ReBaNO: Reduced Basis Neural Operator Mitigating Generalization Gaps and Achieving Discretization Invariance

Haolan Zheng* Yanlai Chen† Jiequn Han‡ Yue Yu§

Abstract

We propose a novel data-lean operator learning algorithm, the Reduced Basis Neural Operator (ReBaNO), to solve a group of PDEs with multiple distinct inputs. Inspired by the Reduced Basis Method and the recently introduced Generative Pre-Trained Physics-Informed Neural Networks, ReBaNO relies on a mathematically rigorous greedy algorithm to build its network structure offline adaptively from the ground up. Knowledge distillation via task-specific activation function allows ReBaNO to have a compact architecture requiring minimal computational cost online while embedding physics. In comparison to state-of-the-art operator learning algorithms such as PCA-Net, DeepONet, FNO, and CNO, numerical results demonstrate that ReBaNO significantly outperforms them in terms of eliminating/shrinking the generalization gap for both in- and out-of-distribution tests and being the only operator learning algorithm achieving strict discretization invariance.

1 Introduction

With the advent of deep neural networks, deep learning has achieved enormous success in various fields of science and engineering, especially in solving Partial Differential Equations (PDEs) [36, 41, 42, 46]. In particular, in the problems where repeated and computationally expensive simulations are demanded, deep neural networks are competitive as a fast and reliable solver. As an emerging approach, the paradigm of operator learning has been shown to be computationally efficient compared to traditional numerical methods. Unlike classical deep learning applications that require approximations to the mappings between finite-dimensional vector spaces (for example, from image pixellation or text embeddings), operator learning models aim to infer mappings between infinite-dimensional function spaces. When solving a group of PDEs with distinct inputs, conventionally we need a fine discretization to solve each PDE accurately using certain computational methods. Nevertheless, repeated simulations are computationally prohibitive when dealing with complex systems, especially nonlinear problems.

*Department of Mathematics, University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA. Email: hzheng1@umassd.edu. Research is supported by National Science Foundation grant DMS-2208277. The code of ReBaNO is available at https://github.com/haolanzheng/rebano.

†Department of Mathematics, University of Massachusetts Dartmouth, North Dartmouth, MA 02747, USA. Email: yanlai.chen@umassd.edu. Research is supported in part by National Science Foundation grant DMS-2208277 and by Air Force Office of Scientific Research grant FA9550-25-1-0181.

‡Flatiron Institute, New York, NY 10010, USA. Email: jhan@flatironinstitute.org .

§Department of Mathematics, Lehigh University, Bethlehem, PA 18015, USA. Email: yuy214@lehigh.edu. Research is supported in part by National Institute of Health grant 1R01GM157589-01.

Recently, Neural Operators (NOs) [34, 36, 42, 43, 46, 57, 64] were proposed as data-driven deep learning framework for learning operators, and they provide a promising path to alleviate the prohibitive computational cost from repeatedly solving PDEs. With NOs, one can train a model using labeled data from PDE solutions of multiple inputs, and then employ the model to provide an efficient prediction on new and unseen inputs. Based on the universal approximation theorem [13], neural operators provide discretization convergent approximations to ground truth operators from the data, which outperform the finite-dimensional operators given by traditional deep neural networks. The convergence and error bounds are also guaranteed under some assumptions [17, 34, 38, 68]. However, there are three limitations of data-driven neural operators. First, a neural operator requests a large number of annotated input-output pairs for training. To generate sufficiently large training datasets, it usually involves computationally expensive numerical simulations or costly experiments. Second, pure data-driven models can not guarantee fundamental physics laws, such as the conservation laws. The physics knowledge needs to be implicitly inferred from data, leading to poor generalizability to unseen inputs, in particular, Out-of-Distribution (OOD) inputs [16, 50]. Third, neural operators struggle in super/sub-resolution tests when they are trained in the low-resolution regime, meaning that the neural operator does not learn a truly mesh-free approximator to the target operator [47]. To alleviate the first and second limitations, Physics-Informed Neural Operators (PINOs) [25, 43] were introduced, which attempt to embed physical constraints in the loss function. However, PINOs still face difficulties from highly non-convex optimization, long training time and sensitivity to the choice of loss weights.

To address these issues, we propose the **Reduced Basis Neural Operator (ReBaNO)**, a data-lean reduced basis-driven operator learning algorithm inspired by the Physics-Informed Neural Network (PINN) [55] and the Generative Pre-Trained Physics-Informed Neural Network (GPT-PINN) [15]. ReBaNO leverages a reduced basis approach. By drawing from the success of PINNs in approximating individual PDE solutions, our method features a similar design of an *offline-online* decomposition. During the offline stage, ReBaNo uses a small number of representative full-order PINN solutions to build up a reduced space during the offline stage. On the online stage, given a new instance of input, the corresponding solution can be obtained efficiently by fine-tuning a lightweight network with a single hidden layer, where each unit, conceptually acting as a 'neuron', encapsulates a precomputed full PINN. Compared to existing neural operator methods, our approach has two features. First, it requires *no* high-fidelity training data. Second, while it significantly reduces generalization errors for OOD predictions, while maintaining a competitive accuracy for In-Distribution (ID) predictions. Furthermore, ReBaNO intrinsically adheres to a mesh-free paradigm by integrating mesh-independent PINNs as functional building blocks.

Our main contributions at present include the following.

1. we introduce a physics-informed operator learning framework based on a mathematically rigorous reduced basis approach, which adaptively constructs a low-dimensional surrogate space from full-order PINN solutions to improve generalization while maintaining efficiency and accuracy;

2. we propose an offline-online decomposition paradigm, eliminating the need for training data and enabling efficient physics-informed inference through a minimal network structure;

3. we demonstrate that ReBaNO consistently outperforms state-of-the-art operator learning methods in terms

of reducing the generalization gaps in OOD tests and being the only one achieving strict discretizations invariance.

The remainder of this paper is structured as follows. First, in Section 2 we briefly review operator learning models, reduced basis method (RBM), and PINNs. Next, we present problem setting and build a unified framework of operator learning models in Section 3. In Section 4, we entail the design of ReBaNO, which is empirically validated in Section 5, where we compare the performance of ReBaNO with other models on three benchmark problems and showcase the better generalizability and superior mesh invaraiance of ReBaNO. Finally, we conclude this paper in Section 6.

2 Background and Related Work

Predicting complex physical responses is ubiquitous in many scientific and engineering applications [4, 10, 12, 19, 21, 22, 29, 31, 32, 45, 52, 69]. Traditionally, the governing law is given as a PDE, and numerical methods, such as finite difference method (FDM) and finite element method (FEM), are developed to discretize the domain into a grid/basis and approximate the PDE solutions on this grid/basis [8, 40, 49]. Unavoidably, since these methods usually rely on a highly fine grid, the generation of grid/basis is time-consuming and may impose truncation errors. To accelerate the PDE solving procedure, reduced basis method was introduced, characterized by the greedy algorithm and the efficiency of its offline-online decomposition. Moreover, neural networks, which have demonstrated revolutionary success in modern artificial intelligence tasks, have also been shown to be powerful in solving scientific problems [19, 36, 46, 55]. In particular, PINNs and operator learning paradigm pave two distinct ways towards scientific machine learning.

This section is devoted to a concise overview of operator learning models in Subsection 2.1, a short introduction to PINN and its variational form in Subsection 2.2, and then a brief review of RBM in Subsection 2.3.

2.1 Neural Operators

Unlike classical neural network architectures for vector-to-vector mappings, neural operators seek to approximate the mappings between infinite-dimensional Banach spaces as applications of machine learning to PDEs [1, 11, 25, 26, 28, 41, 42, 44, 46, 51, 67]. As the emerging successful paradigm, most of NOs are data-driven deep learning models such as DeepONet [46], PCA-Net [5], Fourier Neural Operator (FNO) [42] and its variants [43, 60, 64], and Convolutional Neural Operator (CNO) [57]. In addition, transformer-based methods, such as Galerkin Transformer [11], Transolver [48, 65], take advantage of attention mechanism to capture physical correlations and handle general geometrics. These neural operators are often employed to approximate the mapping between spatial and/or spatio-temporal function pairs. Compared with classical neural networks, the NOs are more favorable due to their resolution independence, convergence guarantee, and fast inference. More specifically, NOs can be evaluated on any given discretization rather than the mesh on which they are trained [42, 71]. And the convergence of NOs to the target operator and the error estimate are proved given certain assumptions based on the universal approximation theorem for operators [13, 34–36]. Moreover, once NOs are trained, they can be evaluated highly efficiently in a single forward pass. However, despite these

features, purely data-driven NOs still suffer from the data challenge: they require a large set of paired data and do not generalize well when test samples are out of the training regime. To reduce this generalization gap, PINO [43] and physics-informed DeepONet [25, 62] were introduced, where a PDE-based loss is added to the training loss as a penalization term. When applying the PDE loss in the test phase, a test-time optimization is performed to take advantage of both the learned neural operator and the additional governing equation, enabling a more accurate solution function and a smaller generalization error on the querying instance [43]. However, computing derivatives of batched outputs through backpropagation demands a large memory overhead. Moreover, the highly complex landscape of the physics-informed loss leads to a long optimization procedure, even the failure of convergence. And one needs to carefully fine-tune the loss weights of the data loss and the PDE loss to enhance the performance.

2.2 Physics-Informed Neural Network

PINNs have become a powerful tool for solving various PDEs arising from numerous forward and inverse problems [55]. Instead of approximating the operator between function spaces, PINNs employ deep neural networks trained on several sets of collocation points in the physical domain and its boudaries to approximate the solutions to a given PDE by minimizing the loss that consists of the PDE residual as well as boundary/initial conditions. Specifically, assume that the solution $u \in \mathcal{U}$ satisfies the following time-dependent PDE defined on a bounded domain $\Omega \subset \mathbb{R}^m$ and its boundary $\partial\Omega$

$$\frac{\partial u}{\partial t} + (\mathcal{N}u)(\mathbf{x}, t) = g(\mathbf{x}, t), \qquad (\mathbf{x}, t) \in \Omega \times [0, T], \tag{1}$$

$$(\mathcal{B}u)(\mathbf{x}, t) = 0, \qquad (\mathbf{x}, t) \in \partial\Omega \times [0, T], \tag{2}$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{3}$$

where $\mathcal{N}$ encodes a differential operator and $\mathcal{B}$ the boundary conditions. The PINN solution $u_{\text{NN}}(\mathbf{x}, t; \theta)$ parameterized by neural network parameters $\theta \in \Theta$ is identified by minimizing the loss function:

$$\mathcal{L}[u_{\text{NN}}(\mathbf{x}, t; \theta)] = \left\| \frac{\partial u_{\text{NN}}}{\partial t} + (\mathcal{N}u_{\text{NN}}) - g \right\|_{L^2}^2 + \|(\mathcal{B}u_{\text{NN}})\|_{L^2}^2 + \|u_{\text{NN}} - u_0\|_{L^2}^2 \tag{4}$$

The L^2 norms for the three terms above are defined in the domains $\Omega \times [0, T]$, $\partial\Omega \times [0, T]$, and $\Omega \times \{t = 0\}$, respectively. This loss can be discretized and evaluated on three sets of collocation points $\{(\mathbf{x}_i^R, t_i^R\}_{i=1}^{s_R}$ for the residual, $\{(\mathbf{x}_i^B, t_i^B\}_{i=1}^{s_B}$ for the boundary condition, and $\{\mathbf{x}_i^I\}_{i=1}^{s_I}$ for the initial condition.

However, the loss Eq. (4) takes the strong form of the PDE causing PINN to possibly fail to provide accurate solutions to the PDEs with weak regularity. Therefore, we resort to the so-called Variational PINN (VPINN) first proposed in [**?**], which applies the weak form of the PDE when calculating the residual loss. Let the weak form of the PDE be $a(u, v) = \ell(v)$ $(v \in \mathcal{V})$ where $\mathcal{V}$ denotes the test function space and $\ell(v)$ corresponds to the source term g in Eq. (1). In this work, we use the loss from the formulation of Robust VPINN [**?**]

$$\mathcal{L}[u_{\text{NN}}(\mathbf{x}, t; \theta)] = R(u_{\text{NN}})^T G^{-1} R(u_{\text{NN}}) + \mathcal{L}_b(u_{\text{NN}}) + \mathcal{L}_i(u_{\text{NN}}) \tag{5}$$

where R is a vector-valued function, the element of which is defined as the residual of the weak form with respect to a set of test functions $\{\varphi_i(x) \in \mathcal{V}\}_{i=1}^{N_f}$. That is, $R(u)_n \coloneqq r(u, \varphi_n) \coloneqq \ell(\varphi_n) - a(u, \varphi_n)$ $(n = 1, 2, \ldots, N_f)$. G is the Gram matrix and is defined by $G_{mn} \coloneqq (\varphi_n, \varphi_m)_{\mathcal{V}}$ ($(\cdot, \cdot)_{\mathcal{V}}$ denotes the inner product in the function space $\mathcal{V}$ and $\|\cdot\|_{\mathcal{V}}$ the norm induced by the inner product). $\mathcal{L}_b$ and $\mathcal{L}_i$ are the same losses in Eq. (4) of the boundary conditions and the initial conditions.

Although PINN and its variants have obtained promising results on various physics modeling tasks, PINNs face several challenges. (1) For problems involving complex physics [20, 56, 63], the PDE constraint induces highly nonlinear optimization landscapes [23, 61], resulting in slow or even failed convergence. (2) For every new instance of boundary conditions/physical parameters, a new training needs to be performed in PINNs. As a result, PINNs become expensive when multiple instances of PDEs need to be solved, such as in inverse PDE and design problems [39, 59]. To alleviate the second challenge, several strategies have been proposed. Examples include the TL-PINN approach based on a transfer-learning philosophy to fine-tune a pre-trained model and fit the new physics constraint [18, 24, 53, 66], the GPT-PINN approach which leverages a meta-learning idea to adaptively infer the parametric dependence of the system and expands a meta-network [14, 15, 33, 70], and so on. However, to the best of our knowledge, none of these approaches is capable of resolving the optimization challenge in PINNs.

2.3 Reduced Basis Method

As ReBaNO is inspired by RBM, we devote this subsection to a brief review of RBM. RBM [27, 30, 54] is a popular model order reduction (MOR) technique capable of rigorously and efficiently simulating parametric PDEs. Its signature, in comparison to other MOR approaches, is a greedy algorithm integrated in an offline-online learning procedure. The offline (i.e., training) stage is devoted to a judicious exploration of the parameter-induced solution manifold driven by an error estimator. It relies on a high-fidelity numerical procedure for the parameter to solution map $\mu \to u(x; \mu) \in X_h$. We call this Full-Order Model, FOM(μ, X_h). RBM selects a number of representative parameter values $\{\mu^n\}_{n=1}^N$ via a mathematically rigorous greedy algorithm [6] to form an N-dimensional subspace of X_h, $X_N \coloneqq \text{span}\,\{u(\cdot; \mu^n)\}_{n=1}^N$. During the online stage, the method computes an approximation $u_N(\cdot, \mu)$ in the surrogate space for any unseen (but still in distribution) parameter value via the ansatz $u_N(\cdot, \mu) = \sum_{n=1}^N c_n(\mu) u(\cdot, \mu^n)$. Thanks to this ansatz and the greedy algorithm iteratively called to build the surrogate space from ground up, in comparison to other reduction techniques (e.g. proper orthogonal decomposition (POD)-based approaches [3, 37]), the number of full order inquiries RBM takes offline is minimum, i.e., equal to the dimension of the surrogate space.

3 A Unified Operator Learning Framework

In this section, we aim to outline a unified operator learning framework to place our discussions in context. In Subsection 3.1, we first introduce the notation and establish the problem setting. In Subsection 3.2, we elucidate the details of the framework and give a unified description of the NOs that we study in our experiments.

3.1 Problem Setting

Without losing generality, we consider a family of PDEs of the following general form:

$$\frac{\partial u}{\partial t} + (\mathcal{N}_f u)(\mathbf{x}, t) = g(\mathbf{x}, t), \qquad (\mathbf{x}, t) \in \Omega \times [0, T] \tag{6}$$

$$(\mathcal{B}u)(\mathbf{x}, t) = h(\mathbf{x}, t), \qquad (\mathbf{x}, t) \in \partial\Omega \times [0, T] \tag{7}$$

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega \tag{8}$$

for some $h \in \mathcal{U}_b, u_0 \in \mathcal{U}_i$ and $\Omega \subset \mathbb{R}^n$ (for some positive integer n) a bounded domain, where $\mathcal{N}_f : \mathcal{U} \times \mathcal{F} \to \mathcal{U}^*$ is a (non)linear differential operator dependent on an input function $f \in \mathcal{F}$ and $\mathcal{B}$ encodes the boundary condition. Assume that the solution $u : \Omega \times [0, T] \to \mathbb{R}^{d_u}$ belongs to a Banach function space $\mathcal{U}$, and that the boundary function space $\mathcal{U}_b$ and the initial function space $\mathcal{U}_i$ are also Banach spaces. The source term g on the right-hand side is from the dual space of $\mathcal{U}$, denoted by $\mathcal{U}^*$. We also assume that the input function space $\mathcal{F}$ is a Banach space of functions and is defined as $\mathcal{F} \coloneqq \{f : \Omega' \to \mathbb{R}^{d_f},\ \Omega' \subset \mathbb{R}^m\}$ (for some positive integer m).[1] For simplicity of our discussion, we assume $\Omega' = \Omega$. That means, the input function (e.g., parameter field) and the solution are defined on the same physical domain.

The goal of operator learning is to approximate the groundtruth solution operator Ψ mapping between two infinite-dimensional Banach spaces $\mathcal{F}$ and $\mathcal{U}$, i.e., $\Psi : \mathcal{F} \to \mathcal{U}$. To this end, we construct a parametric operator

$$\Psi^{\sim} : \mathcal{F} \times \Theta \to \mathcal{U} \tag{9}$$

where Θ is the trainable parameter space and we seek an optimal $\theta^* \in \Theta$ so that the resulting operator $\Psi^*(\cdot) \coloneqq \Psi^{\sim}(\cdot, \theta^*) \approx \Psi(\cdot)$. The optimal parameter θ^* is found by minimizing the loss function $\mathcal{L}(\Psi, \Psi^{\sim}; \theta)$, i.e.

$$\theta^* = \arg\min_{\theta \in \Theta} \mathcal{L}(\Psi, \Psi^{\sim}; \theta). \tag{10}$$

For data-driven models, we define the loss function as the Bochner norm of the approximation error

$$\mathcal{L}_d(\Psi, \Psi^*; \theta) \coloneqq \mathbb{E}_{f \sim \mu} \|\Psi(f) - \Psi^{\sim}(f; \theta)\|^2_{L^2_\mu(\mathcal{F}; \mathcal{U})} = \int_{\mathcal{F}} \|\Psi(f) - \Psi^{\sim}(f; \theta)\|^2_{\mathcal{U}} \, \mathrm{d}\mu(f)\,, \tag{11}$$

where μ is a probability measure supported on $\mathcal{F}$. To compute this loss, we need N_p i.i.d. Monte Carlo samples of the input functions $\{f_i\}_{i=1}^{N_p}$ from μ and possibly a set of corresponding (approximate) solutions $\{u_i\}_{i=1}^{N_p}$ with $u_i \approx \Psi(f_i)$ for training. Thus, the loss can be approximated by the empirical risk.

$$\mathcal{L}_d(\Psi, \Psi^*; \theta) \approx \frac{1}{N_p} \sum_{i=1}^{N_p} \|u_i - \Psi^{\sim}(f_i; \theta)\|^2_{\mathcal{U}}. \tag{12}$$

This empirical risk is usually computed using the L^2 relative error instead of the $\mathcal{U}$-norm.

For physics-driven models, the loss function is independent of the approximated solution data and consists

[1] We remark that, for notational simplicity, we focus on variations with respect to f only, though the formulation naturally extends to learning operators involving variations of g and h as well.

of the PDE residual along with the boundary and the initial condition

$$\mathcal{L}_p(\Psi^\sim;\theta) \coloneqq \mathbb{E}_{f\sim\mu}\left[\left\|\frac{\partial\Psi^\sim(f;\theta)}{\partial t} + \mathcal{N}_f\Psi^\sim(f;\theta) - g\right\|_{\mathcal{U}^*} + \|\mathcal{B}\Psi^\sim(f;\theta) - h\|_{\mathcal{U}_b} + \|\Psi^\sim(f;\theta) - u_0\|_{\mathcal{U}_i}\right]. \quad (13)$$

The above three norms are defined over the domains $\Omega \times [0,T]$, $\partial\Omega \times [0,T]$ and $\Omega \times \{t = 0\}$, respectively. This loss function is exactly the same as the PINN loss Eq. (4) when the norms are replaced with the L^2 norm. Although there has been work indicating that proper norms should be chosen for convergence and error control (see, for example, [7]), we adopt the L^2 norm in our experiments for simplicity and practical use. In this case, the empirical risk of the physics-informed loss becomes

$$\mathcal{L}_p(\Psi^\sim;\theta) \approx \frac{1}{N_p}\sum_{i=1}^{N_p}\left[\left\|\frac{\partial\Psi^\sim(f_i;\theta)}{\partial t} + \mathcal{N}_f\Psi^\sim(f_i;\theta) - g\right\|_{L^2}^2 + \|\mathcal{B}\Psi^\sim(f_i;\theta) - h\|_{L^2}^2 + \|\Psi^\sim(f_i;\theta) - u_0\|_{L^2}^2\right]. \quad (14)$$

In practice, the function data of $\{f_i\}_{i=1}^{N_p}$, $\{u_i\}_{i=1}^{N_p}$ and $\{g, h, u_0\}$ are always measured or evaluated at specific locations in the spatial or spatio-temporal domain. To be concrete, we discretize the domains into three sets of collocations points of $\Omega \times [0,T]$, $\partial\Omega \times [0,T]$ and $\Omega \times \{t = 0\}$ of size s_R, s_B and s_I, respectively. Therefore, we define

$$C_R = \{(\mathbf{x}_i, t_i) \in \Omega \times [0,T]\}_{i=1}^{s_R}, \quad (15)$$

$$C_B = \{(\mathbf{x}_i, t_i) \in \partial\Omega \times [0,T]\}_{i=1}^{s_B}, \quad (16)$$

$$C_I = \{\mathbf{x}_i \in \Omega\}_{i=1}^{s_I}. \quad (17)$$

In this setting, we assume that the data for $\{f_i, u_i\}_{i=1}^{N_p}$ and g are available on C_R (although f_i and u_i are not necessary to be evaluated at the same locations), for h on C_B and for u_0 on C_I so that the loss Eq. (12) and Eq. (14) can be evaluated at the corresponding sites.

With this problem setup, we attempt to give a unified description of operator learning models.

3.2 Encoder-Decoder Framework

In this subsection, we unify all relevant operator learning models within a common framework. Many operator learning models adopt an autoencoder-like structure, where the central idea is to approximate the infinite-dimensional output $u \in \mathcal{U}$ through a latent finite-dimensional representation $u_N \in \mathcal{U}_N$, where $\mathcal{U}_N$ is a learned N-dimensional latent space. Accordingly, the key task is to capture this latent structure of the output space and approximate the mapping from the input function $f \in \mathcal{F}$ to $u_N \in \mathcal{U}_N$. Therefore, the architectures of many operator learning models can be split into two components: (1) an encoder $\mathcal{E}_i : \mathcal{F} \to \mathcal{U}_N$ that maps the input function to its latent representation, and (2) a decoder $\mathcal{D}_o : \mathcal{U}_N \to \mathcal{U}$ that lifts the latent representation back to the full output space. That is, the target operator $\Psi : \mathcal{F} \to \mathcal{U}$ can be approximated as

$$\Psi \approx \mathcal{D}_o \circ \mathcal{E}_i, \quad (18)$$

see Fig. 1 for a schematic plot. This encoder-decoder decomposition is the abstraction of the architectures of many neural operators. It extends the applications of autoencoders to learning mappings between infinite-dimensional function spaces.

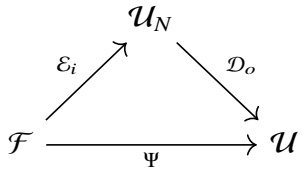

Figure 1: The key workflow of operator learning models. Ψ denotes the solution operator to be approximated. The encoder $\mathcal{E}_i$ maps the input space $\mathcal{F}$ to the latent space $\mathcal{U}_N$, and the decoder $\mathcal{D}_o$ maps $\mathcal{U}_N$ back to the output space $\mathcal{U}$.

Note that operator learning models can only learn the corresponding discrete representations of these operators given discretizations C_R, C_B and C_I. Ideally, the learned discrete operators should be alias-free and preserve *continuous-discrete-equivalence* for robust operator learning, meaning that the continuous solution can be recovered from the discrete output and the discrete output maintains the continuous properties. But we are not going to address this issue in this work, we refer to [2] for more detailed discussion about alias-free operator learning. In the following discussions, we also use these notations for discrete representations of the input encoder and the output decoder.

In this work, we consider four representative operator models for comparison: PCA-Net, DeepONet, FNO and CNO. In the following, we will present the descriptions of how they fit into the Encoder-Decoder framework Eq. (18). For an overview, we refer the reader to Table 1.

Table 1: An overview of the selected models under the Encoder-Decoder framework

Model	Encoder ($\mathcal{E}_i$)	Decoder ($\mathcal{D}_o$)
PCA-Net	PCA + NN	linear
DeepONet	NN	linear
FNO	linear layer + Fourier layers	linear layer
CNO	linear layer + composition of D, R, I, U blocks	linear layer
ReBaNO	A single-layer NN loss minimizer	linear decoder linear, via greedy selected PINNs

* PCA: principal component analysis; NN: neural network

PCA-Net To deliver this paragraph properly, we assume that $\mathcal{F}$ and $\mathcal{U}$ are also generic real separable Hilbert spaces equipped with inner product $(\cdot,\cdot)_{\mathcal{F}}$ and $(\cdot,\cdot)_{\mathcal{U}}$. PCA-Net applies Principal Component Analysis (PCA) onto the data pairs $\{f_i, u_i\}_{i=1}^{N_p}$ evaluated on C_R, generating a reduced basis $\{\phi_i\}_{i=1}^{M}$ for the input space $\mathcal{F}$ and a reduced basis $\{\psi_i\}_{i=1}^{N}$ for the output space $\mathcal{U}$ for some predefined positive integers M and N. Then all

$\{f_i, u_i\}_{i=1}^{N_p}$ are projected onto these bases through

$$\alpha_{ki} \coloneqq (\phi_k, f_i)_{\mathcal{F}},\ k = 1, 2, \ldots, M \tag{19}$$

$$\beta_{li} \coloneqq (\psi_l, u_i)_{\mathcal{U}}, l = 1, 2, \ldots, N \tag{20}$$

for $i = 1, 2, \ldots, N_p$. Then PCA-Net uses a feedforward neural network $\Psi_{\text{NN}}^{\text{PCA}} : \mathbb{R}^M \times \Theta \to \mathbb{R}^N$ to learn the mapping between the PCA coefficients by feeding the data $\{\boldsymbol{\alpha}_i \in \mathbb{R}^M\}_{i=1}^{N_p}$ and $\{\boldsymbol{\beta}_i \in \mathbb{R}^N\}_{i=1}^{N_p}$. Let the operator $\mathcal{R} : \mathcal{F} \to \mathbb{R}^M$ and $\mathcal{S} : \mathbb{R}^N \to \mathcal{U}$ be defined as

$$\mathcal{R}(f) \coloneqq \boldsymbol{\alpha}(f) = \{\langle \phi_k, f \rangle_{\mathcal{F}}\}_{k=1}^{M} \tag{21}$$

$$\mathcal{S}(\boldsymbol{\beta}) \coloneqq \sum_{l=1}^{N} \beta_l \psi_l \tag{22}$$

for any $f \in \mathcal{F}$ and $\boldsymbol{\beta} \in \mathbb{R}^N$. Then, for a given $f \in \mathcal{F}$, the prediction given by PCA-Net is

$$\Psi_{\text{PCA}}^{\sim}(f; \theta) = \sum_{l=1}^{N} \Psi_{\text{NN},l}^{\text{PCA}}(\mathcal{R}f; \theta)\psi_l, \tag{23}$$

where the subscript l in $\Psi_{\text{NN},l}^{\text{PCA}}$ denotes the l-th component. Therefore, the input encoder of PCA-Net is

$$\mathcal{E}_i^{\text{PCA}} = \Psi_{\text{NN}}^{\text{PCA}} \circ \mathcal{R} \tag{24}$$

while the output decoder $\mathcal{D}_o$ is exactly the operator $\mathcal{S}$ defined in Eq. (22).

DeepONet DeepONets consist of two trainable neural networks, a branch net and a trunk net. The branch net encodes the input function, while the trunk net outputs temporally dependent basis functions that span the solution space. In the original DeepONet [46], the branch net Ψ_{NN}^{b} takes as input the discrete evaluations of the input function at fixed locations, $\tilde{f} \coloneqq [f(x_1), f(x_2), \ldots, f(x_M)] \in \mathbb{R}^M$, and maps them to an N-dimensional latent vector $\mathbf{b}(\tilde{f}) = \{b_i(\tilde{f})\}_{i=1}^{N}$ via

$$\Psi_{\text{NN}}^{b} : \mathbb{R}^M \times \Theta \to \mathbb{R}^N. \tag{25}$$

In our implementation, we follow an improved version of the branch net [16, 47], where a PCA transform is first applied to the input function f, and the branch net learns the mapping from the PCA coefficients to the latent representation:

$$\mathbf{b} = \Psi_{\text{NN}}^{\text{PCA}}(\mathcal{R}f; \theta),$$

which is identical to the PCA-Net. The trunk net $\Psi_{\text{NN}}^{t} : \Omega \times \Theta \to \mathbb{R}^N$ outputs the basis functions for reconstructing the solution. If we let $\boldsymbol{\psi}_{\text{NN}}(\cdot; \theta^t) \coloneqq \Psi_{\text{NN}}^{t}(\cdot; \theta^t) : \Omega \times \Theta \to \mathbb{R}^N$, combining the output of the branch net, the prediction given by DeepONet is

$$\Psi_{\text{DON}}^{\sim}(f; \theta)(\mathbf{x}, t) = \sum_{l=1}^{N} \Psi_{\text{NN},l}^{\text{PCA}}(\mathcal{R}f; \theta^b)\psi_{\text{NN},l}(\mathbf{x}, t; \theta^t),\ (\mathbf{x}, t) \in \Omega \times [0, T] \tag{26}$$

where θ is the collection of θ^b and θ^t. This architecture shares the same encoder-decoder structure as PCA-Net.

FNO Contrary to PCA-Net and DeepONet, FNO is a kernel-based approach that leverages integral kernel layers to capture local and nonlocal features. By definition (see Eq. (6) in [36]), the architecture can be divided into three components.

$$\Psi_{\text{FNO}}^{\sim}(f;\theta) = \mathcal{Q} \circ \mathcal{K} \circ \mathcal{P}(f) \tag{27}$$

where the lifting layer $\mathcal{P}$ is a learnable linear layer, defined in a pointwise fashion as $\mathcal{P}(f) \coloneqq W_0 f + b_0$ for some $W_0 \in \mathbb{R}^{d_f' \times d_f}$ and $b_0 \in \mathbb{R}^{d_f'}$. It "lifts" the input function space $\mathcal{F} = \{f : \Omega \to \mathbb{R}^{d_f}\}$ to a higher-dimensional latent feature space $\mathcal{F}_r \coloneqq \{f : \Omega \to \mathbb{R}^{d_f'}\}$ for $d_f' > d_f$. Then the input will go through several consecutive Fourier layers

$$\mathcal{K} = \mathcal{K}_L \circ \mathcal{K}_{L-1} \circ \cdots \circ \mathcal{K}_2 \circ \mathcal{K}_1 \tag{28}$$

for some positive integer L. For $0 < l < L$, a single layer $\mathcal{K}_l : v_l \mapsto v_{l+1}$is pointwise defined as

$$v_{l+1}(y) = \sigma_{l+1}\left(W_{l+1} v_l(y) + \int_\Omega \kappa_l(x-y) v_l(x)\,\mathrm{d}x + b_l\right),\ y \in \Omega \tag{29}$$

where σ is a nonlinear activation function and $\{W_l, b_l, \kappa_l\}_{l=1}^L$ are weights, biases, and parametric kernels. The computation of the convolution integral is carried out in the Fourier domain. Then, the projection layer $\mathcal{Q}$ projects the output v_L back into the solution space $\mathcal{U}$, defined by $u_{\text{FNO}} = W_{L+1} v_L + b_{L+1}$. Therefore, the encoder of FNO is the composition of the lift layer and a set of Fourier layers

$$\mathcal{E}_i^{\text{FNO}} = \mathcal{K} \circ \mathcal{P} \tag{30}$$

while the decoder is the projection layer, i.e.,

$$\mathcal{D}_o^{\text{FNO}} = \mathcal{Q} \tag{31}$$

As such, the parameter θ of FNO is the collection of all learnable neural network parameters, weights and biases $\{W_l, b_l\}_{l=0}^{L+1}$ and parametric kernels $\{\kappa_l\}_{l=1}^L$.

CNO CNO is also a kernel-based neural operator. It can be viewed as a modified U-Net [58]. Similar to FNO, it also includes a lifting layer $\mathcal{P}$ and a projection layer $\mathcal{Q}$. However, the layers inside learn the input encoder $\mathcal{E}_i$ in a very different way. For a given input, it will successively go through downsampling blocks (D blocks), residual blocks (R blocks), invariant blocks (I blocks) and upsampling blocks (U blocks). Such architecture effectively helps the neural network to capture the local features by downsampling and convolutional operations in the downsampling blocks and reconstruct the solution by upsampling and concatenating with the global features through the residual connections (see [57] for more details). Hence, the encoder of CNO is the composition of a lift layer and a set of D, R, I, U blocks and the projection layer works as the decoder.

It is worth mentioning that the input pairs are evaluated on a discrete grid of size s_R points. Therefore, the PCA input encoder Eq. (24) depends on the mesh size s_R, and therefore PCA-Net and DeepONet are not resolution invariant on the input end. In opposite, each layer of FNO works as an operator and the operations

involved are resolution independent, and for inputs with different resolutions, CNO down/upsamples both input and output data to a fixed resolution, so CNO is also invariant with respect to the input/output resolutions.

After the problem setup and framework building, we are ready to introduce ReBaNO in the next section.

4 Reduced Basis Neural Operator (ReBaNO)

We devote this section to a systematic description of the methodology for the proposed neural operator. In contrast to traditional data-driven operator learning models, ReBaNO relies on less high-fidelity data thanks to its judicious greedy strategy in determining which specific high-fidelity data to leverage. Moreover, ReBaNO fine-tunes the model online when a new input function is presented by injecting the corresponding physics as opposed to simply evaluating the trained map. It is these two features that lead to ReBaNO possessing stronger generalizability and mesh invariance.

4.1 The Key Idea of ReBaNO

Inspired by RBM and GPT-PINN, the Reduced Basis Neural Operator adopts a nonlinear physics-based encoder followed by a linear decoder (Fig. 2). This nonlinear-linear pattern was proven to be more theoretically advantageous than other patterns [9]. It builds the mapping in an offline-online fashion, relying on high-fidelity PINN solutions trained on a small number of data instances. In fact, to learn the operator $\Psi : \mathcal{F} \to \mathcal{U}$, ReBaNO iteratively identifies N input-output pairs via a greedy algorithm $\{(f_i(\mathbf{x}, t), u_i(\mathbf{x}, t) = \Psi_h(f_i)(\mathbf{x}, t))\}$ offline. Here, Ψ_h is a high-fidelity approximation of Ψ capable of resolving the input-output map at least at certain input points. ReBaNO builds a rank-n ($1 \le n \le N$) approximation of Ψ in the following fashion: for an arbitrary input function $f \in \mathcal{F}$, this rank-n ReBaNO parameterize the operator $\Psi_n : \mathcal{F} \times \Theta \to \mathcal{U}$ by $\mathbf{c} =: \{c_i\}_{i=1}^n \in \mathbb{R}^n$

$$\Psi_n(f)(\mathbf{x}, t) = \tilde{u}(\mathbf{x}, t; \mathbf{c}) \coloneqq \sum_{i=1}^{n} c_i u_i(\mathbf{x}, t). \tag{32}$$

In essence, ReBaNO adopts a simple one-layer instance-wise network with zero bias, customized activation functions $\{u_i\}_{i=1}^n$, and $\mathbf{c} \coloneqq \{c_i\}_{i=1}^n$ being the trainable weights.

4.2 Physics-Driven Online Tuning

These weights $\{c_i\}_{i=1}^n$ are tuned online by ensuring that the physical law corresponding to the map $f(x, t) \mapsto u(x, t) = \Psi(f)(x, t)$ is satisfied. The ReBaNO solution is sought to satisfy the same PDE Eq. (6) and boundary/initial conditions. The ReBaNO parameter $\mathbf{c}(f)$ is identified per instance by minimizing the loss function Eq. (14) for $N_p = 1$.

We remark that, thanks to the design of offline-online framework and that the activation functions $\{u_i(x, t)\}_{i=1}^n$ are pre-trained and the $\mathbf{c}$-dependence is linear, the relevant derivatives in the loss function can be computed and stored in advance. The consequence is that the weights $\{c_i\}_{i=1}^n$ can be obtained rapidly. In fact, when $\mathcal{N}_f$ in Eq. (6) is a linear operator, no optimization method is needed, as it simplifies to a small least squares problem.

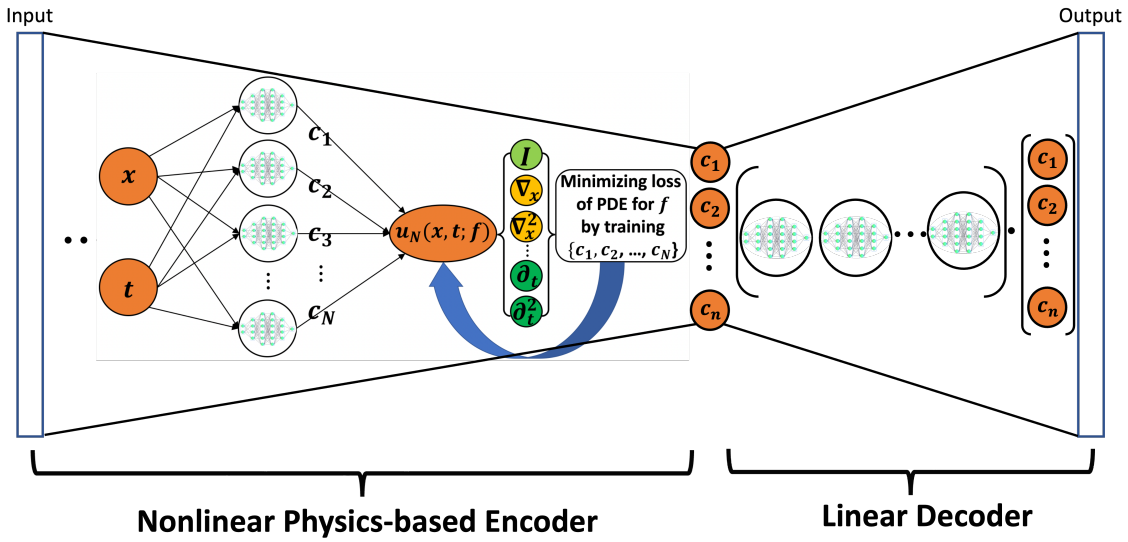

Figure 2: Schematics of ReBaNO. Given an input function, the nonlinear encoder encodes it to $\mathbf{c}$ which then is decoded to the output by a linear operator. Both encoder and decoder are adaptively generated from scratch through an offline training process allowing this minimalist design.

4.3 Knowledge Distillation via Customized Activations

Algorithm 1 ReBaNO: Greedy Knowledge Distillation

Input: Discrete set F for input f, high-fidelity approximate operator Ψ_h (PINN solver), training loss $\mathcal{L}_p$

1: Use the high-fidelity solver to obtain u_1 at a random input f_1, $u_1 = \Psi_h(f_1)$. Precompute quantities necessary for quickly evaluating Eq. (14). Set $n = 2$.
2: **while** *stopping criteria not met,* **do**
3: Train the rank-$(n-1)$ ReBaNO $\Psi_{n-1}(f)$ (i.e., solving the corresponding latent code $\mathbf{c} \in \mathbb{R}^{n-1}$) for all input f in the discretized input set and record the indicator $\Delta_{n-1}(f) := \min_{\mathbf{c}} \mathcal{L}_p[\Psi_{n-1}(f); \mathbf{c}]$.
4: Choose $f_n = \arg\max_{f \in F} \Delta_{n-1}(f)$.
5: Use the high-fidelity solver to obtain u_n at f_n, $u_n = \Psi_h(f_n)$. Precompute quantities necessary for quickly evaluating Eq. (14).
6: Update the ReBaNO network by adding a neuron to the hidden layer to construct $\Psi_n(f)$, and set $n \leftarrow n + 1$.
7: **end while**

Output: Sequence of operators $\{\Psi_n : 1 \le n \le N\}$ approximating the unknown operator Ψ.

ReBaNO features a network-of-networks design with the outer network distilling knowledge from the inner ones. During the offline stage, the outer network grows one neuron at a time self-adaptively by selecting a representative input via a Greedy algorithm. Given a set of samples $F = \{f_i\}_{i=1}^{N_p}$ through which ReBaNO investigates during the training phase, let the set of selected input functions be $\{f_i : f_i \in F\}_{i=1}^{n}$ and their corresponding output (PINN solutions to the PDE) be $\{u_i^{\text{PINN}}\}_{i=1}^{n}$. The ReBaNO network Eq. (32) amounts to adopting these problem-dependent networks $\{u_i^{\text{PINN}}\}_{i=1}^{n}$ as its activation functions. Such an adaptive architecture and the knowledge distillation enable automatic learning of the latent low-dimensional structure of the to-be-learned map.

We end this subsection by describing the iterative and greedy procedure leading to these customized activations. Its main steps are outlined in Algorithm 1. The ReBaNO network adaptively "learns" the input-output relations and "grows" its sole hidden layer one neuron/network at a time in the following fashion. We first randomly select one input instance and obtain the associated (highly accurate) output $u_1(x,t)$. The algorithm then decides how to "grow" itself by scanning the entire discrete input space and, for each case, training this reduced operator Ψ_1 (of 1 hidden layer with 1 neuron). As it scans, it records an error indicator - terminal loss of $\Psi_1(f)$. The next chosen input instance is the one generating the largest error indicator. The algorithm then proceeds by obtaining the high-fidelity output for that input using a PINN solver and therefore grows Ψ_1 to be Ψ_2 featuring two neurons with customized (but pre-trained) activation functions. At every step, we select the input instance that is approximated most badly by the current approximate operator Ψ_n.

4.4 ReBaNO Under the Encoder-Decoder Framework

After introducing the building blocks of ReBaNO, we end this section with the Encoder-Decoder decomposition of ReBaNO. ReBaNO constructs a surrogate output space $\mathcal{U}_N$ spanned by the reduced basis $\{u_i^{\text{PINN}}(\cdot;\theta_i)\}_{i=1}^N$ (here every $\theta_i \in \Theta_{\text{PINN}}$ denotes the set of network parameters of each PINN) by distilling knowledge of the full PINN solutions by a greedy algorithm during offline training. Hence, in the terminology of Section 3, the ReBaNO output decoder $\mathcal{D}_o^{\text{ReBaNO}}$ is learned by physics-driven greedy selection. The output decoder is linear and explicitly defined as the linear combination of the reduced basis

$$\mathcal{D}_o^{\text{ReBaNO}}(\mathbf{c})(\mathbf{x},t) \coloneqq \sum_{i=1}^{N} c_i u_i^{\text{PINN}}(\mathbf{x},t;\theta_i),\ (\mathbf{x},t) \in \Omega \times [0,T] \tag{33}$$

for $\mathbf{c} \in \mathbb{R}^N$. On the other hand, the input functions are mapped to the output latent space through the loss function Eq. (14) in the optimization problem during the online phase. In contrast to aforementioned neural operators in Section 3, ReBaNO directly maps the input function to the latent representation of the solution by the online fine-tuning on the coefficient $\mathbf{c}$ as the parameter of a single-layer neural network that minimizes the physics-informed loss Eq. (14), i.e.,

$$\mathcal{E}_i^{\text{ReBaNO}} : f \mapsto \mathbf{c}, \text{ and } \mathcal{E}_i^{\text{ReBaNO}}(f) \coloneqq \arg\min_{\mathbf{c}\in\mathbb{R}^N} \mathcal{L}_p(\tilde{u};\mathbf{c}) \tag{34}$$

where the loss function is defined on a single input $f \in \mathcal{F}$. In summary, the target operator is approximated by

$$\Psi^{\sim}(f;\theta) = \mathcal{D}_o^{\text{ReBaNO}} \circ \mathcal{E}_i^{\text{ReBaNO}}(f), \tag{35}$$

where each component is defined in Eq. (33) and Eq. (34), and the parameter θ collects the PINN parameters $\{\theta_i \in \Theta_{\text{PINN}}\}_{i=1}^N$ along with the coefficients $\mathbf{c}$.

5 Numerical Results

In this section, we compare the proposed ReBaNO with other data-driven learning approaches (PCA-Net, DeepONet, FNO and CNO) to approximating maps between infinite-dimensional function spaces using three

benchmark problems. In Subsection 5.1, we start with the one-dimensional Poisson equation; Subsection 5.2 studies the map from the permeability field to the solution for 2D steady Darcy flow; and in Subsection 5.3 we consider the Navier-Stokes equation. Because operators are expected to predict the outputs for the inputs in or out of the distribution seen in the training, we perform both in-distribution and out-of-distribution tests in our numerical experiments. The OOD datasets are generated from a similar measure from which the training data are sampled but with smaller correlation lengths. In this section, we focus on the discussion of the more challenging OOD tests. In the following numerical examples, unless otherwise specified, 1000 different input-output pairs are used for training data-driven models and 200 for testing. ReBaNO utilizes the same 1000 inputs as the discrete input data F for greedy selection. We use a relative L^2 loss function for the training of data-driven models. Additional information for the experimental setup is given in Section A. We emphasize here that during the sampling process of all examples, the absolute values of all random numbers are bounded by 4. As a baseline example, to probe the benefit of physics constraints in improving the generalizability, we also contrast ReBaNO with PINO in Poisson's example. Due to the highly complex loss landscapes and the non-convexity of the optimization, PINO errors plateau above 20% when applied to Darcy flow and Navier-Stokes problems. Therefore, the PINO results of Darcy and Navier-Stokes are not included here. All neural networks are trained on a single NVIDIA A100 using 32GB memory. For readers interested in the size and computational cost of each model, we refer to Subsection A.4. The code for all numerical experiments can be found at `https://github.com/haolanzheng/rebano`.

In our numerical experiments, we use L^2 relative error to measure the accuracy of the predictions given by all models

$$e = \frac{\|u_{\text{pred}} - u\|_{L^2}}{\|u\|_{L^2}} \tag{36}$$

where u_{pred} is the prediction and u the high-fidelity solution. Meanwhile, to measure the generalizability, we propose a metric of the ratio of the mean relative error on test datasets to that on training datasets

$$r = \frac{e^{\text{test}}_{\text{mean}}}{e^{\text{train}}_{\text{mean}}} \tag{37}$$

A model has good generalizability if the ratio is close to 1.

For an overview of the performance of all models, the benchmarks of three examples are listed in the Table 2. Based on the results of these benchmark problems, ReBaNO exhibits significantly better generalizability even to both ID and OOD datasets compared to all other data-driven models. More importantly, as shown below, the accuracy of ReBaNO shows strict discretization invariance, while the accuracy of FNO and CNO significantly deteriorates when the grid size deviates from where FNO and CNO are trained.

5.1 1D Poisson Equation

As the first numerical example, we consider 1D Poisson equation define on the interval $\Omega = (0, 1)$

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f(x),\ x \in \Omega \tag{38}$$

$$u(0) = u(1) = 0 \tag{39}$$

Table 2: Poisson (top), Darcy flow (middle), and Navier-Stokes (bottom) benchmarks of all models. Note that PINO results are not included for Darcy and Navier-Stokes due to its errors plateauing above 20% when applied to these two examples.

model	# of params	$e^{\text{train}}_{\text{mean}}$	$e^{\text{train}}_{\text{max}}$	$e^{\text{test}}_{\text{mean}}$		$e^{\text{test}}_{\text{max}}$		$e^{\text{test}}_{\text{mean}}/e^{\text{train}}_{\text{mean}}$		Data
				ID	OOD	ID	OOD	ID	OOD	
PCA-Net	17 670	0.003	0.051	0.011	0.100	0.310	0.767	3.965	35.919	Yes
DeepONet	19 021	0.003	0.028	0.011	0.087	0.451	0.933	3.811	30.511	Yes
FNO	94 657	0.002	**0.010**	0.003	0.015	0.040	0.138	1.564	6.976	Yes
CNO	103 989	0.002	0.011	0.005	0.052	0.134	1.414	1.162	23.941	Yes
PINO	94 657	0.002	0.014	0.002	**0.007**	0.075	0.095	1.199	**4.090**	Yes
ReBaNO	$8 + 901 \times 8$	**0.001**	0.021	**0.001**	0.008	**0.023**	**0.071**	**1.001**	6.975	**No**
PCA-Net	410 733	0.025	0.064	0.057	0.065	0.167	0.157	2.309	2.634	Yes
DeepONet	384 601	0.018	0.038	0.061	0.069	0.165	0.223	3.419	3.899	Yes
FNO	412 929	0.020	0.085	0.032	0.038	0.089	0.083	1.545	1.876	Yes
CNO	359 877	**0.004**	**0.011**	**0.010**	**0.013**	**0.040**	**0.038**	2.489	3.247	Yes
ReBaNO	$48 + 8361 \times 48$	0.045	0.107	0.044	0.048	0.100	0.104	**0.976**	**1.085**	**No**
PCA-Net	86 221	0.052	0.961	0.317	0.859	0.959	2.578	6.097	16.349	Yes
DeepONet	51 601	0.047	0.731	0.379	1.315	1.610	5.514	7.996	27.755	Yes
FNO	58 961	**0.002**	**0.005**	**0.002**	**0.006**	**0.005**	**0.013**	1.018	3.781	Yes
CNO	93 783	0.004	0.008	0.005	0.021	0.008	0.044	1.384	5.690	Yes
ReBaNO	$20 + 2222 \times 20$	0.036	0.069	0.036	0.056	0.072	0.135	**0.996**	**1.569**	**No**

*The last column indicates whether high-fidelity solutions are needed for training the models. The boldface highlights the best performer of each benchmark.

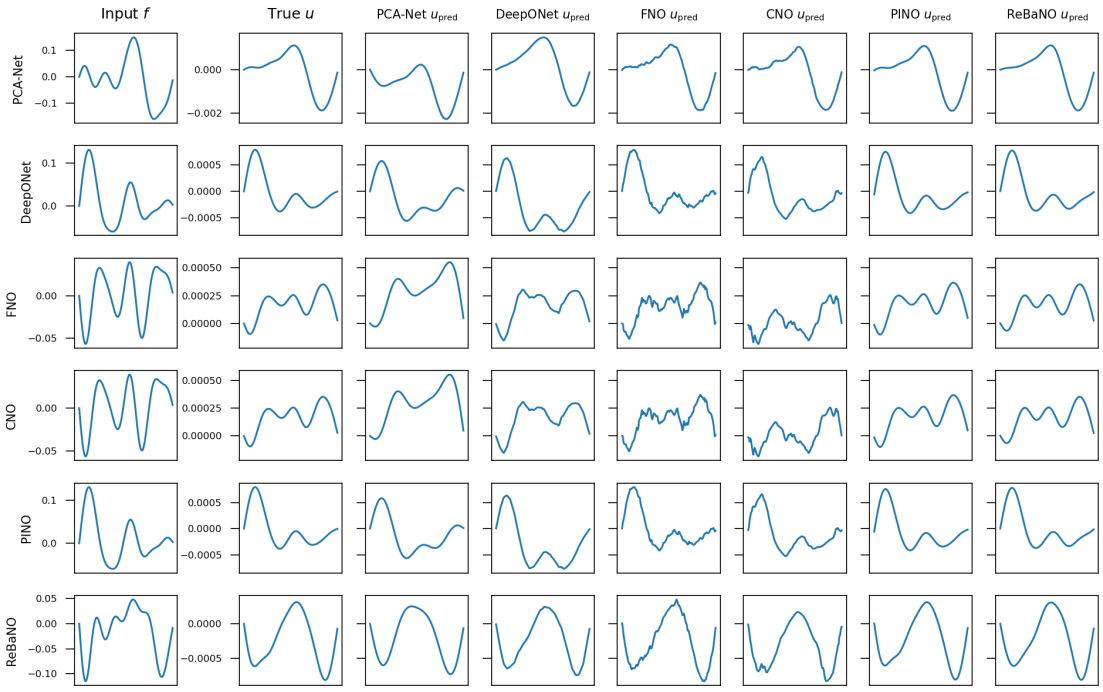

Figure 3: Poisson: inputs and true outputs in the OOD test resulted in worst case errors for each model. PCA-Net is composed of six hidden layers and 64 neurons per layer. DeepONet consists of a branch net with four hidden layers of width 60 and a trunk net with three hidden layers of width 60. FNO is composed of one lifting layer, one projection layer and two Fourier neural layers with 64 neurons for each layer. CNO employs 4 up/down sampling blocks, two residual blocks and another two in the neck. PINO uses the same architecture as the FNO. 8 pre-trained full PINNs are used in the implementation of our ReBaNO.

The data of the source term $f(x)$ are sampled from a centered Gaussian measure $N(0, C)$ with covariance $C = (-\mathrm{d}^2/\mathrm{d}x^2 + 1)^{-2}$, where $-\mathrm{d}^2/\mathrm{d}x^2$ is the Laplacian defined on Ω with homogeneous Dirichlet boundary condition. The data for OOD test are generated from a similar Gaussian with different covariance $C = (-\mathrm{d}^2/\mathrm{d}x^2 + 25)^{-2}$. The solutions to the Poisson equation are obtained analytically. We aim to learn a map from $f(x)$ to the solution $u(x)$: $\Psi : f \mapsto u$.

Accuracy Test Fig. 3 shows the inputs, true outputs, and predicted outputs given by six models for the inputs from the OOD dataset that result in the worst test errors of each model. For this simple 1D problem, the models succeed in capturing the main features of the solutions. PCA-Net and DeepONet give relatively poor accuracy compared to other models. To make a reasonable comparison, we present the pointwise errors in the OOD test on the worst-test-error case of each model in Fig. 4 Top. For example, the first row gives six graphs of pointwise errors given by PCA-Net, DeepONet, FNO, CNO, PINO and ReBaNO (from left to right), respectively of the case that results in the worst test error given by PCA-Net. We can see that, comparing PINO and ReBaNO to FNO, the physical constraint smooths the pointwise error. For the same input, FNO, CNO, PINO, and ReBaNO achieve similarly small test errors compared to PCA-Net and DeepONet.

In Fig. 4 Bottom, we also show the box plot of relative L^2 errors. The numbers printed on the top of each column are the ratios of the mean test errors to the mean training errors provided by all models. The benchmarks of the six models on the Poisson problem are listed in Table 2 (top). From the table and Fig. 4 Bottom, among all purely data-driven models, FNO gives the most consistent level of accuracy in the OOD test. Compared to ID tests, there are wider generalization gaps in all models, while physics-informed models outperform all data-driven models in terms of generalizability. And PCA-Net and DeepONet generalize poorly to OOD data, where the test errors are 30 times more than the training errors. Among all models, PINO achieves the smallest ratio in the OOD test while ReBaNO yields the smallest errors.

Ablation Study on Neuron Selection To see the importance of greedy selection, we randomly sample six neurons (i.e., the set $\{\Psi_h(f_i)\}_{i=1}^6$ for random source terms $\{f_i(x)\}_{i=1}^6$) from the same input data. From Fig. 5 Left, it is clear that with neurons selected using greedy algorithm the largest loss of ReBaNO is more than 10 times smaller than that with randomly selected neurons.

Discretization Invariance Test As discussed in previous work (see Refs. [36, 42]), it is important for a robust operator learning model to be invariant with respect to different discretizations or resolutions. In Fig. 5 Right, we compare the test errors of FNO, CNO, and ReBaNO when varying the grid size on which the input functions are evaluated. Specifically, all models are trained on a grid of size 128. The test inputs for $f(x)$ are sampled from the same Gaussian measure as for the training data, but with different numbers of grid points $s = 32, 50, 64, 100, 128, 200, 256, 400, 512, 800, 1024$. As mentioned in Subsection 3.2, DeepONet and PCA-Net both depend on input resolution. Therefore, we do not compare with DeepONet and PCA-Net. From Fig. 5 Right, we show the mean and maximum relative L^2 error of FNO, CNO, and ReBaNO. We see that the mean and maximum errors given by FNO and CNO increase significantly in super/sub-resolution tests. In the sub-resolution regime, the error is amplified at most more than 10 times, and is increased with a factor of 5 in the super-resolution regime. ReBaNO presents highly consistent accuracy across all resolutions, implying that

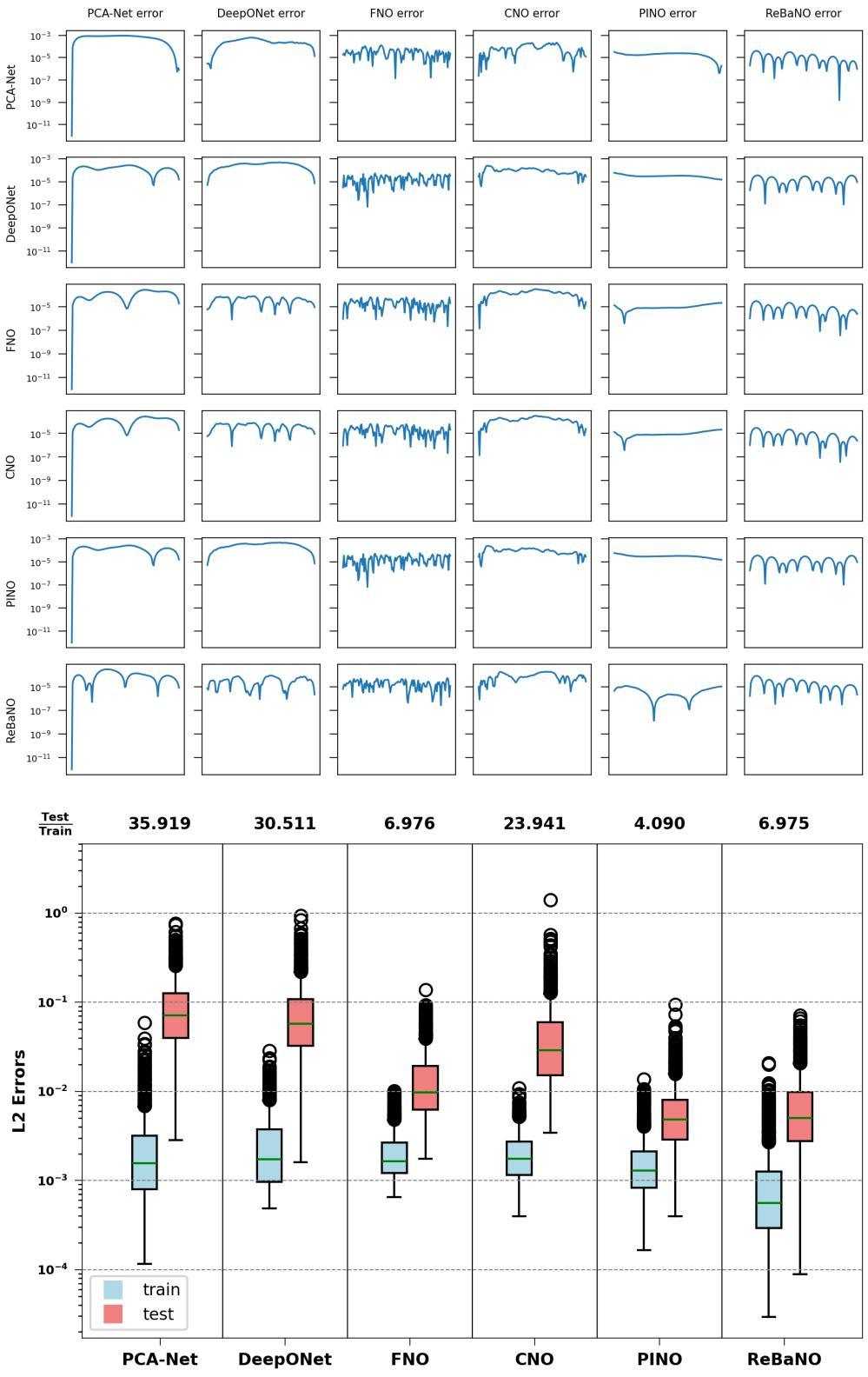

Figure 4: Poisson OOD test pointwise (top) and L^2 relative (bottom) errors. Each subfigure on the top part in each row presents pointwise errors given by the six models for the cases that result in the worst-test-error cases of PCA-Net, DeepONet, FNO, CNO, PINO, and ReBaNO (from top to bottom). Shown on the bottom are the box plots of the L^2 relative errors of the six models. The numbers printed out are the ratios between mean test errors and mean training errors, measuring the generalizability of each model.

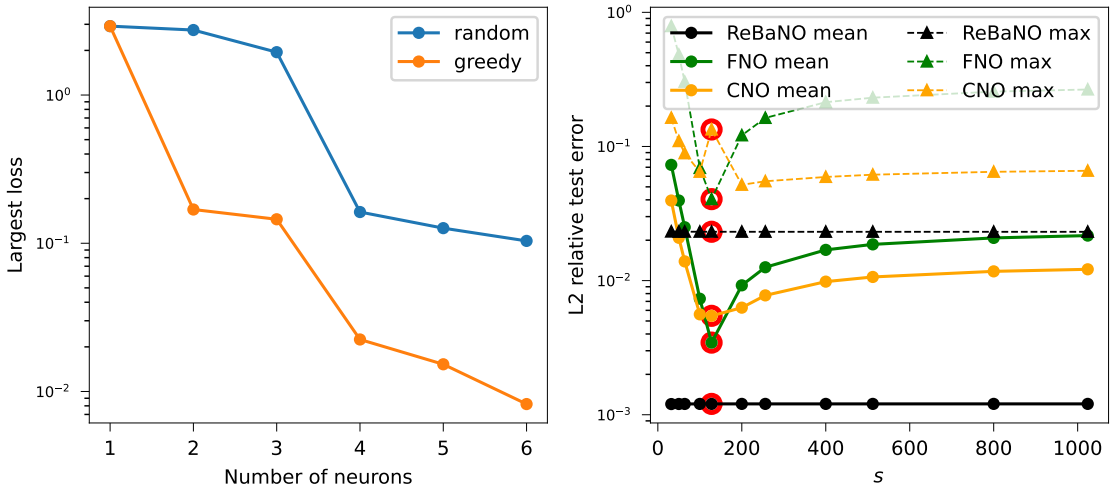

Figure 5: Left: Poisson ablation test on greedy algorithm, showing the largest loss as the number of neurons, selected with or without greedy algorithm, increases. Right: Discretization Invariance Test. Shown are the variations of the mean and max test errors of FNO, CNO and ReBaNO with respect to different discretizations (the number of grid points varies as $s = 32, 50, 64, 100, 128, 200, 256, 400, 512, 800, 1024$). The large red dots denote that all models are trained on the grid of size 128.

our ReBaNO successfully learns a discretized approximator that preserves continuous-discrete-equivalence to the ground-truth operator. In low-resolution regime, the errors of FNO and CNO are dominant by the aliasing error arising from the discretizations of $f(x)$. In the high-resolution regime, the frequency truncation mainly contributes to the errors. Note that the maximum error of the CNO reaches a local maximum at $s = 128$ where it is trained. This is because the CNO is trained with the mean L^2 loss, not the sup-norm, and the CNO up/downsamples the inputs to the fixed input grid, meaning that the CNO may overlook the discrepancies when dealing with coarser inputs, and the errors are also smoothed with higher resolutions. The reason why ReBaNO presents strong discretization invariance is that PINNs learn continuous maps from the physical domain to solution values in a mesh-free manner instead of dealing with mappings between functions, leading to strictly consistent accuracies of the ReBaNO across all discretizations.

5.2 2D Steady Darcy Flow

Next we consider two-dimensional steady-state Darcy flow problem defined on the unit square $\Omega = [0, 1]^2$.

$$-\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \ \mathbf{x} \in \Omega \tag{40}$$

$$u(\mathbf{x}) = 0, \ \mathbf{x} \in \partial\Omega \tag{41}$$

where $a \in L^\infty(\Omega; \mathbb{R}_+)$ is the permeability field. The force term $f(\mathbf{x}) = 1$ is fixed. Following [42], the function $a(\mathbf{x})$ is sampled from the measure $\mu = T_\# N(0, C)$ as the pushforward of a Gaussian measure under the operator T with the covariance $C = (-\Delta + 9)^{-2}$, where $-\Delta$ is the Laplacian defined on Ω and a homogeneous Neumann boundary condition is imposed. The operator T is defined in a piecewise fashion as

$$T(x) = \begin{cases} 12, & x \geq 0 \\ 3, & x < 0. \end{cases} \tag{42}$$

The OOD data are sampled from a similar Gaussian with a covariance $C = (-\Delta + 64)^{-2}$. High-fidelity solutions are computed using FEM over a 100×100 grid. We seek to learn a map from the permeability field $a(\mathbf{x})$ to the solution $u(\mathbf{x})$: $\Psi : a \mapsto u$.

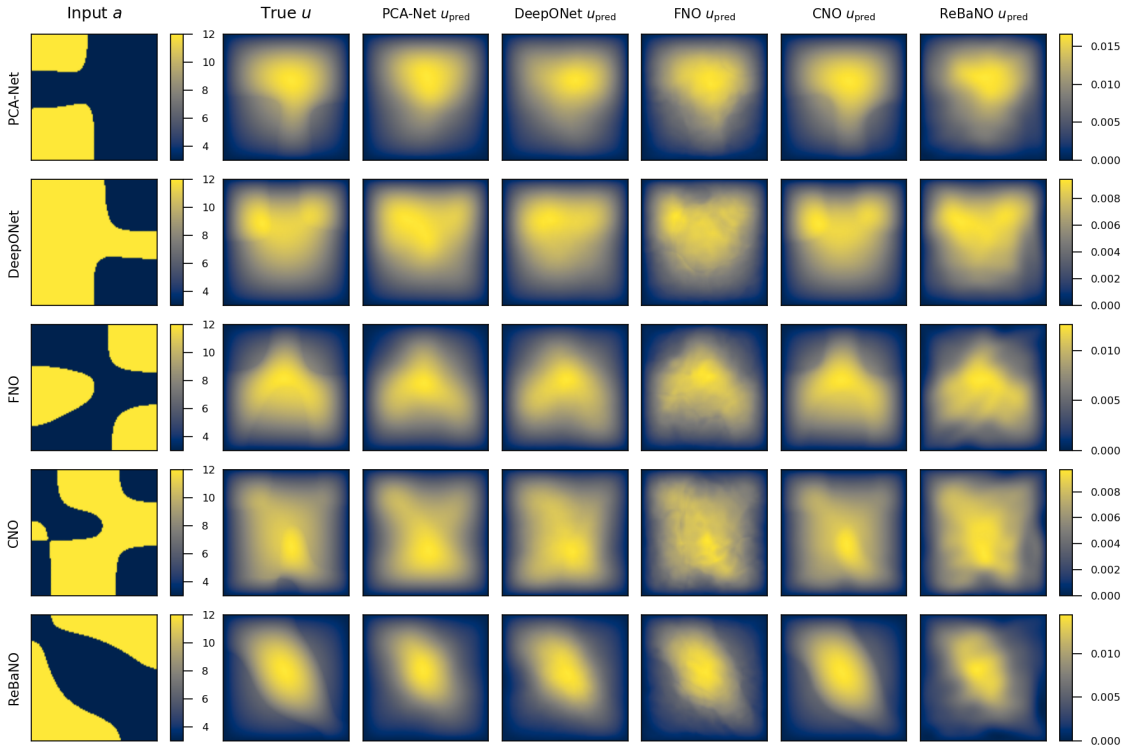

Figure 6: Darcy flow: inputs and true outputs in OOD test that result in worst case errors. PCA-Net is composed of five hidden layers and 200 neurons per layer and DeepONet consists of a branch net with five hidden layers of width 200 and a trunk net with four hidden layers of the same width. FNO is composed of one lifting layer, one projection layer and two Fourier neural layers with 32 neurons for each layer. CNO is composed of four up/down sampling blocks, four residual blocks and two residual blocks in the neck. 48 pre-trained RVPINNs are used in the implementation of our ReBaNO.

Accuracy Test Similarly to the 1D Poisson example, Fig. 6 shows the input, true output, and predicted output given by five models for the inputs from OOD data that result in worst test errors of each model. All models succeed in capturing the main features of the solution, where CNO achieves the best accuracy. From the similar comparison of pointwise errors in worst-test-error cases presented in Fig. 7 Top, we find that FNO predictions exhibit more explicit granularity. The box plot of L^2 relative errors in OOD test is outlined in Fig. 7 Bottom and benchmarks for the Darcy flow problem are listed in Table 2 (middle). Fig. 7 Bottom and Table 2 (middle) also indicate that ReBaNO presents the most comparable performance between the training data and the OOD test data, while the error of all data-driven models increases by a factor of 2 or 3.

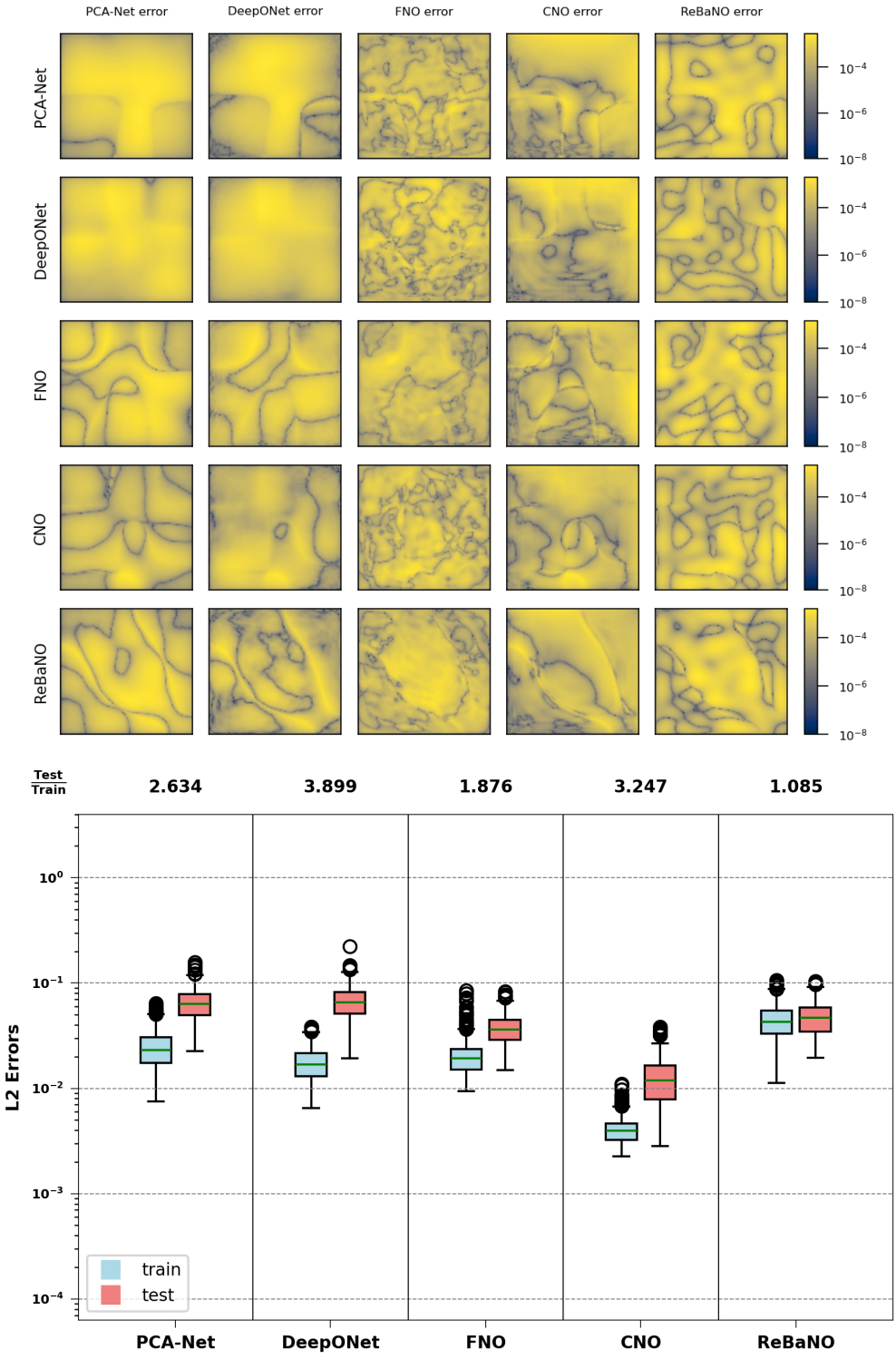

Figure 7: Darcy flow OOD test pointwise (top) and L^2 relative (bottom) errors. Each subfigure on the top part in each row presents pointwise errors given by the five models on the cases that result in the worst-test-error cases of PCA-Net, DeepONet, FNO, CNO and ReBaNO (from top to bottom). Shown on the bottom are the box plots of the L^2 relative errors of the five models. The numbers printed out are the ratios between mean test errors and mean training errors, measuring the generalizability of each model.

5.3 2D Navier-Stokes Equation

In this subsection, we study the two-dimensional Navier-Stokes equation.

$$\frac{\partial \omega}{\partial t} + \mathbf{u} \cdot \nabla \omega = f' + \nu \Delta \omega, \ (\mathbf{x}, t) \in \Omega \times [0, T] \tag{43}$$

$$\omega = -\Delta \psi \tag{44}$$

$$\omega(\mathbf{x}, 0) = \omega_0(\mathbf{x}), \ \mathbf{x} \in \Omega \tag{45}$$

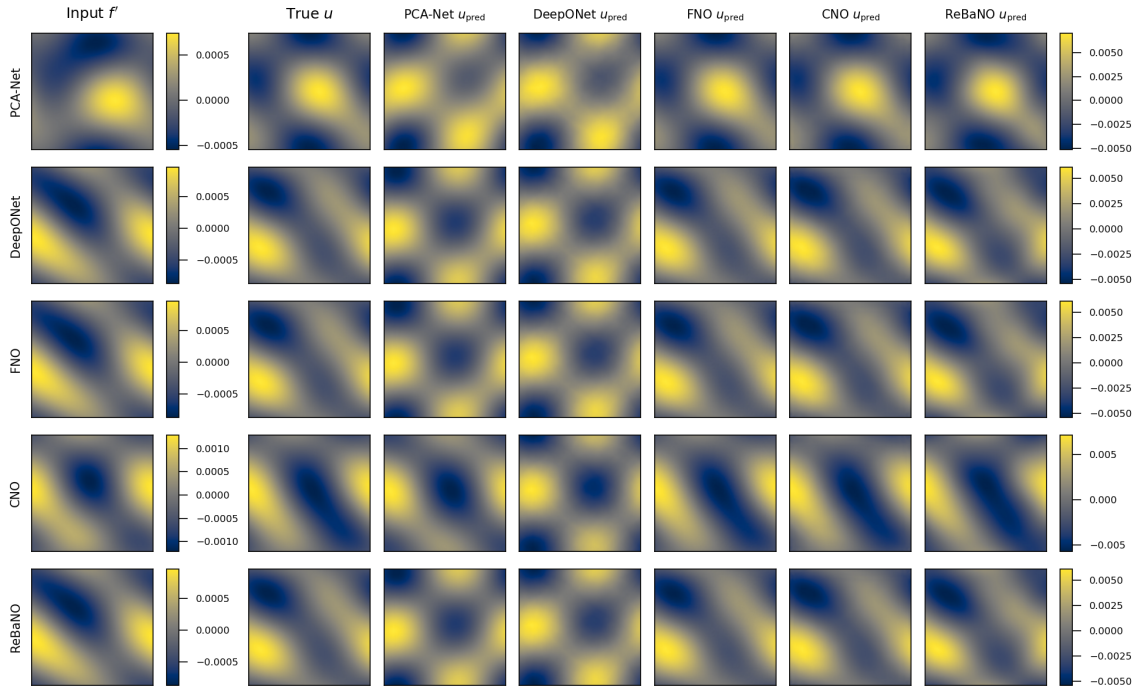

Figure 8: Navier-Stokes: inputs and true outputs of the vorticity resulted in worst case test errors. PCA-Net is composed of four layers and 200 neurons per layer and DeepONet consists of a branch net with four layers of width 100 and a trunk net with three layers of the same width. FNO is composed of one lifting layer, one projection layer and two Fourier neural layers with 18 neurons for each layer. CNO uses four up/down sampling blocks and four residual blocks inside along with two residual blocks in the neck. 20 pre-trained PINNs are used in the implementation of our ReBaNO.

Here we consider a square domain $\Omega = [0, 2\pi]^2$ with periodic boundary conditions, where $\mathbf{u}$ is the physical velocity field and it is related to ψ by $\mathbf{u} = (\partial_y \psi, -\partial_x \psi)$. And $f'(\mathbf{x})$ is the curl of the external force and ν the viscosity constant. We aim to learn the map from f' to the vorticity field ω at a fixed moment $T = 10$: $\Psi : f'(\mathbf{x}) \mapsto \omega(\mathbf{x}, T)$. We sample the data of f' from a centered Gaussian with covariance $C = (-\Delta + 9)^{-4}$, where the Laplacian $-\Delta$ is defined on Ω subject to a periodic boundary condition. The initial condition ω_0 is fixed and sampled from the same measure. OOD data are samples from a similar measure with $C = (-\Delta + 25)^{-4}$. Note that Eqs. (43) to (45) and periodic boundary conditions can only determine the stream function ψ up to an arbitrary additive constant. We can impose an additional condition to eliminate arbitrariness. In this work, we

enforce ψ to be a mean-free function over the spatial domain Ω:

$$\int_{\Omega} \psi = 0 \tag{46}$$

We use pseudo-spectral method with a tiny time step size ($\Delta t = 1/5000$) to solve the Navier-Stokes equation over a 100×100 grid.

Accuracy Test In Fig. 8, it shows the input, true vorticity and predicted vorticity given by the five models for the inputs from OOD data that lead to the worst test errors of each model. A similar pointwise error comparison plot is in Fig. 9 Top. For this problem, the vorticity field is well-correlated with the external force. DeepONet performs worst on its worst-test-error case.

The box plot of L^2 relative errors for vorticity in the OOD test is plotted in Fig. 9 Bottom and the corresponding benchmarks are listed in Table 2 (bottom). PCA-Net and DeepONet show markedly lower accuracy and generalizability than the other models on OOD inputs. When extrapolating to OOD data, FNO and CNO errors increase by a factor of 3 and 5, respectively, whereas ReBaNO error increases only by approximately 50%.

6 Conclusion

This work advances a reduced basis-driven physics-informed operator learning approach. We have introduced a data-lean operator learning algorithm, the Reduced Basis Neural Operator (ReBaNO). Guided by a mathematically rigorous greedy algorithm and with the notion of embedding physics, ReBaNO leverages knowledge distillation via task-specific activation function. After an offline-training stage which relies on minimal amount of data, ReBaNO's online solver is extremely lightweight, with a network consisting of only one hidden layer in which each neuron is a full pre-trained PINN. Tests against four state-of-the-art operator learning models reveal the fact that ReBaNO is the only NO achieving full discretization invariance, and ReBaNO's capability in significantly shrinking the generalization gap for both in-distribution and out-of-distribution tests. Future work includes theoretical convergence rate analysis and augmentation toward robust OOD performance when boundary conditions are concerned. Both are extremely challenging topics in the model reduction and scientific machine learning communities.

References

[1] Anima Anandkumar, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Nikola Kovachki, Zongyi Li, Burigede Liu, and Andrew Stuart. Neural Operator: Graph Kernel Network for Partial Differential Equations. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[2] Francesca Bartolucci, Emmanuel de Bezenac, Bogdan Raonic, Roberto Molinaro, Siddhartha Mishra, and Rima Alaifari. Representation Equivalent Neural Operators: A Framework for Alias-free Operator Learning. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023.

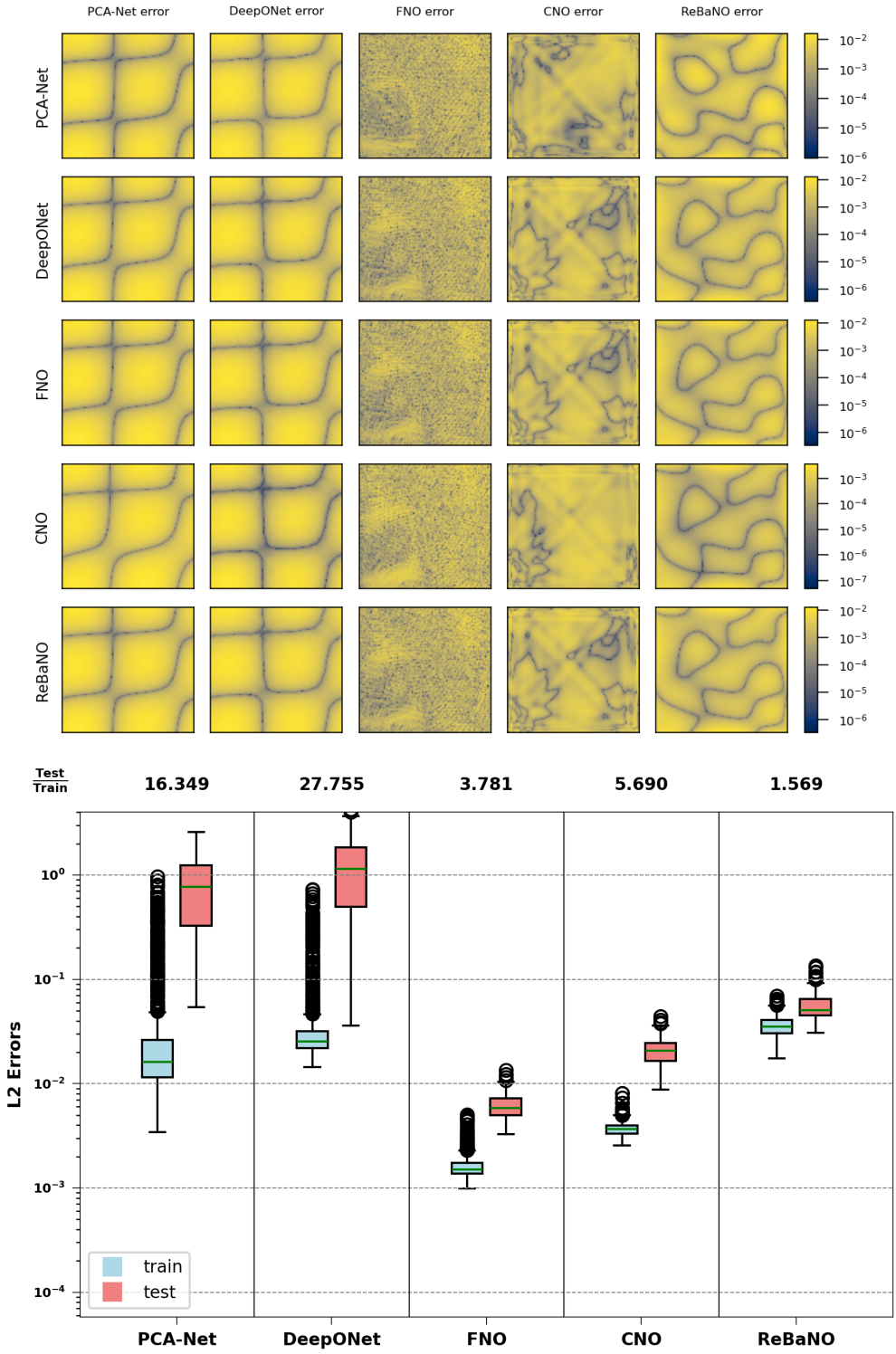

Figure 9: Navier-Stokes OOD test pointwise (top) and L^2 relative (bottom) errors. Each subfigure on the top part in each row presents pointwise errors given by the models for the cases that result in the worst-test-error cases of PCA-Net, DeepONet, FNO, CNO and ReBaNO (from top to bottom). Shown on the bottom are the box plots of the L2 relative errors of the five models. The numbers printed out are the ratios between mean L^2 relative test errors and mean L^2 relative train errors, measuring the generalizability of each model.

[3] P Benner, S Gugercin, and K Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, jan 2015.

[4] Gilles Besnard, François Hild, and Stéphane Roux. "finite-element" displacement fields analysis from digital images: application to portevin–le châtelier bands. *Experimental mechanics*, 46(6):789–803, 2006.

[5] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B. Kovachki, and Andrew M. Stuart. Model Reduction And Neural Networks For Parametric PDEs. *The SMAI Journal of computational mathematics*, 7:121–157, 2021.

[6] P Binev, Albert Cohen, W Dahmen, R Devore, G Petrova, and P Wojtaszczyk. Convergence rates for greedy algorithms in reduced basis methods. *SIAM Journal on Mathematical Analysis*, 43(3):1457–1472, 2011.

[7] Andrea Bonito, Ronald DeVore, Guergana Petrova, and Jonathan W. Siegel. Convergence and error control of consistent PINNs for elliptic PDEs. *arXiv preprint arXiv:2406.09217*, 2025.

[8] Susanne C Brenner. *The mathematical theory of finite element methods*. Springer, 2008.

[9] Patrick Buchfink, Silke Glas, Bernard Haasdonk, and Benjamin Unger. Model reduction on manifolds: A differential geometric framework. *Physica D: Nonlinear Phenomena*, 468:134299, 2024.

[10] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mechanica Sinica*, pages 1–12, 2022.

[11] Shuhao Cao. Choose a transformer: Fourier or galerkin. *Advances in neural information processing systems*, 34:24924–24940, 2021.

[12] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4):045002, 2019.

[13] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE transactions on neural networks*, 6:911–917, 1995.

[14] Yanlai Chen, Yajie Ji, Akil Narayan, and Zhenli Xu. TGPT-PINN: Nonlinear model reduction with transformed GPT-PINNs. *Computer Methods in Applied Mechanics and Engineering*, 430:117198, 2024.

[15] Yanlai Chen and Shawn Koohy. GPT-PINN: Generative pre-trained physics-informed neural networks toward non-intrusive meta-learning of parametric pdes. *Finite Elements in Analysis and Design*, 228:104047, 2024.

[16] Maarten V. de Hoop, Daniel Zhengyu Huang, Elizabeth Qian, and Andrew M. Stuart. The Cost-Accuracy Trade-Off in Operator Learning with Neural Networks. *Journal of Machine Learning*, 1:299–341, 2022.

[17] Beichuan Deng, Yeonjong Shin, Lu Lu, Zhongqiang Zhang, and George Em Karniadakis. Approximation rates of deeponets for learning operators arising from advection–diffusion equations. *Neural Networks*, 153:411–426, 2022.

[18] Shaan Desai, Marios Mattheakis, Hayden Joy, Pavlos Protopapas, and Stephen Roberts. One-shot transfer learning of physics-informed neural networks. *arXiv preprint arXiv:2110.11286*, 2021.

[19] Weinan E, Jiequn Han, and Linfeng Zhang. Machine-learning-assisted modeling. *Physics Today*, 74(7):36–41, 2021.

[20] Olga Fuks and Hamdi A Tchelepi. Limitations of physics informed machine learning for nonlinear two-phase transport in porous media. *Journal of Machine Learning for Modeling and Computing*, 1(1), 2020.

[21] J Ghaboussi, JH Garrett Jr, and Xiping Wu. Knowledge-based modeling of material behavior with neural networks. *Journal of engineering mechanics*, 117(1):132–153, 1991.

[22] Jamshid Ghaboussi, David A Pecknold, Mingfu Zhang, and Rami M Haj-Ali. Autoprogressive training of neural network constitutive models. *International Journal for Numerical Methods in Engineering*, 42(1):105–126, 1998.

[23] Vignesh Gopakumar, Stanislas Pamela, and Debasmita Samaddar. Loss landscape engineering via data regulation on pinns. *Machine Learning with Applications*, 12:100464, 2023.

[24] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[25] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. Physics-informed deep neural operator networks. In *Machine Learning in Modeling and Simulation: Methods and Applications*, pages 219–254. Springer, 2023.

[26] Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. Multiwavelet-based operator learning for differential equations. *Advances in neural information processing systems*, 34:24048–24062, 2021.

[27] Bernard Haasdonk. *Chapter 2: Reduced Basis Methods for Parametrized PDEs: A Tutorial Introduction for Stationary and Instationary Problems*, pages 65–136.

[28] Jiequn Han, Xu-Hui Zhou, and Heng Xiao. An equivariant neural operator for developing nonlocal tensorial constitutive models. *Journal of Computational Physics*, 488:112243, 2023.

[29] Qizhi He, Devin W Laurence, Chung-Hao Lee, and Jiun-Shyan Chen. Manifold learning based data-driven modeling for soft biological tissues. *Journal of Biomechanics*, 117:110124, 2021.

[30] Jan S. Hesthaven, Gianluigi Rozza, and Benjamin Stamm. *Certified reduced basis methods for parametrized partial differential equations*. SpringerBriefs in Mathematics. Springer, Cham; BCAM Basque Center for Applied Mathematics, Bilbao, 2016. BCAM SpringerBriefs.

[31] Siavash Jafarzadeh, Stewart Silling, Ning Liu, Zhongqiang Zhang, and Yue Yu. Peridynamic neural operators: A data-driven nonlocal constitutive model for complex material responses. *Computer Methods in Applied Mechanics and Engineering*, 425:116914, 2024.

[32] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[33] Michail Koumpanakis and Ricardo Vilalta. Meta-learning loss functions of parametric partial differential equations using physics-informed neural networks. In *International Conference on Discovery Science*, pages 183–197. Springer, 2024.

[34] Nikola Kovachki, Samuel Lanthaler, and Siddhartha Mishra. On universal approximation and error bounds for fourier neural operators. *Journal of Machine Learning Research*, 22(290):1–76, 2021.

[35] Nikola B. Kovachki, Samuel Lanthaler, and Andrew M. Stuart. Chapter 9 - Operator learning: Algorithms and analysis. In Siddhartha Mishra and Alex Townsend, editors, *Handbook of Numerical Analysis*, volume 25 of *Numerical Analysis Meets Machine Learning*, pages 419–467. Elsevier, 2024.

[36] Nikola B Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew M Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces with applications to PDEs. *Journal of Machine Learning Research*, 24(89):1–97, 2023.

[37] K Kunisch and S Volkwein. Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics. *SIAM J. Numer. Anal.*, 40(2):492—-515 (electronic), 2002.

[38] Samuel Lanthaler. Operator learning with PCA-Net: upper and lower complexity bounds. *Journal of Machine Learning Research*, 24(318):1–67, 2023.

[39] Doksoo Lee, Wei Chen, Liwei Wang, Yu-Chin Chan, and Wei Chen. Data-driven design for metamaterials and multiscale systems: A review. *Advanced Materials*, 36(8):2305254, 2024.

[40] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*. SIAM, 2007.

[41] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Andrew Stuart, Kaushik Bhattacharya, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. *Advances in Neural Information Processing Systems*, 33:NeurIPS 2020, 2020.

[42] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[43] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-Informed Neural Operator for Learning Partial Differential Equations. *ACM / IMS J. Data Sci.*, 1:9:1–9:27, 2024.

[44] Ning Liu, Siavash Jafarzadeh, and Yue Yu. Domain agnostic fourier neural operators. *Advances in Neural Information Processing Systems*, 36, 2024.

[45] Ning Liu, Xuxiao Li, Manoj R. Rajanna, Edward W. Reutzel, Brady Sawyer, Prahalada Rao, Jim Lua, Nam Phan, and Yue Yu. Deep neural operator enabled digital twin modeling for additive manufacturing. *Advances in Computational Science and Engineering*, 2:174–201, Sun Sep 01 00:00:00 UTC 2024.

[46] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[47] Lu Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

[48] Huakun Luo, Haixu Wu, Hang Zhou, Lanxiang Xing, Yichen Di, Jianmin Wang, and Mingsheng Long. Transolver++: An Accurate Neural Solver for PDEs on Million-Scale Geometries. In *Forty-Second International Conference on Machine Learning*, 2025.

[49] Yvon Maday and Einar M Rønquist. A reduced-basis element method. *Journal of scientific computing*, 17:447–459, 2002.

[50] Nick McGreivy and Ammar Hakim. Weak baselines and reporting biases lead to overoptimism in machine learning for fluid-related partial differential equations. *Nature Machine Intelligence*, pages 1–14, 2024.

[51] Yong Zheng Ong, Zuowei Shen, and Haizhao Yang. Integral Autoencoder Network for Discretization-Invariant Learning. *Journal of Machine Learning Research*, 23:1–45, 2022.

[52] David Pfau, James S Spencer, Alexander GDG Matthews, and W Matthew C Foulkes. Ab initio solution of the many-electron schrödinger equation with deep neural networks. *Physical Review Research*, 2(3):033429, 2020.

[53] Konstantinos Prantikos, Stylianos Chatzidakis, Lefteri H Tsoukalas, and Alexander Heifetz. Physics-informed neural network with transfer learning (tl-pinn) based on domain similarity measure for prediction of nuclear reactor transients. *Scientific Reports*, 13(1):16840, 2023.

[54] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. *Reduced Basis Methods for Partial Differential Equations*, volume 92 of *UNITEXT*. Springer International Publishing, Cham, 2016.

[55] Maziar Raissi, Paris Perdikaris, and George 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*, 378:686–707, 2019.

[56] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[57] Bogdan Raonic, Roberto Molinaro, Tim De Ryck, Tobias Rohner, Francesca Bartolucci, Rima Alaifari, Siddhartha Mishra, and Emmanuel de Bezenac. Convolutional Neural Operators for robust and accurate learning of PDEs. In *Thirty-Seventh Conference on Neural Information Processing Systems*, 2023.

[58] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing.

[59] Jonthan D Smith, Zachary E Ross, Kamyar Azizzadenesheli, and Jack B Muir. Hyposvi: Hypocentre inversion with stein variational inference and physics informed neural networks. *Geophysical Journal International*, 228(1):698–710, 2022.

[60] Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized Fourier Neural Operators. In *The Eleventh International Conference on Learning Representations*, 2022.

[61] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[62] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

[63] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[64] Gege Wen, Zongyi Li, Kamyar Azizzadenesheli, Anima Anandkumar, and Sally M. Benson. U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances in Water Resources*, 163:104180, 2022.

[65] Haixu Wu, Huakun Luo, Haowen Wang, Jianmin Wang, and Mingsheng Long. Transolver: A fast transformer solver for PDEs on general geometries. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *ICML'24*, pages 53681–53705, Vienna, Austria, 2024. JMLR.org.

[66] Chen Xu, Ba Trung Cao, Yong Yuan, and Günther Meschke. Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Computer Methods in Applied Mechanics and Engineering*, 405:115852, 2023.

[67] Huaiqian You, Yue Yu, Marta D'Elia, Tian Gao, and Stewart Silling. Nonlocal kernel network (NKN): A stable and resolution-independent deep neural network. *Journal of Computational Physics*, page arXiv preprint arXiv:2201.02217, 2022.

[68] Huaiqian You, Quinn Zhang, Colton J Ross, Chung-Hao Lee, and Yue Yu. Learning deep implicit fourier neural operators (ifnos) with applications to heterogeneous material modeling. *Computer Methods in Applied Mechanics and Engineering*, 398:115296, 2022.

[69] Linfeng Zhang, Jiequn Han, Han Wang, Roberto Car, and Weinan E. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Physical Review Letters*, 120(14):143001, 2018.

[70] Lu Zhang, Huaiqian You, Tian Gao, Mo Yu, Chung-Hao Lee, and Yue Yu. MetaNO: How to transfer your knowledge on learning hidden physics. *Computer Methods in Applied Mechanics and Engineering*, 417:116280, 2023.

[71] Jianwei Zheng, Ni Xu, Junwei Zhu, Xiaoqin Zhang, et al. Alias-free mamba neural operator. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

A Experimental setup

A.1 Poisson Setup

The full PINNs are trained on grid of size 128 uniformly discretized over $\Omega = [0, 1]$ through 40000 epochs with a fixed learning rate of 0.0005. The architecture of each PINN is $[1, 20, 20, 20, 1]$ and the activation function is tanh. And 100 epochs are used for both training and testing for fine tuning with a fixed learning rate of 0.8 using L-BFGS optimizer. Unless otherwise specified, the optimizer in all training and testing processes of ReBaNO is Adam. ReBaNO leverages 8 pre-trained neurons for this problem. As for data-driven models, PCA-Net has six hidden layers of width 64 and DeepONet consists of a branch net with four hidden layers of width 60 and a trunk net with three hidden layers of the same width. FNO is composed of two Fourier neural layers along with one lifting layer and one projection layer with 64 neurons per layer, and the maximum number of Fourier modes is set to be 10. PINO takes the same architecture as FNO. CNO consists of four down/up sampling blocks, two residual blocks in the middle, and two residual blocks in the neck. The channel multiplier is 8. All data-driven models are trained using Adam optimizer through 1000 epochs with an initial learning rate of 0.001 which is halved every 100 epochs. PINO is trained through 10000 epochs with the same learning rate halved every 1000 epochs. The activation function ReLU is employed. 1000 instances of $f(x)$ are used for testing in all models.

A.2 Darcy flow Setup

For Darcy flow, Robust Variational PINN learns the solution in a more appropriate manner because it leverages the weak-form loss function. We train RVPINN using 8×8 elements with 20×20 Gauss quadrature points for each element through 60000 epochs with an initial learning rate of 0.001 and it is halved after every 10000 epochs. The test functions we use are the tensor product of one-dimensional FEM piecewise linear functions. In the greedy search, the fine-tuning is carried over 8×8 elements with 16×16 Gauss quadrature points for each using the L-BFGS optimizer through 100 epochs for both training and testing with a fixed learning rate of 0.8. The architecture of each RVPINN is $[2, 40, 40, 40, 40, 40, 40, 1]$ and the activation function is sin. 48 neurons are trained for Darcy flow. Meanwhile, PCA-Net has five hidden layers of width 200 and DeepONet consists of a branch net with five hidden layers of width 200 and a trunk net with four hidden layers of the same width. FNO is composed of two Fourier neural layers along with one lifting layer and one projection layer

with 32 neurons per layer, and the maximum number of Fourier modes is set to be 10. CNO consists of four down/up sampling blocks, four residual blocks in the middle, and two residual blocks in the neck. The channel multiplier is 8. All data-driven models are trained using Adam optimizer through 1000 epochs with an initial learning rate of 0.001 which is halved every 100 epochs.

A.3 Navier-Stokes Setup

The full PINNs for representative solutions of NS equation are trained through 40000 epochs with an initial learning rate of 0.005 halved after every 5000 epochs on a 100×100 grid with 100 time steps. The architecture of each PINN is $[3, 20, 20, 20, 20, 20, 20, 2]$ which outputs both vorticity and stream field, and the activation function is cos and 5000 epochs are used during the offline and the online stage of fine tuning, with a fixed learning rate of 0.005, on the same mesh. 20 pre-trained neurons are developed for ReBaNO. In addition, PCA-Net is composed of four layers and 200 neurons per layer and DeepONet consists of a branch net with four layers of width 100 and a trunk net with three layers with the same width. FNO is composed of two Fourier neural layers with 20 neurons for each layer along with one lifting layer and one projection layer. The maximum Fourier modes used is set to be 6. CNO consists of four down/up sampling blocks, two residual blocks in the middle, and two residual blocks in the neck. The channel multiplier is 4. All data-driven models are trained using Adam optimizer through 1000 epochs with an initial learning rate of 0.001 which is halved every 100 epochs.

A.4 Model size and computational cost

Table 3: Size and computational cost of each model for Poisson (top), Darcy flow (middle), and Navier-Stokes (bottom).

model	# of params	train time	infer time/N_{test}
PCA-Net	17 670	111.098 s	0.163 ms
DeepONet	19 021	132.788 s	0.230 ms
FNO	94 657	761.609 s	1.235 ms
CNO	103 989	504.188 s	2.441 ms
PINO	94 657	9808.844 s	3.698 ms
ReBaNO	$8 + 901 \times 8$	979.533 s	0.399 s
PCA-Net	410 733	116.913 s	0.132 ms
DeepONet	384 601	179.098 s	0.511 ms
FNO	412 929	1105.024 s	3.111 ms
CNO	359 877	2672.568 s	4.222 ms
ReBaNO	$48 + 8361 \times 48$	6.328 h	1.940 s
PCA-Net	86 221	94.039 s	0.112 ms
DeepONet	51 601	149.216 s	0.189 ms
FNO	58 961	287.531 s	0.569 ms
CNO	93 783	2232.984 s	4.919 ms
ReBaNO	$20 + 2222 \times 20$	40.774 h	14.092 s

*Training data generation time for the three equations are 0.211 ms/case, 15.405 s/case, and 7.797 s/case respectively. PINN training time for Poisson, Darcy flow, and Navier-Stokes are 0.029 h/case, 0.101 h/case, and 1.952 h/case respectively.