neural network are all linear in problem size. Moreover, the proposed method is able to outperform ORAS, both as a stationary algorithm and preconditioner for FGMRES on structured grids with two subdomains, as considered in the conventional ORAS literature. On more general cases, such as unstructured grids with arbitrary subdomains of bounded size, our method outperforms RAS consistently, both as a stationary algorithm and preconditioner for FGMRES.

In the third chapter, 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 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.

In the fourth chapter, we present a novel graph neural network (GNN) architecture for a demanding entity resolution task: cross-device user matching, which determines if two devices belong to the same user based only on their anonymized internet logs. Our method comprises of designing an effective hierarchical structure for achieving long-range message passing in the graph obtained from device URL logs. After passing device logs through such a hierarchical GNN, we employ a cross-attention mechanism to effectively compare device logs against each other to determine if they belong to the same user. We demonstrate that our method outperforms available baselines by at least 5%, while having the same training time as the previous state-of-the-art method, establishing the effectiveness of our proposed method.

Finally, we discuss a few next steps worth considering in the direction of this thesis. One of the main obstacles regarding training on grids larger than a few thousand nodes has been lack of support for automatic

differentiation for sparse matrix operation [12]. Recently, [78] has developed the software for mitigating this shortcoming. Therefore, as a next step, to fully leverage the benefits of the MG-GNN architecture, we are aiming to train the model on grids orders of magnitude larger than the current training set. Future research may explore whether such alteration would result in more powerful learned solvers, and if global error modes are better captured. Moreover, in a recent study [84], researchers have utilized convolutional neural networks (CNNs) to learn smoothers for AMG on structured grids. Using CNNs will limit the method only to structured grids. Future research may explore learning optimal smoothers for AMG using GNNs in order to generalize the method to unstructured meshes.

References

[1] A. Toselli and O. Widlund, *Domain decomposition methods—algorithms and theory*, ser. Springer Series in Computational Mathematics. Springer-Verlag, Berlin, 2005, vol. 34, pp. xvi+450, ISBN: 3-540-20696-5. DOI: 10.1007/b137868.

[2] A. Quarteroni and A. Valli, *Domain decomposition methods for partial differential equations*, ser. Numerical Mathematics and Scientific Computation. The Clarendon Press, Oxford University Press, New York, 1999, pp. xvi+360, Oxford Science Publications, ISBN: 0-19-850178-1.

[3] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2015, pp. x+238, ISBN: 978-1-611974-05-8. DOI: 10.1137/1.9781611974065.ch1.

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

[5] M. J. Gander and F. Kwok, "Optimal interface conditions for an arbitrary decomposition into subdomains," in *Domain decomposition methods in science and engineering XIX*, ser. Lect. Notes Comput. Sci. Eng. Vol. 78, Springer, Heidelberg, 2011, pp. 101–108. DOI: 10.1007/978-3-642-11304-8_9.

[6] X.-C. Cai and M. Sarkis, "A restricted additive Schwarz preconditioner for general sparse linear systems," *Siam Journal on Scientific Computing*, vol. 21, no. 2, pp. 792–797, 1999.

[7] A. St-Cyr, M. J. Gander, and S. J. Thomas, "Optimized multiplicative, additive, and restricted additive Schwarz preconditioning," *SIAM Journal on Scientific Computing*, vol. 29, no. 6, pp. 2402–2425, 2007.

[8] F. Magoulès, D. B. Szyld, and C. Venet, "Asynchronous optimized Schwarz methods with and without overlap," *Numer. Math.*, vol. 137, no. 1, pp. 199–227, 2017, ISSN: 0029-599X. DOI: 10.1007/s00211-017-0872-z.

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

[10] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

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

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

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

[14] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber, "Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review," *GAMM-Mitteilungen*, vol. 44, no. 1, e202100001, 2021.

[15] ——, "Machine learning in adaptive domain decomposition methods—predicting the geometric location of constraints," *SIAM Journal on Scientific Computing*, vol. 41, no. 6, A3887–A3912, 2019.

[16] K. Li, K. Tang, T. Wu, and Q. Liao, "D3M: A deep domain decomposition method for partial differential equations," *IEEE Access*, vol. 8, pp. 5283–5294, 2019.

[17] W. Li, X. Xiang, and Y. Xu, "Deep domain decomposition method: Elliptic problems," in *Mathematical and Scientific Machine Learning*, PMLR, 2020, pp. 269–286.

[18] V. Mercier, S. Gratton, and P. Boudier, "A coarse space acceleration of deep-DDM," *arXiv preprint arXiv:2112.03732*, 2021.

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

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

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

[22] H. Gao and S. Ji, "Graph u-nets," in *ICML*, PMLR, 2019, pp. 2083–2092.

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

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

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

[26] J. Lian and X. Xie, "Cross-device user matching based on massive browse logs: The runner-up solution for the 2016 cikm cup," *arXiv preprint arXiv:1610.03928*, 2016.

[27] M. C. Phan, Y. Tay, and T.-A. N. Pham, "Cross device matching for online advertising with neural feature ensembles: First place solution at cikm cup 2016," *arXiv preprint arXiv:1610.07119*, 2016.

[28] M. C. Phan, A. Sun, and Y. Tay, "Cross-device user linking: Url, session, visiting time, and device-log embedding," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017, pp. 933–936.

[29] H. Huang, S. Guo, C. Li, J. Sheng, L. Wang, J. Li, J. Liu, and S. Zhong, "Two-tier graph contextual embedding for cross-device user matching," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 730–739.

[30] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, PMLR, 2014, pp. 1188–1196.

[31] U. Tanielian, A.-M. Tousch, and F. Vasile, "Siamese cookie embedding networks for cross-device user matching," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 85–86.

[32] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," *arXiv preprint arXiv:1511.06939*, 2015.

[33] R. Qiu, H. Yin, Z. Huang, and T. Chen, "Gag: Global attributed graph neural network for streaming session-based recommendation," in *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, 2020, pp. 669–678.

[34] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang, "Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1441–1450.

[35] Z. Pan, F. Cai, W. Chen, H. Chen, and M. De Rijke, "Star graph neural networks for session-based recommendation," in *Proceedings of the 29th ACM international conference on information & knowledge management*, 2020, pp. 1195–1204.

[36] B. Li, W. Wang, Y. Sun, L. Zhang, M. A. Ali, and Y. Wang, "Grapher: Token-centric entity resolution with graph convolutional neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8172–8179.

[37] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*. SIAM, 2015.

[38] B. F. Smith, *Domain decomposition methods for partial differential equations*. Springer, 1997.

[39] A. Toselli and O. Widlund, *Domain decomposition methods-algorithms and theory*. Springer Science & Business Media, 2004, vol. 34.

[40] H. A. Schwarz, "Ueber einige Abbildungsaufgaben.," 1869.

[41] V. G. Korneev and U. Langer, *Dirichlet-Dirichlet domain decomposition methods for elliptic problems: h and hp finite element discretizations*. World Scientific, 2015.

[42] M. Dryja and O. B. Widlund, "Schwarz methods of Neumann-Neumann type for three-dimensional elliptic finite element problems," *Communications on pure and applied mathematics*, vol. 48, no. 2, pp. 121–155, 1995.

[43] J. Côté, M. J. Gander, L. Laayouni, and S. Loisel, "Comparison of the Dirichlet-Neumann and optimal Schwarz method on the sphere," in *Domain decomposition methods in science and engineering*, Springer, 2005, pp. 235–242.

[44] M. J. Gander, "Optimized Schwarz methods," *SIAM Journal on Numerical Analysis*, vol. 44, no. 2, pp. 699–731, 2006.

[45] F. Nataf, "Recent developments on optimized Schwarz methods," *Domain Decomposition Methods in Science and Engineering XVI*, pp. 115–125, 2007.

[46] X. Chen, M. J. Gander, and Y. Xu, "Optimized Schwarz methods with elliptical domain decompositions," *Journal of Scientific Computing*, vol. 86, no. 2, pp. 1–28, 2021.

[47] T. F. Chan and J. Zou, "Additive Schwarz domain decomposition methods for elliptic problems on unstructured meshes," *Numerical Algorithms*, vol. 8, no. 2, pp. 329–346, 1994.

[48] M. Griebel and P. Oswald, "On the abstract theory of additive and multiplicative Schwarz algorithms," *Numerische Mathematik*, vol. 70, no. 2, pp. 163–180, 1995.

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

[50] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[51] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[52] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *arXiv preprint arXiv:1606.09375*, 2016.

[53] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[54] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*, PMLR, 2017, pp. 1263–1272.

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

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

[57] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[58] W. N. Bell, "Algebraic multigrid for discrete differential forms," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2008.

[59] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms (Second edition)*. MIT Press, 2001.

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

[61] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[62] G. E. Box, "A note on the generation of random normal deviates," *Ann. Math. Statist.*, vol. 29, pp. 610–611, 1958.

[63] A. Katrutsa, T. Daulbaev, and I. Oseledets, "Black-box learning of multigrid parameters," *Journal of Computational and Applied Mathematics*, vol. 368, p. 112 524, 2020, ISSN: 0377-0427. DOI: https://doi.org/10.1016/j.cam.2019.112524.

[64] A. Loukas, "What graph neural networks cannot learn: Depth vs width," *arXiv preprint arXiv:1907.03199*, 2019.

[65] N. Schlömer, *pygmsh: A Python frontend for Gmsh*, (GPL-3.0 License), 2021. DOI: 10.5281/zenodo.1173105. [Online]. Available: https://github.com/nschloe/pygmsh.

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

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

[68] N. Bell, L. N. Olson, and J. Schroder, "PyAMG: Algebraic multigrid solvers in python," *Journal of Open Source Software*, vol. 7, no. 72, p. 4142, 2022, (code is MIT licensed). DOI: 10.21105/joss.04142. [Online]. Available: https://doi.org/10.21105/joss.04142.

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

[70] L. Zhao and L. Akoglu, "Pairnorm: Tackling oversmoothing in GNNs," *arXiv preprint arXiv:1909.12223*, 2019.

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

[72] A. Taghibakhshi, N. Nytko, T. U. Zaman, S. MacLachlan, L. Olson, and M. West, "Mg-gnn: Multigrid graph neural networks for learning multilevel domain decomposition methods," *arXiv preprint arXiv:2301.11378*, 2023.

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

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

[75] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *ICML*, PMLR, 2019, pp. 3734–3743.

[76] J. Xu, "Iterative methods by space decomposition and subspace correction," *SIAM Review*, vol. 34, no. 4, pp. 581–613, 1992.

[77] W. L. Wan, T. F. Chan, and B. Smith, "An energy-minimizing interpolation for robust multigrid methods," *SIAM Journal on Scientific Computing*, vol. 21, no. 4, pp. 1632–1649, 1999.

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

[79] A. Taghibakhshi, M. Ma, A. Aithal, O. Yilmaz, H. Maron, and M. West, "Hierarchical graph neural network with cross-attention for cross-device user matching," *arXiv preprint arXiv:2304.03215*, 2023.

[80] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, Citeseer, vol. 242, 2003, pp. 29–48.

[81] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 346–353.

[82] T. Chen and R. C.-W. Wong, "Handling information loss of graph neural networks for session-based recommendation," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1172–1180.

[83] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[84] R. Huang, R. Li, and Y. Xi, "Learning optimal multigrid smoothers via neural networks," *SIAM Journal on Scientific Computing*, no. 0, S199–S225, 2022.