Unsupervised convolution neural operator preconditioning for the solution of some heterogeneous fluid PDEs

Yanfei Xiang

Unsupervised convolution neural operator preconditioning for the solution of some heterogeneous fluid PDEs

Yanfei Xiang*

Contents

*Corresponding author: yfxiang0amber@gmail.com and yanfei.xiang@inria.fr. This work was finished when the author was in Concace, Inria project with Airbus CR & T and Cerfacs, Inria Centre at the University of Bordeaux, France.

Abstract

This work exclusively focuses on utilizing convolution neural operator learning to accelerate the solution of some heterogenous PDEs (including Poisson equations, Darcy flow, Convection-Diffusion equations) via the non-linear preconditioning applied to the flexible GMRES method. Convolutional neural networks with U-Net architecture is employed for learning this neural operator preconditioning. For the sake of learning general information, the neural operator is trained with randomly generated datasets using an unsupervised approach. The neural operator is trained with 2 V100 GPUs and the training time is around 1 hour. The trained neural operator exhibits significant generalization features with respect to different aspects. That includes the ability to address varying source terms, permeability terms, diffusivity terms, velocity field for convection, and varying boundary conditions for these heterogeneous fluid equations. Furthermore, it demonstrates promising results in addressing Convection-Diffusion equations under challenging conditions, such as convection-dominant cases and scenarios with a wider range of convection and diffusion terms. Overall, this work demonstrates the efficiency of applying the neural operator learning as non-linear flexible preconditioner for subspace iterative linear solver for addressing heterogeneous fluid equations with varying parameters and varying boundary conditions.

Keywords — Scientific Machine Learning; Convolutional Neural Networks; Unsupervised Neural Operator Learning; Heterogeneous fluid equations; Poisson equations; Heterogeneous Darcy flow; Heterogeneous Convection-Diffusion equations; Subspace iterative methods on CPUs and/or on GPUs; Flexible Preconditioning; Mixed-Precision calculation.

1 Introduction

1.1 Scientific background and major challenges

Fluid simulation modeled by heterogeneous Partial Differential Equations (PDEs) has numerous applications across various engineering fields, including Computational Fluid Dynamics (CFD), plasma physics, parametric studies, and the formulation of inverse problems, to name a few. Numerical solutions are indispensable in practical applications, as analytic solutions for PDEs are rarely available. Numerous general-purpose numerical methods have been developed to address this need, with iterative solvers [94] among the most commonly used approaches for obtaining numerical solutions to PDEs. While iterative methods are theoretically well-founded, they often struggle with slow convergence and lack robustness, particularly when applied to complex problems from industrial applications, such as fluid simulations. These issues can be mitigated through preconditioning [94, Chapter 9-10], a technique that transforms the original system into an equivalent one with the same solution. And the transformed system is likely to be easier to solve with iterative solvers. However, constructing an effective numerical preconditioner can be as challenging as directly solving the original system. Moreover, a preconditioner tailored to one specific problem often lacks generalizability to other similar systems, which limits its applicability for heterogeneous systems. In recent decades, machine learning techniques, particularly deep learning [22] with deep neural networks (DNNs) and convolutional neural networks (CNNs), have gained widespread application in scientific computing, especially for problems related to PDEs [60, 20, 62, 47, 38, 66, 28, 105]. This emerging field, often called Scientific Machine Learning (SciML) [77], encompasses the application of machine learning techniques for scientific computing. Two common trends in this field involve: (1) training neural networks (NNs) as a solver, like Physics-Informed Neural Networks (PINNs) [69] and its related DeepXDE library [68]; and (2) training NNs for general function approximation through neural operator learning [59, 14]. In recent years, these approaches have gained significant attention, leading to extensive research. For a comprehensive overview of developments in PINNs, we refer readers to the review in [24]. For more details on neural operator learning, including various neural network architectures, techniques and physical applications, further references include [67, 86, 48, 65, 64, 39, 34]. These recent methods show considerable promise, particularly when trained effectively. This promise largely stems from their ability to approximate complex functions in diverse scientific contexts, such as non-linear PDEs [2, 85] and high-dimensional PDEs [99], which are often challenging for traditional iterative methods. However, these promising SciML approaches still face unconquered challenges, including limited rigorous mathematical analysis, lack of guarantees for correctness or convergence, limited attainable accuracy, and limitations in handling unseen scenarios.

1.2 Motivation and methods overview

Given the challenges faced by both traditional iterative methods and recent SciML approaches, this work focuses on combining the strengths of these two categories to leverage their respective advantages to enhance performance in heterogeneous fluid simulations. From iterative methods side, it is know that effective preconditioning can accelerate convergence and improve robustness. Within category of preconditioned iterative methods, the flexible GMRES method (FGMRES) [93] can be preconditioned by preconditioner with non-linear property, and the NNs also owns highly non-linear property. From SciML approaches side, there are impressive applications of CNNs with U-Net [90] architecture (characterized by its "U" shape, formed by encoding and decoding layers) for the fluid [21, 34] and wave [102, 4] simulations. Based on these prior works, we explore an integration through *CNNs preconditioning*, leveraging the learning capability of CNNs with U-Net architecture to develop a generalized non-linear preconditioner for the FGMRES method. This preconditioner, while not intended to achieve high attainable accuracy (a typical limitation of many SciML approaches), is instead optimized for robustness and adaptability across various scenarios. The trained CNNs inference is utilized as a non-linear flexible preconditioner that takes the Krylov basis as input to improve the effectiveness of the FGMRES method. This integrated approach is designed to tackle heterogeneous linear PDEs from fluid physics, including Poisson equations, heterogeneous Darcy flows, and heterogeneous Convection-Diffusion equations with non-uniform source terms. These types of equations typically present challenges for traditional numerical preconditioners, especially when dealing with complex boundary conditions and varying parameters within the equations. To enable the CNN-based preconditioner to generalize effectively across different problem instances, we use unsupervised learning on datasets generated from random realizations of these heterogeneous PDEs. This approach demonstrates the potential efficiency of training CNNs preconditioner without requiring pre-existing physical dataset. It also enables the CNNs preconditioner to capture broadly applicable patterns and characteristics of each fluid equations, improving its adaptability and performance across a range of diverse conditions. In short, this work aims to establish an unsupervised convolution neural operator preconditioning technique that improves the efficiency of traditional iterative solvers in addressing complex fluid simulations.

1.3 Key contributions

The core contribution of this work lies in proposing a novel integrated approach through *CNNs preconditioning* to bridge the numerical iterative methods and the unsupervised SciML approaches to let them benefit from each other. The integrated approach is designed for accelerating the simulations for some linear heterogeneous fluid equations with varying parameters and boundary conditions (BCs). This integrated approach has four main attractive properties: (1) *New hybrid point.* We introduce a novel hybrid point through flexible preconditioning to glue the traditional iterative methods and recent SciML approaches to leverage their respective advantages. With mathematical backbones of the iterative methods, we can utilize the black-box-like SciML approach is a relatively reliable way. With learning ability of the SciML methods, we can extend the application of numerical methods to even wider range of complex problems. (2) *Greedy application scope.* This CNNs preconditioning is capable of learning general functions across an entire class of heterogeneous fluid PDEs, with random parameters generated according to specified distributions. Importantly, this CNN-based preconditioner requires only a single training phase and can then be applied to different problems within this class without needing to be retrained. This feature is particularly advantageous given that the training process is typically resource-intensive [103]. (3) *New loss metric.* Inspired by the common stopping criterion used in the iterative methods, we define the loss function as the relative residual of the discrete PDEs. This relative residual is computed directly from the PDE information and the algebraic properties of the network output, which approximates the PDE solution. As a result, the network is trained in an unsupervised manner with randomly generated fluid parameters and the local sparsity structure of the discrete PDEs. This training approach does not require a ground-truth solution—often unavailable in practical scenarios—nor does it rely on the numerical solutions (like generated from a finite element method) to the PDEs, which are typically costly to obtain. By focusing on minimizing the relative residual with random parameters, this way allows for a more flexible and efficient training process, enabling the network to learn generalizable function without dependence on precomputed or exact solutions. (4) *Generalization.* While this method is designed for learning CNNs preconditioning for fluid equations with varying BCs, it is suffice to train the networks on small problems with the simplest

zero-Dirichlet BCs. Generalizability of the trained CNNs preconditioning is tested from diverse aspects. That includes the ability to address varying BCs, source terms, permeability terms, diffusivity terms, velocity field for convection, and the domain size of these heterogeneous fluid equations. Furthermore, it also shows promising results for addressing wider range of the convection dominant situations, which is the challenging case for the Convection-Diffusion equations. Extensive experiments and comprehensive evaluations have verified the significant generalization ability of this unsupervised CNNs preconditioning for various heterogeneous fluid equations.

1.4 Previous efforts in integrating machine learning and iterative methods

Vast amount of researches have focused on integrating recent machine learning (ML) techniques with traditional iterative methods to enhance their combined effectiveness for PDEs simulations. Key efforts to accelerate linear iterative methods (both subspace-like method and multilevel solver) and non-linear iterative solvers, as well as to simplify their usability can be summarized as follows (though not exhaustively).

- *Preconditioning learning.* Staring from efforts to enhance a neural networks (NNs) solver in [102], [109, Chapter 5] initially demonstrated that the trained NNs inferences of NNs solver can also serve as an effective non-linear preconditioner for Krylov subspace methods, including the FGMRES and the flexible FOM (FFOM) methods [94]. With similar idea, [92] illustrated using neural operator as non-linear preconditioner for the flexible conjugate gradient method [80]. More recently, [30] introduced greedy learning to optimize the linear parametrization used as preconditioning, capable of functioning in various forms—ranging from scalar and vector to full linear operator. In the context of multilevel preconditioning, [4] applied deep learning to multigrid method to accelerate the convergence for solving Helmholtz equations. Within multilevel domain decomposition framework, [76] proposed a conjugate gradient method [45] preconditioned with a multilevel Graph Neural Networks (GNNs) [5] preconditioner for solving Poisson equation with irregular meshes. Another ML-based multiscale preconditioner can be found at [63] for elliptic equations in porous structure. If view recycling subspace methods [74, 81] as a type of preconditioning realized by projection, [107] explored learning-based recycling subspaces applied to the GMRES method [95] for accelerating electromagnetic simulations. Refer to [1, 96, 35] for some other ML-based preconditioners.

- *Initial guess learning.* Except for preconditioning, initial guess also effects convergence rate of the iterative methods. For the linear solver side, [70] introduced the use of deep learning to learn effective initial guesses aimed at accelerating the restarted GMRES method [94]. For the non-linear solver side, [2] explored learning optimal initial guesses with neural operator to accelerate the performance of Newton's method when applied to non-linear PDEs.

- *Optimal parameters learning.* [55] demonstrated that a bandit online ML algorithm can effectively select optimal parameters ϖ from a given interval (like $\varpi \in (0, 2)$) to speed up the iterative Successive Over-Relaxation solver (SOR) [111] when applied to solve sequences of linear systems.

- *Alternative algorithms.* Based on Tompson et. al.'s pioneering work on novel NNs method for fluid simulations [106], [50] proposed an alternative strategy that applies the NNs method or the Jacobi algorithm for simulating incompressible flows with a large Richardson number. Also, [87] proposed to intersperse Krylov iterations and NNs correction to accelerate wave simulation.

- *Multigrid and algebraic multigrid.* Multigrid (MG) techniques [15] exploit discretizations with different mesh sizes (like hierarchical meshes) of PDE to obtain optimal convergence from relaxation techniques. Its general version with coarser and finer levels is termed as algebraic multigrid (AMG). From NN for MG side, follow the prior work [47] on integrating the structure of multigrid V-cycles and U-Net, [40] proposed a new neural multigrid solver. Besides a merging PINN-MG method was proposed at [27], where the high- and low-frequency components are respectively solved by iterative methods and PINNs. On the MG for NN side, based on the connections between NN and MG, [41] devised modified NN models with fewer weights and hyperparameters. Prolongation defined in MG relates different scales, [37] proposed to train NN to map discrete PDEs to prolongation operators, which is then extended to AMG on irregular meshes via GNNs at [71]. For other NN-enhanced AMG works, refer to [17, 3] for learning optimal parameter constructed in AMG by NN for acceleration.

- *Domain decomposition methods.* Domain decomposition methods (DDMs) [94, Chapter 14] represent a class of techniques based on the principle of divide-and-conquer. From ML for DDMs side, refer to [43] for an example of ML-enhanced DDMs approach. From DDM for ML side, refer to FBPINNs [75] and its further extensions [26, 42] for improving the performance of PINNs by DDMs' divide-and-conquer idea. Except for these DDMs-enhanced PINNs, refer to [46, 108, 57] for other novel neural network architectures improved by DDMs. For more comprehensive information, we refer the reader to [44, 58], which provide detailed surveys on the integration of ML and DDMs.
- *Recommendation system.* This direction focuses on learning ML-based recommendation system to optimize selection of iterative subspace-like method, preconditioner, and parameter for specific PDEs, reducing the need for extensive manual tuning. Examples include SALSA and Lighthouse projects [10, 101, 53, 100]. Refer to [19] for auto-select best solvers, among three available solvers, for transport problems. Refer to [82] for using reinforcement learning [104] to select optimal restarting parameter, among a group of predefined values, for accelerating restarted GMRES method. Refer to [110] for selecting properly iterative algorithms and preconditioners based on features of the sparse matrix.

On the other hand, the integration of machine learning with non-iterative numerical linear algebra methods has also been explored in various contexts, as outlined below.

- *Randomization.* Random feature [61, 84] has been applied to [78] for developing operator learning with some convergence guarantees. It has also been used at [18] for combining the advantages of numerical approaches and ML-based methods. On the other hand, randomized singular value decomposition [72] has been applied to data-driven methods in [13, 12] for acceleration.
- *Reduced order models.* Reduced-order models (ROMs) have achieved a lot of successes in reducing the computational cost of traditional numerical methods across many disciplines, like [29, 83]. For their interaction with ML approaches, refer to [11] for introducing ROMs into data-driven methods for wave simulations. Additionally, refer to GAROM [23], POD-LSTM ROM [9], NNsPOD-ROM [36] and other related works by G. Rozza and his colleagues [1] [52, 51, 54, 98, 33, 89, 25, 88], for more further explorations of ROMs-enhanced ML approaches.

1.5 Structure and notations

The remainder of this paper is organized as follows. We begin by recalling the general mathematical preliminaries for addressing PDEs using numerical approaches in Section 2. This section outlines the problem formulations of PDEs, including boundary conditions (BCs), their discretization, and the iterative methods used for solving the resulting discrete systems. Based on previous discussions about the PDEs and the iterative methods for discrete PDEs, Section 3 details the development of a new convolutional neural network (CNN) based operator preconditioning framework. This framework introduces a general and effective CNN-based preconditioner designed to enhance the performance of traditional iterative methods for solving sequences of discrete heterogeneous fluid PDEs. In Section 4, we demonstrate that the advantages of integrating these ideas through various examples featuring novel physical parameters and PDE properties, such as diverse BCs. It turns out the proposed CNN-based preconditioner exhibits significant network generalizability across wider range of applications. Finally, we present concluding remarks in Section 5, summarizing the findings, implications and some further perspectives of this work.

Key notations used in this paper are summarized below. The symbol $||\cdot||$ denotes the Euclidean norm by default for both vectors and matrices. The superscript ^{H} denotes the transpose conjugate and ^{T} stands for transpose. The notation $\mathbb{C}$ and $\mathbb{R}$ respectively refer to the complex and real number field. For convenience of the algorithm illustration and presentation, some `Python` notations and functions are used. Because much notation is involved, we make certain choices to improve the readability of the chapter. The scales are denoted by lowercase letters; matrices with multiple columns are described by uppercase letters, like matrix C; lowercase blackboard bold letters, e.g., $\mathbf{x}$ represent (column) vectors. Without special note, a subscript $_j$ for a vector or a matrix is used to indicate that the vector or matrix is obtained at iteration j, and a positive subscript integer $_m$ represents the maximal iteration number of each Krylov cycle. The inner product between two vectors $\mathbf{x}$ and $\mathbf{y}$ is formed as $\langle \mathbf{x}, \mathbf{y} \rangle$. Furthermore, the space spanned by the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_j$ is denoted as $\mathrm{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_j\}$.

[1] https://people.sissa.it/ grozza/

2 Mathematical preliminaries

For mathematical completeness, this section provides a brief overview of the heterogeneous fluid PDEs defined on a two-dimensional (simplified as 2D) domain and the numerical concepts addressed in this paper. It include descriptions of their corresponding boundary conditions (BCs), discretization techniques, and the iterative methods used to solve the resulting discrete systems. To simplify the presentation for 2D domain, we perform the following derivations in two dimensions, denoted as the x and y directions, respectively.

2.1 Problem formulation

We consider three common heterogeneous fluid PDEs shown in Section 2.1.1-2.1.3, incorporating varying classical BCs, such as Dirichlet, Neumann, Cauchy, and their combinations across different boundary segments.

2.1.1 Poisson equations

One of the most common PDEs encountered in various areas of engineering is the Poisson equation:

$$\nabla^2 u(x,y) = \rho(x,y), \tag{1}$$

where $x, y \in \mathbb{R}$ is a point in the 2-dimensional domain $\Omega \subset \mathbb{R}^2$, $\rho(x,y)$ is the given source function, and $u(x,y)$ is the solution to be computed. Here, we consider addressing heterogeneous Poisson equation, that is Equation (1) with non-uniform source term $\rho(x,y) \neq 0$ that varies from point to point.

2.1.2 Heterogeneous Darcy flow

Divergence form of elliptic equations [31] appear in various applications, notably in modeling groundwater flow through porous media as described by Darcy's law [7]. For simplicity and brevity, reusing the notations from Equation (1), this linear elliptic problem is expressed as:

$$-\nabla\big(p(x,y)\nabla u(x,y)\big) = \rho(x,y), \tag{2}$$

where $p(x,y)$ represents the permeability of the porous medium, and $p(x,y)$ is strictly positive almost everywhere within the domain Ω (i.e., $p(x,y) : \mathbb{R}^2 \to \mathbb{R}_+$). The heterogeneous Darcy flow considered in this work refers to Equation (2), characterized by both non-uniform permeability $p(x,y)$ and non-uniform source term $\rho(x,y)$ over domain Ω.

2.1.3 Heterogeneous Convection-Diffusion equations

The Convection-Diffusion equation is a parabolic PDE that captures the interplay between convective (advection) and diffusive effects. Reusing the notations from Equation (1), we consider the linear heterogeneous Convection-Diffusion equation of the following form:

$$-\nabla\big(d(x,y)\nabla u(x,y)\big) + c(x,y)\nabla u(x,y) = \rho(x,y), \tag{3}$$

where $c(x,y) : \mathbb{R}^2 \to \mathbb{R}_+$ represents the heterogeneous velocity or force field responsible for convection, and $d(x,y) : \mathbb{R}^2 \to \mathbb{R}_+$ denotes the heterogeneous diffusivity or viscosity field associated with diffusion. Within Equation (3), the $d(x,y), c(x,y), \rho(x,y)$ are given functions satisfying non-uniform distributions over domain Ω.

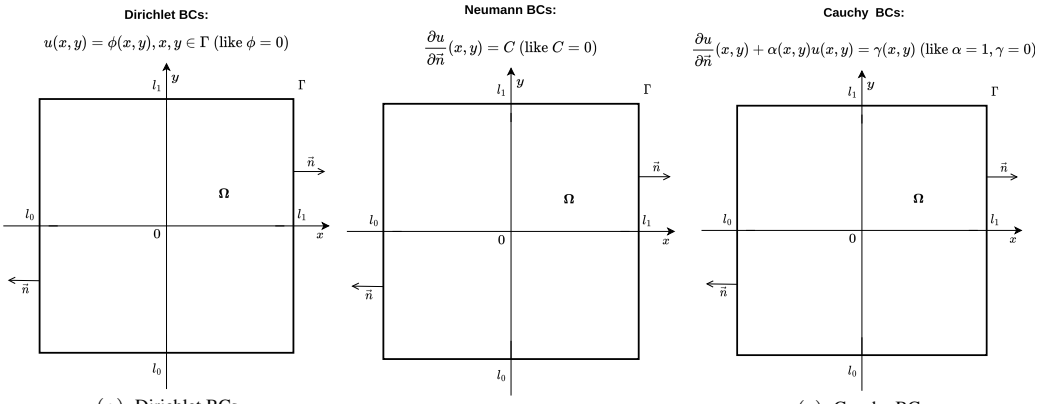
Additionally, the behavior of Equation (3) varies based on the ratio between the convection term $c(x,y)$ and the diffusion term $d(x,y)$. It includes the balanced case where $\frac{d(x,y)}{c(x,y)} \approx 1$ (means convection and diffusion effects are balanced), the diffusion-dominant case where $\frac{d(x,y)}{c(x,y)} \gg 1$ (means diffusion significantly outweighs convection), and the convection-dominant case where $\frac{d(x,y)}{c(x,y)} \ll 1$ (means convection dominates diffusion). Among these, the convection-dominant case is the most challenging to address. In this work, we tackle all these three cases with CNN-based preconditioning. For further insights into this equation, including its analysis and numerical methods, we refer readers to [91].

2.1.4 Boundary conditions

For the three 2D heterogeneous fluid PDEs described in Equation (1)-(3), we consider three classical boundary conditions (BCs): Dirichlet, Neumann, and Cauchy described at

$$\begin{cases} \text{Dirichlet BCs: } u(x,y) = \phi(x,y) \text{ (like } \phi(x,y) = 0); \\ \text{Neumann BCs: } \dfrac{\partial u}{\partial \vec{n}}(x,y) = C \text{ (like } C = 0); \\ \text{Cauchy BCs: } \dfrac{\partial u}{\partial \vec{n}}(x,y) + \alpha(x,y)u(x,y) = \gamma(x,y) \text{ (like } \alpha(x,y) = 1, \gamma(x,y) = 0). \end{cases} \tag{4}$$

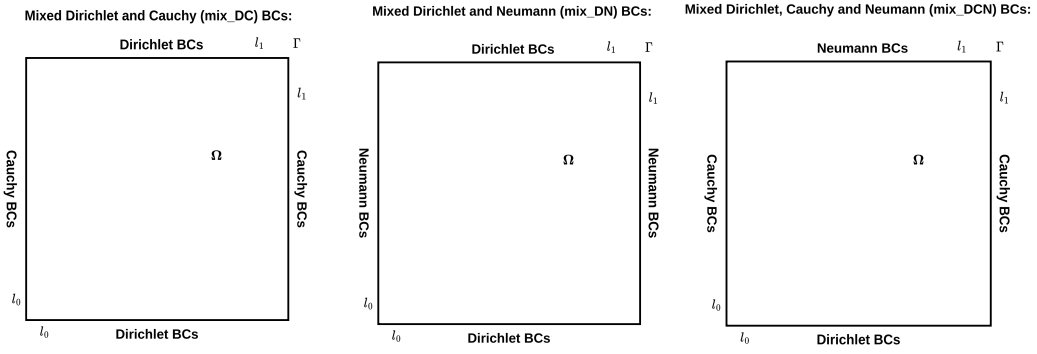
Specifically, we consider applying these three classical BCs on a regular 2D square domain Ω with its boundary side Γ illustrated in Figure 1. Furthermore, their mixtures (i.e., their combinations across different



(a). Dirichlet BCs (b). Neumann BCs (c). Cauchy BCs

Figure 1: Illustration of the three type of classical BCs shown in Equation (4).

boundary sides) shown in Figure 2 are also respectively applied to these three types of fluid equations.



(a). mix_DC BCs (b). mix_DN BCs (c). mix_DCN BCs

Figure 2: Illustration of the three type of mixed classical BCs based on Equation (4).

As a summary, this work focuses on training CNN-based preconditioners to address Equation (1)-(3) under six different BCs, including the three classical BCs shown in Figure 1 and their three mixtures shown in Figure 2. Although the NNs approach is designed to handle these heterogeneous fluid equations with varying BCs, it is sufficient to train the NNs on small-scale problems with the simplest zero-Dirichlet BCs, where the Dirichlet BCs is specified as $\phi(x,y) = 0$ in Equation (4) or in Figure 1 (a). with $\phi(x,y) = 0$.

2.2 Discretization

Addressing the discrete fluid equations (1)-(3) on a regular 2D square domain involves employing numerical differentiation methods such as the fast Fourier transform (FFT) and its discrete variant [49, 32],

or the Finite Difference Method (FDM) [94, Chapter 2] [73] with regular uniform meshes in both the x and y directions. In this work, the differentiation operation for training the CNN-based preconditioner is carried out using FFT. Assume Ω is a 2D domain defined over the square region $\left[[l_0^x, l_1^x], [l_0^y, l_1^y]\right]$ for both the x and y directions. To discretize this 2D domain Ω, consider N-discretisation points such that $\mathbf{x} = [x_1, x_2, \ldots, x_N] \in \mathbb{R}^N$, where $N = n^2$ and n represents the number of discretisation points along each direction. Specially for this section, the subscript $_i$ for a scale refers to the i-th discretisation point over the 2D domain Ω. Note that the setting of N is problem-depending. Generally, for a fixed domain region, a smaller value of N refers to a coarser discretization sufficient for simpler problems, while a lager value of N corresponds to finer discretization for more complex problems. Let's $\mathcal{F}$ and $\mathcal{F}^{-1}$ respectively refers to the direct and inverse Fourier transform, the way to approximate the differentiation parts shown in Equation (1)-(3) can be described as the following Proposition 1.

Proposition 1 *([97, Proposition 2]) Let $\mathbf{f} = \{f_1, \ldots, f_N\}$ be a discretisation of a differentiable function $f(t)$ on $t \in [0, T]$ such that $f(t)$ and $f'(t)$ are absolutely integrable, where $f_k = f(k\frac{T}{N})$ for $k \in \{1, \ldots, N\}$. Let $\mathbf{f}' = \{f'_1, \ldots, f'_N\}$ be a discretisation of the derivative $f'(t)$, where $f'_k = f'(k\frac{T}{N})$. Then we can approximate $\mathbf{f}'$ by the direct and inverse Fourier Transform on $\mathbf{f}$ as*

$$\mathbf{f}' \approx \mathcal{F}^{-1}\{i\boldsymbol{\omega}\mathcal{F}\{\mathbf{f}\}\}, \tag{5}$$

where $\boldsymbol{\omega} = \{\omega_1, \ldots, \omega_N\}$ is the discrete Fourier domain with $\omega_n = 2\pi\frac{N}{T}$.

Note that, as indicated by the original PINNs [69] framework, derivatives can also be computed by the automatic differentiation algorithm (`autograd`) [6] [79, Section 8.2], which operates with respect to the trainable NNs parameters θ (i.e., the trainable weights and biases of the NNs). However, given the number of parameters θ is typically much larger than the grid point N, the `autograd` algorithm is generally slower and memory-consuming. Consequently, in our approach, we opt for numerical differentiation with FFT to approximate the derivatives. Refer to [65, Section 3.3] for more discussions in the comparison of the available derivatives used for the NNs learning.

Discretization with FFT and given BCs discussed in Section 2.1.4, the analytical expression of Equation (1)-(3) with fixed heterogeneous terms, including the permeability $p(x, y)$, the diffusion $d(x, y)$ or the convection $c(x, y)$, or the source $\rho(x, y)$, can be respectively transformed into the following discrete fluid linear systems:

$$\text{Poisson eqs.:} \quad A\mathbf{u} = \mathbf{b}\,, \tag{6}$$

$$\text{Darcy flow:} \quad A(\mathbf{p})\mathbf{u} = \mathbf{b}\,, \tag{7}$$

$$\text{Convection-Diffusion eqs.:} \quad A(\mathbf{d}, \mathbf{c})\mathbf{u} = \mathbf{b}\,. \tag{8}$$

Here $A \in \mathbb{C}^{N\times N}$ in Equation (6) is a Hermitian matrix [2], if applying the Dirichlet BCs, corresponds to the discrete Laplace operator ∇^2 with $\nabla^2 := \sum_{j=1}^{N} \frac{\partial^2}{\partial x_j^2}$, here the differentiation term $\frac{\partial}{\partial x_j}$ is computed by formulation (5). Following the same differentiation, $A(\mathbf{p}) \in \mathbb{C}^{N\times N}$ in Equation (7) and $A(\mathbf{d}, \mathbf{c}) \in \mathbb{C}^{N\times N}$ in Equation (8) are two general matrices respectively correspond to the discrete heterogeneous Darcy flow operator $-\nabla\big(p(x, y)\nabla\big)$ and Convection-Diffusion operator $-\nabla\big(d(x, y)\nabla\big) + c(x, y)\nabla$ with $\nabla := \sum_{j=1}^{N} \frac{\partial}{\partial x_j}$. Within these two general matrices, the $\mathbf{p} \in \mathbb{R}_+^N$, $\mathbf{d} \in \mathbb{R}_+^N$ and $\mathbf{c} \in \mathbb{R}_+^N$ are three real vectors obtained from the discrete permeability $p(x, y)$, diffusion $d(x, y)$ and convection $c(x, y)$ terms. The $\mathbf{b} = [\rho(x_1), \ldots, \rho(x_N)]^T \in \mathbb{R}^N$ is a vector refers to the right-hand side constructed from the discrete source term $\rho(x, y)$, and vector $\mathbf{u} = [u(x_1), \ldots, u(x_N)]^H \in \mathbb{C}^N$ is the discrete solution to be computed.

The elements of fluid permeability $\mathbf{p}$, diffusivity $\mathbf{d}$ or velocity $\mathbf{c}$ can be the same over the whole domain if it is composed by a single material. While for a domain composed by many different materials or for an unknown region, their values become variable or random. Similarly, in some practical applications, the source term $\mathbf{b}$ can also be variable, like the flow starts from a single source location or from multiple locations. In summary, these non-uniform permeability $\mathbf{p}$, diffusivity $\mathbf{d}$, velocity $\mathbf{c}$ and source term $\mathbf{b}$ over the whole domain represent to the diverse heterogeneous cases. Additionally, under the context of dynamic systems, these heterogeneous fluid terms can further change their non-uniform distributions over time. On the other hand, the domain size of the considered fluid equations, such as the value of N with a given

[2] The discrete Laplace operator with Dirichlet BCs leads to a Hermitian or symmetric matrix. While this is not true when applying with other type of classical BCs illustrated in Section 2.1.4. Refer to Figure 18 in Appendix A for an example of the illustration.

discrete step size, may not be fixed as well. Under these dynamic heterogeneous case (further with varying discretisation points N), Equation (6)-(8) can be further extended to a series of linear systems of the form

$$A\mathbf{u}^{(\ell)} = \mathbf{b}^{(\ell)}, \text{ with } A := \nabla^2 \in \mathbb{C}^{N\times N}, \ \ell = 1, 2, ...L, \tag{9}$$

$$A(\mathbf{p}^{(\ell)})\mathbf{u}^{(\ell)} = \mathbf{b}^{(\ell)}, \text{ with } A(\mathbf{p}^{(\ell)}) := -\nabla\big(\mathbf{p}^{(\ell)}\nabla\big), \tag{10}$$

$$A(\mathbf{d}^{(\ell)}, \mathbf{c}^{(\ell)})\mathbf{u}^{(\ell)} = \mathbf{b}^{(\ell)}, \text{ with } A(\mathbf{d}^{(\ell)}, \mathbf{c}^{(\ell)}) := -\nabla\big(\mathbf{d}^{(\ell)}\nabla\big) + \mathbf{c}^{(\ell)}\nabla \in \mathbb{C}^{N\times N}, \tag{11}$$

where, associated with the ℓ-th family from each discrete fluid heterogeneous equations, the value of $N, L \in \mathbb{R}_+$ can be problem-depending. Using Equation (11), an extension of Equation (8), we illustrate the notations meaning. Here, $A(\mathbf{d}^{(\ell)}, \mathbf{c}^{(\ell)}) \in \mathbb{C}^{N\times N}$ is a general matrix of the ℓ-th system, $\mathbf{d}^{(\ell)} \in \mathbb{R}_+^N$, $\mathbf{c}^{(\ell)} \in \mathbb{R}_+^N$ and $\mathbf{b}^{(\ell)} \in \mathbb{R}^N$ are three given vectors. The corresponding solution vector to be computed is $\mathbf{u}^{(\ell)} \in \mathbb{C}^N$. Under dynamic cases, both the coefficient matrix $A(\mathbf{d}^{(\ell)}, \mathbf{c}^{(\ell)})$ and right-hand side $\mathbf{b}^{(\ell)}$ change from one family to the next.

For simplicity and notational convenience, in the rest of this paper we drop heterogeneous permeability parameter $\mathbf{p}$ in Equation (10) and other two heterogeneous parameters $\mathbf{d}, \mathbf{c}$ in Equation (11). Additionally, we drop the superscript $^{(\ell)}$ in $\mathbf{b}^{(\ell)}$ and $\mathbf{u}^{(\ell)}$ whenever we consider solving the current ℓ-th family of linear systems in the entire sequence of families from given fluid equations. We indicate the heterogeneous parameter(s) of fluid equations and the superscript for a family order explicitly when necessary. That is, suppose that the current ℓ-th family of linear systems from given discrete fluid equations to be solved is

$$A\mathbf{u} = \mathbf{b}, \tag{12}$$

where, $A \in \mathbb{C}^{N\times N}$ is the current square nonsingular matrix, $\mathbf{b} \in \mathbb{R}^N$ is the right-hand side, and $\mathbf{u} \in \mathbb{C}^N$ is the solution to be computed.

For a straightforward illustration, we refer readers to Figure 18 in Appendix A for the visualization of small-sale discrete fluid equations with different BCs performed by FFT. Corresponding visualizations realized by FDM are also shown in Figure 19-21 in Appendix A. By comparing Figure 18 to Figure 19-21, it can be observed that discrete PDE performed by FFT exhibit a less sparsity structure. For example, in the case of Dirichlet BCs, the matrix row obtained by FDM has at most 5 non-zero elements, while the matrix row obtained by FFT contains more non-zero elements. Note that the CNN-based preconditioning not only leverages the statistical properties of fluid data but also captures the local sparsity structure of the discrete fluid equations. To enhance the effectiveness of this preconditioning, it is suggested to train with discrete fluid equations realized through FFT, which preserves more data information over the whole domain. This approach allows the NNs to learn more richer information across whole domain, improving its generalization and effectiveness.

2.3 Subspace iterative methods

When solving large linear system such as Equation (12), attractive approaches often involve Krylov subpaces methods [94], like the generalized minimum residual (GMRES) norm method [95]. Krylov subspace methods can achieve fast convergence when working with an effective preconditioner, which can transform the spectra of the original system into another well-conditioned system whose spectra clustered around 1. Generally, the preconditioning is an approximation of the inverse of the original coefficient matrix. Let A and M respectively refers to the coefficient matrix of a given fluid equation and the corresponding preconditioner. Then an effective preconditioner satisfying $M \sim A^{-1}$. The psesudocode of the flexible GMRES (FGMRES) method [93] preconditioned by a non-linear flexible preconditioner in each iteration step is recalled at Algorithm 1. The psesudocode of GMRES can be simply obtained by removing Step 4 and $\mathbf{z}_j$ of the Algorithm 1, and changing its Step 5 into $\mathbf{w}_j := A\mathbf{v}_j$ (j refers to the iteration step). Here the flexible Arnoldi procedure is realized by the modified Gram-Schmidt (MGS) orthogonalization process. The Krylov basis $\text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_j\}$ is orthonormal to each other. For the preconditioned Krylov basis $\text{span}\{\mathbf{z}_1, \ldots, \mathbf{z}_j\}$ computed from Step 4, its condition number is depended on the preconditioning quality.

3 Convolution neural operator preconditioning approach

To construct a powerful preconditioner that approximates A^{-1} is as challenging as solving the linear system directly. Besides, for the parametric linear systems such as Equation (10) or Equation (11), a new

Algorithm 1 *Flexible GMRES (FGMRES) [93, Algorithm 9.6]:*

1: Compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{u}_0$, $\beta = \|\mathbf{r}_0\|^2$, and $\mathbf{v}_1 = \mathbf{r}_0/\beta$
/ Flexible Arnoldi procedure */*
2: **for** $j = 1, \ldots, m$ **do**
3: Choose a (possibly non-linear) preconditioning operator M_j (M_j is an approximation of A^{-1})
4: Compute $\mathbf{z}_j := M_j(\mathbf{v}_j)$
5: Compute $\mathbf{w}_j := A\mathbf{z}_j$
/ Modified Gram-Schmidt procedure */*
6: **for** $i = 1, \ldots, j$ **do**
7: $h_{i,j} := \langle \mathbf{w}_j, \mathbf{v}_i \rangle$
8: $\mathbf{w}_j := \mathbf{w}_j - h_{i,j}\mathbf{v}_i$
9: **end for**
10: Compute $h_{j+1,j} = \|\mathbf{w}_j\|_2$ and $\mathbf{v}_{j+1} = \mathbf{w}_j/h_{j+1,j}$
11: Save $V_{j+1} := [\mathbf{v}_1, \ldots, \mathbf{v}_{j+1}]$, $Z_j := [\mathbf{z}_1, \ldots, \mathbf{z}_j]$, and an upper Hessenberg matrix $\underline{H}_j = \{h_{i,j}\}_{1 \le i \le j+1; 1 \le j \le m}$
12: **end for**
13: Compute $\mathbf{y}_m = \text{argmin}_{\mathbf{y}} \|\beta \mathbf{e}_1 - \underline{H}_m \mathbf{y}\|$, and $\mathbf{u}_m = \mathbf{u}_0 + Z_m \mathbf{y}_m$
14: **if** the stopping criterion is met **then**
15: Return $\mathbf{u}_m$ for approximation of the solution, and stop the algorithm
16: **else**
17: Restart with $\mathbf{u}_0 = \mathbf{u}_m$ and go to Step 1
18: **end if**

algebra preconditioner is necessary when the involved heterogeneous parameter(s) changes, like the change in permeability $\mathbf{p}$, diffusion $\mathbf{d}$, or convection $\mathbf{c}$ and/or the discretisation points N. This let its numerical simulations with purely algebra preconditioning becomes even more challenging. Instead of constructing an algebra preconditioner, this work aims to learning a general convolutional neural networks (CNNs) preconditioning operator. This preconditioning operator is intended to function as a non-linear flexible preconditioner in FGMRES for accelerating the solution of Equation (12) which represents three types of discrete fluid equations. The goal of this CNN-based preconditioning aligns with the motivation behind neural operator learning [59, 14]. It focus on learning a general function that captures the relationships within PDEs, rather than directly solving the PDEs with neural networks (NNs) like PINNs [69] do. To achieve this goal, we investigate the use of CNNs with the U-Net architecture [90], which has been widely applied for the fluid [21, 34] and wave [102, 4] simulations. This study aims to evaluate whether the trained inference from the networks can serve as an effective preconditioner for the considered heterogeneous fluid equations with varying BCs, even when all associated physical and numerical parameters vary.

Assume the CNN-based preconditioning operator is denoted as $\mathcal{F}_\theta$ with trainable parameters θ (i.e., the trainable weights and biases of the NNs). We focus on learning an operator $\mathcal{F}_\theta$

$$\mathcal{F}_\theta : \mathcal{P} \longrightarrow \mathcal{U}, \tag{13}$$

with input $\mathbf{b}^{(\ell)} \in \mathcal{P},\ \ell = 1, \ldots, L$ (here L denotes the size of the training datasets, similar to the meaning shown in Equation (9)-(11)) and output solution $\mathbf{u}_\theta^{(\ell)} \in \mathcal{U}$ returned from CNNs with θ. Our target is to find the learning operator such that

$$\mathcal{F}_\theta(\mathbf{b}^{(\ell)}) \sim \mathbf{u}_\theta^{(\ell)}, \text{ with } i = 1, \ldots, L, \tag{14}$$

which means the learned operator $\mathcal{F}_\theta$ approximates to the inverse of the coefficient matrix A^{-1} since we have $\mathbf{u} = A^{-1}(\mathbf{b})$ from Equation (12). Because of this, the learned operator $\mathcal{F}_\theta$ can be used as a non-linear preconditioner in the FGMRES method, that is to change the M_j in Step 4 of Algorithm 1 into the learned operator $\mathcal{F}_\theta$ (i.e., $M_j(\mathbf{v}_j) \rightarrow \mathcal{F}_\theta(\mathbf{v}_j)$). Since there is no information about the data structure of the Krylov basis $\mathbf{v}_j$, the neural operator is trained with randomly generated datasets, and the value of L is set by ourself. Refer to [112, 21] for more examples of using the random dataset for the SciML training. Given our purpose is to learn an operator to be used as preconditioner rather than to learn a NNs solver, the required approximation accuracy in formula (14) is not high. This reduce the training cost and also naturally circumvents the well-known limited attainable accuracy issue of SciML. In short, the trained CNN-based

preconditioning operator shown in formula (14) may not act as an effective CNNs solver but can be used an effective preconditioner for accelerating the solution of the discrete heterogeneous fluid Equation (12).

Note that in the training phases for various heterogeneous fluid equations, it is necessary to deal with networks with multiple non-zero inputs associated with different physical parameter(s). For example, except for the source term $\mathbf{b}^{(\ell)}$, the diffusion term $\mathbf{d}^{(\ell)}$ and the convection term $\mathbf{c}^{(\ell)}$ are also necessary as the inputs for training an effective operator preconditioner for the discrete Convection-Diffusion equations (11). To deal with multiple inputs, we apply the concatenation to them at the input neural network layer. Besides, given each point of the domain corresponds to an element of the Krylov basis, we suggest the neural network should see data over the whole domain for learning an effective preconditioner. Moreover, except for applying the trained operator as a flexible preconditioner to the FGMRES method considered in this paper, it can also be applied to other iterative methods. For example, refer to [109, Chapter 5] for using a trained U-Net inference as the non-liner preconditioner for the FGMRES and the FFOM methods.

3.1 U-Net architecture

The neural network architecture considered in this work is depicted in Figure 3, which is based on the U-Net architecture [90]. It is named for its distinctive U-shaped architecture, consisting of a contracting path and a more or less symmetric expanding path. The U-Net is a type of convolutional neural network (CNN) specifically developed for biomedical image segmentation tasks in computer vision. It excels in applications requiring the segmentation of images to identify specific objects or regions. Furthermore, due to its convolutional nature, the architecture supports multiple input channels, allowing for the integration of diverse input data. Since its publication, it has been widely used by the SciML-wave [4, 102] and SciML-CFD [21, 34] communities. Similar to [90, Section 2], the U-Net architecture depicted in Figure 3 consists of 23 convolutional layers in total, and it is composed of the following key components:

1. **Input Layer:** The input layer of the network typically consists of a convolutional layer that produces the segmentation map. This map mirrors the full context available in the input images (selected information used for learning) to the seamless segmentation of arbitrarily large images, which is then passed to the subsequent convolutional encoder-decoder network. This ensures both the input images and their corresponding segmentation maps are used to train the network.

2. **Contracting Path (Encoder):** The left top-to-bottom parts of the U-shape is called the contracting path or encoder. It's made up of a series of convolution and pooling layers. These layers reduce the spatial dimensions of the input image while increasing the number of feature channels. This helps the network to capture different levels of information and extract features at different scales.

3. **Bottleneck:** At the bottom of the U-shape there's a bottleneck layer. This is where the network captures the most abstract and consolidated features from the input image.

4. **Expanding Path (Decoder):** The right bottom-to-top parts of the U-shape is called the expanding path or decoder. It's responsible for gradually upsampling the features back to the original image size. Each step in the decoder involves upsampling the feature map and combining it with the feature map from the corresponding layer in the contracting path through a process called skip connections.

5. **Skip Connections:** Skip connections are a critical feature of the U-Net architecture, setting it apart from a standard encoder-decoder network. They connect the feature maps from the contracting path to the corresponding layers in the expansive path. These connections help to preserve fine-grained spatial information that can be lost in the downsampling process. By fusing information from multiple scales, the network can better localize objects and produce more accurate segmentation results.

6. **Final Layer:** The final layer of the network typically consists of a convolutional layer that produces the segmentation map. This map assigns a label to each pixel in the image, indicating which object or category it belongs to, while resizing arbitrarily large images back to the original dimensions of the input images.

The strength of the U-Net architecture lies in its ability to handle both contextual information (via the contraction path) and precise localization (via the expansion path). The skip connections play an important role in achieving this balance. U-Net has been widely used in medical image analysis, where tasks such as

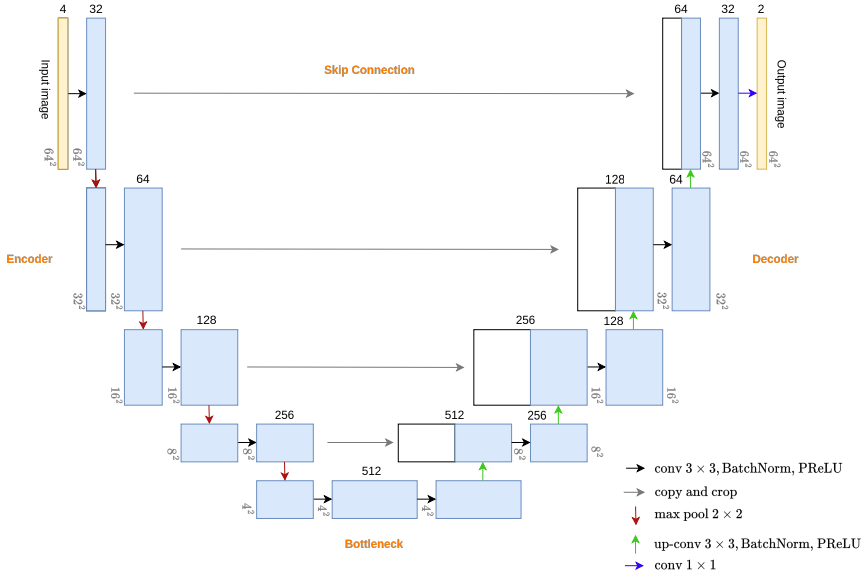


Figure 3: U-Net architecture with 4 depth (example for 4×4 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent feature maps copied from the contracting path, which are then cropped and concatenated with the feature maps obtained from the expanding path. The arrows denote the different convolutional operations.

segmenting organs or tumors from medical scans are critical. Its flexibility and effectiveness in handling segmentation tasks has made it a popular choice for various image analysis applications outside the medical field. Since a matrix or vector can also be viewed as a type of image, this architecture can also be used to learn a highly non-linear preconditioner for the matrix or PDE operator.

3.2 Loss-function

Within a discrete 2D domain Ω defined for Equation (12), the Loss-function of the CNN-based preconditioning operator is defined as the following relative mean squared error (RMSE) that related to the relative residual of the linear system:

$$\min_{\theta} \frac{\|A\mathbf{u}_\theta - \boldsymbol{b}\|_2^2}{\|\boldsymbol{b}\|_2^2} \tag{15}$$

where A relates to the coefficient matrix of the discrete fluid equations we considering for training. For example, A relates to both the heterogeneous diffusion term $\mathbf{d}$ and the convection term $\mathbf{c}$ if we consider training a CNN-based preconditioning operator for the Convection-Diffusion equations (11). Thus the training dataset, in this case, include the heterogeneous $\mathbf{d}$, $\mathbf{c}$ terms as well as the heterogeneous source term $\mathbf{b}$. Given no true solution: $\mathbf{u} = A(\mathbf{c})^{-1}\mathbf{b}$ is required in the Loss-function (15), the training process is under the context of unsupervised learning. With this definition, the training dataset is trained without data-normalization that commonly used in the training process of NNs.

3.3 Training neural operator preconditioners for heterogeneous fluid PDEs

This section aims to train three types of general neural operator preconditioners, $\mathcal{F}_\theta$, to accelerate the solution of a series of heterogeneous or parametric fluid equations described in Equation (9)-(11). To be specific, Section 3.3.1-3.3.3 respectively details the training of these three type of neural operator preconditioners for the discrete fluid equations defined on a 2D domain $[[-20, 20], [-20, 20]]$, discretized into a 64×64 grid, resulting in $N = n^2 = 64^2 = 4096$ discretisation points. Note that although our neural operator preconditioners are designed to handle these heterogeneous fluid equations with varying BCs

illustrated in Section (2.1.4), it is sufficient to train these preconditioners on small-scale problems with the simplest zero-Dirichlet BCs, where the Dirichlet BCs is specified as $\phi(x, y) = 0$ in Equation (4).

These preconditioners are also designed to exhibit strong generalizability by effectively addressing novel testing cases that the neural operators have not encountered during the training process. To train these preconditioning operators, we employ the CNNs with U-Net architecture. Based on its architecture described in Figure 3 of Section 3.1, we construct a 2D U-Net operator with 4 `depth` blocks training with the PReLU activation as well as batch normalization. These blocks are paired with 2-d convolution and de-convolution networks respectively realized by `torch.nn.Conv2d` and `torch.nn.ConvTranspose2d` with the size of the convolving kernel equals to 8. The effective preconditioning operator, $\mathcal{F}_\theta$, is designed to approximate A^{-1}. After training, its inputs include the Krylov basis that without relying on any special data structure. Based on this and the purpose to learn general information, we train the U-Net operator $\mathcal{F}_\theta$ with randomly generated datasets. For instance, for the source term considered in these three fluid equations, we generate random datasets for the source term $\mathbf{b} = x + yj \in \mathbb{C}^{n^2}$, where both $x \sim \mathcal{N}(0, 1)$ and $y \sim \mathcal{N}(0, 1)$, following a standard normal distribution.

3.3.1 Poisson operator

Building on earlier discussions, we demonstrate the process of training the U-Net preconditioning operator for a series of parametric Poisson equations (9). As depicted in Figure 4, the training process begins with the input, which consists of a randomly generated heterogeneous source term $\mathbf{b} \in \mathbb{C}^{n^2}$ sampled from a standard normal distribution $\mathcal{N}(0, 1)$. The U-Net architecture applied to Poisson equations is shown in Figure 3. Whereas in this case, the input layer of Figure 3 has two input channels, corresponding to both the real and image components of the complex source term $\mathbf{b}$. The output is the approximated solution $\mathbf{u}_\theta \in \mathbb{C}^{n^2}$ obtained in lower attainable precision. These input and output, along with the discrete Poisson operator A, are combined into the Loss-function. Then, the optimization techniques such as Stochastic Gradient Descent (SGD) are applied to minimize the Loss-function, recursively updating the trainable parameters θ.

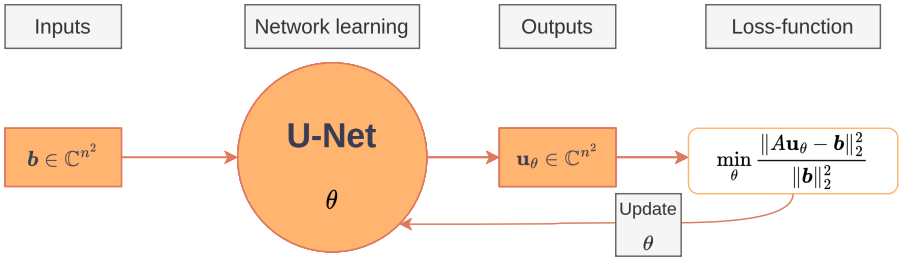


Figure 4: U-Net with Dirichlet BCs concept applied to the Poisson operator on 2D.

Describe Figure 4 in a mathematical way as the operator map shown in formula (13), we have

$$\mathcal{F}_\theta : (\mathbf{b}) \longrightarrow \mathbf{u}_\theta, \tag{16}$$

where $\mathcal{F}_\theta$ is the trained U-Net operator to accelerate the solution of the 2D parametric Poisson equations (9).

3.3.2 Darcy flow operator

According to the discrete heterogeneous Darcy flows described in Equation (10) and the framework discussed in Section 2.1.2, Figure 5 illustrates the training process for developing a CNN-based preconditioning operator specific to Darcy flows. Compared to Figure 4, which is tailored to Poisson equations, an additional input channel is introduced in Figure 5 to incorporate the heterogeneous permeability term $\mathbf{p} \in \mathbb{R}_+^{n^2}$. This additional input allows the neural network to learn from the spatially varying properties of permeability, crucial for modeling heterogeneous Darcy flows. The U-Net architecture depicted in Figure 3 is also applied to Darcy flows. While in this setup, the input layer in Figure 3 is

configured with three input channels, corresponding to the components of the complex source $\mathbf{b}$ and the real permeability $\mathbf{p}$. Follow the random setting for generating the training dataset, the permeability term $\mathbf{p}$ is randomly sampled from an uniform distribution over the interval $[1, 2]$, denoted as $\mathcal{U}(1, 2)$.

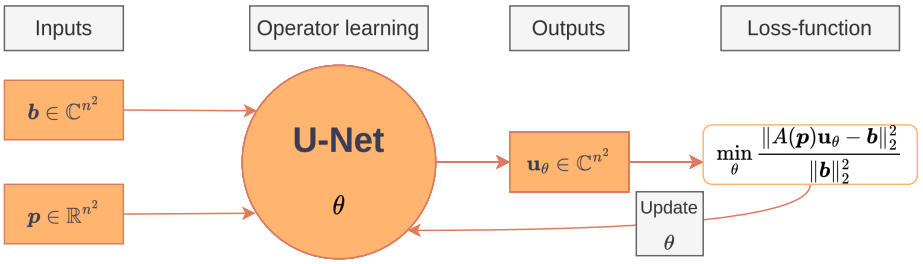


Figure 5: U-Net with Dirichlet BCs concept applied to the heterogeneous Darcy flow operator on 2D.

Similarly, describe Figure 5 in a mathematical way, we have the preconditioning operator

$$\mathcal{F}_\theta : ([\mathbf{b},\ \mathbf{p}]) \longrightarrow \mathbf{u}_\theta \tag{17}$$

to accelerate the solution of the 2D parametric Darcy flows described in Equation (10). Here, the notation $[*, \ldots, *]$ represents concatenates the multiple inputs before passing them through the input layer of the convolutional neural network.

3.3.3 Convection-Diffusion operator

According to the discrete heterogeneous Convection-Diffusion equations illustrated in Equation (11) and the discussions shown in earlier sections, we describe the processes for training CNN-based preconditioning operator for the Convection-Diffusion equations in Figure 6. In contrast to Figure 4, which involves only a single input, two additional variables are introduced in Figure 6. These extra inputs correspond to the heterogeneous diffusion term $\mathbf{d} \in \mathbb{R}_+^{n^2}$ and the convection term $\mathbf{c} \in \mathbb{R}_+^{n^2}$. Incorporating these extra terms allows the CNN-based preconditioning operator to adapt to the varying physical properties of the heterogeneous Convection-Diffusion equations. The U-Net architecture depicted in Figure 3 is directly applied to this Convection-Diffusion case without any modification. For training the operator, both $\mathbf{d}$ and $\mathbf{c}$ are generated randomly, each sampled from an uniform distribution $\mathcal{U}(1, 2)$.

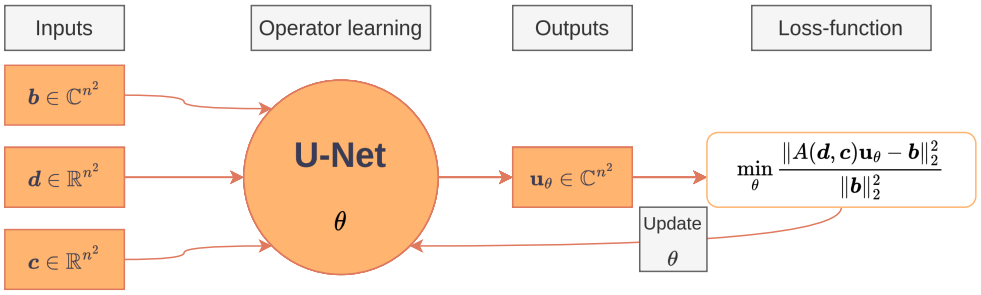


Figure 6: U-Net with Dirichlet BCs concept applied to the heterogeneous Convection-Diffusion operator on 2D.

Similarly, describe Figure 6 in a mathematical way, we have the preconditioning operator

$$\mathcal{F}_\theta : ([\mathbf{b},\ \mathbf{d},\ \mathbf{c}]) \longrightarrow \mathbf{u}_\theta \tag{18}$$

to accelerate the solution of the 2D parametric Convection-Diffusion equations (11).

3.4 Training details and some remarks

To train the 2D CNN-based preconditioning operators described in formulas (16)-(18), 10000 random samples are generated for each training dataset(s), including $\mathbf{b}$, $\mathbf{p}$, or $\mathbf{d}$ and $\mathbf{c}$, resulting in $L = 10000$ in operator learning formula (14). The training is conducted using the Adam optimizer [56], with the maximum number of epochs (denoted as `max_epoch`) set to 500, i.e., `max_epoch` $= 500$. The learning rate is initialized at 0.001 and gradually decreases to a minimum value of 10^{-5}. Refer to Table 1-3 for more details about the fluid operator parameters, the model and training hyper-parameters of U-Net model designed for fluid operators on a 2D domain with zero-Dirichlet BCs. This U-Net preconditioning model, configured with a `depth` of 4, is trained under the single precision (i.e., `float32` and `complex64`) on 2 NVIDIA V100 GPUs located a Sirocco node of the PlaFRIM cluster [3]. This node also includes 2x 16-core Intel Skylake CPUs and 384 GB of memory. With these training settings, the U-Net model, comprising 831 K trainable parameters, is efficiently trained with these training resources. The network architecture and the fluid operator training utilities are implemented using the `Pytorch` and `Pytorch_lightning` libraries. The training time required for this U-Net preconditioning model for the three types of discrete fluid equations is summarized in Table 4.

Table 1: Fluid Operator in 2D and Fast Fourier Transforms (FFT) parameters

Parameter	**Value**
Limits of the Domain	$[[-20, 20], [-20, 20]]$
Discretisation points n in Domain	$[64, 64]$
Axes for which FFT is applied	$[-2, -1]$

Table 2: U-Net Model Hyper-parameters

Parameter	**Value**
Input Channels	2 or 3 or 4
Output Channels	2
Activation Function	PReLU
Depth	4
Channels per Layer	32
Channels per State	32
Convolving kernel size, Stride, Padding	8, 2, 3
Batch normalization	True
Bias in concolutional layers	True

Table 3: Training Hyper-parameters

Parameter	**Value**
Optimizer	Adam
Batch Size	32
Gradient Clipping value	1
Learning Rate (LR)	10^{-3}
Minimal LR	10^{-5}
Seed for Random Number	42

Table 4: Training time for learning the three Fluid operators

Fluid operator	**Trainable time**
Poisson equations	1.28 hr
Darcy flows	1.33 hr
Convection-Diffusion equations	1.44 hr

Since each point in the computational domain corresponds to an element of the Krylov basis, it is essential for the neural network to be exposed to data spanning the entire domain to learn an effective preconditioner. Although random datasets are used for training, it is recommended to select training datasets with a broader range of values to increase the diversity of the training data. For instance, datasets following a standard normal distribution, i.e., $\mathcal{N}(0, 1)$, capture a broader range of data compared to those adhering to an uniform distribution over the interval $[-1, 1]$, i.e., $\mathcal{U}(-1, 1)$. Training with diverse datasets not only enriches the learning process but also significantly improves the network's generalizability to unseen scenarios.

[3] https://www.plafrim.fr

3.5 FGMRES preconditioned by the trained convolution neural operator

Based on previous discussions in Section 3, we have trained three general CNN-based preconditioning operators, $\mathcal{F}_\theta$, as detailed in formulas (16)-(18), tailored for solving three types of discrete fluid equations. This section elaborates on how these trained preconditioning operators are incorporated into the FGMRES framework to accelerate the solution process. Using the pseudocode of the original FGMRES shown in Algorithm 1 as a foundation, the CNN-based FGMRES algorithm preconditioned with the trained operators modifies Step 4 as follows:

- For the discrete Poisson equations (9), Step 4 becomes:

$$\mathbf{z}_j := \mathcal{F}_\theta(\mathbf{v}_j), \tag{19}$$

  where $\mathcal{F}_\theta$ approximate A^{-1} is the preconditioning operator trained with formula (16), $\mathbf{v}_j$ is the Krylov basis and $\mathbf{z}_j$ is the preconditioned Krylov basis.

- For the discrete Darcy flows (10), Step 4 becomes:

$$\mathbf{z}_j := \mathcal{F}_\theta(\mathbf{v}_j, \mathbf{p}), \tag{20}$$

  where $\mathcal{F}_\theta$ is the preconditioning operator trained with formula (17), $\mathbf{p}$ represents the heterogeneous permeability term.

- For the discrete Convection-Diffusion equations (11), Step 4 becomes:

$$\mathbf{z}_j := \mathcal{F}_\theta(\mathbf{v}_j, \mathbf{d}, \mathbf{c}), \tag{21}$$

  where $\mathcal{F}_\theta$ is the preconditioning operator trained with formula (18), $\mathbf{d}$ and $\mathbf{c}$ are the heterogeneous diffusion and convection terms.

Compare to original FGMRES, the CNN-based preconditioning steps shown in Equation (19)-(21) are performed in `float32`, consistent with the precision used during the training processes. The other remaining operations in the FGMRES algorithm with CNN-based preconditioning are conducted in `float64`. Thus this CNN-based FGMRES algorithm is realized under the mixed-precision calculations.

4 Numerical results

Section 3 detailed the training process for CNN-based preconditioning operators using an U-Net architecture with 4 `depth`. It also demonstrated the utility of the trained network inferences for non-linear preconditioning within the FGMRES framework. The training was conducted on a 2D domain $[[-20, 20], [-20, 20]]$, discretized with $N = n^2 = 64 \times 64 = 4096$ points. In this section, we focus on illustrating different numerical features of these trained U-Net preconditioners, including testing similar but unseen scenarios and checking network generalizability from various aspects. Specifically, in these testing phases, numerical experiments are carried out with different parametric fluid Equation (9)-(11) that the neural operators never seen before. Without special notes, zero-Dirichlet boundary conditions (BCs) is applied to these fluid equations, and the fluid systems used in these processes own the same N size as used in the training process. We display results of FGMRES (i.e., GMRES preconditioned by this trained U-Net preconditioner with 4 `depth`). The relevant results of GMRES is presented as well for the comparison. A classical stopping criterion for the numerical linear algebra solvers is based on backward error analysis and consists of stopping the iteration when

$$\eta_b = \frac{\|A\mathbf{u} - \boldsymbol{b}\|}{\|\boldsymbol{b}\|} \leq \varepsilon. \tag{22}$$

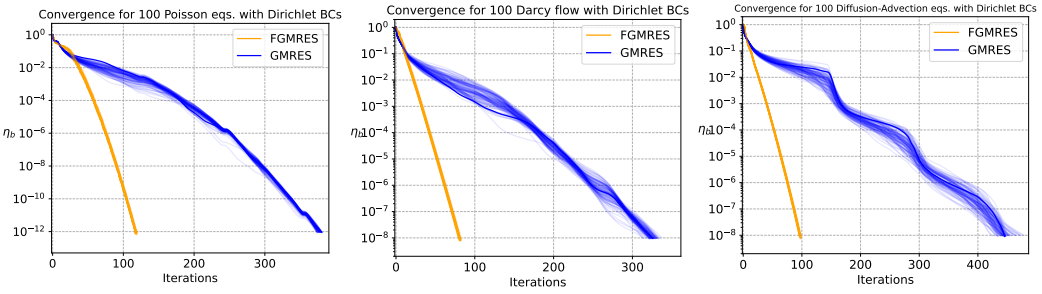
Here we set $\varepsilon = 10^{-12}$ by default for the solution of the parametric Poisson equations (9), and $\varepsilon = 10^{-8}$ for the heterogeneous Darcy flows (10) and the heterogeneous Convection-Diffusion equations (11). Besides during this testing section, the source term is defined as a real vector, represented in the form $\mathbf{b} = x + 0j \in \mathbb{C}^{64^2}$ with $x \in \mathbb{R}^{64^2}$. The performances of the involved algorithms are evaluated in terms of the number of iterations, denoted as its, as well as the execution time, denoted as ET, required to

converge. Given the full GMRES without restart could be quite time-consuming for solving larger problem, we consider GMRES (also FGMRES) with restarts. Without special notes, the number of restarts is set to be 10, and the maximum dimension of the Krylov search space m shown in Algorithm 1 within each of the restarts is 512. Thus the whole maximum iteration (denoted as $maxIts$) is 5120. We stop the algorithm when satisfying formula (22) or when $its = maxIts$. The symbol * denotes the algorithm diverges or it is unable to reach target accuracy within $maxIts$. Besides, instead of showing results up to $maxIts$, we may cut off our figures at some early iterations to have a clear viewer for the better comparison.

Recall that this 2D U-Net inference was training on GPUs whereas the classical numerical linear algebra solvers have been historically implemented on CPUs. In the following subsections, the GMRES and FGMRES solvers are implemented in `Python` prototype, supporting both CPU and GPU backends for computations with `numpy` and `cupy` Python libraries. Without special notes, those two involved solvers in each of the subsections are running on the same GPUs devices for a fair comparison.

4.1 Testing results of U-Net with parameters saved at different epoch

In this section, we check the effect of using varying parameters (i.e., the trainable weights and biases saved in different epoch) for the 2D U-Net trained with `max_epoch` $= 500$. Within this testing phases, we test the fluid equations (9)-(11) with the source term in form $\mathbf{b} = x + 0j \in \mathbb{C}^{64^2}$ with $x \in \mathbb{R}^{64^2}$ satisfying a standard normal distribution $\mathcal{N}(0, 1)$ on the domain size 64×64. The numerical results of GMRES and FGMRES are presented in Table 5 and Figure 7. Comparing the results of these two solvers, it is evident that FGMRES, with the aid of the trained CNN-based preconditioning operators, achieves the stopping criteria with significantly fewer iterations its and reduced elapsed time ET compared to standard GMRES.



Poisson eqs.: `epoch` $= 467$, `val_loss` $= 0.05100$

Darcy flows: `epoch` $= 401$, `val_loss` $= 0.15009$

Convection-Diffusion eqs.: `epoch` $= 483$, `val_loss` $= 0.09784$

Figure 7: Compare the relative residual history of GMRES and FGMRES (preconditioned by the trained 2D U-Net with 4 `depth`) of different fluid equations with zero-Dirichlet boundary condition.

For the following sections, we exclusively evaluate the trained CNN-based preconditioning operators using the parameter configurations saved at the `epoch` highlighted in bold, indicating the lowest value of the validation loss (denoted as `val_loss`), for each type of fluid equation presented in Table 5.

4.2 Network generalizability

The following sections delve into the generalization capabilities of the trained CNN-based preconditioning operators from different aspects. To evaluate the network's generalizability, we test its performance under various heterogeneous parameters present in the fluid equations (9)-(11), extending beyond the conditions used during training. This involves varying the source terms and boundary conditions (BCs) for the three types of fluid equations as detailed in Section 4.2.1 and Section 4.2.2, respectively. Additionally, we examine changes in permeability for Darcy flows and diffusion for Convection-Diffusion equations as discussed in Section 4.2.3, as well as variations in the heterogeneous convection term for Convection-Diffusion equations, covered in Section 4.2.4. Finally, we assess the effect of altering the domain size for these fluid equations in Section 4.2.5.

Table 5: Results of solving different fluid equations (simplified as eqs.) with Dirichlet boundary condition solved by FGMRES preconditioned by the trained U-Net with different parameters saved different `epoch`.

# `fluid eqs.`	`epoch`	`val_loss`	L	(its_{max}, its_{min})	ET
Poisson eqs.: ($\varepsilon = 10^{-12}$) FGMRES:	496	0.05109	100	(123, 119)	198.4347s
			1000	(126, 119)	2857.8662s
	476	0.05105	100	(328, 118)	312.3358s
			1000	(334, 118)	3863.7267s
	467	0.05100	100	(122, 118)	193.3075s
			1000	(198, 118)	2715.7512s
GMRES:			100	(382, 374)	3518.5219s
Darcy flow ($\varepsilon = 10^{-8}$) FGMRES:	449	0.15024	100	(89, 81)	116.4421s
			1000	(91, 79)	1644.4513s
	434	0.15047	100	(84, 80)	106.9640s
			1000	(90, 79)	1477.5602s
	401	0.15009	100	(84, 80)	106.6865s
			1000	(88, 80)	1493.3070s
GMRES:			100	(338, 314)	2149.1766s
Convection-Diffusion eqs. ($\varepsilon = 10^{-8}$) FGMRES:	484	0.09791	100	(100, 97)	141.7242s
			1000	(101, 96)	2027.8065s
	483	0.09784	100	(100, 96)	150.8635s
			1000	(101, 96)	2046.7538s
	471	0.09801	100	(100, 96)	140.9959s
			1000	(102, 96)	2001.6087s
GMRES:			100	(481, 425)	4021.3439s

4.2.1 In terms of varying the source term

During the training process, the random source term $\mathbf{b} = x + yj \in \mathbb{C}^{64^2}$ with $x \neq 0, y \neq 0$ and satisfy standard normal distribution $\mathcal{N}(0, 1)$ on domain 64×64. In this section, we study the network generalizability of the trained CNN-based preconditioning operators by varying source term $\mathbf{b} = x + 0j$ with different settings for x. Specifically, we consider testing $\mathbf{b}$ satisfying an uniform distribution over the interval $[-1, 1]$, denoted as $\mathcal{U}(-1, -1)$; 1 `dirac distr.` [4]; 4 `dirac distr.` [5]; and other normal distribution, such as $\mathcal{N}(1, 1)$ with a mean and variance of 1, and $\mathcal{N}(0, 2)$ with a mean of 0 and a variance of 2. Furthermore, based on the control variables method, the setting for the BCs, permeability $\mathbf{p} \in \mathbb{R}^{n^2}$, diffusion $\mathbf{d} \in \mathbb{R}^{n^2}$ and convection $\mathbf{c} \in \mathbb{R}^{n^2}$ and domain Ω are the same as the ones used in the training process. With these different settings for $\mathbf{b}$ on domain 64×64, Figure 8 (a)-(c) presents the convergence histories of solving the three types of fluid equations (9)-(11) with $L = 1, 5,$ or 100. Corresponding numerical results of GMRES and FGMRES are reported in Table 6, from which we have similar observations discussed in Section 4.1. That is thanks to the applying of CNN-based operator preconditioners, the FGMRES can significant reducing the its and ET when compared to GMRES. Additionally, these results also show that changing the setting of the source term does not effect the performance of these trained CNN-based preconditioners. The real part of the solutions $\mathbf{u}$ for the three types of fluid equations, as solved by FGMRES under 1 `dirac distr.` and 4 `dirac distr.` settings for the source terms $\mathbf{b}$, are respectively visualized in the upper and lower plots of Figure 9, which also exhibits the location of the source terms.

4.2.2 In terms of varying the boundary conditions

This section evaluates the generalizability of the trained CNN-based preconditioners by changing boundary conditions (BCs) applied to the three types of fluid equations (9)-(11). Recall that these CNN-based preconditioners were trained under zero-Dirichlet BCs context. Here, we test these fluid

[4] one element of $\mathbf{b}$ equals to 1 in the grid position (32, 32) and others equal to zeros elsewhere

[5] four elements of $\mathbf{b}$ equal to 1 in the grid position (20, 20), (20, 44), (44, 20), (44, 44) and others equal to zeros elsewhere

Table 6: Numerical results of in terms of its or (its_{max}, its_{min}) and ET for Section 4.2.1 for solving a series of fluid equations (9)-(11) with zero-Dirichlet BCs and with $L = 1$, $L = 5$, $L = 100$, or $L = 1000$.

# `fluid eqs.`	# $\mathbf{b} \in \mathbb{C}^{64^2}$	Method	L	its or (its_{max}, its_{min})	ET
Poisson eqs. with $\varepsilon = 10^{-12}$ epoch=467 val_loss=0.05100	1 `dirac`	GMRES	1	344	10.2736s
		FGMRES	1	110	1.7600s
	4 `dirac`	GMRES	1	342	9.5586s
		FGMRES	1	109	1.6670s
	$\mathcal{N}(0,1)$	GMRES	5	(382, 374)	130.6876s
		FGMRES	5	(120, 118)	11.4340s
		FGMRES	100 / 1000	(122, 118) / (198, 118)	193.3075s / 2715.7512s
	$\mathcal{N}(1,1)$	GMRES	5	(532, 517)	241.6918s
		FGMRES	5	(116, 116)	10.8798s
		FGMRES	100 / 1000	(117, 116) / (117, 115)	190.9784s / 2626.5332s
	$\mathcal{N}(0,2)$	GMRES	5	(380, 379)	128.9121s
		FGMRES	5	(120, 118)	11.2273s
		FGMRES	100 / 1000	(123, 118) / (198, 118)	192.7416s / 2770.7412s
	$\mathcal{U}(-1,-1)$	GMRES	5	(381, 378)	134.2098s
		FGMRES	5	(120, 119)	11.9112s
		FGMRES	100 / 1000	(196, 118) / (198, 118)	213.5582s / 2746.9780s
Darcy flow with $\varepsilon = 10^{-8}$ epoch=484 val_loss=0.17501	1 `dirac`	GMRES	5	(325, 304)	102.7121s
		FGMRES	5	(76, 74)	6.6270s
		FGMRES	100 / 1000	(79, 71) / (81, 71)	89.8052s / 1339.9521s
	4 `dirac`	GMRES	5	(323, 303)	202.9047s
		FGMRES	5	(76, 72)	7.7406s
		FGMRES	100 / 1000	(77, 72) / (84, 71)	101.8716s / 882.2990s
	$\mathcal{N}(0,1)$	GMRES	5	(331, 323)	106.5965s
		FGMRES	5	(84, 81)	7.2482s
		FGMRES	100 / 1000	(84, 80) / (88, 80)	106.6865s / 1493.3070s
	$\mathcal{N}(1,1)$	GMRES	5	(339, 335)	255.4367s
		FGMRES	5	(81, 79)	8.3139s
		FGMRES	100 / 1000	(82, 79) / (83, 78)	147.5519s / 973.8013s
	$\mathcal{N}(0,2)$	GMRES	5	(335, 322)	116.6252s
		FGMRES	5	(82, 81)	7.3133s
		FGMRES	100 / 1000	(85, 80) / (92, 79)	110.1259s / 1579.4930s
	$\mathcal{U}(-1,-1)$	GMRES	5	(327, 322)	105.6639s
		FGMRES	5	(83, 81)	7.2214s
		FGMRES	100 / 1000	(85, 80) / (88, 80)	151.4606s / 1560.9539s
Convection-Diffusion eqs. with $\varepsilon = 10^{-8}$ epoch=483 val_loss=0.09784	1 `dirac`	GMRES	5	(444, 410)	171.3612s
		FGMRES	5	(85, 83)	7.3855s
		FGMRES	100 / 1000	(86, 83) / (87, 82)	106.8275s / 1672.3916s
	4 `dirac`	GMRES	5	(428, 403)	161.7185s
		FGMRES	5	(83, 82)	7.1995s
		FGMRES	100 / 1000	(85, 81) / (86, 81)	104.0529s / 1223.1725s
	$\mathcal{N}(0,1)$	GMRES	5	(460, 444)	187.2677s
		FGMRES	5	(99, 98)	9.2856s
		FGMRES	100 / 1000	(100, 96) / (102, 96)	140.9959s / 2001.6087s
	$\mathcal{N}(1,1)$	GMRES	5	(479, 455)	203.9701s
		FGMRES	5	(98, 96)	8.8070s
		FGMRES	100 / 1000	(99, 94) / (99, 94)	134.8118s / 1314.6095s
	$\mathcal{N}(0,2)$	GMRES	5	(459, 431)	186.6549s
		FGMRES	5	(99, 97)	8.9579s
		FGMRES	100 / 1000	(100, 96) / (101, 96)	137.0601s / 1581.9260s
	$\mathcal{U}(-1,-1)$	GMRES	5	(473, 442)	195.1171s
		FGMRES	5	(98, 97)	8.8451s
		FGMRES	100 / 1000	(100, 96) / (101, 96)	140.4030s / 1382.0640s

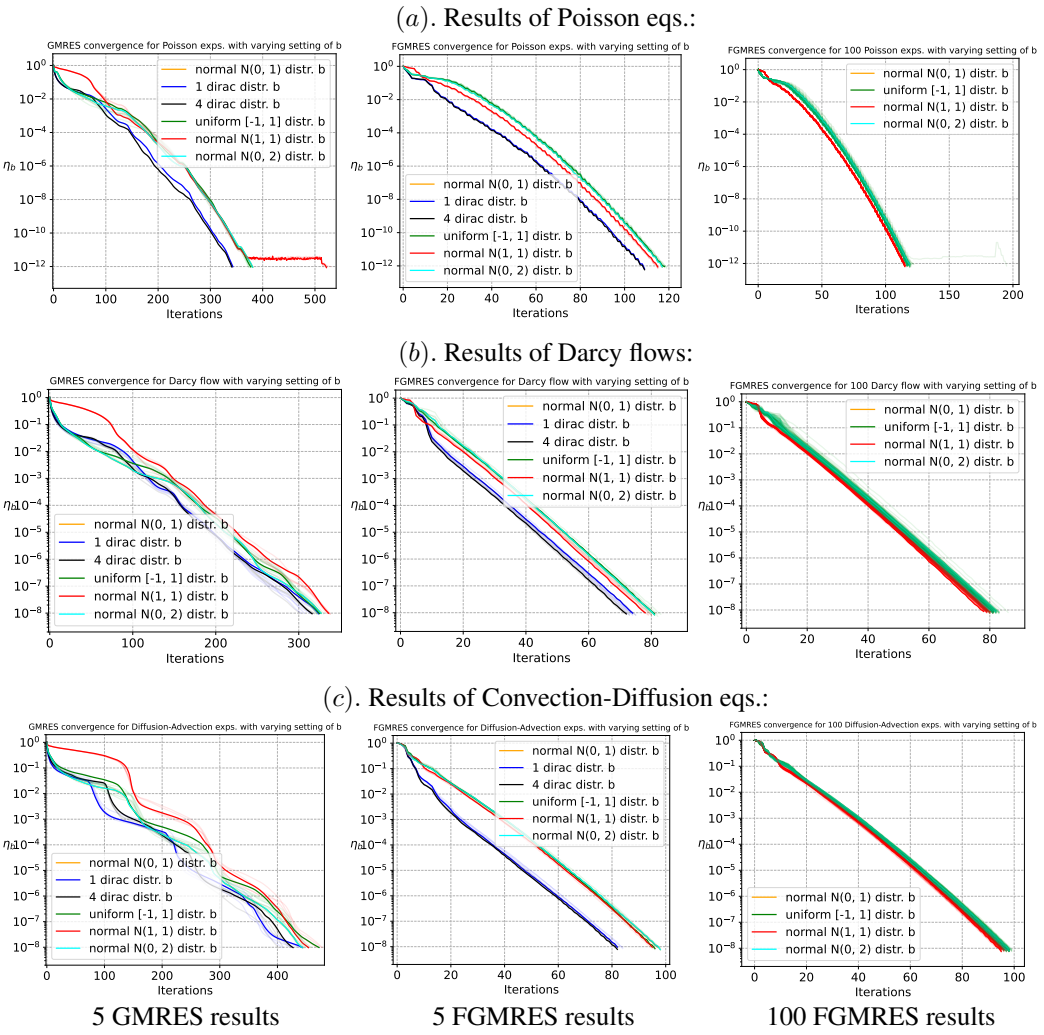


Figure 8: Check network generalizability in terms of varying source term $\mathbf{b}$ (for Section 4.2.1) by using FGMRES for the three types of fluid examples (simplified as exps.) (9)-(11) with zero-Dirichlet BCs and with $L = 1, 5,$ or 100.

equations with other kinds of classical BCs, including Neumann BCs and Cachy BCs shown in Figure 1 $(b) - (c)$, as well as their three types of mixtures shown in Figure 2. For simplicity, we test these fluid equations with $\mathbf{b}$ satisfying 1 `dirac distr.` setting shown in Section 4.2.1, and use the same other heterogeneous parameters, such as $\mathbf{p}, \mathbf{d}, \mathbf{c} \in \mathbb{R}^{n^2}$ satisfying the same uniform distribution $\mathcal{U}(1, 2)$ as used in the training process. The discrete Poisson equation with Neumann BCs is well-known to produce a linear system with singular coefficient matrix. For this reason, we exclude this singular case from our tests. The numerical results of remaining cases are presented in Figure 10 and Table 7. From these results, we observed similarly significant testing and generalizability performance of these trained CNN-based preconditioners as highlighted in Section 4.1-4.2.1. Notably, we found that while GMRES struggles to converge under most BCs aside from the simplest zero-Dirichlet BCs, FGMRES, enhanced by the trained general CNN-based preconditioners, successfully converges in all cases except for the mix_DN case. Figure 11 provides visualizations of the solutions obtained for the three types of fluid equations under various BCs. These visualizations clearly highlight the significant variations in solutions arising from the different applied BCs. It is well established that the choice of BCs profoundly influences the properties of the resulting discrete systems. This section underscores the potential of training a general neural network preconditioner using simple BCs, such as zero-Dirichlet BCs, to effectively address more complex cases involving diverse BCs.

It is important to note that, following the control variable method, only zero-Dirichlet BCs will be considered in the remainder of this work. These sections are specifically dedicated to evaluating the network

Table 7: Numerical results in terms of its for Section 4.2.2 for solving three fluid examples with varying BCs, and with $\mathbf{b} \in \mathbb{C}^{64^2}$ satisfying 1 `dirac distr.` setting and with the same other parameters $\mathbf{p}, \mathbf{d}, \mathbf{c} \in \mathbb{R}_{+}^{n^2}$.

# `fluid eqs.`	# BCs	Method	its	ET
Poisson eqs. with $\varepsilon = 10^{-12}$ epoch=467 val_loss=0.05100	Dirichlet BCs	GMRES	344	29.8149s
		FGMRES	110	8.9809s
	Cauchy BCs	GMRES	5120*	734.4874s
		FGMRES	405	51.1992s
	mix_{DC} BCs	GMRES	4582	542.4576s
		FGMRES	344	38.3996s
	mix_{DN} BCs	GMRES	5120*	633.1670s
		FGMRES	5120*	691.7072s
	mix_{DCN} BCs	GMRES	5120*	667.4544s
		FGMRES	428	50.0176s
Darcy flow with $\varepsilon = 10^{-8}$ epoch=484 val_loss=0.17501	Dirichlet BCs	GMRES	319	21.5340s
		FGMRES	74	3.2535s
	Neumann BCs	GMRES	5120*	508.6574s
		FGMRES	436	40.8165s
	Cauchy BCs	GMRES	5120*	1013.8389s
		FGMRES	500	101.6460s
	mix_{DC} BCs	GMRES	5120*	495.3070s
		FGMRES	388	33.8340s
	mix_{DN} BCs	GMRES	5120*	557.6320s
		FGMRES	5120*	556.9307s
	mix_{DCN} BCs	GMRES	5120*	518.2221s
		FGMRES	510	56.9348s
Convection-Diffusion eqs. with $\varepsilon = 10^{-8}$ epoch=483 val_loss=0.09784	Dirichlet BCs	GMRES	394	37.6104s
		FGMRES	83	3.6068s
	Neumann BCs	GMRES	5120*	553.2043s
		FGMRES	436	46.7377s
	Cauchy BCs	GMRES	5120*	662.6147s
		FGMRES	509	72.9558s
	mix_{DC} BCs	GMRES	5120*	556.2829s
		FGMRES	374	34.7866s
	mix_{DN} BCs	GMRES	5120*	547.6752s
		FGMRES	5120*	580.0284s
	mix_{DCN} BCs	GMRES	5120*	571.4277s
		FGMRES	489	51.3269s

*Note that symbol * denotes the algorithm diverges or it is unable to reach target accuracy within $maxIts = 5120$.

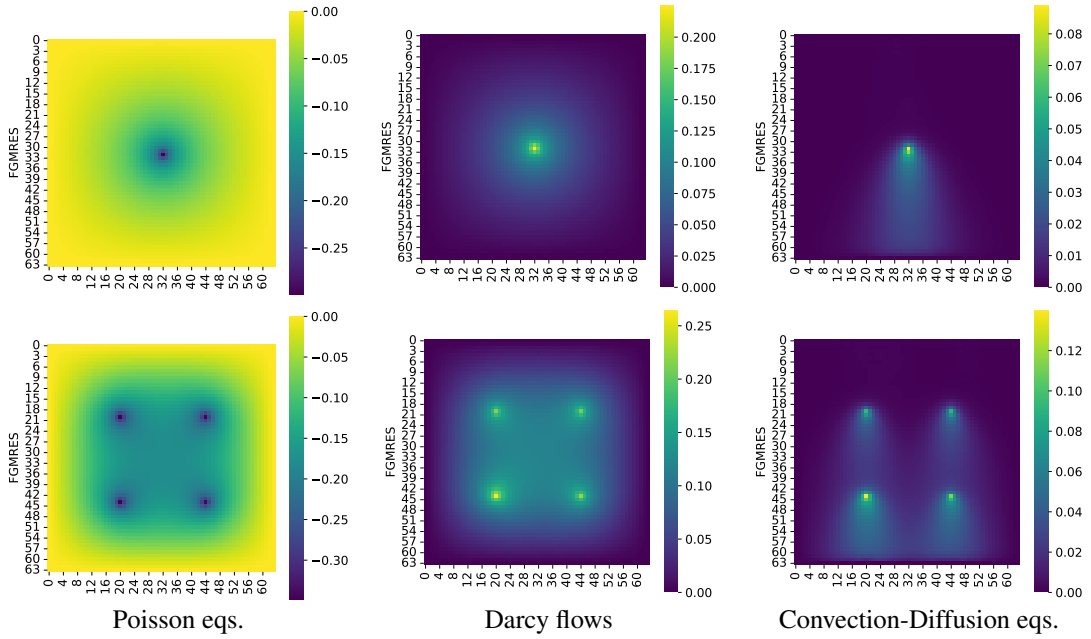


Figure 9: Visualize the FGMRES solution $\mathbf{u}$ for three different heterogeneous fluid equations with zero-Dirichlet BCs and with the source term $\mathbf{b}$ satisfying 1 `dirac distr.` or 4 `dirac distr.` setting for Section 4.2.1.

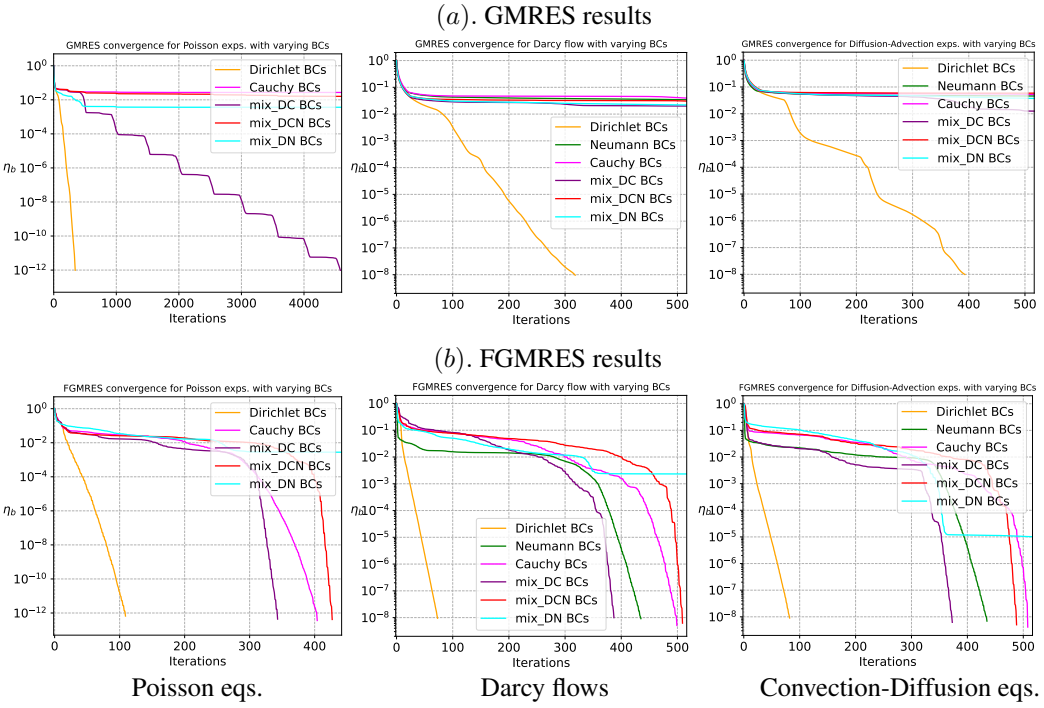


Figure 10: Check network generalizability of FGMRES in terms of varying BCs for Section 4.2.2.

generalizability of the trained preconditioners across other aspects.

4.2.3 In terms of varying the permeability or diffusion term

In this section, we test the network generalizability of the trained CNN-based preconditioners by varying the range of the heterogeneous permeability $\mathbf{p} \in \mathbb{R}^{64^2}$ and diffusion $\mathbf{d} \in \mathbb{R}^{64^2}$, which are uniformly distributed over the interval $[1, 2]$ during the training process, denoted as $\mathbf{p}, \mathbf{d} \in \mathbb{R}^{64^2} \sim \mathcal{U}(1, 2)$.

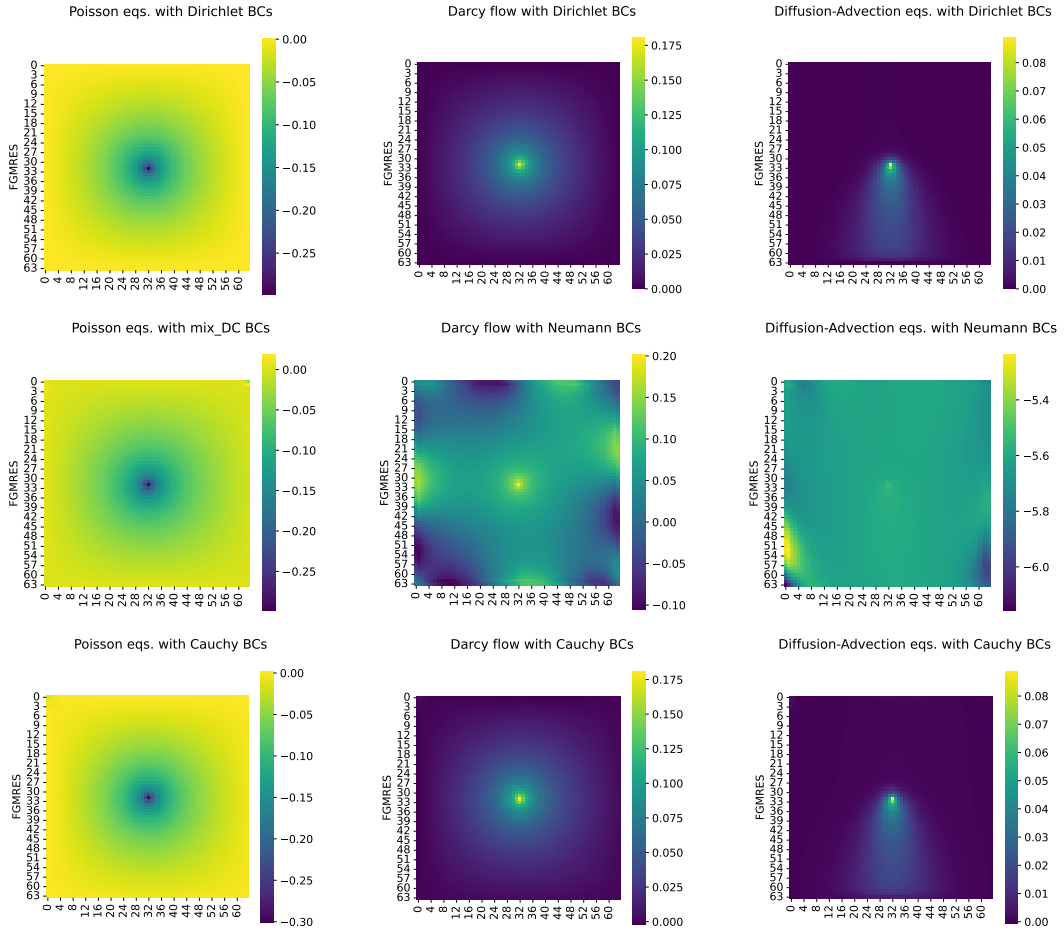


Figure 11: Visualize FGMRES solution $\mathbf{u} \in \mathbb{C}^{64^2}$ for Section 4.2.2 with 1 `dirac distr.` $\mathbf{b}$ and varying BCs.

To be specific, this section tests the trained CNN-based preconditioners with $\mathbf{p}, \mathbf{d}$ satisfying different uniform distribution. For example, $\mathbf{p}, \mathbf{d} \in [1, 2] \sim \mathcal{U}(1, 2)$, $\mathbf{p}, \mathbf{d} \in [1.25, 1.75] \sim \mathcal{U}(1.25, 1.75)$, $\mathbf{p}, \mathbf{d} \in [1, 2.5] \sim \mathcal{U}(1, 2.5)$, $\mathbf{p}, \mathbf{d} \in [0.5, 2] \sim \mathcal{U}(0.5, 2)$, $\mathbf{p}, \mathbf{d} \in [0.5, 2.5] \sim \mathcal{U}(0.5, 2.5)$, $\mathbf{p}, \mathbf{d} \in [2, 2.5] \sim \mathcal{U}(2, 2.5)$, $\mathbf{p}, \mathbf{d} \in [0.5, 1] \sim \mathcal{U}(0.5, 1)$, $\mathbf{p}, \mathbf{d} \in [0.5, 5] \sim \mathcal{U}(0.5, 5)$, and $\mathbf{p}, \mathbf{d} \in [0.1, 2] \sim \mathcal{U}(0.1, 2)$. Besides, we also test a structured $\mathbf{p}$ with a square shape in the center of the domain 64×64. That is part of its elements satisfying $\mathbf{p}[20 : 44, 20 : 44] = 2$ and others $\mathbf{p} = 1$ elsewhere. The same square setting applies to $\mathbf{d}$ as well. For these different settings of $\mathbf{p}$ and $\mathbf{d}$, similarly based on the control variable method, we apply the same fixed source term $\mathbf{b} \in \mathbb{R}^{64^2}$, satisfying 1 `dirac distr.` setting as mentioned in Section 4.2.1.

The convergence histories of the Darcy flows with varying permeability terms $\mathbf{p}$ and the Convection-Diffusion equations with varying diffusion terms $\mathbf{d}$ are shown in Figure 12 and Figure 13, respectively. The corresponding numerical results are detailed in Tables 8-9. Comparing these results of FGMRES with GMRES highlights the remarkable network generalizability of the trained CNN-based preconditioners in terms of handling variations in the permeability of Darcy flows and the diffusion term of Convection-Diffusion equations. Specially, from the left plot of Figure 12, it is evident that GMRES struggles to solve discrete Darcy flows when the permeability parameter $\mathbf{p}$ takes on smaller values, such as those over the interval $[0.1, 2]$, or when $\mathbf{p}$ spans a broader range like $[0.5, 5]$. In contrast, FGMRES successfully addresses all these challenging cases, even though the neural networks were not exposed to such values in training, including those in the intervals $[0.1, 1]$ and $[2, 5]$. Same observations apply to the left plot of Figure 13 for the Convection-Diffusion equations with varying diffusion terms $\mathbf{d}$.

Furthermore, Figure 14 provides a visualization of the FGMRES solutions $\mathbf{u}$ for Darcy flows under different permeability terms $\mathbf{p}$. The figure reveals that the solution field becomes less smooth when the permeability term $\mathbf{p}$ corresponds to the more challenging cases, such as those defined over the intervals

Table 8: Numerical results of in terms of its or (its_{max}, its_{min}) and ET for Section 4.2.3 for solving L ($L = 1$, $L = 100$ or $L = 1000$) Darcy flow equations (7) with varying permeability $\mathbf{p}$.

# `fluid eqs.`	# $\mathbf{p} \in \mathbb{R}^{64^2}$ on	Method	L	its or (its_{max}, its_{min})	ET
Darcy flow with $\varepsilon = 10^{-8}$ epoch=484 val_loss=0.17501	$\mathcal{U}(1,2)$	GMRES	1	326	21.6160s
		FGMRES	1	72	3.1138s
		FGMRES	100	(78, 72)	88.3086s
		FGMRES	1000	(82, 70)	1325.9235s
	$\mathcal{U}(1.25,1.75)$	GMRES	1	303	18.1218s
		FGMRES	1	69	2.9415s
		FGMRES	100	(74, 68)	83.3381s
		FGMRES	1000	(74, 67)	1156.8312s
	$\mathcal{U}(1,2.5)$	GMRES	1	332	38.6903s
		FGMRES	1	83	2.9369s
		FGMRES	100	(94, 81)	153.8347s
		FGMRES	1000	(94, 80)	1670.6245s
	$\mathcal{U}(0.5,2)$	GMRES	1	379	28.2535s
		FGMRES	1	81	3.2983s
		FGMRES	100	(102, 80)	112.1254s
		FGMRES	1000	(102, 79)	1645.0857s
	$\mathcal{U}(0.5,2.5)$	GMRES	1	433	37.5742s
		FGMRES	1	95	3.5771s
		FGMRES	100	(135, 99)	200.5330s
		FGMRES	1000	(146, 94)	2647.4697s
	$\mathcal{U}(2,2.5)$	GMRES	1	292	17.5402s
		FGMRES	1	88	3.3292s
		FGMRES	100	(92, 86)	169.8228s
		FGMRES	1000	(95, 85)	1743.9364s
	$\mathcal{U}(0.5,1)$	GMRES	1	320	23.4513s
		FGMRES	1	72	3.0770s
		FGMRES	100	(75, 70)	84.4250s
		FGMRES	1000	(75, 69)	1197.2716s
	$\mathcal{U}(0.5,5)$	GMRES	1	4566	890.0464s
		FGMRES	1	322	47.0591s
		FGMRES	100	(376, 265)	4574.4856s
	$\mathcal{U}(0.1,2)$	GMRES	1	5120*	532.7237s
		FGMRES	1	455	45.2398s
		FGMRES	100	(990, 390)	4368.7206s
	square shape	GMRES	1	335	21.5060s
		FGMRES	1	62	2.8410s

Table 9: Numerical results of in terms of its or (its_{max}, its_{min}) and ET for Section 4.2.3 for solving a series of Convection-Diffusion equations (8) with $L = 1$, $L = 100$ or $L = 1000$ and with varying diffusion term $\mathbf{d}$.

# `fluid eqs.`	# $\mathbf{d} \in \mathbb{R}^{64^2}$ on	Method	L	its or (its_{max}, its_{min})	ET
Convection-Diffusion eqs. with $\varepsilon = 10^{-8}$ epoch=483 val_loss=0.09784	$\mathcal{U}(1,2)$	GMRES	1	410	34.3245s
		FGMRES	1	85	3.3083s
		FGMRES	100	(86, 82)	117.3535s
		FGMRES	1000	(86, 82)	1060.2771s
	$\mathcal{U}(1.25,1.75)$	GMRES	1	402	35.4889s
		FGMRES	1	80	3.2790s
		FGMRES	100	(81, 78)	96.5203s
		FGMRES	1000	(81, 78)	990.5940s
	$\mathcal{U}(1,2.5)$	GMRES	1	401	30.2844s
		FGMRES	1	94	3.5878s
		FGMRES	100	(97, 92)	139.8715s
		FGMRES	1000	(99, 91)	1447.3728s
	$\mathcal{U}(0.5,2)$	GMRES	1	483	46.9776s
		FGMRES	1	96	3.5172s
		FGMRES	100	(105, 92)	178.2433s
		FGMRES	1000	(107, 91)	1516.0654s
	$\mathcal{U}(0.5,2.5)$	GMRES	1	501	58.2023s
		FGMRES	1	113	4.0818s
		FGMRES	100	(132, 103)	217.5746s
		FGMRES	1000	(138, 103)	1801.6917s
	$\mathcal{U}(2,2.5)$	GMRES	1	370	25.7981s
		FGMRES	1	98	3.5404s
		FGMRES	100	(101, 95)	156.7992s
		FGMRES	1000	(103, 95)	2062.0618s
	$\mathcal{U}(0.5,1)$	GMRES	1	592	52.6816s
		FGMRES	1	95	3.5800s
		FGMRES	100	(97, 90)	128.7326s
		FGMRES	1000	(98, 89)	1946.0813s
	$\mathcal{U}(0.5,5)$	GMRES	1	1438	131.4927s
		FGMRES	1	294	18.7631s
		FGMRES	100	(364, 227)	2198.2898s
	$\mathcal{U}(0.1,2)$	GMRES	1	3072	414.1471s
		FGMRES	1	270	26.5968s
		FGMRES	100	(360, 203)	1337.9980s
	square shape	GMRES	1	562	53.5004s
		FGMRES	1	82	3.2674s
		FGMRES	100	(84, 81)	103.6888s
		FGMRES	1000	(84, 80)	1185.3164s

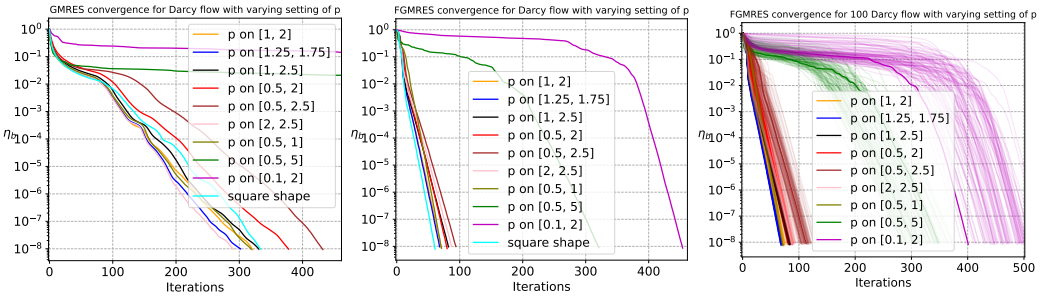


Figure 12: Check network generalizability of FGMRES in terms of varying the permeability term $\mathbf{p}$ for a series of discrete Darcy flows ($L = 1$ or $L = 100$) with zero-Dirichlet BCs for Section 4.2.3.

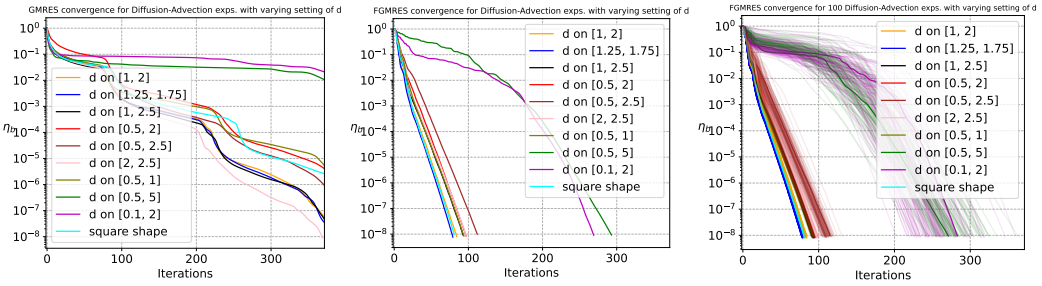


Figure 13: Check network generalizability of FGMRES in terms of varying the diffusion term $\mathbf{d}$ for a series of discrete Convection-Diffusion eqs. ($L = 1$ or $L = 100$) with zero-Dirichlet BCs for Section 4.2.3.

$[0.5, 5]$ and $[0.1, 2]$. This reduced smoothness further reflects the increased complexity introduced by these parameter ranges, and smaller values, which pose significant challenges to traditional GMRES solver but are effectively addressed by this general CNN-based preconditioners.

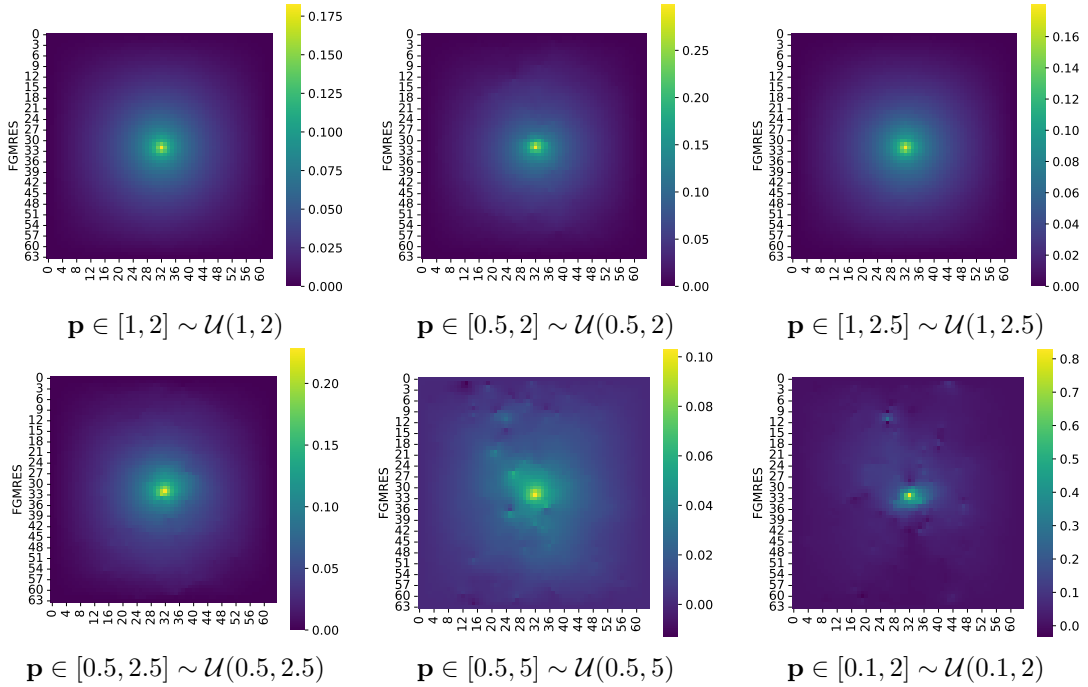


Figure 14: Visualize the FGMRES solution $\mathbf{u}$ for the Darcy flow with different permeability $\mathbf{p}$, with zero-Dirichlet BCs and with the source term $\mathbf{b}$ satisfying 1 `dirac distr.` for Section 4.2.3.

4.2.4 In terms of varying the convection term

This section tests the network generalizability of the trained CNN-based preconditioner for the Convection-Diffusion equations by varying the range of the heterogeneous convection term $\mathbf{c} \in \mathbb{R}^{64^2} \sim \mathcal{U}(1,2)$, which is uniformly distributed over the range $[1,2]$ during the training process. To be specific, we test the trained CNN-based preconditioner with $\mathbf{c}$ satisfying different uniform distribution. For example, $\mathbf{c} \in [1,2] \sim \mathcal{U}(1,2)$, $\mathbf{c} \in [1.25, 1.75] \sim \mathcal{U}(1.25, 1.75)$, $\mathbf{c} \in [1, 2.5] \sim \mathcal{U}(1, 2.5)$, $\mathbf{c} \in [0.5, 2] \sim \mathcal{U}(0.5, 2)$, $\mathbf{c} \in [0.5, 2.5] \sim \mathcal{U}(0.5, 2.5)$, $\mathbf{c} \in [2, 2.5] \sim \mathcal{U}(2, 2.5)$, $\mathbf{c} \in [0.5, 1] \sim \mathcal{U}(0.5, 1)$, $\mathbf{c} \in [0.5, 5] \sim \mathcal{U}(0.5, 5)$, and $\mathbf{c} \in [0.1, 2] \sim \mathcal{U}(0.1, 2)$. Besides, we also test a structured $\mathbf{c}$ with a square shape as described in precious Section 4.2.3, i.e., $\mathbf{c}[20:44, 20:44] = 2$ and others $\mathbf{c} = 1$ elsewhere. For these different settings of $\mathbf{c}$, we also apply the same fixed source term $\mathbf{b} \in \mathbb{C}^{64^2}$, satisfying the `1 dirac distr.` setting, and the same diffusion term $\mathbf{d}$ satisfying $\mathcal{U}(1,2)$. Convergence histories and numerical results are reported in Figure 15 and Table 10, respectively. These results illustrate the significant generalizability of the trained CNN-based preconditioner in addressing varying convection terms.

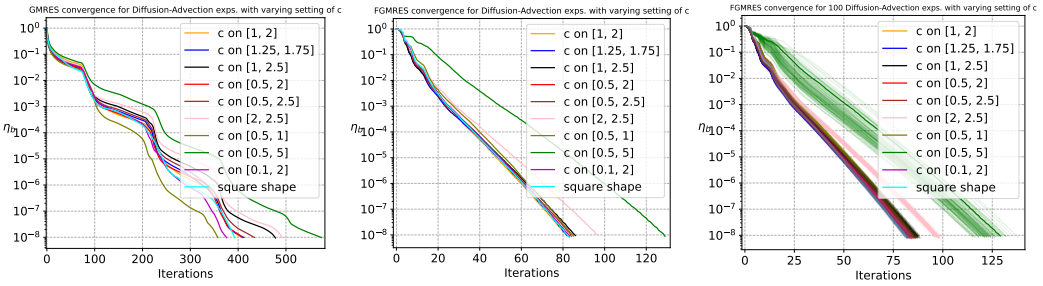


Figure 15: Check network generalizability of FGMRES in terms of varying the convection term $\mathbf{c}$ (for Section 4.2.4) for heterogeneous Convection-Diffusion equations (8) with $L = 1$ or $L = 100$ and with zero-Dirichlet BCs.

Furthermore, as discussed in Section 2.1.3, the behavior of Convection-Diffusion equations varies based on the ratio between the convection term $\mathbf{c}$ and the diffusion term $\mathbf{d}$. It includes the balanced case where $\frac{\mathbf{d}}{\mathbf{c}} \approx 1$, the diffusion-dominant where $\frac{\mathbf{d}}{\mathbf{c}} \gg 1$, and the convection-dominant case where $\frac{\mathbf{d}}{\mathbf{c}} \ll 1$. Among these, the convection-dominant case is the most challenging to address. In this section, we tackle all these three cases with the trained CNN-based preconditioner, which was trained under the balanced case. Numerical results and parts of the visualizations are reported in Table 11 and Figure 16, respectively. Similar effective conclusions apply to the trained CNN-based preconditioner. An intriguing observation from the results of the balanced case in Table 11 is that the CNN-based preconditioner successfully handles a range 100 times wider, up to $[100, 200]$, despite being trained only on values within the interval $[1,2]$. This further demonstrates the remarkable generalization capability of this general CNN-based preconditioner to operate effectively well beyond its training range.

4.2.5 In terms of varying the domain size

Given these general CNN-based preconditioners were trained on domain size 64×64, this section tests the network generalizability of the trained preconditioners by varying the domain size. Specifically, we test the trained CNN-based preconditioners on domain 64×64 (i.e., $n^2 = 64^2$), 128×128 (i.e., $n = 128^2$, denoted as 2^2*), 256×256 (i.e., $n = 256^2$, denoted as 4^2*), 384×384 (i.e., $n = 384^2$, denoted as 6^2*), and 512×512 (i.e., $n = 512^2$, denoted as 8^2*). The distribution of the permeability $\mathbf{p} \in \mathbb{R}^{n^2}$, the diffusion $\mathbf{d} \in \mathbb{C}^{n^2}$ and the convection $\mathbf{c}$ are the same distribution settings as the ones used in the training process. That means over these different domain size, $\mathbf{p}, \mathbf{d}, \mathbf{c}$ satisfying uniform distribution $\mathcal{U}(1,2)$ over the interval $[1,2]$. While the source $\mathbf{b} = x + 0j$ with x satisfying standard normal distribution $\mathcal{N}(0,1)$. The corresponding numerical results are presented in Figure 17 and Table 12. These results indicate the effectiveness of the trained CNN-based preconditioners in significantly reducing the iteration steps its and execution times ET required by GMRES, even when applied them to a 64 times larger domain size, reaching up to 512×512. This improvement is particularly pronounced in the results for Darcy flows and Convection-Diffusion equations, demonstrating the scalability and robustness of the preconditioners across varying domain sizes.

Table 10: Numerical results in terms of its or (its_{max}, its_{min}) and ET for solving a series of Convection-Diffusion equations (8) ($L = 1$, $L = 100$ and $L = 1000$) with varying setting for convection $\mathbf{c}$, but with the same diffusion $\mathbf{d}$ satisfying $\mathcal{U}(1, 2)$ and the same 1 `dirac distr.` setting of $\mathbf{b}$ for Section 4.2.4.

# $\mathbf{c} \in \mathbb{R}^{64^2}$ on	Method	L	its or (its_{max}, its_{min})	ET
$\mathcal{U}(1, 2)$	GMRES	1	408	35.5709s
	FGMRES	1	85	3.3399s
	FGMRES	100	(86, 82)	114.5619s
	FGMRES	1000	(86, 82)	1546.4189s
$\mathcal{U}(1.25, 1.75)$	GMRES	1	413	37.2114s
	FGMRES	1	83	3.3286s
	FGMRES	100	(86, 82)	107.0795s
	FGMRES	1000	(86, 82)	1629.3463s
$\mathcal{U}(1, 2.5)$	GMRES	1	479	3.4219s
	FGMRES	1	87	3.4219s
	FGMRES	100	(90, 86)	118.8076s
	FGMRES	1000	(90, 85)	1785.3667s
$\mathcal{U}(0.5, 2)$	GMRES	1	410	35.6083s
	FGMRES	1	84	3.3302s
	FGMRES	100	(85, 82)	105.6585s
	FGMRES	1000	(87, 81)	1542.7184s
$\mathcal{U}(0.5, 2.5)$	GMRES	1	435	37.1271s
	FGMRES	1	85	3.3394s
	FGMRES	100	(88, 84)	121.8107s
	FGMRES	1000	(88, 83)	1712.9678s
$\mathcal{U}(2, 2.5)$	GMRES	1	493	49.9054s
	FGMRES	1	98	3.8491s
	FGMRES	100	(101, 95)	138.8298s
	FGMRES	1000	(101, 96)	2064.2818s
$\mathcal{U}(0.5, 1)$	GMRES	1	359	25.8455s
	FGMRES	1	86	3.3555s
	FGMRES	100	(89, 85)	116.5543s
	FGMRES	1000	(89, 85)	1666.9948s
$\mathcal{U}(0.5, 5)$	GMRES	1	575	54.4889s
	FGMRES	1	130	4.6627s
	FGMRES	100	(139, 116)	222.5920s
	FGMRES	1000	(149, 115)	3115.4121s
$\mathcal{U}(0.1, 2)$	GMRES	1	377	30.7106s
	FGMRES	1	84	3.3652s
	FGMRES	100	(87, 83)	113.1981s
	FGMRES	1000	(88, 82)	1599.6767s
square shape	GMRES	1	394	31.2774s
	FGMRES	1	84	3.3934s
	FGMRES	100	(84, 81)	126.8673s
	FGMRES	1000	(85, 81)	1541.9930s

Table 11: Numerical results in terms of its or (its_{max}, its_{min}) and ET for Section 4.2.4 for solving three cases of Convection-Diffusion equations (8) with heterogeneous diffusion $\mathbf{d}$ and convection $\mathbf{c}$, and with 1 `dirac.` source $\mathbf{b}$.

# $\mathbf{d}, \mathbf{c} \in \mathbb{R}^{64^2}$ on	Method	L	its or (its_{max}, its_{min})	ET
Balanced case with $\frac{\mathbf{d}}{\mathbf{c}} \approx 1$:				
$\mathbf{d}, \mathbf{c} \in \mathcal{U}(1, 2)$	GMRES	1	408	35.5709s
	FGMRES	1	85	3.3399s
	FGMRES	100	(86, 82)	114.5619s
	FGMRES	1000	(86, 82)	1546.4189s
$\mathbf{d}, \mathbf{c} \in \mathcal{U}(25, 50)$	GMRES	1	420	50.2798s
	FGMRES	1	233	17.5740s
	FGMRES	100	(249, 222)	2392.9016s
$\mathbf{d}, \mathbf{c} \in \mathcal{U}(50, 100)$	GMRES	1	460	54.7812s
	FGMRES	1	250	20.2132s
	FGMRES	100	(292, 232)	2592.4430s
$\mathbf{d}, \mathbf{c} \in \mathcal{U}(100, 200)$	GMRES	1	468	57.3766s
	FGMRES	1	283	24.2253s
	FGMRES	100	(294, 251)	3099.6522s
Diffusion-dominant case with $\frac{\mathbf{d}}{\mathbf{c}} \gg 1$:				
$\mathbf{d} \in \mathcal{U}(25, 50), \mathbf{c} \in \mathcal{U}(1, 2)$	GMRES	1	325	21.8579s
	FGMRES	1	276	16.8212s
	FGMRES	100	(288, 267)	3612.8073s
$\mathbf{d} \in \mathcal{U}(50, 100), \mathbf{c} \in \mathcal{U}(1, 2)$	GMRES	1	325	30.7067s
	FGMRES	1	302	30.4414s
	FGMRES	100	(318, 288)	4053.1707s
Convection-dominant case with $\frac{\mathbf{d}}{\mathbf{c}} \ll 1$:				
$\mathbf{d} \in \mathcal{U}(1, 2), \mathbf{c} \in \mathcal{U}(25, 50)$	GMRES	1	2854	283.7008s
	FGMRES	1	1021	107.3359s
	FGMRES	10	(1369, 1019)	1365.1770s
$\mathbf{d} \in \mathcal{U}(1, 2), \mathbf{c} \in \mathcal{U}(50, 100)$	GMRES	1	4436	597.5046s
	FGMRES	1	3937	374.7240s
	FGMRES	10	(3996, 3371)	3547.3119s

Table 12: Numerical results in terms of its or (its_{max}, its_{min}) and ET for Section 4.2.5 for solving three fluid equations (9)-(11).

# `fluid eqs.`	# Domain	Method	L	its or (its_{max}, its_{min})	ET
Poisson eqs. with $\varepsilon = 10^{-12}$	64×64	GMRES	1	382	26.5632s
		FGMRES	1	120	4.0153s
		FGMRES	100	(122, 118)	193.3075s
	128×128	GMRES	1	1012	94.5204s
		FGMRES	1	348	24.9729s
		FGMRES	100	(616, 228)	4881.7191s
	256×256	GMRES	1	2999	329.9844s
		FGMRES	1	532	57.8837s
		FGMRES	100	(711, 535)	6695.4944s
	384×384	GMRES	1	4815	477.1662s
		FGMRES	1	810	79.2054s
		FGMRES	100	(865, 570)	14348.0632s
	512×512	GMRES	1	5120*	559.6234s
		FGMRES	1	5120*	596.2346s
		FGMRES	100	(5120*, 5120*)	58078.9011s
Darcy flow with $\varepsilon = 10^{-8}$	64×64	GMRES	1	329	22.0342s
		FGMRES	1	81	3.2237s
		FGMRES	100	(84, 80)	106.6865s
	128×128	GMRES	1	664	104.8114s
		FGMRES	1	154	9.6673s
		FGMRES	100	(164, 150)	589.8466s
	256×256	GMRES	1	1681	298.0206s
		FGMRES	1	225	17.0054s
		FGMRES	100	(240, 224)	1453.7784s
	384×384	GMRES	1	2987	340.3831s
		FGMRES	1	324	25.2482s
		FGMRES	100	(345, 316)	2321.4541s
	512×512	GMRES	1	4367	488.2045s
		FGMRES	1	450	59.9249s
		FGMRES	100	(460, 430)	4599.2886s
Convection-Diffusion eqs. with $\varepsilon = 10^{-8}$	64×64	GMRES	1	440	35.0634s
		FGMRES	1	99	3.5620s
		FGMRES	100	(100, 96)	150.8635s
	128×128	GMRES	1	1000	105.5346s
		FGMRES	1	164	7.1934s
		FGMRES	100	(178, 162)	656.0577s
	256×256	GMRES	1	2380	225.8275s
		FGMRES	1	250	14.8938s
		FGMRES	100	(265, 241)	1303.9984s
	384×384	GMRES	1	3632	399.3302s
		FGMRES	1	343	29.9250s
		FGMRES	100	(359, 328)	3011.5191s
	512×512	GMRES	1	4642	551.4533s
		FGMRES	1	447	64.4661s
		FGMRES	100	(470, 427)	5384.0654s

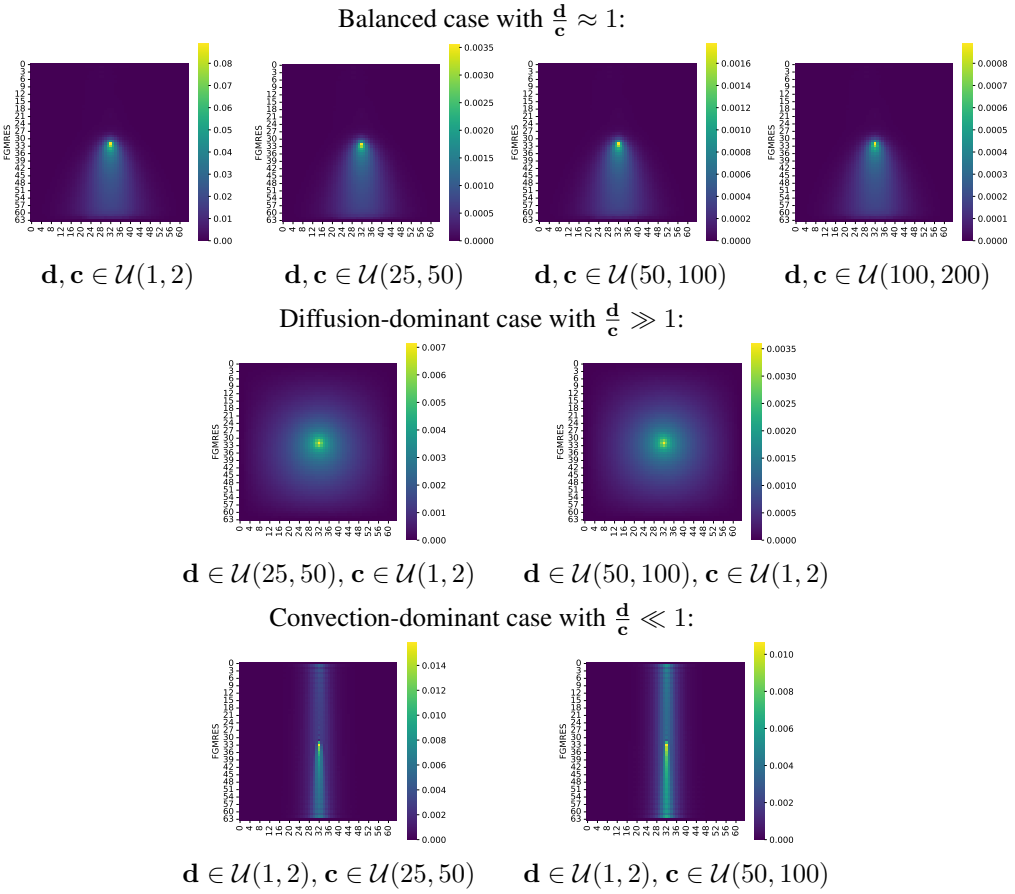


Figure 16: Visualize the FGMRES solution $\mathbf{u}$ for the Convection-Diffusion eqs. with different diffusion term $\mathbf{d}$, convection term $\mathbf{c}$, zero-Dirichlet BCs and the source term $\mathbf{b}$ satisfying 1 `dirac distr.` for Section 4.2.4.

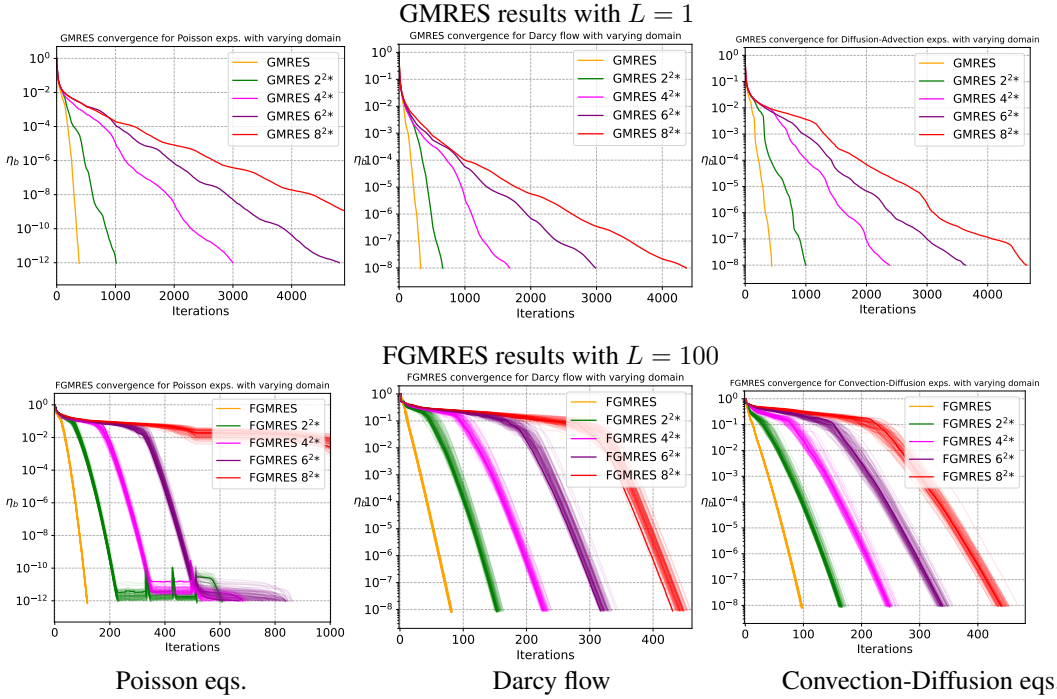


Figure 17: Check network generalizability of FGMRES in terms of varying the domain size (for Section 4.2.5) for different fluid equations (9)-(11) with zero-Dirichlet BCs, and compare it to GMRES.

5 Concluding remarks

Scientific machine learning has been widely used for scientific computing to solve or to enhance the simulation of the partial differential equations. In this paper, we focus on using U-Net composed by convolutional neural networks (CNNs) for accelerating the solution of the different parametric heterogeneous fluid equations with different parameters and classical boundary conditions. The learning process is implemented under the unsupervised context, and it aims to approximate the inverse of the parametric fluid operators. The learned operators are used as a non-linear flexible preconditioner in the numerical linear algebra methods, such as the flexible GMRES method considered in this work, to improving its effectiveness. Extensive numerical results demonstrate that the trained CNN-based preconditioners exhibit exceptional performance during testing phases and showcase impressive generalizability. These preconditioners effectively adapt to a wider range of heterogeneous fluid equations with varying parameters and varying BCs, affirming their robustness and versatility. This work illustrates the promising direction in integrating the traditional numerical iterative methods and the recent unsupervised scientific machine learning approaches through non-linear preconditioning to let them benefit from each other.

Future work will aim to extend this CNN-based preconditioning approach to fluid equations on unstructured and irregular domains. Additionally, it will explore the integration of diverse domain with the generative geometry techniques, such as Geometry-Informed Neural Networks [8] and geometric deep learning [16]. These advancements aim to address applications in plasma-fluid and incompressible fluid simulations. Extension to the time-depended problems should be followed as well.

Acknowledgments

This work was completed during the author's postdoctoral research under the supervision of Luc Giraud (Inria, France) and Paul Mycek (Cerfacs, France). Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), University of Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (refer to https://www.plafrim.fr). The author extends heartfelt gratitude to Mehdi Ettaouchi (EDF, France) for the insightful discussions they shared on Convection-Diffusion equations.

References

[1] J. Ackmann, P. D. Düben, T. N. Palmer, and P. K. Smolarkiewicz. Machine-Learned Preconditioners for Linear Solvers in Geophysical Fluid Flows. *arXiv*, 2020. https://doi.org/10.48550/arXiv.2010.02866.

[2] J. Aghili, E. Franck, R. Hild, V. Michel-Dansac, and V. Vigon. Accelerating the convergence of Newton's method for nonlinear elliptic PDEs using Fourier neural operators. *Communications in Nonlinear Science and Numerical Simulation*, 140(2), 2025. https://doi.org/10.1016/j.cnsns.2024.108434.

[3] P. F. Antonietti, L. Dede, and M. Caldana. Accelerating algebraic multigrid methods via artificial neural networks. *Vietnam J. Math.*, 51:1–36, 2023. https://doi.org/10.1007/s10013-022-00597-w.

[4] Y. Azulay and E. Treister. Multigrid-Augmented Deep Learning Preconditioners for the Helmholtz Equation. *SIAM J. Sci. Comput.*, 45(3):S127–S151, 2022. https://doi.org/10.1137/21M1433514.

[5] P. W. Battaglia, J. B. Hamrick, V. Bapst, and et. al. Relational inductive biases, deep learning, and graph networks. *arXiv*, 2018. https://doi.org/10.48550/arXiv.1806.01261.

[6] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic Differentiation in Machine Learning: a Survey. *J. Mach. Learn. Res.*, 18:1–43, 2018. https://www.jmlr.org/papers/volume18/17-468/17-468.pdf.

[7] J. Bear and M. Y. Corapcioglu. *Fundamentals of Transport Phenomena in Porous Media*. Springer Dordrecht, 2012. https://doi.org/10.1007/978-94-009-6175-3.

[8] A. Berzins, A. Radler, E. Volkmann, S. Sanokowski, S. Hochreiter, and J. Brandstetter. Geometry-Informed Neural Networks. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2402.14009.

[9] L. Besabe, M. Girfoglio, A. Quaini, and G. Rozza. Data-driven reduced order modeling of a two-layer quasi-geostrophic ocean model. *Results in Engineering*, 25:103691, 2024. https://doi.org/10.1016/j.rineng.2024.103691.

[10] S. Bhowmick, V. Eijkhout, Y. Freund, E. Fuentes, and D. Keyes. Application of Machine Learning in Selecting Sparse Linear Solvers. *J. High Perf. Comput.*, 2006. https://icl.utk.edu/files/publications/2006/icl-utk-287-2006.pdf.

[11] L. Borcea, J. Garnier, A. V. Mamonov, and J. Zimmerling. When Data Driven Reduced Order Modeling Meets Full Waveform Inversion. *SIAM Rev.*, 66(3):501–532, 2024. https://doi.org/10.1137/23M1552826.

[12] N. Boullé, D. Halikias, and A. Townsend. Elliptic PDE learning is provably data-efficient. *PNAS*, 120(39):e2303904120, 2023. https://doi.org/10.1073/pnas.2303904120.

[13] N. Boullé and A. Townsend. Learning Elliptic Partial Differential Equations with Randomized Linear Algebra. *Found Comput Math*, 23:709–739, 2023. https://doi.org/10.1007/s10208-022-09556-w.

[14] N. Boullé and A. Townsend. Chapter 3 - A mathematical guide to operator learning. In *Numerical Analysis Meets Machine Learning*, volume 25, pages 83–125. Elsevier, 2024. https://doi.org/10.1016/bs.hna.2024.05.003.

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

[16] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges. *arXiv*, 2021. https://doi.org/10.48550/arXiv.2104.13478.

[17] M. Caldana, P. F.. Antonietti, and L. Dede. A deep learning algorithm to accelerate algebraic multigrid methods in finite element solvers of 3D elliptic PDEs. *Computers & Mathematics with Applications*, 167:217–231, 2024. https://doi.org/10.1016/j.camwa.2024.05.013.

[18] J. R. Chen, X. R. Chi, W. N. E, and Z. W. Yang. Bridging Traditional and Machine Learning-based Algorithms for Solving PDEs: The Random Feature Method. *J. Mach. Learn.*, 1:268–298, 2022. https://doi.org/10.4208/jml.220726.

[19] J.-Z. Chen, J. K. Patel, and R. Vasques. Solver Recommendation For Transport Problems in Slabs Using Machine Learning. *arXiv*, 2019. https://doi.org/10.48550/arXiv.1906.08259.

[20] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. *arXiv*, 2018. https://doi.org/10.48550/arXiv.1806.07366.

[21] L. Cheng, E. A. Illarramendi, G. Bogopolsky, M. Bauerheim, and B. Cuenot. Using neural networks to solve the 2D Poisson equation for electric field computation in plasma fluid simulations. *arXiv*, 2021. https://doi.org/10.48550/arXiv.2109.13076.

[22] F. Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[23] D. Coscia, N. Demo, and G. Rozza. Generative adversarial reduced order modelling. *Sci. Rep.*, 14(3826), 2024. https://doi.org/10.1038/s41598-024-54067-z.

[24] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli. Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next. *J. Sci. Comput.*, 92(88), 2022. https://doi.org/10.1007/s10915-022-01939-z.

[25] N. Demo, M. Tezzele, and G. Rozza. A DeepONet multi-fidelity approach for residual learning in reduced order modeling. *Adv. Model. and Simul. in Eng. Sci.*, 10(12), 2023. https://doi.org/10.1186/s40323-023-00249-9.

[26] V. Dolean, A. Heinlein, S. Mishra, and B. Moseley. Multilevel domain decomposition-based architectures for physics-informed neural networks. *CMAME*, 2024. https://doi.org/10.1016/j.cma.2024.117116.

[27] D. Dong, W. Suo, J. Kou, and W. Zhang. PINN-MG: A Multigrid-Inspired Hybrid Framework Combining Iterative Method and Physics-Informed Neural Networks. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2410.05744.

[28] V. Dwivedi and B. Srinivasan. A Normal Equation-Based Extreme Learning Machine for Solving Linear Partial Differential Equation. *ASME. J. Comput. Inf. Sci. Eng.*, 22(1):014502, 2022. https://doi.org/10.1115/1.4051530.

[29] V. Ehrlacher, D. Lombardi, O. Mula, and F. X. Vialard. Nonlinear model reduction on metric spaces. Application to one-dimensional conservative PDEs in Wasserstein spaces. *arXiv*, 2020. https://doi.org/10.48550/arXiv.1909.06626.

[30] P. Fahy, M. Golbabaee, and M. J. Ehrhardt. Greedy Learning to Optimize with Convergence Guarantees. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2406.00260.

[31] D. Gilbarg and N. S. Trudinger. *Elliptic Partial Differential Equations of Second Order*. Springer Berlin, Heidelberg, 2015. https://doi.org/10.1007/978-3-642-61798-0.

[32] D. S. Gilliam. Mathematics 5342 Discrete Fourier Transform. Technical report, Texas Tech University, 1998.

[33] I. C. Gonnella, M. W. Hess, G. Stabile, and G. Rozza. A two-stage deep learning architecture for model reduction of parametric time-dependent problems. *Computers & Mathematics with Applications*, 149:115–127, 2023. https://doi.org/10.1016/j.camwa.2023.08.026.

[34] V. Gopakumar, S. Pamela, L. Zanisi, Z.-Y. Li, A. Anandkumar, and M. Team. Fourier Neural Operator for Plasma Modelling. *arXiv*, 2023. https://doi.org/10.48550/arXiv.2302.06542.

[35] M. Götz and H. Anzt. Machine Learning-Aided Numerical Linear Algebra: Convolutional Neural Networks for the Efficient Preconditioner Generation. In *2018 IEEE/ACM 9th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (scalA)*. IEEE, 2018. https://doi.org/10.1109/ScalA.2018.00010.

[36] H. Gowrachari, N. Demo, G. Stabile, and G. Rozza. Non-intrusive model reduction of advection-dominated hyperbolic problems using neural network shift augmented manifold transformations. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2407.18419.

[37] D. Greenfeld, M. Galun, R. Kimmel, I. Yavneh, and R. Basri. Learning to Optimize Multigrid PDE Solvers. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2415–2423, 2019. https://proceedings.mlr.press/v97/greenfeld19a.html.

[38] X.-X. Guo, W. Li, and F. Iorio. Convolutional Neural Networks for Steady Flow Approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016. https://doi.org/10.1145/2939672.2939738.

[39] J. K. Gupta and J. Brandstetter. Towards Multi-spatiotemporal-scale Generalized PDE Modeling. *Transactions on Machine Learning Research*, 2023. https://openreview.net/forum?id=dPSTDbGtBY.

[40] X. Han, F. Hou, and H. Qin. UGrid: An Efficient-And-Rigorous Neural Multigrid Solver for Linear PDEs. In *Proceedings of the 41st International Conference on Machine Learning*, pages 17354–17373, July 2024. https://arxiv.org/abs/2408.04846.

[41] J. He and J. Xu. MgNet: A unified framework of multigrid and convolutional neural network. *Sci. China Math.*, 62:1331–1354, 2019. https://doi.org/10.1007/s11425-019-9547-2.

[42] A. Heinlein, A. A. Howard, D. Beecroft, and P. Stinis. Multifidelity domain decomposition-based physics-informed neural networks and operators for time-dependent problems. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2401.07888.

[43] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Machine Learning in Adaptive Domain Decomposition Methods—Predicting the Geometric Location of Constraints. *SIAM J. Sci. Comput.*, 41(6), 2019. https://doi.org/10.1137/18M1205364.

[44] A. Heinlein, A. Klawonn, M. Lanser, and J. Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations-A review. *GAMM-Mitteilungen*, 44(1), 2021. https://doi.org/10.1002/gamm.202100001.

[45] M. Hestenes. The conjugate gradient method for solving linear systems. *PSAM*, VI, Numerical Analysis:83–102, 1956.

[46] A. A. Howard, B. Jacob, S. H. Murphy, A. Heinlein, and P. Stinis. Finite basis Kolmogorov-Arnold networks: domain decomposition for data-driven and physics-informed problems. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2406.19662.

[47] J.-T. Hsieh, S.-J. Zhao, S. Eismann, L. Mirabella, and S. Ermon. Learning Neural PDE Solvers with Convergence Guarantees. *arXiv*, 2019. https://doi.org/10.48550/arXiv.1906.01200.

[48] D. Z. Huang, N. H. Nelsen, and M. Trautner. An operator learning perspective on parameter-to-observable maps. *Foundations of Data Science*, 7(1):163–225, 2025. https://doi.org/10.3934/fods.2024037.

[49] J. O. Smith III. *Mathematics of the Discrete Fourier Transform (DFT): With Audio Applications*. W3K Publishing, North Charleston, 2nd edition edition, 2007. http://ccrma.stanford.edu/ jos/mdft/.

[50] E. A. Illarramendi, A. Alguacil, M. Bauerheim, A. Misdariis, B. Cuenot, and E. Benazera. Towards an hybrid computational strategy based on Deep Learning for incompressible flows. In *AIAA AVIATION 2020 FORUM*. American Institute of Aeronautics and Astronautics, 2020. https://doi.org/10.2514/6.2020-3058.

[51] A. Ivagnes, G. Stabile, A. Mola, T. Iliescu, and G. Rozza. Hybrid data-driven closure strategies for reduced order modeling. *Appl. Math. Comput.*, 448, 2023. https://doi.org/10.1016/j.amc.2023.127920.

[52] A. Ivagnes, G. Stabile, and G. Rozza. Parametric Intrusive Reduced Order Models enhanced with Machine Learning Correction Terms. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2406.04169.

[53] E. Jessup, P. Motter, B. Norris, and K. Sood. Performance-Based Numerical Solver Selection in the Lighthouse Framework. *SIAM J. Sci. Comput.*, 38(5):S750–S771, 2016. https://doi.org/10.1137/15M1028406.

[54] M. Khamlich, G. Stabile, G. Rozza, L. Környei, and Z. Horváth. A physics-based reduced order model for urban air pollution prediction. *Comput. Methods Appl. Mech. Engrg.*, 417, 2023. https://doi.org/10.1016/j.cma.2023.116416.

[55] M. Khodak, E. Chow, M. F. Balcan, and A. Talwalkar. Learning to Relax: Setting Solver Parameters Across a Sequence of Linear System Instances. In *The Twelfth International Conference on Learning Representations*, 2024. https://openreview.net/forum?id=5t57omGVMw.

[56] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, 2014. https://doi.org/10.48550/arXiv.1412.6980.

[57] A. Klawonn, M. Lanser, and J. Weber. A Domain Decomposition–Based CNN-DNN Architecture for Model Parallel Training Applied to Image Recognition Problems. *SIAM J. Sci. Comput.*, 46(5), 2024. https://doi.org/10.1137/23M1562202.

[58] A. Klawonn, M. Lanser, and J. Weber. Machine learning and domain decomposition methods - a survey. *Comput. Sci. Eng.*, 1(2), 2024. https://doi.org/10.1007/s44207-024-00003-y.

[59] N. Kovachki, Z.-Y. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs. *J. Mach. Learn. Res.*, pages 1–97, 24 2023. http://jmlr.org/papers/v24/21-1524.html.

[60] I. Lagaris, A. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–9, 2000. http://dx.doi.org/10.1109/72.870037.

[61] S. Lanthaler and N. H. Nelsen. Error Bounds for Learning with Vector-Valued Random Features. *arXiv*, 2023. https://doi.org/10.48550/arXiv.2305.17170.

[62] H. Li, J. Schwab, S. Antholzer, and M. Haltmeier. NETT: solving inverse problems with deep neural networks. *Inverse Problems*, 36(6):065005, 2020. https://doi.org/10.1088/1361-6420/ab6d57.

[63] K. Li, S. M. Khan, and Y. Mehmani. Machine learning for preconditioning elliptic equations in porous microstructures: A path to error control. *Computer Methods in Applied Mechanics and Engineering*, 427:117056, 2024. https://doi.org/10.1016/j.cma.2024.117056.

[64] Z.-Y. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. In *International Conference on Learning Representations*, 2021. https://openreview.net/forum?id=c8P9NQVtmnO.

[65] Z.-Y. Li, H. K. Zheng, N. Kovachki, D. Jin, H. X. Chen, B. Liu, K. Azizzadenesheli, and A. Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *ACM/JMS Journal of Data Science*, 1:1–27, May 2024. https://doi.org/10.1145/3648506.

[66] Z.-C. Long, Y.-P. Lu, X.-Z. Ma, and B. Dong. PDE-Net: Learning PDEs from Data. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 3208–3216. PMLR, 2018. https://proceedings.mlr.press/v80/long18a.html.

[67] L. Lu, P. Z. Jin, G. F. Pang, Z. Q. Zhang, and G. Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3:218–229, March 2021. https://doi.org/10.1038/s42256-021-00302-5.

[68] L. Lu, X.-H. Meng, Z.-P. Mao, and G. Em Karniadakis. DeepXDE: A Deep Learning Library for Solving Differential Equations. *SIAM Rev.*, 63(1):208–228, 2021. https://doi.org/10.1137/19M1274067.

[69] L. Lu, R. Pestourie, W.-J. Yao, Z.-C. Wang, F. Verdugo, and S. G. Johnson. Physics-Informed Neural Networks with Hard Constraints for Inverse Design. *SIAM J. Sci. Comput.*, 43(6):B1105–B1132, 2021. https://doi.org/10.1137/21M1397908.

[70] K. Luna, K. Klymko, and J. P. Blaschke. Accelerating GMRES with Deep Learning in Real-Time. *arXiv*, 2021. https://doi.org/10.48550/arXiv.2103.10975.

[71] I. Luz, M. Galun, H. Maron, R. Basri, and I. Yavneh. Learning Algebraic Multigrid Using Graph Neural Networks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119, pages 6489–6499, 2020. https://proceedings.mlr.press/v119/luz20a.html.

[72] P.-G. Martinsson and J. A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020. https://doi.org/10.1017/S0962492920000021.

[73] P. Milbradt and W. Abu Abed. Generalized Stabilization Techniques in Computational Fluid Dynamics. In *Proceedings of the 8th International Conference on Hydro-Science and Engineering, Nagoya, Japan. Nagoya: Nagoya Hydraulic Research Institute for River Basin Management*. 2008. https://hdl.handle.net/20.500.11970/110078.

[74] R. B. Morgan. GMRES with Deflated Restarting. *SIAM J. Sci. Comput.*, 24(1):20–37, 2002. https://doi.org/10.1137/S1064827599364659.

[75] B. Moseley, A. Markham, and T. Nissen-Meyer. Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. *AdvCM*, 49(62), 2023. https://doi.org/10.1007/s10444-023-10065-9.

[76] M. Nastorg, J.-M. Gratien, T. Faney, M. A. Bucci, G. Charpiat, and M. Schoenauer. Multi-Level GNN Preconditioner for Solving Large Scale Problems. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2402.08296.

[77] B. Nathan, A. Frank, B. Timo, H. Aric, K. Yannis, N. Habib, P. Manish, P. Abani, S. James, W. Stefan, W. Karen, and L. Steven. Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence. Technical report, U.S. DOE Office of Science, Washington, DC, 2019. https://doi.org/10.2172/1478744.

[78] N. H. Nelsen and A. M. Stuart. Operator Learning Using Random Features: A Tool for Scientific Computing. *SIAM Rev.*, 66(3):535–571, 2024. https://doi.org/10.1137/24M1648703.

[79] J. Nocedal and S. J. Wright. *Numerical optimization*, pages 1–664. Springer Series in Operations Research and Financial Engineering. Springer Nature, 2006.

[80] Y. Notay. Flexible Conjugate Gradients. *SIAM J. Sci. Comput.*, 22(4):1444–1460, 2000. https://doi.org/10.1137/S1064827599362314.

[81] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti. Recycling Krylov Subspaces for Sequences of Linear Systems. *SIAM J. Sci. Comput.*, 28(5):1651–1674, 2006. https://doi.org/10.1137/040607277.

[82] L. Peairs and T.-Y. Chen. Using reinforcement learning to vary the m in GMRES(m). *Procedia Computer Science*, 4:2257–2266, 2011. https://doi.org/10.1016/j.procs.2011.04.246.

[83] A. Quaini, O. San, A. Veneziani, and T. Iliescu. Bridging Large Eddy Simulation and Reduced-Order Modeling of Convection-Dominated Flows through Spatial Filtering: Review and Perspectives. *Fluids*, 9(8), 2024. https://www.mdpi.com/2311-5521/9/8/178.

[84] A. Rahimi and B. Recht. Random Features for Large-Scale Kernel Machines. In *Neural Information Processing Systems*, 2007. https://api.semanticscholar.org/CorpusID:877929.

[85] M. Raissi. Deep Hidden Physics Models: Deep Learning of Nonlinear Partial Differential Equations. *J. Mach. Learn. Res.*, 19(25):1–24, 2018. http://jmlr.org/papers/v19/18-046.html.

[86] B. Raonić, R. Molinaro, T. D. Ryck, T. Rohner, F. Bartolucci, R. Alaifari, S. Mishra, and E. d. Bézenac. Convolutional Neural Operators for robust and accurate learning of PDEs. *arXiv*, 2023. https://doi.org/10.48550/arXiv.2302.01178.

[87] G. Rizzuti, A. Siahkoohi, E. Chow, and F. J. Herrmann. Learned Iterative Solvers for the Helmholtz Equation. In *81st EAGE Conference and Exhibition 2019*. European Association of Geoscientists & Engineers, 2019. https://doi.org/10.3997/2214-4609.201901542.

[88] F. Romor, G. Stabile, and G. Rozza. Non-linear Manifold Reduced-Order Models with Convolutional Autoencoders and Reduced Over-Collocation Method. *J. Sci. Comput.*, 94(74), 2023. https://doi.org/10.1007/s10915-023-02128-2.

[89] F. Romor, D. Torlo, and G. Rozza. Friedrichs' systems discretized with the Discontinuous Galerkin method: domain decomposable model order reduction and Graph Neural Networks approximating vanishing viscosity solutions. *arXiv*, 2023. https://doi.org/10.48550/arXiv.2308.03378.

[90] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. https://doi.org/10.48550/arXiv.1505.04597.

[91] H.-G. Roos, M. Stynes, and L. Tobiska. *Robust Numerical Methods for Singularly Perturbed Differential Equations*. Springer Berlin, Heidelberg, 2008. https://doi.org/10.1007/978-3-540-34467-4.

[92] 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 *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 42766–42782. PMLR, 2024.

[93] Y. Saad. A Flexible Inner-Outer Preconditioned GMRES Algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993. https://doi.org/10.1137/0914028.

[94] Y. Saad. *Iterative Methods for Sparse Linear Systems, 2nd ed.* SIAM, Philadelphia, 2003.

[95] Y. Saad and M. H. Schultz. GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, 1986. https://doi.org/10.1137/0907058.

[96] J. Sappl, L. Seiler, M. Harders, and W. Rauch. Deep Learning of Preconditioners for Conjugate Gradient Solvers in Urban Water Related Problems. *arXiv*, 2019. https://doi.org/10.48550/arXiv.1906.06925.

[97] M. Shpakovych. Neural network preconditioning of large linear systems. Research report, Inria Centre at the University of Bordeaux, 2023. https://hal.inria.fr/hal-04254315v1/.

[98] P. Siena, M. Girfoglio, F. Ballarin, and G. Rozza. Data-Driven Reduced Order Modelling for Patient-Specific Hemodynamics of Coronary Artery Bypass Grafts with Physical and Geometrical Parameters. *J. Sci. Comput.*, 94(38), 2023. https://doi.org/10.1007/s10915-022-02082-5.

[99] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comp. Phys.*, 375(15):1339–1364, 2018. https://doi.org/10.1016/j.jcp.2018.08.029.

[100] K. Sood. Iterative Solver Selection Techniques for Sparse Linear Systems. Ph.D. dissertation, University of Oregon, 2019. https://www.cs.uoregon.edu/Reports/PHD-201905-Sood.pdf.

[101] K. Sood, B. Norris, and E. R. Jessup. Lighthouse: a taxonomy-based solver selection tool. In *Proceedings of the 2nd International Workshop on Software Engineering for Parallel Systems*. ACM, 2015. http://dx.doi.org/10.1145/2837476.2837485.

[102] A. Stanziola, S. R. Arridge, B. T. Cox, and B. E. Treeby. A Helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound. *J. Comp. Phys.*, 441(2):110430, 2021. http://dx.doi.org/10.1016/j.jcp.2021.110430.

[103] E. Strubell, A. Ganesh, and A. McCallum. Energy and Policy Considerations for Deep Learning in NLP. *arXiv*, 2019. https://doi.org/10.48550/arXiv.1906.02243.

[104] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf.

[105] N. Tathawadekar, N. A. K. Doan, C. F. Silva, and N. Thuerey. Hybrid Neural Network PDE Solvers for Reacting Flows. *arXiv*, 2021. https://doi.org/10.48550/arXiv.2111.11185.

[106] J. Tompson, K. Schlachter, P. Sprechmann, and K. Perlin. Accelerating Eulerian Fluid Simulation with Convolutional Networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 3424–3433. PMLR, 2017. https://proceedings.mlr.press/v70/tompson17a.html.

[107] R. Trivedi, L. Su, J. Lu, M. F. Schubert, and J. Vuckovic. Data-driven acceleration of photonic simulations. *Sci. Rep.*, 9:19728, 2019. https://doi.org/10.1038/s41598-019-56212-5.

[108] C. Verburg, A. Heinlein, and E. C. Cyr. DDU-Net: A Domain Decomposition-based CNN for High-Resolution Image Segmentation on Multiple GPUs. *arXiv*, 2024.

[109] Y.-F. Xiang. *Solution of large linear systems with a massive number of right-hand sides and machine learning*. Ph.D. theses, Université de Bordeaux, December 2022. https://theses.hal.science/tel-03967557.

[110] K. Yamada, T. Katagiri, H. Takizawa, M. Kazuo, M. Yokokawa, T. Nagai, and M. Ogino. Preconditioner Auto-Tuning Using Deep Learning for Sparse Iterative Algorithms. In *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE, 2018. https://doi.org/10.1109/CANDARW.2018.00055.

[111] D. M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, 1971.

[112] A. G. Özbay, A. Hamzehloo, S. Laizet, P. Tzirakis, G. Rizos, and B. Schuller. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering*, 2(e6), 2021. https://doi.org/10.1017/dce.2021.7.

A Visualization of three 25×25 coefficient matrices

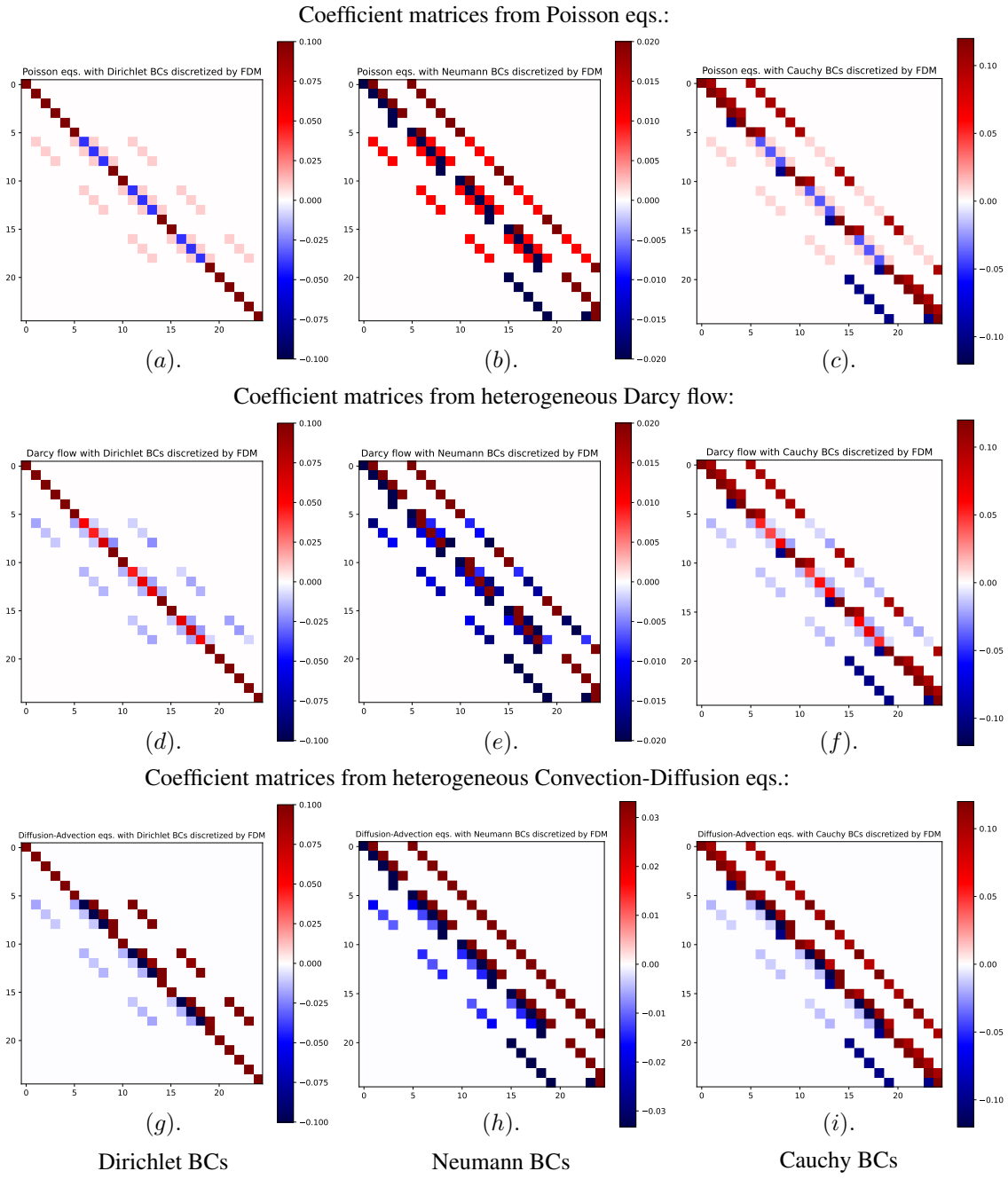
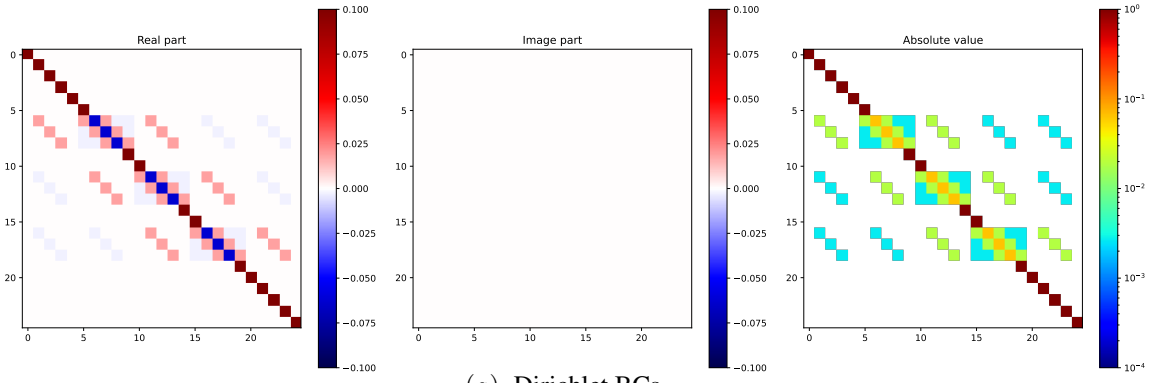
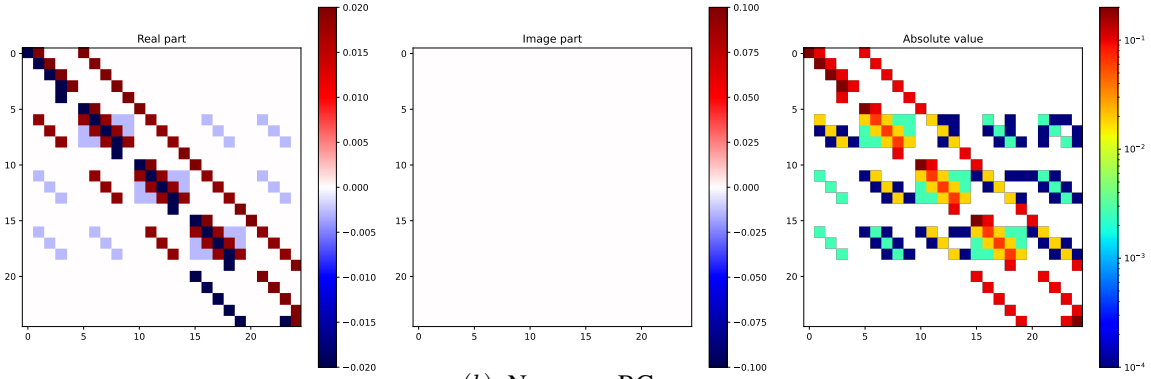


Figure 18: Visualize three 25×25 coefficient matrices (their elements have been scaled with 0.1 for better visualization) obtained from applying FDM discretisation for the three heterogeneous fluid equations with the three type of classical BCs shown in Equation (4).

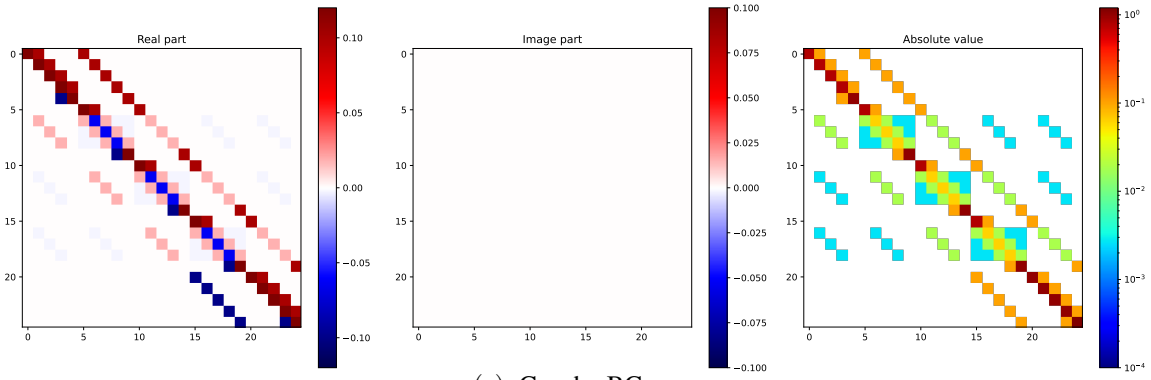
These 25×25 coefficient matrices discretized by FFT have zero image parts.



(a). Dirichlet BCs

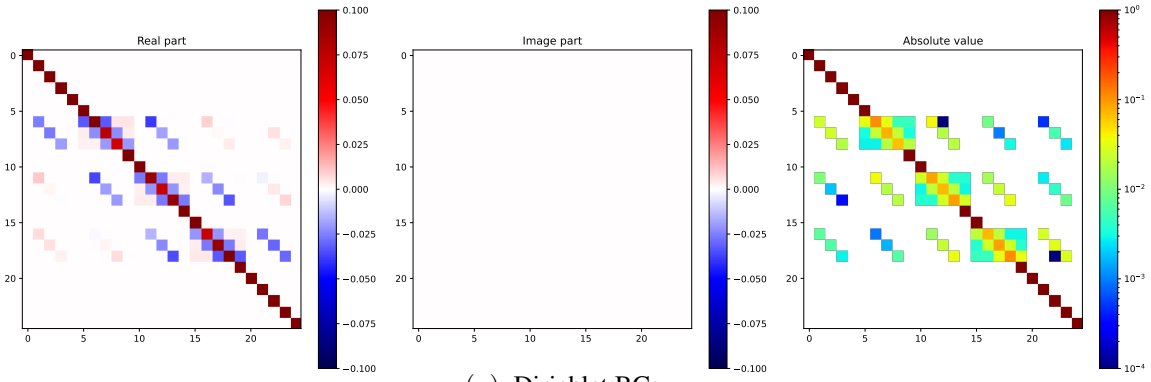


(b). Neumann BCs

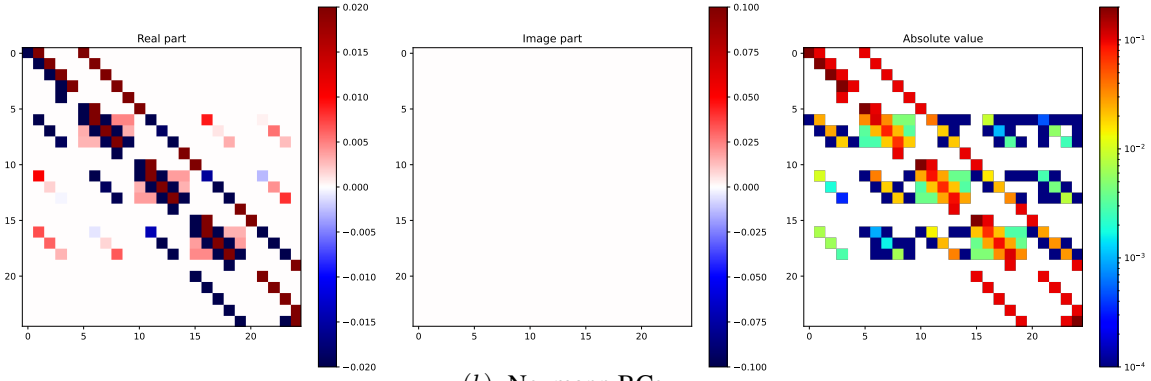


(c). Cauchy BCs

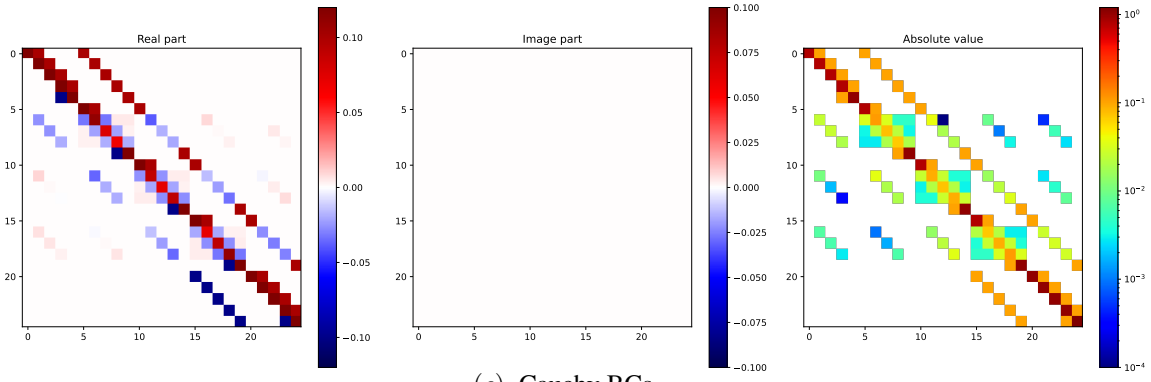
Figure 19: Visualize 25×25 coefficient matrices (their real and image elements have been scaled with 0.1 for better visualization) from Poisson eqs. discretized by FFT with the three type of classical BCs shown in Equation (4).



(a). Dirichlet BCs

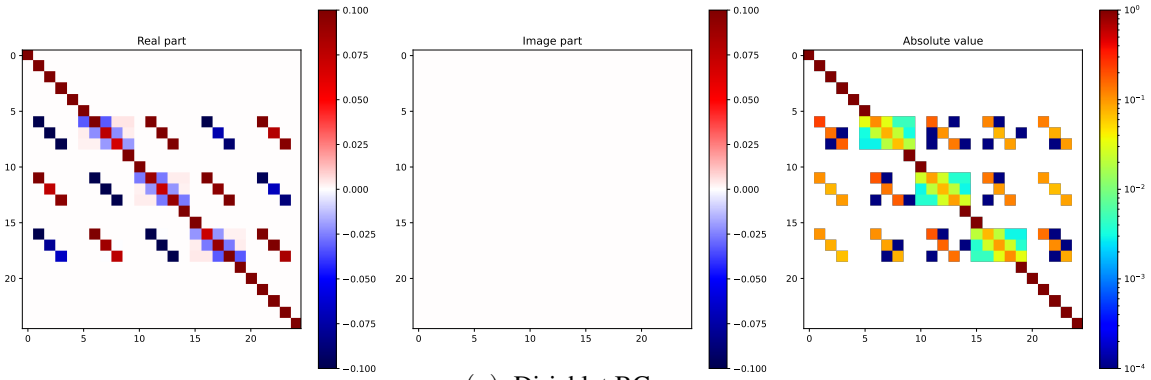


(b). Neumann BCs

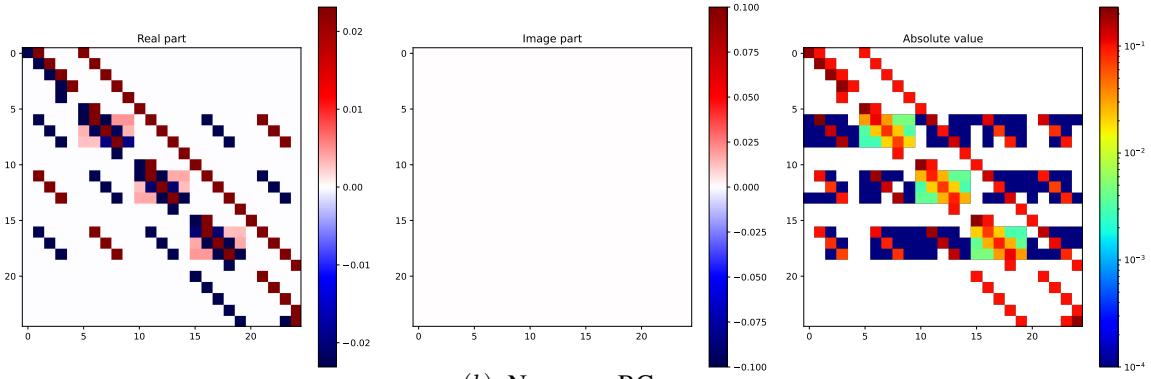


(c). Cauchy BCs

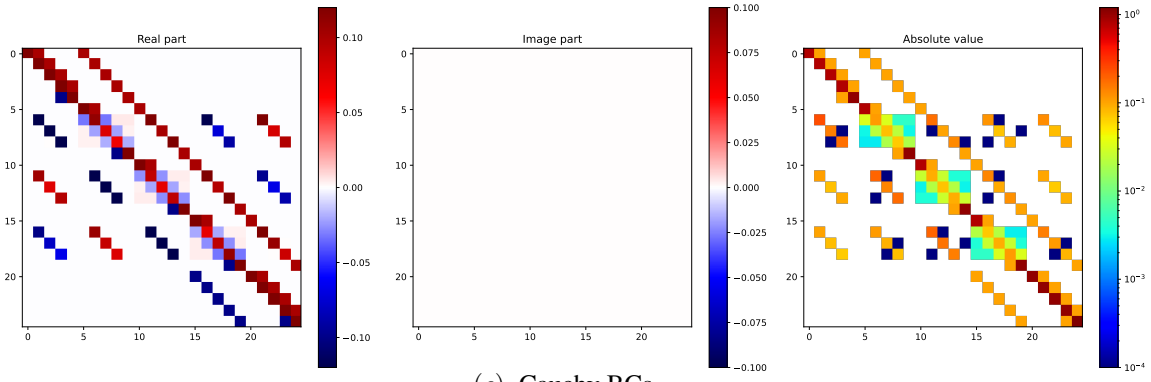
Figure 20: Visualize 25×25 coefficient matrices (their real and image elements have been scaled with 0.1 for better visualization) from Darcy flow discretized by FFT with the three type of classical BCs shown in Equation (4).



(a). Dirichlet BCs



(b). Neumann BCs



(c). Cauchy BCs

Figure 21: Visualize 25×25 coefficient matrices (their real and image elements have been scaled with 0.1 for better visualization) from Convection-Diffusion eqs. discretized by FFT with the three type of classical BCs shown in Equation (4).