Fast Resolution Agnostic Neural Techniques to Solve Partial Differential Equations

Hrishikesh Viswanath[a,*], Md Ashiqur Rahman[a], Abhijeet Vyas[a], Andrey Shor[a], Beatriz Medeiros[d], Stephanie Hernandez[d], Suhas Eswarappa Prameela[b,c,d], Aniket Bera[a]

[a]*Department of Computer Science, Purdue University, West Lafayette, IN, USA*
[b]*Department of Materials Science and Engineering, MIT, Cambridge, MA, USA*
[c]*Department of Aeronautics and Astronautics, MIT, Cambridge, MA, USA*
[d]*Hopkins Extreme Materials Institute, Johns Hopkins University, Baltimore, MD, USA*

arXiv:2301.13331v1 [cs.AI] 30 Jan 2023

Abstract

Numerical approximations of partial differential equations (PDEs) are routinely employed to formulate the solution of physics, engineering and mathematical problems involving functions of several variables, such as the propagation of heat or sound, fluid flow, elasticity, electrostatics, electrodynamics, and more. While this has led to solving many complex phenomena, there are still significant limitations. Conventional approaches such as Finite Element Methods (FEMs) and Finite Differential Methods (FDMs) require considerable time and are computationally expensive. In contrast, machine learning-based methods such as neural networks are faster once trained, but tend to be restricted to a specific discretization. This article aims to provide a comprehensive summary of conventional methods and recent machine learning-based methods to approximate PDEs numerically. Furthermore, we highlight several key architectures centered around the neural operator, a novel and fast approach (∼1000x) to learning the solution operator of a PDE. We will note how these new computational approaches can bring immense advantages in tackling many problems in fundamental and applied physics.

Keywords: Machine learning, Neural networks, Neural operators, Fourier neural operator, Geo-FNO, Graph neural operator, Physics informed neural operator, Finite element method, Finite volume method, Finite difference method, DeepONet, Spectral neural operator, Adaptive Fourier neural operator, Burgers equation, Darcy Flow equation, Navier Stokes equation, Kolmogorov Flow

1. Introduction

Partial differential equations (PDEs) are an integral tool in mathematically modeling the physical world. They allow one to describe how a quantity changes with respect to multiple variables and have allowed physicists to model various phenomena in fluid flow, electrodynamics, and quantum mechanics. An example family of generic PDEs can be represented as shown in equation 1,

$$\begin{aligned}(L_a u)(x) &= f(x), \quad x \in D, \\ u(x) &= 0, \quad x \in \delta D\end{aligned} \tag{1}$$

for some $a \in A, f \in L$, where A, L are Banach spaces, D is the domain of the PDE and $u : D \to \mathbf{R}, u \in U$ is the solution function. While PDEs are all around us, it is oftentimes very difficult for one to solve them analytically. The best that one can achieve is an approximation of the true solution of the PDE. The most popular approaches to solving PDEs are numerical methods such as finite difference methods (FDMs) Godunov & Bohachevsky (1959), finite element methods (FEMs) Zienkiewicz et al. (2005), and finite volume methods (FVMs) Eymard et al. (2000) as they are able to approximate solutions to PDEs with high amounts of accuracy. However, they are computationally expensive.

Finite difference methods solve PDEs by converting them into linear algebraic equations called finite difference equations. These equations are obtained by discretizing the domain of the functions involved in the PDEs and representing the derivatives as differences according to the first principles of calculus Jordan & Jordán (1965). The different discretization schemes result in different methods tailored to specific applications such as gas-dynamics Sod (1978) and heat transfer Özişik et al. (2017). An exhaustive study of applying this method to various families of PDEs has been published by Smith et al. (1985). Finite element methods, on the other hand, generate algebraic equations by applying fundamental physics laws on static or moving quantum of a system Hughes (2012) and have been used to solve several different problems in physics such as the fluid flow problem Donea & Huerta (2003)

*Corresponding author
Email address: hviswan@purdue.edu (Hrishikesh Viswanath)

and strain localization in metals Roters et al. (2011).

Numerical methods have traditionally been used to solve PDEs, but in an effort to reduce the computational cost and achieve greater accuracy, they are being replaced by data-driven methods in the current scientific era. These methods come under the overarching paradigm of machine learning approaches, which utilize data-driven algorithms that allow a program to learn and improve from experience. The modern face of machine learning is neural networks, which have led to a variety of advances in many scientific endeavors. Recent advances in deep learning, a subfield of machine learning which studies algorithms known as artificial neural networks, have allowed researchers to develop neural network architectures capable of approximating the solution of PDEs through large amounts of data. Deep neural networks that can approximate any function to an arbitrary precision Cybenko (1989) consist of multiple hidden layers and serve as a mapping between inputs and outputs LeCun et al. (2015). Deep neural networks have been applied to a multitude of problems in material physics, thermodynamics, and fluid dynamics problems Cai et al. (2022), Mao et al. (2020), Cai et al. (2021b), Thais et al. (2022) due to their innate ability to learn complex relationships between physical entities. To solve PDEs using neural networks, a dataset of inputs $\{x_i\}_1^n$ and their solution mappings $\{u(x_i)\}_1^n$ is used to train or learn an approximate function map $u^+ : D \to R$ of the solution function u, where R is the range of u. The accuracy of the predictions of u depends on the complexity of the function class u^+, which in turn depends on the neural network architecture. The ability of the network to predict $u(x)$ for $x \notin \{x_i\}_1^n$ is known as generalization and is vital to solving PDEs accurately. In particular, physics informed neural networks (PINNs) Raissi et al. (2019), Deep-ONet Lu et al. (2019), and neural operator Li et al. (2020a) architectures have shown great success in learning PDEs.

Neural operator is an emerging deep learning technique that is different from a typical neural network in several ways. While neural networks are able to approximate any function, which is a map between finite-dimensional spaces Cybenko (1989), to approximate an operator, which is a map between infinite dimensional spaces, a network of infinite length is required Guss & Salakhutdinov (2019). Thus, neural networks fail to accurately approximate solution operators of PDEs, which are mappings between infinite dimensional spaces. The need for more accurate solutions has motivated the development of neural operators: a generalization of neural networks to map between infinite dimensional spaces Kovachki et al. (2021). Neural operators are composed of linear integral operators and nonlinear activation functions. The result is an operator capable of approximating highly nonlinear solution operators of PDEs. Furthermore, neural operators are resolution-invariant and up to ≈1000x faster than traditional neural networks in approximating the solution of PDEs Li et al. (2020a). Current state-of-the-art neural operator architecture includes the graph neural network (GNO) Li et al. (2020c), Fourier neural network (FNO) Li et al. (2020a) and its variant geo-Fourier neural network (Geo-FNO) Li et al. (2022), and physics informed neural network (PINO) Rosofsky & Huerta (2022).

This paper aims to provide a brief overview of popular numerical methods and emphasize on several machine learning-based methods to numerically approximate PDEs. Figure 1 provides an overview of both conventional and machine learning-based approaches to solving PDEs. We will highlight several key architectures centered around the neural operator, a novel and fast approach (1000x) to learning the solution operator of a PDE. We will note how these new computational approaches can bring immense advantages in tackling many problems in fundamental and applied physics. Sections 2 and 3 describe some of the conventional and neural network-based architectures, while sections 4 through 8 talk about different types of neural operator-based architectures. Section 9 discusses various fields where neural operators have been successful. Tables 6 and 5, in the appendix section, provide the definitions of the mathematical notations and the GitHub links to neural operator models respectively.

2. Conventional Solvers

In this section, we will discuss some of the conventional approaches that have been used to solve PDEs, such as finite difference methods, finite element methods and finite volume methods.

2.1. Finite difference method

The finite difference method (FDM) is a class of grid-point techniques in numerical methods used to approximate the solution to differential equations. FDMs work by discretizing a function in an infinite domain into a finite, space-time grid Moczo et al. (2004). Each point in the grid represents a value of the particular function being approximated. FDMs are able to approximate the solution to a PDE at each grid-point by approximating the derivatives with finite difference formulas Godunov & Bohachevsky (1959). The more grid-points available, the more accurate the solution.

FDMs have been used to help understand earthquake physics Aochi et al. (2013), conservation laws Sod (1978), and fluid flow simulations Fadlun et al. (2000). Due to its simplicity, the FDM is very fast at solving PDEs on structured grids. However, FDM struggles with complex geometries and is often less accurate than finite element

and finite volume methods.

2.2. Finite element method

The finite element method (FEM) is a numerical method for solving PDEs in which a complex domain is discretized into a collection of small, simple domains. These simple domains are referred to as finite elements and are used to construct approximation functions over each element Reddy (2019). The FEM creates a sparse matrix representing the discretization and solves it via a sparse matrix solver to achieve a global solution. The FEM allows one to solve complex PDEs since the method is adaptable over difficult domains. Furthermore, the FEM offers very accurate results dependent on discretization. The disadvantages of the FEM are that it is computationally expensive and that the accuracy of the solution depends on the resolution of the mesh. The FEM is utilized heavily in problems involving complex geometries and for modeling heat transfer Wilson et al. (1974), electromagnetic potential Li et al. (2017), and quantum mechanics Searles & von Nagy-Felsobuki (1988).

2.3. Finite volume method

Finite volume methods (FVM) are capable of evaluating PDEs numerically through the use of algebraic equations. FVMs utilize a volume integral formation of the PDE and discretize the geometry of the PDE through the use of a finite collection of volumes. Then, FVMs evaluate each volume integral at each volume. FVMs are often used to approximate solutions to physical problems that arise from conservation laws. These include computational fluid dynamics Acharya et al. (2007), dynamic solid mechanics Slone et al. (2003), and stress analysis Demirdžić & Muzaferija (1994). The key advantage of FVMs is the ability to be used on unstructured grids and complex geometries. However, FVMs are computationally intensive and usually restricted to first- and second-order PDEs.

3. Neural network architectures

In this section, we go over some of the most commonly used neural network based architectures for solving PDEs, such as fully connected neural networks, convolutional neural networks and physics informed neural networks. A neural network is a mesh of layers, where each layer consists of a set of nodes, which computes the weighted sum of the input and applies a non-linear activation to it. These networks are also called feed-forward networks because the input flows forward from the input layer to the output layer through one or more hidden layers Bebis & Georgiopoulos (1994).

3.1. Fully Connected neural networks

Fully connected neural networks (FCNs) are the most basic neural network architecture. They consist of a multitude of layers, including an input layer, one or more hidden layers, and an output layer Yegnanarayana (2009). These types of networks are considered "feed-forward" because the information in this structure only moves forward, from one layer to the next, until it reaches the output layer Sazli (2006). This architecture has been an attractive approach to solving PDEs because of its generalizability Lagaris et al. (1998).

Many approaches have been proposed to approximate the solution to PDEs using FCNs. One such approach is to create a model consisting of the sum of two terms. The first term is used to satisfy the boundary conditions, whereas the second term is an FCN that is trained to satisfy the PDE itself Lagaris et al. (1998). An FCN can be used here to approximate the solution to the PDE because FCNs are universal function approximators Hornik et al. (1989). Through this approach, one can achieve a differentiable, closed analytic form of the solution to the PDE Lagaris et al. (1998). While this approach can be applied to both PDEs and ODEs, it struggles to handle PDEs of higher dimensions as the number of training points increases greatly. This, in turn, makes the proposed method computationally exhaustive. Furthermore, this approach is less precise than Finite Element methods in approximating a solution of the PDE Lagaris et al. (1998).

Another common issue is the difficulty in solving PDEs with complex boundary conditions. In fact, there are times when numerical methods cannot approximate solutions to such PDEs Tadmor (2012). The method proposed in Sirignano & Spiliopoulos (2018) is known as the deep Galerkin method (DGM). This method utilizes an FCN that is trained on a set of randomly sampled time and space points. The key strength of this method is that it is able to satisfy the differential operator and initial/boundary conditions. Furthermore, this approach is meshfree; hence it can approximate solutions to PDEs that standard numerical methods may struggle with. Lastly, this method is capable of solving PDEs in high dimensional spaces, unlike the approach described by Lagaris et al. (1998).

While these approaches have shown success in solving PDEs, there are still many limitations. One such limitation is that FCNs are unable to capture spatial and temporal data, primarily because of how FCNs process data. FCNs deal with data in a one-dimensional spatially-invariant format. Therefore it is hard for FCNs to capture spatial information Livingstone et al. (1997). Another issue with FCNs is that they cannot outperform numerical methods in low-dimensional spaces Lagaris et al. (1998). Lastly, FCNs primarily depend on a specific

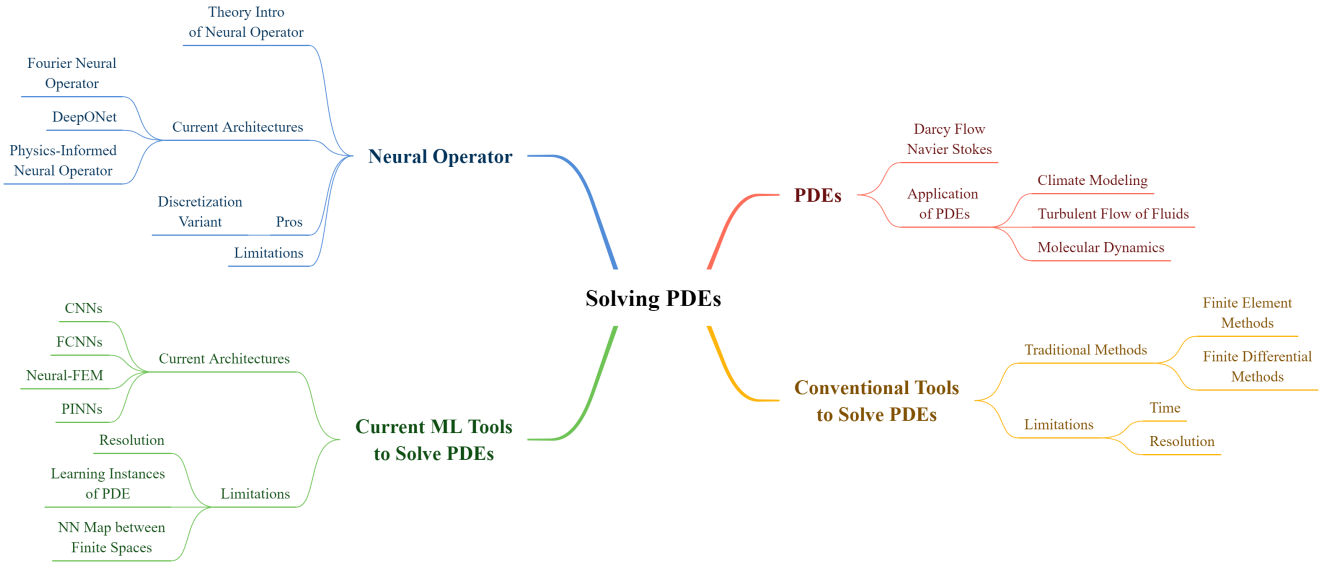


Figure 1: The above figure represents the logical flow of how PDEs can be solved using various methods, highlighting existing ML techniques and various families of neural operator based techniques

discretization of the grid from which training points are sampled and hence need to be retrained each time the discretization changes Li et al. (2020b).

3.2. Convolutional neural networks

While FCNs can approximate PDEs, they fail to capture potentially vital information, such as spatiotemporal features. CNNs are the key to learning spatial features O'Shea & Nash (2015). Traditionally, the CNN architecture has been utilized in image processing tasks Browne & Ghidary (2003), Naranjo-Torres et al. (2020), Ciresan et al. (2011), but have in recent years been trained to solve PDEs due to their ability to learn complex patterns in image-formatted data. CNNs have been shown to be able to approximate the solution of elliptical PDEs Winovich et al. (2019) and have been the foundation for hybrid architectures such as PhyGeoNet Gao et al. (2021).

Standard CNN architecture differs from FCN architecture by utilizing three distinct layers: convolutional layers, pooling layers, and fully-connected layers O'Shea & Nash (2015). As the name suggests, the convolutional layers apply convolution to the input. This allows the network to learn local features in an image. The pooling layers are applied to convolutional layers in order to further generalize the local features. Pooling can be a function such as max, min or avg, which is applied to subsets of the input. The resulting matrix will have fewer elements than the original. Lastly, the fully connected layers are utilized in order to aid with classification problems Aghdam & Heravi (2017).

ConvPDE-UQ is a framework that utilizes CNNs to construct lightweight numerical solvers that can solve elliptical PDEs Winovich et al. (2019). These lightweight solvers utilize the representative power of a neural network to approximate the solution to a PDE on a general domain in a single forward pass by using Green's functions Winovich et al. (2019). This allows the solvers to be more computationally efficient than conventional solvers such as FEMs. One key limitation of this architecture is that it cannot be easily extended to modeling systems that are inhomogeneous and contain mixed boundary conditions Winovich et al. (2019).

Physics-informed geometry-adaptive convolutional neural networks (PhyGeoNet) are an extension of CNNs. This architecture utilizes convolutional neural networks (CNNs) with a physics constraint. CNNs are often utilized when one needs to learn spatiotemporal data O'Shea & Nash (2015). The key disadvantage of CNN architecture is that it can only handle rectangular grids; hence utilizing a pure CNN architecture on complex geometries is challenging. PhyGeoNet is a physics-constrained CNN architecture capable of learning solutions to parametric PDEs on irregular geometries and nonuniform grids without the need to utilize labeled data Gao et al. (2021). This architecture is capable of learning the solutions to parametric PDEs by utilizing an elliptic mapping from the irregular physical domain to a regular reference domain Gao et al. (2021). While PhyGeoNet shows promise in solving PDEs, the architecture is constrained by the fact that it is only capable of solving steady-state parametric PDEs and thus cannot be applied to dynamic systems.

3.3. Physics informed neural networks

PINNs are a neural network architecture that utilize the laws of physics to help guide their training. In deep learning paradigms, neural networks learn by minimizing a convex loss function. In machine learning, the loss function typically contains the loss (ex: L2-Norm) and a regularization term which helps alleviate the risk of overfitting. With PINNs, the loss function also contains a knowledge-based term Raissi et al. (2019). The knowledge-based term allows one to incorporate existing physical laws into the training of the neural network. This allows for a mapping that allows the neural network to be consistent with existing physical laws.

Since PDEs most frequently define physical laws, PINNs can be used to learn the solution to PDEs. The loss function in PINN architecture embeds boundary conditions, initial conditions, PDE residuals for a finite amount of points, and space-time domain boundary Cuomo et al. (2022) into the knowledge-based term. Upon training, PINNs are able to map between an input point in the integration domain and the estimated solutions of the differential equation at that point. Unlike other supervised learning techniques, PINNs are capable of taking into account the underlying physics of the problem instead of just using existing data.

Traditional PINNs have shown limited accuracy in solving PDEs, especially on multi-scale dynamic systems. A variation that is capable of addressing this limitation is gradient-enhanced PINNs (gPINNS). Traditional PINNs use the knowledge-based term to encode the PDE residual into the loss function; however, gPINNs also utilize the gradient information of the PDE residual in the loss function Yu et al. (2022), allowing for much more accurate results after training.

Another typical issue with PINNs when solving nonlinear PDEs is when those PDEs have discontinuous solutions. Hybrid PINNs utilize ideas from convolutional neural networks (CNNs) and FVMs in their architecture. Instead of using automatic differentiation to solve the PDE, this method utilizes an approximation to the differential operator, hence allowing the architecture to have a convergent rate and avoiding the problem of discontinuity Fang (2021).

4. Neural operator based models

To address the challenges posed by traditional neural network-based architectures in solving PDEs, a new learning mechanism called operator learning was proposed by Li et al. (2020a). The intuition behind operator learning was to design mesh-independent, resolution-invariant models to solve PDEs. This means one could train a model on a 40x40 matrix but test it against 256x256 matrices. The primary difference between a neural network and a neural operator is that while a neural network learns the mapping between finite-dimensional spaces, a neural operator is designed to learn the mapping between functional spaces. The key characteristic of a neural operator is that it attempts to learn the mapping between two infinite dimensional spaces using a finite collection of input-output pairs. The input could be a Gaussian field, and the output could be the solution of the PDE on the field. An important advantage of neural operators is that they are mesh-independent because a single set of parameters may be used for different discretizations Li et al. (2020a). The training requires only the input-output pairs and no knowledge of the PDEs being learned.

Let G^+ be a (potentially non-linear) map from the input space A to the solution space U, where A and U are separable Banach function spaces and take the values $\mathcal{R}^{d_a}$ and $\mathcal{R}^{d_u}$, i.e., if we draw a_j and the corresponding solution u_j from A and U respectively, then $u_j = G^+(a_j)$. The neural operator is a parametric map that aims to approximate G^+. This approximation is the operator G denoted as G: A x $\Theta \rightarrow U$. The learning problem would then be an optimization problem where the cost function $\mathcal{C} : U \times U \rightarrow \mathcal{R}$ is an L_2 Bochner norm-based distance generating function defined on U. The problem then becomes $\min_{\theta \in \Theta} \mathbf{E}_a[\mathcal{C}(G_\theta(a), G^+(a)]$, which would be the discrepancy in the actual map versus the operator. While a neural network learns a map from the input domain D to an element in U using samples of input-output pairings $\{x_i, u(x_i)\}_{i=1}^n$, a neural operator learns a mapping between A to U using samples of $\{u_i, a_i\}_{i=1}^n$ The algorithm, as proposed in Li et al. (2020a) is an iterative approach and is explained in the following subsections.

4.1. Operator Learning

As presented in Kovachki et al. (2021), operator learning is a 3-step process - *lifting*, *iterative kernel integration*, and *projection*. The first step involves mapping the input a to the first hidden state v_0 by a pointwise function $\mathcal{R}^{d_a} \rightarrow \mathcal{R}^{d_{v_0}}$. This is performed by a local operator. d_{v_0} is chosen such that it is bigger than d_a. The second step, or the iterative Kernel Integration, is when each hidden representation is mapped to the next hidden representation by summing a local linear operator, a non-local integral kernel operator, and a bias function. The sum has a fixed point-wise non-linearity. $\{v_t : \mathbf{D_t} \rightarrow \mathcal{R}^{d_{v_t}}\} \rightarrow \{v_{t+1} : \mathbf{D_{t+1}} \rightarrow \mathcal{R}^{d_{v_{t+1}}}\}$, for all $t = 1...T-1$. The final projection step is when the representation corresponding to the final hidden layer, v_T, is mapped to the output u. in this case, the space d_{v_T} is larger than the space d_u. This operation is once again performed by a local operator. These sequences of operation are very similar to the functioning of a finite-dimensional neural network, and as presented in

Li et al. (2020a), they can be mathematically written as follows:

$$G_\theta = Q \circ \sigma_T(W_{T-1} + \kappa_{T-1} + b_{T-1}) \circ ... \circ \sigma_1(W_0 + \kappa_0 + b_0) \circ P \tag{2}$$

In equation 2, P and Q are the lifting and projecting maps, respectively, acting point-wise, projecting to higher and lower dimensional spaces. W are the point-wise linear operations, κ are the kernel integral operations, b are the biases, and σ is some non-linear activation function acting point-wise.

5. DeepONet

Deep operator network (DeepONet) is formulated around the observation that a neural operator is a universal operator approximator Lu et al. (2019). The universal approximation theorems for neural operators are derived following the key result that a single non-linear layer in a neural network is capable of approximating any continuous operator Chen & Chen (1995).

The DeepONet architecture consists of two encoder networks. The first encoder is used for encoding the input function and the second encoder is used for encoding the location of the output functions. This architecture has been shown to significantly outperform traditional fully connected neural network architecture on dynamic systems and PDEs. The key strength of this architecture lies in its capability of accurately approximating complex mappings between infinite dimensional Banach spaces, hence making it a natural choice for being utilized in learning the solution operator to a PDE.

While the original DeepONet architecture has shown its ability to learn PDEs, its main downside is the high amounts of training data required. Oftentimes, generating this data can be very computationally expensive, and even with large training datasets, DeepONet may fail to learn the underlying physical principles of the equation. Physics-informed DeepONets are advanced DeepONets that utilize these physical principles as a regularization mechanism, hence leading to predictions that abide by the governing physical laws described. Automatic differentiation is used to integrate these physical laws as penalties while the model trains Wang et al. (2021).

DeepONet and its variants have been used to solve a plethora of problems, including material physics Goswami et al. (2022), electrodynamics Cai et al. (2021a), and aerothermodynamics Sharma Priyadarshini et al. (2021). A particularly interesting use of DeepONet comes from utilizing this model for medical image simulation, particularly for aortic dissection. Aortic dissection is a lethal condition which is characterized by the tear of the aorta, which is a blood vessel used to deliver oxygen to the rest of the body DeSanctis et al. (1987). The DeepONet architecture has been used to predict dissection progression in a heterogeneous aortic wall, which is a very complex physical environment Yin et al. (2022).

Since the DeepONet architecture is capable of learning the non-linear operator between infinite dimensional Banach spaces, it is capable of learning oscillatory continuous functions. These types of functions are often found in earthquake physics. An interesting problem that DeepONet has been able to solve is how buildings respond to seismic excitation over a certain time period. The DeepONet architecture utilizes scaling techniques that convert a high-frequency function to a low-frequency function, which then allows this architecture to learn a range of frequencies for the low-frequency function. This multi-scale architecture, known as multi-scale DeepONet, has allowed scientists to map seismic excitation to building responses Liu & Cai (2021).

6. Graph neural operator

Graph neural network is a machine learning architecture that is built upon the traditional neural network. The additional information that a graph neural network aims to capture are interactions between constituent objects, which in the case of the PDE, would be the kernel. Traditional neural networks do not capture such relationships. A graph neural network represents these relationships as edges between features, which are represented as vertices. Learning relationships between these objects occurs through a process called message passing. This technique allows nodes to learn the information held by neighboring nodes through an aggregating function, which combines the embeddings (feature vector) of the neighboring nodes. An update function learns the new embeddings for the nodes. This process is described in equation 3.

$$v_{t+1}(x) = \sum_{y \in N(x)} F(h_x, h_y, e_{xy}) \tag{3}$$

In the above equation, the new embedding is an aggregate function of the current embedding with the embeddings of all neighboring nodes and F is an arbitrary function of hidden layer embeddings h_x, h_y and the edge e_{xy}. Message passing allows the network to learn the kernel, which in turn approximates the solution to the PDE.

Graph neural networks, however, ignore long-range relationships because of scaling issues. These approximations of relationships are ineffective when trying to solve a PDE since they lead to issues with generalization. To capture long-range relationships, Li et al. (2020c) discuss multipole methods. They propose an architecture of linear complexity called multipole graph neural operator, which captures long-range relationships between objects.

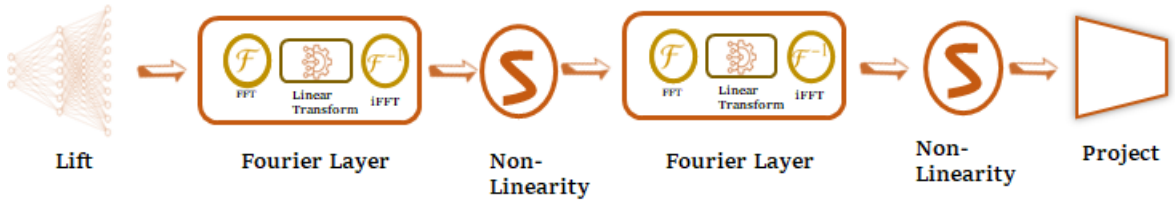


Figure 2: Architecture of the basic Fourier neural operator

The model recursively adds points to the kernel matrix and is equivalent to the multi-level formulation. The long-range relationships are modeled through the addition of inducing points, which form sub-graphs.

6.1. Kernel operator

Kernel integration is a key component of the iterative learning process that occurs in the Graph Neural operator. This process helps learn the mapping between function spaces in PDEs. The process is described in equation 18 and is equivalent to message passing in graphs. Every spatial point in the discretized input can be represented by x-y coordinates. The values at that point can be represented as functions a(x) and a(y) from the input function space. A kernel operator can be defined to act on these data values in the following manner

$$(K_a u)(x) = \int_D k_\phi(a(x), a(y), x, y)u(y)dy \tag{4}$$

Equation 4 can then be converted to an iterative neural network-based architecture in equation 5.

$$v^{(t)} = \sigma((W + K_a)v^{(t-1)}) \tag{5}$$

6.2. Kernel decomposition

The interactions between the nodes of the graph are divided based on their range of interaction. The decomposition of a single kernel into a series of kernels, each representing a different range of interaction, is the feature responsible for the linear complexity of the graph neural operator. The graph is broken down into L levels, where the first level represents the shortest level interactions while the last level, L, is the coarsest and represents the longest range of interactions. The first level is full rank but very sparse, but the last level is dense but of low rank. The coarse graph is derived recursively from the dense graph through the inducing points method or Nystrom approximation, which is shown in equation 6.

$$K_{nn} \approx K_{nm}K_{mm}K_{mn} \tag{6}$$

Nystrom approximation allows the graphs to be unstructured and of arbitrary sizes.

6.3. V cycle Algorithm

The authors Li et al. (2020c) proposed V-cycle algorithm to iteratively and efficiently compute the matrix factorization. The algorithm contains two passes - the downward pass, where the algorithm starts with the fine graph and updates it by applying a downward transition. In the upward pass, the algorithm starts with the coarse graph and updates it by applying an upward transition. The two passes are defined in equations 7 and 8, respectively.
Downward pass

$$\check{v}_{l+1}^{(t+1)} = \sigma(\hat{v}_{l+1}^{(t)} + K_{l+1,l}\check{v}_l^{(t+1)}) \tag{7}$$

Upward pass

$$\hat{v}_l^{(t+1)} = \sigma((W_l + K_{l,l})\check{v}_l^{(t+1)} + K_{l,l-1}\hat{v}_{l-1}^{(t+1)}) \tag{8}$$

The above process facilitates multi-resolution matrix factorization. This process, combined with Nystrom approximation, leads to O(m) complexity, which has been proven through empirical studies. The kernel integration and the V-cycle algorithm allow the GNO to learn mesh invariant solutions to parametric PDEs and make it invariant to different discretizations. However, in problems with regular meshes, GNO architectures are outperformed by Fourier neural operators.

7. Fourier neural operator

Fourier neural operators (FNOs) are another class of neural operators that use the properties of the Fourier transforms to perform the calculation of the integration operation at each layer of the neural operator. The motivation behind using Fourier transforms instead of convolution is that they are faster, and PDEs are inherently continuous, and representing them in Fourier space is more efficient. Convolution in physical space is equivalent to multiplication in Fourier space. Linear transformations are therefore performed in the Fourier space. However, activation functions are applied in the original space because they help recover non-periodic boundaries, which are left by the Fourier space. The entire pipeline is shown in Figure 2.

7.1. Architecture

Fourier neural operator utilizes the operator learning technique implemented by Li et al. (2020a). It has been applied in spatial and temporal settings, such as modeling the flow of fluids with time, specifically for weather modeling and forecast. As specified in Li et al. (2020a), The iterative updates to a state v_i to reach the state v_{i+1} is denoted by equation 9.

$$v_{i+1}(x) = \sigma(W v_i(x) + \kappa v_i(x)) \tag{9}$$

In the above equation,κ denotes the kernel integral transformation which is defined in equation 10.

$$[\kappa v_t](x) = \int_D \kappa(y) v_t(y) dy \tag{10}$$

The kernel integral operator is replaced with a convolution operator defined in the Fourier space. This becomes the basis for the Fourier neural operator. Equations 11 and 12 define Fourier transform and inverse Fourier transform.

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle} dx \tag{11}$$

$$(\mathcal{F}^{-1}f)_j(x) = \int_D f_j(k) e^{2i\pi\langle x,k\rangle} dk \tag{12}$$

The kernel integral operator can be represented as follows

$$[\kappa v_t](x) = \mathcal{F}^{-1}(\mathcal{F}(\kappa)\mathcal{F}(v_t))(x) \tag{13}$$

In equation 13, $\mathcal{F}(\kappa)$ is the Fourier transform of a periodic function κ and can therefore be approximated as a Fourier series. By truncating the series at a maximal number of modes k_{max}, the series can be represented with a finite-dimensional parameterization.

8. Variants of Fourier neural operator

The neural operator has been fine-tuned and modified for specific instances and situations. The following section discusses the different variations of the basic neural operator architecture. Figures 3 and 4 illustrate the flow of various neural operators. 5 provides links to the source code for various neural operators.

8.1. GANO and UNO

Generative adversarial neural operator Rahman et al. (2022a) and U-shaped neural operator Rahman et al. (2022c) are the operator variants of Generative adversarial networks and U-Net, which are typically used for generative modeling. These two architectures allow for memory-efficient implementations of deeper neural operators. The generator network of GANO and the UNO network have similar architecture, comprising an encoder network and a decoder network with skip connections. However, in this paradigm, the input function spaces are mapped to vector-valued function spaces with steadily decreasing domains in the encoder network and increasing domains in the decoder network.

GANOs also use a discriminator neural functional, anologous to the discriminator network in a GAN, to facilitate adversarial learning. This discriminator differentiates between the generated solutions to the PDE and the ground truth PDE solution. This network is optimized for Polish function spaces. The adversarial learning is done using Wasserstein formulation, which allows for a bounded norm in infinite dimensional space. This architecture is both resolution and discretization invariant, similar to the basic FNO an is good at learning probability measures on function spaces. Another advantage of GANOs over regular GANs is that they do not suffer from modal collapse.

8.2. Adaptive Fourier neural operator

AFNO is a powerful sequence-to-sequence generative model similar to a Vision Transformer that is built by stacking individual operator networks. The network learns the representations through token mixing in the Fourier domain. The model treats tokens as continuous objects in infinite space. The network has a block diagonal structure, where the weight matrix is divided into weight blocks and the kernel operates on them independently. This architecture is the basis for FourcastNet Pathak et al. (2022), a weather prediction model that has shown tremendous promise in forecasting weather, matching the accuracy of ECMWF Integrated Forecasting System. Weather systems are applications of complex physical entities that can be modeled as PDEs. Generative models such as VFiT and FourCastNet can be used to model other complex physical systems such as meteor trajectories, volcanoes, predicting the deformations on the surface of space vehicles, etc.

8.3. Implicit Fourier neural operator

Implicit Fourier neural operator (IFNO) was designed by You et al. (2022) address some of the limitations of the basic Fourier neural operator network such as it's tendency to overfit as the number of layers increases and it's susceptibility to vanishing gradient. IFNO is an integral operator based architecture where the operator learning is defined to be an implicit mapping, modeled as a fixed point. Through this technique, the PDEs can be implicitly expressed as a set of equations that can be solved using methods such as the Newton-Raphson method, where approximations to the solution $\mathbf{V}$ are successively improved until a precise value is achieved. This process is defined in equation 14.

$$\mathbf{V}[l+1] = \mathbf{V}[l] + \mathcal{R}(\mathbf{V}[l], F) \tag{14}$$

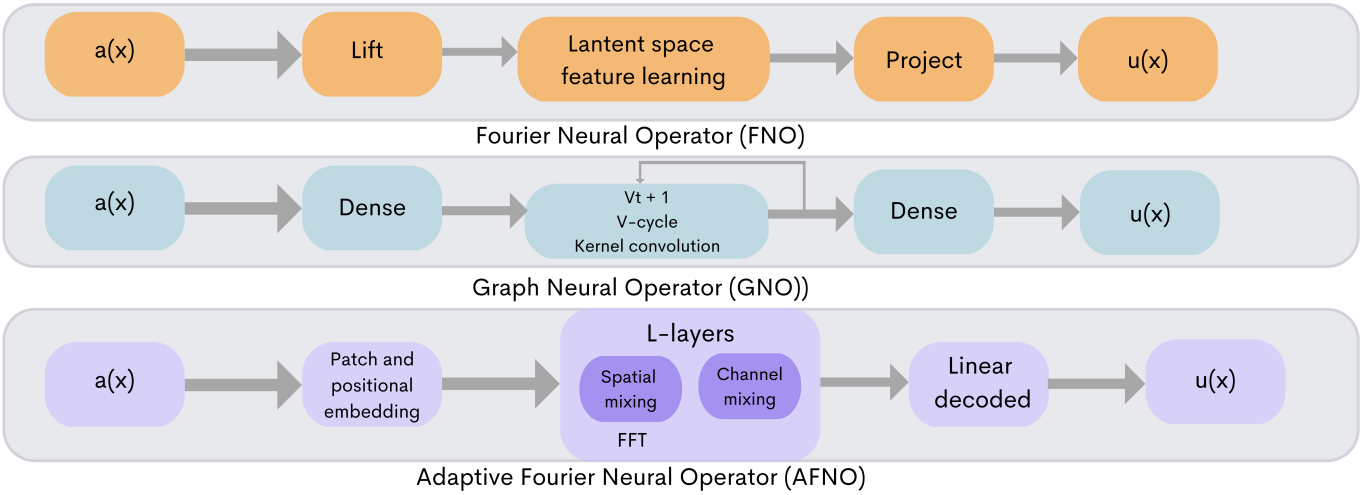


Figure 3: The logical pipeline of the main neural operator architectures. The above figure highlights the differences in the building blocks of the neural architecture

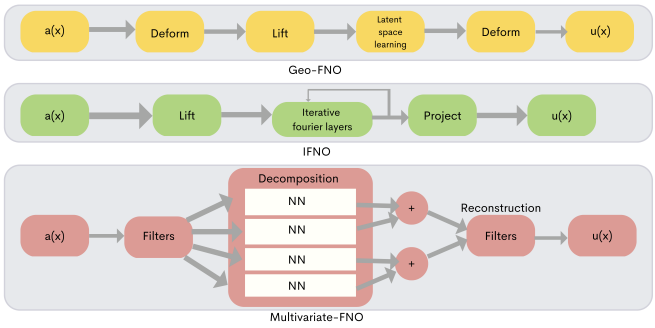


Figure 4: The logical pipeline of geo-FNO, implicit FNO and multivariate FNO, which are all modified versions of the basic neural operator architecture

where $\mathbf{V}[l]$ is the current approximation of the solution, $\mathcal{R}$ is the operator that improves the approximation, and F is the input vector. The network update equation is denoted in equation 15.

$$f(x,\Delta t) = f(x,\Delta t) + \sigma(Wf(x,\Delta t) + \mathcal{F}^{-1}(\mathcal{F}(\kappa v_t)\mathcal{F}(f(.;\Delta t)))(x) + b) \quad (15)$$

The key difference between this method and the vanilla FNO is that, in this case, the parameters of the hidden layers are considered independent. The total number of trainable parameters does not depend on the number of layers. As the layer deepens, it becomes the analog of discretized ordinary differential equations. The weight update is simply a time discretization of the ODE.

You et al. (2022) further showed that this architecture can model heterogenity and material defects. These claims were empirically verified against hyperelastic, brittle and anisotropic materials.

The specific applications where this method showed promising results were as follows

- Porous medium flow in a 2-dimensional setting with heterogenous permeability field.
- Fiber material deformation in hyperelastic and anisotropic settings, represented by the Holzapfel-Gasser-Odgen (HGO) model. Two boundary conditions were considered - the Dirichlet boundary with uniform uniaxial displacement applied on the right, and top edges and the Neumann boundary with uniaxial tension applied on the top edge.
- Fracture mechanisms in brittle glass ceramics, which as modeled as the Darcy-Flow equation.

8.4. Geo Fourier neural operator

Geo-Fourier neural operator is an architecture developed by Li et al. (2022) and is designed to be 'geometry-aware'. This architecture addressed the input limitations of regular FNO, which is that it only works with rectangular domains with uniform meshes. FNOs were made to work in irregular domains by embedding them within larger rectangular domains, however, this was an inefficient way to represent irregularity, and geo-FNO was designed to fix this issue. Rather than embedding in a larger regular mesh, geo-FNO deforms the non-uniform meshes into uniform meshes on which FFT can be applied. This architecture can be used when the input is represented by non-uniform meshes or point clouds or if the input domain is represented as signed distance functions. The network performs two operations - deforming the non-uniform input into a uniform mesh, followed by operator learning in latent space. Traditional methods do not work when the input is deformed because the deformed mesh in Fourier space does not correspond to the original system.

However, since the FNO approximates through training data, it is not constrained by this limitation.

The deformation from the physical space to the computational space is done using an adaptive moving mesh defined by a coordinate transformation. Geo-FNO can be used in both structural and fluid mechanics problems, and it performs well on Euler's Equation for flow over the airfoil and the plastic forging problem defined by equation 16.

$$\rho^s \frac{\partial^2 \mathcal{U}}{\partial t^2} + \nabla . \varsigma = 0 \tag{16}$$

In the above equation, ρ^s refers to the mass density, $\mathcal{U}$ is the displacement and ς is the stress tensor. However, geo-FNOs have only been tested on regular homeomorphic topologies and more studies are needed to exhaustively determine the range of problems that can be solved by this architecture.

8.5. Multiwavelet neural operator

The motivation for this architecture stems from the fact that the solution to any PDE can be found by learning the inverse operator mapping from input to output. To do this, the authors of Gupta et al. (2021) suggest decomposing the kernel using fine wavelets. Embedding the inverse wavelet filters allows for projecting the kernel into multiwavelet polynomial bases. Repeated multiwavelet transform aids in learning complex dependencies and resolution-independent solutions. The purpose of this architecture is to achieve compact representations of the operator and to exploit orthogonality properties & vanishing moments to capture complex information about physical systems and data streams. The model maps the multiwavelet transform of the input to the output at a very fine scale and has two components - the decomposition network and the reconstruction network. The network computes multiwavelet coefficients of the output at a coarse level using four neural networks. The reconstruction network uses the output of the four networks to compute the multiwavelet coefficients of the output at a finer level. This has a recurrent structure, and it continues until the finest level is achieved. This model was empirically shown to perform well on the 2D-Navier Stokes equation, where the wavelet transform was applied to velocity. It has also shown promise in generating finer resolution outputs when trained on low-resolution data. However, it cannot generalize to high frequency signals from low frequency ones.

8.6. Spectral neural operator

Spectral neural operator (SNO) is a recently proposed neural architecture that has been designed to address opaque outputs and aliasing errors which can be potentially observed in vanilla FNOs due to parameterization of the output function Fanaskov & Oseledets (2022). This model is designed to not suffer from aliasing errors and perform lossless operations on functions.

Typically, when FNOs are applied on inputs with coarser grids, then, the activation functions mitigate these aliasing errors. However, when the grid is refined, these errors disappear but FNOs would try to mitigate them further, causing outputs to deviate from the ground truth, resulting in errors in the final solution.

To elaborate further, any function when represented using Fourier Series with k $< |N|$ terms, can be structured on a uniform grid of 2N + 1 points. However, when activation function is applied, the output will have higher frequency points and this means, the grid will need to have more than 2N + 1 points. However, while training, the FNO will learn to fix these errors in representing higher harmonics. In situations where the higher harmonics also fit the grid with 2N + 1 points, the bias of the FNO induced during the training will cause it to try to fit the non-existent high harmonics.

To fix this issue, Spectral neural operator is defined with a fixed number of harmonics. It also decouples the interpolation process from the function mapping process. The mapping is proposed in equation 17.

$$\sum_i g_i(x) d_i = \sum_i g_i(x) b_i \tag{17}$$

In the above equation, $g_i(x)$ are either complex exponential or Chebyshev polynomial. d_i and b_i can be stacked finite-dimensional vectors. Spectral neural operator performs the above mapping and, importantly, preserves the structure of the series. To perform this mapping on finer grids, Chebyshev or Trigonometric interpolation should be used.

SNO has been shown to outperform vanilla FNO on integration, differentiation, parametric ODE, elliptic equation, KdV equation and non-linear Schrodinger equation. However, it does not perform as well on Burger's equation with low viscosity. The limitations of this version of SNO are that it is subject to Gibb's phenomenon and can only work on smooth input and output data. Basis functions used for SNO are non-adaptive.

9. Physics Informed neural operator

In this section, we will discuss a hybrid neural operator architecture called physics informed neural operator (PINO), which learns the mapping between function spaces through data-driven training while being subjected to physics constraints. This architecture is not to be confused with physics informed neural network (PINN), which learns the solution function as opposed to the solution operator.

PINO was designed to address the following key limitations of existing approaches Li et al. (2021b)

- Data-driven methods perform poorly if the training data is noisy or insufficient
- Physics-based approaches require a lot of computation power and may not optimize when the constraints are more complex.

Due to the fact that PINO learns the solution operator as opposed to just the solution to a particular instance of the PDE, this architecture is resolution invariant. This means that even when the operator is trained on low-resolution or coarse data, it accurately manages to predict the solution to high-resolution test instances. Furthermore, low-resolution data can be combined with high-resolution PDE constraints without any degradation in accuracy. It has been empirically shown Li et al. (2021b) that PINO has better generalization capabilities than regular FNO and requires far less training data.

PINO architecture comprises of a sequence of linear integral operators followed by non-linearity, which allows the operator to learn non-linear continuous operators. Operator learning, however, is done with two loss functions - a data-based loss function, similar to the ones used by other neural operators, and a physics-based loss function, used by PINN-based architectures.

The data-based loss function is given by equation 18.

$$\mathcal{L}_{data} = ||u - G_\theta(a)||^2 = \int_D (u(x) - G_\theta(a)(x))^2 dx \quad (18)$$

In the above equation, u denotes the actual solution, G_θ represents the operator and $G_\theta(a)$ represents the predicted solution.

The physics-based loss function is defined on both stationary and dynamic systems. A simple stationary system may be defined with bounded domain D as follows.

$$\begin{aligned} \mathcal{P}(u, a) &= 0 \qquad in \;\; D \subset \mathbf{R}^d \\ u &= g \qquad in \;\; \partial D \end{aligned} \quad (19)$$

The physics-based loss function for the stationary system presented in equation 19 is defined as follows.

$$\mathcal{L}_{pde} = ||\mathcal{P}(a, u_\theta)||^2_{L^2(D)} + \alpha||u_\theta|\partial D - g||^2_{L^2(D)} \quad (20)$$

In equation 20, $\mathcal{P}$ is a partial differential operator. a is the instance and u is the solution. u_θ is the neural network parameterized by θ.

Let a dynamic system be defined as follows

$$\begin{aligned} \frac{du}{dt} &= \mathcal{R}(u) \;\; in \;\; D \times (0, \infty) \\ u &= g \;\; in \;\; \partial D \times (0, \infty) \\ u &= a \;\; in \;\; \bar{D} \times 0 \end{aligned} \quad (21)$$

In equation 21, a = u(0) is the initial condition, $\mathcal{R}$ is the non-linear partial differential operator defined on Banach spaces. The Loss on this system is calculated in equation 22.

$$\begin{aligned} \mathcal{L}_{pde}(a, u_\theta) = ||\frac{du_\theta}{dt} - \mathcal{R}(u_\theta)||^2_{L^2(T;D)} + \\ \alpha||u_\theta|\partial D - g||^2_{L^2(T;\partial D)} + \beta||u_\theta|_{t=0} - a||^2_{L^2(D)} \end{aligned} \quad (22)$$

The above losses allow PINO to be constrained by physics while also being trained with data. Furthermore, instance-wise fine-tuning is done after training by using an operator loss. To compute the exact derivatives for the loss functions, point-wise differentiation approaches are used.
PINO architecture, when fine-tuned, has shown promise in learning specific equations such as Long Temporal Transient Flow, Chaotic Kolmogorov Flow, Transfer Reynolds Numbers and Lid Cavity Flows. However, it has yet to be tested on different kinds of geometry and it remains to be seen whether it can function well on irregular geometry.

10. Applications in physics problems

In this section, we discuss various applications of neural operators in material physics problems. Table 4 summarizes the advantages, limitations and applications of various neural operator architectures. Figure 11 highlights the key applications where neural operators have been successful in outperforming the conventional methods.

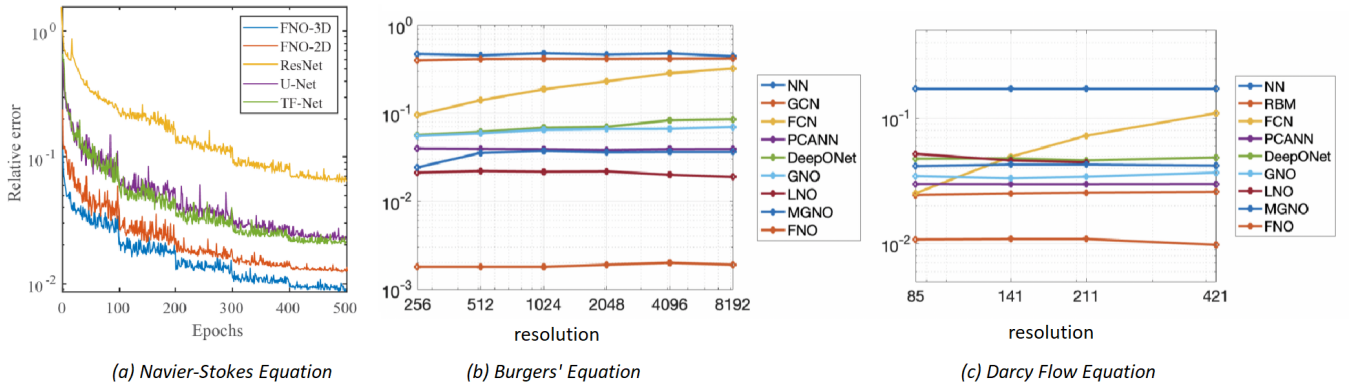
10.1. Solving partial differential equations

We shall first discuss the key PDEs for which neural operator architectures have been designed. A large part of this subsection is devoted to exploring the Navier-Stokes family of equations due to their broad range of applications, which will be described in the following sections. Figure 5 depicts the relative errors of various machine learning models in learning these PDEs.

The Navier-Stokes family of equations were developed to model the behavior of viscous fluids. They represent the conservation of mass and momentum of Newtonian Fluids. The 2D Navier Stokes equation is described in equation 23.

$$\begin{aligned} \frac{\partial w(x,t)}{\partial t} + u(x,t).\nabla w(x,t) - \nu\Delta w(x,t) &= f(x) \\ \nabla.u(x,t) &= 0, \quad x \in (0,1)^2, t \in [0,T] \\ w_0(x) &= w(x, t=0), \quad x \in (0,1)^2 \end{aligned} \quad (23)$$

In the above equations, w represents the vorticity. The neural operator learns to map the vorticity up to time T to vorticity at a time t>T. These equations are derived from Newton's second law to fluid mechanics, under the assumption that stress is dependent on viscosity and pressure. Navier Stokes Equations have various applications beyond modeling the flow of a liquid through



(a) Navier-Stokes Equation (b) Burgers' Equation (c) Darcy Flow Equation

Figure 5: The figure highlights the performance of neural operator-based architecture against neural network-based architectures. (a) represents how the relative error decreases with the number of Epochs on the Navier-Stokes Equation. It can be seen that FNO outperforms other architectures. (b) represents how the relative error varies with resolution on Burgers' Equation. FNO has a significantly lower error than Multipole Graph neural operator (MGNO), Graph neural operator (GNO), Low-Rank neural operator (LNO), etc. (c) shows that FNO outperforms all other models on Darcy Flow Equation. Image taken from Kovachki et al. (2021)

a pipe. They are also useful in aerodynamics by modeling the flow of air around a wing, and they can even be used to model weather Pathak et al. (2022). One of its derivatives, the Burgers equation, is particularly useful for modeling the one-dimensional flow of fluids as well as wave propagation and even vehicular traffic movement Musha & Higuchi (1978); Taigbenu (1999).

Another derivative of Navier Stokes, known as Darcy Flow, is useful for modeling the flow of fluids through a porous medium. Because of the wide range of physical phenomena that can be modeled by these three equations, methods of solving them are critical for several academic and commercial applications. However, these equations, which take the form of non-linear PDEs, are expensive to solve numerically.

In the following subsections, we will discuss the empirical results of neural operators on these families of PDEs. The following two sections describe the Poisson equation and Darcy flow, which belong to the class of equations described in (**??**) and the FNO approximates the maps $G^+ : f \to a$ and $G^+ : u \to a$ respectively. The remaining sections deal with evolution equations where $a = u_0 = u(., 0)$ are the initial conditions and $u(., t)$, the evolved states at a fixed time t, are the solution function. The operator approximates a map $G^+ : a = u(., 0) \to u = u(., t)$ for some t. The datasets, ie. $\{f_i, u_i\}_{i=1}^n$ or $\{a_i, u_i\}_{i=1}^n$ as discussed earlier are generated using appropriate numerical solvers.

10.1.1. Burgers equation

The Burgers equation is a non-linear PDE, which is represented in equation 24.

$$\partial_t u(x,t) + \partial_x(u^2(x,t)/2) = \nu \partial_x^2 u(x,t) \; x \in (0,1), t \in (0,1] \tag{24}$$
$$u(x,0) = u_0(x) \; x \in (0,1)$$

with periodic boundary conditions where the initial condition $a = u_0 \in L^2_{per}((0,1);\mathbf{R})$ is an element in the class of Bochner measurable functions whose norms $\|f\|_{(0,1);\mathbf{R}}$ lie in the standard L^2 space and $\nu \in \mathbf{R}_+$ is the viscosity coefficient. We aim to learn the operator mapping the initial condition to the solution at time $t = 1$, $G^+ : L^2_{per}((0,1);\mathbf{R}) \times \mathbf{R}_+ \to H^r_{per}((0,1);\mathbf{R})$ defined by $u_0 \to u(., 1)$ for any $r > 0$.

Li et al. (2020a) showed that FNO architectures outperformed other architectures (Traditional CNN based architectures, autoencoder, Graph neural networks) and provided the following loss measures against the Burgers Equation, as shown in table 1.

Model	s=256	s=512	s=1024	s=2048
FNO	0.0149	0.0158	0.0160	0.0146
MGNO	0.0243	0.0355	0.0374	0.0360
GNO	0.0555	0.0594	0.0651	0.0663
DeepONet	0.0569	0.0617	0.0685	0.0702

Table 1
Performance of different neural operators against Burgers Equation. The data has been taken from Kovachki et al. (2021)

10.1.2. Darcy Flow equation

Li et al. (2020a) trained a Fourier neural operator to learn the solution operator to the steady state 2D Darcy Flow

Equation, shown in equation 25, with a Dirichlet boundary $a \in L^{\infty}((0,1)^2; \mathcal{R}_+)$, where a is the diffusion coefficient and $f \in L^2((0,1)^2; \mathcal{R})$ is the forcing function.

$$-\nabla.(a(x)\nabla u(x)) = f(x) \tag{25}$$

Table 2 highlights the performance of various neural operator architectures against the Darcy Flow equation.

Model	s=85	s=128	s=141	s=211	s=256	s=421
FNO	0.0108	0.0111	0.0109	0.0109	0.0107	0.0098
MGNO	0.0416	0.0547	0.0428	0.0428	0.0542	0.0420
GNO	0.0346	-	0.0332	0.0342	-	0.0369
DeepONet	0.0476	-	0.0479	0.0462	-	0.0487
UNO	0.0075	-	0.0072	0.0070	-	0.0068
MWT	-	0.0074	-	-	0.0072	-

Table 2
Performance of different neural operator-based architectures against Darcy Flow Equation. It can be seen that UNO outperforms other models, with Multiwavelet models (MWT) having similar performance values as UNO. The values represent the relative L2 error. The data has been taken from Kovachki et al. (2021), Gupta et al. (2021) and Rahman et al. (2022c)

10.1.3. Navier Stokes equation

The Navier-Stokes equation is defined as follows.

$$\partial_t w(x,t) + u(x,t).\nabla w(x,t) = \nu \Delta w(x,t) + f(x) \tag{26}$$

In equation 26, $u \in \mathrm{C}([0,\ \mathrm{T}\ ];\ H^r_{per}((0,\ 1)^2;\ \mathbf{R}^2))$ for any $r > 0$ is the velocity field, $w = \nabla \times u$ is the vorticity, $w \in L_{per}((0,1)^2;\ \mathbf{R})$ is the initial vorticity, $\nu \in \mathbf{R}_+$ is the viscosity coefficient, and $f \in L^2_{per}((0,1)^2; \mathbf{R})$ is the forcing function.

Li et al. (2020a) aims to map the vorticity at a given time T to vorticity at a later time in the future. They argue that vorticity is more challenging to model than velocity. Table 3 contrasts the performance of FNO and UNO on the 2D and 3D variants of the Navier Stokes equation.

Model	ν=1e-3; T=50	ν=1e-4; T=30	ν=1e-5; T=20
FNO-3d	0.0086	0.1918	0.1893
FNO-2d	0.0128	0.1559	0.1556
UNO-3d	0.0080	0.0830	0.1120
UNO-2d	0.0050	0.0590	0.0700

Table 3
Performance of FNO and UNO architectures against Navier Stokes Equation at fixed N = 1000. It can be observed that UNO outperforms FNO. The values represent the relative L2 error. The data has been taken from Rahman et al. (2022c) Kovachki et al. (2021)

10.1.4. Flow equations

In their work Wen et al. (2022) explore the possibility of applying FNOs to solve multiphase flow equation, specifically with CO_2 and Water, given by the following equations 27 and 28.

$$\frac{\partial(\varphi \Sigma_p S_p \rho_p X_p^{CO_2})}{\partial t} = -\nabla.[\Phi^{CO_2}|_{adv} + \Phi^{CO_2}|_{dif}] + q^{CO_2} \tag{27}$$

$$\frac{\partial(\varphi \Sigma_p S_p \rho_p X_p^{water})}{\partial t} = -\nabla.[F^{water}|_{adv} + F^{water}|_{dif}] + q^{water} \tag{28}$$

φ is the porosity, S_p is the saturation of phase p, and X_p^{η} is the mass fraction of component η (water or CO_2)in phase p. For both components, the advective mass flux $F^{\eta}|_{adv}$ is obtained by summing over phases p. To solve this type of PDE, two types of variables are sampled - scalars and fields. Field variables are horizontal & vertical permeability, porosity, and injection perforation. The domain of this problem is defined to be a bounded and open set and a modified version of FNO called U-FNO is proposed to solve this problem.

The primary difference between a regular Fourier layer and a U-Fourier Layer is that the U-Fourier layer also includes a U-Net embedded within it to enhance the representation power of the layer through local convolutions. However, since it is a CNN architecture, it is less flexible than an Operator.

10.1.5. Photoacoustic equation

2D Photoacoustic equation is another PDE that can be solved using neural operators, as shown by Guan et al. (2021). A slight modification is made to the way the input is fed to the neural network. The training data is not normalized beforehand (a term that refers to centering the data w.r.t its moments) with Gaussian normalization prior to training. They show that both FNO-2D and FNO-3D can be used to solve this equation. The former performs 2d convolutions in space for a fixed interval of time, which is then recurrently propagated in time to solve the next interval. In the case of 3D FNO, the network performs 3D convolutions in space-time. They argue that while both can be applied, the FNO-3D performs better than its 2D counterpart.

The photoacoustic pressure wave that Guan et al. (2021) used as the basis for their experiments was defined in equation 29.

$$(\partial_{tt} - c_0^2 \Delta) p(r,t) = 0, p(r, t=0) = x, \partial_t p(r, t=0) = 0 \tag{29}$$

Here $p(r,t)$ is the photoacoustic pressure wave at position r and time t. c is the speed of sound.

10.2. Weather modelling

Pathak et al. (2022) discuss an FNO-based architecture - FourcastNet, which aims to model weather data at a res-

olution of 0.25°, which is approximately equal to the area of 900Km^2 near the equator. At such high resolutions, the predictions of their model can be compared with those of the Integrated Forecasting System (IFS). The key advantage of FourCastNet is that it is 45,000 times faster than NWP models on a node-hour basis, making it efficient for the generation of large ensemble forecasts. The model can be used to generate forecasts of massive weather phenomena such as Tornadoes, hurricanes, and extreme precipitations. Once trained, it consumes less power than IFS to generate forecasts. Some of the other advantages of the FourCastNet, as mentioned by Pathak et al. (2022), are 1. It has eight times the resolution of typical DL-based weather modeling systems. 2. It predicts lead times of up to a week with exceptional levels of accuracy. 3. It is rapid and inexpensive once trained.

10.3. High resolution simulation of SARS-COV-2 replication transcription complex dynamics

The replication of the SARS-COV-2 virus is done primarily by a multi-domain protein. Conventional tools such as all-atom molecular dynamics (AAMD) have shown to be limited in representing the molecular dynamics in high resolution and timescale. Trifan et al. (2021) propose a graph neural operator based model to simulate the molecular dynamics from the AAMD data. The model was used to learn time-dependent conformational changes within the molecules. The GNO model was able to predict protein backbone conformation up to 5ps.

10.4. Thermochemical curing of composites

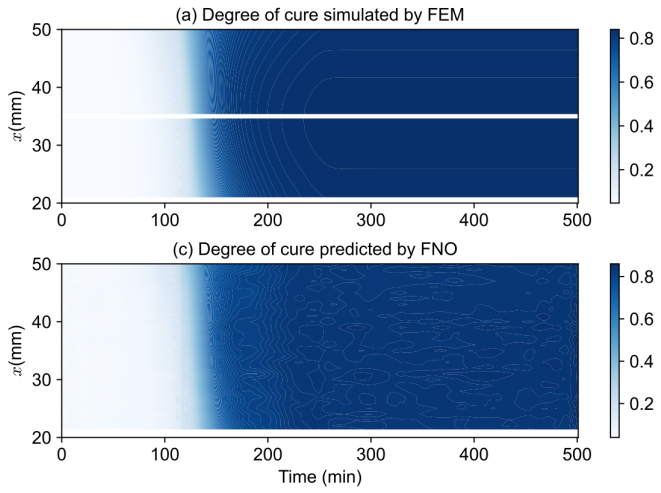


Figure 6: The figure represents the degree of cure predicted by the FEM method vs. the same done by FNO. The figure has been taken from Chen et al. (2021)

The composite curing process is strongly dependent on the temperature gradient, which influences the strength and mechanical properties of the resulting material. Chen et al. (2021) propose a residual FNO architecture that learns the mapping between the curing cycle and the temperature history. The model leverages the time resolution invariance property of neural operators and learns with smaller training datasets. The general heat transfer equation that the model learns is as follows

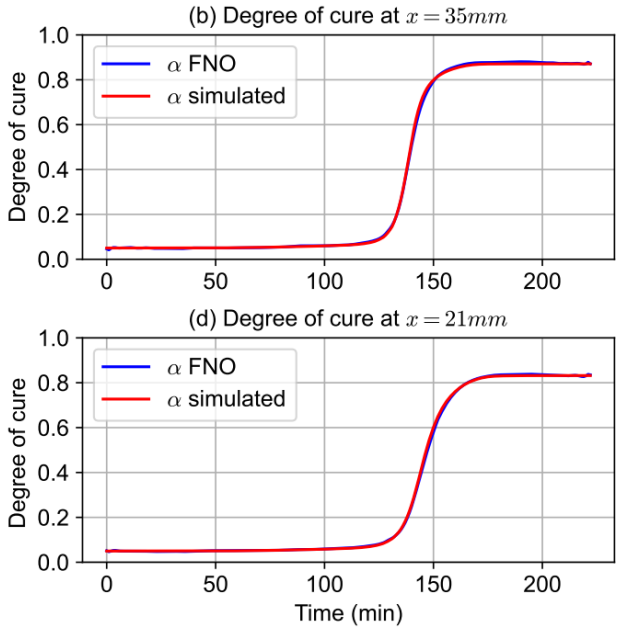


Figure 7: These graphs represent the degree of cure predicted by FNO and FEM methods at x = 35mm and 21mm. It can be observed that the two curves almost entirely overlap for the whole duration. The maximum prediction errors were within 0.02. This figure has been taken from Chen et al. (2021)

$$\rho C \frac{\partial \mathbf{T}}{\partial t} = \frac{\partial}{\partial x}(\lambda_x \frac{\partial \mathbf{T}}{\partial x}) + \frac{\partial}{\partial y}(\lambda_y \frac{\partial \mathbf{T}}{\partial y}) + \frac{\partial}{\partial z}(\lambda_z \frac{\partial \mathbf{T}}{\partial z}) + \mathcal{Q} \quad (30)$$

In equation 30, $\mathbf{T}$ is the temperature, ρ and C are density and specific heat capacity, respectively, λ is the directional thermal conductivity, and $\mathcal{Q}$ is the internal heat source. The boundary conditions used for this problem were Dirichlet boundaries, Neumann boundaries, and Robin boundaries. The architecture comprises K Fourier Layers, similar to a ResNet neural network. The cure period is discretized into finite intervals and each cure cycle is sampled as a vector from the space of n discrete cycles. A key advantage of residual FNO is that it leverages the domain knowledge to train the model using less training data. Rather than learning the mapping over the entire hypothesis space, it learns the mapping with respect to the specific input. For instance, if T_a is the cure cycle and $\mathcal{G}_\theta(T_a)$ is the learned temperature history, the network learns the difference $\mathcal{G}_\theta(T_a) - T_a$, restricting the size of the hypothesis space.

The authors empirically showed that this is an efficient way to model the curing process due to the limited number of Fourier modes. Using domain knowledge, the

model was able to generate more accurate results. Figure 6 visually represents the differences between the degree of cure predicted by the finite element method and by the neural operator method. Figure 7 is the plot of the trends in predicting the degree of cure with time by the two methods.

10.5. Coastal flood Modelling

Jiang et al. (2021) developed a digital twin of the earth's coastlines using FNO to predict sea surface levels on coastlines. They extended the basic FNO to learn multivariate dynamics. They built surrogate models of NEMO (Nucleus for European Modelling of the Ocean) and used these models to train the FNO to predict sea surface height. Their empirical evaluations showed that FNO-based models are approximately 45 times faster than systems such as NEMO and are better suited for real-time predictions of coastal flooding.

10.6. Continuous spatio-temporal dynamics

Stochastic PDEs are used to model various physical systems under the influence of randomness. Finite difference and spectral Galerkin methods require high computation power and high-resolution meshes. SPDEs can be defined by the following equation

$$du_t = (\mathbf{L}u_t + F(u_t))dt + G(u_t)d\mathcal{W}_t \tag{31}$$

In equation 31, $\mathcal{W}_t$ represents a Weiner process, F and G are continuous operators and $\mathbf{L}$ is the linear differential operator. From the operator perspective, W is the continuous embedding of the spatio-temporal data stream. To solve the system, u(t), with the initial condition, is projected into the latent space and the ODE solver solves it in the latent space, and the solution is mapped back into the original space. Salvi et al. (2021) have proposed a neural operator to solve the following equations

- Stochastic Ginzburg-Landau equation

$$\begin{aligned} \partial_t u - \Delta u = 3u - u^3 + \xi \qquad (32)\\ u(t,0) = u(t,1)\\ u(0,x) = u_0(x), (t,x) \in [0,T] \times [0,1] \end{aligned}$$

  Where the operator learns the solution along the noise path ξ.

- Stochastic Korteweg-De Vries equation

$$\begin{aligned} \partial_t u + 0.1\partial_x^3 u = 6u\partial_x u + \xi \qquad (33)\\ u(t,0) = u(t,1)\\ u(0,x) = u_0(x), (t,x) \in [0,T] \times [0,1] \end{aligned}$$

  This equation describes the propagation of nonlinear waves on the surface of fluids under random perturbations. The equation becomes stochastic when the noise is defined to be the partial sum approximation of a Q-Weiner Process.

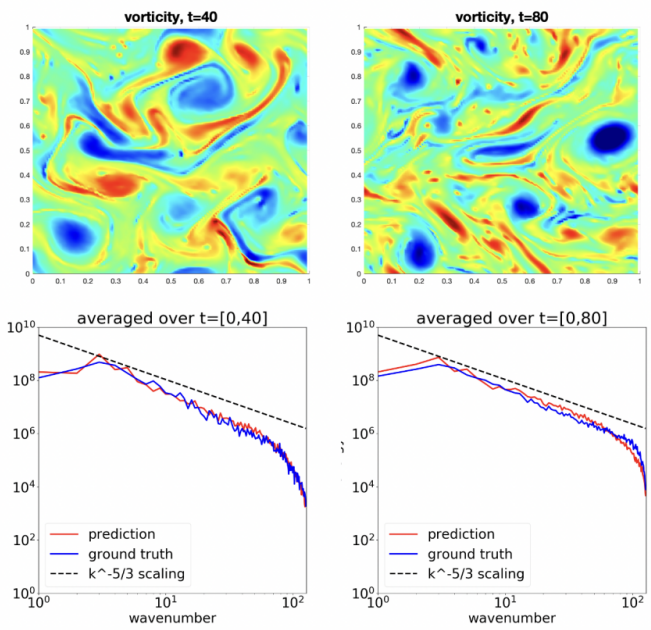


Figure 8: This figure shows the simulation of Kolmogorov flow by the Markov neural operator (MNO) with an initial condition generated from a random Gaussian field. The model captures the energy spectrum that converges to the cascade rate of $k^{-5/3}$. Image taken courtesy of Li et al. (2022)

- Stochastic Navier Stokes equation in 2D

$$\partial_t u - \nu\Delta w = -u.\nabla w + f + 0.05\xi \tag{34}$$

  This equation describes the incompressible flow of fluids under force. w refers to the vorticity and ν is the viscosity coefficient. The noise is set to a Q-Weiner process.

10.7. Disspative dynamics in chaotic systems

Chaotic systems tend to be unpredictable because they are susceptible to minor perturbations Li et al. (2021a). However, their long-term trajectories depend on an invariant measure called the global attractor. This problem has been previously approached using recurrent neural networks but has only worked for very short trajectories. The dissipativity and Markovian properties of these systems can be modeled using neural operator because they exhibit invariance. Dissipativity can be described as a compact set into which all other bounded sets evolve over time. The solution operator maps these initial conditions into the solution set.

To train the neural operator, dissipativity is imposed on the mesh by augmenting data on the outer shell. The authors consider finite-dimensional Lorenz-63 system, infinite-dimensional Kuramoto-Sivashinsky equations, and Kolmogorov flows. For their work, the authors limit themselves to ergodic systems, and the model does not make predictions outside the trained distribution. The operator learns the mapping in single time steps so that

Model	*Advantages*	*Limitations*	*Applications*
Fourier neural operator	Faster Resolution invariant discretization invariant Data driven Doesn't need to know the underlying PDE Zero shot Super resolution Li et al. (2020a)	Parameterization may lead to opaque outputs and aliasing errors Fanaskov & Oseledets (2022) Only works on rectangular domains with uniform meshes Li et al. (2022) Overfits with deeper networks Susceptible to Vanishing Gradient Rahman et al. (2022a) Constrained by availability of training data. Li et al. (2021b)	Burger's Equation Darcy Flow Equation Navier Stokes Equation Li et al. (2020a) Coastal Flood modelling Jiang et al. (2021) Photoacoustic Equation Guan et al. (2021) Chaotic systems Li et al. (2021a) Seismic wave progressions Yang et al. (2021)
GANO/UNO	Memory efficient implementations of deeper networks Optimized for Polish and Banach spaces Works well for bounded norms in infinite spaces Good at learning probability measures Don't suffer from modal collapse Rahman et al. (2022a)	Only works on rectangular domains with uniform meshes Li et al. (2022) Parameterization may lead to opaque outputs and aliasing errors. Fanaskov & Oseledets (2022)	Volcanic deformations Rahman et al. (2022a) Video Interpolation Viswanath et al. (2022) Dyadic Human motion prediction Rahman et al. (2022b)
DeepONet	Accurately approximate mappings between infinite dimensional banach spaces Learns oscillatory continuous functions Can learn the mapping from high frequency functions to low frequency functions Lu et al. (2019)	Requires high amount of training data May fail to learn underlying physical principles	Material Physics Goswami et al. (2022) Electrodynamics Cai et al. (2021a) Aerothermodynamics Sharma Priyadarshini et al. (2021) Medical imaging DeSanctis et al. (1987) Effects of seismic waves on buildings Liu & Cai (2021)
Graph neural operator	Learns long range dependencies in graph like data Linear time complexity Discretization invariant Can learn Mesh invariant solutions Li et al. (2020c)	Outperformed by Fourier neural operator on all PDEs with regular meshes. Li et al. (2020a)	Burgers Equation Darcy Flow Equation Li et al. (2020c) Protein dynamics in SARS-COV-2 virus Trifan et al. (2021)
PINO	Doesn't suffer from generalization errors that other operators suffer from Overcomes the limitations of purely physics based and purely data driven approaches. Incorporate constraints at different resolutions - combine coarse resolution data and high resolution data. Li et al. (2021b)	Has not been tested rigorously on High dimensional PDEs Li et al. (2021b)	Long Temporal Transient Flow Kolmogorov Flows Wave Equation Non-Linear Shallow Water Equation Li et al. (2021b)
Adaptive FNO	Powerful generative model Pathak et al. (2022) Efficient Token Mixer Adaptive Weight sharing among tokens Quasi-Linear Time Complexity highly parallelized Outperforms self-attention mechanisms Guibas et al. (2021)	Can be modified through wavelet transforms to better capture locality Guibas et al. (2021)	Climate Modelling Weather forecast Hurricane prediction Pathak et al. (2022) Generative imaging Guibas et al. (2021)
geo-FNO	geometry Aware Input can be irregular meshes, point clouds As fast as FNO but more efficient and accurate Li et al. (2022)	Only been tested on regular homeomorphic topologies Can be potentially expanded into PINOs but hasn't been empirically verified. Li et al. (2022)	Structural and Fluid Mechanics Problems Euler's equation for Airfoil flow Li et al. (2022)
Implicit FNO	Doesn't suffer from Vanishing Gradient Less prone to overfitting Hidden Layer parameters are independent Has the ability to learn material responses directly from DIC displacement tracking measurements. You et al. (2022)	Has long training times despite having fewer parameters due to the iterative algorithm used for learning. You et al. (2022)	Model Heterogenity and Material Defects in anisotropic and hyperelastic setting Porous Medium Flow Fracture mechanisms You et al. (2022)
Multiwavelet FNO	Compact Representation of data Resolution-independent solutions Learn complex dependencies Gupta et al. (2021)	Performance degrades if the Kernel used for data generation is changed. Cannot generalize to high frequency signals from low frequency ones Gupta et al. (2021)	Burgers Equation (1D) Navier-Stokes Equation (2D) Darcy Flow Equation (2D) Korteweg-de Vries Equation (1D) Gupta et al. (2021)
Spectral FNO	Doesn't suffer from aliasing errors Lossless operations on Functions Preserves the structure of the functions Fanaskov & Oseledets (2022)	Doesn't perform well on Burger's Equations Suffers from Gibbs Phenomenon Only works on smooth input/output Basis functions used are non-adaptive Fanaskov & Oseledets (2022)	Basic Integration, Differentiation Parametric ODEs Elliptic Equations KdV Equation Non-Linear Schrodinger Equation Fanaskov & Oseledets (2022)

Table 4
The table highlights the key advantages and limitations of the most recent operator-based neural architectures for solving PDE and other physics problems

the input would be the system at time step t, and the operator maps it to the system at time step t+1. Long-term predictions are made with repeated composition. Figure 8 highlights the simulation of Kolmogorov flow in a 40 second span from an initial Gaussian Random Field.

10.8. Seismic Wave propagation

Earthquakes and seismic activities are described by various forms of wave equations. However, solving these equations can be computationally expensive. Yang et al. (2021) discuss neural operators as a way to overcome the computation constraints by training the networks on an ensemble of wave equation data represented as low-resolution matrices. The neural operator is trained to learn the 2D acoustic wave equation, defined as the mapping from velocity, defined on an irregular mesh, to the wavefield solution. The network is also used to perform waveform inversions through reverse mode automatic differentiation. Figure 9 illustrates the difference between the inverted waveforms obtained through SEM and neural operator based methods.

10.9. Inelastic impact problems

The training data for the operator is generated as random fields with the von Karman covariance function. The solution is obtained using the Spectral-Element method by applying the Ricker Wavelet source. An added advantage of a neural operator is that its ability to perform automatic differentiation, which is equivalent to the adjoint state method, allows for full waveform inversion without having to compute the adjoint wavefield.

In their work, Liu et al. (2022) design neural operators to learn the inelastic deformation of polycrystalline solids, where polycrystalline refers to materials composed of disjoint grains, where each grain is made of the same material but exhibits different orientations based on the reference viewpoint. Plastic deformations occur through the mechanisms of slip and twinning. The problem is defined by the kinetic relation between crystallographic orientation, inelastic deformation gradient, slip activity, and twin volume fractions. The study is motivated by the fact that fine-scale behavior influences the behavior on a coarser scale. The authors propose two models to learn this re-

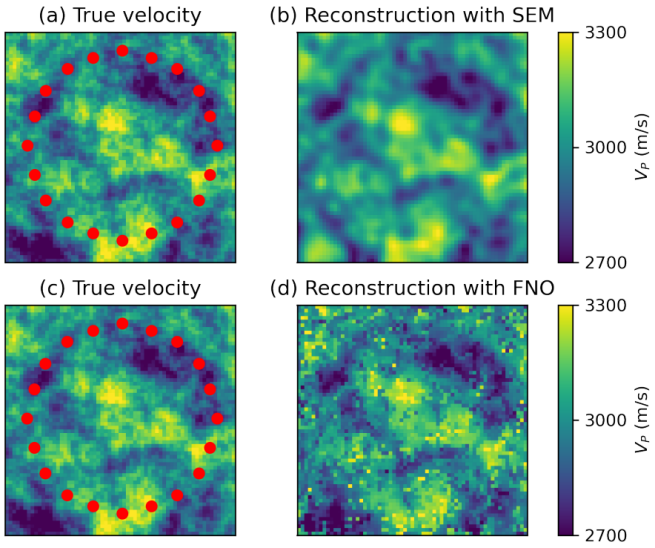


Figure 9: The figure compares the waveform inversion obtained through FNO vs the same obtained through SEM and adjoint tomography. The waveforms on the left (a) and (c) represent the true velocity and the source locations are denoted by the red circles. The relative misfit between true and inverted velocity in case of SEM is 0.028 while for FNO it is 0.0319. Image taken from Yang et al. (2021)

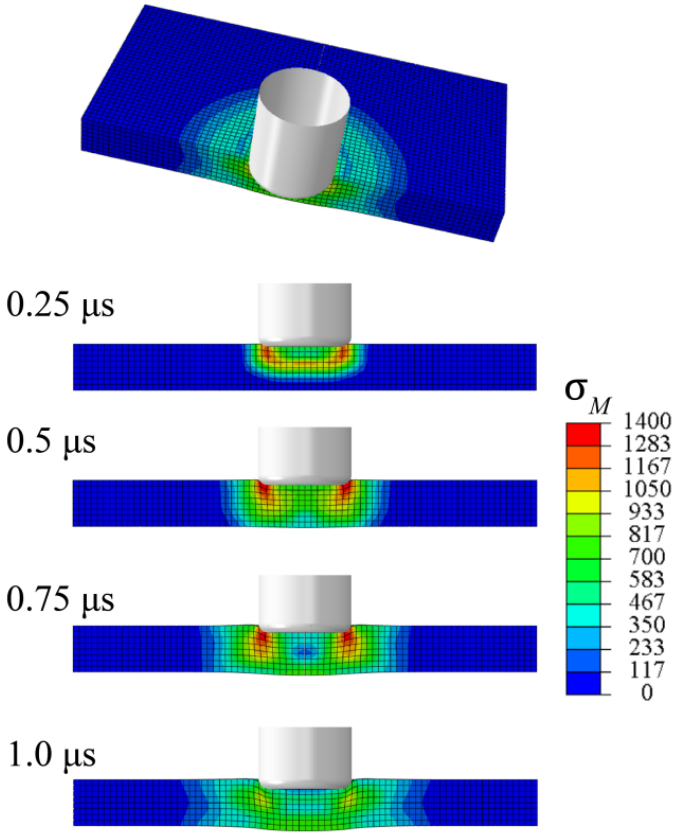


Figure 10: This Figure illustrates the 3D deformation experienced by a plate due to projectile impact. Each figure denotes the deformation in a 25 micro-second interval. Each subfigue represents the axial cross-section with Von Mises stress measure. Figure taken courtesey of Liu et al. (2022)

lation.

- *2DFFT* This model represents the problem in 2 dimensions with two slip systems and no twinning. The problem is solved using Fast Fourier Transform. The input to the network is the deformation history. The operator uses this information to approximate Cauchy stress.
- *3D Taylor* This is the 3-dimensional setting to study the behavior of materials such as magnesium. To solve this system, Taylor averaging assumption is used to assume that the deformation gradient is uniform on the unit cell.

Figure 10 portrays how the neural operator simulates projectile impact on a plate. It shows the axial cross section of the plate subjected to impact.

10.10. Forecasting subcritical cylinder Wakes

In their work, Renn et al. (2023) propose using Fourier neural operators to temporally forecast velocity fields to study the Karman vortex street phenomenon. They use this approach to determine how physical fluid flows evolve with time. The neural operator is trained on flow data obtained through particle image velocimetry and predicts ten-time steps. They show that the model predicts the velocity fields over the entire range of Reynolds numbers. While the model achieves low errors in predicting, it fails to learn unsteady turbulent dynamics. This could be because of the insufficient spatial resolution of the model. The authors speculate that training the model to learn smaller secondary flows could help predict the flows at larger lengths.

10.11. Video interpolation

More recently, neural operator-based models have been used for computer vision problems. In their work, Viswanath et al. (2022) model the problem of video frame interpolation as implicitly learning a PDE, where the complex motions exhibited by the moving objects within a video follow a PDE. A UNO was used to learn these complex trajectories to predict intermediate frames in a video.

10.12. Human motion analysis in dyadic setting

Prediction of reactions to human actions in dyadic or two-person settings is an interesting problem that has applications in gaming and 3D simulations. In their work, Rahman et al. (2022b) explore using Conditional GANO-based architectures to predict the reactions of humans conditioned on the movements of their dyadic partners. The model accurately learns from long sequences of motions at any arbitrary temporal resolution.

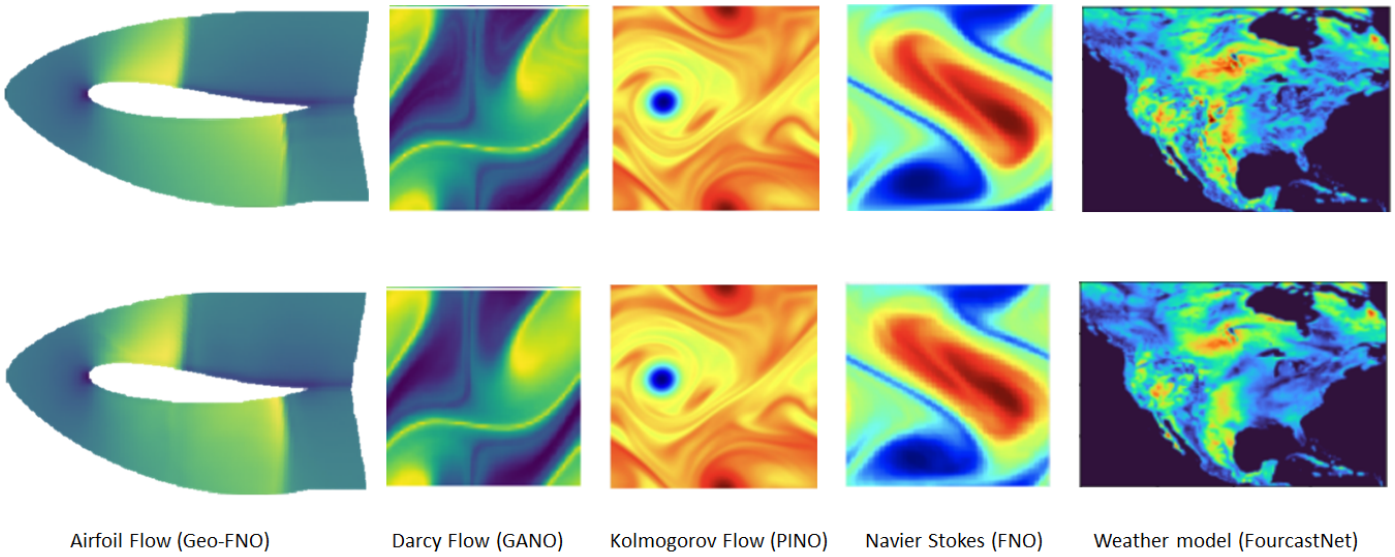


Figure 11: The above figure highlights the performance of various neural operator based architectures in solving different types of problems. It provides a visual comparison between generated solution an ground truth. The images on the top are the ground truth values while the ones on the bottom row are generated by the operator models. The first image on the left represents the airfoil Flow simulation generated by the geo-FNO architecture Liu et al. (2022). The second one is the Darcy Flow simulation generated by the GANO architecture Rahman et al. (2022a). The third one represents the Kolmogorov Flow generated by the PINO architecture Li et al. (2021b), the fourth pair represents the Navier-Stokes equations simulated by the vanilla FNO Li et al. (2020a) and the last one is the weather forecast model generated by the FourcastNet Pathak et al. (2022).

11. Scalability

Finite element approaches are generally slow and imprecise as they suffer from a trade-off between mesh resolution and computation time. The higher the resolution, the higher the computation cost per evaluation. Moreover, the results of a computation are only valid for a single instance of a PDE. Changing initial or boundary conditions or any other parameter requires the expensive computation to be re-run.

Neural network based methods are capable of approximating a PDE with decent accuracy. Although training them is computationally expensive, each computation of the forward evaluation is much faster than the conventional approaches. Some of these methods, such as the ConvPDE-UQ framework, are even able to parameterize the initial and boundary conditions as part of the input to the model in order to enable computation of any system within a given family of PDEs on any domain without having to retrain the model (Winovich et al. (2019)). However, neural network approaches are still mesh-dependent. The resolution that can be achieved from the model is determined by the resolution of the training data, which is computed via conventional methods. Thus the computational cost for training is still sensitive to resolution. This is where a neural operator is advantageous as it attempts to learn the function and hence, the mapping between infinite dimensional spaces. The result is that they are resolution invariant in that they can be trained with data that has a relatively low resolution and will still be able to evaluate at a higher resolution with the same error rate as in low resolution. Li et al. (2021b) demonstrated that their neural operator variant, the Fourier neural operator, is capable of accurately learning PDEs with zero-shot super-resolution. They further proved that the Fourier neural operator achieves superior accuracy compared to neural network-based solvers while still enjoying the same benefits in terms of fast computation of the forward evaluation and generalization to any instance within a PDE family.

All of these features of FNOs- low evaluation cost, zero-shot super resolution, and generalization- have significant implications in terms of computational efficiency. Li et al. (2021b) highlight this by presenting the Bayesian Inversion Problem, where they use a function space Markov chain Monte Carlo (MCMC) method (Cotter et al. (2013)) to draw samples from the posterior distribution of the initial vorticity in Navier-Stokes. MCMC are a set of algorithms for sampling data from distributions. The approach involves constructing Markov chains for the desired distribution, and a sample of this distribution would be a set of states of the Markov chain. Metropolis-Hastings is a well-known MCMC algorithm. In this experiment, they compare FNOs and conventional solvers. While both are able to achieve similar results, they vary substantially in terms of computation time. The MCMC using the traditional solver took 2.2 seconds per computation, whereas the FNO took only 0.005s per computation. The FNO, however, requires a one-time

training, which in this scenario took approximately 12 hours. The traditional solver, on the other hand, requires no such training. Hence, for a small number of data points, the conventional solver is more efficient. The benefits of FNOs are instead realized at a larger scale because the model only has to be trained once and can be used to quickly evaluate any instance of the PDE. To illustrate this point, let T represent the total computation time in seconds to make n evaluations. For the traditional solver, the computation time has the form of a fixed rate ($T = 2.2n$ in this particular setting), whereas for FNO, computation time would behave as a smaller rate plus a one-time cost ($T = 43200 + 0.005n$ in their setup). The authors generated 30,000 data points using each method. Using the FNO, this took 12 hours for training plus an additional 2.5 minutes for the 30,000 forward evaluations, whereas the traditional solver took a total of 18 hours for the same number of evaluations (Li et al. (2021b)).

These advantages in computation time extend to cost savings, especially at large scale. To illustrate the difference in cost at a large scale, consider the cost of running both models with the same experimental setup as above on a p3.2xlarge EC2 instance on AWS, which has 8 vCPU and 61 GiB (or approx. 65 GB) of memory. The P3 instances feature the NVIDIA V100 GPU making it the most similar of the AWS compute options to the NVIDIA V100 GPU with 16 GB memory used by Li et al. (2021b). For the purposes of this demonstration, assume these instances have the same hardware performance as the hardware used by Li et al. The on-demand rate for this instance, which is the cheapest of the P3s, is $3.06 per hour. Thus, training the FNO would cost approximately $36.72. Once trained, the FNO can theoretically perform up to 720000 computations per hour, while the conventional solver would take 440 hours to do the same number of evaluations. For 100,000 evaluations, the FNO would cost under $40, including training time, whereas the conventional solver would cost approximately $187. Beyond the cost savings, this computational efficiency has important implications for environmental impact, as cloud computing is known to consume significant energy resulting in a large carbon footprint.

In designing mechanical systems involving aerodynamics or fluid dynamics, it is common for engineers to have to compute the forward operation of Navier Stokes several thousands of times varying the coefficients with each evaluation in order to recover certain properties of the system. Fourestey & Moubachir (2005), for example, investigated inverse problems of the Navier Stokes equations in the context of designing bridge decks under wind loads. Solving the inverse problem of a given PDE is also critical to tuning predictive modelling systems such as those for weather and climate modelling (Kashinath et al. (2021)). Using conventional numerical methods to solve inverse problems of a given PDE can be prohibitively slow and expensive. The scalability of FNOs can make this problem not only feasible but cost-effective, which has significant implications for mechanical design.

12. Future work - Adaptability

In this paper, we have illustrated the advantages that FNOs have over traditional numerical solvers in terms of computational performance, as well as their superior accuracy when compared to neural network-based solvers. The remaining question to be answered is how these benefits can be realized in academic research and commercial applications. The development of numerical methods such as FDM and FEM began as early as the 1940s Hrennikoff (1941). However, these methods did not see wide-spread use until the 1960s when open-source programs for FEM began to be developed, such as Nastran developed by NASA Butler & Michel (1971) and SAP IV developed at UC Berkeley Gran & Yang (1978). Since then, many tools for the numerical modeling of physical systems have been built and made widely available to researchers and engineers, including MATLAB, COMSOL, and Autodesk Simulation. The question now becomes- *how can the same be done with FNO-based solvers?*

As FNOs are still relatively new computational methods, there is more work that is needed to develop them further and more variants of them are likely to be developed over the next few years. Nevertheless, existing variants already have the potential to greatly accelerate physics-based modelling. However, as with many newly developed neural networks, they are unlikely to see practical use or widespread adoption by physics researchers until they are integrated into a software package that can reliably be used without a firm background in machine learning. Developing this software and enabling it to be used for practical applications would further motivate future research and development of these approaches for PDE approximation. An important factor in achieving this is open-source. Li et al. have already made their code open-source for their work in Li et al. (2020a). Open-source accelerates development and enables standardization in software packages. This standardization is central to the repeatability of results and is hence, critical for research.

It is important to note that numerical solvers are still required to produce the training data for FNOs. As with all machine learning approaches, the quality of this data, in terms of the size and spread of training data, as well as accuracy, impacts the performance of the FNO. In this respect, machine learning paradigms such as that of active learning Settles (2012), which uses learning theoretic arguments to come up with strategies to select training data, can be used to improve the sample complexity (a term used in machine learning for the number of samples required to achieve a particular

accuracy) of learning based techniques. In particular, the learning algorithm would decide on the pairs of $\{u_i\}_{i=1}^N$ for which the solver should generate the corresponding $\{f_i\}$ or $\{a_i\}$ with the aim of selecting the right pairs such that the network is able to approximate the solver function with the lowest number of pairs N. Meta-learning is another paradigm that can be used to improve the sample complexity of learning algorithms. This method aims to learn some underlying connection to different machine-learning tasks. While there are several notions of meta-learning, a popular algorithm is MAML Finn et al. (2017), which aims to learn an initialization for gradient-based learning approaches from different tasks sharing a common structure. This learned initialization, when used for the gradient-based learning approaches, allows for better sample complexity Saunshi et al. (2020).

Due to the dependency of FNOs on the numerical solvers, one potential future for FNOs would be to integrate them into FEM software packages. This would make them more accessible to physics researchers allowing them to leverage the computational benefits of FNOs to accelerate their work. The software could train an FNO for a user-inputted PDE with data it produces using FEM and a sampling method based on active learning or meta-learning. The trained FNO would then be used in computation of the forward evaluation with a mesh and other parameters defined by the user via the same interfaces they use today as the software would handle the parameterization of those inputs into the structure expected by the FNO. Of course there are many variants of FNOs each created for different types of systems involving PDEs where they achieve superior performance over the vanilla version. Users of FEM software would not be aware of the distinctions between them. Hence, this software would have to be able to provide the user with some recommendation of which variant, if any, to use in which scenarios. A deeper study of the performance of different variants across different use-cases would be required in order to build this knowledge base. This paper is a starting point for that.
While the accuracy of FNO-based solvers is invariant to resolution, it is still true that a given FNO-based solver could have higher accuracy on certain systems over others due to the non-deterministic nature of machine-learning. This would be an additional consideration for researchers using this software. One approach to contend with this would be to have the software also produce test data along with the training data and provide a report of accuracy on this test set so that the researcher can weigh the accuracy-efficiency trade off between FNO and conventional methods in order to decide which approach to use. If they opt for the FNO approach, they could also include this report in their findings.

13. Conclusion

The goal of this paper was to characterize the numerous methods that have been developed for approximating PDEs in order to highlight the potential opportunities that exist for further development of these technologies. We have provided a dense overview of these methods from conventional solvers to more novel approaches developed in recent years that leverage neural networks to approximate the mapping between finite spaces as well as those capable of approximating the mapping between function spaces by leveraging the neural operator. We have provided details about several variants of each type of PDE solver and have compared the performance of these variants across several applications particularly those in the field of physics. We have also articulated the advantages of FNO-based solvers as compared to conventional solvers and the implications this could have for the future of physics modelling. Furthermore, we proposed a high-level vision for how this technology could eventually be used to accelerate computation-based physics research.

References

Acharya, S., Baliga, B., Karki, K., Murthy, J., Prakash, C., & Vanka, S. P. (2007). Pressure-based finite-volume methods in computational fluid dynamics, .

Aghdam, H. H., & Heravi, E. J. (2017). Guide to convolutional neural networks. *New York, NY: Springer*, *10*, 51.

Aochi, H., Ulrich, T., Ducellier, A., Dupros, F., & Michea, D. (2013). Finite difference simulations of seismic wave propagation for understanding earthquake physics and predicting ground motions: Advances and challenges. In *Journal of Physics: Conference Series* (p. 012010). IOP Publishing volume 454.

Bebis, G., & Georgiopoulos, M. (1994). Feed-forward neural networks. *Ieee Potentials*, *13*, 27–31.

Browne, M., & Ghidary, S. S. (2003). Convolutional neural networks for image processing: an application in robot vision. In *Australasian Joint Conference on Artificial Intelligence* (pp. 641–652). Springer.

Butler, T. G., & Michel, D. (1971). *NASTRAN: A summary of the functions and capabilities of the NASA structural analysis computer system* volume 260. Scientific and Technical Information Office, National Aeronautics and Space ....

Cai, S., Mao, Z., Wang, Z., Yin, M., & Karniadakis, G. E. (2022). Physics-informed neural networks (pinns) for fluid mechanics: A review. *Acta Mechanica Sinica*, (pp. 1–12).

Cai, S., Wang, Z., Lu, L., Zaki, T. A., & Karniadakis, G. E. (2021a). Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, *436*, 110296.

Cai, S., Wang, Z., Wang, S., Perdikaris, P., & Karniadakis, G. E. (2021b). Physics-informed neural networks for heat transfer problems. *Journal of Heat Transfer*, *143*.

Chen, G., Li, Y., Meng, Q., Zhou, J., Hao, X. et al. (2021). Residual fourier neural operator for thermochemical curing of composites. *arXiv preprint arXiv:2111.10262*, .

Chen, T., & Chen, H. (1995). Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, *6*, 911–917.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Twenty-second international joint conference on artificial intelligence*.

Cotter, Roberts, Stuart, & White (2013). Mcmc methods for functions: Modifying old algorithms to make them faster. *Statist. Sci.*, *28*, 424 – 446.

Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., & Piccialli, F. (2022). Scientific machine learning through physics-informed neural networks: Where we are and what's next. *arXiv preprint arXiv:2201.05624*, .

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*, 303–314.

Demirdžić, I., & Muzaferija, S. (1994). Finite volume method for stress analysis in complex domains. *International journal for numerical methods in engineering*, *37*, 3751–3766.

DeSanctis, R. W., Doroghazi, R. M., Austen, W. G., & Buckley, M. J. (1987). Aortic dissection. *New England Journal of Medicine*, *317*, 1060–1067.

Donea, J., & Huerta, A. (2003). *Finite element methods for flow problems*. John Wiley & Sons.

Eymard, R., Gallouët, T., & Herbin, R. (2000). Finite volume methods. *Handbook of numerical analysis*, *7*, 713–1018.

Fadlun, E., Verzicco, R., Orlandi, P., & Mohd-Yusof, J. (2000). Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations. *Journal of computational physics*, *161*, 35–60.

Fanaskov, V., & Oseledets, I. (2022). Spectral neural operators. *arXiv preprint arXiv:2205.10573*, .

Fang, Z. (2021). A high-efficient hybrid physics-informed neural networks based on convolutional neural network. *IEEE Transactions on Neural Networks and Learning Systems*, .

Finn, C., Abbeel, P., & Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning* (pp. 1126–1135). PMLR.

Fourestey, G., & Moubachir, M. (2005). Mcmc methods for functions: Modifying old algorithms to make them faster. *Computer Methods in Applied Mechanics and Engineering*, *194*, 877–906.

Gao, H., Sun, L., & Wang, J.-X. (2021). Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, *428*, 110079.

Godunov, S., & Bohachevsky, I. (1959). Finite difference method for numerical computation of discontinuous solutions of the equations of fluid dynamics. *Matematičeskij sbornik*, *47*, 271–306.

Goswami, S., Yin, M., Yu, Y., & Karniadakis, G. E. (2022). A physics-informed variational deeponet for predicting crack path in quasi-brittle materials. *Computer Methods in Applied Mechanics and Engineering*, *391*, 114587.

Gran, C. S., & Yang, T. (1978). Nastran and sap iv applications on the seismic response of column-supported cooling towers. *Computers & Structures*, *8*, 761–768.

Guan, S., Hsu, K.-T., & Chitnis, P. V. (2021). Fourier neural operator networks: A fast and general solver for the photoacoustic wave equation. *arXiv preprint arXiv:2108.09374*, .

Guibas, J., Mardani, M., Li, Z., Tao, A., Anandkumar, A., & Catanzaro, B. (2021). Adaptive fourier neural operators: Efficient token mixers for transformers. *arXiv preprint arXiv:2111.13587*, .

Gupta, G., Xiao, X., & Bogdan, P. (2021). Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, *34*, 24048–24062.

Guss, W. H., & Salakhutdinov, R. (2019). On universal approximation by neural networks with uniform guarantees on approximation of infinite dimensional maps. *arXiv preprint arXiv:1910.01545*, .

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*, 359–366.

Hrennikoff, A. (1941). Solution of problems of elasticity by the framework method, .

Hughes, T. J. (2012). *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation.

Jiang, P., Meinert, N., Jordão, H., Weisser, C., Holgate, S., Lavin, A., Lütjens, B., Newman, D., Wainwright, H., Walker, C. et al. (2021). Digital twin earth–coasts: Developing a fast and physics-informed surrogate model for coastal floods via neural operators. *arXiv preprint arXiv:2110.07100*, .

Jordan, C., & Jordán, K. (1965). *Calculus of finite differences* volume 33. American Mathematical Soc.

Kashinath, K., Mustafa, M., Albert, A., Wu, J., Jiang, C., Esmaeilzadeh, S., Azizzadenesheli, K., Wang, R., Chattopadhyay, A., Singh, A. et al. (2021). Physics-informed machine learning: case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A*, *379*, 20200093.

Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2021). Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, .

Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, *9*, 987–1000.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*, 436–444.

Li, S., Cui, X., & Li, G. (2017). Multi-physics analysis of electromagnetic forming process using an edge-based smoothed finite element method. *International Journal of Mechanical Sciences*, *134*, 244–252.

Li, Z., Huang, D. Z., Liu, B., & Anandkumar, A. (2022). Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020a). Fourier neural operator for parametric partial differential equations. *arXiv preprint*

arXiv:2010.08895, .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2020b). Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., & Anandkumar, A. (2021a). Markov neural operators for learning chaotic systems. *arXiv preprint arXiv:2106.06898*, .

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Stuart, A., Bhattacharya, K., & Anandkumar, A. (2020c). Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, *33*, 6755–6766.

Li, Z., Zheng, H., Kovachki, N., Jin, D., Chen, H., Liu, B., Azizzadenesheli, K., & Anandkumar, A. (2021b). Physics-informed neural operator for learning partial differential equations. *arXiv preprint arXiv:2111.03794*, .

Liu, B., Kovachki, N., Li, Z., Azizzadenesheli, K., Anandkumar, A., Stuart, A. M., & Bhattacharya, K. (2022). A learning-based multiscale method and its application to inelastic impact problems. *Journal of the Mechanics and Physics of Solids*, *158*, 104668.

Liu, L., & Cai, W. (2021). Multiscale deeponet for nonlinear operators in oscillatory function spaces for building seismic wave responses. *arXiv preprint arXiv:2111.04860*, .

Livingstone, D. J., Manallack, D. T., & Tetko, I. V. (1997). Data modelling with neural networks: advantages and limitations. *Journal of computer-aided molecular design*, *11*, 135–142.

Lu, L., Jin, P., & Karniadakis, G. E. (2019). Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, .

Mao, Z., Jagtap, A. D., & Karniadakis, G. E. (2020). Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, *360*, 112789.

Moczo, P., Kristek, J., & Halada, L. (2004). The finite-difference method for seismologists. *An Introduction*, *161*.

Musha, T., & Higuchi, H. (1978). Traffic current fluctuation and the burgers equation. *Japanese journal of applied physics*, *17*, 811.

Naranjo-Torres, J., Mora, M., Hernández-García, R., Barrientos, R. J., Fredes, C., & Valenzuela, A. (2020). A review of convolutional neural network applied to fruit image processing. *Applied Sciences*, *10*, 3443.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, .

Özişik, M. N., Orlande, H. R., Colaco, M. J., & Cotta, R. M. (2017). *Finite difference methods in heat transfer*. CRC press.

Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K. et al. (2022). Fourcastnet: A global data-driven high-resolution weather model using adaptive fourier neural operators. *arXiv preprint arXiv:2202.11214*, .

Rahman, M. A., Florez, M. A., Anandkumar, A., Ross, Z. E., & Azizzadenesheli, K. (2022a). Generative adversarial neural operators. *arXiv preprint arXiv:2205.03017*, .

Rahman, M. A., Ghosh, J., Viswanath, H., Azizzadenesheli, K., & Bera, A. (2022b). Pacmo: Partner dependent human motion generation in dyadic human activity using neural operators. *arXiv preprint arXiv:2211.16210*, .

Rahman, M. A., Ross, Z. E., & Azizzadenesheli, K. (2022c). U-no: U-shaped neural operators. *arXiv e-prints*, (pp. arXiv–2204).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, *378*, 686–707.

Reddy, J. N. (2019). *Introduction to the finite element method*. McGraw-Hill Education.

Renn, P. I., Wang, C., Lale, S., Li, Z., Anandkumar, A., & Gharib, M. (2023). Forecasting subcritical cylinder wakes with fourier neural operators. *arXiv preprint arXiv:2301.08290*, .

Rosofsky, S. G., & Huerta, E. A. (2022). Applications of physics informed neural operators. *arXiv preprint arXiv:2203.12634*, .

Roters, F., Eisenlohr, P., Bieler, T. R., & Raabe, D. (2011). *Crystal plasticity finite element methods: in materials science and engineering*. John Wiley & Sons.

Salvi, C., Lemercier, M., & Gerasimovics, A. (2021). Neural stochastic pdes: Resolution-invariant learning of continuous spatiotemporal dynamics. *arXiv preprint arXiv:2110.10249*, .

Saunshi, N., Zhang, Y., Khodak, M., & Arora, S. (2020). A sample complexity separation between non-convex and convex meta-learning. In *International Conference on Machine Learning* (pp. 8512–8521). PMLR.

Sazli, M. H. (2006). A brief review of feed-forward neural networks. *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, *50*.

Searles, D. J., & von Nagy-Felsobuki, E. I. (1988). Numerical experiments in quantum physics: Finite-element method. *American Journal of Physics*, *56*, 444–448.

Settles, B. (2012). Active learning. *Synthesis lectures on artificial intelligence and machine learning*, *6*, 1–114.

Sharma Priyadarshini, M., Venturi, S., & Panesi, M. (2021). Application of deeponet to model inelastic scattering probabilities in air mixtures. In *AIAA AVIATION 2021 FORUM* (p. 3144).

Sirignano, J., & Spiliopoulos, K. (2018). Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, *375*, 1339–1364.

Slone, A., Bailey, C., & Cross, M. (2003). Dynamic solid mechanics using finite volume methods. *Applied mathematical modelling*, *27*, 69–87.

Smith, G. D., Smith, G. D., & Smith, G. D. S. (1985). *Numerical solution of partial differential equations: finite difference methods*. Oxford university press.

Sod, G. A. (1978). A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws. *Journal of computational physics*, *27*, 1–31.

Tadmor, E. (2012). A review of numerical methods for nonlinear partial differential equations. *Bulletin of the American Mathematical Society*, *49*, 507–554.

Taigbenu, A. E. (1999). Burgers equation. In *The Green Element Method* (pp. 195–216). Boston, MA: Springer US. URL: https://doi.org/10.1007/978-1-4757-6738-4_7. doi:10.1007/978-1-4757-6738-4_7.

Thais, S., Calafiura, P., Chachamis, G., DeZoort, G., Duarte, J., Ganguly, S., Kagan, M., Murnane, D., Neubauer, M. S., & Terao, K. (2022). Graph neural networks in particle physics: Implementations, innovations, and challenges. *arXiv preprint arXiv:2203.12852*, .

Trifan, A., Gorgun, D., Li, Z., Brace, A., Zvyagin, M., Ma, H., Clyde, A., Clark, D., Salim, M., Hardy, D. J. et al. (2021). Intelligent resolution: Integrating cryo-em with ai-driven multi-resolution simulations to observe the sars-cov-2 replication-transcription machinery in action. *bioRxiv*, .

Viswanath, H., Rahman, M. A., Bhaskara, R., & Bera, A. (2022). Nio: Lightweight neural operator-based architecture for video frame interpolation. *arXiv preprint arXiv:2211.10791*, .

Wang, S., Wang, H., & Perdikaris, P. (2021). Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, *7*, eabi8605.

Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A., & Benson, S. M. (2022). U-fno—an enhanced fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, *163*, 104180.

Wilson, E., Bathe, K., & Peterson, F. (1974). Finite element analysis of linear and nonlinear heat transfer. *Nuclear engineering and design*, *29*, 110–124.

Winovich, N., Ramani, K., & Lin, G. (2019). Convpde-uq: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, *394*, 263–279.

Yang, Y., Gao, A. F., Castellanos, J. C., Ross, Z. E., Azizzadenesheli, K., & Clayton, R. W. (2021). Seismic wave propagation and inversion with neural operators. *The Seismic Record*, *1*, 126–134.

Yegnanarayana, B. (2009). *Artificial neural networks*. PHI Learning Pvt. Ltd.

Yin, M., Ban, E., Rego, B. V., Zhang, E., Cavinato, C., Humphrey, J. D., & Em Karniadakis, G. (2022). Simulating progressive intramural damage leading to aortic dissection using deeponet: an operator–regression neural network. *Journal of the Royal Society Interface*, *19*, 20210670.

You, H., Zhang, Q., Ross, C. J., Lee, C.-H., & Yu, Y. (2022). Learning deep implicit fourier neural operators (ifnos) with applications to heterogeneous material modeling. *arXiv preprint arXiv:2203.08205*, .

Yu, J., Lu, L., Meng, X., & Karniadakis, G. E. (2022). Gradient-enhanced physics-informed neural networks for forward and inverse pde problems. *Computer Methods in Applied Mechanics and Engineering*, *393*, 114823.

Zienkiewicz, O. C., Taylor, R. L., & Zhu, J. Z. (2005). *The finite element method: its basis and fundamentals*. Elsevier.

14. Appendix

Model	Link
FNO	https://github.com/neural-operator/fourier_neural_operator
FourCastNet	https://github.com/NVlabs/FourCastNet
GANO	https://github.com/kazizzad/GANO
geo-FNO	https://github.com/neural-operator/Geo-FNO
GNO	https://github.com/neural-operator/graph-pde
MWT-NO	https://github.com/gaurav71531/mwt-operator
PINO	https://github.com/neural-operator/PINO
SNO	https://github.com/vlsf/sno
UNO	https://github.com/ashiq24/UNO

Table 5
This table provides links to the source code for various neural operator architectures

Term	Meaning
A	Banach Function Space
a(x)	Input function
α	Hyperparameter for neural network
b	Bias
β	Hyperparameter for neural network
C	Specific heat capacity
$\mathcal{C}$	Cost Function
c	Speed of Sound
D	Domain
d_i, b_i	Finite dimensional vectors
e_{xy}	Edge from node x to node y in a graph neural network
$\mathcal{F}$	Fourier Transform
f(x)	Function
$\mathcal{F}^{-1}$	Inverse Fourier Transform
G	neural operator
G^+	Non-Linear map between Function Spaces
g_i	Complex Exponential/Chebyshev polynomial
h_i	Hidden layer Embedding
k	directional Thermal Conductivity
K, κ	Kernel
L	Banach Function Space
$\mathbf{L}$	Linear Differential Operator
$\mathcal{L}$	Loss Function
λ	Directional Thermal Conductivity
N	Natural Numbers
n	Arbitrary Natural number
P	Lifting Map
p(r,t)	photoacoustic pressure wave at position r, time t
$\mathcal{P}$	Partial Differential Operator
Φ	flux
φ	Porosity
$\mathcal{Q}$	internal heat source
Q	Projecting Map
R	Non-Linear Partial Differential Operator
$\mathcal{R}$	Real Numbers
ρ	density
ς	stress tensor
S_p	Saturation of phase p
σ	Non-Linear Activation
$\mathbf{T}$	temperature
t, T	time
U	Banach Function Space
u(x)	solution function
$\mathcal{U}$	Velocity
u_θ	neural network parameterized by θ
v_i, $v^{(i)}$	ith output of a neural network
$\mathbf{V}$	neural network Approximation of Solution
ν	Viscosity coefficient
W	Weight
$\mathcal{W}$	Weiner Process
w	Vorticity
X	mass fraction
ξ	noise
x_i	ith input element
∇	Gradient Operator
Δ	Laplacian Operator

Table 6
A summary of mathematical notations used in the article