Numer. Math. Theor. Meth. Appl.
doi: 10.4208/nmtma.OA-2024-0124

Vol. **18**, No. 1, pp. 31-65
February 2025

# R-Adaptive DeepONet: Learning Solution Operators for PDEs with Discontinuous Solutions Using an R-Adaptive Strategy

Yameng Zhu[1], Jingrun Chen[2] and Weibing Deng[1,*]

[1] *School of Mathematics, Nanjing University, Nanjing 210093, P.R. China*
[2] *School of Mathematical Sciences and Suzhou Institute for Advanced Research, University of Science and Technology of China, Suzhou 215127, P.R. China*

Received 24 October 2024; Accepted (in revised version) 7 January 2025

**Abstract.** When DeepONet approximates solution operators of partial differential equations (PDEs) with discontinuous solutions, it poses a foundational approximation lower bound due to its linear reconstruction property. Inspired by the moving mesh method, we propose an R-adaptive DeepONet method, which consists of: (1) the output data representation is transformed from the physical domain to the computational domain using the equidistribution principle; (2) the maps from input parameters to the solution and the coordinate transformation function over the computational domain are learned using DeepONets separately; (3) the solution over the physical domain is obtained via post-processing methods such as the interpolation method. Additionally, we introduce a solution-dependent weighting strategy in the training process to reduce the error. We establish an upper bound for the reconstruction error based on piecewise linear interpolation and show that the introduced R-adaptive DeepONet can reduce this bound. Moreover, for two prototypical PDEs with sharp gradients or discontinuities, we prove that the approximation error decays at a superlinear rate with respect to the trunk basis size, unlike the linear decay observed in vanilla DeepONets. Numerical experiments on several PDEs with discontinuous solutions are conducted to verify the advantages of the R-adaptive DeepONet over available variants of DeepONet.

**AMS subject classifications**: 47-08, 47H99, 65D15, 65M50, 68Q32, 68T05, 68T07

**Key words**: Scientific machine learning, neural operators, DeepONet, R-adaptive method.

## 1. Introduction

Many interesting phenomena in physics and engineering are described by partial differential equations whose solutions contain sharp gradient regions or discontinuities.

*Corresponding author. *Email addresses:* dg21210022@smail.nju.edu.cn (Y. Zhu), jingrunchen@ustc.edu.cn (J. Chen), wbdeng@nju.edu.cn (W. Deng)

The most common types of such PDEs are hyperbolic systems of conservation laws [10], such as Euler equations, inviscid Burgers' equation, etc. It is well-known that solutions of these PDEs develop finite-time discontinuities such as shock waves, even when the initial and boundary data are smooth. Other examples include convection-dominated equations, reaction-diffusion equations, and so on. It is challenging for traditional numerical methods because resolving these discontinuities, such as shock waves and contact discontinuities, requires petite grid sizes. Moreover, characterizing geometric structures, especially in terms of effectively suppressing numerical oscillations near discontinuous interfaces and maintaining the steepness of transition interfaces, is difficult. Specialized numerical methods such as adaptive finite element methods [1] and discontinuous Galerkin finite element methods [12] have been successfully used in this context, but their high computational cost limits their wide use.

At the same time, data-driven approaches are becoming a competitive and viable means for solving these challenging problems. Deep neural networks (DNNs) have shown promising potential for solving both forward and inverse problems associated with PDEs [3]. Numerous researchers have explored methods that utilize DNNs for solving PDEs (see [13, 21] and references therein).

Machine learning for PDEs primarily focuses on learning solutions by training a mapping from the computational domain to the solution. This process, known as the solution parameterization, encompasses techniques such as the deep Ritz method [14], deep Galerkin method [34], and physics informed neural networks (PINNs) [5, 32]. These methods utilize DNNs to represent the solution and integrate the PDE information into the loss function. The approximate solution is obtained by minimizing the loss function. Since proposed, these methods have been successfully applied to solve both forward and inverse problems for various linear and nonlinear PDEs [5, 18, 30]. Note that these approaches are tailored to specific instances of PDEs. Consequently, if the coefficients or initial conditions associated with the PDEs change, the model has to be retrained, resulting in poor generalization ability across different PDEs.

Along another line, there is ongoing work on parameterizing the solution map using DNNs, referred to as operator learning [2, 8, 17, 20, 26, 28]. In [8], Chen and Chen introduced a novel learning architecture based on neural networks, termed operator networks, and demonstrated that these operator networks possess an astonishing universal approximation property for infinite-dimensional nonlinear operators. Recently, the authors of [28] replaced the shallow branch and trunk networks in operator networks with DNNs and proposed the deep operator network (DeepONet). Since proposed, it has been successfully applied to a variety of problems with differential equations [6, 9, 27, 39]. In [26], Li *et al.* proposed Fourier neural operators based on a nonlinear generalization of the kernel integral representation for some operators and makes use of the convolutional or Fourier network structure.

Although DeepONets have demonstrated good performance across diverse applications, some studies have pointed out that DeepONets fail to efficiently approximate solution operators of PDEs with sharp gradients or discontinuities [22, 24]. In [22], the authors gave a fundamental lower bound on the approximation error of DeepONets and

show that there are fundamental barriers to the expressive power of operator learning methods based on linear reconstruction. This is of particular relevance for problems in which the optimal lower bound exhibits a slow decay in terms of the number of basis functions $n$, due to the slow decay of the eigenvalues of the covariance operator. To reduce the approximation error, the resolution must be high enough, i.e., we need a large $n$. However, this may lead to a dramatic increase in computing costs. Therefore, a method with a small approximation error and moderate computational cost is highly desirable.

Variants of DeepONets have been developed to overcome this limitation. Hadorn [15] investigated the behavior of DeepONet to understand the challenges in detecting sharp features in the target function when the number of basis $n$ is small. They proposed Shift-DeepONet, which adds two neural networks to shift and scale the input function. Venturi and Casey [37] analyzed the limitations of DeepONet using singular value decomposition and proposed a flexible DeepONet (flexDeepONet) by adding a pre-net and an additional output in the branch net. Seidman *et al.* [33] introduced a nonlinear manifold decoder (NOMAD) framework, utilizing a neural network that takes the output of the branch net as input along with the query location. Recently, Lee *et al.* [25] proposed a HyperDeepONet, which leverages the expressive power of hypernetworks to learn complex operators with a smaller set of parameters. These methods address the limitations of linear reconstruction by modifying the structure of DeepONet, allowing the trunk basis to incorporate information about the input parameters.

Traditional numerical methods, such as finite difference method and finite element method (FEM), rely on the linear reconstruction using a linear space of basis functions over a predefined mesh to approximate the solution. For solutions with sharp gradients or discontinuities, a fine mesh is needed to resolve local singularities which may lead to significant computational time and data storage. Therefore, researchers have introduced the moving mesh (R-adaptive) method to adaptively and automatically optimize and adjust mesh configurations based on solution characteristics, see [19, 35] and references therein. The core concept involves adjusting grid distribution through strategic methods without altering the number of mesh grids and their topological connections. This process ensures grids concentrate in regions where solution variations are pronounced. Consequently, this adaptive approach enhances numerical simulation accuracy without increasing computational costs.

To overcome the limitation of linear reconstruction in DeepONet, in this study, we propose a new framework inspired by the moving mesh method, called R-adaptive DeepONet. It employs different learning strategies while maintaining the vanilla structure in the original DeepONet. To this end, we introduce a solution-dependent coordinate transformation from the physical domain to the computational domain. The transformed coordinates are then used as the input to the trunk net, similar to traditional R-adaptive methods. This enables adaptive adjustment of basis functions in DeepONet based on the property of the output solution. Specifically, we first transform the representation of the output data from the physical domain to the computational

domain using the equidistribution principle. This yields two output datasets: the coordinate transform function and the solution over the computational domain. Second, we use two DNN models to learn the maps from the input parameters to the coordinate transform function and the solution over the computational domain separately. We emphasize that while learning the forward coordinate transformation from the physical to the computational domain can ensure the injectivity, it retains the singularity of the original solution which is difficult to learn. Therefore, we propose an alternative approach using inverse coordinate transform learning. Although the inverse coordinate transformation does not guarantee a bijection, the functions over the output domain are smoother, making it easier to learn. Given the choice of the inverse coordinate transform, directly predicting the solution value for a given arbitrary coordinate becomes impractical. Thus, we finally recover the solution using post-processing methods such as the (linear) interpolation method. It is worth mentioning that, according to the error analysis of the operator composition, we introduce two novel solution-related weights to the training process of each component.

We establish an upper bound for the reconstruction error using piecewise linear interpolation and demonstrate that our proposed R-adaptive DeepONet can reduce this bound. Additionally, we rigorously prove that R-adaptive DeepONet can efficiently approximate the prototypical PDEs with sharp gradients or discontinuities. Specifically, the approximation error decays at a superlinear rate with respect to the trunk basis size, while the vanilla DeepONet exhibits at best the linear decay rate [22].

To illustrate the effectiveness of our approach, we compare the performance of several DeepONet models for the linear advection equation, the Burgers' equation with low viscosity, and the compressible Euler equations of gas dynamics. The results consistently demonstrate that our R-adaptive DeepONet outperforms vanilla DeepONet and competes effectively with Shift DeepONet.

The remainder of this paper is structured as follows. In Section 2, we give a brief introduction to DeepONet. And then discuss the details of the R-adaptive DeepONet in Section 3. In Section 4, we show some theoretical results. In Section 5, we present some numerical results. Finally, some conclusions and comments are given.

## 2. Operator learning and DeepONet

### 2.1. Problem setting

The goal of operator learning is to learn a mapping from one infinite-dimensional function space to another by using a finite collection of observations of input-output pairs from this mapping. We formalize this problem as follows. Let $\mathcal{X}$ and $\mathcal{Y}$ be two Banach spaces of functions defined on bounded domains $D_{\mathcal{X}} \subset \mathbb{R}^{d_{\mathcal{X}}}, D_{\mathcal{Y}} \subset \mathbb{R}^{d_{\mathcal{Y}}}$ respectively and $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ be a (typically) non-linear map. Suppose we have observations $\{a^{(i)}, u^{(i)}\}_{i=1}^{N}$ where $a^{(i)} \sim \mu$ are i.i.d. samples drawn from some probability measure $\mu$ supported on $\mathcal{X}$ and $u^{(i)} = \mathcal{G}(a^{(i)})$. We aim to build an approximation of $\mathcal{G}$ by constructing a parametric map $\mathcal{G}_\theta : \mathcal{X} \to \mathcal{Y}$ with parameters $\theta \in \mathbb{R}^{\text{para}}$ such that $\mathcal{G}_\theta \approx \mathcal{G}$.

Sometimes the input function space $\mathcal{X}$ can be parameterized by a finite dimensional vector space $\bar{\mathcal{X}}$. Thus, the original objective operator $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ can also be equivalently expressed as $\bar{\mathcal{G}} : \bar{\mathcal{X}} \to \mathcal{Y}$. For example, if we consider the mapping from the initial density, velocity, and pressure $(\rho_0, u_0, p_0)$ to the energy $E$ at some time $T$ in the sod shock tube problem, we can parameterize the initial data by the left and right states $(\rho_L, u_L, p_L), (\rho_R, u_R, p_R)$ and the location of the initial discontinuity $x_0$. In this case, the input function space is equivalent to a 7-dimensional vector space. For convenience, we still write $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ instead of distinguishing between $\mathcal{G}$ and $\bar{\mathcal{G}}$.

We are interested in controlling the error of the approximation of the average for $\mu$. In particular, assuming $\mathcal{G}$ is $\mu$-measurable, we aim to control the $L^2_\mu(\mathcal{X};\mathcal{Y})$ Bochner norm of the approximation as follows:

$$\|\mathcal{G} - \mathcal{G}_\theta\|_{L^2_\mu(\mathcal{X};\mathcal{Y})} := \mathbb{E}_{a\sim\mu}\|\mathcal{G}(a) - \mathcal{G}_\theta(a)\|^2_{\mathcal{Y}} = \int_{\mathcal{X}} \|\mathcal{G}(a) - \mathcal{G}_\theta(a)\|^2_{\mathcal{Y}} \, \mathrm{d}\mu(a). \tag{2.1}$$

## 2.2. A brief introduction to DeepONet

DeepONets [28] present a specialized deep learning architecture for operator learning that encapsulates the universal approximation theorem for operators [8]. Here we provide a brief introduction to the effective application of DeepONets for learning operators.

To construct a DeepONet, we first need to encode the input parameter function. In [28], the authors use a fixed collection of training sensors $\{x_1, x_2, \ldots, x_m\} \subset D_{\mathcal{X}}$ to encode the input function $a$ by the point values $\mathcal{E}(a) := \mathcal{E}(a(x_1), a(x_2), \ldots, a(x_m))$ in $\mathbb{R}^m$. As we mentioned before, sometimes the input function space $\mathcal{X}$ contains a finite-dimensional parameterization and we can encode $a \in \mathcal{X}$ by this parameterization directly. DeepONet is formulated in terms of two neural networks:

(1) Branch-net $\boldsymbol{\beta}$: it maps the point values $\mathcal{E}(a)$ to coefficients

$$\boldsymbol{\beta}\big(\mathcal{E}(a)\big) = \big(\beta_1(\mathcal{E}(a)), \ldots, \beta_n(\mathcal{E}(a))\big),$$

resulting in a mapping

$$\boldsymbol{\beta} : \mathbb{R}^m \;\to\; \mathbb{R}^n, \quad \mathcal{E}(a) \;\mapsto\; \boldsymbol{\beta}\big(\mathcal{E}(a)\big). \tag{2.2}$$

(2) Trunk-net $\boldsymbol{\tau}(y) = (\tau_1(y), \ldots, \tau_n(y))$: it is used to define a mapping

$$\boldsymbol{\tau} : D_{\mathcal{Y}} \;\to\; \mathbb{R}^n, \quad y \;\mapsto\; \boldsymbol{\tau}(y). \tag{2.3}$$

While the branch net provides the coefficients, the trunk net provides the "basis" functions in an expansion of the output function of the form

$$\mathcal{G}^{\mathrm{DON}}(a)(y) = \sum_{k=1}^{n} \beta_k(a)\tau_k(y), \quad a \in \mathcal{X}, \quad y \in D_{\mathcal{Y}}$$

with $\beta_k(a) = \beta_k(\mathcal{E}(a))$. The resulting mapping $\mathcal{G}^{\mathrm{DON}} : \mathcal{X} \to \mathcal{Y}$, $a \mapsto \mathcal{G}^{\mathrm{DON}}(a)$ is referred to as the vanilla DeepONet.

**Limitation of DeepONet.** Although DeepONets have been proven to be universal within the class of measurable operators [22], a fundamental lower bound on the approximation error has also been identified.

**Theorem 2.1** (Lanthaler *et al.* [22, Theorem 3.4])**.** *Let $\mathcal{X}$ be a separable Banach space, $\mathcal{Y}$ a separable Hilbert space, and let $\mu$ be a probability measure on $\mathcal{X}$. Let $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$ be a Borel measurable operator with $\mathbb{E}_{a\sim\mu}[\|\mathcal{G}(a)\|_{\mathcal{Y}}^2] < \infty$. Then the following lower approximation bound holds for any DeepONet $\mathcal{N}^{DON}$ with trunk-/branch-net dimension $n$:*

$$E(\mathcal{N}^{DON}) := \mathbb{E}_{a\sim\mu}\Big[\|\mathcal{N}^{DON}(a) - \mathcal{G}(a)\|_{\mathcal{Y}}^2\Big]^{1/2} \geq \mathcal{E}_{opt} =: \sqrt{\sum_{j>n} \lambda_j}, \tag{2.4}$$

*where the optimal error $\mathcal{E}_{opt}$ is written in terms of the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots$ of the covariance operator $\Gamma_{\mathcal{G}_{\#\mu}} := \mathbb{E}_{u\sim\mathcal{G}_{\#\mu}}[(u \otimes u)]$ of the push-forward measure $\mathcal{G}_{\#\mu}$.*

The same lower bound applies to any operator approximation of the form $\mathcal{N}(a) = \sum_{k=1}^n \beta_k(a)\tau_k$, where $\beta_k : \mathcal{X} \to \mathbb{R}$ are arbitrary functionals. This bound, for example, also holds for the PCA-Net architecture discussed in [2, 17]. In [23], the authors referred to any operator learning architecture of this form as a method with "linear reconstruction", since the output function $\mathcal{N}(a)$ is restricted to the linear $n$-dimensional space spanned by the $\tau_1, \ldots, \tau_n \in \mathcal{Y}$.

When the eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots$ of the covariance operator $\Gamma_{\mathcal{G}_{\#\mu}}$ decay slowly, approximation using DeepONet may become inaccurate. For instance, the solution operators of advection PDEs and the Burgers' equation are challenging to approximate accurately when using DeepONet with a small number of basis functions $n$ (see [23]).

## 2.3. Variant models of DeepONet

Several variants of DeepONet have been developed to overcome its limitations. Hadorn [15] proposed the Shift-DeepONet. The main idea is that a scale net $\boldsymbol{A} = (A_k)_{k=1}^n$,

$$\boldsymbol{A} : \mathbb{R}^m \;\to\; \mathbb{R}^{n\times d_{\mathcal{Y}}\times d_{\mathcal{Y}}}, \quad \mathcal{E}(a) \;\mapsto\; \boldsymbol{A}(a) := \big(A_1(a), A_2(a), \ldots, A_n(a)\big),$$

where $A_k(a)$ is matrix-valued functions, and a shift net $\boldsymbol{\gamma} = (\gamma_k)_{k=1}^n$, with

$$\gamma_k : \mathbb{R}^m \to \mathbb{R}^{n\times d_{\mathcal{Y}}}, \qquad \mathcal{E}(a) \mapsto \boldsymbol{\gamma}(a) := \big(\gamma_1(a), \gamma_2(a), \ldots, \gamma_n(a)\big)$$

to scale and shift the input query position $y$, while retaining the DeepONet branch- and trunk-nets $\boldsymbol{\beta}, \boldsymbol{\tau}$ defined in (2.2) and (2.3), respectively. The Shift-DeepONet $\mathcal{N}^{\text{sDON}}$ is an operator of the form

$$\mathcal{N}^{\text{sDON}}(a)(y) = \sum_{k=1}^n \beta_k(a)\tau_k\big(A_k(a)\cdot y + \gamma_k(a)\big).$$

This approach incorporates the information of the input parameter function $a$ into the trunk basis, allowing the Shift-DeepONet to overcome the limitations of linear reconstruction.

Similar to the Shift-DeepONet, Venturi & Casey proposed the flexible DeepONet (flexDeepONet) [37], using the additional network, pre-net, to give the bias between the input layer and the first hidden layer, thus introducing the information of $a$ to the trunk basis. NOMAD, developed by Seidman *et al.* [33], devised a nonlinear output manifold using a neural network that takes the output of the branch net $\{\beta_i\}_{i=1}^n$ and the query location $y$, to overcome the limitation of vanilla DeepONet. Lee *et al.* [25] went a step further. They used a hypernetwork to share the information of input $a$ to all parameters of the trunk network and proposed a general model HyperDeepONet.

All these methods incorporate information from the input function $a$ into the trunk basis to overcome the limitation of linear reconstruction. In practical performance, they do not differ significantly. To validate the effectiveness of our proposed method, we use Shift DeepONet as a representative among these models to compare with the R-adaptive DeepONet in this paper.

## 3. Proposed methodology: R-adaptive DeepONet

### 3.1. R-adaptive DeepONet

Many traditional numerical methods rely on linear reconstruction and encounter similar limitations when facing local singularities. R-adaptive methods, also known as moving mesh methods, effectively alleviate these issues. In R-adaptive computations, the number of basis functions remains fixed, but they dynamically adjust based on the problem characteristics. This adaptation reduces errors without significantly increasing computational costs. In Appendix A, we provide a brief introduction to the R-adaptive method and its associated equidistribution principle.

Inspired by the R-adaptive method, we propose a new learning strategy based on DeepONet for operator learning of PDEs with local singularity, termed R-adaptive DeepONet.

Formally, given $\mathcal{G} : \mathcal{X} \to \mathcal{Y}$, $a \mapsto u(y)$, we introduce a homeomorphism $\tilde{y} = y(\xi) : D_{\mathcal{Y}} \to D_{\mathcal{Y}}, \xi \mapsto y(\xi)$, which maps the computational domain to the physics domain. This allows us to divide the original operator into two new operators as follows:

$$\mathcal{T} : a \;\mapsto\; \tilde{y}(\xi) \quad \text{and} \quad \tilde{\mathcal{G}} : a \;\mapsto\; \tilde{u}(\xi) = u\big(\tilde{y}(\xi)\big),$$

where $\mathcal{T}$ maps $a$ to the coordinate transform function $\tilde{y} = y(\xi)$, and $\tilde{\mathcal{G}}$ defines the map to the solution in the computational domain. The original object operator to be learned can be represented as

$$\mathcal{G}(a)(y) = \tilde{\mathcal{G}}(a) \circ \big(\mathcal{T}(a)\big)^{-1}(y), \tag{3.1}$$

where $(\mathcal{T}(a))^{-1} : y \mapsto \xi(y)$ represents the inverse function of $\tilde{y} = y(\xi)$. Since the objective operator $\mathcal{G}(a)$ can be represented by these two operators, we can use two

independent DeepONets to learn these two operators as follows:

$$\mathcal{T}_{\theta_T} \approx \mathcal{T} : a \ \mapsto \ y(\xi) \quad \text{and} \quad \tilde{\mathcal{G}}_{\theta_G} \approx \tilde{\mathcal{G}} : a \ \mapsto \ \tilde{u}(\xi),$$

where $\theta_T$ and $\theta_G$ represent the parameters of the two models, respectively. For clarity, we will refer to $\mathcal{T}$ as the adaptive coordinate operator and $\tilde{\mathcal{G}}$ as the adaptive solution operator. The corresponding $\mathcal{T}_{\theta_T}$ and $\tilde{\mathcal{G}}_{\theta_G}$ are termed the adaptive coordinate and adaptive solution DeepONets respectively. Together, the pair $\{\mathcal{T}_{\theta_T}, \tilde{\mathcal{G}}_{\theta_G}\}$ is then called an R-adaptive DeepONet system.

Since our approach is data-driven, generating appropriate training data for the models $\mathcal{T}_{\theta_T}$ and $\tilde{\mathcal{G}}_{\theta_G}$ using the equidistribution principle is crucial. Given observations $\{a^{(i)}, u^{(i)}\}_{i=1}^N$, we first preprocess the sampled data. This involves determining the corresponding coordinate transform function $y^{(i)}(\xi)$ for each target function $u^{(i)}(y)$ and obtaining the solution on the computational domain $\tilde{u}^{(i)}(\xi)$. As a result, we generate training data sets $\{a^{(i)}, \{y^{(i)}(\xi_j)\}\}_{i=1}^N$ and $\{a^{(i)}, \{\tilde{u}^{(i)}(\xi_j)\}\}_{i=1}^N$ for the two independent models, respectively. In this step, we use the mesh generator proposed by Ceniceros and Hou [7]. Other mesh generation methods can be found in [19]. We emphasize that for problems with discontinuous solutions, the R-adaptive DeepONet needs smaller training datasets than other DeepONets since the output functions $y(\xi)$ and $\tilde{u}(\xi)$ are both smooth, which allow sparser sampling data to capture most features of them.

Here, we choose to learn the mapping from $a$ to $y(\xi)$ instead of $\xi(y)$, since the coordinate transform $y \mapsto \xi(y)$ retains the singularity of the output function, while the inverse $\xi \mapsto y(\xi)$ is relatively smooth, and thus is easier to learn. Our goal is to obtain the output function in terms of $y$, but the prediction process yields two functions in terms of $\xi$. To determine the value of $u$ at $y$, we must first find the corresponding $\xi$ and use it as the input of the learned $\tilde{\mathcal{G}}(a)$ to predict the function value. However, due to the black-box nature of neural networks, deducing the input $\xi$ from output $y$ is challenging. Consequently, post-processing is necessary to make accurate predictions. After training, we have two independent models mapping the input $a$ to two functions of $\xi$. Given a fixed $a$ and $\xi \in D_y$, we can get a pair $\{y(\xi), \tilde{u}(\xi)\}$, which forms a mesh grid in the graph of $u = \mathcal{G}(a)$. By using a uniform mesh $\{\xi_j\}$ as the input of the trunk net, we generate a set of points $\{y(\xi_j), \tilde{u}(\xi_j)\}$ that provides a discrete representation of $u$. These points are densely distributed in places where $u$ has singularity, and sparsely distributed in places where $u$ is smooth, hence effectively capturing the function $u$. Using these discrete points, we can reconstruct the output function $u$ by the local interpolation method.

## 3.2. Training settings

In Section 2.1, we set the target to minimize the $L^2_\mu(\mathcal{X};\mathcal{Y})$ Bochner norm of the approximation (see (2.1)). In our model, if we assume that the adaptive coordinate operator $\mathcal{T}$ is known and only consider learning the mapping $\tilde{\mathcal{G}}_{\theta_G}$, the corresponding approximation error $E_{\tilde{\mathcal{G}}}$ can be written as follows:

$$E_{\tilde{\mathcal{G}}} = \left\| \mathcal{G} - \tilde{\mathcal{G}}_{\theta_G} \circ \mathcal{T}^{-1} \right\|_{L^2_\mu(\mathcal{X};\mathcal{Y})}$$

$$
\begin{aligned}
&= \mathbb{E}_{a\sim\mu}\big\|\mathcal{G}(a) - \tilde{\mathcal{G}}_{\theta_G}(a)((\mathcal{T}(a))^{-1}\,\big\|_{\mathcal{Y}}^2 \\
&= \mathbb{E}_{a\sim\mu}\int_{D_{\mathcal{Y}}} \big|\mathcal{G}(a)(y) - \tilde{\mathcal{G}}_{\theta_G}(a)((\mathcal{T}(a))^{-1}(y))\big|^2 \mathrm{d}y \\
&= \mathbb{E}_{a\sim\mu}\int_{D_{\mathcal{Y}}} \big|\tilde{\mathcal{G}}(a)(\xi) - \tilde{\mathcal{G}}_{\theta_G}(a)(\xi)\big|^2 \big|\det(J(\mathcal{T}(a)(\xi)))\big| \mathrm{d}\xi.
\end{aligned}
$$

Therefore, in the loss function, we naturally introduce the weight $|\det(J(\mathcal{T}(a)(\xi)))|$. To prevent this weight from being zero or too large, we modify it to

$$
w_{\tilde{\mathcal{G}}}(a,\xi) := \min\left\{M, \sqrt{1+|\det(J(\mathcal{T}(a)(\xi)))|^2}\right\}, \tag{3.2}
$$

where $M$ is the upper bound we set for this weight, and $|\det(J(\mathcal{T}(a)(\xi)))|$ for weight computing is obtained from the data pre-processing. Therefore, in the training process, we aim to minimize the weight empirical loss function

$$
\mathcal{L}_{\tilde{\mathcal{G}}} := \frac{1}{N_1 \times N_2}\sum_{k=1}^{N_1}\sum_{j=1}^{N_2} \big|\tilde{u}_k(\xi_j) - \tilde{\mathcal{G}}_{\theta_G}(a_k)(\xi_j)\big|^2 w_{\tilde{\mathcal{G}}}(a_k,\xi_j), \tag{3.3}
$$

where $N_1$ denotes the number of sampled inputs $a_k$, and $N_2$ denotes the number of sensors $\xi_j$. Generally, according to the equidistribution principle, $w_{\tilde{\mathcal{G}}}(a,\xi)$ is relatively small in places where $u$ has singularities. This weighting ensures that the model training is more concentrated over the areas where $u$ is smooth.

In parallel, we can write the approximation error of $\mathcal{T}$ as

$$
\begin{aligned}
E_{\mathcal{T}} &= \big\|\mathcal{G} - \tilde{\mathcal{G}}\circ\mathcal{T}_{\theta_T}^{-1}\big\|_{L^2_\mu(\mathcal{X};\mathcal{Y})} \\
&= \mathbb{E}_{a\sim\mu}\int_{D_{\mathcal{Y}}} \big|\mathcal{G}(a)(y) - \mathcal{G}(a)(\mathcal{T}_{\theta_T}(a)\circ(\mathcal{T}(a))^{-1}(y))\big|^2 \mathrm{d}y \\
&= \mathbb{E}_{a\sim\mu}\int_{D_{\mathcal{Y}}} \big|\mathcal{G}(a)(y) - \mathcal{G}(a)(\mathcal{T}_{\theta_T}(a)(\xi))\big|^2 \big|\det(J(\mathcal{T}(a)(\xi)))\big| \mathrm{d}\xi \\
&\approx \mathbb{E}_{a\sim\mu}\int_{D_{\mathcal{Y}}} \big|\nabla\mathcal{G}(a)\big|^2 \big|y - \mathcal{T}_{\theta_T}(a)(\xi)\big|^2 \big|\det(J(\mathcal{T}(a)(\xi)))\big| \mathrm{d}\xi.
\end{aligned}
$$

So the corresponding weight can be chosen as

$$
w_{\mathcal{T}}(a,\xi) := \min\left\{\bar{M}, \sqrt{1+|\nabla\mathcal{G}(a)|^4|\det(J(\mathcal{T}(a)(\xi)))|^2}\right\}, \tag{3.4}
$$

where $\bar{M}$ is the upper bound we set for this weight. The density function is usually of the form $\rho = \sqrt{1+\beta|\nabla u|^2}$, where $\beta$ is a constant. According to (A.1), we can see that $|\det(J(\mathcal{T}(a)(\xi)))|$ is inversely proportional to $\rho$. So here we can see that $w_{\mathcal{T}}(a,\xi)$ computed according to (3.4) has the opposite performance to $w_{\tilde{\mathcal{G}}}(a,\xi)$. $w_{\mathcal{T}}(a,\xi)$ is small in places where $u$ is smooth and large in places where $u$ has singularities.

Moreover, for convenience and accuracy of post-processing, a well-structured mesh is crucial. The coordinate transform functions learned by DeepONet do not inherently

guarantee untangling. To prevent mesh tangling, it is essential to ensure that the Jacobian determinant of the transformation function $y(\xi)$ satisfies $\det(J(\mathcal{T}_{\theta_T}(a)(\xi))) > 0$. Therefore, we incorporate a regularization term into the loss function of the coordinate learning process. The modified loss function becomes

$$\mathcal{L}_{\mathcal{T}} := \frac{1}{N_1 \times N_2} \sum_{k=1}^{N_1} \sum_{j=1}^{N_2} \Big[ \alpha_1 |\tilde{y}_k(\xi_j) - \mathcal{T}_{\theta_T}(a_k)(\xi_j)|^2 w_{\mathcal{T}}(a_k, \xi_j) + \alpha_2 \mathrm{ReLU}^2 \left( -\det(J(\mathcal{T}_{\theta_T}(a_k)(\xi_j))) \right) \Big], \tag{3.5}$$

where $\alpha_1$ and $\alpha_2$ are regularization parameters, and $w_{\mathcal{T}}(a, \xi)$ represents the weighting factor emphasizing singular regions in $u$.

## 4. Theoretical analysis

In this section, we provide the theoretical foundation for the effectiveness of our proposed strategy. In traditional numerical methods, R-adaptive methods alleviate the limitations of linear reconstruction by dynamically adjusting the basis functions, thereby reducing approximation errors. Similarly, our proposed R-adaptive DeepONet method can also reduce the errors caused by linear reconstruction. First, we demonstrate the feasibility of the R-adaptive DeepONet in reducing reconstruction errors. Second, we rigorously prove the validity of the proposed method for two prototypical PDEs. In this section, we introduce the shorthand notation $A \lesssim B$ and $B \gtrsim A$ for the inequality $A \leq CB$ and $B \geq CA$, where $C$ denote generic constant independent of the number of trunk net basis functions and the mesh size unless otherwise stated. The notation $A \simeq B$ is equivalent to the statement $A \lesssim B$ and $B \lesssim A$.

### 4.1. Reconstruction error of DeepONets

#### 4.1.1. Bounds of the reconstruction error

In [22], the authors present a natural decomposition of DeepONets into three components: an encoder $\mathcal{E}$ that maps the infinite-dimensional input space into a finite-dimensional space, an approximator $\mathcal{A}$, often a neural network, maps one finite-dimensional space into another, and a trunk net-induced affine reconstructor $\mathcal{R}$ that maps the finite-dimensional space into the infinite-dimensional output space. The total DeepONet approximation error is then decomposed into encoding, approximation, and reconstruction errors.

Suppose $\mathcal{P}_\tau : \mathcal{Y} \to \mathrm{span}\{\tau_1(y), \ldots, \tau_n(y)\}$ is the projection operator that maps the solution function space $\mathcal{Y}$ to the linear span of trunk basis functions. The $L^2$ projection error can be defined as

$$E_{\mathcal{P}_\tau} := \|\mathcal{P}_\tau - \mathrm{Id}\|_{L^2(\mathcal{G}_{\#\mu})} = \left( \int_{\mathcal{Y}} \|\mathcal{P}_\tau u - u\|^2 \mathrm{d}(\mathcal{G}_{\#\mu})(u) \right)^{1/2}.$$

We define the reconstruction error as $E_{\mathcal{R}} := \inf_\tau E_{\mathcal{P}_\tau}$. The reconstruction error is closely related to the Kolmogorov $n$-width [31]. According to Kolmogorov $n$-width theory, we can provide a lower bound for the reconstruction error

$$E_{\mathcal{R}} \geq \sqrt{\sum_{j>n} \lambda_j},$$

where $\lambda_n \geq \lambda_{n+1} \geq \cdots$ are defined as in Theorem 2.1. This lower bound is fundamental as it reveals that the spectral decay rate for the covariance operator $\Gamma_{\mathcal{G}_{\#\mu}}$ of the push-forward measure essentially determines how low the approximation error of DeepONets can be for a given output dimension $n$ of the trunk nets. However, in practice, one does not have access to the form of the nonlinear operator $\mathcal{G}$, not to mention the covariance operator $\Gamma_{\mathcal{G}_{\#\mu}}$. Therefore, we aim to derive an upper bound on this error that is easier to be analyzed.

Since the reconstruction error $E_{\mathcal{R}}$ represents the projection error onto the linear space constructed by the optimal $n$ trunk basis functions in DeepONet, we can compare $E_{\mathcal{R}}$ with the projection error of a linear reconstruction system constructed using another, possibly non-optimal, set of $n$ basis functions. A straightforward choice for comparison is the linear finite element reconstruction. In [16], the authors proved that a linear finite element function in $\mathbb{R}^d$ with $N$ degrees of freedom can be represented by a ReLU DNN with at most $\mathcal{O}(d)$ hidden layers, and the number of neurons is at most $\mathcal{O}(\kappa^d N)$, where $\kappa \geq 2$ depends on the shape regularity of the underlying finite element grid. By utilizing the basis functions and interpolation operator defined here, we can extend this result, as stated in [16, Corollary 3.1], to the DeepONet structure.

**Lemma 4.1.** *Suppose that $\Omega \subset D_{\mathcal{Y}}$ is a bounded domain, and $\mathcal{M}_h$ is a locally convex finite element mesh on $\Omega$ consisting of a set of simplexes and degrees of freedom $n$. Define the corresponding nodal basis function as $\{\phi_1(y), \ldots, \phi_n(y)\}$, and $\Pi_h$ the interpolation operator on $\mathcal{M}_h$. Then there exists a ReLU-activated trunk net $\boldsymbol{\tau} : \mathbb{R}^{d_{\mathcal{Y}}} \to \mathbb{R}^n$, with*

$$\mathrm{depth}(\boldsymbol{\tau}) = \mathcal{O}(d_{\mathcal{Y}}), \quad \mathrm{size}(\boldsymbol{\tau}) = \mathcal{O}(\kappa^{d_{\mathcal{Y}}} n)$$

*such that the reconstruction error has the upper bound $E_{\mathcal{R}} \leq E_{FEM}$, where $E_{FEM} := \|\Pi_h - \mathrm{Id}\|_{L^2(\mathcal{G}_{\#\mu})}$ denotes the FEM interpolation error.*

It is a well-known result in the FEM literature that for a convex polyhedral domain $\Omega \subset \mathbb{R}^d$ and a regular finite element mesh $\mathcal{M}_h$, the following estimate holds:

$$\|u - \Pi_h u\|_{L^2(\Omega)} \lesssim h^2 \|u\|_{H^2(\Omega)}, \quad \forall u \in H^2(\Omega),$$

where $h = \max_{K \in \mathcal{M}_h} \mathrm{diam}(K)$, and $\Pi_h$ denotes the FEM interpolation operator.

Now combining the FEM interpolation error estimate and Lemma 4.1 yields the following upper bound of the reconstruction error.

**Theorem 4.1.** *Suppose $\Omega$ is a bounded convex domain in $\mathbb{R}^d$ and $\mathcal{G}$ defines a mapping $\mathcal{G} : \mathcal{X} \to H^2(\Omega)$. Then there exists a trunk net $\boldsymbol{\tau} : \mathbb{R}^d \to \mathbb{R}^n$, with*

$$\mathrm{depth}(\boldsymbol{\tau}) = \mathcal{O}(d), \quad \mathrm{size}(\boldsymbol{\tau}) = \mathcal{O}(\kappa^d n),$$

*where $\kappa$ is a constant depending only on $\Omega$, and the associated reconstruction error satisfies*

$$E_{\mathcal{R}} \lesssim n^{-2/d} \left( \int_{\mathcal{Y}} |u|^2_{H^2(\Omega)} \mathrm{d}(\mathcal{G}_{\#\mu})(u) \right)^{1/2} =: \overline{E_{\mathcal{R}}}.$$

**Remark 4.1.** The reconstruction error estimate here is based on the approach [22], but with different basis functions, yielding similar convergence rates. Our method also highlights the connection between FEM and ReLU-activated DeepONets, allowing us to derive an upper bound by comparing with the linear finite element interpolation error on a uniform mesh. However, uniform meshes may not be ideal for functions with local singularities, as finer meshes are often needed to maintain accuracy, leading to higher computational costs.

We denote the upper bound of the reconstruction error in Theorem 4.1 as $\overline{E_{\mathcal{R}}}$. In the next part, we will show that our proposed R-adaptive DeepONet has the property of reducing this upper bound of the reconstruction error.

### 4.1.2. R-adaptive to lessen the upper bound of the reconstruction error

Since our proposed R-adaptive DeepONet framework differs from the vanilla DeepONet, we first need to define its reconstruction error.

Suppose $\mathcal{P}_{\tau_{\mathcal{T}}} : \mathcal{Y} \to \mathrm{span}\{\tau_{\mathcal{T},1}(\xi), \ldots, \tau_{\mathcal{T},n}(\xi)\}$ is the projection operator that maps $\mathcal{Y}$ to the linear span of trunk basis functions of the adaptive coordinate DeepONet $\mathcal{T}_{\theta_T}$. Similarly $\mathcal{P}_{\tau_{\tilde{\mathcal{G}}}}$ is the projection operator that maps $\mathcal{Y}$ to the linear span of trunk basis functions of the adaptive solution DeepONet $\tilde{\mathcal{G}}_{\theta_G}$. Then we define the reconstruction error of the R-adaptive DeepONet as

$$E^{\mathrm{RA}}_{\mathcal{R}} := \inf_{\tau_{\mathcal{T}}, \tau_{\tilde{\mathcal{G}}}} \left\| \mathcal{P}_{\tau_{\tilde{\mathcal{G}}}} \tilde{\mathcal{G}}(a) \circ (\mathcal{P}_{\tau_{\mathcal{T}}} \mathcal{T}(a))^{-1} - \mathrm{Id} \right\|_{L^2(\mu)}, \tag{4.1}$$

where $\mathcal{T}, \tilde{\mathcal{G}}$ are introduced in Section 3.1, representing the ground adaptive coordinate and solution operators respectively. Moreover, $\mathcal{T}, \tilde{\mathcal{G}}$ satisfy $\tilde{\mathcal{G}}(a) \circ (\mathcal{T}(a))^{-1} = \mathcal{G}(a)$. We will explain the reason behind this definition below. The encoding error arises from the discretization of the input parameter $a$. The approximation error can be viewed as the error associated with learning the linear reconstruction coefficients $\beta(a)$, which is primarily influenced by the branch net. On the other hand, the reconstruction error represents the error due to the inherent linear reconstruction structure in DeepONet, which is affected by the trunk net. Therefore, in our proposed framework with two DeepONets, we focus solely on the error caused by the trunk net. Combining this error with (3.1), we derived the reconstruction errors presented in (4.1).

Assume that $\Omega \subset \mathbb{R}^d$ is polyhedral. $\mathcal{M}_h$ is an affine family of simplicial mesh for $\Omega$, with the reference element $\hat{K}$ being chosen as an equilateral $d$-simplex with unit volume. For any element $K$ in $\mathcal{M}_h$, we denote $F_K : \hat{K} \to K$ as the invertible affine

mapping satisfying $K = F_K(\hat{K})$. Then, for any $u \in H^2(\Omega)$, we have the following error estimate for piecewise linear interpolation:

$$\|u - \Pi_h u\|^2_{L^2(\Omega)} \lesssim \sum_{K\in\mathcal{M}_h} \|F_K'\|^4 \cdot |u|^2_{H^2(K)} =: E(u, \mathcal{M}_h),$$

where $F_K'$ denotes the Jacobian matrix of mapping $F_K$, and $|u|_{H^2(K)}$ denotes the $H^2$ semi-norm of $u$. In [19], the authors give a lower bound of $E(u, \mathcal{M}_h)$

$$E(u, \mathcal{M}_h) \geq N^{-4/d} \left( \sum_{K\in\mathcal{M}_h} |K| \langle u \rangle^{2d/(d+4)}_{H^2(K)} \right)^{(d+4)/d},$$

where $N$ is the number of the elements, and

$$\langle u \rangle_{H^2(K)} = \left( \frac{1}{|K|} |u|^2_{H^2(K)} \right)^{1/2}.$$

The lower bound can be attained via an optimal mesh $\mathcal{M}_h^*$ which equidistributes the density function $\rho_K = \langle u \rangle^{2d/(d+4)}_{H^2(K)}$. Note that there is a continuous coordinate transformation $x_u : \xi \mapsto x$ from the original uniform simplicial mesh $\mathcal{M}_h$ to the optimal mesh $\mathcal{M}_h^*$, which is linear on each element $K \in \mathcal{M}_h$. Through this transform, $\tilde{u} = u \circ x_u$ becomes a function of $\xi$ and can be well approximated on a uniform mesh of $\xi$. Let $\Pi_\xi$ denote the linear finite element interpolation operator on the uniform mesh of $\xi$ and $\xi_u : x \mapsto \xi$ the inverse of $x_u$, then we can bound the interpolation error by

$$\|u - (\Pi_\xi \tilde{u}) \circ \xi_u\|_{L^2(\Omega)} \lesssim n^{-2/d} \left( \sum_{K\in\mathcal{M}_h^*} |K| \langle u \rangle^{2d/(d+4)}_{H^2(K)} \right)^{(d+4)/2d}. \tag{4.2}$$

Here, we have applied the Euler's formula for polyhedra [38] to address the relationship between the number of elements $N$ and the degrees of freedom $n$.

Based on the previous analysis, we employ ReLU trunk nets to emulate linear finite element space over a predetermined fixed mesh, and then approximate the target function space through function composition. This allows us to derive an upper bound for the reconstruction error. From (4.2) and Lemma 4.1, it follows that:

**Theorem 4.2.** *Suppose $\Omega \subset \mathbb{R}^d$ and $\mathcal{G}$ defines a mapping $\mathcal{G} : \mathcal{X} \to H^2(\Omega)$. Then there exists a coordinate transform DeepONet with trunk net $\boldsymbol{\tau}_\mathcal{T} : \mathbb{R}^d \to \mathbb{R}^n$ and a transformed solution DeepONet with trunk net $\boldsymbol{\tau}_{\tilde{\mathcal{G}}} : \mathbb{R} \to \mathbb{R}^n$, with*

$$\begin{aligned} &\mathrm{depth}(\boldsymbol{\tau}_\mathcal{T}) = \mathcal{O}(d), \quad \mathrm{depth}(\boldsymbol{\tau}_{\tilde{\mathcal{G}}}) = \mathcal{O}(d), \\ &\mathrm{size}(\boldsymbol{\tau}_\mathcal{T}) = \mathcal{O}(\kappa^d n), \quad \mathrm{size}(\boldsymbol{\tau}_{\tilde{\mathcal{G}}}) = \mathcal{O}(\kappa^d n), \end{aligned}$$

*where $\kappa$ is a constant dependent with $d$, and the associated reconstruction error satisfies*

$$\begin{aligned} E_{\mathcal{R}}^{RA} &\leq \mathbb{E}_{\mathcal{G}_{\#\mu}}\big[\min_{\mathcal{M}_{h,u}} E(u, \mathcal{M}_{h,u})\big] \\ &= \mathbb{E}_{\mathcal{G}_{\#\mu}}\big[E(u, \mathcal{M}_{h,u}^*)\big] \\ &\leq \min_{\mathcal{M}_h} \mathbb{E}_{\mathcal{G}_{\#\mu}}[E(u, \mathcal{M}_h)] \leq \overline{E_{\mathcal{R}}}. \end{aligned}$$

*Here $\mathcal{M}_{h,u}$ means the mesh depends on $u$ and $\mathcal{M}_{h,u}^*$ denotes the optimal mesh for each $u$.*

By adding a coordinate transform (learned by another DeepONet with the same size of the original model), the upper bound for the reconstruction error in the R-adaptive DeepONet may be smaller than that of the vanilla DeepONet. This theorem implies the reduction of the reconstruction error in the R-adaptive DeepONet compared to the vanilla DeepONet.

## 4.2. Approximation properties for concrete examples

The previous subsection theoretically demonstrates that the proposed framework can reduce the upper bound of the reconstruction error, but we do not directly show its advantages over the vanilla DeepONet. This is because the form of the operator $\mathcal{G}$ varies significantly across different problems, making it challenging to use a unified framework for analysis. In this subsection, we select two prototypical PDEs widely used to analyze numerical methods for transport-dominated PDEs. We rigorously prove that the proposed method efficiently approximates operators stemming from discontinuous solutions of PDEs, whereas vanilla DeepONets fail to do so. The chosen PDEs are the linear advection equation and the nonlinear inviscid Burgers' equation, which are the prototypical examples of hyperbolic conservation laws. Detailed descriptions of the exact operators and corresponding approximation results using both vanilla and our proposed reconstruction methods are presented below.

### 4.2.1. Linear advection equation

Consider the one-dimensional linear advection equation

$$\partial_t u + a\partial_x u = 0, \quad u(\cdot, t=0) = \bar{u} \tag{4.3}$$

on a $2\pi$-periodic domain $\mathbb{T}$, with constant speed $a \in \mathbb{R}$. The underlying operator is $\mathcal{G}_{\text{adv}} : L^2(\mathbb{T}) \to L^2(\mathbb{T}), \bar{u} \mapsto \mathcal{G}_{\text{adv}}(\bar{u}) := u(\cdot, T)$, obtained by solving the PDE (4.3) with initial data $\bar{u}$ up to some final time $t = T$. As input measure $\mu \in \text{Prob}(\mathcal{X})$, we consider random input functions $\bar{u} \sim \mu$ given by the square (box) wave of height $h$, width $w$ and centered at $\zeta$,

$$\bar{u}_\zeta(x) = h\mathbb{1}_{[-\omega/2, \omega/2]}(x - \zeta). \tag{4.4}$$

In the following we let $h = 1, w = \pi$, and $\zeta \in [0, 2\pi]$ be uniformly distributed.

Following [22], we observe that the translation invariance of the problem implies that the Fourier basis is optimal for spanning the output space. Given the discontinuous nature of the underlying functions, the eigenvalues of the covariance operator for the push-forward measure decay linearly at most in $n$. Consequently, the lower bound implies a linear decay of error in terms of the number of trunk net basis functions. As a consequence, the following result is established, as stated in [23].

**Theorem 4.3** (Lanthaler *et al.* [23, Theorem 3.1])**.** *Let $n \in \mathbb{N}$. For any DeepONet $\mathcal{N}^{\mathrm{DON}}$ with $n$ trunk-/branch-net output functions, satisfying*

$$\sup_{\bar{u}\sim\mu} \|\mathcal{N}^{\mathrm{DON}}(\bar{u})\|_{L^\infty} \leq M < \infty,$$

*we have the lower bound*

$$\mathcal{E}(\mathcal{N}^{\mathrm{DON}}) := \mathbb{E}_{\bar{u}\sim\mu}\Big[\|\mathcal{G}_{\mathrm{adv}}(\bar{u}) - \mathcal{N}^{\mathrm{DON}}(\bar{u})\|_{L^2}^2\Big]^{1/2} \gtrsim n^{-1}.$$

*Consequently, for a given $\epsilon > 0$, to achieve $\mathcal{E}(\mathcal{N}^{\mathrm{DON}}) \leq \epsilon$ with DeepONet, we need at least $n \gtrsim \epsilon^{-1}$ trunk and branch net basis functions.*

In contrast to the previous DeepONet results, we now present an efficient approximation result for R-adaptive DeepONet.

**Theorem 4.4.** *For any $\epsilon > 0$, there exist two DeepONets $\mathcal{T}_{\theta_T}, \tilde{\mathcal{G}}_{\theta_G}$, both with $n$ trunk-/branch-net output functions, and assume that $\mathcal{T}_{\theta_T} : [-\pi, \pi] \to [-\pi, \pi], \xi \mapsto x(\xi)$ is bijective. Then the $L^2$-error of the R-adaptive DeepONet system $\{\mathcal{T}_{\theta_T}, \tilde{\mathcal{G}}_{\theta_G}\}$ satisfies*

$$\mathcal{E} := \mathbb{E}_{\bar{u}\sim\mu}\left[\|\mathcal{G}_{\mathrm{adv}}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\|_{L^2}^2\right]^{1/2} \leq \epsilon$$

*with $n \simeq \epsilon^{-2/3}$.*

The detailed proof, presented in Appendix B, is based on the fact that the reconstruction error is determined by the approximation error of the optimal reconstruction basis functions. Therefore, if we find a set of basis functions represented by trunk nets that satisfy the error bounds, then the approximation error of the optimal reconstruction basis functions is naturally smaller than the approximation error of this set of basis functions. By construction, we show that the finite element basis functions on a uniform mesh can be represented by trunk nets and satisfies the error bounds. Hence, we complete the proof.

### 4.2.2. Inviscid Burgers' equation

Next, we consider the inviscid Burgers' equation in one-space dimension, which is the prototypical example of nonlinear hyperbolic conservation laws

$$\partial_t u + \partial_x\left(\frac{1}{2}u^2\right) = 0, \quad u(\cdot, t=0) = \bar{u}, \tag{4.5}$$

on the $2\pi$-periodic domain $\mathbb{T}$. It is well-known that discontinuities in the form of shock waves can appear in finite time even for smooth $\bar{u}$. Consequently, solutions of (4.5) are interpreted in the sense of distributions and entropy conditions are imposed to ensure uniqueness. Thus, the underlying solution operator is $\mathcal{G}_{\mathrm{Burg}} : L^2(\mathbb{T}) \to L^2(\mathbb{T}), \bar{u} \mapsto \mathcal{G}_{\mathrm{Burg}}(\bar{u}) = u(\cdot, T)$, with $u$ being the entropy solution of (4.5) at final time $T$. Given $\zeta \sim \mathrm{Unif}([0, 2\pi])$, we define the random field

$$\bar{u}_\zeta(x) := -\sin(x - \zeta),$$

and we define the input measure $\mu \in \mathrm{Prob}(L^2(\mathbb{T}))$ as the law of $\bar{u}_\zeta$. Then, similarly, we can rewrite the underlying operator as $\mathcal{G}_{\mathrm{Burg}} : [0, 2\pi] \to L^2(\mathbb{T}), \zeta \mapsto \mathcal{G}_{\mathrm{Burg}}(\bar{u}_\zeta) := u_\zeta(\cdot, T)$.

Also, translation invariance and local discontinuous can be observed in this problem. This leads to the following conclusion, as presented in [23].

**Theorem 4.5** (Lanthaler *et al.* [23, Theorem 3.4])**.** *Assume that $\mathcal{G}_{\mathrm{Burg}} = u(\cdot, T)$, for $T > \pi$ and $u$ is the entropy solution of* (4.5) *with initial data $\bar{u} \sim \mu$. Then the $L^2$-error for any DeepONet $\mathcal{N}^{\mathrm{DON}}$ with $n$ trunk-/branch-net output functions is lower-bounded by*

$$\mathcal{E}(\mathcal{N}^{\mathrm{DON}}) := \mathbb{E}_{\bar{u}\sim\mu}\left[\|\mathcal{G}_{\mathrm{Burg}}(\bar{u}) - \mathcal{N}^{DON}(\bar{u})\|_{L^2}^2\right]^{1/2} \gtrsim n^{-1}.$$

*Consequently, for a given $\epsilon > 0$, achieving an error $\mathcal{E}(\mathcal{N}^{\mathrm{DON}}) \lesssim \epsilon$ requires at least $n \gtrsim \epsilon^{-1}$.*

Similar to that in the analysis of linear advection equation, in contrast to the vanilla DeepONet, we have the following result for efficient approximation of $\mathcal{G}_{\mathrm{Burg}}$ with R-adaptive DeepONet, whose proof is an almost exact repetition of the proof of Theorem 4.4, which is arranged in Appendix C for convenience of the reader.

**Theorem 4.6.** *Assume that $T > \pi$. For any $\epsilon > 0$, there exist two DeepONets $\mathcal{T}_{\theta_T}, \tilde{\mathcal{G}}_{\theta_G}$, both with $n$ trunk-/branch-net output functions, and assume that $\mathcal{T}_{\theta_T} : [-\pi, \pi] \to [-\pi, \pi]$, $\xi \mapsto x(\xi)$ is bijective. Then the $L^2$-error of the R-adaptive DeepONet system $\{\mathcal{T}_{\theta_T}, \tilde{\mathcal{G}}_{\theta_G}\}$ satisfies*

$$\mathbb{E}_{\bar{u}\sim\mu}\left[\|\mathcal{G}_{\mathrm{Burg}}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G} \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\|_{L^2}^2\right]^{1/2} \leq \epsilon$$

*with $n \simeq \epsilon^{-2/3}$.*

## 5. Numerical experiments

In this section, we present several numerical results to evaluate the performance of our proposed R-adaptive DeepONet framework, comparing it with vanilla DeepONets and Shift-DeepONets. We focus on three test problems: Burgers' equation, commonly used to benchmark neural operators; linear advection equations in 1D; and compressible Euler equations in one dimension, which is representative of hyperbolic systems of

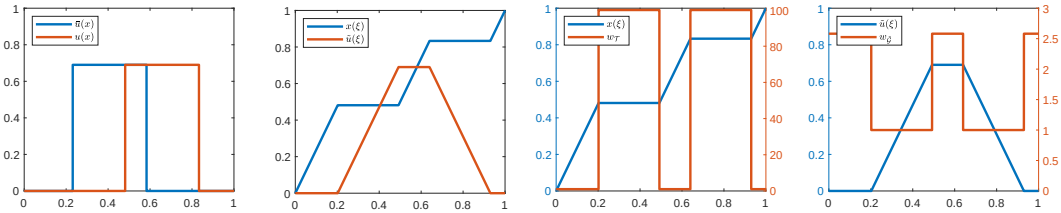
conservation laws. Through these experiments, we aim to highlight the potential advantages of the proposed framework. We use the relative $L^2$ norm as the error metric employed throughout all numerical experiments to assess model performance. In the next part, for simplicity, we use DON as a shorthand for DeepONet.

## 5.1. Linear advection equation

We take the linear advection equation (4.3) as the first example to echo our theoretical analysis in the previous section. Here we set $\Omega = [0, 1]$ and $a = 1$. The initial data is given by (4.4) corresponding to square waves, with initial heights, widths, and shifts uniformly distributed in $[0.2, 0.8]$, $[0.05, 0.3]$ and $[0, 0.5]$, respectively. We aim to learn the underlying solution operator $\mathcal{G}_{\text{adv}} : \bar{u} \mapsto \mathcal{G}_{\text{adv}}(\bar{u}) = u(\cdot, T = 0.25)$, which maps the initial data $\bar{u}$ to the solution at the terminal time $T = 0.25$. Since $\bar{u}$ is controlled by a parameter $\zeta$, the underlying operator is equivalent to $\mathcal{G} : \zeta \mapsto \mathcal{G}_{\text{adv}}(\bar{u}_\zeta) := u_\zeta(\cdot, T)$. Therefore, we try to learn the map $\mathcal{G}$ instead of $\mathcal{G}_{\text{adv}}$. The training and testing samples of the solutions for vanilla DON and Shift-DON are generated by sampling the underlying exact solution, which are obtained by translating the initial data sampled on $2048$ uniformly distributed grids by $0.25$. The training data of R-adaptive DON is obtained by preprocessing this batch of data. We use density function

$$\rho(x) = \sqrt{1 + |u'(x)|^2} \tag{5.1}$$

to obtain the equidistributed coordinate transform functions $x(\xi)$ and corresponding adaptive solution functions $\tilde{u}(\xi) = u(x(\xi))$. Moreover, according to (3.2) and (3.4), we calculate the weights $w_{\tilde{\mathcal{G}}}$ and $w_{\mathcal{T}}$ for the training of the R-adaptive DON. Fig. 1 shows an example of the processed data. As can be seen from Figs. 1(a) and 1(b), the discontinuity in the original data $u(x)$ has been alleviated after preprocessing and has become a smoother transition, and the corresponding coordinate transformation function $x(\xi)$ is also smooth and has no discontinuity. Furthermore, in Fig. 1(c) and 1(d), we show the calculated weights $w_{\tilde{\mathcal{G}}}$ and $w_{\mathcal{T}}$. We can see that $w_{\tilde{\mathcal{G}}}$ and $w_{\mathcal{T}}$ satisfy certain properties as described in Section 3. $w_{\tilde{\mathcal{G}}}$ is relatively small in places where $u$ has singularities, while $w_{\mathcal{T}}$ is just the opposite.



(a) Initial condition and final output (b) Adaptive coordinate and solution (c) Adaptive coordinate and weight (d) Adaptive solution and weight

Figure 1: Illustration of an example of processed data for advection equation.

To ensure a fair comparison, we used models with similar structures. For vanilla DON, the main body of Shift-DON, and the two sub-DON in R-adaptive Net, we employed the same architecture: both the branch and trunk nets have $4$ layers, each containing $256$ neurons. For the scale and shift nets in Shift-DON, we also used a structure of $4$ layers with $256$ neurons per layer. This approach ensures that the number of parameters in each model remains comparable.

For each model, we use a training set with $1000$ samples and a validation set with $200$ samples. The training is performed with the ADAM optimizer, with learning rate $10^{-3}$ for $100000$ epochs and a learning rate decay of $10\%$. We compute the relative $L^2$-error on the validation set every $2000$ epoch. The validation error throughout the training process is shown in Fig. 2(a).

It can seen that the validation error of the adaptive solution DON and adaptive coordinate DON decay rapidly, ending up much lower than that of the vanilla DON. Since the validation error indicates the ability of the model to approximate the target dataset, it means that the adaptive solution DON and coordinate DON can approximate their target $\tilde{u}(\xi)$ and $x(\xi)$ well. The reason for this can be understood by examining the eigenvalues of the covariance matrices of the target datasets. In Fig. 2(b) we show the eigenvalues of the covariance operators of the three data sets, the original solutions $\{u(x)\}$, the processed adaptive solution $\{\tilde{u}(\xi)\}$ and coordinate transform functions $\{x(\xi)\}$. As can be seen from the figure, the eigenvalues of the latter two sets decay much faster than those of the former. From (2.4) we know that the corresponding reconstruction error is also smaller. This demonstrates that our preprocessing effectively reduces the lower bound of the reconstruction error, highlighting the feasibility and advantages of our proposed method.

In the testing part, we also use a data set with $200$ samples to calculate the testing error. We use the trained models to predict the solution values at $2048$ grid points uniformly distributed over $[0, 1]$ and calculate the approximate $L^2$ error. For the testing error of R-adaptive DON, we use one-dimensional piecewise-linear interpolation to get the solutions on uniformly distributed grids over $[0, 1]$ and then compare it with the exact solutions.

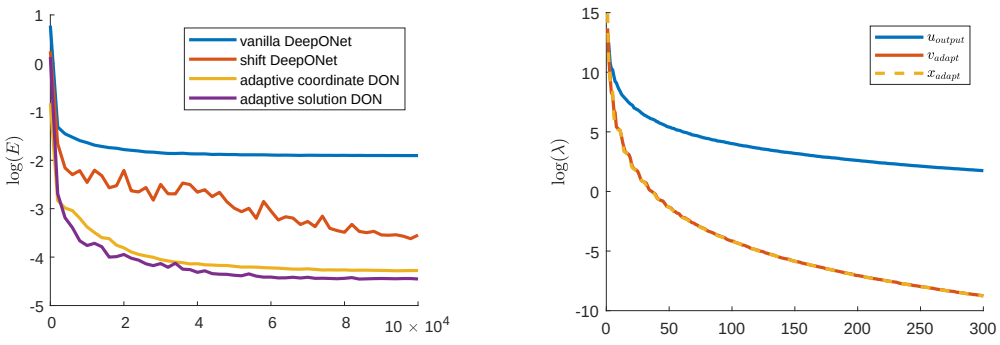



Figure 2: Left: The validation error of different models during training. Right: Eigenvalues of the covariance operators of different data sets.

First, we show the relative testing $L^2$ error of different models trained using output datasets with different sampling densities and verify the advantage of R-adaptive DON that smaller output datasets can be used for good performance. Table 1 shows the testing errors of models trained using output data sampled on $16, 32, 64, 128$ uniformly distributed grids over $[0, 1]$. It is observed that as the number of sampling points increases, the approximation performance of vanilla DON and the Shift-DON improves, resulting in a gradual decrease in testing error. In contrast, R-adaptive DON shows a relative insensitivity to the density of output training data, with the error remaining relatively stable. Therefore, compared to the vanilla DON and the Shift-DON, the accuracy of the R-adaptive DON is less sensitive to the number of sampling points. We also note that the R-adaptive DON trained with data sampled on $16$ uniform grids achieves prediction accuracy comparable to that of Shift-DON trained with sampled on $128$ uniform grids. This implies that the proposed method can achieve similar accuracy with a smaller training dataset, hence can reduce the storage requirements during training. This advantage is particularly significant in high-dimensional situations.

Next, we present a set of numerical examples to validate the effectiveness of introducing adaptive weights as described in Section 3.2. We conducted four groups of experiments using the R-adaptive DON architecture. The training output data is sampled on $2048$ uniformly distributed grids over $[0, 1]$. In the first two groups, we only learn the adaptive solution operator $\tilde{\mathcal{G}}$ with upper bounds of the weights: $\bar{w}_{\tilde{\mathcal{G}}} = 1$ and $\bar{w}_{\tilde{\mathcal{G}}} = 2$ respectively. Note that $\bar{w}_{\tilde{\mathcal{G}}} = 1$ indicates training without using weights. In this way, we can demonstrate the effectiveness of introducing weight $w_{\tilde{\mathcal{G}}}$. The results are shown in Table 2. It can be seen that the introduction of adaptive weight $w_{\tilde{\mathcal{G}}}$ reduces the approximation error effectively, consistent with the analysis in Section 3.2. In the latter two groups of experiments, we change the strategy of learning adaptive coordinates while keeping the part of the adaptive solution unchanged, aiming to show the effectiveness of introducing weight $w_{\mathcal{T}}$. The results also show that the introduction of weight $w_{\mathcal{T}}$ can improve the model's performance.

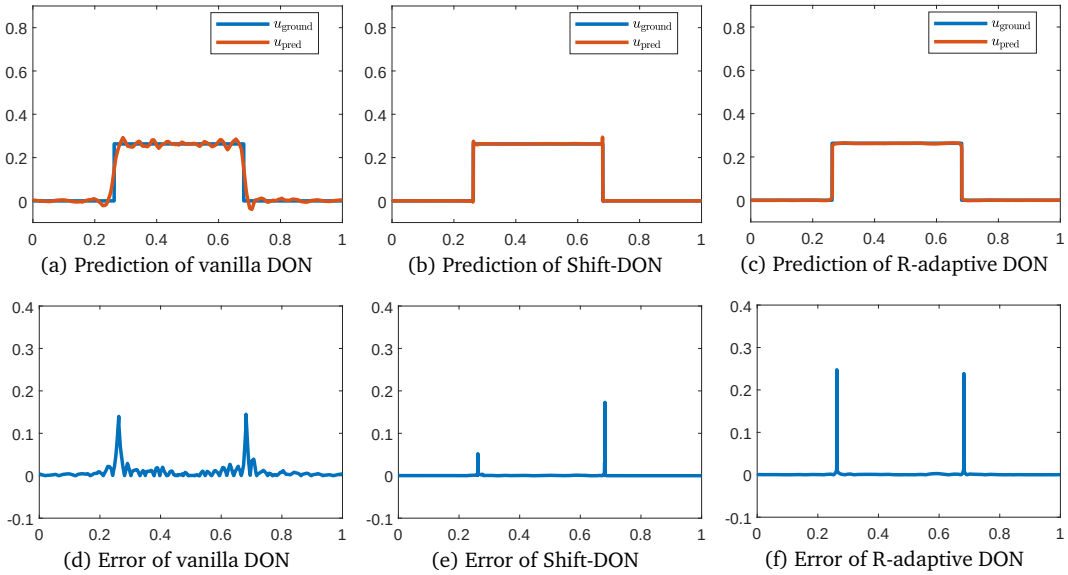
In the end of this subsection, we provide an example of predictions from different models to visually demonstrate that R-adaptive DON can effectively approximate

Table 1: Relative testing $L^2$ error of different models trained using data of different resolutions.

| Sampling points | Vanilla DON | Shift-DON | R-adaptive DON |
|---|---|---|---|
| 16 | $8.17 \times 10^{-2}$ | $1.90 \times 10^{-2}$ | $6.95 \times 10^{-3}$ |
| 32 | $4.25 \times 10^{-2}$ | $1.68 \times 10^{-2}$ | $6.57 \times 10^{-3}$ |
| 64 | $2.79 \times 10^{-2}$ | $1.13 \times 10^{-2}$ | $6.96 \times 10^{-3}$ |
| 128 | $2.46 \times 10^{-2}$ | $6.37 \times 10^{-3}$ | $6.62 \times 10^{-3}$ |

Table 2: Relative testing $L^2$ errors of R-adaptive DON for linear advection equation.

| Model | $\bar{w}_{\tilde{\mathcal{G}}} = 1$ & $x_{\text{ground}}$ | $\bar{w}_{\tilde{\mathcal{G}}} = 2$ & $x_{\text{ground}}$ | $\bar{w}_{\tilde{\mathcal{G}}} = 2$ & $\bar{w}_{\mathcal{T}} = 2$ | $\bar{w}_{\tilde{\mathcal{G}}} = 2$ & $\bar{w}_{\mathcal{T}} = 100$ |
|---|---|---|---|---|
| *Error* | $5.75 \times 10^{-5}$ | $3.36 \times 10^{-5}$ | $1.69 \times 10^{-2}$ | $6.54 \times 10^{-3}$ |




(a) Prediction of vanilla DON (b) Prediction of Shift-DON (c) Prediction of R-adaptive DON

(d) Error of vanilla DON (e) Error of Shift-DON (f) Error of R-adaptive DON

Figure 3: An example of the prediction results of the three models for linear advection equation.

problems with discontinuities. Here, we use the R-adaptive DON framework with the adaptive solution DON and coordinate DON trained with adaptive weight as shown in the last column in Table 2. From Fig. 3, it can be seen that vanilla DON does not approximate the solution operator well, and its prediction results oscillate wildly due to the existence of discontinuities. Both Shift-DON and R-adaptive DON can grasp the discontinuities and approximate the smooth region well. In addition, Shift-DON leaves small oscillations at the discontinuities, while R-adaptive does not oscillate, but naturally polishes the function. Compared to Fig. 3(e), Fig. 3(f) shows larger errors near the discontinuous point. This phenomena arises because the neural network is used to approximate the coordinate transformation $x(\xi)$, which may introduce slight inaccuracies, and these small deviations in coordinate learning can lead to significant errors in the solution near discontinuities, where the values of the solution change abruptly. The misalignment caused by the coordinate learning in R-adaptive DON can be mitigated by improving the accuracy of the coordinate approximation, thereby reducing its impact in regions near discontinuities.

## 5.2. Viscous Burgers' equation

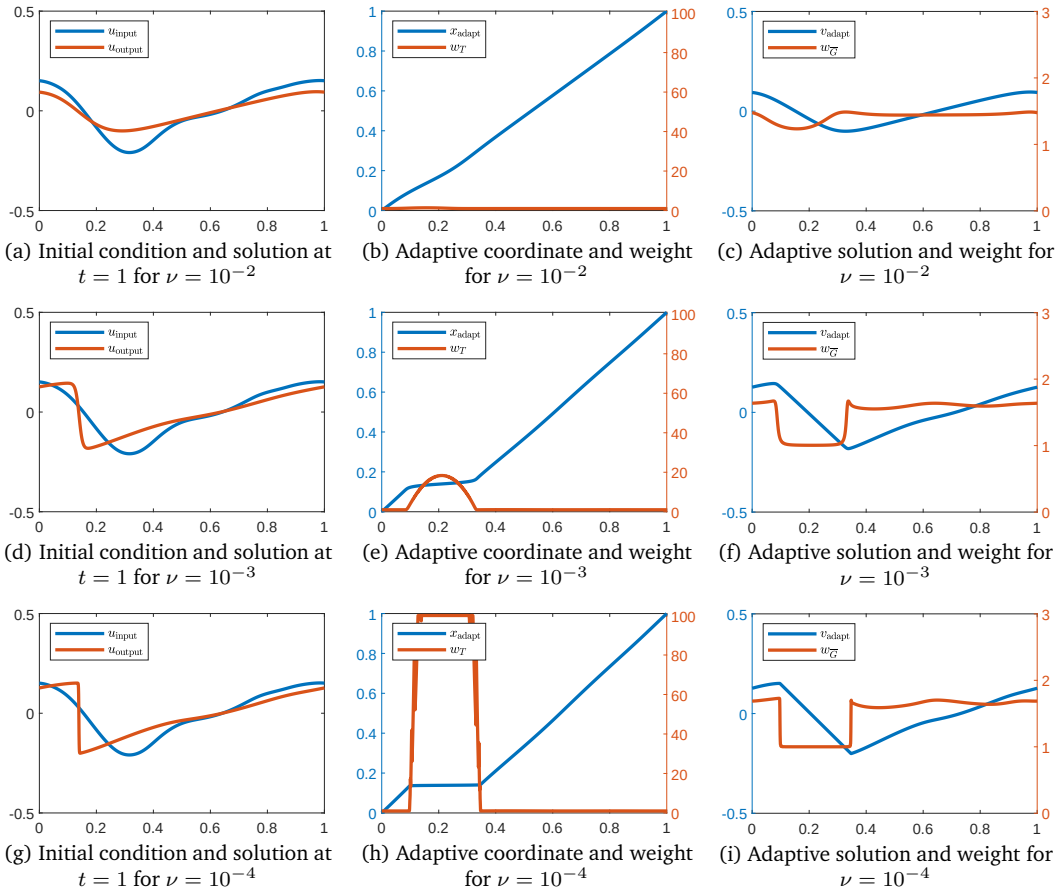
Next, we consider the one-dimensional viscous Burgers' equation

$$\begin{aligned} &\frac{\partial}{\partial t}u(x,t)+\frac{1}{2}\frac{\partial}{\partial x}\big(u(x,t)\big)^2=\nu\frac{\partial^2}{\partial x^2}u(x,t), \quad x\in[0,1], \quad t\in[0,1],\\ &u(x,0)=u_0(x), \qquad\qquad\qquad\qquad\qquad\qquad\quad x\in[0,1] \end{aligned} \tag{5.2}$$

with periodic boundary conditions and a fixed viscosity $\nu$.

When the viscosity coefficient is large, the solution of the Burgers' equation will not exhibit significant singularities. However, as the viscosity coefficient decreases, the solution gradually approaches that of the corresponding inviscid Burgers' equation, resulting in regions with large gradients. In this experiment, we use several different viscosity coefficients such as $\nu = 5 \times 10^{-2}, 10^{-2}, 10^{-3}, 10^{-4}$. Our goal is to learn the solution operator mapping initial conditions $u(x, 0)$ to the solution at $T = 1$.

To obtain a set of training data, we randomly sample $1000$ input functions from a Gaussian random field (GRF) $\mathcal{N}(0, 25^2(-\Delta + 5^2 I)^{-4})$ and solve the Burgers' equation using the Chebfun package with a spectral Fourier discretization and a fourth-order stiff time stepping scheme with a time-step size of $10^{-4}$. We generate test data sets by sampling another $200$ input functions from the same GRF. On the input side, we sample the initial data on a uniformly distributed grid of $128$ points over $[0, 1]$ as the input parameters for training the models. The data preprocessing is similar to the previous test, and the density function (5.1) is also used. As before, we show some examples of the processed data in Fig. 4. When the viscosity coefficient is relatively large, the solution does not exhibit singularities. In this case, the adaptive solution obtained



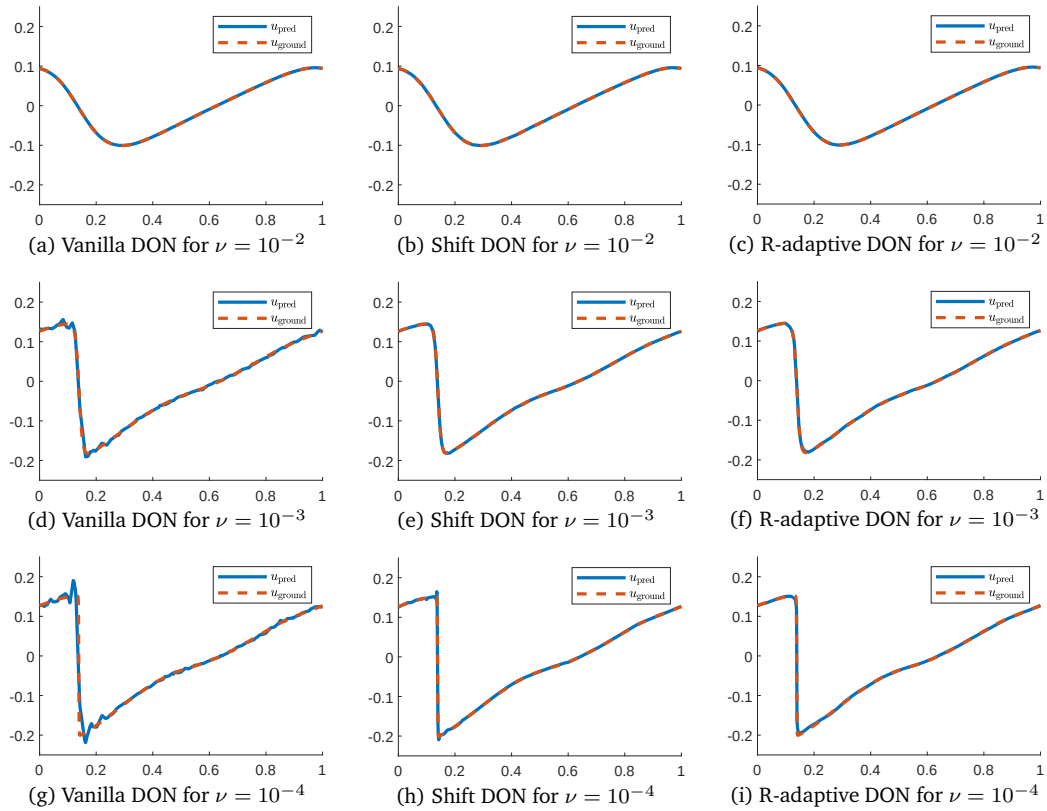

(a) Initial condition and solution at $t = 1$ for $\nu = 10^{-2}$
(b) Adaptive coordinate and weight for $\nu = 10^{-2}$
(c) Adaptive solution and weight for $\nu = 10^{-2}$
(d) Initial condition and solution at $t = 1$ for $\nu = 10^{-3}$
(e) Adaptive coordinate and weight for $\nu = 10^{-3}$
(f) Adaptive solution and weight for $\nu = 10^{-3}$
(g) Initial condition and solution at $t = 1$ for $\nu = 10^{-4}$
(h) Adaptive coordinate and weight for $\nu = 10^{-4}$
(i) Adaptive solution and weight for $\nu = 10^{-4}$

Figure 4: Illustration of an example of processed data for Burgers' equation.

through preprocessing is close to the original data, and the coordinate transformation is approximately an identity mapping. However, as the viscosity coefficient decreases, the adaptive solution obtained through preprocessing becomes smoother and free of singularities compared to the original data. Additionally, we have shown the graphs of the adaptive weights, whose properties are consistent with our analysis in Section 3.2.

We use the same network structures and training strategies as in the previous experiment. The testing errors for different operator learning strategies are presented in Table 3. As shown in the table, vanilla DON approximates the solution operator well when the viscosity coefficient is large. As the viscosity coefficient decreases and the solution exhibits local singularities, the performance of vanilla DON degrades. In contrast, R-adaptive DON performs better than vanilla DON at low viscosity levels and even achieves smaller relative errors than Shift-DON. This may indicate that R-adaptive DON has an advantage over Shift-DON in approximating problem whose solution exhibits large gradients rather than discontinuities, such as convection-dominated diffusion equations. We will explore this in future work.

To illustrate the prediction results more intuitively, we present some prediction examples in Fig. 5. As seen in the figures, vanilla DON performs well in approximating the solution when the viscosity coefficient is large, and both Shift-DON and R-adaptive




(a) Vanilla DON for $\nu = 10^{-2}$ (b) Shift DON for $\nu = 10^{-2}$ (c) R-adaptive DON for $\nu = 10^{-2}$

(d) Vanilla DON for $\nu = 10^{-3}$ (e) Shift DON for $\nu = 10^{-3}$ (f) R-adaptive DON for $\nu = 10^{-3}$

(g) Vanilla DON for $\nu = 10^{-4}$ (h) Shift DON for $\nu = 10^{-4}$ (i) R-adaptive DON for $\nu = 10^{-4}$

Figure 5: Illustration of an example of processed data for Burgers' equation.

Table 3: Relative testing $L^2$ error of different models for Burgers' equation.

| Model | $\nu = 5 \times 10^{-2}$ | $\nu = 10^{-2}$ | $\nu = 10^{-3}$ | $\nu = 10^{-4}$ |
|---|---|---|---|---|
| Vanilla DON | $5.41 \times 10^{-5}$ | $3.93 \times 10^{-4}$ | $1.00 \times 10^{-2}$ | $3.45 \times 10^{-2}$ |
| Shift-DON | $1.58 \times 10^{-4}$ | $6.32 \times 10^{-4}$ | $1.15 \times 10^{-2}$ | $3.93 \times 10^{-2}$ |
| R-adaptive DON | $1.31 \times 10^{-4}$ | $4.90 \times 10^{-4}$ | $8.35 \times 10^{-3}$ | $2.44 \times 10^{-2}$ |

DON also provide accurate predictions. However, as the viscosity coefficient decreases, the solution of Burgers' equation develops a large local gradient, causing the solution predicted by vanilla DON to oscillate, especially near the singularity region. In contrast, both Shift- and R-adaptive DONs capture the local singularity characteristics effectively. Additionally, as noted in Section 5.1, Shift-DON produces minor oscillations while R-adaptive DON polishes the solutions around the singularity.

## 5.3. Shock tube

In this subsection, we consider the motion of an inviscid gas described by the Euler equations of aerodynamics. The governing equations can be written as

$$\begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}_t + \begin{pmatrix} \rho u \\ \rho u^2 + p \\ (E+p)u \end{pmatrix}_x = 0$$

with $\rho, u$ and $p$ denoting the fluid density, velocity, and pressure. $E$ represents the total energy per unit volume

$$E = \frac{1}{2}\rho u^2 + \frac{p}{\gamma - 1},$$

where $\gamma = c_p/c_v$ is the gas constant which equals to $1.4$ for a diatomic gas considered here.

We restrict the equation to $D = [-5, 5]$ and consider the initial data corresponding to a shock tube of the form

$$\rho_0 = \begin{cases} \rho_L, & x \le x_0, \\ \rho_R, & x > x_0, \end{cases} \quad u_0 = \begin{cases} u_L, & x \le x_0, \\ u_R, & x > x_0, \end{cases} \quad p_0 = \begin{cases} p_L, & x \le x_0, \\ p_R, & x > x_0, \end{cases}$$

parameterized by the left and right states $(\rho_L, u_L, p_L)$, $(\rho_R, u_R, p_R)$, and the location of the initial discontinuity $x_0$. As proposed in Lye *et al.* [29], these parameters are, in turn, drawn from the measure

$$\begin{aligned} &\rho_L = 0.75 + 0.45g(z_1), \quad u_L = 0.5 + 0.5g(z_3), \quad p_L = 2.5 + 1.6g(z_4), \\ &\rho_R = 0.4 + 0.3g(z_2), \quad u_R = 0, \quad p_R = 0.375 + 0.325g(z_5), \\ &x_0 = 0.5g(z_6) \end{aligned}$$

with $z = [z_1, \ldots, z_6] \sim \text{Unif}([0,1]^6)$ and $g(z) = 2z - 1$. We aim to approximate the operator $\mathcal{G} : [\rho_0, \rho_0 u_0, E_0] \mapsto E(1.5)$. As in the previous subsections, we simplify this mapping to $\mathcal{G} : z \mapsto E(1.5)$.

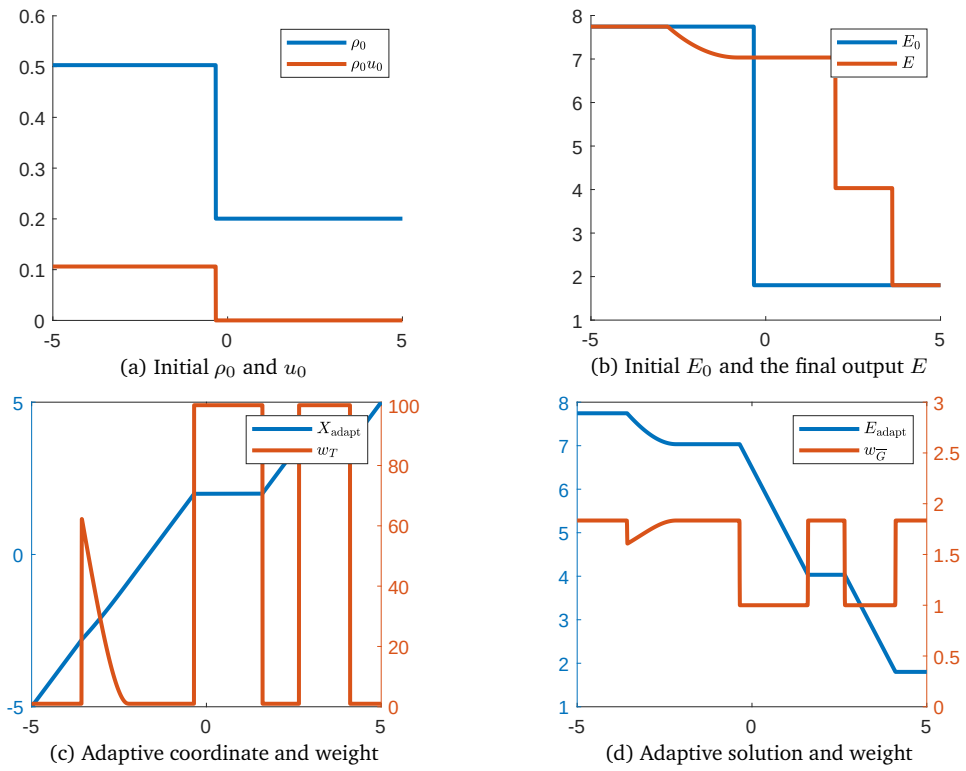
The training (and testing) output is generated through the analytic method in [36]. The rest of the concretes are similar to those in Subsections 5.1 and 5.2. As in the previous subsections, we first show an example of the processed data in Fig. 6.

In the testing part, for the prediction results of R-adaptive DON, we still obtained them by piecewise linear interpolation to the uniformly distributed grids through the output of two sub-DONs. The results are summarized in Table 4 and an example of the output is shown in Fig. 7.

From Table 4, we can see that R-adaptive DON has stronger performance than vanilla DON when approximating the solution of the Sod shock tube problem. And, from Fig. 7, we can see that although the error is larger than that of Shift-DON, R-adaptive DON can catch the discontinuity just as well as Shift-DON.

Table 4: Testing $L^2$ error of different models for Sod shocktube problem.

| Model | Vanilla DON | Shift-DON | R-adaptive DON |
|---|---|---|---|
| Error | $4.77 \times 10^{-4}$ | $2.71 \times 10^{-5}$ | $9.24 \times 10^{-5}$ |



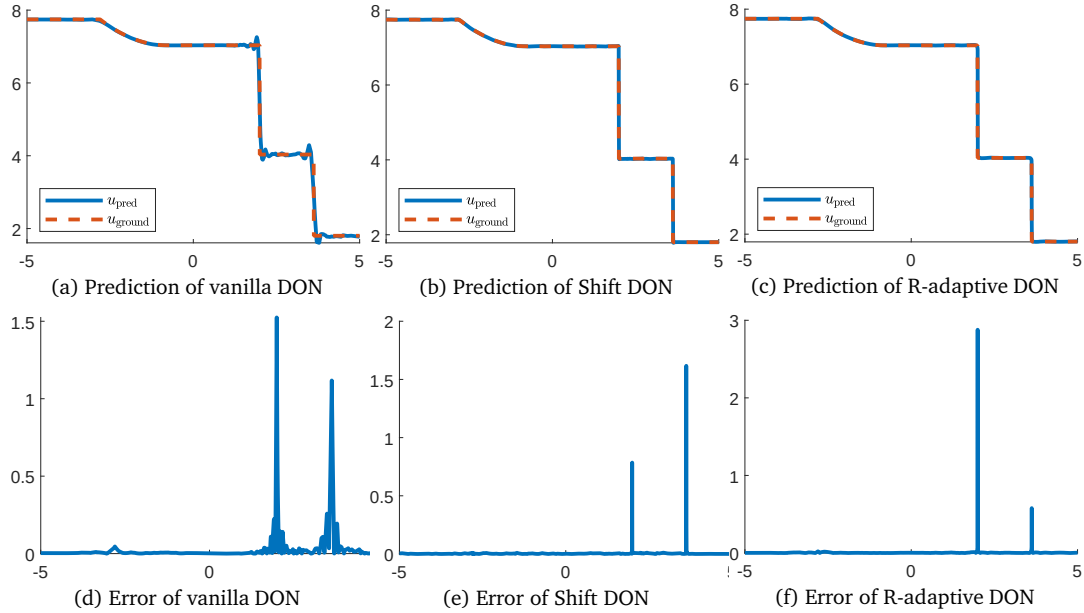

(a) Initial $\rho_0$ and $u_0$

(b) Initial $E_0$ and the final output $E$

(c) Adaptive coordinate and weight

(d) Adaptive solution and weight

Figure 6: Illustration of an example of processed data for Sod shock tube problem.




(a) Prediction of vanilla DON (b) Prediction of Shift DON (c) Prediction of R-adaptive DON

(d) Error of vanilla DON (e) Error of Shift DON (f) Error of R-adaptive DON

Figure 7: Illustration of an example of outputs for Sod shock tube problem.

## 6. Conclusion and discussion

In this paper, we have proposed a DeepONet learning framework based on the R-adaptive method to address the limitations of vanilla DeepONet representations. Inspired by the introduction of adaptive coordinates in R-adaptive methods, our framework tackles the challenge of representing problems with local singularities by separately learning the adaptive coordinate transform function and the corresponding solution over the computation domain. Additionally, we have derived two solution-dependent weighting strategies in the training process to reduce the final error.

We have established an upper bound on the reconstruction error of DeepONet using error estimation from the piecewise linear interpolation and theoretically demonstrated that our R-adaptive DeepONet framework can reduce this upper bound, indicating its potential for problems with local singularities or discontinuities.

In numerical experiments, we selected several typical partial differential equations with local singularities and used the R-adaptive DeepONet to solve them. We compared its results with those of vanilla DeepONet and Shift-DeepONet. It is shown that R-adaptive DeepONet generally outperforms vanilla DeepONet with smaller approximation errors.

Furthermore, we observed that the Shift-DON method performs well in most cases due to its straightforward and simple structure. However, due to its inherent reliance on using continuous functions to approximate discontinuous solutions, it inevitably exhibits Gibbs phenomena near points of discontinuity. Additionally, to achieve sufficiently accurate solutions, Shift-DON typically requires a large amount of training

data to capture features such as the locations of discontinuities. Therefore, when the training data is limited, Shift-DON may struggle to achieve the desired accuracy.

On the other hand, R-adaptive DON handles the discontinuity of PDE solutions by introducing adaptive coordinate transformations, achieving a composite discontinuous representation through the locally constant nature of the adaptive coordinates near discontinuities. Both the adaptive solution and the adaptive coordinates are relatively smooth functions, enabling effective training with a smaller amount of data. As a result, R-adaptive DON has an advantage over Shift-DON in scenarios where adaptive sampling is feasible, as it can achieve effective training with fewer data points. We have also observed that in cases where the solution exhibits large gradients rather than discontinuities, R-adaptive DON outperforms Shift-DON. We will conduct further research on this point in the future.

## Acknowledgments

The authors wish to thank the referees for their constructive suggestions and comments that helped us to improve the presentation of this work.

This work was partially supported by the NSF of China (Grant 12171237), by the Ministry of Science and Technology of China (Grant 2020YFA0713803), and by the NSFC Major Research Plan (Grants 92270001, 92370205).

## Appendix A. A brief introduction to R-adaptive method and equidistribution

In this appendix, we provide a brief introduction to the R-adaptive method and its associated equidistribution principle, see [4, 19] for details.

Suppose that we have a PDE with solution $u(x,t)$, which is posed in a physical domain $\Omega_P \subset \mathbb{R}^d$ with independent spatial variable $x \in \mathbb{R}^d$ for each time $t$. Conceptually, an R-adaptive method generates a moving mesh, continuously mapping a suitable computational space $\Omega_C$ into $\Omega_P$. To achieve this, we assume that a computational coordinate $\xi \in \mathbb{R}^d$ is continuously mapped to the physical coordinate so that $x = x(\xi,t)$. The basis of the R-adaptive methods is that a fixed set of mesh grids (with fixed connectivity) in $\Omega_C$ is moved by this map to a moving set of grids in $\Omega_P$ where the solution is developing an interesting structure. As a result, a fixed set of basis functions (corresponding to the fixed mesh grids) in $\Omega_C$ is mapped to the adaptive basis functions in $\Omega_P$ for each $t$. We write the function in computational coordinate $\xi$ corresponding to $u(x,t)$ as $\tilde{u}(\xi,t) = u(x(\xi,t),t)$. The structure of the function set $\{\tilde{u}(\xi,t)\}_t$ is much less complicated than $\{u(x,t)\}_t$, allowing us to linearly reconstruct $\{\tilde{u}(\xi,t)\}_t$ with fewer basis functions than $\{u(x,t)\}_t$.

The equidistribution principle plays a fundamental role in the mesh adaptation process. This concept, originating from de Boor [11], is a powerful method for identifying a suitable mapping. To implement it, we introduce a (time-dependent) Stieltjes mea-

sure $\rho(x,t)\mathrm{d}x$ into the physical domain. The scalar function $\rho(x,t) > 0$, known as the mesh density specification function (or monitor function), is designed to be large in regions of $\Omega_P$ where the mesh grids need to be clustered. This function is often defined indirectly via the solution, such that $\rho(x,t) = \rho(x, u(x,t), \nabla u(x,t), \dots, t)$. We do not consider the specific choice of the function $\rho$ here. More detailed discussions can be found in [19].

Now introduce an arbitrary non-empty set $K \subset \Omega_C$ in the computational domain, with a corresponding image set $x(K,t) \subset \Omega_P$. The map $x(\cdot,t)$ equidistributes the respective density function $\rho$ if the Stieltjes measure of $K$ and $x(K,t)$ normalized over the measure of their respective domains are the same. This implies that

$$\frac{\int_K \mathrm{d}\xi}{\int_{\Omega_C} \mathrm{d}\xi} = \frac{\int_{x(K,t)} \rho(x,t)\mathrm{d}x}{\int_{\Omega_P} \rho(x,t)\mathrm{d}x}.$$

It follows from a change of variables that

$$\frac{\int_K \mathrm{d}\xi}{\int_{\Omega_C} \mathrm{d}\xi} = \frac{\int_K \rho(x(\xi,t),t)|J(\xi,t)|\mathrm{d}\xi}{\int_{\Omega_P} \rho(x,t)\mathrm{d}x},$$

where

$$J(\xi,t) = \det\left(\frac{\partial x(\xi,t)}{\partial \xi}\right).$$

As the set $K$ is arbitrary, the map $x(\xi,t)$ must obey the identity

$$\rho\big(x(\xi,t),t\big)|J(\xi,t)| = \frac{\int_{\Omega_P} \rho(x,t)\mathrm{d}x}{\int_{\Omega_C} \mathrm{d}\xi} =: \sigma(t). \tag{A.1}$$

We shall refer to (A.1) as the equidistribution equation, and it must always be satisfied by the map $x(\xi,t)$. By solving the mesh equation (A.1) and the original problem simultaneously, we can obtain the adaptive mesh and the corresponding adaptive solution.

## Appendix B. Proof of Theorem 4.4

Recall that with initial data $\bar{u}_\zeta = \bar{u}_0(\cdot - \zeta)$, the solution at $t = T$ can be written as

$$\mathcal{G}(\bar{u}_\zeta)(x) = \bar{u}(x - aT - \zeta) = \mathbb{1}_{[-\pi/2,\pi/2]}(x - aT - \zeta).$$

Given $\delta > 0$, let

$$\begin{aligned}\mathcal{G}_\delta(\bar{u}_\zeta)(x) = \frac{1}{\delta}\sigma\left(x + \frac{\pi}{2} + \frac{\delta}{2} - aT - \zeta\right) - \frac{1}{\delta}\sigma\left(x + \frac{\pi}{2} - \frac{\delta}{2} - aT - \zeta\right)\\ - \frac{1}{\delta}\sigma\left(x - \frac{\pi}{2} + \frac{\delta}{2} - aT - \zeta\right) + \frac{1}{\delta}\sigma\left(x - \frac{\pi}{2} - \frac{\delta}{2} - aT - \zeta\right),\end{aligned}$$

where $\sigma$ is the rectified linear unit (ReLU). We have that $\mathcal{G}_\delta \to \mathcal{G}$ as $\delta \to 0$, or

$$\|\mathcal{G}_\delta(\bar{u}) - \mathcal{G}(\bar{u})\|_2^2 = 4\int_0^{\delta/2} \left(\frac{x}{\delta}\right)^2 \mathrm{d}x = \frac{\delta}{6}, \quad \forall \bar{u} \sim \mu. \tag{B.1}$$

Since $\delta$ is arbitrary, we can try to approximate $\mathcal{G}_\delta$ instead of $\mathcal{G}$. We divide the proof into the following four steps:

**Step 1:** In the first step, we divide the object operator $\mathcal{G}_\delta$ into two parts. For each $\bar{u} \sim \mu$, we introduce a coordinate transform

$$x = x(\xi) : [-\pi, \pi] \ \to \ [-\pi, \pi]$$

to its image function $\mathcal{G}_\delta(\bar{u})$, referred to as $u$ for convenience, satisfying the equidistribution relation $(\rho x_\xi)_\xi = 0$, for the mesh density function

$$\rho(x) = \sqrt{1 + (\pi^2 - 2\pi\delta)u_x^2},$$

and the boundary conditions $x(-\pi) = -\pi, x(\pi) = \pi$. So we can get the object function in the transformed variable $\tilde{u}(\xi) = u(x(\xi))$. For example, when $\zeta = -aT$, i.e.

$$\begin{aligned} u(x) = &\frac{1}{\delta}\sigma\left(x + \frac{\pi}{2} + \frac{\delta}{2}\right) - \frac{1}{\delta}\sigma\left(x + \frac{\pi}{2} - \frac{\delta}{2}\right) \\ &- \frac{1}{\delta}\sigma\left(x - \frac{\pi}{2} + \frac{\delta}{2}\right) + \frac{1}{\delta}\sigma\left(x - \frac{\pi}{2} - \frac{\delta}{2}\right), \end{aligned}$$

we have that

$$\begin{aligned} x(\xi) = &-\pi + \left(2 - \frac{2\delta}{\pi}\right)(\xi + \pi) + \left(\frac{4\delta}{\pi} - 2\right)\sigma\left(\xi + \frac{3\pi}{4}\right) + \left(2 - \frac{4\delta}{\pi}\right)\sigma\left(\xi + \frac{\pi}{4}\right) \\ &+ \left(\frac{4\delta}{\pi} - 2\right)\sigma\left(\xi - \frac{\pi}{4}\right) + \left(2 - \frac{4\delta}{\pi}\right)\sigma\left(\xi - \frac{3\pi}{4}\right), \\ \tilde{u}(\xi) = &\frac{2}{\pi}\sigma\left(\xi + \frac{3\pi}{4}\right) - \frac{2}{\pi}\sigma\left(\xi + \frac{\pi}{4}\right) - \frac{2}{\pi}\sigma\left(\xi - \frac{\pi}{4}\right) + \frac{2}{\pi}\sigma\left(\xi - \frac{3\pi}{4}\right). \end{aligned}$$

Note that for each $\bar{u}$, there is a unique $\tilde{u}(\xi)$ and a strictly increasing $x(\xi)$ corresponding to it. Let us call these two mappings $\tilde{\mathcal{G}}$ and $\mathcal{T}$, respectively, as $\tilde{\mathcal{G}} : \bar{u} \mapsto \tilde{u}(\xi), \mathcal{T} : \bar{u} \mapsto x(\xi)$. So the objective operator $\mathcal{G}_\delta$ is divided into two parts, $\mathcal{G}_\delta(\bar{u}) = \tilde{\mathcal{G}}(\bar{u}) \circ (\mathcal{T}(\bar{u}))^{-1}$.

**Step 2:** Let $\{\xi_i = -\pi + i2\pi/n\}_{i=0}^n$ be the uniform grid nodes on $[-\pi, \pi]$, and $\{\phi_i\}$ be the corresponding piecewise-linear basis functions. For a given $\tilde{u}(\xi)$, define its finite element interpolation $\tilde{u}_I := \sum_{i=0}^n \tilde{u}(\xi_i)\phi_i$. Note that $\tilde{u}$ itself is piece-linear, with four corner points that are $\pi/2$ apart. So $\tilde{u}_I$ is equal to $\tilde{u}$ in the intervals without corner points. Suppose that a corner point is $\xi_j + \eta \in [\xi_j, \xi_{j+1}]$, without loss of generality, we assume that the slope on its left is $0$ and right is $2/\pi$. The $L^2$ error on $[\xi_j, \xi_{j+1}]$ can be

estimated as

$$\begin{aligned}\|\tilde{u}-\tilde{u}_I\|^2_{L^2_{[\xi_j,\xi_{j+1}]}} &= \int_0^{\eta}\left|\frac{n}{\pi^2}\left(\frac{2\pi}{n}-\eta\right)\tilde{\xi}\right|^2 \mathrm{d}\tilde{\xi}+\int_{\eta}^{2\pi/n}\left|\frac{n}{\pi^2}\left(\frac{2\pi}{n}-\eta\right)\tilde{\xi}-\frac{2}{\pi}(\tilde{\xi}-\eta)\right|^2 \mathrm{d}\tilde{\xi}\\ &= \frac{1}{3}\frac{n^2}{\pi^4}\left(\frac{2\pi}{n}-\eta\right)^2\eta^3+\frac{1}{3}\frac{n^2}{\pi^4}\left(\frac{2\pi}{n}-\eta\right)^3\eta^2\\ &= \frac{2}{3\pi^3}n\left(\frac{2\pi}{n}-\eta\right)^2\eta^2 \le \frac{2\pi}{3}n^{-3}.\end{aligned}$$

So we have

$$\|\tilde{u}-\tilde{u}_I\|_{L^2[-\pi,\pi]} \le Cn^{-3/2},$$

where $C=\sqrt{8\pi/3}$ since there are only four corner points.

Since piecewise linear functions can be represented by ReLU neural networks, there is a neural network $\tau:\mathbb{R}\to\mathbb{R}^{n+1}$, mapping $\xi$ to $(\phi_0(\xi),\dots,\phi_n(\xi))$ precisely. And by the universal approximation property of the neural networks, given arbitrary $\varepsilon>0$, there exists a neural network $\beta: L^1(\mathbb{T})\cup L^\infty(\mathbb{T})\to\mathbb{R}^{n+1}$ such that

$$\sup_{\bar{u}\sim\mu}\|\beta(\bar{u})-(\tilde{u}(\xi_0),\dots,\tilde{u}(\xi_n))\|_{l^\infty}<\varepsilon.$$

Then,

$$\begin{aligned}&\|\tilde{\mathcal{G}}(\bar{u})-\beta(\bar{u})\cdot\tau(\cdot)\|_{L^2[-\pi,\pi]}\\ &\le \|\tilde{u}-\tilde{u}_I\|_2+\|\tilde{u}_I-\beta(\bar{u})\cdot\tau(\cdot)\|_2\\ &\lesssim n^{-3/2}+\frac{\varepsilon}{n}\lesssim n^{-3/2}.\end{aligned}$$

Let $\tilde{\mathcal{G}}_{\theta_G}$ be a DeepONet with trunk net $\tau$ and branch net $\beta$, then we have

$$\|\tilde{\mathcal{G}}(\bar{u})-\tilde{\mathcal{G}}_{\theta_G}(\bar{u})\|_{L^2[-\pi,\pi]}\lesssim n^{-3/2}. \tag{B.2}$$

**Step 3:** For $x(\xi)$, define its finite element interpolation

$$x_I(\xi):=\sum_{i=0}^{n}x(\xi_i)\phi_i(\xi).$$

Since $x_I(\xi)$ is strictly increasing, we denote its inverse as $\xi_I(x)$. $\xi_I(x)$ is linear in each interval $[x_i,x_{i+1}]$, where $x_i=x(\xi_i)$, and $\xi_I(x_i)=\xi(x_i)=\xi_i$ for each $i$, where $\xi(x)$ is the inverse of $x(\xi)$. So $\xi_I(x)$ is equal to $\xi(x)$ in those $[x_i,x_{i+1}]$ without corner points. Suppose a corner point of $x(\xi)$ is $\xi_j+\eta\in[\xi_j,\xi_{j+1}]$, without loss of generality, we assume that the slope on its left is $2-2\delta/\pi$ and right is $2\delta/\pi$.

For simplicity, let $k_1=2-2\delta/\pi, k_2=2\delta/\pi$. We can calculate that the corner point is $(\xi_j+\eta, x_j+k_1\eta)$, and

$$x_{j+1}=x_j+k_1\eta+k_2\left(\frac{2\pi}{p}-\eta\right).$$

So we have

$$x(\xi) = \begin{cases} k_1(\xi - \xi_j) + x_j, & \xi \in [\xi_j, \xi_j + \eta], \\ k_2(\xi - \xi_j - \eta) + x_j + k_1\eta, & \xi \in [\xi_j + \eta, \xi_{j+1}], \end{cases}$$
$$x_I(\xi) = k_3(\xi - \xi_j) + x_j, \qquad \xi \in [\xi_j, \xi_{j+1}],$$

where

$$k_3 = \frac{n}{2\pi}\left(k_1\eta + k_2\left(\frac{2\pi}{n} - \eta\right)\right).$$

Correspondingly,

$$\xi(x) = \begin{cases} \dfrac{1}{k_1}(x - x_j) + \xi_j, & x \in [x_j, x_j + k_1\eta], \\ \dfrac{1}{k_2}(x - x_j - k_1\eta) + \xi_j + \eta, & x \in [x_j + k_1\eta, x_{j+1}], \end{cases}$$
$$\xi_I(x) = \frac{1}{k_3}(x - x_j) + \xi_j, \qquad x \in [x_j, x_{j+1}].$$

Then the $L^2$ error on $[x_j, x_{j+1}]$ with respect to $x$ can be estimated as

$$\begin{aligned} &\|\xi(x) - \xi_I(x)\|^2_{L^2_{[x_j, x_{j+1}]}} \\ &= \int_{x_j}^{x_{j+1}} |\xi(x) - \xi_I(x)|^2 \mathrm{d}x \\ &= \int_{x_j}^{x_j + k_1\eta} \left|\left(\frac{1}{k_1} - \frac{1}{k_3}\right)(x - x_j)\right|^2 \mathrm{d}x \\ &\quad + \int_{x_j + k_1\eta}^{x_{j+1}} \left|\left(\frac{1}{k_2} - \frac{1}{k_3}\right)(x - x_j - k_1\eta)\right|^2 \mathrm{d}x \\ &= \int_0^{k_1\eta} \left|\left(\frac{1}{k_1} - \frac{1}{k_3}\right)\tilde{x}\right|^2 \mathrm{d}\tilde{x} + \int_0^{k_2(2\pi/n - \eta)} \left|\left(\frac{1}{k_2} - \frac{1}{k_3}\right)\tilde{x}\right|^2 \mathrm{d}\tilde{x} \\ &= \frac{1}{3}\left(\left(\frac{1}{k_1} - \frac{1}{k_3}\right)^2 (k_1\eta)^3 + \left(\frac{1}{k_2} - \frac{1}{k_3}\right)^2 \left(k_2\left(2\frac{2\pi}{n} - \eta\right)\right)^3\right) \\ &= \frac{1}{3}\frac{(k_1 - k_2)^2\eta^2(2\pi/n - \eta)^2}{k_1\eta + k_2(2\pi/n - \eta)} \\ &\le \frac{1}{3}\frac{(k_1 - k_2)^2}{k_1}\eta\left(\frac{2\pi}{n} - \eta\right)^2 \\ &\le \frac{1}{3}\frac{(k_1 - k_2)^2}{k_1}\frac{4}{27}\left(\frac{2\pi}{n}\right)^3. \end{aligned}$$

Since $(k_1 - k_2)^2/k_1 \to 2\pi$ as $\delta \to 0$, we have the result

$$\|\xi(x) - \xi_I(x)\|_{L^2_{[x_j, x_{j+1}]}} \lesssim n^{-3/2}.$$

Similar to Step 2, we can build a DeepONet $\mathcal{T}_{\theta_T}$ such that

$$\left\|(\mathcal{T}(\bar{u}))^{-1} - (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2_{[-\pi,\pi]}} \lesssim n^{-3/2}. \tag{B.3}$$

**Step 4:** It is obvious that

$$\begin{aligned}
&\left\|\mathcal{G}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \\
\leq\ & \|\mathcal{G}(\bar{u}) - \mathcal{G}_\delta(\bar{u})\|_{L^2} + \left\|\mathcal{G}_\delta(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \\
\leq\ & \|\mathcal{G}(\bar{u}) - \mathcal{G}_\delta(\bar{u})\|_{L^2} + \left\|\tilde{\mathcal{G}}(\bar{u}) \circ (\mathcal{T}(\bar{u}))^{-1} - \tilde{\mathcal{G}}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \\
&\quad + \left\|\tilde{\mathcal{G}}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1} - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \\
=:\ & \mathrm{I}_1 + \mathrm{I}_2 + \mathrm{I}_3.
\end{aligned}$$

From (B.1), it follows that $\mathrm{I}_1 = \delta/6$. For $\mathrm{I}_2$, from (B.3), it follows that

$$\mathrm{I}_2 \leq \mathrm{Lip}\big(\tilde{\mathcal{G}}(\bar{u})(\xi)\big)\left\|(\mathcal{T}(\bar{u}))^{-1}(x) - (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}(x)\right\|_{L^2} \lesssim \frac{2}{\pi} n^{-3/2}.$$

Further, for $\mathrm{I}_3$, from (B.2), it follows that

$$\begin{aligned}
\mathrm{I}_3 &= \int_{\mathbb{T}} \left|\tilde{\mathcal{G}}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}(x) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}(x)\right|^2 \mathrm{d}x \\
&= \int_{\mathbb{T}} \left|\tilde{\mathcal{G}}(\bar{u})(\xi) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u})(\xi)\right|^2 (\mathcal{T}_{\theta_T}(\bar{u}))_\xi \mathrm{d}\xi \\
&\leq \left(2 - \frac{2}{\pi}\delta\right) \left\|\tilde{\mathcal{G}}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u})\right\|_{L^2}^2 \lesssim \left(2 - \frac{2}{\pi}\delta\right) n^{-3}.
\end{aligned}$$

Hence,

$$\left\|\mathcal{G}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \lesssim \frac{\delta}{6} + \sqrt{\left(2 - \frac{2}{\pi}\delta\right)} n^{-3/2} + \frac{2}{\pi} n^{-3/2}.$$

Since $\delta$ is arbitrary, we have the final result

$$\left\|\mathcal{G}(\bar{u}) - \tilde{\mathcal{G}}_{\theta_G}(\bar{u}) \circ (\mathcal{T}_{\theta_T}(\bar{u}))^{-1}\right\|_{L^2} \lesssim n^{-3/2}, \quad \forall \bar{u} \sim \mu.$$

## Appendix C. Proof of Theorem 4.6

It is well known that the inviscid Burgers' equation can be solved using the method of characteristics. For general initial data $\bar{u}(x) = -\sin(x - \zeta)$, the solution $u(x,t)$ is given by

$$u(x,t) = \begin{cases} -\sin\left(\Psi_T^{-1}(x - \zeta + 2\pi)\right), & \text{if} \quad x < \zeta, \\ -\sin\left(\Psi_T^{-1}(x - \zeta)\right), & \text{if} \quad x \geq \zeta, \end{cases}$$

where

$$\Psi_T(x_0) = x_0 - T\sin(x_0 - \zeta)$$

is the characteristic mapping associated with the initial data.

Similar to the proof of Theorem 4.4, we first approximate the solution of the equation using a simple continuous function. Here, we take the case $\bar{u}(x) = -\sin(x-\zeta)$ as an example. For the more general case of $\bar{u}(x) = -\sin(x-\zeta)$, we can leverage the translation invariance property of the Burgers' equation to obtain similar results by shifting the solution accordingly.

At time $t = T > 1$, characteristic curves intersect at $x = \pi$, leading to the formation of a shock and causing the solution to become discontinuous at that point. To address this discontinuity, we employ a continuous function to approximate the solution.

Given $\delta > 0$, we define a continuous function $u_\delta(x)$ on the interval $[0, 2\pi]$ satisfying the following conditions:

1. For $x \in [0, \pi - \delta] \cup [\pi + \delta, 2\pi]$, the function $u_\delta(x)$ coincides with the original solution $u(x, T)$

$$u_\delta(x) = u(x, T), \quad x \in [0, \pi - \delta] \cup [\pi + \delta, 2\pi].$$

2. In the interval $[\pi - \delta, \pi + \delta]$, the function $u_\delta(x)$ is defined as a linear function that smoothly connects the values at $x = \pi - \delta$ and $x = \pi + \delta$, ensuring continuity of the approximation.

As $\delta \to 0$, the function $u_\delta(x)$ converges to the discontinuous solution $u(x, T)$. Specifically, for any $\epsilon_1 > 0$, there exists $\delta > 0$ such that

$$\|u_\delta - u(\cdot, T)\|_2^2 < \epsilon_1.$$

This ensures that the continuous approximation $u_\delta(x)$ can approach the original solution $u(x, T)$ in the $L^2$-norm as closely as desired, depending on the choice of $\delta$.

Thus, we provide an approximation $\mathcal{G}_\delta$ of the Burgers' equation solution operator $\mathcal{G}_{\text{Burg}}$, where $\mathcal{G}_\delta(\bar{u}) = u_\delta$, and

$$\|\mathcal{G}_\delta(\bar{u}) - \mathcal{G}_{\text{Burg}}(\bar{u})\|_2^2 < \epsilon_1, \quad \forall \bar{u} \sim \mu.$$

Then, similar to the Step 1 in Appendix B, we now divide the operator $\mathcal{G}_\delta$ into two parts. Consider again the initial condition $\bar{u}(x) = -\sin(x - \pi)$. We introduce a coordinate transform $x = x(\xi) : [0, 2\pi] \to [0, 2\pi]$, satisfying the equidistribution relation for the mesh density function

$$\rho(x) = \sqrt{1 + u_x^2}$$

with the boundary conditions $x(0) = 0$ and $x(2\pi) = 2\pi$. To handle the discontinuity at $x = \pi$, we truncate the mesh density function $\rho(x)$ so that it remains constant for $x \in [0, \pi - \delta] \cup [\pi + \delta, 2\pi]$. This truncation is reasonable because the gradient of $u_\delta$ in the interval $[\pi - \delta, \pi + \delta]$ is large, significantly different from that in $[0, \pi - \delta] \cup [\pi + \delta, 2\pi]$.

After truncation, we set the value of $\rho(x)$ in $[0, \pi-\delta] \cup [\pi+\delta, 2\pi]$ such that its ratio to the value in $[\pi-\delta, \pi+\delta]$ is $\delta : (\pi-\delta)$. Then we have

$$x(\xi) = \begin{cases} 4\dfrac{\pi-\delta}{\pi}\xi, & \xi \in \left[0, \dfrac{\pi}{4}\right], \\ \pi - \delta + 4\dfrac{\delta}{\pi}\left(\xi - \dfrac{\pi}{4}\right), & \xi \in \left[\dfrac{\pi}{4}, \dfrac{3\pi}{4}\right], \\ \pi + \delta + 4\dfrac{\pi-\delta}{\pi}\left(\xi - \dfrac{3\pi}{4}\right), & \xi \in \left[\dfrac{3\pi}{4}, 2\pi\right], \end{cases}$$

$$\tilde{u}(\xi) = \begin{cases} -\sin\left(\Psi_T^{-1}\left(4\dfrac{\pi-\delta}{\pi}\xi + \pi\right)\right), & \xi \in \left[0, \dfrac{\pi}{4}\right], \\ \text{linear}, & \xi \in \left[\dfrac{\pi}{4}, \dfrac{3\pi}{4}\right], \\ -\sin\left(\Psi_T^{-1}\left(\delta + 4\dfrac{\pi-\delta}{\pi}\left(\xi - \dfrac{3\pi}{4}\right)\right)\right), & \xi \in \left[\dfrac{3\pi}{4}, 2\pi\right]. \end{cases}$$

By this approach, we decompose $u_\delta(x)$ into two parts, and the gradients of both $x(\xi)$ and $\tilde{u}(\xi)$ can be well controlled. The remainder of the proof follows similar steps as the linear transport equation case and is omitted here.

## References


[1] W. BANGERTH AND R. RANNACHER, *Adaptive finite element methods for differential equations*, Lectures in Mathematics ETH Zürich, Birkhäuser Verlag, 2003.

[2] K. BHATTACHARYA, B. HOSSEINI, N. B. KOVACHKI, AND A. M. STUART, *Model reduction and neural networks for parametric PDEs*, SMAI J. Comput. Math. 7 (2021), 121–157.

[3] J. BLECHSCHMIDT AND O. G. ERNST, *Three ways to solve partial differential equations with neural network — A review*, GAMM-Mitt. 44 (2021), e202100006.

[4] C. J. BUDD AND J. F. WILLIAMS, *Moving mesh generation using the parabolic Monge-Ampère equation*, SIAM J. Sci. Comput. 31 (2009), 3438–3465.

[5] S. CAI, Z. MAO, Z. WANG, M. YIN, AND G. E. KARNIADAKIS, *Physics-informed neural networks (PINNs) for fluid mechanics: A review*, Acta Mech. Sin. 37 (2021), 1727–1738.

[6] S. CAI, Z. WANG, L. LU, T. A. ZAKI, AND G. E. KARNIADAKIS, *DeepM & Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks*, J. Comput. Phys. 436 (2021), 110296.

[7] H. D. CENICEROS AND T. Y. HOU, *An efficient dynamically adaptive mesh for potentially singular solutions*, J. Comput. Phys. 172 (2001), 609–639.

[8] T. CHEN AND H. CHEN, *Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems*, IEEE Trans. Neural Networks Learn. Syst. 6 (1995), 911–917.

[9] P. CLARK DI LEONI, L. LU, C. MENEVEAU, G. E. KARNIADAKIS, AND T. A. ZAKI, *Neural operator prediction of linear instability waves in high-speed boundary layers*, J. Comput. Phys. 474 (2023), 111793.

[10] C. M. DAFERMOS AND C. M. DAFERMOS, *Hyperbolic Conservation Laws in Continuum Physics*, Vol. 3, Springer, 2005.

[11] C. DE BOOR, *Good approximation by splines with variable knots. II*, in: Conference on the Numerical Solution of Differential Equations, Lecture Notes in Mathematics, Vol. 363, Springer, (1974), 12–20.
[12] V. DOLEJŠÍ AND M. FEISTAUER, *Discontinuous Galerkin Method. Analysis and Applications to Compressible Flow*, Springer Series in Computational Mathematics, Vol. 48, Springer, 2015.
[13] W. E, *Machine learning and computational mathematics*, Commun. Comput. Phys. 28 (2020), 1639–1670.
[14] W. E AND B. YU, *The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems*, Commun. Math. Stat. 6 (2018), 1–12.
[15] P. S. HADORN, *Shift-Deeponet: Extending Deep Operator Networks for Discontinuous Output Functions*, ETH Zurich, Seminar for Applied Mathematics, 2022.
[16] J. HE, L. LI, J. XU, AND C. ZHENG, *ReLU deep neural networks and linear finite elements*, J. Comput. Math. 38 (2020), 502–527.
[17] J. S. HESTHAVEN AND S. UBBIALI, *Non-intrusive reduced order modeling of nonlinear problems using neural networks*, J. Comput. Phys. 363 (2018), 55–78.
[18] B. HUANG AND J. WANG, *Applications of physics-informed neural networks in power systems — A review*, IEEE Trans. Power Syst. 38 (2023), 572–588.
[19] W. HUANG AND R. D. RUSSELL, *Adaptive Moving Mesh Methods*, Applied Mathematical Sciences, Vol. 174, Springer, 2011.
[20] Y. KHOO, J. LU, AND L. YING, *Solving parametric PDE problems with artificial neural networks*, Eur. J. Appl. Math. 32 (2021), 421–435.
[21] S. KOLLMANNSBERGER ET AL., *Deep Learning in Computational Mechanics*, Springer, 2021.
[22] S. LANTHALER, S. MISHRA, AND G. E. KARNIADAKIS, *Error estimates for DeepONets: A deep learning framework in infinite dimensions*, Trans. Math. Appl. 6 (2022), doi:10.1093/imatrm/tnac001.
[23] S. LANTHALER, R. MOLINARO, P. HADORN, AND S. MISHRA, *Nonlinear Reconstruction for Operator Learning of PDEs with Discontinuities*, Tech. Rep. 2022-42, Seminar for Applied Mathematics, ETH Zürich, 2022.
[24] S. LANTHALER AND A. M. STUART, *The curse of dimensionality in operator learning*, arXiv: 2306.15924, 2023.
[25] J. Y. LEE, S. W. CHO, AND H. J. HWANG, *Hyperdeeponet: Learning operator with complex target function space using the limited resources via hypernetwork*, arXiv:2312.15949, 2023.
[26] Z. LI ET AL., *Fourier Neural Operator for Parametric Partial Differential Equations*, in: International Conference on Learning Representations, 2021.
[27] C. LIN, M. MAXEY, Z. LI, AND G. E. KARNIADAKIS, *A seamless multiscale operator neural network for inferring bubble dynamics*, J. Fluid Mech. 929 (2021), A18.
[28] L. LU, P. JIN, G. PANG, Z. ZHANG, AND G. E. KARNIADAKIS, *Learning nonlinear operators via deeponet based on the universal approximation theorem of operators*, Nat. Mach. Intell. 3 (2021), 218–229.
[29] K. O. LYE, S. MISHRA, AND D. RAY, *Deep learning observables in computational fluid dynamics*, J. Comput. Phys. 410 (2020), 109339.
[30] Z. MAO, A. D. JAGTAP, AND G. E. KARNIADAKIS, *Physics-informed neural networks for high-speed flows*, Comput. Methods Appl. Mech. Engrg. 360 (2020), 112789.
[31] A. PINKUS, *N-Widths in Approximation Theory*, Springer Science & Business Media, 1985.
[32] M. RAISSI, P. PERDIKARIS, AND G. E. KARNIADAKIS, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear*

*partial differential equations*, J. Comput. Phys. 378 (2019), 686–707.
[33] J. SEIDMAN, G. KISSAS, P. PERDIKARIS, AND G. J. PAPPAS, *Nomad: Nonlinear manifold decoders for operator learning*, in: Advances in Neural Information Processing Systems, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (Eds), Vol. 35, Curran Associates Inc., (2022), 5601–5613.
[34] J. SIRIGNANO AND K. SPILIOPOULOS, *DGM: A deep learning algorithm for solving partial differential equations*, J. Comput. Phys. 375 (2018), 1339–1364.
[35] T. TANG, R. LI, AND Z. ZHANG, *Moving Mesh Methods for Partial Differential Equations*, Science Press, 2023.
[36] E. F. TORO, *Riemann solvers and numerical methods for fluid dynamics: A practical introduction*, Springer Science & Business Media, 2013.
[37] S. VENTURI AND T. CASEY, *SVD perspectives for augmenting DeepONet flexibility and interpretability*, Comput. Methods Appl. Mech. Engrg. 403 (2023), 115718.
[38] D. B. WEST, *Introduction to Graph Theory*, Prentice Hall Inc., 1996.
[39] M. YIN, E. ZHANG, Y. YU, AND G. E. KARNIADAKIS, *Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems*, Comput. Methods Appl. Mech. Engrg. 402 (2022), 115027.