fluids

Article

Fourier Neural Operator Networks for Solving Reaction–Diffusion Equations

Yaobin Hao and Fangying Song *

School of Mathmatics and Statistics, Fuzhou University, Fuzhou 350108, China; 220320010@fzu.edu.cn
* Correspondence: fangying_song@fzu.edu.cn

Abstract: In this paper, we used Fourier Neural Operator (FNO) networks to solve reaction–diffusion equations. The FNO is a novel framework designed to solve partial differential equations by learning mappings between infinite-dimensional functional spaces. We applied the FNO to the Surface Quasi-Geostrophic (SQG) equation, and we tested the model with two significantly different initial conditions: Vortex Initial Conditions and Sinusoidal Initial Conditions. Furthermore, we explored the generalization ability of the model by evaluating its performance when trained on Vortex Initial Conditions and applied to Sinusoidal Initial Conditions. Additionally, we investigated the modes (frequency parameters) used during training, analyzing their impact on the experimental results, and we determined the most suitable modes for this study. Next, we conducted experiments on the number of convolutional layers. The results showed that the performance of the models did not differ significantly when using two, three, or four layers, with the performance of two or three layers even slightly surpassing that of four layers. However, as the number of layers increased to five, the performance improved significantly. Beyond 10 layers, overfitting became evident. Based on these observations, we selected the optimal number of layers to ensure the best model performance. Given the autoregressive nature of the FNO, we also applied it to solve the Gray–Scott (GS) model, analyzing the impact of different input time steps on the performance of the model during recursive solving. The results indicated that the FNO requires sufficient information to capture the long-term evolution of the equations. However, compared to traditional methods, the FNO offers a significant advantage by requiring almost no additional computation time when predicting with new initial conditions.

Keywords: Fourier Neural Operator (FNO); Surface Quasi-Geostrophic (SQG) equation; Gray–Scott equation; multi-layer convolution

check for updates

Citation: Hao, Y.; Song, F. Fourier Neural Operator Networks for Solving Reaction–Diffusion Equations. *Fluids* **2024**, *9*, 258. https://doi.org/10.3390/fluids9110258

Academic Editors: Giuliano De Stefano, Yonghua Yan and Yong Yang

Received: 10 September 2024
Revised: 15 October 2024
Accepted: 4 November 2024
Published: 6 November 2024

1. Introduction

Reaction–diffusion equations are complex nonlinear partial differential equations, and they are typically solved using numerical methods such as the spectral method, the finite difference method, and the finite volume method [1,2]. It is well known that numerically simulating such complex partial differential equations is computationally expensive. These methods typically require high-resolution grids to capture detailed features, leading to high computational and storage demands. Additionally, there are limitations, in terms of generalization capability. When replacing the initial conditions with new ones, it is necessary to re-solve the simulation.

In recent years, deep learning has shown tremendous potential for solving complex nonlinear problems. The success of neural networks in areas such as image recognition and natural language processing has prompted researchers to explore their applications in scientific computing [3–6]. In addressing Partial Differential Equations (PDEs), many successful deep learning methods have emerged, which can be broadly classified into three categories: image-to-image regression, Physics-Informed Neural Networks (PINNs), and operator learning.

Convolutional Neural Networks (CNNs), as a representative image regression method, are widely used in image processing and computer vision fields [7,8], automatically extracting features and performing classification through local receptive fields and weight sharing. CNNs have also been used in solving PDEs [9–11], leveraging their ability to learn complex patterns and representations from data. However, when applied to solving partial differential equation problems, they still face challenges related to data requirements and generalization capabilities. Physics-Informed Neural Networks (PINNs) incorporate physical constraints directly into the training process of the neural network [12–15], enabling it to not only fit the training data but also adhere to the physical laws governing the PDEs being solved. Due to their ability to avoid complex mesh generation and handle irregular domains and intricate boundary conditions, PINNs have broad applications in structural mechanics [16], biomedical science [17], materials science [18], and meteorology [19]. In addition to traditional PINNs, physics–data combined machine learning methods have emerged as a powerful approach, particularly for reduced-order modeling of nonlinear dynamical systems in small-data regimes. These methods combine physics-based models with machine learning techniques to leverage both limited observational data and the underlying governing laws, thus improving generalization and accuracy in scenarios where data are sparse. For instance, Meng et al. [20] proposed a composite neural network framework that combines physics and data to solve reduced-order models of fluid dynamics effectively. Similarly, Lee and Carlberg [21] developed a physics-constrained autoencoder for nonlinear reduced-order modeling in mechanical systems, demonstrating the potential of these approaches in handling high-dimensional and highly nonlinear problems under data limitations. Additionally, NSFnets [22] have been successfully applied to solve incompressible Navier–Stokes equations, showcasing the strength of PINNs in fluid dynamics. However, PINNs also face the same issue as traditional numerical methods, namely, the need to retrain the model for each new instance.

Operator learning methods aim to learn mappings from an input function space to an output function space [23–25]. Unlike traditional numerical methods that rely on explicit mathematical models, operator learning involves learning the mapping between inputs and outputs to automatically discover and represent operators or transformations, thereby enabling the handling of complex systems and dynamic processes. Among the most mature operator learning methods are DeepONet and the Fourier Neural Operator (FNO). DeepONet, proposed by Lu et al. [24], has been widely applied to various problems and has demonstrated strong performance in different scientific [26] and engineering domains [27]. The Fourier Neural Operator (FNO) is a method proposed by Li et al. [25]. It is designed by approximating Green's function governed by differential equations. The FNO utilizes the Fourier transform to handle integral terms, which not only accelerates computations but also transforms the original data into the frequency domain. This allows for the efficient processing of high-dimensional data and the capturing of global properties. In previous studies, the FNO has demonstrated outstanding performance in handling Darcy flow with zero boundary conditions, Burgers' equation, and the Navier–Stokes equation with periodic boundary conditions [25]. In addition, an improved model based on the FNO, the U-FNO [28], has been proposed, to solve the problem of water flow through rock pores in multiphase flow simulations. The U-FNO model can achieve prediction accuracy comparable to traditional Convolutional Neural Networks (CNNs) with less training data, while also exhibiting higher computational efficiency and generalization capability.

The goal of this study was to apply Fourier neural operators to solve the Gray–Scott and SQG models, demonstrating their effectiveness in capturing the complex dynamics of these systems. By comparing with traditional numerical methods, we wished to highlight the advantages of the FNO, in terms of accuracy, efficiency, and scalability. This study also examined the effects of different convolutional layers and input time steps, to better understand the impact of each parameter on the model and to further optimize its performance.

Our experimental results show that compared to traditional numerical methods, such as spectral methods [29,30] and the finite difference method [31], the absolute error of the FNO reaches 10^{-2}. Once the FNO model is trained, it can simultaneously predict multiple datasets based on the batch size used during training. For the SQG equation, using the Fourier pseudo-spectral method takes an average of 43 s to generate one set of simulation data. However, after the FNO model is trained, it only takes 2.38 s to predict two sets of data. Additionally, we found in the experiments that as the number of convolutional layers increases, the training accuracy improves exponentially with the increase in the number of parameters. In the Gray–Scott model experiments, we discovered that the FNO requires sufficient information; short time steps fail to capture the changes in the solution over long time spans, leading to less accurate future time step predictions.

The remainder of this paper is organized as follows. In the Section 2, we introduce the SQG and GS models. Section 3 provides a detailed explanation of the FNO network architecture and the numerical methods used for data generation. We present the experimental results in Section 4. We provide the conclusions in Section 5. The Appendix A provides some intermediate experimental results obtained during the process of achieving the desired outcomes, for reference and further analysis.

2. Reaction–Diffusion Equations

In this paper, we focused on investigating two kinds of reaction–diffusion equations. The first kind was SQG equations, and the second kind was GS. A brief introduction is given, as follows.

2.1. Surface Quasi-Geostrophic Equation

The Surface Quasi-Geostrophic (SQG) is a partial differential equation used to describe the evolution of temperature and vorticity in the hydrodynamics of the Earth's atmosphere and ocean surfaces. The SQG equation is a simplified model derived from the Quasi-Geostrophic equation [32], which assumes that the potential vortex is mainly concentrated in the surface layer. Its mathematical form is similar to that of the two-dimensional vorticity equation, but it is richer and more complex in modeling physical phenomena because it takes into account the effect of temperature gradients. In this paper, we considered the 2D inviscid SQG equation. The 2D inviscid form of the SQG equation is as follows:

$$\theta_t + u \cdot \nabla\theta = 0 \tag{1}$$

Here, θ represents buoyancy, u is the horizontal velocity field, and ∇ is the horizontal gradient operator. In the domain $\Omega = (0, 2\pi)^2$, we performed numerical simulations of the SQG equation using the Fourier pseudo-spectral method, which showed good simulation results. The Fourier pseudo-spectral method is an efficient numerical technique that represents variables in the frequency domain, allowing for accurate handling of periodic boundary conditions and high computational efficiency. The detailed procedure for the numerical solution is shown in Numerical Methods Section 3.2. The initial conditions were as follows:

$$\theta(\boldsymbol{x}, 0) = -\exp\left(-\frac{16}{\pi^2}\left((x - k\pi)^2 + 36(y - k\pi)^2\right)\right), k \in (0, 1) \tag{2}$$

where k was a bias variable. By introducing the bias variable k, we could generate different initial conditions at various positions. We conducted numerical simulations using the Fourier pseudospectral method for 100 sets of initial conditions, with a spatial grid resolution of 256×256. The simulation time range was $T = [0, 20]$, with a time step of 0.005. During the simulation, data were sampled every 200 time steps (i.e., every one unit of real time), resulting in a total of 20 samples. This setup ensured that the data had sufficient spatial and temporal resolution under different initial conditions and at different times, facilitating subsequent model training and analysis.

2.2. Gray–Scott Model

The Gray–Scott model is an autocatalytic reaction–diffusion model that includes the presence of stable single spots, the self-replication of spots, the spontaneous formation of stripes and hexagonal spot arrays, and Turing patterns [33]. The Gray–Scott equations describe an autocatalytic reaction–diffusion process between two chemical constituents with concentrations u and v:

$$\begin{aligned}\frac{\partial u}{\partial t} &= D_u \nabla^2 u - uv^2 + F(1-u) \\ \frac{\partial v}{\partial t} &= D_v \nabla^2 v + uv^2 - (F+k)v\end{aligned} \tag{3}$$

The corresponding chemical reactions under periodic boundary conditions are as follows [1]:

$$\begin{gathered}U + 2V \rightarrow 3V \\ V \rightarrow P\end{gathered} \tag{4}$$

We used the finite difference method to perform numerical simulations of the Gray–Scott model, which involved different chemical substances U, V, and P. Here, u and v represented the concentrations of the chemical substances U and V, respectively. The model parameters included D_u and D_v, which were the diffusion rates of U and V; k, which was the rate at which V was converted to P; and F, which represented the rate of replenishment of U and the consumption rates of U, V, and P during the reaction. The simulation domain was set as $[-1, 1]^2$, and it was divided into a grid of 101×101 cells. We randomly generated 1 to 5 squares of size 36×36 in different locations within the domain. The concentration value inside the squares was set to 1, to represent the initial reactant U, while the region outside the squares represented the initial concentration of reactant V. Since u and v satisfied the conservation relation $u + v = u_0 + v_0$ during the reaction, this could be simplified to a single-variable problem. We performed numerical simulations under periodic boundary conditions, with the time range set to [0, 10,000] and a time step of $dt = 0.25$. The diffusion coefficients were set to $D_u = 1.0$ and $D_v = D_u/2$, with a reaction rate of $F = 0.055$ and a conversion rate of $k = 0.063$. The simulation time was long enough to ensure the system reached a steady state, allowing for effective analysis of the model training results. Data were sampled every 200 time units, resulting in 50 samples for subsequent model training and analysis. We generated 1000 sets of different initial conditions, including varying positions and numbers of squares, to ensure the comprehensive training and effectiveness of the model. Detailed simulation and data processing procedures are provided in Numerical Methods Section 3.3.

3. Numerical Methods

3.1. FNO

In this section, we briefly introduce the process of solving PDEs by the FNO. Consider the following PDE:

$$\begin{aligned}(\mathcal{L}_a u)(x) &= f(x), \quad && x \in D \\ u(x) &= 0, && x \in \partial D\end{aligned} \tag{5}$$

where f is a function determined by $\mathcal{L}_a$. For a second-order elliptic partial differential equation, $\mathcal{L}_a$ is $-div(a\nabla\cdot)$. One method to solve the analytical solution of the partial differential equation is the Green's function representation. Using this method, we can obtain

$$\mathcal{L}_a G(x, \cdot)) = \delta_x \tag{6}$$

where δ_x is the Dirac measure centered at x on $\mathbb{R}^d$, and where $G(x, \cdot)$ represents the Green's function associated with the operator $\mathcal{L}_a$. Using the Green's function method, we can derive the formal solution of Equation (5) as follows:

$$\begin{aligned}(\mathcal{L}_a u)(x) &= \int_D (\mathcal{L}_a G(x, \cdot))(y) f(y) dy \\ &= \int_D \delta_x(y) f(y) dy \\ &= f(x)\end{aligned} \tag{7}$$

$$u(x) = \int_D G_a(x, y) f(y)\, dy \tag{8}$$

Naturally, Green's function can be parameterized using a neural network to approximate $\mathcal{L}_a$. Guided by Equation (8), Li et al. [25] proposed the following iterative architecture for $t = 0, 1, 2, \cdots, T-1$:

$$v_{t+1}(x) = \sigma\Big(W v_t(x) + \int_D \kappa_\phi(x, y, a(x), a(y)) v_t(y) \nu_x(\mathrm{d}y)\Big), \tag{9}$$

where ν_x is a fixed Borel measure for each $x \in D$, and where the parameters ϕ in the matrix W and the kernel κ_ϕ are learned from the data. To solve the integral term in Equation (9), the authors replaced the integral with the Fourier transform, resulting in the Fourier Neural Operator. Let $\mathcal{F}$ denote the Fourier transform of the function f, and let $\mathcal{F}^{-1}$ denote its inverse transform:

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle} \mathrm{d}x, \quad \left(\mathcal{F}^{-1} f\right)_j(x) = \int_D f_j(k) e^{2i\pi\langle x,k\rangle} \mathrm{d}k$$

By removing the dependence on the function a and assuming the term $\kappa_\phi(x, y) = \kappa_\phi(x - y)$, we find that $\int_D \kappa_\phi(x, y, a(x), a(y)) v_t(y) \nu_x(\mathrm{d}y)$ becomes a convolution operator. Therefore, parameterizing κ_ϕ in the Fourier space allows the use of the fast Fourier transform for integral computation. Applying the convolution theorem, we obtain

$$(\mathcal{K}(a; \phi) v_t)(x) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(v_t)\big)(x), \quad \forall x \in D$$

Therefore, κ_ϕ can be parameterized in the Fourier space. The integral operator in the Fourier space can be defined as

$$(\mathcal{K}(\phi) v_t)(x) = \mathcal{F}^{-1}\big(R_\phi \cdot (\mathcal{F} v_t)\big)(x), \quad \forall x \in D$$

where R_ϕ is the complex matrix learned from data in the Fourier space. Equation (9) can be updated to $v_{t+1}(x) = \sigma(W v_t(x) + (\mathcal{K}(a; \phi) v_t)(x)), \quad \forall x \in D$.

To improve the model's capacity to capture intricate relationships, the raw input data are transformed into a high-dimensional feature space. The authors designed the following framework:

$$\begin{aligned}&v_0(x) = P(x, a(x), a_\epsilon(x), \nabla a_\epsilon(x)) + p \\ &v_{t+1}(x) = \sigma(W v_t(x) + (\mathcal{K}(a; \phi) v_t)(x)) \\ &u(x) = Q v_T(x) + q\end{aligned} \tag{10}$$

where σ is an activation function, P and Q are analogous to domain encoders and decoders, P transforms input data from physical space (spatial domain) to Fourier space (frequency domain), while Q transforms data from Fourier space (frequency domain) to physical space (spatial domain). Figure 1 shows the overall procedure more clearly.

More details can be found in the original article [23,25]; in addition, article [34] proves that the FNO can approximate any continuous operator.

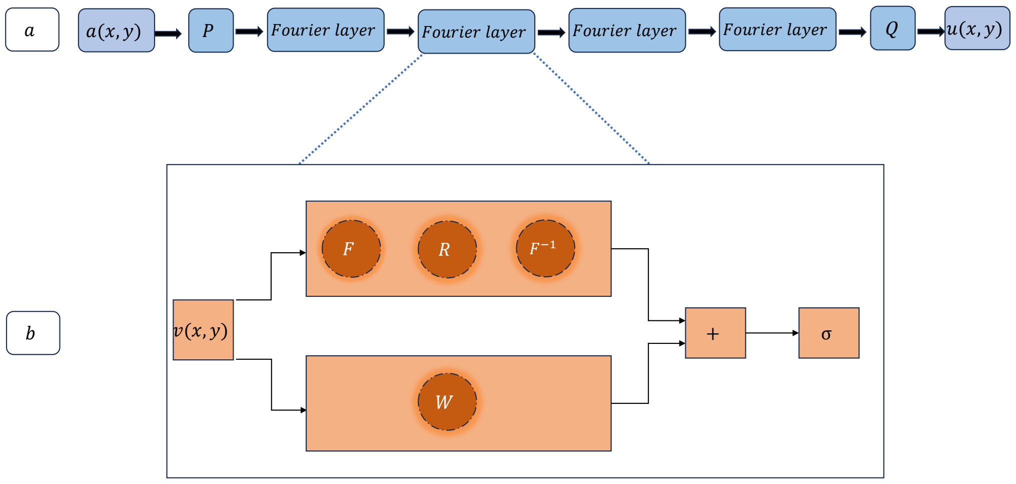


Figure 1. (**a**) the structural framework of the complete Fourier Neural Operator; (**b**) the structure of a Fourier layer.

3.2. Fourier Pseudo-Spectral Method for Solving the SQG Equation

Reviewing the Surface Quasi-Geostrophic (SQG) equation

$$\theta_t + u \cdot \nabla\theta = 0, \tag{11}$$

where θ is the buoyancy and u is the velocity field, for the SQG model, the velocity field is given by the stream function ψ:

$$u = \left(-\frac{\partial\psi}{\partial y}, \frac{\partial\psi}{\partial x}\right) \tag{12}$$

The relationship between the buoyancy and the stream function is

$$\theta = -\frac{\partial\psi}{\partial z} \quad \text{at}, z = 0 \quad \text{and} \quad \Delta\psi = 0 \quad \text{for}, z \neq 0 \tag{13}$$

In the Fourier transform, we consider the horizontal spatial variables x and y, defining the two-dimensional Fourier transform and its inverse as

$$\hat{\theta}(k_x, k_y, t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \theta(x, y, t)e^{-i(k_x x + k_y y)}\, dx\, dy \tag{14}$$

$$\theta(x, y, t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \hat{\theta}(k_x, k_y, t)e^{i(k_x x + k_y y)}\, dk_x\, dk_y \tag{15}$$

Similarly, the Fourier transform of the stream function ψ is $\hat{\psi}$, and we have

$$\hat{\theta}(k_x, k_y, t) = -i|k|\hat{\psi}(k_x, k_y, t) \tag{16}$$

Next, the Fourier transform is applied to the SQG equation:

$$\theta_t + u \cdot \nabla\theta = 0 \tag{17}$$

Thus, in the frequency domain, the SQG equation becomes

$$\hat{\theta}_t + (\widehat{u \cdot \nabla\theta}) = 0 \tag{18}$$

We need to calculate the Fourier transform of the nonlinear term $\widehat{u \cdot \nabla\theta}$. The velocity field u can be expressed in the frequency domain as

$$\hat{u} = \left(-i\frac{k_y}{|k|}\hat{\psi}, i\frac{k_x}{|k|}\hat{\psi}\right) \tag{19}$$

We perform a Fourier transform on $\nabla\theta$:

$$\widehat{\nabla\theta} = (ik_x\hat{\theta}, ik_y\hat{\theta}) \tag{20}$$

We calculate the nonlinear term $\widehat{u \cdot \nabla\theta}$:

$$\widehat{u \cdot \nabla\theta} = \mathcal{F}\left(\left(-i\frac{k_y}{|k|}\psi\right)(ik_x\theta) + \left(i\frac{k_x}{|k|}\psi\right)(ik_y\theta)\right) \tag{21}$$

Using the convolution theorem, this nonlinear term becomes a convolution of the Fourier transform:

$$\widehat{u \cdot \nabla\theta} = \sum_{\mathbf{k}'}\left(-\frac{k'_y}{|k'|}k_x + \frac{k'_x}{|k'|}k_y\right)\hat{\psi}(\mathbf{k}')\hat{\theta}(\mathbf{k}-\mathbf{k}') \tag{22}$$

Finally, we obtain the SQG equation in the Fourier space:

$$\frac{\partial\hat{\theta}(\mathbf{k},t)}{\partial t} + \sum_{\mathbf{k}'}\left(-\frac{k'_y}{|k'|}k_x + \frac{k'_x}{|k'|}k_y\right)\hat{\psi}(\mathbf{k}')\hat{\theta}(\mathbf{k}-\mathbf{k}') = 0 \tag{23}$$

This equation can be solved numerically, using the Runge–Kutta method. With Δt as the time step, we have

$$\hat{\theta}(\mathbf{k}, t+\Delta t) = \hat{\theta}(\mathbf{k}, t) - \Delta t\sum_{\mathbf{k}'}\left(-\frac{k'_y}{|k'|}k_x + \frac{k'_x}{|k'|}k_y\right)\hat{\psi}(\mathbf{k}')\hat{\theta}(\mathbf{k}-\mathbf{k}') \tag{24}$$

Using the inverse Fourier transform, the updated $\hat{\theta}(\mathbf{k}, t+\Delta t)$ can be transformed back to the physical space. The solution θ can be derived as follows:

$$\theta(x, y, t+\Delta t) = \mathcal{F}^{-1}\big(\hat{\theta}(k_x, k_y, t+\Delta t)\big) \tag{25}$$

3.3. Differential Interpolation for Solving Gray–Scott Model

$$\begin{aligned}\frac{\partial u}{\partial t} &= D_u\nabla^2 u - uv^2 + F(1-u)\\ \frac{\partial v}{\partial t} &= D_v\nabla^2 v + uv^2 - (F+k)v\end{aligned} \tag{26}$$

For the two-dimensional Gray–Scott Model, the variables u and v satisfy conservation laws. We first discretize the Laplacian operator, using the five-point finite difference method:

$$\nabla u = -4u_{i,j} + u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} \tag{27}$$

For the continuous form of the reaction terms described above, we can discretize it using the five-point stencil for the Laplacian operator. The discretized iterative scheme is as follows:

For U :

$$U^{t+\Delta t}_{i,j} = U^t_{i,j} + \Delta t\left(D_U\Delta U^t_{i,j} - U^t_{i,j}\cdot\left(V^t_{i,j}\right)^2 + F\cdot\left(1-U^t_{i,j}\right)\right) \tag{28}$$

For V :

$$V^{t+\Delta t}_{i,j} = V^t_{i,j} + \Delta t\left(D_V\Delta V^t_{i,j} + U^t_{i,j}\cdot\left(V^t_{i,j}\right)^2 - (k+F)\cdot V^t_{i,j}\right) \tag{29}$$

The discrete form of the boundary conditions for the matrix U is as follows:

$$\begin{aligned} U_{0,j} &= U_{n-2,j}, \quad U_{n-1,j} = U_{1,j} \quad \text{for } j = 0,1,\ldots,n-1 \\ U_{i,0} &= U_{i,n-2}, \quad U_{i,n-1} = U_{i,1} \quad \text{for } i = 0,1,\ldots,n-1 \end{aligned} \tag{30}$$

The discrete form of the boundary conditions for the matrix V is as follows:

$$\begin{aligned} V_{0,j} &= V_{n-2,j}, \quad V_{n-1,j} = V_{1,j} \quad \text{for } j = 0,1,\ldots,n-1 \\ V_{i,0} &= V_{i,n-2}, \quad V_{i,n-1} = V_{i,1} \quad \text{for } i = 0,1,\ldots,n-1 \end{aligned} \tag{31}$$

After setting the initial conditions, we use the method as above to solve the model.

4. Numerical Results

In this section, we present the numerical results. We compared the FNO solutions with numerical solutions of the reaction–diffusion equations. The results show that the error between the FNO solution and the numerical solution was less than 10^{-2}, indicating that the FNO achieves high precision in solving these complex models.

4.1. Surface Quasi-Geostrophic Equation

We trained the FNO model using the numerical results, and we studied the impact of different convolutional layers on the training outcome. In the FNO training process, we used information from 10 time steps as input ($T_{\text{in}} = 10$), and we set the output size to 1 (step = 1). Each FNO autoregressively advanced to the 20th time instance to allow the neural operator to learn the function's temporal evolution at that point ($T_{\text{out}} = 20$). During training, 90 datasets were used as the training set and 10 datasets were used as the test set. The model was trained with a four-layer convolutional network, using the Adam optimizer for 500 epochs, with a learning rate of 0.001. In the Fourier Neural Operator (FNO), each Fourier layer had 12 modes and 20 widths. The modes represented the number of frequency components retained, while the widths represented the dimension of each frequency component, configured to effectively capture the complex features of the input data. The choice of 12 modes was based on the experimental results, where different mode values (6, 12, 18, 24) were tested on the dataset, with all other parameters kept consistent. Figure 2 shows the training and testing loss achieved after 500 epochs for different modes. As shown in the figure, both the training and testing loss exhibited significant decreases up to 12 modes. However, beyond 12 modes, the reduction in loss became minimal, indicating a plateau where further increases in the number of modes provided diminishing returns, in terms of performance improvement. Therefore, based on the observed trend where the loss plateaued after 12 modes, we chose 12 modes as the optimal configuration. This selection struck a balance between minimizing loss and maintaining computational efficiency.

After the model training, we simulated two new sets of data to evaluate the performance of the models. These two newly simulated datasets were neither part of the training set nor the test set. The input consisted of data from the first 10 time steps, to predict the next time step (i.e., the 11th time step). For the subsequent forward pass, the model received time steps 1–11, where the 10 time steps were composed of time steps 2–10 and the first prediction of time step 11. This sliding window-update-and-prediction process was repeated until the desired number of time steps was predicted. Figure 3 shows the test results of the trained model on these two datasets. The figure shows that our model learning from the data can predict the numerical solution exactly.

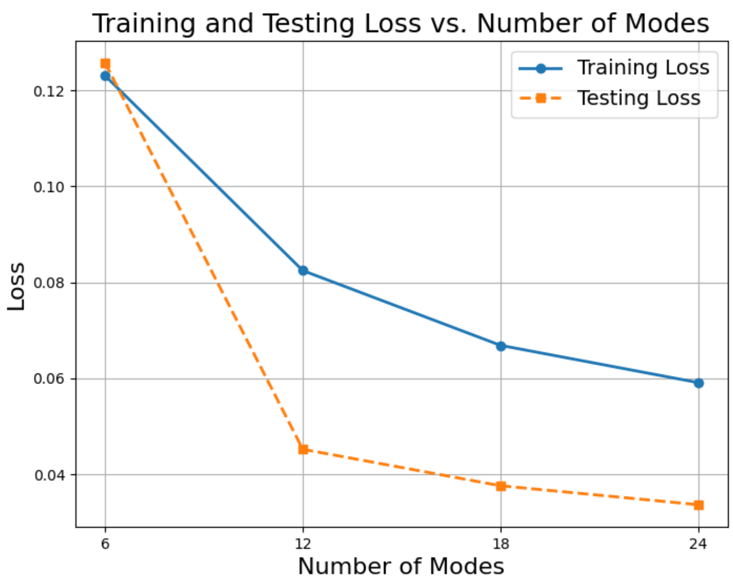


Figure 2. The plot illustrates the training and testing loss after 500 epochs for different numbers of Fourier modes (6, 12, 18, 24).

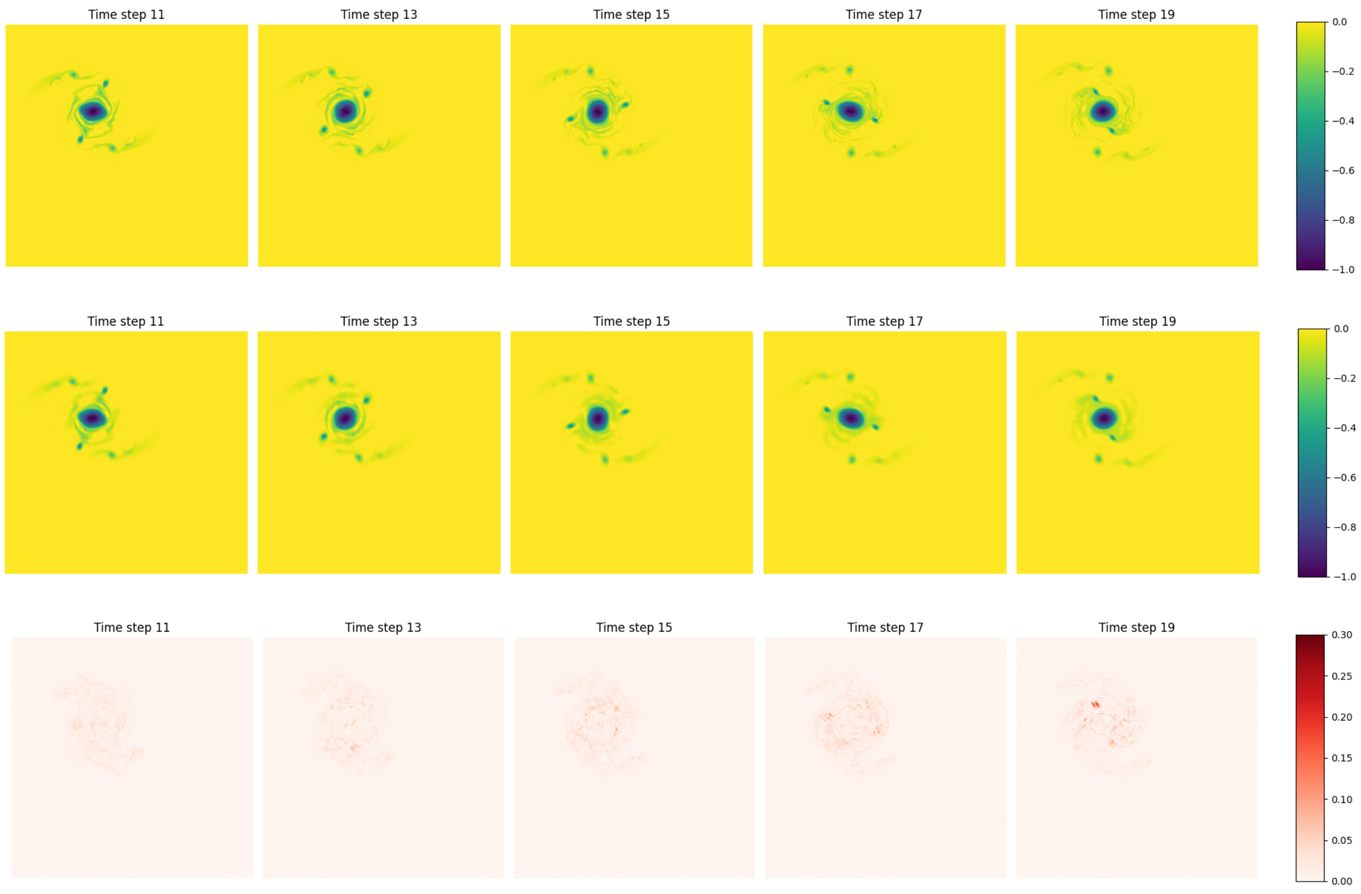


Figure 3. The first row of images shows the values of the solution at time steps 11, 13, 15, 17, and 19 resulting from the numerical method. The second row of images displays the predicted values of the FNO at these same time steps. The third row of images shows the absolute error between the numerical solution and the FNO prediction.

In order to investigate the effect of different numbers of convolutional layers (i.e., the number of parameters) on the model training effect, we designed an experiment. We used

100 sets of data to train models on different numbers of convolutional layers (three, four, five, and six layers, respectively). During the training process, each model was trained for 500 epochs, with the learning rate set to 0.001. Figure 4 illustrates the variation of the loss function during the training process. It can be observed that while the loss generally decreased as the number of convolutional layers increased, there was a slight exception where the four-layer model had a slightly higher loss compared to the two-layer and three-layer models. However, from five layers onwards, the loss significantly decreased. In the experiments, it was found that overfitting occurred after 10 layers, highlighting the importance of selecting an appropriate number of layers. Additionally, Table 1 shows the number of model parameters corresponding to different numbers of convolutional layers, as well as the final training and test losses after 500 epochs of training.

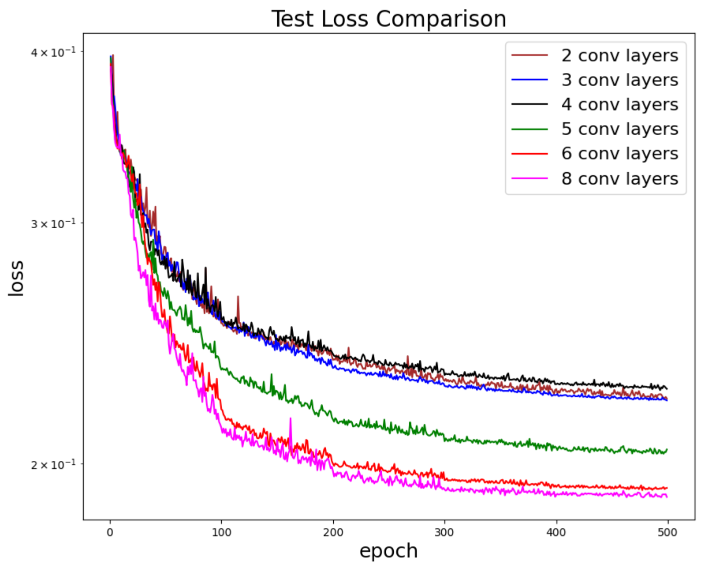
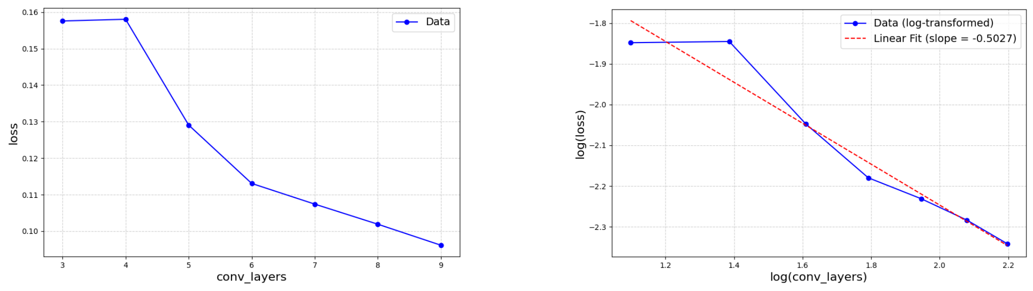


Figure 4. Comparison of test errors across various models with different numbers of convolutional layers (y-axis: logarithmic scale of loss values).

Table 1. Performance of different configurations. All the computation was carried out on a single Nvidia A40 GPU with 24 GB memory.

Config	Parameters	Time Per Epoch(s)	Train Loss	Test Loss
FNO-3 Conv	695,657	27.01	0.157565	0.222477
FNO-4 Conv	926,517	28.03	0.158038	0.226834
FNO-5 Conv	1,157,377	28.12	0.129072	0.204844
FNO-6 Conv	1,388,237	31.33	0.113070	0.192028

It can be observed that each additional layer increased the number of parameters by 230,460. This increase was due to the model's width and modes. Specifically, each Fourier layer contained two complex weight tensors, each with dimensions (in_channels, out_channels, modes1, modes2). For a given number of input and output channels (20) and Fourier modes (12), the total number of real parameters for each weight tensor was 230,400 (two weight tensors, each containing $20 \times 20 \times 12 \times 12 \times 2$ real parameters). Additionally, considering the `Conv2d` layer in each layer (which added $20 \times 20 + 20 = 420$ parameters) and the `BatchNorm2d` layer (which added $2 \times 20 = 40$ parameters), the total number of additional parameters was 230,860. We used the trained model to make predictions on two completely new datasets that were not included in the original training and test sets. Specifically, we used the model to predict the result at time step 11, using data from time steps 0 to 10, then added the prediction for time step 11 to the dataset. This process was iteratively repeated until predictions were made for up to time step 20. Figure 5a shows the error achieved after training for 500 epochs with different numbers of convolutional layers. To investigate the relationship between the number of convolutional layers and the final error, we applied a logarithmic transformation to both the number of layers and the error values. As shown in Figure 5b, the resulting linear relationship suggests that the final error followed an exponential decay with respect to the number of convolutional layers.



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

Figure 5. The above figure shows the effect of different convolutional layer variations on the experimental results: (**a**) comparison of test errors for different convolutional layers; (**b**) taking logarithms of both the test error and the number of convolutional layers.

Through these experiments, we evaluated the effect of different numbers of convolutional layers on the model error, to determine the optimal model structure. The results show that the FNO model performs well with different numbers of convolutional layers, especially when more layers are used, and that the model's solving ability is more significant.

For a model trained entirely with the Vortex Initial Condition, we next considered evaluating its effectiveness on a completely different initial condition, the Sinusoidal Initial Condition. The Sinusoidal Initial Condition is defined as follows:

$$\theta(\boldsymbol{x}, 0) = \sin(x)\sin(y) + \cos(y) \tag{32}$$

A bias variable k was also introduced, so that the initial conditions in Equation (2) could also be generated at different locations, and the form with the introduction of the bias variable was as follows:

$$\theta(\boldsymbol{x}, 0) = \sin(x - k\pi)\sin(y - k\pi) + \cos(y - k\pi), k \in (0, 1) \tag{33}$$

Using the Sinusoidal Initial Condition, we generated two sets of data with the Fourier pseudo-spectral method. When predicting with the model trained on the Vortex Initial Condition, the results were not very satisfactory (see Appendix A for the prediction results). The unsatisfactory results were likely due to the fact that the Vortex Initial Condition involves complex rotational patterns, whereas the Sinusoidal Initial Condition represents simpler periodic patterns. The Vortex Initial Condition and the Sinusoidal Initial Condition have different spatial frequencies and modes. If a model is highly adapted to a certain pattern, it may not effectively capture other patterns. To address this, we considered training the model simultaneously on both the Vortex Initial Condition and the Sinusoidal Initial Condition, to test whether the FNO had generalization capabilities.

To further demonstrate the generalizability of the FNO, we similarly used this bias variable to generate 100 sets of initial conditions at different locations for numerical simulations using the Fourier spectral method. Since Initial Condition 1 (Equation (2)) and Initial Condition 2 (Equation (32)) were both valid for SQG, and now that we were employing operator learning, to validate the effectiveness of the operator learning, we combined the two sets of data. Thus, we now had 200 datasets for training. We selected a six-layer convolutional network FNO for model training, with the modes still set to 12 and the width set to 20. The model was trained using the Adam optimizer for 500 epochs, with a learning rate of 0.001. After training the model, we generated two sets of new initial conditions and fitted the model to these conditions. The predicted results are shown in Figures 6 and 7.

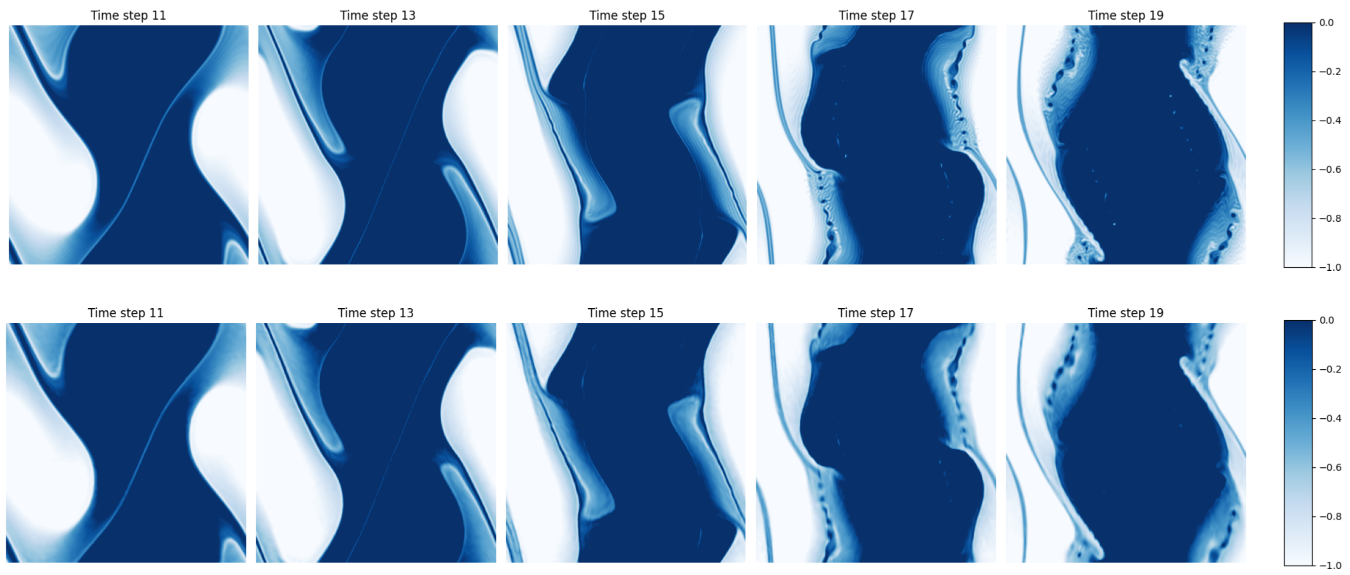


Figure 6. Numerical solutions (**first row**) and predicted values (**second row**) for the Sinusoidal Initial Condition at time steps 11, 13, 15, 17, and 19 from left to right, respectively.

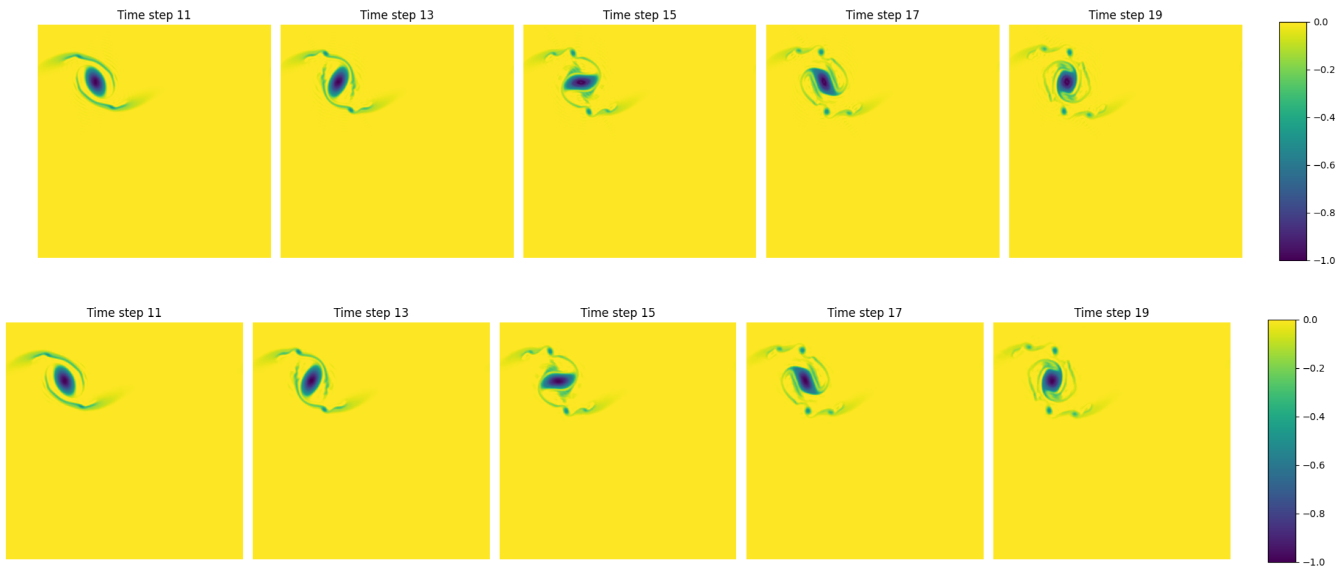


Figure 7. Numerical solutions (**first row**) and predicted values (**second row**) of the Vortex Initial Condition at 11, 13, 15, 17, and 19 from left to right, respectively.

Figures 6 and 7 demonstrate the generalization ability of the Fourier Neural Operator (FNO). Even where initial conditions with significant differences were used during training, the FNO was still able to effectively capture and learn the behavior of the entire operator. Such results indicate that the FNO has strong generalization abilities, allowing it to extract stable patterns and regularities from diverse input data. This feature enables the FNO to perform well with complex and varied datasets, demonstrating its potential for a wide range of applications. However, it is worth noting that while we trained the model with both vortex and periodic initial conditions, the prediction performance of the FNO was relatively weaker when encountering unseen initial conditions, such as those from unstable modes.

4.2. Gray–Scott

To investigate the impact of different input time steps on the model's performance, we selected an FNO with four convolutional layers, where the modes and width were set to 12 and 20, respectively. In the training process for solving the Gray–Scott model, we examined the performance of the model's autoregressive recursion up to the 20th time step for input steps of 10, 8, 6, 4, and 2, all with an output step of 1, respectively, during the training process.

Input step length: Refers to the number of past time steps used by the model at each prediction step. For example, an input step length of 10 meant that the model used data from the past 10 time steps to predict the next time step.

Output step length: Refers to the number of future time steps generated by the model at each prediction step. For example, an output step length of 1 meant that the model predicted only the next time step's data at each step.

Autoregressive propagation: The model not only relied on the initial historical data for predictions but also used each step's prediction as input for the next step, propagating step-by-step until the desired time step was reached.

The core idea of this approach was to explore and compare the predictive performance of the model under different historical data window sizes with different input step sizes. Autoregressive recursion then allowed the model to better simulate the dynamics of the time series data during the stepwise prediction process.

We took these different time steps to investigate the model, and we gave only the $[0, 10]$ data solved to the 50th time step (both T = 10,000), using the trained model. The following figure shows the numerical solution, followed by the prediction results for input steps of 2, 4, 6, 8, and 10 at time steps [10, 20], respectively.

As shown in Figure 8, we observed that when solving the Gray–Scott model with the FNO, shorter input step lengths resulted in worse prediction performance, leading to the "shorter step length degradation phenomenon". When solving equations numerically, the choice of step size is critical. A shorter input step size means that the model can only use less historical data to make predictions, which can cause several problems:

1. **Insufficient information**: Shorter input step lengths are unable to capture changes in the equation's solution over a long time span. In such cases, the model may fail to recognize important patterns and trends, leading to less accurate predictions for future time steps.

2. **Error accumulation**: In time series prediction, especially with autoregressive models, the prediction error at each step affects subsequent predictions. When the input step length is too short, error accumulation becomes more pronounced because each prediction relies on the previous step's result, which may be inaccurate, due to insufficient information.

3. **Numerical stability**: Shorter step lengths may lead to numerical instability, especially when solving complex equations. A short step length might not effectively smooth out the variations in the solution, causing numerical oscillations or deviations from the true solution.

Therefore, selecting an appropriate input step length is crucial for ensuring the predictive performance of the FNO model. A step length that is too short can lead to insufficient historical information, thereby affecting prediction accuracy, which is known as the shorter-step-length degradation phenomenon. In Figure 9, we analyze how the L2 error between the prediction results and the numerical results varied with step length, where the L2 error was defined as $\epsilon_i = \frac{\|\mathbf{E}_i\|_2}{\|\mathbf{U}_i\|_2} = \frac{\|\mathbf{U}_i - \mathbf{U}_{\text{hat},i}\|_2}{\|\mathbf{U}_i\|_2}$, where i was the ith time step, U_i was the numerical solution for the ith time step, and $U_{\text{hat},i}$ was the prediction result of the FNO corresponding to different time steps.

The L2 error results aligned with the expected outcomes. As the time steps progressed, the L2 error associated with shorter input step lengths increased over time. The findings indicate that larger input step lengths result in smaller L2 errors, thereby improving the model's overall performance.

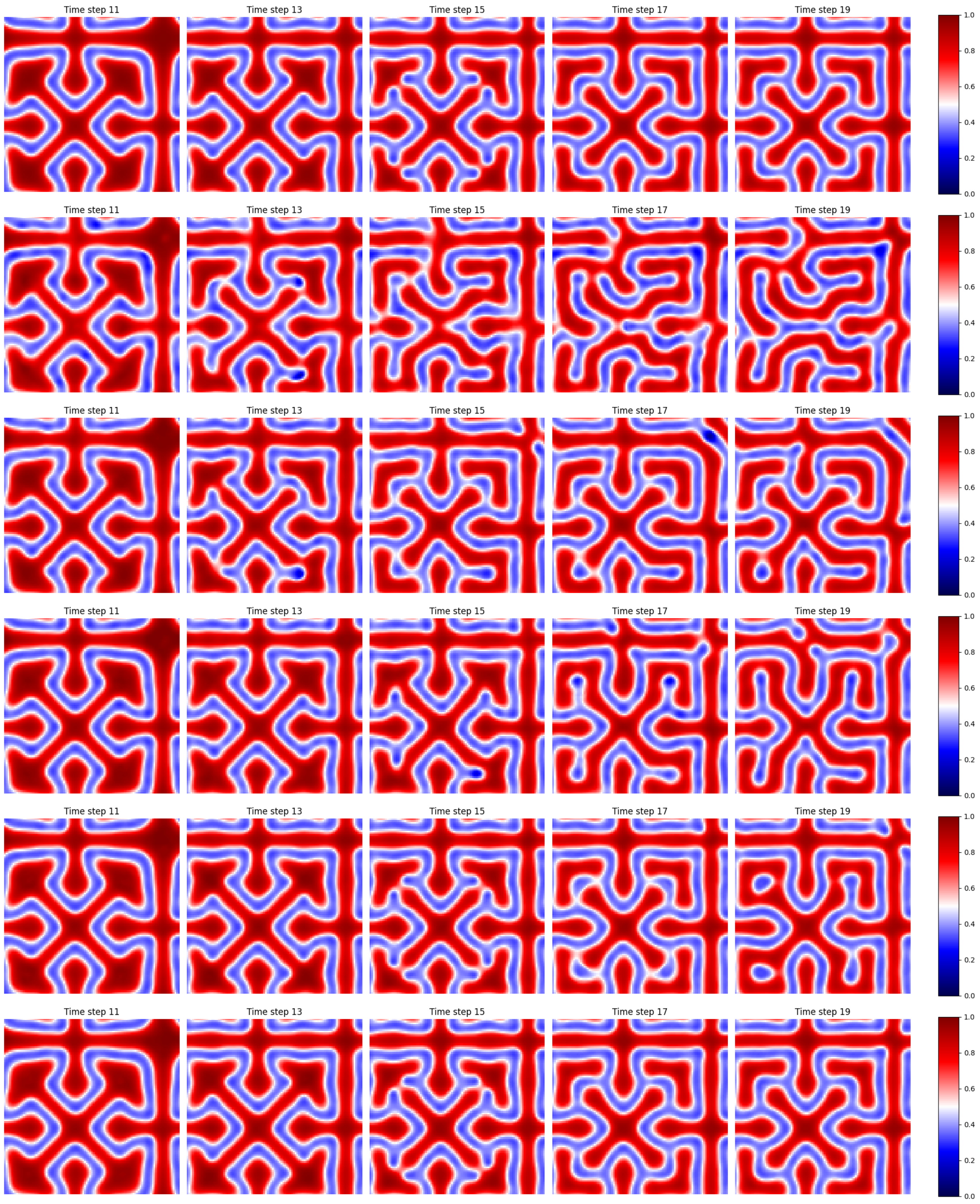


Figure 8. The first row of images shows the numerical solution, followed by the results of predicting at 11, 13, 15, 17, and 19, using input steps of 2, 4, 6, 8, and 10 from top to bottom.

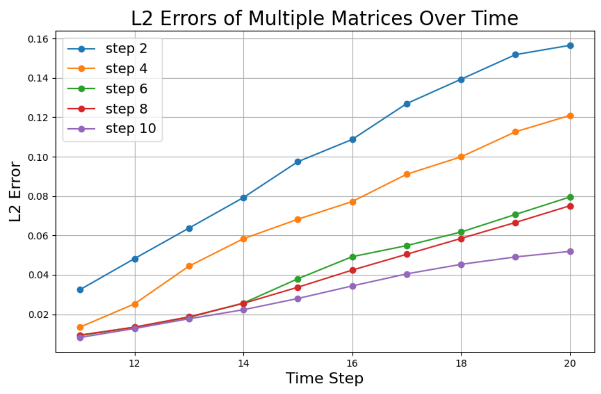


Figure 9. Comparison of predicted L2 errors for different input time steps.

5. Conclusions

In this work, we generated experimental data for the SQG equation, using the Fourier pseudospectral method, and for the Gray–Scott equation, using the finite difference method. Our study demonstrated the accuracy of the FNO in solving complex nonlinear PDE problems, and it achieved a relative error of 10^{-2} compared to numerical methods. In this paper, we also conducted extensive hyperparameter experiments. First, we investigated the modes (frequency parameters) used during training, we analyzed their impact on the experimental results, and we determined the most suitable modes for this study. Next, we experimented with the number of convolutional layers. The results showed that the model's performance did not differ significantly when using two, three, or four layers, with the performance of two or three layers slightly surpassing that of four layers. However, when the number of layers increased to 5, the model's performance improved significantly, although overfitting became evident with more than 10 layers. Based on these observations, we ultimately selected the most suitable parameters, specifically modes = 12 and layers = six, to ensure the best model performance. In the experiments with the SQG equation, although using only 100 datasets for training achieved a prediction error in the order of 10^{-2} compared to the data generated by the spectral method, the model's performance was unsatisfactory when predicting initial conditions that were significantly different and had not been encountered before. In the study of the Gray–Scott model with different input time steps, due to the FNO model's implicit learning of temporal relationships, short time steps may fail to capture information changes over long time spans, leading to inaccurate predictions of future time steps, with the issue of error accumulation worsening over time.

In our future work, we aim to extend the FNO to three-dimensional spaces. Our initial tests on the three-dimensional Navier–Stokes equations have yielded promising results. Additionally, we will explore the application of the FNO to various boundary conditions beyond the nonzero and periodic cases, thereby enhancing its applicability to a broader range of models.

Author Contributions: Software, Y.H.; Investigation, Y.H.; Data curation, Y.H.; Writing—review & editing, F.S.; Supervision, F.S.; Project administration, F.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the NSF of China (grant No. 11901100) and the Natural Science Foundation of Fujian Province of China (Grant No. 2020J05111).

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Acknowledgments: We thank the anonymous referees for their useful suggestions.

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

Appendix A

Appendix A.1. More Results

The figure below shows the results of using a model trained on the Vortex Initial Condition to predict the Sinusoidal Initial Condition, where the performance was found to be suboptimal. To verify whether the FNO remained effective when the two initial conditions differed significantly, we incrementally added 5 different datasets generated from the Sinusoidal Initial Condition, combining them with 100 datasets generated from the Vortex Initial Condition for training. We then used two new datasets from the Sinusoidal Initial Condition to evaluate the model's performance.

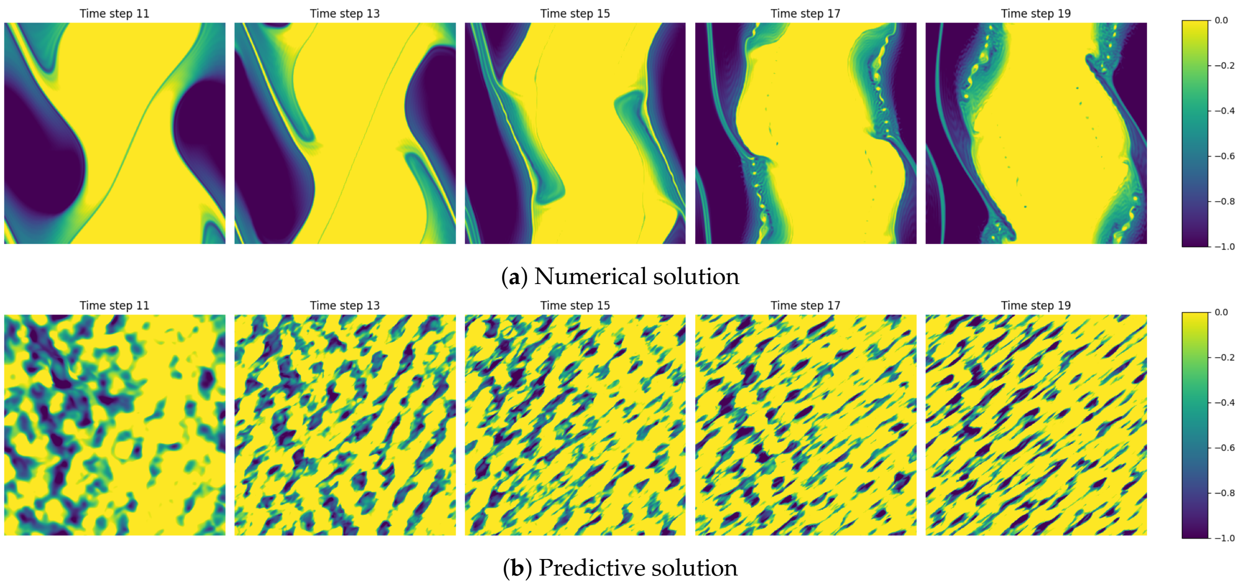


Figure A1. (**a**) the values of the solution at time steps 11, 13, 15, 17, and 19 resulting from the numerical method; (**b**) the figure shows the results of a model trained using only the Vortex Initial Conditions to predict the Sinusoidal Initial Conditions at time steps 11, 13, 15, 17, and 19.

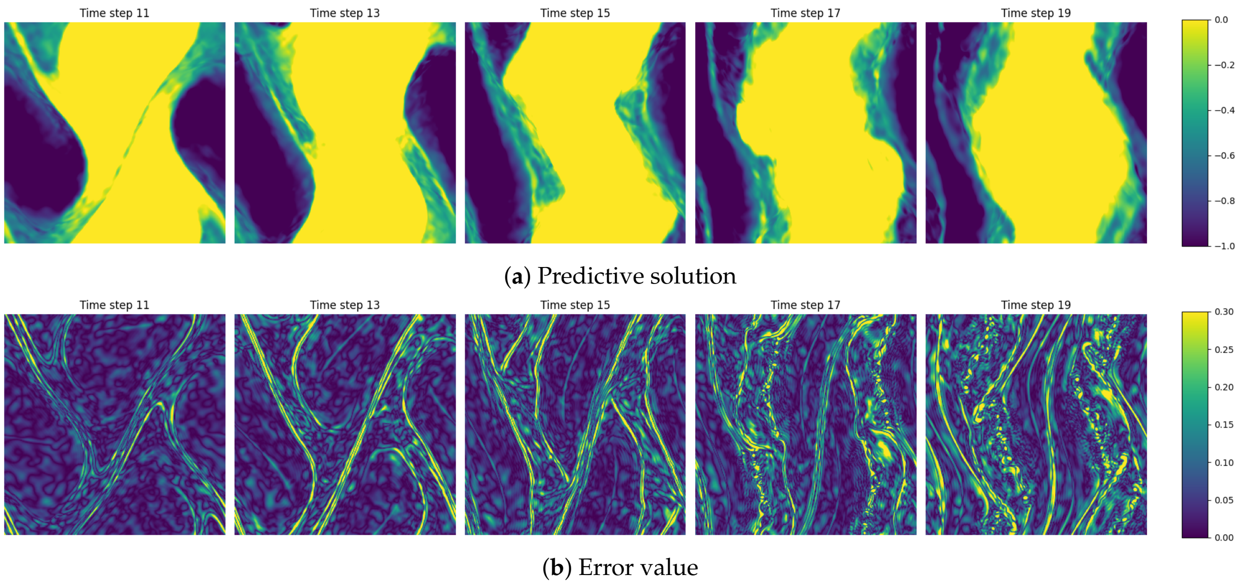
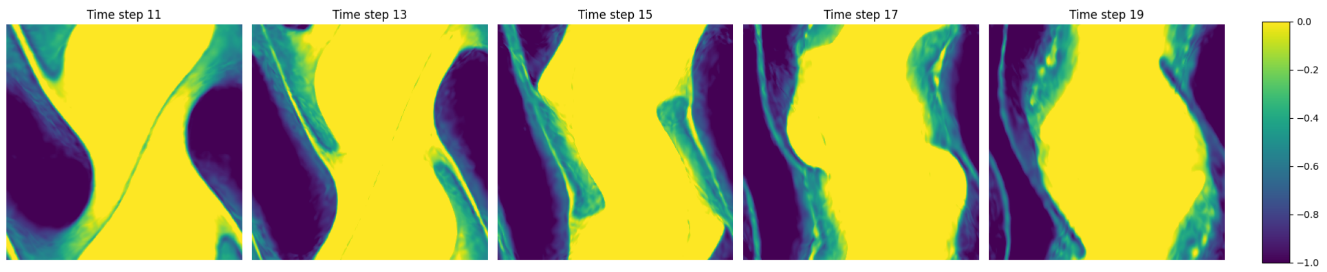
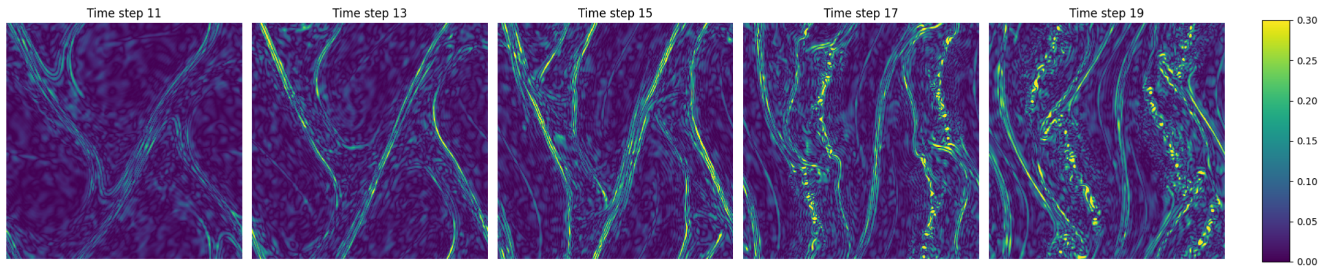


Figure A2. (**a**) the figure shows the results of a model trained using a combination of 100 Vortex Initial Conditions and five Sinusoidal Initial Conditions, to predict the Sinusoidal Initial Conditions at time steps 11, 13, 15, 17, and 19; (**b**) the error between the numerical solution and the prediction is demonstrated.

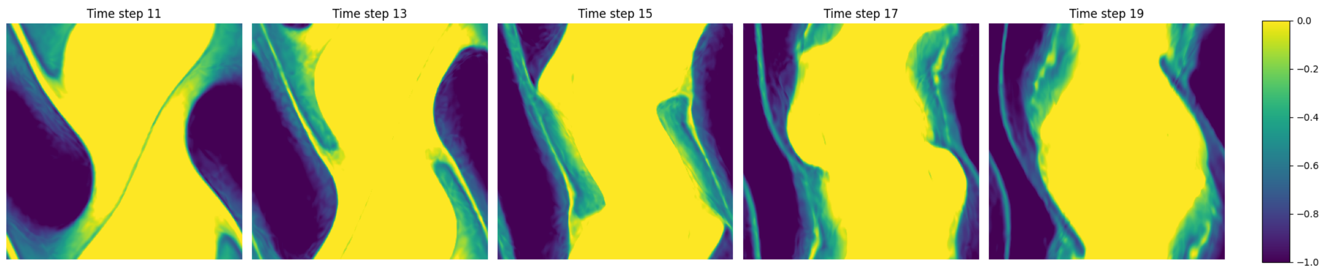


(**a**) Predictive solution

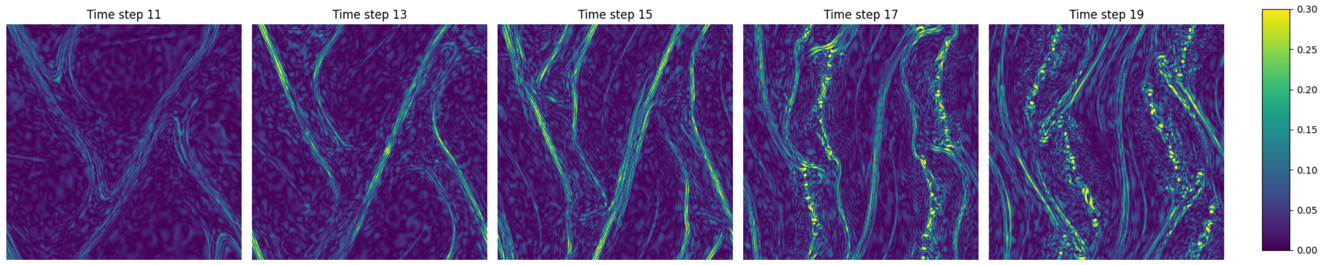


(**b**) Error value

Figure A3. (**a**) the figure shows the results of a model trained using a combination of 100 Vortex Initial Conditions and 15 Sinusoidal Initial Conditions, to predict the Sinusoidal Initial Conditions at time steps 11, 13, 15, 17, and 19; (**b**) the error between the numerical solution and the prediction is demonstrated.



(**a**) Predictive solution



(**b**) Error value

Figure A4. (**a**) the figure shows the results of a model trained using a combination of 100 Vortex Initial Conditions and 25 Sinusoidal Initial Conditions, to predict the Sinusoidal Initial Conditions at time steps 11, 13, 15, 17, and 19; (**b**) the error between the numerical solution and the prediction is demonstrated.

References

1. Li, Q.; Song, F. Splitting spectral element method for fractional reaction-diffusion equations. *J. Algorithms Comput. Technol.* **2020**, *14*, 1748302620966705. [CrossRef]
2. Shi, S.; Song, F. Scalar auxiliary variable approache for the surface quasi-geostrophic equation. *J. Algorithms Comput. Technol.* **2023**, *17*, 17483026231176203. [CrossRef]
3. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; et al. Language models are few-shot learners. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 1877–1901.
4. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
5. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*. [CrossRef]
6. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
7. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
8. Fukami, K.; Fukagata, K.; Taira, K. Super-resolution reconstruction of turbulent flows with machine learning. *J. Fluid Mech.* **2019**, *870*, 106–120. [CrossRef]
9. Long, Z.; Lu, Y.; Ma, X.; Dong, B. Pde-net: Learning pdes from data. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 3208–3216.
10. Zhu, Y.; Zabaras, N.; Koutsourelakis, P.S.; Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.* **2019**, *394*, 56–81. [CrossRef]
11. Gao, H.; Sun, L.; Wang, J.X. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *J. Comput. Phys.* **2021**, *428*, 110079. [CrossRef]
12. 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]
13. Raissi, M. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *J. Mach. Learn. Res.* **2018**, *19*, 1–24.
14. Karniadakis, G.E.; Kevrekidis, I.G.; Lu, L.; Perdikaris, P.; Wang, S.; Yang, L. Physics-informed machine learning. *Nat. Rev. Phys.* **2021**, *3*, 422–440. [CrossRef]
15. Cai, S.; Mao, Z.; Wang, Z.; Yin, M.; Karniadakis, G.E. Physics-informed neural networks (PINNs) for fluid mechanics: A review. *Acta Mech. Sin.* **2021**, *37*, 1727–1738. [CrossRef]
16. Rao, C.; Sun, H.; Liu, Y. Physics-informed deep learning for computational elastodynamics without labeled data. *J. Eng. Mech.* **2021**, *147*, 04021043. [CrossRef]
17. Kissas, G.; Yang, Y.; Hwuang, E.; Witschey, W.R.; Detre, J.A.; Perdikaris, P. Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. *Comput. Methods Appl. Mech. Eng.* **2020**, *358*, 112623. [CrossRef]
18. Goswami, S.; Anitescu, C.; Chakraborty, S.; Rabczuk, T. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theor. Appl. Fract. Mech.* **2020**, *106*, 102447. [CrossRef]
19. Geneva, N.; Zabaras, N. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *J. Comput. Phys.* **2020**, *403*, 109056. [CrossRef]
20. Meng, X.; Karniadakis, G.E. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *J. Comput. Phys.* **2020**, *401*, 109020. [CrossRef]
21. Lee, K.; Carlberg, K.T. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J. Comput. Phys.* **2020**, *404*, 108973. [CrossRef]
22. Jin, X.; Cai, S.; Li, H.; Karniadakis, G.E. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *J. Comput. Phys.* **2021**, *426*, 109951. [CrossRef]
23. Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv* **2020**, arXiv:2003.03485.
24. 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]
25. Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Liu, B.; Bhattacharya, K.; Stuart, A.; Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv* **2020**, arXiv:2010.08895.
26. Morra, P.; Meneveau, C.; Zaki, T.A. ML for fast assimilation of wall-pressure measurements from hypersonic flow over a cone. *Sci. Rep.* **2024**, *14*, 12853. [CrossRef]
27. Zhu, M.; Feng, S.; Lin, Y.; Lu, L. Fourier-DeepONet: Fourier-enhanced deep operator networks for full waveform inversion with improved accuracy, generalizability, and robustness. *Comput. Methods Appl. Mech. Eng.* **2023**, *416*, 116300. [CrossRef]
28. Wen, G.; Li, Z.; Azizzadenesheli, K.; Anandkumar, A.; Benson, S.M. U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Adv. Water Resour.* **2022**, *163*, 104180. [CrossRef]
29. Constantin, P.; Wu, J. Behavior of solutions of 2D quasi-geostrophic equations. *SIAM J. Math. Anal.* **1999**, *30*, 937–948. [CrossRef]

30. Song, F.; Zhang, J.; Wang, J. Convergence analysis and error estimate of second-order implicit–explicit scheme for Gray-Scott model. *Int. J. Comput. Math.* **2021**, *98*, 2330–2340. [CrossRef]
31. Abbaszadeh, M.; Dehghan, M. A reduced order finite difference method for solving space-fractional reaction-diffusion systems: The Gray-Scott model. *Eur. Phys. J. Plus* **2019**, *134*, 620. [CrossRef]
32. Held, I.M.; Pierrehumbert, R.T.; Garner, S.T.; Swanson, K.L. Surface quasi-geostrophic dynamics. *J. Fluid Mech.* **1995**, *282*, 1–20. [CrossRef]
33. Munafo, R.P. Stable localized moving patterns in the 2-D Gray-Scott model. *arXiv* **2014**, arXiv:1501.01990.
34. Kovachki, N.; Lanthaler, S.; Mishra, S. On universal approximation and error bounds for Fourier neural operators. *J. Mach. Learn. Res.* **2021**, *22*, 1–76.