nature portfolio

https://doi.org/10.1038/s42256-021-00302-5

Supplementary information

Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators

In the format provided by the authors and unedited

Supplementary Information for

Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators

Lu Lu[1], Pengzhan Jin[2,3], Guofei Pang[2], Zhongqiang Zhang[4], and George Em Karniadakis[2,*]

[1]Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
[2]Division of Applied Mathematics, Brown University, Providence, RI 02912, USA
[3]LSEC, ICMSEC, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, China
[4]Department of Mathematical Sciences, Worcester Polytechnic Institute, Worcester, MA 01609, USA
[*]e-mail: george_karniadakis@brown.edu

S1 Neural networks to approximate nonlinear operators

We list in Table S1 the main symbols and notations that are used throughout this paper.

Table S1: **Notation.**

X	a Banach space with norm $\|\cdot\|_X$
$\mathbb{R}^d$	Euclidean space of dimension d
K	a compact set in a Banach space
$C(K)$	Banach space of all continuous functions defined on K with norm $\|f\|_{C(K)} = \max_{x\in K} \|f(x)\|$
V	a compact set in $C(K)$
$u(x)$	an input function/signal
$s(x)$	an output function/signal
f	a function or functional
G	an operator
(TW)	all the Tauber-Wiener functions
$S(\mathbb{R}^d)$	Schwartz functions in tempered distribution theory, i.e., rapidly decreasing and infinitely differentiable functions
$S'(\mathbb{R}^d)$	Tempered distributions, i.e., linear continuous functionals defined on $S(\mathbb{R}^d)$
σ	an activation function
$\{x_1, x_2, \ldots, x_m\}$	m sensor points
n, p	neural network size hyperparameters in Theorem 1
N	number of basis functions
M	value of some upper bound

Let $C(K)$ denote the Banach space of all continuous functions defined on a compact set $K \subset X$ with sup-norm $\|f\|_{C(K)} = \max_{x\in K} |f(x)|$, where X is a Banach space. We first review the definition and sufficient condition of Tauber-Wiener (TW) functions [4], and the definition of continuous operator.

Definition S1 (**TW**). *If a function $\sigma : \mathbb{R} \to \mathbb{R}$ (continuous or discontinuous) satisfies that all the linear combinations $\sum_{i=1}^{N} c_i\sigma(\lambda_i x + \theta_i)$, $\lambda_i \in \mathbb{R}$, $\theta_i \in \mathbb{R}$, $c_i \in \mathbb{R}$, $i = 1, 2, \ldots, N$, are dense in every $C([a, b])$, then σ is called a Tauber-Wiener (TW) function.*

Theorem S2 (**Sufficient condition for TW**). *Suppose that σ is a continuous function, and $\sigma \in S'(\mathbb{R})$ (tempered distributions), then $\sigma \in (TW)$, if and only if σ is not a polynomial.*

It is easy to verify that all the activation functions we use nowadays, such as sigmoid, tanh and ReLU, are TW functions.

Definition S3 (**Continuity**). *Let G be an operator between topological spaces X and Y. We call G continuous if for every $\epsilon > 0$, there exists a constant $\delta > 0$ such that*

$$\|G(x) - G(y)\|_Y < \epsilon$$

for all $x, y \in X$ satisfying $\|x - y\|_X < \delta$.

We recall the following two main theorems of approximating nonlinear continuous functionals and operators due to Chen & Chen [4].

Theorem S4 (**Universal Approximation Theorem for Functional**). *Suppose that $\sigma \in (TW)$, X is a Banach space, $K \subset X$ is a compact set, V is a compact set in $C(K)$, f is a continuous*

functional defined on V, then for any $\epsilon > 0$, there are a positive integer n, m points $x_1, \ldots, x_m \in K$, and real constants c_i, θ_i, ξ_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$, such that

$$\left| f(u) - \sum_{i=1}^{n} c_i \sigma \left(\sum_{j=1}^{m} \xi_{ij} u(x_j) + \theta_i \right) \right| < \epsilon$$

holds for all $u \in V$.

Theorem S5 (**Universal Approximation Theorem for Operator**). *Suppose that $\sigma \in (TW)$, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$, then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$.

The necessary and sufficient conditions of the compactness are given by the following Arzelà-Ascoli Theorem.

Definition S6 (**Uniform Boundedness**). *Let V be a family of real-valued functions on the set K. We call V uniformly bounded if there exists a constant M such that*

$$|f(x)| \leq M$$

for all $f \in V$ and all $x \in K$.

Definition S7 (**Equicontinuity**). *Let V be a family of real-valued functions on the set K. We call V equicontinuous if for every $\epsilon > 0$, there exists a $\delta > 0$ such that*

$$|f(x) - f(y)| < \epsilon$$

for all $f \in V$ and all $x, y \in K$ satisfying $|x - y| < \delta$.

Theorem S8 (**Arzelà-Ascoli Theorem**). *Let X be a Banach space, and $K \subset X$ be a compact set. A subset V of $C(K)$ is pre-compact (has compact closure) if and only if V is uniformly bounded and equicontinuous.*

S2 Proof of Theorem 2

In this section, we provide a simple proof of Theorem 2.

Proof. Let $\boldsymbol{g} = (g_1, \cdots, g_p)^T$ and $\boldsymbol{f} = (f_1, \cdots, f_p)^T$. We choose $g_k(x) = \sum_{i=1}^{n} c_i^k \sigma(\sum_{j=1}^{m} \xi_{ij}^k x_j + \theta_i^k)$ for $x = (x_1, \cdots, x_m)^T \in \mathbb{R}^m$, and $f_k(y) = \sigma(w_k \cdot y + \zeta_k)$ for $y \in \mathbb{R}^d$, where all the notations involved are defined in Theorem 1. Then this theorem holds immediately following Theorem 1.

Note that the domains of $\boldsymbol{g}$ and $\boldsymbol{f}$ can be restricted on compact sets. In fact, $\boldsymbol{f}$ is defined on K_2, and $\boldsymbol{g}$ can be defined on $[-M, M]^m$, where $M > 0$ is a uniform bound of the functions in V. Therefore, $\boldsymbol{g}$ and $\boldsymbol{f}$ can be approximated by neural networks according to the universal approximation theorem of functions. □

S3 Data generation

Here, we list the details of the data generation, including function spaces V and reference solutions. We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may appear in multiple data points with different values of y. For example, a dataset of size 10000 may only be generated from 100 u trajectories, and each evaluates $G(u)(y)$ for 100 y locations.

Gaussian random field (GRF) or Gaussian process (GP). In most examples, we use the mean-zero GRFs for the input function $u(x)$:

$$u(x) \sim \mathcal{GP}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp(-\|x_1 - x_2\|^2/2l^2)$ is the Gaussian kernel with a length-scale parameter $l > 0$. The length-scale l determines the smoothness of the sampled function, and larger l leads to smoother u.

Orthogonal polynomials. Let $M > 0$ and P_i be i-th orthogonal polynomial basis. We define the N-dimensional orthogonal polynomial space as

$$V_{\text{Org}} = \left\{ \sum_{i=0}^{N-1} a_i P_i(x) : |a_i| \leq M \right\}.$$

We generate the samples of input function $u(\cdot)$ from the space V_{Org} by randomly sampling the expansion coefficient a_i from $[-M, M]$.

In the paper, we consider four different bases $P_i(\cdot)$:

1. the Chebyshev polynomial of the first kind $T_i(x)$;
2. the Legendre polynomial $L_i(x)$;
3. the poly-fractonomial polynomial $(1+x)^{0.5} J_i^{-0.5,0.5}(x)$, where $J_i^{\alpha,\beta}(x)$ is the Jacobi polynomial of degree i;
4. the Zernike polynomial $Z_i(r, \theta)$, where $Z_i(r, \theta)$ is the single indexing Zernike polynomial whose index i is linked to the indices of double-indexing one $Z_n^m(r, \theta)$ by $i = \frac{n(n+2)+m}{2}$.

Reference solutions. The reference solutions of all ODE systems are obtained by Runge-Kutta (4, 5), and the reference solutions of all deterministic PDEs are obtained by a second-order finite difference method.

The exact solution of the stochastic ODE (Problem 5) is

$$y = y_0 e^{K(t)},$$

where $K(t) = \int_0^t k(s)\, ds$ is again Gaussian process with mean zero.

The exact solution of the stochastic PDE (Problem 6) is

$$u(x) = C(\omega) \int_0^x e^{-b(y;\omega)}\, dy - \int_0^x \int_0^y e^{-b(y;\omega)} f(z)\, dz\, dy,$$

where

$$C(\omega) = \left(\int_0^1 e^{-b(y;\omega)}\, dy \right)^{-1} \int_0^1 \int_0^y e^{-b(y;\omega)} f(z)\, dz\, dy.$$

Numerical solutions are obtained by approximating $k(t;\omega)$ or $b(x;\omega)$ with their truncated Karhunen-Loeve expansions.

S4 Gaussian random field with the Gaussian kernel

Suppose that $X(t) \sim \mathcal{GP}(0, \exp(-\frac{|x-y|^2}{l^2}))$. Then, by the Wiener-Khinchin Theorem, we have

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \cos(\omega t) \exp(-\frac{l^2\omega^2}{8}) dW(\omega) - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \sin(\omega t) \exp(-\frac{l^2\omega^2}{8}) dB(\omega),$$

where W and B are independent standard Brownian motions [13]. Apply the change of variable $\lambda = l\omega$ and write $X(t)$ as

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \cos(\frac{\lambda}{l} t) \exp(-\frac{\lambda^2}{8}) dW(\lambda) - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \sin(\frac{\lambda}{l} t) \exp(-\frac{\lambda^2}{8}) dB(\lambda).$$

Applying a linear interpolation Π_1 on the interval $[t_i, t_{i+1}]$, then

$$\begin{aligned}
\mathbb{E}[(X(t) - \Pi_1 X(t))^2] &= 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\cos(\frac{\lambda}{l} t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&\quad + 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\sin(\frac{\lambda}{l} t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&\leq (\pi)^{\frac{1}{2}} (t_{i+1} - t_i)^4 \int_{\mathbb{R}^+} (\frac{\lambda}{l})^4 \exp(-\frac{\lambda^2}{4}) d\lambda \\
&= 24\pi \frac{(t_{i+1} - t_i)^4}{l^4},
\end{aligned}$$

where we recalled the error estimate of the linear interpolation on $[a, b]$ (by Taylor's expansion)

$$|(I - \Pi_1) g(t)| = \frac{1}{2}(b-a)^2 |g''(\xi)|,$$

where ξ lies in between a and b. Then by the fact that $X(t) - \Pi_1 X(t)$ is Gaussian, we have

$$|X(t) - \Pi_1 X(t)| \leq C \frac{(t_{i+1} - t_i)^2}{l^2}, \quad \epsilon > 0,\, t \in [t_i, t_{i+1}],$$

where C is an absolute value of a Gaussian random variable with mean zero and variance $\sqrt{24\pi}$. In conclusion, taking a piecewise linear interpolation of $X(t)$ with m points will lead to convergence with order $\mathcal{O}(\frac{1}{m^2 l^2})$.

S5 Number of sensors for identifying nonlinear dynamical systems

This part is the proof of Theorem 3. All the concepts and notations involved here are defined in the paragraph of Theorem 3.

Lemma. *$W \coloneqq \bigcup_{i=1}^{\infty} W_i$ is compact.*

Proof. At first, we prove that W is pre-compact. For any $\varepsilon > 0$, by (3), there exists an m_0 such that

$$\|u - \mathcal{L}_m(u)\|_C < \frac{\varepsilon}{4}, \quad \forall u \in V, \forall m > m_0.$$

Since W_{m_0} is a compact set subject to equicontinuity, there exists a $\delta > 0$ such that

$$|x - y| < \delta \Rightarrow |u(x) - u(y)| < \frac{\varepsilon}{2}, \quad \forall u \in W_{m_0}, \forall x, y \in [a, b].$$

Now for all $u \in W$ and all $x, y \in [a, b], |x - y| < \delta$, if $u \in W_{m_0}$, naturally we have $|u(x) - u(y)| < \frac{\varepsilon}{2} < \varepsilon$, otherwise, $u \in \bigcup_{i=m_0+1}^{\infty} U_i$. Suppose that $u = \mathcal{L}_m(v), m > m_0, v \in V$, then there holds

$$\begin{aligned}
|u(x) - u(y)| =& |u(x) - v(x) + v(x) - v(y) + v(y) - u(y)| \\
\leq& |\mathcal{L}_m(v)(x) - v(x)| + |v(x) - v(y)| + |\mathcal{L}_m(v)(y) - v(y)| \\
\leq& 2\|\mathcal{L}_m(v) - v\|_C + |v(x) - v(y)| \\
<& 2 \cdot \frac{\varepsilon}{4} + \frac{\varepsilon}{2} = \varepsilon,
\end{aligned}$$

which shows the quicontinuity of W. In addition, it is obvious that W is uniformly bounded, so that we know W is pre-compact by applying the Arzelà-Ascoli Theorem.

Next we show that W is closed. Let $\{w_i\}_{i=1}^{\infty} \subset W$ be a sequence which converges to a $w_0 \in C[a, b]$. If there exists an m such that $\{w_i\} \subset W_m$, then $w_0 \in W_m \subset W$. Otherwise, there is a subsequence $\{\mathcal{L}_{i_n}(v_{i_n})\}$ of $\{w_i\}$ such that $v_{i_n} \in V$ and $i_n \to \infty$ as $n \to \infty$. Then we have

$$\begin{aligned}
\|v_{i_n} - w_0\|_C =& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n}) + \mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\
\leq& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n})\|_C + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\
\leq& \kappa(i_n, V) + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C,
\end{aligned}$$

which implies that $w_0 = \lim_{n\to\infty} v_{i_n} \in V \subset W$. □

Next we show the proof of Theorem 3.

Proof. For $u \in V$ and $u_m \in U_m$, according to the bound (3) and the Lipschitz conditions of the right-hand side of the ODE, we can derive that the operator is Lipschitz continuous.

$$\begin{aligned}
\|G(u)(d) - G(u_m)(d)\|_2 &\leq c \int_a^d \|G(u)(t) - G(u_m)(t)\|_2 dt + c \int_a^d |u(t) - u_m(t)| dt \\
&\leq c \int_a^d \|G(u)(t) - G(u_m)(t)\|_2 dt + c(b - a)\kappa(m, V).
\end{aligned}$$

Here $\kappa(m, V)$ is defined in (3). By Grönwall's inequality, we have

$$\|G(u)(d) - G(u_m)(d))\|_2 \leq c(b - a)\kappa(m, V)e^{c(b-a)}.$$

Define $S_m = \{(u(x_0), u(x_1), \cdots, u(x_m)) \in \mathbb{R}^{m+1} | u \in V\}$. Then S_m is a compact set in $\mathbb{R}^{m+1}$ as it is the image of a continuous map on the compact set V. As there is a bijective map between S_m and U_m, we may define a vector-valued function $\boldsymbol{\varphi}$ on S_m by

$$\boldsymbol{\varphi}(u(x_0), u(x_1), \cdots, u(x_m)) = G(u_m)(d).$$

Because G is a continuous operator over V, $\boldsymbol{\varphi}$ is a continuous function over S_m. For any $\varepsilon > 0$, make m large enough so that $c(b-a)\kappa(m, V)e^{c(b-a)} < \varepsilon$. By the universal approximation theorem of neural network for high-dimensional functions, there exist $\mathcal{W}_1 \in \mathbb{R}^{n\times(m+1)}, b_1 \in \mathbb{R}^{m+1}, \mathcal{W}_2 \in \mathbb{R}^{K\times n}, b_2 \in \mathbb{R}^K$, such that

$$\|\boldsymbol{\varphi}(u(x_0), \cdots, u(x_m)) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 < \varepsilon - c(b-a)\kappa(m, V)e^{c(b-a)}$$

holds for all $(u(x_0), \cdots, u(x_m)) \in S_m$. Hence, we have

$$\begin{aligned}
&\|(Gu)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\
\leq &\|(Gu)(d) - (Gu_m)(d)\|_2 + \|(Gu_m)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\
< &c(b-a)\kappa(m, V)e^{c(b-a)} + \varepsilon - c(b-a)\kappa(m, V)e^{c(b-a)} = \varepsilon.
\end{aligned}$$

In summary, by choosing the value of m so that it makes $c(b-a)\kappa(m, V)e^{c(b-a)}$ less than ε is sufficient to achieve accuracy ε. □

S6 Wasserstein metric for Gaussian processes

Different metrics have been proposed to measure the distance between two Gaussian processes, and we will use the 2-Wasserstein metric in this study. Suppose we have two GPs $f_1 \sim \mathcal{GP}(m_1, k_1)$ and $f_2 \sim \mathcal{GP}(m_2, k_2)$ defined on X, where m_1 and m_2 are the mean functions, and k_1 and k_2 are the covariance functions. Each covariance function k_i has an associated covariance operator $K_i : L^2(X) \to L^2(X)$ defined by

$$[K_i\phi](x) = \int_X k_i(x, s)\phi(s)ds, \qquad \forall \phi \in L^2(X).$$

Then the 2-Wasserstein metric between the two Gaussian processes f_1 and f_2 is given by

$$W_2(f_1, f_2) \coloneqq \left\{ \|m_1 - m_2\|_2^2 + \mathrm{Tr}\left[K_1 + K_2 - 2\left(K_1^{\frac{1}{2}} K_2 K_1^{\frac{1}{2}} \right)^{\frac{1}{2}} \right] \right\}^{\frac{1}{2}},$$

where Tr is the trace of the operator. More details can be found in [5].

In practice, we may not be able to compute the 2-Wasserstein metric analytically. To estimate it numerically, we can estimate the trace of the operator from the trace of the corresponding covariance matrix. For example, let $\hat{K}_1 \in \mathbb{R}^{N\times N}$ be a covariance matrix of k_1, i.e., the i^{th}-row and j^{th}-column entry of $\hat{K}_1$ is $\hat{K}_1[i, j] = k_1(x_i, x_j)$, where $\{x_1, x_2, \ldots, x_N\}$ are N equispaced points in the domain of f_1. Then we have $\mathrm{Tr}K_1 \approx \mathrm{Tr}\hat{K}_1/N$, and we can estimate other traces in the same way.

S7 Parameters

In all problems we use ReLU activation function, except that for fractional differential operators we use the hyperbolic tangent. We use the Adam optimizer [8] with learning rate 0.001, except that for the Legendre transform it is 0.0001. The other parameters and network sizes are listed in Tables S2 and S3, unless otherwise stated. In this study, we tune these hyperparameters manually, while they may also be automatically chosen via the techniques in automated machine learning (AutoML) [7]. Here, we list a few hints based on our experience that are useful for hyperparameter tuning:

- The number of sensors m is determined by the smoothness of the input function, and a large m is required for oscillatory input functions.
- If the training error does not decrease, then it may be helpful to increase the network size (depth and width of the trunk and branch nets), decrease the learning rate, or increase the training iterations.
- If the difference between training error and test error is large, i.e., a large generalization error, then it may be helpful to increase the training dataset size, or decrease the network size.

Table S2: **Default parameters for each problem, unless otherwise stated.** We note that one data point is a triplet $(u, y, G(u)(y))$, and thus a specific input u may generate multiple data points with different values of y. In our dataset, for each u we randomly select P different values of y, and thus the total number of data points is equal to $P \times \#u$.

Case	Input function space	# Sensors m	$\#u$ Train	P	$\#u$ Test	# Iterations
Antiderivative	GRF ($l = 0.2$)	100	10^4	1	10^5	5×10^4
Legendre transform	GRF ($l = 0.2$)	200	500	20	1000	2×10^6
Fractional (1D)	–	15	10^4	100	10^4	3×10^4
Fractional (2D)	Zernike	225	5000	2250	5000	5×10^3
Nonlinear ODE	GRF ($l = 0.2$)	100	10^4	1	10^5	10^5
Gravity pendulum	GRF ($l = 0.2$)	100	10^4	1	10^5	10^5
Diffusion-reaction	GRF ($l = 0.2$)	100	–	–	10^6	5×10^5
Advection (4 cases)	GRF	100	500	100	1000	5×10^5
Advection-diffusion	GRF ($l = 0.5$)	100	500	100	1000	5×10^5
Stochastic ODE/PDE (pathwise)	–	20	10^4	100	10^4	5×10^4
Stochastic PDE (mean/SD)	–	30	1000	10	1000	2×10^4

Table S3: **DeepONet size for each problem, unless otherwise stated.**

Case	Trunk depth	Trunk width	Branch depth	Branch width
Antiderivative (stacked and unstacked)	3	40	2	40
Legendre transform	3	100	2	100
Fractional (1D)	4	40	3	40
Fractional (2D)	4	60	3	60
Nonlinear ODE (stacked and unstacked)	3	40	2	40
Gravity pendulum	3	40	2	40
Diffusion-reaction	3	100	2	100
Advection (4 cases)	3	100	2	100
Advection-diffusion	3	100	2	100
Stochastic ODE/PDE (pathwise)	3	100	2	100
Stochastic PDE (mean/SD)	3	100	2	100

S8 Computational cost

Table S4: **Computational cost (hours) of training of DeepONet and NN baselines.** DeepONet size is listed in Table S3, and the network sizes of other models are chosen as the sizes with which the best accuracy is obtained. All networks are trained using 4 cores on an Intel Xeon Gold 6242 processor.

Case	DeepONet	FNN	ResNet	CNN	Seq2Seq
Antiderivative	0.10	1.39	0.07	–	0.77
Legendre transform	10.63	–	–	–	–
Fractional (1D)	14.24	–	–	–	–
Fractional (2D)	76.41	–	–	2.28	–
Nonlinear ODE	0.27	–	–	–	–
Gravity pendulum	0.19	0.31	–	–	1.71
Diffusion-reaction	12.25	–	3.69	–	–
Advection (4 cases)	11.96	–	–	–	–
Advection-diffusion	14.51	–	–	–	–
Stochastic ODE/PDE (pathwise)	27.23	–	–	–	–
Stochastic PDE (mean/SD)	0.13	–	–	–	–

Table S5: **Computational cost (seconds) of running inference of DeepONet, NN baselines and numerical solvers.** DeepONet size is listed in Table S3, and the network sizes of other models are chosen as the sizes with which the best accuracy is obtained. The numerical solvers used for each case are descried in Section S3. The network inferences and numerical solvers are performed using 1 core on an Intel Xeon Gold 6242 processor.

Case	DeepONet	FNN	ResNet	CNN	Seq2Seq	Numerical solver
Antiderivative	1.3×10^{-4}	9.9×10^{-4}	3.5×10^{-4}	–	1.4×10^{-2}	8.0×10^{-4}
Legendre transform	1.4×10^{-4}	–	–	–	–	1.2×10^{-5}
Fractional (1D)	1.5×10^{-4}	–	–	–	–	117.6
Fractional (2D)	2.2×10^{-4}	–	–	3.5×10^{-4}	–	43.1
Nonlinear ODE	1.3×10^{-4}	–	–	–	–	9.5×10^{-4}
Gravity pendulum	1.3×10^{-4}	3.8×10^{-4}	–	–	1.3×10^{-2}	1.4×10^{-3}
Diffusion-reaction	3.7×10^{-4}	–	3.6×10^{-3}	–	–	1.9×10^{-2}
Advection (4 cases)	3.7×10^{-4}	–	–	–	–	9.0×10^{-3}
Advection-diffusion	3.7×10^{-4}	–	–	–	–	9.0×10^{-3}
Stochastic ODE/PDE (pathwise)	1.4×10^{-4}	–	–	–	–	Analytical solution
Stochastic PDE (mean/SD)	1.4×10^{-4}	–	–	–	–	Analytical solution

S9 Antiderivative

Table S6: **Errors of different network architectures trained to learn the antiderivative operator (Fig. 2A).** The p-values of two-sided T-tests indicate whether the test error of the unstacked DeepONet with bias is significantly smaller than that of other networks.

Network	Training MSE	Test MSE	p-value
FNN (best)	$8.6 \times 10^{-6} \pm 1.7 \times 10^{-6}$	$7.0 \times 10^{-5} \pm 4.1 \times 10^{-6}$	0.000
ResNet (best)	$8.2 \times 10^{-5} \pm 2.3 \times 10^{-5}$	$1.3 \times 10^{-4} \pm 3.8 \times 10^{-5}$	0.000
Seq2Seq (best)	$2.2 \times 10^{-6} \pm 1.7 \times 10^{-6}$	$8.9 \times 10^{-6} \pm 5.7 \times 10^{-6}$	0.002
Seq2Seq 10× (best)	$1.6 \times 10^{-6} \pm 1.4 \times 10^{-6}$	$2.6 \times 10^{-6} \pm 1.9 \times 10^{-6}$	0.391
Stacked (no bias)	$3.1 \times 10^{-6} \pm 1.7 \times 10^{-6}$	$1.1 \times 10^{-5} \pm 5.6 \times 10^{-6}$	0.000
Stacked (bias)	$2.4 \times 10^{-6} \pm 7.3 \times 10^{-7}$	$9.8 \times 10^{-6} \pm 2.2 \times 10^{-6}$	0.000
Unstacked (no bias)	$1.7 \times 10^{-6} \pm 4.5 \times 10^{-7}$	$2.3 \times 10^{-6} \pm 5.7 \times 10^{-7}$	0.386
Unstacked (bias)	$1.6 \times 10^{-6} \pm 6.0 \times 10^{-7}$	$2.0 \times 10^{-6} \pm 6.2 \times 10^{-7}$	–

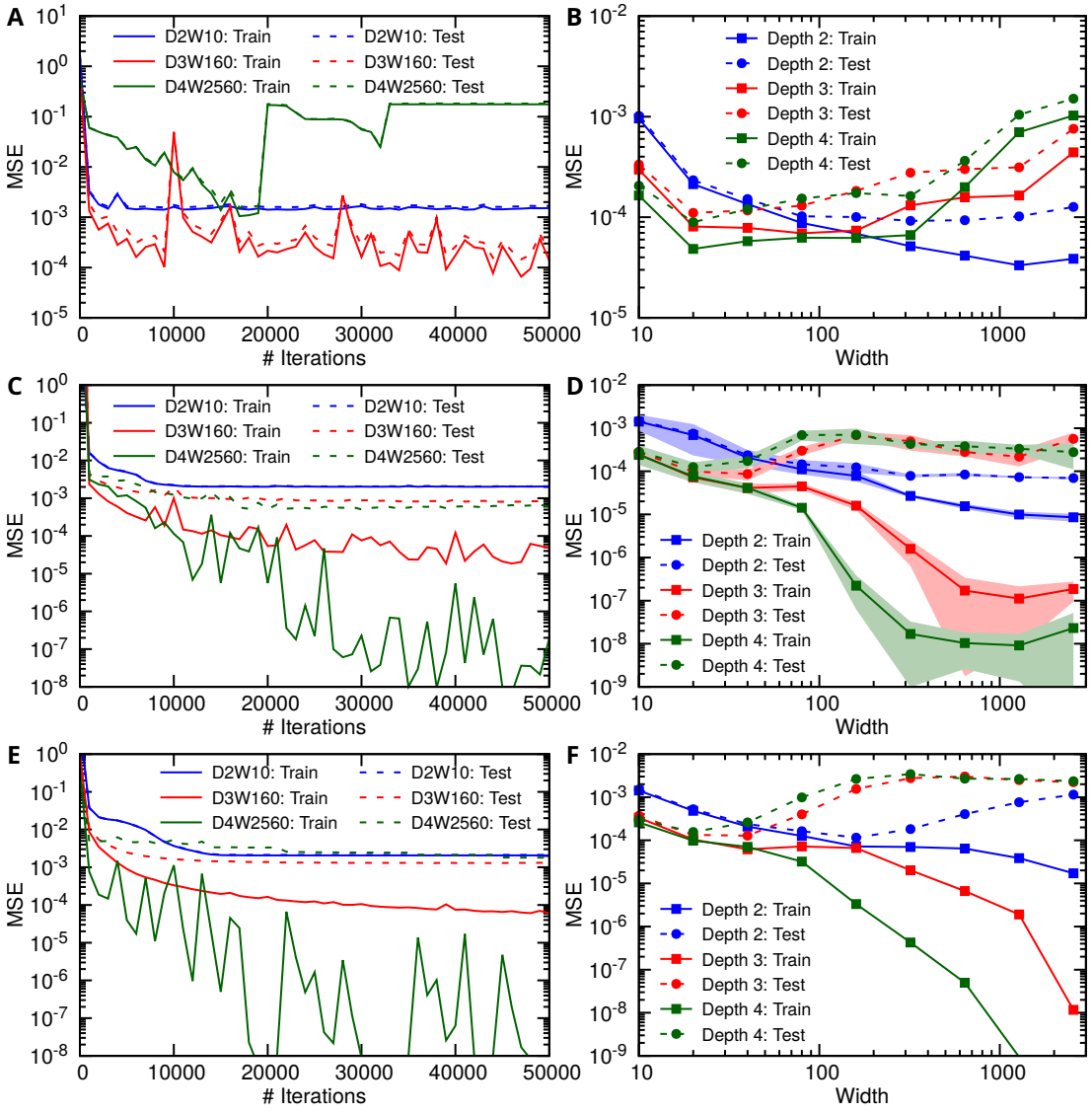

Fig. S1: **Errors of FNNs trained to learn the antiderivative operator (linear case).** (**A** and **B**) Learning rate 0.01. (**C** and **D**) Learning rate 0.001. (**E** and **F**) Learning rate 0.0001. (A, C, E) The solid and dash lines are the training error and test error during the training process, respectively. The blue, red and green lines represent FNNs of size (depth 2, width 10), (depth 3, width 160), and (depth 4, width 2560), respectively. (B, D, F) The blue, red and green lines represent FNNs of depth 2, 3 and 4, respectively. The shaded regions in (D) are the one-standard-derivation (SD) from 10 runs with different training/test data and network initialization. For clarity, only the SD of learning rate 0.001 is shown (Fig. D). The number of sensors for u is $m = 100$.

S10 Legendre transform

We consider the Legendre transform of a function $f(x)$:

$$\textbf{Problem 7} \qquad \mathcal{J}_n\{f(x)\} = \int_{-1}^{1} P_n(x) f(x) dx, \tag{S1}$$

where $P_n(x)$ is a Legendre polynomial of degree n. We use DeepONet to learn the Legendre transform $f(x) \mapsto \mathcal{J}_n\{f(x)\}$. Because the Legendre transform decays fast to zero w.r.t. n for a smooth $f(x)$, it is sufficient to consider the first 20 coefficients, i.e., $n \in \{0, 1, 2, \ldots, 19\}$. To generate the dataset, we sampled 500 random $f(x)$ from a GRF with the Gaussian kernel of the length scale $l = 0.2$, and computed the Legendre transform at the first 20 coefficients for each $f(x)$. Thus the dataset size is equal to $10000 (= 500 \times 20)$. The test MSE is $3.2 \times 10^{-7} \pm 1.9 \times 10^{-7}$, and two prediction examples are shown in Fig. S2.

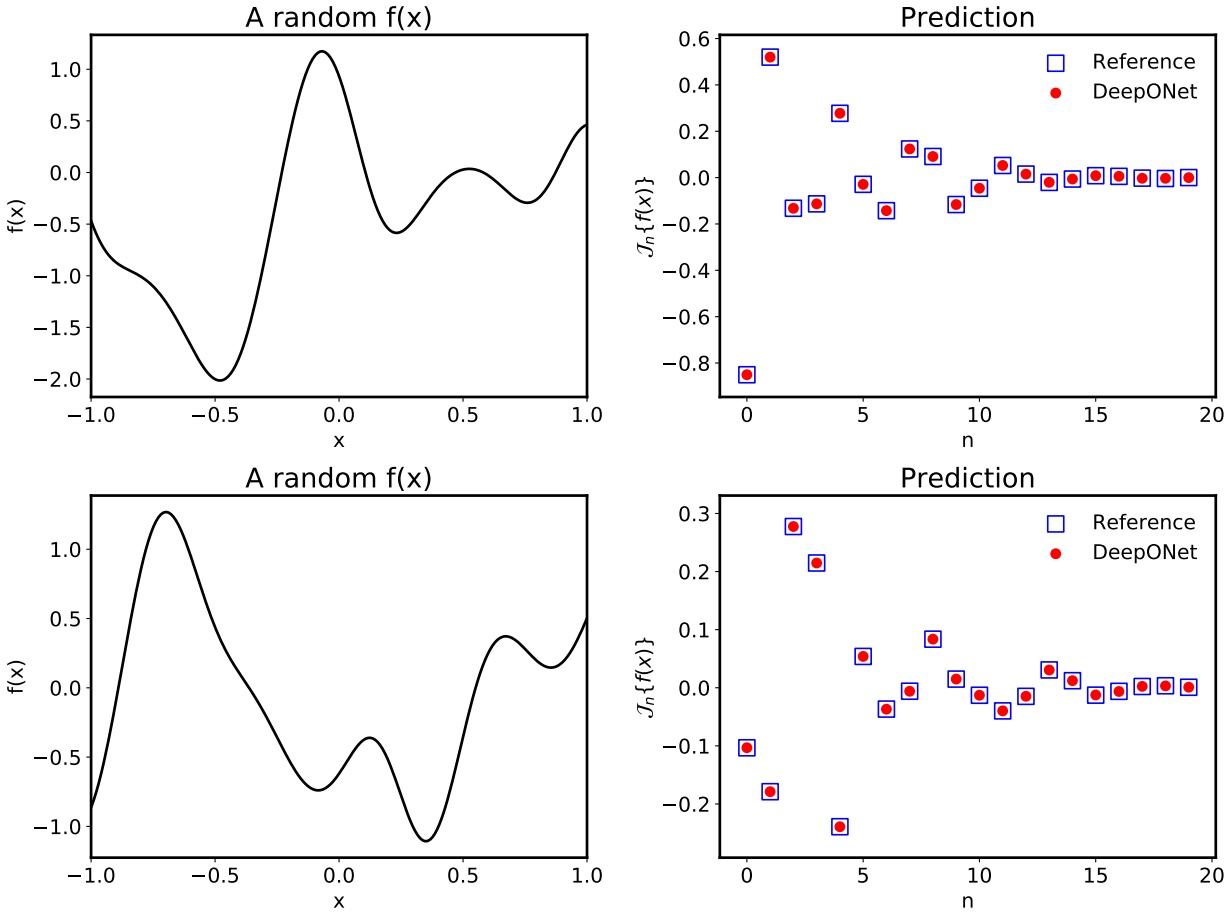

Fig. S2: **DeepONet prediction of the Legendre transform for two random $f(x)$.**

S11 Nonlinear ODE system

We consider a nonlinear 1D ODE system

$$\textbf{Problem 1.C} \quad \frac{ds(x)}{dx} = g(s(x), u(x), x),\ x \in (0, 1],\ \text{where } g(s(x), u(x), x) = -s^2(x) + u(x),$$

with an initial condition $s(0) = 0$. Because $s(x)$ may explode for certain u, we compute the test MSE by removing the 1‰ worst predictions. During the network training, the training MSE and test MSE of both stacked and unstacked DeepONets decrease, but the correlation between training MSE and test MSE of unstacked DeepONets is tighter (Fig. S3A), i.e., smaller generalization error. This tight correlation between training and test MSE of unstacked DeepONets is also observed across multiple runs with random training dataset and network initialization (Fig. S3B). Moreover, the test MSE and training MSE of unstacked DeepONets follow almost a linear correlation

$$\text{MSE}_{\text{test}} \approx 10 \times \text{MSE}_{\text{train}} - 10^{-4}.$$

Fig. S3C shows that unstacked DeepONets have smaller test MSE due to smaller generalization error. DeepONets work even for out-of-distribution predictions, see three examples of the prediction in Fig. S4.

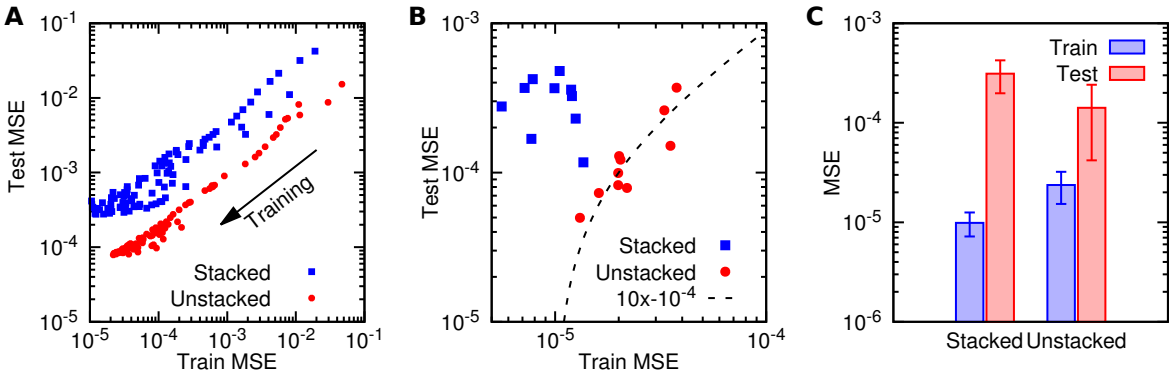

Fig. S3: **Nonlinear ODE: unstacked DeepONets have smaller generalization error and smaller test MSE than stacked DeepONets.** (**A**) The correlation between the training MSE and the test MSE of one stacked/unstacked DeepONet during the training process. (**B**) The correlation between the final training MSE and test MSE of stacked/unstacked DeepONets in 10 runs with random training dataset and network initialization. The training and test MSE of unstacked DeepONets follow a linear correlation (black dash line). (**C**) The mean and one-standard-derivation of data points in B.

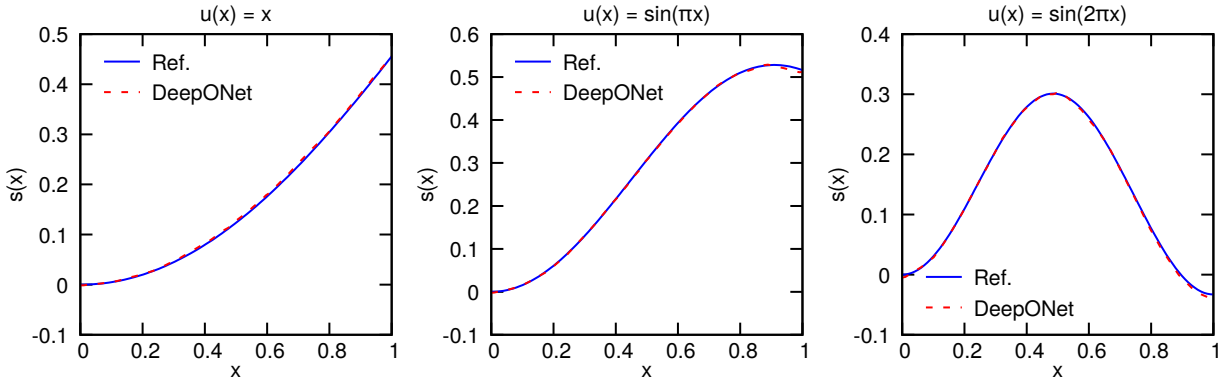

Fig. S4: **Nonlinear ODE: predictions of a trained unstacked DeepONet on three out-of-distribution input signals.** The blue and red lines represent the reference solution and the prediction of a DeepONet.

S12 Gravity pendulum with an external force

S12.1 Number of sensors

The number of sensors required to distinguish two input functions depends on the value of k, prediction time T, and the input function space. For the case with the default parameters $k = 1$, $T = 1$ and $l = 0.2$, when the number of sensors m is small, the error decays exponentially as we increase the number of sensors (Fig. S5A):

$$\text{MSE} \propto 4.6^{-\#\text{sensors}}.$$

When m is already large, the effect of increasing m is negligible. The transition occurs at ∼10 sensors, as indicated by the arrow.

To predict s for a longer time, more sensors are required (Fig. S5B). For example, predicting until $T = 5$ requires ∼25 sensors. As we aim to investigate the transition point of the sensor number for different T, we use the same setup for all T, including the training dataset size, the network size, and the training iterations. The setup is not optimal for large T, and thus the error for large T is large. To achieve a better accuracy for large T, we need to tune the hyperparameters.

If the function u is less smooth corresponding to smaller l, it also requires more sensors (Fig. S5C). However, the number of sensors is not sensitive to k (Fig. S5D). Although it is hard to quantify the exact dependency of m on T and l, by fitting the computational results we show that

$$m \propto \sqrt{T} \quad \text{and} \quad m \propto l^{-1}.$$

In Theorem 3, m should be large enough to make $Te^{cT}\kappa(m, V)$ small. In Section S4 we show theoretically that $m \propto l^{-1}$ for the GRF function space with the RBF kernel, which is consistent with our computational result here. Te^{cT} in the bound is loose compared to the computational results.

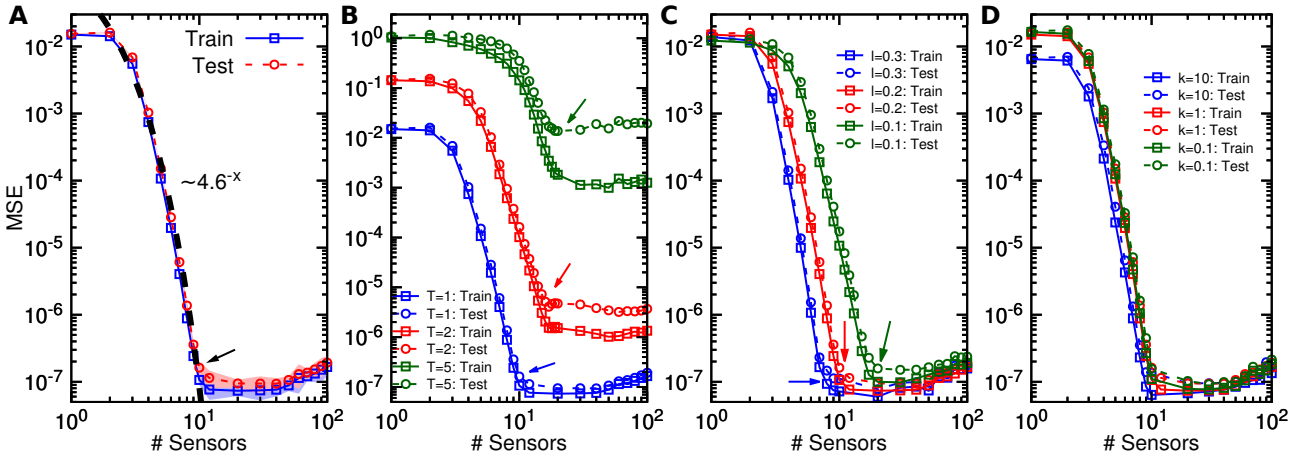

Fig. S5: **Gravity pendulum: required number of sensors for different T, k and l.** (**A**) Training MSE (square symbols) and test MSE (circle symbols) decrease as the number of sensors increases in the case $k = 1$, $T = 1$ and $l = 0.2$. Training and test MSE versus the number of sensors in different conditions of (**B**) T, (**C**) l, and (**D**) k. For clarity, the standard deviation is not shown. The arrow indicates where the rate of the error decay diminishes.

S12.2 Error tendency and convergence

Here, we investigate the error tendency under different conditions, including network size, and training dataset size. We take $T = 3$ for the following experiments in this subsection. By varying the network width, we can observe that there is a best width to achieve the smallest error (Fig. S6A). It is reasonable that increasing the width from 1 to 100 would decrease the error, but the error would instead increase when the width further increases. This could be due to the increased optimization error, and a better learning rate may be used to find a better network. To examine the effect of training dataset size, we choose networks of width 100 to eliminate the network size effect, and the training and test errors using different dataset size are shown in Fig. S6B. The test and generalization errors of networks with width 50 and 200 are shown in Fig. 3.

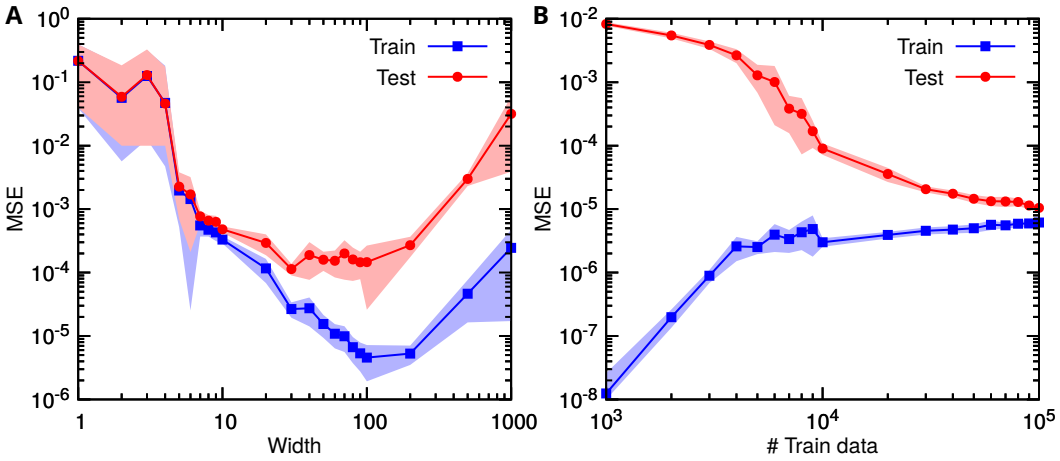

Fig. S6: **Gravity pendulum: error tendency and convergence.** (**A**) Training and test errors first decrease and then increase, when network width increases. (**B**) Test error decreases when more training data are used (width 100).

S12.3 Input function spaces

Next, we investigate the effect of different function spaces, including GRF with different length scale l and the space of Chebyshev polynomials with different number of bases. For a fixed sensor number, we observe that there exists a critical length scale, around where the training and test errors change rapidly, see the arrows in Fig. S7A. This sharp transition around the critical value is also observed in the space of Chebyshev polynomials with different number of bases (Fig. S7B). The relations between the critical values and the sensor numbers are

$$l \propto m^{-1} \quad \text{and} \quad \#\text{Bases} \propto \sqrt{m}.$$

The inverse relation between l and m is consistent with the theoretical results in Section S4 and the computational results in Section S12.1. Therefore, when the function space complexity increases, one may increase m to capture the functions well.

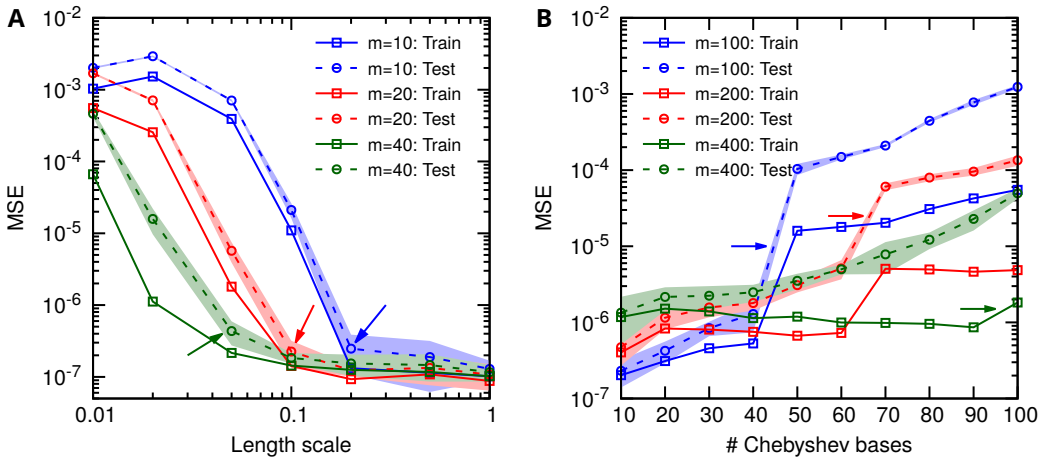

Fig. S7: **Gravity pendulum: errors for different input function spaces.** Training error (solid line) and test error (dash line) for (**A**) GRF function space with different length scale l. The colors correspond to the number of sensors as shown in the inset. (**B**) Chebyshev polynomials for different number of basis functions.

S13 Diffusion-reaction system with a source term

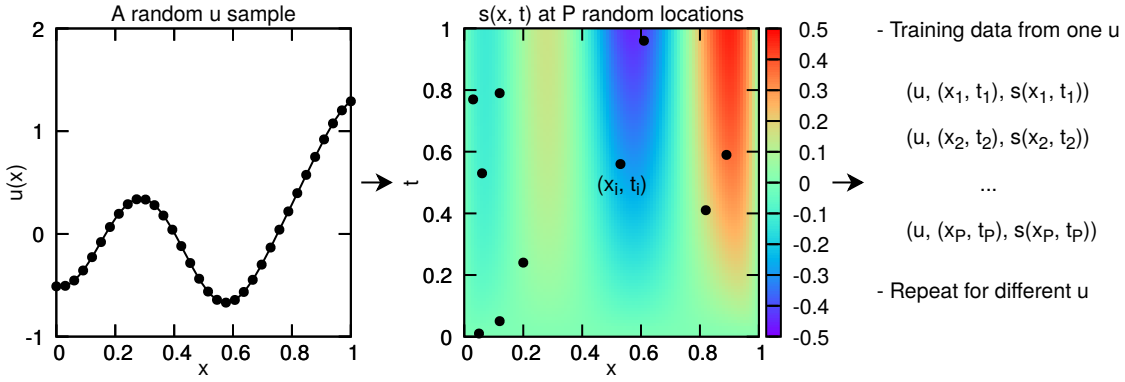

Fig. S8: **Learning a diffusion-reaction system.** (left) An example of a random sample of the input function $u(x)$. (middle) The corresponding output function $s(x,t)$ at P different (x,t) locations. (right) Pairing of inputs and outputs at the training data points. The total number of training data points is the product of P and the number of samples of u.

Table S7: **Learning a diffusion-reaction system.** Mean and standard deviation of training error and test error of ResNets from 10 independent runs.

# Residual block	Width	Training error	Test error
1	20	$2.4 \times 10^{-3} \pm 4.8 \times 10^{-4}$	$9.9 \times 10^{-2} \pm 4.8 \times 10^{-2}$
1	50	$2.4 \times 10^{-4} \pm 2.6 \times 10^{-5}$	$4.3 \times 10^{-1} \pm 1.3 \times 10^{-1}$
1	100	$2.2 \times 10^{-5} \pm 2.0 \times 10^{-6}$	$1.4 \times 10^{-1} \pm 1.8 \times 10^{-2}$
2	20	$2.2 \times 10^{-3} \pm 2.3 \times 10^{-5}$	$6.0 \times 10^{-1} \pm 1.8 \times 10^{-1}$
2	50	$5.8 \times 10^{-5} \pm 7.3 \times 10^{-6}$	$2.6 \times 10^{-1} \pm 5.6 \times 10^{-2}$
3	20	$1.1 \times 10^{-3} \pm 1.3 \times 10^{-4}$	$6.1 \times 10^{-1} \pm 3.0 \times 10^{-1}$
3	50	$2.8 \times 10^{-5} \pm 3.6 \times 10^{-6}$	$2.1 \times 10^{-1} \pm 6.4 \times 10^{-2}$

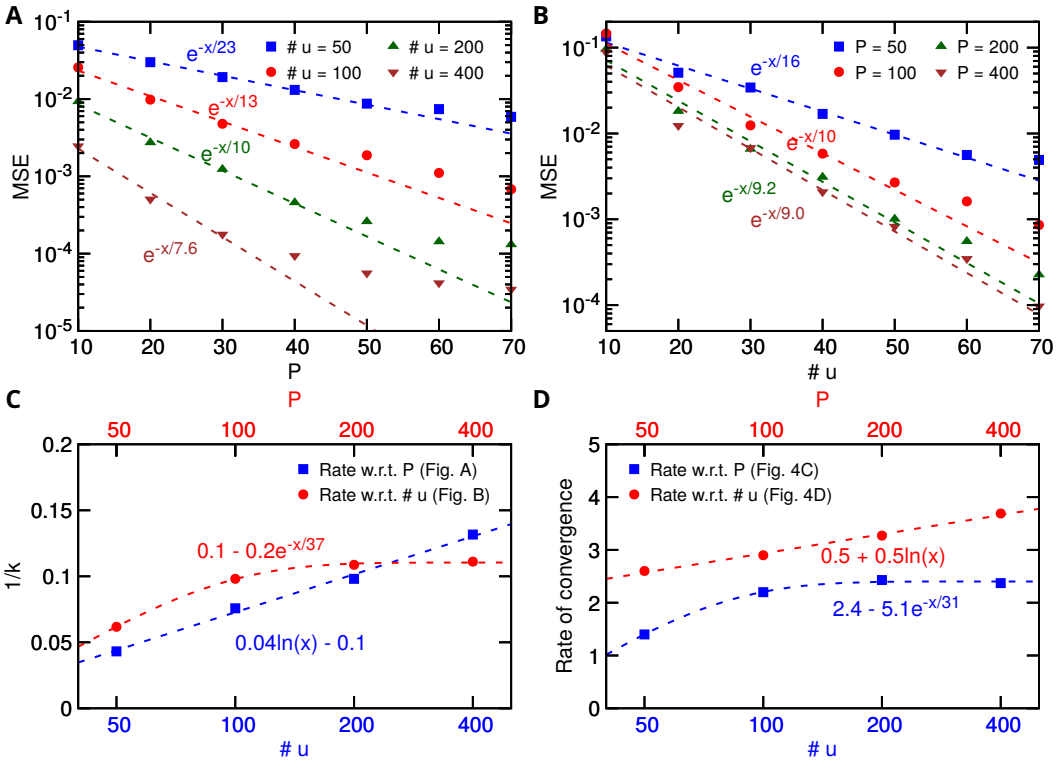

Fig. S9: **Diffusion-reaction system: error convergence rates for different number of training data points.** (**A**) Exponential convergence of test error with respect to P for different number of u samples. (**B**) Exponential convergence of test error with respect to the number of u samples for different values of P. (**C**) The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ versus the number of u samples or the values of P. (**D**) The polynomial rates of convergence versus the number of u samples or the values of P.

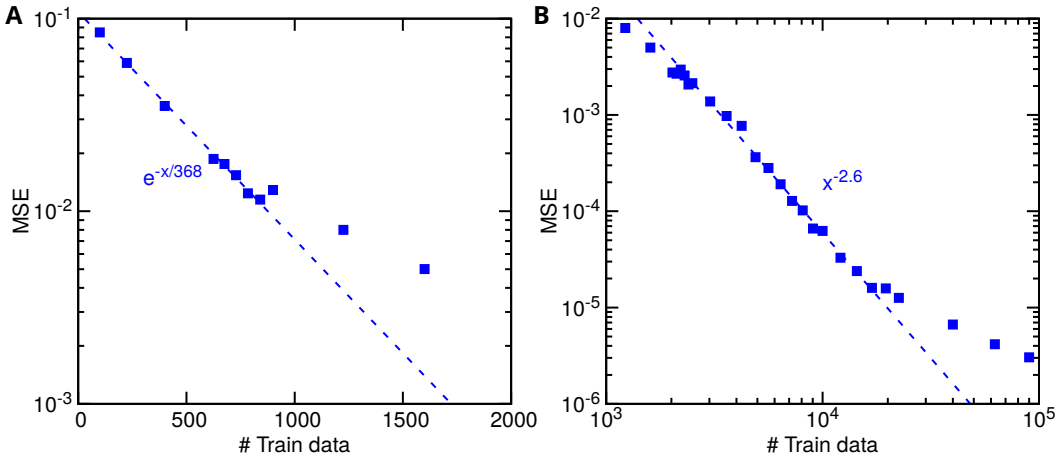

Fig. S10: **Diffusion-reaction system: error convergence rates for different number of training data points when $P = \#u$.** (**A**) Exponential convergence and (**B**) polynomial convergence.

S14 Advection equation

Consider the advection equation

$$\textbf{Problem 8} \quad \frac{\partial s}{\partial t} + a(x)\frac{\partial s}{\partial x} = 0, \quad x \in [0,1], t \in [0,1],$$

and here we learn four operators mapping from different input functions to the solution $s(x,t)$. The choices of $a(x)$, IC/BC and DeepONet input function are listed in Table S8. We note that in these four cases, $a(x)$ is positive. In Cases I and IV we use a periodic BC, and thus the IC is also chosen to be periodic.

To generate the training dataset, we solve the system using a finite difference method on a 100 by 100 grid, except for the Case I with the analytical solution $s(x,t) = f(\sin^2(\pi(x-t))$. Then for each solution $s(x,t)$ we randomly select $P = 100$ points out of these $10000 (= 100 \times 100)$ grid points. The dataset size is equal to the product of P by the number of input function samples. We sample $f(x)$ in Table S8 from a GRF with the Gaussian kernel of different length scale l, and we use 500 random f samples for training.

Table S8: **Setup and results of four different cases of the advection equation.** Mean and standard deviation of the test MSE error are obtained from 10 independent runs.

Case	$a(x)$	IC $s(x,0)$	BC	Input	Length scale l	Test MSE	Example
I	1	$f(\sin^2(\pi x))$	Periodic	$s(x,0)$	0.5	$1.8 \times 10^{-6} \pm 4.2 \times 10^{-7}$	Fig. S11
II	1	$xf(x)$	$s(0,t) = 0$	$s(x,0)$	0.2	$3.7 \times 10^{-7} \pm 1.8 \times 10^{-7}$	Fig. S12
III	$1 + 0.1 \cdot f(x)$	x^2	$s(0,t) = \sin(\pi t)$	$f(x)$	0.2	$1.6 \times 10^{-5} \pm 6.8 \times 10^{-6}$	Fig. S13
IV	$1 + 0.1 \cdot \frac{f(x)+f(1-x)}{2}$	$\sin(2\pi x)$	Periodic	$f(x)$	0.2	$3.2 \times 10^{-5} \pm 1.4 \times 10^{-5}$	Fig. S14

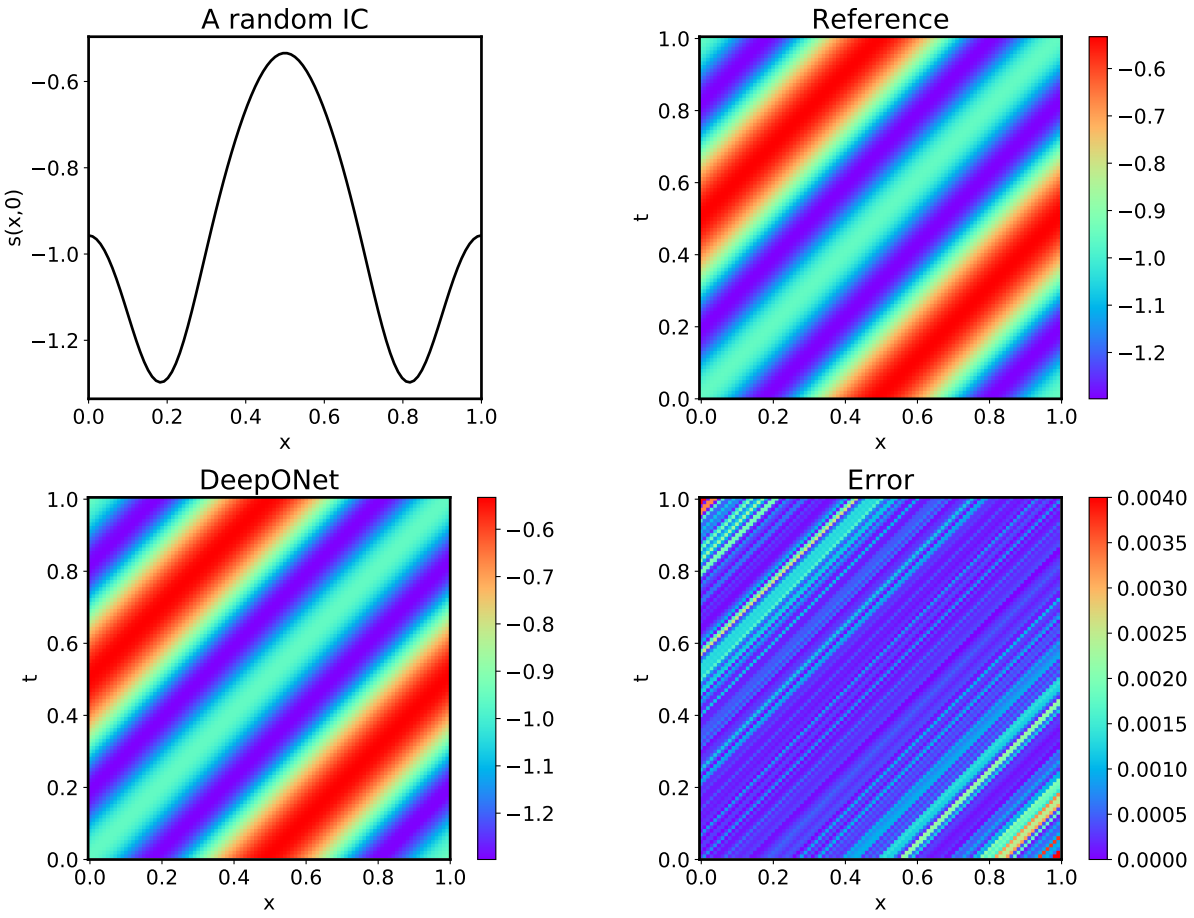

Fig. S11: **DeepONet prediction of the advection equation in Case I for a random IC.**

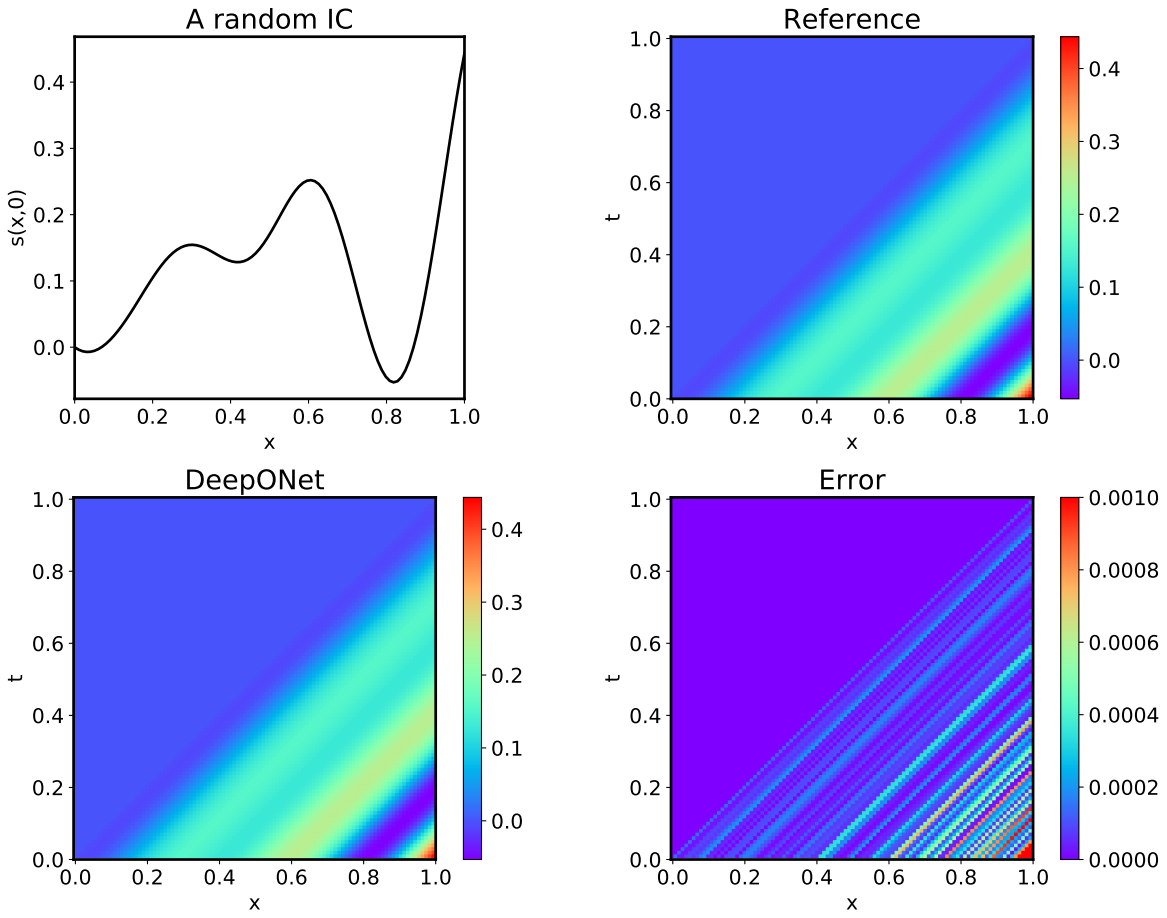

Fig. S12: **DeepONet prediction of the advection equation in Case II for a random IC.**

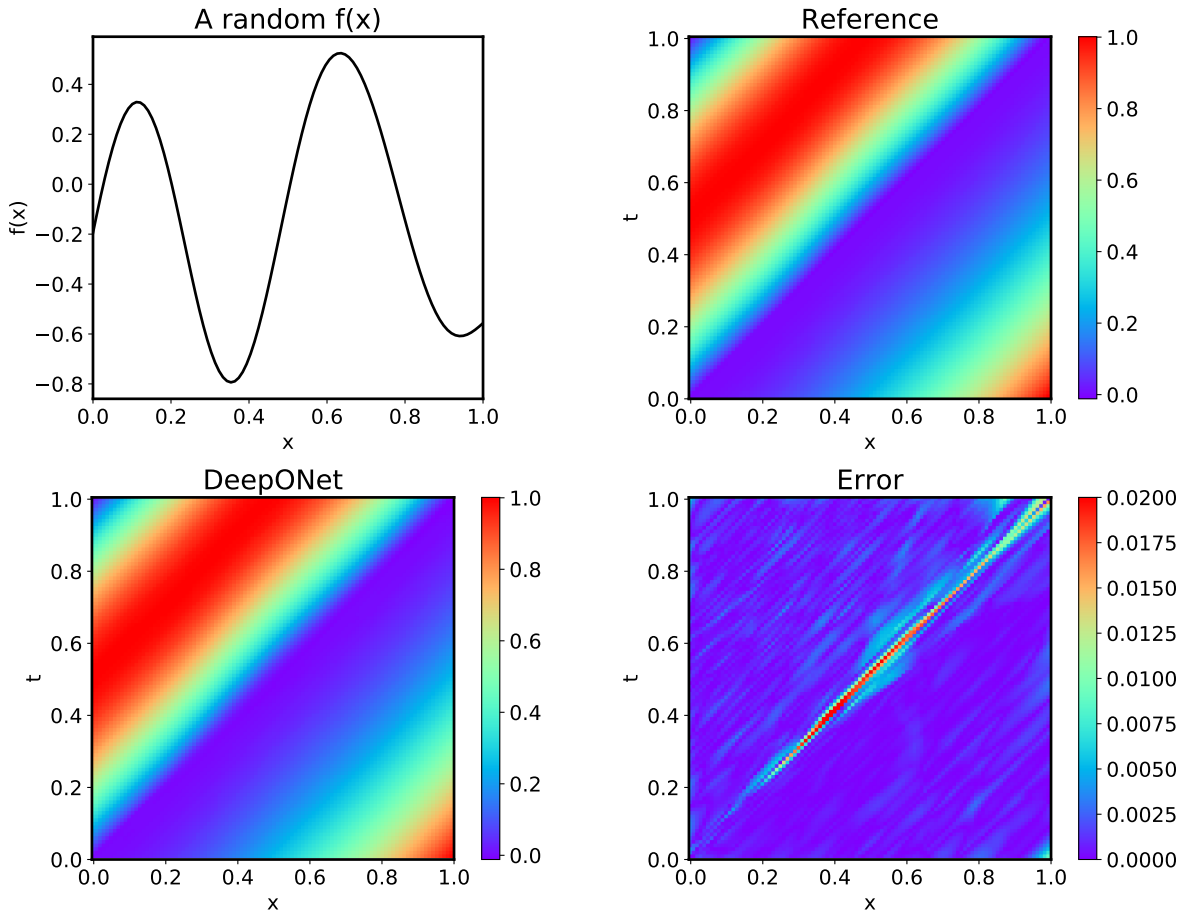

Fig. S13: **DeepONet prediction of the advection equation in Case III for a random $f(x)$.**

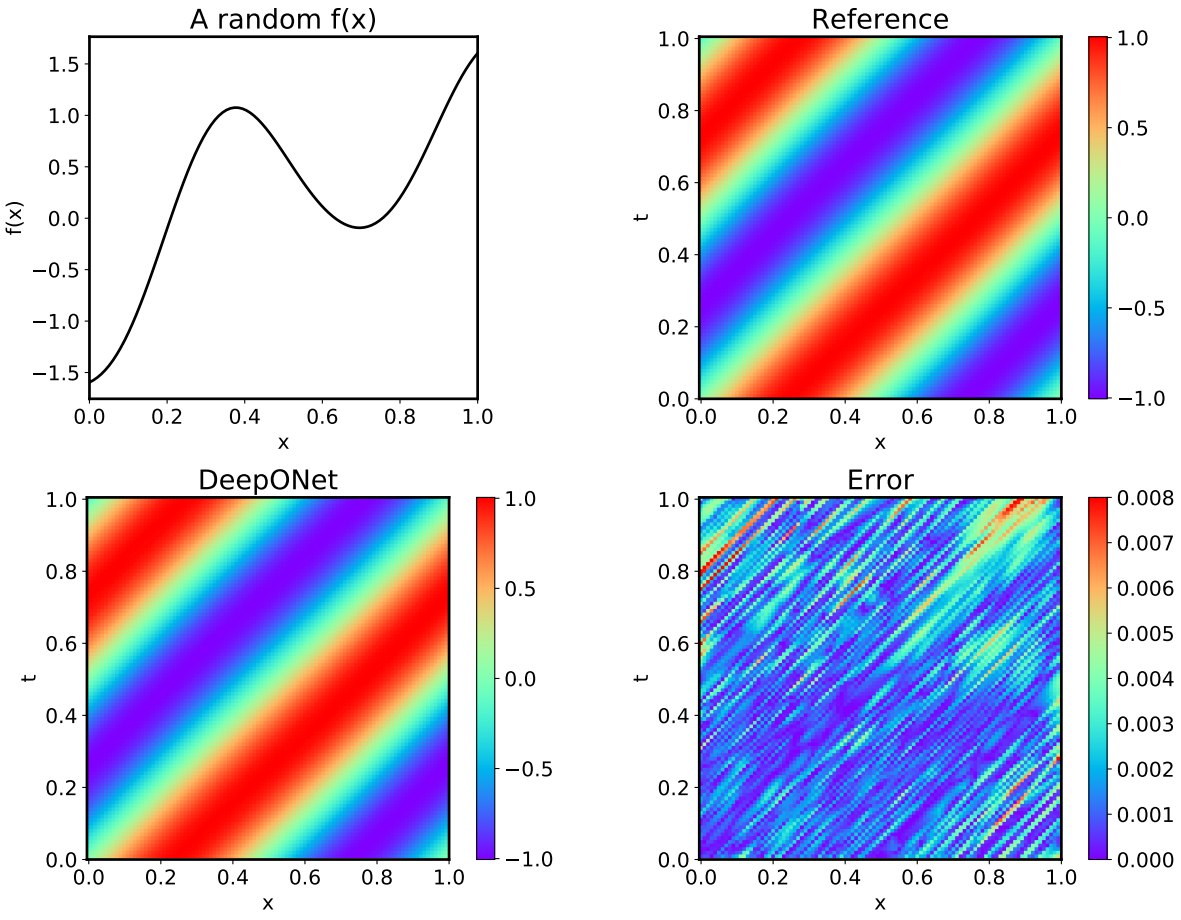

Fig. S14: **DeepONet prediction of the advection equation in Case IV for a random $f(x)$.**

S15 Advection-diffusion equation

Consider the advection-diffusion equation

$$\textbf{Problem 9} \qquad \frac{\partial s}{\partial t} + \frac{\partial s}{\partial x} - D\frac{\partial^2 s}{\partial x^2} = 0, \quad x \in (0,1), t \in (0,1], \tag{S2}$$

with the periodic boundary condition and initial condition $s(x,0) = f(\sin^2(\pi x))$, where $D = 0.1$ is the diffusion coefficient and $f(x)$ is sampled from a GRF with the Gaussian kernel of the length scale $l = 0.5$. Here, $s(x,0)$ is the input function, and we use DeepONets to learn the operator mapping from $s(x,0)$ to the solution $s(x,t)$. To generate the training dataset, we solve the system using a finite difference method on a 100 by 100 grid, and then for each solution $s(x,t)$ we randomly select $P = 100$ points out of these $10000 (= 100 \times 100)$ grid points. The dataset size is equal to the product of P by the number of $s(x,0)$ samples which is set to 500 here. The test MSE is $8.7 \times 10^{-6} \pm 1.1 \times 10^{-6}$ and Fig. S15 is an example for prediction.

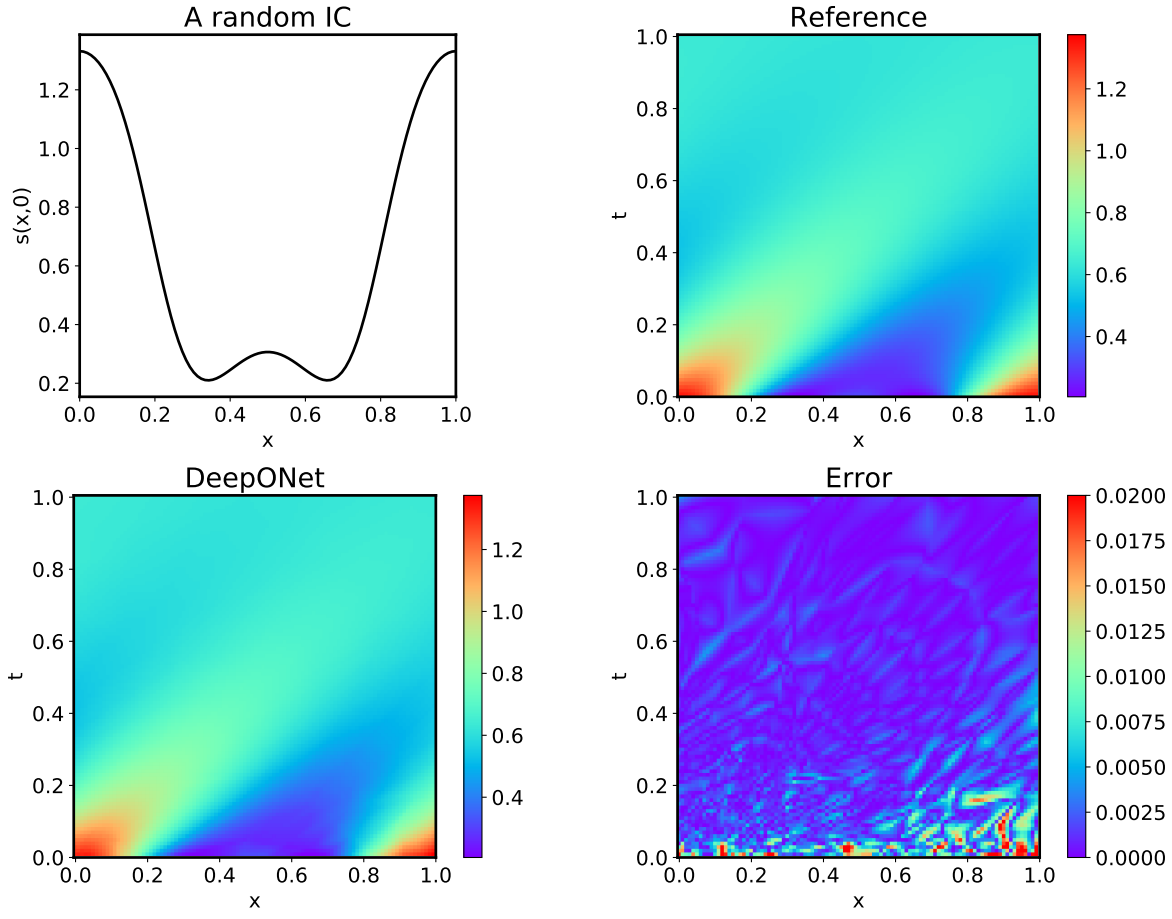

Fig. S15: **DeepONet prediction of the advection-diffusion equation for a random IC.**

S16 Sequence-to-sequence learning

In this section, we introduce the Seq2Seq models used in the antiderivative and gravity pendulum problems.

S16.1 Network architecture

A Seq2Seq model consists of an encoder and a decoder, and we use recurrent neural networks (e.g., LSTM or GRU) as the encoder and decoder [12, 1, 2]. Specifically, take the GRU cell as an example, let $f_{en}(x, h)$ be the encoder cell which maps an input $x \in \mathbb{R}^d$ and a hidden state $h \in \mathbb{R}^s$ to a new hidden state $h' \in \mathbb{R}^s$, where s denotes the hidden size of corresponding cell. Then, for an input sequence

$$\boldsymbol{x} = (x_1, \cdots, x_l), \quad x_i \in \mathbb{R}^d,$$

the encoding process can be written as

$$h_0 = \mathbf{0}, \quad h_i = f_{en}(x_i, h_{i-1}), \quad i = 1, 2, \cdots, l.$$

Before the decoding stage, we apply an attention mechanism [1] to $(h_1, \cdots, h_l)$ as follows. Let $W = (w_{ij}) \in \mathbb{R}^{l \times l}$ be the attention matrix subject to $\sum_j w_{ij} = 1$ and $w_{ij} > 0$, which is implemented by the softmax activation to a weight matrix. Subsequently, the processed hidden states are expressed as

$$\tilde{h}_i = \sum_{j=1}^{l} w_{ij} \cdot h_j, \quad i = 1, 2, \cdots, l.$$

Then, we feed $(\tilde{h}_1, \tilde{h}_2, \cdots, \tilde{h}_l)$ to the decoder as

$$\hat{h}_0 = \mathbf{0}, \quad \hat{h}_i = f_{de}(\tilde{h}_i, \hat{h}_{i-1}), \quad i = 1, 2, \cdots, l,$$

where the decoder cell $f_{de} : \mathbb{R}^s \times \mathbb{R}^s \to \mathbb{R}^s$ is also a GRU cell with input dimension s and hidden size s. Finally, we have the outputs by feeding $(\hat{h}_1, \hat{h}_2, \cdots, \hat{h}_l)$ to a linear layer:

$$\boldsymbol{y} = (y_1, \cdots, y_l), \quad y_i = W_{out} \cdot \hat{h}_i + b_{out} \in \mathbb{R}^n, \quad i = 1, 2, \cdots, l,$$

where $W_{out} \in \mathbb{R}^{n \times s}$ and $b_{out} \in \mathbb{R}^n$ are trainable parameters. In our problems, $d = n = 1$, and the window size l is selected as 100. If the LSTM cell is used instead of the GRU cell, the only modification is adding the cell states in the encoding and decoding stages.

S16.2 antiderivative

We use a Seq2Seq model for **Problem 1.A**. For each input function u sampled from GRF ($l = 0.2$), we use $m = 100$ sensors uniformly distributed on the time interval $[0, 1]$ for both u and its corresponding antiderivative s. The training and test errors for different training dataset and network hyperparameters are in Table S9, and the smallest test error is obtained by using the GRU cell and hidden size 5.

S16.3 Gravity pendulum

We investigate the error convergence of Seq2Seq for **Problem 1.B**. For each input function u sampled from GRF ($l = 0.2$), we use $m = 100$ sensors uniformly distributed on the time interval $[0, 3]$ for both u and its corresponding solution s_1. Based on the accuracy of different architectures for the antiderivative problem, here we choose the GRU cell and the hidden size to be 5.

Table S9: **Learning the antiderivative operator via Seq2Seq.** Mean and standard deviation of training and test errors from 10 independent runs. In the training dataset, $P = 100$.

$\#u$ Train	Cell	Hidden size	Training MSE	Test MSE
100	LSTM	2	$1.4 \times 10^{-5} \pm 1.1 \times 10^{-5}$	$8.5 \times 10^{-5} \pm 1.2 \times 10^{-4}$
100	LSTM	5	$1.0 \times 10^{-5} \pm 4.8 \times 10^{-6}$	$2.3 \times 10^{-4} \pm 1.6 \times 10^{-4}$
100	LSTM	10	$4.0 \times 10^{-6} \pm 2.2 \times 10^{-6}$	$2.5 \times 10^{-4} \pm 1.4 \times 10^{-4}$
100	LSTM	20	$2.0 \times 10^{-6} \pm 7.6 \times 10^{-7}$	$7.1 \times 10^{-4} \pm 1.4 \times 10^{-3}$
100	GRU	2	$2.2 \times 10^{-6} \pm 1.7 \times 10^{-6}$	$8.9 \times 10^{-6} \pm 5.7 \times 10^{-6}$
100	GRU	5	$1.6 \times 10^{-6} \pm 4.1 \times 10^{-7}$	$2.0 \times 10^{-5} \pm 1.3 \times 10^{-5}$
100	GRU	10	$1.1 \times 10^{-6} \pm 2.4 \times 10^{-7}$	$4.0 \times 10^{-5} \pm 2.6 \times 10^{-5}$
100	GRU	20	$6.9 \times 10^{-7} \pm 6.3 \times 10^{-8}$	$8.4 \times 10^{-5} \pm 1.2 \times 10^{-4}$
1000	LSTM	2	$1.1 \times 10^{-5} \pm 6.3 \times 10^{-6}$	$1.9 \times 10^{-5} \pm 1.2 \times 10^{-5}$
1000	LSTM	5	$1.2 \times 10^{-5} \pm 5.4 \times 10^{-6}$	$2.7 \times 10^{-5} \pm 1.6 \times 10^{-5}$
1000	LSTM	10	$4.6 \times 10^{-6} \pm 9.8 \times 10^{-7}$	$1.5 \times 10^{-5} \pm 7.3 \times 10^{-6}$
1000	LSTM	20	$2.3 \times 10^{-6} \pm 5.8 \times 10^{-7}$	$1.2 \times 10^{-5} \pm 1.2 \times 10^{-5}$
1000	GRU	2	$4.2 \times 10^{-6} \pm 7.1 \times 10^{-6}$	$5.7 \times 10^{-6} \pm 7.5 \times 10^{-6}$
1000	GRU	5	$1.6 \times 10^{-6} \pm 1.4 \times 10^{-6}$	$2.6 \times 10^{-6} \pm 1.9 \times 10^{-6}$
1000	GRU	10	$1.7 \times 10^{-6} \pm 3.6 \times 10^{-7}$	$4.4 \times 10^{-6} \pm 1.8 \times 10^{-6}$
1000	GRU	20	$8.6 \times 10^{-7} \pm 1.1 \times 10^{-7}$	$4.7 \times 10^{-6} \pm 2.3 \times 10^{-6}$

S17 Fractional differential operators

Below we detail the data generation and NN architectures for learning the 1D Caputo fractional derivative and the 2D fractional Laplacian.

S17.1 1D Caputo derivative operator

The V spaces we consider are Legendre polynomial space, poly-fractonomial space, and GRF, all of which have been mentioned in Section S3. For the Legendre and poly-fractonomial spaces, we take $M = 2$ and $N = 11$. The expansion coefficients are taken from Sobol sequence, a quasi-random sequence. For the GRF, we set $l = 0.1$ and $l = 0.3$ in the Gaussian kernel for training and test sets, respectively.

Given the discrete version of the input function $u(\cdot)$, i.e., $[u(x_1), \cdots, u(x_m)]^T$, the evaluation point y, and the fractional order α, the reference solution $G(u)(y, \alpha)$ is computed by using the $L1$ scheme [11], a finite difference scheme for approximating Caputo fractional derivative. We take 2000 finite difference points in order to ensure that the computed value of output function is sufficiently accurate.

The training set is determined by the following parameters. The number of different samples of $u(\cdot)$ is $N_u = 10000$. The number of the equi-spaced "sensors" for $u(\cdot)$ is $m = 15$. The numbers of different equi-spaced evaluation points y and α for the output function $G(u)(y, \alpha)$ are both $N_y = N_\alpha = 10$. The number of training points is calculated as $N_u \times N_y \times N_\alpha = 10^6$. The same parameters are taken for the test set.

S17.2 2D fractional Laplacian operator

The V space, where $u(r, \theta)$ is sampled from, is an orthogonal polynomial space spanned by the Zernike polynomial $Z_i(r, \theta)$, where r and θ are the radius and angle in the polar coordinate, respectively. We take 15 basis functions $\{Z_0, Z_1, \cdots, Z_{14}\}$, namely, $N = 15$, and set $M = 2$ for sampling the expansion coefficients a_i in Section S3.

Given the discrete version of the input function $u(\cdot)$, i.e., $[u(x_1), \cdots, u(x_m)]^T$, the evaluation point y, and the fractional order α, the reference solution $G(u)(y, \alpha)$ is computed by using the vector Grünwald-Letnikov formula [10] for approximating Riesz fractional Laplacian. We take 16 Gauss quadrature points for evaluating the integral with respect to differentiation direction and take 2000 finite difference points to approximate fractional directional derivative along a specific differentiation direction.

For DeepONet and CNN models, the raw data are the same. We choose $N_u = 5000$, $N_x = 15 \times 15 = 225$, $N_y = 15 \times 15 = 225$, and $N_\alpha = 10$. But the raw data are organized in different ways for the two models. The DeepONet model organizes the data in a loose way, and thus the size of training set is large, namely, $N_u \times N_y \times N_\alpha = 1.125 \times 10^7$. In contrast, the CNN model leverages the spatial structure of the raw data and rearranges the data into a compact form (image). The size of training set is only $N_u = 5000$. Also, the parameters for the test set are the same as those for the training one.

For the CNN, the filter size is fixed to be 3×3, and the maxpooling window size is 2×2. The input images are shaped to be a 4D array having the shape $[N_u, H_x, W_x, 1]$, where the height (H_x) and width (W_x) of the image for contour plot of one sample of $u(\cdot)$ are taken to be $H_x = W_x = \sqrt{N_x}$. The output images are arranged to another 4D array having the shape $[N_u, H_y, W_y, N_\alpha]$, where the height ($H_y$) and width ($W_y$) of the images for contour plots of $\{G(u)(y, \alpha_1), G(u)(y, \alpha_2), \cdots, G(u)(y, \alpha_{N_\alpha})\}$ are taken to be $H_y = W_y = \sqrt{N_y}$. We denote by N_{chan} the number of channels in a convolution

layer. From the input images to the latent vector, we successively have a convolutional layer with $N_{chan} = 32$, a maxpooling layer, a convolutional layer with $N_{chan} = 64$, and a dense layer with width 64. We take zero padding and stride size 1 for all the above layers. From the latent vector to the output images, we successively have a dense layer with width 64, a dense layer with width 1024 (then reshaped to the shape $[N_u, 4, 4, 64]$), a convolutional layer with $N_{chan} = 32$, padding size 2, and stride 1 (then upsampled to the shape $[N_u, 13, 13, 1]$), and a convolutional layer with $N_{chan} = N_\alpha$, padding size 2 and stride size 1. The CNN architecture we just mentioned are rather similar to encoder and decoder in an autoencoder. We borrow the idea of latent space in the autoencoder, and link the two parts in the CNN using the latent space. The dimensionality of the latent space is set to be 20 in the current example. If we remove the autoencoder structure and instead use a standard CNN, the performance becomes only slightly worse.

S18 Hölder continuity of the operators in this work

For all the explicit and implicit operators in our examples, the operators G are Hölder continuous.

$$\|G(f) - G(g)\|_X \leq C \|f - g\|_Y^{\alpha}, \quad 0 < \alpha \leq 1.$$

Here $C > 0$ depends on f and g and the operator G. Here X and Y are Banach spaces and they refer to the space of continuous functions on a compact set unless otherwise stated.

Let $G_{\mathbb{N}}$ be the approximated G using DeepONet. Let f_h be an approximation of f in Y, possibly by collocation or neural networks. Then

$$\|G(f) - G_{\mathbb{N}}(f_h)\|_X \quad \leq \quad \|G(f) - G(f_h)\|_X + \|G(f_h) - G_{\mathbb{N}}(f_h)\|_X \leq C \|f - f_h\|_Y^{\alpha} + \varepsilon,$$

where ε is the user-defined accuracy as in the universal approximation theorem by neural networks and thus the key is to verify the operator G is Hölder continuous.

The explicit operators and their Lipschitz continuity ($\alpha = 1$) are presented below.

1. (Simple ODE, **Problem 1.A**) $G(u)(x) = s_0 + \int_0^x u(s)\, ds$.

$$\max_{x \in [0,b]} |G(u)(x) - G(v)(x)| \leq b \max_{x \in [0,b]} |u - v|.$$

2. (Caputo derivative, **Problem 2**) The operator is Lipschitz continuous with respect to its argument in weighted Sobolev norms; see e.g. [14, Theorem 2.4].

3. (Integral fractional Laplacian, **Problem 3**) The operator is Lipschitz continuous with respect to its argument in weighted Sobolev norms, see e.g. [6, Theorem 3.3].

4. (Legendre transform, **Problem 7** in Equation (S1)). The Lipschitz continuity of the operator $G(u)(n) = \int_{-1}^{1} P_n(x) u(x)\, dx$ can be seen as follows. For any non-egative integer,

$$\max_n |G(u)(n) - G(v)(n)| \leq \max_n \int_{-1}^{1} |P_n(x)|\, dx \max_{x \in [-1,1]} |u - v| \leq C \max_{x \in [-1,1]} |u - v|$$

   where $C = \max_n \big(\int_{-1}^{1} dx \big)^{1/2} \big(\int_{-1}^{1} |P_n(x)|^2\, dx \big)^{1/2} = \max_n (2 \frac{2}{2n+1})^{1/2} \leq 2$.

5. The linear operator from **Problem 9** in (S2) in Lipschitz continuous with respect to the initial condition in the norm in the space of continuous functions, from he classical theory for linear parabolic equations [9, Chapter IV],

The implicit operators from dynamic systems and differential equations are more complicated. The Hölder continuity of these operators are elaborated in the following text.

The implicit operator from **Problem 1**. The Lipschitz continuity of the operator has been shown in the proof of Theorem 3. For Problem 1.C, the solution is still bounded as long as the input u is not too large. With a bounded solution s, the nonlinear term s^2 is Lipschitz continuous, and thus, the Lipschitz continuity of the operator is still valid, as in the proof of Theorem 3.

The implicit operator from **Problem 4** is Lipschitz with respect to the right-hand side u of not large values, from the classical theory for quasi-linear parabolic equations [9, Chapter V].

The implicit operator from **Problem 5**. $dy(t,\omega) = k(t,\omega) y(t,\omega)\, dt, \quad y(0) = y_0$. Here $k(t)$ is a Gaussian process with mean zero and a smooth covariance function. The exact solution is

$y = y_0 \exp(\int_0^t k(s)\,ds)$. Then the solution operator is Lipschitz continuous in the pathwise sense. Let $k_1(t)$ and $k_2(t)$ be the coefficients in Problem 5 and $y_1(t)$ and $y_2(t)$ are the corresponding solutions. By the mean value theorem, $|e^x - e^y| \le |x-y|\,(e^x + e^y)$ holds for all $x, y \in \mathbb{R}$ and

$$\begin{aligned}
|y_1(t) - y_2(t)| &= |y_0| \left| \exp(\int_0^t k_1(s)\,ds) - \exp(\int_0^t k_2(s)\,ds) \right| \\
&\le |y_0| \,(\exp(\int_0^t k_1(s)\,ds) + \exp(\int_0^t k_2(s)\,ds)) \left| \int_0^t k_1(s) - k_2(s)\,ds \right| \\
&\le |y_0| \exp(\int_0^t k_1(s)\,ds) + \exp(\int_0^t k_2(s)\,ds)\, |k_1 - k_2|_{C([0,T])}\,.
\end{aligned}$$

Then for any $t \in [0,T]$,

$$\|G(k_1)(\cdot,\omega) - G(k_2)(\cdot,\omega)\|_{C[0,T]} \le C(\omega,t)\, |k_1(\cdot,\omega) - k_2(\cdot,\omega)|_{L^\infty([0,t])}\,,$$

where $C(\omega,T) = |y_0|\,(\max_{t\in[0,T]} \exp(\int_0^t k_1(s,\omega)\,ds) + \max_{t\in[0,T]} \exp(\int_0^t k_2(s,\omega)\,ds))$ has moments of any order, according to [3, Proposition 2.3].

The implicit operator from **Problem 6**. The operator can be written as $u = G(b)$ and $u_i = G(b_i)$, $i = 1, 2$ satisfying the following equations:

$$-\mathrm{div}(e^{b_i(x)} \nabla u_i) = f(x), \quad x \in D = (0,1), \qquad u_i(x) = 0, \quad x \in \partial D,$$

Then $\|u_i(\omega)\|_{H^1} \le (\min_{x\in D} e^{-b_i})^{-1}\, \|f\|_{H^{-1}}$, where H^1 and H^{-1} are standard Sobolev-Hilbert spaces. The difference $u_1 - u_2$ satisfies the following

$$-\mathrm{div}(e^{b_1(x)} \nabla (u_1 - u_2)) = \mathrm{div}((e^{b_1(x)} - e^{b_2(x)}) \nabla u_2), \quad x \in D, \qquad u_1 - u_2 = 0, \quad x \in \partial D.$$

Then by the stability of the elliptic equation , we have

$$\begin{aligned}
\|u_1(\omega) - u_2(\omega)\|_{H^1} &\le (\min_{x\in D} e^{b_i(x)})^{-1} \left\| (e^{b_1} - e^{b_2}) \nabla u_2 \right\|_{L^2} \\
&= (\min_{x\in D} e^{b_1(x)})^{-1} \left\| e^{b_1} - e^{b_2} \right\|_{C(D)} \|u_2\|_{H^1} \\
&\le (\min_{x\in D} e^{b_1(x)})^{-1} (\min_{x\in D} e^{b_2(x)})^{-1} \left\| e^{b_1} - e^{b_2} \right\|_{C(D)} \|f\|_{H^{-1}}\,.
\end{aligned}$$

Then by the mean value theorem, $|e^x - e^y| \le |x-y|\,(e^x + e^y)$ holds for all $x, y \in \mathbb{R}$. Thus,

$$\|u_1(\omega) - u_2(\omega)\|_{H_0^1} \le (\min_{x\in D} e^{b_1(x)})^{-1} (\min_{x\in D} e^{b_2(x)})^{-1} (\left\| e^{b_1} \right\|_{C(D)} + \left\| e^{b_2} \right\|_{C(D)})\, \|b_1 - b_2\|_{C(D)}\, \|f\|_{H^{-1}}\,.$$

According to [3, Proposition 2.3], all the random variables $\min_{x\in D} e^{b_i(x)})^{-1}$ and $\left\| e^{b_i} \right\|_{C(D)}$, $i = 1, 2$ have any moments of finite order. Then, we obtain the pathwise Lipschitz continuity of the operator G

$$\|\mathcal{G}(b_1)(\omega) - \mathcal{G}(b_2)(\omega)\|_{H^1} \le C(\omega)\, \|b_1 - b_2\|_{C(D)}\,.$$

Here $C(\omega) = (\min_{x\in D} e^{b_1(x)})^{-1} (\min_{x\in D} e^{b_2(x)})^{-1} (\left\| e^{b_1} \right\|_{C(D)} + \left\| e^{b_2} \right\|_{C(D)})\, \|f\|_{H^{-1}}$.

The implicit operator in **Problem 8** is also Lipschitz continuous. The solution can be represented as $s_0(X^{-1}(t,x))$, where $X(t,x)$ satisfies the following equation

$$\dot{X}(t,x) = a(t, X(t,x)), \quad X(0,x) = x.$$

As there exists $a_0 > 0$ such that $a(t,x) > a_0 > 0$ in the setting of Problem 8, $X(t,x) > 0$ is away from zero. Denote the solution with coefficient a and initial condition u_0 by $u_0(X_a^{-1})$ and the solution with coefficient b and initial condition v_0 by $v_0(X_b^{-1})$. We then can derive the Lipschitz continuity of the operator from the following facts that

$$u_0(X_a^{-1}(t,x)) - v_0(X_b^{-1})(t,x) = \big(u_0(X_a^{-1}(t,x)) - u_0(X_b^{-1}(t,x))\big) + \big(u_0(X_b^{-1}(t,x)) - v_0(X_b^{-1})(t,x)\big)$$

and that u_0, v_0 are Lipschitz continuous and X_a and X_a^{-1} (X_b and X_b^{-1}) are Lipchitz continuous with respect to a (b).

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] D. Britz, A. Goldie, M. Luong, and Q. Le. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*, 2017.

[3] J. Charrier. Strong and weak error estimates for elliptic partial differential equations with random coefficients. *SIAM Journal on Numerical Analysis*, 50(1):216–246, 2012.

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

[5] M. Gelbrich. On a formula for the l^2 Wasserstein metric between measures on Euclidean and Hilbert spaces. *Mathematische Nachrichten*, 147(1):185–203, 1990.

[6] Z. Hao, H. Li, Z. Zhang, and Z. Zhang. Sharp error estimates of a spectral Galerkin method for a diffusion-reaction equation with integral fractional Laplacian on a disk. *Submitted*, 2019. `https://www.researchgate.net/publication/335892990`.

[7] X. He, K. Zhao, and X. Chu. AutoML: A survey of the state-of-the-art. *arXiv preprint arXiv:1908.00709*, 2019.

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[9] O. A. Ladyženskaja, V. A. Solonnikov, and N. N. Ural'ceva. *Linear and quasilinear equations of parabolic type*. AMS, Providence, R.I., 1968.

[10] M. M. Meerschaert, J. Mortensen, and H.-P. Scheffler. Vector grunwald formula for fractional derivatives. *Fractional Calculus and Applied Analysis*, 7(1):61–82, 2004.

[11] Z. Sun and X. Wu. A fully discrete difference scheme for a diffusion-wave system. *Applied Numerical Mathematics*, 56(2):193–209, 2006.

[12] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[13] Z. Zhang and G. E. Karniadakis. *Numerical methods for stochastic partial differential equations with white noise*. Springer, 2017.

[14] Z. Zhang, F. Zeng, and G. E. Karniadakis. Optimal error estimates of spectral Petrov-Galerkin and collocation methods for initial value problems of fractional differential equations. *SIAM Journal on Numerical Analysis*, 53(4):2074–2096, 2015.