axioms

MDPI

Article

An Efficient Method for Solving Two-Dimensional Partial Differential Equations with the Deep Operator Network

Xiaoyu Zhang [1,2], Yichao Wang [3], Xiting Peng [2,3,*] and Chaofeng Zhang [4]

[1] School of Artificial Intelligence, Shenyang University of Technology, Shenyang 110870, China; xy.zhang@sut.edu.cn
[2] Shenyang Key Laboratory of Information Perception and Edge Computing, Shenyang University of Technology, Shenyang 110870, China
[3] School of Information Science and Engineering, Shenyang University of Technology, Shenyang 110870, China; wangyichao@smail.sut.edu.cn
[4] School of Information and Electronic Engineering, Advanced Institute of Industrial Technology, Tokyo 140-0011, Japan; zhang-chaofeng@aiit.ac.jp
* Correspondence: xt.peng@sut.edu.cn

Abstract: Partial differential equations (PDEs) usually apply for modeling complex physical phenomena in the real world, and the corresponding solution is the key to interpreting these problems. Generally, traditional solving methods suffer from inefficiency and time consumption. At the same time, the current rise in machine learning algorithms, represented by the Deep Operator Network (DeepONet), could compensate for these shortcomings and effectively predict the solutions of PDEs by learning the operators from the data. The current deep learning-based methods focus on solving one-dimensional PDEs, but the research on higher-dimensional problems is still in development. Therefore, this paper proposes an efficient scheme to predict the solution of two-dimensional PDEs with improved DeepONet. In order to construct the data needed for training, the functions are sampled from a classical function space and produce the corresponding two-dimensional data. The difference method is used to obtain the numerical solutions of the PDEs and form a point-value data set. For training the network, the matrix representing two-dimensional functions is processed to form vectors and adapt the DeepONet model perfectly. In addition, we theoretically prove that the discrete point division of the data ensures that the model loss is guaranteed to be in a small range. This method is verified for predicting the two-dimensional Poisson equation and heat conduction equation solutions through experiments. Compared with other methods, the proposed scheme is simple and effective.

Keywords: partial differential equations; deep operator network; predicting the solution; machine learning

MSC: 68T07; 35Q68

Citation: Zhang, X.; Wang, Y.; Peng, X.; Zhang, C. An Efficient Method for Solving Two-Dimensional Partial Differential Equations with the Deep Operator Network. *Axioms* **2023**, *12*, 1095. https://doi.org/10.3390/axioms12121095

Academic Editor: Behzad Djafari-Rouhani

Received: 31 October 2023
Revised: 23 November 2023
Accepted: 27 November 2023
Published: 29 November 2023

1. Introduction

A partial differential equation (PDE) is a kind of equation that deals with the derivatives of unknown variables to time and space variables. These equations can describe natural phenomena (e.g., hydrodynamics [1] and electromagnetism [2]). In addition, it also describes practical applications (e.g., solving population problems [3] and urban traffic problems [4]). People often pay more attention to the analysis and solution of complex PDE models. In recent decades, the research of numerical methods for solving PDEs has never been interrupted. There are finite difference methods [5], finite element methods [6], spectral methods [7], boundary element methods [8], finite volume methods [9], Monte Carlo methods [10], and adaptive mesh refinement. With the development of the research,

many new numerical methods appear (e.g., the fast curvilinear finite difference method [11], multiscale RBF collocation method [12], Haar wavelet collocation method [13], and sinc-collocation method [14]). However, we often need to analyze and design PDE solution algorithms via numerical methods. The calculation cost and resource consumption will increase as PDEs become more and more complex. Meanwhile, we can simplify PDEs by using the variational method, separation of variables method, Green function method, transformation method, and deep learning method. These make it possible to solve PDEs quickly and accurately.

With the rapid development of artificial intelligence technology, people pay more and more attention to using deep learning methods to analyze complex problems. A mesh-free algorithm called the deep Galerkin method [15] is proposed to approximate high-dimensional PDEs. In [16], the authors propose a scheme that embeds constraints in the Physics-informed neural networks of loss functions to predict PDEs with specific conditions. In addition, some PDE models have successfully solved practical application problems. For instance, Physics-informed neural networks [16,17] are used to predict the fluid flow in porous media [18] and simulate heat transfer [19]. Although these methods have successfully solved various problems, they are time-consuming, laborious, and not universal. When changing conditions, these strategy models need to be retrained.

The latest application of deep learning in PDEs comes from the universal approximation theory that shows that a neural network can accurately approximate any continuous nonlinear function or operator [20]. Based on this theory, Lu et al. [21] recently proposed a new neural network architecture called the Deep Operator Network (DeepONet). The architecture is composed of branch and trunk networks, which solve a class of partial differential equations through learning operators (maps between infinite-dimensional function spaces). To demonstrate the capability and effectiveness of DeepONet, some researchers have also analyzed the error and convergence and proved theoretically it can overcome the disaster of dimension [22]. Recent theories suggest that DeepONet has successfully solved various problems. In [23], it is used as a learning operator to predict the multiscale bubble dynamics. In [24], DeepONet is used to predict the linear unstable waves in the high-speed boundary layer according to the data provided in advance. In addition, DeepM&Mnet predicts a two-dimensional electric pair flow field [25] and predicts the coupled flow and finite rate chemical reaction after impacting normal [26] by integrating multiple parallel DeepONets. We consider that the high-dimensional problems most widely exist in a real scenario. Therefore, we focus on solving two-dimensional PDEs by using DeepONet.

For the original model, the branch network receives a function, which is a vector consisting of m discrete points, and the trunk network input is the information of spatial and time variables. However, when we use it to solve the two-dimensional PDEs, the input function is expressed as a matrix, and the spatial variables increase to two. At this time, the input function cannot be the branch network input, and the variable information of the trunk network input also increases. To cope with these challenges, Cai et al. [25] use a fully connected network and a pre-trained DeepONet to predict the 2D electroconvection problem by minimizing the total loss. In [22,27], the input function can be regarded as an image and the branch network can be replaced by a convolutional neural network (CNN) [28] to learn the mapping between two variables. However, these methods increase the complexity of the model and lack complete theoretical support.

To deal with model redundancy, we designed an efficient framework to solve two-dimensional PDEs based on DeepONet. First, starting with the input function, the input matrix is reshaped into a vector. The input function can be regarded as the input of the branch network. Second, because the two-dimensional PDEs contain more location and time information, we add spatial and time variable information as the input of the trunk network. Finally, according to the data form of the branch and trunk networks, we expand the data set required by DeepONet to approximate the solutions of PDEs. Firstly, the reduced dimension input function and the added spatial and temporal variable information form a binary. This part represents the input information of DeepONet. Then,

the numerical solution of the corresponding variable is obtained according to the numerical solver, and these three parts together form a data set. In addition, we also designed a method to randomly select the location information from the corresponding time variables to consist of the data set. The proposed method can predict the solution of PDEs efficiently and accurately without increasing the complexity of the model. Moreover, it can solve a class of PDEs when only the input function is changed, so it is universal. All told, the main contributions of this paper are as follows.

1. To obtain enough two-dimensional PDE data for training, we design a method to construct binary functions in the classical function space and obtain the corresponding PDE solutions through the difference method to form point-value data sets. To prove the DeepONet output is closer to the actual value, we analyze the constructed function theoretically that can be applied for model training.
2. To solve the challenge of the form of the input data, we rely on DeepONet to design a new data dimensionality reduction algorithm. This method flattens the matrix data representing the function into vectors for inputting the branch network and randomly selects the location and time information as the trunk network input.
3. The proposed strategy is performed on the two-dimensional Poisson equation and the heat conduction equation experiments. Compared with other methods, we verify that it is accurate and efficient in predicting two-dimensional PDEs.

This paper is organized as follows. The proposed method, data set, and training scheme are introduced in Section 2. Next, we formalize the problem and theoretically prove the relationship between the discrete points and model accuracy. The fourth part shows the experimental results of the proposed method on two-dimensional Poisson equations and heat conduction equations. Finally, we summarize this paper.

2. Methodology

In this section, we first review the work on deep operator networks and present a method for solving two-dimensional PDEs based on it. Next, we explain the source of the data set and the proposed training strategies.

2.1. Deep Operator Network Architecture

Here, we introduce the background of DeepONet. It is a simple neural network architecture that consists of two subnetworks. One is a branch network, which encodes the discrete points of the input function. The other one is a trunk network, which encodes the coordinate position of the output function. Each subnetwork is a feed-forward neural network (FNN). This network predicts PDE solutions by learning nonlinear operators from input to output. Generally, we define many points $[x_1, x_2, \ldots, x_m]$ in domain f. $[f(x_1), f(x_2), \ldots, f(x_m)]$ represent the function f and the branch network inputs. The trunk network inputs are the time and space variables x, t, and two network outputs must be the same dimensional vectors (Figure 1) to conduct a dot-product operation. If Ψ is an operator that maps from input to solution, the final output $\Psi(f)(x, t)$ is the solution of the PDEs.

When solving a one-dimensional partial differential equation, the input function is input into a branch network as a finite number of equidistant discrete points. The trunk network inputs are spatial variables and time variable information. However, when we solve two-dimensional PDEs, the input function is expressed as a matrix that cannot be the branch network inputs. Therefore, we propose a scheme to overcome this problem. For the branch network input, we convert the matrix representing the two-dimensional function into a vector (Figure 2). For the trunk network input, we extend it to two spatial variables and the time variable information. The proposed approach also uses a bias term for the output. Algorithm 1 represents the dimensionality reduction process of the input function. It means taking out each row of the matrix and appending it to the end of the previous row.

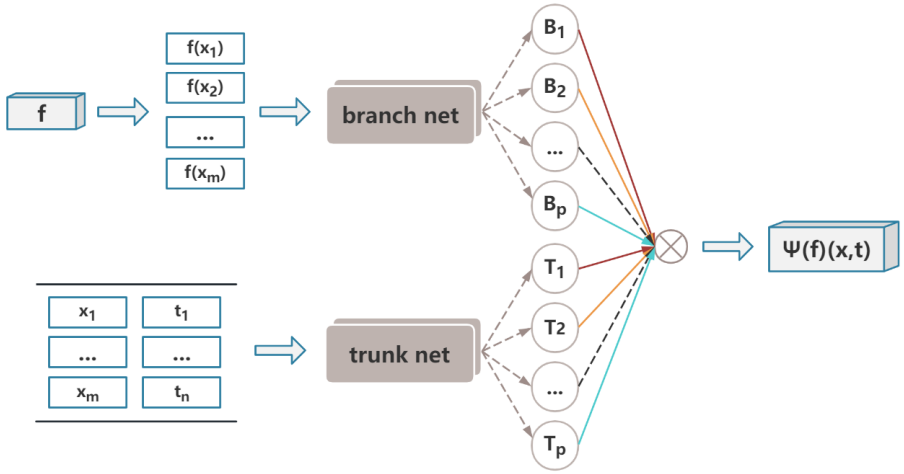


Figure 1. DeepONet architecture.

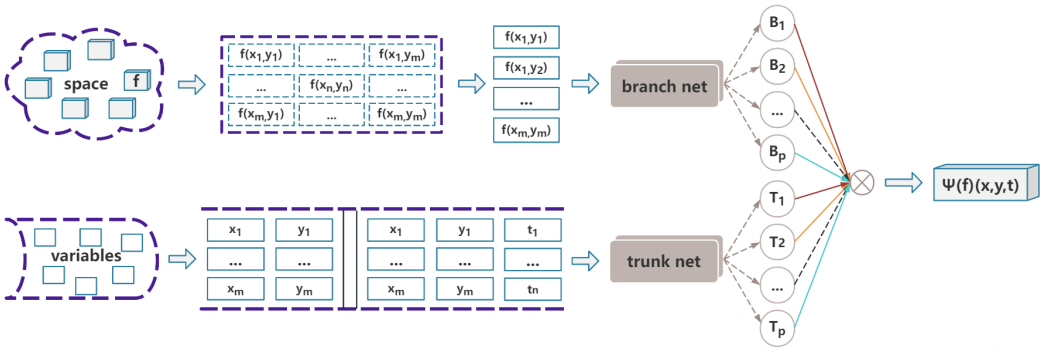


Figure 2. The proposed method.

Algorithm 1 Function dimensionality reduction

input function $f(x, y)$
output discrete points vector F represents the function $f(x, y)$
1: using matrix F to represent function $f(x, y)$
2: **for** each row of the matrix **do**
3: append to the previous line
4: **end for**
5: thus, matrix F becomes vector F

2.2. The Data Set and Training Scheme

The data set consists of two parts: input and output. Due to input functions being essential for model training, we designed a random sampling input functions method. It is divided into two steps. In step one, we obtain univariate functions in the Gaussian random field (GRF).

$$R \sim Q(0, k_l(x_1, x_2)) \tag{1}$$
$$Y \sim Q(0, k_l(x_1, x_2)) \tag{2}$$

where R, Y are different input signals, k is the radial-basis function (RBF) kernel, $l > 0$ is a length-scale parameter, and $k_l(x_1, x_2) = exp(-||x_1 - x_2||^2/2l^2)$. If l is larger, the function R and Y is smoother. In step two, we randomly choose two univariate functions for a linear combination. c_1, c_2 are constants.

$$F = c_1 R + c_2 Y \tag{3}$$

The variable information of the trunk network is location points randomly selected according to the range of independent variables. In addition, the most important is to obtain PDE solutions to train the model. Because it is difficult to gain the analytical solution, we consider using finite difference methods to obtain the numerical solution. A training point is a triple $(F,(x,y,t),\Psi(F)(x,y,t))$. It may correspond to multiple training data points at different positions for specific input. For example, if the training data points are 36,000 and the number of functions is 200, each input function includes 180 training data points at different locations. Because the data points are randomly selected, they have different positions in various inputs. Then, we consider two methods of combining the data sets. In one, we randomly form triplets with two spatial variables and a time variable x,y,t; the input function F; and the result value $\Psi(F)(x,y,t)$. In the other, we randomly sample points in each period and form triples $(F,(x,y,t),\Psi(F)(x,y,t))$. Algorithms 2 and 3 display the two training methods.

Algorithm 2 Training strategy one

input input function F, independent variable x,y,t
output triples $(F,(x,y,t),\Psi(F)(x,y,t))$
A random combination of x, y, t
set $x_j=[x_1,\ldots,x_m],y_i=[y_1,\ldots,y_m],t_k=[t_1,\ldots,t_n].\,j,i=1,2,\ldots,m.\,k=1,2,\ldots,n.$
for training points **do**
    randomly select x_j
    randomly select y_i
    randomly select t_k
end for
form the matrix (x_j,y_i,t_k) and change into (x,y,t)
Processing of PDE solution
obtain PDE solutions through solvers
transform into the matrix $\Psi(F)(x,y,t)$
combine with matrix (x,y,t) and get two tuples $((x,y,t),\Psi(F)(x,y,t))$
add F and get triples $(F,(x,y,t),\Psi(F)(x,y,t))$

Algorithm 3 Training strategy two

input input function F, independent variable x,y,t
output triples $(F,(x,y,t),\Psi(F)(x,y,t))$
Processing of time period
set $t_k=[t_1,\ldots,t_n],k=1,2,\ldots,n.$
A random combination of x,y,t
set $x_j=[x_1,\ldots,x_m],y_i=[y_1,\ldots,y_m].\,j,i=1,2,\ldots,m.$
for n times **do**
    for training points **do**
        randomly select x_j
        randomly select y_i
    end for
end for
combine t_k to form the matrix (x_j,y_i,t_k) and change into (x,y,t)
Processing of PDE solution
obtain PDE solutions through solvers
transform into the matrix $\Psi(F)(x,y,t)$
combine with (x,y,t) and get two tuples $((x,y,t),\Psi(F)(x,y,t))$
add F and get triples $(F,(x,y,t),\Psi(F)(x,y,t))$

3. Problem Formulation

In this part, we demonstrate an influence on model accuracy with the number of discrete points. We keep the two-dimensional input function sampling space from the GRF

and design a method to sample it. Because the sampling space remains unchanged and the input function is a two-dimensional function, it is necessary to extend the proof. Suppose that the two-dimensional heat conduction problem with the source term is described as follows:

$$\begin{cases} \frac{\partial s}{\partial t} = g[s(x,y,t), f(x,y), (x,y)] \\ IC : s(x,y,0) = s_0 \\ BC : 0 \end{cases} \tag{4}$$

$f(x,y) = f_1(x) + f_2(y) \in V$ (compact set of Banach space), and the domain of s is $[n,k] \times [n,k] \to \mathbb{R}^k$. Ψ is an operator that is from f to s, ($\Psi_f(x,y,t) = s$). Ψ_f satisfies

$$\Psi_f(x,y,t) = s_0 + \iiint g[\Psi_f(l,p,h), f(l,p), (l,p)]dldpdh$$

and choose $(m+1) \times (m+1)$ points in the domain.

$$x_j = n + \frac{j(k-n)}{m}, j = 1, 2, \ldots, m.$$

$$y_i = n + \frac{i(k-n)}{m}, i = 1, 2, \ldots, m.$$

where $x_j, y_i \in [n,k]$. Define the function $f_m(x,y) = f_m(x) + f_m(y)$.

$$f_m(x) = f(x_j) + \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} + x_j}(x - x_j)$$

$$f_m(y) = f(y_i) + \frac{f(y_{i+1}) - f(y_i)}{y_{i+1} + y_i}(y - y_i)$$

$$f_m(x,y) = f(x_j) + \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} + x_j}(x - x_j) + f(y_i) + \frac{f(y_{i+1}) - f(y_i)}{y_{i+1} + y_i}(y - y_i)$$

Y is an operator that is from $f(x,y)$ to $f_m(x,y)$. $P_m = \Upsilon_m(f)|f \in V$, V is a compact set of the Banach space; V is compact and Y is also compact. Hence, $Z_m := V \cap P_m$ also remain compact. According to Lemma 1, $Z := \bigcup_{i=1}^{\infty} Z_i$ is also compact. Ψ is a continuous implicit operator, and $\Psi(Z)$ is a compact set. Suppose $g(s, f, (x,y))$ satisfies the Lipschitz condition, $s, f \in \Psi(Z) \times Z$, there exists a constant c such that

$$||g(s_1, f, (x,y)) - g(s_2, f, (x,y))||_2 \leqslant c||s_1 - s_2||_2$$

$$||g(s, f_1, (x,y)) - g(s, f_2, (x,y))||_2 \leqslant c||f_1 - f_2||$$

To achieve this condition, g is differentiable about f and s on $\Psi(Z) \times Z$. For $f \in V, f_m \in P_m$, c_1, c_2 and $s(m,V)$ are constant such that

$$\underset{x,y \in [n,k]}{Max} |f(x,y) - f_m(x,y)| \leqslant (c_1 + c_2)s(m,V) \tag{5}$$

$(c_1 + c_2)s(m,V) \to 0$ when $m \to \infty$ According to Theorem 1, we have $s \sim \frac{1}{m^2 l^2}$. Based on the above deduction, we obtain Theorem 2.

Lemma 1. *$Z := \bigcup_{i=1}^{\infty} Z_i$ is compact.*

Proof. For any $\varepsilon > 0$, by (5), there exists an m_0, such that:

$$||f - \Upsilon_m(f)||_c < \frac{\varepsilon}{4}, \forall f \in V, \forall m > m_0$$

Z_{m_0} is a compact set, and there exists a $\delta > 0$ such that:

$$|(x_1, y_1) - (x_2, y_1)| < \delta \Longrightarrow |f(x_1, y_1) - f(x_2, y_1)| < \frac{\varepsilon}{2}$$

$$|(x_1, y_1) - (x_1, y_2)| < \delta \Longrightarrow |f(x_1, y_1) - f(x_1, y_2)| < \frac{\varepsilon}{2}$$

$\forall f \in Z_{m_0}, \forall x_1, x_2, y_1, y_2 \in [n, k]$. All the variables satisfy the above conditions and $f \in Z$. If $f \in Z_{m_0}$, we have

$$|f(x_1, y_1) - f(x_2, y_1)| < \frac{\varepsilon}{2} < \varepsilon$$

$$|f(x_1, y_1) - f(x_1, y_2)| < \frac{\varepsilon}{2} < \varepsilon$$

otherwise, $f \in \bigcup_{i=m_0+1}^{\infty} P_i$; suppose $f = \Upsilon_m(k), m > m_0, k \in V$,

$$|f(x_1, y_1) - f(x_2, y_1)| = |s(x_1, y_1) - k(x_1, y_1) + k(x_1, y_1) - k(x_2, y_1) + k(x_2, y_1) - s(x_2, y_1)|$$

$$\leq |\Upsilon_m(k)(x_1, y_1) - k(x_1, y_1)| + |k(x_1, y_1) - k(x_2, y_1)| + |\Upsilon_m(k)(x_2, y_1) - k(x_2, y_1)|$$

$$\leq 2|\Upsilon_m(k) - k|_c + |k(x_1, y_1) - k(x_2, y_1)| \leq 2 \times \frac{\varepsilon}{4} + \frac{\varepsilon}{2} = \varepsilon.$$

Similarly, $|f(x_1, y_1) - f(x_1, y_2)| \leq \varepsilon$. Z is uniformly bounded and successive, so it is pre-compact through the Arzelà–Ascoli theorem. Define $\{z_i\}_{i=1}^{\infty} \subset Z$, which is a sequence that converges to $z_0 \in c[n, k]$. If an m exists such that $\{z_i\} \subset Z_m$, we have $z_0 \in Z_m \in Z$. Otherwise, we have a subsequence $\{Y_{i_n}(v_{i_n})\}$ of $\{z_i\}$, such that $v_{i_n} \subset V$, when $n \to \infty, i_n \to \infty$, and we have

$$||v_{i_n} - z_0||_c = ||v_{i_n} - \Upsilon_{i_n}(v_{i_n}) + \Upsilon_{i_n}(v_{i_n}) - z_0||_c$$

$$\leq ||v_{i_n} - \Upsilon_{i_n}(v_{i_n})||_c + ||\Upsilon_{i_n}(v_{i_n}) - z_0||_c$$

$$\leq \kappa(i_n, V) + ||\Upsilon_{i_n}(v_{i_n}) - z_0||_c.$$

It means $z_0 = \lim_{n \to \infty} v_{i_n} \in V \subset Z$. Hence, Z is closed. □

Theorem 1. *$R(s), Y(c) \sim Q(0, k_l(x_1, x_2))$, using a piecewise linear interpolation of $R(s), Y(c)$ with m points will convergence with order $O(\frac{2}{(ml)^2})$.*

Proof. Suppose that $R(s), Y(c)$ are functions that sample in the Gaussian random field.

$$R(s) = \sqrt{2}(\pi)^{\frac{1}{4}}\left(\int_{\mathbb{R}^+} \sqrt{l}cos(ks)e^{-\frac{l^2k^2}{8}}dW(k) - \sqrt{l}sin(ks)e^{-\frac{l^2k^2}{8}}dB(k)\right)$$

$$Y(c) = \sqrt{2}(\pi)^{\frac{1}{4}}\left(\int_{\mathbb{R}^+} \sqrt{l}cos(kc)e^{-\frac{l^2k^2}{8}}dW(k) - \sqrt{l}sin(kc)e^{-\frac{l^2k^2}{8}}dB(k)\right)$$

W, B are independent standard Brownian motions [29]. Setting $T = lk$ and applying linear interpolations Π_1, Π_2 in $[s_i, s_{i+1}], [c_i, c_{i+1}]$:

$$H[(R(s) - \Pi_1 R(x))^2] = 2(\pi)^{\frac{1}{2}}[(\int((I - \Pi_1)cos(\frac{T}{l}s))^2 + \int((I - \Pi_1)sin(\frac{T}{l}s))^2)e^{-\frac{T^2}{l}}dT]$$

$$\leq (\pi)^{\frac{1}{2}}(S_{i+1} - S_i)^4 \int (\frac{T}{l})^4 e^{-\frac{T^2}{l}}dT = 24\pi(\frac{S_{i+1} - S_i}{l})^4.$$

Similarly, $H[(Y(c) - \Pi_1 Y(c))^2] \leq 24\pi(\frac{S_{i+1} - S_i}{l})^4$. By the linear interpolation error, we have:

$$|(I - \Pi_1)g(s)| = \frac{(b - a)^2|f''(\varepsilon_1)|}{2}$$

$$|(I-\Pi_2)g(s)| = \frac{(b-a)^2|f''(\varepsilon_2)|}{2}$$

Applying the Borel–Cantelli lemma, we have

$$|R(s)-\Pi_1 R(s)+Y(c)-\Pi_2 Y(c)| \le \frac{k[(S_{i+1}-S_i)^{2-\epsilon}+(C_{i+1}-C_i)^{2-\epsilon}]}{l^2}$$

k is a positive number of a Gaussian random variable with finite variance. Applying the piecewise linear interpolation of $R(s)$ and $Y(c)$ with m points, we can obtain convergence with order $O(\frac{2}{(ml)^2})$. □

Theorem 2. *Suppose that m is a positive integer, making* $c(k-n)(c_1+c_2)s(m,V)e^{c(k-n)} < \varepsilon$, *for* $d,e \in [n,k]$, *function* $f \in V$, *there exists* $W_1 \in \mathbb{R}^{n\times(m+1)}$, $b_1 \in \mathbb{R}^{(m+1)}$, $W_2 \in \mathbb{R}^{k\times n}$, *such that:*

$$||(\Psi)(d,e) - (W_2 \cdot \sigma(W_1 \cdot [f(x_1,y_1),\ldots,f(x_m,y_m)]^T + b_1) + b_2)||_2 < \varepsilon.$$

Proof. For $f \in V$, $f_m \in P_m$, by (5) and the Lipschitz condition, such that:

$$||(\Psi_f)(d,e) - (\Psi_{f_m})(d,e)||_2 \le c\iint |(\Psi_f)(x,y) - (\Psi_{f_m})(x,y)|dxdy$$

$$+c\iint |f(x,y) - f_m(x,y)|dxdy \le c\iint |(\Psi_f)(x,y) - (\Psi_{f_m})(x,y)|dxdy$$

$$+c(k-n)^2(c_1+c_2)s(m,V)$$

According to the Gronwall inequality, we obtain

$$||(\Psi_f)(d,e) - (\Psi_{f_m})(d,e)||_2 \le c(k-n)^2(c_1+c_2)s(m,V)e^{\iint 1dxdy}$$

$$\le c(k-n)^2(c_1+c_2)s(m,V)e^{c(k-n)^2}$$

Define $L_m = (f(x_1,y_1), f(x_1,y_2),\ldots,f(x_m,y_m)) \in \mathbb{R}^{m+1}$, $f \in V$, and L_m is a compact set. There is a mapping relationship between L_m and P_m. We define a vector-valued function on L_m, such that:

$$\rho(f(x_1,y_1), f(x_1,y_2),\ldots,f(x_m,y_m)) = (\Psi_{f_m})(d,e)$$

For any $\varepsilon > 0$, when $m \to \infty$, $c(k-n)^2(c_1+c_2)s(m,V)e^{c(k-n)^2} < \varepsilon$. Applying the universal approximation theorem of a neural network for high-dimensional functions:

$$||\rho(f(x_1,y_1), f(x_1,y_2),\ldots,f(x_m,y_m)) - (W_2 \cdot \sigma(W_1 \cdot [f(x_1,y_1),\ldots,f(x_m,y_m)]^T + b_1) + b_2)||_2$$

$$\le \varepsilon - c(k-n)^2(c_1+c_2)s(m,V)e^{c(k-n)^2})$$

Thus, we have:

$$||(\Psi_f)(d,e) - (W_2 \cdot \sigma(W_1 \cdot [f(x_1,y_1),\ldots,f(x_m,y_m)]^T$$

$$+b_1) + b_2)||_2 \le ||(\Psi_f)(d,e) - (\Psi_{f_m})(d,e)||_2 + ||(\Psi_f)(d,e)$$

$$-(W_2 \cdot \sigma(W_1 \cdot [f(x_1,y_1),\ldots,f(x_m,y_m)]^T + b_1) + b_2)||_2$$

$$\le c(k-n)^2(c_1+c_2)s(m,V)e^{c(k-n)^2}) + \varepsilon$$

$$-c(k-n)^2(c_1+c_2)s(m,V)e^{c(k-n)^2}) = \varepsilon$$

□

The number of discrete points only affects the smoothness of the PDE curve. The more discrete points, the smoother the curve. With the increase in discrete points, the mesh segmentation becomes finer. The amount of data required by the model will continue to increase, but it has no impact on the accuracy. We can always make the model achieve a better effect.

4. Experimental Results

In this section, we conduct experiments using the proposed method. For the data sets, we obtain functions and solutions through the GRF and the numerical solver of the PDEs. In the prediction aspect, we exhibit the prediction ability of the proposed method based on the two-dimensional Poisson and heat conduction equation. In addition, we analyze the experimental results by the losses and compare the predictions with numerical solutions. The loss function is the mean square error (MSE). As shown in the following,

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \widehat{Y}_i)^2$$

n is the number of data, $\widehat{Y}_i$ is the predicted value, and Y is the true value. This indicator can directly show the prediction effect.

4.1. Experimental Setup

The underlying framework for implementing DeepONet is TensorFlow. We train DeepONet using the Windows 10 operating system, with a 32 G memory, an NVIDIA GeForce RTX 3080 Ti graphics card, and Python as the programming language. In addition, the hyperparameters are set as follows. The learning rate is 0.001, the epoch is 20,000, the activation function is ReLU, and the optimizer is Adam. The width and depth of the branch and the trunk network are 40 and 2.

4.2. Poisson Equation

The two-dimensional Poisson equation is described as:

$$(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2}) = f(x,y), \quad x,y \in [0,1]$$

with zero boundary conditions and a fixed f. We aim to predict $s(x,y)$ over the domain $[0,1]$ for any $f(x,y)$. The objective is to learn an operator from function f to s. The selected function space is the GRF, the number of discrete points is 20×20, the number of training and test functions is 200/40, and the number of training data points of each function is 120. They are the solutions of the PDEs, which can be obtained by the five-point difference method, as shown in Figure 3.

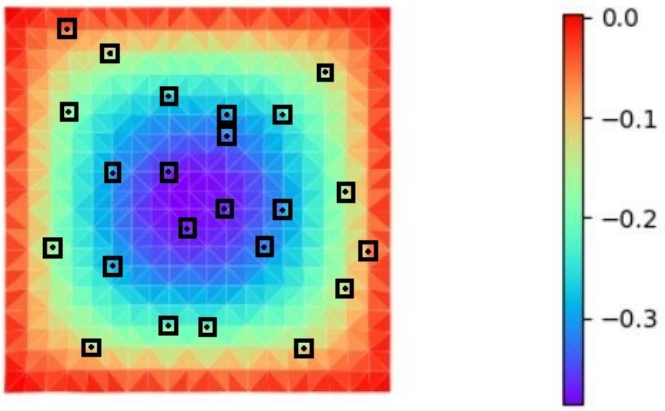


Figure 3. Training points in Poisson equation.

The training result is shown in Figure 4. The loss will decrease with the iterations' increase and fluctuate in a small range until stable. The best test loss is 8.04×10^{-6}. Next, we use the trained model to predict the PDE. We choose the input function $f = -5e^{\frac{[(x-\frac{1}{4})^2+(y-\frac{1}{4})^2]}{4}}$, and the numerical solution and the prediction of DeepONet are shown in Figures 3 and 5.

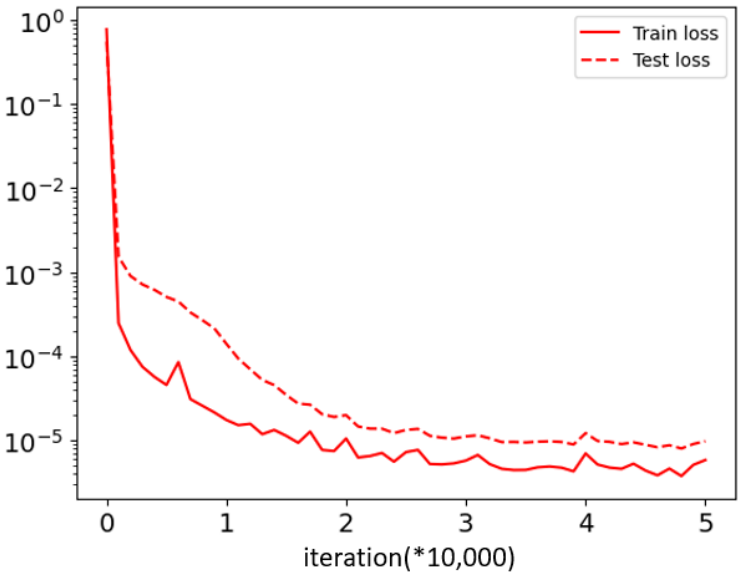


Figure 4. Training and test losses.

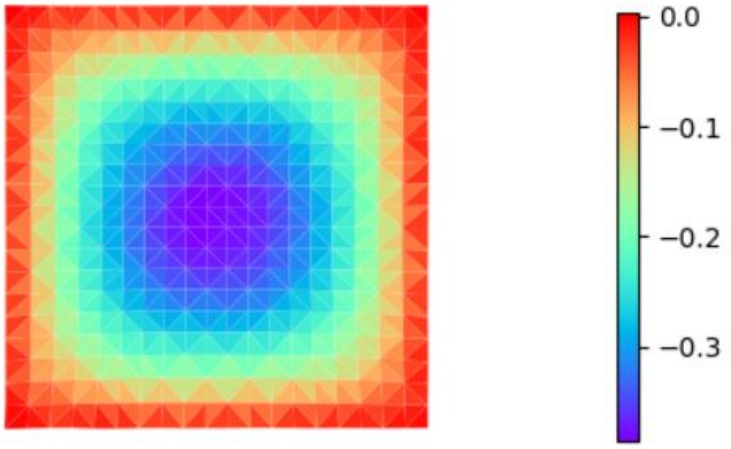


Figure 5. Prediction result.

Compared with Figure 3, the prediction result of DeepONet is almost the same as the numerical result. Moreover, the models also show good generalization. When the input functions are $f_1 = xsin(\pi x) + ysin(\pi y)$ and $f_2 = -xsin(\pi x) - ycos(\pi y)$, the numerical solution and the prediction are shown in Figure 6. The prediction effects are excellent when the functions are in the same function space.

Figure 6a,b are the DeepONet prediction results, and Figure 6c,d are the PDE numerical solutions. The results indicate that DeepONet can accurately predict the two-dimensional Poisson equation. Compared with the traditional numerical solution methods, the DeepONet framework is flexible because it is a pre-trained model. As long as the corresponding input functions are set, DeepONet can predict this kind of PDE.

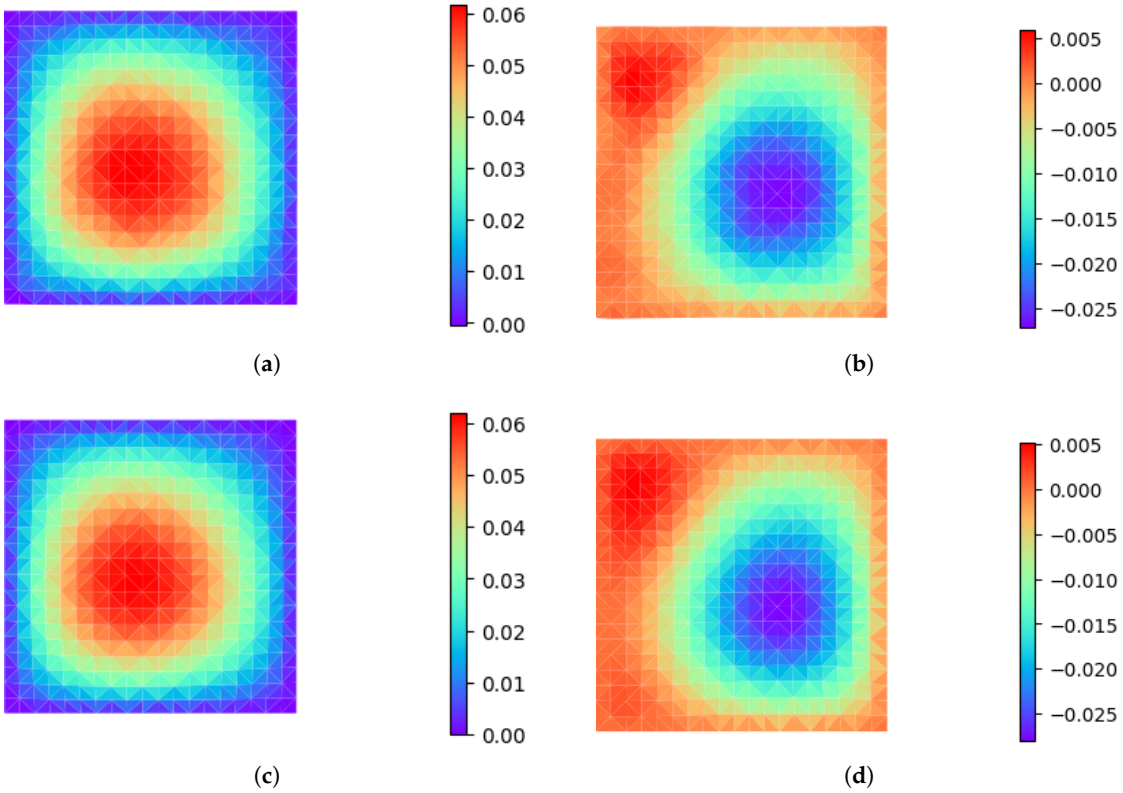


Figure 6. The predictions and numerical solutions. (**a**,**b**) represent the DeepONet prediction of s_1, s_2, and (**c**,**d**) represent the true value of s_1, s_2.

4.3. Heat Conduction Equation

The two-dimensional heat conduction PDE with the source term $f(x, y)$ is described by

$$\frac{\partial s}{\partial t} = a^2\left(\frac{\partial^2 s}{\partial x^2} + \frac{\partial^2 s}{\partial y^2}\right) + f(x, y), \quad x, y, t \in [0, 1]$$

with a zero boundary condition, and the initial condition is

$$s_0 = e^{-10[(x-\frac{1}{2})^2+(y-\frac{1}{2})^2]}$$

where $a = 1$ is a constant. We will train DeepONet to learn the implicit operator that is a mapping from $f(x, y)$ to $s(x, y, t)$. According to the dimension reduction algorithm of the input function and the model training method under a multivariable, we show six different experimental results for DeepONet. These experiments include the number of input functions, the number of training points, increasing discrete points, different function spaces, different training methods, and different models. We analyze the experimental results by comparing the prediction and numerical solution. The purpose is to find the best model and accurately predict the two-dimensional heat conduction equation.

4.3.1. Number of Input Functions

The data set size is expressed as the product of the input function and the training data points. One of the factors affecting it is the number of input functions. To maintain the unity of other variables, we set them in Table 1.

Table 1. The number of input functions is different, and other parameters are unchanged.

Case	Training Data Points	Functions	Discrete Points	Space	MSE
1	180	50/10	20×20	GRF	7.64×10^{-4}
2	180	60/12	20×20	GRF	4.43×10^{-4}
3	180	80/16	20×20	GRF	2.87×10^{-4}
4	180	110/22	20×20	GRF	1.59×10^{-4}
5	180	150/30	20×20	GRF	3.28×10^{-5}
6	180	200/40	20×20	GRF	2.53×10^{-5}
7	180	300/60	20×20	GRF	1.46×10^{-5}

To control the variables, we only change the number of functions. After obtaining the results through the numerical solver, 180 data points are randomly selected. These points are part of the numerical solutions. Each function randomly selects data points at different locations, as shown in Figure 7.

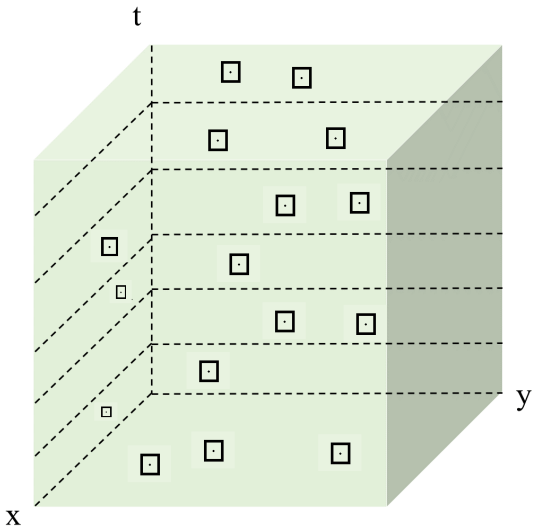


Figure 7. Training points in heat conduction equation.

For the different numbers of functions, the training results are different. The losses of the models are shown in Figure 8.

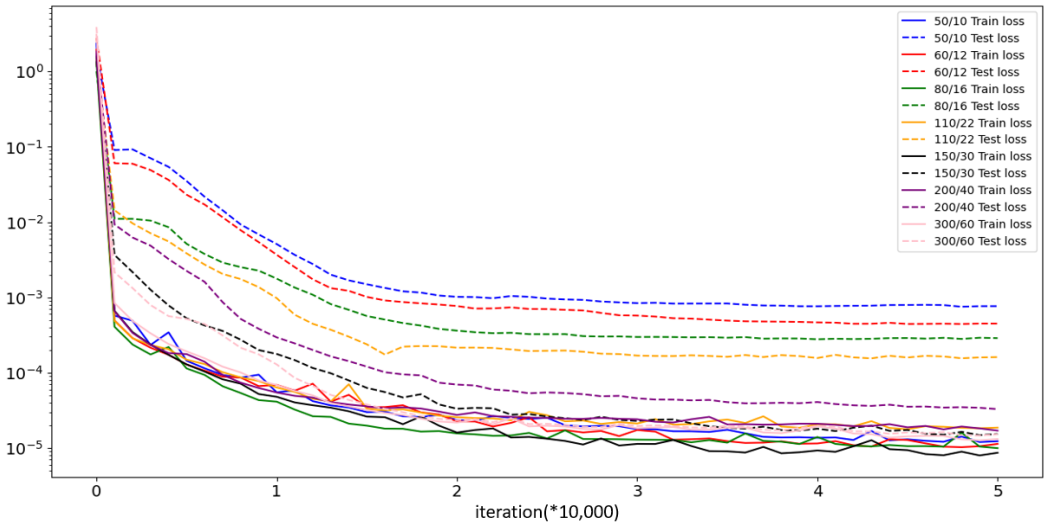


Figure 8. Model training and test losses in different input functions.

The dotted lines represent the test loss, and the solid lines represent the training loss. The different colors represent the model training status corresponding to the number of input functions. When the number of input functions is small, the model will be underfitted, and the test loss is far greater than the training loss. With the increase in the number of input functions, the gap between the test and training loss becomes smaller and smaller, and the test loss of the model becomes lower until the number of functions reaches convergence at 300/60. At this time, the optimal loss of the trained model is 1.46×10^{-5}. Generally, we evaluate the model generalization ability on the test set. When the test loss is low, the model generalization ability is strong. Therefore, the number of functions we choose is 300/60.

We set function $f(x, y) = e^{-\frac{[(x-\frac{1}{4})^2+(y-\frac{1}{4})^2]}{4}}$, and DeepONet predicts the best results, which are shown in Figure 9. Figure 9a–f show the solution of the two-dimensional heat conduction equation predicted by DeepONet and the process of the PDE changing with time. For comparison, we show the corresponding numerical solution in Figure 10.

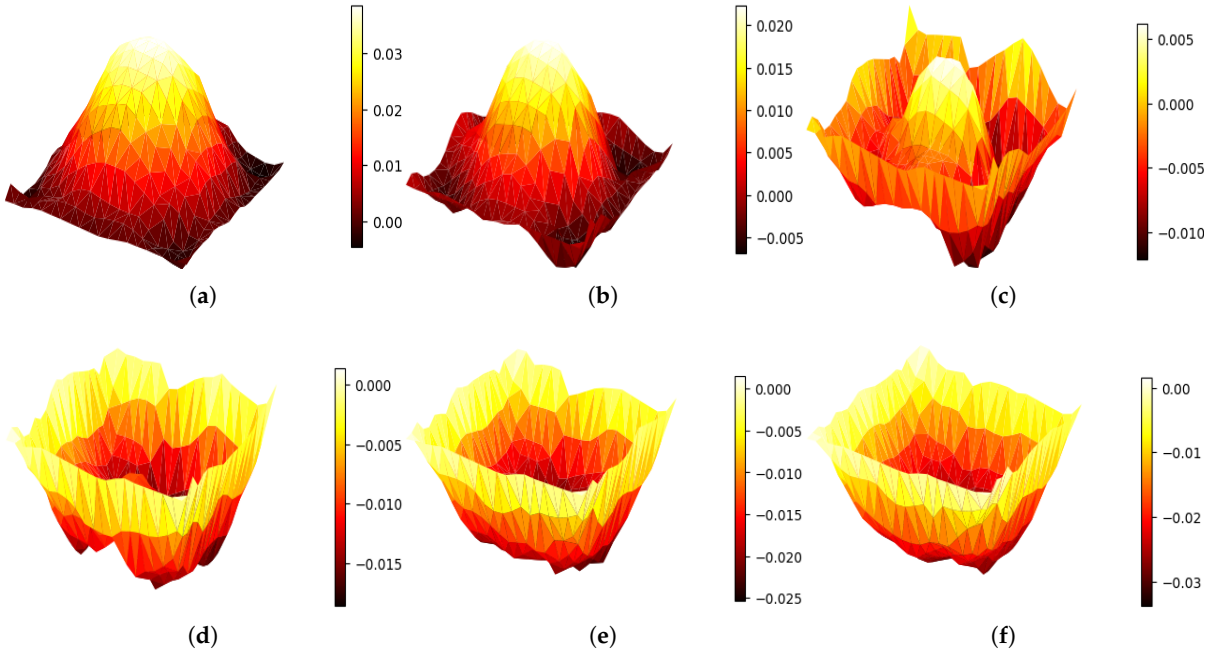


Figure 9. Prediction results for six time periods. (**a**–**f**) represent the prediction results from t_1 to t_6 time periods, respectively

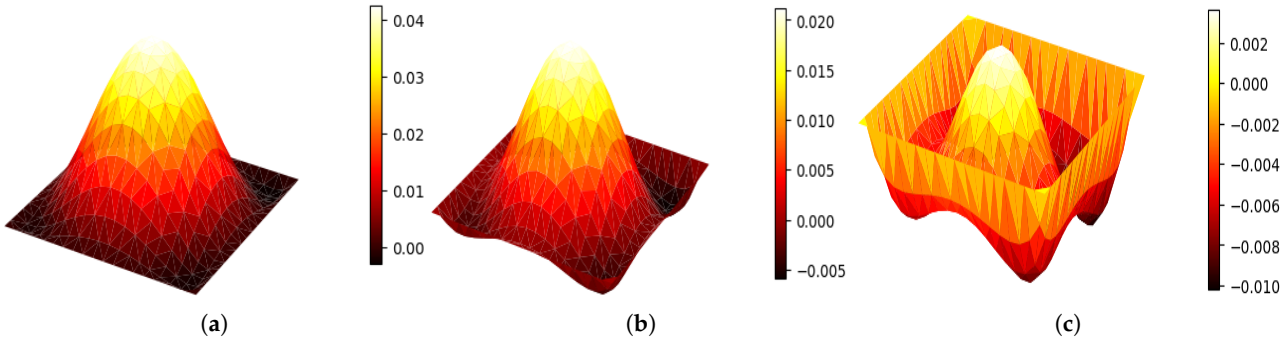
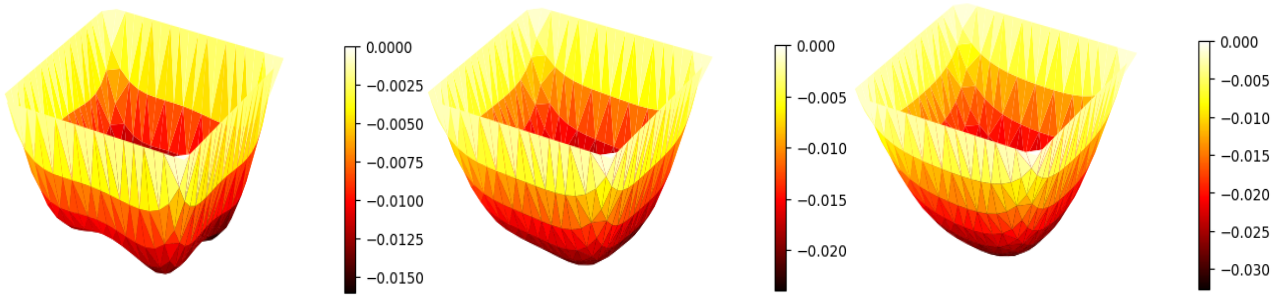


Figure 10. *Cont.*



(**d**) (**e**) (**f**)

Figure 10. Numerical results for six time periods. (**a**–**f**) represent the numerical results from t_1 to t_6 time periods, respectively

For seven cases, the MSE is shown in Table 1. It shows that the larger the number of functions, the smaller the MSE, and the better the model training, but the training time is longer and longer. When the number of functions reaches 200/40, the model loss is $\sim 10^{-5}$ and does not decrease. At this time, the model has reached convergence.

4.3.2. Number of Training Data Points

Training data points are a significant part of the data sets, and their number also determines the size of the data sets. To improve the model training efficiency, we will find appropriate data points. We randomly select 20, 40, 70, 110, 160, 220, and 300 training data points in each corresponding input function. Similarly, we designed Table 2 to show the control variables.

According to the number of input functions, periods, and training data points, we can calculate the training times of each data point in the grid. When choosing 40 as the number of training data points, the training time of each data point in the grid is $200 \times 40/400 \times 6$, which is about 3. The specific experimental results are shown in Figure 11. It shows the model loss under different data points, and the colors represent the distinction of the training data points. With the increase in the iterations, the loss change also shows the same trend. When the number of training data points is less than 110, there exists a big gap.

Table 2. The number of training data points is different, and the other parameters are unchanged.

Periods	Training Data Points	Functions	Discrete Points	Space	MSE
6	20	200/40	20×20	GRF	1.17×10^{-3}
6	40	200/40	20×20	GRF	1.24×10^{-4}
6	70	200/40	20×20	GRF	5.36×10^{-5}
6	110	200/40	20×20	GRF	2.43×10^{-5}
6	160	200/40	20×20	GRF	1.81×10^{-5}
6	220	200/40	20×20	GRF	1.59×10^{-5}
6	300	200/40	20×20	GRF	1.84×10^{-5}

When the number of training data points is more than 110, the test loss gradually reaches the fitting state. It is 10^{-5}. As more and more training data points are selected, the test loss is gradually consistent with the training loss. Especially, the number of iterations exceeds 20,000. When the number of training data points is 220, the test loss is almost the same as the training loss, and the model generalization is the strongest. The best test loss in the different training data points is shown in Table 2.

In Table 2, the test losses become smaller with the increase in the training points. After 70 training points, the best test loss remains at $\sim 10^{-5}$. We select around 220 training points, the 20×20 grid size, and the $200 \times 220/400 \times 6 \approx 18$ training times. The model training effect is best when each point is trained about 18 times.

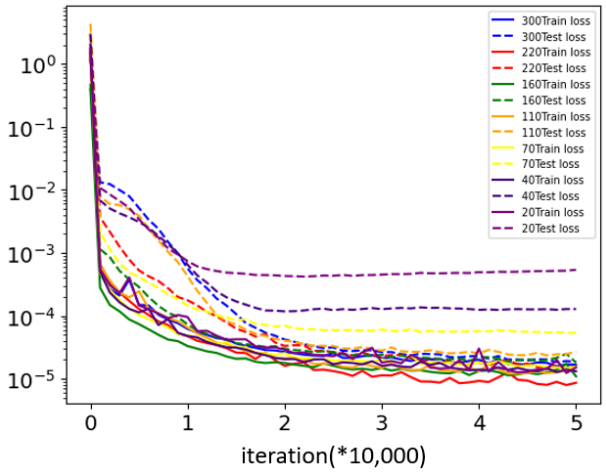


Figure 11. Model training and test losses in different training data points.

4.3.3. Different Function Spaces

In the previous parts, we mentioned the method of the sampling function. Firstly, we randomly sample univariate functions in the GRF and then a random linear combination between two univariate functions. However, according to the work of [21], the classical function space can also select a Chebyshev polynomial. Therefore, in this part, we also consider sampling the function in this space as the data sets and training the model. According to the comparison of the GRF, we explore which function space is more suitable for sampling two-dimensional functions.

The numbers of the functions and data points are set to 200 and 220. The input function from the Chebyshev polynomial space is as follows: let $k > 0$, and F_i are Chebyshev polynomials of the first kind. We define the orthogonal polynomials of degree N as:

$$V_p = \sum_{i=0}^{N-1} a_i F_i(x) : |a_i| \leqslant k$$

We obtain the input functions from V_p by randomly sampling a_i from $[-k, k]$. The model training loss corresponding to the two function spaces is shown in Figure 12. When the sampling space comes from the GRF or Chebyshev polynomials, the blue and red lines in Figure 12 represent the corresponding training and test losses. By comparing the loss changes in the model, we find that the model loss represented by red is slightly greater than that of blue. The experimental results show that under the state of the other same parameters, the generalization error of the model training is small in the GRF.

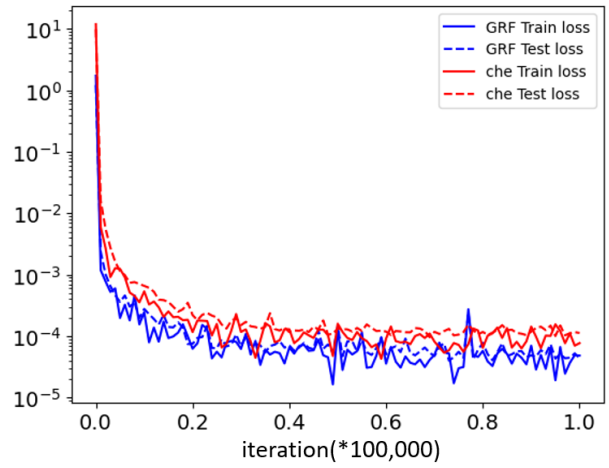


Figure 12. Model training and test losses in different function spaces.

The prediction results with the same f are shown in Figure 13. It shows that when the function space is Chebyshev polynomials, DeepONet cannot predict the solutions well. Although the generalization abilities of the model are poor in this state, they can improve in the GRF.

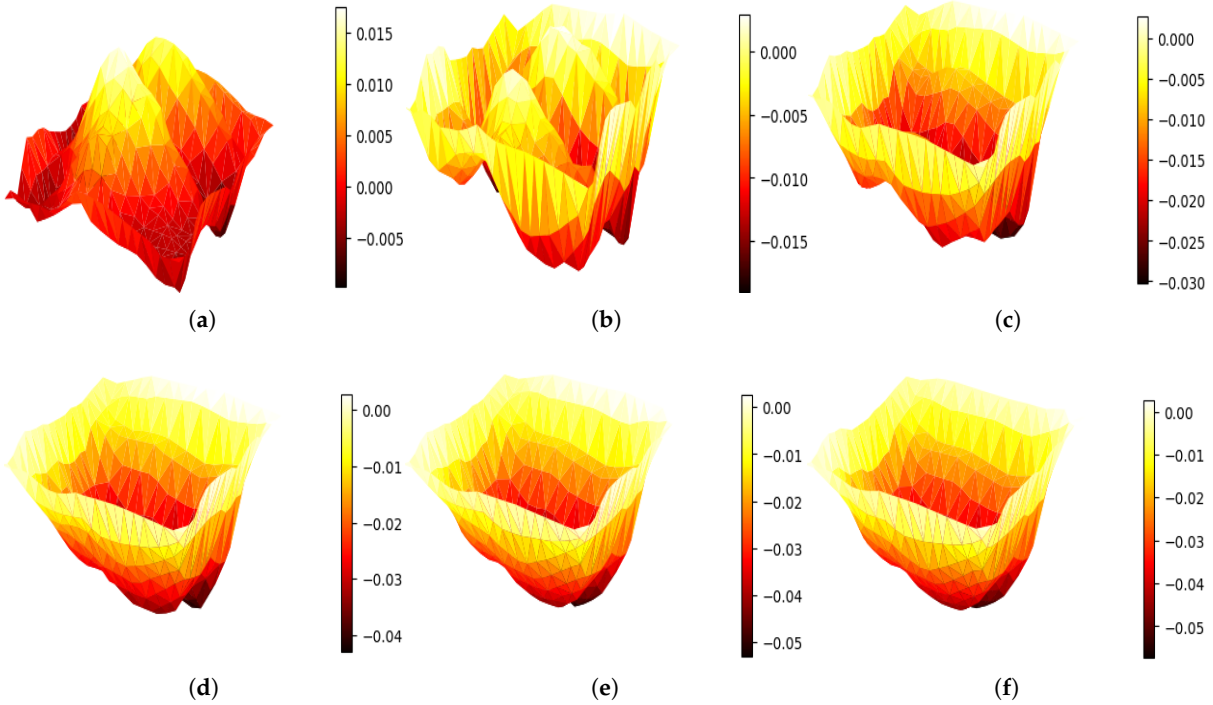
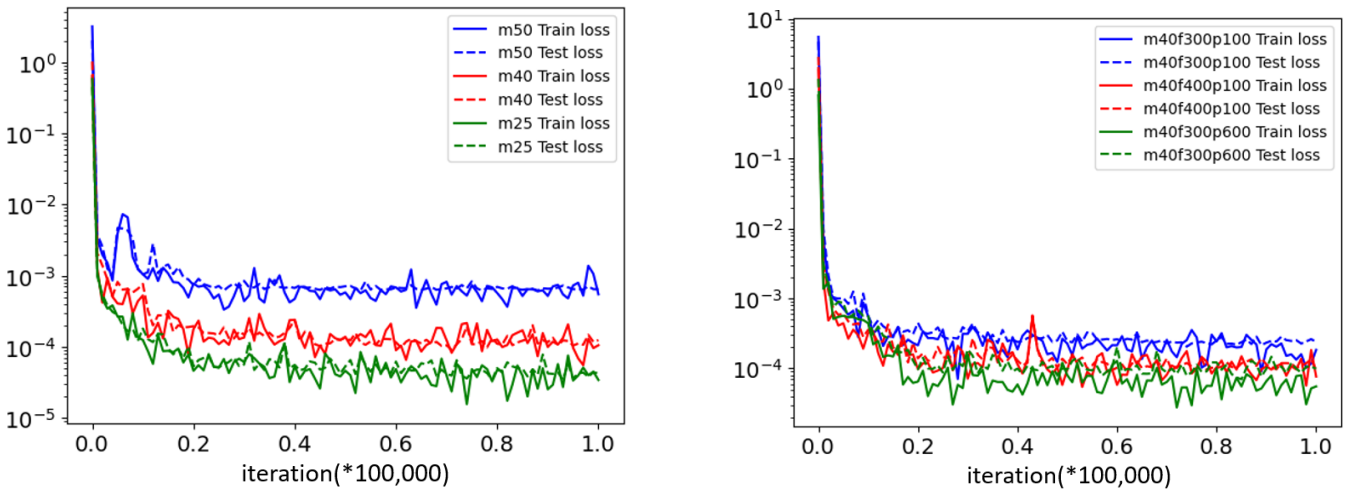


Figure 13. Prediction results when function space is Chebyshev polynomials. (**a**–**f**) represent the prediction results from t_1 to t_6 time periods, respectively

4.3.4. Discrete Points

According to the theoretical analysis in part 3, the more discrete points, the more able to represent the input function. We send the discrete input function as a branch network input to DeepONet. As long as the loss of the model is less than ε, DeepONet can accurately predict the solution of the two-dimensional partial differential equation. However, when the number of m gains, the mesh and function image become denser and smoother. If the data set of the same size is maintained, the larger m is, there will be fewer training times of each data point, which may lead to a bad result for the model training, as shown in Figure 14a.

Blue, red, and green are the changes in the model training loss when m = 50, 45, and 25. When the same data set is maintained, the model loss will increase to the expansion of several m. However, if the number of m increases, we can improve the number of training data points or input functions (expand the training times of each data point) to reduce the model loss. The results are shown in Figure 14b. While keeping the other parameters unchanged and increasing the training data points, the green line in Figure 14b shows the loss change during the model training. The training and test losses decrease to the best state with the iterations' increase. Then, it tends to be stable. When increasing the number of input functions, the loss change is shown by the red line in Figure 14b. The test and training losses keep a similar trend, and the optimal test loss is smaller than the blue line. Therefore, we can increase the number of input functions (expand the data set) while increasing the training data points, and it can ensure that the model training is better, as shown in Figure 15.



(**a**) (**b**)

Figure 14. Model training and test losses. The loss will be reduced if the input functions and training points are increased.

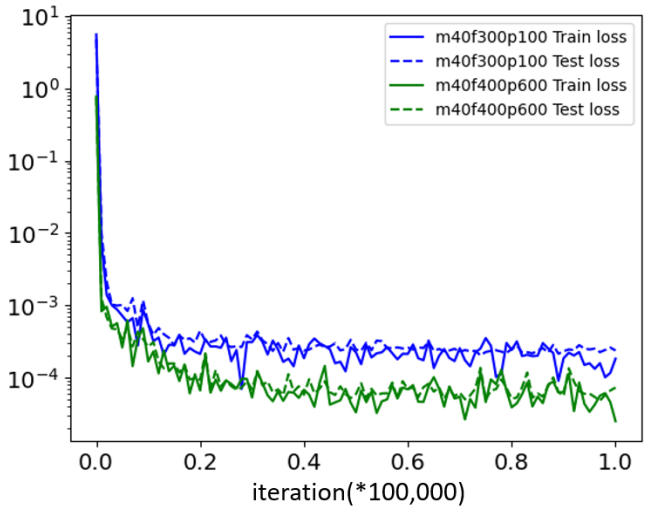


Figure 15. Model training and test losses after enlarging data set.

The blue line indicates the change in the loss when the discrete points increase to 40×40. The green line represents the change in the model loss after the data set increases. In comparison, the best loss of the blue lines is greater than that of the green. According to the records in Table 3, for different discrete points, the optimal loss of the model will eventually reach 10^{-5}. It shows that when the discrete points gradually increase, the model will always ensure that the optimal MSE is small enough to make the prediction results accurately.

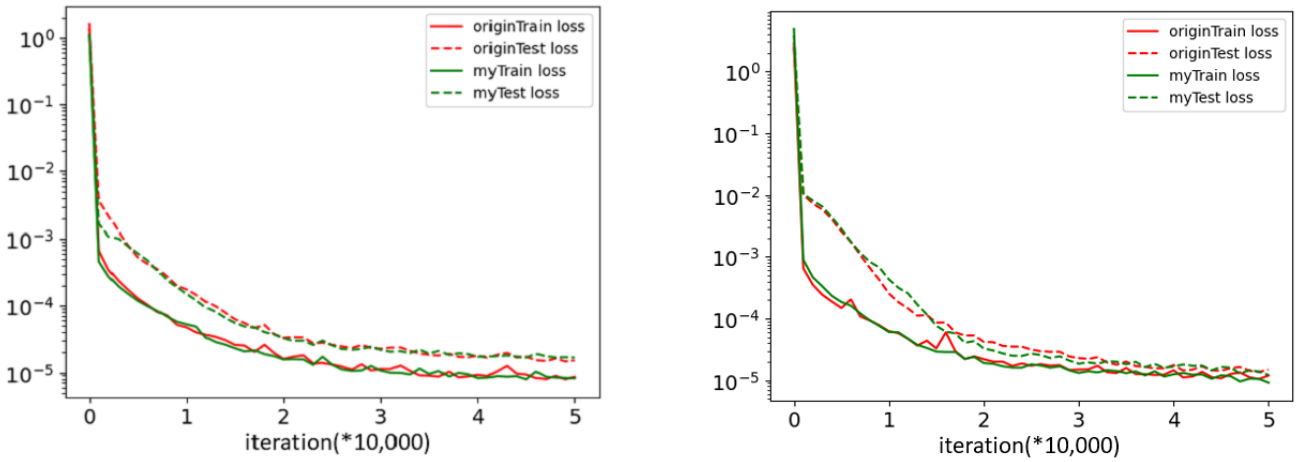
Table 3. The MSE of different discrete points.

m	Data Points	Functions	MSE
20	220	200/40	1.48×10^{-5}
25	320	300/60	1.61×10^{-5}
40	600	400/80	5.79×10^{-5}
50	600	400/80	6.76×10^{-5}

4.3.5. Different Training Method

In part two, we give two training methods. The first is to extend the training method to a two-dimensional PDE according to the work of [21] in Algorithm 2. The second is to divide the period into n equal parts into blocks and randomly select training points in each block for training in Algorithm 3. In this part, we reproduce the two training methods and illustrate that these methods are effective according to the change in the model training loss. We obtain the solution of the PDE corresponding to six time periods and use these two methods to train the model. The results are shown in Figure 16a.

In Figure 16a, red and green represent the change in the model training loss corresponding to the origin and our algorithms. From the figure, we can see that no matter the training or test loss, they almost maintain the same change trajectory, and the loss gradually decreases and tends to fit with the increase in the iterations. Finally, the best test loss remains at 10^{-5}. However, we speculate that the reason may be that there are too few periods to see the advantages. Therefore, we will select ten discrete-time variable training periods, and the change in the model loss is shown in Figure 16b. The experimental results show that the two training algorithms can train the model well. Both can minimize the loss of the model.



(**a**) Six time periods (**b**) Ten time periods

Figure 16. Model training and test losses in different periods.

4.3.6. Different Models

By the work of [21], we considered two models, FNN and Residual Networks (ResNet), because the input has no specific structure. In this part, we still regard these two models as comparative models. We apply the improved data set to train the two models, and the experimental results are shown in Figure 17.

The red lines indicate the training result of FNN. With the increase in iterations, the training loss decreases continuously. However, the test loss will not change when it reaches a certain extent. The green lines represent the training results of ResNet. We can see that the training and test loss changes are very tortuous and unstable. The training results of the two models are much worse than DeepONet.

The best training error of DeepONet is 1.28×10^{-5}, and the best test error is 3.56×10^{-5}. The best training error of FNN is 6.29×10^{-5}, and the best test error is 1.27×10^{-4}. The best training error of ResNet is 8.98×10^{-5}, and the test error is 3.01×10^{-4}. Through the training of FNN and ResNet, we found that whether the FNN or ResNet model are used, their training results are worse than DeepONet.

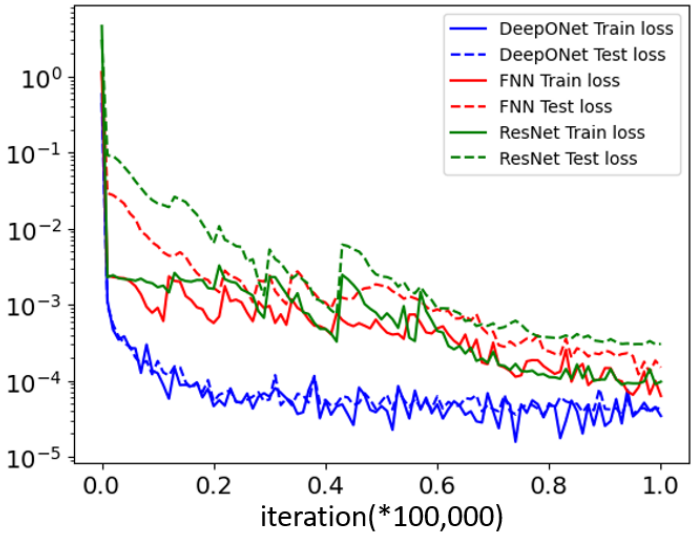


Figure 17. Compared with FNN and ResNet.

In addition, DeepM&Met indicates that a two-dimensional PDE can also be predicted by using a fully connected network and the pre-trained DeepONet. This method mainly solves multi-physical and multiscale problems, but adding an FNN is redundant for a two-dimensional PDE with simple prediction. In addition, the model complexity and prediction time will increase when the branch network is replaced by a convolution neural network. As shown in Figure 18, the prediction accuracy of the method using a CNN to replace the branch network is consistent with our proposed method. However, in terms of the training time and prediction time, our method shows absolute advantages, as shown in Table 4.

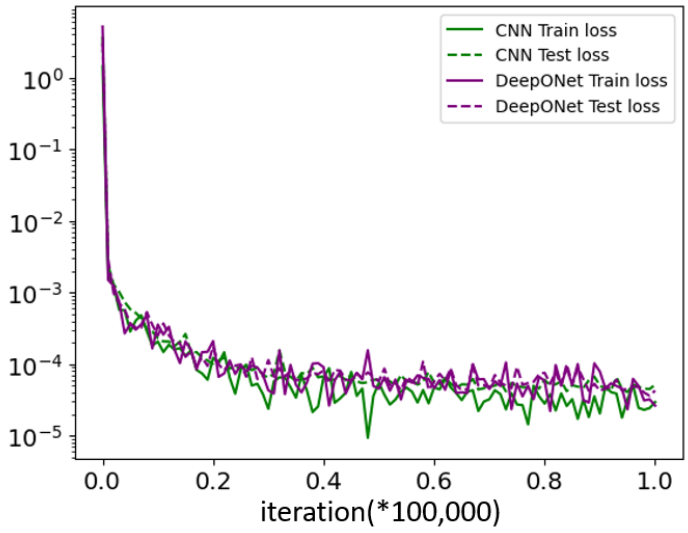


Figure 18. Compared with CNN.

Table 4. Comparison with CNN.

Metrics	Our Method	CNN
Loss	5.34×10^{-5}	5.63×10^{-5}
Training time	75 s	348 s
Prediction time	0.009 s	0.03 s

The above experiments indicate that our proposed method has a strong operator learning ability. When dealing with a two-dimensional PDE, it can predict the corresponding solution by learning the implicit operator. We also analyze various factors affecting the training of the DeepONet model and select the optimal parameters. Although the prediction accuracy is lower than that of the numerical solver and close to the methods based on a CNN, our proposed method saves time and labor in solving two-dimensional PDEs.

5. Conclusions

This paper presents an efficient method based on DeepONet for solving two-dimensional PDEs and takes the two-dimensional Poisson and heat conduction equation as research problems. Moreover, we theoretically analyze the influence of the number of discrete points corresponding to the two-dimensional functions on the model accuracy and verify it through experiments. In addition, many experimental results show that the proposed method can accurately predict two-dimensional PDEs. Compared with the numerical solution, the mean square error of the prediction solution can reach 10^{-5}. Compared with other methods, it is efficient. However, our approach considers many simple nonlinear input functions and ignores some complex function forms. In the future, we would like to improve the algorithm and expand the model to solve more complex partial differential equations.

Author Contributions: Writing—original draft preparation, Y.W.; writing—review and editing, X.Z. and X.P.; funding acquisition, C.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This study was supported by the "Chunhui Plan" Cooperative Research for the Ministry of Education (Grant No. HZKY20220407), the Natural Science Foundation of Liaoning Province (Grant No. LJKZ0136), the JSPS KAKENHI (Grant No. JP22K17884, JP23KJ1005), National Key Research and Development Plan project (Grant No. 2022YFE011400), and the Xingliao Talents Program of Liaoning Province (Grant No. XLYC2203046).

Data Availability Statement: The data are unavailable due to privacy.

Acknowledgments: The authors are thankful to the anonymous reviewers and editors for their valuable comments and suggestions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Øksendal, B. Stochastic partial differential equations and applications to hydrodynamics. In *Stochastic Analysis and Applications in Physics*; Springer: Berlin/Heidelberg, Germany, 1994; pp. 283–305.
2. Constantin, P. *Analysis of Hydrodynamic Models*; SIAM: Philadelphia, PA, USA , 2017.
3. Lenhart, S.; Xiong, J.; Yong, J. Optimal controls for stochastic partial differential equations with an application in population modeling. *SIAM J. Control Optim.* **2016**, *54*, 495–535. [CrossRef]
4. Thonhofer, E.; Palau, T.; Kuhn, A.; Jakubek, S.; Kozek, M. Macroscopic traffic model for large scale urban traffic network design. *Simul. Model. Pract. Theory* **2018**, *80*, 32–49. [CrossRef]
5. Boureghda, A. A modified variable time step method for solving ice melting problem. *J. Differ. Equations Appl.* **2012**, *18*, 1443–1455. [CrossRef]
6. Rao, S.S. *The Finite Element Method in Engineering*; Butterworth-heinemann: Oxford, UK, 2017.
7. Shen, J.; Tang, T.; Wang, L.L. *Spectral Methods: Algorithms, Analysis and Applications*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2011; Volume 41.
8. Aliabadi, F.M. Boundary element methods. In *Encyclopedia of Continuum Mechanics*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 182–193.
9. Moukalled, F.; Mangani, L.; Darwish, M.; Moukalled, F.; Mangani, L.; Darwish, M. *The Finite Volume Method*; Springer: Berlin/Heidelberg, Germany, 2016.
10. Rubinstein, R.Y.; Kroese, D.P. *Simulation and the Monte Carlo Method*; John Wiley & Sons: Hoboken, NJ, USA, 2016.
11. Sharma, D.; Goyal, K.; Singla, R.K. A curvelet method for numerical solution of partial differential equations. *Appl. Numer. Math.* **2020**, *148*, 28–44. [CrossRef]
12. Liu, Z.; Xu, Q. A Multiscale RBF Collocation Method for the Numerical Solution of Partial Differential Equations. *Mathematics* **2019**, *7*, 964. [CrossRef]

13. Aziz, I.; Amin, R. Numerical solution of a class of delay differential and delay partial differential equations via Haar wavelet. *Appl. Math. Model.* **2016**, *40*, 10286–10299. [CrossRef]
14. Fahim, A.; Araghi, M.A.F.; Rashidinia, J.; Jalalvand, M. Numerical solution of Volterra partial integro-differential equations based on sinc-collocation method. *Adv. Differ. Equations* **2017**, *2017*, 362. [CrossRef]
15. Sirignano, J.; Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **2018**, *375*, 1339–1364. [CrossRef]
16. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **2019**, *378*, 686–707. [CrossRef]
17. Fang, Z. A high-efficient hybrid physics-informed neural networks based on convolutional neural network. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 5514–5526. [CrossRef] [PubMed]
18. Wang, K.; Chen, Y.; Mehana, M.; Lubbers, N.; Bennett, K.C.; Kang, Q.; Viswanathan, H.S.; Germann, T.C. A physics-informed and hierarchically regularized data-driven model for predicting fluid flow through porous media. *J. Comput. Phys.* **2021**, *443*, 110526. [CrossRef]
19. Cai, S.; Wang, Z.; Wang, S.; Perdikaris, P.; Karniadakis, G.E. Physics-informed neural networks for heat transfer problems. *J. Heat Transf.* **2021**, *143*, 060801. [CrossRef]
20. Chen, T.; Chen, H. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Trans. Neural Netw.* **1995**, *6*, 911–917. [CrossRef]
21. Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; Karniadakis, G.E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* **2021**, *3*, 218–229. [CrossRef]
22. Deng, B.; Shin, Y.; Lu, L.; Zhang, Z.; Karniadakis, G.E. Convergence rate of DeepONets for learning operators arising from advection-diffusion equations. *arXiv* **2021**, arXiv:2102.10621.
23. Lin, C.; Li, Z.; Lu, L.; Cai, S.; Maxey, M.; Karniadakis, G.E. Operator learning for predicting multiscale bubble growth dynamics. *J. Chem. Phys.* **2021**, *154*, 104118. [CrossRef]
24. Di Leoni, P.C.; Lu, L.; Meneveau, C.; Karniadakis, G.; Zaki, T.A. Deeponet prediction of linear instability waves in high-speed boundary layers. *arXiv* **2021**, arXiv:2105.08697.
25. Cai, S.; Wang, Z.; Lu, L.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *J. Comput. Phys.* **2021**, *436*, 110296.
26. Mao, Z.; Lu, L.; Marxen, O.; Zaki, T.A.; Karniadakis, G.E. DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. *J. Comput. Phys.* **2021**, *447*, 110698.
27. Lu, L.; Meng, X.; Cai, S.; Mao, Z.; Goswami, S.; Zhang, Z.; Karniadakis, G.E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Comput. Methods Appl. Mech. Eng.* **2022**, *393*, 114778. [CrossRef]
28. Li, Z.; Liu, F.; Yang, W.; Peng, S.; Zhou, J. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *33*, 6999–7019. [CrossRef]
29. Zhang, Z.; Karniadakis, G.E. *Numerical Methods for Stochastic Partial Differential Equations with White Noise*; Springer: Berlin/Heidelberg, Germany, 2017.