Transfer Learning Enhanced DeepONet for Long-Time Prediction of Evolution Equations

Wuzhe Xu,[1*] **Yulong Lu,** [1] **Li Wang** [2]

[1] Department of Mathematics and Statistics, University of Massachusetts Amherst
[2] School of Mathematics, University of Minnesota
wuzhexu@umass.edu, yulonglu@umass.edu, wang8818@umn.edu

Abstract

Deep operator network (DeepONet) has demonstrated great success in various learning tasks, including learning solution operators of partial differential equations. In particular, it provides an efficient approach to predict the evolution equations in a finite time horizon. Nevertheless, the vanilla DeepONet suffers from the issue of stability degradation in the long-time prediction. This paper proposes a *transfer-learning* aided DeepONet to enhance the stability. Our idea is to use transfer learning to sequentially update the DeepONets as the surrogates for propagators learned in different time frames. The evolving DeepONets can better track the varying complexities of the evolution equations, while only need to be updated by efficient training of a tiny fraction of the operator networks. Through systematic experiments, we show that the proposed method not only improves the long-time accuracy of DeepONet while maintaining similar computational cost but also substantially reduces the sample size of the training set.

1 Introduction

Solving partial differential equations (PDEs) through deep learning approach has attracted extensive attention recently. Thanks to the universal approximation theorem of neural networks, it is natural to approximate solutions of PDEs using neural network. Many popular neural network based methods have been proposed recently, such as Deep Ritz Method (Yu et al. 2018), Deep Galerkin Method (Sirignano and Spiliopoulos 2018), Physics Informed Neural Networks (PINNs) (Raissi, Perdikaris, and Karniadakis 2019) and the Weak Adversarial Networks (Zang et al. 2020). In spite of the great success of these methods in solving various PDEs, the neural networks need to be re-trained if one seeks solutions corresponding to different initial conditions (ICs), boundary conditions (BCs) or parameters for the same PDEs. Instead, the recently proposed parametric operator learning methods, such as DeepONet (Lu, Jin, and Karniadakis 2019) and FNO (Li et al. 2020) enable learning of PDEs corresponding to varying BCs or ICs without re-training the networks. However, there is one important caveat in the aforementioned operator neural networks. Namely they are essentially supervised learning and often require solving large number of PDEs to form the training data, which can be extremely expansive, especially when PDEs of interest lie in high dimensional spaces. To overcome this issue, Wang et al. (Wang, Wang, and Perdikaris 2021; Wang and Perdikaris 2021) proposed the physics-informed DeepONet, which uses only the physical information (for instance the governing law of the PDEs) to construct loss function and thus making DeepONet self-supervised. Nevertheless, in practice the physics-informed DeepONets are more difficult to train compared to its vanilla version since the exact differential operators act on the networks and make the convergence behavior highly depends on the underlying physics problem.

Recently DeepONet has also been applied to learning the propagators of evolution equations; see e.g. (Liu and Cai 2022; Wang and Perdikaris 2021). The basic idea is to employ DeepONets to learn the solution operator of a PDE within a short time interval subject to a collection of (random) initial conditions. The solution of the PDE at later times can be computed as recursive actions of the trained network operator on solutions obtained at the prior steps. However, the approximation accuracy of solutions can deteriorate in the long-run for at least two reasons. First, due to the approximation error, the trained DeepONet, as a surrogate propagator, may be expansive even if the exact propagator is non-expansive, which leads to the accumulation of approximation error in time and hence makes it difficult to predict the solution in the long-run. Second, during the time-evolution of PDEs, the functions that a propagator inputs and outputs can vary in time, even though the form of the propagator within a fixed time-slot may remain unchanged (e.g. when the dynamics is autonomous). Taking diffusion equation as an example, one observes that the functions in the range space of the propagator or the semigroup are much smoother than those in the domain, and for this reason, the solutions in later times become increasingly more regular than those in earlier times. Furthermore, some evolution equations may develop various complexities in a long time-horizon, such as turbulence and scale separations. For those equations, iterating a DeepONet surrogate that is usually only trained in a single (short) time-frame using a finite collection of initial functions may fail to capture the correct regularity or complexity of the solutions in the long time.

Transfer learning (Bozinovski and Fulgosi 1976; Do and Ng 2005) is an important class of machine learning tech-

*Wuzhe Xu is the corresponding author.

niques that use one neural network trained for one task for a new neural network trained for a different related task. The idea is that the knowledge or important features of one problem gained by training the former neural nets can be transferred to other problems. Transfer learning has been widely used in image recognition (Yin et al. 2019; Jin, Cruz, and Gonçalves 2020), natural language processing (Ruder et al. 2019) and recently in PINNs (Goswami et al. 2019; Obiols-Sales et al. 2021; Song and Tartakovsky 2022; Desai et al. 2021). To the best of our knowledge, the present work is the first work to employ transfer learning for learning solution operators of evolutionary PDEs.

1.1 Our contributions

We propose a novel physics-informed DeepONet approach based on the transfer learning for predicting time-dependent PDEs. Different from the existing usage of DeepONets in learning the propagators of PDEs where the learned propagators are treated constant in time, we use transfer learning to sequentially update the learned propagators as time evolves. The resulting time-changing DeepONets offer several advantages compared to the vanilla counterparts: (1) the evolving DeepONets can better adapt to the varying complexities associated to the evolution equations; (2) the DeepONets are updated in a computationally efficient way that the hidden layers are frozen once trained and only the parameters in the last layer are re-trained.

We hereby highlight the major contributions of the proposed method:

- Time marching with the transfer-learning tuned DeepONet gives more accurate and robust long-time prediction of solutions of PDEs while still maintaining low computational cost.
- The proposed method is applied to various types of evolutionary PDEs, including the reaction diffusion equations, Allen-Cahn and Cahn-Hilliard equations, the Navie-Stokes equation and multiscale linear radiative transfer equations.
- Through extensive numerical results, we show that our method can significantly reduce the training sample size needed by DeepONet to achieve the same (or even higher) accuracy.

1.2 Related works

Transfer-learning has been previously combined with physics informed neural networks for solving PDEs problems arising from diverse fields, including the phase-field modeling of fracture (Goswami et al. 2019), super-resolution of turbulent flows (Obiols-Sales et al. 2021), training of CNNs on multi-fidelity data (e.g. multi-resolution images of PDE solutions on fine and coarse meshes) (Song and Tartakovsky 2022), etc. In (Chakraborty et al. 2022), transfer-learning was also applied as a domain adaption method for learning solutions of PDEs defined on complex geometries. The recent paper (Desai et al. 2021) proposed a one-shot transfer learning strategy that freezes the hidden layers of a pre-trained PINN and reduces the training neural networks for solving new differential equations to optimizing only the last (linear) layer. This approach eliminates the need of re-training the whole network parameters while still produces high-quality solutions by tuning a small fraction of parameters in the last layer. The present paper marry this transfer learning idea with DeepONet for learning the propagators of evolution equations in order to predict the long time evolution.

While we are finalizing the current paper, we are aware of a recent preprint (Goswami et al. 2022) where transfer learning was exploited together with DeepONet for learning PDEs under conditional shift. The purpose there is to train a source PDE model with sufficient labeled data from one source domain and transfer the learned parameter to a target domain with limited labeled data. The technology developed there is mainly applied for transferring the knowledge of a solution operator trained on a system of PDEs from one domain to another. Different from (Goswami et al. 2022), we leverage transfer learning to successively tuning the surrogate models of propagators learned via physics-informed DeepONet so that the tuned operator networks can adaptively track the evolving propagators that carry evolving inputs and outputs. The proposed approach is proven to be more accurate and robust for learning the long-time evolution of PDEs.

2 Numerical method

Problem set-up Consider the initial boundary value problem for a general evolution equation:

$$\begin{cases} \partial_t f(t,\boldsymbol{x}) = \mathcal{L}(f(t,\boldsymbol{x})), \\ f(t,\boldsymbol{x}) = \phi(\boldsymbol{x}),\ \boldsymbol{x} \in \partial\Omega_x, \\ f(0,\boldsymbol{x}) = f_0(\boldsymbol{x})\,,\ \boldsymbol{x} \in \Omega_x\,. \end{cases} \tag{1}$$

Throughout the paper, we assume that the equations are dissipative in the sense that $\int_{\Omega_x} f\mathcal{L}f d\boldsymbol{x} \leq 0$. Given a time step size Δt, we consider the semi-discrete approximation $f^n(\boldsymbol{x})$ of the solution $f(n\Delta t, \boldsymbol{x})$ to (1) defined by the backward Euler discretization:

$$f^{n+1}(\boldsymbol{x}) = (I - \Delta t\mathcal{L})^{-1} f^n(\boldsymbol{x}) := \mathcal{P}^{\Delta t} f^n(\boldsymbol{x})\,. \tag{2}$$

Our goal is to approximate the propagator

$$\mathcal{P}^{\Delta t} : f^n(\boldsymbol{x}) \mapsto f^{n+1}(\boldsymbol{x})$$

by an operator neural network $\mathcal{P}_{NN}$ so that only one forward pass of the neural network achieves time-marching solutions from one step to the next, and that the evolution dynamics can be captured in a long time-horizon.

It is important to point out that the backward Euler scheme is not the only choice for time-marching. One can extend it to high order time discretization schemes such as Runge-Kutta methods, as long as Δt is chosen such that $\mathcal{P}^{\Delta t}$ is a non-expanding operator. We will make this point more clear in Section 3 and Appendix C.2. To ease the notation, the superscript n, $n+1$ and Δt will be omitted in the following context if it does not cause any confusion.

2.1 Physics-informed DeepONet

Let Ω_x be a compact set in $\mathbb{R}^d$ and let $\mathcal{X}$ be a compact subspace of the space $C(\Omega_x)$ of continuous function defined on Ω_x. Then according to the universal approximation theorem (Chen and Chen 1995), an operator $\mathcal{P} : \mathcal{X} \to \mathcal{X}$ can be

approximated by a parametric operator $\mathcal{P}_{NN}$ with arbitrary accuracy. That is, for any $\varepsilon > 0$, there exists a sufficiently large parametric neural network $\mathcal{P}_{NN}$, such that

$$\int_{\mathcal{X}} \int_{\Omega_x} |\mathcal{P}(f)(\boldsymbol{x}) - \mathcal{P}_{NN}(f)(\boldsymbol{x})|^2 d\boldsymbol{x} d\mu(f) < \varepsilon\,.$$

Here μ denotes a probability measure on $\mathcal{X}$. In practice, μ is chosen as a Gaussian measure induced by the law of a Gaussian random field. Several operator networks have been proposed recently, including DeepONets (Lu, Jin, and Karniadakis 2019; Wang, Wang, and Perdikaris 2021) and various neural operators (Bhattacharya et al. 2020; Li et al. 2020; Kovachki et al. 2021). In this paper, we adopt DeepONet as the basic architecture and refine it with transfer learning. The vanilla DeepONet takes the following form:

$$\begin{aligned}\mathcal{P}_{NN}(f)(\boldsymbol{x};\theta,\xi) &\\ &= \sum_{k=1}^{p} b_k^{NN}(f(\boldsymbol{y}_1),\cdots,f(\boldsymbol{y}_N);\theta) t_k^{NN}(\boldsymbol{x};\xi) \\ &=: \sum_{k=1}^{p} b_k^{NN}(f;\theta) t_k^{NN}(\boldsymbol{x};\xi). \end{aligned} \tag{3}$$

The operator network $\mathcal{P}_{NN}$ consists of two sub-networks: the branch net b^{NN} is paramterized by θ and maps an encoded input function $\{f(\boldsymbol{y}_i)\}_{i=1}^N$ to p scalars b_k^{NN}, and the trunk net $t^{NN} = \{t_k^{NN}\}_{k=1}^p$ is parameterized by ξ and forms a directionary of functions in the output space. Both networks can be modified in practice to fit with various set-ups of PDEs, such as the boundary condition. A diagram of the DeepONets architecture we use in this paper is shown in Figure 2. The vanilla DeepONets is often trained in a supervised fashion and requires pairs of input-output functions. To be more specific, given N_s randomly sampled functions $\{f_s(\boldsymbol{x})\}_{s=1}^{N_s}$ one needs to prepare reference solutions $\{\mathcal{P}(f_s)(\boldsymbol{x})\}_{s=1}^{N_s}$ either analytically or using conventional high-fidelity numerical solvers. Then one trains the $\mathcal{P}_{NN}$ by minimizing the loss function

$$\begin{aligned}\frac{1}{2N_s}\sum_{s=1}^{N_s}\Big(&\|\mathcal{P}_{NN}(f_s)(\cdot;\theta,\xi) - \mathcal{P}(f_s)(\cdot)\|^2_{L^2(\Omega_x)} \\ &+ \|\mathcal{P}_{NN}(f_s)(\cdot;\theta,\xi) - \phi_s(\cdot)\|^2_{L^2(\partial\Omega_x)}\Big).\end{aligned}$$

However, in reality it can be extremely expensive to obtain the outputs $\mathcal{P}(f_s)$, especially when the underlying physical principles are complicated and the dimension of the problem is high. To this end, (Wang, Wang, and Perdikaris 2021) proposed a physics-informed DeepONet which makes the learning procedure above self-supervised. More precisely, we turn to minimizing the new loss function

$$\begin{aligned}\frac{1}{2N_s}\sum_{s=1}^{N_s}\Big(&\|\mathcal{P}^{-1}\big(\mathcal{P}_{NN}(f_s)(\cdot;\theta,\xi)\big) - f_s(\cdot)\|^2_{L^2(\Omega_x)} \\ &+ \|\mathcal{P}_{NN}(f_s)(\cdot;\theta,\xi) - \phi_s(\cdot)\|^2_{L^2(\partial\Omega_x)}\Big).\end{aligned} \tag{4}$$

Note that the introduction of $\mathcal{P}^{-1}$ in (4) completely avoids the evaluations of $\mathcal{P}(f_s)$. The boundary term in (4) can be further eliminated in practice because the networks can be modified to satify the boundary conditions (see e.g. (Lu et al. 2022)). We also observe through numerical experiments that eliminating the the boundary loss can substantially improves the training efficiency. The physics-informed DeepONet has been applied to learning evolution equations (Wang and Perdikaris 2021). For equation (1), instead of first discretizing it in time, they consider time as an additional input variable, and try to learn an operator $\mathcal{P}^I$ that maps the initial condition to the solutions over an time-interval $[0, t_0]$:

$$\mathcal{P}^I : f(0,\boldsymbol{x}) \mapsto f(t,\boldsymbol{x}), \quad \text{for} \quad t \in [0,t_0]\,.$$

The corresponding loss function to be minimized is

$$\begin{aligned}L(\theta,w,\xi) = \frac{1}{2N_s}\sum_{s=1}^{N_s}(&\|\partial_t(\mathcal{P}^I_{NN}(f_s))(t,\boldsymbol{x}) \\ &- \mathcal{L}(\mathcal{P}^I_{NN}(f_s))(t,\boldsymbol{x})\|^2_{L^2(\Omega_x\times[0,t_0])} \\ &+ \|\mathcal{P}^I_{NN} f_s(\boldsymbol{x}) - g_s(\boldsymbol{x})\|^2_{L^2(\partial\Omega_x\times[0,t_0])}),\end{aligned} \tag{5}$$

where $\mathcal{P}^I_{NN}$ is the neural network approximator to $\mathcal{P}^I$. Once trained, $\mathcal{P}^I_{NN}$ can be applied to $f(t_0,\boldsymbol{x})$ to get the solution $f(t,\boldsymbol{x})$ over $[t_0, 2t_0]$. Repeating this process enables one to obtain approximation solutions in any finite time. However, this methodology may suffer from long-time instability. In fact, let us illustrate this using the Allen-Cahn equation (16) in one dimension. Figure 1 shows that the average L^2 errors of approximated solutions learned by DeepONets using both the single-shot loss (4) (labeled as DeepONet) and the time-integrated loss (5) (labeled as CONT DeepONet). Both errors accumulate rapidly as time increases, indicating the instability of the learned DeepONets in prediction of long-time solution. In contrast, our transfer-learning assisted DeepOnet dramatically reduces the error and stabilizes the prediction. For completeness, we also compare them with the Fourier Neural Operator(FNO) (Li et al. 2020), which is another state-of-the-art operator learning method.

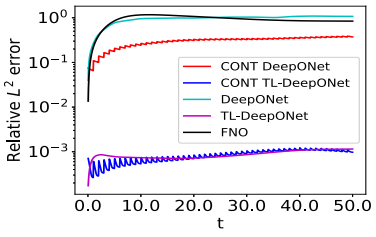

Figure 1: The relative L^2 error in time (defined in (22)) for 1D Allen Cahn (16). The networks with "CONT" refer to networks trained by using (5) whereas the others are trained by using (4). The prefix "TL" means tuned by transfer learning. See implementation details in Appendix C.3.

2.2 DeepONet with transfer learning

The main idea of transfer learning is to train a neural network on a large data set and then partially freeze and apply it to a

related but unseen task. Inspired by (Desai et al. 2021), we employ the transfer learning technique to successively correct the trained DeepONet at the prediction steps: we freeze the majority of the well-trained DeepONet and merely re-train the weights in the last hidden layer of the branch net by fitting the same physics-informed loss (4) defined by the underlying PDEs. To be more precise, by separating the parameters θ in the hidden layers and the parameter w in the last layer of the branch net, we rewrite the branch net as

$$b_k^{NN}(f;\theta,w) = \sum_{j=1}^{q} w_j h_{k,j}(f;\theta)\,,$$

where $h = \{h_{k,j}\}$ are the outputs of the last hidden layer of the branch net and $w = \{w_j\}$ are the weights in the last layer. Inserting this into (3) gives

$$\mathcal{P}_{NN}(f)(\boldsymbol{x};\theta,w,\xi) = \sum_{k=1}^{p}\sum_{j=1}^{q} w_j h_{k,j}^{NN}(f;\theta) t_k^{NN}(\boldsymbol{x};\xi)\,. \tag{6}$$

The architecture of the new operator network is illustrated in Figure 2. In the training step, the optimal parameters (θ^*, w^*, ξ^*) of the DeepONet (6) can be obtained by minimizing the empirical loss (4). Later in each prediction step, we freeze the value of θ^* and ξ^*, but update w^* by re-training the loss (4) with newly-predicted solution as the initial condition. Namely with the predicted solution f_n at step n, we seek w_{n+1}^* defined by

$$\begin{aligned} w_{n+1}^* \in \arg\min_{w} \\ \frac{1}{2N_s}\sum_{s=1}^{N_s}\Big(&\|\mathcal{P}^{-1}\mathcal{P}_{NN}(f)(\boldsymbol{x};\theta^*,w,\xi^*) - f_n(\boldsymbol{x})\|^2_{L^2(\Omega_x)} \\ +&\|\mathcal{P}_{NN}(f)(\boldsymbol{x};\theta^*,w,\xi^*) - \phi(\boldsymbol{x})\|^2_{L^2(\partial\Omega_x)}\Big), n = 1,2,\cdots. \end{aligned} \tag{7}$$

Note that $w_1^* = w^*$. The optimal sequence of weights w_n^* defines a sequence of operator networks $\mathcal{P}_{NN}^n := \mathcal{P}_{NN}^n(\theta^*, w_n^*, \xi^*)$, which can be used to approximate the solution at $t = n\Delta t$ by

$$f(n\Delta t) \approx \mathcal{P}_{NN}^n \circ \mathcal{P}_{NN}^{n-1} \circ \mathcal{P}_{NN}^1(f_0).$$

It is interesting to note that the proposed method shares some similarities with the classical Galerkin approximation. In fact, the operator network (6) can be further rewritten as

$$\begin{aligned} \mathcal{P}_{NN}(f)(\boldsymbol{x}) &= \sum_{j=1}^{q} w_j \left(\sum_{k=1}^{p} h_{k,j}^{NN}(f;\theta) t_k^{NN}(\boldsymbol{x};\xi)\right) \\ &=: \sum_{j=1}^{q} w_j \phi_j(\boldsymbol{x};f)\,, \end{aligned}$$

Observe that ϕ_j playing the role of basis functions in Galerkin methods, and w_j being the corresponding weight. However, unlike most Galerkin methods which often use handcraft bases, such as piecewise polynomials and trigonometric functions, here the bases are learned from the problem itself, and vary with the function they approximate. This seemly minor change reduces substantially the number of bases needed in the output space, as shown by extensive numerical tests in Section 4. To minimize (7), it amounts to solving a system of N equations with q unknowns, where N is the total number of fixed sensors in the branch net. This is achieved by least square minimization. Since $q \ll N$, the computational complexity of finding the least square solution is only $\mathcal{O}(q^2 N)$. In practice, we further reduce the computational complexity by sub-sampling N_c grid points out of N in the transfer learning step.

3 Theoretical result

In this section, we analyze the long time stability of the learned operator $\mathcal{P}_{NN}$. First let $\mathcal{X}$ be a Banach space and assume that the original propagator $\mathcal{P}$ (i.e., $\mathcal{P}^{\Delta t}$ in (2)): $\mathcal{X} \to \mathcal{X}$ is non-expansive such that

$$\|\mathcal{P}\|_{\mathcal{X}} := \sup_{f\in\mathcal{X}, \|f\|_{\mathcal{X}}=1} \|\mathcal{P}f\|_{\mathcal{X}} \le 1. \tag{8}$$

In the case that $\mathcal{X} = L^2(\Omega_x)$, assumption(8) follows from the dissipative assumption of $\mathcal{L}$; see Appendix B.1 for more details. Let $\mathcal{U} \subseteq \mathcal{X}$ be a linear subspace, we also assume that

$$\mathcal{P}f \in \mathcal{U}, \quad \forall f \in \mathcal{U}\,. \tag{9}$$

The theorem below shows that the long-time prediction error of the operator network can be bounded by the loss function.

Theorem 1. Assume (8) and (9) hold. If the neural network approximator $\mathcal{P}^{NN}$ satisfies that the maximum loss over the set $\mathcal{U}$ is less than δ, i.e.

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|\mathcal{P}^{-1}\mathcal{P}_{NN}f - f\|_{\mathcal{X}} \le \delta\,, \tag{10}$$

and that

$$\mathcal{P}_{NN}f \in \mathcal{U}, \quad \forall f \in \mathcal{U}, \tag{11}$$

then the following long-time stability holds

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} \le \delta K(1+\delta)^K. \tag{12}$$

Moreover, if we further assume that

$$\|\mathcal{P}\|_{\mathcal{X}} \le \eta < 1 \tag{13}$$

and (10) holds with $\delta \le \frac{1}{2}(1-\eta)$, then we have

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} \le \delta K\Big(\frac{1+\eta}{2}\Big)^{K-1}. \tag{14}$$

The proof of Theorem 1 is provided in Appendix B.2.

Remark 1. 1. If the error tolerance $\delta = \Delta t^2$ with Δt being the time-discretization stepsize and the number of iterations $K = \frac{T}{\Delta t}$, then (12) becomes

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} \le e^{T\Delta t} T\Delta t\,.$$

When assumption (13) holds, the estimate above improves to

$$\begin{aligned} \sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} &\le \big(\frac{1+\eta}{2}\big)^{\frac{T}{\Delta t}} T\Delta t \\ &\le C\Delta t, \end{aligned}$$

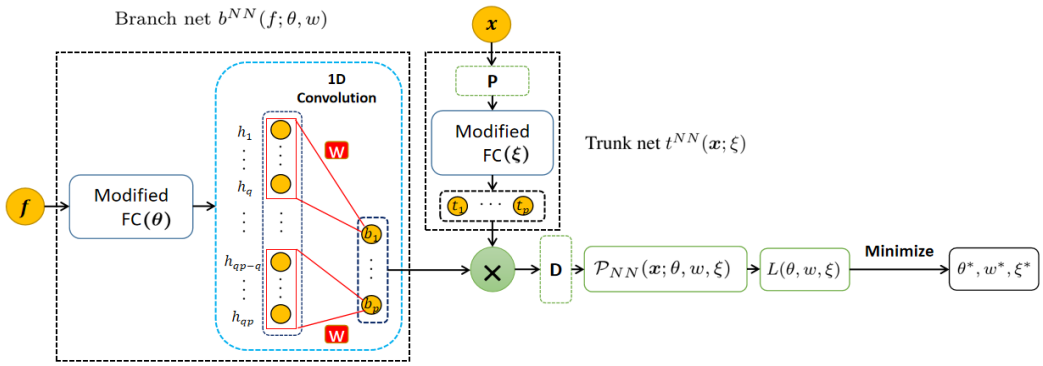

Figure 2: The architecture of transfer learning aided physics-informed DeepONet. Here P and D are optional layers that enforce periodic and Dirichlet boundary conditions, respectively. The block named Modified FC is a modified fully connected neural networks architecture introduced in (Wang, Wang, and Perdikaris 2021). The parameter w (in the red box) denotes the tunable weights in the last hidden layer of the branch net. In the transfer learning step, only w will be re-trained while the θ, ξ are frozen.

where C is a constant independent of T and Δt, suggesting that the prediction error is of the order $\mathcal{O}(\Delta t)$ uniformly in time.

2. We comment on the assumptions made in Theorem 1. In practice, physics-informed loss (7) is trained so that condition (10) is fulfilled for some subspace space $\mathcal{U}$, e.g. $\mathcal{U} = \{e^{i\mathbf{k}\cdot\mathbf{x}}\}_{|\mathbf{k}|\le K_0}$. Assumption (9) holds for such a choice of $\mathcal{U}$ when the evolution equation involves diffusion. Assumption (11) holds in particular when sine or cosine activation function is used in the operator network $\mathcal{P}_{NN}$.

4 Numerical experiments

In this section, we demonstrate the effectiveness of transfer learning enhanced DeepONet and show its advantages over the vanilla DeepONet through several evolutionay PDEs, including reaction diffusion equation, Allen-Cahn and Cahn-Hilliard equations, Navier-Stokes equation and multiscale linear radiative transfer equation. The equations of consideration are equipped with either Dirichlet or periodic boundary conditions. In all the test problems, our goal is to predict the long time evolution of the equations obtained by successive actions of the propagators learned via DeepONets. More concretely, we first build the first-step neural operator approximation $\mathcal{P}^1_{NN}$ to the propagator $\mathcal{P} = \mathcal{P}^{\Delta t}$ by minimizing the physics-informed loss (4) with M training initial data. The operator network $\mathcal{P}^1_{NN}$ is then gradually tuned to $\mathcal{P}^j_{NN}, j = 2, \cdots K$ via updating the weights w in the last-layer of its trunk nets. With the learned (and adjusted) operators $\mathcal{P}^j_{NN}, j = 1, \cdots K$, the solution of a PDE at time $t = K\Delta t$ with an initial condition f_0 can then be obtained approximately by $\mathcal{P}^K_{NN} \circ \cdots \circ \mathcal{P}^1_{NN} f_0$. We remark that the M training data is constructed as a subset of a larger training set of size $N_s \times N_p$, which consists of pointwise evaluations of N_s randomly sampled functions at N_p physical locations (sensors). We refer to Appendix C.1 for detailed discussions on the data generating process and treatment of boundary conditions in various test problems. Choices of parameters for the operator networks and the training process are discussed in the end of Appendix C.1. In all the numerical results to follow, we quantify the performance of the proposed method by measuring the aggregated relative prediction error over a time horizon $[0, T]$; see the precise definition of the relative error in Appendix C.1. The codes used for the numerical experiments will be published on the website https://github.com/woodssss/TL-PI-DeepONet.

4.1 Reaction diffusion equation

Consider the reaction diffusion equation

$$\begin{cases} \partial_t f = d\Delta f + kf^2, x \in \Omega_x := [0,1] \\ f(t,\boldsymbol{x}) = 0, \quad \boldsymbol{x} \in \partial\Omega_x = \{0,1\}, \\ f(0,\boldsymbol{x}) = f_0(\boldsymbol{x}), \end{cases} \tag{15}$$

where $d = k = 0.001$. In this example, we train DeepOnets and our transfer learning enhanced DeepONets using two different loss functions, with one based on the physics-informed loss within a single time-step $\Delta t = 0.05$ (c.f. (4)), and the other based on the aggregated physics-informed loss (5) in the time window $t \in [0, 1]$. We refer to the DeepONets trained using the latter loss as *CONT DeepONets* and reserve *DeepONets* for the one trained by the former loss. The numerical results of different DeepONets with varying training sample sizes M are shown in Table 1. Our proposed method provides more accurate and more robust prediction of the solutions with little extra computational cost. In particular, the prediction error of vanilla DeepONets and CONT DeepONets increase dramatically as time increases from 0.2 to 50, while transfer learning can significantly reduces the error and stabilizes the prediction in the long time. In addition, our method also substantially reduces the size of training data to achieve the same order of prediction accuracy. Note that, since trained only within a single time step, DeepONets take far less training time than the corresponding CONT DeepONets while maintain comparable accuracy. The similar trade-off of accuracy and training cost applies to other experiments. For this reason, in subsequent examples we will only report results

on our proposed method and the vanilla DeepONet, and exclude the results from CONT DeepONets. Note also that the results obtained in Table 1 are for propagators defined by the backward Euler scheme. One can also consider propagators defined by higher order time-discretization schemes and their neural network approximation. We refer the numerical results obtained using the Crank-Nicolson method to Table 8 in Appendix C.2.

Neural network	M	t_1	t_2	$T = 0.2$	$T = 50$
CONT	1000	43030	0.26	7.95e-2	4.17e-1
DeepONet	3000	79375	0.28	1.01e-2	4.15e-2
	10000	83465	0.26	3.40e-3	1.34e-2
CONT	1000	43030	0.93	**5.38e-3**	**1.95e-3**
TL-DeepONet	3000	79375	0.98	**1.87e-3**	**6.88e-4**
	10000	83465	0.93	**1.84e-3**	**4.87e-4**
DeepONet	1000	2575	5.02	2.75e-1	1.69e0
	3000	4313	4.9	1.05e-1	1.60e0
	10000	5854	4.8	9.19e-2	1.87e0
TL-DeepONet	1000	2575	9.86	**1.03e-3**	**1.52e-3**
	3000	4313	7.1	**8.34e-4**	**1.19e-3**
	10000	5854	8.2	**8.19e-4**	**9.05e-4**

Table 1: Results on reaction diffusion equation. Here t_1 is the training time and t_2 is the averaged time of predicting the solution trajectories among the time interval $[0, 50]$ based on 30 test initial conditions. The last two columns to the right are the averaged relative L^2 error within $[0, T]$.

4.2 Allen-Cahn and Cahn-Hilliard equations

In the second example, we consider Allen-Cahn equation

$$\begin{cases} \partial_t f = d_1 \Delta f + d_2 f(1 - f^2), \\ f(0, \boldsymbol{x}) = f_0(\boldsymbol{x}), \end{cases} \tag{16}$$

and Cahn-Hilliard equation

$$\begin{cases} \partial_t f = \Delta g, \\ g = -d_1 \Delta f + d_2(f^3 - f), \\ f(0, \boldsymbol{x}) = f_0(\boldsymbol{x}), \end{cases} \tag{17}$$

both equipped with periodic boundary conditions. They are prototype models for the motion of anti-phase boundaries in crystalline solids. The computational domain is $\Omega := [0, 1]^d$ with $d = 1, 2$. We are interested in learning the propagator $\mathcal{P} = \mathcal{P}^{\Delta t}$ with $\Delta t = 0.05$ and used it to predict the solutions $f(t, \boldsymbol{x})$ for every $t \leq T = 50$. The results on Allen-Cahn equation are shown in Table 2 (1D) and Table 3 (2D). See also Figure 1 for a plot of evolving relative errors on 1D Allen-Cahn equation. Similar results for 1D Cahn-Hilliard equation are presented in Table 4 and Figure 3 compares the snapshots of predicted solutions to the 2D Cahn-Hilliard equation. In all the results, for a fixed d_2, the relative errors increase as d_1 decreases because the transition layers of solutions are increasingly sharper and hence make the numerical resolution more challenging. Similar to the previous example, our proposed method provides more accurate prediction of solutions than the vanilla DeepONets among all the configurations of parameters. We note that the average trajectory prediction times of TL-DeepONets increase for about 3 times compared to those of the vanilla DeepONets while the prediction errors of the former decrease by at least two orders of magnitude.

Neural network	M	d_1=1e-3	d_1=5e-4	d_1=1e-4
DeepONet	1000	1.13e0	1.33e0	1.29e0
	3000	1.33e0	1.18e0	1.23e0
	10000	1.01e0	8.95e-1	1.43e0
TL-DeepONet	1000	**9.25e-4**	**9.64e-4**	**2.16e-2**
	3000	**7.78e-4**	**8.83e-4**	**1.81e-2**
	10000	**5.81e-4**	**7.94e-4**	**1.16e-2**

Table 2: Results on 1D Allen-Cahn equation: the time-average of relative prediction errors within $[0, 50]$. The average trajectory prediction time is 5.1s for DeepONet and 16.1s for TL-DeepONet.

Neural network	M	d_1=4e-3	d_1=2e-3	d_1=1e-3
DeepONet	1000	9.96e-1	1.01e0	1.02e0
	10000	9.96e-1	1.00e0	1.00e0
TL-DeepONet	1000	**6.54e-3**	**8.43e-3**	**1.01e-2**
	10000	**4.96e-3**	**6.46e-3**	**9.01e-3**

Table 3: Results on 2D Allen-Cahn equation: the time-average of the relative prediction errors within $[0, 10]$. The average trajectory prediction time is 6.1s for DeepONet and 29.5s for TL-DeepONet.

Neural network	M	d_1=4e-6	d_1=2e-6	d_1=1e-6
DeepONet	1000	9.43e-1	9.47e-1	9.44e-1
	3000	9.54e-1	9.58e-1	9.39e-1
	10000	9.65e-1	9.58e-1	9.38e-1
TL-DeepONet	1000	**1.04e-2**	**1.36e-2**	**4.25e-2**
	3000	**8.11e-3**	**9.04e-3**	**3.86e-2**
	10000	**2.29e-3**	**7.89e-3**	**3.03e-2**

Table 4: Results on 1D Cahn-Hilliard equation: the time-average of the relative prediction errors within $[0, 50]$. the average trajectory prediction time is 4.4s for DeepONet and 12.3s for TL-DeepONet.

4.3 Navier-Stokes equation

Consider the 2D Navier-Stokes equation in the vorticity form:

$$\begin{cases} \partial_t w(\boldsymbol{x}, t) + u(\boldsymbol{x}, t) \cdot \nabla w(\boldsymbol{x}, t) = \nu \Delta w(\boldsymbol{x}, t) + f(\boldsymbol{x}), \\ w(\boldsymbol{x}, 0) = w_0(\boldsymbol{x}) \end{cases} \tag{18}$$

with periodic boundary condition and source $f(\boldsymbol{x}) = 0.1(\sin(2\pi(x + y)) + \cos(2\pi(x + y)))$. We would like to learn the propagator $\mathcal{P}^{\Delta t}$ with $\Delta t = 0.01$ and apply it to predict the solution $w|_{(0,1)^2 \times (\Delta t, T)}$. Table 5 shows the results with varying values of viscosity ν. Note that the prediction

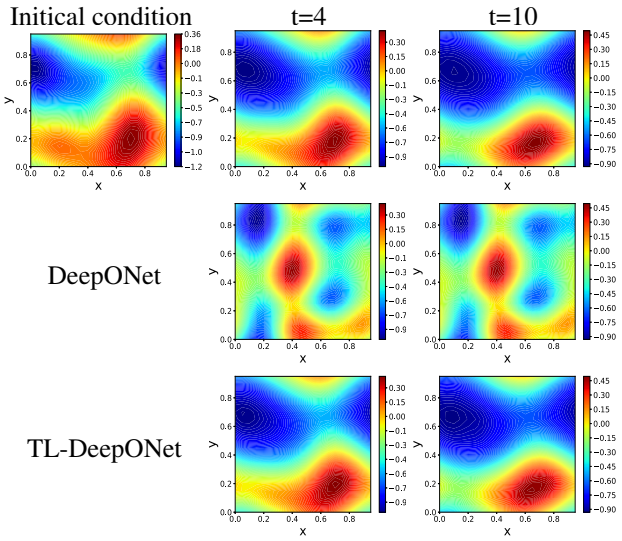

Figure 3: Results on 2D Cahn-Hilliard equation: snapshots of reference solutions (top), and of approximate solutions predicted by DeepONet (middle) and TL-DeepONet (bottom).

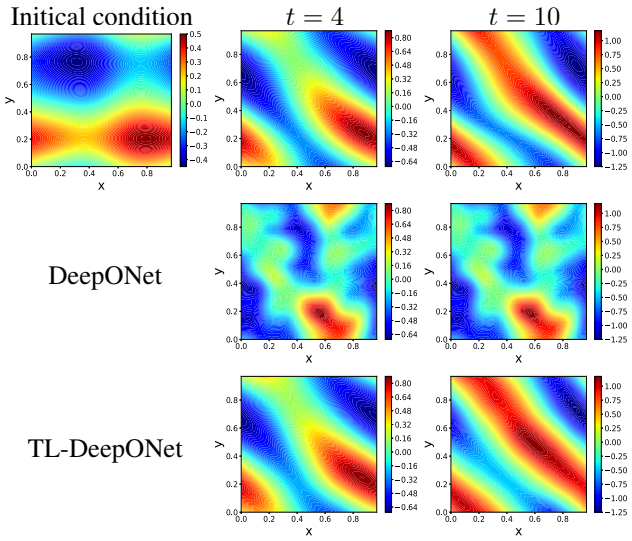

Figure 4: Results on Navier-Stokes equation with $\nu = 0.001$: snapshots of reference solutions (top), and of approximate solutions predicted by DeepONet (middle) and TL-DeepONet (bottom).

error increases as ν decreases. TL-DeepONets reduces the errors of DeepONets by two orders of magnitudes although the prediction time of the former increases for less than 4 times. Figure 4 shows the snapshots of solutions to (18) with $\nu = 0.001$ at two different times.

Neural network	ν=1e-1	ν=1e-2	ν=1e-3	ν=1e-4
DeepONet	9.95e-1	1.02e0	9.96e-1	1.04e0
TL-DeepONet	**1.41e-2**	**1.07e-2**	**3.35e-2**	**9.42e-2**

Table 5: Results on 2D Navier-Stokes equation: the relative prediction errors within $[0, 10]$. The average trajectory prediction time is 5.3s for DeepONet and 24.8s for TL-DeepONet.

4.4 Multiscale linear radiative transfer equation

Consider the linear multiscale radiative transfer equation:

$$\begin{cases} \varepsilon \partial_t f + \boldsymbol{v} \cdot \nabla f = \frac{1}{\varepsilon} \mathcal{L} f, \ t \in [0, T], \ (\boldsymbol{x}, \boldsymbol{v}) \in \Omega_x \times \mathcal{S}^{d-1}, \\ f(t, \boldsymbol{x}, \boldsymbol{v}) = \phi(\boldsymbol{x}), \ (\boldsymbol{x}, \boldsymbol{v}) \in \Gamma_-, \\ f(0, \boldsymbol{x}, \boldsymbol{v}) = f_0(\boldsymbol{x}, \boldsymbol{v}). \end{cases} \tag{19}$$

Here $\varepsilon > 0$ is the Knudsen number which is a dimensionless parameter that determines the physical regime of the equation, $\mathcal{L}(f) = \frac{1}{|\mathcal{S}^{d-1}|} \int_{\mathcal{S}^{d-1}} f d\boldsymbol{v} - f =: \langle f \rangle - f$, and $\Gamma_- = \{(x, v) : x \in \partial\Omega_x, \ v \cdot n_x < 0\}$ is the inflow part of the boundary. In this example, we aim to learn the propagator $\mathcal{P}^{\Delta t}$ with $\Delta t = 0.01$ and employ it to predict the solution $f(t, \boldsymbol{x}, \boldsymbol{v})$ for $t \in [0, 10]$. We mainly consider (19) in one and two physical dimensions and refer to Appendix C.5 for a detailed discussion on the experiment set-up and the numerical method. Table 6 displays the results corresponding to different Knudsen numbers. The transfer learning enhanced DeepONets reduces the relative error by one or two orders of magnitude although increase the prediction time by around 4 times. Figure 9 and Figure 8 in Appendix C.5 show several snapshots of solutions to (19) with $\varepsilon = 1$ and $\varepsilon = 10^{-4}$ respectively.

RTE	Neural network	t_2	relative error
1D $\varepsilon = 1$	DeepONet	4.6	3.06e-1
	TL-DeepONet	22.5	**1.52e-2**
1D $\varepsilon =$ 1e-4	DeepONet	5.1	3.74e-1
	TL-DeepONet	21.9	**5.52e-3**
2D $\varepsilon = 1$	DeepONet	79.8	3.58e-1
	TL-DeepONet	431.3	**2.19e-2**
2D $\varepsilon =$ 1e-4	DeepONet	83.1	2.37e0
	TL-DeepONet	379.3	**8.93e-3**

Table 6: Results on the radiative transfer equation: the relative prediction errors over the time-horizon $[0, 10]$.

5 Conclusion

In this paper, we proposed a new physics-informed DeepONet based on transfer learning for learning evolutionary PDEs. This is achieved in two steps: first learn the propagators and then predict the solutions by successive actions of propagators on the initial condition. The experimental results demonstrated that the proposed method improves substantially upon the vanilla DeepONet in terms of long-time accuracy and stability while maintains low computational cost. The proposed method also reduced the training sample size needed to achieve the same order of prediction accuracy of the vanilla DeepONets.

Acknowledgement

L. Wang is partially supported by NSF grant DMS-1846854. Y. Lu thanks NSF for the support via the award DMS-2107934.

References

Bhattacharya, K.; Hosseini, B.; Kovachki, N. B.; and Stuart, A. M. 2020. Model reduction and neural networks for parametric PDEs. *arXiv preprint arXiv:2005.03180*.

Bozinovski, S.; and Fulgosi, A. 1976. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, volume 3, 121–126.

Chakraborty, A.; Anitescu, C.; Zhuang, X.; and Rabczuk, T. 2022. Domain adaptation based transfer learning approach for solving PDEs on complex geometries. *Engineering with Computers*, 1–20.

Chen, T.; and Chen, H. 1995. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4): 911–917.

Desai, S.; Mattheakis, M.; Joy, H.; Protopapas, P.; and Roberts, S. 2021. One-Shot Transfer Learning of Physics-Informed Neural Networks. *arXiv preprint arXiv:2110.11286*.

Do, C. B.; and Ng, A. Y. 2005. Transfer learning for text classification. *Advances in neural information processing systems*, 18.

Goswami, S.; Anitescu, C.; Chakraborty, S.; and Rabczuk, T. 2019. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *arXiv preprint arXiv:1907.02531*.

Goswami, S.; Kontolati, K.; Shields, M. D.; and Karniadakis, G. E. 2022. Deep transfer learning for partial differential equations under conditional shift with DeepONet. *arXiv preprint arXiv:2204.09810*.

Harris, C. R.; Millman, K. J.; van der Walt, S. J.; Gommers, R.; Virtanen, P.; Cournapeau, D.; Wieser, E.; Taylor, J.; Berg, S.; Smith, N. J.; Kern, R.; Picus, M.; Hoyer, S.; van Kerkwijk, M. H.; Brett, M.; Haldane, A.; del Río, J. F.; Wiebe, M.; Peterson, P.; Gérard-Marchant, P.; Sheppard, K.; Reddy, T.; Weckesser, W.; Abbasi, H.; Gohlke, C.; and Oliphant, T. E. 2020. Array programming with NumPy. *Nature*, 585(7825): 357–362.

Jin, B.; Cruz, L.; and Gonçalves, N. 2020. Deep facial diagnosis: deep transfer learning from face recognition to facial diagnosis. *IEEE Access*, 8: 123649–123661.

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

Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; and Anandkumar, A. 2020. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*.

Liu, L.; and Cai, W. 2022. DeepPropNet–A Recursive Deep Propagator Neural Network for Learning Evolution PDE Operators. *arXiv preprint arXiv:2202.13429*.

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

Lu, L.; Meng, X.; Cai, S.; Mao, Z.; Goswami, S.; Zhang, Z.; and Karniadakis, G. E. 2022. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393: 114778.

Lu, Y.; Wang, L.; and Xu, W. 2022. Solving multiscale steady radiative transfer equation using neural networks with uniform stability. *Research in the Mathematical Sciences*, 9(3): 1–29.

Obiols-Sales, O.; Vishnu, A.; Malaya, N. P.; and Chandramowlishwaran, A. 2021. SURFNet: Super-resolution of Turbulent Flows with Transfer Learning using Small Datasets. In *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 331–344. IEEE.

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

Ruder, S.; Peters, M. E.; Swayamdipta, S.; and Wolf, T. 2019. Transfer learning in natural language processing. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Tutorials*, 15–18.

Sirignano, J.; and Spiliopoulos, K. 2018. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375: 1339–1364.

Song, D. H.; and Tartakovsky, D. M. 2022. TRANSFER LEARNING ON MULTIFIDELITY DATA. *Journal of Machine Learning for Modeling and Computing*, 3(1): 31–47.

Virtanen, P.; Gommers, R.; Oliphant, T. E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; van der Walt, S. J.; Brett, M.; Wilson, J.; Millman, K. J.; Mayorov, N.; Nelson, A. R. J.; Jones, E.; Kern, R.; Larson, E.; Carey, C. J.; Polat, İ.; Feng, Y.; Moore, E. W.; VanderPlas, J.; Laxalde, D.; Perktold, J.; Cimrman, R.; Henriksen, I.; Quintero, E. A.; Harris, C. R.; Archibald, A. M.; Ribeiro, A. H.; Pedregosa, F.; van Mulbregt, P.; and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272.

Wang, S.; and Perdikaris, P. 2021. Long-time integration of parametric evolution equations with physics-informed deeponets. *arXiv preprint arXiv:2106.05384*.

Wang, S.; Wang, H.; and Perdikaris, P. 2021. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances*, 7(40): eabi8605.

Yin, X.; Yu, X.; Sohn, K.; Liu, X.; and Chandraker, M. 2019. Feature transfer learning for face recognition with under-represented data. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 5704–5713.

Yu, B.; et al. 2018. The deep Ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1): 1–12.

Zang, Y.; Bao, G.; Ye, X.; and Zhou, H. 2020. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411: 109409.

Table 7: Table of notations

Notation	Meaning
$\mathcal{P}$ or $\mathcal{P}^{\Delta t}$	Target propagator
$\mathcal{P}_{NN}$	Neural network approximator
$\mathcal{U}$	The Banach space of input functions
$\{\boldsymbol{x}_i^r\}_{i=1}^{N_x^r}$	The interior sensors
$\{\boldsymbol{x}_i^b\}_{i=1}^{N_x^b}$	The boundary sensors
$\{f_s\}_{s=1}^{N_s}$	Randomly sampled functions used for training
θ, ξ	Neural network parameters
$b^{NN}(\cdot, \theta)$	The branch net of DeepONet
$h^{NN}(\cdot, \theta)$	The output functions defined by the last hidden layer in the branch net
$t^{NN}(\cdot, \xi)$	The trunk net of DeepONet
$w = \{w_j\}_{j=1}^q$	Weights in the last layer of the branch net
N_p	Number of sensors
N_c	Number of grid points in transfer learning step
M	Number of training sample pairs
p	Number of output features
q	Number of tunable weights at the last hidden layer of the branch net

A Table of notations

A table of notations is given in Table 7.

B Theoretical analysis

B.1 Validation of assumption (8)

Here we will give a simple justification for the assumption (8). Consider the L^2 norm as an example:

$$\|\mathcal{P}\|_2 = \sup_{f\in L^2(\Omega_x), \|f\|_2=1} \|\mathcal{P}f\|_2 \,,$$

then (8) is fulfilled if the underlying dynamics is stable under this norm:

$$\frac{d}{dt}\|f\|_2^2(t) \le 0\,,$$

where f is the solution to (1). Indeed, multiplying (1) by f and integrating in $\boldsymbol{x}$, we have

$$\frac{d}{dt}\frac{1}{2}\|f\|_2^2 = \int_{\Omega_x} f\mathcal{L}f \mathrm{d}\boldsymbol{x} \le 0\,.$$

Then for the semi-discrete in time version

$$f^{n+1} - f^n = \Delta t \mathcal{L} f^{n+1}$$

multiplying it by f^{n+1} and integrating in $\boldsymbol{x}$, it becomes

$$\begin{aligned}\frac{1}{2}(\|f^{n+1}\|_2^2 - \|f^n\|_2^2 + \|f^{n+1}-f^n\|_2^2) \\ = \int_{\Omega_x} f^{n+1}\mathcal{L}f^{n+1}\mathrm{d}x \le 0\,,\end{aligned}$$

which readily leads to

$$\|f^{n+1}\|_2 = \|\mathcal{P}f^n\|_2 \le \|f^n\|_2\,.$$

B.2 Proof of Theorem 1

Proof. From (8)-(10), one sees that

$$\begin{aligned}\sup_{f\in\mathcal{U}} \|\mathcal{P}f - \mathcal{P}_{NN}f\|_{\mathcal{X}} &= \sup_{f\in\mathcal{U}} \|\mathcal{P}(f - \mathcal{P}^{-1}\mathcal{P}_{NN}f)\|_{\mathcal{X}} \\ &\le \delta\|f\|_{\mathcal{X}}.\end{aligned} \tag{20}$$

Therefore one obtains that

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|\mathcal{P}_{NN}f\|_{\mathcal{X}} \le 1+\delta\,.$$

Let $f\in\mathcal{U}$ with $\|f\|_{\mathcal{X}} = 1$. It follows from above that

$$\begin{aligned}&\|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} \\ &= \|(\mathcal{P}-\mathcal{P}_{NN})(\mathcal{P}^{K-1} + \mathcal{P}^{K-2}\mathcal{P}_{NN} + \cdots + \mathcal{P}_{NN}^{K-1})f\|_{\mathcal{X}} \\ &\le \sum_{l=0}^{K-1} \|(\mathcal{P}-\mathcal{P}_{NN})\mathcal{P}^l\mathcal{P}_{NN}^{K-1-l}f\|_{\mathcal{X}} \\ &\le \delta K(1+\delta)^K\,,\end{aligned} \tag{21}$$

which proves (12) after taking supreme on f.

Next, if (13) holds, and δ is chosen to be $\delta \le \frac{1}{2}(1-\eta)$, then we have from (20) that

$$\sup_{f\in\mathcal{U}, \|f\|_{\mathcal{X}}=1} \|\mathcal{P}_{NN}f\| \le \eta + \delta \le \frac{1+\eta}{2}.$$

Inserting above into (21) leads to

$$\begin{aligned}\|(\mathcal{P})^K f - (\mathcal{P}_{NN})^K f\|_{\mathcal{X}} &\le \sum_{l=0}^{K-1} \|(\mathcal{P}-\mathcal{P}_{NN})\mathcal{P}^l\mathcal{P}_{NN}^{K-1-l}f\|_{\mathcal{X}} \\ &\le \delta\sum_{l=0}^{K-1}\eta^l(\eta+\delta)^{K-1-l} \\ &\le \delta K\Big(\frac{1+\eta}{2}\Big)^{K-1}.\end{aligned}$$

This proves (14). □

C Experiment details

In this section, we provide the details on the numerical experiments of Section 4.

C.1 Data generation and configuration of training

Data generation For all numerical experiments, we use uniform mesh with N_x^d grid points for discretization of the spatial domain $\Omega_x \subseteq \mathbb{R}^d$, and N_v Gaussian quadrature points for discretizing the velocity variable $\boldsymbol{v}\in\mathcal{S}^{d-1}$ in the radiative transfer equation only. We generate N_s initial conditions $\{f_s(\boldsymbol{x})\}_{s=1}^{N_s}$ for training and N_e functions $\{f_e(\boldsymbol{x})\}_{e=1}^{N_e}$ for testing. The training set consists of N_b functions that sampled from a centered Gaussian random field as well as forward passes of those functions through up to n_t times actions of the propagator. This gives $N_s = n_t \times N_b$ training functions. The random functions may be post-processed so that they satisfy the boundary condition of the PDE. Details on the post-processing methods can be found in the subsequent sections. The final training data of size M is constructed as a

subset of a larger training set of size $N_s \times N_p$, which consists of pointwise evaluations of N_s randomly sampled functions at N_p physical locations (sensors). Unless otherwise specified, we set $N_b = 100$, $n_t = 20$ in 1D test problems and $N_b = 100$, $n_t = 100$ in 2D test problems. Here a training set of size $M = 1000$ may only use 50 functions evaluating at 20 grid points in the domain.

Two error measures. To quantify the performance of our neural nets, we measure two relative errors of neural operator approaches. The first is the relative error at a single time-step $t^k := k\Delta t$:

$$\frac{1}{N_e}\sum_{j=1}^{N_e}\sqrt{\frac{\sum_{i=1}^{N_x}\left(\mathcal{P}_{NN}^k(f_j)(\boldsymbol{x}_i^r)-\mathcal{P}^k(f_j)(\boldsymbol{x}_i^r)\right)^2}{\sum_{i=1}^{N_x}\left(\mathcal{P}^k(f_j)(\boldsymbol{x}_i^r)\right)^2}}, \quad (22)$$

and the second is the relative error over a long time horizon (or equivalently multiple time steps)

$$\sqrt{\frac{\sum_{i=1}^{N_x}\sum_{j=1}^{N_e}\sum_{n=1}^{K}\left(\mathcal{P}_{NN}^n(f_j)(\boldsymbol{x}_i^r)-\mathcal{P}^n(f_j)(\boldsymbol{x}_i^r)\right)^2}{\sum_{i=1}^{N_x}\sum_{j=1}^{N_e}\sum_{n=1}^{K}\left(\mathcal{P}^n(f_j)(\boldsymbol{x}_i^r)\right)^2}}. \quad (23)$$

Neural networks and training parameters In 1D examples, we use the modified fully connected architecture with depth of 5 layers and width of 100 neurons for both branch and trunk nets, and the design of the optional layers P and D in Figure 2 will be detailed in each of the following examples. The batch size is chosen to be 100 with ADAM optimizer, where the initial learning rate $lr = 0.001$ and a 0.95 decay rate in every 5000 steps. Same architecture is used in 2D examples except that a depth of 6 layers is used. In the transfer learning step, to solve the optimization problem (7), we use the lstsq function (with rcond=1e-6) from Numpy(Harris et al. 2020) for linear operators and the leastsq function in Scipy(Virtanen et al. 2020) (using default setting with $ftol$ =1e-5, $xtol$ =1e-5) for nonlinear operators. All of the neural networks are trained on a single K40m GPU, and the prediction step is computed on a AMD Ryzen 7 3700x Processor.

C.2 Further details on reaction diffusion equation

To generate initial conditions that satisfy zero boundary condition, i.e., $f_0(0) = f_0(1) = 0$ we first sample $a(x) \sim \mathcal{GP}(0, K_l(x_1, x_2))$ with

$$K_l(x_1, x_2) = e^{-\frac{(x_1-x_2)^2}{2l^2}}, \quad (24)$$

and then let $f_0(x) = a(x)x(1-x)$. Likewise, to enforce the same boundary condition for the output of the neural net, i.e., $\mathcal{P}_{NN}(f)(1) = \mathcal{P}_{NN}(f)(1) = 0$, we employ an additional layer D in Figure 2 that multiplies the output of trunk nets by $x(1-x)$.

Crank-Nicolson scheme for reaction diffusion equation To demonstrate the improvement on the efficiency of using the higher order in time scheme at the transfer learning step, we apply Crank-Nicolson scheme for the nonlinear reaction diffusion equation. As displayed in Table 8, the second order scheme with $\Delta t = 0.4$ reduces the prediction time compared with first order scheme with $\Delta t = 0.05$ by a factor of $1/6$.

Neural network	Δt	t_2	Relative error
TL-DeepONet	$\Delta t = 0.05$	7.1	**1.78e-3**
TL-DeepONet 2nd	$\Delta t = 0.1$	4.53	**4.91e-4**
	$\Delta t = 0.2$	2.36	**2.63e-3**
	$\Delta t = 0.4$	1.24	**9.53e-3**

Table 8: Comparison of first order and second order in time method for reaction diffusion equation with various Δt. Here t_2 is the averaged time of predicting the solution trajectories among the time interval $[0, 50]$ based on 30 test initial conditions.

Implementation details on Table 1 and Table 8 Consider 1D nonlinear reaction diffusion equation (15) with $d = k = 0.001$. We use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 0.2$ in (24) and we use $N_x = 64$ uniform spatial grids for spatial discretization. The loss functions of CONT DeepONet and CONT TL-DeepONet (c.f. (5)) are calculated using 20 uniform temporal steps on $[0, 1]$. For DeepONet and TL-DeepONet, we set the maximum iteration number $N_{iter} = 100000$ and adopt the stopping criterion that the empirical loss is below 1e-6. For CONT DeepONet and CONT TL-DeepONet, we set maximum iteration number $N_{iter} = 200000$ and use stopping criterion that the empirical loss is below 1e-6. We use $p = 100$ features for all four operator networks. In the transfer learning steps of CONT TL-DeepONet, we subsample $N_c = 400 < 64 \times 20$ grid points and update $q = 25$ weights defined in (6). In the transfer learning steps of TL-DeepONet, we set $N_c = 32 < 64$ and $q = 15$ instead. Additionally, we fix $\Delta t = 0.05$ in Table 1 and $M = 3000$ in Table 8.

C.3 Further details on Allen-Cahn and Cahn-Hilliard equation

In all three examples, we consider periodic boundary conditions. To this end, the initial condition is generated from $f_0(\boldsymbol{x}) \sim \mathcal{GP}(0, K_l^p(\boldsymbol{x}_1, \boldsymbol{x}_2))$, where the covariance kernel has the desired periodicity. In particular, the kernel in one dimension reads:

$$K_l^p(x_1, x_2) = e^{-\frac{\sin^2(\pi(x_1-x_2))}{2l^2}} \quad (25)$$

and in two dimension takes the form:

$$K_l^p(\boldsymbol{x}_1, \boldsymbol{x}_2) = e^{-\frac{\sin^2(\pi(\boldsymbol{x}_{1,1}-\boldsymbol{x}_{2,1}))+\sin^2(\pi(\boldsymbol{x}_{1,2}-\boldsymbol{x}_{2,2}))}{2l^2}}. \quad (26)$$

To enforce the periodic boundary condition to the output of the trunk net, we employ an additional layer P (see Figure 2) in the truck net, which upsizes x to $\{\cos 2\pi x, \sin 2\pi x\}$.This way, the input of the trunk net already has the desired periodicity and will be maintained throughout. Analogously, an additional layer P, which plays the role of upsizing (x, y) to $\{\cos 2\pi x, \sin 2\pi x, \cos 2\pi y, \sin 2\pi y\}$, is leveraged in the trunk net in 2D case.

Implementation details on Figure 1 and Table 2, 3 Consider Allen-Cahn equation (16) with $d_2 = 0.1$. In 1D case,

we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 0.5$ in (25) and use $N_x = 64$ uniform spatial grids for spatial discretization. For DeepONet and TL-DeepONet, we let time step size $\Delta t = 0.05$, set the maximum iteration number $N_{iter} = 100000$ and adopt the stopping criterion that the empirical loss is below 1e-6. For CONT DeepONet and CONT TL-DeepONet, we use additional 20 uniform grids on time span $[0, 1]$, set maximum iteration number $N_{iter} = 200000$ and use stopping criterion that the empirical loss is below 1e-6. We use $p = 100$ features for all four operator networks. In the transfer learning steps of CONT TL-DeepONet, we subsample $N_c = 400 < 64 \times 20$ grid points and update $q = 25$ weights defined in (6). In the transfer learning steps of TL-DeepONet, we set $N_c = 32 < 64$ and $q = 15$ instead. In 2D case, we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 1$ in (26). We use $\Delta t = 0.01$ for time step size and $N_x = N_y = 20$ uniform spatial grids for spatial discretization. We set the maximum iteration number $N_{iter} = 200000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 120$. In the transfer learning step, we subsample $N_c = 144 < 20 \times 20$ and update $q = 40$ weights defined in (6). Additionally, we fix $d_1 = 0.0005$ and $M = 3000$ in Figure 1. For FNO in Figure 1, we choose the time step size $\Delta t = 0.05$, prepare 50 Input & Ouput function pairs, set the maximum iteration number $N_{iter} = 100000$ and adopt the stopping criterion that the empirical loss is below 1e-6.

Implementation details on Table 4 and Figure 3 Consider Cahn-Hilliard equation (17) with $d_2 = 0.001$. In 1D case, we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 0.5$ in (25). We use $\Delta t = 0.05$ for time step size and $N_x = 64$ uniform spatial grids for spatial discretization. We set the maximum iteration number $N_{iter} = 100000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 100$. In the transfer learning step, we subsample $N_c = 32 < 64$ and update $q = 15$ weights defined in (6). In 2D case, we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 1$ in (26). We use $\Delta t = 0.01$ for time step size and $N_x = N_y = 20$ uniform spatial grids for spatial discretization. We set the maximum iteration number $N_{iter} = 200000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 100$. In the transfer learning step, we subsample $N_c = 144 < 20 \times 20$ and update $q = 25$ weights defined in (6). Additionally, we fix d_1 =2e-6 and $M = 30000$ in Figure 3.

C.4 Further details on Navier-Stokes equation

The data generation of Navier-Stokes equation and the design of optional layer P in Figure 2 are exactly same as the one to Allen-Cahn and Cahn-Hilliard equation in Section C.3.

Calculation of Navie-Stokes equation By introducing the stream function $\psi(x,t)$, the velocity field and the vorticity can be found from $u(x,t) = (\frac{\partial \psi}{\partial y}, -\frac{\partial \psi}{\partial x})$ and $w(x,t) = -\Delta\psi(x,t)$. Therefore, we rewirte the equation (18) as

$$\begin{cases} w = -(\partial_{xx}\psi + \partial_{yy}\psi), \\ \partial_t w + \partial_y\psi\partial_x w - \partial_x\psi\partial_y w - \nu(\partial_{xx}w + \partial_{yy}w) - \nu f = 0, \\ w(x,0) = w_0(x)\,, \end{cases} \tag{27}$$

and use the following semi-discretization scheme in the computation of the numerical solutions:

$$\begin{cases} w^{n+1} - w^n + \Delta t\partial_y\psi^n\partial_x w^{n+1} - \partial_x\psi^n\partial_y w^{n+1} \\ \qquad - \nu(\partial_{xx}w^{n+1} + \partial_{yy}w^{n+1}) - \nu f = 0, \\ w^{n+1} + (\partial_{xx}\psi^{n+1} + \partial_{yy}\psi^{n+1}) = 0. \end{cases}$$

Supplementary examples of Navier-Stokes equation Here we provide the snapshots of solution to (18) with $\nu = 0.1, 0.01, 0.0001$ in Figures 5, 6, 7 respectively.

Implementation details on Table 5 and Figures 5,6,4, 7 Consider 2D Navier-Stokes equation (18). We use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 1$ in (26). We use $\Delta t = 0.01$ for time step size and $N_x = N_y = 20$ uniform spatial grids for spatial discretization. We set the maximum iteration number $N_{iter} = 200000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 120$. In the transfer learning step, we subsample $N_c = 144 < 20 \times 20$ and update $q = 40$ weights defined in (6).

C.5 Details on Multiscale linear radiative transfer equation

Computation of multiscale radiative transfer equation Following (Lu, Wang, and Xu 2022), instead of discretizing the original radiative transfer equation (19), we consider a new system of equations based on its micro-macro decomposition. More concretely, let $f = \rho + \varepsilon g,\ \rho = \langle f \rangle$. We consider

$$\begin{cases} \partial_t \rho + \langle \boldsymbol{v} \cdot \nabla g \rangle = 0 \\ \varepsilon^2 g_t + \varepsilon \boldsymbol{v} \cdot \nabla g - \varepsilon \langle \boldsymbol{v} \cdot \nabla g \rangle + \boldsymbol{v} \cdot \nabla \rho = \mathcal{L} g, \\ \rho(t,\boldsymbol{x}) + \varepsilon g(t,\boldsymbol{x},\boldsymbol{v}) = \phi(\boldsymbol{x}),\ (\boldsymbol{x},\boldsymbol{v}) \in \Gamma_- \\ f(0,\boldsymbol{x},\boldsymbol{v}) = f_0(\boldsymbol{x},\boldsymbol{v}). \end{cases} \tag{28}$$

It was shown in (Lu, Wang, and Xu 2022) that the PINN loss based on (19) suffers from the instability issue when the small Knudsen number ε is small while the PINN loss based on the system (28) above is uniformly stable with respect to the small Knudsen number in the sense that the L^2-error of the neural network solution is uniformly controlled by the loss. We consider (19) and its equivalent system (28) in one and two dimensions. To enforce the inflow boundary condition

$$f(t,\boldsymbol{x},\boldsymbol{v}) = \phi(\boldsymbol{x}),\ (\boldsymbol{x},\boldsymbol{v}) \in \Gamma_-\,,$$

we first parameterize f as follows:

$$f(t,\boldsymbol{x},\boldsymbol{v}) = \mathcal{N}_1(t,\boldsymbol{x})A(\boldsymbol{x}) + C(\boldsymbol{x}) + \varepsilon\mathcal{N}_2(t,\boldsymbol{x},\boldsymbol{v})B(\boldsymbol{x},\boldsymbol{v})\,, \tag{29}$$

where $A(\boldsymbol{x})$, $B(\boldsymbol{x},\boldsymbol{v})$ and $C(\boldsymbol{x})$ are determined according to the specific boundary conditions in each example, whilst $\mathcal{N}_1(t,\boldsymbol{x})$ and $\mathcal{N}_2(t,\boldsymbol{x},\boldsymbol{v})$ are to-be approximated by the neural nets. In other words, instead of using two neural nets

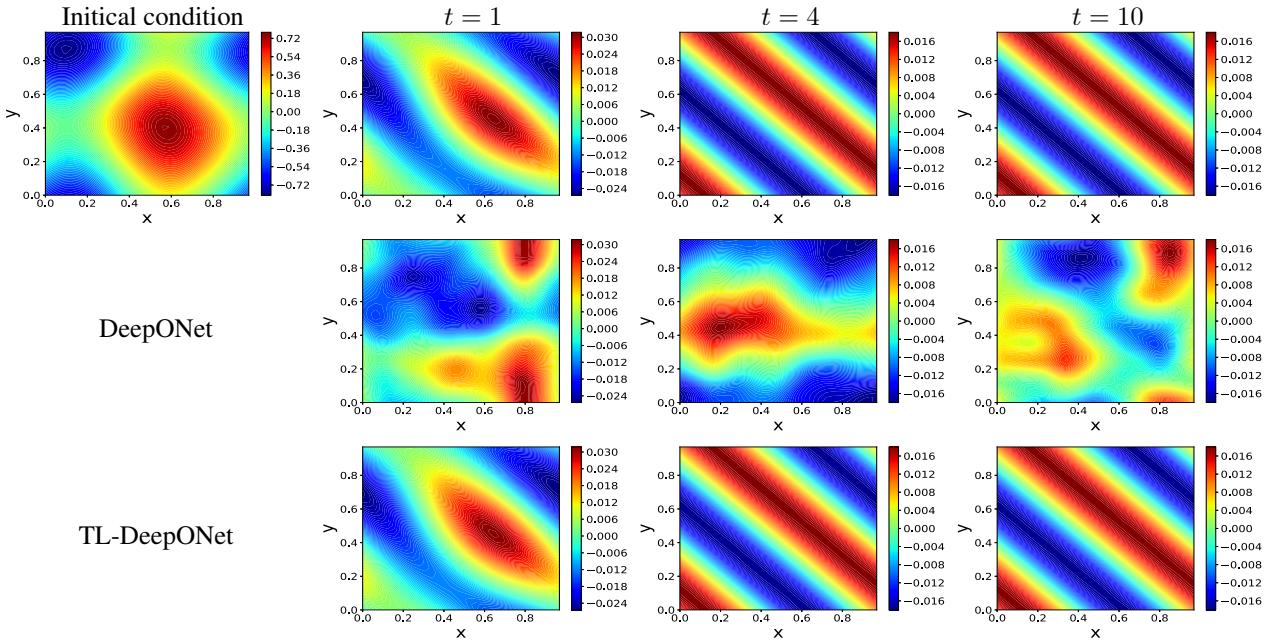

Figure 5: Results on Navier-Stokes equation with $\nu = 0.1$: snapshots of reference solutions (top), and of approximate solutions predicted by DeepONet (middle) and TL-DeepONet (bottom).

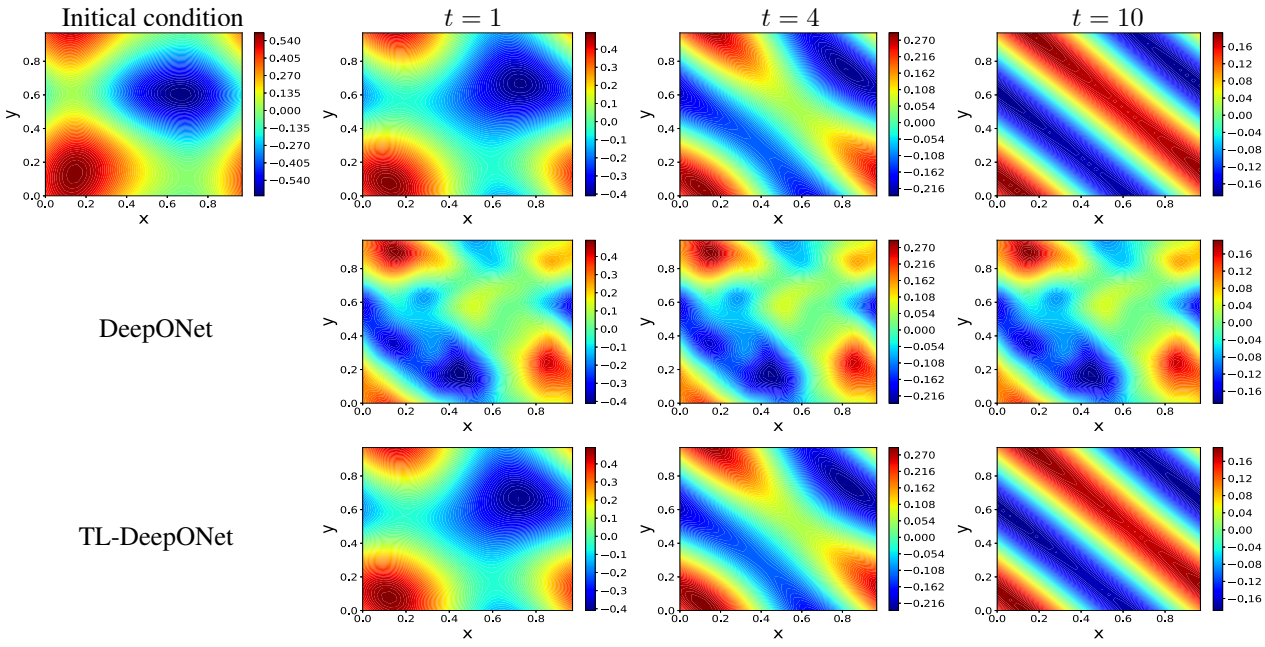

Figure 6: Results on Navier-Stokes equation with $\nu = 0.01$: snapshots of reference solutions (top), and of approximate solutions predicted by DeepONet (middle) and TL-DeepONet (bottom).

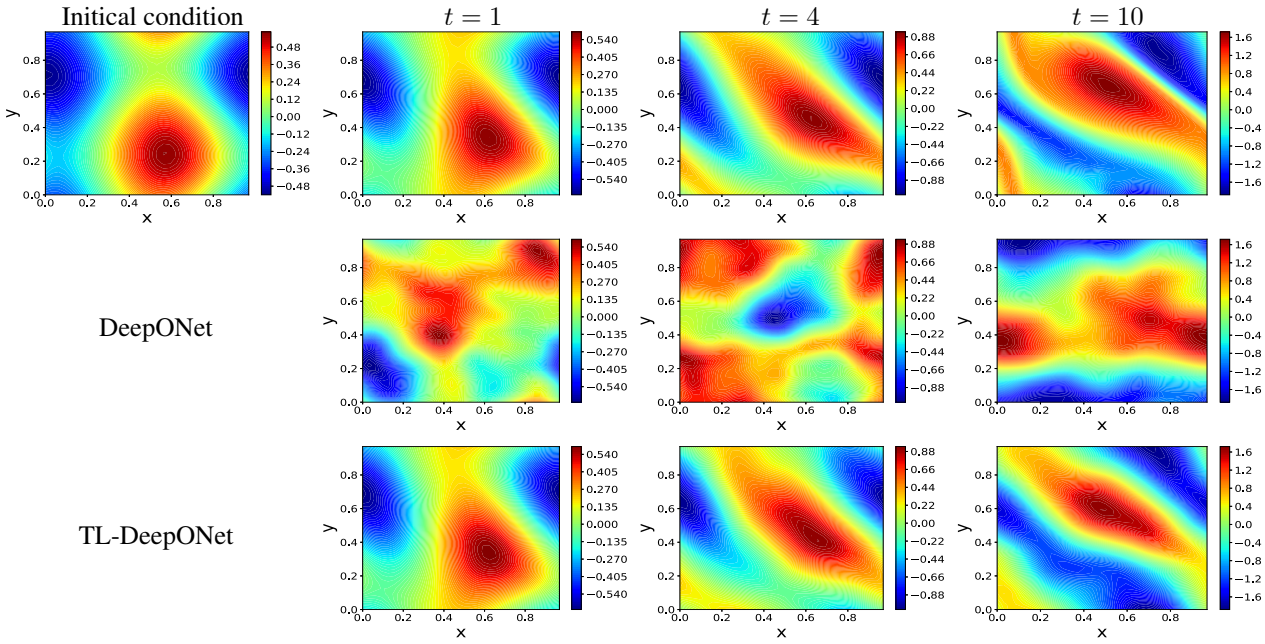

Figure 7: Results on Navier-Stokes equation with $\nu = 0.0001$: snapshots of reference solutions (top), and of approximate solutions predicted by DeepONet (middle) and TL-DeepONet (bottom).

to approximate $\rho(t, \boldsymbol{x}) = \langle f \rangle$ and $g(t, \boldsymbol{x}, \boldsymbol{v}) = f - \rho$, we approximate $\mathcal{N}_1$ and $\mathcal{N}_2$, and recover ρ and g via

$$\rho(t, \boldsymbol{x}) = \mathcal{N}_1(t, \boldsymbol{x})A(\boldsymbol{x}) + C(\boldsymbol{x}) + \varepsilon \langle \mathcal{N}_2(t, \boldsymbol{x}, \boldsymbol{v})B(\boldsymbol{x}, \boldsymbol{v}) \rangle ,$$

and

$$g(t, \boldsymbol{x}, \boldsymbol{v}) = \mathcal{N}_2(t, \boldsymbol{x}, \boldsymbol{v})B(\boldsymbol{x}, \boldsymbol{v}) - \langle \mathcal{N}_2(t, \boldsymbol{x}, \boldsymbol{v})B(\boldsymbol{x}, \boldsymbol{v}) \rangle .$$

1D Example Here the computational domain is $(x, v) \in [0, 1] \times [-1, 1]$ and the inflow boundary condition takes the form:

$$f(0, v > 0) = 1, \ f(1, v < 0) = \frac{1}{2} .$$

Then in (29) we use

$$\begin{aligned} A(x) &= x(1 - x) , \\ B(x, v) &= R(v)x + R(-v)(1 - x) , \\ C(x) &= (1 - \frac{1}{2}x) , \end{aligned}$$

where $R(v)$ is the ReLU function. In the same vein, we generate the initial data by first sampling $a(x, v) \sim \mathcal{GP}(0, K_l(x_1, v_1), (x_2, v_2))$ with

$$K_l((x_1, v_1), (x_2, v_2)) = e^{-\frac{(x_1 - x_2)^2 + (v_1 - v_2)^2}{2l^2}} , \qquad (30)$$

and then selecting those that are strictly positive to construct

$$f_0(x, v) = a(x, v)B(x, v) + C(x) .$$

2D Example Consider the spatial domain as $(x, y) \in [0, 1]^2$, and velocity variable is $(v_x, v_y) = (\cos \alpha, \sin \alpha)$ with $\alpha \in [0, 2\pi]$. The inflow boundary condition reads:

$$f(t, 0, y, v_x > 0) = \frac{1}{2} - (y - \frac{1}{2})^2,$$

$$f(t, 1, y, v_x < 0) = f(t, x, 0, v_y > 0) = f(t, x, 1, v_y < 0) = \frac{1}{2}.$$

Then to guarantee that f in (29) satisfies this condition, we choose

$$\begin{aligned} A(x, y) &= x(1 - x)y(1 - y) , \\ B(x, y, v_x, v_y) &= R(v_x)xy(1 - y) + R(-v_x)(1 - x)y(1 - y) \\ &\quad + R(v_y)yx(1 - x) + R(-v_y)(1 - y)x(1 - x) \\ &\quad + R(v_x)R(v_y)xy + R(v_x)R(-v_y)x(1 - y) \\ &\quad + R(-v_x)R(v_y)(1 - x)y \\ &\quad + R(-v_x)R(-v_y)(1 - x)(1 - y) , \\ C(x, y) &= (\frac{1}{4} - (y - \frac{1}{2})^2)(1 - x) + \frac{1}{4}. \end{aligned}$$

Similarly, the initial conditions are generated by first sampling $a(x, y, \alpha) \sim \mathcal{GP}(0, K_l(x_1, y_1, \alpha_1), (x_2, y_2, \alpha_2))$ with

$$K_l((x_1, y_1, \alpha_1), (x_2, y_2, \alpha_2)) = e^{-\frac{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (\alpha_1 - \alpha_2)^2}{2l^2}} . \qquad (31)$$

and then retaining only the positive samples to construct f_0 via:

$$f_0(x, y, \alpha) = a(x, y, \alpha)B(x, y, \cos(\alpha), \sin(\alpha)) + C(x, y).$$

Supplementary examples of radiative transfer equation Here we provide the snapshots of solution to (19) in 2D with $\varepsilon = 0.0001$ and 1 in Figure 8 and Figure 9, respectively.

Implementation details on Table 6 and Figures 8,9 Consider multiscale radiative transfer equation (19). In 1D case, we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 1$ in (30). We use $\Delta t = 0.01$ for time step size, $N_x = 32$ uniform spatial grids for spatial discretization and $N_v = 16$ Gaussian

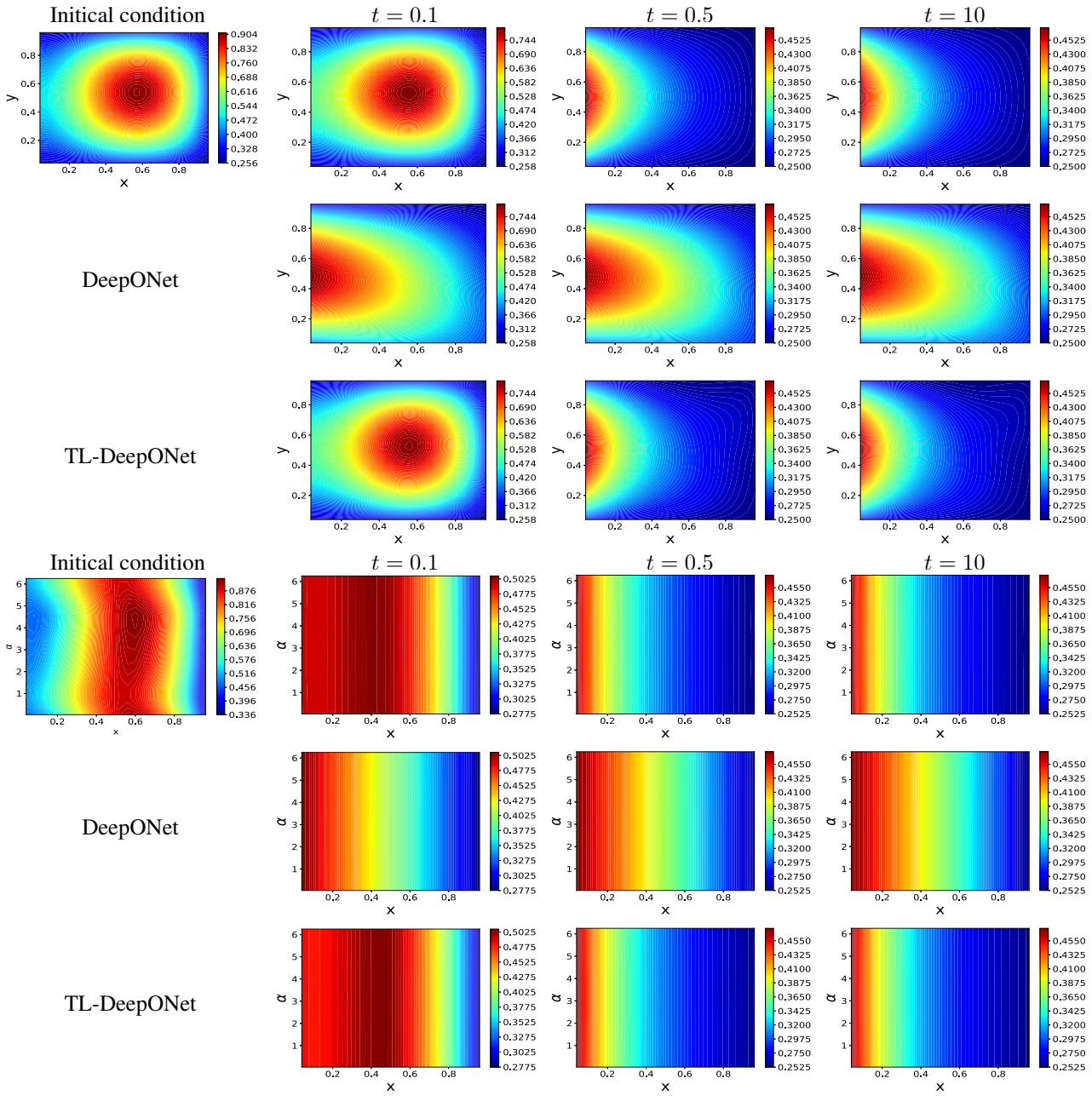

Figure 8: Results on 2D radiative transfer equation with ε =1e-4. The top three rows are snapshots of $\rho(t, x, y)$ of reference solutions, approximate solutions predicted by DeepONet and approximate solutions predicted by TL-DeepONet respectively. The bottom three rows are snapshots of $f(t, x, y = 0.5, \alpha)$ of reference solutions, approximate solutions predicted by DeepONet and approximate solutions predicted by TL-DeepONet respectively.

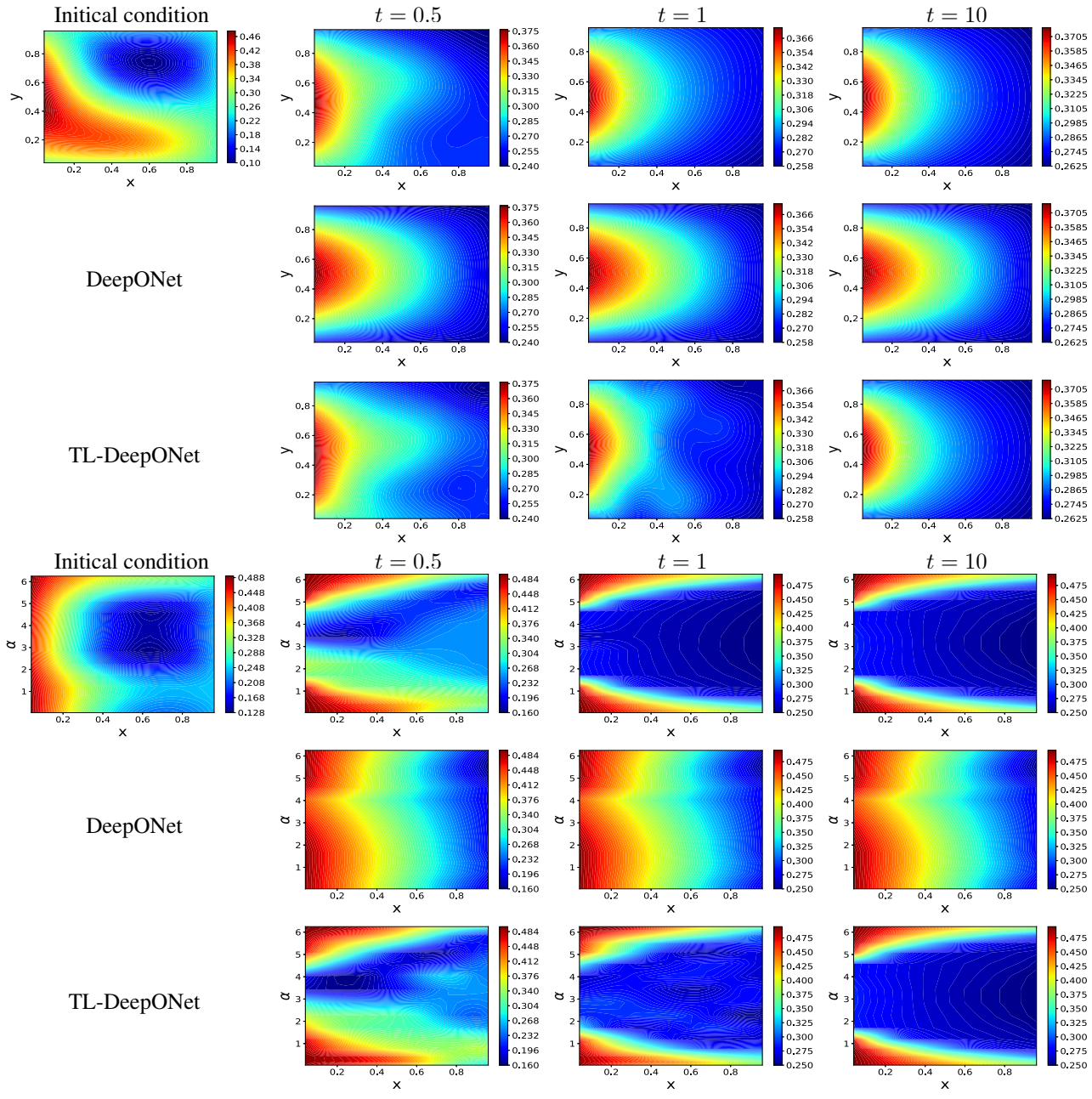

Figure 9: Results on 2D radiative transfer equation with $\varepsilon = 1$. The top three rows are snapshots of $\rho(t, x, y)$ of reference solutions, approximate solutions predicted by DeepONet and approximate solutions predicted by TL-DeepONet respectively. The bottom three rows are snapshots of $f(t, x, y = 0.5, \alpha)$ of reference solutions, approximate solutions predicted by DeepONet and approximate solutions predicted by TL-DeepONet respectively.

quadrature points for velocity discretization. We set the maximum iteration number $N_{iter} = 200000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 100$. In the transfer learning step, we update $q = 40$ weights defined in (6). In 2D case, we use $N_e = 30$ test functions drawn from the Gaussian process defined above with the length scale $l = 1$ in (31). We use $\Delta t = 0.01$ for time step size, $N_x = N_y = 24$ uniform spatial grids for spatial discretization and $N_v = 16$ Gaussian quadrature points for velocity discretization. We set the maximum iteration number $N_{iter} = 300000$, adopt the stopping criterion that the empirical loss is below 1e-6 and use number of feature $p = 150$. In the transfer learning step, we update $q = 120$ weights defined in (6).