



University of Bristol  
Faculty of Engineering  
Department of Computer Science

---

# Accelerating Algebraic Multigrid Using Machine Learning

Application to problems originating from fluid simulation

---

by

*Roussel Desmond Nzoyem*

A dissertation submitted to the University of Bristol in accordance with the requirements of the Summer Project unit within the Interactive Artificial Intelligence CDT.

August 2022

---

**Supervisor:**

Prof. **Simon McIntosh-Smith**, Ph.D.  
Department of Computer Science  
Faculty of Engineering  
University of Bristol  
Woodland Road  
Bristol, BS8 1UB  
United Kingdom

**Co-Supervisor:**

Dr. **Tom Deakin**, Ph.D.  
Department of Computer Science  
Faculty of Engineering  
University of Bristol  
Woodland Road  
Bristol, BS8 1UB  
United Kingdom

**Co-Supervisor:**

Mr. **Thorben Louw**, MSc  
Lead Consultant  
Equal Experts  
Farringdon Rd  
London, EC1M 3JU  
United Kingdom

---

# Abstract

High-fidelity fluid simulations require solving extremely large and sparse linear systems, often involving millions of unknowns. With its fast convergence properties, Algebraic Multigrid (AMG) is the method of choice in several application areas. This said, classical AMG suffers from a number of issues, calling Machine Learning (ML) to the rescue. Recent years have seen a flourish of ML tactics to accelerate AMG. However, published work tends to focus on small and unrepresentative problems. Moreover, these methods tend to be developed dissociatively, not involving human end users in the process.

With insights from potential users, we aim to build a ML-augmented AMG framework that is: (i) *robust* i.e. reusable in a variety of problems arising from fluid simulation; (ii) computationally *cheap* i.e. converges faster than the classical AMG; (iii) works for structured as well as *unstructured* grids; (iv) works for small systems as well as *large* systems, while taking advantage of their *sparsity*; and (v) *scalable* i.e. data- and model-parallelisable with efficient memory management.

The main contributions of this project are as follows: (1) a Graph Neural Network methodology for learning prolongation operators, built around DGL and PyTorch; (2) a Cloud Formation stack on AWS to run experiments, containing all the dependencies our implementation requires; (3) experiments indicating that our current AI model is better than the classical approach in about 20% of cases; and (4) a survey that, among other remarks, endeared participants to using (AI-enabled) AMG for their various problems.

Our work entails much efficient usage of HPC resources. By noticing that ML can be leveraged not only when solving linear systems, we pave the way for ML to be efficiently used in other parts of the high-fidelity simulation pipeline. Moreover, the consideration of users and their needs is a positive step towards broader Design Space Exploration of prolongation operators, model hyperparameters, and physical parameters influencing mechanical engines in operation.

**Keywords:**

HPC-AI, AMG, GNN, Physics-Informed ML, Unstructured grid.



---

# Acknowledgements

First of all, I would like to express my gratitude to my thesis supervisors, Prof. Simon McIntosh-Smith, Dr. Tom Deakin, Mr. Thorben Louw. They have been a constant source of encouragement and insight during this research, helping me with numerous problems. From the start, they helped review my original synopsis, suggesting new avenues and proposing to abandon others (those would have never been achievable in hindsight).

Special thanks go to the management of the Interactive AI CDT, who maintained a pleasant and flexible environment for this research. Not only did they provide funding for the research, but they also provided credits through the AWS Public Sector Cloud Credit Initiative for Research Programs, allowing us to run various experiments on Amazon Web Services.

I would like to express my thanks to my colleagues from the Interactive AI CDT Mauro Comi, Matt Clifford, Davide Turco, and Tony Fang for their support, ideas, and for their valuable comments throughout the project.

Thank you to the University of Bristol's Research Ethics Committee at the Faculty of Engineering for expediting my application to launch a survey. Particularly, I am grateful to Miss Ingrid Hoxha whose quick actions allowed me to launch and analyse the survey in time.

Also, I am grateful to all the participants of our survey who gave precious minutes of their time, despite there being no prize money at the end. Their detailed responses were valuable to say the least.

Finally, my greatest thanks go to my family members, for their infinite patience during the past three months.

---

# Contents

<b>Abbreviations</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Problem statement . . . . .	2
1.3 Related work . . . . .	3
1.4 Contributions of this thesis . . . . .	3
1.5 Structure of the thesis . . . . .	3
<b>2 Background and state-of-the-art</b>	<b>5</b>
2.1 Background on linear solvers . . . . .	5
2.1.1 Direct methods . . . . .	5
2.1.2 Iterative methods . . . . .	6
2.1.3 Multigrid methods . . . . .	9
2.2 Previous results and related work . . . . .	14
2.2.1 Greenfeld et al. . . . .	14
2.2.2 Luz et al. . . . .	15
2.2.3 Louw and McIntosh-Smith . . . . .	15
2.3 Background on interactive design . . . . .	16
<b>3 Overview of our approach</b>	<b>19</b>
3.1 Unsupervised learning problem . . . . .	19
3.1.1 Loss functions . . . . .	20
3.1.2 Input and output data . . . . .	21
3.2 Graph Neural Networks . . . . .	22
3.2.1 Background on GNNs . . . . .	22
3.2.2 Graph operations in our problem . . . . .	23
3.2.3 Deep Graph Library . . . . .	24
3.3 Training strategy and resources . . . . .	25
3.3.1 Block-diagonalisation . . . . .	26
3.3.2 Amazon Web Services . . . . .	27
3.4 Method behind the survey . . . . .	27

---

<b>4</b>	<b>Results, discussion, and challenges</b>	<b>31</b>
4.1	Performance of AI-enabled AMG . . . . .	31
4.1.1	Description of the experiment . . . . .	31
4.1.2	Results and discussion . . . . .	32
4.2	Insights from the user survey . . . . .	33
4.2.1	Awareness of AMG . . . . .	33
4.2.2	Features for successful AMG . . . . .	34
4.2.3	Adoption of AI-enabled solvers . . . . .	35
4.3	Challenges . . . . .	36
4.3.1	Making use of edge features in GNNs . . . . .	36
4.3.2	Minimizing the spectral radius . . . . .	37
4.3.3	Testing on large matrices . . . . .	37
4.3.4	Other challenges . . . . .	38
<b>5</b>	<b>Conclusion</b>	<b>39</b>
5.1	Summary . . . . .	39
5.2	Future Work . . . . .	40
	<b>Bibliography</b>	<b>41</b>

---

## List of Figures

1.1	Comparison of serial performance (single CPU clock frequency and threads) against parallel computing (number of CPU cores) from 1970 to 2018. The parallel computing era begins about 2005, when the core count in CPU chips begins to rise, while the clock frequency and power consumption plateaus. Image reprinted from [1]. . . . .	2
2.1	Restriction of a smooth quantity from a fine grid $\Omega_f$ to a coarse grid $\Omega_c$ resulting in a good approximation (a), compared to the restriction of an oscillatory quantity resulting in a bad approximation (b). Image adapted from [2]. . . . .	10
2.2	Illustration of GMG in a multi-level 2D scenario. In each V-cycle, the solution is smoothed, and a residual is computed and propagated to the coarser grid. At the coarsest level, a direct solver (or another smoother) is applied, and the solutions are then iteratively interpolated to finer grids. Image adapted from [nVidia]. . . . .	12
2.3	Inputs and outputs of the network in Greenfeld et al. [3]. The discs denote the (fine) grid points, where the black discs mark the subset of points selected as coarse grid points. The input of the network consists of the $3 \times 3$ stencils of the five points, denoted by the red cycles. The black arrows illustrate the output of the network, i.e., the contribution of the prolongation of one coarse point to its eight fine grid neighbours. Image reprinted from [3]. . . . .	15
2.4	An element of the Thingi10K dataset created by Louw and McIntosh-Smith [4]. Using the geometry from (a), a FEM fluid simulation is carried out to obtain the large and sparse matrix visualized in (b). Images reprinted from [4]. . . . .	16
3.1	Illustration of our unsupervised learning process. A particular attention was placed on the pre- and post-processing steps, which will be covered throughout this report. But mainly, the loss functions and the GNN model will be showcased in the sequel. .	20
3.2	A sparse matrix $A$ and its associated graph $\mathcal{G}_A$ . The sparsity pattern defines the set of C-nodes 1; 4; 6, and to which nodes each C-node contributes (denoted by the red cells). In the graph in lower right, red edges represent the set $E^{sp}$ corresponding to the sparsity pattern. The prolongation matrix $P$ is formed by assigning weights to the sparsity pattern. Image adapted from [5]. . . . .	22
3.3	2D Euclidean convolution (a) and graph convolution (b). Graph convolutions only consider immediate neighbours. When attempting to increase this receptive field, the over-smoothing issue becomes apparent. Image reprinted from [6]. . . . .	23



3.4	Encode-Process-Decode GNN architecture implemented. Two separate MLPs are used for encoding edge and node features, while only one is used for decoding. During our project, we developed two Process architectures. In the first, all three Graph Convs layers were GraphSAGE convolutions. In the second architecture, the first layer was a GraphSAGE, the second was a NNConv, and the third was omitted. Image adapted from [4]. . . . .	24
3.5	Decrease in loss function during training ( <code>loss</code> ) and validation ( <code>eval_loss</code> ) for one of our held models (named 89621, with 3 GraphSAGE convolutions). Other than the loss function, several other quantities were measured and visualized with TensorBoard.	26
3.6	Visualization of our AWS Cloud Formation stack. MATLAB access is provided through a UoB academic licence. The GPU powering the EC2 instance is one of <code>g4dn.xlarge</code> , an NVIDIA T4. Image obtained through AWS's CCloud Formation template designer.	27
3.7	Screenshot of the introductory section of our survey. In total, it contained 5 sections, of which 4 are not displayed here. The survey can be taken at this link. . . . .	29
4.1	Circular domain with a square hole used for testing the generalization of our method on a diffusion problem using FEM. Image reprinted from [5]. . . . .	32
4.2	Descriptive and thematic results about awareness of linear solvers. . . . .	34
4.3	Word cloud for the responses to "What other features do you consider when choosing a solver?". . . . .	35
4.4	Histogram for the responses to "What part of the simulation pipeline would you use AI in?". . . . .	35
4.5	Full GN block as described by Battaglia et al. [7]. It defines parametrized functions for updating vertices $V$ , edges $E$ , and the global node $\mathbf{u}$ independently. This framework is missing in DGL, which is why we plan on implementing it for our work. Image reprinted from [7]. . . . .	36
4.6	Spectral radius loss function constantly stuck around 1 on AWS. Faced with this difficulty, it is important to consider more stable backpropagation strategies through eigen-decomposition in the future, as suggested in Wang et al. [8]. . . . .	37

---

## List of Tables

3.1	Popular graph neural network libraries written in Python, with GitHub numbers as of 18th May 2022. Based on this table and multiple other factors, we chose DGL for our implementations. . . . .	24
4.1	Results obtained when evaluating our three-layered GraphSAGE GNN model. These indicate that in about 20% of problems, our model outperforms classical AMG. Results for sizes greater than 65536 were not obtained, (probably) due to insufficient memory on AWS. . . . .	32
4.2	Results obtained by Luz et al. [5] when evaluating their Encode-Process-Decode GNN model. . . . .	33

---

## List of Algorithms

2.1	Multigrid V-cycle with $N$ levels to solve $A\mathbf{x} = \mathbf{b}$ . . . . .	11
-----	---	----



---

# Abbreviations

## Common Sets

$\mathbb{R}$	Real numbers set
$\mathbb{R}^n$	Set of real-valued vectors of size $n$
$\mathbb{R}^{n \times n}$	Set of real-valued matrices of size $n \times n$

## Common Mathematical Functions and Operators

$\mathbf{b}$	Vector $\mathbf{b}$
$b_i$	the $i^{\text{th}}$ element of vector $\mathbf{b}$
$\ \mathbf{b}\ $	Norm of vector $\mathbf{b}$
$A$	Matrix $A$
$a_{ij}$	Element of matrix $A$ at the $i^{\text{th}}$ row, and the $j^{\text{th}}$ column
$A^{-1}$	Inverse matrix to matrix $A$
$A^T$	Transposed matrix to matrix $A$
$\ A\ $	Norm of matrix $A$
$\text{cond } A$	Condition number of matrix $A$
$\max \{a, b\}$	Maximum of $a$ and $b$ , $a$ when $a \geq b$ , $b$ when $a < b$
$\sup B$	Supremum of set $B$
$O(n)$	The big $O$ notation

## Miscellaneous Abbreviations

<b>AMG</b>	Algebraic Multigrid
<b>GMG</b>	Geometric Multigrid
<b>AWS</b>	Amazon Web Services
<b>GNN</b>	Graph Neural Network
<b>DGL</b>	Deep Graph Library
<b>MLP</b>	Multilayer Perceptron
<b>SPD</b>	Positive Semi-Definite
<b>CSR</b>	Compressed Sparse Row



# Introduction

## 1.1 Motivation

Linear equations are pervasive in our everyday lives. In science and engineering, these equations are often coupled together to form linear systems. Numerical simulation of modern engineering problems can easily incorporate millions or even billions of degrees of freedom. This is the case for high-fidelity fluid simulations in gas-turbine engines during operation<sup>1</sup>.

On one hand, serial computing has shown its limits, with manufacturers finding it harder to build chips with increased clock frequency [9]. This has lead the community to rely heavily on parallel computing, which, as opposed to serial computing, has only risen in popularity since the mid 2000s (see fig. 1.1). Today, high-fidelity simulations are achieved in HPC facilities, taking advantage of different degrees of parallelism. On the other hand, the methods we use as numerical solvers for these large problems must be extremely fast to be viable. Furthermore, they must make use of some form of parallelism.

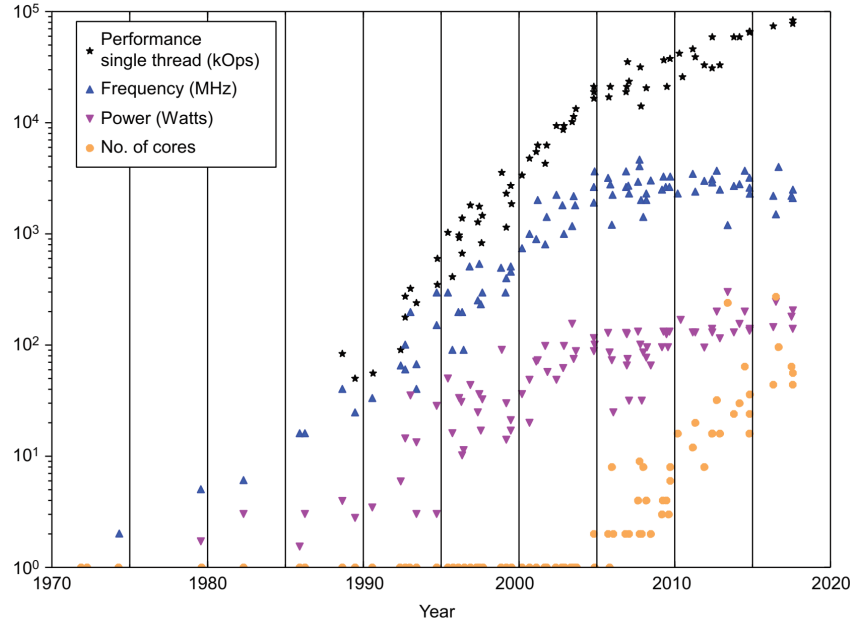
Algebraic Multigrid (AMG) is a relatively recent iterative method [10], which is firmly established as state-of-the-art for many large problems. This said, AMG still requires expert customisation to function optimally. It was designed for specific classes of matrices, and still struggles to take full advantage of the sparsity we encounter in fluid simulation.

All things considered, HPC resources are finite, and we need to use them in a smarter way i.e. develop strategies to accelerate linear solvers. AMG happens to be built upon the crucial *prolongation* step, which lends itself very well to the unsupervised machine learning (ML) paradigm. While several methods have tried leveraging ML for speed-up in solver time [3, 5], most have focused on structured<sup>2</sup> problems [3]. Our goal is to develop an AMG method augmented with AI that is: robust, computationally cheap, works for large and sparse matrices, works for problems defined on unstructured grids, and scalable with distributed computation.

Finally, we realize that AI-augmented methods such as ours need careful consideration of its users. Upon realizing the benefits of AMG, we were curious as to why it is not unilaterally adopted by the community. To the best of our knowledge, no studies has investigated such questions, es-

<sup>1</sup>A typical project where such problems are encountered is the ASiMoV industrial partnership.

<sup>2</sup>A structured grid is one that is always ordered, generally using three counters or indices  $(i, j, k)$  in 3D.



**Figure 1.1:** Comparison of serial performance (single CPU clock frequency and threads) against parallel computing (number of CPU cores) from 1970 to 2018. The parallel computing era begins about 2005, when the core count in CPU chips begins to rise, while the clock frequency and power consumption plateaus. Image reprinted from [1].

pecially when it pertains to AMG. This is why we decided to build on participatory design ideas to interactively build our methodology, placing human needs at the centre of our decisions.

## 1.2 Problem statement

The problem is to rapidly solve the linear system:

$$\mathbf{Ax} = \mathbf{b}, \quad (1.1)$$

where  $A$  is a large and sparse  $n \times n$  square matrix;  $\mathbf{x}$  and  $\mathbf{b}$  are column vectors of size  $n$ . We aim to solve eq. (1.1) by enriching AMG with a Graph Neural Network (GNN) model to help predict better prolongation operators. We identified Deep Graph Library (DGL) as the scalable package by excellence to build said model.

The immediate added value of our work is the potential speed-up in solving large problems in computational fluid dynamics and other related fields in engineering. Within the ASiMoV project, this should allow complete certification of jet engines based on their high-accuracy simulation during operation. Additionally, given that AMG is used by virtually all the biggest HPC facilities worldwide, our work will help save millions in financial cost of operation, all the while addressing climate change. Indeed, amidst the energy crisis currently plaguing Europe [11], streamlining operations in HPC facilities should keep them running for less long, hence saving vital energy.

Our immediate hypothesis is that AI-enabled AMG should be faster than its classical counterpart. That said, our project has a wider long-term hypothesis to test: how efficient is AI at improving high-fidelity simulations ?



### 1.3 Related work

Our work builds on several impactful results obtained within the past three years. First, Greenfeld et al. [3] developed a deep residual feed-forward network to predict the prolongation operator when facing problems on a 2D structured grid. They trained one neural network for each class of PDEs, and demonstrated improved convergence on 2D diffusion problems.

The next seminal work is that of Luz et al. [5]. It builds on Greenfeld et al. [3] by leveraging some of the same training strategies. Quite distinctly though, it focuses on unstructured grids, outperforming classical AMG on graph Laplacian problems, 2D FEM problems, and spectral clustering.

The third and most recent piece of work that underlined our study is that of Louw and McIntosh-Smith [4]. It not only showed that the model in [3] could be simplified and still outshine classical AMG, but it also introduced a new dataset specifically tailored to fluid simulations. Their biggest finding was that the model in [5] is not ready for large distributed applications.

### 1.4 Contributions of this thesis

Our work in this project is method-oriented, i.e. focuses more on the development of a methodology rather than conducting detailed experiments. Below are the major contributions to the problem statement:

- (i) A Graph Neural Network methodology<sup>3</sup> for learning prolongation operators built around Deep Graph Library and PyTorch. Our GNN model is based on [5] and includes: encoding, message-passing, and decoding steps. It is trained with an unsupervised objective function.
- (ii) A Cloud Formation stack on Amazon Web Services to run experiments, containing all the dependencies our methodology requires. Built from a MATLAB template, this stack will be maintained for future experiments.
- (iii) Experiments indicating that the current version of our AI-enabled AMG method is better than the classical approach in about 20% of cases, for both V- and W-cycles, for graph Laplacian and 2D diffusion problems.
- (iv) A user survey with insights from the community that stands to benefit from our research. At the end of it, participants expressed increased interest in AI-enabled AMG. The survey, among other observations, stressed the importance of a careful approach to disseminating our findings. It considerably informed future experiments and research directions.

### 1.5 Structure of the thesis

The thesis is organized into 5 chapters as follows:

1. *Introduction*: Describes the motivation behind our efforts together with our goals.
2. *Background and state-of-the-art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.

<sup>3</sup><https://github.com/desmond-rn/accelerating-amg>

## 1. INTRODUCTION

---

3. *Overview of our approach*: Introduces our unsupervised learning approach based on Graph Neural Networks, and the design of a user questionnaire.
4. *Results, discussion, and challenges*: Describes our experiments and the accompanying results and discussion; and presents challenges faced during the project.
5. *Conclusion*: Concludes the thesis by summarizing the results of our project, and suggests topics for further research.

## Background and state-of-the-art

Our efforts accelerating AMG benefited from a long body of research. Solving linear problems has always been at the heart of many science and engineering problems. In the first section of this chapter, we present the contextual and technical background needed to understand the most important aspects of linear solvers and AMG. The second section is dedicated to the series of recent and influential work that inspired ours. The third section gives a general background and motivation for surveys and questionnaires in applications like ours.

### 2.1 Background on linear solvers

The context of our project is the linear problem (with  $A \in \mathbb{R}^{n \times n}$ ,  $\mathbf{x} \in \mathbb{R}^n$ , and  $\mathbf{b} \in \mathbb{R}^n$ ):

$$A\mathbf{x} = \mathbf{b}, \tag{1.1}$$

which arises from the discretization of partial differential equations in engineering. Specifically, our linear systems originate from *high-fidelity* simulations of complete gas-turbine engines during operation. These include several coupled physical effects: thermo-mechanics, electro-magnetics, and more importantly, computational fluid dynamics. Our problem is complicated by the fact that the matrix is  $A$  is both extremely *large* (millions of degrees of freedom), and incredibly *sparse* (less than 1% dense). As such, any viable solver must take advantage of these properties. Furthermore, given that we solve systems like (1.1) hundreds or thousands of times for simple simulations, it puts further constraints on the method, which now has to be *robust* too. Let's review some techniques developed over centuries to tackle this problem, beginning with *direct methods*.

#### 2.1.1 Direct methods

These are methods that theoretically give the exact solution<sup>1</sup> to eq. (1.1) in a finite number of steps. Direct methods are much more effective when the matrix  $A$  is either diagonal, orthogonal, or triangular. This is why the matrix  $A$  is often decomposed into better manageable chunks before attempting to obtain the solution  $\mathbf{x}$ . The most intuitive but naive approach is to invert  $A$ , and thus

<sup>1</sup>Computationally, the exact solution can only be obtained in the absence of round-off errors.

$\mathbf{x} = A^{-1}\mathbf{b}$ . Regardless of the strategy, computing  $A^{-1}$  is prohibitively expensive for our use cases (at least  $O(n^3)$ ). Besides, invertibility of  $A$  is not always guaranteed. Examples of other much improved methods include Gaussian elimination by LU decomposition, or the and Cholesky algorithm [12].

In the case of LU decomposition,  $A$  is decomposed into a product of lower and upper triangular matrices,  $L$  and  $U$  respectively. And then backward and forward substitutions are applied to successively eliminate variables. That said, the average cost of using direct methods is  $O(n^3)$ , which is evidently not good enough (considering the enormous size of  $n$ ). On top of its huge computational cost, direct Gaussian elimination by LU decomposition suffers from several drawbacks [13]:

- It may introduce *fill-in*:  $L$  and  $U$  may have non-zero elements at locations where the original matrix  $A$  had zeros, putting further strain on sparse memory storage.
- Sometimes we may not really need to solve the system exactly, like it is the case for some nonlinear problems. This is impossible because direct methods, by definition, cannot be interrupted.
- Sometimes, we have a pretty good idea of the solution; especially in instationary simulations, where the previous time-step's solution can serve as a *warm* start for the current time-step. Direct methods cannot take advantage of this feature.

### 2.1.2 Iterative methods

The other family of methods are *iterative*, more commonly known as *relaxation methods*. They lead to approximate solutions to (1.1) after a (potentially infinite) number of steps, by improving on an initial guess. The basic mechanism is the following: given  $A\mathbf{x} = \mathbf{b}$ , we write  $A = W - V$ . We then have  $W\mathbf{x} = V\mathbf{x} + \mathbf{b}$ , which leads to the fixed point iteration:

$$W\mathbf{x}_{k+1} = V\mathbf{x}_k + \mathbf{b}. \quad (2.1)$$

Let's suppose that  $W$  is much easier to invert than  $A$  (assumed to be non-singular). Given an initial guess  $\mathbf{x}_0$ , the quantity

$$\mathbf{e}_0 := \mathbf{x} - \mathbf{x}_0, \quad (2.2)$$

is the initial *error*, and

$$A\mathbf{e}_0 = \mathbf{b} - A\mathbf{x}_0 =: \mathbf{r}_0, \quad (2.3)$$

which we call *residual*<sup>2</sup>. Since  $\mathbf{x} = \mathbf{x}_0 + \mathbf{e}_0 = \mathbf{x}_0 + A^{-1}\mathbf{r}_0$ , we find  $\tilde{\mathbf{e}}$  using

$$\tilde{\mathbf{e}} = W^{-1}\mathbf{r}_0, \quad (2.4)$$

and we take

$$\mathbf{x}_1 = \mathbf{x}_0 + \tilde{\mathbf{e}}, \quad (2.5)$$

as our new guess. The hope is that  $\mathbf{x}_1$  is closer to  $\mathbf{x}$  than  $\mathbf{x}_0$  was to  $\mathbf{x}$ . The process is repeated to find  $\mathbf{x}_2, \mathbf{x}_3, \dots$ , and so on.

---

<sup>2</sup>It is important to note that the residual is computable, whereas the error isn't.

At each step, iterative methods require evaluating a stopping criterion, e.g. checking the norm of the residual error  $\|\mathbf{r}\| = \|\mathbf{b} - A\mathbf{x}\|$  against a tolerance. In the case of a full matrix  $A$ , relaxation schemes cost about  $O(n^2)$  for each iteration, to be contrasted with about  $\frac{2}{3}O(n^3)$  floating-point operations needed for Gaussian Elimination by LU decomposition<sup>3</sup>. Within a prescribed tolerance, iterative methods are very competitive against direct methods, especially if the number of iterations required for convergence is either fixed or scales sub-linearly with respect to  $n$ .

### 2.1.2.1 Stationary iterative methods

Optimal splitting of  $A$  into  $W$  and  $V$  is not an easy task. Ideally,  $W$  should contradictorily be (i) close to  $A$ , or rather  $W^{-1}$  should be close to  $A^{-1}$  as pointed out in eq. (2.4); and (ii) much easier to invert than  $A$ . This duality has given rise to a multitude of stationary methods. If we write  $A = D + E + F$ , where  $D$  is the *diagonal* of  $A$ ,  $E$  is its strictly *upper* triangular part, and  $F$  its strictly *lower* triangular part, we can distinguish the following types of relaxations [5]:

- **Richardson** [14]: where  $W = \theta^{-1}I$ , with  $\theta \in \mathbb{R}$ , and  $I$  the identity matrix the same size as  $A$ . This is one of the simplest linear iteration methods.
- **Jacobi** [15]:  $W = D$ . This method is quite slow compared to others below, but easily parallelisable to take advantage of HPC.
- **Gauss-Siedel** [16, 17]:  $W = D + E$ . Also quite slow, but has good *smoothing* properties, beneficial for Multigrid methods.
- **Successive over-relaxation (SOR)** [18]:  $W = \omega^{-1}D - E$ . A Gauss-Siedel variant with an optimizable parameter  $\omega > 0$ , whose cost of computing should not be neglected.

A linear iteration method is said to be in *second normal form* if it satisfies eq. (2.1) and is *consistent* i.e. it has a fixed point solution [19, p.22]. However, a more interesting form when analysing relaxation methods is the *third normal form*, which additionally requires  $W$  being regular. In this form, we have:

$$\mathbf{x}_{k+1} = W^{-1}V\mathbf{x}_k + W^{-1}\mathbf{b}, \quad (2.6)$$

and then we can write the error as

$$\begin{aligned} \mathbf{e}_{k+1} &= \mathbf{x}_{k+1} - \mathbf{x} \\ &= W^{-1}V\mathbf{x}_k + W^{-1}\mathbf{b} - \mathbf{x} \\ &= W^{-1}V\mathbf{e}_k + \underbrace{(W^{-1}V - I)\mathbf{x} + W^{-1}\mathbf{b}}_{=0 \text{ since } V=W-A} \\ &= (I - W^{-1}A)\mathbf{e}_k, \end{aligned}$$

thus highlighting the *error propagation matrix*  $M$ , defined such that  $\mathbf{e}_{k+1} = M\mathbf{e}_k$ , and written as:

$$M = I - W^{-1}A. \quad (2.7)$$

<sup>3</sup>This simplistic estimation only considers floating point operations. It doesn't take into account data movement between elements of the computer's memory hierarchy, which can be crucial, especially in multiprocessor architectures.

This quantity is central to the convergence analysis of all iterative methods, and to ours in particular. The asymptotic convergence of the relaxation scheme is governed by the *spectral radius*<sup>4</sup> of  $M$ , which should be smaller than 1. The smaller it is, the faster we converge [13].

As a final note, the *third normal form* is often expressed as [19, p.23]:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - W^{-1}(A\mathbf{x}_k - \mathbf{b}), \quad (2.8)$$

in order to highlight the fact that the solution at one iteration is the same as the solution at the previous iteration, corrected by a term proportional to the residual.

### 2.1.2.2 Nonstationary iterative methods

Iterative methods like Jacobi, Gauss-Siedel, or SOR are qualified as *stationary* because they do not take advantage of the information accumulated through the iterations. *Nonstationary* methods try to address this problem by finding the solution in an expanding Krylov subspace  $\mathcal{K}$ :

$$\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}^k(A; \mathbf{r}_0) \equiv \mathbf{x}_0 + \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, A^2\mathbf{r}_0, \dots, A^{k-1}\mathbf{r}_0\}. \quad (2.9)$$

The surrogate goal is then to minimize each new residual's norm while requiring it to be *orthogonal* to the subspace. This should (theoretically) lead to  $\mathbf{r} = 0$  and termination after a finite number of steps.

Conjugate Gradient (CG) is the archetypical Krylov subspace method [20]. It applies to positive semi-definite (SPD) matrices. In CG, the linear problem (1.1) is recast as the convex optimization of the quadratic function

$$\varphi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A \mathbf{x} - \mathbf{x}^T \mathbf{b} + c, \quad (2.10)$$

where  $c \in \mathbb{R}$ . The function is then minimized using gradient descent. The direction of steepest descent (closely related to the residual) is chosen to be  $A$ -conjugate (or  $A$ -orthogonal) to the existing subspace, and minimizes the energy-norm (or  $A$ -norm):

$$\|\mathbf{e}_k\|_A = \sqrt{\mathbf{e}_k^T A \mathbf{e}_k}. \quad (2.11)$$

CG is guaranteed to converge in under  $n$  iterations. However, this number might still be too large considering the size of the problems we are tackling.

Other well-known Krylov subspace methods include: Generalized minimal residual method (GMRES) [21] which finds orthogonal directions by making use of the Lanczos and Arnoldi algorithms [22, 23]; Biconjugate gradient stabilized method (BiCG-Stab) with good parallelisable properties [24]; Quasi-Minimal Residuals (QMR) [25]; and many more.

### 2.1.2.3 Preconditioning

A factor that influences the convergence rate of any iterative method is the *condition number*<sup>5</sup>, defined as:

$$\text{cond } A = \|A\| \times \|A^{-1}\|, \quad (2.12)$$

<sup>4</sup>The maximum of the absolute values of a matrix's eigenvalues.

<sup>5</sup>Although this factor is secondary to the spectral radius of the error propagation matrix.

where  $\|A\|$  is the *matrix norm* of  $A$  associated with a *vector norm*  $\|\cdot\|$ , such that:

$$\|A\| = \sup_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \quad (2.13)$$

In short, the condition number indicates how stable a linear system is. A good conditioning means the solution to  $A\mathbf{x}' = \mathbf{b}'$  remains close to  $\mathbf{x}$  if  $\mathbf{b} \approx \mathbf{b}'$ . A blossoming field of research is the complex art of finding good preconditionners for specific problems, i.e. define a matrix  $P$  and solve the surrogate system

$$P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}, \quad (2.14)$$

rather than  $A\mathbf{x} = \mathbf{b}$ . The matrix  $P$  should not only be relatively easy to invert, but it is also desirable that  $\text{cond}(P^{-1}A) \ll \text{cond}(A)$ . Preconditioning is also important because it can lead to systems that possess the most crucial property for fast convergence: a good distribution/clustering of eigenvalues [13].

Important results have shown that instability and non-convergence of relaxation schemes is caused by the small frequency (large wavelength) component of the residual error, which amplifies with successive iterations [26, p.201]. The same effect has also been demonstrated when the method is stable. Indeed, the convergence rate is primarily dictated by eigenvectors whose associated eigenvalues have relatively small frequencies<sup>6</sup>. Furthermore, solving linear systems on “coarser-grids” facilitates the smoothing of those small frequencies.

### 2.1.3 Multigrid methods

Multigrid methods are a special family of iterative methods developed to quickly smooth out small frequency components of the error [26, 19]. Multigrid methods create a hierarchy of smaller grids and make use of relaxation on fine grids for rapid convergence (up to  $O(n)$  for a grid with  $n$  nodes, in the optimal Geometric Multigrid case [4]). In order to understand Algebraic Multigrid (AMG) and how we plan to improve it using Machine Learning, it is important to explain how the Geometric Multigrid (GMG) method works.

#### 2.1.3.1 Geometric Multigrid (GMG)

In GMG, the hierarchy of meshes that are employed to rapidly reduce the residual errors is made available to the solver. In other words, the original GMG approach requires the problem geometry. The GMG algorithm with two levels – the original fine grid  $\Omega_f$  and the derived coarse grid  $\Omega_c$  – is summarized below [26, p.205].

Let  $\mathbf{x}^*$  be the exact solution to eq. (1.1). We write:

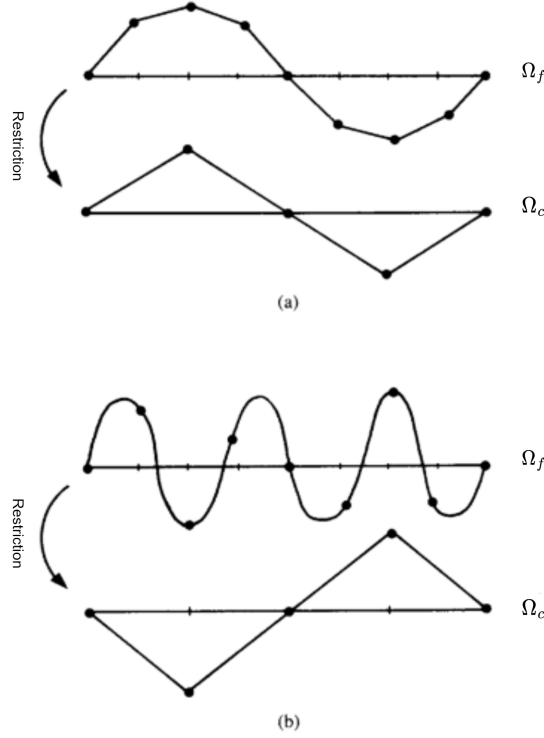
$$\mathbf{e}^n = \mathbf{x}^* - \mathbf{x}^n \quad \text{and} \quad \mathbf{r}^n = \mathbf{b} - A\mathbf{x}^n, \quad (2.15)$$

where  $n$  designates the iteration step. GMG assumes that the error can be decomposed into an oscillatory (high-frequency) part  $\mathbf{e}^{n,\text{osc}}$  and a smooth (low-frequency) part  $\mathbf{e}^{n,\text{smth}}$ :

$$\mathbf{e}^n = \mathbf{e}^{n,\text{osc}} + \mathbf{e}^{n,\text{smth}}, \quad (2.16)$$

<sup>6</sup>Eigenvalues can often be written as sinusoidal functions of the problem or the grid size, when applicable.

and bets on the fact that  $\mathbf{e}^{n,\text{osc}}$  converges to 0 much faster than its smooth counterpart [19, p.267]. GMG then builds on the intuitive idea that smooth quantities do not require high-resolution grids to be accurately fitted. In fact, coarser meshes are more suitable, and computation in them is orders of magnitude faster. Figure 2.1 illustrates the benefit of approximating a smooth 1D quantity on a coarse grid rather than a fine one.



**Figure 2.1:** Restriction of a smooth quantity from a fine grid  $\Omega_f$  to a coarse grid  $\Omega_c$  resulting in a good approximation (a), compared to the restriction of an oscillatory quantity resulting in a bad approximation (b). Image adapted from [2].

With  $\mathcal{S}(\cdot)$  denoting the relaxation method (preferably weighted Jacobi or Gauss-Seidel for reasons stated in section 2.1.2.1)<sup>7</sup>, the steps for two-level GMG are listed below:

1. Generate meshes and their relationships, deriving prolongation and restriction operators ( $P$  and  $R$  respectively)
2. Set an initial guess for the solution on the finest mesh:  $\mathbf{x}_f^0$
3. Pre-smooth (i.e. perform several relaxation sweeps) on the fine grid:  $\mathbf{x}_f^{n,\text{presmth}} = \mathcal{S}^{\sigma_1}(\mathbf{x}_f^n)$ , thus setting  $\mathbf{e}_f^{n,\text{osc}} \approx 0$
4. Compute the residual on the fine grid:  $\mathbf{r}_f^n = \mathbf{b}_f - A_f \mathbf{x}_f^{n,\text{presmth}}$
5. Transfer the fine-grid residual to the coarse grid (**restriction**):  $\mathbf{r}_c^n = R \mathbf{r}_f^n$
6. Determine the coarse grid operator using the Galerkin product  $A_c = R A_f P$ . Note that it could be easier to reassemble  $A_c$  than to use the Galerkin product.

<sup>7</sup>The exponent  $\sigma$  in  $\mathcal{S}^\sigma$  indicates the number of relaxation sweeps applied.



7. Compute the error on the coarse grid (either *direct solve* or another *smoothing*):  $A_c \mathbf{e}_c^n = \mathbf{r}_c^n$ .
8. Transfer the coarse-grid error to the fine grid (***prolongation*** or ***interpolation***<sup>8</sup>):  $\mathbf{e}_f^{n,*} = P \mathbf{e}_c^n$ , hoping that  $\mathbf{e}_f^{n,*} \approx \mathbf{e}_f^{n,\text{smth}} \approx \mathbf{e}_f^n$
9. Update the fine-grid solution (*correction*):  $\mathbf{x}_f^{n,\text{postsmth}} = \mathbf{x}_f^{n,\text{presmth}} + \mathbf{e}_f^{n,*}$
10. Post-smooth on the fine grid:  $\mathbf{x}_f^{n+1} = \mathcal{S}^{\sigma_2}(\mathbf{x}_f^{n,\text{postsmth}})$
11. Check for convergence

The difference between the two-level algorithm and the multi-level version appears at step 7 above. Noticing that the equation  $A_c \mathbf{e}_c^n = \mathbf{r}_c^n$  is a linear system similar to the original problem, we can solve it with another Multigrid method, yielding the recursive process known as *Multigrid V-cycle*<sup>9</sup> (see algorithm 2.1)<sup>10</sup>. An illustrative image (with  $N = 3$ ) is presented in fig. 2.2.

---

**Algorithm 2.1** Multigrid V-cycle with  $N$  levels to solve  $A\mathbf{x} = \mathbf{b}$ .

---

**Input:**  $A, \mathbf{b}$

**Input:**  $\mathbf{x}_0$

▷ Initial guess

**Input:**  $\mathcal{S}, \sigma_1, \sigma_2$

▷ Relaxation scheme

**Input:**  $\delta$

▷ Residual tolerance

**Output:**  $\mathbf{x}$

```

1: procedure MULTILEVEL( $A_k, \mathbf{f}, \mathbf{u}, k$ )           ▷ Solves  $A_k \mathbf{u} = \mathbf{f}$  ( $k < N$  is the current grid level)
2:    $\mathbf{u} = \mathcal{S}^{\sigma_1}(A_k, \mathbf{f}, \mathbf{u})$                  ▷ Compulsory pre-smoothing
3:   if ( $k < N - 1$ ) then
4:      $P_k = \text{determine\_interpolant}(A_k)$ 
5:      $R_k = \text{determine\_restrictor}(A_k)$ 
6:      $\mathbf{r}_{k+1} = R_k(\mathbf{b} - A_k \mathbf{u})$                  ▷ Restriction
7:      $A_{k+1} = R_k A_k P_k$ 
8:      $\mathbf{v} = 0$ 
9:     MULTILEVEL( $A_{k+1}, \mathbf{r}_{k+1}, \mathbf{v}, k + 1$ )
10:     $\mathbf{u} = \mathbf{u} + P_k \mathbf{v}$                              ▷ Prolongation and correction
11:     $\mathbf{u} = \mathcal{S}^{\sigma_2}(A_k, \mathbf{f}, \mathbf{u})$                  ▷ Optional post-smoothing
12:   else
13:      $\mathbf{u} = \text{direct\_solve}(A_k, \mathbf{f})$ 
14:   end if
15: end procedure

16:  $\mathbf{x} = \mathbf{x}_0$ 
17: repeat                                           ▷ Iterate until convergence
18:    $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ 
19:   MULTILEVEL( $A, \mathbf{b}, \mathbf{x}, 0$ )
20: until  $\|\mathbf{r}\| \leq \delta$ 

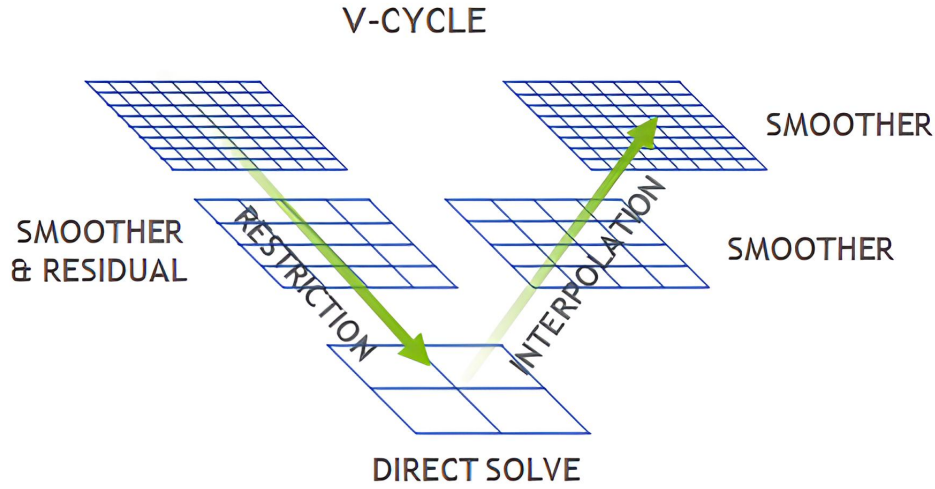
```

---

<sup>8</sup>Not all prolongation strategies are interpolative. Nevertheless, the two terms will be synonymous in this report.

<sup>9</sup>Other variations exist and report overall better performance, like the W- and F-cycles.

<sup>10</sup>Note that lines 12 to 14 are optional and should only be included if the problem size at level  $N - 1$  is small enough.



**Figure 2.2:** Illustration of GMG in a multi-level 2D scenario. In each V-cycle, the solution is smoothed, and a residual is computed and propagated to the coarser grid. At the coarsest level, a direct solver (or another smoother) is applied, and the solutions are then iteratively interpolated to finer grids. Image adapted from [nVidia].

### 2.1.3.2 Algebraic Multigrid (AMG)

While GMG is one of the most powerful method for solving linear systems, its formulation and execution are inherently tied to a grid, hampering its generality. The key distinction between the Algebraic Multigrid (AMG) and GMG method is that AMG doesn't require meshes for the restriction nor prolongation operations, making it easily reusable. Upon inputting values for  $A$  and  $\mathbf{b}$ , AMG will automatically decide which equations are agglomerated based on an energy-like connection between them [26]. Given the absence of grids, AMG typically defines the restriction operator as  $R = P^T$ . Furthermore, the coarse-grid operator  $A_c$  is computed from the fine grid operator using the Galerkin product  $A_c = P^T A_f P$  (see algorithm 2.1, which also applies to AMG depending on lines 4. and 5.).

Most of AMG rests on two fundamental concepts. The first concept (*smooth error*) was addressed in section 2.1.3.1. The second important concept is that of *strong dependence* or *strong influence*. Several AMG methods exist, each with their own coarsening strategy and definition of strong dependence. The earliest AMG coarsening strategy is due to Ruge and Stüben [10], and from now will be referred to as “classical” AMG. We will focus on the Ruge-Stüben (RS) algorithm in the sequel.

**Overview of the classical AMG algorithm.** For simplicity of notations, a node defined by the variable  $x_i$  will be denoted as node  $i$ , or just  $i$ . In the absence of a grid for visual reference, classical AMG defines two nodes  $i$  and  $j$  as *strongly* connected if given a threshold  $0 < \theta \leq 1$ , we have [27]:

$$|a_{ij}| \geq \theta \max_{k \neq i} |a_{ik}|. \quad (2.17)$$

If  $i$  is connected to  $j$ , but not strongly connected, it is said to be *weakly* connected. Next, an array  $\lambda$  is created ( $\lambda_i$  is the number of strong connections to node  $i$ ).

The Ruge-Stüben algorithm then defines two sets forming a partition over the original “grid” (this constitutes the **first phase** of the whole process):

- C-points: the set of points selected to be part of the coarse grid.

- $F$ -points: the set of points *not* selected to be in  $C$ <sup>11</sup>.

In a mandatory *first pass*, the Ruge-Stüben algorithm assign each node to either  $C$  or  $F$  following the steps below [27]:

1. Find the unassigned grid point  $i$  which has the largest  $\lambda_i$  value; add the point to  $C$ .
2. Add every  $j$  strongly connected to  $i$  to  $F$ .
3. For each node  $j$  in step 2, find every  $k$  strongly connected to  $j$  and increase  $\lambda_k$  by one.
4. If every node is either in the fine set or coarse set, end the first pass. Else return to step 1.

The *second pass* checks that no fine node is connected to another fine node. If any such nodes are found, one of the two will be made a coarse node.

The **final phase** in all coarsening algorithms is to compute the weights, i.e. how coarse nodes influence fine nodes during prolongation. First, a few definitions are needed. For each fine-grid point  $i$ , define  $N_i$ , the neighbourhood of  $i$ , which contains all  $j \neq i$  such that  $a_{ij} \neq 0$ . These points can be divided into three categories [27]:

- The *neighbouring coarse-grid points* that strongly influence  $i$ ; this is the coarse interpolation set for  $i$ , denoted by  $C_i$ .
- The *neighbouring fine-grid points* that strongly influence  $i$ , denoted by  $D_i^s$ .
- *Weakly connected neighbours*, i.e. the points that do not strongly influence  $i$ , denoted by  $D_i^w$  (it may contain both coarse- and fine-grid points).

The weights can now be calculated using the formula:

$$w_{ij} = -\frac{a_{ij} + \sum_{l \in D_i^s} \left( \frac{a_{il}a_{lj}}{\sum_{k \in C_i} a_{lk}} \right)}{a_{ii} + \sum_{n \in D_i^w} a_{in}} \quad \forall i \in F \text{ and } j \in C. \quad (2.18)$$

Then the prolongation operator  $P$  is constructed row-wise.  $P$  has one row for each original node  $i \in C \cup F$ , and one column for each coarse node  $j \in C$ . If  $i$  is a coarse node, then the  $i$ -th row will be the *identity*, i.e.

$$p_{ij} = \begin{cases} 1 & \text{at the corresponding column } j \text{ (not necessarily } j = i), \\ 0 & \text{elsewhere.} \end{cases} \quad (2.19)$$

If  $i$  is a fine node, then the corresponding row of the matrix of weights is used according to (2.18).

Despite its relative simplicity, the classical Ruge-Stüben coarsening strategy described above served as a good baseline in our experiments. In our evaluation, we considered improved strategies, like Cleary-Luby-Jones-Plassman (CLJP) coarsening [28], a graph-based parallelisable approach. Still, we acknowledge that other heuristics provide (better) coarsening strategies for specific problems, more suited for the real world. Among those, we find some that could even lead to lower complexities for harder problems, like the Falgout coarsening [29], PMIS and HMIS [30], and many more.

<sup>11</sup>To avoid confusion in the sequel,  $F$ -points will simply be referred to as fine points, while  $C \cup F$  will form the original set of points defining the problem  $\mathbf{Ax} = \mathbf{b}$ .

**The error propagation matrix.** The error propagation matrix  $T$  for the core<sup>12</sup> two-level Multigrid V-cycle is of particular importance. Starting at the coarse-grid correction step,  $T$  is obtained by expressing  $\mathbf{x}_f^{n+1}$  in terms of  $\mathbf{x}_f^n$ , by leveraging the steps described in the GMG section (see 2.1.3.1):

$$\begin{aligned}
\mathbf{x}_f^{n+1} &= \mathbf{x}_f^n + \mathbf{e}_f^{n,*} \\
&= \mathbf{x}_f^n + P\mathbf{e}_f^n \\
&= \mathbf{x}_f^n + P[A_c^{-1}\mathbf{r}_f^n] && \text{(assuming direct solve on coarse grid)} \\
&= \mathbf{x}_f^n + P(A_c^{-1})R(\mathbf{b} - A_f\mathbf{x}_f^n) \\
&= \mathbf{x}_f^n - [P(P^T A_f P)^{-1} P^T](A_f\mathbf{x}_f^n - \mathbf{b}).
\end{aligned}$$

Based on the third normal form presented in eq. (2.8) and the error propagation formula in eq. (2.7), we deduce:

$$T = I - [P(P^T A_f P)^{-1} P^T]A. \quad (2.20)$$

One important property of  $T$  is that it is an  $A$ -orthogonal projection, which highlights the close relationship with other Krylov subspace methods. To complete the error propagation matrix for the complete two-level multigrid V-cycle, we define  $S$  as the prolongation matrix for the smoother  $\mathcal{S}(\cdot)$ . We land on:

$$M = S^{\sigma_2} \left( I - P [P^T A P]^{-1} P^T A \right) S^{\sigma_1}. \quad (2.21)$$

The meaning of  $M$  for the two-level Multigrid algorithm is central to our project.  $M$  has been used extensively in recent work involving AI. Indeed, recent techniques using Machine Learning to predict a prolongation operator often require minimizing the error propagation matrix. As a consequence, a particular care should be given to the selection of  $P$  on which  $M$  is dependant, whether it derives naturally from a fine grid (GMG), ad-hoc from strong and weak connections (classical AMG), or from a Graph Neural Network (AI-augmented AMG).

## 2.2 Previous results and related work

Our goal is to find the prolongation operator  $P$  in such away that the residual error is quickly decreased. Several works have tried predicting better prolongation operators using Machine Learning. This constitutes the setup and part of the problem that Louw and McIntosh-Smith [4] tried to solve.

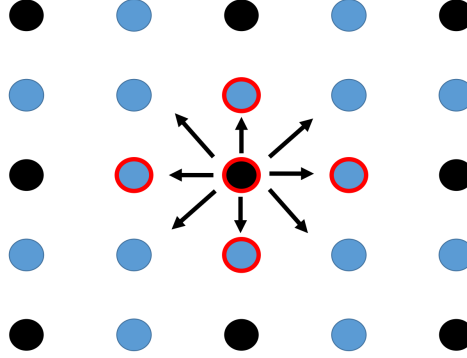
One of the earliest ML-based methods is the Deep Multigrid Method [31]. Despite its substantial upsides, the method requires retraining of a new deep neural network for each new matrix  $A$ , making it unsuitable for our case. In contrast, Greenfeld et al. [3] and Luz et al. [5] were able to “train once and use anywhere”. Moreover, while [31] tries to minimize the spectral radius  $\rho(M)$  of the error propagation matrix  $M$ , [3] and [5] attempt to minimize a surrogate instead (the Frobenius norm  $\|M\|_F$ ), which appears to be more stable. Indeed, computing  $\rho(M)$  requires inverting the quantity  $P^T A P$  (see eq. (2.21)), which, as we have stated in subsection 2.1.1, is a costly operation. As for  $\|M\|_F$ , this can be avoided, or the very least improved (see section 3.3.1).

### 2.2.1 Greenfeld et al.

A summary of the deep residual feed-forward network-based method proposed in Greenfeld et al. [3] is presented in [4]. The main (and limiting) assumption of the work is that the grid is structured. As such,

<sup>12</sup>The core is made up of all the steps of the complete two-level V-cycle except for pre- and post-smoothing.

the coarse grid is contained within the fine grid. For each point in the coarse grid, we seek to determine how it contributes towards neighbouring fine points (those not in the coarse grid). The input to the network are the  $3 \times 3$  stencils of the point itself and of its 4 neighbours, which is flattened and results in a vector in  $\mathbb{R}^{45}$ . The output is the vector in  $\mathbb{R}^8$  with the desired weightings (see fig. 2.3). If other neighbours need computing, their weighting can be found algebraically. As stated earlier, this method is only suited for structured grids and requires strong integration with the geometry of the problem, which constitutes a considerable limitation.



**Figure 2.3:** Inputs and outputs of the network in Greenfeld et al. [3]. The discs denote the (fine) grid points, where the black discs mark the subset of points selected as coarse grid points. The input of the network consists of the  $3 \times 3$  stencils of the five points, denoted by the red cycles. The black arrows illustrate the output of the network, i.e., the contribution of the prolongation of one coarse point to its eight fine grid neighbours. Image reprinted from [3].

### 2.2.2 Luz et al.

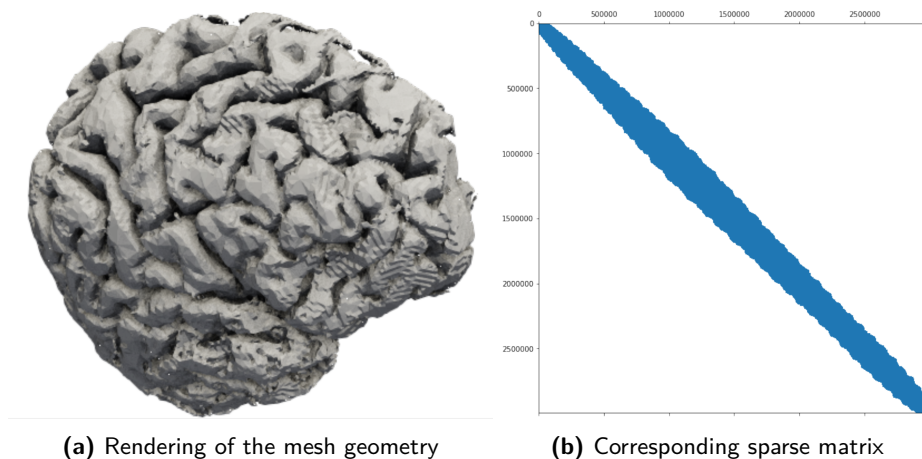
Graph Neural Networks have been key to scientific discovery over the last years [7, 32, 33] thanks to their strong inductive bias. They allow dealing with unstructured grid, reason why they were a key contribution in the work of Luz et al. [5]. The work took advantage of the encode-process-decode architecture proposed by [7] (and their accompanying code for GraphNets)<sup>13</sup> to learn generalizable prolongation matrices  $P$ . Despite sampling the entries for  $A$  from a log-normal distribution (and making sure  $A$  could be expressed as the Laplacian of a certain graph), they demonstrated generalization properties on 2D Finite Elements method problems. That said, GraphNets is heavily outdated, to the point that reproducing results in [5] was impossible. As such, upgrading their work into a more robust and scalable framework was a primary milestone for our project. Given that [5] formed an important part of our work, further aspects of its methodology and results will be presented in chapters 3 and 4.

### 2.2.3 Louw and McIntosh-Smith

In their 2021 paper, Louw and McIntosh-Smith [4] attempted recent techniques from [3] and [5], and investigated data- and model-parallel distributed approaches for training and especially inference. After adapting both techniques, they found (despite several re-engineering tricks) that time and memory complexity were still prohibitively high for the kind of matrices in the dataset<sup>14</sup> they created (see fig. 2.4). They concluded that further work is required before these methods are ready for general application to real-world problems. That said, their work showed that there is hope for simpler models to achieve good results, especially for the work in [3].

<sup>13</sup>[https://github.com/deepmind/graph\\_nets](https://github.com/deepmind/graph_nets)

<sup>14</sup>Thingy10K: a dataset of 30,000 high-dimensional and sparse matrices (see fig. 2.4).



**Figure 2.4:** An element of the Thingi10K dataset created by Louw and McIntosh-Smith [4]. Using the geometry from (a), a FEM fluid simulation is carried out to obtain the large and sparse matrix visualized in (b). Images reprinted from [4].

Our work builds on results from Greenfeld et al. [3], Luz et al. [5], and more importantly Louw and McIntosh-Smith [4], in order to build a robust single-node neural network model for large and sparse matrices. In an interactive process, our work will open the door to efficient design-space exploration in three senses: (i) users selecting weights for their prolongation operator, (ii) users customizing hyperparameters of their model for reusability on different PDEs via Bayesian optimization, and (iii) users contributing to the form and function in shape optimization of gas-turbine engines in simulation. Finally, our work wouldn't have succeeded had it not been for the large community of users that stands to gain from our efforts.

### 2.3 Background on interactive design

As stated in the sections above, the Algebraic Multigrid (AMG) method is undeniably the fastest in numerical analysis. And yet the method is only used in selected situations. Indeed, there is no “one method fits all” in numerical analysis. In this multiphysics, multiscale era, fluid simulation experts must often hand-pick the most suitable method for their problem. And when that method is AMG, the choice of the coarsening strategy and the prolongation operator is critical for the swift convergence of the solver. This is one of the problems our AI-augmented approach is trying to solve.

Our efforts will go unnoticed if we do not understand why so many people are not adopting AMG techniques in the first place. Admittedly, there are mathematical restriction on the type of problems AMG should be comfortably applied to ( $M$ -matrices, SPD, symmetric, etc). But still, after months researching this method, it is quite surprising how little it is adopted by academicians and companies alike. This might be linked to the fact that unlike direct and other iterative solvers, AMG is not taught in Numeral Analysis undergraduate curriculums. Gaining insight here would help better introduce people to AMG, and thus use our software.

Any AI-enabled method needs to be scrutinized, even for low-stakes applications like ours — the consequences of a bad prediction are low compared to other AIs that can have life-altering influence. But would people use it for high-precision applications like the certification of a jet-engine?

In summary, there are several questions that need answering before our envisioned software can be deployed to aid researchers and private companies:

- Are people aware of AMG? Is it among their top choice solvers?
- Are people aware of other fields that rely on core AMG strategies (namely coarsening and prolongation)?

- Why would people prefer direct solvers or other iterative solvers over AMG? Would things be different had they been exposed to AMG earlier in their curriculum?
- Can people rate the difficulty to implement and incorporate AMG into existing solvers?
- Would people use an AI-enhanced method for high-precision applications?
- What features would people like in an AI-enabled AMG simulation package/library/software?
- While exploring the design space of possible prolongation operators, how involved would people like to be ?
- Given the scale of the problems, how to present candidate matrices to the user? Would/could the user overwrite the AIs prediction?

These questions (and more) motivated a survey, whose results will be analysed to inform current and future directions. To the best of our knowledge, no questionnaire investigating the same technical issues has ever been organized on this scale. We were not able to find one with results tailored to our needs, further motivating us to carry out a questionnaire of the highest standard whose results could have substantial impact.





## Overview of our approach

Our project is method-oriented, i.e. the bulk of the time was spent developing a methodology to learn prolongation operators that would accelerate AMG. In this chapter, we describe the unsupervised learning problem, where we will, among others, present the non-trivial inputs and outputs of the model. The model is subsequently detailed in the Graph Neural Network section. Finally, we summarize our training strategy and showcase the approach underlining the survey we designed.

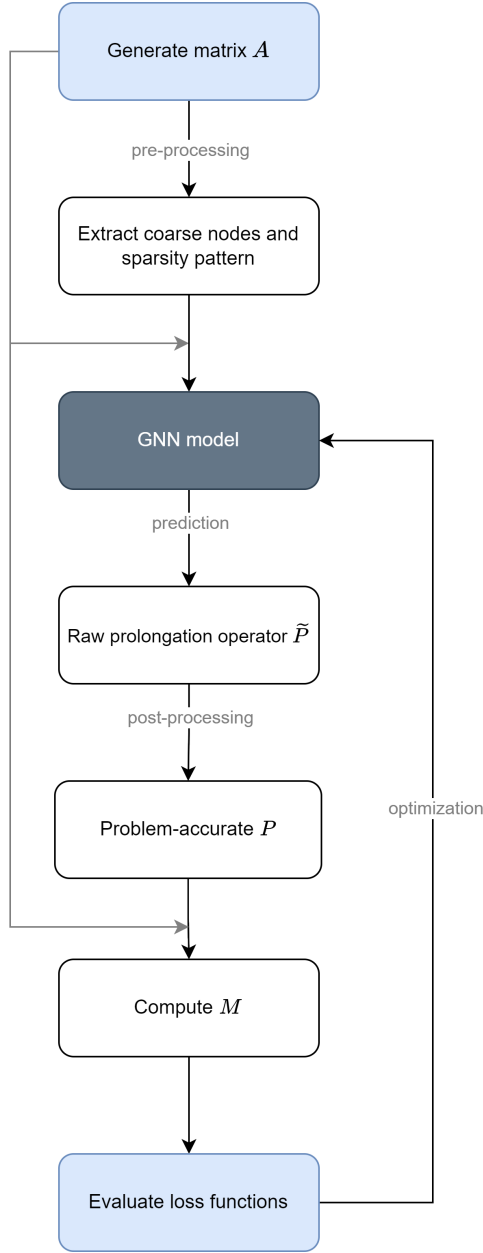
### 3.1 Unsupervised learning problem

For any AMG method, constructing the prolongation operator  $P$  entails several phases. As we saw in section 2.1.3.2 depicting classical AMG, the Ruge-Stüben coarsening algorithm selects the C- and F-points in a preliminary phase. In the subsequent phase, the weights  $w_{ij}$  (or future entries of the matrix  $P$ ) are computed. Our task is to learn these weights through a neural network. To that end, the coarse nodes will be derived from a pre-existing coarsening strategy, namely CLJP [28, 34]. Furthermore, the sparsity pattern (i.e. which coarse nodes contribute to the interpolation of a fine node) is derived from that coarsening strategy.

More formally, our task is to learn a mapping  $P = P_\theta(A)$ , where  $A$  is a sparse and large matrix,  $\theta$  the model parameters, and  $P$  the prolongation matrix. The resulting  $P$  should yield fast convergence of the AMG method, and be sparse for computational efficiency. As we saw at the end of section 2.1.3.2, this convergence is mainly governed by the spectral radius  $\rho(M)$  of the error propagation matrix  $M(A, P)$ , which we aim to minimize. This results in the unsupervised learning problem below:

$$\min_{\theta} \mathbf{E}_{A \sim \mathcal{D}} \rho(M(A, P_\theta(A))), \quad (3.1)$$

where  $\mathcal{D}$  denotes the distribution from which the entries of  $A$  are sampled (a log-normal distribution in our case). A summary of the process can be viewed at fig. 3.1.



**Figure 3.1:** Illustration of our unsupervised learning process. A particular attention was placed on the pre- and post-processing steps, which will be covered throughout this report. But mainly, the loss functions and the GNN model will be showcased in the sequel.

#### 3.1.1 Loss functions

Computing the spectral radius of  $M$  requires its eigen (or spectral) decomposition. However, backpropagation through eigen decomposition tends to be numerically unstable [8], and is a rather costly operation. Indeed, eigen decomposition can carry a complexity of about  $O(n^3)$ , which is considerably expensive – although it should be recognized that this cost is only incurred during training.

For those reasons, we borrowed an idea from Greenfeld et al. [3], and replaced the spectral radius in (3.1) with a surrogate: the Frobenius norm. In doing this, we rely on the fact that the Frobenius norm bounds the spectral radius from above, yielding a differentiable quantity without the need for expensive eigen decomposition. Aside from the Frobenius loss function  $\mathcal{L}_{\text{frob}}$ , two other function were introduced.

The second loss functions we introduced is intended to enforce physical constraints on  $P$ . Specifically, a suitable  $P$  must have the sum of its entries along the rows equal to 1. Intuitively, this means the weights contributing to the interpolation of a fine point must sum to 1 (regardless of whether that point is also a coarse point or not). Previous work in Luz et al. [5] avoided this function by dividing (or normalizing) the row entries against their sum, before using the matrix  $P$  to compute  $M$  (see eq. (2.21)). This approach results in a slow and unstable decrease in the loss function, as we found in our efforts. This observation led to us introducing our physics-informed loss function  $\mathcal{L}_{\text{norm}}$ .

Like the second loss function, the third function  $\mathcal{L}_{\text{neg}}$  enforces the idea that weights of  $P$  should all be positive. Although not strictly required by the coarsening methods, it made sense to further restraint our network's range of extrapolation in order to improve its results. All in all, the unsupervised learning problem in (3.1) becomes:

$$\min_{\theta} \mathbf{E}_{A \sim \mathcal{D}} [\mathcal{L}_{\text{frob}}(M_{\theta}(A)) + \mathcal{L}_{\text{norm}}(P_{\theta}(A)) + \mathcal{L}_{\text{neg}}(P_{\theta}(A))], \quad (3.2)$$

with:

$$\mathcal{L}_{\text{frob}}(M) = \sqrt{\sum_{i=1}^n \sum_{j=1}^n m_{i,j}^2}, \quad (3.3)$$

$$\mathcal{L}_{\text{norm}}(P) = \left\| \left( \sum_{j=1}^n p_{i,j} \right)_{i=1,\dots,n} - (1)_{i=1,\dots,n} \right\|_1, \quad (3.4)$$

$$\mathcal{L}_{\text{neg}}(P) = \sum_{\substack{i,j=1,\dots,n \\ p_{ij} < 0}} \|p_{ij}\|_1. \quad (3.5)$$

### 3.1.2 Input and output data

The matrix  $A$  can be represented as a directed graph  $\mathcal{G}_A(V, E)$ , with as many vertices (or nodes)  $v \in V$  as there are unknowns, and as many weighed edges  $e \in E$  as there are non-zeros entries (which should amount to about  $O(n)$  given our targeted interest for sparse matrices). Upon representing  $A$  as a graph, one must consider the remainder of the pre- and post-processing.

As a matter of fact,  $A$  is not the unique input to our model. As introduced earlier and notably in fig. 3.1, the input data must include the list of coarse nodes and a sparsity pattern. While it is relatively easy to extract this information from an established coarsening method, it is non-trivial to include it as input to the GNN. Here, we once again drew inspiration from Luz et al. [5] and defined a set of node features  $\{f_v\}_{v \in V}$  and a set of edge features  $\{f_e\}_{e \in E}$ . To represent node features, we used one-hot encoding to indicate whether a node is coarse or not:

$$f_v = \begin{cases} [1, 0] & \text{if } v \text{ is a C-point,} \\ [0, 1] & \text{if } v \text{ is not a C-point.} \end{cases} \quad (3.6)$$

A one-hot encoding representation also helped indicate whether an edge was part of the sparsity pattern  $E^{\text{sp}} \in E$  or not:

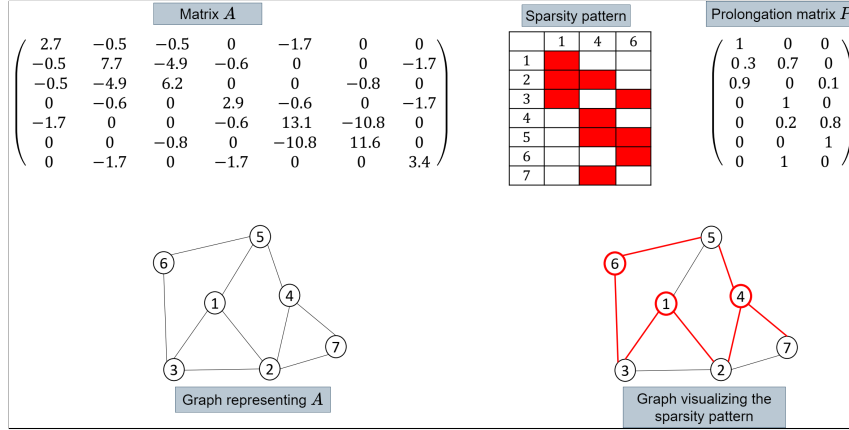
$$f_e = \begin{cases} [a_{ij}, 1, 0] & \text{if } e \in E^{\text{sp}}. \\ [a_{ij}, 0, 1] & \text{if } e \notin E^{\text{sp}}. \end{cases} \quad (3.7)$$

With the above representations, the question of predicting values for the prolongation matrix  $P$  amounts to assigning a set of values  $\{p_e\}_{e \in E^{\text{sp}}}$ . Two potential problems should be addressed here. First, it is clear that the predicted  $P_{\theta}(A)$  inherits the square shape of  $A$ , from which we must slice out only the coarse columns. At this step, it is extremely important to correctly order the coarse nodes list, without which it quickly becomes

### 3. OVERVIEW OF OUR APPROACH

easy to take one coarse column for another. Luz et al. [5] relied on MATLAB for this operation, while we found PyTorch to be miles better. Second, one can hardly stop the model from predicting entries for  $e \notin E^{\text{SP}}$ . For this reason, the original prolongation matrix (baseline  $P$ ) produced by the upstream coarsening strategy is included as part of the post-processing. In short, our predicted  $P_\theta(A)$  should only contain entries where the baseline  $P$  had non-zeros entries. All other entries are set to zero.

These operations on input and output data are illustrated in fig. 3.2. As this section demonstrates, non-trivial pre- and post-processing operations are required to perform training and inference on our model. A considerable portion of our summer project was spent on streamlining this arduous process.



**Figure 3.2:** A sparse matrix  $A$  and its associated graph  $G_A$ . The sparsity pattern defines the set of C-nodes 1; 4; 6, and to which nodes each C-node contributes (denoted by the red cells). In the graph in lower right, red edges represent the set  $E^{\text{SP}}$  corresponding to the sparsity pattern. The prolongation matrix  $P$  is formed by assigning weights to the sparsity pattern. Image adapted from [5].

## 3.2 Graph Neural Networks

Graphs are ubiquitous in the real world. They represent entities and their relationship as nodes and links respectively. In recent years, their effectiveness has been proven in many application areas among which social networks, point clouds, knowledge graphs, traffic networks, and molecular structures analysis. The research community has achieved great success on various tasks such as node classification [35, 36, 37], link prediction [38, 39, 40], and graph classification [41, 32, 42].

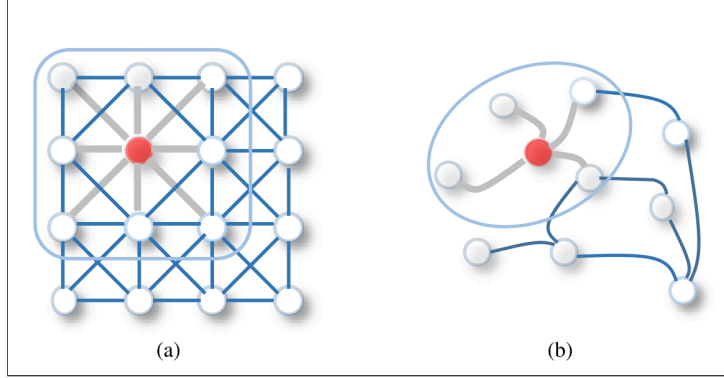
Our work focuses on unstructured grids, which are hardly representable as Euclidean<sup>1</sup> data. GNNs can handle unstructured grids by considering each cell in the discretized domain as a node. An edge is added between two nodes if the corresponding cells are adjacent. This is translated into non-zeros entries in the matrix  $A$  when assembled, from which a graph can be more accurately obtained as presented in section 3.1.2.

### 3.2.1 Background on GNNs

A Graph Neural Network (GNN) updates node, edge, and global features of a graph, layer after layer. GNNs have shown significant success in the field of graph representation learning. The central element at the heart of this success is the concept of *graph convolution*, which represents one of the most important graph operations. Contrary to Convolutional Neural Networks (CNNs) or Recursive Neural Networks (RNNs) that are biased towards knowledge from the spatial and temporal locality of data, GNNs adapt to the structure of the data [43]. Indeed, the inductive bias in GNNs can either be very strong or very weak, part of the reason for their massive success [7].

<sup>1</sup>By Euclidean data, we refer to text and images for which Deep Learning tools were primarily developed.

Similar to convolutions on Euclidean data, graph convolutions perform neighbourhood aggregation. Using this operation, many architectures of GNNs have proliferated [37, 33, 7], with the most representative being the Graph Convolutional Network (GCN) by Kipf and Welling [37]. However, one layer of these neighbourhood aggregation methods only considers immediate neighbours, with the performance decreasing when going deeper, trying to enable larger receptive fields (see fig. 3.3). Several recent studies attribute this performance deterioration to the over-smoothing issue, which states that repeated propagation makes node representations of different classes indistinguishable [44]. In our work, we avoided the over-smoothing issue by using no more than three convolution layers in our GNN architectures.



**Figure 3.3:** 2D Euclidean convolution (a) and graph convolution (b). Graph convolutions only consider immediate neighbours. When attempting to increase this receptive field, the over-smoothing issue becomes apparent. Image reprinted from [6].

### 3.2.2 Graph operations in our problem

Our main consideration when choosing a GNN architecture is whether they consider *edge features* (where the entries of  $A$  are stored). Other requirements were also taken into account, like stated in [5]: *efficiency*: runtime of the mapping from  $A$  to  $P$  should be proportional to the number of non-zero elements,  $O(n)$ ; and *flexibility*: the architecture should be able to process graphs of different sizes.

Using Deep Graph Library (DGL) and its pool of readily available convolution layers, it quickly became evident that the latter two requirements (efficiency and flexibility) would be satisfied regardless of our choice of model among their offerings. Finding a layer that efficiently made use of edge features was the limiting factor. The main graph convolutions we opted for in our study are the GraphSAGE [36], and the NNConv [32], readily available in the DGL library. GraphSAGE can be summarized with the equations below:

$$\begin{aligned} h_{\mathcal{N}(i)}^{(l+1)} &= \text{aggregate} \left( \{e_{ji} h_j^l, \forall j \in \mathcal{N}(i)\} \right), \\ h_i^{(l+1)} &= \sigma \left( W \cdot \text{concat}(h_i^l, h_{\mathcal{N}(i)}^{(l+1)}) \right), \\ h_i^{(l+1)} &= \text{norm}(h_i^{(l+1)}), \end{aligned} \quad (3.8)$$

where  $h_i$  is the feature for node  $i$ , and  $e_{ji}$  is the scalar weight on the edge from node  $j$  to node  $i$ . As for the NNConv in [32], it can be simplified with the following equation:

$$h_i^{l+1} = h_i^l + \text{aggregate} \left( \{f_{\Theta}(e_{ij}) \cdot h_j^l, j \in \mathcal{N}(i)\} \right), \quad (3.9)$$

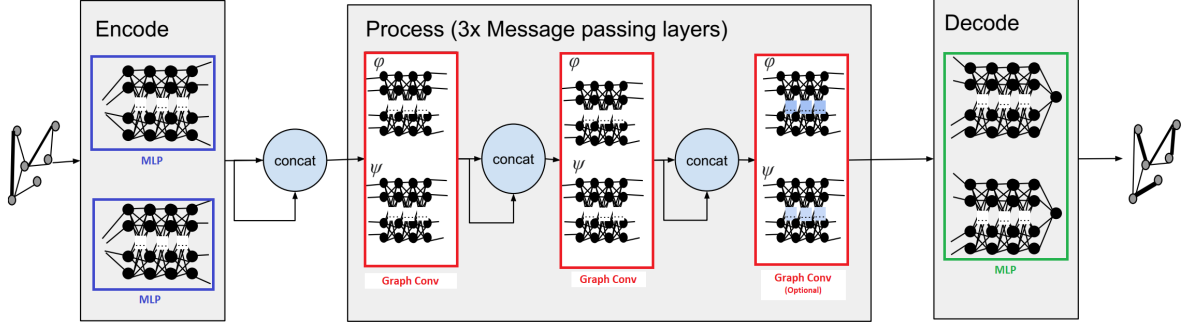
where  $f_{\Theta}$  is a function with learnable parameters.

Alongside convolutions, GNNs allow numerous other operations; quite similar to deep learning on Euclidean data. Our inputs were multiplied by linear dense layers, followed by an activation function<sup>2</sup>. The

<sup>2</sup>Here, like with graph convolutions, we used a ReLu activation function after each layer.

### 3. OVERVIEW OF OUR APPROACH

operations were chained together in an MLP to build up representations for our inputs (encoding), or extract prolongation weights (decoding). Our resulting architecture, heavily inspired from [5] and [4] is presented in fig. 3.4.



**Figure 3.4:** Encode-Process-Decode GNN architecture implemented. Two separate MLPs are used for encoding edge and node features, while only one is used for decoding. During our project, we developed two Process architectures. In the first, all three Graph Convs layers were GraphSAGE convolutions. In the second architecture, the first layer was a GraphSAGE, the second was a NNConv, and the third was omitted. Image adapted from [4].

#### 3.2.3 Deep Graph Library

The rise of deep learning on graphs was accompanied by the proliferation of packages and libraries. Over time in the scientific community, the most cited packages are Deep Graph Library (DGL), Pytorch Geometric (PyG), Graph Nets, Spektral, Dive Into Graphs (DIG), etc. (see table 3.1). Previous work on AMG relied on GraphNets [5]; but as table 3.1 and a quick look at their GitHub repositories suggest, GraphNets and the latter two are deprecated, leaving only two serious contenders: DGL and PyG. Our decision to migrate existing work and build our entire framework around DGL (data loading, model, evaluation, etc.) was motivated by several factors: active maintenance; performance; storage and memory.

Framework	Stars	Forks	Contributors
PyTorch Geometric (PyG)	14.6k	2.6k	238
Deep Graph Library (DGL)	9.6k	2.2k	195
Graph Nets	5.1k	780	10
Spektral	2.1k	287	17
Dive Into Graphs (DIG)	1.1k	155	225

**Table 3.1:** Popular graph neural network libraries written in Python, with GitHub numbers as of 18th May 2022. Based on this table and multiple other factors, we chose DGL for our implementations.

**Active maintenance of DGL.** Ever since its open-source publication in [45], DGL has seen consistent contribution from its creators and its users (see table 3.1). Moreover, the DGL Team at the origin of DGL has announced plans for further improvements. When we get stuck on a bug, we benefit from abundant help from the community.

**Better performance.** GNN models built with DGL show better inference speed when compared to models built with GraphNets and PyG [45].

**Storage and memory.** Deep Graph Library holds key optimization tactics crucial when working with large matrices. Indeed, the size of the matrices we consider often prevents fitting into memory. Given that sparse operations are not fully supported both in TensorFlow and PyTorch, one has to rely on dense representations to perform inversion, block-diagonalisation and matrix norms. Specifically, the following memory management points made DGL the unequivocal choice for our task:

- DGL has a much better memory management due to its usage of *fused* message passing kernels [4].
- It allows *half-precision* (FP-16). Although half-precision is not fully supported, and as of this writing, is only present in development branches.
- While most frameworks allow distributed *training* (both data- and model-parallel), only a few allow for distributed *inference*<sup>3</sup>.

Our work with DGL started after taking the widely popular Blitz Introduction to DGL<sup>4</sup>. Officially, DGL relies on a traditional deep learning backend. They offer three choices: PyTorch, TensorFlow, and MXNet. The Blitz tutorial and a preliminary investigation quickly showed that PyTorch was the only viable choice. Not only most tutorials are taught with PyTorch, but also, some crucial features are only implemented in PyTorch (e.g. half-precision). Our entire codebase (including the tutorials we completed), is available on our public [GitHub](#) repository.

### 3.3 Training strategy and resources

The network is trained to minimize the error propagation matrix of the two-level multigrid operator. We essentially implemented the same three-stage training strategy as [5]. In the **first stage**, the GNN was trained on 256000 randomly generated problems of size  $n = 64$ , in a single epoch<sup>5</sup>. For memory efficiency, Luz et al. [5] performed 1000 runs of 256 problems each. Instead, we opted for 100 runs of 2560 problems each. Along with the new loss functions in section 3.1.1, this sizing tactic contributed to the stability of the training process. In the **second stage**, we generated 128000 problems of size  $n = 128$  and we applied the trained network to generated prolongation operators for each of those problems. We then computed the corresponding coarse matrices  $A_c = P^T A_f P$ . This choice of  $n$  is motivated by the fact that the CLJP coarsening algorithm selects roughly half of the nodes. This way, the coarsened problems are approximately the same size as the original problems. The **final stage** was to generate 128000 additional problems of size 64, shuffle them with the coarsened problems, and continue training the network on the combined set of 256000 problems for another epoch.

All our experiments were conducted within the PyTorch framework [46] using an NVIDIA T4 GPU on AWS. We used a batch size of 32 and picked the Adam optimizer [47] with the dynamic learning rate scheduler ReduceLROnPlateau. Starting at  $3 \times 10^{-3}$ , it reduced the learning rate by multiplying it with a decay factor  $\gamma = 0.95$  if we noticed that our loss value hadn't decreased over the previous 100 evaluations. A complete training took about 10 hours, amounting to about 24000 loss function evaluations and backpropagation steps.

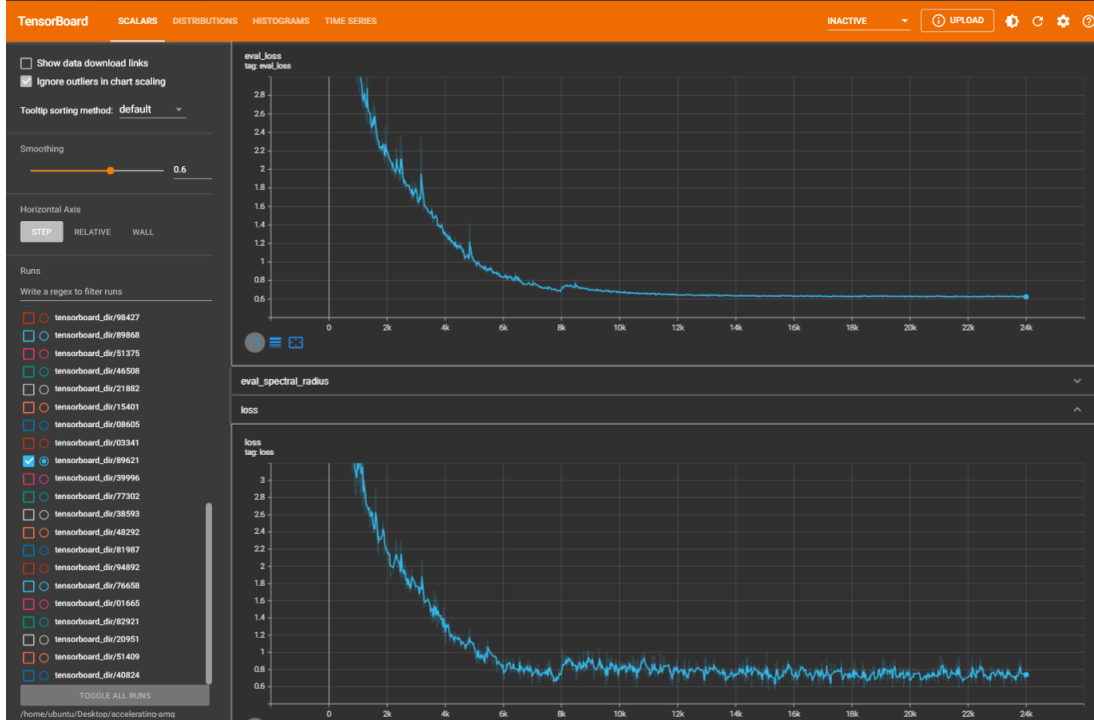
<sup>3</sup>We only focused on single-node training and prediction for the summer project. Multi-node will be investigated as part of future work.

<sup>4</sup><https://docs.dgl.ai/tutorials/blitz/index.html>

<sup>5</sup>Here is how we employ the terms epoch, run, and batch: within an *epoch*, several *runs* can be made. A fresh set of problems is generated at the start of each run. Within a run, *batches* are used.



### 3. OVERVIEW OF OUR APPROACH



**Figure 3.5:** Decrease in loss function during training (loss) and validation (eval\_loss) for one of our held models (named 89621, with 3 GraphSAGE convolutions). Other than the loss function, several other quantities were measured and visualized with TensorBoard.

Training models for such large matrices as long as we did requires specialized tools and resources. Moreover, our problem is one with multiple requirements in terms of software, libraries, and packages. In fact, just launching a training loop and making a prediction was one of the early main milestones of the project. It turned out to be one of our major achievements when we succeeded at said task.

#### 3.3.1 Block-diagonalisation

Training by diagonalization of block-circulant matrices is a radically new approach introduced by Greenfield et al. [3] to mitigate the cost of computing the error propagation matrix and its Frobenius norm. As indicated in eq. (2.21) as follows,

$$M = S^{\sigma_2} \left( I - P \left[ P^T A P \right]^{-1} P^T A \right) S^{\sigma_1}, \quad (2.21)$$

computing  $M$  requires inverting  $A_c = P^T A P$ , which can be prohibitively expensive. Greenfield et al. [3] found that if  $A$  was block-circulant, then  $M$  could inherit those properties, assuming  $A$  could be unitarily block-diagonalised by an appropriate Fourier basis. After this, operating (computing of norm, inverting, etc.) on a block-circulant matrix boils down to operating on each of the much smaller chunks of its block-diagonal version.

Luz et al. [5] also relied on block-circulant matrices. For instance, their matrices in the first stage of the training strategies were composed of  $b = 16$  unique square blocks of size  $k = 4$  (hence the relation  $n = kb = 64$ ). Since  $A$  itself must be block-circulant, the value of  $k$  and  $b$  must be provided as part of the synthetic data creation process. Moreover, imposing a block-circulant structure on  $A$  entails a periodic structure too. This whole process is further complicated by the fact that  $A$  must be created with its graph Laplacian in mind. These intricacies led us to embrace [5]’s MATLAB implementation within Python. For that, we dedicated two weeks early in the project to follow two courses on using MATLAB: A Beginner’s Guide to Numerical Methods in MATLAB, and MATLAB Master Class: Go from Beginner to Expert in MATLAB.

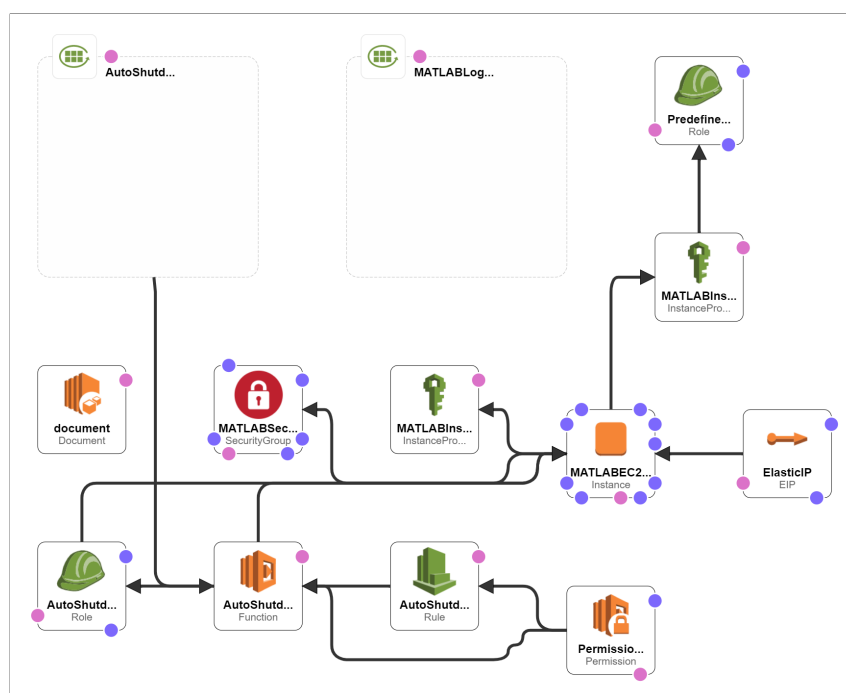


That said, we acknowledge that clarifying this and streamlining the dataset creation and diagonalization processes should be a priority. Despite spending much time trying to train on block-diagonalised matrices, we consistently failed to see any decrease in loss function; causing us to abandon this path entirely.

### 3.3.2 Amazon Web Services

The code we inherited from [5] was outdated and difficult to set up. The long list of requirements is displayed on their repository<sup>6</sup>. The difficulty in setting up these requirements was only exacerbated by the code’s reliability on MATLAB and its official interface for Python: MATLAB Engine. Installing MATLAB Engine on any facility requires admin privileges, which naturally comes with security concerns. Moreover, careful precautions are needed to install the correct and compatible version with Python<sup>7</sup>, while addressing the NumPy array optimization trick used by [5]<sup>8</sup>.

These difficulties in setting up the working pipeline lead us to consider AWS, and its CloudFormation stack for MATLAB. Upon setting up the stack, we were able to train for considerably longer, and test for larger matrices. The architecture we set up is presented in fig. 3.6, and the code taking advantage of this hardware is available on our GitHub repository<sup>9</sup>.



**Figure 3.6:** Visualization of our AWS Cloud Formation stack. MATLAB access is provided through a UoB academic licence. The GPU powering the EC2 instance is one of g4dn.xlarge, an NVIDIA T4. Image obtained through AWS's CCloud Formation template designer.

### 3.4 Method behind the survey

Our survey was design to collect ideas, opinions, and feedback from the community<sup>10</sup>. In total, it contained 42 questions divided in 5 sections, although some could be avoided through branching. We relied on *open-ended*

<sup>6</sup><https://github.com/ilayluz/learning-amg>

<sup>7</sup>Versions of Python Compatible with MATLAB Products by Release

<sup>8</sup><https://stackoverflow.com/a/45290997>

<sup>9</sup><https://github.com/desmond-rn/accelerating-amg>

<sup>10</sup><https://forms.office.com/r/qpbCvqfNY3>

### 3. OVERVIEW OF OUR APPROACH

---

questions (40%) so that participants would express themselves unencumbered. Participants were encouraged to answer questions in English, although those in French were also considered. Participants were also presented with *multiple-choices* (42%) and *ratings* question (14%) to express their level of approval of our suggestions. A screenshot of the survey is presented in fig. 3.7.

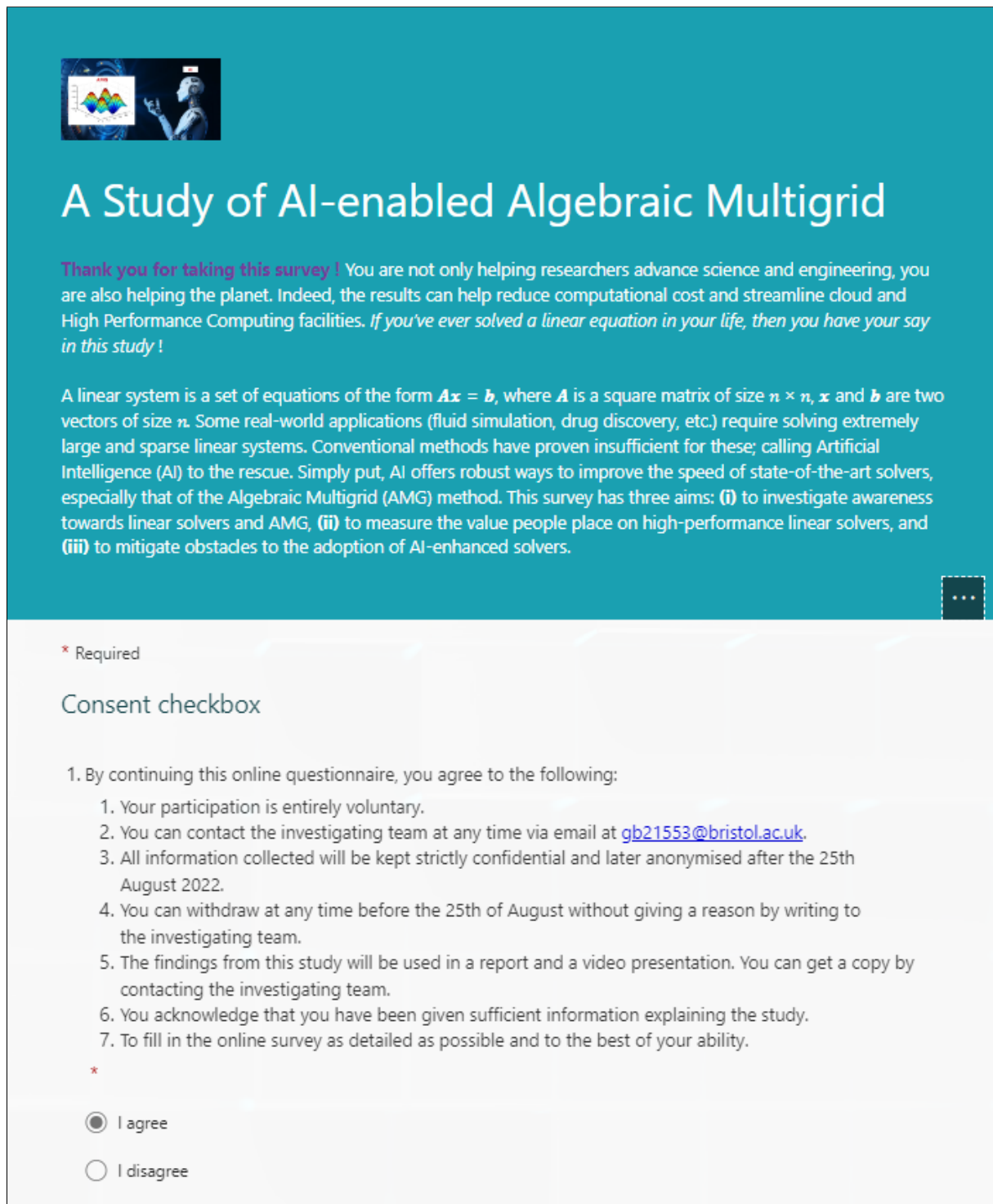
A statistical analysis using ANOVA and  $\chi$ -squared independence tests was originally planned, but later cancelled. Instead, we adjusted the survey so that a robust *descriptive* analysis could be carried. Parallel to that, the open-ended questions were designed to be probed via *thematic* coding and analysis, much like a transcribed audio or video survey.

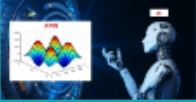
In terms of resources used to develop and analyse the survey, we mostly relied on the Office 365 platform as offered by the University of Bristol. Namely, we used MS Word to write draft questionnaires for our Ethics application, then MS Forms to format and serve the survey. We recruited participants in LinkedIn's HPC and ML communities and decided to test out the SurveyCircle<sup>11</sup> tool (although no additional data came out of it). MS Forms provided charts for easy descriptive statistics. Also, a readily available word-cloud feature considerably helped our thematic analysis.

Obtaining the Ethics approval for the survey was an uphill battle. Starting the process on 1st August, it took us 13 days to be granted permission to launch the survey. We had planned to recruit participants in several stages. However, in light of the volume of the information to analyse and the limited time left for the write-up, we stopped after the first phase, upon collecting 18 detailed responses (more than our 15 minimum).

---

<sup>11</sup><https://www.surveycircle.com/en/>





## A Study of AI-enabled Algebraic Multigrid

**Thank you for taking this survey !** You are not only helping researchers advance science and engineering, you are also helping the planet. Indeed, the results can help reduce computational cost and streamline cloud and High Performance Computing facilities. *If you've ever solved a linear equation in your life, then you have your say in this study !*

A linear system is a set of equations of the form  $A\mathbf{x} = \mathbf{b}$ , where  $A$  is a square matrix of size  $n \times n$ ,  $\mathbf{x}$  and  $\mathbf{b}$  are two vectors of size  $n$ . Some real-world applications (fluid simulation, drug discovery, etc.) require solving extremely large and sparse linear systems. Conventional methods have proven insufficient for these; calling Artificial Intelligence (AI) to the rescue. Simply put, AI offers robust ways to improve the speed of state-of-the-art solvers, especially that of the Algebraic Multigrid (AMG) method. This survey has three aims: **(i)** to investigate awareness towards linear solvers and AMG, **(ii)** to measure the value people place on high-performance linear solvers, and **(iii)** to mitigate obstacles to the adoption of AI-enhanced solvers.

\* Required

### Consent checkbox

1. By continuing this online questionnaire, you agree to the following:

1. Your participation is entirely voluntary.
2. You can contact the investigating team at any time via email at [gb21553@bristol.ac.uk](mailto:gb21553@bristol.ac.uk).
3. All information collected will be kept strictly confidential and later anonymised after the 25th August 2022.
4. You can withdraw at any time before the 25th of August without giving a reason by writing to the investigating team.
5. The findings from this study will be used in a report and a video presentation. You can get a copy by contacting the investigating team.
6. You acknowledge that you have been given sufficient information explaining the study.
7. To fill in the online survey as detailed as possible and to the best of your ability.

\*

☒ I agree

☐ I disagree

**Figure 3.7:** Screenshot of the introductory section of our survey. In total, it contained 5 sections, of which 4 are not displayed here. The survey can be taken at [this link](#).



## Results, discussion, and challenges

In this section, we will present and compare our results while providing a critical analysis, both for the AI-enabled AMG and the user survey. We will then highlight the major challenges we faced and solutions we see moving forward.

### 4.1 Performance of AI-enabled AMG

Following the methodology we developed in chapter 3, we trained several GNNs, with the most representative being the architecture depicted in fig. 3.4 that resulted in the training and validation loss values at fig. 3.5. Before we can present the results we obtained when evaluating the model, let's summarize the basis for comparing it with classical AMG.

#### 4.1.1 Description of the experiment

Our main evaluation strategy matched the Graph Laplacian strategy as described in [5]. To generate a test point  $A$ , we sampled points uniformly on the unit square, and computed a Delaunay triangulation. Each edge was then given a random weight sampled from a standard log-normal distribution, and the corresponding graph Laplacian matrix was constructed. The right-hand side, non-influential in the overall convergence rate of the method, was set to  $\mathbf{b} = 0$  for all data points. With an initial guess  $\mathbf{x}_0$  randomly sampled from a Normal distribution, successive iterations should yield  $\mathbf{x} = 0$  as a solution. For each homogenous problem  $A\mathbf{x} = 0$ , we built a hierarchy of at most 12 levels, each time relying on our model to provide a matrix to move from one level to the next.

For our AI-enabled AMG and for classical AMG, we performed 80 AMG iterations, each time collecting the *residual factor*, i.e. the ratio of the residual norms of the last two iterations  $\frac{\|\mathbf{r}_{k+1}\|}{\|\mathbf{r}_k\|}$ . To avoid epsilon-zero and floating-point errors, we only considered residuals  $\mathbf{r}_k$  such that  $\|\mathbf{r}_k\| \geq 10^{-12}$ . We applied this to problems of size ranging from 1024 to 32768. For each problem size, the described experiment was repeated 100 times, with a different matrix  $A$  at each run. The success rate of our method was given by the number of times our residual factor after 80 AMG iterations was smaller than the residual factor for classical AMG. This whole experiment was conducted both for V- and W-cycles alike (see table 4.1).

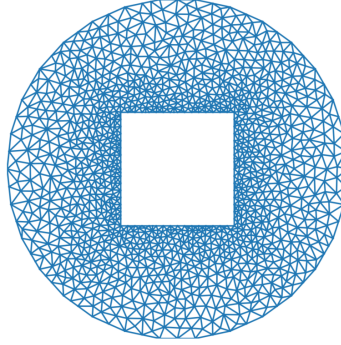
Like Luz et al. [5], we also tested our model for generalization<sup>1</sup> on a complex 2D FEM problem. We

<sup>1</sup>Our network was not only trained on graph Laplacian problems. Testing on 2D FEM problems is a stepping stone towards 3D Navier-Stokes equations.

generated matrix  $A$  and vector  $\mathbf{b}$  corresponding to the Poisson diffusion PDE with Dirichlet boundary conditions:

$$\begin{cases} -\nabla \cdot (\mathbf{g} \nabla \mathbf{u}) = \mathbf{f} & \text{in } \Omega \\ \mathbf{u} = 0 & \text{on } \partial\Omega \end{cases} \quad (4.1)$$

where the diffusion coefficients  $g_i$  were sampled from a log-normal distribution with a log-mean of 0 and a log-standard deviation of 0.5. The domain we considered for this was also inspired by [5]: a circular domain with a square hole discretized using a triangular mesh (see fig. 4.1).



**Figure 4.1:** Circular domain with a square hole used for testing the generalization of our method on a diffusion problem using FEM. Image reprinted from [5].

#### 4.1.2 Results and discussion

We ran our experiments for Graph Laplacian problem and 2D FEM with a varying degree of success. The success rates were compiled and presented in table 4.1.

Problem size	Graph Laplacian		2D FEM	
	V-cycle	W-cycle	V-cycle	W-cycle
1024	3%	3%	4%	4%
2048	9%	10%	8%	12%
4096	15%	13%	10%	11%
8192	18%	23%	16%	19%
16384	21%	16%	15%	14%
32768	27%	25%	16%	23%
65536	-	-	-	-
131072	-	-	-	-
262144	-	-	-	-
400000	-	-	-	-

**Table 4.1:** Results obtained when evaluating our three-layered GraphSAGE GNN model. These indicate that in about 20% of problems, our model outperforms classical AMG. Results for sizes greater than 65536 were not obtained, (probably) due to insufficient memory on AWS.

We can see from table 4.1 that our model hardly outperformed AMG, with its success rate peaking at 27%. That said, the success rate tended to increase with matrix size, which is a positive sign. As table 4.1 points out, we were unable to test larger matrices on AWS, despite maintaining our matrices in CSR format throughout the experiments. When we tried running experiments with  $n = 65536$  for example, the process

simply stopped, without outputting any insufficient RAM or VRAM memory error, nor runtime error. That said, we suspect that increasing the specifications of our EC2 instance on AWS<sup>2</sup> should solve this issue.

Louw and McIntosh-Smith [4] reported similar results during testing. When they applied the Luz et al. [5] network to one of their large matrices built from Thingi10K, they reported only a 1.2% success rate. Given that we trained and tested with the same synthetic dataset as [5], we expected results similar to theirs (see table 4.2).

Problem size	Graph Laplacian		2D FEM	
	V-cycle	W-cycle	V-cycle	W-cycle
1024	97%	83%	87%	88%
2048	98%	91 %	94%	85%
4096	98%	91%	99%	84%
8192	99%	84%	99%	90%
16384	99%	79%	96%	88%
32768	98%	78%	96%	96%
65536	100%	79%	98%	87%
131072	100%	76%	96%	94%
262144	100%	83%	97%	77%
400000	98%	82%	96%	89%

**Table 4.2:** Results obtained by Luz et al. [5] when evaluating their Encode-Process-Decode GNN model.

The limited success rate we observed can be attributed to several factors. First, the fact that our GNN convolutional layers were not suitable and didn't make good use of edge features. This marks one of the major points of improvements in future work, and is further detailed in the section 4.3 below. Also, we faced the problem of normalizing the rows of  $P$  before computing  $M$  and back-propagating, as it resulted in slow and unstable decrease in the loss function.

Another interesting result we observed was that indeed for W-cycles, the convergence rate we used to compute the success rate is almost equal to the spectral radius of the error iteration matrix  $M$  (which was fully computed during testing for visualization purposes). This observation fully aligns with those from [19] and [5].

## 4.2 Insights from the user survey

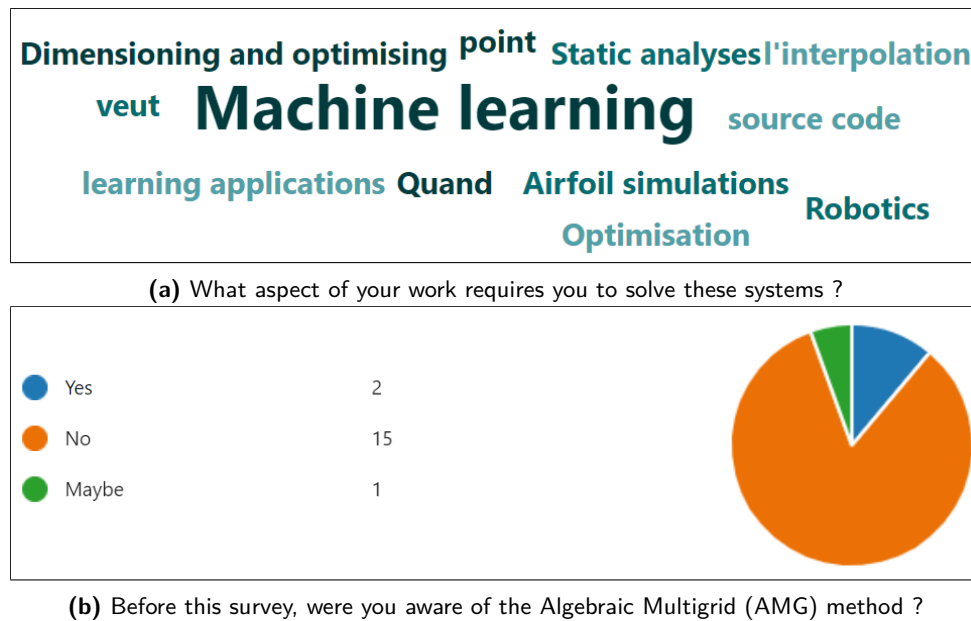
Our survey delivered both expected and surprising results. The first observation we made is that it took a participant 24:08 minutes on average to complete the 5 sections. While it took some only 7:16 minutes, it took others as much as 134:28 minutes, as they had to research some technical terms used in the question statements. Given that we had planned for 10 minutes, we realize that a more targeted approach should be prioritized for future investigations.

### 4.2.1 Awareness of AMG

In the first section of our survey, we investigated awareness of linear solving techniques. Starting with a few demographic questions, it was clear that our audience was mostly made of engineers and research scientists (61%). As the word cloud in fig. 4.2a suggests, Machine Learning was the primary work aspect that required solving linear systems. Our participants stressed that the systems they encountered were not particularly large

<sup>2</sup>Although not originally planned, the AWS architecture is one of the most important deliverables of this project. It took much time and attempts to get right.

nor sparse. Indeed, More than 44% indicated that their systems had less than 1000 unknowns, and more than 89% indicated a sparsity of more than 1%. This is probably the reason they majoritarilly opted for direct solving as their method of choice, despite the drawbacks we highlighted in chapter 2. Unsurprisingly, only 2 out of 18 fully knew about AMG (see fig. 4.2b), as they learned it as part of their research. This confirms our hypothesis when building this survey (see section 2.3). Its confirmation with these observations suggests that particular care should be placed on preparation of the material and on the approach when disseminating our research.



**Figure 4.2:** Descriptive and thematic results about awareness of linear solvers.

### 4.2.2 Features for successful AMG

The next section of our survey focused on market study to find which features users valued most. In this more technical section, we started by checking how people handled systems too large for HPC resources at hand. This was non-conclusive, which wasn't alarming, considering that our participants faced relatively small problems, and thus had the resources to solve them. We proceeded to explain what AMG could do: save HPC resources and speed up solving time. When asked what were their biggest concerns and what would it take for them to switch to AMG, one participant wrote:

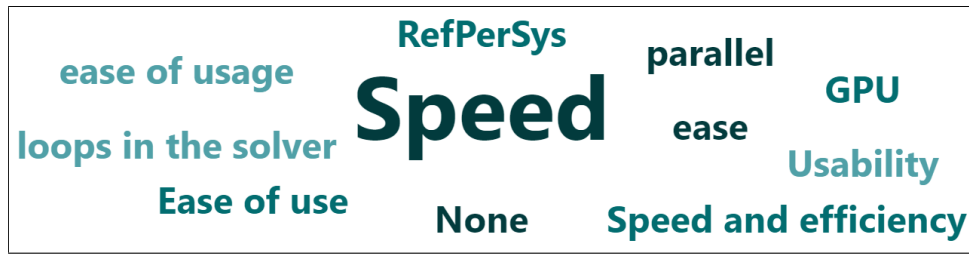
*Depends on how many unknown variables my system has. If it has more than 10000 unknown variables, then I think it will be worth it for me to replace my LU solver with AMG.*

This response embodies the general attitude that transpired throughout the survey.

In the second section, we also enquired on which format would users prefer for AMG: they responded with Python package (40 %), a C/C++ software (33%) and a fully fledge software with GUI (13%); although some users proposed High Performance languages like Symphony<sup>3</sup> that we hadn't anticipated. We interpret this as: no format/language should be neglected; we will keep using Python for rapid prototyping, and then explore portability to other formats based on the specific needs of our fluid simulation application domain. The section ended with an open-ended question asking participants to cite what other features they could consider. As highlighted by the word cloud in fig. 4.3, "speed" was the theme that overshadowed the rest (ease of use, usability, etc.).

<sup>3</sup><https://symfony.com/>





**Figure 4.3:** Word cloud for the responses to “What other features do you consider when choosing a solver ?”.

The interest people showed in “speed” alongside “accuracy” is one of the most impactful findings in our survey. It helped design future experiments to test our AI-enabled methodology. Contrary to the process described in section 4.1.1, we plan on testing whether our algorithm runs faster than the classical AMG, while ensuring that the residual errors after 80 iterations all fall within a specified ballpark. This way, we will have an experiment that takes into account the accuracy of the methods and their direct execution times.

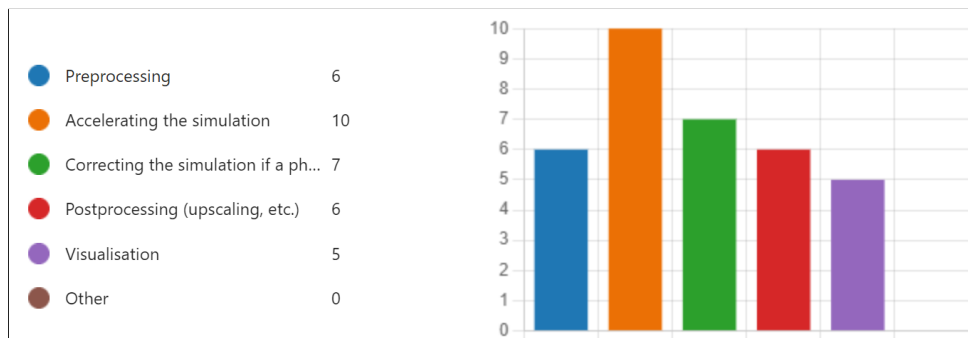
#### 4.2.3 Adoption of AI-enabled solvers

The last section of our survey focused on the use of AI in accuracy-critical applications like ours. Participants were emphatic in that they wanted accuracy out of such our system; even more than speed, as promoted in the non-AI-enabled AMG case above. many users also proposed *community support* as key to their decision to embrace an AI-enabled method. This result tells us that community outreach skills and strategies should be developed in tandem with our core findings.

We carried out a conjoint analysis, trying to pinpoint which features users would trade for which. It was evident from the responses that such trade-offs depends on the particular problem at hand. This was a clear indication that our question lacked precision. This will be corrected in future versions of the survey.

Given that most users hardly understood how prolongation and restriction operators worked in AMG, we asked them to imagine the opposite (that they were in fact experts in the field). It then came as no small surprise that people were rather inclined to somehow alter the AI’s prediction for  $P$ . Only 11% of participants said they would fully trust the AI, assuming it had proven accuracy. These questions should be further investigated, this time with actual AMG experts.

The final part of this section investigated the more general use of AI in simulations. Participants expressed interest in using AI in virtually every stage of the simulation pipeline: preprocessing, accelerating, correcting, post-processing, visualization, etc. (see fig. 4.4). Similarly, participants agreed (with more than 75%) that they would be more comfortable using a hybrid AI+physics+human system. When asked to elaborate on their answer on this, participants mentioned terms like *reduced basis approximation*, *looping within solvers*, and *differentiable physics*; terms that will inform our future research directions.



**Figure 4.4:** Histogram for the responses to “What part of the simulation pipeline would you use AI in ?”.

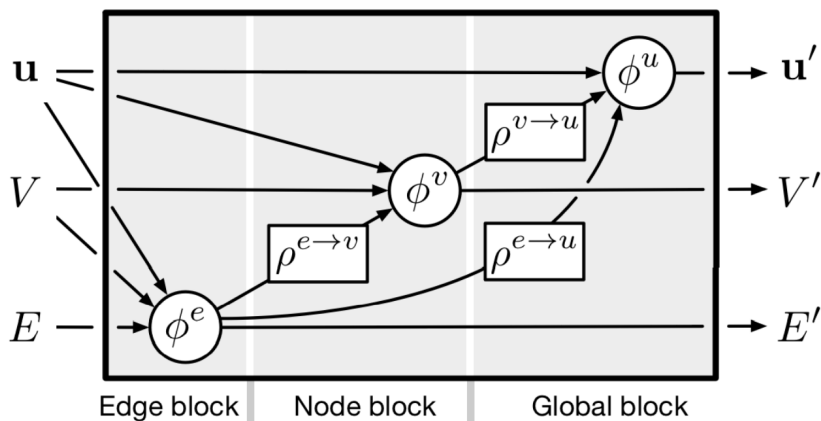
Furthermore, participants responded well (more than 75%) to *design space exploration* by shape optimization as an interactive approach they would be interested in. At the end of the survey, users indicated that given what they now knew on AMG, they would consider using it for their projects. This suggests that our survey had a positive impact, without even presenting any definitive AI/HPC/AMG result. Future surveys and interviews will focus on people that face larger problems with scarce HPC resources. Our recruitment strategy will thus be more targeted.

### 4.3 Challenges

We will now detail the major challenges we faced in our efforts to interactively build a method to accelerate AMG. Some of these challenges were: GNN architectures able to better handle edge features were not implemented in DGL; the learning curve we faced for many libraries (MATLAB, PyAMG, etc.); navigating the over-modularised code from [5] (likely the reason block-diagonalisation was unfruitful), and managing the UoB Ethics Board for approval to launch our survey.

#### 4.3.1 Making use of edge features in GNNs

One of the main problems we faced was choosing a GNN. As stated in 3.2.1, three considerations came up: (1) edge features, (2) efficiency (or scalability), and (3) flexibility. The Battaglia et al. [7] framework<sup>4</sup> clearly fits this description by separating parametrized update functions for edges and nodes (see fig. 4.5); which justified their usage in previous work by [5] and [4]. However, to the best of our knowledge, the Graph Network (GN) from [7] is not implemented in DGL; despite DGL having numerous readily available graph convolutional layers.



**Figure 4.5:** Full GN block as described by Battaglia et al. [7]. It defines parametrized functions for updating vertices  $V$ , edges  $E$ , and the global node  $u$  independently. This framework is missing in DGL, which is why we plan on implementing it for our work. Image reprinted from [7].

With the absence of the GN block [7], we compromised by using the readily available GraphSAGE [36] and NNConv [32] convolutions. We found that DGL’s implementation of NNConv<sup>5</sup> is not as explained in the corresponding paper, further incentive to switch to [7]. That said, during the project, we didn’t have enough expertise to exert granular control over DGL’s convolution modules. As fixing this will likely “fix” our results, it is the priority moving forward; along with building a convenient NNConv (which should be much easier, and serve as a baseline GNN model).

<sup>4</sup>This paper was published to the wider community with the GraphNets package, now deprecated.

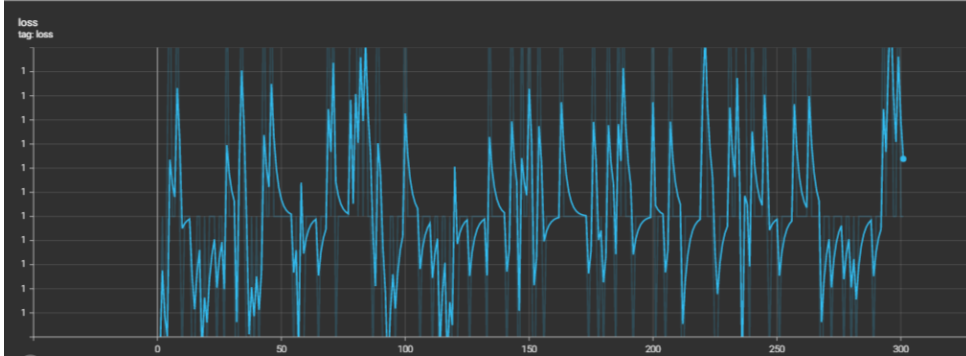
<sup>5</sup><https://docs.dgl.ai/en/0.8.x/generated/dgl.nn.pytorch.conv.NNConv.html>

Besides the GraphSAGE and NNConv, a recent architecture that meets all our need, and that appears to be fully supported in DGL is the Equivariant Graph Neural Networks (EGNN) [48]. EGNN is heavily inspired by NNConv. More than equivariance to permutations, EGNN enforces spatial equivariance (rotations, translations, etc.) on graphs, which requires spacial coordinates of each node (to be provided as part of the inputs). However, this requirement is hard to satisfy, especially considering that we are focusing on unstructured grids and Algebraic Multigrid. Nevertheless, the code for EGNN is openly available in DGL and could easily be moulded into a satisfying NNConv layer.

These difficulties in failing to appropriately make use of edge features should explain the unsatisfactory quality of our results. We believe that adapting the EGNN to build a NNConv layer that updates edge features as frequently as node features should considerably improve the results. More importantly, coding our model from scratch following the framework in [7] should fully reproduce the architecture in [5] and their results, which will then be scaled to larger matrices through data- and model- parallelism.

#### 4.3.2 Minimizing the spectral radius

Minimizing the spectral radius was possible on a desktop PC, but unsuccessful on AWS. As conveyed in fig. 4.6, its value was stuck around 1. We need to figure out why this is happening. AWS was chosen for its ease to set up and credit incentives provided by the University of Bristol. Once the model has shown sufficient promise, we plan to transition away from AWS, and rely on UoB’s HPC facilities for training and testing. But that only works if we do not rely on MATLAB for block-diagonalisation.



**Figure 4.6:** Spectral radius loss function constantly stuck around 1 on AWS. Faced with this difficulty, it is important to consider more stable backpropagation strategies through eigen-decomposition in the future, as suggested in Wang et al. [8].

In our investigation, we found that Luz et al. [5] relied on MATLAB Engine for Python (MATLAB’s official interface to Python) for too many tasks (matrix construction, diagonalization, padding, selection of coarse nodes, etc.); even when quicker options were available. We improved those aspects with PyTorch’s illustrious library of functions, both for dense and sparse functions. The only aspect we retained is the creation of block-circulant matrices and their corresponding diagonalization using Matlab Engine. We feel that this is vital for reproducibility and will only be further investigated when porting the work to UoB’s HPC.

#### 4.3.3 Testing on large matrices

Ensuring that  $A$  was block-diagonalisable forced training on a very limited class of matrices: those with random weights sampled from a log-normal distribution, arranged in such a way that graph Laplacian patterns would repeat in different chunks of the graph. Given that we are mostly interested in problems arising from fluid simulations, it could have been more helpful to train on more specific datasets.

Additionally, it was not possible to test large matrices, especially those from the Thingi10K dataset (where  $n$  could have millions of rows). As showcased in table 4.1, we could not even test matrices larger than  $n =$

65536. Several reasons might explain this: (1) we chose a weak GPU on AWS to save cost during the prototyping phase, (2) our model is very heavy and needs to be simplified by using less learnable parameters, and fewer layers in our MLPs (currently containing 4 layers each). Optimization with half-precision and distributed inference could help alleviate some of these issues.

### 4.3.4 Other challenges

As stated in section 4.1.2, Louw and McIntosh-Smith [4] reported similar results when applying the Luz et al. [5] framework, obtaining about 1.2% success rate<sup>6</sup>. This was the motivation to first achieve good results on the original dataset. Despite our numerous attempts at running the code from [5], we found errors after errors; even when downgrading our packages to fit the ones highlighted in the README<sup>7</sup> file. This problem was only worsened by the lack of a `requirements.txt` file to recreate a suitable Python environment for reproducibility. That said, running their code and obtaining the results they reported is still a goal, a stepping stone towards the larger objectives presented in chapter 1.

One other issue that was raised during this project is that of time-management. We had to adjust our plans and expectations at least twice. First, the original timeline was quite aggressive and was quickly shifted to focus on AMG first, and AMG only. The second time we had to adjust was 1 month into the project, to redefine the interactive component. We ended up designing a survey, for which we had little experience, and thus had to set aside time to learn and master the art of crafting effective technical surveys. This meant less time for coding, tweaking, and evaluating our GNN model.

---

<sup>6</sup>Although it should be noted that this test was done on a completely different dataset.

<sup>7</sup><https://github.com/ilayluz/learning-amg>

---

## Conclusion

### 5.1 Summary

In this work, we tested the hypothesis that AI-augmented AMG could outperform its classical counterpart. To this end, we developed a methodology around DGL and PyTorch, and performed experiments on AWS, all the while considering the needs of the end users. In the end, we showed that AI-enabled AMG was superior in about 20% of cases. Taken together with the impressive work that preceded ours, we can say that our hypothesis was confirmed.

The main contributions of the dissertation thesis are as follows:

1. A GNN methodology<sup>1</sup> for learning prolongation operators built around Deep Graph Library and PyTorch. Our GNN model is based on [5] and includes: encoding, message-passing, and decoding steps. It is trained with an unsupervised objective function.
2. A Cloud Formation stack on Amazon Web Services to run experiments, containing all the dependencies our methodology requires. Built from a MATLAB template<sup>2</sup>, this stack will be maintained for future experiments.
3. Experiments indicating that the current version of our AI-enabled AMG method is better than the classical approach in about 20% of cases, for both V- and W-cycles, for graph Laplacian and 2D diffusion problems.
4. A user survey with insights from the community that stands to benefit from our research. At the end of it, participants expressed increased interest in AI-enabled AMG. The survey also stressed the importance for careful approach to disseminating our findings, and informed future experiments and research directions.

Throughout our work, we faced a number of challenges. The main one being the lack of a readily available Encode-Process-Decode GNN architecture in DGL. Additionally, we observed inconsistencies when back-propagating directly through the spectral radius. Also, we were unable to test certain sizes of matrices, despite developing our code around CSR matrices. Finally, we faced time-management issues, especially while including the survey, and had to readjust priorities during the project.

---

<sup>1</sup><https://github.com/desmond-rn/accelerating-amg>

<sup>2</sup><https://github.com/mathworks-ref-arch/matlab-on-aws>

### 5.2 Future Work

We’ve learned a lot during this project. Our encouraging findings suggest we can improve our success rate, and include Machine Learning in various other stages of the fluid simulation pipeline: pre- / post-processing, simulations, error projection, solution upscaling. This said, in the immediate future, we will design an Encode-Process-Decode architecture from scratch using basic DGL building blocks, by drawing inspiration from the code of other modules. This should yield much better results than those presented in table 4.1. Once this is done, we plan to simplify our methodology by using AI to predict the coarse nodes and the sparsity pattern, hence not requiring a pre-existing coarsening strategy to supplement our approach.

The overarching idea behind this project has always been Design Space Exploration, where humans and machines would compete and collaborate to produce the optimal set of parameters. In the long run, one (or all) of the following ideas will be explored.

**Design space exploration of the prolongation matrix  $P$ .** This means a user could theoretically choose to alter the AI’s prediction of  $P$ , by making use of the physical meaning behind  $P$  (neighbours that bear a strong *physical* connection to a node should have large weights). This appears to be extremely difficult, especially given the size of the  $P$ . Our survey helped to address this question, including how to visualize these quantities.

**Design space exploration in terms of hyperparameter optimization.** Here, we would systematically find the best hyperparameters for our GNN model, enabling easy model reusability across different fluid simulation families of PDEs. Explainable Bayesian optimization would help analyse parameters and features in real time, and explain their contribution to each user decision with comprehensible graphics [49].

**Design space exploration in terms of shape optimization.** This applies to specific designs used in aerospace, namely gas-turbines. Here, recent tools like physics-informed ML and differentiable simulation together with automatic differentiation would help draw a direct connection between the engine’s physical parameters and the objective function, thus allowing for better and faster mechanical designs.

---

## Bibliography

- [1] R. Robey and Y. Zamora, *Parallel and high performance computing*. Simon and Schuster, 2021.
- [2] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A multigrid tutorial*. SIAM, 2000.
- [3] 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.
- [4] T. Louw and S. McIntosh-Smith, “Applying recent machine learning approaches to accelerate the algebraic multigrid method for fluid simulations,” in *Smoky Mountains Computational Sciences and Engineering Conference*. Springer, 2021, pp. 40–57.
- [5] 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.
- [6] 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*, vol. 32, no. 1, pp. 4–24, 2020.
- [7] 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.
- [8] W. Wang, Z. Dang, Y. Hu, P. Fua, and M. Salzmann, “Backpropagation-friendly eigendecomposition,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [9] W. Nawrocki, “Physical limits for scaling of integrated circuits,” in *Journal of Physics: Conference Series*, vol. 248, no. 1. IOP Publishing, 2010, p. 012059.
- [10] J. W. Ruge and K. Stüben, “Algebraic multigrid,” in *Multigrid methods*. SIAM, 1987, pp. 73–130.
- [11] C. Hutter and E. Weber, “Russia-ukraine war: Short-run production and labour market effects of the energy crisis,” IAB-Discussion Paper, Tech. Rep., 2022.
- [12] C. Benoit, “Note sur une méthode de résolution des équations normales provenant de l’application de la méthode des moindres carrés à un système d’équations linéaires en nombre inférieur à celui des inconnues (procédé du commandant cholesky),” *Bulletin géodésique*, vol. 2, no. 1, pp. 67–77, 1924.
- [13] C. Greif, “Numerical solution of linear systems,” *Vancouver BC*, 2008.
- [14] L. F. Richardson, “Ix. the approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam,” *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character*, vol. 210, no. 459-470, pp. 307–357, 1911.
- [15] C. G. Jacobi, “Ueber eine neue auflösungsart der bei der methode der kleinsten quadrate vorkommenden lineären gleichungen,” *Astronomische Nachrichten*, vol. 22, no. 20, pp. 297–306, 1845.



- [16] C. F. Gauss, *Supplementum theoriae combinationis observationum erroribus minimis obnoxiae*, 1828.
- [17] P. L. Seidel, *Ueber ein verfahren, die gleichungen, auf welche die methode der kleinsten quadrate führt, sowie lineäre gleichungen überhaupt, durch successive annäherung aufzulösen*. Verlag d. Akad., 1873, vol. 11.
- [18] D. M. Young, "Iterative methods for solving partial differential equations of elliptic type," Ph.D. dissertation, Harvard University, Cambridge, MA, USA, 1950.
- [19] W. Hackbusch, *Iterative solution of large sparse systems of equations*, 2nd ed. Springer, 2016, vol. 95.
- [20] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving," *Journal of research of the National Bureau of Standards*, vol. 49, no. 6, p. 409, 1952.
- [21] Y. Saad and M. H. Schultz, "Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.
- [22] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," 1950.
- [23] W. E. Arnoldi, "The principle of minimized iterations in the solution of the matrix eigenvalue problem," *Quarterly of applied mathematics*, vol. 9, no. 1, pp. 17–29, 1951.
- [24] B. Sousedik and J. Novotny, "Parallelisation of bicg-stab algorithm for finite element computations," *UMBC Mathematics Department Collection*, 2003.
- [25] R. W. Freund and N. M. Nachtigal, "Qmr: a quasi-minimal residual method for non-hermitian linear systems," *Numerische mathematik*, vol. 60, no. 1, pp. 315–339, 1991.
- [26] S. Mazumder, *Numerical methods for partial differential equations: finite difference and finite volume methods*. Academic Press, 2015.
- [27] Cerwinsky, Derrick and Douglas, Craig C., "An introduction to algebraic multigrid (amg) algorithms," URL: <https://pdfslide.net/documents/an-introduction-to-algebraic-multigrid-amg-algorithms-.html?page=1>, 05 2015.
- [28] A. J. Cleary, R. D. Falgout, V. E. Henson, and J. E. Jones, "Coarse-grid selection for parallel algebraic multigrid," in *International Symposium on Solving Irregularly Structured Problems in Parallel*. Springer, 1998, pp. 104–115.
- [29] R. D. Falgout and U. M. Yang, "hypre: A library of high performance preconditioners," in *International Conference on computational science*. Springer, 2002, pp. 632–641.
- [30] H. De Sterck, U. M. Yang, and J. J. Heys, "Reducing complexity in parallel algebraic multigrid preconditioners," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 4, pp. 1019–1039, 2006.
- [31] A. Katrutsa, T. Daulbaev, and I. Oseledets, "Deep multigrid: learning prolongation and restriction matrices," *arXiv preprint arXiv:1711.03825*, 2017.
- [32] 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.
- [33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
- [34] L. Olson and J. Schroder, "Pyamg: Algebraic multigrid solvers in python v4. 0," URL <https://github.com/pyamg/pyamg>, 2018.



- 
- [35] H. Gao, Z. Wang, and S. Ji, "Large-scale learnable graph convolutional networks," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1416–1424.
  - [36] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.
  - [37] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.
  - [38] 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.
  - [39] M. Zhang and Y. Chen, "Weisfeiler-lehman neural machine for link prediction," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 575–583.
  - [40] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Thirty-second AAAI conference on artificial intelligence*, 2018.
  - [41] H. Gao and S. Ji, "Graph u-nets," in *international conference on machine learning*. PMLR, 2019, pp. 2083–2092.
  - [42] J. Lee, I. Lee, and J. Kang, "Self-attention graph pooling," in *International conference on machine learning*. PMLR, 2019, pp. 3734–3743.
  - [43] M. Liu, H. Gao, and S. Ji, "Towards deeper graph neural networks," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 338–348.
  - [44] T. Chen, K. Zhou, K. Duan, W. Zheng, P. Wang, X. Hu, and Z. Wang, "Bag of tricks for training deeper graph neural networks: A comprehensive benchmark study," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.
  - [45] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *ICLR workshop on representation learning on graphs and manifolds*, 2019.
  - [46] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
  - [47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
  - [48] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 9323–9332.
  - [49] Pyzer-Knapp, Edward O. and Jordan, Kirk E. and Porter, Christopher N. and Turek, David W., "Beyond bigger, faster, cheaper machines: Intelligent simulation," URL: [https://research.ibm.com/science/documents/AI-Enriched-Simulation\\_Beyond\\_Bigger\\_Faster\\_Cheaper\\_Machines-Intelligent\\_Simulation.pdf](https://research.ibm.com/science/documents/AI-Enriched-Simulation_Beyond_Bigger_Faster_Cheaper_Machines-Intelligent_Simulation.pdf), 01 2022.