MOX-Report No. 34/2023

A Deep Learning algorithm to accelerate Algebraic Multigrid methods in Finite Element solvers of 3D elliptic PDEs

Caldana, M.; Antonietti, P. F.; Dede', L.

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it https://mox.polimi.it

A Deep Learning algorithm to accelerate Algebraic Multigrid methods in Finite Element solvers of 3D elliptic PDEs

Matteo Caldana[a,*], Paola F. Antonietti[a], Luca Dede'[a]

[a] *MOX, Dipartimento di Matematica, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy*

April 21, 2023

Abstract

Algebraic multigrid (AMG) methods are among the most efficient solvers for linear systems of equations and they are widely used for the solution of problems stemming from the discretization of Partial Differential Equations (PDEs). The most severe limitation of AMG methods is the dependence on parameters that require to be fine-tuned. In particular, the strong threshold parameter is the most relevant since it stands at the basis of the construction of successively coarser grids needed by the AMG methods. We introduce a novel Deep Learning algorithm that minimizes the computational cost of the AMG method when used as a finite element solver. We show that our algorithm requires minimal changes to any existing code. The proposed Artificial Neural Network (ANN) tunes the value of the strong threshold parameter by interpreting the sparse matrix of the linear system as a black-and-white image and exploiting a pooling operator to transform it into a small multi-channel image. We experimentally prove that the pooling successfully reduces the computational cost of processing a large sparse matrix and preserves the features needed for the regression task at hand. We train the proposed algorithm on a large dataset containing problems with a highly heterogeneous diffusion coefficient defined in different three-dimensional geometries and discretized with unstructured grids and linear elasticity problems with a highly heterogeneous Young's modulus. When tested on problems with coefficients or geometries not present in the training dataset, our approach reduces the computational time by up to 30%.

Key words: algebraic multigrid methods, partial differential equations, finite element method, elliptic problems, linear elasticity, deep learning, convolutional neural networks.

AMS subject classification: 65N22, 65N30, 65N55, 68T01

1 Introduction

Multigrid methods [14, 60] are among the state-of-the-art solvers for large linear systems that come from the discretization of Partial Differential Equations (PDEs). They are applicable to a wide range of discretizations such as polyhedral discontinuous Galerkin [5, 9] and virtual elements methods [7]. However, their drawback is that they rely on a sequence of coarser grids to solve the problem. The algebraic multigrid (AMG) methods are a highly scalable [11] generalization that is capable of building this hierarchy of grids algebraically. The main challenge behind the algebraic construction of a coarser mesh is the selection of the prolongation operator, which in turns relies on seeing the sparse matrix of the linear system as a weighted graph and defining a strong threshold parameter to partition the aforementioned graph.

The AMG was first introduced in the 80s [16, 17, 18, 51] and gained momentum very quickly is the subsequent years [20, 24, 54, 57]. In recent year modifications have been proposed to the AMG to improve

*Corresponding author: matteo.caldana@polimi.it

its efficiency, like smoothed aggregation [55, 56] and extend its range of application to different discretization techniques, like discontinuous Galerkin [8] and Ritz-type finite element methods [19, 21, 36], and to different physics equations like Maxwell's equations [13, 38], magnetohydrodynamics [1], Navier-Stokes equations [47, 59], linear elasticity [29] and multiphase poromechanics [61]. There is also a wide literature that tackles the AMG from a theoretical point of view, the foundation was laid down in [15, 25, 62, 63] and the most recent survey is found in [64]. In particular, we consider the finite element (FE) [46] discretization of diffusion and linear elasticity problems with highly heterogeneous coefficients in 3D. These problems represent a challenging benchmark since they feature a large number of unknowns and a very ill-conditioned system matrix. Consequently, the efficient application of the AMG method is the key to obtain fast and robust convergence.

In this work, we propose to use Deep Learning (DL) [42] techniques to find the optimal value of the strong threshold parameter, depending on the problem we want to solve. In particular, Artificial Neural Networks (ANNs) have gained widespread popularity for a variety of applications. They are versatile tools that are progressively being used in scientific computing [40, 44, 58], particularly in the numerical approximation of PDEs and model order reduction [27, 48]. ANNs can also be utilized within a data-driven framework to provide alternative closure models, which involve learning input-output relationships in complex physical processes [49, 50]. Since we treat the sparse matrix of the linear system like an image, part of the ANN that we employ is comprised by a Convolutional Neural Network (CNN). CNNs are among the most applied neural architectures to analyze visual imagery and achieved breakthrough results [30, 34, 35, 39, 41]. Today, they are only surpassed by visual transformer [22], which however have a much more complex architecture and are much more difficult to optimize. Moreover, CNNs have already been successfully applied in the field of scientific computing [12, 23].

The combination of multigrids methods and machine learning has already been used in literature. For instance, in [28, 37] there is the first attempt of optimizing the multigrid parameters, in [2, 3, 6] DL is used to perform grid refinement ad agglomeration, in [43, 45] Graph Neural Networks (GNNs) are used to learn the prolongation operator and in [53] reinforcement learning is used to perform graph coarsening, in [32, 33] DL is used to enhance domain decomposition. In this paper, we propose a DL-based algorithm that is able to tune on-the-fight the strong threshold parameter so as to minimize the computational time needed to solve the linear system at hand. The main difference with the previous works is that our algorithm is completely non intrusive: following the approach described in [4], it does not require any change to existing code (neither to the FE nor the AMG solver). This guarantees a wider range of applicability and means that we can rely on all the classical theoretical results regarding convergence. Indeed, we show that our algorithm is able to perform a fine tune of the AMG parameters just by analyzing the sparse matrix of the linear system we want to solve. In particular, we train an ANN to predict the expected computational cost of solving a linear system given as input a certain value of the strong threshold parameter and a small multi channel image representing the sparse matrix of the linear system. In order to build this representation we propose to employ the pooling operator used in CNN. We show that the pooling operator allows for a cheap evaluation that does not lose relevant information for the task at hand. We found that the proposed ANN-enhanced AMG method allows to reduce significantly the computational cost (elapsed time) needed to solve the linear system compared to employing the pre-defined choice of the parameters based on trial-and-error, experience, and literature.

This work is organized as follows. Section 2 introduces the mathematical framework of the AMG methods, in particular we define and show the importance of the strong threshold parameter. In Section 3 we present our algorithm. We describe the architecture of the ANN and show how to apply the pooling operator to a large sparse matrix. Section 4 is devoted to the numerical validation on our method. We first make preliminary sensitivity analysis of the hyperparameters of the ANN. We then apply our algorithm to a family of elliptic problems with a highly heterogeneous diffusion coefficient on structured and unstructured meshes and to a family of linear elasticity problems with a highly heterogeneous Young's modulus. Finally, in Section 5 we draw our conclusions with future developments.

Algorithm 1 One Iteration of the V-cycle of the AMG method
$\mathbf{u}^{(k)} = \texttt{vcycle}^k(\mathbf{u}^{(k)}, \mathbf{f}^{(k)}, \{(\mathrm{A}^{(j)}, \mathrm{B}_1^{(j)}, \mathrm{B}_2^{(j)})\}_{j=k}^{M}, \{(\mathrm{I}_j^{j+1}, \mathrm{I}_{j+1}^{j})\}_{j=k}^{M-1}, \nu_1, \nu_2)$

if $k = M$ **then**
  $\mathbf{u}^{(\mathbf{M})} = \texttt{gaussian\_elimination}(\mathrm{A}^{(M)}, \mathbf{f}^{(M)})$
else
  $\mathbf{u}^{(k)} \leftarrow \texttt{smooth}^{\nu_1}(\mathrm{A}^{(k)}, \mathrm{B}_1^{(k)}, \mathbf{u}^{(k)}, \mathbf{f}^{(k)})$
  $\mathbf{r}^{(k+1)} \leftarrow \mathrm{I}_k^{k+1}(\mathbf{f}^{(k)} - \mathrm{A}^{(k)}\mathbf{u}^{(k)})$
  $\mathbf{e}^{(k+1)} \leftarrow \texttt{vcycle}^{k+1}(\mathbf{u}^{(k)}, \mathbf{f}^{(k)}, \{(\mathrm{A}, \mathrm{B}_1, \mathrm{B}_2)^{(j)}\}_{j=k+1}^{M}, \{(\mathrm{I}_j^{j+1}, \mathrm{I}_{j+1}^{j})\}_{j=k+1}^{M-1}, \nu_1, \nu_2)$
  $\mathbf{u}^{(k)} \leftarrow \mathbf{u}^{(k)} + \mathrm{I}_{k+1}^{k}\mathbf{e}^{(k+1)}$
  $\mathbf{u}^{(k)} \leftarrow \texttt{smooth}^{\nu_2}(\mathrm{A}^{(k)}, \mathrm{B}_2^{(k)}, \mathbf{u}^{(k)}, \mathbf{f}^{(k)})$
end if

2 AMG methods

To start, we will explain the fundamental concepts and methods that make up AMG. For further information, refer to [51, 65]. Our goal is to solve the linear system $\mathrm{A}\mathbf{u} = \mathbf{f}$ where $\mathrm{A} \in \mathbb{R}^{n_1 \times n_1}$ is a symmetric positive definite (SPD) matrix with entries a_{ij} and $\mathbf{u}, \mathbf{f} \in \mathbb{R}^{n_1}$ are vectors with entries $(\mathbf{u})_i$. We define a smoothing scheme as a linear iterative method

$$\mathbf{u}_{k+1} = \mathbf{u}_k + \mathrm{B}(\mathbf{f} - \mathrm{A}\mathbf{u}_k), \quad k \geq 0 \tag{1}$$

where $\mathrm{B} \in \mathbb{R}^{n_1 \times n_1}$ and the initial guess $\mathbf{u}_0$ are given. We denote ν applications of (1) as $\texttt{smooth}^{\nu}(\mathrm{A}, \mathrm{B}, \mathbf{u}_0, \mathbf{f})$. The idea of multigrid methods is that after a certain number of iterations ν of Eq. (1), the error is more efficiently reduced by projecting the residual equation $\mathrm{A}\mathbf{e} = \mathbf{r} = \mathbf{f} - \mathrm{A}\mathbf{u}_\nu$ on a coarser space, and interpolating the solution back to the original space to apply the correction $\mathbf{u}_\nu + \mathbf{e}$. If we suppose to have a sequence of interpolation operators $\mathrm{I}_k^{k-1} \in \mathbb{R}^{n_{k-1} \times n_k}$, restriction operators $\mathrm{I}_{k-1}^{k} \in \mathbb{R}^{n_k \times n_{k-1}}$ and grid operators $\mathrm{A}^{(k)} \in \mathbb{R}^{n_k \times n_k}$ for $k = 2, ..., M$ with $\mathrm{A}^{(1)} = \mathrm{A}$, and of smoother $\mathrm{B}^{(k)}$ for $k = 1, ..., M$ with $n_1 > n_2 > ... > n_M$, and a number of pre-smoothing iterations ν_1 and post-smoothing ν_2, the V-cycle multigrid iteration is defined as in Algorithm 1 (notice the usage of the superscript $^{(k)}$ to indicate the different levels). The AMG method aims at finding the operators needed for this task by just relying upon the original matrix A. Since A is SPD we assume that

$$\mathrm{I}_k^{k+1} = (\mathrm{I}_{k+1}^{k})^\top, \quad \mathrm{A}^{(k+1)} = \mathrm{I}_k^{k+1}\mathrm{A}^{(k)}\mathrm{I}_{k+1}^{k}, \quad \forall k = 1, ..., M-1. \tag{2}$$

Hence, all the operators are defined once we have a recipe to build the interpolation operator.

2.1 Interpolation operator and coarse-grid selection

We consider the interpolation operator between the level k and $k+1$. We assume that we can split the variable into two sets: the one that needs interpolation at the fine level k and the one that are kept at the coarse level $k+1$, namely:

$$(\mathrm{I}_{k+1}^{k}\mathbf{x})_i = \begin{cases} (\mathbf{x})_i & \text{if } i \in \mathcal{C}^k, \\ \sum_{k \in \mathcal{C}_i^k} \omega_{ij}^k (\mathbf{x})_j & \text{if } i \in \mathcal{F}^k, \end{cases} \tag{3}$$

where $\mathbf{x} \in \mathbb{R}^{n_{k+1}}$ is a generic vector, $\mathcal{C}^k/\mathcal{F}^k$ is a coarse/fine partition of the set $\mathcal{N}_k = \{1, ..., n_k\}$, $\mathcal{C}_i^k = \{j \in \mathcal{C}^k : a_{ij} \neq 0\}$ is a subset of $\mathcal{C}^k$ that depends on i and ω_{ij}^k is a set of weights.

Since we are working between two levels, from now on, we will omit the superscript k. Before being able to define the value of ω_{ij} we need to define when the unknown $(\mathbf{u})_j$ is important in determining the value of $(\mathbf{u})_i$.

Definition 2.1 *Given a threshold parameter $0 < \theta \leq 1$, the set of of variables on which the variable i strongly depends on is*

$$\mathcal{S}_i = \{j \neq i : -a_{ij} \geq \theta \max_{l \neq i} \{-a_{il}\}, j = 1, ..., n_k\}.$$

Algorithm 2 AMG algorithm
$\mathbf{u} = \texttt{AMG}(\mathbf{u}, \mathrm{A}, \mathbf{f}, \theta, \{(\mathrm{B}_1^{(j)}, \mathrm{B}_2^{(j)})\}_{j=k}^{M}, \nu_1, \nu_2, N_{max}, tol)$

1: build $\{\mathcal{C}^k, \mathcal{F}^k\}_{k=1}^{M}$ using θ by means of CLJP
2: build the operators $\{\mathrm{I}_{j+1}^{j}\}_{j=k}^{M-1}$ employing Eq. (3) and Eq. (4)
3: build the operators $\{\mathrm{I}_{j}^{j+1}\}_{j=k}^{M-1}$ and $\{(\mathrm{A}^{(j)}\}_{j=k}^{M}$ by means of Eq. (2)
4: **while** $k < N_{max}$ **and** $\|\mathrm{A}\mathbf{u} - \mathbf{f}\| / \|\mathbf{f}\| < tol$ **do**
5: $\quad \mathbf{u} \leftarrow \texttt{vcycle}^1(\mathbf{u}, \mathbf{f}, \{(\mathrm{A}^{(j)}, \mathrm{B}_1^{(j)}, \mathrm{B}_2^{(j)})\}_{j=k}^{M}, \{(\mathrm{I}_{j}^{j+1}, \mathrm{I}_{j+1}^{j})\}_{j=k}^{M-1}, \nu_1, \nu_2)$
6: $\quad k \leftarrow k + 1$
7: **end while**

We also define the set of variables j that are influenced by the variable i

$$\mathcal{S}_i^\top = \{j : i \in \mathcal{S}_i, j = 1, ..., n_k\}.$$

The underlying assumption of AMG (see [51]) is that the error satisfies

$$\sum_{j \in \mathcal{C}_i} a_{ij}(\mathbf{e})_j + \sum_{j \in \mathcal{D}_i^s} a_{ij}(\mathbf{e})_j + \sum_{j \in \mathcal{D}_i^w} a_{ij}(\mathbf{e})_j = 0, \quad \forall i = 1, ..., n,$$

where $\mathcal{D}_i^s = \mathcal{F} \cap \mathcal{S}_i$ and $\mathcal{D}_i^w = \{j \in \mathcal{N} : a_{ij} \neq 0, j \notin \mathcal{S}_i\}$. This yields the formula for the weights

$$\omega_{ij} = -\frac{1}{a_{ij} + \sum_{l \in \mathcal{D}_i^w} a_{il}} \left(a_{ij} + \sum_{l \in \mathcal{D}_i^s} \frac{a_{il}\hat{a}_{lj}}{\sum_{m \in \mathcal{C}_i} \hat{a}_{lm}} \right) \text{ where } \hat{a}_{ij} = \begin{cases} 0 & \text{if } a_{ij}a_{ii} > 0, \\ a_{ij} & \text{otherwise.} \end{cases} \tag{4}$$

The last ingredient that we need to build the interpolation operator is the $\mathcal{C}/\mathcal{F}$ splitting. Among the several techniques that can be used, we outline the CLJP (Cleary-Luby-Jones-Plassman) algorithm. First, we construct the graph of variable dependencies $G = (V, E)$ with vertices $V = \{1, ..., n\}$ and edges $E = \{(i, j) \in V \times V : j \in \mathcal{S}_i\}$. For each vertex, we define the measure $\eta(i) = |\mathcal{S}_i^\top| + \tilde{\eta}$, where $\tilde{\eta}$ is a random number in $(0, 1)$ used to break ties. Then, we update η and G in the following way until all vertexes are designed as either $\mathcal{C}$ or $\mathcal{F}$. Whenever a update to $\eta(j)$ is such that $\eta(j) < 1$, j is flagged as $\mathcal{F}$.

1. Build the independent set $D = \{i \in V : \eta(i) > \eta(j) \forall j \in \mathcal{S}_i \cap \mathcal{S}_i^\top\}$ for the graph G.
2. For each $i \in D$
   2.1. For each $j \in \mathcal{S}_i$, decrement $\eta(j)$ and remove the edge (i, j) from E.
   2.2. For each $j \in \mathcal{S}_i^\top$, remove (j, i) from E
      2.2.1. For each $k \in \mathcal{S}_j^\top \cap \mathcal{S}_i^\top$, decrement $\eta(j)$ and remove (k, j) from E.
3. Every point in D is flagged as $\mathcal{C}$.

This algorithm is applied at each level until the size of the grid operator n_k is smaller than a certain given value n_{min}, that we fix equal to two. Thus, given the strong threshold θ we are able to construct the succession of operators needed for the V-cycle. Algorithm 2 showcases the full AMG method.

3 ANN-enhanced AMG

We now introduce the ideas behind the proposed AMG-ANN algorithm. We have shown that the strong threshold parameter θ heavily conditions the construction of the interpolation operator that stands at the basis of the AMG method. In literature, this value is usually fixed to 0.25 and 0.5 when applying the AMG to the solution of linear systems that stem from the discretization of 2D and 3D PDEs, respectively. Our algorithm aims at making an accurate choice of θ so as to minimize the computational cost of the AMG method. To this end, we devise an algorithm that exploits DL techniques to predict the value of θ that minimizes the elapsed CPU time needed to solve the given linear system $\mathrm{A}\mathbf{u} = \mathbf{f}$. Namely, we propose to use an ANN $\mathscr{F}$ to predict the computational time t, that is the number of seconds needed to solve a linear system with the AMG method (including the setup phase), and its confidence of that prediction σ^2, that is

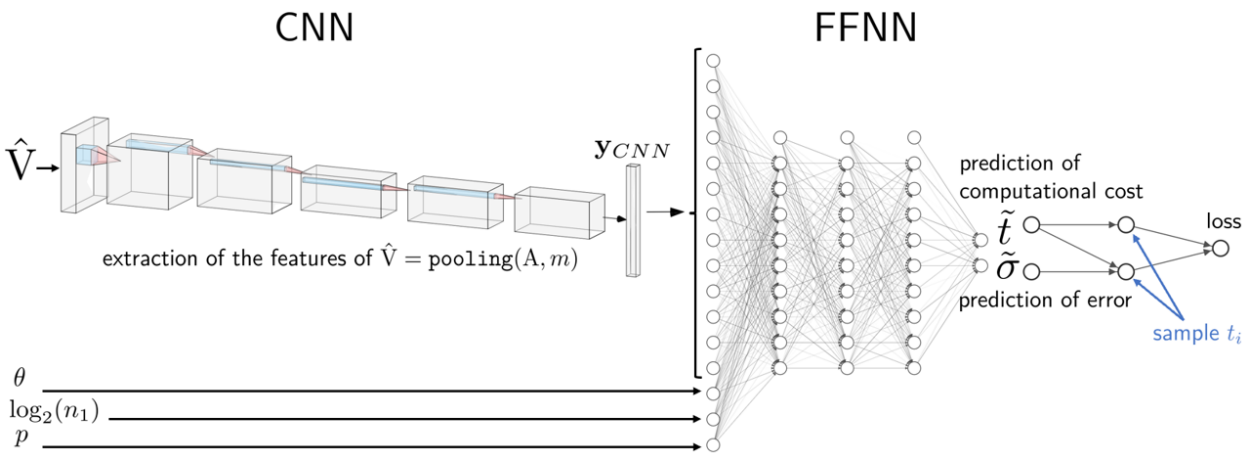


Figure 1: Architecture of the proposed ANN used to predict the optimal value of the strong threshold parameter θ.

the square of the committed error, given as input a small multi-channel image V extracted from the matrix A, the FE degree p, the logarithm of the size of A and the value of the strong threshold parameter θ. Then, online, we perform a 1D optimization to find the optimal θ^* that minimizes the predicted cost $\tilde{t}$ (we use the tilde to denote quantities approximated by the ANN). The architecture of the ANN is shown in Figure 1.

We start by briefly reviewing how we collect data and explaining how the extraction of V from A is performed. We then describe in details our approach in Algorithm 4 and we introduce a way to measure its performance. To make the article self-contained we discuss the basics of DL in Appendix A.

3.1 Gathering and smoothing data

Since the measurements of t are affected by error, we repeat the measurements r times with r between two and 100, where r is chosen inversely proportional to the mean elapsed time in the first two measurements. Indeed, the uncertainty in the measurements is caused by the tasks scheduled by the operating system in concurrence with the AMG solver that perturb the CPU load, hence if t is larger the uncertainty is smaller since these perturbations average out. This hypothesis is supported by our measurements, indeed we have observed that the sample variance of our measurements is inversely proportional to t (fixed r). The elapsed CPU time t is then defined as the mean of the measurements.

Moreover, we regularize data using Savitzky-Golay filter [52]. Namely, we fit a polynomial to a successive sub-sets of adjacent data points (called window) by means of least squares. We use a window of size 21 and polynomial degree 7 for uniformly sampled values of θ. The two parameters were selected through a process of manual tuning, whereby various combinations were tested and assessed for their performance on a subset of problems within the dataset. The selection was guided by visual inspection of the resulting fits, with consideration given to balancing model complexity and accuracy. To further validate the result of the filtering we have checked that the filter maintains the positive sign of the data and we manually reviewed the cases where the difference between the minimum of the filtered and unfilterd (raw) data is larger than a certain threshold. In these cases we checked if it would be appropriate to increase the degree of the polynomial or change the size of the window. Indeed, the smoothing sometimes changes the position of the minima in sharp valleys. For consistency all simulations where carried out on the same Intel Xeon Gold 6238R node of the HPC cluster at MOX.

3.2 Applying the pooling operation to large sparse matrices

Relying on the properties of the pooling operation that made it so widely used (see Appendix A), we aim to apply the pooling to the sparse matrix A. Indeed, A can be seen as a large black and white image and we want to apply the pooling operation (before the training) to reduce the computational cost associated with handling such a large amount of data, and to prune unimportant features. We call $\mathrm{V} = \texttt{pooling}(\mathrm{A}, m) \in \mathbb{R}^{m \times m \times 4}$ the result of the pooling of A of size m. We report the details in Algorithm 3. Here, we are assuming that

Algorithm 3 Pooling algorithm $\mathrm{V} = \texttt{pooling}(\mathrm{A}, m)$

access A in COO form and extract its: size n_1, `val`, `row`, `col`
initialize V to an $m \times m \times 4$ dense tensor with all zero entries $v_{ijl} = 0$
$q \leftarrow n_1/m$, $p \leftarrow n_1 \bmod m$, $t \leftarrow (q+1)p$
for $k = 0$ **to** `val.size()` $- 1$ **do**
  $i \leftarrow \texttt{row}[k]/(q+1)$ **if** $(\texttt{row}[k] < t)$ **else** $(\texttt{row}[k] - t)/q + p$
  $j \leftarrow \texttt{col}[k]/(q+1)$ **if** $(\texttt{col}[k] < t)$ **else** $(\texttt{col}[k] - t)/q + p$
  $v_{ij1} \leftarrow \max\{\max\{0, \texttt{val}[k]\}, v_{ij1}\}$
  $v_{ij2} \leftarrow \max\{\max\{0, -\texttt{val}[k]\}, v_{ij2}\}$
  $v_{ij3} \leftarrow v_{ij3} + \texttt{val}[k]$
  $v_{ij4} \leftarrow v_{ij4} + 1$
end for

the sparse matrix A is stored in coordinate list format, however the algorithm can be easily generalized to other sparse storage formats. The only difference with the pooling operation usually applied in CNNs is that instead of extracting the maximum, we extract the following features (always in a rectangular neighborhood): maximum of the positive part, maximum of the negative part, the sum, the number of non-zeros (nnz) entries. The insight behind extracting these features is the following: the positive part, the negative part and the sum are relevant in the definition of the weights of the interpolation operator Eq. (4), the nnz count is an indicator of the sparsity of the neighborhood. The algorithm has low complexity, namely $O(nnz)$, that in the case of FE means $O(p \cdot n_1)$. Moreover, we have experimentally proved that its elapsed CPU time is negligible with respect to the one to solve the linear system. This is a necessary condition for this algorithm to be worthwhile. We remark that Algorithm 3 could be easily generalized to work in parallel.

3.3 The AMG-ANN algorithm

We now have all the ingredients to describe the proposed algorithm in detail. We employ an ANN $\mathscr{F}$ to predict, as target the computational cost t and the square of the error committed by the ANN itself σ^2. The insight behind σ is that, since the elapsed CPU time t is polluted with noise due to the measurements, in this way we are able to know when the ANN is confident on its own prediction. We employ as inputs of the ANN the vector formed by the pooling of the matrix a $\mathrm{V} = \texttt{pooling}(\mathrm{A}, m)$, where the size m is an hyperparameter that we tune (see Section 4.1.1), the FEM degree p, the logarithm of the size of A $\log_2(n_1)$ and the strong threshold θ. Namely,

$$\mathscr{F}(\mathrm{V}, p, \log_2(n_1), \theta; \boldsymbol{\gamma}) = (\tilde{t}, \tilde{\sigma})$$

and we optimize (the superscript i indicates the i-th sample of the dataset of size T):

$$\min_{\boldsymbol{\gamma}} \frac{1}{T} \sum_{i=1}^{T} \mathrm{MSE}(t^i, \tilde{t}^i) + \mathrm{MSE}(\tilde{\sigma^i}^2, (t^i - \tilde{t}^i)^2).$$

Let us remark that p is not needed as input of the neural network. Indeed, the information about p is embedded into the matrix A: empirical results show that if we add p as input of the network, we need a lower number of epochs to reach the same loss, but it is still possible to effectively train the network without p. Hence, our algorithm is not limited to problems stemming from FE discretizations.

Given the matrix A associated to the linear system we want to solve, then our algorithm prescribes the default literature value of the strong threshold θ every time the weighted average variance $\hat{\sigma}$ of the map $\mathrm{A} \mapsto \theta$,

$$\hat{\sigma} = \frac{1}{181} \sum_{j=10}^{190} (1 - \tilde{t}^j)\tilde{\sigma}^j, \text{ where } (\tilde{t}^j, \tilde{\sigma}^j) = \mathscr{F}(\mathrm{V}, p, \log_2(n_1), \frac{j}{200}; \boldsymbol{\gamma})$$

is greater than a certain threshold $\bar{\sigma}$. This weighted average ensures that the variance of the prediction is more relevant if closer to the (expected) minima. The value $\bar{\sigma}$ is calibrated offline on the validation dataset

once after the training of the ANN. For each matrix A^i in the dataset we compute the relative error indicator $\hat{\sigma}^i$. Then we propose to choose $\bar{\sigma}$ as the ordinate of the elbow point of the sorted errors indicators $\hat{\sigma}^i$. On the other hand, if $\hat{\sigma} < \bar{\sigma}$, the algorithm prescribes as θ the value θ^* found by solving the optimization problem

$$\theta^* = \underset{\theta \in (0,1]}{\operatorname{argmin}} \, (\mathscr{F}(\mathrm{V}, p, \log_2(n_1), \theta; \boldsymbol{\gamma}))_1. \tag{5}$$

However, we empirically found that by solving the discretization of the above with two hundreds linearly spaced points, we have an estimate of θ^* that, for our aim, is indistinguishable from the solution of the continuous problem with gradient descent. There are two reasons to use this approach instead of predicting directly θ^*: we can quantify the expected improvement with respect to using the standard literature value of θ, and we do not have to solve a new optimization problem just to add one sample to the dataset. The architecture of $\mathscr{F}$ is comprised by two components. The first one leverages a CNN to extract the relevant features from the matrix V. These features are flattened in a intermediate dense layer, where they are concatenated with the other features $p, \log_2(n_1)$ and θ and fed into a dense feed forward network, which predicts the computational cost t. On the output layer we clip the prediction of the normalized computational cost between zero and one and use a softplus activation function for the variance estimate.

3.3.1 Evaluating the performance of the algorithm

A small loss is not a good indicator of the performance of our algorithm. Indeed, the choice of θ is subordinate to the map $\mathrm{A} \mapsto \theta^*$ defined by Eq. (5). With this in mind, we introduce the following quantities of interest. Let A be fixed, and let:

- t_{ANN} be the computational time of the AMG-ANN algorithm
- $t_{0.25}$ be the computational time of the AMG method for $\theta = 0.25$
- t_{MIN} be the computational time of the AMG method with

$$\theta^* = \underset{\theta \in \text{dataset for A}}{\operatorname{argmin}} \; t(\theta; \mathrm{A}).$$

- $P = 1 - \frac{t_{\mathrm{ANN}}}{t_{0.25}}$ be the performance index of the AMG-ANN algorithm
- $P_{\mathrm{MAX}} = 1 - \frac{t_{MIN}}{t_{0.25}}$ be the best performance of the AMG-ANN algorithm (according to the dataset).

Moreover, we can compound the quantities over different A and define PB as the percentage of cases where $P \geq 0$, P_m as the average of P and P_M as the median of P. The ratio P/P_{MAX} gives a measure of how well the ANN has learned the data.

3.3.2 Normalization

Normalization of data is a necessary step to assure fast convergence in neural networks. To this end we employ the following logarithmic normalization for each channel of the input V since it has been shown in [4] to outperform a classical linear normalization in this kind of applications:

$$\hat{v}_{ij} = \frac{\log(|v_{ij}| + 1)}{\max_{i,j} |\log(|v_{ij}| + 1)|} \frac{v_{ij}}{|v_{ij}|}. \tag{6}$$

One of the main properties of this normalization is to preserve the sparsity pattern of A. In the Appendix B we shows some examples of the combination of pooling and normalization for some matrices A and confront them with the relative sparsity pattern. Concerning the output, we normalize the data linearly between zero and one, independently for each subset of outputs in our dataset defined by fixing a matrix A. Algorithm 4 shows the complete ANN-enhanced AMG.

Algorithm 4 ANN-enhanced AMG
$\mathbf{u} = \texttt{ANN\_AMG}(\mathbf{u}, \mathrm{A}, \mathbf{f}, \{(\mathrm{B}_1^{(j)}, \mathrm{B}_2^{(j)})\}_{j=k}^{M}, \nu_1, \nu_2, N_{max}, tol)$

1: $\mathrm{V} \leftarrow \texttt{pooling}(\mathrm{A}, m)$ (Algorithm 3)
2: normalize V by means of Eq. (6), obtaining $\hat{\mathrm{V}}$
3: $(\tilde{t}^j, \tilde{\sigma}^j) \leftarrow \mathscr{F}(\hat{\mathrm{V}}, p, \log_2(n_1), \frac{j}{200}; \boldsymbol{\gamma}), \quad j = 10, 11, ..., 190$
4: $\hat{\sigma} \leftarrow \frac{1}{181} \sum_{j=10}^{190} (1 - \tilde{t}^j)\tilde{\sigma}^j$
5: $\theta^* \leftarrow \mathrm{argmin}_{\tilde{t}^j} \frac{j}{200}$ **if** $\hat{\sigma} > \bar{\sigma}$ **else** 0.5
6: $\mathbf{u} \leftarrow \texttt{AMG}(\mathbf{u}, \mathrm{A}, \mathbf{f}, \theta^*, \{(\mathrm{B}_1^{(j)}, \mathrm{B}_2^{(j)})\}_{j=k}^{M}, \nu_1, \nu_2, N_{max}, tol)$ (Algorithm 2)

4 Numerical Results

One of the advantages of our algorithm is that it integrates seamlessly into pre-existing code. We carry out several numerical experiments using using deal.II [10] for the construction of the linear system and we rely on BoomerAMG of the library HYPRE [26] as the AMG solver. No changes are made to BoomerAMG, we use it as a black-box solver and only change the value of the strong threshold parameter θ, all the other parameters (choice of pre- and post- smoother B, number of pre- and post- smoothing cycles ν, etc.) are left as default.

Concerning the training of the ANN, we employ a 20-20-60 split of the dataset into training-validation-test, respectively. The splitting is done among problems, this means that a certain matrix A appears just in one of the three datasets. This entails that when we evaluate the algorithm on the test (or validation) dataset, it is making predictions on problems it has never seen before. Notice also that the test dataset is much larger than the other two: this makes sense only when the dataset is large enough so that the neural network can generalize well. Training has been performed using the Adam optimizer with default learning rate 10^{-3} and minibatch size equal to 32. Moreover, we found that applying a suitable learning rate schedule is key to accelerate the optimization. Namely, we employ a learning rate schedule that halves the learning rate with patience 15 epochs.

The convolutional part of the network is composed by either one or two plain convolutional blocks each one ending with a max-pooling layer. For each test case, we tune the number of convolutional layers, the size of the filter, the number of filters the number of dense layers and the width of the dense layers.

4.1 Test Case 1: Highly heterogeneous diffusion problem, unstructured Grids

Let us consider the following parametric elliptic problem

$$\begin{cases} -\mathrm{div}(\mu \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega, \end{cases} \tag{7}$$

where $\mu \in L^\infty(\Omega)$ is a highly heterogeneous piecewise positive constant and Ω varies among four geometries: a simplex, a plate with a hole, a ball and a cylinder. We consider in total eight different discretization of the four domains and their nested refinements. We show the coarsest meshes in the first column of Table 1. The diffusion coefficient μ is a highly heterogeneous piecewise constant that is conforming with the mesh. Namely, μ has a different value equal to 10^{ε_i} on the i-th cell of one of the coarsening of the mesh, where ε_i is randomly chosen between $[0, \varepsilon_{MAX}]$ and $\varepsilon_{MAX} = 1, 3, 10$. Moreover, we again employ four different DoFs numbering to enhance the dataset. In this way, we build a dataset counting 5873 different configurations and thus containing 217301 samples.

4.1.1 Study of hyperparameters and pooling features

In this section we analyze the influence of some hyperparameters on the final loss when we employ a subset of the dataset. This is a preliminary analysis that we made for each test case in order to choose the best value of m (the size of the pooling V), which of the features of V are relevant (that is which of the four layers f of v_{ijf} should we employ as input of the CNN) and the activation function. Namely we performed

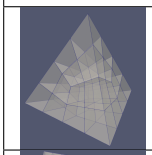
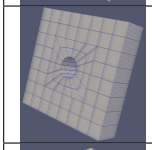
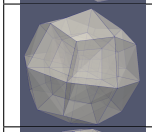
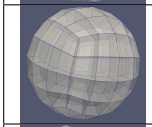
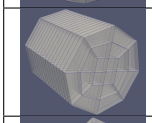
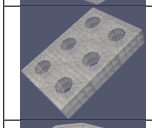
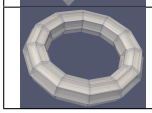
Picture	Geometry	Description
	Simplex	Tetrahedron inscribed in the unit ball centered in $(0, 0, 0)$.
	Plate with hole	The $(-2, 2) \times (-2, 2) \times (-\frac{1}{2}, \frac{1}{2})$ plate with a centered hole of radius 0.4 along the z axis. Discretizations with two or eight slices in the z-direction are considered.
	Ball	Unit ball centered in zero.
	Balanced ball	A variant of the unit ball centered in zero that has a better balance between the size of the cells around the outer curved boundaries and the cell in the interior.
	Cylinder	Cylinder of unit radius and height two centered in zero. We consider three different discretizations with one, two or eight slices along its height.
	Holes	3×2 replica of the "Plate with hole" geometry on the x and y. Only test dataset.
	Torus	Torus of circle radius 2 and inner radius $\frac{1}{2}$. The axis of the torus is the y-axis. Only test dataset.

Table 1: Test case 1: list of meshes used in the dataset.

a grid search over different network architectures. Figure 2 shows the results. The ReLU activation function consistently reaches the smallest loss, hence we employ ReLU activation functions in conjunction with He initialization [31]. The ANN that uses all four the features of the pooling has the smallest loss, thus the whole tensor V is relevant for this task. The pooling size $m = 75$ gives the best trade-off between accuracy and computational cost.

4.1.2 Calibration of the $\bar{\sigma}$ threshold

We discuss how the choice of $\bar{\sigma}$ is made and how it impacts the performance of our algorithm. In Figure 4 we plot the accuracy (PB) and average time reduction (P_m) when choosing as $\bar{\sigma}$ the n-th largest $\hat{\sigma}$ of the validation dataset. Notice there is an accuracy-performance trade-off, indeed, as $\bar{\sigma}$ tends to zero, PB tends to one but at the same time P_m tends to zero. Moreover, the fact that P_m is not strictly decreasing means that the error committed by the algorithm is not just due to noisy measurements. However, by using as $\bar{\sigma}$ the elbow of the ordered $\hat{\sigma}$ we have a gain of 6.1% in terms of accuracy meanwhile the mean performance decreases of only 0.5%.

4.1.3 Evaluation of the algorithm

Starting from the architecture we found in Section 4.1.1, we fine tuned the number of filters, the kernel size of the CNN and the width and depth of the dense part. The ANN has a loss of $2.89 \cdot 10^{-4}$. The performance of our algorithm is evaluated by the indexes described in Section 3.3.1 on the test dataset (see Table 2) and by visualizing the scaling of the AMG with respect to the number of DoFs (see Figure 9). Moreover, we remark that the scaling shown in the bottom row of Figure 9, ideally, should be constant. Indeed, this is true when we apply the AMG to problems where $\max_\Omega \mu / \min_\Omega \mu \sim 1$, however in the cases that we consider is not

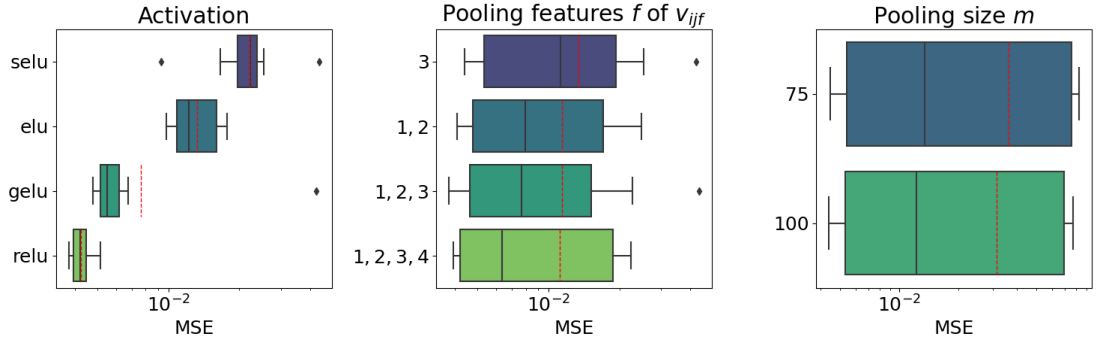


Figure 2: Boxplots of the impact of the choice of the activation function (*left*), the pooling features (*center*) and the pooling size m (*right*), on the validation loss (MSE) of an ANN trained on a subsample of the dataset described in Section 4.1. The red line represents the mean.

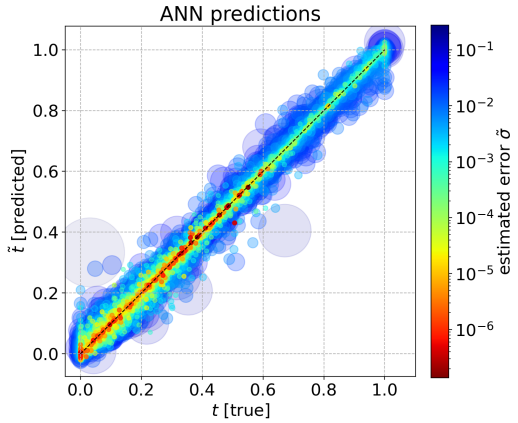


Figure 3: Test case 1: Prediction of the ANN. Color and size are proportional to the estimated error $\tilde{\sigma}$.

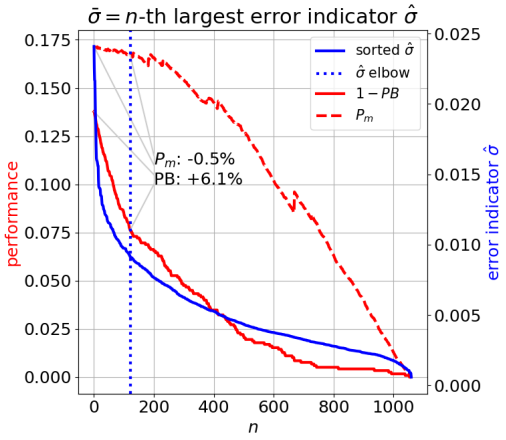


Figure 4: Test case 1: performance accuracy trade-off of the AMG-ANN algorithm depending on the choice of $\bar{\sigma}$. The algorithm is evaluated on the validation set.

possible to achieve this. In any case, the AMG-ANN is able to achieve a much better scaling with respect to the not tuned AMG version. In particular, the algorithm has an accuracy of 92.51% and an produces an average reduction of the computational cost of almost 16% in average. Since in this case the median on P_{MAX} is rather small (82%), there is still margin of improvement for the reduction of the computational cost by tuning the architecture of the ANN.

Figure 3 shows all the predictions made by the trained ANN, notice that the largest points are the furthest from the bisector, thus the ANN is successfully predicting where it is inaccurate. In Figure 5 we highly some relevant predictions. Notice that the map $\theta \mapsto t$ exhibits many different complex patterns and that θ^* varies depending on the problem considered. Hence, it is not possible to find a fixed value θ^* that works well for all the problems. In Figure 6, we showcase relevant cases of when the algorithm is not accurate, in particular, we categorize the errors of the algorithm into three classes (from left to right):

- Noisy measurements: the error is due to the measurement of t. As mentioned before, we reduce this error by repeating the measurements and regularizing data. Moreover, $\tilde{\sigma}$ is often a good indicator of a large error in these cases.
- Generalization error: this takes place when the network is not able to generalize beyond the training set. You can notice that in this case the prediction is completely wrong.
- ANN error: in this category we classify the errors due to the approximation capacity of the ANN, the optimization error and the loss of information occurring during the pooling of A. These type of errors

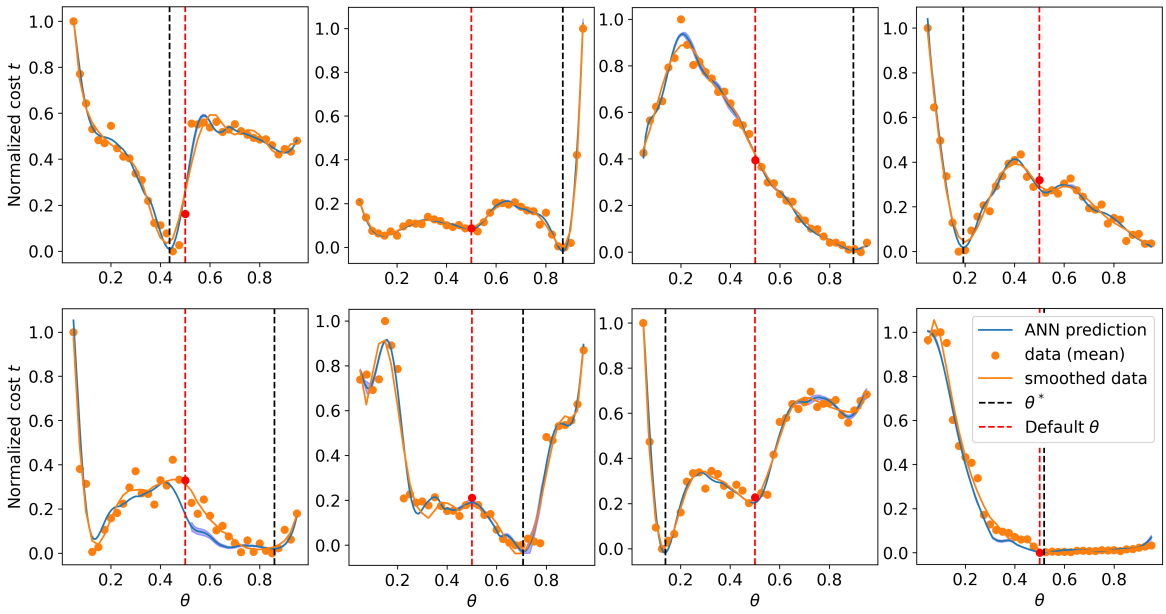


Figure 5: Test case 1: Predictions made by the ANN for some exemplary values of the diffusion coefficient and domain shape. On the y-axis the normalized computational cost t, on the x-axis the value of the strong threshold parameter θ. The dots represent the raw measurements of the computational time.

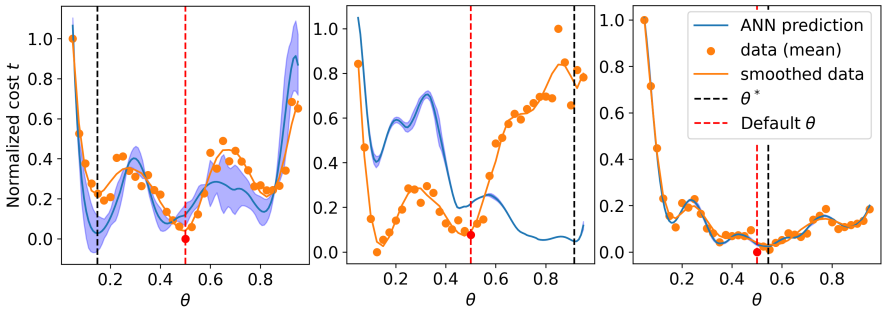


Figure 6: Test case 1: three representative cases of predictions of the normalized computational cost t made by the ANN that leads to sub-optimal values of θ^*. The blue area is the ANN error estimate $\tilde{\sigma}$.

can be mitigated by hyperparameters tuning. However, there are some limitations, for instance, even if we would like m to be large, in order to lose the least amount of information possible, the cost of the evaluation of $\mathscr{F}$ depends quadratically on it, so it must stay small. Moreover, the error estimate $\tilde{\sigma}$ is affected by these kind of errors itself, and it may not be of use in these cases.

4.1.4 Evaluation on unseen domains

We tested the ANN-AMG algorithm by applying it to problems with a diffusion coefficient μ that was not present in the training dataset, but on a known geometry. To prove the generalization capability of the algorithm, we now test it using domains and diffusion coefficients it has never seen before. The two domains we employ are represented in the two last rows of Table 1: a replication of the plate with a hole in the x and y coordinates and a torus. We build the dataset exactly as done before, in total it counts 445 different problems. The performance on the "Holes" geometry are similar to a known geometry: $PB = 86\%$ and $P_M = 18\%$. Thus, we have shown that the algorithm is able to generalize also on domains with a different topology w.r.t. the ones in the training dataset. This result is linked to the fact that a (small) subset of the considered geometry (the plate with the hole) is present in the training dataset. Indeed, the results on the Torus are much worse: $PB = 64\%$ and $P_M = 6\%$. The algorithm is still making predictions that

Algorithm 5 Diffusion μ in a given point $\mathbf{x} \in \Omega = (-1,1)^3$, having fixed the $\texttt{mode}$, $\texttt{size}$ and $\boldsymbol{\varepsilon} \in \mathbb{R}^{\texttt{size}^{(\texttt{mode})}}$. $\mu = \mu(\mathbf{x}; \texttt{mode}, \texttt{size}, \boldsymbol{\varepsilon})$

$j \leftarrow 1$
for $k = 0$ **to** $\texttt{mode}$ **do**
  $j \leftarrow j + \lfloor ((\mathbf{x})_i + 1)\texttt{size}/2 \rfloor \, \texttt{size}^{i-1}$
end for
$\mu \leftarrow 10^{(\boldsymbol{\varepsilon})_j}$

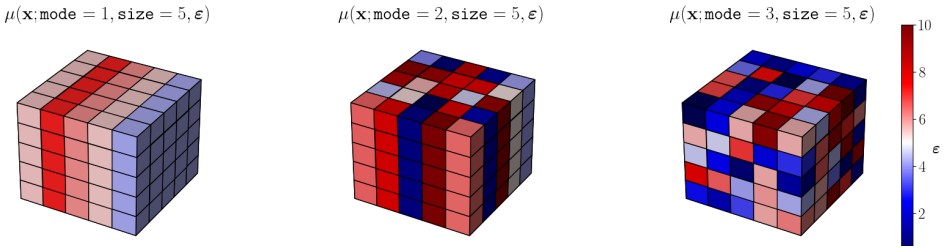


Figure 7: Examples of the diffusion coefficient μ employed in Section 4.2 for some choices of the vector $\boldsymbol{\varepsilon}$ depending on the $\texttt{mode}$ parameter (one to three from left to right) and fixed $\texttt{size}$=5.

are significantly better than any prediction made "at random", however this challenging mesh shows the limitation of our approach. We show the details about the performance in Table 2.

4.2 Test Case 2: Highly heterogeneous diffusion problem, structured grids

Let us consider the same elliptic problem (7) of before. However, here we have $\Omega = (-1,1)^3$ and $\mu \in L^\infty(\Omega)$ is a highly heterogeneous piecewise positive constant defined differently from before. Namely, μ is defined by means of Algorithm 5, where $\texttt{mode} = 1,2,3$ defines if the pattern is either made of slices, lines or is checkerboard like, $\texttt{size} = 1,2,...$ defines how many times the pattern repeats, and the vector $\boldsymbol{\varepsilon} \in \mathbb{R}^{\texttt{size}^{(\texttt{mode})}}$ defines the value of μ. Figure 7 shows a representation of μ for $\texttt{size} = 5$ and $\texttt{mode} = 1,2,3$. The problem is discretized by means of continuous FE of order p on nested Cartesian meshes. The dataset is built by varying $p = 1,2,3$, $\texttt{mode} = 1,2,3$, $\texttt{size} = 2,3,...,10$, the mesh size h between $\frac{\sqrt[3]{2}}{\texttt{size}}$ and $\frac{\sqrt[3]{2}}{\texttt{size}2^{8-p}}$, and choosing at random the components of the vector $\boldsymbol{\varepsilon}$ between 0 and ε_{MAX}, where $\varepsilon_{MAX} = 1,3,10$. In order to test the stability of the algorithm we also use four different DoFs (degrees of freedom) numbering. Namely, when applying the renumbering, the underlying sparsity graph of the matrix A and thus the optimal value of θ stays the same, but the pooling V changes, so the ANN should learn to be inveriant with respect to these changes. The dataset we built counts 5471 different problems and thus contains 202427 samples.

We started by using an architecture with hyperparameters chosen as described in Section 4.1.1 and then tuned the number of filters and the size of the kernel of the CNN and the width and depth of the dense part of the ANN. We obtain an ANN with a loss of $1.55 \cdot 10^{-4}$. Details about its performance and its scaling are shown in Figure 9 and Table 2. In particular, our algorithm has a 93% accuracy with an average reduction of the CPU time of almost 30%, and in half of the cases you can expect a reduction greater than 32%.

4.2.1 What does the ANN learn?

To better understand how the model problem we considered works and what does the ANN learn, we study a specific subset of the model problems defined in the previous section. Namely we consider a diffusion μ which is everywhere constant and equal to one apart from one cell of the mesh, where it is equal to $\mu_{MAX} = 10^{\varepsilon}$ with $\varepsilon = 2,4,8$. We stress that this family of problems was not present in the dataset we employed to train our ANN. Moreover, we define d to be the distance between the cell with the largest diffusion coefficient μ_{MAX} from the center of the domain. Figure 8 (*top-left*) shows a graphical representation of the problem. First we analyze the conditioning of the matrix A. As expected it scales with respect to the mesh size h as h^{-2} and it is linearly proportional to the ratio between the maximum and minimum value of diffusion

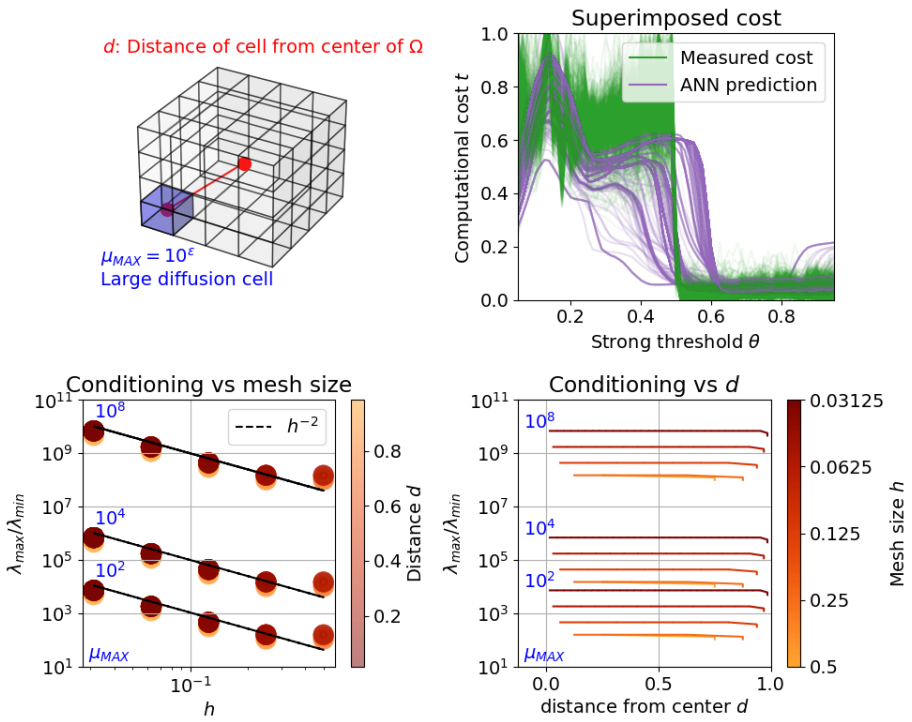


Figure 8: Study of a special subset of problems of Test case 2. (*Top-left*) We consider a uniform diffusion $\mu = 1$ apart from one cell where $\mu = \mu_{MAX} = 10^{\varepsilon}$ and $\varepsilon = 2, 4, 8$. This cell has distance d from the origin. (*Top-right*) Superimposed normalized computational cost t needed to solve the system with the AMG method depending on the strong threshold parameter θ and relative predictions made by the ANN. (*Bottom*) Conditioning number of the matrix A depending on the distance d and the mesh size h.

coefficient μ in the domain (i.e. is proportional to μ_{MAX}). The position of the cell does not affect the condition number. There is just a small reduction of it when the cell touches the boundaries: this is simply due to the Dirichlet boundary conditions, see Figure 8 (*bottom*). We then test our algorithm. Figure 8 (*top-right*) shows that the optimal value of θ is almost always in the same range: between 0.5 and 0.9. Moreover, we see that the ANN is able to capture the overall relation between the normalized computational cost t and the strong threshold. Hence, in all this cases, the predicted value θ^* by the AMG-ANN algorithm is near the true optimum. Finally, to gain some insight on the ANN we computed the feature maps extracted using the convolutional filters learned by the ANN. In Appendix C we show some of them: we can see that from a layer to another the CNN is enhancing the features that deems relevant.

4.3 Test Case 3: Linear Elasticity

We consider the following linear elasticity problem

$$\begin{cases} -\text{div}(\mathbb{C}\nabla_S \mathbf{u}) = \mathbf{f}, & \text{in } \Omega, \\ \mathbf{u} = \mathbf{0}, & \text{on } \partial\Omega, \end{cases} \tag{8}$$

where $\Omega = (-1, 1)^3$ and $\nabla_S \mathbf{u} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^\top)$ is the symmetric gradient and $\mathbb{C}$ is a rank-4 tensor that encodes the stress-strain relationship. Under the assumptions that the material under consideration is isotropic, by introducing the two Lamé coefficients $\lambda \in L^\infty(\Omega)$ and $\mu \in L^\infty(\Omega)$, the coefficient tensor reduces to

$$\mathbb{C}\nabla_S \mathbf{u} = 2\mu\nabla_S \mathbf{u} + \lambda \text{tr}(\nabla_S \mathbf{u})\text{I}$$

where tr is the trace operator and I is the identity matrix. We can rewrite the Lamé parameters in terms of the Young's modulus $E > 0$ and the Poisson ratio $\nu \in (0, \frac{1}{2})$; we have $G = E/(1+\nu)$ and $\beta = \nu/(1-2\nu)$. The problem has been discretized by means of FE of order $p = 1, 2, 3$ on a structured cartesian grid of diameter

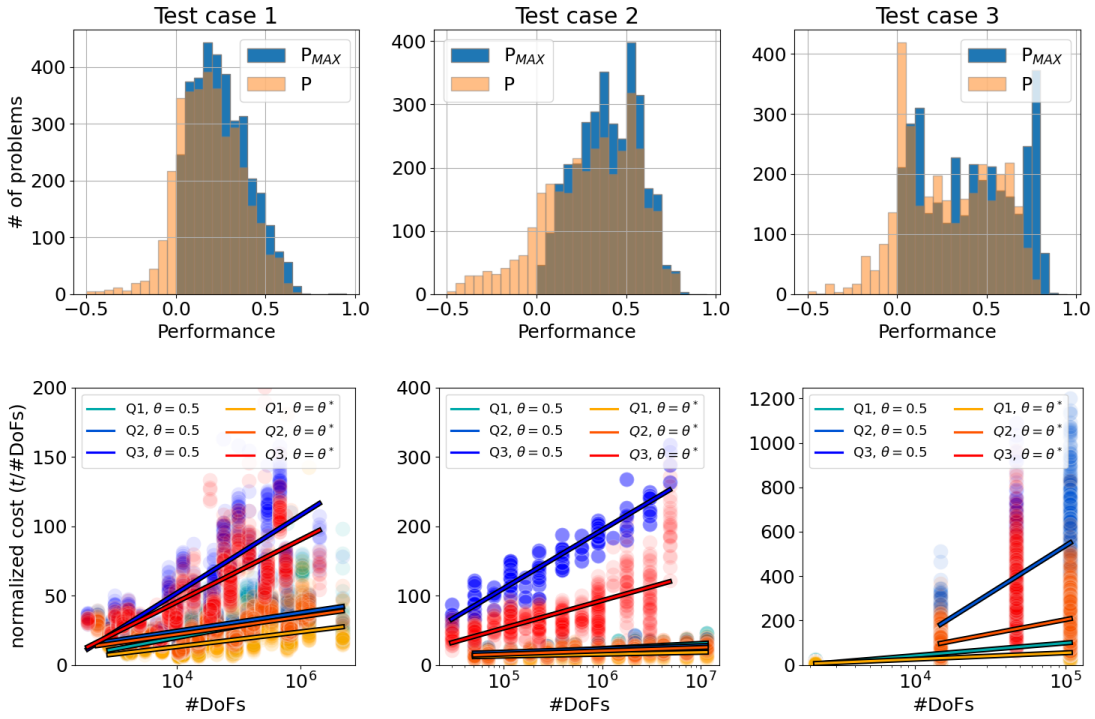


Figure 9: (*Top*) Performance gain P of the AMG-ANN algorithm and maximum theoretical performance P_{MAX} for the three test cases. (*Bottom*) Scatter plot with linear regression of the DoF normalized computational cost of the AMG-ANN algorithm with respect to the number of DoFs and the polynomial degree p. In tones of red the AMG-ANN algorithm and in tones of blue the standard AMG method.

$h = \frac{\sqrt[3]{2}}{2}, ..., \frac{\sqrt[3]{2}}{2^{4-p}}$. We fix the Poisson ration $\nu = 0.29$ and choose a highly heterogeneous Young modulus E. Namely E has the same pattern of the diffusion coefficient μ described in Section 4.2 and is such that `size` is even. We also employ four different DoFs numbering to enhance the dataset. In this way, we build a dataset counting 5873 different problems and thus containing 217301 samples.

As we have done for the previous test case, we start with an architecture found like described in Section 4.1.1 and then we fine tuned the number of filters, the kernel size of the CNN and the width and depth of the dense part. The ANN has a loss of $5.64 \cdot 10^{-4}$. Evaluated on the test dataset, the algorithm has an accuracy of 90.71% and an average reduction of the computational cost of almost 23% in average. Details about its performance and its scaling are shown in Figure 9 and Table 2. There is still margin of improvement concerning the reduction of the cost since P_{MAX} is relatively small, however it is possible to see a remarkable reduction of the scaling of the computational cost in the bottom row of Figure 9.

5 Conclusions

We proposed a DL algorithm to optimize the choice of the strong threshold parameter that stands at the basis of the construction of the levels needed for the AMG method. We have shown that our algorithm can be applied to a wide class of challenging problems coming from the FE discretization of PDEs. Namely, we have provided numerical results for three different test cases concerning a Poisson problem with a highly heterogeneous diffusion coefficient on (*i*) structured and (*ii*) unstructured grids and (*iii*) a linear elasticity problem with a highly heterogeneous Young's modulus. Our algorithm performs better, other than using the standard literature value, in more than 90% of the cases and reduces significantly the computational cost (up to 30% on average).

The algorithm hinges upon the pooling operator to compress a large sparse matrix into a small tensor to feed into an ANN. Indeed, we have proved that the pooling operator reduces the computational cost and preserves relevant information needed to perform the complex regression task at hand. Moreover, our

Test case	$\bar{\sigma}$	PB	P (avg/median)		P/P_{MAX} (median)
1: Unstructured	∞	86.58%	16.73%	17.90%	85.54%
	$\tilde{\sigma}$ elbow	92.51%	15.91%	15.46%	-
1: Holes	∞	80.82%	15.09%	18.02%	76.06%
	$\tilde{\sigma}$ elbow	86.01%	15.13%	17.76%	-
1: Torus	∞	63.88%	0.85%	5.69%	45.72%
	$\tilde{\sigma}$ elbow	65.08%	0.01%	3.30%	-
2: Structured	∞	87.97%	30.51%	32.14%	93.50%
	$\tilde{\sigma}$ elbow	93.03%	29.54%	19.42%	-
3: Elasticity	∞	83.72%	24.01%	21.40%	88.59%
	$\tilde{\sigma}$ elbow	90.71%	22.60%	18.54%	-

Table 2: Evaluation of the performance of the AMG-ANN algorithm for each test case depending on the choice of the threshold error $\bar{\sigma}$.

algorithm can be introduced with minimal changes into any existing code that uses an AMG solver.

In future works we plan to generalize our algorithm by using one single ANN to make predictions on linear systems stemming from different underlying PDEs and being able to predict the optimal value of other parameters of the AMG method such as the type of smoother and the number of smoothing steps. We also plan to try more advanced computer vision models, such as transformers [22]. Finally, we aim to further investigate the properties of the pooling operator applied to sparse matrices and gain a better understanding of what does the CNN learn.

Acknowledgments. M.C., P.F.A and L.D. are members of the INdAM Research group GNCS. P.F.A has been partially funded by the research projects PRIN17 (n. 201744KLJL) and PRIN 2020 (n. 20204LN5N5) funded by Italian Ministry of University and Research (MUR) and partially funded by European Union - NextGenerationEU. L.D. has been partially funded by the research project PRIN 2020 (n. 20204LN5N5) funded by MUR. L.D. acknowledges the support by the FAIR (Future Artificial Intelligence Research) project, funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, investment 1.3, line on Artificial Intelligence), Italy.

CRediT authorship contribution statement P.F.A. and L.D.: Conceptualization, Methodology, Review and editing, Supervision, Project administration, Funding acquisition. M.C.: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Visualization, Original draft.

Declaration of competing interests The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

A Basic concepts of Deep Learning

A deep learning regression model is a function $\mathscr{F} : \mathbb{R}^{N_{IN}} \to \mathbb{R}^{N_{OUT}}$ that maps an input $\mathbf{x}$ to an output $\tilde{\mathbf{y}}$ and depends on a vector of parameters $\boldsymbol{\gamma}$. The parameters $\boldsymbol{\gamma}$ are chosen by minimizing the error (loss) evaluated on a training dataset of known input-output couples $\{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^T$, namely

$$\min_{\boldsymbol{\gamma}} \frac{1}{T} \sum_{i=1}^{T} \mathscr{L}(\mathbf{y}^i, \mathscr{F}(\mathbf{x}^i, \boldsymbol{\gamma}))$$

where $\mathscr{L}$ is the loss function. Namely, we employ the mean squared error (MSE). The optimization is performed by means of gradient descent updates that uses a mini-batch of the training dataset to compute $\nabla_{\boldsymbol{\gamma}} \mathscr{F}$ with automatic differentiation.

Feed-forward neural networks Let $\mathbf{y}^{(0)} = \mathbf{x}$ and $\mathbf{y}^{(L)} = \tilde{\mathbf{y}}$, a dense feed-forward neural network (FFNN) of depth L is the composition of L functions called layers defined as

$$\mathbf{y}^{(l)} = h^{(l)}(\mathrm{W}^{(l)}\mathbf{y}^{(l-1)} + \mathbf{b}^{(l)}), \quad l = 1, ..., L$$

where $\mathrm{W}^{(l)} \in \mathbb{R}^{N_l \times N_{l-1}}$ (weights) and $\mathbf{b}^{(l)} \in \mathbb{R}^{N_l}$ (biases) are the parameters $\boldsymbol{\gamma}$, and $h^{(l)}$ is a scalar non-linear activation function applied component-wise.

Convolutional neural networks Convolutional neural networks (CNNs) are neural networks that use convolution instead of matrix multiplication in at least one of their layers. They have great success in computer vision tasks thank to their ability to exploit the structured data format of images. Indeed, convolution layers enjoy three properties: shared parameters, that is, each parameter is tied to multiple component of the input; sparse interactions, that is, each component of $\mathbf{y}^{(l)}$ depends only on a subset of the components of $\mathbf{y}^{(l-1)}$, (this also entails a lower number of parameters and thus a greater efficiency); equivariance to translation, that is the application of a translation and a convolution can be interchanged to obtain the same result. The last ingredient of a CNNs layer is the pooling operator. After several parallel convolutions and a non-linear activation, we replace the output of the layer at a certain location with a summary statistic of the nearby outputs. A popular option is the max pooling [39], which reports the maximum value over a rectangular neighborhood. The aim of the pooling operation is to reduce the computational cost, increase statistical efficiency and adding invariance to small translations.

B Pooling visualization

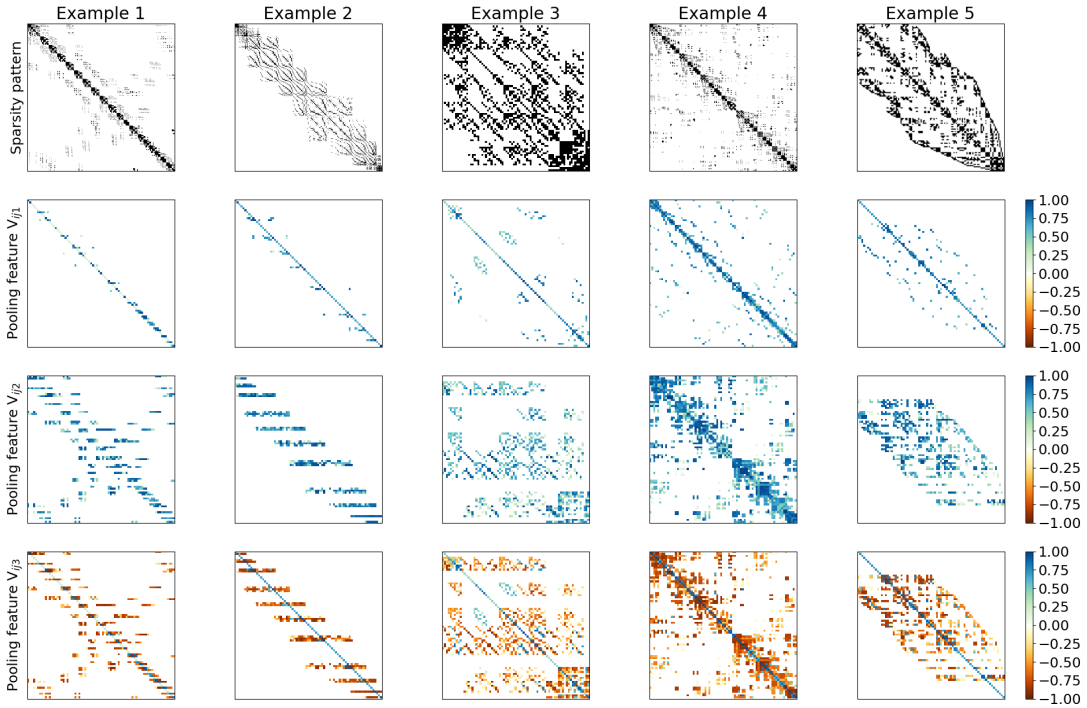


Figure 10: Visual representation of the normalized pooling results $\hat{\mathrm{V}}$ that is fed into the ANN for some exemplary problems taken from the dataset described in Section 4.1.

C CNN feature maps visualization

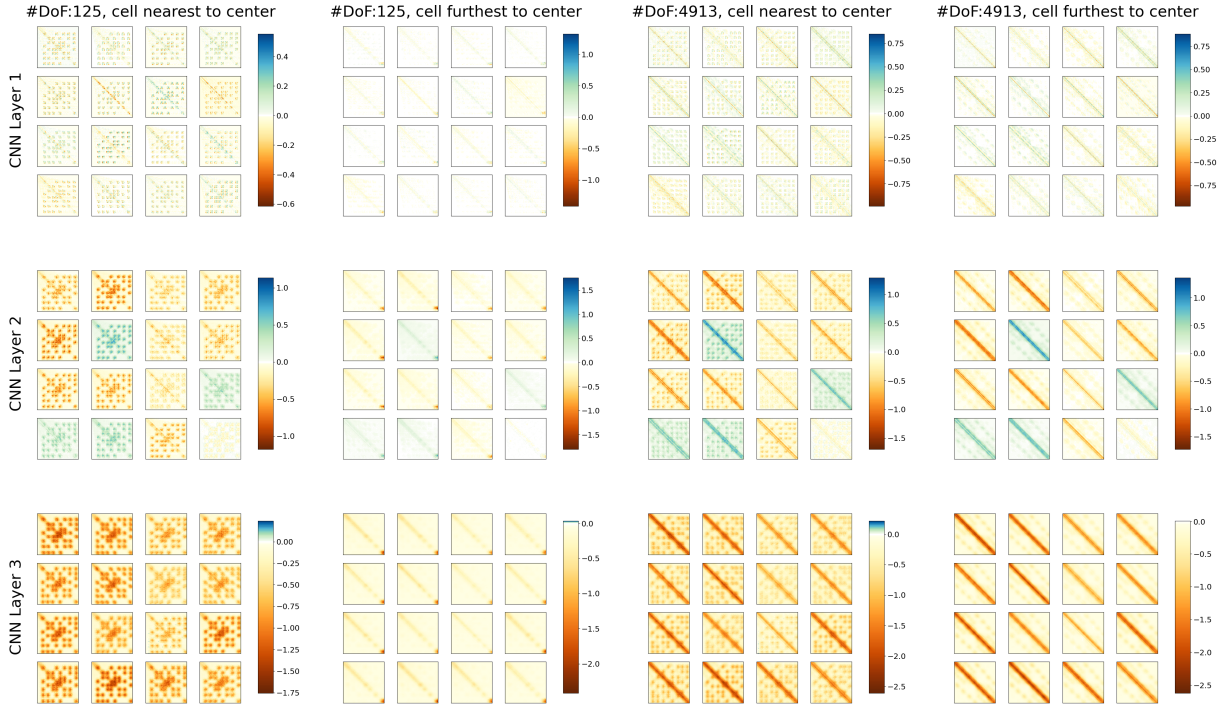


Figure 11: Example of feature maps of the CNN layers for the problems described in Section 4.2.1.

References

[1] J. H. Adler, T. R. Benson, E. C. Cyr, S. P. MacLachlan, and R. S. Tuminaro. Monolithic multigrid methods for two-dimensional resistive magnetohydrodynamics. *SIAM Journal on Scientific Computing*, 38(1):B1–B24, 2016.

[2] P. Antonietti, F. Dassi, and E. Manuzzi. Machine learning based refinement strategies for polyhedral grids with applications to virtual element and polyhedral discontinuous galerkin methods. *Journal of Computational Physics*, 469:111531, 2022.

[3] P. Antonietti, N. Farenga, E. Manuzzi, G. Martinelli, and L. Saverio. Agglomeration of polygonal grids using graph neural networks with applications to multigrid solvers. *arXiv preprint arXiv:2210.17457*, 2022.

[4] P. F. Antonietti, M. Caldana, and L. Dede'. Accelerating algebraic multigrid methods via artificial neural networks. *Vietnam Journal of Mathematics*, pages 1–36, 2023.

[5] P. F. Antonietti, P. Houston, X. Hu, M. Sarti, and M. Verani. Multigrid algorithms for hp hp-version interior penalty discontinuous galerkin methods on polygonal and polyhedral meshes. *Calcolo*, 54:1169–1198, 2017.

[6] P. F. Antonietti and E. Manuzzi. Refinement of polygonal grids using convolutional neural networks with applications to polygonal discontinuous galerkin and virtual element methods. *Journal of Computational Physics*, 452:110900, 2022.

[7] P. F. Antonietti, L. Mascotto, and M. Verani. A multigrid algorithm for the p-version of the virtual element method. *ESAIM: Mathematical Modelling and Numerical Analysis*, 52(1):337–364, 2018.

[8] P. F. Antonietti and L. Melas. Algebraic multigrid schemes for high-order nodal discontinuous galerkin methods. *SIAM Journal on Scientific Computing*, 42(2):A1147–A1173, 2020.

[9] P. F. Antonietti and G. Pennesi. V-cycle multigrid algorithms for discontinuous galerkin methods on non-nested polytopic meshes. *Journal of Scientific Computing*, 78(1):625–652, 2019.

[10] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The deal.II finite element library: Design, features, and insights. *Computers & Mathematics with Applications*, 81:407–422, 2021.

[11] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang. Scaling hypre's multigrid solvers to 100,000 cores. *High-performance scientific computing: algorithms and applications*, pages 261–279, 2012.

[12] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, and S. Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64:525–545, 2019.

[13] P. B. Bochev, C. J. Garasi, J. J. Hu, A. C. Robinson, and R. S. Tuminaro. An improved algebraic multigrid method for solving maxwell's equations. *SIAM Journal on Scientific Computing*, 25(2):623–642, 2003.

[14] J. H. Bramble. *Multigrid Methods*. Chapman and Hall/CRC, 2019.

[15] J. H. Bramble, J. E. Pasciak, J. P. Wang, and J. Xu. Convergence estimates for multigrid algorithms without regularity assumptions. *Mathematics of Computation*, 57(195):23–45, 1991.

[16] A. Brandt. Algebraic multigrid (AMG) for sparse matrix eqations. *Sparsity and its Applications*, pages 257–284, 1984.

[17] A. Brandt. Algebraic multigrid theory: The symmetric case. *Applied Mathematics and Computation*, 19(1-4):23–56, 1986.

[18] A. Brandt, S. MacCormick, and J. Ruge. *Algebraic Multigrid (AMG) for Automatic Multigrid Solution with Application to Geodetic Computations*. National Geodetic Survey and Air Force Office of Scientific Research and National Science Foundation, 1983.

[19] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge. Algebraic multigrid based on element interpolation (AMGe). *SIAM Journal on Scientific Computing*, 22(5):1570–1592, 2001.

[20] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM, 2000.

[21] T. Chartier, R. D. Falgout, V. Henson, J. Jones, T. Manteuffel, S. McCormick, J. Ruge, and P. S. Vassilevski. Spectral AMGe (ρ AMGe). *SIAM Journal on Scientific Computing*, 25(1):1–26, 2003.

[22] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[23] M. Eichinger, A. Heinlein, and A. Klawonn. Stationary flow predictions using convolutional neural networks. In *Numerical Mathematics and Advanced Applications ENUMATH 2019: European Conference, Egmond aan Zee, The Netherlands, September 30-October 4*, pages 541–549. Springer, 2020.

[24] R. D. Falgout. An introduction to algebraic multigrid. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2006.

[25] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov. On two-grid convergence estimates. *Numerical linear algebra with applications*, 12(5-6):471–494, 2005.

[26] R. D. Falgout and U. M. Yang. hypre: A library of high performance preconditioners. In *International Conference on computational science*, pages 632–641. Springer, 2002.

[27] S. Fresca, L. Dede', and A. Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *Journal of Scientific Computing*, 87:1–36, 2021.

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

[29] M. Griebel, D. Oeltz, and M. A. Schweitzer. An algebraic multigrid method for linear elasticity. *SIAM Journal on Scientific Computing*, 25(2):385–407, 2003.

[30] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[31] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern Recognition*, pages 770–778, 2016.

[32] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Machine learning in adaptive domain decomposition methods—predicting the geometric location of constraints. *SIAM Journal on Scientific Computing*, 41(6):A3887–A3912, 2019.

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

[34] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[35] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5MB model size. *arXiv preprint arXiv:1602.07360*, 2016.

[36] J. E. Jones and P. S. Vassilevski. AMGe based on element agglomeration. *SIAM Journal on Scientific Computing*, 23(1):109–133, 2001.

[37] A. Katrutsa, T. Daulbaev, and I. Oseledets. Black-box learning of multigrid parameters. *Journal of Computational and Applied Mathematics*, 368:112524, 2020.

[38] T. V. Kolev and P. S. Vassilevski. Parallel auxiliary space amg for h (curl) problems. *Journal of Computational Mathematics*, pages 604–623, 2009.

[39] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[40] G. Kutyniok, P. Petersen, M. Raslan, and R. Schneider. A theoretical analysis of deep neural networks and parametric PDEs. *Constructive Approximation*, 55(1):73–125, 2022.

[41] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks*, 3361(10):1995, 1995.

[42] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

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

[44] S. Mishra. A machine learning framework for data driven acceleration of computations of differential equations. *arXiv preprint arXiv:1807.09519*, 2018.

[45] N. S. Moore, E. Cyr, and C. Siefert. Learning an algebriac multrigrid interpolation operator using a modified graphnet architecture. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2022.

[46] A. Quarteroni and A. Valli. *Numerical approximation of partial differential equations*, volume 23. Springer Science & Business Media, 2008.

[47] M. Raw. Robustness of coupled Algebraic Multigrid for the Navier-Stokes equations. In *34th Aerospace sciences meeting and exhibit*, page 297, 1996.

[48] F. Regazzoni, L. Dede', and A. Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *Journal of Computational Physics*, 397:108852, 2019.

[49] F. Regazzoni, L. Dede', and A. Quarteroni. Machine learning of multiscale active force generation models for the efficient simulation of cardiac electromechanics. *Computer Methods in Applied Mechanics and Engineering*, 370:113268, 2020.

[50] F. Regazzoni, M. Salvador, L. Dedè, and A. Quarteroni. A machine learning method for real-time numerical simulations of cardiac electromechanics. *Computer Methods in Applied Mechanics and Engineering*, 393:114825, 2022.

[51] J. W. Ruge and K. Stüben. Algebraic multigrid. In *Multigrid methods*, pages 73–130. SIAM, 1987.

[52] A. Savitzky and M. J. Golay. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964.

[53] A. Taghibakhshi, S. MacLachlan, L. Olson, and M. West. Optimization-based algebraic multigrid coarsening using reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12129–12140, 2021.

[54] U. Trottenberg, C. W. Oosterlee, and A. Schuller. *Multigrid*. Elsevier, 2000.

[55] P. Vaněk, M. Brezina, and J. Mandel. Convergence of algebraic multigrid based on smoothed aggregation. *Numerische Mathematik*, 88:559–579, 2001.

[56] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996.

[57] P. S. Vassilevski. *Multilevel block factorization preconditioners: Matrix-based analysis and algorithms for solving finite element equations*. Springer Science & Business Media, 2008.

[58] R. Vinuesa and S. L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022.

[59] J. M. Weiss, J. P. Maruszewski, and W. A. Smith. Implicit solution of preconditioned navier-stokes equations using algebraic multigrid. *AIAA Journal*, 37(1):29–36, 1999.

[60] P. Wesseling. *An Introduction to Multigrid Methods*. An Introduction to Multigrid Methods. R.T. Edwards, 2004.

[61] J. A. White, N. Castelletto, S. Klevtsov, Q. M. Bui, D. Osei-Kuffuor, and H. A. Tchelepi. A two-stage preconditioner for multiphase poromechanics in reservoir simulation. *Computer Methods in Applied Mechanics and Engineering*, 357:112575, 2019.

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

[63] J. Xu and L. Zikatanov. The method of alternating projections and the method of subspace corrections in hilbert space. *Journal of the American Mathematical Society*, 15(3):573–597, 2002.

[64] J. Xu and L. Zikatanov. Algebraic multigrid methods. *Acta Numerica*, 26:591–721, 2017.

[65] U. M. Yang et al. Boomeramg: A parallel algebraic multigrid solver and preconditioner. *Applied Numerical Mathematics*, 41(1):155–177, 2002.

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

33/2023 Caldana, M.; Antonietti, P. F.; Dede', L.
A Deep Learning algorithm to accelerate Algebraic Multigrid methods in Finite Element solvers of 3D elliptic PDEs

32/2023 Gambarini, M.; Ciaramella, G.; Miglio, E.; Vanzan, T.
Robust optimization of control parameters for WEC arrays using stochastic methods

31/2023 Orlando, G.
Assessing ChatGPT for coding finite element methods

30/2023 Antonietti, P. F.; Bonizzoni, F.; Verani, M.
A DG-VEM method for the dissipative wave equation

28/2023 Zingaro, A.; Vergara, C.; Dede', L.; Regazzoni, F.; Quarteroni, A.
A comprehensive mathematical model for myocardial perfusion

29/2023 Carbonaro, D.; Mezzadri, F.; Ferro, N.; De Nisco, G.; Audenino, A.L.; Gallo, D.; Chiastra, C.; Morbiducci, U.; Perotto, S.
Design of innovative self-expandable femoral stents using inverse homogenization topology optimization

Fumagalli, A.; Panzeri, L.; Formaggia, L.; Scotti, A.; Arosio, D.
A mixed-dimensional model for direct current simulations in presence of a thin high-resistivity liner

27/2023 Beirao da Vega, L.; Canuto, C.; Nochetto, R.H.; Vacca, G.; Verani, M.
Adaptive VEM for variable data: convergence and optimality

26/2023 Artoni, A.; Antonietti, P.F.; Corradi, R.; Mazzieri, I.; Parolini, N.; Rocchi, D.; Schito P.; Semeraro, F.F.;
AeroSPEED: a high order acoustic solver for aeroacoustic applications

25/2023 Bonetti, S.; Botti, M.; Mazzieri, I.; Antonietti, P.F.
Numerical modelling of wave propagation phenomena in thermo-poroelastic media via discontinuous Galerkin methods