REDUCING OPERATOR COMPLEXITY OF GALERKIN COARSE-GRID OPERATORS WITH MACHINE LEARNING

RU HUANG[*], KAI CHANG[*], HUAN HE[†], RUI PENG LI[‡], AND YUANZHE XI[*]

Abstract. We propose a data-driven and machine-learning-based approach to compute non-Galerkin coarse-grid operators in multigrid (MG) methods, addressing the well-known issue of increasing operator complexity. Guided by the MG theory on spectrally equivalent coarse-grid operators, we have developed novel machine learning (ML) algorithms that utilize neural networks (NNs) combined with smooth test vectors from multigrid eigenvalue problems. The proposed method demonstrates promise in reducing the complexity of coarse-grid operators while maintaining overall MG convergence for solving parametric partial differential equation (PDE) problems. Numerical experiments on anisotropic rotated Laplacian and linear elasticity problems are provided to showcase the performance and comparison with existing methods for computing non-Galerkin coarse-grid operators.

Key words. machine learning, multigrid methods, operator complexity, neural networks

AMS subject classifications. 65M55, 65F08, 65F10, 15A60

1. Introduction.

Multigrid (MG) methods are one of the most efficient and scalable iterative methods for solving linear systems of equations

$$Au = f \tag{1.1}$$

where the coefficient matrix $A \in \mathbb{R}^{N\times N}$ is sparse and large, and $u \in \mathbb{R}^N$ and $f \in \mathbb{R}^N$ are the solution and right-hand-side vectors respectively. For the systems that arise from elliptic-type partial differential equations (PDEs), MG methods often exhibit optimal linear computational complexities. Nevertheless, there is ongoing research focused on further improving the efficiency and scalability of MG methods, in particular for large-scale and challenging problems. By and large, the overall efficiency of iterative methods is determined by not only the convergence rate of the iterations but also the arithmetic complexity per iteration and the corresponding throughput on the underlying computing platform. In this work, we address a common issue in MG methods which is the growth of the coarse-grid operator complexity in the hierarchy. This operator is typically computed as the (Petrov–)Galerkin product from the operators in the fine level. Assuming A is symmetric positive definite (SPD) and $R = P^{\mathsf{T}}$, the Galerkin operator is optimal in the sense that it yields an orthogonal projector as the coarse-grid correction that guarantees to reduce the A-norm of the error. However, this operator can lead to the issue of decreasing operator sparsity, particularly at deeper levels of the MG hierarchy. This can impair the overall performance of MG by introducing challenges in terms of computational efficiency, memory requirements, and the communication cost in distributed computing environments [28, 2].

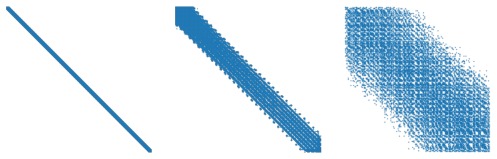
[*]Department of Mathematics, Emory University, Atlanta, GA 30322 ({ru.huang,kai.chang,yxi26}@emory.edu). The work is supported by NSF awards OAC 2003720 and DMS 2208412.

[†]Department of Biomedical Informatics, Harvard University, Boston, MA 02130 (huan_he@hms.harvard.edu)

[‡]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P. O. Box 808, L-561, Livermore, CA 94551 (li50@llnl.gov). This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD program under Project No. 23-FS-031 and 24-ERD-033.

Moreover, the increasing operator complexity can also affect the effectiveness and robustness of other MG components such as the coarsening and interpolation algorithms [26]. To demonstrate this problem, we consider classical MG methods for solving the 3-D Poisson's equation discretized on a $100 \times 100 \times 100$ grid with a 7-point stencil. The sparsity patterns of the operator matrix $A^{(l)}$ at the levels $l = 0, 3, 5$ are shown in Table 1. From these patterns, it is evident that the matrix bandwidth increases as the level goes deeper, as well as the stencil size (i.e., the average number of nonzeros per row). The increased sparsity often leads to not only a growth in computational cost but also an increase in data movement, which corresponds to the communication expense in parallel solvers. Figure 1 shows the time spent in the computation and communication in the first 6 levels of the MG hierarchies for solving a 3-D Poisson's problem. As depicted, there is a steep increase in the computational cost at level 2, coinciding with the level where the communication cost reaches its maximum.

TABLE 1
The sparsity patterns of $A^{(0)}$, $A^{(3)}$, and $A^{(5)}$ in a Multigrid hierarchy for solving the 3-D Poisson's equation on 4 processes (top). The size of the operator matrix (N), the number of nonzeros (NNZ) and the average number of nonzeros per row (RNZ) from the top 4 levels (bottom).



level	N	NNZ	RNZ
0	1,000,000	6,940,000	7
1	499,891	8,418,739	17
2	115,515	5,328,543	46
3	14,479	1,125,707	78

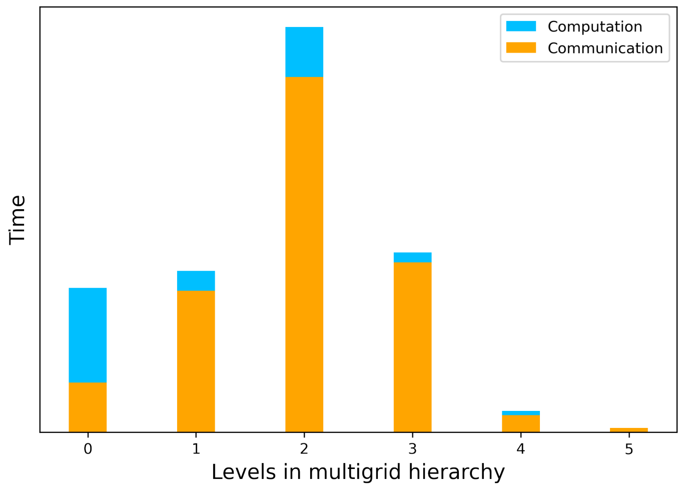


FIG. 1. *The cost of computation and communication in the first 6 levels of parallel Algebraic MG methods for solving the 3-D Poisson's equation on 8192 processes of Blue Waters, a Cray XK/XE supercomputer at the National Center for Supercomputing Applications. Image source: [5]*

One approach to reducing the coarse-grid operator complexity is to “sparsify” the (Petrov–)Galerkin operator after it is computed, i.e., removing some nonzeros outside a given sparsity pattern. The obtained sparsified operator is often called a “non-Galerkin” coarse-grid operator. The methods developed in [36, 33] leverage algebraically smooth basis vectors and the approximations to the fine grid operator to explicitly control the coarse grid sparsity pattern. The algorithms introduced in [11, 32] first determine the patterns of the sparsified operator based on heuristics on the path of edges in the corresponding graph and then compute the numerical values to ensure the spectral equivalence to the Galerkin operator for certain types of PDEs. Improving the parallel efficiency of AMG by reducing the communication cost with the non-Galerkin operator was discussed in [4]. These existing algorithms for computing non-Galerkin operators are usually based on heuristics on the associated graph and the characteristic of the underlying PDE problem, such as the information of the near kernels of A. Therefore, they are problem-dependent, and often times it can be difficult to devise such heuristics that are suitable for a broader class of problems.

Recently, there has been a line of work in the literature to leverage data-driven and machine learning (ML) based methods to improve the robustness of MG. In particular, [27, 24, 12] deal with learning better prolongation operators. Techniques of deep reinforcement learning (DRL) are exploited in [31] to better tackle the problem of Algebraic MG coarsening combined with the diagonal dominance ratio of the F-F block. Both the works in [21] and [25] focus on the problem of designing better smoothers. In [21], smoothers are directly parameterized by multi-layer convolution neural networks (CNNs) while [25] optimizes the weights in the weighted Jacobi smoothers. In this paper, we follow this line of research and propose a data-driven and ML-based method for non-Galerkin operators. In this work, we restrict our focus on problems on *structured meshes* and with *constant coefficients*. The innovations and features of the proposed method are summarized as follows: 1) Introduction of a multi-level algorithm based on ML methods to sparsify all coarse-grid operators in the multigrid hierarchy; 2) Successful reduction of operator density while preserving the convergence behavior of the employed multigrid method; 3) Applicability of the proposed NN model to a class of parametric PDEs with parameters following specific probability distributions; 4) Ability to train the sparsified coarse-grid operator on each level in parallel once the training data is prepared; 5) Flexibility for the user to choose the average number of non-zero entries per row in the coarse-grid operators, with a minimum threshold requirement. To the best of our knowledge, our proposed work is the first to utilize ML models for controlling sparsity within multigrid hierarchies.

The rest of the paper is organized as follows. We first briefly review the preliminaries of AMG methods and the non-Galerkin algorithms in section 2. We elaborate on our proposed sparsification algorithm in section 3. Numerical experiments and results are presented in section 4. Finally, we conclude in section 5.

2. MG preliminaries and coarse-grid operators.

In this section, we give a brief introduction to MG methods and the Galerkin coarse-grid operators. The MG method is a multilevel method that utilizes a hierarchy of grids, consisting of fine and coarse levels, and constructs coarse-level systems at different scales that can capture the essential information of the fine-level system while reducing the problem size. MG algorithms employ techniques such as coarsening, relaxation, restriction and interpolation to transfer information between the grid levels to accelerate the solution process. Algorithm 2.1 presents the most commonly used MG V-cycle scheme. It uses

ν steps of pre- and post-smoothing, where M and M^T are the smoothing operators. Matrices R and P are the restriction and prolongation operators, respectively. The coarse-grid operator is computed in Step 3 via the Galerkin product, $A_g = RAP$. The aim of smoothing is to quickly annihilate the high-frequency errors via simple iterative methods such as relaxation, whereas the low-frequency errors are targeted by the Coarse-Grid Correction (CGC) operator, $I - P(RAP)^{-1}RA$. When A is SPD and $R = P^\mathsf{T}$, the CGC operator is A-orthogonal with the Galerkin operator A_g.

Algorithm 2.1 Multigrid V-Cycle for solving $A^{(l)}u^{(l)} = f^{(l)}$ at level l

```
1: Pre-smoothing: $u^{(l)} := (I - (M^{(l)})^{-1}A^{(l)})u^{(l)} + (M^{(l)})^{-1}f^{(l)}$ for $\nu$ steps
2: Compute residual $r^{(l)} = f^{(l)} - A^{(l)}u^{(l)}$ and the restriction $r^{(l+1)} = R^{(l)}r^{(l)}$
3: Compute Galerkin operator $A_g = R^{(l)}A^{(l)}P^{(l)}$ and let $A^{(l+1)} = A_g$
4: if $l = L - 1$ then
5:     Solve $A^{(l+1)}u^{(l+1)} = r^{(l+1)}$ with an arbitrary method
6: else
7:     Let $u^{(l+1)} = 0$ and $f^{(l+1)} = r^{(l+1)}$. Go to Step 1 with $l := l + 1$.
8: end if
9: Prolongate and correct: $u^{(l)} := u^{(l)} + P^{(l)}u^{(l+1)}$
10: Post-smoothings: $u^{(l)} := (I - (M^{(l)})^{-\mathsf{T}}A^{(l)})u^{(l)} + (M^{(l)})^{-\mathsf{T}}f^{(l)}$ for $\nu$ steps
```

2.1. Non-Galerkin operators. Naive approaches, such as indiscriminately removing nonzero entries in the Galerkin operators based on the magnitude often result in slow convergence of the overall MG method (see the example provided in Section 3 of [11]). To address the aforementioned challenges arising from the increased operator complexity, alternative operators, denoted by A_c, that are not only sparser than A_g but also spectrally equivalent have been studied and have been used in lieu of the Galerkin operator A_g [11, 23]. We say two matrices are spectrally equivalent defined in [11] as follows:

Definition 2.1. *SPD matrices A_g and A_c are spectrally equivalent if*

$$0 < \alpha \le \lambda(A_g^{-1}A_c) \le \beta, \tag{2.1}$$

with α and β both close to 1, where $\lambda(\cdot)$ denotes eigenvalues of a matrix.

The convergence rate of MG can be analyzed through the spectral radius of the error propagation matrix. For example, the two-grid error propagation matrix corresponding to the V-cycle in Algorithm 2.1 reads

$$E_g = (I - M^{-\mathsf{T}}A)^\nu (I - PA_g^{-1}RA)(I - M^{-1}A)^\nu. \tag{2.2}$$

With the replacement of A_g by A_c, it becomes

$$E_c = (I - M^{-\mathsf{T}}A)^\nu (I - PA_c^{-1}RA)(I - M^{-1}A)^\nu. \tag{2.3}$$

The spectrum property of E_c is analyzed in the following theorem.

Theorem 2.2 ([11]). *Denoting by B_g and B_c respectively the corresponding preconditioning matrices defined as $B_g^{-1} = (I - E_g)A^{-1}$ and $B_c^{-1} = (I - E_c)A^{-1}$ and assuming A_c and A_g are both SPD and*

$$\eta = \|I - A_cA_g^{-1}\|_2 = \|I - A_g^{-1}A_c\|_2 < 1, \tag{2.4}$$

for the preconditioned matrix, we have

$$\kappa(B_c^{-1}A) \leq \frac{1+\eta}{1-\eta}\kappa(B_g^{-1}A), \tag{2.5}$$

and moreover

$$\rho(E_c) \leq \max\left(\frac{\lambda_{\max}(B_g^{-1}A)}{1-\eta} - 1, 1 - \frac{\lambda_{\min}(B_g^{-1}A)}{1+\eta}\right), \tag{2.6}$$

where $\lambda_{\max}(\cdot)$ *and* $\lambda_{\min}(\cdot)$ *are the largest and smallest eigenvalues respectively,* $\kappa(\cdot)$ *denotes the condition number and* $\rho(\cdot)$ *denotes the spectrum radius.*

The quantity η measures the degree of spectral equivalence between the operators A_g and A_c, i.e., only when η is small, these operators are spectrally equivalent, since $\rho(\cdot) \leq \|\cdot\|_2$. Clearly, the condition number of the preconditioned matrix and the two-grid convergence with respect to E_c deteriorate as η increases. With fixed $B_g^{-1}A$, we can establish a criterion for the convergence of E_c with respect to η, as shown in the next result.

Corollary 2.3. *Suppose* $\eta < 1 - \lambda_{\max}(B_g^{-1}A)/2$. *Then, the two-grid method* (2.3) *converges.*

Proof. Note that $\eta < 1-\lambda_{\max}(B_g^{-1}A)/2$ implies that $\lambda_{\max}(B_g^{-1}A)/(1-\eta)-1 < 1$. Since $\lambda(B_g^{-1}A) > 0$, $1 - \lambda_{\min}(B_g^{-1}A)/(1+\eta) < 1$. Therefore, from (2.6), it follows that $\rho(E_c) < 1$. ◻

2.2. Spectrally equivalent stencils. In this paper, we focus on structured matrices that can be represented by stencils and grids, for the definitions and notations, see, e.g., [34]. These structured matrices exhibit a unique property wherein two spectrally equivalent stencils can determine two sequences of spectrally equivalent matrices with increasing sizes and thus ensures the convergence of E_c with increasing matrix sizes.

Definition 2.4 ([1, 6]). *Let* $\{A_j\}$ *and* $\{B_j\}$ *be two sequences of (positive definite) matrices with increasing size* N_j, *where* A_j *and* $B_j \in \mathbb{R}^{N_j\times N_j}$. *If* A_j *and* B_j *are spectrally equivalent as defined in* (2.1) *for all* j *with* α *and* β *that are independent of* N_j, *then the sequences* $\{A_j\}$ *and* $\{B_j\}$ *are called spectrally equivalent sequences of matrices.*

The above definition yields the definition of spectrally equivalent stencils given as follows.

Definition 2.5 ([6]). *Suppose the sequences of matrices* $\{A_j\}$ *and* $\{B_j\}$ *are constructed with the stencils* $\mathcal{A}$ *and* $\mathcal{B}$ *respectively, where for any given* j, A_j *and* B_j *have the same size. We call* $\mathcal{A}$ *and* $\mathcal{B}$ *are spectrally equivalent if* $\{A_j\}$ *and* $\{B_j\}$ *are spectrally equivalent sequences of matrices.*

At the end of this section, we provide an example of spectrally equivalent stencils. Consider the following 9-point stencil that was used in the study of the MG method for circulant matrices [6]:

$$\begin{bmatrix} c & b & c \\ a & -2(a+b)-4c & a \\ c & b & c \end{bmatrix}. \tag{2.7}$$

It was proved that the associated 5-point stencil

$$\begin{bmatrix} & b+2c & \\ a+2c & -2(a+b)-8c & a+2c \\ & b+2c & \end{bmatrix}. \tag{2.8}$$

is spectrally equivalent to (2.7). Results for the 7-point stencil in 3-D that is spectrally equivalent to a 27-point stencil can also be found in [6].

2.3. Numerical heuristics for spectral equivalence. Directly optimizing (2.4), which involves a matrix norm, to find a spectrally equivalent A_c to A_g appears to be challenging. Instead, a more viable approach is via test vectors that correspond to the low-frequency modes of A_g (see, e.g., [11, 36]). These low-frequency modes represent the algebraically smooth modes at a coarse level, which are important for the interpolation to transfer to the fine level within the MG hierarchy. From the perturbed error propagation operator (2.3), it follows that after the pre-smoothing steps, the remaining error, denoted by e, that is algebraically smooth in terms of A (i.e., $Ae \approx 0$) needs to be efficiently annihilated by the coarse-grid. This smooth error (of low frequency) is ideally in the range of interpolation operator P, meaning that, $e = Pe_c$ with some coarse-grid error e_c. Furthermore, e_c is smooth with respect to A_g with a proper R, since $A_g e_c = RAPe_c = RAe$ is small. Therefore, for an effective CGC with non-Galerkin coarse-grid operator A_c, it is essential for $(I - PA_c^{-1}RA)e = (I - PA_c^{-1}RA)Pe_c = P(I - A_c^{-1}A_g)e_c$ to be small, which implies $A_g e_c \approx A_c e_c$. That is to enforce the accuracy of $A_c e_c$ compared to $A_g e_c$ with the low-frequency vector e_c on the coarse level.

In this paper, we adopt the approach of multigrid eigensolver (MGE) [7] to compute the smooth vectors in the MG hierarchy. First, consider two-grid MG methods. The Rayleigh quotient of Pe_c with respect to A reads

$$r(A, Pe_c) = \frac{(APe_c, Pe_c)}{(Pe_c, Pe_c)} = \frac{(P^{\mathsf{T}}APe_c, e_c)}{(P^{\mathsf{T}}Pe_c, e_c)} = \frac{(A_g e_c, e_c)}{(Te_c, e_c)}, \tag{2.9}$$

where $T = P^{\mathsf{T}}P$. Therefore, the desired smooth modes that minimize (2.9) relate to the eigenvectors that correspond to the small eigenvalues of the generalized eigenvalue problem

$$A_g u = \lambda T u, \quad T = P^{\mathsf{T}}P. \tag{2.10}$$

For MG methods with more than 2 levels, we can compute the smooth vectors at each coarse level by recursively applying (2.10) at the previous fine level.

3. An ML method for coarse-grid operators.

We aim to utilize ML techniques to compute non-Galerkin operators in the MG method for solving (1.1), where A is a stencil-based coefficient matrix that corresponds to PDE problems discretized on Cartesian grids. On a given MG level $l > 1$, with stencil $\mathcal{A}_g^{(l)}$ associated with the Galerkin matrix $A_g^{(l)}$, we construct a sparser stencil $\mathcal{A}_c^{(l)}$ in the following 3 steps, which are explained in detail below and illustrated in Figure 2.

Step 1. Select the pattern of $\mathcal{A}_c^{(l)}$, where the corresponding entries are assumed to be nonzero. The NN in this step, denoted by $F_{\Theta^{(l)}}$, is parametrized by $\Theta^{(l)}$. It computes the *location probability*, i.e., the probability of a nonzero entry appears at a location, for each of the stencil entries of $\mathcal{A}_g^{(l)}$. We apply the NN $F_{\Theta^{(l)}}$ to the vectorized stencil $v_g^{(l)} = \mathrm{vec}(\mathcal{A}_g^{(l)})$, i.e., the vector reshaped from the stencil array, followed by a *softmax* layer. Therefore, the output of the NN can be written as

$$\mathcal{P}^{(l)} = \mathrm{softmax}(F_{\Theta^{(l)}}(v_g^{(l)})), \tag{3.1}$$

which is then reshaped back to match the shape of $\mathcal{A}_g^{(l)}$. Given that $0 < \mathcal{P}^{(l)} < 1$, each entry can be interpreted as the probability of the entry appearing (being nonzero) in the sparsified stencil $\mathcal{A}_c^{(l)}$. With that, we select the largest k entries of $\mathcal{P}^{(l)}$,

$$\mathcal{I}^{(l)} = \left\{ i \,\middle|\, \mathcal{P}^{(l)}(i) \text{ is one of the largest } k \text{ entries of } \mathcal{P}^{(l)} \right\}, \tag{3.2}$$

where $\mathcal{I}^{(l)}$ denotes the set of the indices of those entries, from which we build a mask Boolean vector $\mathcal{M}^{(l)}$ defined as

$$\mathcal{M}^{(l)}(i) = \begin{cases} 1, & \text{if } i \in \mathcal{I}^{(l)} \\ 0, & \text{otherwise} \end{cases}, \tag{3.3}$$

that determines the positions of the nonzeros in the non-Galerkin stencil.

Step 2. Compute the numerical values of the nonzero entries. The NN in this step, denoted by $G_{\Psi^{(l)}}$, is parametrized by $\Psi^{(l)}$ which is applied to the same input as in Step 1. The output from NN of this step reads

$$\mathcal{V}^{(l)} = G_{\Psi^{(l)}}(v_g^{(l)}), \tag{3.4}$$

which determines the numerical values of the nonzero entries.

Step 3. Construct $\mathcal{A}_c^{(l)}$ by point-wise multiplication. The non-Galerkin stencil is computed by the Hadamard (or element-wise) product

$$\mathcal{A}_c^{(l)} = \mathcal{M}^{(l)} \odot \mathcal{V}^{(l)}. \tag{3.5}$$

We summarize these steps in Algorithm 3.1. The MG V-cycle using the sparsified coarse grid is outlined in Algorithm 3.2, which closely resembles Algorithm 2.1, whereas, instead of using the Galerkin operator for coarser levels, the non-Galerkin operator A_c is constructed from the sparsified stencil generated by Algorithm 3.1.

Remark 3.1. A few remarks on Algorithm 3.1 and Algorithm 3.2 follow. To begin with, the parameter k of Algorithm 3.1 signifies the number of the nonzero entries in the sparsified stencil. This effectively gives us the ability to directly manipulate the complexity of the resulting non-Galerkin operator. Secondly, in the NN implementations, we ensure that the shapes of $\mathcal{M}^{(l)}$ and $\mathcal{V}^{(l)}$ are identical. This enables the proper application of the Hadamard product. Lastly, it is assumed that the NNs, F_{Θ_l} and G_{Ψ_l}, have undergone sufficient training. Therefore, Step 7 of Algorithm 3.2 involves merely the application of the trained NNs.

Algorithm 3.1 SparsifyStencil

```
Input: $\mathcal{A}_g, F_\Theta, G_\Psi, k$
1: Apply the NNs to compute $\mathcal{P} = F_\Theta(\mathcal{A}_g)$ and $\mathcal{V} = G_\Psi(\mathcal{A}_g)$
2: $\mathcal{M}$ has the same shape as $\mathcal{V}$ and has a value of 1 at the entries corresponding to
   the $k$ largest values of $\mathcal{P}$, with 0 elsewhere.
3: return $\mathcal{A}_c = \mathcal{M} \odot \mathcal{V}$
```

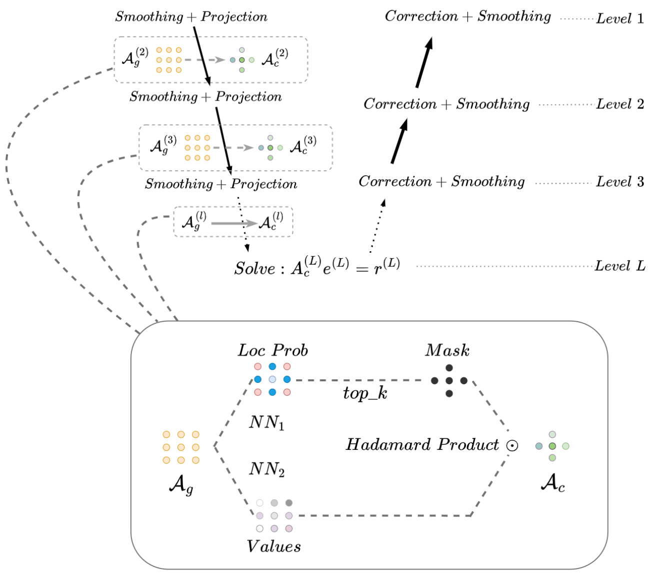


FIG. 2. *Illustration of the ML algorithm for computing coarse-grid operators with NNs.*

Algorithm 3.2 MG V-Cycle with sparsified coarse-grid operator

```
 1: Pre-smoothing: $u^{(l)} := (I - (M^{(l)})^{-1} A^{(l)}) u^{(l)} + (M^{(l)})^{-1} f^{(l)}$ for $\nu$ steps
 2: Compute the residual $r^{(l)} = f^{(l)} - A^{(l)} u^{(l)}$ and restriction $r^{(l+1)} = R^{(l)} r^{(l)}$
 3: Compute the Galerkin operator $A_g^{(l+1)} = R^{(l)} A^{(l)} P^{(l)}$
 4: if $l = L - 1$ then
 5:     Solve $A_g^{(l+1)} u^{(l+1)} = r^{(l+1)}$ with an arbitrary method
 6: else
 7:     Apply Algorithm 3.1: $\mathcal{A}_c^{(l+1)} = \mathrm{SparsifyStencil}(\mathcal{A}_g^{(l+1)}, F_{\Theta^{(l+1)}}, G_{\Psi^{(l+1)}}, k)$
 8:     Let $A^{(l+1)} = A_c^{(l+1)}$, $u^{(l+1)} = 0$ and $f^{(l+1)} = r^{(l+1)}$. Go to Step 1 with $l := l+1$
 9: end if
10: Prolongate and correct: $u^{(l)} := u^{(l)} + P^{(l)} u^{(l+1)}$
11: Post-smoothings: $u^{(l)} := (I - (M^{(l)})^{-\mathsf{T}} A^{(l)}) u^{(l)} + (M^{(l)})^{-\mathsf{T}} f^{(l)}$ for $\nu$ steps
```

3.1. NN training algorithm. In this section, we delve into the specifics of the training algorithm that enables NNs to generate a sparser coarse-grid operator stencil than the Galerkin operator stencil, without impairing the overall convergence of the MG method. A key component of Algorithm 3.2 is line 7 where F_Θ and G_Ψ are the pre-trained NNs. The loss function, another crucial component, is pivotal to the training procedure. Based on the discussion in subsection 2.3, we aim to minimize the discrepancy between $A_g v$ and $A_c v$ where v is an algebraically smooth vector.

Denote by $\beta \in \mathbb{R}^p$ parameters of the problem to solve, which possesses a probability distribution p_β in $\mathscr{B}$. The loss function tied to F_Θ and G_Ψ is defined as:

$$\mathcal{L}_\beta \left(F_\Theta, G_\Psi, \mathcal{A}_g^\beta, \{v_j^\beta\}_{j=1}^s \right) = \sum_{j=1}^{s} \| A_g^\beta v_j^\beta - A_c^\beta v_j^\beta \|_2^2, \tag{3.6}$$

where $\{v_j^\beta\}$ represents the set of algebraically smooth vectors, s is the number of these vectors, and $\mathcal{A}_c^\beta$ is computed by Algorithm 3.1. The objective is to minimize the expectation of $\mathcal{L}_\beta$ under the distribution of β, symbolized as $\mathbb{E}_{\beta\sim p_\beta}[\mathcal{L}_\beta]$, throughout the training. It is worth mentioning that instead of explicitly forming the matrices A_g^β and A_c^β, we adopt a stencil-based approach where the matrix-vector multiplications are performed as the convolutions of the stencils $\mathcal{A}_g^\beta$ and $\mathcal{A}_c^\beta$ with vectors that are padded with zero layers, assuming zero boundary conditions are used. The stencil-based approach and the convolution formulation greatly enhance memory efficiency during training.

3.2. Details of Training and Testing. In this section, we provide details of the training and testing algorithms.

Architecture of the multi-head attention. We use multi-head attention [35] to compute both location probability in Step 1 with $F_{\Theta^{(l)}}$ and numerical values in Step 2 with $G_{\Psi^{(l)}}$ as discussed in section 3. We adopt the standard architecture that comprises a set of n_h independent attention heads, each of which extracts different features from each stencil entry of $\mathcal{A}_g^{(l)}$. In essence, each head generates a different learned weighted sum of the input values, where the weights are determined by the attention mechanism and reflect the importance of each value. The weights are calculated using a softmax function applied to the scaled dot-product of the input vectors. The output from each head is then concatenated and linearly transformed to produce the final output.

The multi-head attention mechanism in our study is formally defined as follows: Let $v_g^{(l)}$ denote the input vectors. For each attention head $h_i, i = 1, \ldots, n_h$, we first transform the inputs using parameterized linear transformations, W_i^Q, W_i^K, and W_i^V to produce the vectors of query Q_i, key K_i, and value V_i as follows:

$$Q_i = v_g^{(l)} W_i^Q, \quad K_i = v_g^{(l)} W_i^K, \quad V_i = v_g^{(l)} W_i^V. \tag{3.7}$$

The attention scores for each input vector in head h_i are then computed using the scaled dot-product of the query and key vectors, followed by a softmax function:

$$\text{Attention}_i = \text{softmax}\left(\frac{Q_i K_i^T}{\sqrt{d_k}}\right) V_i \ , \tag{3.8}$$

where d_k is the dimension of the key vectors. This process captures the dependencies among the input vectors based on their similarities. The output of each attention head h_i is then concatenated, and a linear transformation is applied using a parameterized weight matrix W^O with softmax activation, which ensures positive outputs:

$$\text{MultiHead}(v_g^{(l)}) = \text{Concat}(\text{Attention}_1, \ldots, \text{Attention}_{n_h}) W^O. \tag{3.9}$$

This architecture empowers the model to learn complex PDE stencil patterns effectively. The design is flexible, and the number of heads can be adjusted as per the complexity of the task.

Intuition of selection of multi-head attention. We first briefly explain why multi-head attention is beneficial for PDE stencil learning than other types of NNs. This work is about teaching the NNs to generate stencils, which are essentially small patterns or templates used in the discretization of PDEs. These stencils represent the relationship between a grid point and its neighbors. In the context of PDE stencil learning, multi-head attention can be highly beneficial for several reasons:

1. Feature diversification: The multi-head attention allows the model to focus on various features independently, and thus, can capture a wider variety of patterns in the data. For PDE stencil learning, this means that the model can understand the relationships between different grid points more comprehensively.

2. Context awareness: Attention mechanisms inherently have the capacity to consider the context, i.e., the relationships between different parts of the input data. In PDE stencil learning, this translates to understanding the interactions between a grid point and its surrounding neighboring points.
3. Flexibility: Multi-head attention adds flexibility to the model. Each head can learn to pay attention to different features, making the model more adaptable. In the context of PDEs, this means that one head can learn to focus on local features (such as the values of nearby points), while another might focus more on global or structural aspects.

We observed empirically that our proposed architecture outperforms standard deep NNs (vanilla deep NNs) in stencil learning. This improvement is primarily attributed to the attention mechanism, which effectively addresses local structure capturing challenge inherent in fully connected layers. PDE problems often involve spatial and temporal structures that are local in nature. Fully connected networks, due to their global connectivity pattern, may struggle to effectively learn these local structures. They treat input data as a flattened array without considering the spatial correlations, which are critical in PDEs and yield sub-optimal performance. Through our empirical experiments, we found that the MG method, enhanced with attention-based operator learning, significantly outperforms models based solely on fully connected layers. Specifically, it achieved a 30% reduction in the number of iterations required to converge on a 2-D linear elasticity problem. This finding aligns with other studies, such as [13], which demonstrated that fully connected layers suffer from learning the complex dependencies among a grid.

Details of training and testing. Parameters β_i of PDE problems are sampled from distribution $\mathscr{B}$ according to the probability density function p_β to get the set of N_t parameters, $\{\beta_i\}$, $i = 1, \dots, N_t$. Then, we construct the corresponding set of fine-grid stencils $\{\mathcal{A}^{\beta_i}\}$. For all the tests in this paper, we use full coarsening, full-weighting restriction, and the corresponding bi-linear interpolation for all the levels of MG. At each level $l > 1$, the ML model is built with the Galerkin stencils $\{(\mathcal{A}_g^{\beta_i})^{(l)}\}$ and a set of smooth test vectors $\{(v_j^{\beta_i})^{(l)}\}$, $j = 1, \dots, s_l$, associated with each of the stencil, using the loss function

$$\mathcal{L} = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}_{\beta_i} \left(F_{\Theta^{(l)}}, G_{\Psi^{(l)}}, (\mathcal{A}_g^{\beta_i})^{(l)}, \{(v_j^{\beta_i})^{(l)}\} \right) \tag{3.10}$$

that is used to approximate $\mathbb{E}_{\beta \sim p_\beta}[\mathcal{L}_\beta]$. The complete training procedure is summarized in Algorithm 3.3. It is important to note that the NN trainings are independent of each other on different levels. Therefore, the training of the NNs for each level can be carried out simultaneously once the training data is prepared, taking advantage of parallel computing resources. The testing set is constructed with parameters that differ from those in the training set. This means that we test the model on a set of PDE parameters $\{\beta_j\}$, $j = 1, \dots, N_v$, that have not been encountered by the models during training. The purpose of the testing set is to assess the generalization capability of new problem instances.

4. Numerical Results.

We report the numerical results of the proposed ML-based non-Galerkin coarse-grid method in this section. All the ML models in the work[1] were written with PyTorch 1.9.0 [29]. We use PyAMG 4.2.3 [3] to build the

[1]The codes is available at https://anonymous.4open.science/r/Sparse-Coarse-Operator-11C7

Algorithm 3.3 Training NNs for computing coarse-grid operator at level l

```
Input: Interpolation operator $P^{(l)}$, Galerkin coarse-grid stencils $\{(\mathcal{A}_g^{\beta_i})^{(l)}\}$, the number of test vectors $s_l$ and the target stencil complexity $k$
Output: NNs $F_{\Theta^{(l)}}$ and $G_{\Psi^{(l)}}$
1: Generate test vectors $(v_j^{\beta_i})^{(l)}$, $j = 1, \ldots, s_l$, for each $(\mathcal{A}_g^{\beta_i})^{(l)}$ as follows
     • Compute $T^{(l)} = (P^{(l)})^\top P^{(l)}$
     • Compute the eigenvalues and vectors of
           $$(A_g^{\beta_i})^{(l)} u = \lambda_i^{(l)} T^{(l)} u,$$
       where $(A_g^{\beta_i}))^{(l)}$ is the coefficient matrix at level $l$ corresponding to stencil $(\mathcal{A}_g^{\beta_i})^{(l)}$.
     • The test vectors are the eigenvectors associated with the $s_l$ smallest eigenvalues
2: Initialize $G_{\Theta^{(l)}}$ and $G_{\Psi^{(l)}}$
3: repeat
4:     Apply Algorithm 3.1: $(\mathcal{A}_c^{\beta_i})^{(l)} = \text{SparsifyStencil}\left((\mathcal{A}_g^{\beta_i})^{(l)}, F_{\Theta^{(l)}}, G_{\Psi^{(l)}}, k\right)$
5:     Compute the gradient of the loss (3.10)
6:     Update the weights $\Theta^{(l)}$ and $\Psi^{(l)}$
7: until the prescribed number of training epochs is reached
```

MG hierarchy. All the experiments were performed on a workstation with Intel Core i7-6700 CPUs. The multi-head attention model we implemented has a total dimension of 256 and comprises 8 heads, which is also incorporated a dropout scheme with the rate of 0.7.

4.1. Evaluation Metrics.

In this section, we evaluate the performance of the proposed ML-based approach by comparing the average number of iterations required by the MG method using different coarse-grid operators to converge. Additionally, we verify the spectral equivalence of the Galerkin and sparsified non-Galerkin stencils by computing the spectra of the corresponding matrices on meshes of various sizes.

4.2. Spectrally equivalent stencils.

We first examine the proposed ML-based method on the 9-point stencil problem (2.7) that allows direct evaluation of the learned non-Galerkin operator by the comparison with the theoretical result (2.8), which is a spectrally equivalent 5-point stencil. We use the 9-point stencil $\mathcal{A}$ of the form (2.7) with $a = 1.417$, $b = 2.720$ and $c = 0.000114$, i.e.,

$$\mathcal{A} = \begin{bmatrix} 0.000114 & 2.720 & 0.000114 \\ 1.417 & -8.27456 & 1.417 \\ 0.000114 & 2.720 & 0.000114 \end{bmatrix} \tag{4.1}$$

as the fine-level A-operator, and the 2-D full-weighting stencil,

$$\mathcal{R} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}, \tag{4.2}$$

for the restriction operator. Thus, the stencil of the Galerkin operator is

$$\mathcal{A}_g = \begin{bmatrix} 0.129294125 & 0.42140075 & 0.129294125 \\ 0.095650750 & -1.55182350 & 0.095650750 \\ 0.129294125 & 0.42140075 & 0.129294125 \end{bmatrix}, \tag{4.3}$$

which has the same form as $\mathcal{A}$. From (2.8), a 5-point stencil that is spectrally equivalent to (4.3) is given by

$$\mathcal{A}_c = \begin{bmatrix} 0 & 0.679989 & 0 \\ 0.354239 & -2.068456 & 0.354239 \\ 0 & 0.679989 & 0 \end{bmatrix}. \tag{4.4}$$

Using Algorithm 3.3 with the prescribed number of nonzeros in the stencil $k = 5$ on the 31×31 grid, the pre-trained NNs produced the following 5-point stencil

$$\mathcal{A}_{nn} = \begin{bmatrix} 0 & 0.663 & 0 \\ 0.347 & -2.024 & 0.348 \\ 0 & 0.666 & 0 \end{bmatrix}, \tag{4.5}$$

denoted by $\mathcal{A}_{nn}$, which is close to the theoretical result (4.4). To assess the convergence behavior of the MG method, we solve a linear system using the coefficient matrix defined by (4.1). We conduct these tests on larger-sized grids and use the two-grid MG methods employing $\mathcal{A}_g$, $\mathcal{A}_c$, and $\mathcal{A}_{nn}$ as the coarse-level operators, respectively. The right-hand-side vector is generated randomly. The stopping criterion of MG iterations with respect to the relative residual norm is set to be 10^{-6}. The results are shown in Table 2, from which we can observe that all three methods require the same number of iterations.

Table 2
The number of iterations required by the two-level MG methods for solving a linear system corresponding to the coefficient matrix stencil (4.1) to 10^{-6} accuracy in terms of the relative residual and the α and β in (2.1) for different grid sizes. The MG methods utilize $\mathcal{A}_g$, $\mathcal{A}_c$, and $\mathcal{A}_{nn}$ as the coarse-level operator respectively. The tests are carried out on grid sizes up to 511×511.

		grid size						
		63	95	127	191	255	383	511
$\mathcal{A}_c$	α	1.0073	1.0021	1.0018	1.0013	1.0011	1.0004	1.0002
	β	1.9990	1.9996	1.9998	2.0000	2.0001	2.0001	2.0002
$\mathcal{A}_{nn}$	α	0.9771	0.9804	0.9796	0.9780	0.9778	0.9776	0.9776
	β	1.9556	1.9563	1.9566	1.9567	1.9568	1.9569	1.9568

	grid size						
	63	95	127	191	255	383	511
$\mathcal{A}_g$	11	10	10	10	10	10	10
$\mathcal{A}_c$	11	10	10	10	10	10	10
$\mathcal{A}_{nn}$	11	10	10	10	10	10	10

4.3. 2-D rotated Laplacian problem. In this section, we consider the 2-D anisotropic rotated Laplacian problem

$$-\nabla \cdot (T_{\theta,\xi} \nabla u(x, y)) = f(x, y), \tag{4.6}$$

where the 2×2 vector field $T_{\theta,\xi}$ parameterized by θ and ξ is defined as

$$T_{\theta,\xi} = \begin{bmatrix} \cos^2\theta + \xi \sin^2\theta & \cos\theta \sin\theta(1-\xi) \\ \cos\theta \sin\theta(1-\xi) & \sin^2\theta + \xi \cos^2\theta \end{bmatrix}$$

with θ being the angle of the anisotropy and ξ being the conductivity.

We show that the proposed approach is not limited to a particular set of parameters but remains effective across a range of values for both ξ and θ. In the first set of experiments, we fix the value ξ while allowing θ to follow a uniform distribution within a specified interval. We conduct 12 experiments where each $\xi \in \{100, 200, 300, 400\}$ is paired up with θ sampled from intervals $\{(\pi/4, \pi/3), (\pi/3, 5\pi/12), (\pi/2, 7\pi/12)\}$. The MG methods for solving these problems use full-coarsening, full-weighting restriction, and the Gauss-Seidel method for both pre-smoothing and post-smoothing. MG V-cycles are executed until the residual norm is reduced by 6 orders of magnitude. The number of the nonzero elements is 9 in the Galerkin coarse-grid stencil across the MG levels, whereas we choose to reduce the number to 5 for the non-Galerkin operator.

During the training phase of each experiment, the model is provided with 5 instances for a given ξ with different θ values evenly distributed within a chosen interval. For example, for $\xi = 100$ and $\theta \in (\pi/4, \pi/3)$, the parameters for the 5 instances of (ξ, θ) are selected as follows:

$$\{(100, \pi/4), (100, 3.25\pi/12), (100, 3.5\pi/12), (100, 3.75\pi/12), (100, \pi/3)\}. \tag{4.7}$$

The size of the fine-level matrix in the training instances is set to be 31×31. In the testing phase, 10 distinct θ values are randomly selected from the chosen interval. The MG parameters are the same as those used in the training phase. In the testing, it should be noted that the fine-level problem size is 511×511, which is approximately 256 times larger than that in the training instances. This larger problem size in the testing allows for a more rigorous evaluation of the performance of MG and the ability to handle larger-scale problems. We record the number of iterations required by the 3-level MG method to converge with the Galerkin and non-Galerkin operators, shown in Table 3. These results indicate that the convergence behavior of the MG method remains largely unchanged when the alternative sparser non-Galerkin coarse-grid operators are used as replacements.

TABLE 3
The average number of iterations required by the 3-level MG to converge with the Galerkin and non-Galerkin coarse-grid operators for solving (4.6) with different PDE parameter ξ and θ. The mesh size is 511×511. The parameters are selected so that $\xi \in \{100, 200, 300, 400\}$ is fixed and θ is randomly sampled from a uniform distribution in each interval. The iteration number is averaged over 10 different sampled θ values.

	ξ	θ		
		$(\pi/6,\ \pi/4)$	$(\pi/4, \pi/3)$	$(\pi/2,\ 7\pi/12)$
$\mathcal{A}_g$	100	92.1	102.8	126.9
$\mathcal{A}_{nn}$		89.0	93.0	135.2
$\mathcal{A}_g$	200	191.7	196.6	203.1
$\mathcal{A}_{nn}$		174.2	177.8	204.9
$\mathcal{A}_g$	300	248.0	269.7	342.3
$\mathcal{A}_{nn}$		246.5	231.4	356.2
$\mathcal{A}_g$	400	337.1	351.1	438.2
$\mathcal{A}_{nn}$		326.3	327.7	441.5

In the second set of experiments, we keep the parameter θ fixed and vary ξ following a uniform distribution within the selected intervals. A total of 12 experiments were conducted where each $\theta \in \{\pi/6, \pi/4, \pi/3, 5\pi/12\}$ is paired with ξ sampled from the intervals $\{(5, 10), (80, 100), (100, 200)\}$. The MG configurations used in these experiments remain the same as in the previous set. The training and testing processes are also similar. For each experiment, we train the model using 5 different instances evenly distributed within the selected intervals and then test it with 10 randomly sampled ξ values from the same interval. The size of the fine-level linear system in the training instances is set to be 31×31, while in each testing instance, it has a much larger size that is 511×511. The averaged numbers of iterations from all the experiments are presented in Table 4.

Table 4

The average number of iterations required by the 3-level MG to converge with the Galerkin and non-Galerkin coarse-grid operators for solving (4.6) with different PDE parameter ξ and θ. The mesh size is 511×511. The parameters are selected such that $\theta \in \{\pi/6, \pi/4, \pi/3, 5\pi/12\}$ is fixed and ξ is randomly sampled from a uniform distribution in each interval. The iteration number is averaged over 10 different sampled ξ values.

	θ	ξ		
		(100, 200)	(80, 100)	(5, 10)
$\mathcal{A}_g$	$\pi/6$	90.4	72.1	13.5
$\mathcal{A}_{nn}$		100.2	84.4	13.8
$\mathcal{A}_g$	$\pi/4$	172.5	105.2	14.1
$\mathcal{A}_{nn}$		123.1	79.0	15.9
$\mathcal{A}_g$	$\pi/3$	99.4	80.9	14.3
$\mathcal{A}_{nn}$		79.1	88.8	15.4
$\mathcal{A}_g$	$5\pi/12$	92.5	76.4	16.5
$\mathcal{A}_{nn}$		107.4	88.2	16.6

In the subsequent experiment, we specifically consider the Laplacian problem with parameters $\theta = \pi/6$ and $\xi = 0.1$ as an example to demonstrate the measurement of spectral equivalence as defined in (2.1). We examine the eigenvalues of $A_{nn}^{-1}A_g$ on meshes of varying sizes. The real parts of the eigenvalues are depicted in Figure 3. We observe that all the eigenvalues are bounded by $\alpha = 0.65, \beta = 0.9$ in Definition 2.4, and the distribution of eigenvalues remains consistent regardless of the mesh size. This observation suggests the presence of spectral equivalence between the two coarse-grid operators across meshes of different sizes.

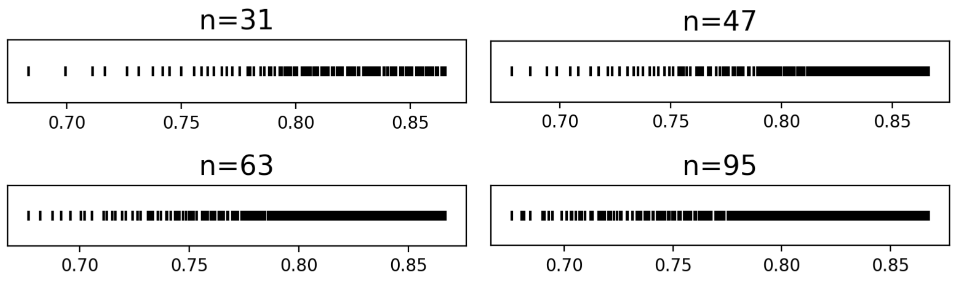


Fig. 3. *The real parts of the eigenvalues of $A_{nn}^{-1}A_g$ on meshes of different sizes ($n \times n$) for solving the rotated Laplacian problem with $\theta = \pi/6$ and $\xi = 0.1$.*

The target stencil complexity k in Algorithm 3.1 is a parameter left to be chosen by the users. It is an adjustable parameter that allows users to control the sparsity level in the trained NN-model and of the resulting coarse-grid operator. The appropriate value of k typically depends on the problem domain and the desired balance between accuracy and computational efficiency. It may be necessary to perform experiments to determine the optimal value of k for a particular application. In the final experiment, we perform this study for the rotated Laplacian problem with $\xi = 10$ and $\theta = \pi/4$. Note that the Galerkin operator has a 9-point stencil, so we vary the stencil complexity from 4 to 6 in the non-Galerkin operator and record the convergence behavior of the corresponding MG method. The results, as depicted in Figure 4, show the findings regarding the convergence behavior of the MG method with different values of k in the sparsified stencil $\mathcal{A}_{nn}$. Notably, when $k = 4$, the MG method fails to converge. However, for $k = 5$ and $k = 6$, the convergence behavior closely resembles that of the 9-point Galerkin operator. This observation suggests that a minimum number of nonzeros in the stencil of $k = 5$ appears to be required for $\mathcal{A}_{nn}$ to achieve convergence, which coincides with the operator complexity of the fine-grid operator.

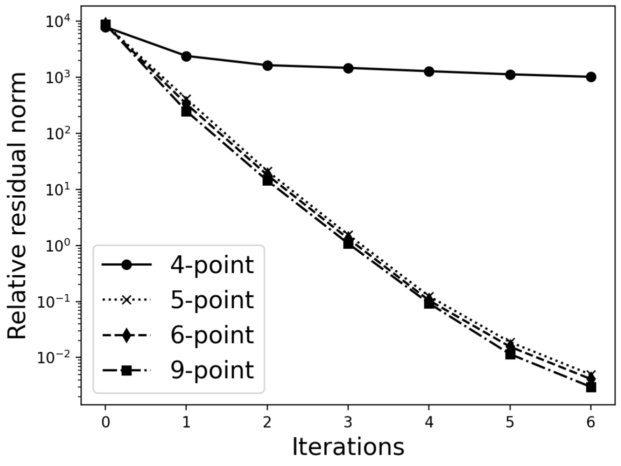


FIG. 4. *The convergence in terms of the residual norm of the two-grid MG methods using the coarse-grid operator from the NN model of stencil complexity $k = 4, 5, 6$ and the Galerkin operator for solving the rotated Laplacian problem with $\xi = 10$ and $\theta = \pi/4$.*

4.4. 2-D linear elasticity problem.

In this section, we consider the 2-D time-independent linear elasticity problem in an isotropic homogeneous medium:

$$\mu\nabla^2 u + (\mu+\lambda)\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial x \partial y}\right) + f_x = 0, \tag{4.8}$$

$$\mu\nabla^2 v + (\mu+\lambda)\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 u}{\partial x \partial y}\right) + f_y = 0, \tag{4.9}$$

where u and v are the solution in the direction of x- and y-axis respectively, f_x and f_y are forcing terms, and μ and λ are Lame coefficients that are determined by Young's modulus E and Poisson's ratio ν as

$$\mu = \frac{E}{2(1+\nu)}\ , \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}\ . \tag{4.10}$$

In our tests, we set $E = 10^{-5}$ and vary the value of ν. For the discretization, we adopt the optimal 2-D 9-point stencil in terms of local truncation errors [22] on rectangular

Cartesian grid with the mesh step sizes h and $b_y h$, respectively, along the x- and y-axes (b_y is the aspect ratio of the mesh),

$$\mathcal{A}_{uu} = \begin{bmatrix} a_{uu}^{nw} & a_{uu}^{n} & a_{uu}^{ne} \\ a_{uu}^{w} & 1 & a_{uu}^{e} \\ a_{uu}^{sw} & a_{uu}^{s} & a_{uu}^{se} \end{bmatrix}, \quad \mathcal{A}_{uv} = \begin{bmatrix} a_{uv}^{nw} & 0 & a_{uv}^{ne} \\ 0 & 0 & 0 \\ a_{uv}^{sw} & 0 & a_{uv}^{se} \end{bmatrix}, \tag{4.11}$$

where the coefficients are given by

$$a_{uu}^{n} = a_{uu}^{s} = \frac{(b_y^2-1)\lambda + 2(b_y^2-2)\mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)}\ , \tag{4.12}$$

$$a_{uu}^{w} = a_{uu}^{e} = -\frac{2\lambda b_y^2 + 4\mu b_y^2 + \lambda + \mu}{2(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)}\ , \tag{4.13}$$

$$a_{uu}^{nw} = a_{uu}^{ne} = a_{uu}^{sw} = a_{uu}^{se} = \frac{-\lambda b_y^2 - 2\mu b_y^2 + \lambda + \mu}{4(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)}\ , \tag{4.14}$$

$$a_{uv}^{nw} = a_{uv}^{se} = -a_{uv}^{ne} = -a_{uv}^{sw} = \frac{3b_y(\lambda+\mu)}{8(2\lambda b_y^2 + \lambda + 4(b_y^2+1)\mu)}\ . \tag{4.15}$$

These stencils define the 2×2 block linear system

$$\begin{bmatrix} A_{uu} & A_{uv} \\ A_{vu} & A_{vv} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f_x \\ f_y \end{bmatrix}, \tag{4.16}$$

where $A_{uu} = A_{vv}$ and $A_{vu} = A_{uv}^{\mathsf{T}}$. A node-based MG approach is used to solve (4.16), where the same red-black coarsening is used in u-u and v-v blocks and the interpolation and restriction operators have the same block form

$$R = \begin{bmatrix} R_{uu} & R_{uv} \\ R_{vu} & R_{vv} \end{bmatrix}, \quad P = \begin{bmatrix} P_{uu} & P_{uv} \\ P_{vu} & P_{vv} \end{bmatrix}, \tag{4.17}$$

which interpolate and restrict within and across the two types of variables u and v. The stencils of the operators in (4.17) are given by, respectively,

$$\mathcal{R}_{uu} = \mathcal{R}_{vv} = \frac{1}{8}\left[\begin{matrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{matrix}\right], \quad \mathcal{P}_{uu} = \mathcal{P}_{vv} = \frac{1}{4}\left]\begin{matrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{matrix}\right[, \tag{4.18}$$

$$\mathcal{R}_{uv} = \frac{1}{8}\left[\begin{matrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{matrix}\right], \quad \mathcal{P}_{uv} = \frac{1}{4}\left]\begin{matrix} & 1 & \\ -1 & 0 & -1 \\ & 1 & \end{matrix}\right[, \tag{4.19}$$

$$\mathcal{R}_{vu} = \frac{1}{8}\left[\begin{matrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{matrix}\right], \quad \mathcal{P}_{vu} = \frac{1}{4}\left]\begin{matrix} & -1 & \\ 1 & 0 & 1 \\ & -1 & \end{matrix}\right[. \tag{4.20}$$

Here we use the reversed bracket notation [34] to represent column stencils as a fan-out operation as opposed to the fan-in operation of row stencils. As stated in [8], to interpolate exactly the smoothest function that is locally constant, it requires the interpolation weights for u-u and v-v to sum to 1 and for the u-v and v-u weights to sum to 0. The Gauss-Seidel smoother is used with the MG V-cycle and the iterations are stopped when the relative residual norm is below 10^{-6}.

We train the NN model on 4 different instances with $\nu \in \{0.1, 0.2, 0.3, 0.4\}$ to reduce the complexity of the Galerkin operator by 50%. The coarse-grid Galerkin operator has the same block structure as (4.16) and only 2 distinct stencils due to the symmetry of the matrix. In the training, we combine these 2 stencils and pass them to the NNs as the input. It turns out that the NN model trained in this way yields better coarse-grid operators than learning the stencils of u-u and u-v separately.

The mesh size used in the training set is set to be 9×9. We test the model on instances with ν randomly drawn from each interval of $\{(0.1, 0.2), (0.2, 0.3), (0.3, 0.4)\}$. The size of the mesh used in the testing is 65×65. The average numbers of iterations are presented in Table 5. Similar to the results observed in the rotated Laplacian problems, the convergence behavior of the two-grid MG method is not negatively affected by the replacement with the non-Galerkin coarse-grid operator obtained from the NN model.

TABLE 5
The average number of iterations required by the 2-grid MG to converge with the Galerkin and non-Galerkin coarse-grid operators for solving (4.8) with 10 different Poisson's ratios ν randomly sampled from each interval. The mesh size is set to be 65×65.

ν	$(0.1, 0.2)$	$(0.2, 0.3)$	$(0.3, 0.4)$
$\mathcal{A}_g$	10.1	10.2	10.6
$\mathcal{A}_{nn}$	11.0	10.7	11.5

4.5. Comparison with existing non-Galerkin methods. In this section, we compare the performance of the proposed NN-based algorithm with the Sparsified Smooth Aggregation (SpSA) method proposed in [32] for solving the rotated Laplacian problem. The SpSA method is based on Smooth Aggregation (SA) AMG methods. In these methods, a tentative aggregation-based interpolation operator P_t is first constructed, followed by a few steps of smoothing of P_t that generate the final interpolation operator P, which is typically considerably denser than P_t. The SpSA algorithm aims to reduce the complexity of the Galerkin operator $P^\mathsf{T} AP$ to have the same sparsity pattern as $P_t^\mathsf{T} AP_t$. Given that we utilize the standard Ruge-Stüben MG (as opposed to SA AMG) combined with the NN-based approach, conducting a direct comparison between the two approaches becomes challenging due to the different MG hierarchies obtained. To ensure an equitable comparison, we impose a requirement that the number of nonzero entries per row in the coarse-level operator generated by SpSA should not be smaller than the operator produced by our algorithm. Consequently, any observed disparities in performance can be attributed to the specific characteristics of the selected sparsity pattern and numerical values of the coarse-grid operator, rather than the variations in the level of the sparsity. The number of iterations required by the GMRES method preconditioned by 3-level MG methods with different coarse-grid operators for solving the rotated Laplacian problem (4.6) are presented in Table 6 and Table 7, with varied PDE coefficients. For more than 70% of cases, the MG method with NN-based coarse-grid operators exhibits better performance compared to SpSA, as it requires fewer iterations to converge to the 10^{-6} stopping tolerance and achieves a convergence rate that is much closer to that using the Galerkin operator. There are a few exceptions where SpSA outperforms the NN-based method, and in some cases, it performs even better than the MG method using the Galerkin operator.

Table 6

The average number of iterations required by the GMRES method preconditioned by 3-level MG methods with different coarse-grid operators for solving (4.6) with different sets of PDE parameters. The mesh size is 511×511. The parameters are selected so that $\theta \in \{\pi/6, \pi/4, \pi/3, 5\pi/12\}$ is fixed and ξ is randomly sampled from a uniform distribution in each interval. The iteration number is averaged over 10 different sampled θ's. $\mathcal{A}_g$ denotes the Galerkin coarse-grid operator, $\mathcal{A}_{nn}$ is the coarse-grid operator obtained from Algorithm 3.3, and SpSA refers to the coarse-grid operator from the Sparsified Smooth Aggregation (SpSA) algorithm [32]. The numbers in the brackets are the operator complexities.

	ξ	θ		
		$(\pi/6,\ \pi/4)$	$(\pi/4,\pi/3)$	$(\pi/2,\ 7\pi/12)$
$\mathcal{A}_g$	100	11.3 (1.31)	11.1 (1.31)	11.5 (1.31)
$\mathcal{A}_{nn}$		16.5 (1.17)	16.9 (1.17)	19.9 (1.17)
SA		35.5 (1.65)	35.0 (1.70)	10.2 (2.14)
SpSA		41.7 (1.29)	38.7 (1.21)	13.2 (1.42)
$\mathcal{A}_g$	200	15.9 (1.31)	15.3 (1.31)	14.5 (1.31)
$\mathcal{A}_{nn}$		20.5 (1.17)	19.6 (1.17)	29.8 (1.17)
SA		44.1 (1.65)	44.8 (1.66)	9.6 (2.31)
SpSA		51.8 (1.19)	47.9 (1.19)	14.9 (1.42)
$\mathcal{A}_g$	300	18.1 (1.31)	21.5 (1.31)	17.9 (1.31)
$\mathcal{A}_{nn}$		25.4 (1.17)	33.1 (1.17)	25.6 (1.17)
SA		45.7 (1.72)	51.3 (1.63)	11.2 (2.08)
SpSA		54.7 (1.23)	53.5 (1.17)	16.7 (1.43)
$\mathcal{A}_g$	400	21.1 (1.31)	20.2 (1.31)	19.9 (1.31)
$\mathcal{A}_{nn}$		27.2 (1.17)	30.9 (1.17)	26.2 (1.17)
SA		52.7 (1.63)	53.5 (1.66)	11.0 (2.11)
SpSA		61.3 (1.19)	57.6 (1.19)	18.2 (1.42)

5. Conclusion.

In this work, we propose an ML-based approach for computing non-Galerkin coarse-grid operators to address the issue of increasing operator complexity in MG methods by sparsifying the Galerkin operator in different MG levels. The algorithm consists of two main steps: choosing the sparsity pattern of the stencil and computing the numerical values. We employ NNs in both steps and combine their results to construct a non-Galerkin coarse-grid operator with the desired lower complexity. The NN training algorithm is guided by the MG convergence theory, ensuring the spectral equivalence of coarse-grid operators with respect to the Galerkin operator. We showed that spectrally equivalent sparser stencils can be learned by advanced ML techniques that exploit multi-head attention.

The NN model is trained on parametric PDE problems that cover a wide range of parameters. The training dataset consists of small-size problems, while the testing problems are significantly larger. Empirical studies conducted on rotated Laplacian and linear elasticity problems provide evidence that the proposed ML method can construct non-Galerkin operators with reduced complexity while maintaining the overall convergence behavior of MG. A key feature of our method is its ability to generalize to problems of larger sizes and with different PDE parameters that were not encountered in the training for in-distribution test sets. This means that the algorithm can effectively handle a wide range of problem settings, expanding its practical applicability. By generalizing to new problem instances, the algorithm amortizes the training cost and reduces the need for retraining for every specific problem scenario. *It is impor-*

TABLE 7
The average number of iterations required by the GMRES method preconditioned by 3-level MG methods with different coarse-grid operators for solving (4.6) with different sets of PDE parameters. The mesh size is 511×511. The parameters are selected so that $\theta \in \{\pi/6, \pi/4, \pi/3, 5\pi/12\}$ is fixed and ξ is randomly sampled from a uniform distribution in each interval. The iteration number is averaged over 10 different sampled ξ's. $\mathcal{A}_g$ denotes the Galerkin coarse-grid operator, $\mathcal{A}_{nn}$ is the coarse-grid operator obtained from Algorithm 3.3, and SpSA refers to the coarse-grid operator from the Sparsified Smooth Aggregation (SpSA) algorithm [32]. The numbers in the brackets are the operator complexities.

	θ	ξ		
		$(100, 200)$	$(80, 100)$	$(5, 10)$
$\mathcal{A}_g$	$\pi/6$	11.6 (1.31)	10.4 (1.31)	4.4 (1.31)
$\mathcal{A}_{nn}$		19.9 (1.17)	16.4 (1.17)	7.1 (1.17)
SA		24.8 (1.98)	22.4 (1.98)	11.5 (1.98)
SpSA		32.2 (1.38)	29.0 (1.38)	13.7 (1.38)
$\mathcal{A}_g$	$\pi/4$	14.2 (1.31)	11.3 (1.31)	4.6 (1.31)
$\mathcal{A}_{nn}$		18.2 (1.17)	15.5 (1.17)	10.0 (1.17)
SA		39.2 (1.63)	33.1 (1.63)	13.3 (1.62)
SpSA		42.8 (1.17)	35.8 (1.17)	14.2 (1.17)
$\mathcal{A}_g$	$\pi/3$	11.2 (1.31)	10.2 (1.31)	4.7 (1.31)
$\mathcal{A}_{nn}$		18.8 (1.17)	16.4 (1.17)	7.1 (1.17)
SA		26.5 (1.98)	24.2 (1.98)	12.1 (1.98)
SpSA		34.1 (1.38)	30.9 (1.38)	14.5 (1.38)
$\mathcal{A}_g$	$5\pi/12$	11.2 (1.31)	10.1 (1.31)	4.8 (1.31)
$\mathcal{A}_{nn}$		28.8 (1.17)	19.1 (1.17)	9.1 (1.17)
SA		13.9 (1.93)	12.8 (1.93)	10.0 (1.91)
SpSA		18.5 (1.38)	17.1 (1.38)	10.8 (1.37)

tant to note that the true generalizability capability (out-of distribution test sets) for deep learning approaches requires the development of large-scale foundation models, a large-scale pretrained-model that can be used to conduct unseen tasks. This work is an initial step towards that goal.

In the future, we plan to extend this work to sparsify unstructured coarse grid operators by exploiting the Graph Convolution Networks (GCNs). We also plan to explore the Equivariant Neural Networks [10] to enforce the symmetry in the sparsified coarse-grid operators if the fine level operator is symmetric. In addition, we plan to investigate the real-world applications including saddle point system[18], efficient tensor algebra [16, 20, 14], modern generative models [9, 19, 17], multi-time series analysis techniques [15, 30] to solve time-dependent PDEs.

REFERENCES

[1] O. Axelsson, *On multigrid methods of the two-level type*, in Multigrid methods, Springer, 1982, pp. 352–367.
[2] A. H. Baker, T. Gamblin, M. Schulz, and U. M. Yang, *Challenges of scaling algebraic multigrid across modern multicore architectures*, in 2011 IEEE International Parallel & Distributed Processing Symposium, 2011, pp. 275–286, https://doi.org/10.1109/IPDPS.2011.35.
[3] N. Bell, L. N. Olson, and J. Schroder, *Pyamg: algebraic multigrid solvers in python*, Journal of Open Source Software, 7 (2022), p. 4142.
[4] A. Bienz, R. D. Falgout, W. Gropp, L. N. Olson, and J. B. Schroder, *Reducing parallel*

communication in algebraic multigrid through sparsification, SIAM Journal on Scientific Computing, 38 (2016), pp. S332–S357.

[5] A. Bienz, W. D. Gropp, and L. N. Olson, *Reducing communication in algebraic multigrid with multi-step node aware communication*, The International Journal of High Performance Computing Applications, 34 (2020), pp. 547–561.

[6] M. Bolten and A. Frommer, *Structured grid AMG with stencil-collapsing for d-level circulant matrices.* Bergische Universität Wuppertal, Wuppertal, Germany. Preprint BUW-SC 07/4, 2007.

[7] A. Brandt, J. Brannick, K. Kahl, and I. Livshits, *Bootstrap AMG*, SIAM Journal on Scientific Computing, 33 (2011), pp. 612–632.

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

[9] D. Cai, Y. Ji, H. He, Q. Ye, and Y. Xi, *Autm flow: atomic unrestricted time machine for monotonic normalizing flows*, in Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, J. Cussens and K. Zhang, eds., vol. 180 of Proceedings of Machine Learning Research, PMLR, 01–05 Aug 2022, pp. 266–274.

[10] T. Cohen and M. Welling, *Group equivariant convolutional networks*, in Proceedings of The 33rd International Conference on Machine Learning, M. F. Balcan and K. Q. Weinberger, eds., vol. 48 of Proceedings of Machine Learning Research, PMLR, 2016, pp. 2990–2999.

[11] R. D. Falgout and J. B. Schroder, *Non-galerkin coarse grids for algebraic multigrid*, SIAM Journal on Scientific Computing, 36 (2014), pp. C309–C334.

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

[13] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, *Hippo: recurrent memory with optimal polynomial projections*, in Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS'20, Red Hook, NY, USA, 2020, Curran Associates Inc.

[14] H. He, J. Henderson, and J. C. Ho, *Distributed tensor decomposition for large scale health analytics*, in The World Wide Web Conference, WWW '19, New York, NY, USA, 2019, Association for Computing Machinery, p. 659–669.

[15] H. He, O. Queen, T. Koker, C. Cuevas, T. Tsiligkaridis, and M. Zitnik, *Domain adaptation for time series under feature and label shifts*, in International Conference on Machine Learning, 2023.

[16] H. He, Y. Xi, and J. C. Ho, *Fast and accurate tensor decomposition without a high performance computing machine*, in 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 163–170.

[17] H. He, S. Zhao, Y. Xi, and J. Ho, *AGE: Enhancing the convergence on GANs using alternating extra-gradient with gradient extrapolation*, in NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications, 2021.

[18] H. He, S. Zhao, Y. Xi, J. Ho, and Y. Saad, *GDA-AM: ON THE EFFECTIVENESS OF SOLVING MIN-IMAX OPTIMIZATION VIA ANDERSON MIXING*, in International Conference on Learning Representations, 2022.

[19] H. He, S. Zhao, Y. Xi, and J. C. Ho, *Meddiff: Generating electronic health records using accelerated denoising diffusion model*, 2023, https://arxiv.org/abs/2302.04355.

[20] J. Henderson, H. He, B. A. Malin, J. C. Denny, A. N. Kho, J. Ghosh, and J. Ho, *Phenotyping through semi-supervised tensor factorization (psst)*, AMIA ... Annual Symposium proceedings. AMIA Symposium, 2018 (2018), pp. 564–573.

[21] R. Huang, R. Li, and Y. Xi, *Learning optimal multigrid smoothers via neural networks*, SIAM Journal on Scientific Computing, 45 (2023), pp. S199–S225.

[22] A. Idesman and B. Dey, *Compact high-order stencils with optimal accuracy for numerical solutions of 2-d time-independent elasticity equations*, Computer Methods in Applied Mechanics and Engineering, 360 (2020), p. 112699.

[23] T. Jonsthovel, A. Lukyanov, E. Wobbes, and C. Vuik, *Monotone non-galerkin algebraic multigrid method applied to reservoir simulations*, (2016), cp-494-00156, https://doi.org/https://doi.org/10.3997/2214-4609.201601896, https://www.earthdoc.org/content/papers/10.3997/2214-4609.201601896.

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

[25] D. Kuznichov, *Learning relaxation for multigrid*, arXiv preprint arXiv:2207.11255, (2022).

[26] R. P. Li, B. Sjögreen, and U. M. Yang, *A new class of amg interpolation methods based on*

matrix-matrix multiplications, SIAM Journal on Scientific Computing, 43 (2021), pp. S540–S564, https://doi.org/10.1137/20M134931X, https://doi.org/10.1137/20M134931X, https://arxiv.org/abs/https://doi.org/10.1137/20M134931X.

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

[28] W. B. Mitchell, R. Strzodka, and R. D. Falgout, *Parallel performance of algebraic multigrid domain decomposition*, Numerical Linear Algebra with Applications, 28 (2021), p. e2342, https://doi.org/https://doi.org/10.1002/nla.2342, https://onlinelibrary.wiley.com/doi/abs/10.1002/nla.2342, https://arxiv.org/abs/https://onlinelibrary.wiley.com/doi/pdf/10.1002/nla.2342.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep learning library*, Advances in neural information processing systems, 32 (2019).

[30] O. Queen, T. Hartvigsen, T. Koker, H. He, T. Tsiligkaridis, and M. Zitnik, *Encoding time-series explanations through self-supervised model behavior consistency*, 2023, https://arxiv.org/abs/2306.02109.

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

[32] E. Treister and I. Yavneh, *Non-galerkin multigrid based on sparsified smoothed aggregation*, SIAM Journal on Scientific Computing, 37 (2015), pp. A30–A54.

[33] E. TREISTER, R. Zemach, and I. YAVNEH, *Algebraic collocation coarse approximation (acca) multigrid*, in 12th Copper Mountain Conference on Iterative Methods, 2012.

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

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017.

[36] R. Wienands and I. Yavneh, *Collocation coarse approximation in multigrid*, SIAM Journal on Scientific Computing, 31 (2009), pp. 3643–3660.