Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2024-0123

Vol. **37**, No. 2, pp. 420-456
February 2025

Approximating Numerical Fluxes Using Fourier Neural Operators for Hyperbolic Conservation Laws

Taeyoung Kim[1] and Myungjoo Kang[1,*]

[1] *Department of Mathematical Sciences, Seoul National University, Seoul 08826, South Korea.*

Received 20 May 2024; Accepted (in revised version) 7 August 2024

Abstract. Traditionally, classical numerical schemes have been employed to solve partial differential equations (PDEs) using computational methods. Recently, neural network-based methods have emerged. Despite these advancements, neural network-based methods, such as physics-informed neural networks (PINNs) and neural operators, exhibit deficiencies in robustness and generalization. To address these issues, numerous studies have integrated classical numerical frameworks with machine learning techniques, incorporating neural networks into parts of traditional numerical methods. In this study, we focus on hyperbolic conservation laws by replacing traditional numerical fluxes with neural operators. To this end, we developed loss functions inspired by established numerical schemes related to conservation laws and approximated numerical fluxes using Fourier neural operators (FNOs). Our experiments demonstrated that our approach combines the strengths of both traditional numerical schemes and FNOs, outperforming standard FNO methods in several respects. For instance, we demonstrate that our method is robust, has resolution invariance, and is feasible as a data-driven method. In particular, our method can make continuous predictions over time and exhibits superior generalization capabilities with out-of-distribution (OOD) samples, which are challenges that existing neural operator methods encounter.

AMS subject classifications: 65M06, 65M08, 68T07, 35L65

Key words: Scientific machine learning, neural operator, FNO, numerical analysis, conservation laws, PDE.

1 Introduction

1.1 Numerical schemes

Throughout the 20th century, with advancements in computer technology, methods of solving numerical problems requiring massive computation have been developed. A

*Corresponding author. *Email addresses:* legend@snu.ac.kr (T. Kim), mkang@snu.ac.kr (M. Kang)

primary focus of this domain is the derivation of numerical solutions for partial differential equations (PDEs). Hence, various methodologies such as the finite difference method (FDM), finite volume method (FVM), and finite element method (FEM) have been formulated. Notably, the FDM and FVM are primarily used to solve computational fluid dynamics (CFD) problems. These include diverse schemes such as the upwind scheme, which calculates derivatives along the flow direction [35]; the Godunov scheme, which resolves the Riemann problem at each time step for time marching [24]; and the Lax–Friedrich scheme, which is essentially a forward-in-time, centered-in-space scheme with an artificial dissipation term [36]. An important mathematical theorem in numerical schemes, Godunov's theorem, states that monotonic schemes cannot exceed the first-order accuracy [24]. Consequently, higher-order schemes exhibit spurious oscillations. To mitigate this, various schemes employing slope/flux limiters [3,4] and methods, such as ENO [5], weighted essentially non-oscillating (WENO) [6], and TENO [11], have been developed.

1.2 Physics-inspired machine learning

In addition to classical numerical approaches, efforts have been made to employ artificial neural networks as surrogate models for PDE solvers. The two main topics in these fields are PINN-type and neural operator-type methods. Remarkable advances in parallel computing, especially in graphical processing units (GPUs), have catalyzed the feasibility of deep learning technologies, thereby drawing substantial attention toward deep learning research. In this context, research on PINNs has attracted increasing interest. PINN-type research has considered [13], [21], [22], and many of their variations. Despite their numerous variations and advancements, Physics-Informed Neural Networks (PINNs) have several limitations. These include the nonconvexity of loss surfaces, as highlighted by [8], the eigenvalue imbalance of losses analyzed via the Neural Tangent Kernel (NTK) approach [12], and challenges in training highlighted by the decay of singular values of solutions, an indicator derived from the Kolmogorov n-width, which suggests difficulties in training PINNs [23]. Moreover, the neural operator domain features methodologies such as DeepONet [14], the Fourier neural operator (FNO) [17], and the graph neural operator (GNO) [16], among other variations [15] [34]. DeepONet is structured through the nonlinear expansion of basis functions, whereas the FNO and GNO approximate the operator's kernel function nonlinearly. FNOs are trained by tuning the weight function in the frequency space, and GNOs are trained by learning the graph kernel matrix. Studies by [14] and [18] have demonstrated the universal approximation properties of DeepONet and FNO. In terms of generalization error, analyses focused on Rademacher complexity have been conducted by [19], [37], and [38]. However, neural operators still face generalization challenges, particularly when inferring OOD samples and repeated inferences.

1.3 Various approaches of combining numerical schemes and neural networks

Efforts to harness the merits of both physics-inspired machine learning methods and classical numerical schemes have led to various innovative methodologies. For instance, differentiable physics [7] integrates numerical solvers and neural networks to enhance the robustness against input variability. [10] combined reinforcement learning and the WENO scheme. Moreover, strategies have been proposed to replace or supplement the numerical limiters or weights of the WENO scheme using neural networks [9, 39]. Studies similar to ours include [2] and [40], which endeavored to construct a Riemann solver via neural networks; and [1], which replaced the numerical flux of the hyperbolic conservation law with a neural network whose input was a stencil.

1.4 Our contribution

This study introduces the flux Fourier neural operator (Flux FNO), a neural operator model designed to predict subsequent states from given initial conditions based on an approximated flux. Unlike previous models, such as that of [1], our model not only encompasses the loss pertaining to time marching, but also incorporates consistency loss, thereby enhancing consistency of approximated flux. Unlike the stencil-based input in [1], our model processes the entire state at each time step. Moreover, because our model is based on FNO, it has the property of resolution invariance, meaning it functions effectively even at resolutions different from those of the training data. The Flux FNO model exhibited notable improvements over existing FNO models, including enhanced generalization capabilities, superior performance with OOD samples, and continuous and long-term prediction capabilities. It retains the advantages of the existing FNO, such as the capacity to learn from experimental values, and achieves computational efficiency through a constant inference time irrespective of the complexity of the approximated numerical flux, thus ensuring computational efficiency compared with more complex numerical schemes. We present algorithms for Flux FNO and have validated their performance through experiments with one-dimensional (1D) inviscid Burgers' equations, 1D linear advection equations, and other types of conservation laws, demonstrating their superior performance and generalization capabilities compared to existing models. Our method integrates seamlessly with complex numerical schemes, including higher-order Runge–Kutta (RK) methods. Additionally, we validated the effectiveness of our loss function using an ablation study.

2 Preliminaries

In this section, we introduce the concepts fundamental and necessary for understanding the method we have designed. First, we explain what hyperbolic conservation laws are, which are the subject we aim to solve, and introduce several representative equations

used in the experiments. Additionally, we discuss the numerical schemes that have motivated our methodology and have been used to generate data. Lastly, we introduce the neural network model, FNO, which serves as a component of our designed methodology.

2.1 Hyperbolic conservation laws

Conservation laws are systems of partial differential equations written in the following form (for 1D case):

$$U_t + F(U)_x = 0,$$

where $U = [u_1, \cdots, u_m]^T$ is a vector of conserved variables, and $F(U) = [f_1, \cdots, f_m]^T$ a vector of fluxes, with the input of each f_i being U. We can write (2.1) in quasi-linear form as follows:

$$U_t + \frac{\partial F(U)}{\partial U}\frac{\partial U}{\partial x} = 0.$$

If the Jacobian $\frac{\partial F(U)}{\partial U}$ has m real eigenvalues and is diagonalizable, we say that (2.1) is hyperbolic. Moreover, if the dimension of the conserved variables is 1 $(m = 1)$, then we say (2.1) is a scalar conservation law. With appropriate initial and boundary conditions, (2.1) composes a hyperbolic conservation law problem. An important problem is the Riemann problem, which involves an initial condition with a single discontinuity. Because the solution of the hyperbolic conservation law can form a discontinuity (shock), a naïve finite-difference scheme does not work, and careful treatment of these shocks is required. This class of hyperbolic conservation laws addresses several scientific and engineering problems, particularly with types of gas dynamics. Here, we present some of these problems.

One of the simplest type of conservation laws is a linear advection equation:

$$u_t + a u_x = 0,$$

where a is a constant. For initial condition $u(x,0) = u_0(x)$, the solution to the linear advection equation is simply a translation of initial condition $u(x,t) = u_0(x - at)$. For a multidimensional linear advection equation, similar to the 1D case, the solutions are simply translations in which the velocity is the coefficient.

Next, the inviscid Burgers' equation is a basic conservation law problem. This problem permits the formation of complex waves at discontinuities, such as shock and rarefaction waves. Therefore, its behavior has often been studied to analyze conservation laws. The governing equation is as follows:

$$u_t + u u_x = 0.$$

Additionally, to verify that our methodology works well with vector-valued problems, we will also consider the 1D shallow water equation. The 1D shallow water equation is

written as follows:

$$H_t+(UH)_x=0,$$
$$(UH)_t+\left(U^2H+\frac{1}{2}gH^2\right)_x=0.$$

Here, H represents the height, U corresponds to the velocity, and UH physically represents the mass velocity. g is the acceleration due to gravity. This equation is used to describe the flow of fluid beneath the pressure surface.

2.2 Numerical schemes

We may use the finite difference method to solve PDEs in the form of (2.1). However, if we use arbitrary FDMs, the numerical result may converge to an incorrect solution when discontinuity exists, and the method may be unstable. Therefore, we must restrict our schemes, which result in correct and stable solutions. A theorem exists regarding certain types of numerical methods that guarantee good quality. These methods are called "conservative methods" and have the following form:

$$U_j^{n+1}=U_j^n-\frac{k}{h}[\hat{F}(U_{j-p}^n,\cdots,U_{j+q}^n)-\hat{F}(U_{j-p-1}^n,\cdots,U_{j+q-1}^n)],$$

where $\hat{F}$ is a function of $p+q+1$ arguments. $\hat{F}$ denotes the numerical flux function. The notation (U_j^n) refers to a discretized function where the superscript indicates the temporal index and the subscript indicates the positional index. h and k represent the space step and time step, respectively. We define the numerical flux as consistent when the following conditions are satisfied:

$$\hat{F}(u,\cdots,u)=F(u),\quad \forall u\in\mathbb{R},$$
$$|\hat{F}(U_{j-p},\cdots,U_{j+q})-F(u)|\leq K\max_{-p\leq i\leq q}|U_{j+i}-u|,$$

where K is the Lipschitz constant of $\hat{F}$. We employ Eqs. (2.2) and (2.2) as loss functions in our model to approximate a consistent numerical flux using the FNO model. Many issues must be solved in numerical schemes, such as obtaining a higher accuracy in time while maintaining the total variation diminishing (TVD) property (Appendix A), handling shocks via a limiter or weighted essentially non-oscillating (WENO) schemes [4,6]. We briefly introduce some of these methods and discuss the combination of our method and these methods, showing the flexibility of our model's implementation.

First, to obtain a higher accuracy in time, we use the following multi-step method, which is called the RK method.

$$U^{(i)}=\Sigma_{k=0}^{i-1}\left(\alpha_{ik}U^{(k)}+\frac{\Delta t\beta_{ik}}{\Delta x}[\hat{F}(U_{j-p}^{(k)},\cdots,U_{j+q}^{(k)})-\hat{F}(U_{j-p-1}^{(k)},\cdots,U_{j+q-1}^{(k)})]\right),$$
$$i=1,\cdots,m,$$
$$U^{(0)}=U^n,\quad U^{n+1}=U^{(m)}.$$

Under suitable conditions for the coefficients α_{ik} and β_{ik} (see CFL conditions in Section A), the RK method is known to exhibit the TVD property [41]. In Sections 3 and 4, we algorithmically demonstrate that our methodology can be integrated with the RK method, and we prove experimentally that it produces better results. By doing so, we show that it can synergize and be compatible with classical methodologies.

Moreover, higher-order linear schemes exhibit oscillations at discontinuities (Godunov's theorem [24]). Thus, limiting methods exist that combine high and low-order fluxes by manipulating the flux coefficient using a function known as a limiter [26]. The formula for a flux function manipulated by a limiter is as follows:

$$F(U;j)=F_L(U;j)+\Phi(U;j)[F_H(U;j)-F_L(U;j)],$$

where $\Phi(U;j)$ is a limiter function, F_H is a high-order numerical flux, and F_L is a low-order numerical flux function. The notation for the flux mentioned implies $F(U;j)=F(U_{j-p},\cdots,U_{j+q})$, and similarly for $F_H(U;j)$ and $F_L(U;j)$. The dataset for the Burgers' equation is generated using a numerical scheme with the minmod limiter, which is defined as follows:

$$\Phi(U;j)=\max\left(0,\min\left(1,\frac{U_j-U_{j-1}}{U_{j+1}-U_j}\right)\right).$$

Since we approximate numerical fluxes using a non-linear neural network function in our methodology, we will be able to approximate the entire flux function with a limiter applied.

2.3 Fourier neural operator

Although various types of neural network models can be used in our designed methodology, we have chosen the Fourier Neural Operator (FNO) as a component of our model. The FNO efficiently handles global processing of functions through the use of Fourier transform convolutions with kernels, and additionally, it can computationally manage both global and local features of operators efficiently by processing local data through an auxiliary network, typically a Convolutional Neural Network (CNN). Our model approaches temporal dimension locally but can address spatial dimensions globally, making the FNO suitable due to its features mentioned. Especially since the global characteristics of solutions are pronounced in analytical solutions or physical observations rather than numerical solvers, we chose the FNO as our foundational model, keeping this in mind. The FNO is a neural network model that can handle functional data. Unlike traditional neural network models that process fixed-size vectors, the FNO can handle arbitrarily vectorized functional data, regardless of its predetermined architecture. As shown in Fig. 1, the FNO consists of a lifting layer, Fourier layers with convolutional neural network (CNN) layers, and a projection layer. The following definitions describe the detailed mathematical formulae of an FNO.

Definition (General FNO). Let $\tilde{D}\subseteq\mathbb{R}^d$ represent the domain for both input and output functions within our problem scope. To analyze these functions computationally, we

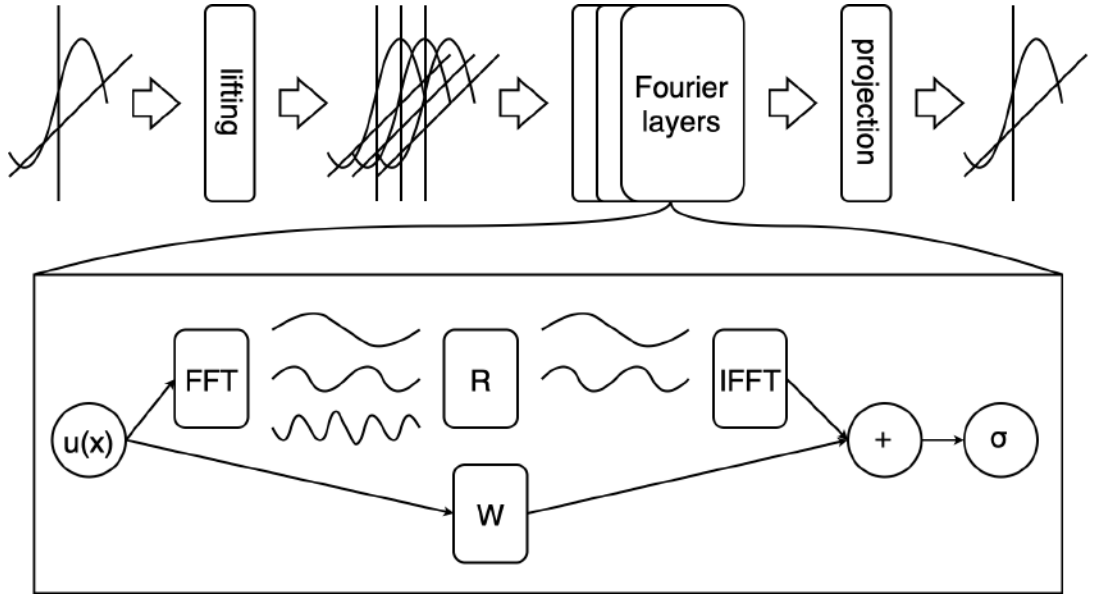

Figure 1: Schematic of 1D FNO.

discretize their respective spaces into $\mathbb{R}^{N\times d_a}$ for input functions and $\mathbb{R}^{N\times d_u}$ for output functions. Here, N denotes the number of grid points within the discretized domain $\tilde{D}$, while d_a and d_u represent the dimensions of the codomains for input and output functions, respectively. Subsequently, FNO $G(a;\theta):\mathbb{R}^{N\times d_a}\to\mathbb{R}^{N\times d_u}$ is defined as follows:

$$G(\cdot;\theta)=\mathcal{N}_Q\circ\mathcal{A}_L\circ\mathcal{A}_{L-1}\cdots\circ\mathcal{A}_1\circ\mathcal{N}_P,$$

where $\mathcal{N}_P$ and $\mathcal{N}_Q$ are neural networks used for lifting and projection. Moreover, each $\mathcal{A}_i$ is a Fourier layer, L represents the depth of the Fourier layers. We assume that $\mathcal{N}_Q$ and $\mathcal{N}_P$ are fully connected networks. Each Fourier layer is a composition of activation functions with a sum of convolutions by a function parametrized by a tensor and other neural networks. Moreover, in the Fourier layers, only a subset of the frequencies is utilized. The frequencies employed in the model are represented by index set $K=\{(k_1,\cdots,k_d)\in\mathbb{Z}^d:|k_j|\le k_{\max,j},\ j=1,\cdots,d\}$. The detailed formula for the FNO is as follows:

$$\begin{aligned}
&v_0:=\mathcal{N}_P(a|_X)=(\mathcal{N}_P(a_{\mathbf{x}\cdot})_j)_{\mathbf{x}\in\mathbf{X},\ j=1,\cdots,d_{v_0}},\\
&v_{t+1}:=\mathcal{A}_{t+1}(v_t)=\sigma\left(A_{t+1}v_t+\mathcal{F}^{-1}\left(R_{t+1}\cdot(\mathcal{F}(v_t))\right)\right),\quad t=0,\cdots,L-1,\\
&G(a;\theta):=\mathcal{N}_Q(v_L)=(\mathcal{N}_Q(v_{L\mathbf{x}\cdot})_j)_{\mathbf{x}\in\mathbf{X},\ j=1,\cdots,d_{v_L}},
\end{aligned}$$

where $a|_X$ is the discretized functional data of a and the lifting and projection layers ($\mathcal{N}_P$ and $\mathcal{N}_Q$) act on the vector value of the discretized functional data and not on the entire vector. σ is an activation function that acts on the output of each neuron. Each $d_{v_i}, i=$

$0,\cdots,L$ represents the dimensionality of the function value at the respective layer. $\mathcal{F}$ and $\mathcal{F}^{-1}$ are discrete Fourier and inverse Fourier transform, respectively. R_t are high-rank tensors representing kernel functions, each with a size of $d_t \times d_t \times k_{\max,1} \times \cdots \times k_{max,d}$. A_t can be any neural network as long as it has a resolution invariance property. We select A_t as the CNN layer because a CNN layer with suitable hyperparameters (stride, padding, kernel size, etc.) can handle arbitrary resolutions. We use the CNN layer as a layer for processing local information, but since a kernel size larger than 1 may exhibit different behavior depending on the resolution, we set the kernel size to 1. For this purpose, we describe the CNN layers as follows.

CNN layer. A kernel tensor of a certain size sweeps across the input tensors such that for each index of the output, an inner product with the kernel and local components of the input tensor centered on that index are produced. For example, for a d-rank input tensor of size $N_1 \times \cdots \times N_d$, we consider a d-rank tensor kernel K of size $c_1 \times \cdots \times c_d$. We denote this CNN layer by kernel $C(c_1 \times \cdots \times c_d)$; the tensor that passes through the CNN layer with kernel K is defined as follows:

$$C(c_1,\cdots,c_d)(x_{x_1\cdots x_d})_{z_1\cdots z_d} = \sum_{j_1=0}^{c_1-1} \cdots \sum_{j_d=0}^{c_d-1} K_{j_1,\cdots,j_d} x_{z_1+j_1,\cdots,z_d+j_d}.$$

Because the positional dimensions of the tensors must be maintained, the CNN layers in our study are restricted to kernels with odd sizes. To fit the dimensions, we added padding to the input tensor of the CNN layer. For example, for $N_1 \times \cdots \times N_d$-dimensional tensors, $x_{x_1\cdots x_d}$, and CNN layer $C(c_1,\cdots,c_d)$, we added padding of $\frac{c_i-1}{2}$ elements to the each sides of input tensor of the CNN layer, denoted as $\tilde{x}$. Then, $C(c_1,\cdots,c_d)(\tilde{x}_{x_1\cdots x_d})$ has the same dimensions as the input tensor. Because we fixed the number of channels in the Fourier layers, for a CNN layer with multiple channels, without confusion, we used the same notation $C(c_1,\cdots,c_d)$, and the detailed formula for such a multi-channel CNN layer can be written as follows:

$$C(c_1,\cdots,c_d)(x_{x_1\cdots x_d})_{z_1\cdots z_d j} = \sum_{k=1}^{d_u} \sum_{j_1=0}^{c_1-1} \cdots \sum_{j_d=0}^{c_d-1} K_{j_1,\cdots,j_d,k,j} x_{z_1+j_1,\cdots,z_d+j_d,k}.$$

Definition (FNO with CNN layer). Consider the same settings and notations as in the previously defined general FNO; the only difference is that the Fourier layer now consists of CNN layer and convolution with a parameterized function

$$\begin{aligned} v_{t+1} :=& \mathcal{A}_{t+1}(v_t) = \sigma\Big(C_{t+1}(c_1,\cdots,c_d)(\tilde{v}_t) + \mathcal{F}^{-1}\Big(R_{t+1}\cdot(\mathcal{F}(v_t))\Big)\Big), \\ =& \sigma\Big(\sum_{k=1}^{d_u} \sum_{j_1=0}^{c_1-1} \cdots \sum_{j_d=0}^{c_d-1} K_{t+1,jk,j_1,\cdots,j_d} \tilde{v}_{t,x_1+j_1,\cdots,x_d+j_d,k} \\ &+ \sum_{\mathbf{z},\mathbf{k}\in K,k} D^{\dagger}_{\mathbf{xk}} R_{t+1,\mathbf{k},jk} D_{\mathbf{kz}} v_{t,\mathbf{z}k} \Big). \end{aligned}$$

$D_{\mathbf{kz}}$ represents the components of the discrete Fourier transform. For a detailed description of the FNO model, see [16], and to understand the CNN layer, [42].

3 Algorithms for Flux Fourier Neural Operator

In this section, we propose a novel loss function for the Flux FNO, which is pivotal to our study. Subsequently, we describe the algorithms used for training and inference. For different numerical schemes that adapt the Flux FNO, slight modifications to our loss function or algorithms are necessary, which we also discuss. For ease of implementation, we assumed periodic boundary conditions.

Definition (Loss function). Motivated by (2.2), we constructed the following loss function for the one-step time-marching method

$$\mathcal{L}_{tm}(U)=\Sigma_{n=0}^{N}\|U^{n+1}-U^{n}+\frac{t_n}{k}[G(U^{n}_{-p},\cdots,U^{n}_{+q};\theta)-G(U^{n}_{-p-1},\cdots,U^{n}_{+q-1};\theta)]\|_2^2.$$

Here, $G(V_1,\cdots,V_{p+q+1};\theta)$ represents an FNO model with parameter θ, and each U^n,U^{n+1},U^n_i denotes vectorized functional data. U denotes all functional data over the time interval. U^n_i indicates that the vector components have been shifted by i. t_n represents the discretization spacing of the time variable for each step. Furthermore, we consider another loss inspired by (2.2)

$$\mathcal{L}_{consi}(U)=\sum_{n=0}^{N}\|G(U^n,\cdots,U^n;\theta)-F(U^n)\|_2^2.$$

Here, F denotes a physical flux in the conservation law, which is different from the numerical flux function $\hat{F}$ from (2.2). The physical flux function is used when defining Eq. (2.1). By combining $\mathcal{L}_{tm}$ and $\mathcal{L}_{consi}$, we train our model based on the subsequent loss for a given dataset of functions $\{U_i\}_{i=1,\cdots,m}$:

$$\mathcal{L}(\{U_i\},G(\cdot;\theta))=\Sigma_{i=1}^{m}(\mathcal{L}_{tm}(U_i)+\lambda\mathcal{L}_{consi}(U_i)),\quad 0\leq\lambda.$$

We now describe the training and inference algorithms as indicated in Algorithms 1 and 2. Fig. 2 shows the inference pipeline. Initially, we introduce an algorithm for a scalar conservation law within a 1D periodic domain using a first-order time step. For multidimensional cases, add the flux for each spatial axis. Here, the functional data are structured as [batch size, N_t, N_x, 1], where N_t and N_x represent the number of grid points in time and space, respectively. Notably, each batch contains the continuous evolution of function, rather than randomly chosen function snapshots. Next, we explain the algorithms now integrated with the RK method in Algorithms 3 and 4. Although not shown in the algorithm, designing the input dimension of the FNO model's lifting layer to consider the concatenated vector dimensions (in this case, $p+q+1$ dimensions. The additional 1 is for the positional embedding of the domain.) is necessary. For vector-valued

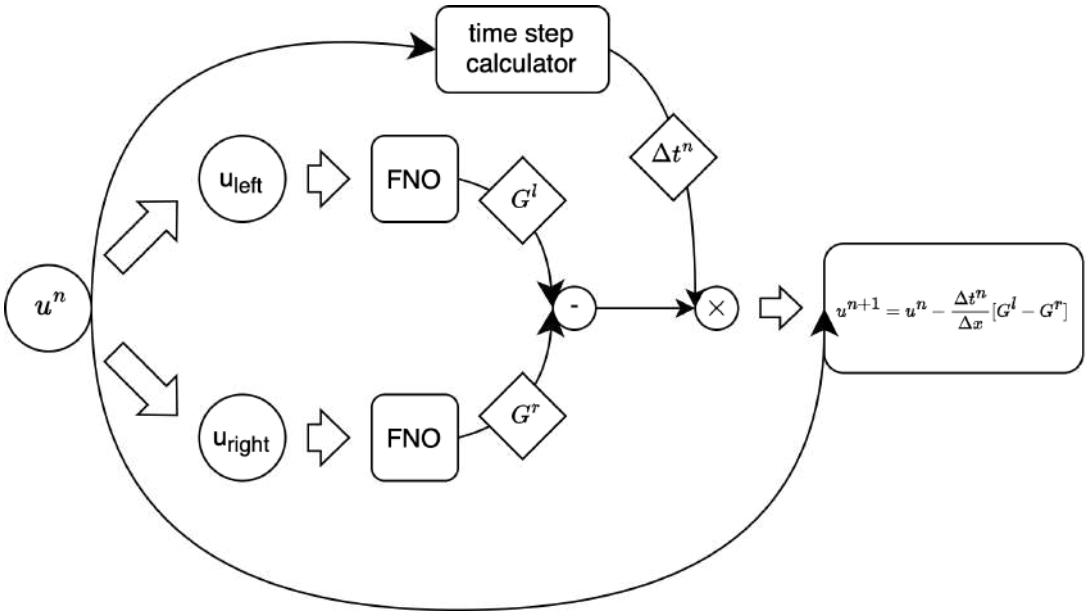

Figure 2: Schematic of the inference structure for flux Fourier neural operator (Flux FNO).

cases, the algorithm remains identical, except that for an N-dimensional vector-valued function, the concatenated function would have dimensions $N(p+q)$. It's important to note that neural networks can approximate nonlinear functions, so numerical fluxes that incorporate nonlinear schemes like Flux limiters or WENO can also be approximated end-to-end. Certainly, combining methods such as flux-limiters and WENO with our approach is also conceivable.

Furthermore, we present the following theorem to estimate the inference of the Flux FNO. This theorem combines the statistical attributes of the neural network model and properties of classical numerical theory. Notably, the actual experimental results demonstrate a better performance than the following theorem (e.g., robustness to OOD samples, stability, and faster convergence rate). For future work, it is possible to consider research on the estimation of out-of-distribution (OOD) samples or stability analysis of approximated flux. In the theorem, we clarify assumptions about the dataset. Suppose $D\big|_{t=0}$ is a dataset composed of m functions generated from the distribution $\mathcal{D}$. Then, let $D\big|_{t=\tilde{t}}$ be a dataset composed of m functions which are solutions at time $t=\tilde{t}$ to the initial conditions for each corresponding data point in $D\big|_{t=0}$. Since each data point in $D\big|_{t=\tilde{t}}$ is independent, we can treat them as i.i.d. samples from a hypothetical distribution.

Theorem 3.1 (Error Estimation for Inference of Flux FNO)**.** *Assume that an FNO model $G(\cdot;\theta)$ is trained on the dataset $D=\bigcup_{i=0}^{T} D\big|_{t=t_i}, t_0=0$. where $D\big|_{t=0}$ consists of m functions generated using the distribution $\mathcal{D}$. Furthermore, the support of each hypothetical distribution for $D\big|_{t=t_k}$ is a compact ball with a radius of B under the norm $\|\cdot\|_{p*}$. The approximation errors,*

$\epsilon_{tm}^{t_k}$ for loss $\mathcal{L}_{tm}$ and $\epsilon_{consi}^{t_k}$ for $\mathcal{L}_{consi}$ on $D\big|_{t=t_k}$, are evaluated (note that these are not relative L^2 losses but L^2 losses, and the approximation errors considered only for $D\big|_{t=t_k}$). For the vectorized function set $\{U^0,\cdots,U^T\}$, whose initial conditions are generated from the distribution $\mathcal{D}$, the following estimation error is probable at a minimum of $1-\delta$, where U^{k+1} represents data from the distribution, $\tilde{U}^{k+1}$ is inferred by the Flux FNO, and h is the time-step interval.

$$\begin{aligned}
&\|U^{k+1}-\tilde{U}^{k+1}\|_2^2\\
\leq&\min\Bigg(C_3\gamma\frac{\epsilon_{tm}^{t_k}B}{\sqrt{m}}+\epsilon_{tm}^{t_k2}\Big(1+\sqrt{\frac{2log(\frac{4}{\delta})}{m}}\Big),\\
&\quad h\Big(C_1\gamma\frac{\epsilon_{consi}^{t_k}B}{\sqrt{m}}+\epsilon_{consi}^{t_k2}\Big(1+\sqrt{\frac{2log(\frac{4}{\delta})}{m}}\Big)+C_2\epsilon(h)\Big)\Bigg).
\end{aligned}$$

Here, $\gamma=\gamma_{p,q}(G(\cdot;\theta))$ represents the capacity of the model $G(\cdot;\theta)$, as defined by Kim (2024). This theorem implies that the convergence of the scheme with an increasing dataset size and a decreasing time-step interval h. It can be inferred from the proof of the theorem, if the solution exhibits good regularity, $\epsilon(h)$, which depends on the solution, decreases rapidly as h decreases. The Appendix provides a detailed explanation of this theorem and its associated definitions.

Algorithm 1 An algorithm for training (Basic case)

Input: Dataset $\mathcal{U}=((U_{b,i,j,1}),(\Delta t_{b,i}))$
Output: trained FNO model $G(\cdot;\theta)$

for `epoch`$=1,\cdots,E$ **do**
  for `Batch ∈ Train loader` **do**
    $\tilde{U}^n_{-\tilde{j}}\leftarrow$ roll $U_{b,n,\cdot,1}$ by $\tilde{j}$ in the third index for $\tilde{j}=-q,\cdots,p+1$
    $U^l\leftarrow$ concatenate $(\tilde{U}^n_{-p},\cdots,\tilde{U}^n_q)$ along fourth index
    $U^r\leftarrow$ concatenate $(\tilde{U}^n_{-p-1},\cdots,\tilde{U}^n_{q-1})$ along fourth index ▷ Thus, the concatenated function is now a (p+q)-dimensional vector-valued function
    $\mathcal{L}_{tm}(\text{Batch})\leftarrow\sum_{b=1}^{B}\sum_{n=0}^{N_l-1}\|U_{b,n+1,\cdot,\cdot}-U_{b,n,\cdot,\cdot}+\frac{\Delta t_{b,n}}{k}[G(U^l;\theta)-G(U^r;\theta)]_{b,n,\cdot,\cdot}\|_2^2$ ▷ N_t is the number of grid points along time axis and B is batch size.
    $V^{p+q}\leftarrow$ concatenate U p+q times.
    $\mathcal{L}_{consi}(\text{Batch})\leftarrow\sum_{b=1}^{B}\sum_{n=0}^{N_l-1}\|G(V^{p+q}_{b,n,\cdot,\cdot};\theta)-F(U_{b,n,\cdot,\cdot})\|_2^2$
    Calculate backpropagation for $\mathcal{L}_{tm}(U)+\lambda\mathcal{L}_{consi}(U)$ ▷ λ is a weight for the loss.
    Take an optimization step.
  end for
end for

Algorithm 2 An algorithm for inference (Basic case)

Input: FNO model $G(\cdot;\theta)$, initial condition U_0, and target time T
Output: $u(x,T)|_{\mathbf{x}}$

$t \leftarrow 0$
$U \leftarrow U_0$
while $t < T$ **do**
  Calculate Δt according to based numerical scheme.
  if $t+\Delta t > T$ **then**
    $\Delta t \leftarrow T-t$
  end if
  $U \leftarrow U - \frac{\Delta t}{k}[G(U^l;\theta) - G(U^r;\theta)]$ ▷ U^l and U^r are constructed in the same manner as in Algorithm 1
  $t \leftarrow t+\Delta t$
end while

Algorithm 3 An algorithm for training (Combined with TVD-RK)

Input: Dataset $\mathcal{U} = ((U_{b,i,j,1}),(\Delta t_{b,i}))$
Output: trained FNO model $G(\cdot;\theta)$

for `epoch`$=1,\cdots,E$ **do**
  for `Batch ∈ Train loader` **do**
    $\tilde{U}^n_{-\tilde{j}} \leftarrow$ roll $U_{b,n,\cdot,1}$ by $\tilde{j}$ in the third index for $\tilde{j} = -q,\cdots,p+1$
    $U^l \leftarrow$ concatenate $(\tilde{U}^n_{-p},\cdots,\tilde{U}^n_q)$ along fourth index
    $U^r \leftarrow$ concatenate $(\tilde{U}^n_{-p-1},\cdots,\tilde{U}^n_{q-1})$ along fourth index ▷ Thus, the concatenated function is now a p+q dimension vector-valued
    $U^0 \leftarrow U$
    for $\tilde{k} = 1,\cdots,l$ **do**
      $\hat{U}^{\tilde{k}} \leftarrow \frac{1}{\tilde{k}}[G(U^{\tilde{k},l};\theta) - G(U^{\tilde{k},r};\theta)]$
      $U^{\tilde{k},\cdot} \leftarrow \Sigma_{s=0}^{\tilde{k}-1}(\alpha_{\tilde{k}s}U^{s,\cdot} + \Delta t_{b,n}\beta_{\tilde{k}s}\hat{U}^{s,\cdot})$ ▷ Each α and β are selected to satisfy the CFL condition
    end for
    $\mathcal{L}_{tm}(\text{Batch}) \leftarrow \sum_{b=1}^{B}\sum_{n=0}^{N_t-1}\|U_{b,n+1,\cdot,\cdot} - U^l_{b,n,\cdot,\cdot}\|_2^2$
    $V^{p+q} \leftarrow$ concatenate U p+q times.
    $\mathcal{L}_{consi}(\text{Batch}) \leftarrow \sum\sum_{b=1}^{B}\sum_{n=0}^{N_t-1}\|G(V^{p+q}_{b,n,\cdot,\cdot};\theta) - F(U_{b,n,\cdot,\cdot})\|_2^2$
    Calculate backpropagation for $\mathcal{L}_{tm}(U) + \lambda\mathcal{L}_{consi}(U)$
    Take an optimization step.
  end for
end for

Algorithm 4 An algorithm for inference (Combined with TVD-RK)

Input: FNO model $G(\cdot;\theta)$, initial condition U_0, and target time T
Output: $u(x,T)|_{\mathbf{x}}$

$t \leftarrow 0$
$U \leftarrow U_0$
while $t < T$ **do**
  Calculate Δt according to based numerical scheme.
  if $t+\Delta t > T$ **then**
    $\Delta t \leftarrow T-t$
  end if
  for $\tilde{k}=1,\cdots,l$ **do**
    $\hat{U}^{\tilde{k}} \leftarrow \frac{1}{\tilde{k}}[G(U^{\tilde{k},l};\theta)-G(U^{\tilde{k},r};\theta)]$
    $U^{\tilde{k},\cdot} \leftarrow \Sigma_{s=0}^{\tilde{k}-1}(\alpha_{\tilde{k}s}U^{s,\cdot}+\Delta t\beta_{\tilde{k}s}\hat{U}^{s,\cdot})$ ▷ Each α and β are selected to satisfy the CFL condition
  end for
  $U \leftarrow U^l$
  $t \leftarrow t+\Delta t$
end while

4 Experiments

We conducted experiments based on 1D linear advection and inviscid Burgers' equations, which are fundamental hyperbolic conservation laws. First, to demonstrate the robustness of our method compared with existing neural operator methods, we compared the results of our method with those of existing FNO models for long-term prediction tasks and inferences on out-of-distribution (OOD) samples. Second, to demonstrate the compatibility of our method with classical schemes, we combined our method with a more complex scheme and obtained positive results. Additionally, we conducted experiments on the 1D shallow water equations, which are vector-valued problems extending beyond scalar conservation laws, and we also conducted experiments on the 2D linear advection equation to demonstrate the effectiveness of our methodology in higher dimensions. Finally, we conducted an ablation study on the loss function, omitting the consistency loss, to verify its significance in performance enhancement. As explained in Section 3, for 1D problems, the data is in the form of [batch size,N_t,N_x,1], where N_t and N_x represent the number of grid points along the time and spatial axes, respectively. For vector-valued shallow water equation, the data takes the form of [batch size,N_t,N_x,2], where the last index is now 2, corresponding to the number of physical states. In the case of the 2D-linear advection problem, the data is in the form of [batch size,N_t,N_x,N_y,1], with the problem becoming two-dimensional and N_y being added as an index representing the number of grid points along the y-axis.

4.1 Long-term continuous inference

Dataset Specification. For our experiments, we generated two types of training datasets governed by 1D linear advection and 1D Burgers' equations. The governing equations are as follows:

$$\frac{\partial u}{\partial t}+c\frac{\partial u}{\partial x}=0,$$
$$\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}=0.$$

The first equation represents the 1D linear advection equation. We selected $c=1$; therefore, the solution translates to the right with equal time and position scales. The second equation is the 1D Burgers' equation. We used the same Gaussian random field (GRF) generator as the initial condition for both problems. Input functions, which are the initial conditions of the problems, were generated using the GRF with covariance function $k(x,y)=e^{-100(x-y)^2}$ and discretized into a 256-dimensional vector. The exact solutions to the linear advection equations are simple translations. Therefore, we constructed a solution for linear advection from time 0 to 1 with $\Delta t=\Delta x$. We employed a second-order Godunov-type scheme with a second-order RK method for time marching and a minmod limiter for generating the solutions of Burgers' equation. Typically, a time-adaptive method is used for the numerical scheme of conservation laws owing to shocks. However, because the original FNO results are merely snapshots of functions, we used constant time intervals for the comparison with our method, provided that the characteristic lines did not collide. The training dataset of the linear advection case comprised 100 functions in $C^\infty([0,1]\times[0,1])$ and that of the Burgers' equation case comprised 10 functions in $L^2([0,0.3]\times[0,1])$. For the test dataset, each dataset comprised 10 functions drawn from the same distribution, and each function belonged to $C^\infty([0,5]\times[0,1])$ and $L^2([0,0.6]\times[0,1])$, respectively. Table 1 summarizes the dimensions of the datasets.

Table 1: Specifications of training and testing datasets.

	$(\Delta t,\Delta x)$	Number of functions	Domain of function	Overall shape of dataset
Training dataset for linear advection	$(2^{-8},2^{-8})$	100	$[0,1]\times[0,1]$	[100,256,256,1]
Test dataset for linear advection	$(2^{-8},2^{-8})$	10	$[0,5]\times[0,1]$	[10,1280,256,1]
Training dataset for Burgers	$(10^{-2}2^{-8},2^{-8})$	10	$[0,0.3]\times[0,1]$	[760,100,256,1]
Test dataset for Burgers	$(10^{-2}2^{-8},2^{-8})$	10	$[0,0.6]\times[0,1]$	[10,1520,256,1]

Table 2: Specification of architectures and hyperparameters.

	width	depth of Fourier layers	Number of modes	Batch size (Advection, Burgers)
Flux FNO	64	1	5	(1, 1)
1D FNO	64	1	5	(1, 1)
1D FNO(heavy)	32	3	20	(1, 1)
2D FNO	64	3	(10, 10)	(10, 10)

Architecture and Hyperparameters of FNOs. For the Flux FNO, we used a basic FNO combined with CNN layers. The architecture of the model included a maximum frequency of 5, width of 64, and depth of 1. All architectures of FNOs we used had fixed common points: the lifting layer, which is a one-layer FCN; projection layer, which is a two-layer FCN; GELU activation function; and a CNN with a kernel size of 1. The training environments were identical. We used the Adam optimizer with a learning rate of 1e-1 and weight decay of 1e-3, and the scheduler was StepLR with a step size of 50 and gamma of 0.5. The total number of epochs was 1000, and the λ for the loss was selected to be 0.01 throughout all experiments. We compared our method with existing 1D and 2D FNO models. In the 1D FNO case, we trained two architectures. One model was the same as the one used in Flux FNO, while the other was a heavier model. The 1D FNO is designed to receive an input function and locally output the solution for the next Δt immediately. Because 2D FNOs use 2D functional data as input, the training dataset for the 2D FNO was slightly modified. We adjusted the training dataset for linear advection to [99,256,256,1] for each input and target function (with the target function shifted by 1 in the first index), and for Burgers to [759,100,256,1]. Thus, the 2D FNO takes a function over a fixed time interval and outputs a snapshot of the function for the next time interval of the same length, advancing progressively. Table 2 summarizes the architectures and hyperparameters.

Comparison of Results. Figs. 3, 4, 5, and 6 and Tables 3 and 4 qualitatively and quantitatively show the performance of each model. Although all models were trained on the same datasets, the regular 1D and 2D FNO showed significantly poor performances compared to Flux FNO, even within the time interval ($0 \le t \le 0.3$ for Burgers, $0 \le t \le 1$ for advection) of the training samples. By contrast, for Flux FNO, we proposed an inference well over the entire time interval ($0 \le t \le 0.6$ for Burgers, $0 \le t \le 5$ for advection) of the test samples. Note that Flux FNO performed well despite the light architecture of the neural networks. The performance of the 2D FNO is somewhat better than that of the 1D FNO, primarily because the vanilla FNO is well-suited for approximating distributions across a broad function space. In our experiments, the 1D FNO must contend with distributions of functions that dynamically change at each time step. This variability leads to poor training convergence and an inability to effectively learn specific distributions.

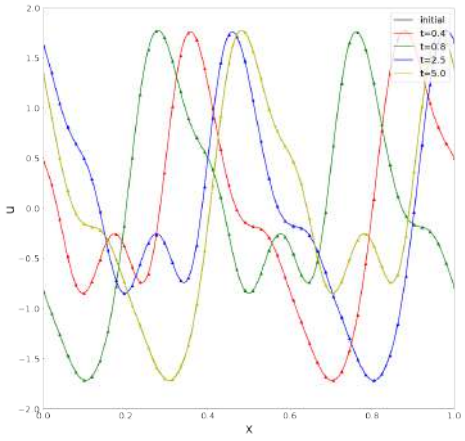

Figure 3: Output of Flux FNO (dashed line with triangle markers) compared with the exact solutions (solid line) for the 1D linear advection problem.

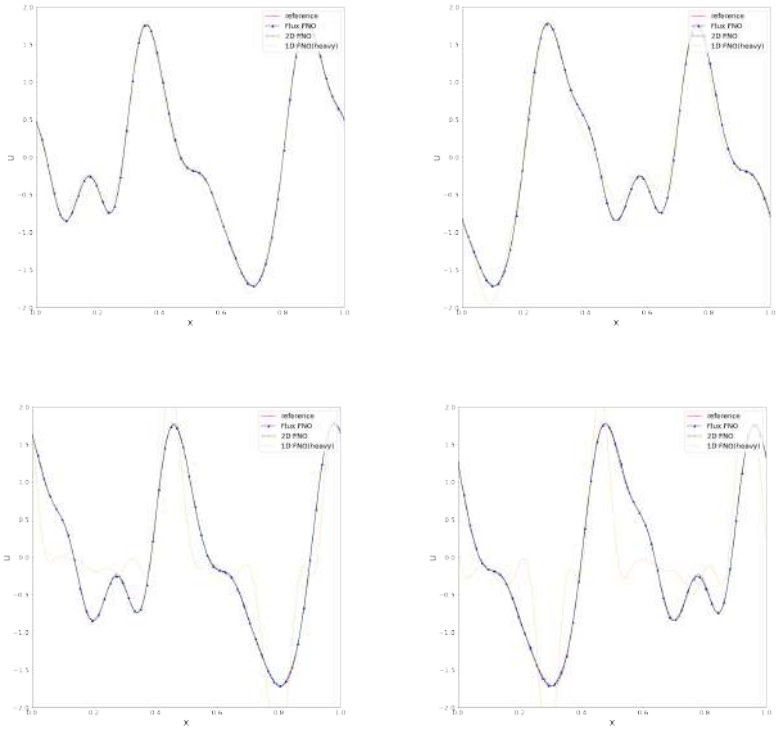

Figure 4: Comparison of Flux FNO output with exact solutions and other FNO models at $t=0.4$ (top left), $t=0.8$ (top right), $t=2.5$ (bottom left), and $t=5.0$ (bottom right) for the 1D linear advection problem.

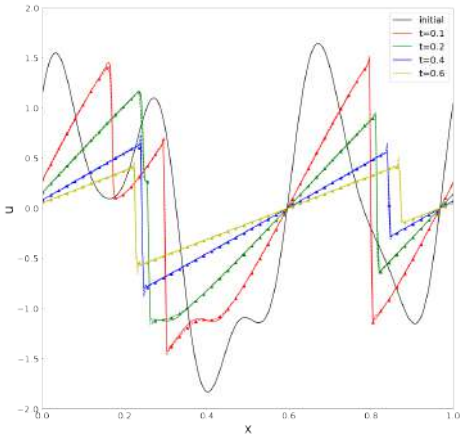

Figure 5: Output of Flux FNO (dashed line with triangle markers) compared with the exact solutions (solid line) for the 1D Burgers' equation problem.

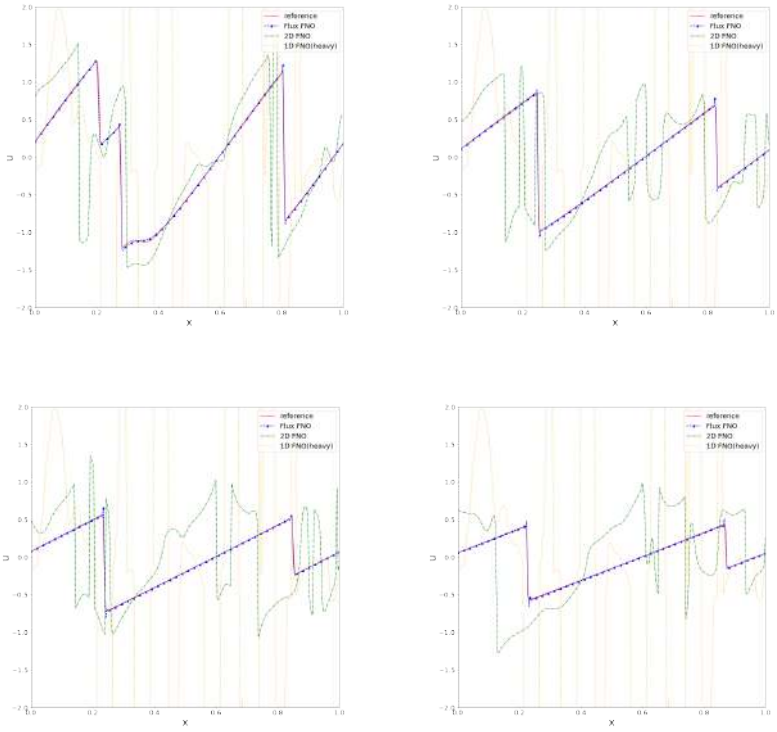

Figure 6: Comparison of Flux FNO output with the exact solutions and other FNO models at $t=0.15$ (top left), $t=0.30$ (top right), $t=0.45$ (bottom left), and $t=0.60$ (bottom right) for the 1D Burgers' equation problem.

Table 3: Quantitative results of each model for the 1D linear advection problem. Each value represents the mean over the test dataset.

(relative L^2, L^∞)	$t=0.4$	$t=0.8$	$t=2.5$	$t=5.0$	On $[0,1]\times[0,0.6]$
Flux FNO	**(8.25e-4, 3.17e-3)**	**(1.67e-3, 6.37e-3)**	**(5.18e-3, 2.02e-2)**	**(1.04e-2, 4.53e-2)**	**(1.34e-2, 4.53e-2)**
1D FNO(heavy)	(7.80e-2, 1.73e-1)	(1.68e-1, 3.54e-1)	(4.87e-1, 1.02)	(6.25e-1, 1.19)	(1.02, 1.21)
1D FNO	(5.46e-1, 1.07)	(1.84, 2.19)	(3.67, 3.17)	(4.27, 3.28)	(8.06, 4.67)
2D FNO	(6.17e-3, 1.24e-2)	(6.82e-3, 1.44e-2)	(1.32e-2, 2.76e-2)	(1.94e-2, 4.67e-2)	(2.84e-2, 5.17e-2)

Table 4: Quantitative results of each model for the 1D Burgers' equation problem. Each value represents the mean over the test dataset.

(relative L^2, L^∞)	$t=0.15$	$t=0.30$	$t=0.45$	$t=0.60$	On $[0,1]\times[0,0.6]$
Flux FNO	**(0.048, 0.30)**	**(0.049, 0.21)**	**(0.051, 0.15)**	**(0.052, 0.13)**	**(0.040, 0.48)**
1D FNO(heavy)	(4.68, 7.85)	(6.55, 7.53)	(8.86, 7.31)	(11.20, 7.32)	(5.39, 8.66)
1D FNO	(5.07, 4.89)	(10.08, 4.94)	(15.02, 4.92)	(19.09, 4.81)	(7.78, 5.35)
2D FNO	(0.93, 2.24)	(1.36, 1.65)	(1.88, 1.43)	(2.21, 1.26)	(1.08, 3.28)

4.2 Generalization ability

In this section, we describe the experiment with OOD samples. Throughout this section, we use the model trained in Section 4.1 and an additional 1D FNO model trained with different datasets for a fair comparison with our method. First, we solve for the initial condition function, which is simple but has a cusp or discontinuity. We solve the initial OOD condition of the function generated using the GRF with more fluctuations. We then demonstrate the resolution invariance of our method using initial conditions with different grid points.

Inferences from OOD samples. We made inferences on OOD samples to show that our model has a generalization ability. For the linear advection cases, two types of initial

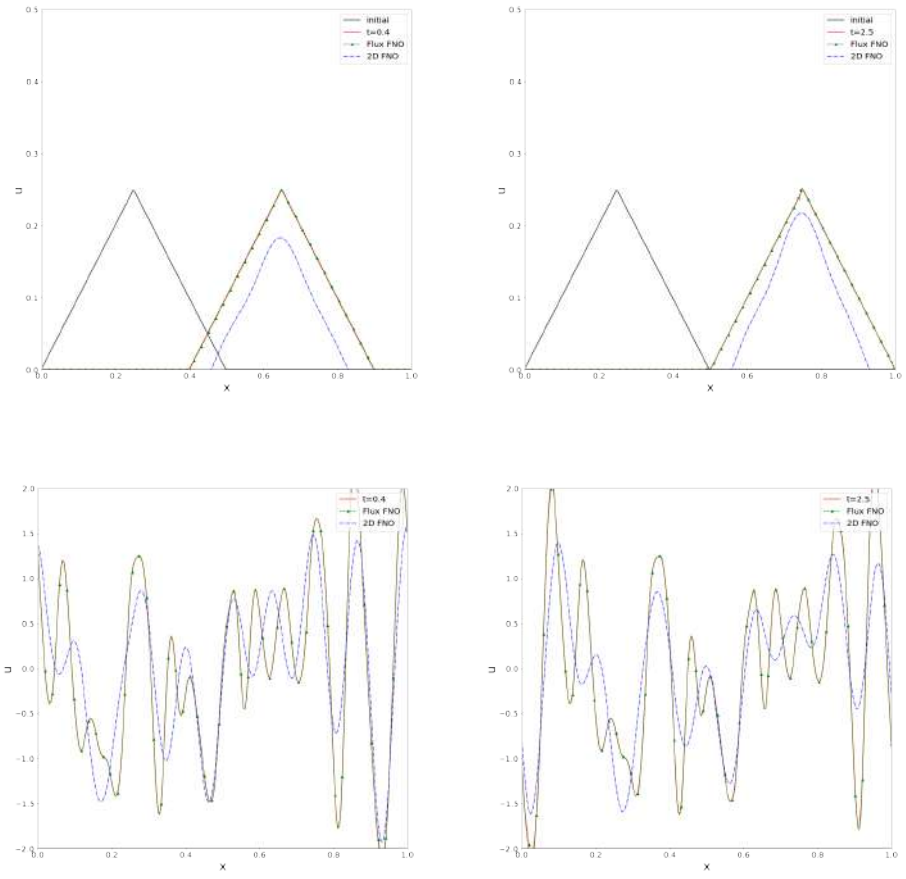

Figure 7: Inference of Flux FNO on out-of-distribution samples for the 1D linear advection problem: triangular pulse (top) and GRF with different covariance (bottom).

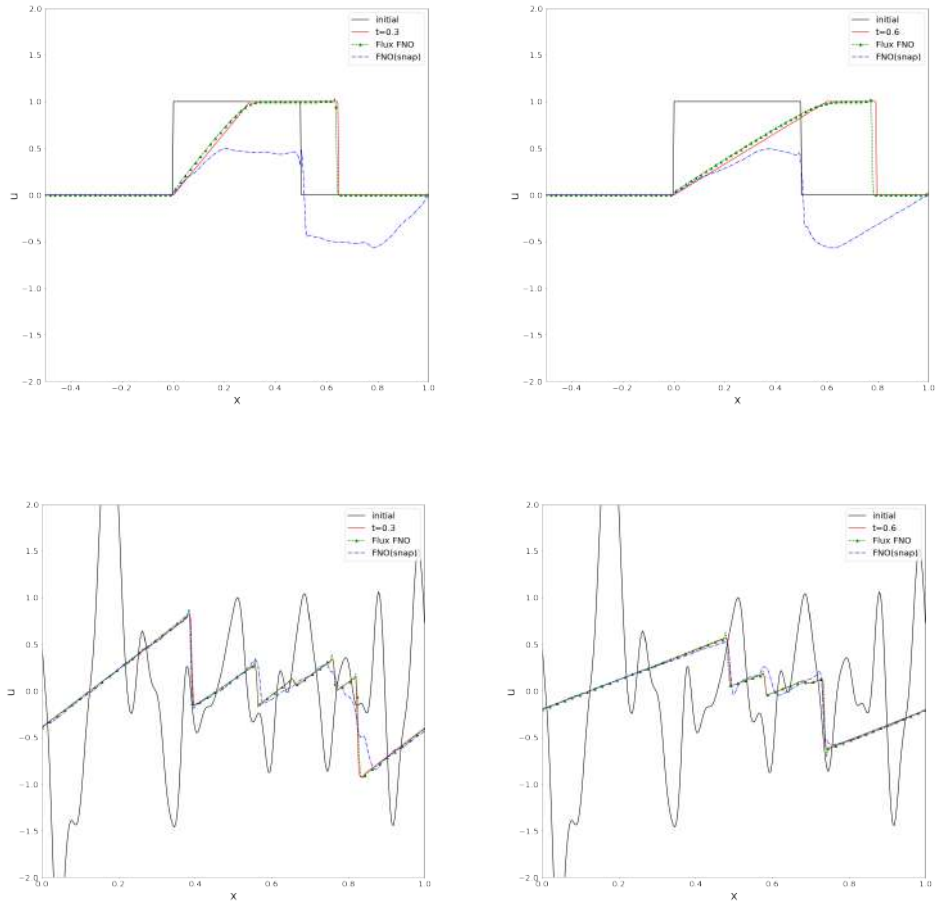

Figure 8: Inference of Flux FNO on out-of-distribution samples for the 1D Burgers equation problem: square wave (top) and GRF with different covariance (bottom).

conditions exist. One is a triangular pulse, and the other is a function generated using the GRF with more fluctuations (the covariance function is $e^{(\frac{x-y}{0.03})^2}$). We compared this with the 2D FNO model, which showed the best performance among all the comparison groups. We also considered two types of initial conditions for the Burgers' equation case. One is a square wave, in which the exact solution can be solved, and the GRF, similar to the advection case, is used to generate the other. For comparison, we did not consider models from the comparison group in Section 4.1, because of the poor performances of all models. Because the FNO may perform poorly on continuous and repeated inferences, particularly for nonlinear cases, we trained an additional FNO model. We constructed a dataset such that the input functions are similarly generated, but the target functions were fixed on time as a solution function at time $t=0.3$ for the Burgers case. After training

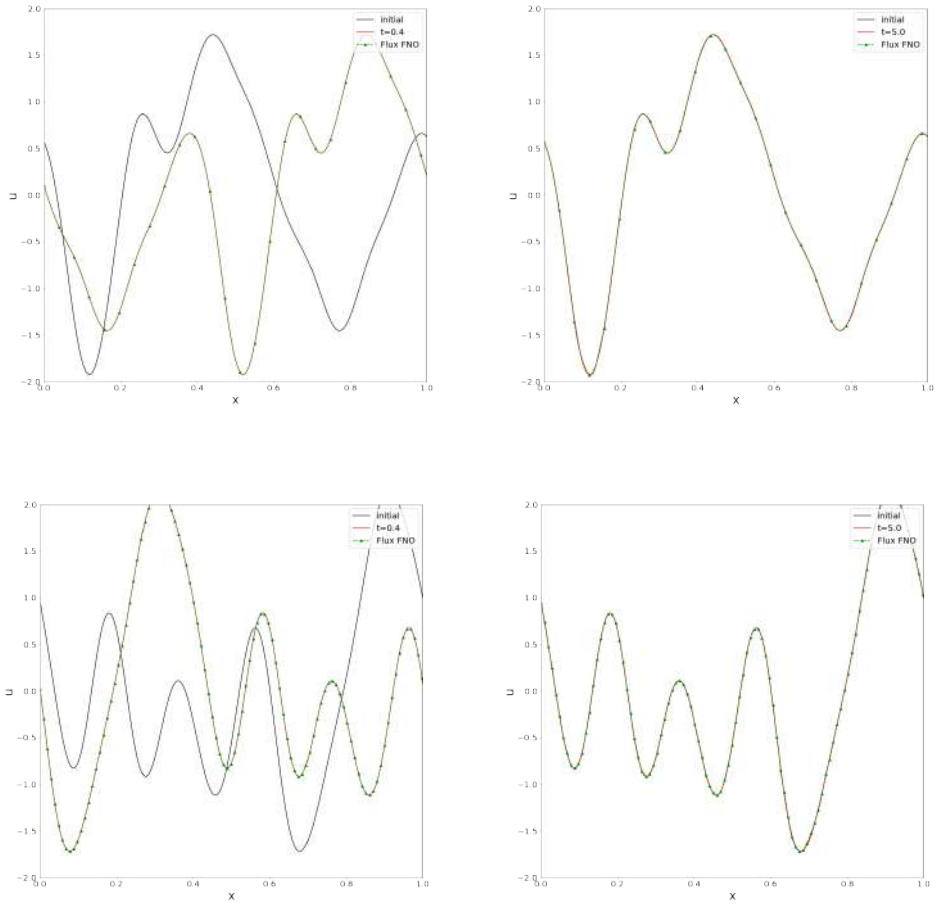

Figure 9: Inference of Flux FNO on different resolution samples for the 1D linear advection problem: resolution of 128 (top) and resolution of 512 (bottom).

a regular FNO (FNO(snap)) on this dataset, we infer twice using this FNO (snap). As shown in Figs. 7 and 8, the generalization ability of our method for OOD samples is much better than that of existing models.

Inferences at different resolutions. We show that our model can handle different resolutions, essentially inheriting the original FNO's property. We sampled the initial condition with the GRF of the same covariance as the training distribution. However, the numbers of nodes were 128 and 512, respectively. Figs. 9 and 10 show that our method is consistent across different resolutions. Therefore, our method is superior to those that approximate the function from a stencil to the flux value, which require the training of another network for different resolutions.

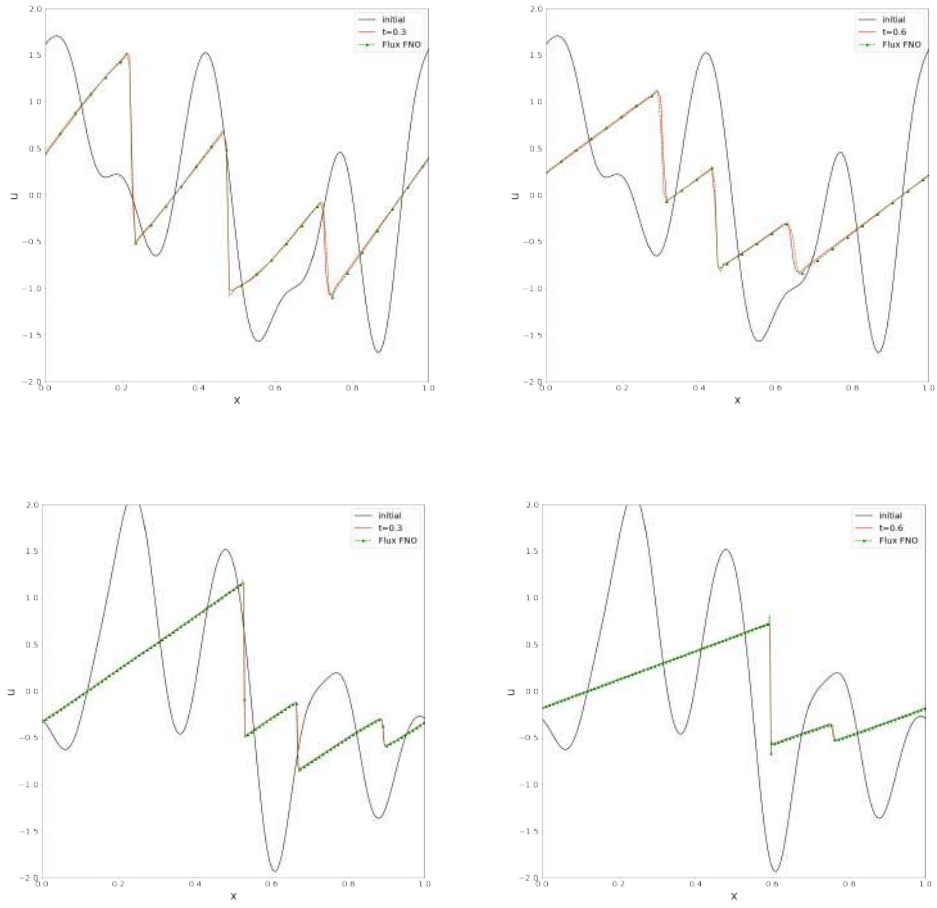

Figure 10: Inference of Flux FNO on different resolution samples for the 1D Burgers' equation problem: resolution 128 (top) and resolution 512 (bottom).

4.3 Combined with 2nd order Runge-Kutta (RK2) method

We conducted experiments to show that our method can be combined with more complex numerical schemes. We selected RK2, which improves the order in the time dimension. We trained our model following Algorithm 3. The architecture and training environment settings were the same as those of the Burgers equation settings in Section 4.1. The training converged well, and Table 5 presents the performance of the model. As shown in Table 5, Flux FNO with RK2 exhibited a better performance. We anticipate that more high-order RK methods will be compatible with our methods, and that methods such as the limiter and WENO can be combined compatibly.

Table 5: Results of baseline and Flux FNO combined with the RK2 method. Each value represents the mean over the test dataset.

relative L^2	$t=0.15$	$t=0.30$	$t=0.45$	$t=0.60$	On entire time interval
Flux FNO	0.048	0.049	0.051	0.052	0.040
Flux FNO with RK2	**0.046**	**0.043**	**0.042**	**0.041**	**0.037**

4.4 Results on other conservation problems

In this section, we experimentally verify the effectiveness of our methodology for vector-valued, multi-dimensional cases. Experiments were conducted for the 1D shallow water equation and the 2D linear advection equation, testing performance quantitatively and qualitatively in each scenario.

4.4.1 1D shallow water equation

As mentioned in Section 2.1, the 1D shallow water equation is described by:

$$\begin{aligned}&\frac{\partial H}{\partial t}+\frac{\partial(UH)}{\partial x}=0,\\&\frac{\partial(UH)}{\partial t}+\frac{\partial\left(U^2H+\frac{1}{2}gH^2\right)}{\partial x}=0.\end{aligned}$$

We have generated the training dataset assuming periodic boundary conditions for this equation. The initial conditions are set as $H_0=0.5+0.02a_0$, $(UH)_0=0.1a_1$, where a_0,a_1 are sampled from the same Gaussian random field as the training datasets in Section 4.1. The numerical solution was obtained using a combination of the Lax–Friedrichs and Lax–Wendroff schemes with a minmod limiter, and the function's domain is $[0,0.1]\times[0,1]$. The spatial domain is discretized at $\Delta x=2^{-8}$, and the temporal domain is discretized at $\Delta t=0.1\Delta x$. The various hyperparameters and architecture are almost identical to those described in Section 4.1, with the only difference being that the output of the FNO model is now 2-dimensional vector value. the shape of the training dataset is $[1000,256,256,2]$ and for training, we choose a batch size of 1. The result of the Flux FNO for a test sample (which was sampled from the same distribution as the training dataset) is demonstrated in Figs. 11 and 13. As you can see, our model not only accurately predicts over the learned functions' time domain $[0,0.1]$, but also performs well in longer time predictions $[0,0.2]$. For comparison purposes, as we did in Sections 4.1 and 4.2, we trained 2D FNO as comparison group. the model adheres to the specifications outlined in Sections 4.1 and 4.2. And the batch size is 10. The shapes of the input and target for the training dataset is $[7000,32,256,2]$ for 2D FNO. For 2D FNO, the model takes a function defined up to a time domain of $\Delta t=0.0125$ and outputs the progressed function over a temporal length of Δt. We measured performance at $t=0.05,0.1,0.15,0.2$, thus to reach these times, the

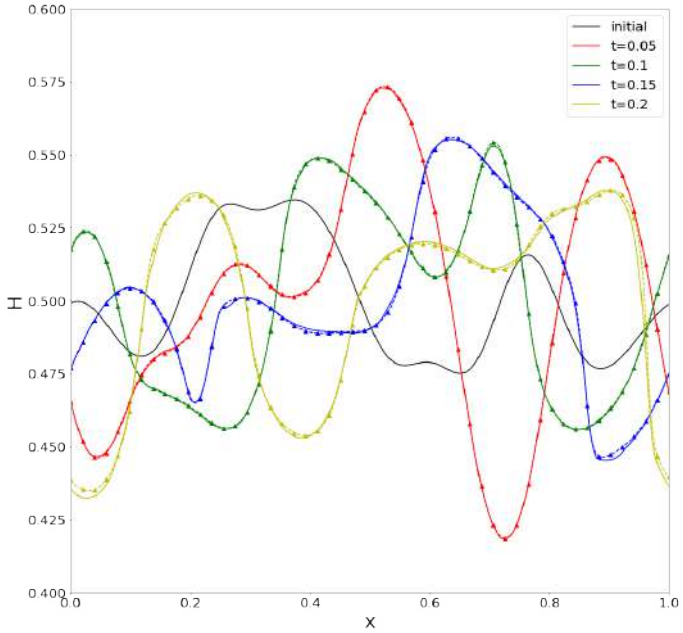

Figure 11: Output (H) of Flux FNO (dashed line with triangle markers) compared with the exact solutions (solid line) for the 1D Shallow water equation problem.

Table 6: Quantitative results of each model for the 1D Shallow water equation problem. Each value represents the mean over the test dataset.

(relative L^2, L^∞)	$t=0.05$	$t=0.1$	$t=0.15$	$t=0.2$
Flux FNO	(**3.58e-3**, **4.57e-3**)	(**5.63e-3**, **5.51e-3**)	(**9.56e-3**, **6.26e-3**)	(**1.54e-2**, **7.34e-3**)
2D FNO	(7.74e-3, 9.80e-3)	(1.21e-2, 1.58e-2)	(1.59e-2, 1.55e-2)	(1.88e-2, 1.57e-2)

Flux FNO iterated 128 times between each time interval, while 2D FNO iterated 4 times. The test samples used for comparison were generated from the same distribution as the training dataset, and averages were taken over a total of 10 samples. The results of the comparison can be found in Table 6. We have also conducted experiments in the same manner as in Section 4.2 to demonstrate the generalization capability of our model for this problem. The results are included in Appendix C. Although there is no remarkable difference in the test performance for samples from the same distribution as shown in the Table 6, as can be seen in Fig. 14, our model demonstrates remarkable generalization capabilities on out-of-distribution samples.

4.4.2 2D linear advection equation

We apply our methodology to one of the simplest multi-dimensional problems, the 2D linear advection equation. The governing equation is set as follows:

$$\frac{\partial u}{\partial t}+\frac{1}{4}\frac{\partial u}{\partial x}+\frac{1}{4}\frac{\partial u}{\partial y}=0.$$

The solution to this equation involves the function u translating at a speed of $\frac{1}{4}$ along each axis. We have assumed periodic boundary conditions and created the training dataset using a 2D Gaussian random field that follows a power spectrum of $P(k)\propto k^{-2.5}$ for the initial conditions. Since the solution over time is merely a translation, we generated it by rolling the sampled initial conditions. The function's domain is $[0,1]\times[0,1]\times[0,1]$ with each axis discretized at 2^{-6}. With this setting, we generated 500 training samples. For this problem, we set the maximal frequency mode of FNOs to 4 and the width to 32. Since there are two axes, we created one model for the flux corresponding to each axis, assigning two FNOs within one Flux FNO unit. The learning rate is set at 1e-4, the λ for loss at 0.25, weight decay at 2e-4, and the scheduler is CosineAnnealingWarmRestarts with a step of 100 and eta_min at 1e-5. The shape of the training dataset is [500,64,64,64,1] and for training, we choose a batch size of 4. The result of the Flux FNO for a test sample (which was sampled from the same distribution as the training dataset) is demonstrated in Fig. 12. Similar to the 1D shallow water equation, we conducted comparison experiments and tested the generalization ability of our models using a 3D FNO for this application. The shape of the training dataset for the 3D FNO is [1500,16,64,64,1], with both input and target configured in this format. And the batch size is 25. The 3D FNO processes a time domain length of 0.25. Performance was measured at $t=0.5,1.0,1.5,2.0$, which required the Flux FNO to iterate 32 times between each time interval, whereas the 3D FNO iterated only 2 times for each interval. The test samples used for comparison were generated from the same distribution as the training dataset, and averages were taken over a total of 10 samples. The results of the comparison can be found in Table 7. Additionally, we have conducted experiments to demonstrate the generalization capability of our model for this problem. The results are included in Appendix C. As can be seen in Figs. 12 and 15, the results from the Flux FNO exhibit some blurring, but overall, the model is able to infer the general shape of the solution. Upon comparing Figs. 15 and 16, it is evident that Flux FNO better preserves the form of the solution.

Table 7: Quantitative results of each model for the 2D Linear advection equation problem. Each value represents the mean over the test dataset.

(relative L^2, L^∞)	$t=0.5$	$t=1.0$	$t=1.5$	$t=2.0$
Flux FNO	(**0.0365**, **0.126**)	(**0.0574**, **0.191**)	(**0.0746**, **0.239**)	(**0.0891**, **0.277**)
3D FNO	(0.110, 0.345)	(0.189, 0.580)	(0.287, 0.899)	(0.399, 1.315)

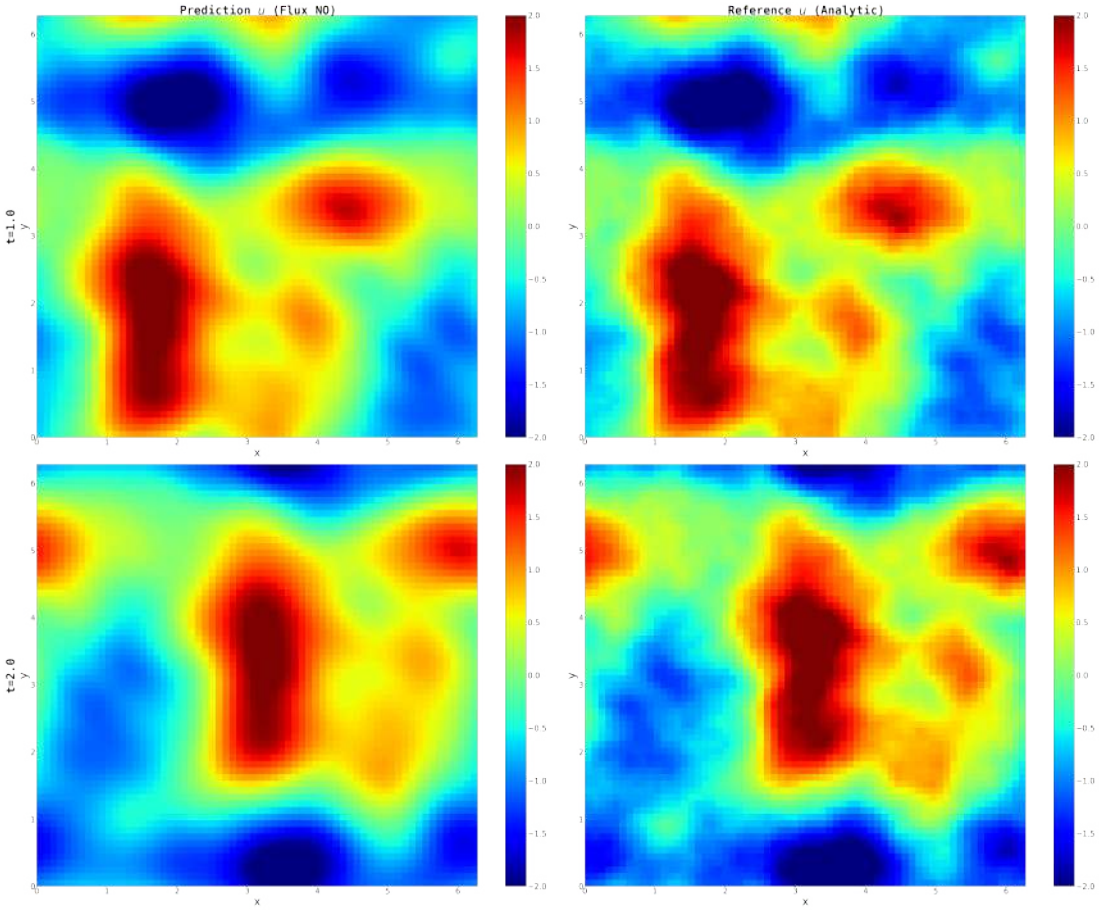

Figure 12: Output of Flux FNO (left) compared with the exact solutions (right) for the 2D linear advection problem.

4.5 Ablation study

The difference between Flux FNO and existing methods that approximate the numerical flux via neural network is that the numerical flux approximated by the FNO model obtains consistency through consistency loss. To verify whether this consistency positively influences the model performance, we conducted an ablation study by removing the consistency loss term from our original loss. We compared the performance with a baseline on various datasets; the governing equation was the Burgers' equation, and the architecture and training settings were the same as those in Section 4.1. Table 8 presents the results. As shown in Table 6, the performance of the baseline model improves by a large margin for relative L^2.

Table 8: Results of baseline and Flux FNO without consistency loss. Each value represents the mean over the test dataset.

relative L^2	inference time	GRF(c=0.1)			GRF(c=0.03)	Square wave
	resolution	128	256	512	256	256
Flux FNO	$t=0.3$	**0.065**	**0.026**	**0.033**	0.13	**0.081**
	$t=0.6$	**0.046**	**0.11**	**0.043**	**0.14**	**0.15**
w/o consistency loss	$t=0.3$	0.13	0.26	0.19	**0.095**	0.38
	$t=0.6$	0.13	0.24	0.15	0.21	0.71

5 Conclusion

In this study, we replace the numerical flux with a neural operator model to solve hyperbolic conservative laws. Through experiments, we demonstrated that our method possesses a superior generalization ability compared with existing FNO models, particularly in terms of long-term prediction and inferences on OOD samples. By combining the RK2 method with our proposed method, we showed that our method is compatible with classical schemes, thereby improving the performance. In the results with linear advection samples, we did not rely on data from numerical schemes but on exact solutions. This implies that when the physical flux is known, exact solutions or experimental data can be used to approximate unknown numerical fluxes. One limitation of our method is its iterative nature, which makes it slower than the original FNO, which produces results in the form of snapshots. However, we anticipate that when we approximate high-order and complex numerical schemes, our method will offer a superior time complexity compared with classical schemes. In future studies, we aim to extend the experiments to more complex hyperbolic conservation systems (such as Euler equations) and conduct a theoretical analysis to guarantee the stability of our method.

Acknowledgments

The authors appreciate the financial support provided by the Data-driven Flow Modeling Research Laboratory funded by the Defense Acquisition Program Administration under Grant [UD230015SD]. This work is supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (Nos. RS-2024-00421203, RS-2024-00406127).

Appendices

A Courant–Friedrichs–Lewy Condition (CFL Condition) and Total Variation Diminishing (TVD) Property

The Courant–Friedrichs–Lewy (CFL) condition is a necessary criterion for the stability of numerical solutions. When a solution progresses over time, the physical propagation of the wave should be less than the numerical propagation speed to ensure its stability. The CFL condition for the 1D case is commonly written as follows:

$$\Delta t \leq \frac{C\Delta x}{|u|_\infty},$$

where u represents the wave speed, and C is a Courant number, which is typically less than one. This ensures that the numerical method can adequately capture the physical phenomena within the time-step and spatial resolution constraints. A multi-dimensional case can be formulated similarly, considering the propagation speeds and spatial resolutions in all relevant dimensions. The total variation is defined as the sum of all differences across the spatial dimensions of a discretized function, and it is mathematically expressed as follows:

$$TV(u) := \sum_j |u_{j+1} - u_j|.$$

We say that numerical scheme has TVD property if the total variation of numerical solution does not increase:

$$TV(u^{n+1}) \leq TV(u^n).$$

As mentioned in the text, it is known that high-order linear numerical schemes in spatial dimensions can lead to spurious oscillations at discontinuities and shocks, thereby increasing the total variation [24]. However, it is also known that even if high-order schemes are maintained in the time dimension, they can possess the TVD (Total Variation Diminishing) property under specific conditions [43], which are as follows:

Lemma A.1. *The Runge-Kutta method* (2.2) *is TVD under the CFL condition* (A) *where C is as follows:*

$$C = \min_{i,k} \frac{\alpha_{ik}}{\beta_{ik}},$$

provided that $\alpha_{ik} \geq 0$, $\beta_{ik} \geq 0$.

B Detailed definition of capacity and Proof of Theorem 3.1

In this section, we present a detailed definition of capacity and provide the proof of Theorem 3.1. For simplicity, we consider an FNO with CNN layers. We require the following definition and lemma from [37].

Definition (Weight Norms and Capacity). For the multi-rank tensor $M_{i_1,\cdots,i_m,j_1,\cdots,j_k}$, we define the following weight norm:

$$\|M_{i_1,\cdots,i_m,j_1,\cdots,j_k}\|_{p:\{i_1,\cdots,i_m\},q:\{j_1,\cdots,j_k\}} := \sqrt[q]{\sum_{j_1\cdots j_m}\left(\sqrt[p]{\sum_{i_1\cdots i_k} M_{i_1,\cdots,i_m,j_1,\cdots,j_k}{}^{p}}\right)^{q}}.$$

For $p=\infty$ or $q=\infty$, we consider the sup-norm instead of the aforementioned definition. Now, considering an FNO with a Fourier layer of depth D, we denote Q and P as the projection and lifting weight matrices, respectively. We then define $\|\cdot\|_{p,q}$, where p is the index for positions, frequencies, and inputs, and q is the output index. We define the following norms for the weights and capacities of the entire neural network: In the $\|\cdot\|_{p,q}$ norm for the kernel tensor of the CNN layer, p is the index of the kernels and input, and q is the output index.

$$\|(K_i,R_i)\|_{p,q} := \|K\|_{p,q}\sqrt[p*]{c_1\cdots c_d} + \sqrt[p*]{k_{\max,1}\cdots k_{\max,d}}\|R\|_{p,q},$$
$$\gamma_{p,q}(h) := \|P\|_{p,q}\|Q\|_{p,q}\|\prod_{i=1}^{D}\|(K_i,R_i)\|_{p,q}.$$

Lemma B.1 (Posterior Estimation of Generalization Error and Expected Error in the ReLU Activation Case)**.** *Consider FNO with CNN layers with fixed architecture and training samples* $\{(a_i,u_i)\}_{i=1,\cdots,m}$ *with* $\|a_i\|_{p*}\le B$ *for all i. Suppose* $h(\cdot;\theta)$ *is a trained FNO such that* $\|h(a;\theta)-u\|_2\le\epsilon^2$ *for all training samples, and* $1\le p\le 2$, $1\le q\le p*$. *Then, with a confidence level of at least* $1-\delta$, *we obtain the following estimates, where* $L_{\mathcal{D}}$ *represents the loss over the distribution, and* L_S *represents the loss over training dataset S:*

$$L_{\mathcal{D}}(h(\cdot;\theta))-L_S(h(\cdot;\theta))\le\epsilon C\gamma_{p,q}(h(\cdot;\theta))\frac{\min\{p*,4\log(2d_a)\}B}{\sqrt{m}}+\epsilon^2\sqrt{\frac{2\log 4/\delta}{m}},$$
$$\Longrightarrow L_{\mathcal{D}}(h(\cdot;\theta))\le\epsilon C\gamma_{p,q}(h(\cdot;\theta))\frac{\min\{p*,4\log(2d_a)\}B}{\sqrt{m}}+\epsilon^2\left(1+\sqrt{\frac{2\log 4/\delta}{m}}\right),$$

where C is constant dependent on model architecture.

Now, we present the proof of Theorem 3.1.

Proof. We begin with two equations derived from the conservation law and numerical scheme. We assume that the initial condition ($t=k$) is the same, and for simplicity, we assume G takes two pairs of inputs

$$\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u(x,t_{k+1})dx=\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u(x,t_k)dx-\int_{t_k}^{t_{k+1}} f(u(x_{j+\frac{1}{2}},t))dt+\int_{t_k}^{t_{k+1}} f(u(x_{j-\frac{1}{2}},t))dt, \quad \text{(B.1)}$$
$$\tilde{U}_j^{k+1}=\tilde{U}_j^k-\frac{h}{k}\Big(G(\tilde{U}_j^k,\tilde{U}_{j+1}^k)-G(\tilde{U}_{j-1}^k,\tilde{U}_j^k)\Big). \quad \text{(B.2)}$$

Let us denote $U^{k+1} := \frac{1}{k}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u(x,t_{k+1})dx$ and $U^k := \frac{1}{k}\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} u(x,t_k)dx$. Based on our assumption, we have $U^k = \tilde{U}^k$; let $\hat{F}$ be a consistent numerical flux approximation. We can then express the physical flux as a numerical flux. By subtracting (B.1) from (B.2), we obtain the following equation:

$$
\begin{aligned}
& U_j^{k+1} - \tilde{U}_j^{k+1} \\
=& \int_{t_k}^{t_{k+1}} \Big(\hat{F}(u(x_{j-\frac{1}{2}},t),u(x_{j-\frac{1}{2}},t)) - G(U_{j-1}^k,U_j^k)\Big)dt \\
& - \int_{t_k}^{t_{k+1}} \Big(\hat{F}(u(x_{j+\frac{1}{2}},t),u(x_{j+\frac{1}{2}},t)) - G(U_j^k,U_{j+1}^k)\Big)dt \\
=& \int_{t_k}^{t_{k+1}} \Big(\hat{F}(u(x_{j-\frac{1}{2}},t),u(x_{j-\frac{1}{2}},t)) - \hat{F}(U_{j-1}^k,U_j^k) + \hat{F}(U_{j-1}^k,U_j^k) - G(U_{j-1}^k,U_j^k)\Big)dt \\
& - \int_{t_k}^{t_{k+1}} \Big(\hat{F}(u(x_{j+\frac{1}{2}},t),u(x_{j+\frac{1}{2}},t)) - \hat{F}(U_j^k,U_{j+1}^k) + \hat{F}(U_j^k,U_{j+1}^k) - G(U_j^k,U_{j+1}^k)\Big)dt, \\
\Longrightarrow\quad & |U_j^{k+1} - \tilde{U}_j^{k+1}|^2 \\
\leq & \|\hat{F}(u(x_{j-\frac{1}{2}},\cdot),u(x_{j-\frac{1}{2}},\cdot)) - \hat{F}(U_{j-1}^k,U_j^k)\|_2^2 + \|\hat{F}(u(x_{j+\frac{1}{2}},\cdot),u(x_{j+\frac{1}{2}},\cdot)) - \hat{F}(U_j^k,U_{j+1}^k)\|_2^2 \\
& + h|\hat{F}(U_{j-1}^k,U_j^k) - G(U_{j-1}^k,U_j^k)|^2 + h|\hat{F}(U_j^k,U_{j+1}^k) - G(U_j^k,U_{j+1}^k)|^2.
\end{aligned}
$$

By applying the consistency property and Lemma B.1, and summing over the positional index, we obtain the following inequality, with a probability of over $1-\delta$:

$$
\begin{aligned}
\Longrightarrow\quad & \|U^{k+1} - \tilde{U}^{k+1}\|_2^2 \\
\leq & C_1 h\gamma \frac{\epsilon_{consi}^{t_k} B}{\sqrt{m}} + h\epsilon_{consi}^{t_k 2}\Big(1 + \sqrt{\frac{2\log(\frac{4}{\delta})}{m}}\Big) \\
& + C_2 h sup_{j,t\in(t_k,t_{k+1})}\Big(\max(|u(x_{j+\frac{1}{2}},t) - U_{j-1}^k|^2, \\
& \quad |u(x_{j+\frac{1}{2}},t) - U_j^k|^2, |u(x_{j-\frac{1}{2}},t) - U_j^k|^2, |u(x_{j-\frac{1}{2}},t) - U_{j-1}^k|^2)\Big).
\end{aligned}
$$

where C_1 is a coefficient derived from Lemma B.1, and C_2 is dependent on the Lipschitz constant of numerical flux $\hat{F}$. As time step h approaches zero, the second term in the inequality above approaches zero. Let us abbreviate it as $C_2 h\epsilon(h)$. We also obtain the following inequality from the time-marching loss and Lemma B.1, with a probability of over $1-\delta$

$$
\begin{aligned}
& \|U^{k+1} - \tilde{U}^{k+1}\|_2^2 \\
\leq & C_3\tilde{\gamma}\frac{\epsilon_{tm}^{t_k} B}{\sqrt{m}} + \epsilon_{tm}^{t_k 2}\Big(1 + \sqrt{\frac{2\log(\frac{4}{\delta})}{m}}\Big).
\end{aligned}
$$

Here, $\tilde{\gamma}$ represents the capacity when the output structure of the Flux FNO is considered. Without losing generality, we can incorporate the effect of this structure into the constant C_3 and simply denote it as C_3. Finally, we derive the following inequality, with a probability of over $1-\delta$

$$\|U^{k+1}-\tilde{U}^{k+1}\|_2^2$$
$$\leq \min\left(C_3\gamma\frac{\epsilon_{tm}^{t_k}B}{\sqrt{m}}+\epsilon_{tm}^{t_k 2}\left(1+\sqrt{\frac{2\log(\frac{4}{\delta})}{m}}\right), h\left(C_1\gamma\frac{\epsilon_{consi}^{t_k}B}{\sqrt{m}}+\epsilon_{consi}^{t_k 2}\left(1+\sqrt{\frac{2\log(\frac{4}{\delta})}{m}}\right)+C_2\epsilon(h)\right)\right).$$

This completes the proof. □

C Additional figures for Section 4.4

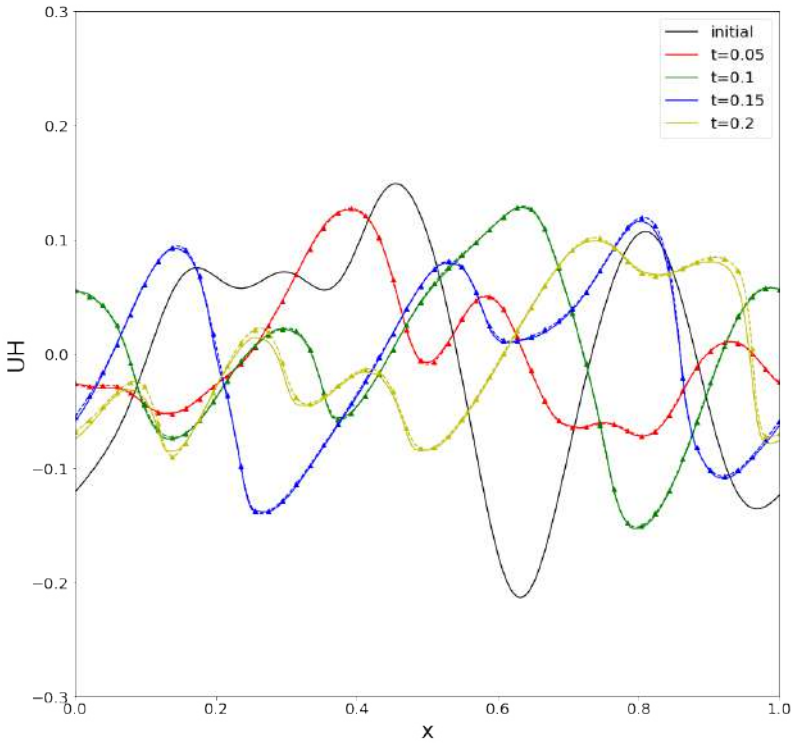

Figure 13: Output (UH) of Flux FNO (dashed line with triangle markers) compared with the exact solutions (solid line) for the 1D Shallow water equation problem.

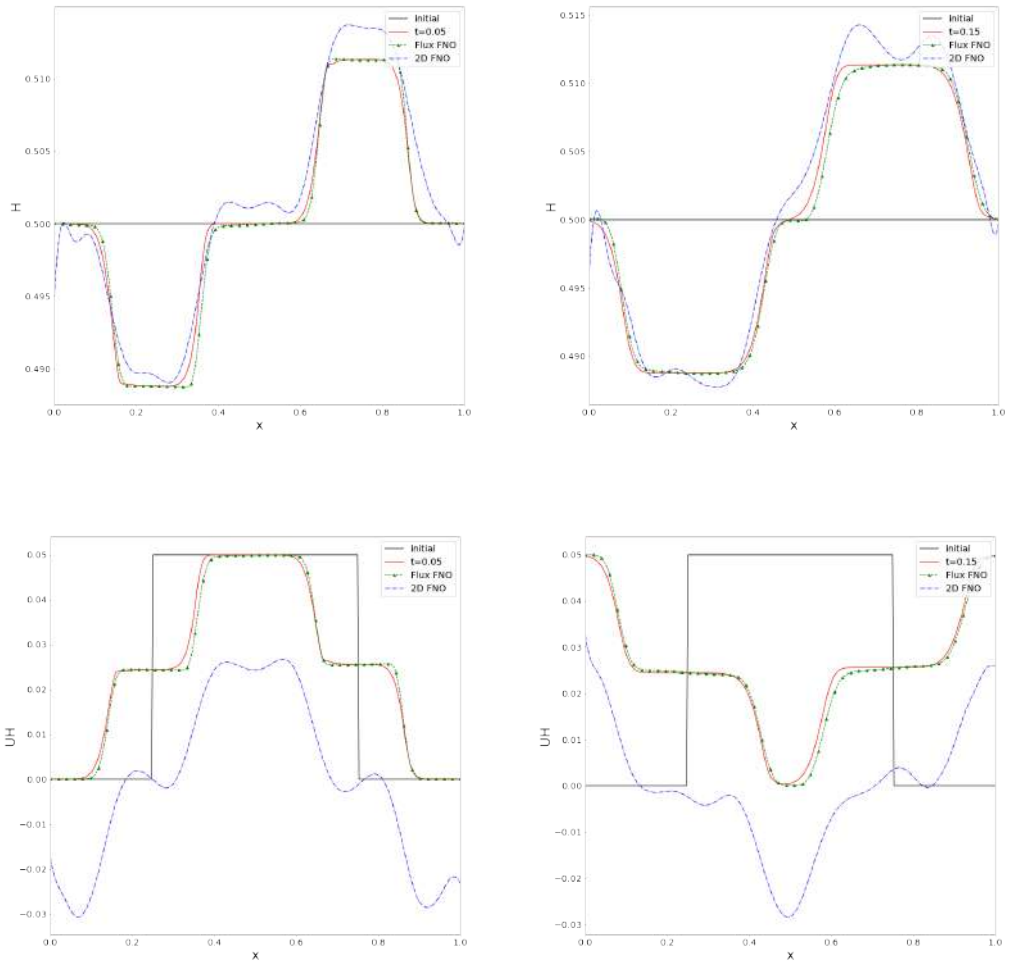

Figure 14: Inference of Flux FNO on out-of-distribution samples for the 1D Shallow water equation problem with initial condition is square pulse: U (top), UH (bottom).

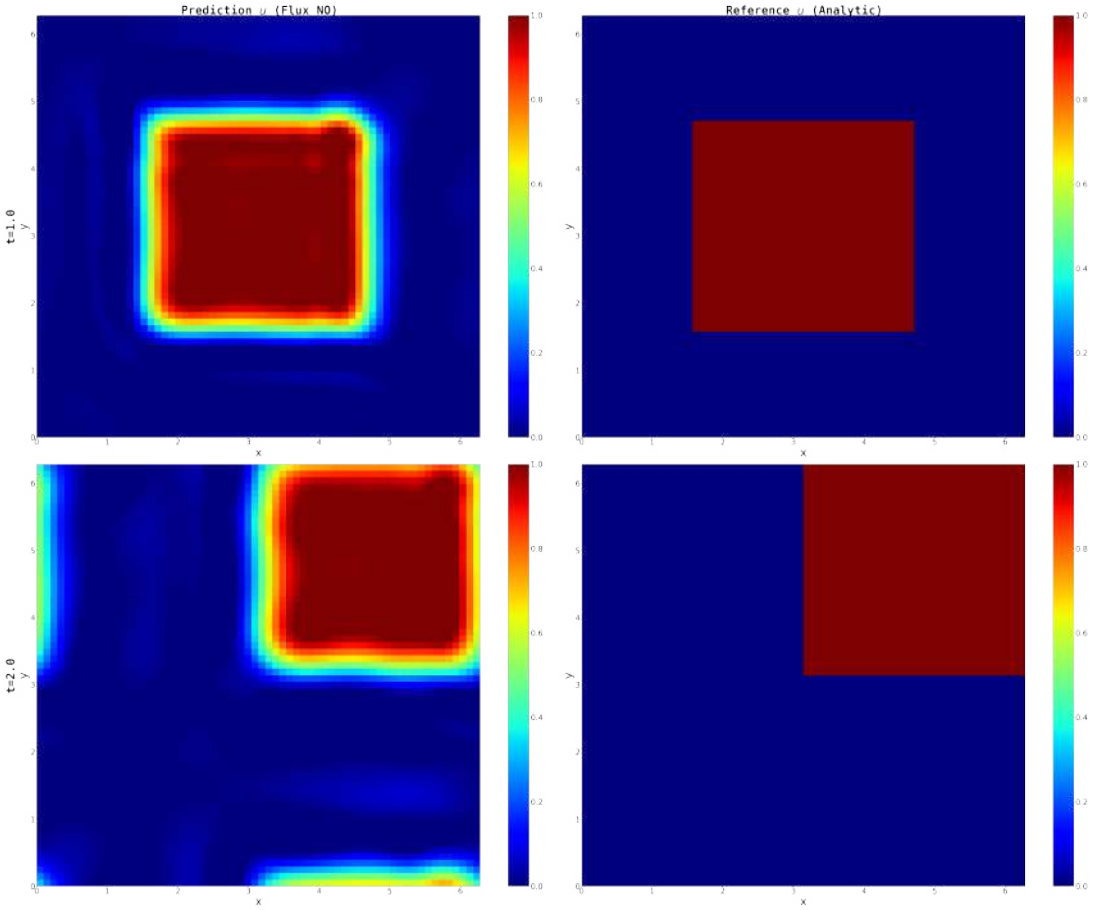

Figure 15: Inference of Flux FNO on Out-of-Distribution Samples for the 2D Linear Advection Problem: Comparison of Output from Flux FNO (left) with Exact Solutions (right) for Square Pulse Initial Condition.

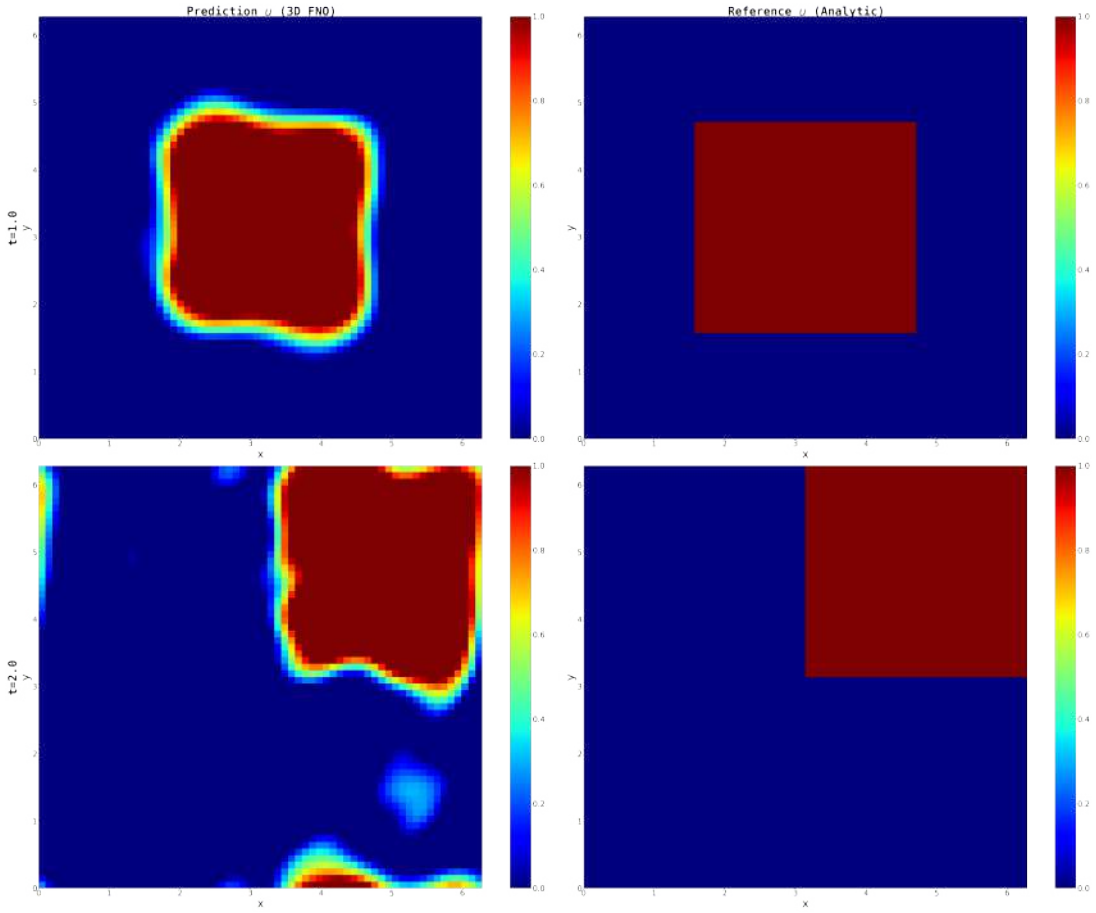

Figure 16: Inference of 3D FNO on Out-of-Distribution Samples for the 2D Linear Advection Problem: Comparison of Output from 3D FNO (left) with Exact Solutions (right) for Square Pulse Initial Condition.

D Experimental details

All experiments were conducted using Pytorch 2.0.1, with Python 3.9.6. The specifications of the hardware environment are provided in Table 9.

Table 9: Specifications of computer hardware.

CPU	GPU	RAM
Intel i9-10900	Nvidia RTX3080	64GB

References

[1] Chen, Z., Gelb, A. & Lee, Y. Designing Neural Networks for Hyperbolic Conservation Laws. *ArXiv*. **arXiv:2211.14375** (2022).

[2] Ruggeri, M., Roy, I., Mueterthies, M., Gruenwald, T. & Scalo, C. Neural-network-based Riemann solver for real fluids and high explosives; application to computational fluid dynamics. *Physics Of Fluids*. **34**, 116121 (2022).

[3] Leer, B. Towards the ultimate conservative difference scheme. II. Monotonicity and conservation combined in a second-order scheme. *Journal Of Computational Physics*. **14**, 361-370 (1974).

[4] Sweby, P. High resolution schemes using flux limiters for hyperbolic conservation laws. *SIAM Journal On Numerical Analysis*. **21**, 995-1011 (1984).

[5] Harten, A., Engquist, B., Osher, S. & Chakravarthy, S. Uniformly high order essentially non-oscillatory schemes III. *Journal Of Computational Physics*. **71**, 231-303 (1987).

[6] Liu, X., Osher, S. & Chan, T. Weighted essentially non-oscillatory schemes. *Journal Of Computational Physics*. **115**, 200-212 (1994).

[7] Holl, P., Koltun, V. & Thuerey, N. Learning to Control PDEs with Differentiable Physics. *ICLR 2020*. (2020).

[8] Basir, S. & Senocak, I. Critical Investigation of Failure Modes in Physics-informed Neural Networks. *AiAA SCITECH 2022 Forum*. (2022).

[9] Kossaczká, T., Ehrhardt, M. & Günther, M. Enhanced fifth order WENO shock-capturing schemes with deep learning. *Results In Applied Mathematics*. **12** (2021).

[10] Wang, Y., Shen, Z., Long, Z. & Dong, B. Learning to Discretize: Solving 1D Scalar Conservation Laws via Deep Reinforcement Learning. *ArXiv*. **arXiv:1905.11079** (2020).

[11] Fu, L. A very-high-order TENO scheme for all-speed gas dynamics and turbulence. *Computer Physics Communications*. **244** pp. 117-131 (2019).

[12] Wang, S., Yu, X. & Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal Of Computational Physics*. **449** pp. 110768 (2022).

[13] Cai, Z., Chen, J. & Liu, M. Least-squares ReLU neural network (LSNN) method for linear advection-reaction equation. *Journal Of Computational Physics*. **443** pp. 686-707 (2021).

[14] Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*. **3**, 218-229 (2021).

[15] G. Gupta, X. & Bogdan, P. Multiwavelet-based Operator Learning for Differential Equations. *Advances In Neural Information Processing Systems*. (2021).

[16] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. & Anandkumar, A. Neural Operator: Graph Kernel Network for Partial Differential Equations. *ICLR 2020 Workshop ODE/PDE+DL*. (2020).

[17] Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A. & Anandkumar, A. Fourier Neural Operator for Parametric Partial Differential Equations. *ICLR 2021*. (2021).

[18] Kovachki, N., Lanthaler, S. & Mishra, S. On Universal Approximation and Error Bounds for Fourier Neural Operators. *Journal Of Machine Learning Research*. **22** (2021).

[19] Gopalani, P., Karmakar, S. & Mukherjee, A. Capacity Bounds for the DeepONet Method of Solving Differential Equations. *ArXiv*. **arXiv:2205.11359** (2022).

[20] Holl, P., Koltun, V. & Thuerey, N. Scale-invariant Learning by Physics Inversion. *Advances In Neural Information Processing Systems*. (2022).

[21] Sirignano, J. & Spiliopoulos, K. DGM: A deep learning algorithm for solving partial differential equations. *Journal Of Computational Physics*. **375** pp. 1339-1364 (2018).
[22] Weinan, E. & Yu, B. The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. *Communications In Mathematics And Statistics*. **6** pp. 1-12 (2018).
[23] Mojgani, R., Balajewicz, M. & Hassanzadeh, P. Kolmogorov n–width and Lagrangian physics-informed neural networks: A causality-conforming manifold for convection-dominated PDEs. *Computer Methods In Applied Mechanics And Engineering*. **404** pp. 115810 (2023).
[24] Godunov, S. A difference method for numerical calculation of discontinuous solutions of the equations of hydrodynamics. *Matematicheskii Sbornik*. **47**, 271-306 (1959).
[25] Shahabi, A. & Ghiassi, R. A Robust Second-Order Godunov-Type Method for Burgers' Equation. *International Journal Of Applied And Computational Mathematics*. **8** (2022).
[26] LeVeque, R. Numerical Methods for Conservation Laws. (Birkhäuser Basel, 1992).
[27] Jin, S. Runge-Kutta Methods for Hyperbolic Conservation Laws with Stiff Relaxation Terms. *Journal Of Computational Physics*. **122**, 51-67 (1995).
[28] Pathak, J., Subramanian, S., Harrington, P., Raja, S., Chattopadhyay, A., Mardani, M., Kurth, T., Hall, D., Li, Z., Azizzadenesheli, K., Hassanzadeh, P., Kashinath, K. & Anandkumar, A. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators. *ArXiv*. **arXiv:2202.11214** (2022).
[29] Shalev-Shwartz, S. & Ben-David, S. Understanding Machine Learning: From Theory to Algorithms. (Cambridge University Press, 2014).
[30] Vapnik, V. An overview of statistical learning theory. *IEEE Transactions On Neural Networks*. **10**, 988-999 (1999).
[31] Jakubovitz, D., Giryes, R. & Rodrigues, M. Generalization Error in Deep Learning. (Birkhäuser Cham, 2019).
[32] Valiant, L. A theory of the learnable. *Communications Of The ACM*. **27**, 1134-1142 (1984).
[33] Kovachki, N., Li, Z., Liu, B., Azizzadenesheli, K., Bhattacharya, K., Stuart, A. & Anandkumar, A. Neural Operator: Learning Maps Between Function Spaces. *ArXiv*. **arXiv:2108.08481** (2021).
[34] Wen, G., Li, Z., Azizzadenesheli, K., Anandkumar, A. & Benson, S. U-FNO—An enhanced Fourier neural operator-based deep-learning model for multiphase flow. *Advances In Water Resources*. **163** (2022).
[35] Courant, R., Isaacson, E. & Rees, M. On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications On Pure And Applied Mathematics*. **5**, 243-255 (1952).
[36] Lax, P. Weak solutions of nonlinear hyperbolic equations and their numerical computation. *Communications On Pure And Applied Mathematics*. **7**, 159-193 (1954).
[37] Kim, T. & Kang, M. Bounding the Rademacher Complexity of Fourier neural operators. *Machine Learning*. **113** (2024).
[38] Benitez, J., Furuya, T., Faucher, F., Kratsios, A., Tricoche, X. & Hoop, M. Out-of-distributional risk bounds for neural operators with applications to the Helmholtz equation. *ArXiv*. **arXiv:2301.11509** (2023).
[39] Ray, D. & Hesthaven, J. An artificial neural network as a troubled-cell indicator. *Journal Of Computational Physics*. **367** pp. 166-191 (2018)
[40] Magier, J., Ray, D., Hesthaven, J. & Rohde, C. Constraint-aware neural networks for Riemann problems. *Journal Of Computational Physics*. **409** pp. 109345 (2020).

[41] Gottlieb, S. & Shu, C. Total Variation Diminishing Runge-Kutta Schemes. *Mathematics Of Computation*. **67** pp. 73-85 (1998).
[42] O'Shea, K. & Nash, R. An Introduction to Convolutional Neural Networks. *ArXiv*. **arXiv:1511.08458** (2015).
[43] Shu, C. & Osher, S. Efficient implementation of essentially non-oscillatory shock- capturing schemes. *Journal Of Computational Physics*. **21** pp. 439-471 (1988).