

---

# DEEPOKAN: DEEP OPERATOR NETWORK BASED ON KOLMOGOROV ARNOLD NETWORKS FOR MECHANICS PROBLEMS

---

A PREPRINT

**Diab W. Abueidda\***

Civil and Urban Engineering Department  
New York University Abu Dhabi  
National Center for Supercomputing Applications  
University of Illinois at Urbana-Champaign

**Panos Pantidis**

Civil and Urban Engineering Department  
New York University Abu Dhabi

**Mostafa E. Mobasher<sup>†</sup>**

Civil and Urban Engineering Department  
New York University Abu Dhabi

May 30, 2024

## ABSTRACT

The modern digital engineering design often requires costly repeated simulations for different scenarios. The prediction capability of neural networks (NNs) makes them suitable surrogates for providing design insights. However, only a few NNs can efficiently handle complex engineering scenario predictions. We introduce a new version of the neural operators called DeepOKAN, which utilizes Kolmogorov Arnold networks (KANs) rather than the conventional neural network architectures. Our DeepOKAN uses Gaussian radial basis functions (RBFs) rather than the B-splines. The DeepOKAN is used to develop surrogates for different mechanics problems. This approach should pave the way for further improving the performance of neural operators. Based on the current investigations, we observe that DeepOKANs require a smaller number of learnable parameters than current MLP-based DeepONets to achieve comparable accuracy.

**Keywords** Computational solid mechanics · Deep operator networks · Gaussian radial basis functions · Neural networks · Orthotropic elasticity · Transient analysis

## 1 Introduction

Contemporary science and engineering increasingly depend on advanced physics-based computational models. Numerical simulations offer critical insights for understanding and predicting complex physical phenomena and engineering systems. Real-world problems often involve varying loads, boundary and initial conditions, material properties, and domain geometries. Due to their inherent complexity, multi-physics nature, dimensionality, time dependency, and fidelity requirements, finite element analysis (FEA) models can be computationally intensive, even with high-performance computing platforms [1, 2, 3]. Consequently, relying solely on traditional high-fidelity simulation models for tasks such as computer-aided design, material discovery, and digital twins is often impractical, especially when exploring numerous design scenarios or geometries [4]. In this context, surrogate neural network (NN) models emerge as a promising machine learning approach, capable of rapidly inferring solutions to physical problems without requiring expensive numerical simulations once they are trained [5, 6]. These models hold significant potential across various application domains, including real-time simulations for predictions and controls, design, topology and shape optimizations,

---

\*da3205@nyu.edu

<sup>†</sup>mostafa.mobasher@nyu.edu

sensitivity analysis, and uncertainty quantification, which require extensive forward evaluations with varying parameters [7, 8, 9]. For a comprehensive overview of the application of neural networks in computational mechanics, see [10].

However, most of these approaches necessitate retraining or transfer learning when input parameters such as loads, boundary conditions, material properties, or geometry are altered. This requirement is similarly observed in conventional numerical methods like finite elements (FE), where each new input parameter value mandates a distinct simulation. Recently, researchers devised NNs trained to approximate the underlying physics or mathematical operator for a class of problems, a process known as operator learning [11]. Researchers have proposed various architectures for operator learning, with two notable examples being the Fourier neural operator (FNO) and the deep operator network (DeepONet). The Fourier Neural Operator (FNO) was first introduced by Li et al. [12] to solve partial differential equations with parametric inputs. Drawing inspiration from the Fourier transform used in differential equation solutions, the input function is processed through multiple Fourier layers. The encoded information is then mapped onto the output function space. Each Fourier layer applies the fast Fourier transform (FFT) to its input and filters out high-frequency modes. FNO and its enhanced versions have been successfully used for Burger’s equation, Darcy flow, and the Navier-Stokes equation [12], as well as for elasticity and plasticity problems in solid mechanics [13]. Nonetheless, since the FNO employs FFT, it encounters challenges when dealing with complex geometries or intricate non-periodic boundary conditions. Additionally, Fourier-based operations are computationally expensive when addressing high-dimensional problems.

Lately, Lu et al. [14, 15] introduced the DeepONet, which emerged as another capable architecture for operator learning. DeepONet comprises two sub-networks: a branch network for encoding input functions and a trunk network for encoding input domain geometry. Initially, both networks were designed as multi-layer perceptron (MLP) networks. In the seminal study, DeepONet effectively mapped between unknown parametric functions and solution spaces for several linear and nonlinear partial differential equations (PDEs) while also learning explicit operators such as integrals. DeepONets have been increasingly used to tackle scientific and engineering challenges, such as the inverse design of nanoscale heat transfer systems [16], brittle fracture [17], digital twins [18], and uncertainty quantification [19]. Lu et al. [20] thoroughly compared FNO and DeepONet. Several researchers have proposed different architectures for the branch and/or trunk networks to enhance DeepONet’s versatility and capability to capture complex scenarios [21, 22, 23, 24, 25, 26].

The significance of MLPs is immense, as they are the standard models in machine learning for approximating nonlinear functions, owing to their expressive power as assured by the universal approximation theorem [27, 28]. Recently, Liu et al. [29] proposed a promising alternative to MLPs called Kolmogorov-Arnold networks (KANs). While MLPs are based on the universal approximation theorem [28], KANs are based on the Kolmogorov-Arnold representation theorem [30, 31, 32]. Similar to MLPs, KANs have fully connected architectures. Nevertheless, unlike MLPs, which use fixed activation functions on nodes (neurons), KANs employ activation functions with learnable weights on edges. This difference can make KANs more accurate and interpretable than MLPs for several modeling scenarios. The potential of using the Kolmogorov-Arnold representation theorem to construct neural networks has been explored in various studies [33, 34, 35, 36, 37] before the work of Liu et al. [29]. However, most research has adhered to the original depth-2, width- $(2n + 1)$  representation and has not utilized modern techniques such as backpropagation for training. Liu et al. [29] generalized the original Kolmogorov-Arnold representation to arbitrary widths and depths, thereby revitalizing and contextualizing it within the contemporary deep learning framework. They also conducted extensive empirical experiments to demonstrate its potential as a foundational artificial intelligence and science model, highlighting its accuracy and interoperability. Although the work of Liu et al. [29] was published recently, it inspired other researchers to investigate the topic further [38, 39].

This study introduces the DeepOKAN, where the branch and trunk of DeepONet are KANs rather than the traditional MLPs. While the original implementation utilized splines for function approximation, we have replaced them with radial basis functions (RBFs). This modification has resulted in a significant computational speedup, enhancing the model’s efficiency without compromising accuracy. Adopting RBFs leverages their scalability and flexibility, making the approach more suitable for high-dimensional applications. The DeepOKAN is used to solve several mechanics problems. The paper’s structure is outlined as follows: Section 2 provides an overview of KANs and Gaussian RBFs. It also scrutinizes the DeepOKAN. In Section 3, the DeepOKAN is developed as a neural operator for a few numerical examples. The current version of the paper has one example only. The other examples will be added soon. The paper concludes in Section 4 by summarizing the key results and indicating potential directions for future research.

## 2 Methods

### 2.1 Kolmogorov-Arnold representation theorem

The Kolmogorov-Arnold representation theorem, also known as the superposition theorem, is a fundamental result in approximation theory. It asserts that any continuous multivariate function on a bounded domain can be represented as a composition of a finite number of univariate functions and a set of linear operations (additions). Specifically, for any smooth function  $f : [0, 1]^{n_{\text{in}}} \rightarrow \mathbb{R}^{n_{\text{out}}}$ , the Kolmogorov-Arnold theorem asserts that there exist continuous univariate functions  $\psi_q$  and  $\phi_{p,q}$  such that:

$$f(\mathbf{x}) = \sum_{q=1}^{2n_{\text{in}}+1} \psi_q \left( \sum_{p=1}^{n_{\text{in}}} \phi_{p,q}(x_p) \right), \quad (1)$$

where  $\mathbf{x} = (x_1, \dots, x_{d_{\text{in}}})$ ,  $\phi_{q,p} : [0, 1] \rightarrow \mathbb{R}$ , and  $\psi_q : \mathbb{R} \rightarrow \mathbb{R}$ .

### 2.2 Kolmogorov-Arnold network

The Kolmogorov-Arnold representation theorem states that any continuous multivariable function on a bounded domain can be represented as a superposition of continuous functions of one variable and addition. While no strict restrictions exist on choosing these univariate functions beyond their continuity, practical considerations may influence their specific forms based on the problem context and desired properties. In the KAN implementation by Liu et al. [29], each KAN layer is composed of the sum of the spline function and the SiLU activation function, with learnable coefficients. Splines face a significant curse of dimensionality (COD) due to their inability to leverage compositional structures. On the other hand, MLPs are less affected by COD because of their feature learning capabilities. Still, they are less accurate than splines in low-dimensional settings due to their inefficiency in optimizing univariate functions. A model must capture the compositional structure (external degrees of freedom) and effectively approximate univariate functions (internal degrees of freedom) to learn a function accurately. A more detailed description of KANs can be found in the work of [29].

Here, we provide a brief description. A KAN comprises several KAN layers. Figure 1 shows a KAN with two KAN layers. A KAN layer with  $n_{\text{in}}$ -dimensional inputs and  $n_{\text{out}}$ -dimensional outputs can be represented as a matrix of 1D functions, denoted as:

$$\Psi = \{\phi_{q,p}\}, \quad p = 1, 2, \dots, n_{\text{in}}, \quad q = 1, 2, \dots, n_{\text{out}}, \quad (2)$$

where the functions  $\phi_{q,p}$  have learnable parameters.

The shape of KAN can be expressed as an integer array  $[n_0, n_1, \dots, n_L]$ , where  $n_l$  is the number of nodes (neurons) in the  $l^{\text{th}}$  layer (see Figure 1). The  $i^{\text{th}}$  neuron in the  $l^{\text{th}}$  layer is denoted by  $(l, i)$ , while the activation value of the  $(l, i)$ -neuron is represented by  $x_{l,i}$ . There are  $n_l n_{l+1}$  activation functions between the consecutive layers  $l$  and  $l+1$ . The activation function connecting the neurons  $(l, i)$  and  $(l+1, j)$  is denoted by:

$$\phi_{l,i,j}, \quad l = 0, \dots, L-1, \quad i = 1, \dots, n_l, \quad j = 1, \dots, n_{l+1}. \quad (3)$$

As shown in Figure 1, the activation functions appear on the edges rather than the nodes, unlike MLPs. Specifically, the pre-activation of  $\phi_{l,i,j}$  is  $x_{l,i}$ , while the post-activation of  $\phi_{l,i,j}$  is  $\tilde{x}_{l,i,j}$ . Then, the  $\tilde{x}_{l,i,j}$ 's are summed to determine  $x_{l+1,j}$ , which is the activation value at the  $(l+1, j)$ -neuron:

$$x_{l+1,j} = \sum_{i=1}^{n_l} \tilde{x}_{l,i,j} = \sum_{i=1}^{n_l} \phi_{l,i,j}(x_{l,i}). \quad (4)$$

This can be presented in a matrix form as follows:

$$\mathbf{x}_{l+1} = \underbrace{\begin{pmatrix} \phi_{l,1,1}(\cdot) & \phi_{l,1,2}(\cdot) & \cdots & \phi_{l,1,n_l}(\cdot) \\ \phi_{l,2,1}(\cdot) & \phi_{l,2,2}(\cdot) & \cdots & \phi_{l,2,n_l}(\cdot) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{l,n_{l+1},1}(\cdot) & \phi_{l,n_{l+1},2}(\cdot) & \cdots & \phi_{l,n_{l+1},n_l}(\cdot) \end{pmatrix}}_{\Psi_l} \mathbf{x}_l, \quad (5)$$

where  $\Psi_l$  is the function matrix corresponding to the  $l^{\text{th}}$  KAN layer. Then, a general KAN is expressed as a composition of  $L$  layers:

$$\text{KAN}(\mathbf{x}) = (\Psi_{L-1} \circ \Psi_{L-2} \circ \cdots \circ \Psi_1 \circ \Psi_0) \mathbf{x}. \quad (6)$$

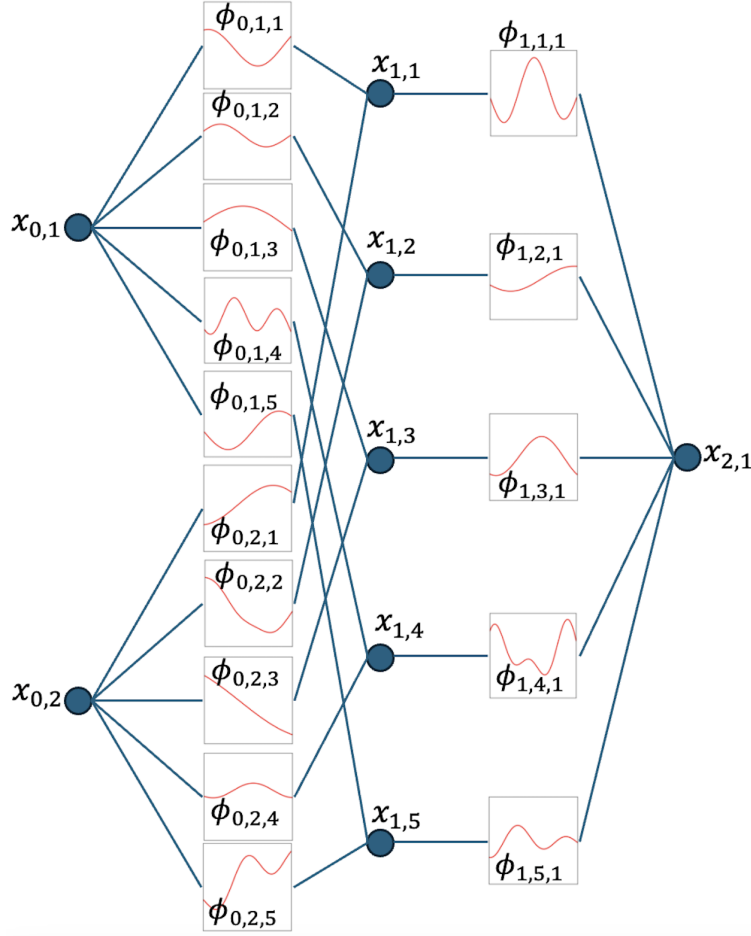


Figure 1: Illustration of the activation functions flowing through the network.

The original Kolmogorov–Arnold representation (see Equation 1) is deemed a special case of the KAB, with two layers and a shape  $[n_0, 2n_0 + 1, 1]$ . On the other hand, the well-known MLPs alternate between linear transformation  $\mathbf{W}$  and nonlinear activation functions  $\sigma$ :

$$\text{MLP}(\mathbf{x}) = (\mathbf{W}_{L-1} \circ \sigma \circ \mathbf{W}_{L-2} \circ \sigma \circ \dots \circ \mathbf{W}_1 \circ \sigma \circ \mathbf{W}_0)\mathbf{x}. \quad (7)$$

MLPs distinctly handle linear transformations  $\mathbf{W}$  and nonlinearities  $\sigma$ , whereas KANs combine both within  $\Psi$ .

### 2.3 RBF-KAN

As mentioned earlier, Liu et al. [29] proposed using B-splines as activation functions within the KAN framework. Here, we propose using RBFs for KANs. An RBF is a real-valued function whose value depends only on the distance from a central point. The Gaussian RBF is a specific type from the radial basis function family. The transformation of each input  $x_i$  using a Gaussian RBF  $R$  is given by:

$$R^l(x_i^l, g_j^l) = \exp\left(-\left(\frac{x_i^l - g_j^l}{\beta}\right)^2\right), \quad i = 1, 2, \dots, n, \quad j = 1, 2, \dots, m \quad (8)$$

where  $n$  is the input dimension (layer  $l$  size), and  $m$  represents the number of grid points (RBF centers). The superscript  $l$  denotes the layer number:  $l = 1, \dots, L$ , where  $L$  is the number of layers.  $g_j$  represents the  $j^{\text{th}}$  point in the RBF grid;  $g_j \in \mathbf{G}$ . The RBF centers  $\mathbf{G}$  can be learnable parameters or fixed points (non-learnable) defined during the initialization of the RBF transformation within the RBF-KAN framework.  $\beta$  is the scaling factor for the RBF, defined as:

$$\beta = \frac{g_{\max} - g_{\min}}{m - 1}. \quad (9)$$

$g_{\max}$  and  $g_{\min}$  denote the maximum and minimum values of the grid, respectively. After transforming the inputs, the transformed features are combined linearly using some learnable weight matrix  $\mathbf{W}$ . The output  $x^{l+1}$  is computed as:

$$x^{l+1} = \mathbf{W}^l \mathbf{R}^l(x^l, G^l) \quad (10)$$

where  $\mathbf{R}$  is a vector of size  $(n \times m)$ ,  $\mathbf{W}$  is a matrix with a size of  $(o, n \times m)$ , where  $o$  is the length of the vector  $x^{l+1}$ . While constructing an RBF-KAN, one can stack several RBF-KAN layers. During the training of an RBF-KAN,  $\mathbf{W}$  and  $\mathbf{G}$  are obtained by minimizing a loss function  $\mathcal{L}$ , which requires finding the gradients of  $\mathcal{L}$  with respect to  $\mathbf{W}$  and  $\mathbf{G}$  using backpropagation:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{W}^l} &= \frac{\partial \mathcal{L}}{\partial x^L} \cdot \mathbf{W}^{L-1} \cdot \frac{\partial \mathbf{R}^{L-1}}{\partial x^{L-1}} \cdot \mathbf{W}^{L-2} \dots \mathbf{W}^{l+1} \cdot \frac{\partial \mathbf{R}^{l+1}}{\partial x^{l+1}} \cdot \mathbf{R}^l \\ \frac{\partial \mathcal{L}}{\partial \mathbf{G}^l} &= \frac{\partial \mathcal{L}}{\partial x^L} \cdot \mathbf{W}^{L-1} \cdot \frac{\partial \mathbf{R}^{L-1}}{\partial x^{L-1}} \cdot \mathbf{W}^{L-2} \dots \mathbf{W}^{l+1} \cdot \frac{\partial \mathbf{R}^{l+1}}{\partial x^{l+1}} \cdot \mathbf{W}^l \cdot \frac{\partial \mathbf{R}^l}{\partial \mathbf{G}^l} \end{aligned} \quad (11)$$

where  $x^L$  is the final output of the RBF-KAN, and  $\frac{\partial \mathcal{L}}{\partial x^L}$  depends on the choice of the loss function.  $\frac{\partial \mathbf{R}^l}{\partial x^l}$  and  $\frac{\partial \mathbf{R}^l}{\partial \mathbf{G}^l}$  are found by differentiating Equation 8:

$$\begin{aligned} \frac{\partial R^l}{\partial x_i^l} &= -\frac{2(x_i^l - g_j^l)}{\beta^2} \exp\left(-\left(\frac{x_i^l - g_j^l}{\beta}\right)^2\right) \\ \frac{\partial R^l}{\partial g_j^l} &= \frac{2(x_i^l - g_j^l)}{\beta^2} \exp\left(-\left(\frac{x_i^l - g_j^l}{\beta}\right)^2\right). \end{aligned} \quad (12)$$

## 2.4 Neural operators

Researchers developed neural networks trained to approximate the underlying physics or mathematical operator for a class of problems, a process known as operator learning. Specifically, neural operators were proposed to learn nonlinear operators, mapping input functions into corresponding output functions. For neural operators, in the infinite functional space  $Q$ , the parameter  $q \in Q$  represents the input functions and  $s \in S$  refers to the unknown solutions of the PDE in the functional space  $S$ . It is assumed that for each  $q$  in  $Q$ , a unique solution exists  $s = s(q)$  in  $S$  corresponding to the governing PDE, which also satisfies the boundary conditions (BCs). Consequently, the mapping solution operator  $F : Q \rightarrow S$  can be defined as:

$$F(q) = s(q). \quad (13)$$

Specifically, for a collection of  $N$  points  $\mathbf{X}$  on a domain, each denoted by its coordinates  $(x_i, y_i)$ , the neural operator considers both the functions  $q_j$  in its branch and positions  $\mathbf{X}$  in its trunk and predicts the solution operator  $\hat{F}(q)(\mathbf{X})$  by fusing intermediate encoded outputs  $b_i$  (from branch) and  $t_i$  (from trunk) in a dot product enhanced by bias  $B$ , as shown in Figure 2. In a larger sense, it is possible to think of  $\hat{F}(q)(\mathbf{X})$  as a function of  $\mathbf{X}$  conditioning on input  $q$ .

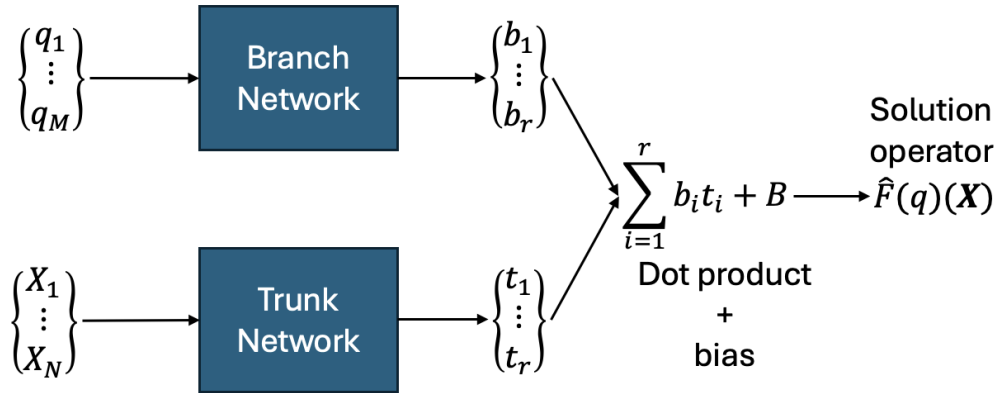


Figure 2: Schematic for neural operators.

The parameter  $q$  can encompass a variety of functional inputs that influence the underlying PDE solution. For instance,  $q$  may represent material properties such as thermal conductivity or elastic modulus. Additionally,  $q$  could denote load

constants or time-dependent input load function, discretized at  $T$  time steps to form an input load vector  $\mathbf{q}$ . This allows the modeling of transient processes where the applied loads vary over time, as shown in one of the examples later. This study leverages neural operators as regressors. Thus, one needs to devise an appropriate loss function. Typical loss functions for regression problems are mean square error and mean absolute error. This paper uses the root mean square deviation (RMSD) as the loss function to be minimized and determine the optimized network parameter. The RMSD is defined as:

$$\mathcal{L} = \text{RMSD} = \sqrt{\frac{1}{N} \sum_{i=1}^N (s_i - \hat{s}_i)^2} \quad (14)$$

where  $\hat{s}$  denotes the solution obtained from the neural operator. This paper considers two types of neural operators: DeepONet and DeepOKAN.

## 2.5 DeepOKAN vs. DeepONet

A DeepONet (see Figure 3) model, which uses MLP networks in both its branch and trunk, serves as a performance baseline. This paper introduces the DeepOKAN (see Figure 4), which replaces the MLP networks with KANs. Specifically, it employs Gaussian RBF KANs, though other KAN types, such as multi-quadratic RBF KANs and B-splines KANs, could also be explored in the context of neural operators. KAN models require significantly fewer learnable parameters than MLP models while achieving equivalent or better accuracy [29]. RBFs have been used before in the context of machine learning [40], but their success was limited. However, the recent development of KANs promises more effective integration. By leveraging the theoretical foundations laid by Kolmogorov and Arnold, KANs offer a compact and efficient representation of functions. This efficiency, combined with higher accuracies, positions KANs as a promising alternative to traditional MLPs across various applications.

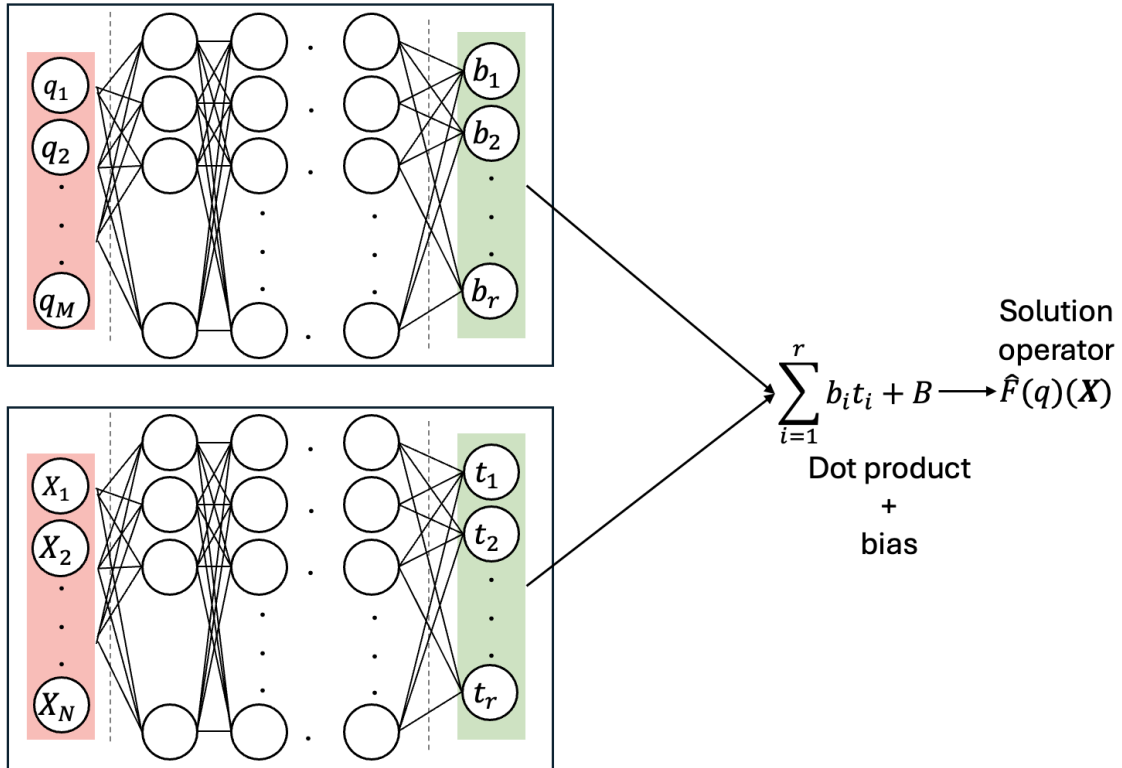


Figure 3: Illustration of the DeepONet.

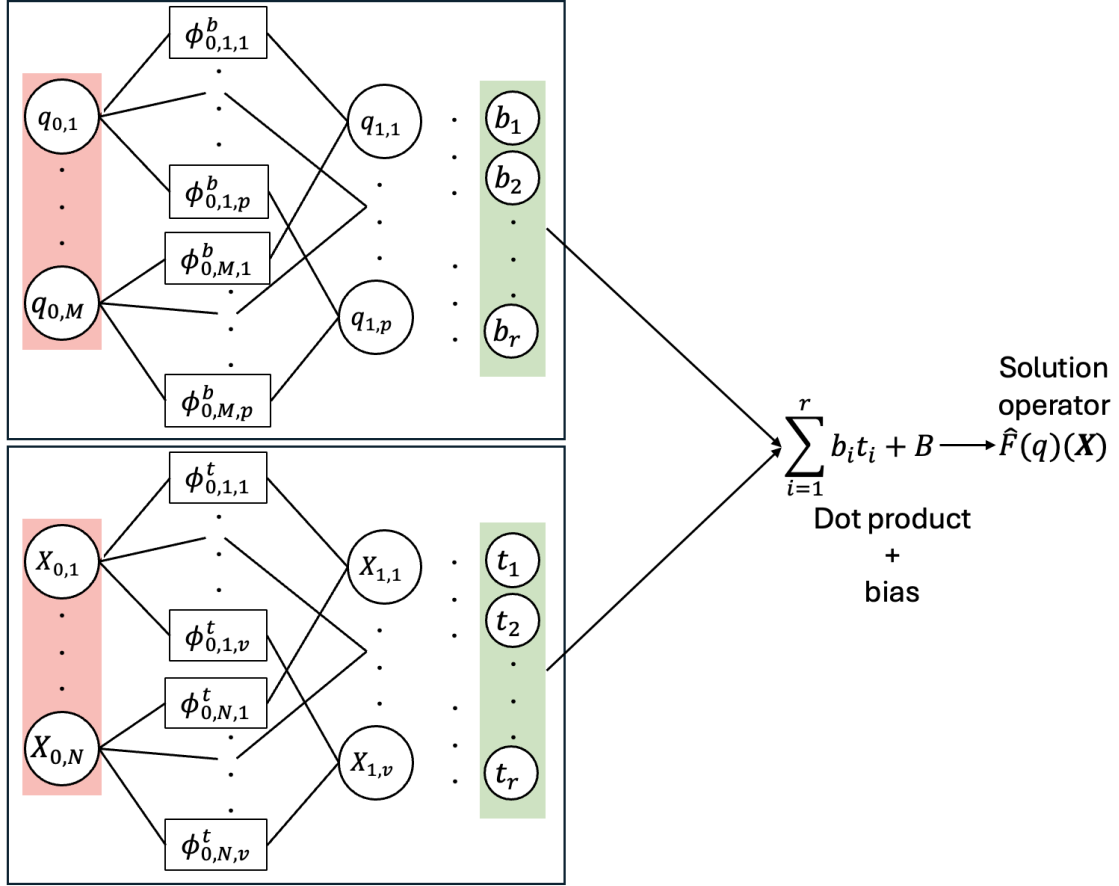


Figure 4: Illustration of the DeepOKAN.  $\phi^t$  and  $\phi^b$  represent the activation functions corresponding to the trunk and branch networks, respectively.  $p$  and  $v$  are hyperparameters defining the width of layer  $l = 1$  in the branch and trunk, respectively. Different layers can possess different widths.

### 3 Numerical examples

#### 3.1 2D orthotropic elasticity

##### 3.1.1 FEA and data generation

Consider a homogeneous, orthotropic, elastic body undergoing small deformations. In the absence of body forces and inertial forces, the equilibrium equation can be expressed as follows:

$$\begin{aligned} \nabla \cdot \boldsymbol{\sigma} &= \mathbf{0}, & \mathbf{x} &\in \Omega, \\ \hat{\mathbf{u}} &= \bar{\mathbf{u}}, & \mathbf{x} &\in \Gamma_u, \\ \boldsymbol{\sigma} \cdot \mathbf{n} &= \bar{\mathbf{t}}, & \mathbf{x} &\in \Gamma_t. \end{aligned} \quad (15)$$

Here,  $\boldsymbol{\sigma}$  represents the Cauchy stress tensor,  $\mathbf{n}$  denotes the normal unit vector,  $\nabla \cdot$  signifies the divergence operator, and  $\nabla$  indicates the gradient operator. Given the assumption of small deformations, the strain  $\boldsymbol{\varepsilon}$  can be described by:

$$\boldsymbol{\varepsilon} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T) \quad (16)$$

The relationship between the strain and stress is written as:

$$\boldsymbol{\sigma} = \mathbb{C} : \boldsymbol{\varepsilon}, \quad (17)$$

where  $\mathbb{C}$  is the fourth-order elastic tensor defining linear elastic behavior. In the case of plane-stress orthotropy, it is assumed that the two-dimensional material has two planes of symmetry aligned with the global  $x$  and  $y$  axes. For the

orthotropic plane-stress case, the compliance matrix  $\mathbf{S}$  is expressed as:

$$[\mathbf{S}] = \begin{bmatrix} \frac{1}{E_x} & -\frac{\nu_{xy}}{E_x} & 0 \\ -\frac{\nu_{yx}}{E_y} & \frac{1}{E_y} & 0 \\ 0 & 0 & \frac{1}{G_{xy}} \end{bmatrix}, \quad (18)$$

with  $E_x$  and  $E_y$  representing Young's moduli in the orthotropic directions,  $G_{xy}$  being the shear modulus, and  $\nu_{xy}$  being the in-plane Poisson's ratio. The constitutive relation symmetry is ensured by:

$$\nu_{xy} = \frac{E_x}{E_y} \nu_{yx}. \quad (19)$$

$\mathbf{S}$  has to be inverted to obtain the stiffness matrix.

In this example, we consider a two-dimensional (2D) domain subjected to boundary conditions and traction forces shown in Figure 5. The bottom and right edges of the domain are subjected to roller boundary conditions. Traction forces are applied at the top edge of the domain, represented by two components,  $t_x$  and  $t_y$ . These forces are sampled from uniform distributions within a specified range:  $t_x, t_y \in [-0.3, 0.3]$ . As demonstrated in Equations 18 and 19, for a 2D orthotropic material, four constants are required:  $E_x$ ,  $E_y$ ,  $\nu_{xy}$ , and  $G_{xy}$ . We sample  $E_x$ ,  $E_y$ , and  $\nu_{xy}$  from specified uniform distributions:  $E_x, E_y \in [5, 20]$ , and  $\nu_{xy} \in [0.15, 0.35]$ .  $G_{xy}$  is computed using the following relationship:

$$G_{xy} = \frac{1}{2} \left( \frac{E_x}{2(1 + \nu_{xy})} + \frac{E_y}{2(1 + \nu_{yx})} \right). \quad (20)$$

Equation 20 does not generally hold for orthotropic materials. However, we compute  $G_{xy}$  using this equation to ensure that it remains within a physically plausible range, as for most materials, it is observed that they are less stiff in shear than in tension or compression. Hence, we would like to highlight that it is an assumption we made, and one might choose a different range to generate the dataset for training. Although  $G_{xy}$  is calculated using Equation 20 in the data generation phase, it is still fed to the neural operators to learn its impact on the material deformation. We assume zero body force and negligible inertia. This leaves us with six parameters to be fed to the branch network:  $t_x, t_y, E_x, E_y, \nu_{xy}$ , and  $G_{xy}$ . The coordinates are fed to the trunk network. A total of 5000 samples were generated using this approach. The dataset was then split into 80% for training and 20% for testing.

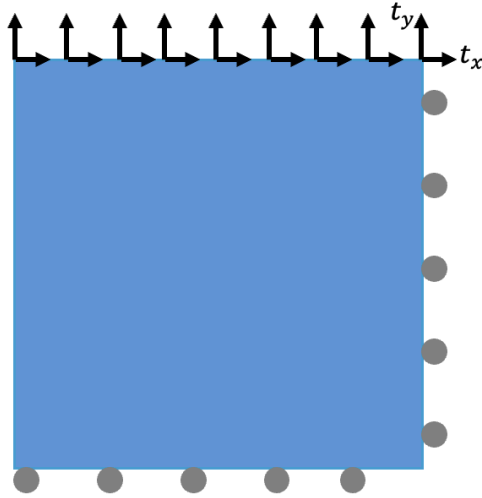


Figure 5: Schematic of the geometry and boundary conditions of the 2D orthotropic material example.

### 3.1.2 Results

In order to compare the training efficiency of the DeepOKAN and DeepONet models, we create the following variations. For the DeepOKAN, both the KAN trunk and branch networks have a width of 6 and 1 hidden layer. Specifically, the resulting architectures (including input and output layers) for the branch and trunk are  $[6, 6, r = 4]$  and  $[2, 6, r = 4]$ , respectively. This yields a setup with 768 learnable parameters. For the DeepONet, the benchmark MLP has 2 hidden layers and 16 neurons per hidden layer for both the trunk and branch. The branch and trunk architectures have the



following shapes:  $[6, 16, 16, r = 4]$  and  $[2, 16, 16, r = 4]$ , respectively. This results in 738 learnable parameters, and the purpose of that model is to have approximately the same number of learnable parameters as in the KAN case. To further increase the approximation capability of the DeepONets, we also adopt the following variations for the MLP hidden layers:  $[16, 16, 16]$ ,  $[16, 16, 16, 16]$ , and  $[64, 64]$ , which result in 1282, 1826, and 9090 learnable parameters, respectively. We train all the networks for 200, 1000, and 5000 epochs using the Adam optimizer. We use a learning rate  $lr = 0.01$  and a learning scheduler every 500 epochs with  $\gamma = 0.5$ . Specifically, the learning rate scheduler is designed to reduce the learning rate by a factor of 0.5 every 500 epochs, allowing the model to make more fine-grained adjustments during training as it approaches convergence. This helps prevent the model from overshooting the optimal solution as learning progresses.

The evolution of the loss function for all the models is shown in Fig. 6. Here, we remark that the learning capability of all the DeepONet variations seems to get quickly stalled, and the loss value remains essentially constant after a few training epochs. This observation holds true regardless of the number of the DeepONet learnable parameters. On the contrary, the DeepOKAN model seems to follow a more healthy and steadily descending trend in its cost function as the number of training epochs increases. This is the first sign that the DeepOKAN shows more numerical flexibility during its training than the DeepONet counterparts.

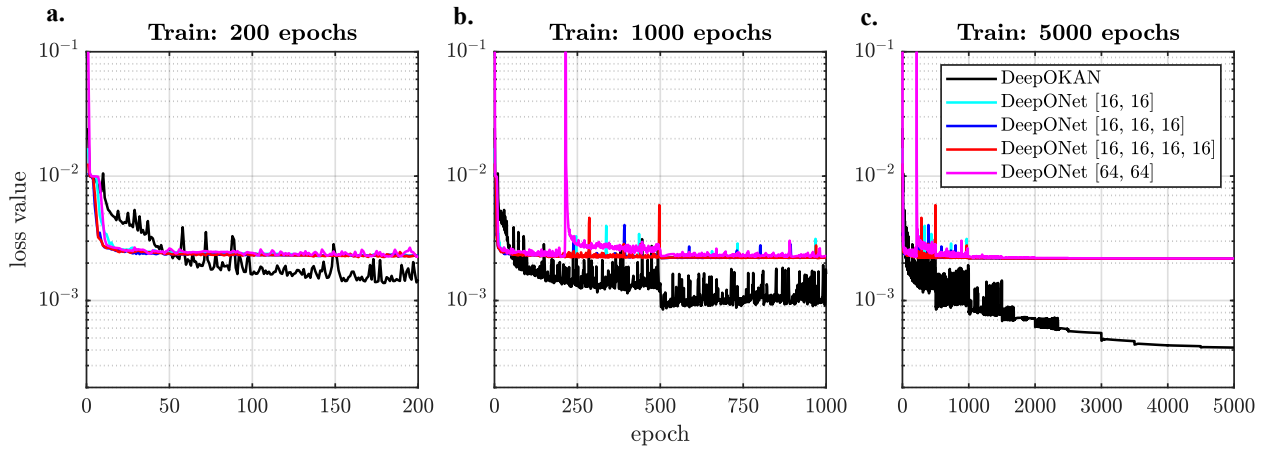


Figure 6: Loss function evolution for the 2D orthotropic elasticity problem, when the networks are trained for **a.** 200, **b.** 1000 and **c.** 5000 Adam epochs.

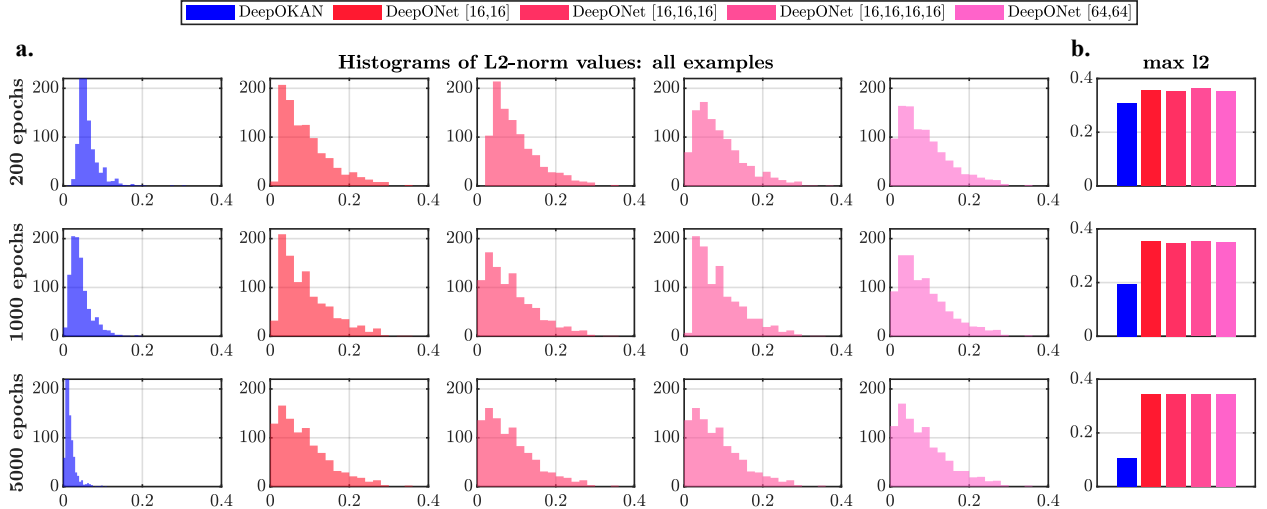


Figure 7: **a.** Histogram of the L2-norm for all the test examples. **b.** Maximum L2-norm value for each case of training epochs.

Next, we evaluate the accuracy of the predictions, and Fig. 7a shows a histogram of the L2-norm values for all the testing examples. The left column corresponds to the DeepOKAN, while the other graphs correspond to the four

DeepONet variations (color description shown in the legend). Each row corresponds to a different number of training epochs (200, 1000, and 5000). Also, to better elucidate the difference in the training efficiency between DeepOKAN and DeepONets, we plot in Fig. 7b the maximum L2-norm value from all the operators. Then, we can make the following observations. First, we note that within the context we investigate, the DeepONet performance remains essentially unaffected by the number of learnable parameters, and increasing the training epochs has only a mild positive impact. On the contrary, the DeepOKAN performance shows a clear improvement with the increased training time. This is shown by the shift of the histograms to the lower range, as the number of epochs increases, and also by the distinct decrease of the maximum L2-norm value as shown in Fig. 7b.

Finally, while monitoring the L2-norm of all examples provides a holistic measure of performance evaluation, comparing the actual prediction fields is also instructive. For this reason, we randomly pick 20 examples from the test dataset, and we plot in Fig 8 the following contours: a) true values, b) predictions and absolute error for the DeepOKAN trained at 5000 epochs, and c) predictions and absolute error for the DeepONet [16,16] trained at 5000 epochs. The colorbars for the predictions are shown in Fig. 8a, and they are kept the same for Figs. 8b and 8c, while the colorbars of the absolute error are all capped between 0 and 0.005. Clearly, we observe that the DeepOKAN predicts with much higher accuracy the queried field, as evidenced by both the prediction and the absolute error contours comparison. Finally, we note that for our comparison, we selected the DeepONet, which had roughly the same number of parameters as the DeepOKAN. However, similar conclusions can be drawn for all the other variations we investigated.

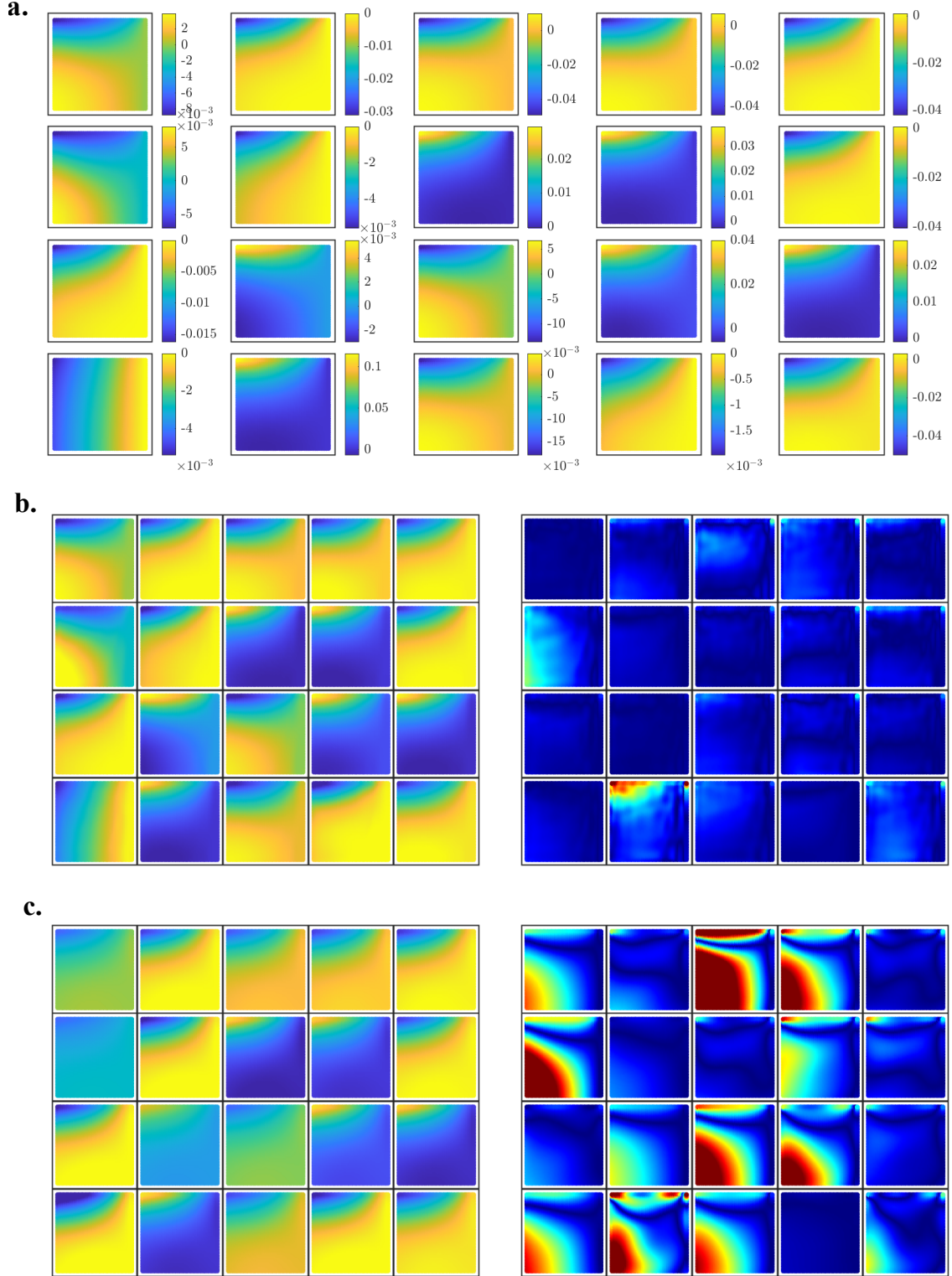


Figure 8: **a.** True values of  $u_x$  for 20 randomly selected test samples. **b.** Predictions and absolute error using the DeepOKAN trained for 5000 Adam epochs. **c.** Predictions and absolute error using the DeepONet [16,16] trained for 5000 Adam epochs.

## 4 Conclusions, limitations, and future work

Neural operators are machine learning models that approximate the solutions of partial differential equations by learning the mapping between function spaces. They provide an efficient and accurate tool for solving complex, high-dimensional problems in physics and engineering. One of the most popular neural operators is the DeepONet. The DeepONet uses MLP networks in the branch and trunk, where the MLP is based on interleaving affine transformations and nonlinear activation functions. MLPs treat the linear transformations and activation functions separately. This paper proposes a new neural operator based on KANs called DeepOKAN. The proposed operator uses KANs in the branch and trunk components, where KANs have learnable activation functions. One may consider different bases for these activation functions. The original work by Liu et al. [29] used B-splines. This paper uses Gaussian RBFs as a basis for the KANs, appearing in both the branch and trunk. The classical data-driven DeepONet framework is effective; however, the proposed DeepOKAN outperforms it. Specifically, within the context of 2D orthotropic elasticity, we observe that a DeepOKAN outperforms a DeepONet with the same number of learnable parameters. We also observe that training DeepOKAN for more epochs yielded a computational boost that was not observed by the DeepONets. More detailed analysis is being conducted on this problem and other solid mechanics cases to explore the differences in training efficiency, accuracy, and expressivity between different operator variations.

## Acknowledgment

This work was partially supported by the Sand Hazards and Opportunities for Resilience, Energy, and Sustainability (SHORES) Center, funded by Tamkeen under the NYUAD Research Institute Award CG013. The authors wish to thank the NYUAD Center for Research Computing for providing resources, services, and skilled personnel. This work was partially supported by Sandooq Al Watan Applied Research and Development (SWARD), funded by Grant No.: SWARD-F22-018.

## Data availability

The data supporting the study's findings will be available upon paper acceptance.

## References

- [1] D. Ammosov, M. Vasilyeva, Online multiscale finite element simulation of thermo-mechanical model with phase change, *Computation* 11 (4) (2023) 71.
- [2] Y. Efendiev, T. Y. Hou, Multiscale finite element methods: theory and applications, Vol. 4, Springer Science & Business Media, 2009.
- [3] M. E. Mobasher, H. Waisman, Dual length scale non-local model to represent damage and transport in porous media, *Computer Methods in Applied Mechanics and Engineering* 387 (2021) 114154.
- [4] M. Torzoni, M. Tezzele, S. Mariani, A. Manzoni, K. E. Willcox, A digital twin framework for civil engineering structures, *Computer Methods in Applied Mechanics and Engineering* 418 (2024) 116584.
- [5] S. Kushwaha, J. He, D. Abueidda, I. Jasiuk, Designing impact-resistant bio-inspired low-porosity structures using neural networks, *Journal of Materials Research and Technology* 27 (2023) 767–779.
- [6] M. Valizadeh, S. J. Wolff, Convolutional neural network applications in additive manufacturing: A review, *Advances in Industrial and Manufacturing Engineering* 4 (2022) 100072.
- [7] B. Paermentier, D. Debruyne, R. Talemi, A machine learning based sensitivity analysis of the gtn damage parameters for dynamic fracture propagation in x70 pipeline steel, *International Journal of Fracture* 227 (1) (2021) 111–132.
- [8] A. Olivier, M. D. Shields, L. Graham-Brady, Bayesian neural networks for uncertainty quantification in data-driven materials modeling, *Computer methods in applied mechanics and engineering* 386 (2021) 114079.
- [9] C. M. Parrott, D. W. Abueidda, K. A. James, Multidisciplinary topology optimization using generative adversarial networks for physics-based design enhancement, *Journal of Mechanical Design* 145 (6) (2023) 061704.
- [10] L. Herrmann, S. Kollmannsberger, Deep learning in computational mechanics: a review, *Computational Mechanics* (2024) 1–51.
- [11] Z. Li, Neural operator: Learning maps between function spaces, in: 2021 Fall Western Sectional Meeting, AMS, 2021.

- [12] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint arXiv:2010.08895.
- [13] Z. Li, D. Z. Huang, B. Liu, A. Anandkumar, Fourier neural operator with learned deformations for pdes on general geometries, *Journal of Machine Learning Research* 24 (388) (2023) 1–26.
- [14] L. Lu, P. Jin, G. Pang, Z. Zhang, G. E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, *Nature machine intelligence* 3 (3) (2021) 218–229.
- [15] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, Deepxde: A deep learning library for solving differential equations, *SIAM review* 63 (1) (2021) 208–228.
- [16] L. Lu, R. Pestourie, S. G. Johnson, G. Romano, Multifidelity deep neural operators for efficient learning of partial differential equations with application to fast inverse design of nanoscale heat transport, *Physical Review Research* 4 (2) (2022) 023210.
- [17] S. Goswami, M. Yin, Y. Yu, G. E. Karniadakis, A physics-informed variational deeponet for predicting crack path in quasi-brittle materials, *Computer Methods in Applied Mechanics and Engineering* 391 (2022) 114587.
- [18] K. Kobayashi, J. Daniell, S. B. Alam, Improved generalization with deep neural operators for engineering systems: Path towards digital twin, *Engineering Applications of Artificial Intelligence* 131 (2024) 107844.
- [19] S. Garg, S. Chakraborty, Vb-deeponet: A bayesian operator learning framework for uncertainty quantification, *Engineering Applications of Artificial Intelligence* 118 (2023) 105685.
- [20] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, G. E. Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data, *Computer Methods in Applied Mechanics and Engineering* 393 (2022) 114778.
- [21] J. He, S. Koric, S. Kushwaha, J. Park, D. Abueidda, I. Jasiuk, Novel deeponet architecture to predict stresses in elastoplastic structures with variable complex geometries and loads, *Computer Methods in Applied Mechanics and Engineering* 415 (2023) 116277.
- [22] J. He, S. Kushwaha, J. Park, S. Koric, D. Abueidda, I. Jasiuk, Sequential deep operator networks (s-deeponet) for predicting full-field solutions under time-dependent loads, *Engineering Applications of Artificial Intelligence* 127 (2024) 107258.
- [23] J. He, S. Kushwaha, J. Park, S. Koric, D. Abueidda, I. Jasiuk, Predictions of transient vector solution fields with sequential deep operator network, arXiv preprint arXiv:2311.11500.
- [24] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, arXiv preprint arXiv:2003.03485.
- [25] J. He, S. Koric, D. Abueidda, A. Najafi, I. Jasiuk, Geom-deeponet: A point-cloud-based deep operator network for field predictions on 3d parameterized geometries, arXiv preprint arXiv:2403.14788.
- [26] W. Zhong, H. Meidani, Physics-informed mesh-independent deep compositional operator network, arXiv preprint arXiv:2404.13646.
- [27] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Mathematics of control, signals and systems* 2 (4) (1989) 303–314.
- [28] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, *Neural networks* 2 (5) (1989) 359–366.
- [29] Z. Liu, Y. Wang, S. Vaidya, F. Ruehle, J. Halverson, M. Soljačić, T. Y. Hou, M. Tegmark, Kan: Kolmogorov-arnold networks, arXiv preprint arXiv:2404.19756.
- [30] N. Kolmogorov, On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables, *American Mathematical Society*, 1961.
- [31] V. I. Arnol'd, On the representation of continuous functions of three variables by superpositions of continuous functions of two variables, *Matematicheskii Sbornik* 90 (1) (1959) 3–74.
- [32] J. Braun, M. Griebel, On a constructive proof of kolmogorov's superposition theorem, *Constructive approximation* 30 (2009) 653–675.
- [33] D. A. Sprecher, S. Draghici, Space-filling curves and kolmogorov superposition-based neural networks, *Neural Networks* 15 (1) (2002) 57–67.
- [34] M. Köppen, On the training of a kolmogorov network, in: *Artificial Neural Networks—ICANN 2002: International Conference Madrid, Spain, August 28–30, 2002 Proceedings* 12, Springer, 2002, pp. 474–479.
- [35] J.-N. Lin, R. Unbehauen, On the realization of a Kolmogorov network, *Neural Computation* 5 (1) (1993) 18–20.

- 
- [36] M.-J. Lai, Z. Shen, The Kolmogorov superposition theorem can break the curse of dimensionality when approximating high dimensional functions, arXiv preprint arXiv:2112.09963.
  - [37] D. Fakhoury, E. Fakhoury, H. Speleers, Exsplinet: An interpretable and expressive spline-based neural network, *Neural Networks* 152 (2022) 332–346.
  - [38] R. Genet, H. Inzirillo, Tkan: Temporal kolmogorov-arnold networks, arXiv preprint arXiv:2405.07344.
  - [39] Z. Li, Kolmogorov-arnold networks are radial basis function networks, arXiv preprint arXiv:2405.06721.
  - [40] S. Satapathy, S. Dehuri, A. Jagadev, S. Mishra, Empirical study on the performance of the classifiers in eeg classification, *EEG Brain Signal Classification for Epileptic Seizure Disorder Detection* (2019) 45–65.