Neural network preconditioning: a case study for the solution of the parametric Helmholtz equation

Luc Giraud, Carola Kruse, Paul Mycek, Maksym Shpakovych, Yanfei Xiang

Neural network preconditioning: a case study for the solution of the parametric Helmholtz equation

Luc Giraud, Carola Kruse, Paul Mycek, Maksym Shpakovych, Yanfei Xiang

RESEARCH REPORT
N° 9593
July 2025
Project-Team Concace

ISSN 0249-6399 ISRN INRIA/RR--9593--FR+ENG

Neural network preconditioning: a case study for the solution of the parametric Helmholtz equation

Luc Giraud*, Carola Kruse†, Paul Mycek†,

Maksym Shpakovych‡, Yanfei Xiang§

Project-Team Concace

Research Report n° 9593 — July 2025 — 19 pages

Abstract: This work presents a hybrid numerical approach for solving linear systems arising from the discretization of the two-dimensional parametric Helmholtz equation. A convolutional neural network based on the U-Net architecture is trained in an unsupervised manner to approximate the inverse of the discretized Helmholtz operator, using a loss function involving the residual norm of the linear system. The trained network is used as a nonlinear preconditioner within the Flexible GMRES (FGMRES) algorithm. Numerical experiments show that while the neural network is not accurate enough to act as a standalone solver, it significantly improves the convergence of FGMRES when employed as a preconditioner. The neural preconditioner demonstrates robust performance and generalization capabilities with respect to variations in the velocity field and the domain size. Comparisons with classical algebraic preconditioners based on sparsified LU factorizations indicate superior efficiency of the neural approach under equivalent conditions. We believe that the proposed method is not tied to a specific neural architecture and can be extended to other parametric PDEs.

Key-words: Preconditioning, neural networks, subspace methods, parametric Helmholtz equation

* Inria, Concace joint project Airbus CR & T, Cerfacs and Inria, France
† Cerfacs, Concace joint project Airbus CR & T, Cerfacs and Inria, France
‡ Inria, Concace joint project Airbus CR & T, Cerfacs and Inria. Current affiliation: Opticalp, France
§ Inria, Concace joint project Airbus CR & T, Cerfacs and Inria. Current affiliation: University of Strasbourg, France

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Préconditonnement par réseaux de neuronnes: un cas d'étude pour la résolution de l'équation de Helmholtz paramétrique

Résumé : Ce travail présente une approche numérique hybride pour la résolution de systèmes linéaires découlant de la discrétisation de l'équation paramétrique bidimensionnelle de Helmholtz. Un réseau neuronal convolutionnel basé sur l'architecture U-Net est entraîné de manière non supervisée pour approximer l'inverse de l'opérateur de Helmholtz discrétisé, en utilisant une fonction de perte impliquant la norme résiduelle du système linéaire. Le réseau entraîné est utilisé comme préconditionneur non linéaire dans l'algorithme Flexible GMRES (FGMRES). Les expériences numériques montrent que si le réseau neuronal n'est pas suffisamment précis pour servir de solveur autonome, il améliore considérablement la convergence de l'algorithme FGMRES lorsqu'il est utilisé comme préconditionneur. Le préconditionneur neuronal démontre des performances robustes et des capacités de généralisation en ce qui concerne les variations du champ de vitesse et de la taille du domaine. Les comparaisons avec les préconditionneurs algébriques classiques basés sur des factorisations LU sparifiées indiquent une efficacité supérieure de l'approche neuronale dans des conditions équivalentes. Nous pensons que la méthode proposée n'est pas liée à une architecture neuronale spécifique et qu'elle peut être étendue à d'autres EDP paramétriques.

Mots-clés : Préconditionnement, réseaux de neuronnes, méthodes de sous-espaces, équation d'Helmholtz paramétique

Contents

1 Introduction

In this work, we focus on solving the Helmholtz equation that is used for modeling the propagation of acoustic waves in heterogeneous media. Specifically, we are interested in the solution of the linear systems resulting from the discretization of this equation in a 2D domain using subspace iterative methods. For these types of linear systems, no generic, efficient preconditioner currently exists. In this respect we investigate the possibility of using machine learning to design such a preconditioner. In more classical approaches such as physics-informed neural networks (PINNs) [7], the trained neural network aims at directly finding the solutions. The main advantage of our hybrid approach is that the subspace method ensures that the quality of the computed solution far surpasses the quality of the solution predicted by a neural network alone. Given that neural networks have a nonlinear response to their inputs and are typically trained in 32-bit arithmetic, we consider the FGMRES [16] subspace method to enable a 64-bit computation suited for ill-conditioned problems. To our knowledge, this idea was first proposed in [23, Chapter 5] and further exploited in [18]. We show that an unsupervised-trained neural network can construct an effective preconditioner with attractive generalization properties. Our conclusions are not restricted to this network choice and our claim is that one may use their preferred network [18] and apply similar ideas to other linear systems.

In recent years, the use of machine learning techniques, particularly neural networks (NNs), has been considered in various ways for developing preconditioners for Krylov subspace methods. One contribution in this field, close to ours, comes from Oseledets and colleagues [15], who introduced a discretization-invariant neural operator used as a nonlinear preconditioner within the flexible conjugate gradient (FCG) method, enabling efficient solutions of elliptic PDEs across multiple linear systems. In their work, Trifonov et al. [20] proposed a GNN-based preconditioner that learns optimal sparsity patterns and improves the convergence of iterative solvers for complex physical models. In a related study, they further refined this concept for conjugate gradient solvers, demonstrating significant reduction in iteration counts and computational cost [21]. Furthermore, Kopaničáková and Karniadakis [9] proposed a hybrid preconditioning strategy that leverages DeepONet models to capture low-frequency error components while classical iterative methods handle high-frequency ones. Their framework demonstrated robust performance and convergence acceleration across a wide range of parameterized linear systems, offering a novel perspective distinct from purely GNN-based or multigrid techniques. This approach complements other neural preconditioning methods and reveals the potential of operator learning frameworks in the preconditioning landscape. Yusuf et al. [24], who designed ILU preconditioners via GNNs for advection-dominated PDEs; and Li et al. [10], who introduced neural operators to accelerate PDE solvers directly. Azulay and Treister [1] combined multigrid methods with deep learning for Helmholtz problems, while Moore [12] explored residual neural networks for constructing adaptive preconditioners. Dimola et al. [4] and Nieto Juscafresa [6] also proposed data-driven techniques tailored for GMRES solvers.

The field of scientific machine learning is rapidly evolving, and the few references provided herein are not intended to offer an exhaustive overview of the state of the art, as such an undertaking would far exceed the scope of the present work.

2 Background

2.1 The Helmholtz equation

We consider the two-dimensional Helmholtz equation, subject to the so-called Sommerfeld radiation condition,

$$\begin{cases} \nabla^2 u + k^2 u = f \quad \text{in } \mathbb{R}^2, \\ \lim_{\|\mathbf{x}\|_2 \to \infty} \|\mathbf{x}\|_2^{1/2} \left(\dfrac{\partial u}{\partial \|\mathbf{x}\|_2}(\mathbf{x}) - \jmath k(\mathbf{x}) u(\mathbf{x}) \right) = 0, \end{cases} \tag{1}$$

where $\jmath \in \mathbb{C}$ denotes the imaginary unit such that $\jmath^2 = -1$, and $u\colon \mathbb{R}^2 \to \mathbb{C}$, $f\colon \mathbb{R}^2 \to \mathbb{C}$, $k\colon \mathbb{R}^2 \to \mathbb{R}^+$ are scalar fields, referred to as the solution field, the source field, and the wavenumber field, respectively. The latter is defined by $k\colon \mathbf{x} \mapsto \omega/c(\mathbf{x})$, where $\omega \in \mathbb{R}^+$ is the angular frequency of the source, and $c\colon \mathbb{R}^2 \to \mathbb{R}^+$ is the velocity field. In what follows, we restrict ourselves to the case $\omega = 1$. Numerically, the unbounded domain $\mathbb{R}^2$ is truncated to a computational, square domain $\Omega := [-L, L]^2 \subset \mathbb{R}^2$ and the derivatives in eq. (1) are discretized using Fourier differentiation [18, 19, 23] with N points in each direction. Consequently, there is a total of $n := N^2$ degrees of freedom. The boundary conditions are (approximately) enforced using perfectly matched layers (PMLs) [2] in the outer region $\Omega_{\text{pml}} := \Omega \setminus \Omega_{\text{inner}}$, where $\Omega_{\text{inner}} := [-L_{\text{inner}}, L_{\text{inner}}]^2$, with $L_{\text{inner}} := L - \ell$ and $\ell < L$, as illustrated in Figure 1.

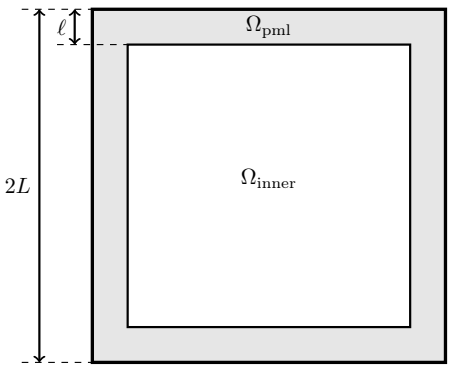


Figure 1: Illustration of the PML.

The discretization of the Helmholtz problem (1) yields a system of linear equations of the form $A(c)x = b$, involving the discrete velocity field $c \in \mathbb{R}^n$ and the associated discretized operator (matrix) $A(c) \in \mathbb{C}^{n \times n}$, as well as the discretized source term $b \in \mathbb{C}^n$ and solution $x \in \mathbb{C}^n$.

2.2 Subspace solver with nonlinear preconditioner

The solution of large linear systems of equations is often approached by iterative methods. Among the advantages of these techniques are their moderate memory consumption and the ability to stop the iterations when the quality of the solution is similar to the possible uncertainty in the matrix entries or right-hand sides, which may arise from discretization errors in the case of a PDE solution or from uncertainty in some input data. Backward error analysis [13, 22] provides powerful techniques for designing meaningful stopping criteria. In recent decades, subspace methods, such as

Krylov subspace methods, have attracted much attention and have become standard techniques for solving linear systems. For unsymmetric problems, the best known method is certainly GMRES [17], which computes the minimum residual norm approximation in a Krylov subspace. This technique relies on the so-called Arnoldi process, which incrementally builds an orthonormal basis of the nested subspaces. At step k, the Arnoldi algorithm computes a set of orthonormal vectors $V_k = [v_1, \ldots, v_k]$ that satisfy the so-called Arnoldi relation, which is written in matrix form:

$$AV_k = V_{k+1}\bar{H}_k \quad \text{with } V_{k+1}^H V_{k+1} = I_{k+1}$$

where $\bar{H}_k \in \mathbb{C}^{(k+1)\times k}$ is upper Hessenberg. The minimum residual norm iterate $x_k \in \text{span}\{v_1, \ldots, v_k\}$, that is $x_k = V_k y_k$ with $y_k = \arg\min_{y\in\mathbb{C}^k}(\|\bar{H}_k y - \beta e_1\|_2)$ where $\beta = \|b\|_2$. In practice, a preconditioner M is used to speed up the convergence of GMRES, where M is expected to approximate A^{-1} somehow. For GMRES it is recommended to use the preconditioner on the right, i.e., GMRES solves $AMt = b$, with $Mt = x$. Thus, at each iteration, GMRES still minimizes the residual norm of the residual associated with the original linear system. The Arnoldi relation is

$$AMV_k = V_{k+1}\bar{H}_k \quad \text{with } V_{k+1}^H V_{k+1} = I_{k+1}.$$

In [16], Saad introduced the idea of a flexible preconditioner, which allows to have a different preconditioning matrix M_i at each iteration, so that the generalized Arnoldi relation becomes

$$AZ_k = V_{k+1}\bar{H}_k \quad \text{with } V_{k+1}^H V_{k+1} = I_{k+1}, \tag{2}$$

where $Z_k = [z_1, \ldots, z_k]$ with $z_i = M_i v_i$ $(i = 1, \ldots, k)$. This idea can easily be extended to the situation where the preconditioner is no longer linear, i.e., $z_i = \mathcal{M}_i(v_i)$, where each $\mathcal{M}_i$ may be a nonlinear operator. In the latter case, Equation (2) still holds and uniquely defines each iterate as long as Z_k remains full rank. In this paper we use this equality with $\mathcal{M}_i() = \mathcal{N}_\theta()$, which is the trained neural network (see section 3). The nonlinearity has two origins; first, the activation functions in the neurons are nonlinear, and second, the preconditioner is computed in 32-bit arithmetic while the rest of the computation is done in 64-bit arithmetic. This truncation due to casting 64 bits to 32 bits is also nonlinear. A sketch of the NN-preconditioned FGMRES algorithm is provided in Algorithm 1.

3 Neural network components

3.1 Network architecture

As stated in Section 2.2, we propose to use a neural network as a nonlinear preconditioner. To do so, we design a neural network based on the U-Net architecture [14], which takes the discretized source and velocity fields as inputs. To make them suitable to the convolutional nature of the U-Net, these 2D fields are recast as image-like tensors. Specifically, recalling that $n = N^2$, the complex-valued source term $b \in \mathbb{C}^n$ may be recast as a $2 \times N \times N$ tensor composed of two $N \times N$ channels representing the real and imaginary parts of the field. Similarly, the real-valued velocity field $c \in \mathbb{R}^n$ may be recast as a $1 \times N \times N$ tensor. These two inputs are combined in a single $3 \times N \times N$ input tensor. The output of the network is a $2 \times N \times N$ tensor, representing a 2D, complex-valued field. As we shall see in Section 3.2, the network will be trained to provide approximations (predictions) $\hat{x}_\theta$ of discrete solution fields x of the linear system $A(c)x = b$ corresponding to a discretized version of the Helmholtz problem (1).

Algorithm 1: Sketch of the NN-preconditioned FGMRES algorithm.

$r_0 = b - Ax_0$, $\beta = \|r_0\|_2$, $v_1 = r_0/\beta$
for $j = 1, \ldots, m$ **do**
  $z_j = \mathcal{N}_\theta(v_j)$
  $w = Az_j$
  for $i = 1, \ldots, j$ **do**
    $h_{i,j} = \langle w, v_i \rangle$
    $w \leftarrow w - h_{i,j} v_i$
  end for
  $h_{j+1,j} = \|w\|_2$
  $v_{j+1} = w/h_{j+1,j}$
end for
$\bar{H}_m = [h_{i,j}]_{1 \le j \le m, 1 \le i \le j+1}$
$Z_m = [z_1, \ldots, z_m]$
$y_m = \arg\min_{y \in \mathbb{C}^m} \|\bar{H}_m y - \beta e_1\|_2$
$x_m = x_0 + Z_m y_m$

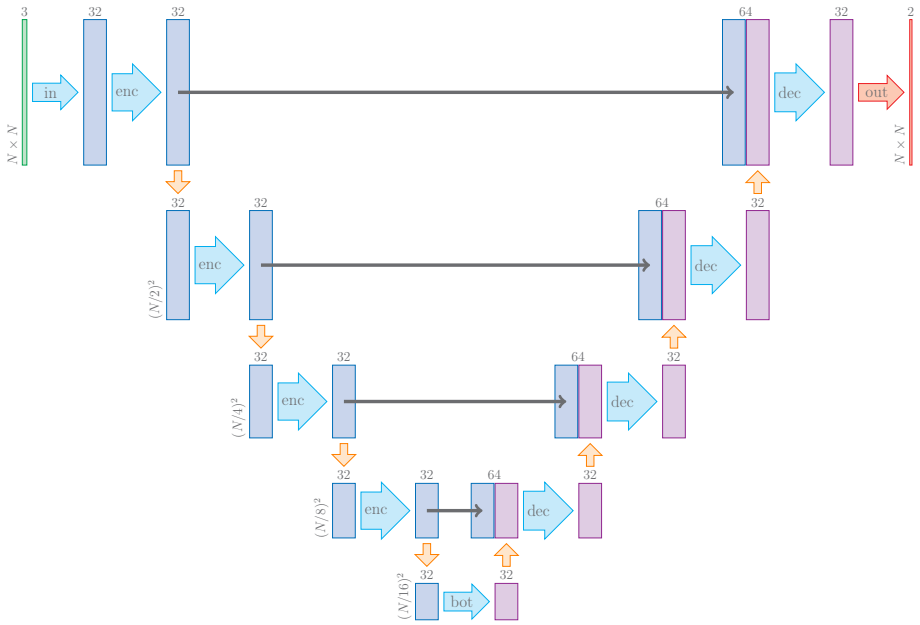


Figure 2: Overview of our U-Net architecture, whose main building blocks are described in Tables 1 and 2.

The specific U-Net architecture prescribed in this work is depicted in Figure 2. The modules represented in Figure 2 are described in Table 1. They consist of sequences of parameterized layers detailed in Table 2. The downscaling convolutions, represented by the downward orange arrows, are designed such that the size of the "image" is halved. Similarly, the upscaling convolutions, repre-

sented by the upward orange arrows, are designed such that the size of the "image" is multiplied by 2. As reported in Table 1, our U-Net architecture comprises a total of 831,650 trainable parameters, which is independent of n owing to its convolutional nature.

Module	Sequence of layers	#params/module	#modules	total #params
in	$\mathrm{Conv}_{3,32,3,1,1}, \mathrm{BN}_{32}, \mathrm{PreLU}_{32}$	992	1	992
bot	$\mathrm{Conv}_{32,32,3,1,1}, \mathrm{BN}_{32}, \mathrm{PreLU}_{32}$	9,344	1	9,344
enc	$\mathrm{Conv}_{32,32,3,1,1}, \mathrm{BN}_{32}, \mathrm{PreLU}_{32}, \mathrm{Conv}_{32,32,3,1,1}$	18,592	4	74,368
dec	$\mathrm{Conv}_{64,64,3,1,1}, \mathrm{BN}_{64}, \mathrm{PreLU}_{64}, \mathrm{Conv}_{64,32,3,1,1}$	55,584	4	222,336
out	$\mathrm{Conv}_{32,2,1,0,1}$	66	1	66
	$\mathrm{Conv}_{32,32,8,3,2}$	65,568	4	262,272
	$\mathrm{ConvT}_{32,32,8,3,2}$	65,568	4	262,272
	skip connection	0	4	0
			Total	**831,650**

Table 1: Description of the modules used in the U-Net architecture depicted in Figure 2. Each module is defined as a sequence of layers detailed in Table 2, and the associated number of parameters is deduced from the values provided therein.

The abstract, functional representation of the neural network with parameters θ is thus a nonlinear operator,

$$\begin{aligned} \mathcal{N}_\theta &: \mathbb{C}^n \times \mathbb{R}^n \to \mathbb{C}^n, \\ &(b, c) \mapsto \mathcal{N}_\theta(b, c) = \mathcal{N}_\theta(c)(b), \end{aligned} \tag{3}$$

where the conversions to and from image-like tensors are implicit.

3.2 Training of the network

The training process aims at finding the parameters $\theta^* \in \Theta$ of the neural network that minimize a certain cost functional $J : \Theta \to \mathbb{R}$, where Θ denotes the parameter space, i.e.,

$$\theta^* \in \arg\min_{\theta \in \Theta} J(\theta). \tag{4}$$

For our specific problem, the cost function takes the form

$$J(\theta) = \mathbb{E}_{b,c}[\mathcal{L}_\theta(b, c)], \tag{5}$$

Layer	**PyTorch name**	**Arguments**	**#params**	**Output size**
$\mathrm{Conv}_{c_{\mathrm{in}},c_{\mathrm{out}},k,p,s}$	`Conv2d`	`in_channels` $= c_{\mathrm{in}}$ `out_channels` $= c_{\mathrm{out}}$ `kernel_size` $= k$ `padding` $= p$ `stride` $= s$	$(k^2 c_{\mathrm{in}} + 1)c_{\mathrm{out}}$	$o = \lfloor (i + 2p - k)/s + 1 \rfloor$
$\mathrm{ConvT}_{c_{\mathrm{in}},c_{\mathrm{out}},k,p,s}$	`ConvTranspose2d`	`in_channels` $= c_{\mathrm{in}}$ `out_channels` $= c_{\mathrm{out}}$ `kernel_size` $= k$ `padding` $= p$ `stride` $= s$	$(k^2 c_{\mathrm{in}} + 1)c_{\mathrm{out}}$	$o = s(i-1) - 2p + k$
BN_c	`BatchNorm2d`	`num_features` $= c$	$2c$	$o = i$
PreLU_c	`PReLU`	`num_parameters` $= c$	c	$o = i$

Table 2: Description and PyTorch name of the NN layers used in the U-Net architecture described in Figure 2 and Table 1, along with their arguments, and the induced number of parameters and output size.

where b and c are the inputs of the network, $\mathcal{L}_\theta$ is a loss function and $\mathbb{E}_{b,c}$ denotes the expectation operator with respect to b and c. In this work, we consider the loss function defined by

$$\mathcal{L}_\theta(b,c) = \frac{\|b - A(c)\mathcal{N}_\theta(b,c)\|_2^2}{\|b\|_2^2} = \frac{\|b - A(c)\hat{x}_\theta\|_2^2}{\|b\|_2^2} = \frac{\|b - \hat{b}_\theta\|_2^2}{\|b\|_2^2}, \tag{6}$$

where $\hat{x}_\theta := \mathcal{N}_\theta(b,c) = \mathcal{N}_\theta(c)(b)$ denotes the prediction of the neural network, and $\hat{b}_\theta = A(c)\hat{x}_\theta$. The neural network is thus trained to be a good approximation of A^{-1}, i.e., roughly speaking, so that $\mathcal{N}_\theta(c) \approx A^{-1}(c)$ for any c. The training process can here be interpreted as unsupervised, in the sense that the prediction $\hat{x}_\theta$ are not (directly) compared to reference values x^* in the loss function. We remark that the residual of the linear system for a given prediction $\hat{x}_\theta = \mathcal{N}_\theta(c)(b)$, namely $b - A(c)\hat{x}_\theta$, which is at the core of the loss definition, corresponds to an algebraic (discretized) version of the residual of the PDE, including the boundary conditions. As such, the loss may be interpreted as being physics-informed. In terms of numerical linear algebra, we further note that for any given $c \in \mathbb{R}^n$, $\mathcal{L}_\theta(b,c) = \eta_b(\hat{x}_\theta)^2$, where $\eta_b(\tilde{x})$ denotes the backward error with respect to b of $\tilde{x}$ as an approximate solution to the linear system $A(c)x = b$ [5, 13]. An alternative interpretation that can be drawn from the last representation of the loss function (6) is that the neural network is trained so that $A(c)\mathcal{N}_\theta(c)$ is a good approximation of the identity operator, similar to an autoencoder [3]. The neural network $\mathcal{N}_\theta$ can then be interpreted as an encoder, trained such that it is decoded by A. The training process is summarized in Figure 3.

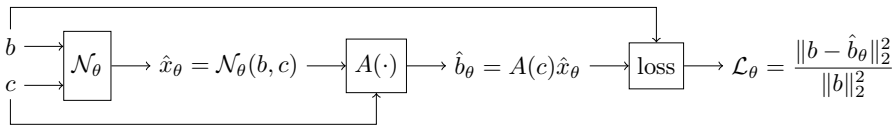


Figure 3: Schematic representation of the training framework.

Parameter	**Value**
Optimizer	Adam [8]
Mini-batch size B	32
Number of epochs	500
Size of data set K	16,000
Training/validation data set split	96%/4%
Learning rate (LR)	10^{-3}
Gradient clipping (norm)	1
LR scheduler (LRS)	`ReduceLROnPlateau`
LRS factor	0.5
LRS patience	10
LRS min LR	10^{-5}

Table 3: Parameters of the training process.

Numerically, the solution of the optimization problem (4) is approximated using the Adam algorithm [8] over 500 epochs, from a data set of $K = 16{,}000$ synthetic pairs of inputs $\mathcal{X} = \{(b^{(k)}, c^{(k)})\}_{k=1}^{K}$, which are cheaply generated from the (prescribed) probability distribution of (b, c) (see below). The data set is split into a training data set $\mathcal{X}_{\text{train}} \subset \mathcal{X}$ of size 15,360 (96% of the data set), on which the training is performed, and a validation data set $\mathcal{X}_{\text{val}} := \mathcal{X} \setminus \mathcal{X}_{\text{train}}$ of size 640 (4% of the data set), on which the model is validated at the end of each epoch. The parameters of the optimizer are summarized in Table 3. At each iteration (gradient step) of the Adam algorithm, the cost function is approximated from a mini-batch $\mathcal{B} \subset \mathcal{X}_{\text{train}}$ of size $B = 32$ as

$$J \approx \hat{J}_{\mathcal{B}} := \frac{1}{B} \sum_{(b,c)\in\mathcal{B}} \mathcal{L}_\theta(b, c). \tag{7}$$

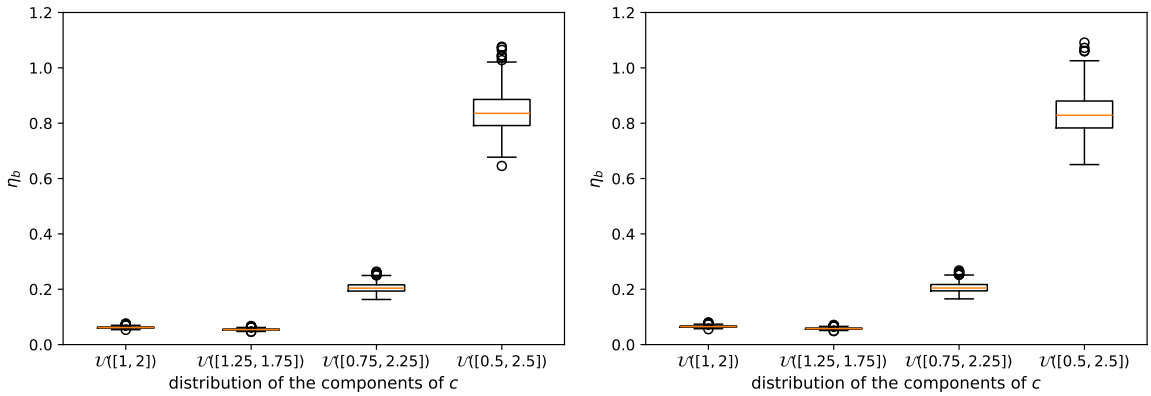
For the experiments presented subsequently in this paper, the network was trained using training and validation data generated with $N = 64$, $L = 20$ and $\ell = 4$. Furthermore, the distribution of (b, c) is defined component-wise and independently for b and c such that the entries of b are independent and normally distributed with zero mean and unit variance (we denote $b \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$), the ones of c are independent and uniformly distributed between 1 and 2 (i.e., $c \overset{\text{iid}}{\sim} \mathcal{U}([1, 2])$). We note that the latter implies that the wavelength $\lambda := 2\pi/k = 2\pi c/\omega$ lies within the range $[2\pi, 4\pi]$, so that there are between 10 and 20 discretization points per wavelength, since $2L/N = 5/8$.

4 Numerical experiments

4.1 Numerical quality and robustness of the trained network as a standalone solver/surrogate

We first investigate the numerical robustness of the trained network used as a surrogate model that serves as stand alone solver. For that purpose we select two sets of trained parameters that have comparable value of the loss function; they are referred to as Training #1 and Training #2. We study their quality and robustness both for problems similar to those used during the training, that is, linear systems arising from the discretization on a 64×64 grid and $c \overset{\text{iid}}{\sim} \mathcal{U}([1,2])$ as well as for other distributions for c and larger grid sizes.

4.1.1 Generalization capabilities with respect to the parameter c



(a) Training #1

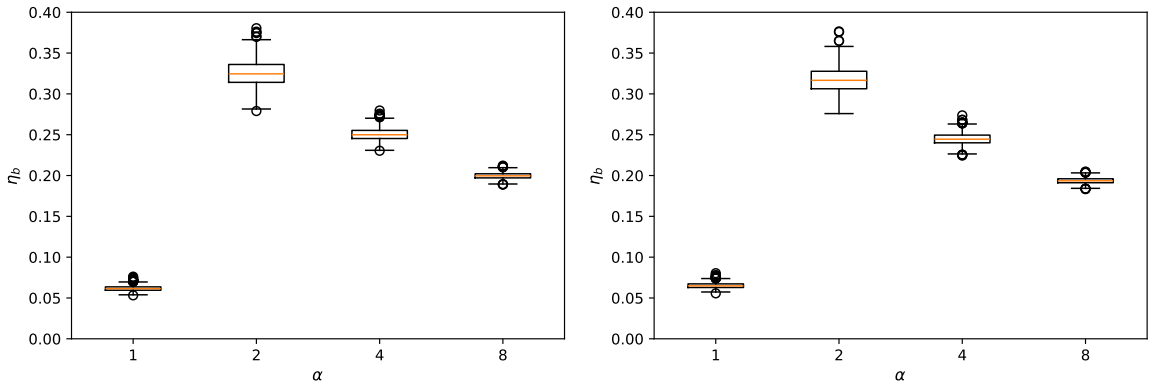
(b) Training #2

Figure 4: Direct prediction from the neural network on problems with c having different distributions, where the neural network has been trained with $\alpha = 1$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([1,2])$. Boxplots of η_b based on 1,000 experiments.

For the set of trained parameters, we display in Figure 4 the boxplots of $\eta_b(\hat{x}_\theta)$ for different distributions of c. These include the distribution used during training (i.e., $c \overset{\text{iid}}{\sim} \mathcal{U}([1,2])$; leftmost boxplot), a distribution strictly contained within the training interval $[1,2]$, namely $[1.25, 1.75]$, and two intervals that progressively extend beyond the training range, namely $[0.75, 2.25]$ and $[0.5, 2.5]$. We note that the latter range implies that $\lambda \in [\pi, 5\pi]$, so that there are still between 5 and 25 discretization points per wavelength. The two figures reveal quasi identical behaviors and trends. The best performance is observed when the network is used to solve systems involving matrices in the training domain, and the performance deteriorates as we move further away. However, for all problem sets, the numerical quality of the predicted solutions by the surrogate model is extremely poor, with a backward error $\eta_b(\hat{x}_\theta)$ that varies between 0.1 and 0.8. Furthermore, the trend and performances are similar for the trained network; the results are not sensitive to the selected network.

4.1.2 Generalization capabilities with respect to the domain size

We conduct similar experiments with both trained networks while varying the domain size, keeping c within the same interval as used for training. As discussed in Section 3.2, the network is trained on a $N \times N$ Cartesian grid, with $N = 64$, discretizing the square domain $[-L, L]^2$, with $L = 20$ and a PML size $\ell = 4$. We scale both N and L by a factor $\alpha \in \{2, 4, 8\}$, so that $L/N = 5/16$ remains fixed, while keeping the PML size $\ell = 4$ and the distribution of $c \overset{\text{iid}}{\sim} \mathcal{U}([1, 2])$ fixed. This results in larger discretization matrices $A(c) \in \mathbb{C}^{n \times n}$ with $n = N^2$, whose size thus grows quadratically with α: $n = 16{,}384$ for $\alpha = 2$, $n = 65{,}536$ for $\alpha = 4$, and $n = 262{,}144$ for $\alpha = 8$; while $n = 4{,}096$ for the reference ($\alpha = 1$). As evidenced by Figure 5, the trends and performance remain similar. The predicted solution exhibits extremely poor numerical quality. At first glance, one might notice an improvement in quality as the domain size increases. However, the differences are not significant, and the overall quality remains very poor. More notably, the variance appears to decrease with larger domain sizes, though we currently have no explanation for this behavior.



(a) Training #1

(b) Training #2

Figure 5: Direct prediction from the neural network on problems with $\alpha \in \{1, 2, 4, 8\}$, where the neural network has been trained with $N = 64$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([1, 2])$. Boxplots of η_b based on 1,000 experiments.

The results presented in the last two subsections indicate that the network cannot be used as a stand alone solver, as it lacks the robustness needed for accurate solutions. In the next section, we will explore its use as a preconditioner within a subspace linear solver (here, FGMRES), where an approximate inverse may be sufficient to accelerate convergence to an accurate solution.

4.2 Numerical quality of the trained network as nonlinear preconditioner

In this section, we report on the numerical effectiveness of the trained networks when used as a nonlinear preconditioner to speed-up the convergence of a subspace method, namely FGMRES. We first compare its performance with respect to an algebraic preconditioner in Section 4.2.1. Then, in the following sections, we investigate the generalization capability with respect to the velocity field parameter c and with respect to the domain size.

4.2.1 Comparison with an algebraic preconditioner

For the sake of comparison, we also report on the numerical performance of a purely algebraic implicit[1] preconditioner built as the LU factorization of an approximation $\tilde{A}(c)$ of $A(c)$ that consists in keeping only the largest entries. More precisely, for a given threshold ϵ we proceed as follows:

$$\begin{cases} \tilde{a}(c)_{ii} = a(c)_{ii}, & \\ \tilde{a}(c)_{ij} = a(c)_{ij}, & \text{if } |a(c)_{ij}| > \epsilon |a(c)_{ii}|, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

We display in Table 4 the ratio of kept entries in the preconditioner and relative distance between the matrix and the implicit preconditioner. We should mention that, due to the chosen discretization technique, the matrix $A(c)$ is quite populated with many nonzero entries but usually does not need to be explicitly formed to solve the associated linear system by a subspace method. Among the large number of nonzero entries, many are of small magnitude which explain the very low nonzero ratio for a moderate distance between the two matrices. As expected the lower the threshold ϵ, the closer to the original matrix and the denser the preconditioner. Since the experiments are run on 100 examples, the quantities reported in Table 4 are the mean values, along with the associated standard deviations indicated between parentheses.

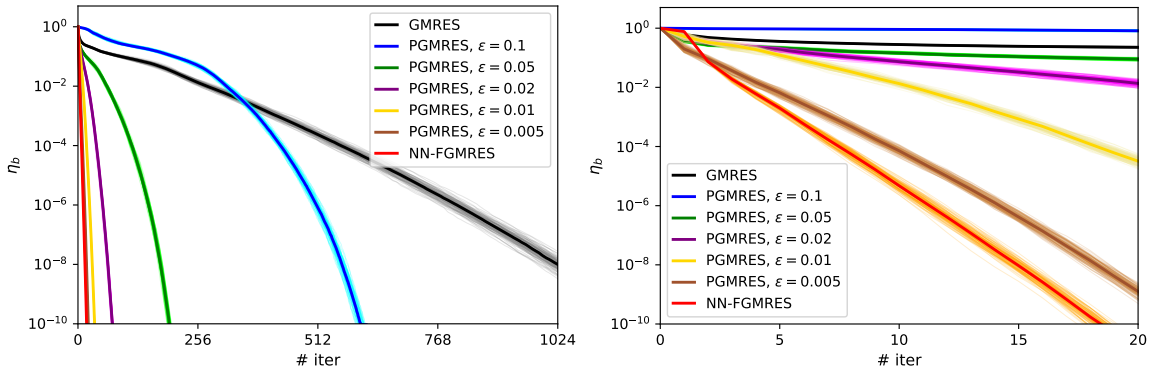
ϵ	$\text{nnz}(\tilde{A}(c))/\,\text{nnz}(A(c))$	$\|\tilde{A}(c) - A(c)\|_{\text{F}}/\|A(c)\|_{\text{F}}$
0.1	$3.4 \cdot 10^{-3}$ $(3.1 \cdot 10^{-6})$	$1.8 \cdot 10^{-1}$ $(6.3 \cdot 10^{-5})$
0.05	$6.5 \cdot 10^{-3}$ $(2.2 \cdot 10^{-6})$	$1.2 \cdot 10^{-1}$ $(2.7 \cdot 10^{-5})$
0.02	$1.5 \cdot 10^{-2}$ $(6.8 \cdot 10^{-6})$	$7.7 \cdot 10^{-2}$ $(1.3 \cdot 10^{-5})$
0.01	$3.2 \cdot 10^{-2}$ $(8.7 \cdot 10^{-6})$	$3.8 \cdot 10^{-2}$ $(1.0 \cdot 10^{-5})$
0.005	$4.9 \cdot 10^{-2}$ $(8.9 \cdot 10^{-6})$	$1.8 \cdot 10^{-2}$ $(7.3 \cdot 10^{-6})$

Table 4: Sparsity of $\tilde{A}$ in terms of proportion of non-zero (denoted as nnz) entries kept compared to A and relative distance of $\tilde{A}$ to A in terms of Frobenius norm, for different values of ϵ. The average value over 100 experiments is reported, along with the standard deviation between parentheses. Note that $\text{nnz}(A(c)) = 7{,}547{,}926$, and $\|A(c)\|_{\text{F}} = 1.1 \cdot 10^{3}$ $(1.8 \cdot 10^{-1})$.

Although the trained network performs poorly as a standalone solver, we highlight its potential to accelerate the convergence of subspace methods like FGMRES. In Figure 6, we present the convergence history of η_b for the solution of 100 linear systems derived from the discretization of the Helmholtz problem with a random velocity field $c \overset{\text{iid}}{\sim} \mathcal{U}([1,2])$, using FGMRES without restart. We display the convergence histories for the 100 linear systems, as well as the median for each system and for all preconditioners built using various sparsification thresholds ϵ. For the sake of completeness, we also display the convergence history of GMRES without a preconditioner, that does not succeed to reach 10^{-10} in 1024 iterations while all the others do. It can be observed that the algebraic implicit preconditioner becomes increasingly effective as ϵ decreases. It is noticeable that $\mathcal{N}_\theta$ outperforms all of them, including the best individual algebraic preconditioner built with $\epsilon = 0.005$, which retains nearly 5% of the nonzero entries of each matrix, as can be seen in the zoom display in Figure 6b. We stress that the algebraic preconditioners require an LU factorization for

[1] implicit in contrast to explicit preconditioners based on approximate inverse techniques

every linear system as c changes, whereas $\mathcal{N}_\theta$ remains the same for all linear systems. Furthermore, all convergence histories using $\mathcal{N}_\theta$ (in orange) are closely aligned and convergence to 10^{-10} is achieved in fewer than 20 iterations. These results illustrate the performance and the robustness of the trained network when used as preconditioner. The next question is what are its capabilities in terms of generalization with respect to the parameter c and with respect to the domain sizes. We investigate these two questions in the next sections.



(a) (b) Zoom on the first 20 iterations.

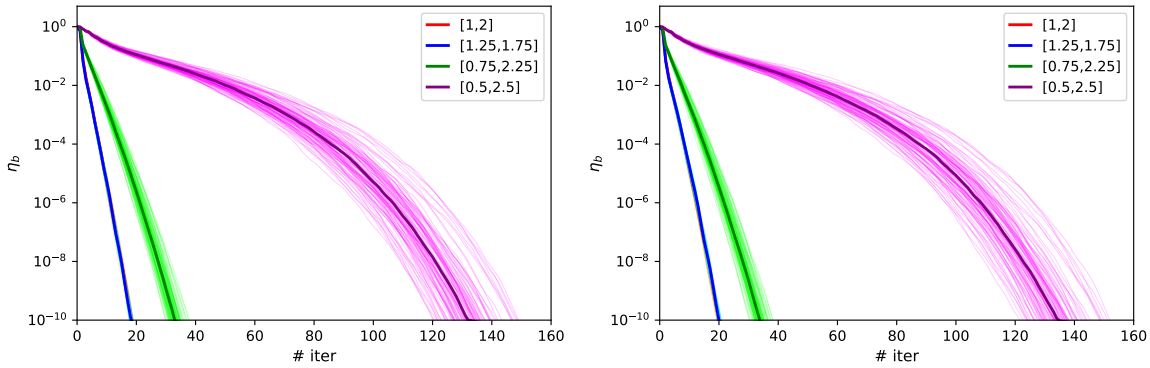
Figure 6: Convergence in terms of η_b of unpreconditioned GMRES, preconditioned GMRES (PGMRES) with the algebraic preconditioners described in Table 4 and FGMRES preconditioned with the trained neural network (NN-FGMRES) corresponding to Training #1, on problems with $\alpha = 1$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([1,2])$ (same as for the neural network training). The iteration-wise median of the convergence history over 100 experiments is displayed as a thick line, while the individual 100 realizations are plotted as thinner, lighter lines.

4.2.2 Generalization capability with respect to the parameter c

As described in Section 3.2, the neural network is trained on a dataset built with $c \in \mathcal{U}([1,2])$ and demonstrates excellent performance as a preconditioner for solving linear systems whose matrices arise from the discretization of problems where c falls within the same range. A natural question arises: how does $\mathcal{N}_\theta$ perform on matrices constructed with c varying within an interval that is either entirely contained within or only partially overlaps the training interval $[1,2]$? To address this, we consider three sets of 100 linear systems each, with c varying in the following intervals: $[1.25, 1.75]$, which is fully contained within the training range; $[0.75, 2.25]$, where the training interval covers two thirds of the range; and finally $[0.5, 2.5]$, where the training interval represents only one third of the values involved in the matrices preconditioned by $\mathcal{N}_\theta$.

The convergence history for these four sets of ranges (including the original range $[1,2]$) is shown in Figure 7 for the two trained networks. The first key observation is that both trained networks exhibit very similar efficiency, indicating that the choice of the network does not significantly impact performance as long as they achieve comparable validation loss during training. The second observation is that when solving problems where c belongs to the training range or a subset of it, the convergence histories are nearly identical, as evidenced by the complete overlap of the curves (e.g., the red curves are entirely covered by the blue ones). Regarding generalization to values outside

the training range, two insights emerge. First, the smaller the overlap with the training range, the greater the degradation in preconditioner performance—$\mathcal{N}_\theta$ converges more slowly for $c \in [0.5, 2.5]$ than for $c \in [0.75, 2.25]$. Second, as the overlap decreases, the variability in the preconditioner's effectiveness increases. Nonetheless, it is worth noting that despite the performance degradation, the preconditioned solver still achieves significant improvements compared to using no preconditioner at all.



(a) Training #1 (b) Training #2

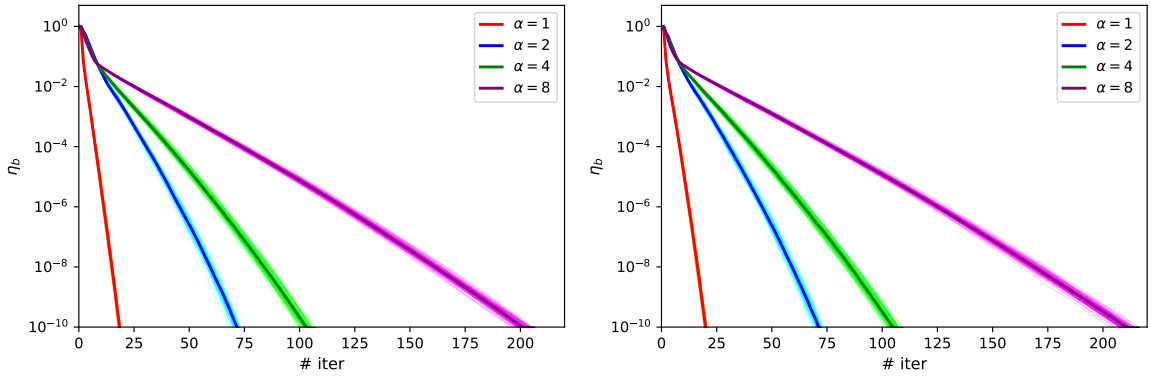
Figure 7: Convergence in terms of η_b of NN-FGMRES on problems with $\alpha = 1$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([a, b])$ with varying ranges $[a, b]$ as reported, where the neural network has been trained with $\alpha = 1$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([1, 2])$. The iteration-wise median of the convergence history over 100 experiments is displayed as a thick line, while the individual 100 realizations are plotted as thinner, lighter lines.

4.2.3 Generalization capability with respect to the domain size

In this section, we analyze the generalization capabilities of the preconditioner concerning the size of the physical domain, which translates into solving larger linear systems. Following the same approach as in previous experiments, we solve 100 linear systems for each system size using $\mathcal{N}_\theta$ with both trained networks. Figure 8 displays the convergence history for all 100 solutions, along with the median convergence history. The results show that as the problem size increases, the number of iterations required for convergence also grows, though the dispersion around the median remains relatively stable. Additionally, the choice of trained network has little impact, as the trends for Training #1 (left graph) and Training #2 (right graph) are very similar. Notably, when increasing the scaling factor from $\alpha = 4$ to $\alpha = 8$, the number of iterations approximately doubles, even though the problem size is quadrupled. This highlights some robustness of the $\mathcal{N}_\theta$ preconditioner with respect to problem size.

5 Further discussion

We have presented results only with U-Net in this work, but preliminary results [18] on 1D examples suggest that using other neural network architectures, trained with the same loss function leads to similar outcomes with different preconditioner efficiency. The objective of this work is to



(a) Training #1 (b) Training #2

Figure 8: Convergence in terms of η_b of NN-FGMRES on problems with $\alpha \in \{1, 2, 4, 8\}$, where the neural network has been trained with $\alpha = 1$ and $c \overset{\text{iid}}{\sim} \mathcal{U}([1, 2])$. The iteration-wise median of the convergence history over 100 experiments is displayed as a thick line, while the individual 100 realizations are plotted as thinner, lighter lines.

demonstrate that machine learning methods, which often struggle with precision issues, can be effectively combined with classical linear algebra techniques—such as suitable subspace methods—to leverage the best of both worlds. This approach ensures solution quality through traditional numerical techniques while benefiting from the acceleration provided by neural networks. It is a highly generic strategy that can be applied in various contexts [20]. Identifying the optimal architecture is beyond the scope of this paper. However, certain machine learning architectures may lack the necessary properties to function effectively as nonlinear preconditioners. For example, as noted in [11], DeepONet constructs surrogates that reside in a low-dimensional space of dimension the output dimension of the trunk net branch. If this space has a lower dimension than that of the linear system to be solved, the resulting nonlinear preconditioner will be "low-rank", potentially leading to a breakdown of FGMRES. Therefore, careful consideration is required when designing the neural network architecture within the DeepONet framework to mitigate this issue. For an illustration of this potential limitation, we refer to [18].

6 Concluding remarks

In this study, we illustrate the potential of using neural networks as nonlinear preconditioners to solve parametric Helmholtz equations using subspace methods. While the accuracy and robustness of trained neural networks as standalone solvers is limited, integrating them as nonlinear preconditioners within the FGMRES algorithm significantly accelerates convergence, even for problems that are far beyond the training distribution. This hybrid approach combines the generalisation capabilities and computational speed of neural networks with the numerical reliability of classical iterative methods.

We would like to emphasise that the proposed strategy is not tied to a specific neural network architecture. Although we opted for a U-Net-based model due to its convolutional structure and efficient use of parameters, other architectures could also produce effective preconditioners, provided they retain sufficient representational capacity. The generic framework can be extended to other classes of linear systems beyond the Helmholtz equation, offering a promising approach to machine learning in scientific computing.

Several promising avenues for future research have emerged. Firstly, incorporating models that are independent of discretisation, such as PINNs or neural operators, which are trained across multiple resolutions, could improve the ability to generalise to arbitrary domain sizes and mesh structures. Secondly, extending the methodology to three-dimensional problems and other PDEs.

The intersection of numerical linear algebra and machine learning continues to pave the way for solving complex, high-dimensional scientific problems. The methodology presented here builds on this growing body of work by demonstrating that approximate learned inverses can be effectively integrated into established numerical schemes.

Acknowledgements

We would like to thank Pierre Benjamin and Sofiane Haddad from Airbus CR & T for fruitful discussions and exchanges that took place during the course of this work.

References

[1] Y. Azulay and E. Treister. "Multigrid-augmented deep learning preconditioners for the Helmholtz equation". In: *SIAM Journal on Scientific Computing* (2022).

[2] J.-P. Berenger. "A Perfectly Matched Layer for the Absorption of Electromagnetic Waves". In: *Journal of Computational Physics* 114.2 (1994), pp. 185–200. ISSN: 0021-9991. DOI: 10.1006/jcph.1994.1159.

[3] C. M. Bishop and H. Bishop. "Autoencoders". In: *Deep Learning: Foundations and Concepts*. Cham: Springer International Publishing, 2024, pp. 563–579. ISBN: 978-3-031-45468-4. DOI: 10.1007/978-3-031-45468-4_19.

[4] N. Dimola, N. R. Franco, and P. Zunino. "Numerical Solution of Mixed-Dimensional PDEs Using a Neural Preconditioner". In: arXiv:2505.08491 (2025). DOI: 10.48550/arXiv.2505.08491.

[5] N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002. DOI: 10.1137/1.9780898718027.

[6] A. N. Juscafresa. *Graph neural network-based preconditioners for optimizing GMRES algorithm*. DiVA Portal. 2024.

[7] G. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang. "Physics-informed machine learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[8] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: arXiv:1412.6980 (2017). DOI: 10.48550/arXiv.1412.6980.

[9] A. Kopaničáková and G. Karniadakis. "DeepONet Based Preconditioning Strategies for Solving Parametric Linear Systems of Equations". In: *SIAM Journal on Scientific Computing* 47.1 (Feb. 2025), C151–C181. ISSN: 1095-7197. DOI: 10.1137/24m162861x.

[10] Z. Li, D. Xiao, Z. Lai, and W. Wang. "Neural Preconditioning Operator for Efficient PDE Solves". In: arXiv:2502.01337 (2025). DOI: 10.48550/arXiv.2505.01337.

[11] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. Karniadakis. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators". In: *Nature Machine Intelligence* 3 (Mar. 2021), pp. 218–229. DOI: 10.1038/s42256-021-00302-5.

[12] N. S. Moore. "Machine learning techniques applied to preconditioning of linear systems". PhD thesis. Texas Tech University, 2022.

[13] J. L. Rigal and J. Gaches. "On the Compatibility of a Given Solution With the Data of a Linear System". In: *Journal of the ACM* 14.3 (July 1967), pp. 543–548. DOI: 10.1145/321406.321416.

[14] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: arXiv:1505.04597 (2015). DOI: 10.48550/arXiv.1505.04597.

[15] A. Rudikov, V. Fanaskov, E. Muravleva, Y. M. Laevsky, and I. Oseledets. "Neural operators meet conjugate gradients: The FCG-NO method for efficient PDE solving". In: *Forty-first International Conference on Machine Learning*. 2024.

[16] Y. Saad. "A flexible inner-outer preconditioned GMRES algorithm". In: *SIAM Journal Scientific Computing* 14 (1993), pp. 461–469.

[17] Y. Saad and M. H. Schultz. "GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems." In: *SIAM Journal on Scientific and Statistical Computing* 7 (1986), pp. 856–869.

[18] M. Shpakovych. *Neural Network Preconditioning of Large Linear Systems*. Technical Report 0518. Inria Centre at the University of Bordeaux, 2023.

[19] A. Stanziola, S. R. Arridge, B. T. Cox, and B. E. Treeby. "A Helmholtz Equation Solver Using Unsupervised Learning: Application to Transcranial Ultrasound". In: *Journal of Computational Physics* 441 (2021), p. 110430. ISSN: 00219991. DOI: 10.1016/j.jcp.2021.110430.

[20] V. Trifonov, A. Rudikov, O. Iliev, Y. M. Laevsky, and I. Oseledets. "Efficient preconditioning for iterative methods with graph neural networks". In: *AI4X 2025 International Conference*. 2025.

[21] V. Trifonov, A. Rudikov, O. Iliev, Y. M. Laevsky, and I. Oseledets. "Learning from linear algebra: A graph neural network approach to preconditioner design for conjugate gradient solvers". In: arXiv:2405.15557 (2024). DOI: 10.48550/arXiv.2405.15557.

[22] J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Notes on Applied Science 32. Also published by Prentice-Hall, Englewood Cliffs, NJ, USA, 1964. Reprinted by Dover Publications, New York, 1994. London, UK: HMSO, 1963.

[23] Y.-F. Xiang. "Solution of Large Linear Systems with a Massive Number of Right-Hand Sides and Machine Learning". PhD thesis. University of Bordeaux, 2022.

[24] S. Yusuf, J. E. Hicken, and S. Pan. "Constructing ILU preconditioners for advection-dominated problems using graph neural networks". In: *AIAA AVIATION Forum*. 2024.

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399