

A comprehensive and FAIR comparison between MLP and KAN representations for differential equations and operator networks

Khemraj Shukla^{a,1}, Juan Diego Toscano^{a,1}, Zhicheng Wang^{a,1}, Zongren Zou^{a,1}, George Em Karniadakis^{a,b,2}

^a*Division of Applied Mathematics, Brown University, Providence, RI 02906, USA*

^b*Pacific Northwest National Laboratory, Richland, WA 99354, USA*

Abstract

Kolmogorov-Arnold Networks (KANs) were recently introduced as an alternative representation model to MLP. Herein, we employ KANs to construct physics-informed machine learning models (PIKANs) and deep operator models (DeepOKANs) for solving differential equations for forward and inverse problems. In particular, we compare them with physics-informed neural networks (PINNs) and deep operator networks (DeepONets), which are based on the standard MLP representation. We find that although the original KANs based on the B-splines parameterization lack accuracy and efficiency, modified versions based on low-order orthogonal polynomials have comparable performance to PINNs and DeepONet although they still lack robustness as they may diverge for different random seeds or higher order orthogonal polynomials. We visualize their corresponding loss landscapes and analyze their learning dynamics using information bottleneck theory. Our study follows the FAIR principles so that other researchers can use our benchmarks to further advance this emerging topic.

Keywords: Scientific machine learning, Kolmogorov-Arnold networks, physics-informed neural networks, operator networks

1. Introduction

Multilayer perceptrons (MLPs) are a class of feedforward artificial neural networks consisting of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer [1, 2]. As stated in the universal approximation theorem [3], MLPs can learn non-linear relationships and patterns in data, making them one of the main building blocks of modern deep learning applications [4–9]. However, due to their complex and deeply nested structure, MLPs lack interpretability [10] and often face challenges such as overfitting, vanishing or exploding gradients, and scalability issues.

As an alternative to MLP, researchers have recently proposed Kolmogorov-Arnold Networks (KANs), a new type of model that aims to be more accurate and interpretable than MLP [11]. KANs are inspired by the Kolmogorov-Arnold representation theorem and can

¹These authors contributed equally to this work and are listed in alphabetical order by last name.

²Corresponding author: george.karniadakis@brown.edu (George Em Karniadakis).

be interpreted as a combination of Kolmogorov Networks [12–17] and MLPs with learnable activation functions [18–20]. KANs and their rapidly growing extensions have shown promising performance in addressing several MLP issues, such as interpretability and catastrophic forgetting in supervised and unsupervised learning tasks [11, 21, 22]. However, their formulation relies on learnable B-Splines as activation functions, significantly increasing their computational cost. To address this problem, several subsequent studies proposed using alternative univariate functions such as radial basis functions (RBF) [23], wavelets [24] or Jacobi polynomials [25–28] (e.g., Chebyshev, Legendre).

KANs have also been explored for solving differential equations and operator learning. Liu et al. [11] combined physics-informed neural networks (PINNs) [29] and KANs to solve a 2D Poisson equation. Similarly, Abueida et al. proposed DeepOKAN [30], an RFB-based KAN operator network, to solve a 2D orthotropic elasticity problem. The authors in [11] and [30] show that KANs significantly outperform MLP; however, these studies were limited to shallow networks and were conducted on simplified problems.

Herein, we employ physics-informed machine learning [31] and KANs to develop PIKANs and operator models (DeepOKANs). In the first part of this study, we systematically compare PINN and PIKAN variations on six benchmarks carefully selected from the current literature [32–35]. To allow a fair comparison between these representation models, we combine them with state-of-the-art optimization techniques such as residual-based attention [36] and eddy viscosity formulations [33], which enable these new models to solve more complex problems. In this section, we analyze PIKAN stability and sensitivity to higher polynomial orders and the number of layers. In the second part, we compare DeepOKANs and DeepONets for two operator learning tasks.

In the last section of this paper, we analyze PIKAN learning dynamics through the lens of the information bottleneck (IB) theory [37, 38]. According to the IB theory, a well-functioning model should retain essential output information while discarding insignificant input details, thereby creating an “information bottleneck” that induces two distinct stages of training “fitting” and “diffusion” [39, 40]. Recently, Anagnostopoulos et al., [36] extended this theory to PINNs and proposed the existence of a third phase named “total diffusion”. Following [36], we analyze the PIKAN training dynamics and identify the three stages of learning observed in PINNs, bridging the gap between both representation models.

This paper is organized as follows. In Section 2, we briefly describe the problem and the representation models. Section 3 compares the performance of both representation models in eight benchmarks, including discontinuous function approximation, structure-preserving Hamiltonian dynamical systems, PDE solution approximation, uncertainty quantification, and operator learning. Finally, Section 4 analyzes the training dynamics of PINNs and PIKANs based on the IB method. We summarize in Section 5.

2. Problem Formulation and representation models

Consider the following nonlinear ODE/PDE:

$$\mathcal{F}_\lambda[u](x) = f(x), x \in \Omega, \quad (1a)$$

$$\mathcal{B}_\lambda[u](x) = b(x), x \in \partial\Omega, \quad (1b)$$

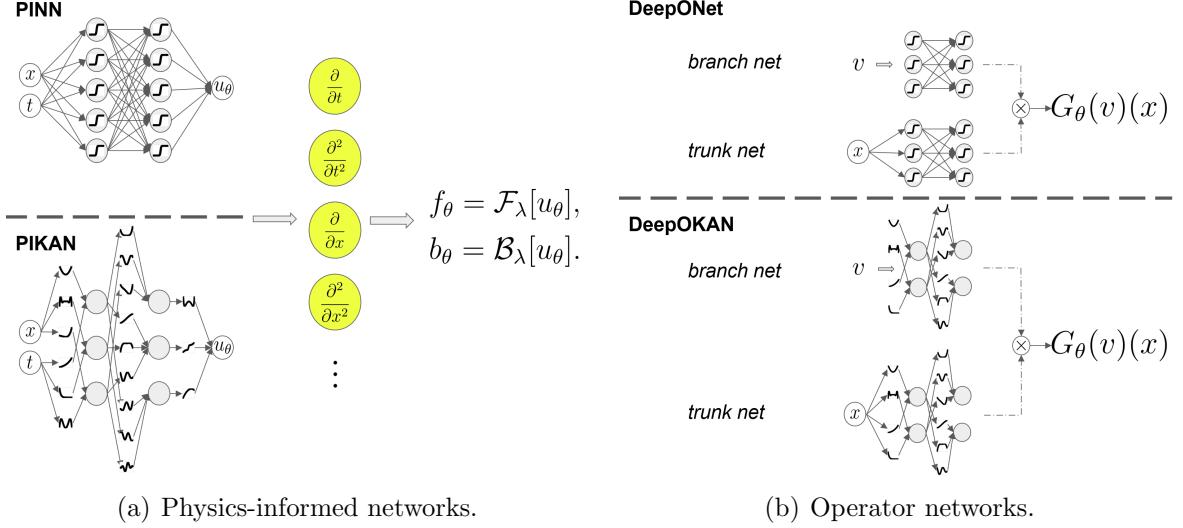


Figure 1: An illustration of MLP and KAN for (a) differential equations and (b) operator networks. We choose DeepONet [34] as the representation model for operator learning. Here activation function for MLP in PINNs and DeepONets is chosen as the hyperbolic tangent only for the demonstration.

where x is the spatial-temporal coordinate, u is the solution, λ is the model parameter, f is the source term, b is the boundary term, and \mathcal{F} and \mathcal{B} are general nonlinear differential and boundary operators, respectively. In the literature [31, 41], there are, generally, two ways to solve (1) with modern machine learning techniques. One is approximating the solution u with parameterized model, u_θ where θ denotes the parameter, constructing physics-informed loss function via automatic differentiation, and finding θ such that the loss function is minimized [29, 42–50]. Representative model is physics-informed neural networks (PINNs) [29]. In this work, we refer to it as the *neural differential equation*. Another one is learning the solution operator, which maps f, b and/or λ to u , using NNs. Representative models are deep operator networks (DeepONets) [34] and Fourier neural operators (FNOs) [51]. We refer to them as *neural operators*. The main difference between neural differential equations and neural operators is that the former targets at solving one specific ODE/PDE, in which the training of NNs gives an approximated solution that maps point to point, while the latter aims to solve a family of ODEs/PDEs, in which NNs map functions to functions.

2.1. Physics-informed neural networks (PINNs)

The PINN method [29] solves the problem involving (1) by modeling the sought solution with a NN, denoted by u_θ , and then modeling f and b with $\mathcal{F}[u_\theta]$ and $\mathcal{B}[u_\theta]$ via automatic differentiation, respectively. The differential equation is explicitly encoded by constructing the physics-informed loss function as follows:

$$\begin{aligned} \mathcal{L}(\theta) = & \frac{w_u}{N_u} \sum_{i=1}^{N_u} \|\alpha_i(u_\theta(x_i^u) - u_i)\|^2 + \frac{w_f}{N_f} \sum_{j=1}^{N_f} \|\alpha_j(\mathcal{F}_\lambda[u_\theta](x_j^f) - f_j)\|^2 + \\ & \frac{w_b}{N_b} \sum_{l=1}^{N_b} \|\alpha_l(\mathcal{B}_\lambda[u_\theta](x_l^b) - b_l)\|^2, \end{aligned} \quad (2)$$

where w_u, w_f, w_b are belief weights for different terms in the loss function, $\|\cdot\|$ is the l^2 norm for finite-dimensional vectors, $\{x_i^u, u_i\}_{i=1}^{N_u}, \{x_j^f, f_j\}_{j=1}^{N_f}, \{x_l^b, b_l\}_{l=1}^{N_b}$ are data for u, f, b , and α_i, α_j and α_l are local weights (such as residual-based attention weights) that balance the loss contribution of the training points i, j and l , respectively. We note that $N_u = 0$ when the ODE/PDE is solved with known λ , which is often referred to as the *forward problem* [29, 45] in the literature, while $N_u \neq 0$ when λ is unknown, which is referred to as the *inverse problem* [52, 53].

Residual-Based Attention. One of the inherent challenges in training neural networks is that the residuals (i.e., point-wise errors) can get overlooked when calculating the cumulative loss function (i.e., summation or mean of the residuals). To address this issue, several studies proposed scaling the loss terms using local multipliers [32, 54]. Local multipliers such as residual-based attention (RBA) weights [54] or self-adaptive weights [32] have shown a remarkable performance in physics-informed neural networks and other supervised learning tasks. These weights balance the contribution of specific training points within each loss term inducing a residual homogeneity [36]. RBA weights are based on the exponentially weighted moving average of the residuals. Thus, since the loss residuals contain information about the high error regions, the obtained multipliers work as an attention mask that helps the optimizer focus on capturing the spatial or temporal characteristics of the specific problem [54].

The update rule for RBA for any training point i on iteration k is given by:

$$\alpha_i^{k+1} \leftarrow (1 - \eta^*)\alpha_i^k + \eta^* \frac{|e_i|}{\|e\|_\infty}, \quad i \in \{0, 1, \dots, N\}, \quad (3)$$

where N is the number of training points, e_i is the residual of the respective loss term for point i and η^* is a learning rate. This is a convergent linear homogeneous recurrence relation that bounds our RBA between zero to one ($\alpha \in [0, 1]$).

2.2. Neural operators (NOs)

NOs solve (1) by approximating the solution operator, denoted as G_θ , using NNs from data [34, 41, 51, 55–57]. Unlike neural differential equations in which the NN maps point to point, i.e. the spatial-temporal coordinate to the value of the function evaluated at this coordinate, NOs address mappings from functions to functions, e.g. the mapping from the source term f to the sought solution u , and can be used as fast solvers for forward problems and physics encoders [51, 58–60] for inverse problems. We denote the input function as v and the output function as u , and then the loss function can be formulated as follows:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \frac{1}{N_u^i} \sum_{j=1}^{N_u^i} \|G_\theta(\mathbf{v}_i)(x_i^j) - u_i^j\|^2, \quad (4)$$

where $\{\mathbf{v}_i, \{x_i^j, u_i^j\}_{j=1}^{N_u^i}\}_{i=1}^N$ are the data for training the NO. Here, N denotes the number of paired data for the input function v and the output function u , \mathbf{v} denotes the finite representation of v , and $N_u^i, i = 1, \dots, N$ is the number of measurements for the i th data for u , which is denoted by $x_i^j, u_i^j, j = 1, \dots, N_u^i$.

2.3. Representation Models

2.3.1. Multilayer Perceptron (MLP)

The output y of a Multilayer Perceptron (MLP) can be described by the following nested formulation, where σ denotes the activation function, $W^{(l)}$ and $b^{(l)}$ are the weights and biases of the l -th layer, respectively:

$$y(\mathbf{x}) = \sigma(W^{(L)}\sigma(W^{(L-1)} \dots \sigma(W^{(1)}\mathbf{x} + b^{(1)}) \dots + b^{(L-1)}) + b^{(L)})$$

In this formula, $\mathbf{x} = (x_1, x_2, \dots)$ represents the input vector, and L is the number of layers. Each layer's output serves as the input for the next layer, culminating in the final output y . This structure, combined with sufficiently many neurons and the right choice of activation function, allows MLPs to approximate virtually any continuous function on compact subsets of \mathbb{R}^n , as stated by the Universal Approximation Theorem [3]. This theorem underpins the ability of neural networks to model complex, nonlinear relationships.

The combination of physics-informed machine learning and MLPs is called physics-informed neural networks (PINNs). Based on this definition, we can define the specific number of parameters ($|\theta|$) of PINNs as follows:

$$|\theta|_{PINN} = H[I + (n_l - 1)H + O] \sim \mathcal{O}(n_l H^2) \quad (5)$$

where I and O are the numbers of inputs and outputs, n_l is the number of hidden layer and H is the number of neurons per hidden layer.

2.3.2. Kolmogorov-Arnold networks (KANs)

Kolmogorov-Arnold networks (KANs) are a novel type of neural network inspired by the Kolmogorov-Arnold representation theorem. This theorem states that any multivariate continuous function $f(\mathbf{x}) = f(x_1, x_2, \dots)$ on a bounded domain can be represented as a finite composition of continuous functions of a single variable, and the binary operation of addition [11]. Motivated by this theorem, [11] proposed approximating $f(\mathbf{x})$ as follows:

$$f(\mathbf{x}) \approx \sum_{i_{L-1}=1}^{n_{L-1}} \phi_{L-1,i_L,i_{L-1}} \left(\sum_{i_{L-2}=1}^{n_{L-2}} \dots \left(\sum_{i_2=1}^{n_2} \phi_{2,i_3,i_2} \left(\sum_{i_1=1}^{n_1} \phi_{1,i_2,i_1} \left(\sum_{i_0=1}^{n_0} \phi_{0,i_1,i_0}(x_{i_0}) \right) \right) \right) \dots \right) \quad (6)$$

The right-hand side of (6) represents a KAN ($KAN(\mathbf{x})$), where L denotes the number of layers, $\{n_j\}_{j=0}^L$ are the numbers of nodes (i.e., neurons) in the j^{th} layer, and $\phi_{i,j,k}$ are the univariate activation functions. The specific form of each $\phi(x)$ defines the variations among different KAN architectures.

Vanilla KAN (PIKAN). In the original implementation [11] proposed defining $\phi(x)$ as a weighted combination of a basis function $b(x)$ and B-splines. In particular:

$$\phi(x) = w_b b(x) + w_s \text{spline}(x)$$

where the basis function $b(x)$ and the spline function $\text{spline}(x)$ are defined as follows:

$$b(x) = \frac{x}{1 + e^{-x}}$$

$$\text{spline}(x) = \sum_i c_i B_i(x)$$

here, c_i , w_b and w_s are trainable parameters. The splines $B_i(x)$, are characterized by the polynomial order k , and the number of grid points g . Notice that, under this formulation, the trainable parameters define the contribution of each univariate function. In this study, we denote PIKAN as the combination of physics-informed machine learning and vanilla KANs. As described in [11], the total number of parameters of KANs (and PIKANs) can be quantified as follows:

$$|\theta|_{\text{PIKAN}} = H[I + (n_l - 1)H(k + g) + O] \sim \mathcal{O}(n_l H^2(k + g)) \quad (7)$$

where, I and O are the numbers of inputs and outputs, n_l is the number of hidden layer, H is the number of neurons per hidden layer, g is the grid size, and k is the polynomial order.

Since the debut of the KAN in April 2024, researchers across the world have been actively exploring the development of KANs tailored for diverse applications. There are quite a few KAN variations appear for testing, see the Github page for the KANs collection [61]. Among them, we would like to mention following KAN variations,

Radial Basis Function(RBF) KANs. It is reported by [23] that using the radial basis functions (RBFs) with Gaussian kernels toapproximates the 3-order B-spline basis, in addition with layer normalization that can prevent the inputs shifting away from the domain of the RBFs, the vanila KAN can accelerate the training without loss of accuracy.

Wavelet KANs. Wavelet that uses orthogonal or semi-orthogonal basis, has the capability to maintain a balance between accurately representing the underlying data structure and avoiding overfitting to the noise. It is reported by [24] that the wavelet KAN is able to enhance the accuracy, speedup the training, and increase the robustness compared MLPs.

Jacobi KANs. Jacobi polynomials are orthogonal polynomials defined on the interval [-1, 1]. They are very popular at high order numerical methods for computational fluid dynamics [62]. The Jacobi polynomials can be calculated recursively. Note that Chebyshev polynomials and Legendre polynomials are special cases of Jacobi polynomials. Implementation of the former can be found in [26], while the latter is available on [63]. It is worth noting that the the work by [27] shows that the Chebyshev KAN is more efficient than the original KAN implementation and it might represent a promising step towards leveraging theoretical foundations and efficient approximation techniques in the field of machine learning.

This study defines cPIKAN as the combination of physics-informed machine learning and Chebyshev KAN. cPIKANs do not require grid points as PIKANs which reduces the number of trainable parameters $|\theta|$ to:

$$|\theta|_{\text{cPIKAN}} = H[I + (n_l - 1)Hk + O] \sim \mathcal{O}(n_l H^2 k) \quad (8)$$

as in the previous models, I and O are the numbers of inputs and outputs, n_l is the number of hidden layer, H is the number of neurons per hidden layer, and k is the polynomial order.

3. Computational experiments

In this section, we present a series of computational experiments comparing the efficacy and accuracy of MLP and KAN-based architectures in solving SciML problems. Specifically, we focus on utilizing KAN-based architectures to solve steady and unsteady partial differential equations, perform operator regression in low and high-dimensional regimes, and explore the applicability of KAN-based architectures for solving PDEs with noisy data by utilizing the Bayesian framework. All these experiments are further benchmarked against contemporary MLP-based architectures.

3.1. Approximation of a discontinuous and oscillatory function

Here, we compare the approximation capability of KAN and Chebyshev-KAN against the MLP architecture. To illustrate this, we select a function that includes a discontinuity and various high and low-frequency modes. The selection of this function aims to evaluate the robustness of KAN and MLP-based architectures in addressing the prevalent phenomenon of spectral bias in neural networks, as discussed in [64]. The function is expressed as follows,

$$y = \begin{cases} 5 + \sum_{k=1}^4 \sin(kx), & x < 0, \\ \cos(10x), & x \geq 0. \end{cases} \quad (9)$$

To approximate function in (9), we implement KAN [23], Chebyshev-KAN [27], Modfied-Chebyshev-KAN and MLP based architectures with hyper-parameter shown in Table 1. To achieve this approximation, we fix a neural network architecture consisting of 2 hidden layers, each containing 40 neurons. However, with this architecture, the number of trainable parameters for Chebyshev-KAN and MLP are in the same order of magnitude, whereas for the KAN architecture, the number of parameters increases by an order of magnitude. Therefore, to ensure a fair comparison, we present the regression results using two architectures for KAN, named KAN-I and KAN-II. KAN-I represents the same number of layers and neurons as MLP and Chebyshev-KAN, while KAN-II represents the same number of parameters as MLP and Chebyshev-KAN. To perform these regressions for all the cases described in Section 3.1, we utilize the Adam optimizer. Before discussing the main approximation results, we want to highlight the instability encountered during the training of Chebyshev-KAN architectures for approximating the function (9). In Figure 2, we present the approximation of (9) using Chebyshev-KAN. Figure 2(a) shows the reference and approximated function plots, indicating a l_2 - relative error of 7.43%. Additionally, the training becomes unstable after 2000 iterations, with the loss converging to a NaN value, as represented in Figure 2(b) by a very high double precision number. In Figure 2(c), we compare the Fourier spectrum of the reference and approximated functions, clearly showing that the network fails to accurately learn high frequencies. To address this instability, we modified the architecture by composing each Chebyshev-KAN layer with a tanh function, except for the last layer. Thus forward pass of modified Chebyshev-KAN layer with 1-hidden layer is expressed as

$$y = (\Phi \circ \tanh \circ \Phi)(x), \quad (10)$$

where Φ defines Chebyshev layer.

In Figure 3 (a)-(d), we present the function approximation results obtained from the KAN-I, KAN-II, modified Chebyshev-KAN (equation (10)), and MLP architectures, respectively. The relative l_2 - errors between the reference and approximated functions for the KAN-I, KAN-II, modified Chebyshev-KAN, and MLP architectures are 0.29%, 0.33%, 0.79%, and 0.81%, respectively. The expressivity of the modified Chebyshev-KAN is similar to that of the MLP architecture, although the MLP is slightly more efficient in terms of runtime. The accuracy of the KAN-I and KAN-II architectures is almost the same, despite the KAN-II having an order of magnitude fewer parameters.

In Figure 4(a)-(d), we display the Fourier spectrum of the reference and approximated functions obtained from the KAN-I, KAN-II, modified Chebyshev-KAN, and MLP-based architectures. All four architectures successfully captured all frequencies and exhibited excellent agreement with the reference spectra. In Figure 5, we display the trajectories of convergence loss for the KAN-I, KAN-II, modified Chebyshev-KAN, and MLP architectures. We conducted the training for 100,000 iterations for Chebyshev-KAN and MLP due to their efficiency, whereas for KAN-I and KAN-II, the training was halted at 20,000 and 25,000 iterations, respectively, as the loss values for all four architectures converged to almost identical values. Notably, the rate of convergence for KAN-I and KAN-II is steeper compared to the modified Chebyshev-KAN and MLP architectures.

Methods	No. of parameters	Degree of polynomial	Rel. l_2 - error	Time: ms/iter.
KAN-I Figure 3a	37041	3	0.29%	1586.15
KAN-II Figure 3b	4317	3	0.32%	182.22
Chebyshev-KAN Figure 2	4352	3	7.43%	2.34
Modified Chebyshev-KAN Figure 3c	4352	3	0.79%	2.30
MLP Figure 3d	3401	-	0.81%	1.30

Table 1: Hyperparameters of networks for approximating the function (9) for results show in Figure 2. Here KAN-I and KAN-II represnts the original KAN [23] however they only differ in number of parameters. The time per iteration is GPU time, measured on an Nvidia’s GeForce RTX-3090 equipped with 24 GB of memory.

3.2. Structure preserving Dynamical System: Hamiltonian neural network (HNN) vs Hamiltonian Chebyshev-KAN (HcKAN)

This study investigates whether Chebyshev-KAN neural networks can effectively predict the phase space of a dynamical system while preserving its energy (Hamiltonian). Traditional Multilayer Perceptrons (MLPs) struggle with this task because they lack the necessary inductive biases to guide their learning. [65] proposed incorporating the Hamiltonian into the training process of MLPs to address this limitation. To showcase the potential of Chebyshev-KAN for such systems, we will use a simple example: an ideal mass-spring system. The Hamiltonian for this system is given by:[66]

$$H(p, q) = \frac{1}{2}kq^2 - \frac{p^2}{2m} \quad (11)$$

The architecture of HNN and HcKAN is shown in Figure 6(a) and Figure 6(b), respectively. We modified the HNN in Figure 6(a) by replacing the MLP layer with with Chebyshev-KAN layer and perform the training for both HNN and HcKAN architecture by minimizing the following loss function

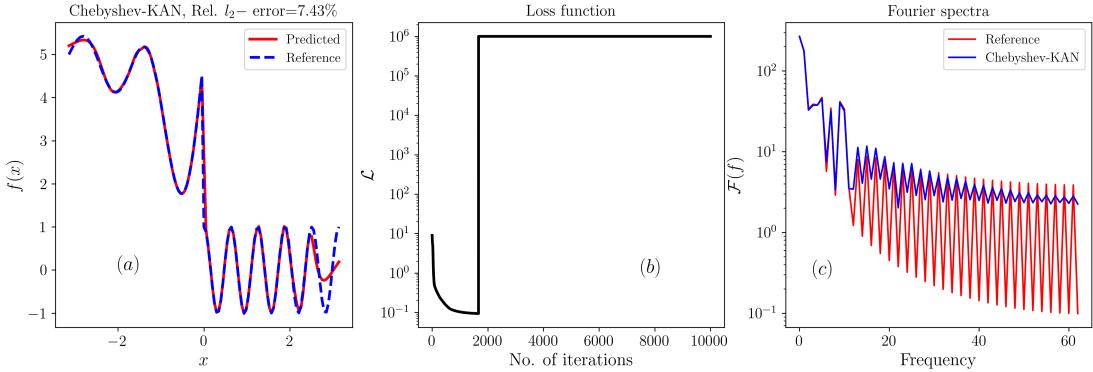


Figure 2: Expressivity of Chebyshev-KAN while approximating the function (9) is shown here. Subfigure (a) compares the reference and approximated functions, with the approximation by Chebyshev-KAN exhibiting a large error of 7.43%. Subfigure (b) depicts the trajectory of the training loss, noting that training becomes unstable after the 2000th iteration, leading to NaN loss values, which are represented using a very high value of order six. Subfigure (c) compares the spectra of the reference and approximated functions, highlighting Chebyshev-KAN’s failure to capture the high frequencies, resulting in the significant error.

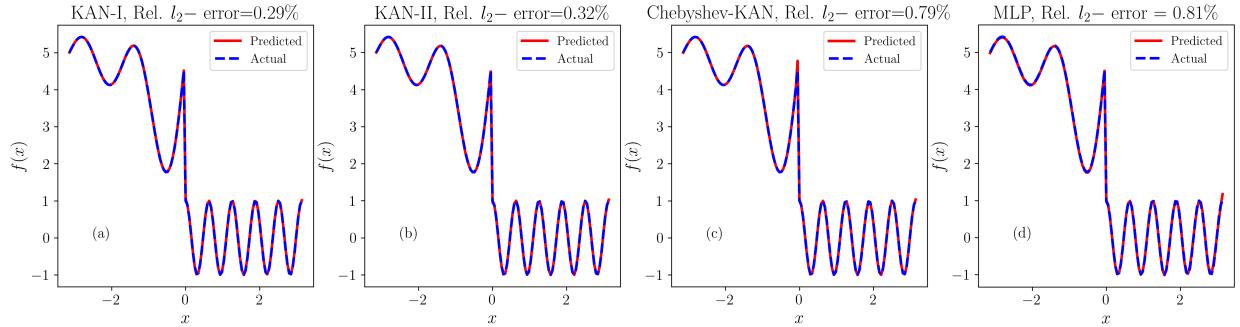


Figure 3: A comparison between reference and approximated of (9) using (a) KAN-I, (b) KAN-II, (c) modified Chebyshev-KAN and (d) MLP based architectures.

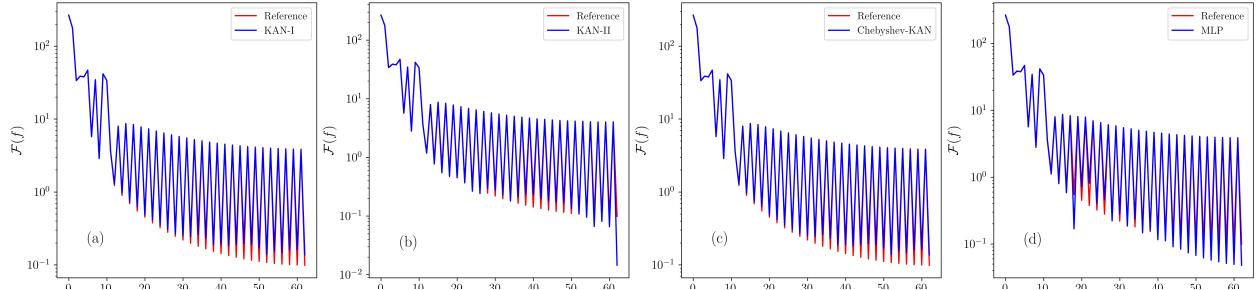


Figure 4: A comparison between the spectrum of reference and approximated function obtained using (a) KAN-I, (b) KAN-II, (c) modified Chebyshev-KAN and (d) MLP architectures. Fourier spectra of approximated function obtained from all the four architectures are in very good agreement with the reference one. The long tail of oscillation represents the discontinuity present in the function (9).

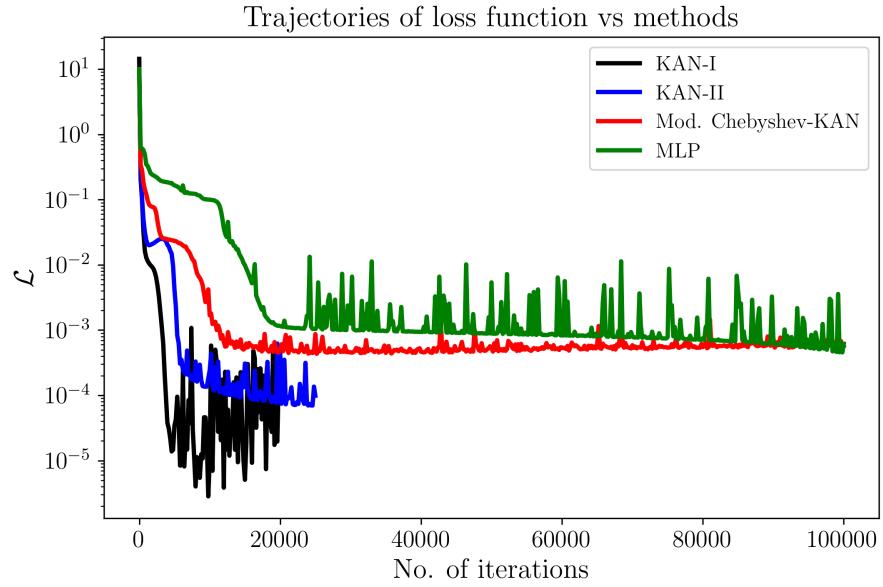


Figure 5: Loss functions for function approximation

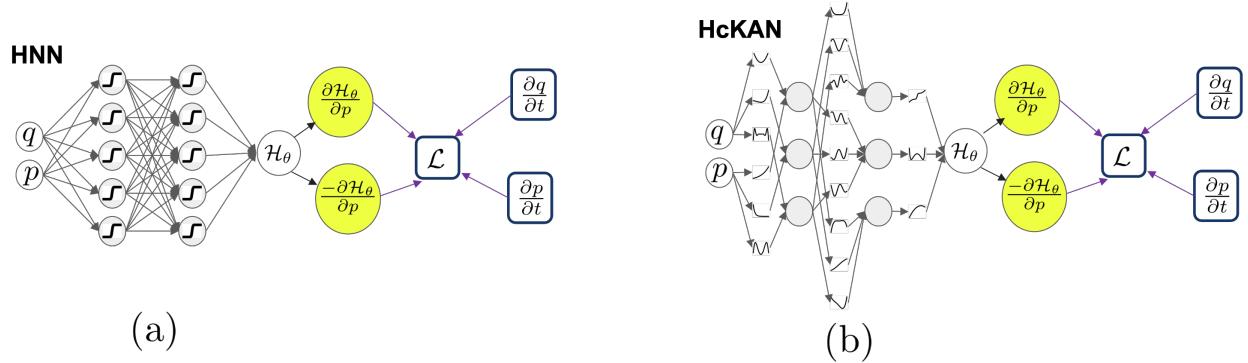


Figure 6: Architectures of (a) HNN [65] and (b) HeKAN used for forecasting the state of the dynamical system defined by (11)

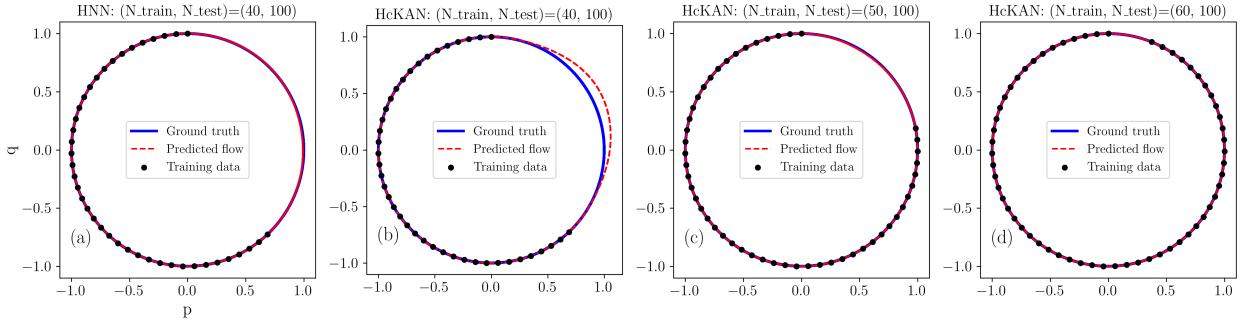


Figure 7: Learned vs actual state space of ideal mass-spring system shown by equation (11). (a) shows the comparison between actual and learned state space (p, q) by HNN [65] by using $[N_{\text{train}}, N_{\text{test}}] = [40, 100]$, N_{train} and N_{test} are number of training and testing samples. (c), (d), and (e) represent the learned and predicted state space using $[N_{\text{train}}, N_{\text{test}}] = [40, 100]$, $[N_{\text{train}}, N_{\text{test}}] = [40, 100]$, and $[N_{\text{train}}, N_{\text{test}}] = [40, 100]$, respectively. The black solid circle represent the training sample however blue and dashed red lines show the actual and predicted p and q , respectively.

$$\mathcal{L} = \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{p}} - \frac{\partial \mathbf{q}}{\partial t} \right\|_2 + \left\| \frac{\partial \mathcal{H}_\theta}{\partial \mathbf{q}} + \frac{\partial \mathbf{p}}{\partial t} \right\|_2, \quad (12)$$

where p and q are position and momentum of the system described by equation (11).

Methods	No. of parameters	Degree of polynomial	N_{train}	Time: ms/iter.
HNN Figure 6a	4417	-	40	1.53
HcKAN Figure 6b	920	3	60	2.97

Table 2: Hyperparameters used for training HNN and HcKAN. The time per iteration is GPU time, measured on an Nvidia’s GeForce RTX-3090 equipped with 24 GB of memory.

The hyperparameters used to train HNN and HcKAN are provided in Table 2. To train these networks, we use the Adam Optimizer with static learning rate of 1e-3. Figure 7(a)-(d) showcases how HNN and HcKAN models can learn the state space (p, q) of the ideal mass-spring system defined by equation 11. Additionally, Figure 8(a)-(d) present the corresponding convergence history of these models during the training process. Panel (a) of Figure 7 focuses on the state space learned by HNN using 40 training samples and shows good agreement between the predicted and actual state space, indicating the model’s strong extrapolation capability. This observation is further supported by the convergence behavior of the training and testing loss curves in Figure 8(a). While the curves exhibit signs of overfitting after approximately 50,000 iterations, we selected the model with the lowest test loss for prediction purposes. In Panel (b) of the figure Figure 7 presents the state space predicted by HcKAN using the same number of training and testing samples as those employed for HNN in panel (a). Initially, we attempted to use HcKAN with the same architecture and number of parameters as the HNN (refer to Table 2). However, the training process became unstable and diverged rapidly. Reducing the number of parameters in HcKAN did not alleviate this

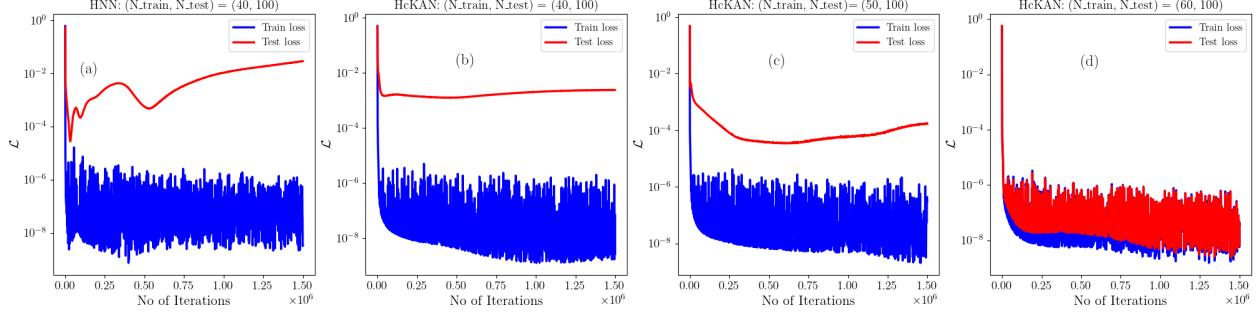


Figure 8: Train and test Loss function showing the convergence history of (a) HNN, (b) HeKAN: [N_train, N_test]=[40, 100], (c) HeKAN: [N_train, N_test]=[50, 100] and (d) (b) HeKAN: [N_train, N_test]=[60, 100].

issue. To achieve stability, we employed a modification of the Chebyshev-KAN architecture, as defined by equation 10, along with a shallower network. Panel (b) of Figure 7 compares the actual and predicted state space (p, q) obtained using the HeKAN model trained with 40 samples and validated with 100 additional samples. The results indicate that the extrapolation capability of HeKAN is not as strong as that of HNN and therefore have larger generalization error. The convergence history of the test loss in Figure Figure 8(b) further corroborates this observation. The test loss remains stagnant at a high value, indicating that the model is not generalizing well to unseen data. This improvement is further supported by the loss plots shown in Figure 8(c) and (d). We observe a substantial decrease in test loss (by several orders of magnitude) when training with a slightly larger dataset. The execution times (runtime) for HNN and HeKAN are presented in Table 2. These measurements were conducted on an Nvidia GeForce RTX-3090 GPU. It is important to note that when HeKAN has a small number of parameters, its runtime per iteration can be higher than that of HNN. This is likely due to underutilized GPU resources; with utilization below 10%, the latency associated with data transfer between DRAM and the processor becomes more significant than the actual computation time.

3.3. Helmholtz Equation

The 2D Helmholtz PDE is defined as follows:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + k^2 u - q(x, y) = 0, \quad (13)$$

where $q(x, y)$ is a forcing term,

$$\begin{aligned} q(x, y) = & -(a_1 \pi)^2 \sin(a_1 \pi x) \sin(a_2 \pi y) \\ & -(a_2 \pi)^2 \sin(a_1 \pi x) \sin(a_2 \pi y) \\ & + k \sin(a_1 \pi x) \sin(a_2 \pi y), \end{aligned} \quad (14)$$

that leads to the analytical solution $u(x, y) = \sin(a_1 \pi x) \sin(a_2 \pi y)$ [32]. For this problem, the boundary conditions are expressed as:

$$u(-1, y) = u(1, y) = u(x, -1) = u(x, 1) = 0, \quad (15)$$

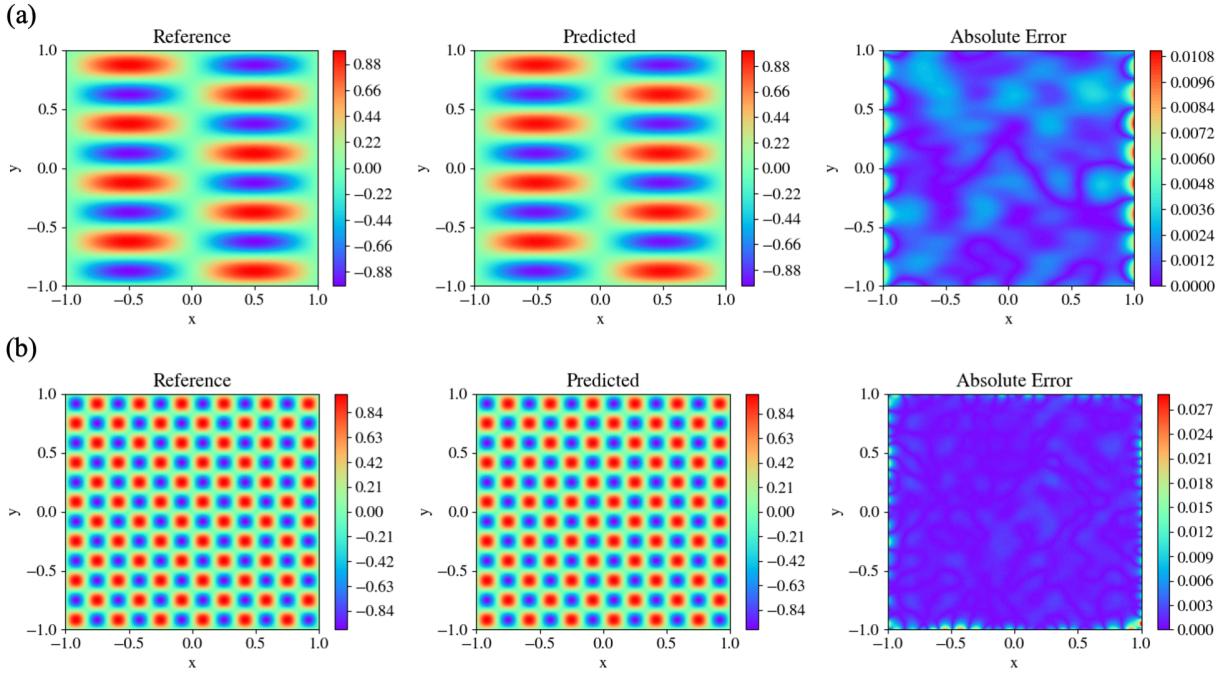


Figure 9: Reference solution with the corresponding cPIKAN+RBA prediction and the absolute error difference for different wave numbers and optimizers. (a) Helmholtz Equation with $a_1 = 1$ and $a_2 = 4$ trained with LBFGS optimizer for 1.8e3 iterations. (b) Helmholtz Equation with $a_1 = a_2 = 6$ trained with Adam optimizer for 5.0e5 iterations.

We approximate the solution of the Helmholtz equation ($a_1 = 1$ and $a_2 = 4$) with a PINN and a PIKAN by minimizing the combined loss function described in equation (16).

$$\mathcal{L} = w_{bc}\mathcal{L}_{bc} + w_{pde}\mathcal{L}_{pde}, \quad (16)$$

here w_{bc} and w_{pde} are global weights that modify the contribution of the averaged loss terms for boundary conditions (\mathcal{L}_{bc}) and PDE residuals (\mathcal{L}_{pde}) which are described as follows:

$$\mathcal{L}_{bc} = \langle (\alpha_i \cdot \sum_{b=1}^2 |\mathcal{R}_{i,b}|)^2 \rangle_i, \quad (17)$$

$$\mathcal{L}_{pde} = \langle (\alpha_j \cdot |\mathcal{R}_j|)^2 \rangle_j, \quad (18)$$

here, $\langle \cdot \rangle$ is the mean operator, $\mathcal{R}_{i,b}$ and \mathcal{R}_j are the residuals for boundary conditions and PDE at points i and j , respectively. α_j and α_i are RBA weights that balance the local contribution within each loss term [54].

Parameter-based analysis. We define suitable architectures to approximately match the number of parameters between all models. PINN has two hidden layers with 16 neurons, cPIKAN (i.e., physics-informed Chebyshev KAN) has two hidden layers with eight neurons and $k = 5$,

and PIKAN (i.e., physics-informed KAN) has a single hidden layer with ten neurons and $k = g = 5$. Additionally, as described in [11], we explore the PIKAN multi-grid approach; for this case, we set $k = 3$, initialize $g = 5$, and divide the training process into three stages, duplicating the number of grid points every 600 iterations. We train our models by minimizing equation (16) for 1800 LBFGS iterations on a sample space of 51×51 collocation points. Following [11]; we set $w_{bc} = 1$ and $w_{pde} = 0.01$, which induces a biased loss function that downscale the PDE contribution. This loss function enables us to train models with few parameters in a few numbers of iterations using second-order optimizes directly. We initialize our RBA weights to one (i.e., $\alpha_i = \alpha_j = 1$) and update them as described in equation (3) with $\eta^* = 1e - 4$.

	Method	N. Params	Optimizer	Iterations	Relative L^2	Time(ms/it)
a	PINN	304	LBFGS	1.8e3	1.03%	64
	PIKAN	300	LBFGS	1.8e3	0.735%	4550
	PIKAN(Multigrid)	240-690	LBFGS	1.8e3	0.476%	3243
	cPIKAN	350	LBFGS	1.8e3	0.530%	183
	PINN+RBA	304	LBFGS	1.8e3	0.354%	108
	cPIKAN+RBA	350	LBFGS	1.8e3	0.376%	243
b	PINN	30300	Adam	2.0e5	0.530%	5.1
	cPIKAN	15840	Adam	2.0e5	0.500%	6.7
	PINN+RBA	30300	Adam	2.0e5	0.206%	7.1
	cPIKAN+RBA	15840	Adam	2.0e5	0.160%	7.4
c	PINN	82304	Adam	5.0e5	4.30%	10.3
	cPIKAN	20960	Adam	5.0e5	N/A	8.0
	PINN+RBA	82304	Adam	5.0e5	1.72%	11.4
	cPIKAN+RBA	20960	Adam	5.0e5	0.381%	8.4

Table 3: Relative L^2 and computational time (ms/it) comparison between different models and training strategies.(a) Parameter-based analysis for solving Helmholtz Equation ($a_1 = 1, a_2 = 4$) using LBFGS optimizer and a biased loss function that downscale the PDE contribution. (b) Computation time-based comparison for solving Helmholtz Equation ($a_1 = 1, a_2 = 4$) using ADAM optimizer with an unbiased loss function. (c) Complexity-based analysis using ADAM optimizer with no global weights for solving the Helmholtz equation with a higher wave number ($a_1 = a_2 = 6$). For the cPIKAN model, N/A represents "not applicable" since loss became undefined after the initial iterations. Time per iteration is measured on Nvidia's GeForce RTX-3090 GPU.

We evaluate the model performance based on the Relative L^2 and training time measured in milliseconds per iteration (ms/it). The cPIKAN with RBA (cPIKAN+RBA) achieves a relative L^2 error of 0.354%, and its prediction and corresponding point-wise error are shown in Figure 9 (a). The results for the remaining methods are detailed in Table 3(a), and Figure 10(a) shows their corresponding Relative L^2 convergence. Since PIKAN does not benefit from GPU parallelization, it is significantly slower than the other models; however, its performance is better than vanilla PINN. The multigrid PIKAN is faster (i.e., average of three stages) than PIKAN and outperforms cPIKAN. However, it is essential to notice that in the last stage, the number of parameters is twice as many as in the other models. For this example, cPIKAN outperforms PINN and vanilla PIKAN, and the best-performing model is PINN+RBA. However, notice that cPIKAN+RBA's final relative L^2 error is comparable.

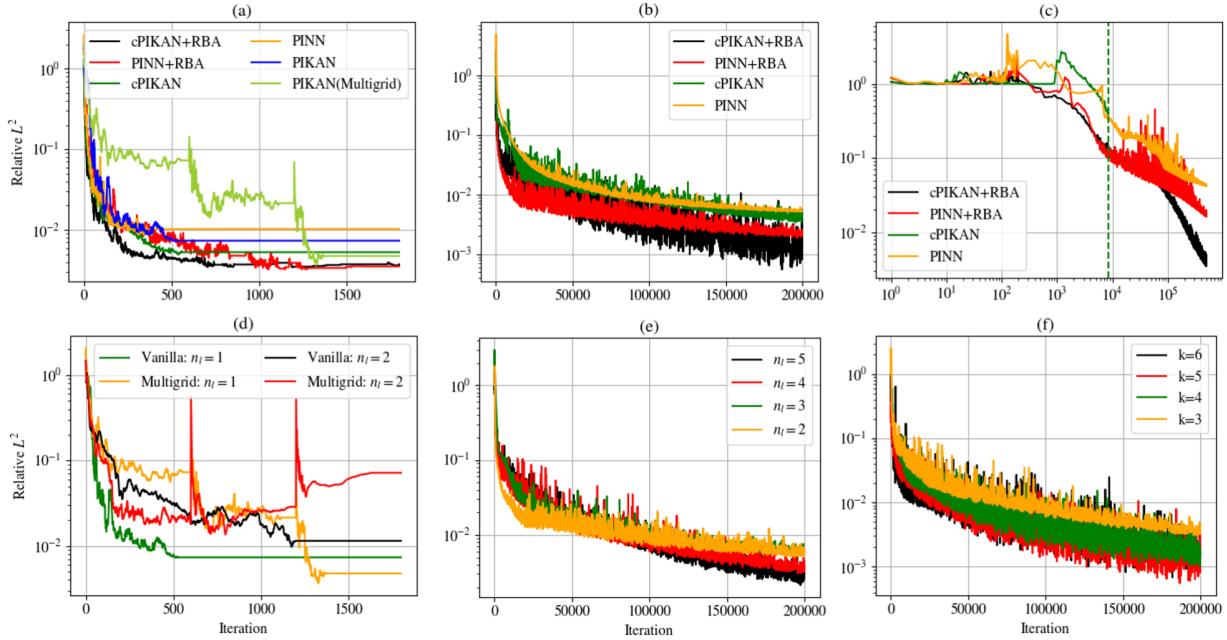


Figure 10: Relative L^2 convergence history. (a) Parameter-based analysis based in the original study [11] using LBFGS and global weights to downscale the PDE residuals for solving the Helmholtz equation ($a_1 = 1, a_2 = 4$). (b) Time-based analysis using ADAM optimizer with no global weights for solving the Helmholtz equation ($a_1 = 1, a_2 = 4$). (c) Complexity analysis using ADAM optimizer with no global weights for solving the Helmholtz equation ($a_1 = 6, a_2 = 6$). The green line represents the iteration where cPIKAN becomes undefined and cannot be trained further. (d) Vanilla PIKAN and multigrid PIKAN sensitivity analysis for a different number of layers for solving the Helmholtz equation ($a_1 = 1, a_2 = 4$) (e)cPIKAN+RBA ($k = 3$) sensitivity analysis for a different number of layers for solving the Helmholtz equation ($a_1 = 1, a_2 = 4$). We report the number of layers up to ($n = 5$) since deeper networks did not converge. (f) cPIKAN+RBA sensitivity analysis for different Chebyshev polynomial orders for solving the Helmholtz equation ($a_1 = 1, a_2 = 4$). We present the results up to degree ($k = 6$) since higher orders did not converge.

Computation time-based analysis. In this section, we analyze the PINN and cPIKAN model for deeper networks (i.e., four hidden layers) and a higher number of collocation points (i.e., 100×100). We define the number of neurons per layer by roughly matching the PINN and cPIKAN’s computational time. In particular, we use 100 and 32 neurons per hidden layer for PINN and cPIKAN, respectively. We train our models using $w_{bc} = w_{pde} = 1$, which induces an unbiased loss function akin to real-world applications. To balance the contribution of each loss term, we use RBA only on the PDE residuals, initializing them to zero (i.e., $\alpha_j = 0$) and updating them interactively with $\eta^* = 1e - 3$ as described in equation (3). Following this approach, the RBAs work as global and local weights that modify the contribution of each training point iteratively by following the network residuals. We train our models for $2.0e5$ iterations using Adam optimizer [67] with a learning rate scheduler that starts in $1e - 3$ and ends in $1e - 4$.

As shown in Table 3(b), cPIKAN marginally outperforms PINN with and without RBA. Additionally, Figure 10(b) shows that combining our based models with RBA accelerates their relative L^2 convergence. For this example, the best-performing model is cPIKAN+RBA,

which achieves a relative L^2 error of 0.160%.

Complexity-based analysis. To increase the problem complexity, we solve the Helmholtz equation with a higher wave number (i.e., $a_1 = a_2 = 6$). This modification induces steeper gradients in the PDE residuals, making it difficult for the neural network to approximate. For PINN, we use six hidden layers with 128 neurons per layer, while for cPIKAN, we use five layers, 32 neurons, and $k = 5$. As in the previous case, we train our model with an unbiased loss function ($w_{bc} = w_{pde} = 1$) and apply RBA (initiated at zero) only in the residuals using $\eta^* = 1e - 3$. We update our network parameters using Adam optimizer for 5e5 iterations with a learning rate schedule from $1e - 3$ to $5e - 5$.

The best-performing model reconstruction and its corresponding point-wise error are illustrated in Figure 9 (b). Table 3(c) shows that cPIKAN+RBA significantly outperforms the other methods, achieving a relative L^2 error of 0.381%. However, notice that the vanilla cPIKAN did not converge (See Figure 10(c)).

3.3.1. Sensitivity Analysis

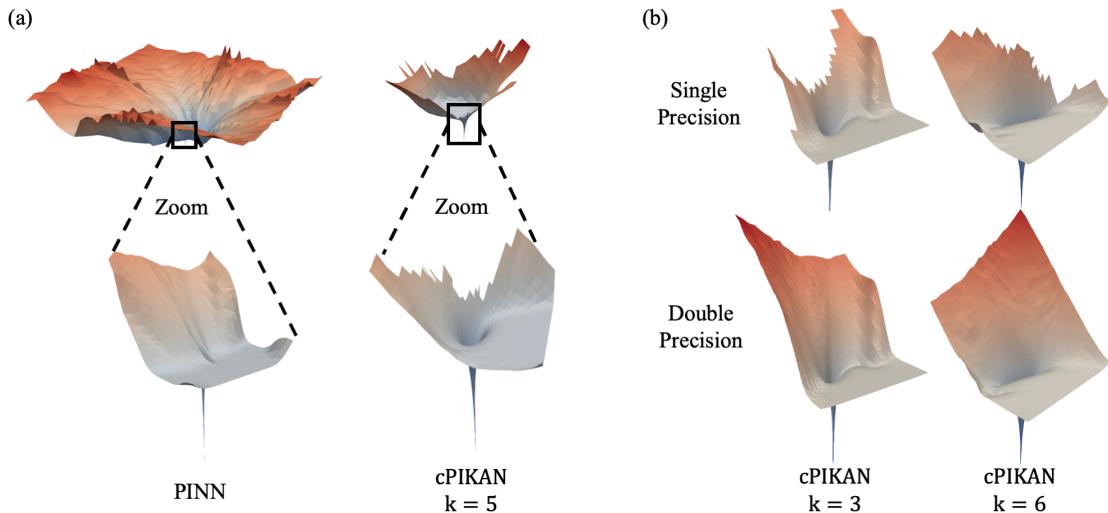


Figure 11: Loss Landscapes.(a)Comparison between PINN and PIKAN ($k = 5$) for solving Helmholtz Equation ($a_1 = 6, a_2 = 6$). Notice that PIKAN’s loss is not defined (empty spaces) for parameters far from the minimum, making it highly sensitive to initialization. (b) Comparison between PINN and PIKAN for different Chebyshev polynomial orders for solving Helmholtz Equation ($a_1 = 1, a_2 = 4$). Notice that the non-defined regions disappear when using double precision, which indicates that the problem lies within the limitations of training a model using a single precision.

As described in the previous sections, cPIKAN performs better than PINNs and significantly reduces the computational overhead of PIKAN. However, it induces more oscillations (See Figure 10(b))and can potentially become unstable (See Figure 10(c)). To explore this behavior, we perform a sensitivity analysis for Helmholtz and study the influence of the number of hidden layers n_l and polynomial order k .

First, we study the effect of n_l on PIKAN ($k = 5$) and PIKAN multigrid, fixing the polynomial order to $k = 5$ and $k = 3$ respectively. Notice that, in this case, increasing

the number of layers hinders the model performance of both models. Then, we analyze the effects of k (Figure 10(e)) and n_l (Figure 10(e)) on cPIKANs. For this case, increasing k or n_l improves cPIKAN's performance; nevertheless, increasing these parameters makes the model unstable. In this example, using $k > 6$ or $n_l > 5$ causes the model's loss function to become undefined after training it for several iterations. To explore this issue, we follow[7, 33] and plot the loss landscape for different values of k . An ideal landscape is smooth, continuous, and convex, which enables the optimizer to converge successfully to the global minimum. However, Figure 11(a) shows that the cPIKAN landscape has empty holes near the edges, which indicate sections where the loss is not defined. Moreover, these regions take over the whole space far from the minimum, suggesting that cPIKAN models are sensitive to initialization. To further analyze this behavior, we train and plot the loss landscape using single and double pressure. Figure 11(b) shows that, for single precision, the non-defined regions grow as we increase k , suggesting that the model becomes unstable for higher polynomial orders. However, it can be observed that these models can be trained, and their loss is defined when using double precision (i.e., float 64), indicating that the instability is related to the numerical approximation inherent in training a model with single precision (i.e., float 32). This indicates that it is possible to train models with higher k or n_l using double precision, yet this type of training increases their computational cost.

3.4. Navier-Stokes equation

In this section, we consider solving the following 2D steady Navier-Stokes equations with PINNs and PIKANs,

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \quad (19)$$

$$u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} = - \frac{\partial p}{\partial x} + \nu \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad (20)$$

$$u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} = - \frac{\partial p}{\partial y} + \nu \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right), \quad (21)$$

where, (u, v) are the velocity component in x, y direction, respectively, p is the pressure, ν is the kinematic viscosity. In particular, when the ν is small, it is recommended by [68] to reformulate equations (19)-(21) into to the following loss functions:

$$e_1 = u \frac{\partial u}{\partial x} + v \frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - (\nu + \nu_E) \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad (22)$$

$$e_2 = u \frac{\partial v}{\partial x} + v \frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - (\nu + \nu_E) \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right), \quad (23)$$

$$e_3 = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}, \quad (24)$$

where ν_E is the artificial viscosity that will be determined during training. Note that ν_E is a scalar, whose construction is adapted from the entropy viscosity method (EVM) [69, 70] for

numerical stabilization in the flow simulation at a high Re . Specifically, ν_E can be computed from:

$$\nu_E = \min(\beta_E \nu, \alpha_E \frac{|r|L^2}{U_\infty^2}), \quad (25)$$

where r is the *predicted entropy residual* and is one of the output variable of the network, In practice, r can be inferred by following equation loss,

$$e_4 = (u - u_m)e_1 + (v - v_m)e_2 - r. \quad (26)$$

Here u_m and v_m are two constants that makes r be non-zero on the boundaries. In all the simulations cases of this section, $u_m = 0.5$, $v_m = 0.5$ are employed unless otherwise stated. We would like to emphasize that non-zero entropy viscosity on the wall boundary is important to improve the accuracy, which is different from the entropy viscosity in the numerical method developed in [70]. We found that $\nu_E > 0$ in the vicinity of the boundary can result into better prediction in both PINNs and PIKANs. In equation (25), $L = 1$, $U_\infty = 1$ are the characteristic length and velocity, respectively. Moreover, α_E and β_E are two tunable hyperparameters whose values may affect the inference accuracy greatly; α_E and β_E can be either constant or descending throughout the training. Nonetheless, in this section, $\alpha_E = 0.03$, $\beta_E = 5$ are used.

Re	Method	polynomial order	Relative L^2 error (u, v, p) %	Time (ms/it)	Num. of parameters
400	PINN	-	0.25, 0.37, 2.25	32	44, 283
	Chebyshev	3	1.13, 1.38, 2.34	61	4, 736
	PIKAN	5	0.20, 0.26, 1.63	81	7, 104
		8	0.18, 0.21, 1.8	112	10, 656
	Legendre	3	2.0, 2.3, 3.2	82	4, 736
	PIKAN	8	0.21, 0.29, 1.77	241	10, 656
	Jacobi	3	0.31, 0.43, 1.96	120	4, 736
	PIKAN	8	0.18, 0.26, 1.67	235	10, 656
	Hermite	3	4.1, 4.9, 9.7	84	10, 656
	PIKAN	8	143, 101, 151	220	10, 656
2000	PINN+RBA	-	0.24, 0.33, 1.83	34	44, 283
	Chebyshev PIKAN+RBA	3	0.18, 0.20, 1.78	92	4, 736
	PINN	-	18.9, 18.8, 22.8	63	44, 283
	Chebyshev	5	109.2, 113.8, 132.05	85	7, 104
	PIKAN	8	105.2, 107.8, 110.23	127	10, 656
2000	PINN+EVM	-	6.4, 5.4, 6.9	64	49, 365
	Chebyshev PIKAN+EVM	8	4.2, 4.5, 8.4	135	20, 736

Table 4: Comparison on performance between PINNs and PIKANs based on different polynomials in solving 2D steady cavity flow at $Re = 400$ and $Re = 2000$. The PINNs, PIKANs, RBA and EVM are implemented on our NSFnet [71]. 10^4 training points, 9×10^5 training epochs for the case of $Re = 400$, 2×10^4 training points, 4×10^5 training epochs for the case of $Re = 2000$. The Adams optimizer [67] is used and the training is performed on an Nvidia’s RTX 4090 GPU. Note that the coefficients of the Jacobi polynomial $P_n^{\alpha, \beta}(x)$ used in this section are: $\alpha = 1$, $\beta = 1$.

We have systematically performed PINNs/PIKANs simulation of 2D steady lid-driven cavity flow at $Re = 400$ and $Re = 2000$. The steady cavity flow is the well-known benchmark case and is frequently used in the validation of numerical method. The details of the computational domain and boundary condition can be found in [33]. In particular, we have compared the Chebyshev, Jacobi, Legendre and Hermite polynomials based PIKANs for the cavity flow at $Re = 400$ and $Re = 2000$. It has been reported that vanilla PINNs can accurately infer the cavity flow at $Re = 400$, but failed to obtain correct solution at $Re = 2000$.

[33]. In order to provide a fair comparison among PINNs and PIKANs, we have kept the same number of residual points, number of training epochs and the optimizer, i.e., the only difference between different cases lies in the network, e.g., in PINN, the multi-layer Perceptron (MLP) is used, while in Chebyshev PIKAN (cPIKAN), the Chebyshev polynomials based Kolmogorov Arnold Network (KAN) is employed.

As shown in Table 4, in total 18 cases have been tested. At $Re = 400$, it could be observed that PIKANs can generate solutions with a comparable accuracy to PINNs. However, the number of trainable parameters used in PIKANs is far less than the one used in PINNs, although they can achieve the same accuracy. On the other hand, the training time for each iteration of PIKANs is four times more than the counterpart in PINNs. Among different variants of PIKANs that are differentiated by the type of polynomials, cPIKAN is the most promising architecture, in terms of the inference accuracy and GPU time of training. Moreover, with RBA, which is the technique that can dynamically and locally adjust the weights on loss function during training, both PINNs and PIKANs can further achieve better inference accuracy at the same computation time.

From aforementioned results of $Re = 400$, it could be concluded that cPIKAN is the best choice for the 2D steady cavity flow. Therefore, in the PIKAN simulation of cavity flow at $Re = 2000$, only the cPIKAN is tested. However, as shown in the bottom 5 rows of Table 4, after training 40,000 epochs, the vanilla PINN manages to achieve the l_2 -relative error lower than 20%, while the vanilla cPIKAN produces solutions of error higher than 100%. However, with the help of EVM, both PINN and cPIKAN can significantly improve the inference accuracy, with the relative error lower than 7%, after the same number of epochs as the vanilla PINN and cPIKAN.

The histories of relative error varying with training epochs of PINN/cPIKANs are plotted in Figure 12. As shown in the left panel of Figure 12, it could be observed that the RBA can speed up the both the PINN and cPIKAN training. From the right panel of Figure 12, it can be seen that the relative error of the vanilla cPIKAN (green line) barely decays with training. However, with EVM, both PINN and cPIKAN can reduce the inference error notably, although the error history of cPIKAN exhibits more oscillations.

The inferred streamlines from the trained PINN/cPIKAN at final training stage are shown in Figure 13. It could be seen that at $Re = 400$, both PINN and cPIKAN successfully reproduce the small eddies at left bottom and right bottom corners. The result of cPIKAN is slightly better than the counterpart of PINN, since the streamlines on the right bottom corners are in closed circles, while streamlines generated by PINN penetrate into the wall. At $Re = 2000$, the streamlines generated by the cPIKAN are totally different from the reference solution, which indicates that the cPIKAN might stuck at a local minimum from the very beginning of the training.

In summary, we found that PIKANs based on the Jacobi type of polynomials can generate comparable accurate solution to that of PINN, while the cost of training can be several times more. PIKANs can suffer from unstable training for the flow at high Re , but with the help of EVM or RBA, PIKANs can return to the correct training trajectory and generate accurate solution.

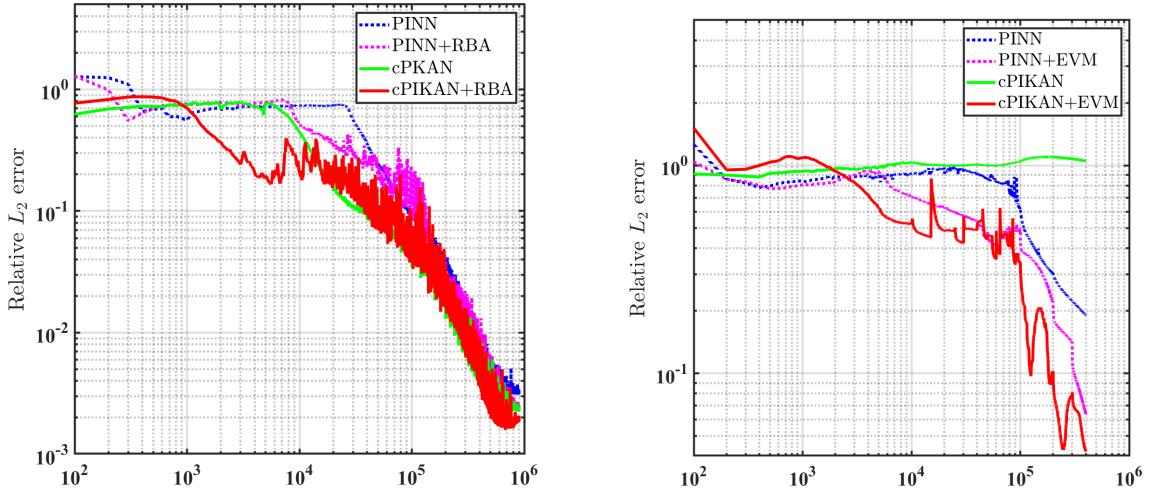


Figure 12: Comparison of the error between PINN and PIKAN in solving the steady cavity flow at $Re = 400$ and $Re = 2000$. Note that the error is computed on a 256×256 uniform mesh, which is different from the residual points used in training.

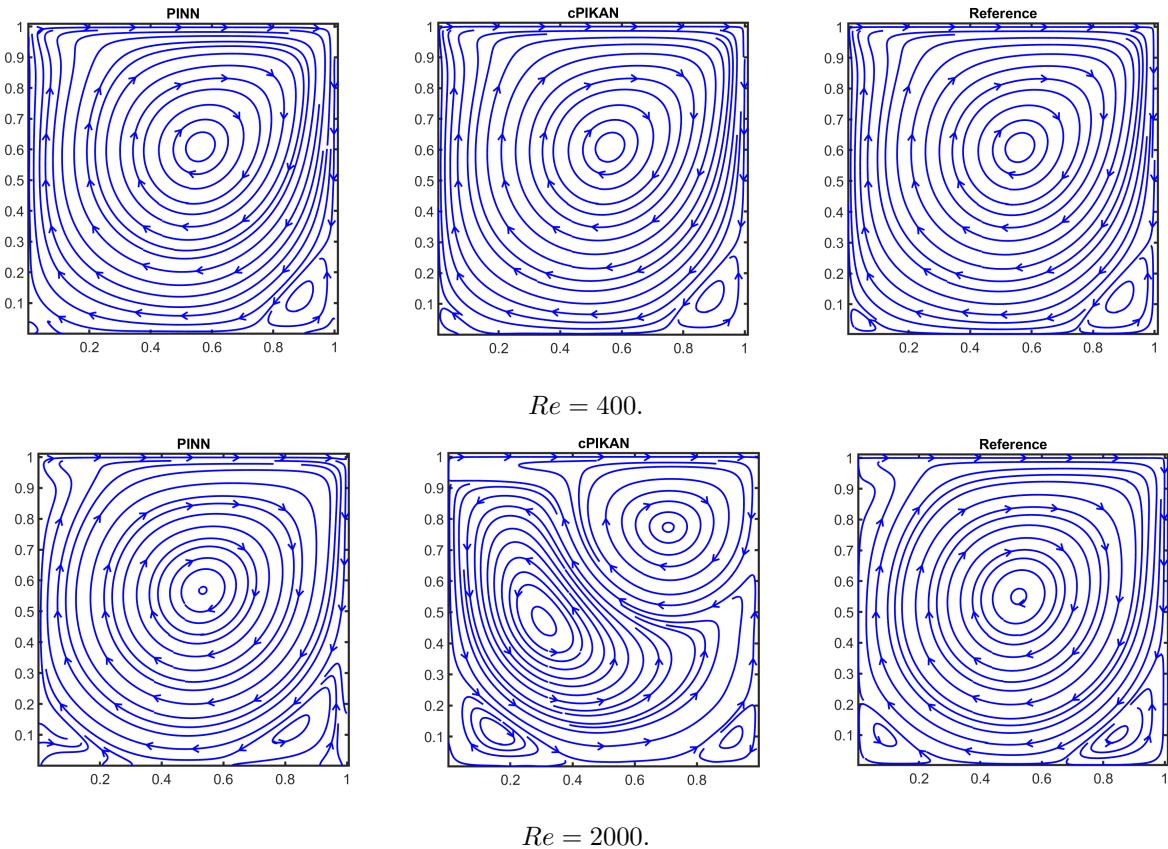


Figure 13: Streamlines of the cavity flow, inferred by the vanila PINN, cPIKAN, as well as the reference solution.

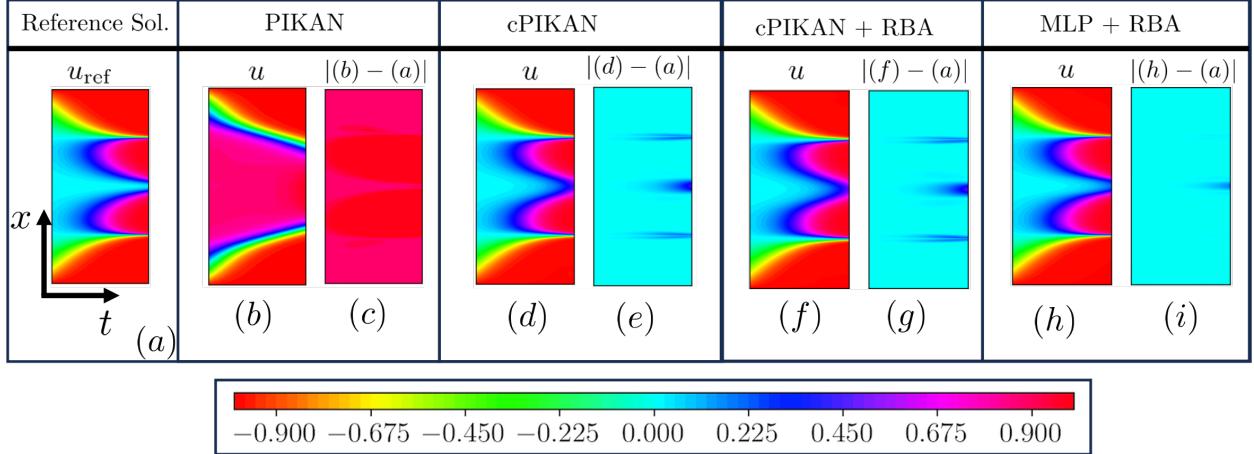


Figure 14: The solution of the Allen-Cahn equation (27) in the spatio-temporal domain of $(x \in [-1, 1], t \in [0, 1])$ is depicted, with the reference solution shown in subfigure (a). The solutions obtained from (b) PIKAN, (d) cPIKAN, (f) cPIKAN+RBA, and (h) MLP architectures are also displayed. Subfigures (c), (e), (g), and (i) represent the absolute pointwise error between the reference solution and the solutions obtained from PIKAN, cPIKAN, cPIKAN with RBA, and MLP with RBA, respectively.

3.5. Allen-Cahn equation

In this example, we investigate the efficacy of PIKAN, cPIKAN, cPIKAN with RBA, and PINN with RBA for solving the 2D (1 + 1D) nonlinear Allen-Cahn equation [72]. The Allen-Cahn equation is expressed as follows,

$$\frac{\partial u}{\partial t} - D \frac{\partial^2 u}{\partial x^2} + 5(u^3 - u) = 0, \quad (27)$$

where $D = 1 \times 10^{-4}$ and $t \in [0, 1]$, $x \in [-1, 1]$ and with following initial and boundary conditions

$$\begin{aligned} u(x, 0) &= x^2 \cos(\pi x) \\ u(-1, t) &= u(1, t) = -1. \end{aligned}$$

The solution of (27) using PIKAN, cPIKAN, cPIKAN with RBA, and PINN with RBA is obtained by minimizing the following loss function

$$\mathcal{L} = w_{ic}\mathcal{L}_{ic} + w_{bc}\mathcal{L}_{bc} + w_{pde}\mathcal{L}_{pde}, \quad (28)$$

where w_{ic} , w_{bc} and w_{pde} are weights that balances the contribution of the averaged loss terms for initial conditions (\mathcal{L}_{ic}), boundary conditions (\mathcal{L}_{bc}) and PDE residuals (\mathcal{L}_{pde}) which are described as follows,

$$\begin{aligned} \mathcal{L}_{ic} &= \left(\sum_{l=1}^{N_i} |\mathcal{R}_{l,ic}| \right)^2 \\ \mathcal{L}_{bc} &= \left(\sum_{b=1}^2 \sum_{i=1}^{N_b} |\mathcal{R}_{i,b}| \right)^2, \\ \mathcal{L}_{pde} &= \langle (\alpha_j \cdot |\mathcal{R}_j|)^2 \rangle_j, \end{aligned} \quad (29)$$

where, $\langle \cdot \rangle$ is the mean operator, $\mathcal{R}_{i,b}$, $\mathcal{R}_{l,ic}$ and \mathcal{R}_j are the residuals for boundary conditions, initial conditions and PDE at points l , i and j , respectively. α_j are the RBA weights.

The solutions of (27) using PIKAN, cPIKAN, cPIKAN with RBA, and PINN with RBA are shown in Figure 14. The parameters used for training these networks are detailed in Table 5. The results in Figure 14 were obtained using the Adam optimizer with a learning rate of 5×10^{-4} . The training was performed as a single batch training till 150,000 iterations. In Figure 14(a), we display the reference solution of (27) computed using the spectral element method [73]. Figure 14(b) presents the solution obtained using PIKAN, while Figure 14(c) shows the absolute pointwise error between the reference and PIKAN solutions. The solution obtained from the PIKAN method did not converge to the reference solution, as relative l_2 -error between PIKAN and reference solutions is 58.39%. In Figure 14(d) and (f), we show the solutions of (27) obtained from cPIKAN and cPIKAN enhanced with RBA, respectively. The absolute pointwise errors in the solutions obtained from cPIKAN and cPIKAN with RBA are shown in Figure 14(e) and (g), respectively. The relative l_2 -errors between the reference solution and those from cPIKAN and cPIKAN with RBA are 5.15% and 5.65%, respectively, indicating almost similar level of accuracy among them. In Figure 14(h) and (i), we show the solution of (27) obtained using PINN (MLP architecture) with RBA and the absolute pointwise error, respectively. The relative l_2 -error between the PINN and the reference solutions is 1.51%. In Figure 15, we show the convergence of all the methods by plotting the loss function Equation 29 against the iterations. It is evident from Figure 15 that the MLP-based architecture, enhanced with RBA, exhibits faster convergence compared to the other methods. A detailed description of parameters, errors, and efficiency (in terms of runtime) is provided in Table 5. The runtime measurements in Table 5 were taken on an Nvidia’s GeForce RTX-3090 GPU. In Table 5 it is noted that runtime for cPIKAN and cPIKAN with RBA (Row 2 and 3) is almost similar despite having 50000 additional RBA parameters. This is caused by latency while moving the data from the DRAM to the processor. As the model and data are very small, the volatile GPU utility does not exceed more than 15%. Therefore, the runtime for cPIKAN and cPIKAN with RBA is dominated by latency.

Method	$[N_i, N_b, N_f]$	No. of parameters	Relative L^2 error	Time: (ms/it)
PINN with RBA	[200, 100, 50000]	50,049	1.51 %	22.93 ms
cPIKAN	[200, 100, 50000]	6,720	5.15 %	39.31 ms
cPIKAN with RBA	[200, 100, 50000]	56,720*	5.65 %	39.28 ms
PIKAN	[200, 100, 50000]	6721	58.39 %	2633.12 ms

Table 5: Details of (hyper) parameters used for solving Allen-Cahn equation using PINN with RBA, cPIKAN, cPIKAN with RBA and PIKAN based architecture. Here N_i , N_b and N_f represents the number of spatio-temporal points used for computing initial, boundary and residual value of u during the training the networks. Time per iteration is measured on Nvidia’s GeForce RTX-3090 GPU. * : 56720 parameters include 6720 model parameter and 50,000 trainable RBA weights.

3.6. Reaction-diffusion equation

In this example, we extend PI-KAN to address noisy data in solving differential equations with uncertainty quantification (UQ) [41, 44, 58–60, 74–76]. In particular, we equip cPIKAN with the Bayesian framework and estimate the posterior distribution of its parameters given

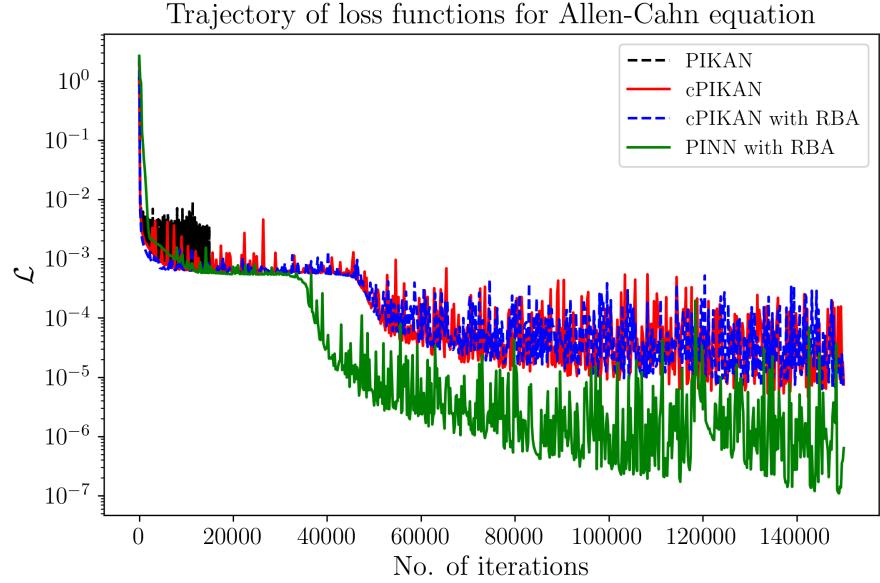


Figure 15: Loss functions showing the convergence of PIKAN, cPIKAN, cPIKAN with RBA and PINN with RBA while computing the solution of Allen-Cahn equation (27). It is to be noted that convergence PINN (MLP architecture) with RBA happened to be faster than any other method.

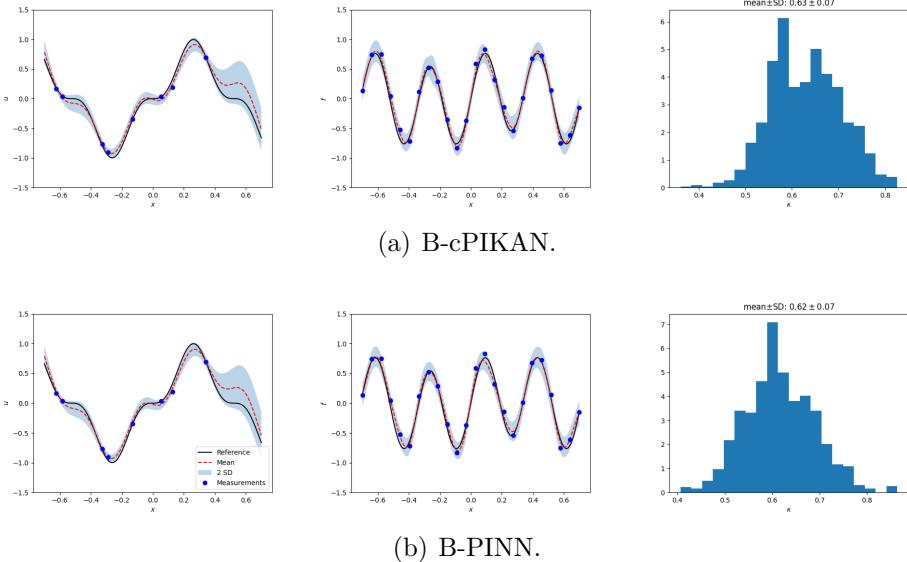


Figure 16: Results from B-cPIKAN and B-PINN for the inverse problem on the 1D steady-state reaction-diffusion equation with noisy data of u and f : from left to right are inferences of u , f , and κ . Here, the epistemic uncertainty of the network is quantified. The B-cPIKAN and B-PINN methods perform similarly in inferring u and f . The B-cPIKAN method is able to provide slightly better inference over κ (the exact is 0.7): the error is lower and the uncertainty bounds the error.

data, i.e. $p(\theta|\mathcal{D})$, using Hamiltonian Monte Carlo (HMC) method [77] (see [41, 59] for more details regarding UQ in SciML). We refer to the Bayesian cPIKAN as B-cPIKAN and demonstrate its capability by solving the following 1D steady reaction-diffusion equation

Methods	Rel. l_2 -error of u	Rel. absolute error of κ	No. of parameters	Time
B-cPIKAN	22.95%	10.04%	480	139s
B-PINN	22.67%	11.90%	481	39s

Table 6: Results from B-cPIKAN and B-PINN for the inverse problem on the 1D steady-state reaction-diffusion equation with noisy data of u and f . The architecture of B-cPIKAN is [1, 10, 10, 1] with degree three for the Chebyshev polynomials while the architecture of B-PINN is [1, 20, 20, 1] with hyperbolic tangent activation function. HMC with the same hyperparameter is employed to sample from both posterior distributions on a standard laptop CPU (13th Gen Intel(R) Core(TM) i9-13900HX with 2.20 GHz processor).

with noisy data:

$$D\partial_{xx}u + \kappa \tanh(u) = f, \quad (30)$$

where f is the source term, $D = 0.01$ denotes the diffusion rate, and $\kappa = 0.7$ the reaction rate which is assumed to unknown. The target is to infer u and κ with uncertainty given noisy data of u and f , i.e. an inverse problem. Specifically, we choose $u(x) = \sin^3(6x)$, $x \in [-0.7, 0.7]$ to be the exact and derive $f(x)$ analytically by plugging $u(x) = \sin^3(6x)$ into (30). Eight measurements of u are randomly sampled from $[-0.7, 0.7]$, following the uniform distribution, and corrupted by additive Gaussian noise with mean zero and standard deviation 0.05, and 24 measurements of f are uniformly sampled and corrupted by additive Gaussian noise with mean zero and standard deviation 0.1.

We employ both the B-cPIKAN and B-PINN method to solve the inverse problem. For B-cPIKAN, the architecture of the network is [1, 10, 10, 1] with degree three for the Chebyshev polynomials, while for B-PINN, the architecture is [1, 20, 20, 1], such that the number of parameters of these two networks is approximately the same. The prior of the B-cPIKAN is chosen to be independent Gaussian with mean zero and standard deviation 0.5, and the prior of the B-PINN is independent Gaussian with mean zero and standard deviation 1. We employ adaptive HMC with initial step size 0.01 and 50 leapfrog steps to sample from both posterior distributions. We set the number of burn-in samples to be 2,000 and the number of posterior samples to be 1,000. An open-source Python library NeuralUQ [59] is utilized for fast and reliable implementation. We note that here we only quantify the epistemic uncertainty of the network. Results are presented in Figure 16 and Table 6, from which we can see the B-cPIKAN and B-PINN perform similarly: the predicted uncertainties are able to bound the errors between the predicted means and the exact. In particular, the predicted uncertainties of u from both methods grow near $x = 0.7$ due to lack of measurements. The B-cPIKAN method is able to provide slightly better inference over κ with higher computational cost: the error is smaller and the uncertainty is able to bound the error.

3.7. 1D Burgers' equation

We consider the 1D Burgers' equation with periodic boundary conditions:

$$\partial_t u + u \partial_x u = \nu \partial_{xx} u, x \in [0, 1], t \in [0, 1]. \quad (31)$$

where $\nu = \frac{0.01}{\pi}$ denotes the viscosity. In this example, we learn the surrogate operator for solution of equation 31, which maps an arbitrary initial condition sampled from a distribution, denoted as u_0 , to the solution of equation 31 at $t = 1$:

$$G : u_0(x) \mapsto u(x, t = 1).$$

Methods	Rel. l_2 -error	No. of parameters	Time: ms/iter
DeepONet	$5.83\% \pm 0.19\%$	63900	1.9
DeepOKAN 1	$2.71\% \pm 0.08\%$	252800	3.9
DeepOKAN 2	$3.02\% \pm 0.13\%$	76400	3.9

Table 7: Rel. l_2 -error for learning the solution operator of the 1D Burgers' equation with viscosity $\nu = \frac{1}{100\pi}$. The architectures of the branch and trunk nets are $[128, 100, 100, 100, 100]$ and $[4, 100, 100, 100]$, respectively, for DeepONet and DeepOKAN 1, and are $[128, 50, 50, 50, 50]$ and $[4, 50, 50, 50]$ for DeepOKAN 2. Here Chebyshev KAN [26] is employed with degree three for DeepOKANs and hyperbolic tangent is used as the activation function for the DeepONet. The time per iteration is measured on an Nvidia's GeForce RTX-3090 GPU with 24 GB of memory.

Methods	1% noise	5% noise	10% noise
DeepONet	$5.83\% \pm 0.19\%$	$5.93\% \pm 0.18\%$	$6.29\% \pm 0.19\%$
DeepOKAN 1	$2.72\% \pm 0.08\%$	$2.94\% \pm 0.09\%$	$3.57\% \pm 0.05\%$
DeepOKAN 2	$3.02\% \pm 0.13\%$	$3.24\% \pm 0.12\%$	$3.91\% \pm 0.13\%$

Table 8: Rel. l_2 -error for learning the solution operator of the 1D Burgers' equation when trained with clean data but tested with noisy input data. Here the noise is additive Gaussian noise with mean zero and different levels of standard deviation, and the noise level is defined as the percentage of the absolute value of the input function evaluated at the grid.

The training and testing data are generated following [51] where the initial condition is sampled from a Gaussian process defined as $\mu = N(0, 49^2(-\Delta + 49I)^{-2.5})$ with embedded periodic boundary condition. A spatial resolution with 128 uniform grids is used to resolve the input and output functions. We apply four Fourier basis

$$\{\cos(2\pi x), \sin(2\pi x), \cos(4\pi x), \sin(4\pi x)\}$$

to the input of the trunk net and data normalization to the output of DeepONets to improve the performance [55]. 1,000 and 200 functions of u_0 and $u(\cdot, t = 1)$ are used for training and testing, respectively, and we normalize the output of the operator network based on the mean and standard deviation of the training data.

We employ one DeepONet and two DeepOKANs to learn the solution operator G . Specifically, the architecture of the DeepONet is $[128, 100, 100, 100, 100]$ for the branch net and $[4, 100, 100, 100]$ for the trunk net, both of which are equipped with tanh activation function, while the architectures of DeepOKANs (DeepOKAN 1/2) are $[128, 100, 100, 100, 100]/[128, 50, 50, 50, 50]$ for the branch net and $[4, 100, 100, 100]/[4, 50, 50, 50]$ for the trunk net. Both DeepOKANs are based on the Chebyshev KAN [26] and have degree three for the Chebyshev polynomials. The Adam optimizer [67] is employed for all operator networks. The learning rate for the DeepONet is 1e-3 for 100k iterations and 1e-4 for another 100k iterations, while for both DeepOKANs it is 1e-4 for 100k iterations and 1e-5 for another 100k iterations. To avoid overfitting, we apply a l_2 regularizer [78] with weighting coefficient 1e-5 to both operator networks. The errors are presented in Table 7. We observe that DeepOKANs perform significantly better than the DeepONet at higher computational cost.

We further test the robustness of these operator networks against noisy input functions. Specifically, networks are trained with clean data while tested with noisy data. Here we

consider Gaussian noise with mean zero added to the value of $u_0(x_i), i = 1, \dots, N_v$ where x_i are the uniform grids on which the input and output functions are resolved. The standard deviation of the noise is proportional to the absolute value of $u_0(x_i)$. We test three noise levels with 1%, 5% and 10% and present results in Table 8. We observe that DeepOKANs are more robust to noisy input functions compared to the DeepONet.

3.8. 120-dimensional Darcy problem

Methods	L^2 relative error	No. of parameters	Time: ms/iter
DeepONet	$1.62\% \pm 0.15\%$	147000	2.3
DeepOKAN	$2.18\% \pm 0.02\%$	585200	8.8

Table 9: Rel. l_2 -error for learning the solution operator of the Darcy problem. The architectures of the branch and trunk nets are [961, 100, 100, 100, 100] and [2, 100, 100, 100], respectively, for both operator networks. Chebyshev KAN [26] is employed with degree three for the DeepOKAN and hyperbolic tangent is used as the activation function for the DeepONet. The time per iteration is measured on an Nvidia’s GeForce RTX-3090 GPU with 24 GB of memory.

Methods	1% noise	5% noise	10% noise
DeepONet	$1.66\% \pm 0.15\%$	$2.25\% \pm 0.12\%$	$3.47\% \pm 0.12\%$
DeepOKAN	$2.18\% \pm 0.02\%$	$2.20\% \pm 0.02\%$	$2.30\% \pm 0.03\%$

Table 10: Rel. l_2 -error for learning the solution operator of the Darcy problem when trained with clean data but tested with noisy input data. The noise is additive Gaussian noise with mean zero and different levels of standard deviation, and the noise level is defined as the percentage of the absolute value of the input function evaluated at the grid.

In this section, we consider a 2D steady-state flow through porous media described by steady-state Darcy’s law expressed as

$$\nabla \cdot (\lambda(x, y) \nabla u(x, y)) = f, x, y \in (0, 1), \quad (32)$$

where the source term $f = -30$. Here λ denotes the hydraulic conductivity field and u the hydraulic head. The boundary condition is specified as follows:

$$u(0, y) = 1, u(1, y) = 0, \partial_{\mathbf{n}} u(x, 0) = \partial_{\mathbf{n}} u(x, 1) = 0. \quad (33)$$

We learn the solution operator which maps $\log(\lambda)$ to u :

$$G : \log(\lambda)(x, y) \mapsto u(x, y),$$

using dataset from [41, 59, 60], in which the logarithm of the conductivity is sampled from a truncated Karhunen-Loève expansion of a Gaussian process with zero mean and the following kernel:

$$k(x, x', y, y') = \exp\left(-\frac{(x - x')^2}{2l^2} - \frac{(y - y')^2}{2l^2}\right), x, x', y, y' \in [0, 1],$$

where $l = 0.25$ denotes the correlation lengths. We use a 31×31 uniform grid to represent $\log(\lambda)$ and u , and the first 120 leading terms of the expansion is kept. To perform training

and testing, 10,000 and 1,000 functions of $\log(\lambda)$ and u are used, respectively. We normalize both the input and output of the operator network based on the mean and standard deviation of the training data for better performance.

We employ one DeepONet and one DeepOKAN to learn the solution operator G . The architectures are [961, 100, 100, 100, 100] for the branch net and [2, 100, 100, 100] for the trunk net, for both DeepONet and DeepOKAN. The DeepONet has tanh as the activation function, while the DeepOKAN is based on Chebyshev KANs and has degree three for the Chebyshev polynomials. The Adam optimizer [67] is employed for both operator networks. The learning rate for the DeepONet is 1e-3 for 100k iterations and 1e-4 for another 100k iterations, while for the DeepOKAN it is 1e-4 for 100k iterations and 1e-5 for another 100k iterations. We impose a l_2 regularizer [78] with weighting coefficient 1e-4 to the DeepOKAN to avoid the overfitting. The errors are presented in Table 9, from which we observe that the DeepONet outperforms the DeepOKAN at lower computational cost. However, as shown in Table 10, the robustness of the DeepOKAN is still better than the DeepONet when they are trained with clean input data while tested with noisy input data. In particular, the DeepOKAN becomes better in accuracy as the noise level grows larger.

4. Learning in PIKANs

4.1. Information bottleneck method

The Information Bottleneck (IB) method offers a perspective on the training and performance of neural networks using principles from information theory. It lays out a framework for determining the ideal balance between compression and prediction in supervised learning, proposing a principle for forming a condensed representation of layer activations \mathcal{T} with respect to an input variable \mathcal{X} , which preserves as much information as possible about an output variable \mathcal{Y} [37, 38]. Central to this theory is the use of mutual information $I(x, y)$, a measure of the information one random variable (y) reveals about another (x). This indicates that the best model representations should retain all relevant information about the output while omitting irrelevant information from the inputs, thereby establishing an “information bottleneck.” A notable insight from IB is that deep learning progresses through two distinct phases: fitting and diffusion, delineated by a phase transition driven by the signal-to-noise ratio (SNR) of the gradients [39, 40, 79]. The theory posits that significant learning happens during the gradual diffusive phase, crucial for the model’s ability to generalize effectively. Anagnostopoulos et al.[36] extended the information bottleneck method to interpret how physics-informed neural networks learn and prosed the existence of a third phase denominated total diffusion. In this section, we will apply this framework to describe the learning dynamics of cPIKANs and PINNs.

4.2. Signal-to-noise ratio(SNR)

As described in [36, 39, 40, 79], the batch-wise signal-to-noise ratio (SNR) is a metric used to identify the training dynamics of neural networks and can be described as follows:

$$\text{SNR} = \frac{\|\mu\|_2}{\|\sigma\|_2} = \frac{\|\mathbb{E}[\nabla_\theta \mathcal{L}_B]\|_2}{\|std[\nabla_\theta \mathcal{L}_B]\|_2} \quad (34)$$

where θ are the network parameters, and $\|\mu\|_2$ and $\|\sigma\|_2$ are the L^2 norms of the batch-wise mean and standard deviation of the total loss gradients ($\nabla_\theta \mathcal{L}_B$). Under this definition, the “signal” represents an idealized gradient that drives the optimizer to minimize the error of all subdomains, and the noise is the perturbation from the ideal gradient related to learning from the average of a finite number of observations. Following [36], we analyze the PINN and cPIKAN training dynamic of full-batch Adam trained with the entire dataset \mathcal{X} , and calculate $\nabla_\theta \mathcal{L}_B$ without performing an update of θ for each i.i.d. batch (\mathcal{B}) so that we can investigate the batch-wise behavior on the same iteration t .

4.3. Stages of Learning

The different stages can be interpreted as a process where the model fits the data (captures relevant information) and then compresses it (discards irrelevant information), further enhancing its generalization ability [80]. Each phase is characterized by the dominant term in the SNR. Highly deterministic regimes are characterized by a high signal and, with it, a high SNR. On the other hand, highly stochastic stages are defined by high noise and low SNR.

Fitting. At the beginning of training, the loss and its gradients are high for all subdomains. This agreement induces an initial high SNR characterized by a signal (i.e., direction) that helps the model reduce the training error of all subdomains. However, as the loss and its gradients (i.e., signal) decrease, the disagreement between subdomains (i.e., noise) increases too, which induces a low SNR. Therefore, the fitting stage can be defined as a deterministic phase characterized by a transition from high to low SNR. As shown in Figure 17(a) and (b), cPIKANs and PINNs present a fitting stage. In particular, their SNR goes from high to low, and their corresponding residuals display an ordered pattern (See Figure 17(c) and (d)).

Diffusion. Once the model has learned to fit the data (i.e., general traits), it starts an exploration stage aiming to find a signal (i.e., direction) that minimizes the training error in all subdomains. During this stage, the network weights start diffusing and aim to improve the model’s generalization capabilities, breaking the initial states’ order. Thus, the diffusion stage is characterized by a low fluctuating SNR. Figure 17(a) and (b) show that cPIKANs and PINNs display a diffusion state. Notice that in this stochastic stage, the residuals become disordered (i.e., Figure 17(a) and (b)), and SNR starts to oscillate.

Total Diffusion. Once the model has identified an optimal signal, the SNR suddenly increases to an equilibrium stage where the model exploits a consistent direction that minimizes the generalization error in all subdomains. During this phase, the model simplifies the internal representations of the learned patterns by keeping the important features and discarding the irrelevant ones, thus effectively reducing its complexity and breaking the order of the corresponding residuals. As shown in Figure 17(a) and (b), PINNs and PIKANs display a total diffusion stage. Notice that, for all representation models, as soon as the optimal direction (i.e., signal) is found (i.e., total diffusion starts), the generalization error (i.e., relative L^2) decreases faster, indicating optimal convergence. Thus, unsurprisingly, the best-performing models (i.e., PINN+RBA and cPIKAN+RBA) transition to total diffusion first.

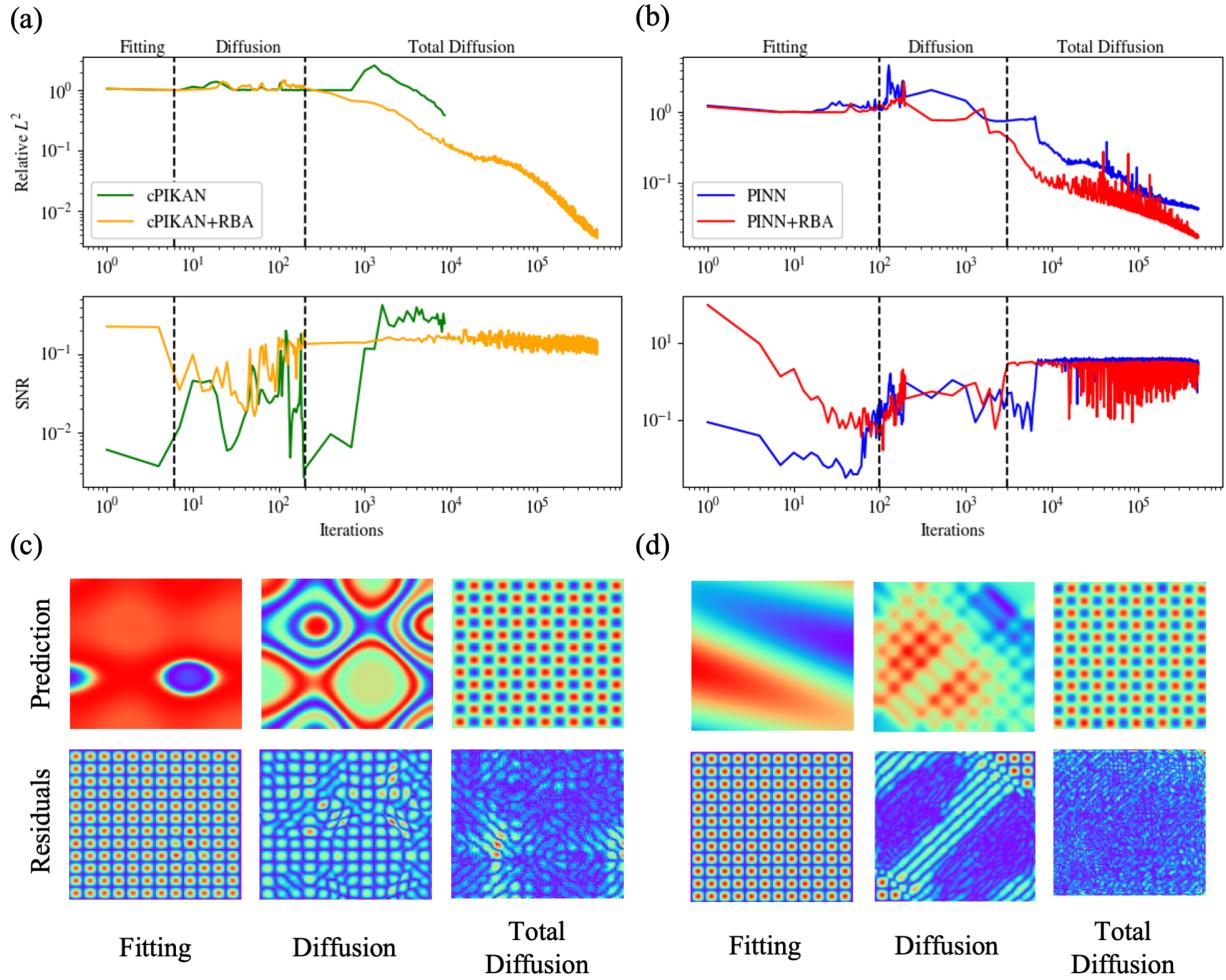


Figure 17: Training dynamics and stages of learning. Relative L^2 convergence and corresponding signal-to-noise ratio (SNR) for (a) cPIKANs and (b) PINNs. The vertical dashed lines indicate the transitions of cPIKANs+RBA and PINNs+RBA. During **fitting**, PIKANs and PINNs SNR go from high to low. This suggests an initial phase where the model closely fits the training data. The **Diffusion** phase is considered an exploratory stage characterized by a fluctuating low SNR. In the last stage, **total diffusion**, the SNR suddenly increases and converges to a critical value, and the generalization (i.e., relative L^2) error decreases faster, suggesting an optimal convergence. Notice that the best-performing models transition to total diffusion faster. Even though cPIKAN became undefined during the initial iterations, the three stages of learning are still identifiable. Prediction at residual distributions at different stages of learning in (c) cPIKAN and (d) PINN. The **fitting** phase is highly deterministic, so the residuals display an ordered pattern. As the SNR decreases and the model transitions to a stochastic **diffusion**, the residuals gradually become disordered. Finally, in **total diffusion** the model reaches an equilibrium state, simplifies internal representations, and reduces their complexity, making the model much more efficient and generalizable. Notice that during this stage, the predictions closely match the analytical solution. This phase is characterized by highly stochastic (i.e., noisy) residuals.

5. Summary

This study investigates the potential and effectiveness of KAN-based representations for tackling key challenges in scientific machine learning. We begin with evaluating KAN and Chebyshev-KAN for approximating discontinuous and oscillatory functions. The investigation revealed that both MLPs and KANs achieved high accuracy in approximating the function. However, training KANs was significantly slower compared to MLP representations. Notably, the Chebyshev-KAN approach exhibited unstable training, leading to rapid divergence. To address this issue, we implemented a modification to the Chebyshev-KAN forward pass by composing it with tanh function. This modification successfully stabilized the training process. Consequently, the function approximation performance of the modified Chebyshev-KAN became comparable to MLPs in terms of both accuracy and runtime efficiency. We then explored the application of Chebyshev-KAN to problems involving structure-preserving and energy-conserving dynamical systems, comparing its efficacy to MLP-based architectures. The results showed that Chebyshev-KAN is not as data-efficient as MLPs. However, with a slightly larger amount of training data, the performance of both models became comparable. Next, we consider the 2D Helmholtz equation. In this example, we demonstrate the capabilities of various neural network architectures, including PINN, PIKAN, PIKAN (multigrid), cPIKAN, PIKAN with RBA, and cPIKAN with RBA to recover the highly oscillatory solutions. Additionally, we analyze the loss landscapes for all these architectures to investigate their convexity and convergence behavior towards minima across different landscapes. Next we consider 2D steady state incompressible Navier-Stokes equation to solve the lid-driven cavity flow problem for low to high Reynold's number. Here in addition to Chebyshev, we also use Jacobi [73] and Legendre polynomials [73]. The Jacobi polynomials based KANs are proved to be promising candidate for physics informed network predict the incompressible flow. Among the various polynomials we have tested, the Chebyshev PIKAN (cPIKAN) shows its advantage in terms of accuracy and training time. Compared to the PINN, cPIKAN can achieve the same accuracy with far less network parameters, but it takes more GPU hours to train. The residual-based attention (RBA) helps to improve both PINN and PIKAN's accuracy, without need to consume more training time. In addition, the vanilla PIKANs suffered from unstable training for flow at Re , shown by the fact that the relative error barely goes down with training. Nonetheless, with the help of the entropy viscosity method (EVM), it can recover the correct training trajectory and obtain accurate solution.

Next we study the KAN based representation for solving the PDE with noisy data. In addressing noisy data with uncertainty quantification in solving differential equations, cPIKANs are compatible with the Bayesian framework and the Bayesian cPIKAN (B-cPIKAN) method is able to provide similar predicted mean and uncertainty as the Bayesian PINN (B-PINN) method [44] at higher computational cost. However, specifying the prior distribution for parameters of the B-cPIKAN is not as straightforward as it is for parameters of the B-PINN, and requires further theoretical and numerical work in the future. As discussed in Section 3, the DeepOKAN, which integrates the DeepONet [34] structure with the Chebyshev KAN architecture [26], has shown competitive performance in operator learning compared to the DeepONet, indicating DeepOKAN as a promising alternative representation model. Additionally, it is significantly more robust to noisy input functions in the testing

stage after being trained with clean data.

Finally, we study the learning behavior of KAN and MLP-based representation using information-bottleneck theory. The Information Bottleneck (IB) method has been effectively extended to the study of cPIKANs. The foundational training dynamics exhibit remarkable similarities despite the architectural distinctions between PINNs and cPIKANs. Both representation models demonstrate a consistent progression through the stages of fitting, diffusion, and total diffusion, as outlined by the IB framework. This insight into network behavior aims to bridge the gap between the representation models and motivate future research to use the IB method as a guide to develop training strategies and new architectures or enhance model performance. Future research directions include

1. Implementing KAN-based representations for solving large-scale partial differential equations (PDEs) using domain decomposition techniques. This approach has the potential to improve the scalability of KAN-based methods for complex problems.
2. Implementing KAN-based representations for solving large-scale partial differential equations (PDEs) using domain decomposition techniques [81]. This approach has the potential to improve the scalability of KAN-based methods for complex problems.
3. Applying KAN-based representations to time-dependent PDEs in two and three dimensions. This would extend the applicability of KAN methods to a wider range of scientific and engineering problems.
4. A rigorous theory for the convergence of KAN for elliptic and parabolic class of PDEs.
5. Extending the applicability of DeepOKAN based architecture for formulating surrogate for industrial complexity problems [57].

References

- [1] S. Haykin, Neural networks: a comprehensive foundation, Prentice Hall PTR, 1998.
- [2] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (4) (1989) 303–314.
- [3] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural networks 2 (5) (1989) 359–366.
- [4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial networks, Communications of the ACM 63 (11) (2020) 139–144.
- [5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).
- [6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
- [7] H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, Advances in neural information processing systems 31 (2018).

- [8] J. D. Toscano, C. Zuniga-Navarrete, W. D. J. Siu, L. J. Segura, H. Sun, Teeth mold point cloud completion via data augmentation and hybrid rl-gan, *Journal of Computing and Information Science in Engineering* 23 (4) (2023) 041008.
- [9] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, *Journal of artificial intelligence research* 4 (1996) 237–285.
- [10] M. Cranmer, Interpretable machine learning for science with pysr and symbolicregression. jl, arXiv preprint arXiv:2305.01582 (2023).
- [11] 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 (2024).
- [12] D. A. Sprecher, S. Draghici, Space-filling curves and Kolmogorov superposition-based neural networks, *Neural Networks* 15 (1) (2002) 57–67.
- [13] 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.
- [14] J. Schmidhuber, Discovering neural nets with low Kolmogorov complexity and high generalization capability, *Neural Networks* 10 (5) (1997) 857–873.
- [15] 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 (2021).
- [16] P.-E. Leni, Y. D. Fougerolle, F. Truchetet, The kolmogorov spline network for image processing, in: *Image Processing: Concepts, Methodologies, Tools, and Applications*, IGI Global, 2013, pp. 54–78.
- [17] J. He, On the optimal expressive power of relu dnns and its application in approximation with kolmogorov superposition theorem, arXiv preprint arXiv:2308.05509 (2023).
- [18] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, *Journal of Computational Physics* 404 (2020) 109136.
- [19] S. Guarnieri, F. Piazza, A. Uncini, Multilayer feedforward networks with adaptive spline activation function, *IEEE Transactions on Neural Networks* 10 (3) (1999) 672–683.
- [20] D. Fakhouri, E. Fakhouri, H. Speleers, ExSpliNet: An interpretable and expressive spline-based neural network, *Neural Networks* 152 (2022) 332–346.
- [21] C. J. Vaca-Rubio, L. Blanco, R. Pereira, M. Caus, Kolmogorov-Arnold Networks (KANs) for Time Series Analysis, arXiv preprint arXiv:2405.08790 (2024).
- [22] M. E. Samadi, Y. Müller, A. Schuppert, Smooth Kolmogorov Arnold networks enabling structural knowledge representation, arXiv preprint arXiv:2405.11318 (2024).

- [23] Z. Li, Kolmogorov-Arnold Networks are Radial Basis Function Networks, arXiv preprint arXiv:2405.06721 (2024).
- [24] Z. Bozorgasl, H. Chen, Wav-KAN: Wavelet Kolmogorov-Arnold Networks (2024). arXiv:2405.12832.
- [25] NLNR, Jacobikan, <https://github.com/mintisan/awesome-kan/> (2024).
- [26] SynodicMonth, ChebyKAN, <https://github.com/SynodicMonth/ChebyKAN/> (2024).
- [27] S. SS, Chebyshev Polynomial-Based Kolmogorov-Arnold Networks: An Efficient Architecture for Nonlinear Function Approximation, arXiv preprint arXiv:2405.07200 (2024).
- [28] S. S. Bhattacharjee, TorchKAN: Simplified KAN Model with Variations, <https://github.com/1ssb/torchkan/> (2024).
- [29] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *Journal of Computational physics* 378 (2019) 686–707.
- [30] D. W. Abueidda, P. Pantidis, M. E. Mobasher, DeepOKAN: Deep Operator Network Based on Kolmogorov Arnold Networks for Mechanics Problems, arXiv preprint arXiv:2405.19143 (2024).
- [31] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, *Nature Reviews Physics* 3 (6) (2021) 422–440.
- [32] L. D. McClenney, U. M. Braga-Neto, Self-adaptive physics-informed neural networks, *Journal of Computational Physics* 474 (2023) 111722.
- [33] Z. Wang, X. Meng, X. Jiang, H. Xiang, G. E. Karniadakis, Solution multiplicity and effects of data and eddy viscosity on Navier-Stokes solutions inferred by physics-informed neural networks, arXiv preprint arXiv:2309.06010 (2023).
- [34] 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.
- [35] C. Wu, M. Zhu, Q. Tan, Y. Kartha, L. Lu, A comprehensive study of non-adaptive and residual-based adaptive sampling for physics-informed neural networks, *Computer Methods in Applied Mechanics and Engineering* 403 (2023) 115671.
- [36] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopoulos, G. E. Karniadakis, Learning in PINNs: Phase transition, total diffusion, and generalization, arXiv preprint arXiv:2403.18494 (2024).
- [37] N. Tishby, F. C. Pereira, W. Bialek, The information bottleneck method, arXiv preprint physics/0004057 (2000).

- [38] N. Tishby, N. Zaslavsky, Deep learning and the information bottleneck principle, in: 2015 ieee information theory workshop (itw), IEEE, 2015, pp. 1–5.
- [39] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, arXiv preprint arXiv:1703.00810 (2017).
- [40] Z. Goldfeld, Y. Polyanskiy, The information bottleneck problem and its applications in machine learning, *IEEE Journal on Selected Areas in Information Theory* 1 (1) (2020) 19–38.
- [41] A. F. Psaros, X. Meng, Z. Zou, L. Guo, G. E. Karniadakis, Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons, *Journal of Computational Physics* 477 (2023) 111902.
- [42] S. Cai, Z. Mao, Z. Wang, M. Yin, G. E. Karniadakis, Physics-informed neural networks (PINNs) for fluid mechanics: A review, *Acta Mechanica Sinica* 37 (12) (2021) 1727–1738.
- [43] Z. Mao, A. D. Jagtap, G. E. Karniadakis, Physics-informed neural networks for high-speed flows, *Computer Methods in Applied Mechanics and Engineering* 360 (2020) 112789.
- [44] L. Yang, X. Meng, G. E. Karniadakis, B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data, *Journal of Computational Physics* 425 (2021) 109913.
- [45] X. Meng, Z. Li, D. Zhang, G. E. Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent PDEs, *Computer Methods in Applied Mechanics and Engineering* 370 (2020) 113250.
- [46] Z. Zou, G. E. Karniadakis, L-HYDRA: Multi-head physics-informed neural networks, arXiv preprint arXiv:2301.02152 (2023).
- [47] Z. Zou, X. Meng, G. E. Karniadakis, Correcting model misspecification in physics-informed neural networks (PINNs), *Journal of Computational Physics* 505 (2024) 112918.
- [48] Z. Zhang, Z. Zou, E. Kuhl, G. E. Karniadakis, Discovering a reaction–diffusion model for Alzheimer’s disease by combining PINNs with symbolic regression, *Computer Methods in Applied Mechanics and Engineering* 419 (2024) 116647.
- [49] P. Chen, T. Meng, Z. Zou, J. Darbon, G. E. Karniadakis, Leveraging multitime Hamilton–Jacobi PDEs for certain scientific machine learning problems, *SIAM Journal on Scientific Computing* 46 (2) (2024) C216–C248.
- [50] P. Chen, T. Meng, Z. Zou, J. Darbon, G. E. Karniadakis, Leveraging Hamilton-Jacobi pdes with time-dependent Hamiltonians for continual scientific machine learning, arXiv preprint arXiv:2311.07790 (2023).

- [51] 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 (2020).
- [52] K. Shukla, P. C. Di Leoni, J. Blackshire, D. Sparkman, G. E. Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, *Journal of Nondestructive Evaluation* 39 (2020) 1–20.
- [53] K. Shukla, A. D. Jagtap, J. L. Blackshire, D. Sparkman, G. E. Karniadakis, A physics-informed neural network for quantifying the microstructural properties of polycrystalline nickel using ultrasound data: A promising approach for solving inverse problems, *IEEE Signal Processing Magazine* 39 (1) (2021) 68–77.
- [54] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopoulos, G. E. Karniadakis, Residual-based attention in physics-informed neural networks, *Computer Methods in Applied Mechanics and Engineering* 421 (2024) 116805.
- [55] 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.
- [56] Z. Zou, A. Kahana, E. Zhang, E. Turkel, R. Ranade, J. Pathak, G. E. Karniadakis, Large scale scattering using fast solvers based on neural operators, arXiv preprint arXiv:2405.12380 (2024).
- [57] K. Shukla, V. Oommen, A. Peyvan, M. Penwarden, N. Plewacki, L. Bravo, A. Ghoshal, R. M. Kirby, G. E. Karniadakis, Deep neural operators as accurate surrogates for shape optimization, *Engineering Applications of Artificial Intelligence* 129 (2024) 107615.
- [58] X. Meng, L. Yang, Z. Mao, J. del Águila Ferrandis, G. E. Karniadakis, Learning functional priors and posteriors from data and physics, *Journal of Computational Physics* 457 (2022) 111073.
- [59] Z. Zou, X. Meng, A. F. Psaros, G. E. Karniadakis, NeuralUQ: A comprehensive library for uncertainty quantification in neural differential equations and operators, *SIAM Review* 66 (1) (2024) 161–190.
- [60] Z. Zou, X. Meng, G. E. Karniadakis, Uncertainty quantification for noisy inputs-outputs in physics-informed neural networks and neural operators, arXiv preprint arXiv:2311.11262 (2023).
- [61] J. Lin, awesome-kan, <https://github.com/SpaceLearner/JacobiKAN/> (2024).
- [62] G. Karniadakis, S. Sherwin, *Spectral/hp Element Methods for Computational Fluid Dynamics*, 2nd edition, Oxford University Press, Oxford, UK, 2005.
- [63] B. Ter-Avanesov, awesome-kan, <https://github.com/Boris-73-TA/OrthogPolyKANs/> (2024).

- [64] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: International conference on machine learning, PMLR, 2019, pp. 5301–5310.
- [65] S. Greydanus, M. Dzamba, J. Yosinski, Hamiltonian neural networks, Advances in neural information processing systems 32 (2019).
- [66] A. Garg, S. S. Kagi, Hamiltonian neural networks (2019).
- [67] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
- [68] Y. He, Z. Wang, H. Xiang, X. Jiang, D. Tang, An artificial viscosity augmented physics-informed neural network for incompressible flow, Applied Mathematics and Mechanics 44 (7) (2023) 1101–1110.
- [69] J.-L. Guermond, R. Pasquetti, B. Popov, Entropy viscosity method for nonlinear conservation law, Journal of Computational Physics 230 (11) (2011) 4248–4267.
- [70] Z. Wang, M. S. Triantafyllou, Y. Constantinides, G. Karniadakis, An entropy-viscosity large eddy simulation study of turbulent flow in a flexible pipe, Journal of Fluid Mechanics 859 (2019) 691–730.
- [71] X. Jin, S. Cai, H. Li, G. E. Karniadakis, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations, Journal of Computational Physics 426 (2021) 109951.
- [72] S. M. Allen, J. W. Cahn, Ground state structures in ordered binary alloys with second neighbor interactions, Acta Metallurgica 20 (3) (1972) 423–433.
- [73] G. Karniadakis, S. J. Sherwin, Spectral/hp element methods for computational fluid dynamics, Oxford University Press, USA, 2005.
- [74] K. Linka, A. Schäfer, X. Meng, Z. Zou, G. E. Karniadakis, E. Kuhl, Bayesian physics informed neural networks for real-world nonlinear dynamical systems, Computer Methods in Applied Mechanics and Engineering 402 (2022) 115346.
- [75] M. Yin, Z. Zou, E. Zhang, C. Cavinato, J. D. Humphrey, G. E. Karniadakis, A generative modeling framework for inferring families of biomechanical constitutive laws in data-sparse regimes, Journal of the Mechanics and Physics of Solids 181 (2023) 105424.
- [76] Z. Zou, T. Meng, P. Chen, J. Darbon, G. E. Karniadakis, Leveraging viscous Hamilton–Jacobi PDEs for uncertainty quantification in scientific machine learning, arXiv preprint arXiv:2404.08809 (2024).
- [77] R. M. Neal, et al., MCMC using Hamiltonian dynamics, Handbook of markov chain monte carlo 2 (11) (2011) 2.
- [78] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv preprint arXiv:1711.05101 (2017).

- [79] R. Shwartz-Ziv, Information flow in deep neural networks, arXiv preprint arXiv:2202.06749 (2022).
- [80] S. J. Anagnostopoulos, J. D. Toscano, N. Stergiopoulos, G. E. Karniadakis, Residual-based attention and connection to information bottleneck theory in PINNs, arXiv preprint arXiv:2307.00379 (2023).
- [81] K. Shukla, A. D. Jagtap, G. E. Karniadakis, Parallel physics-informed neural networks via domain decomposition, Journal of Computational Physics 447 (2021) 110683.