



MOX-Report No. 60/2025

**Deep Learning Accelerated Algebraic Multigrid Methods for
Polytopal Discretizations of Second-Order Differential Problems**

Antonietti, P. F.; Caldana, M.; Gentile, L.; Verani M.

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

mox-dmat@polimi.it

<https://mox.polimi.it>

Deep Learning Accelerated Algebraic Multigrid Methods for Polytopal Discretizations of Second-Order Differential Problems

Paola F. Antonietti^a, Matteo Caldana^a, Lorenzo Gentile^a, Marco Verani^a

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

October 1, 2025

Abstract

Algebraic Multigrid (AMG) methods are state-of-the-art algebraic solvers for partial differential equations. Still, their efficiency depends heavily on the choice of suitable parameters and/or ingredients. Paradigmatic examples include the so-called strong threshold parameter θ , which controls the algebraic coarse-grid hierarchy, as well as the smoother, i.e., the relaxation methods used on the fine grid to damp out high-frequency errors. In AMG, since the coarse grids are constructed algebraically (without geometric intuition), the smoother's performance is even more critical. For the linear systems stemming from polytopal discretizations, such as Polytopal Discontinuous Galerkin (PolyDG) and Virtual Element Methods (VEM), AMG sensitivity to such choices is even more critical due to the significant variability of the underlying meshes, which results in algebraic systems with different sparsity patterns. We propose a novel deep learning approach that automatically tunes the strong threshold parameter, as well as the smoother choice in AMG solvers, for linear systems of equations arising from polytopal discretizations, thereby maximizing AMG performance. We interpret the sparse matrix resulting from polytopal discretization as a grayscale image, and by applying pooling, our neural network extracts compact features that preserve the necessary information at a low computational cost. We test various differential problems in both two- and three-dimensional settings, with heterogeneous coefficients and polygonal/polyhedral meshes, and demonstrate that the proposed approach generalizes well. In practice, we demonstrate that we can reduce AMG solver time by up to 27% with minimal changes to existing PolyDG and VEM codes.

Key words: algebraic multigrid methods, polytopal grids, discontinuous Galerkin, virtual elements, deep learning, convolutional neural networks.

MSC subject classification: 65N22, 65N30, 65N55, 68T01.

1 Introduction

Multigrid methods are state-of-the-art iterative solvers for large, sparse, linear systems stemming from Finite Element discretizations of Partial Differential Equations (PDEs) [1]. A hierarchy of

“coarser” problems forms the key ingredient of Multigrid methods, which enables fast error reduction across multiple levels within iterative Krylov-based algorithms [2].

Within the framework of multigrid methods, Geometric Multigrid methods [3, 4] take advantage of the explicit geometry of the underlying PDEs and computational domain: a sequence of “coarser” problems is built based on employing coarser grids by directly coarsening the original mesh, which allows the method to use geometric information to define interpolation and restriction operators between levels. In contrast, Algebraic Multigrid (AMG) methods [5, 6, 7] do not require knowledge of the underlying geometry; they operate solely on the system of linear equations, analyzing (interpreting it as a weighted graph) the matrix structure to generate coarse levels and the corresponding transfer operators automatically. Therefore, AMG methods are more flexible for use as a “black box” or for problems posed on complicated geometries and/or with highly heterogeneous materials, where computing the sequence of coarser meshes can be challenging.

Over the past ten years, there has been significant progress in developing new discretization methods for PDEs, commonly referred to as polytopal methods, that extend the Finite Element approach to support computational domains partitioned into general polygons (in two dimensions) or polyhedra (in three dimensions) and that offer greater flexibility in handling complex geometries. Examples of polytopal methods include Virtual Element Methods (VEM), see e.g., [8, 9], and the review paper [10]; polytopal Discontinuous Galerkin (PolyDG), see e.g., [11, 12, 13, 14], the monograph [15] and the review paper [16]; Hybrid High-Order Methods, see e.g., [17, 18, 19], the monograph [20] and the review paper [21]; Mimetic Finite Difference Methods, see e.g., [22, 23, 24] and the monograph [25]; Hybridizable Discontinuous Galerkin, see e.g., [26, 27, 28], and weak Galerkin methods, see e.g., [29, 30]. While the development of polytopal discretizations has undergone a very rapid growth in the last ten years, the development of efficient solvers for the resulting large, sparse (and often ill-conditioned) linear systems of equations is still in its infancy.

A key advantage of polytopal methods is that they naturally support agglomeration, i.e., a coarser mesh of arbitrarily shaped elements is obtained by merging an underlying finer grid. Agglomeration significantly reduces computational cost by lowering the number of degrees of freedom required to achieve a given accuracy, while enabling the use of higher polynomial degrees. Mesh agglomeration, which lacks a direct equivalent in traditional finite elements, is crucial for reducing computational complexity in domains with “small” inclusions or complex embedded layers and structures. It also forms the basis of constructing efficient geometric multigrid solvers and adaptive algorithms. Indeed, in the context of multilevel solvers, agglomeration techniques form the basis for automatically generating the sequence of nested/non-nested coarser levels. Recent work [31] has demonstrated that employing Graph Neural Networks (GNNs) for automatic mesh agglomeration can improve the quality and efficiency of the agglomeration steps, with promising results and novel avenues in multigrid solvers. Polytopal mesh agglomeration via geometrical deep learning for two- and three-dimensional heterogeneous domains has been proposed in [31, 32]. Geometric and physical features are incorporated into the GNN model to preserve mesh quality better when aggregating elements in complex domains. Such a distinguishing and essential feature can be implemented at a much greater cost in classical graph-partitioning methods like METIS. R-tree-based agglomeration algorithms of polytopal grids with applications to multilevel methods has also been proposed recently in [33].

For PolyDG discretizations of second-order elliptic problems, geometric agglomeration-based

multigrid solvers have been developed and analysed in [34], where multigrid algorithms for hp -version PolyDG methods on agglomerated polygonal/polyhedral meshes, proving scalability independent of the discretization parameters. In [35], the authors proved uniform convergence of a non-nested agglomeration-driven geometric multigrid scheme, provided sufficient smoothing steps are applied. A massively parallel (agglomeration-based) two-level solver for PolyDG methods is proposed and analyzed in [36]. More recently, a geometric multigrid solver for compact DG discretizations was proposed in [37]. For Virtual Element discretizations of the diffusion problem, [38] proved that a p -multigrid algorithm, where the sequence of coarse levels is obtained by progressively reducing the polynomial degree, leads to a scalable solver. In [39], the authors presented a uniformly convergent geometric multigrid scheme for the lowest-order Virtual Element Method. In [40], the authors provide a numerical study of algebraic multigrid applied to VEM, demonstrating numerically that AMG can be effective for VEM discretizations over general meshes and heterogeneous diffusion. For Hybrid High-Order discretizations, a geometric multigrid method was proposed and analyzed in [41]. A comprehensive comparative study of h -, p -, and hp -coarsening strategies, considering both nested and non-nested mesh hierarchies, was presented in [42]. In the context of mixed or incompressible flow problems, p -multilevel preconditioners for the Stokes equations were investigated in [43].

While the flexibility of polytopal methods offers improved advantages in the context of geometric multigrid solvers, through the use of agglomeration, this generality also complicates things for AMG. Indeed, the sparsity pattern of the matrix may be more irregular; local element shape quality, face/edge connectivity, etc., may vary more, and interpolation/prolongation operators, and algebraic aggregation strategies have to handle more “variability”.

Since its introduction in [44], AMG methods have been widely studied from both algorithmic and theoretical perspectives. Developments such as smoothed aggregation [45, 46] and adaptive variants [47] have broadened its applicability to many finite element discretizations and physical models, including Ritz-type methods [48, 49, 50], discontinuous Galerkin [51, 52, 53], hybrid schemes [43, 54], and problems in fluid dynamics [55, 56], electromagnetism [57, 58], elasticity [59, 60, 61], and poromechanics [62, 63]. The theoretical foundations of AMG methods can be found in [7, 44, 64, 65, 66] for the two-level method; we also refer to the recent review paper [67]. It is well known that AMG performance depends on parameter tuning, including options for coarsening strategies, interpolation operators, and the choice of the smoother. AMG faces further challenges when applied to solve the linear system stemming from polytopal discretizations. The greater variability of the meshes, and consequently of the resulting algebraic systems, makes such a tuning both more difficult and more critical. For example, a poor choice of the strong threshold—governing the coarsening step—can severely degrade solver performance, whereas optimal tuning can dramatically accelerate convergence. Analogously, because AMG relies heavily on smoothers to damp high-frequency errors, the choice of the smoother is even more critical. In this paper, we propose a novel ANN-AMG method, where a deep learning algorithm predicts “on the fly” the optimal AMG strong threshold parameter driving the algebraic coarsening as well as the optimal choice of the smoother. Conceptually, our method, the ANN-AMG method, combines the advantages of adaptive AMG [47], which automatically refines interpolation patterns at additional cost, and calibrated AMG, which can achieve good performance but requires extensive manual tuning. More specifically, we propose using Artificial Neural Networks to automatically tune the strong threshold parameter and the best smoother (from a set of given relaxation methods), thereby minimizing the time-to-solution of the resulting iterative scheme. To achieve our goal, we first interpret the matrix of the algebraic system as a grayscale image. Next, a convolutional and pooling layer generates a compact, multi-channel rep-

resentation that preserves key structural features while simultaneously reducing computational cost. The proposed ANN-AMG method is entirely non-intrusive, requiring no modifications to existing PolyDG or VEM implementations, nor to AMG solvers, thereby guaranteeing compatibility with existing libraries and preserving parallel AMG implementations [68]. Our approach builds upon earlier work that integrated deep learning with multigrid to accelerate iterative solvers, see e.g., [69, 70] for classical Conforming Finite Element Discretizations. However, it extends these ideas in several crucial directions necessary for handling polytopal discretizations. More specifically, in the present work, we propose optimizing the AMG performance with respect to both the choice of the threshold parameter that drives the algebraic coarsening and the choice of the smoother. We introduce a novel pre-processing step to ensure robustness of training data. We also propose a novel ANN architecture with an additional output measuring the model’s prediction confidence. Finally, we propose an improved pooling strategy to capture better variability in the matrix structure and a new training acceleration strategy based on layer freezing. We validate our approach through extensive numerical experiments conducted on both two-dimensional and three-dimensional diffusion and elasticity differential problems, discretized using PolyDG and VEM on complex polygonal and polyhedral meshes. Results demonstrate that our ANN-AMG consistently lowers the computational costs by up to 30% compared to classical AMG with default or manually tuned parameters. We note that VEM and PolyDG discretizations are paradigmatic examples of polytopal methods. In VEM, the degrees of freedom are associated with “geometric” entities such as vertices, edges, faces, or internal moments. In contrast, in PolyDG methods, the local approximation space is “geometry-agnostic,” since a local modal polynomial expansion is used. Consequently, our results demonstrate that the proposed ANN-AMG algorithm can achieve acceleration regardless of whether the discretization space is “virtual and skeleton-based” or “plain polynomial and modal”. Furthermore, considering both diffusion and elasticity problems serves to show that ANN-AMG is robust for both scalar and vector-valued equations, the latter being particularly challenging within the AMG framework due to the need for suitable aggregation strategies. Finally, we emphasize that the computational overhead of the ANN forward pass is negligible compared to the acceleration it provides to the solver. Since the training phase is performed offline, the ANN-AMG algorithm can be generalized to a wide range of differential problems, resulting in symmetric and positive definite algebraic systems.

The remainder of this work is organized as follows. In Section 2, we review the basic principles of AMG methods, with a focus on the role of the strong threshold parameter. In Section 3, we discuss the PolyDG and VEM discretization for both diffusion and linear elasticity problems. Section 4 introduces our deep ANN-AMG algorithm, detailing its architecture and the matrix-to-image pooling representation. In Section 5.1 and Section 5.2, we test our ANN-AMG algorithm on a wide set of numerical benchmarks carried out with either PolyDG and VEM on both two- and three-dimensional diffusion and linear elasticity problems, respectively. Finally, Section 6 concludes with a discussion on the obtained results and outlines directions for future research.

2 Algebraic Multigrid Methods

This section introduces the key steps required to construct the AMG method’s hierarchy of grids and operators. We first describe the coarse–fine partitioning strategy that determines the grid structure, and then define the interpolation operator that transfers information between levels.

AMG [7, 44, 67] is an iterative method for solving large, sparse symmetric positive definite

Algorithm 1 One Iteration of the V-cycle of the AMG method

$\mathbf{u}^{(k)} = \text{vcycle}^k(\mathbf{u}^{(k)}, \mathbf{f}^{(k)}, \{(A^{(j)}, S_1^{(j)}, S_2^{(j)})\}_{j=k}^M, \{(I_j^{j+1}, I_{j+1}^j)\}_{j=k}^{M-1}, \nu_1, \nu_2)$

```

1: if  $k = M$  then
2:    $\mathbf{u}^{(M)} = \text{gaussian\_elimination}(A^{(M)}, \mathbf{f}^{(M)})$ 
3: else
4:    $\mathbf{u}^{(k)} \leftarrow \text{smooth}^{\nu_1}(A^{(k)}, S_1^{(k)}, \mathbf{u}^{(k)}, \mathbf{f}^{(k)})$ 
5:    $\mathbf{r}^{(k+1)} \leftarrow I_k^{k+1}(\mathbf{f}^{(k)} - A^{(k)}\mathbf{u}^{(k)})$ 
6:    $\mathbf{e}^{(k+1)} \leftarrow \text{vcycle}^{k+1}(\mathbf{u}^{(k)}, \mathbf{f}^{(k)}, \{(A^{(j)}, S_1^{(j)}, S_2^{(j)})\}_{j=k+1}^M, \{(I_j^{j+1}, I_{j+1}^j)\}_{j=k+1}^{M-1}, \nu_1, \nu_2)$ 
7:    $\mathbf{u}^{(k)} \leftarrow \mathbf{u}^{(k)} + I_{k+1}^k \mathbf{e}^{(k+1)}$ 
8:    $\mathbf{u}^{(k)} \leftarrow \text{smooth}^{\nu_2}(A^{(k)}, S_2^{(k)}, \mathbf{u}^{(k)}, \mathbf{f}^{(k)})$ 
9: end if

```

(SPD) linear systems of the form

$$A\mathbf{u} = \mathbf{f}, \quad (1)$$

where $A \in \mathbb{R}^{n \times n}$ and $\mathbf{f} \in \mathbb{R}^n$. The method constructs a hierarchy of smaller systems to reduce error across different frequency modes efficiently. Its main components are a smoother, grid transfer operators, and a sequence of coarse-grid operators.

The smoother is a simple iterative solver applied for ν steps, $\mathbf{u}_{k+1} = \mathbf{u}_k + S(\mathbf{f} - A\mathbf{u}_k)$, $k \geq 0$, starting from an initial guess $\mathbf{u}_0$. The matrix $S \in \mathbb{R}^{n \times n}$ defines the method (smoother) that is designed to damp high-frequency components of the error. This operation is denoted as $\text{smooth}^\nu(A, S, \mathbf{u}_0, \mathbf{f})$.

To address low-frequency error, the problem is transferred to a coarser grid. This is achieved using interpolation operators $I_k^{k-1} \in \mathbb{R}^{n_{k-1} \times n_k}$ and restriction operators $I_{k-1}^k \in \mathbb{R}^{n_k \times n_{k-1}}$. For the SPD case, these operators are defined recursively for levels $k = 1, \dots, M-1$ alongside the coarse-grid system matrices $A^{(k+1)}$:

$$I_k^{k+1} = (I_{k+1}^k)^\top, \quad A^{(k+1)} = I_k^{k+1} A^{(k)} I_{k+1}^k, \quad \forall k = 1, \dots, M-1, \quad A^{(1)} = A. \quad (2)$$

The system sizes decrease at each level, $n = n_1 > n_2 > \dots > n_M$. These components are assembled in a recursive procedure, such as the V-cycle shown in Algorithm 1. The V-cycle projects the residual equation onto a coarse space ($A\mathbf{e} = \mathbf{r} = \mathbf{f} - A\mathbf{u}_\nu$), solves the problem there, and interpolates the correction back to the fine space ($\mathbf{u}_\nu + \mathbf{e}$). It uses pre-smoothers $S_1^{(k)}$ for ν_1 steps and post-smoothers $S_2^{(k)}$ for ν_2 steps at each level (k).

2.1 Coarse-Fine Partitioning

The core of AMG is the automated construction of the interpolation operator based on the entries of the matrix A . For clarity, we will describe the construction for a two-level system, omitting the level-identifying superscripts and subscripts k . First, the set of variables $\mathcal{N} = \{1, \dots, n\}$ is partitioned into a set of coarse-grid variables $\mathcal{C}$ and a set of fine-grid variables $\mathcal{F}$. Variables in $\mathcal{C}$ will exist on the next coarser grid, while variables in $\mathcal{F}$ will be interpolated from them. To perform the $\mathcal{C}/\mathcal{F}$ splitting, one must first quantify the coupling between variables based on the matrix entries. Namely, it is crucial to know when a variable i can be interpolated from a variable j , or formally, when the variable i *strongly depends* on the variable j .

Definition 2.1 Given a threshold parameter $0 < \theta \leq 1$, the set of variables on which variable i strongly depends, denoted $\mathcal{S}_i$, is

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

Conversely, we define the set of variables that are strongly influenced *by* the variable i . This “transpose” set, $\mathcal{S}_i^\top$, is given by $\mathcal{S}_i^\top = \{j : i \in \mathcal{S}_j, j = 1, \dots, n_k\}$. Both sets are fundamental for constructing the coarsening and interpolation operators.

We use the CLJP (Cleary-Luby-Jones-Plassman) algorithm to partition the grid. This algorithm models the strong dependencies as a graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$, where an edge $(i, j) \in \mathcal{E}$ exists if $j \in \mathcal{S}_i$. It then iteratively selects an independent set of nodes to form $\mathcal{C}$. A weight $\eta(i) = |\mathcal{S}_i^\top| + \tilde{\eta}$ is assigned to each node i , where $\tilde{\eta}$ is a small random number to break ties. The algorithm proceeds until all nodes are classified as either $\mathcal{C}$ or $\mathcal{F}$. This process is repeated to build the operator hierarchy until the grid size n_k is below a threshold, here set to two.

2.2 Interpolation Operator

With the $\mathcal{C}/\mathcal{F}$ partition established, the interpolation operator I_{k+1}^k is defined as follows. For any vector $\mathbf{x} \in \mathbb{R}^{n_{k+1}}$ on the coarse grid, its interpolated counterpart on the fine grid is given by:

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

where $\mathcal{C}_i^k = \{j \in \mathcal{C}^k : a_{ij} \neq 0\}$ is the set of coarse neighbors of a fine point i , and ω_{ij}^k are the interpolation weights.

The weights are derived from the assumption that the error $\mathbf{e}$ is smooth, meaning $(A\mathbf{e})_i \approx 0$ for all i . This condition can be written as:

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

where $\mathcal{D}_i^s = \mathcal{F} \cap \mathcal{S}_i$ are the strongly connected fine-grid neighbors and $\mathcal{D}_i^w = \{j \in \mathcal{N} : a_{ij} \neq 0, j \notin \mathcal{S}_i\}$ are the weakly connected neighbors. This leads to the following formula for the weights:

$$\omega_{ij} = -\frac{1}{a_{ij} + \sum_{l \in \mathcal{D}_i^w} a_{il}} \left(a_{ij} + \sum_{l \in \mathcal{D}_i^s} \frac{a_{il} \hat{a}_{lj}}{\sum_{m \in \mathcal{C}_i} \hat{a}_{lm}} \right) \quad (4)$$

where $\hat{a}_{ij}$ is defined as a_{ij} if $a_{ij}a_{ii} \leq 0$ and zero otherwise. The complete AMG setup procedure is summarized in Algorithm 2.

3 Model Problems their Polytopal Discretizations

In this section, we introduce the model problems under consideration, the basic notation for PolyDG and VEM discretizations, and discuss the corresponding discrete formulations for diffusion and linear elasticity differential problems that lead to linear systems of the form (1).

Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$ be an open, bounded polygonal/polyhedral domain. The first model problem that we consider is the diffusion equation

$$\begin{aligned} -\nabla \cdot (\kappa \nabla u) &= f & \text{in } \Omega, \\ u &= 0 & \text{on } \partial\Omega, \end{aligned} \quad (5)$$

Algorithm 2 AMG algorithm

$\mathbf{u} = \text{AMG}(\mathbf{u}, A, \mathbf{f}, \theta, \{(S_1^{(j)}, S_2^{(j)})\}_{j=k}^M, \nu_1, \nu_2, N_{max}, tol)$

- 1: build $\{\mathcal{C}^k, \mathcal{F}^k\}_{k=1}^M$ using θ by means of CLJP
 - 2: build the operators $\{I_{j+1}^j\}_{j=k}^{M-1}$ employing Eq. (3) and Eq. (4)
 - 3: build the operators $\{I_j^{j+1}\}_{j=k}^{M-1}$ and $\{(A^{(j)})\}_{j=k}^M$ by means of Eq. (2)
 - 4: **while** $k < N_{max}$ **and** $\|A\mathbf{u} - \mathbf{f}\| / \|\mathbf{f}\| < tol$ **do**
 - 5: $\mathbf{u} \leftarrow \text{vcycle}^1(\mathbf{u}, \mathbf{f}, \{(A^{(j)}, S_1^{(j)}, S_2^{(j)})\}_{j=k}^M, \{(I_j^{j+1}, I_{j+1}^j)\}_{j=k}^{M-1}, \nu_1, \nu_2)$
 - 6: $k \leftarrow k + 1$
 - 7: **end while**
-

where the datum $f \in L^2(\Omega)$ and the function $\kappa : \Omega \rightarrow \mathbb{R}$ is bounded and uniformly positive. For the sake of simplicity, in the following we will assume that κ is piecewise constant over Ω with discontinuities aligned with the mesh.

The second model problem under investigation is the following: find the displacement field $\mathbf{u} : \Omega \rightarrow \mathbb{R}^d$ such that

$$\begin{aligned} -\nabla \cdot \underline{\boldsymbol{\sigma}}(\mathbf{u}) &= \mathbf{f} & \text{in } \Omega, \\ \mathbf{u} &= \mathbf{0} & \text{on } \partial\Omega, \end{aligned} \quad (6)$$

where $\mathbf{f} : \Omega \rightarrow \mathbb{R}^d$ is the body force, and $\underline{\boldsymbol{\sigma}}(\mathbf{u})$ is the Cauchy stress tensor. For isotropic elastic materials, $\underline{\boldsymbol{\sigma}}(\mathbf{u}) = \mathbb{C} : \underline{\boldsymbol{\varepsilon}}(\mathbf{u}) = 2\mu \underline{\boldsymbol{\varepsilon}}(\mathbf{u}) + \lambda \text{tr}(\underline{\boldsymbol{\varepsilon}}(\mathbf{u}))\mathbf{I}$, where $\underline{\boldsymbol{\varepsilon}}(\mathbf{u})$ is the strain tensor, $\lambda, \mu > 0$ are the Lamé parameters, and $\mathbf{I}$ is the identity matrix. The Lamé parameters λ and μ can be expressed in terms of the Young's modulus E and Poisson's ratio ν as follows:

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad (7)$$

respectively.

We next introduce the basic notation, which is instrumental to our discretizations. Let $\mathcal{T}_h$ be a *polytopal* mesh made of general polygons (for $d = 2$) or polyhedra (for $d = 3$). We denote such polytopal elements by K , define by h_K their diameter, and set $h = \max_{K \in \mathcal{T}_h} h_K$. To deal with polygonal and polyhedral elements, we define an *interface* of $\mathcal{T}_h$ as the intersection of the $(d-1)$ -dimensional faces of any two neighboring elements of $\mathcal{T}_h$. If $d = 2$, an interface/face is a line segment and the set of all interfaces/faces is denoted by $\mathcal{F}_h$. When $d = 3$, an interface can be a general polygon that we assume could be further decomposed into a set of planar triangles collected in the set $\mathcal{F}_h$. Let F be an interior face shared by two neighboring elements $K^\pm \in \mathcal{T}_h$, i.e., $F = \partial K^+ \cap \partial K^-$. For regular enough scalar-, vector-, and tensor-valued functions v , $\mathbf{w}$, and $\underline{\boldsymbol{\tau}}$, we define the average and jump operators as

$$\begin{aligned} \llbracket v \rrbracket &= v^+ \mathbf{n}^+ + v^- \mathbf{n}^-, & \llbracket \mathbf{w} \rrbracket &= \mathbf{w}^+ \otimes \mathbf{n}^+ + \mathbf{w}^- \otimes \mathbf{n}^-, \\ \{ \{ v \} \} &= \frac{v^+ + v^-}{2}, & \{ \{ \mathbf{w} \} \} &= \frac{\mathbf{w}^+ + \mathbf{w}^-}{2}, & \{ \{ \underline{\boldsymbol{\tau}} \} \} &= \frac{\underline{\boldsymbol{\tau}}^+ + \underline{\boldsymbol{\tau}}^-}{2}. \end{aligned} \quad (8)$$

The notation $(\cdot)^\pm$ is used to denote the trace on F taken within the interior of $K^\pm$, $\mathbf{n}^\pm$ is the outer unit normal vector to $\partial K^\pm$, and $\mathbf{w} \otimes \mathbf{n} = \mathbf{w} \mathbf{n}^T$. On boundary faces, the definition extends as in [71], i.e.,

$$\llbracket v \rrbracket = v \mathbf{n}, \quad \{ \{ v \} \} = v, \quad \llbracket \mathbf{w} \rrbracket = \mathbf{w} \otimes \mathbf{n}, \quad \{ \{ \mathbf{w} \} \} = \mathbf{w}, \quad \{ \{ \underline{\boldsymbol{\tau}} \} \} = \underline{\boldsymbol{\tau}}.$$

3.1 PolyDG Discretization of Diffusion and Linear Elasticity

We first introduce the scalar broken (discontinuous) polynomial space that serves as the discretization space for PolyDG methods, as

$$V_h^{\text{DG}} = \{v \in L^2(\Omega) : v|_K \in \mathbb{P}^p(K) \quad \forall K \in \mathcal{T}_h\}, \quad (9)$$

where $\mathbb{P}^p(K)$ denotes the space of polynomials of total degree at most $p \geq 1$. We also need its vector-valued counterpart defined as $\mathbf{V}_h^{\text{DG}} = [V_h^{\text{DG}}]^d$, $d = 2, 3$.

The PolyDG approximation to the model problem (5) reads as follow: find $u_h \in V_h^{\text{DG}}$ such that

$$\mathcal{A}_D^{\text{DG}}(u_h, v_h) = \sum_{K \in \mathcal{T}_h} \int_K f v_h \quad \forall v_h \in V_h^{\text{DG}} \quad (10)$$

where

$$\begin{aligned} \mathcal{A}_D^{\text{DG}}(u_h, v_h) = & \sum_{K \in \mathcal{T}_h} \int_K \kappa \nabla_h u_h \cdot \nabla_h v_h - \sum_{F \in \mathcal{F}_h} \int_F \{\kappa \nabla_h u_h\} \cdot \llbracket v_h \rrbracket \\ & - \sum_{F \in \mathcal{F}_h} \int_F \llbracket u_h \rrbracket \cdot \{\kappa \nabla_h v_h\} + \sum_{F \in \mathcal{F}_h} \int_F \sigma \llbracket u_h \rrbracket \cdot \llbracket v_h \rrbracket. \end{aligned}$$

Here, ∇_h denotes the element-wise gradient operator, and the penalty function is defined as

$$\sigma|_F = \begin{cases} \gamma \left\{ \kappa \frac{p_K^2}{h_K} \right\}_{\mathbb{H}}, & F \in \mathcal{F}_h^I \\ \gamma \kappa \frac{p_K^2}{h_K}, & F \in \mathcal{F}_h^B, \end{cases}$$

with $\gamma > 0$ denoting a penalty coefficient to be properly chosen (large enough) and $\{\cdot\}_{\mathbb{H}}$ denoting the harmonic average.

As for the model problem (6), following [72], we introduce

$$\begin{aligned} \mathcal{A}_E^{\text{DG}}(\mathbf{u}_h, \mathbf{v}_h) = & \sum_{K \in \mathcal{T}_h} \int_K \underline{\boldsymbol{\sigma}}(\mathbf{u}_h) : \underline{\boldsymbol{\varepsilon}}(\mathbf{v}_h) - \sum_{F \in \mathcal{F}_h} \int_F \{\underline{\boldsymbol{\sigma}}(\mathbf{u}_h)\} : \llbracket \mathbf{v}_h \rrbracket \\ & - \sum_{F \in \mathcal{F}_h} \int_F \llbracket \mathbf{u}_h \rrbracket : \{\underline{\boldsymbol{\sigma}}(\mathbf{v}_h)\} + \sum_{F \in \mathcal{F}_h} \int_F \eta \llbracket \mathbf{u}_h \rrbracket : \llbracket \mathbf{v}_h \rrbracket, \end{aligned}$$

with η as in [16, eq. (9)]. Hence, the PolyDG approximation to the elasticity problem (6) reads as follows: find $\mathbf{u}_h \in \mathbf{V}_h^{\text{DG}}$ such that

$$\mathcal{A}_E^{\text{DG}}(\mathbf{u}_h, \mathbf{v}_h) = \sum_{K \in \mathcal{T}_h} \int_K \mathbf{f} : \mathbf{v}_h \quad \forall \mathbf{v}_h \in \mathbf{V}_h^{\text{DG}}. \quad (11)$$

Remark 1 (*PolyDG algebraic form*) We observe that each of the discrete problems (10)-(11) yields a linear system of the form (1), where A and $\mathbf{f}$ denote the PolyDG stiffness matrix and the right-hand side corresponding to the chosen bilinear form and functional, respectively, and $\mathbf{u}$ is the corresponding vector of expansion coefficients in the selected basis for the discrete spaces V_h^{DG} and $\mathbf{V}_h^{\text{DG}}$, respectively.

3.2 VEM Discretization of Diffusion and Linear Elasticity

Following [8, 73, 74], in this section, we present the VEM for the diffusion (5) and linear elasticity (6) problems, focusing for the sake of simplicity to the two-dimensional case; we refer to [75] and [76] for the details on the three-dimensional extension.

For any $K \in \mathcal{T}_h$, we first introduce the following scalar- and vector-valued *local* virtual element spaces on K is given by

$$V_h^{\text{VEM}}(K) = \{v \in H^1(K) : v|_F \in \mathbb{P}_p(F) \ \forall F \subset \partial K, \ \Delta v \in \mathbb{P}_{p-2}(K)\}, \quad \mathbf{V}_h^{\text{VEM}}(K) = [V_h^{\text{VEM}}(K)]^2,$$

with the convention that for $p = 1$, $\mathbb{P}_{-1} = \{0\}$. From the above definitions, we introduce the global spaces V_h^{VEM} as $\mathbf{V}_h^{\text{VEM}}$ as

$$V_h^{\text{VEM}} = \{v \in H_0^1(\Omega) : v|_K \in V_h^{\text{VEM}}(K) \ \forall K \in \mathcal{T}_h\}, \quad \mathbf{V}_h^{\text{VEM}} = [V_h^{\text{VEM}}]^2.$$

A common choice of degrees of freedom for V_h^{VEM} , see, e.g., [77], is:

1. values of v at the vertices of K ,
2. on each $F \subset \partial K$ the moments of v up to order $p - 2$, for $p \geq 2$,
3. the internal moments of v up to order $p - 2$, for $p \geq 2$.

This choice of degrees of freedom is unisolvent for the space V_h^{VEM} and allows the computation of the following projection operator

$$\Pi_p^\nabla : V_h^{\text{VEM}}(K) \rightarrow \mathbb{P}_p(K) \quad \int_K \nabla(\Pi_p^\nabla v - v) \cdot \nabla q = 0 \quad \forall q \in \mathbb{P}_p(K),$$

with an additional constraint fixing the constant mode (e.g., element mean). For $\mathbf{V}_h^{\text{VEM}}(K)$, we can define the degrees of freedom analogously (componentwise) and the elastic energy projector as

$$\Pi_p^\nabla : \mathbf{V}_h^{\text{VEM}}(K) \rightarrow [\mathbb{P}_p(K)]^2 \quad \int_K \left(\boldsymbol{\sigma}(\Pi_p^\nabla \mathbf{v} - \mathbf{v}) \right) : \boldsymbol{\varepsilon}(\mathbf{q}) = 0 \quad \forall \mathbf{q} \in [\mathbb{P}_p(K)]^2,$$

together with the orthogonality/rigid-motion constraint to fix rigid body modes (so that the projector is unique).

The Virtual Element discretization of (5) reads: Find $u_h \in V_h^{\text{VEM}}$ such that

$$\mathcal{A}_D^{\text{VEM}}(u_h, v_h) = \sum_{K \in \mathcal{T}_h} \int_K f_h v_h \quad \forall v_h \in V_h^{\text{VEM}},$$

where f_h is a suitable polynomial projection of f , which is computable using the available degrees of freedom, and where

$$\mathcal{A}_D^{\text{VEM}}(w_h, v_h) = \sum_{K \in \mathcal{T}_h} \int_K \kappa_K \nabla(\Pi_p^\nabla w) \cdot \nabla(\Pi_p^\nabla v) + \sum_{K \in \mathcal{T}_h} S_D^K((I - \Pi_p^\nabla)w, (I - \Pi_p^\nabla)v) \quad w, v \in V_h^{\text{VEM}},$$

where $S_D^K(\cdot, \cdot)$ is a *computable* symmetric positive definite stabilization form acting on the “non-polynomial” part of $(I - \Pi_p^\nabla)$ (typical choices for S_D^K include scaled inner products on the degrees of freedom, the so-called “dof-dof” stabilization).

As for the Virtual Element discretization of the linear elasticity problem (6). Find $\mathbf{u}_h \in \mathbf{V}_h^{\text{VEM}}$ such that

$$\mathcal{A}_E^{\text{VEM}}(\mathbf{u}_h, \mathbf{v}_h) = \sum_{K \in \mathcal{T}_h} \int_K \mathbf{f}_h \cdot \mathbf{v}_h|_K \quad \forall \mathbf{v}_h \in \mathbf{V}_h^{\text{VEM}}$$

where $\mathbf{f}_h$ is a suitable polynomial projection of f , which is computable using the available degrees of freedom. The computable bilinear form $\mathcal{A}_E^{\text{VEM}}(\cdot, \cdot)$ is defined as

$$\mathcal{A}_E^{\text{VEM}}(\mathbf{w}_h, \mathbf{v}_h) = \sum_{K \in \mathcal{T}_h} \int_K \underline{\boldsymbol{\sigma}}(\Pi_{\mathbf{p}}^{\nabla} \mathbf{w}) : \underline{\boldsymbol{\varepsilon}}(\Pi_{\mathbf{p}}^{\nabla} \mathbf{v}) + \sum_{K \in \mathcal{T}_h} S_E^K((\mathbf{I} - \Pi_{\mathbf{p}}^{\nabla})\mathbf{w}, (\mathbf{I} - \Pi_{\mathbf{p}}^{\nabla})\mathbf{v})$$

for all $\mathbf{w}_h, \mathbf{v}_h \in \mathbf{V}_h^{\text{VEM}}$ where $S_E^K(\cdot, \cdot)$ is a symmetric positive definite stabilization acting on the kernel of the projector (i.e., the non-polynomial part).

Remark 2 (*VEM algebraic form*) As before, the Virtual Element discretization of both the diffusion and linear elasticity problem yields a linear system of the form (1), where A and $\mathbf{f}$ denote the VEM stiffness matrix and the right-hand side corresponding to the chosen bilinear form and right hand side, respectively, and $\mathbf{u}$ is the corresponding expansion coefficient vectors for the discrete Virtual Element spaces V_h^{VEM} or $\mathbf{V}_h^{\text{VEM}}$, depending on the discretized differential problem.

4 The Neural Network Architecture

In this section, we provide a detailed description of the neural network architecture and the algorithm used to tune the AMG method automatically. First, we provide an overview of the pipeline, introducing the high-level components and procedures that enable the automatic choice of parameters. Then, in each subsection, we will go into the details and precisely define all the components of our method.

The first choice concerns the parameters to be tuned and the employed metric (that is, a scalar index that establishes when a certain combination of parameters is better than another). In Section 2, we have shown that the strong threshold parameter θ critically influences the construction of the interpolation operator and thus the whole hierarchy of levels that stand at the basis of each AMG application. Another key choice that heavily influences the behavior of the AMG methods is the smoother. The importance of these parameters is confirmed by empirical evidence when tested for our applications. For these reasons, we aim to tune θ and the smoother to minimize the computational cost of the AMG method. We remark that the tuning process can be done by optimizing the choice of other parameters, such as the number of pre- and post-smoothing iterations applied at each level, or the choice of $\mathcal{C}/\mathcal{F}$ splitting algorithm.

Concerning the performance metric, we ideally want to minimize the total elapsed time needed to solve the linear system. However, this metric has at least two significant drawbacks: it is machine-dependent and it suffers from measurement error. Techniques to tackle these problems are detailed in Section 4.1. However, in some instances, statistical analysis shows a strong correlation between the elapsed time and the approximate convergence factor

$$\rho = \left(\frac{\|\mathbf{r}^{(k)}\|_2}{\|\mathbf{r}^{(0)}\|_2} \right)^{N_{it}} \quad (12)$$

which measures how quickly the iterative solution contracts towards the exact one. Hence, when possible, we prefer to use ρ as the index of the performance. For each test case, we also considered multivariate polynomial models that take into account the size and number of non-zero elements

of $A^{(k)}$ at each level k . Unfortunately, none of them showed statistical significance in predicting the elapsed time t .

The optimization step is performed at each application of the AMG method, that is, every time we solve the system $\mathbf{A}\mathbf{u} = \mathbf{f}$. In view of this, it seems quite natural that only the matrix A is employed in the tuning procedure. However, the matrix A is a large, sparse matrix of variable size. Neural networks are not well-equipped to handle this kind of data. For this reason, we use a special kind of pooling, first introduced in [69], to prune and compress A into a small multi-channel image $\mathbf{V} \in \mathbb{R}^{m \times m \times f}$. This procedure is outlined, together with the meaning of the hyperparameters m and f , in Section 4.2.

Hence, our goal is to use a neural network $\mathcal{F}$ to predict the computational cost of solving a system $\mathbf{A}\mathbf{u} = \mathbf{f}$, where the input of $\mathcal{F}$ is the matrix A and the parameters that we want to tune (the scalar θ and the categorical variable for the kind of smoother encoded as a one-hot). Although including $\mathbf{f}$ (or a pooled form of $\mathbf{f}$) as ANN input might improve per-problem performance, we omit it because AMG constructs its multilevel hierarchy and interpolation operators from A alone. By limiting the network’s input to information derived from A , we aim to learn features that generalize across problems rather than overfit to particular right-hand sides; we therefore hypothesize improved generalization. The optimal choice of parameters is then found by solving two optimization problems. The first one is the offline training of the ANN, which enables us to learn an approximate map of the computational cost depending on the problem we are solving (A) and the choice of the AMG’s parameters. This step is expensive but it is done only once. The second one is an online optimization step that allows us to find the optimal choice of parameters, namely:

$$\min_{\text{AMG parameters}} \mathcal{F}(A, \cdot). \quad (13)$$

This optimization step takes place in the online phase and must be solved each time we apply the algorithm.

While this choice might seem strange at first, we would like to highlight a few advantages. This approach is more data-efficient. Specifically, if the ANN were trained to directly predict the optimal combination of AMG parameters, we would need to solve a more expensive offline optimization problem, which in turn would require solving many more linear systems $\mathbf{A}\mathbf{u} = \mathbf{f}$. This procedure would be prohibitively expensive. By contrast, if the ANN is trained to predict the computational cost, then every computational cost measurement taken by solving $\mathbf{A}\mathbf{u} = \mathbf{f}$ – for any choice of AMG’s parameters – can be used as a training sample. In this way, we can build a large dataset by solving many inexpensive, small-scale problems, and then supplement it with only a few samples from larger, more costly problems, relying on the ANN’s generalization capability. In other words, by predicting the computational cost instead of the optimal AMG parameters, we build a surrogate model $(A, \text{AMG parameters}) \mapsto c$ which enables a more efficient solution of Eq. 13 than directly measuring c by solving $\mathbf{A}\mathbf{u} = \mathbf{f}$. This feature is incredibly useful when the solution of even one linear system may be costly. Moreover, we have empirically found this approach to be more stable with respect to directly learning the optimal value of parameters. Finally, the search space is practically one-dimensional (since the smoother choice is discrete), thus the cost of performing the optimization is small.

4.1 Handling Measurement Uncertainty

In this section, we describe the procedure used to collect the data for training the neural network when the execution time is taken as performance metric. Since measurements of the elapsed time t are subject to noise, we repeat each measurement multiple times.

To reduce data collection costs while mitigating measurement errors, we adopt the following strategy: the measurement is repeated r times, where r ranges from 2 to 100 and is chosen inversely proportional to the mean elapsed time of the first two measurements. The rationale is that measurement variability arises primarily from operating-system tasks running concurrently with the AMG solver, perturbing CPU load. For larger values of t , these fluctuations tend to average out, resulting in lower variance. Empirical evidence supports this assumption, as we observed that the sample variance of repeated measurements decreases inversely proportionally to the elapsed time (for fixed r). The reported elapsed time t is then taken as the mean of the r repetitions.

To further improve data quality, we apply a Savitzky-Golay filter [78]. We employ a window size of 21 and a polynomial of degree 7 for uniformly sampled values of θ . These parameters were determined through manual tuning, by testing multiple combinations and assessing their performance on representative subsets of the dataset. Selection was guided by visual inspection, balancing smoothness and fidelity to the raw data. Importantly, we verified that the filter preserves the positivity of the data and performed manual checks whenever the difference between the minima of the raw and smoothed signals exceeded a prescribed threshold. In such cases, we adjusted the polynomial degree or window size as appropriate. We note that smoothing can alter the position of minima in sharp valleys, as it tends to attenuate high-frequency features. Improper tuning of the filter may significantly degrade predictive accuracy, whereas carefully calibrated smoothing provides accuracy improvements of several percentage points. Nevertheless, in the absence of proper tuning, we observed that omitting smoothing remains preferable to applying it blindly.

All experiments reported here were carried out in a serial manner in a controlled environment, ensuring that no other processes contributed significant CPU load. To enhance training robustness, we normalize execution times to the interval $[0, 1]$.

Finally, we remark that parallel execution introduces another external factor – the number of CPU cores – which can be treated as an additional AMG parameter. However, this extension is left for future work.

4.2 The Pooling Operator

One of the key steps of our algorithm is passing the information contained in A to the neural network. Even if the structure of a sparse matrix is more akin to a graph than a dense matrix, we prefer to use a neural network with structured input (CNN) due to the size of A . Namely, to make this process scalable to cases where A has millions of entries, we prefer not to directly apply GNNs to A , but instead, we would rather employ a process that can prune and extract information from A much more quickly. In particular, we use a variation of the pooling technique used in CNNs, first introduced in [69].

We denote by $\mathbf{V} = \text{pooling}(A, m) \in \mathbb{R}^{m \times m \times f}$ the pooled representation of A , where $m \in \mathbb{N}$ is a hyperparameter that controls the tensor size and f is the number of features extracted, in our case $f = 4$. Letting $q = \lceil n_1/m \rceil$, the pooled features are defined as

$$\begin{aligned} v_{ij1} &= \max_{i', j'=1, \dots, q} \max \{0, \tilde{a}_{iq+i', jq+j'}\}, & v_{ij2} &= \max_{i', j'=1, \dots, q} \max \{0, -\tilde{a}_{iq+i', jq+j'}\}, \\ v_{ij3} &= \sum_{i'=1}^q \sum_{j'=1}^q \tilde{a}_{iq+i', jq+j'}, & v_{ij4} &= \sum_{i'=1}^q \sum_{j'=1}^q \chi_{(0, +\infty)}(\tilde{a}_{iq+i', jq+j'}), \end{aligned}$$

where χ is the indicator function and $\tilde{a}_{ij} = a_{ij} \chi_{i \leq n_1, j \leq n_1}$.

A reference implementation is reported in Algorithm 3. While the pseudocode assumes that A is stored in coordinate (COO) format, the procedure extends to other sparse storage formats.

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

```
1: Access  $A$  in COO format and extract:  $n_1$ ,  $\text{val}$ ,  $\text{row}$ ,  $\text{col}$ 
2: Initialize  $\mathbf{V}$  as an  $m \times m \times 4$  dense tensor with zero entries
3:  $q \leftarrow n_1/m$ ,  $p \leftarrow n_1 \bmod m$ ,  $t \leftarrow (q+1)p$ 
4: for  $k = 0$  to  $\text{val.size}() - 1$  do
5:    $i \leftarrow \text{row}[k]/(q+1)$  if  $\text{row}[k] < t$ , else  $(\text{row}[k] - t)/q + p$ 
6:    $j \leftarrow \text{col}[k]/(q+1)$  if  $\text{col}[k] < t$ , else  $(\text{col}[k] - t)/q + p$ 
7:    $v_{ij1} \leftarrow \max(\text{val}[k], 0)$ ,  $v_{ij1}$ 
8:    $v_{ij2} \leftarrow \max(-\text{val}[k], 0)$ ,  $v_{ij2}$ 
9:    $v_{ij3} \leftarrow v_{ij3} + \text{val}[k]$ 
10:   $v_{ij4} \leftarrow v_{ij4} + 1$ 
11: end for
```

Unlike conventional CNN pooling, which typically extracts a single maximum, here we compute four complementary features within each neighborhood: maximum of positive entries, maximum of negative entries, sum of all entries, and the number of nonzeros. This choice is motivated by the role of positive, negative, and aggregate values in defining the interpolation weights (Eq. 4), while the nonzero count provides a measure of local sparsity. Although additional features could be incorporated, this selection offers a favorable balance between expressiveness and efficiency. Notably, the features preserve the sparsity structure: pooling a block of zeros yields zero.

The algorithm scales linearly with the number of nonzero entries, i.e., $O(nnz)$, which for finite element discretizations translates to $O(p \cdot n)$. Empirical tests confirm that its runtime is negligible compared to that of solving the linear system, a prerequisite for its practical relevance. Furthermore, Algorithm 3 can be parallelized with minimal effort.

4.2.1 Normalization

To ensure stable and fast convergence, the tensor $\mathbf{V}$ is normalized following a logarithmic normalization scheme [69]. It has been shown that this normalization outperforms standard linear scaling in this context. Namley, for each feature f , the transformation reads

$$\hat{v}_{ijf} = \frac{\log(|v_{ijf}| + 1)}{\max_{i,j} |\log(|v_{ijf}| + 1)|} \frac{v_{ijf}}{|v_{ijf}|}. \quad (14)$$

A key property of this normalization is that zeros map to zero, thereby preserving the sparsity pattern of $\mathbf{V}$.

4.3 The AMG-ANN Algorithm

The core of the algorithm is the neural network $\mathcal{F}$ that predicts the computational cost of the AMG method. As mentioned before, the computational cost for us can either be the approximate convergence factor ρ or the normalized and smoothed wall clock time $\bar{t}$.

The architecture of $\mathcal{F}$ is made up of two main components in series. The first is a CNN that analyzes and encodes a flat latent representation for the pooling tensor $\mathbf{V}$. The hyperparameters that we consider for this component are the size m of $\mathbf{V}$, the number of convolution blocks (ending with a max-pooling layer), the number of convolutions in each block, and the number and size of filters of each convolution. The activation function we use is the ReLU.

The latent representation of $\mathbf{V}$ is concatenated with the parameters of the AMG method we want to tune (the scalar θ and the kind of smoother encoded as a one-hot of size four), the

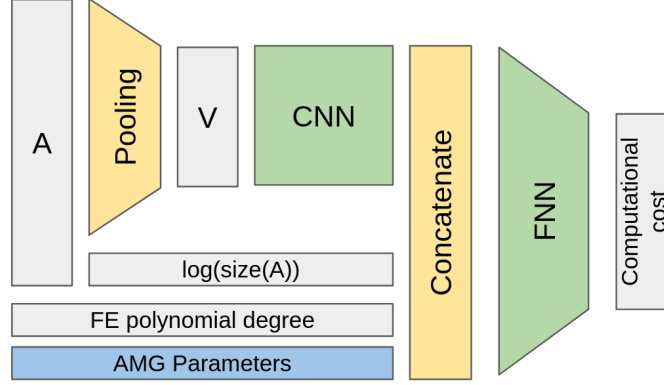


Figure 1: Architecture of the proposed ANN used to predict the optimal value of the AMG parameters.

logarithm of the size of A ($\log(n)$) – since during the pooling this information is lost – and, the polynomial degree p of the basis function used in the FE discretization. This information is not strictly needed; indeed, our experiments show that even omitting the parameter p , we can obtain a neural network with a similar loss. However, when adding the polynomial degree to the inputs, the training is usually less expensive and requires a shorter phase of hyperparameters’ tuning. Hence, our algorithm is not limited to problems stemming from FE discretizations.

The second component of $\mathcal{F}$ is a dense feed-forward neural network that takes as input the aforementioned concatenation layer and outputs a prediction of the computational cost. On the last layer, we add a clipping activation function to ensure that the output lies in $[0, 1]$. The hyperparameters of this component are the depth and the width of the network. To sum up, we have

$$\mathcal{F}(\hat{\mathbf{V}}, \log n, p, \theta, b; \alpha) = \tilde{c} \quad (15)$$

where

$$b \in \mathcal{O}_4 = \left\{ e_i \in \{0, 1\}^4 : \sum_{j=1}^4 (e_i)_j = 1 \right\}$$

is the one-hot representation of the kind of smoother. Namely, it is a vector of four components where only one is not zero. Thus, if the non-zero component is in the i -th position, we choose as smoother the i -th in the following list: Jacobi with successive over-relaxation (SOR), ℓ^1 -Jacobi [79], ℓ^1 -Jacobi with SOR and Jacobi with FCF relaxation [80]. Moreover, we denote by α the parameters of the neural network, while $\tilde{c}$ denotes the approximation of the computational cost c . The architecture of the ANN is shown in Figure 1.

Like most deep learning algorithms, our algorithm works in two phases.

Offline phase We collect, smooth, and normalize the data following the procedure outlined in Section 4.1. Then we train the neural network with the collected samples. Namely, the input-target training pairs for the supervised training are the couples $((\hat{\mathbf{V}}^{(i)}, \log n^{(i)}, p^{(i)}, \theta^{(i)}, b^{(i)}), c^{(i)})$, where the superscript (i) indicates that it is the i -th sample of the dataset. In other words, the ANN receives as input the matrix of the system, the threshold parameters, and the choice of the smoothers, and it has as its target the computational cost. As usual, The optimization step minimizes the MSE error between the predicted cost and the target cost c .

The hyperparameters of the neural network are chosen via a Bayesian optimization algorithm [81]. The procedure uses a standard 60-20-20 split into train, validation, and test datasets. The

data are partitioned at the problem level, meaning that each matrix A appears in exactly one of the three datasets. Consequently, when the algorithm is evaluated on the validation or test set, it is required to make predictions on entirely unseen problems. Training has been performed using the AdamW [82] optimizer. The initial learning rate and the minibatch size are also tuned hyperparameters. The learning rate is managed via a suitable learning rate schedule to improve convergence speed and stability. Namely, we employ a reduced on-plateau learning rate schedule that halves the learning rate with a manually tuned patience. More details on how data is practically generated are given in Sections 5.1, 5.2 since it depends on the physics and the FE discretization of the problem.

Online phase Given as input the matrix A of the linear system to be solved and, optionally, the polynomial degree p of the discretization.

- 1 Compress A by applying the pooling to obtain $\mathbf{V}$ (Section 4.2).
- 2 Normalize V to obtain $\hat{V}$ (Section 4.2.1)
- 3 Obtain (θ^*, b^*) solving the continuous optimization problem

$$(\theta^*, b^*) = \underset{(\theta, b) \in [0, 1] \times \mathcal{O}_4}{\operatorname{argmin}} \mathcal{F}(\hat{\mathbf{V}}, \log n, p, \theta, b) \quad (16)$$

by doing a complete search of the space using the discretization

$$(\theta^*, b^*) = \underset{(\theta, b) \in \text{linespace}(0, 1, 101) \times \mathcal{O}_4}{\operatorname{argmin}} \mathcal{F}(\hat{\mathbf{V}}, \log n, p, \theta, b)$$

- 4 Use (θ^*, b^*) as parameters of the AMG method.

4.4 Evaluating the Model

While obtaining a small MSE test loss between the predicted and target computational cost c during the offline phase is a good indicator that the neural network is learning, it does not measure the actual reduction of cost that our algorithm has on the AMG method. We remark that the choice of parameters (θ, b) employed in the AMG to solve the system is subordinate to the map $A \mapsto (\theta^*, b^*)$ defined by the ‘‘Online phase’’ algorithm of Section 4.3 and, in particular, to Eq. (16). Hence, we introduce the following quantities of interest. Let A be fixed, and let:

- t_{ANN} be the computational time of the AMG-ANN algorithm, that is by using $\theta = \theta^*$ and i -th smoother among SOR-Jacobi, ℓ^1 -Jacobi, SOR- ℓ^1 -Jacobi and FCF-Jacobi, where i is the position of the only non-zero bit of b^* ;
- t_0 be the computational time of the AMG method with the default parameter ($\theta = 0.25$ in 2D, $\theta = 0.5$ in 3D, and SOR-Jacobi smoother);
- t_{MIN} be the computational time of the AMG method with
$$(\theta^*, b^*) = \underset{(\theta, b) \in \text{dataset for } A}{\operatorname{argmin}} t(\theta, b; A);$$
- $P = 1 - \frac{t_{\text{ANN}}}{t_0}$ be the performance index of the AMG-ANN algorithm;
- $P_{\text{MAX}} = 1 - \frac{t_{\text{MIN}}}{t_0}$ be the best performance of the AMG-ANN algorithm.

Moreover, we can compound the quantities over different A and define P_B 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 $P_r = P/P_{MAX}$ gives the accuracy of the neural network, namely a measure of how close the ANN is to performing to the theoretical maximum.

Naturally, all the metrics defined above can be computed also if we employ as a measure of the computational cost the approximate convergence factor ρ . Indeed, we can compute (θ^*, b^*) as explained in the previous section and then compute all the performances indexes (P , P_{MAX} , etc.) by taking measurements of the wall clock time needed to solve the linear system for each choice of (θ, b) present in the dataset. Finally, let us remark that it can happen that, when the matrix A is ill-conditioned, the quantity t_0 is not well defined since the default values of the parameters of the AMG do not allow the solver to converge. In these cases we set $t_0 = \infty$ and thus $P = 1$. In a few of the following test cases, this is not a rare event. Thus, for the sake of completeness, we also introduce the following additional performance indexes.

- P_w as the percentage of cases where $t_0 = \infty$;
- $t_{0,i}$ as computational time of the AMG method with the default value of θ and the i -th smoother;
- $P^i = 1 - \frac{t_{ANN}}{t_{0,i}}$ the performance with respect to the i -th smoother.

5 Numerical Results

This section presents the numerical experiments designed to evaluate the performance of the proposed AMG-ANN algorithm across a variety of discretizations, problem settings, and dimensions. We first describe the computational framework and solver configuration adopted in our study. This provides the necessary context for interpreting the numerical results presented later. The experiments are then organized into two main categories: the diffusion problem and the linear elasticity problem. Within each category, we report several test cases of increasing complexity, spanning 2D and 3D domains, multiple mesh families, and different discretization schemes (VEM and PolyDG).

A key advantage of our algorithm is its ability to integrate seamlessly with existing code bases. In particular, we assemble the linear systems using the **VEM++** [83] and **Vulpes** [84] libraries, while employing BoomerAMG [68] from the HYPRE library [85] as the algebraic multigrid (AMG) solver. BoomerAMG is used strictly as a black-box solver; no modifications are introduced except for adjusting the strong threshold parameter θ and the smoother. All other parameters remain at their default values. The stopping criterion is defined by enforcing a relative residual tolerance of 10^{-8} when AMG is used as a preconditioner for the conjugate gradient (CG) Krylov solver of PETSc [86].

Finally, we describe the method used to choose or sample the threshold parameter θ , which is common across all test cases. For small problems ($n < 20,000$), we use a uniform discretization of the interval $[0.05, 0.95]$ with a step size of 0.025. For moderate problems (up to $n = 100,000$), we employ a coarser discretization with a step size of 0.05. For larger problems, we uniformly randomly sample the interval at 10 or fewer points, depending on the problem size.

5.1 Numerical Results: Diffusion Problem

5.1.1 Test Case 1: VEM Discretization of the Diffusion Problem in 2D

This first test case involves a VEM discretization of problem (5) with discontinuous diffusion coefficients, representing a moderately challenging scenario where the default AMG solver is



Figure 2: Patterns used to partition the domain Ω into Ω_{white} and Ω_{gray} .

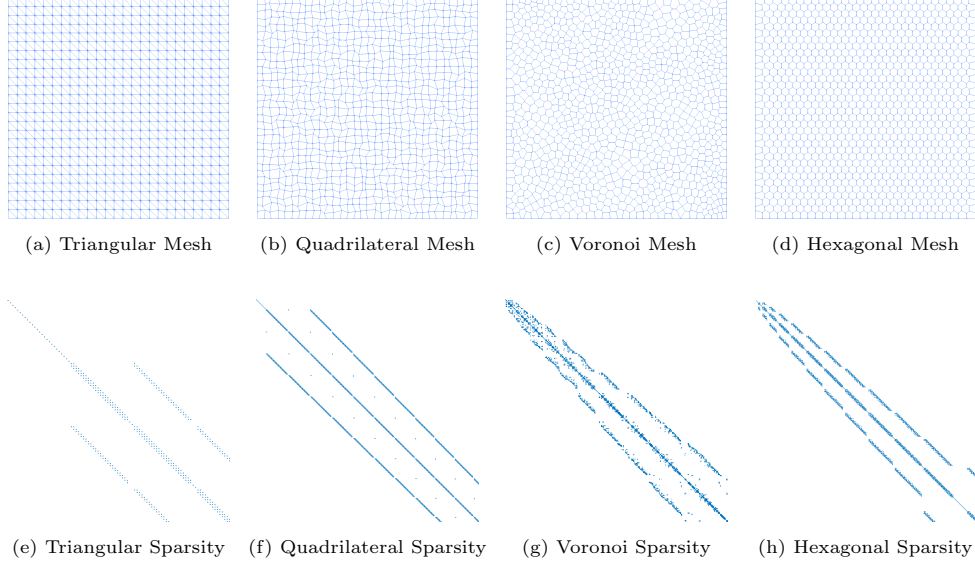


Figure 3: The four mesh types considered in 2D work (top row) and their corresponding system matrix sparsity patterns (bottom row) for the diffusion problem discretized with VEM of polynomial degree $p = 1$.

generally effective. To define the diffusion coefficient κ , we partition the domain Ω into two subdomains Ω_{white} and Ω_{gray} according to 4 different patterns (see Figure 2) and we assign a constant value to κ in each subdomain in the following way:

$$\kappa(x, y) = \begin{cases} 1 & \text{if } (x, y) \in \Omega_{white} \\ 10^\epsilon & \text{if } (x, y) \in \Omega_{gray} \end{cases}. \quad (17)$$

This diffusion coefficient is discontinuous across subdomains, and the magnitude of the jump is controlled by the parameter $\epsilon \in \mathbb{R}$, which will be varied in the dataset generation. The domain Ω is discretized as shown in Figure 3. Namely, we employ 24 distinct meshes, comprising six different refinement levels for each of the four mesh types. The mesh was refined such that the number of elements increases geometrically with each refinement level.

In this case, we observe that ρ and wall clock time are linearly correlated, cf. Figure 4. Thus, we employ ρ as a measure of the computational cost.

In Figure 5, we show the gain in performance and the scaling obtained by using our AMG-ANN algorithm. The histogram shows that the gains are evenly distributed, with no peaks, proving that the algorithm is consistent across different scenarios. The scaling of the computational cost further supports the fact that our algorithm works well as the size of the problem

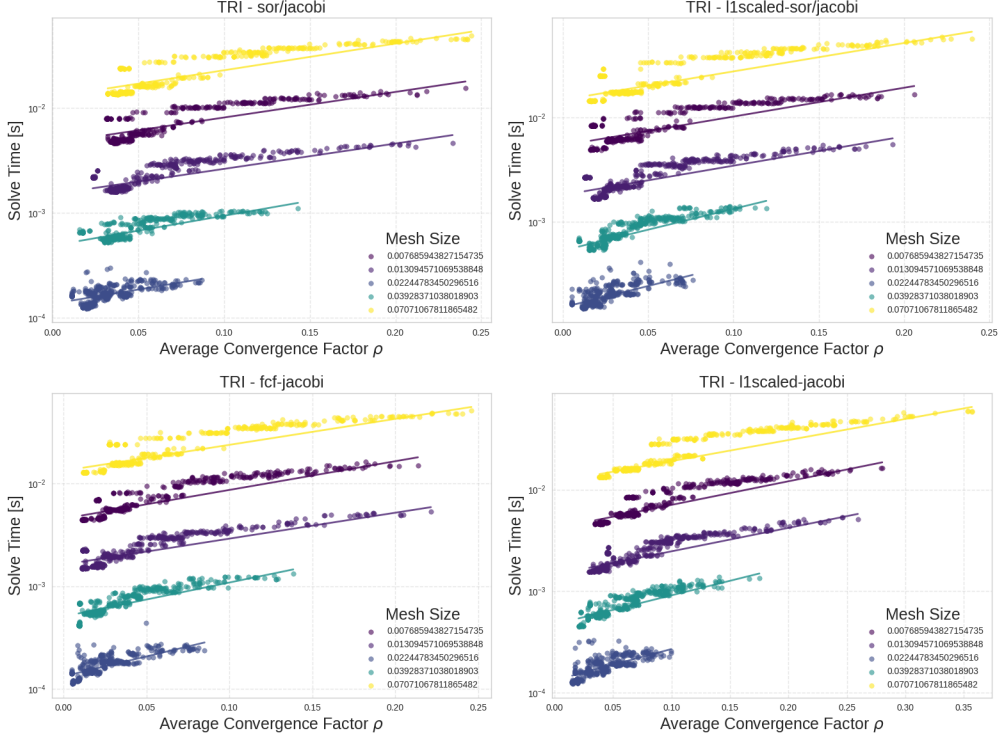


Figure 4: Correlation between the average convergence factor ρ and wall-clock solve time for triangular grid for different smoothers. A clear positive correlation is observed, validating ρ as a reliable proxy for computational cost. Similar results hold for the other mesh families.

increases, which is of utmost importance for large-scale solvers. The results in Table 1 show that the default solver (SOR-Jacobi) always converges ($P_w = 0\%$). Nevertheless, our AMG-ANN algorithm yields substantial improvements. The median performance gain $P_M = 23.7\%$ indicates a significant reduction in computational time. The algorithm is effective in the vast majority of instances, as shown by $P_B = 87.5\%$. The performance ratio $P_r = 70.1\%$ reveals that the ANN successfully captures over two-thirds of the maximum achievable performance improvement identified in our dataset. This confirms the efficacy of the network’s predictions. Finally, these gains in wall-clock time were achieved even though the network was trained using the approximate convergence factor ρ , validating our choice of ρ as an effective and cheaper proxy for computational cost.

5.1.2 Test Case 2: PolyDG Discretization of the Diffusion Problem in 2D

We consider the diffusion problem (5), discretized by means of PolyDG with polynomial degree $p = 1, 2, 3, 4$. The domain $\Omega = (0, 1)^2$ is a unit square discretized in four different ways, as shown in the top row of Figure 3b. Different degrees of refinement are employed when building the dataset, namely, each refinement level contains a number of elements that grows geometrically relative to the previous level up to 200,000. The diffusion coefficient κ is a strongly heterogeneous piecewise constant that is conforming to the mesh. Namely, κ takes the form of $\kappa = 10^{\epsilon_i}$ on the i -th cell of the mesh. The values of ϵ_i are chosen by extracting values for a uniform distribution in $[0, \epsilon_{MAX}]$, where $\epsilon_{MAX} \in \{1, 2, 4\}$. The penalty γ is chosen uniformly at random in $[5, 20]$,

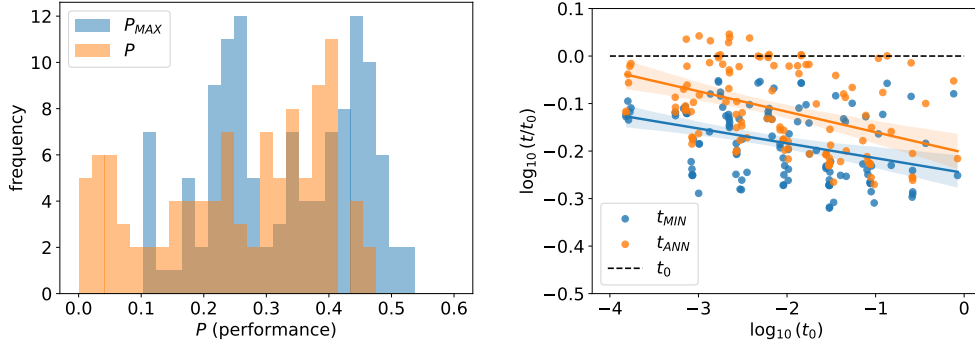


Figure 5: Test Case 1: (Left) Performance gain P of the AMG-ANN algorithm and maximum theoretical performance P_{MAX} . (Right) Scaling of the AMG-ANN computational cost t_{ANN} and the theoretical minimum t_{MIN} with respect to the default one t_0 .

	P_B	P_w	P_m	P_M	P_{MAX}	P_r	P_M^2	P_M^3	P_M^4
Test Case 1	87.5	0	21.1	23.7	33.8	70.1	17.8	22.6	19.1
Test Case 2	100	100	100	100	100	100	21.9	99.6	99.7
Test Case 3	100	100	100	100	100	100	26.9	97.1	96.5
Test Case 4	100	100	100	100	100	100	24.2	100	100
Test Case 5	100	100	100	100	100	100	24.8	97.2	95.5

Table 1: Evaluation of the performance of the AMG-ANN algorithm for each test case.

making sure that the chosen value keeps the matrix A is positive definite.

Details about the performance are shown in Figure 6 and Table 1. This test case significantly increases the difficulty by employing a PolyDG discretization with highly heterogeneous coefficients, which typically produces ill-conditioned matrices. Indeed, the presence of “duplicate” degrees of freedom makes the application of the AMG method challenging. The results underscore a critical strength of our approach: its ability to render intractable problems solvable. As shown in Table 1, the default AMG configuration fails to converge in every single instance ($P_w = 100\%$). Consequently, any convergent parameter set represents an infinite improvement, leading to perfect scores in P_B , P_m , P_M , P_{MAX} , and P_r . The primary contribution of the ANN here is identifying a suitable smoother, as the default Jacobi method is inadequate. The metrics P_M^i allow us to dissect the algorithm’s performance further. The near-perfect scores for $P_M^3 = 99.6\%$ and $P_M^4 = 99.7\%$ confirm that smoothers 3 and 4 are poor choices. The much lower value of $P_M^2 = 21.9\%$ reveals two things: first, that the second smoother is the most robust choice for this problem class, and second, that even after making this correct choice, the ANN’s fine-tuning of the threshold θ provides an additional median time reduction of 21.9%. This highlights the dual benefit of our approach: robust smoother selection and effective parameter optimization.

5.1.3 Test Case 3: PolyDG Discretization of the Diffusion Problem in 3D

We extend the previous test case by considering the 3D version. Namely, we consider $\Omega = (0,1)^3$, discretized with four different types of grids, as shown in Figure 7. We consider various refinements of the grids until we reach 100 thousand degrees of freedom. The diffusion coefficient κ takes the same form as the previous test case.

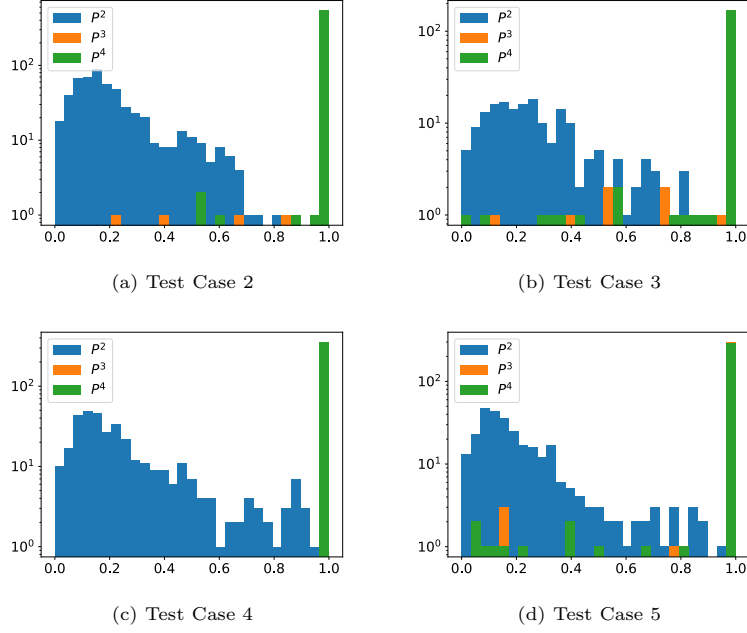


Figure 6: Test Cases 2-5: Performance gain P^i of the AMG-ANN algorithm with respect to the choice of the smoother: (2) ℓ^1 -Jacobi, (3) ℓ^1 -SOR Jacobi and (4) FCF-Jacobi. The ℓ^1 -SOR Jacobi is usually the best, while the other most often do not reach convergence. Results for (1) SOR-Jacobi are omitted because $P^1 = 1$ for all test samples. Indeed, when SOR-Jacobi is employed as a smoother, the AMG method fails to converge (see Table 1).

Extending the previous scenario to three dimensions further increases the conditioning challenges. We show the performance of our algorithm in Figure 6 and Table 1.

As in the 2D case, the default AMG solver consistently fails to converge ($P_w = 100\%$), making the AMG-ANN’s ability to find a convergent configuration essential. The results in Table 1 show that the landscape of optimal smoothers is more complex in 3D. While the performance gains relative to smoothers 3 and 4 remain very high ($P_M^3 = 97.1\%$ and $P_M^4 = 96.5\%$), they are slightly lower than in the 2D case. This indicates that while smoother 2 is still generally the best, the other smoothers are not as uniformly suboptimal, and the best choice may be more dependent on the specific matrix properties. The ANN successfully navigates this more complex decision space. However, the benefit of fine-tuning the threshold parameter θ for the best smoother family is even more pronounced in 3D, yielding a median performance gain of $P_M^2 = 26.9\%$.

5.2 Numerical Results: Linear Elasticity Problem

We now shift to the discretization of the linear elasticity problem, which introduces further complexities due to its block-structured nature.

5.2.1 Test Case 4: PolyDG Discretization of the Linear Elasticity Problem in 2D

We consider the PolyDG discretization of the linear elasticity problem 5. The domain $\Omega = (0, 1)^2$ is a unit square discretized with the grids outlined in the first row of Figure 3b. For each kind of grid, we employ different refinements until the total number of degrees of freedom exceeds one

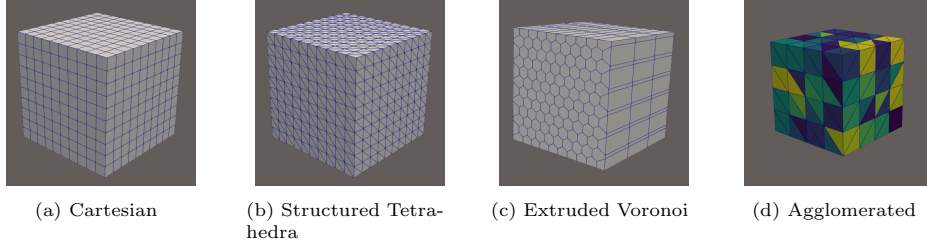


Figure 7: The four mesh types considered in 3D test cases.

hundred thousand.

We can rewrite the Lamé parameters in terms of the Young’s modulus $E > 0$ and the diffusion ratio $\nu \in (0, \frac{1}{2})$ (see Eq. 7). The problem has been discretized with order $p = 1, 2, 3, 4$. We fix the diffusion ratio $\nu = 0.29$ and choose a strongly heterogeneous Young modulus E . Namely, for each cell, we set $E = 10^\epsilon$, where ϵ is chosen at random in $[0, \epsilon_{MAX}]$, where $\epsilon_{MAX} \in \{1, 2, 4\}$. The penalty coefficient γ is chosen so that the matrix A is symmetric and positive definite. Various values in the interval $[4, 20]$ are picked to enhance the dataset.

Similar to the diffusion PolyDG cases, the default AMG parameters frequently fail to yield a convergent solver, making the AMG-ANN indispensable. The numerical results reveal a clear trend: the ℓ^1 -scaled SOR Jacobi smoother consistently emerges as the most effective choice across different mesh types and polynomial orders. Our algorithm reliably identifies this optimal smoother in all instances. The primary benefit is therefore transforming a non-convergent method into a robust and efficient one. The median performance gain attributed solely to the optimization of the threshold θ after selecting the right smoother family is $P_M^2 = 24.2\%$, demonstrating that even when a single smoother is dominant, careful parameter tuning remains crucial for achieving optimal performance.

5.2.2 Test Case 5: PolyDG Discretization of the Linear Elasticity Problem in 3D

This final test case represents the most complex scenario, combining the challenges of 3D geometry, PolyDG discretization, and the block structure of linear elasticity. The computational domain is $\Omega = (0, 1)^3$, discretized with the four mesh families shown in Figure 7. The meshes are progressively refined until the total number of degrees of freedom exceeds 10^5 . The governing equations are the 3D linear elasticity equations in displacement form, see Eq. (6). We fix $\nu = 0.29$, while the Young’s modulus E is chosen to be strongly heterogeneous across elements: for each cell, we set

$$E = 10^\epsilon, \quad \epsilon \sim \mathcal{U}(0, \epsilon_{MAX}), \quad \epsilon_{MAX} \in \{1, 2, 4\}. \quad (18)$$

The stabilization parameter γ is randomly drawn from the interval $[5, 20]$, ensuring that the global stiffness matrix A remains symmetric positive definite. Discretization is performed with PolyDG of order $p = 1, 2, 3, 4$, yielding block-structured stiffness matrices with significant heterogeneity and conditioning challenges.

The numerical results for this case are summarized in Figure 6 and Table 1. As in the 2D elasticity case, the ℓ^1 -scaled SOR Jacobi smoother is consistently selected by the AMG-ANN algorithm as the most effective option, confirming its robustness across discretization orders and mesh types. The baseline AMG setup frequently fails to converge within the prescribed tolerance, especially for high-order discretizations ($p \geq 3$) and agglomerated meshes. Our algorithm always selects a convergent configuration. Performance improvements remain substantial. Optimizing

the threshold parameter θ alone yields a reduction of 24.8% in computational cost. In contrast to the 2D case, the interplay between smoother choice and parameter tuning is more delicate in 3D: certain smoothers are optimal only for specific mesh families, which explains the observed degradation in P_M^3 and P_M^4 relative to P_M^2 . This reflects the higher complexity of the 3D operator spectrum. Overall, these results show that our AMG-ANN framework scales robustly to heterogeneous 3D linear elasticity.

6 Conclusions

In this work, we addressed the challenge of employing Deep Learning to accelerate algebraic multigrid iterative solvers for linear systems of equations stemming from polytopal discretizations of Partial Differential Equations, where the increased flexibility of polygonal and polyhedral meshes comes at the cost of more complex algebraic structures. Algebraic multigrid methods are among the state-of-the-art iterative solvers thanks to their robustness and black-box nature. However, AMG performance is highly sensitive to the smoother and parameter choices, particularly the strong threshold parameter that drives the coarsening strategy. To improve the performance of AMG solvers, we introduced ANN-AMG: a framework that automatically tunes the smoother as well as the strong threshold parameter in AMG solvers using a novel neural network-driven approach. By interpreting the linear system matrix as an image and designing a tailored convolutional architecture and a novel pooling strategy, our ANN-AMG achieves robust and efficient parameter selection on the fly, eliminating the need for manual fine-tuning. The proposed approach is entirely non-intrusive, and it is compatible with existing implementations of the PDE solver and parallel AMG implementations. Numerical experiments on both PolyDG and VEM discretizations of diffusion and elasticity problems in two- and three-dimensions demonstrated that we can reduce up to 27% the computational cost compared to AMG approaches with “manual” tuning. Our results demonstrate the advantages of enhancing algebraic iterative solvers via deep learning to improve the efficient numerical solution of differential problems posed on complex and heterogeneous domains via polytopal discretizations. Further research aims to integrate data-driven adaptivity in multilevel solvers, ranging from coarsening strategies to the design of intergrid operators, as well as to extend ANN-AMG to non-symmetric or indefinite systems and explore transferability across broader classes of PDE models and discretizations.

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.

Funding

This work received funding from the European Union (ERC SyG, NEMESIS, project number 101115663). Views and opinions expressed are, however, those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them. PFA, MC, and MV are members of INdAM-GNCS. The present research is part of the activities of “Dipartimento di Eccellenza 2023-2027”, funded by MUR, Italy.

References

- [1] U. Trottenberg, C. W. Oosterlee, and A. Schuller, *Multigrid*. Elsevier, 2000.
- [2] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.
- [3] P. Wesseling, *An Introduction to Multigrid Methods*. An Introduction to Multigrid Methods, R.T. Edwards, 2004.
- [4] J. H. Bramble, *Multigrid Methods*. Chapman and Hall/CRC, 2019.
- [5] A. Brandt, “Algebraic Multigrid (AMG) for sparse matrix equations,” *Sparsity and its Applications*, pp. 257–284, 1984.
- [6] A. Brandt, “Algebraic Multigrid theory: The symmetric case,” *Applied Mathematics and Computation*, vol. 19, no. 1-4, pp. 23–56, 1986.
- [7] J. W. Ruge and K. Stüben, “Algebraic Multigrid,” in *Multigrid Methods*, pp. 73–130, SIAM, 1987.
- [8] L. Beirão da Veiga, F. Brezzi, A. Cangiani, G. Manzini, L. D. Marini, and A. Russo, “Basic principles of virtual element methods,” *Mathematical Models and Methods in Applied Sciences*, vol. 23, no. 01, pp. 199–214, 2013.
- [9] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo, “The hitchhiker’s guide to the virtual element method,” *Mathematical models and methods in applied sciences*, vol. 24, no. 08, pp. 1541–1573, 2014.
- [10] P. F. Antonietti, G. Manzini, S. Scacchi, and M. Verani, “A review on arbitrarily regular conforming virtual element methods for second- and higher-order elliptic partial differential equations,” *Mathematical Models and Methods in Applied Sciences*, vol. 31, no. 14, pp. 2825 – 2853, 2021.
- [11] P. F. Antonietti, F. Brezzi, and L. D. Marini, “Bubble stabilization of discontinuous Galerkin methods,” *Computer Methods in Applied Mechanics and Engineering*, vol. 198, no. 21-26, pp. 1651 – 1659, 2009.
- [12] P. F. Antonietti, S. Giani, and P. Houston, “hp-version composite discontinuous Galerkin methods for elliptic problems on complicated domains,” *SIAM Journal on Scientific Computing*, vol. 35, no. 3, pp. A1417–A1439, 2013.
- [13] F. Bassi, L. Botti, A. Colombo, D. Di Pietro, and P. Tesini, “On the flexibility of agglomeration-based physical space discontinuous Galerkin discretizations,” *Journal of Computational Physics*, vol. 231, no. 1, pp. 45 – 65, 2012.
- [14] A. Cangiani, E. H. Georgoulis, and P. Houston, “hp-version discontinuous Galerkin methods on polygonal and polyhedral meshes,” *Mathematical Models and Methods in Applied Sciences*, vol. 24, no. 10, pp. 2009–2041, 2014.
- [15] A. Cangiani, Z. Dong, E. H. Georgoulis, and P. Houston, *hp-Version Discontinuous Galerkin Methods on Polygonal and Polyhedral Meshes*. Springer Publishing Company, Incorporated, 1st ed., 2017.
- [16] P. F. Antonietti, A. Cangiani, J. Collis, Z. Dong, E. H. Georgoulis, S. Giani, and P. Houston, “Review of discontinuous Galerkin finite element methods for partial differential equations on complicated domains,” in *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, pp. 281–310, Springer, 2016.
- [17] D. A. Di Pietro, A. Ern, and S. Lemaire, “An arbitrary-order and compact-stencil discretization of diffusion on general meshes based on local reconstruction operators,” *Computational Methods in Applied Mathematics*, vol. 14, no. 4, pp. 461 – 472, 2014.
- [18] D. A. Di Pietro and A. Ern, “Hybrid high-order methods for variable-diffusion problems on general meshes,” *Comptes Rendus Mathématique*, vol. 353, no. 1, pp. 31–34, 2015.
- [19] D. A. Di Pietro and R. Tittarelli, “An introduction to hybrid high-order methods,” in *Numerical Methods for PDEs: State of the Art Techniques*, pp. 75–128, Springer, 2018.
- [20] D. A. Di Pietro and J. Droniou, “The hybrid high-order method for polytopal meshes,” *Number 19 in Modeling, Simulation and Application*, vol. 84, 2020.
- [21] D. A. Di Pietro, A. Ern, and S. Lemaire, “A review of hybrid high-order methods: formulations, computational aspects, comparison with other methods,” *Building bridges: connections and challenges in modern approaches to numerical partial differential equations*, pp. 205–236, 2016.
- [22] J. Hyman, J. Morel, M. Shashkov, and S. Steinberg, “Mimetic finite difference methods for diffusion equations,” *Computational Geosciences*, vol. 6, no. 3, pp. 333–352, 2002.
- [23] K. Lipnikov, G. Manzini, and M. Shashkov, “Mimetic finite difference method,” *Journal of Computational Physics*, vol. 257, pp. 1163–1227, 2014.
- [24] L. B. da Veiga, K. Lipnikov, and G. Manzini, *The mimetic finite difference method for elliptic problems*, vol. 11. Springer, 2014.
- [25] L. Beirão da Veiga, K. Lipnikov, and G. Manzini, “The mimetic finite difference method for elliptic problems,” *Modeling, Simulation and Applications*, vol. 11, pp. 1 – 389, 2014.

- [26] B. Cockburn, “The hybridizable discontinuous Galerkin methods,” in *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 2749–2775, World Scientific, 2010.
- [27] N. C. Nguyen, J. Peraire, and B. Cockburn, “A hybridizable discontinuous Galerkin method for Stokes flow,” *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 9–12, pp. 582–597, 2010.
- [28] N. C. Nguyen and J. Peraire, “Hybridizable discontinuous Galerkin methods for partial differential equations in continuum mechanics,” *Journal of Computational Physics*, vol. 231, no. 18, pp. 5955–5988, 2012.
- [29] J. Wang and X. Ye, “A weak Galerkin finite element method for second-order elliptic problems,” *Journal of Computational and Applied Mathematics*, vol. 241, pp. 103–115, 2013.
- [30] J. Wang and X. Ye, “A weak Galerkin finite element method for the Stokes equations,” *Advances in Computational Mathematics*, vol. 42, no. 1, pp. 155–174, 2016.
- [31] P. F. Antonietti, N. Farenga, E. Manuzzi, G. Martinelli, and L. Saverio, “Agglomeration of polygonal grids using graph neural networks with applications to multigrid solvers,” *Computers & Mathematics with Applications*, vol. 154, pp. 45–57, 2024.
- [32] P. F. Antonietti, M. Caldana, I. Mazzieri, and A. R. Fraschini, “MAGNET: an open-source library for mesh agglomeration by Graph Neural Networks,” *Engineering with Computers*. in press.
- [33] M. Feder, A. Cangiani, and L. Heltai, “R3mg: R-tree based agglomeration of polytopal grids with applications to multilevel methods,” *Journal of Computational Physics*, vol. 526, 2025.
- [34] P. F. Antonietti, P. Houston, X. Hu, M. Sarti, and M. Verani, “Multigrid algorithms for hp-version interior penalty Discontinuous Galerkin methods on polygonal and polyhedral meshes,” *Calcolo*, vol. 54, pp. 1169–1198, 2017.
- [35] P. F. Antonietti and G. Pennesi, “V-cycle multigrid algorithms for discontinuous Galerkin methods on non-nested polytopic meshes,” *Journal of Scientific Computing*, vol. 78, no. 1, pp. 625–652, 2019.
- [36] P. Antonietti, P. Houston, G. Pennesi, and E. Süli, “An agglomeration-based massively parallel non-overlapping additive Schwarz preconditioner for high-order discontinuous Galerkin methods on polytopic grids,” *Mathematics of Computation*, vol. 89, no. 325, pp. 2047–2083, 2020.
- [37] Y. Pan and P.-O. Persson, “Agglomeration-based geometric multigrid solvers for compact discontinuous Galerkin discretizations on unstructured meshes,” *Journal of Computational Physics*, vol. 449, p. 110775, 2022.
- [38] 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*, vol. 52, no. 1, pp. 337–364, 2018.
- [39] P. F. Antonietti, S. Berrone, M. Busetto, and M. Verani, “Agglomeration-based geometric multigrid schemes for the Virtual Element Method,” *SIAM Journal on Numerical Analysis*, vol. 61, no. 1, pp. 223–249, 2023.
- [40] D. Prada and M. Pennacchio, “Algebraic multigrid methods for virtual element discretizations: A numerical study,” *arXiv preprint arXiv:1812.02161*, 2018.
- [41] D. A. Di Pietro, Z. Dong, G. Kanschat, P. Matalon, and A. Rupp, “Homogeneous multigrid for hybrid discretizations: application to HHO methods,” *Numerical Methods for Partial Differential Equations*, vol. 41, no. 5, p. e70023, 2025.
- [42] D. A. Di Pietro, P. Matalon, P. Mycek, and U. Rüde, “High-order multigrid strategies for hybrid high-order discretizations of elliptic equations,” *Numerical Linear Algebra with Applications*, vol. 30, no. 1, p. e2456, 2023.
- [43] L. Botti and D. A. Di Pietro, “p-Multilevel preconditioners for HHO discretizations of the Stokes equations with static condensation,” *Communications on Applied Mathematics and Computation*, vol. 4, no. 3, pp. 783–822, 2022.
- [44] A. Brandt, S. McCormick, 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.
- [45] P. Vaněk, J. Mandel, and M. Brezina, “Algebraic Multigrid by smoothed aggregation for second and fourth order elliptic problems,” *Computing*, vol. 56, no. 3, pp. 179–196, 1996.
- [46] P. Vaněk, M. Brezina, and J. Mandel, “Convergence of Algebraic Multigrid based on smoothed aggregation,” *Numerische Mathematik*, vol. 88, pp. 559–579, 2001.
- [47] M. Brezina, R. Falgout, S. MacLachlan, T. Manteuffel, S. McCormick, and J. Ruge, “Adaptive Algebraic Multigrid,” *SIAM Journal on Scientific Computing*, vol. 27, no. 4, pp. 1261–1286, 2006.
- [48] 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*, vol. 22, no. 5, pp. 1570–1592, 2001.

- [49] J. E. Jones and P. S. Vassilevski, “AMGe based on element agglomeration,” *SIAM Journal on Scientific Computing*, vol. 23, no. 1, pp. 109–133, 2001.
- [50] 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*, vol. 25, no. 1, pp. 1–26, 2003.
- [51] V. A. Dobrev, R. D. Lazarov, P. S. Vassilevski, and L. T. Zikatanov, “Two-level preconditioning of discontinuous Galerkin approximations of second-order elliptic equations,” *Numerical Linear Algebra with Applications*, vol. 13, no. 9, pp. 753–770, 2006.
- [52] P. Bastian, M. Blatt, and R. Scheichl, “Algebraic Multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems,” *Numerical Linear Algebra with Applications*, vol. 19, no. 2, pp. 367–388, 2012.
- [53] P. F. Antonietti and L. Melas, “Algebraic Multigrid schemes for high-order nodal discontinuous Galerkin methods,” *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. A1147–A1173, 2020.
- [54] D. A. Di Pietro, F. Hülsemann, P. Matalon, P. Mycek, and U. Rüde, “Algebraic Multigrid preconditioner for statically condensed systems arising from lowest-order hybrid discretizations,” *SIAM Journal on Scientific Computing*, pp. S329–S350, 2023.
- [55] M. Raw, “Robustness of coupled Algebraic Multigrid for the Navier-Stokes equations,” in *34th Aerospace sciences meeting and exhibit*, p. 297, 1996.
- [56] J. M. Weiss, J. P. Maruszewski, and W. A. Smith, “Implicit solution of preconditioned Navier-Stokes equations using Algebraic Multigrid,” *AIAA Journal*, vol. 37, no. 1, pp. 29–36, 1999.
- [57] 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*, vol. 25, no. 2, pp. 623–642, 2003.
- [58] T. V. Kolev and P. S. Vassilevski, “Parallel auxiliary space AMG for H (curl) problems,” *Journal of Computational Mathematics*, pp. 604–623, 2009.
- [59] M. Griebel, D. Oeltz, and M. A. Schweitzer, “An Algebraic Multigrid method for linear elasticity,” *SIAM Journal on Scientific Computing*, vol. 25, no. 2, pp. 385–407, 2003.
- [60] N. A. Barnafi, L. F. Pavarino, and S. Scacchi, “A comparative study of scalable multilevel preconditioners for cardiac mechanics,” *Journal of Computational Physics*, p. 112421, 2023.
- [61] 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*, vol. 38, no. 1, pp. B1–B24, 2016.
- [62] 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*, vol. 357, p. 112575, 2019.
- [63] A. Arrarás, F. J. Gaspar, L. Portero, and C. Rodrigo, “Multigrid solvers for multipoint flux approximations of the Darcy problem on rough quadrilateral grids,” *Computational Geosciences*, vol. 25, pp. 715–730, 2021.
- [64] R. D. Falgout and P. S. Vassilevski, “On generalizing the Algebraic Multigrid framework,” *SIAM Journal on Numerical Analysis*, vol. 42, no. 4, pp. 1669–1693, 2004.
- [65] R. D. Falgout, P. S. Vassilevski, and L. T. Zikatanov, “On two-grid convergence estimates,” *Numerical linear algebra with applications*, vol. 12, no. 5-6, pp. 471–494, 2005.
- [66] L. T. Zikatanov, “Two-sided bounds on the convergence rate of two-level methods,” *Numerical Linear Algebra with Applications*, vol. 15, no. 5, pp. 439–454, 2008.
- [67] J. Xu and L. Zikatanov, “Algebraic Multigrid methods,” *Acta Numerica*, vol. 26, pp. 591–721, 2017.
- [68] U. M. Yang *et al.*, “BoomerAMG: A parallel Algebraic Multigrid solver and preconditioner,” *Applied Numerical Mathematics*, vol. 41, no. 1, pp. 155–177, 2002.
- [69] P. F. Antonietti, M. Caldana, and L. Dede’, “Accelerating Algebraic Multigrid methods via artificial neural networks,” *Vietnam Journal of Mathematics*, pp. 1–36, 2023.
- [70] M. Caldana, P. F. Antonietti, *et al.*, “A deep learning algorithm to accelerate algebraic multigrid methods in finite element solvers of 3D elliptic PDEs,” *Computers & Mathematics with Applications*, vol. 167, pp. 217–231, 2024.
- [71] D. N. Arnold, F. Brezzi, B. Cockburn, and L. Donatella Marini, “Unified analysis of discontinuous Galerkin methods for elliptic problems,” *SIAM Journal on Numerical Analysis*, vol. 39, no. 5, pp. 1749 – 1779, 2001.
- [72] P. Antonietti and I. Mazzieri, “High-order discontinuous Galerkin methods for the elastodynamics equation on polygonal and polyhedral meshes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 342, pp. 414 – 437, 2018.
- [73] L. Beirão Da Veiga, F. Brezzi, and L. Marini, “Virtual elements for linear elasticity problems,” *SIAM Journal on Numerical Analysis*, vol. 51, no. 2, pp. 794 – 812, 2013.

- [74] L. Beirão Da Veiga, F. Brezzi, L. Marini, and A. Russo, “Virtual element method for general second-order elliptic problems on polygonal meshes,” *Mathematical Models and Methods in Applied Sciences*, vol. 26, no. 4, pp. 729 – 750, 2016.
- [75] L. Beirão da Veiga, F. Dassi, and A. Russo, “High-order virtual element method on polyhedral meshes,” *Computers and Mathematics with Applications*, vol. 74, no. 5, pp. 1110 – 1122, 2017.
- [76] L. Beirão da Veiga, C. Lovadina, and D. Mora, “A virtual element method for elastic and inelastic problems on polytope meshes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 295, pp. 327 – 346, 2015.
- [77] L. Beirão da Veiga, F. Brezzi, L. D. Marini, and A. Russo, “The hitchhiker’s guide to the virtual element method,” *Mathematical Models and Methods in Applied Sciences*, vol. 24, no. 08, pp. 1541–1573, 2014.
- [78] A. Savitzky and M. J. Golay, “Smoothing and differentiation of data by simplified least squares procedures.,” *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.
- [79] A. H. Baker, R. D. Falgout, T. V. Kolev, and U. M. Yang, “Multigrid smoothers for ultraparallel computing,” *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2864–2887, 2011.
- [80] A. Hestenthaler, B. S. Southworth, D. Nordsletten, O. Rohrlé, R. D. Falgout, and J. B. Schroder, “Multilevel convergence analysis of multigrid-reduction-in-time,” *SIAM Journal on Scientific Computing*, vol. 42, no. 2, pp. A771–A796, 2020.
- [81] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.
- [82] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” *arXiv preprint arXiv:1711.05101*, 2017.
- [83] F. Dassi, “Vem++, a C++ library to handle and play with the virtual element method,” *Numerical Algorithms*, pp. 1–43, 2025.
- [84] M. Caldana, F. Dassi, I. Mazzieri, P. F. Antonietti, and L. Beirão da Veiga, “Vulpes: an open-source C++ library for virtual and discontinuous polytopal element methods,” *In preparation*, 2025.
- [85] R. D. Falgout and U. M. Yang, “hypre: A library of high performance preconditioners,” in *International Conference on computational science*, pp. 632–641, Springer, 2002.
- [86] S. Balay, S. Abhyankar, M. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, et al., *PETSc users manual*. Argonne National Laboratory, 2019.

MOX Technical Reports, last issues

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

- 59/2025** Wolf, F.; Botteghi, N.; Fasel, U.; Manzoni, A.
Interpretable and efficient data-driven discovery and control of distributed systems
- 58/2025** Pivato, C.A.; Cozzi, O.; Fontana, N.; Ieva, F., et al.
Clinical outcomes of percutaneous coronary interventions after transcatheter aortic valve replacement
- 57/2025** Guagliardi, O.; Masci, C.; Breschi, V; Paganoni A. ; Tanelli, M.
A Novel DNA-Inspired Framework to Study University Dropout: Insights from Politecnico di Milano
- 56/2025** Tonini, A.; Bui-Thanh, T.; Regazzoni, F.; Dede', L; Quarteroni, A.
Improvements on uncertainty quantification with variational autoencoders
- Tonini, A.; Bui-Thanh, T.; Regazzoni, F.; Dede', L; Quarteroni, A.
Improvements on uncertainty quantification with variational autoencoders
- 53/2025** Zecchi, A. A.; Sanavio, C.; Perotto, S.; Succi, S.
Telescopic quantum simulation of the advection-diffusion-reaction dynamics
- 54/2025** Tomasetto, M.; Williams, J.P.; Braghin, F.; Manzoni, A.; Kutz, J.N.
Reduced order modeling with shallow recurrent decoder networks
- 55/2025** Gimenez Zapiola, A.; Boselli, A.; Menafoglio, A.; Vantini, S.
Hyper-spectral Unmixing algorithms for remote compositional surface mapping: a review of the state of the art
- Gimenez Zapiola, A.; Boselli, A.; Menafoglio, A.; Vantini, S.
- 51/2025** Tomasetto, M.; Braghin, F., Manzoni, A.
Latent feedback control of distributed systems in multiple scenarios through deep learning-based reduced order models