Learning Transient Partial Differential Equations with Local Neural Operators

Ximeng Ye†, Hongyu Li‡, Peng Jiang‡, Tiejun Wang‡*, and Guoliang Qin‡*

1School of Energy and Power Engineering, Xi’an Jiaotong University, Xi’an, 710049, Shaanxi, China.
2State Key Lab for Strength and Vibration, Department of Engineering Mechanics, Xi’an Jiaotong University, Xi’an, 710049, Shaanxi, China.

*Corresponding author(s). E-mail(s): 973wtj@xjtu.edu.cn; glqin@mail.xjtu.edu.cn;
Contributing authors: yeximeng@stu.xjtu.edu.cn; lihongyu_0@163.com; jiangpeng219@mail.xjtu.edu.cn;
†These authors contributed equally to this work.

Abstract

In decades, enormous computational resources are poured into solving the transient partial differential equations for multifarious physical fields. The latest artificial intelligence has shown great potential in accelerating these computations, but its road to wide applications is hindered by the variety of computational domains and boundary conditions. Here, we overcome this obstacle by constructing a learning framework capable of purely representing the transient PDEs with local neural operators (LNOs). This framework is demonstrated in learning several transient PDEs, especially the Navier-Stokes equations, and successfully applied to solve problems with quite different domains and boundaries, including the internal flow, the external flow, and remarkably, the flow across the cascade of airfoils. In these applications, our LNOs are faster than the conventional numerical solver by over 1000 times, which could be significant for scientific computations and engineering simulations.

Keywords: local neural operator, transient partial differential equation, Navier-Stokes equation, domain of dependence, receptive range
1 Introduction

Partial differential equations (PDEs), especially transient PDEs, are the mathematical formulation of physical laws such as the convection and diffusion in fluid flow, the deformation of solid structures, the conduction of heat, etc. To understand these laws and construct the PDEs accurately, human researchers have paid great efforts in experiments, observation, analysis, and summarization. Here, we wonder whether the artificial intelligent (A.I.) agents can summarize the laws of physics as what human researchers have been doing all the time? Once this conception is achieved, not only it can feed our curiosity, but also the A.I. based methods will become powerful tools for exploring the laws of physics, meanwhile, it could assist or partly replace the numerical simulation of physical field and benefit engineering developments. To achieve this goal, this work studies the way to powerful A.I. agents to learn the transient PDEs.

A properly defined learning problem is the prerequisite to construct effective A.I. agents. It determines the trainability and the applicability of the neural networks. For the trainability, we can hardly or never train a neural network to predict the output which is with little relevance to the input. For the applicability, the learning problem essentially contains the assumption of how to apply the trained networks, that is, it determines the scope of application. The following review of recent related literature focus on their learning problem definition.

Learning PDEs as functions. Taking $D$ as the computational domain, and $t$ as the time, the process to solve a transient PDE is to find a solution function $U(D,t)$ satisfying the equations and all boundary conditions in the computational domain. To this end, the pioneers adopted a natural definition of the learning problem, that is, to let the neural networks directly approximate the solution function $U(D,t)$ minimizing the residual of equations $[1, 2]$. Recently, as a modern reproduction, development, and application of this classic idea, the Physics-Informed Neural Networks (PINNs) $[3, 4]$, achieved a great success and earned a lot of attention $[5–10]$. However, though there are encouraging applications reported $[11–18]$, it has encountered a bottleneck in expanding the scope of application, that is, neural networks constructed as functions are limited on a certain case, i.e., the PDE with a certain computation domain and boundaries (for transient PDEs, the initial conditions are also included). This means, even if only the initial condition is changed, the trained model cannot be reused to other cases.

Learning PDEs as operators. More recently, the researchers tried another learning problem to let the neural networks approximate an operator which generates the solution $U(D, t)$ according to an input function. For transient PDEs, specifically, neural networks are defined to be a map from the physical field at $t$ to the physical field at $t + \Delta t$. These neural networks can generate the complete solution $U(D, t)$ by taking the initial condition as the first input and predicting the output recurrently. It also implies that the neural operators get rid of the assumption of a certain initial condition, which could
be the most significant advantage of this learning problem setting. Impressive neural architectures adopting this setting include Deep Operator Network (DeepONet) [19], Fourier Neural Operator (FNO) [20] and its several variants [21–23]. These networks succeeded in learning various PDEs, including the challenging Navier-Stokes (N-S) equation, and have good applications [24, 25]. However, a specific computational domain and boundary are still assumed in the network training. This assumption limits the scope of application in diverse scientific and engineering scenarios.

The ideal A.I. agent with a maximum scope of generalization should essentially learn the PDEs without assumptions on the initial condition, the shape of the computational domain, or the boundary conditions. In fact, the governing equations (PDEs), which represent the essential law of physics, play a primary role and determine the solution function jointly with the other three elements. In other words, the boundary conditions or the computational domain should not affect the laws of physics governed by the transient PDEs. In the existing literature, the neural networks trained for learning PDEs as functions assume all the four elements fixed; the neural networks trained for learning PDEs as operators eliminate the assumption on the initial condition, but are still troubled in dealing with various computational domains and boundaries. For this issue, some primary ideas of conventional numerical schemes are inspiring. For example, the finite element method (FEM) divides the computational domain into small elements and handles them in the same way [26, 27]. Similar ideas are also adopted in the development from spectral method to the spectral element method (SEM) [28–30]. It provides FEM/SEM great flexibility to be independent of the computational domain and the boundary conditions, which enables a wide application of FEM. Moreover, this idea leads us to give out a better definition of the learning problem.

In this work, neural networks learn transient PDEs as local operators. We propose a more general framework to learn the transient PDEs without any assumptions on the initial condition, the shape of the computational domain, or the boundary conditions. Specifically, with the concept ‘domain of dependence’, we raise a unique local-related learning problem to learn the transient PDEs with local neural operators (LNOs). This framework is validated by experiments in learning transient PDEs including Burgers equations, wave equations, and N-S equations. We summarize the advantages and the main contributions of this work as follows.

1. The local neural operator learns pure laws governed by the transient PDEs. By the local-related hypothesis raised herein, the governing equations are stripped from other elements (including the initial condition, boundary condition, and the computational domain) in network training.
2. We provide two strategies here for handling almost all types of boundaries. Hence, the well-trained local neural operators can directly predict solution functions of problems in any computational domain with various initial and boundary conditions without any additional training or optimization. We present several cases for solving Navier-Stokes equations as examples.
3. Thanks to the maximum flexibility provided by the learning framework, any well-trained powerful local neural operator has great potential for reuse in various scenarios of scientific or engineering practices.

2 Hypothesis and learning problem definition

2.1 The local-related hypothesis of transient PDEs

The success of FNO [20] in learning PDEs lies on a basic hypothesis that, there exists a solution operator $G$ which is a map between two function spaces governed by the PDEs. Specifically, for transient PDEs with certain boundary conditions, the solution operator $G : \mathcal{U}_t(D; \mathbb{R}^{d_u}) \rightarrow \mathcal{U}_{t+\Delta t}(D; \mathbb{R}^{d_u})$ is invariant and learnable for one neural operator. Here $\mathcal{U}_t(D)$ equals to $\mathcal{U}(D, t)$; $D$ is the computational domain; $d_u$ is the number of dimensions of $\mathcal{U}$, i.e., the number of physical fields to be computed.

We raise a stronger and also intuitive hypothesis about the solution operator of transient PDEs as follows.

**The local-related hypothesis.** For problems governed by the same transient PDEs, with a limited and constant $\Delta t$, there exists a solution operator $G_l : \mathcal{U}_t(D_1; \mathbb{R}^{d_u}) \rightarrow \mathcal{U}_{t+\Delta t}(D_2; \mathbb{R}^{d_u})$, for any bounded $D_2 \subset D$, the maximum distance $r_{dep}$ between points inside $D_1$ and $D_2$ is limited. $D_1$ is the domain of dependence $^1$ of $D_2$ determined by the PDEs. $r_{dep}$ is called ‘range of dependence’ in this paper.

This hypothesis is formulated intuitively from some ubiquitous physical phenomena. For example, it costs months for the radiative wastewater from the Fukushima disaster to gradually spread all over the world [32]; more rapidly, the ocean wave caused by an underwater earthquake or volcanic eruption travels throughout the Pacific in hours [33]. In another word, when observing these processes using a relatively small $\Delta t$, the influenced range should be limited. Besides, using the concept of Green’s function, this hypothesis is also explained mathematically (see Appendix. A)

2.2 Learning transient PDEs as local neural operators

Following the local-related hypothesis, we attempt to learn the transient PDEs as local neural operator $G_\theta : \mathcal{U}_t(D_3; \mathbb{R}^{d_u}) \rightarrow \mathcal{U}_{t+\Delta t}(D_4; \mathbb{R}^{d_u})$ parameterized by $\theta$ which maps the field function $\mathcal{U}_t$ to $\mathcal{U}_{t+\Delta t}$, where $D_3, D_4 \subset D$. The process to train the local neural operator $G_\theta$ is to optimize $\theta$ to achieve an approximation of the real operator $G_l$ as accurately as possible.

For a local neural operator $G_\theta$, the minimum related range of the output with respect to the input is defined as the ‘receptive range’ $r_{rcp}$. In practice, $r_{rcp}$ depends on the specific architecture design of the neural network. Naturally, there is a basic requirement of the designed local neural operator that $r_{rcp}$ of the local neural operator should be equal or greater than $r_{dep}$ of the

---

$^1$‘Domain of dependence’ is a basic concept in the theory of partial differential equations [31]. Here we use the term to support the local-related hypothesis and the learning problem definition.
transient PDEs to be learned. Otherwise, part of the physical information may be ignored by the network, and the training becomes very hard or even impossible. Figure 1 presents the mean square error (MSE) curves when using the local neural operators with different $r_{rcp}$ to learn the same transient PDEs. The local neural operators achieve competitive MSE with the nonlocal neural operator FNO just when adopting a proper $r_{rcp}$ regarding $r_{dep}$ of the transient PDEs.

A common kind of neural network, the convolutional neural network, is essentially a local neural operator. A 1-D discrete case using the convolutional layer $C$ for mapping $U_t([x_1 - K\Delta x, x_2 + K\Delta x]; \mathbb{R}) \rightarrow U_{t+\Delta t}([x_1, x_2]; \mathbb{R})$ can be written as

$$U_{t+\Delta t}(x) = (C(U_t))(x) = \sum_{i=-K}^{K} \kappa(i) U_t(x + i\Delta x), \ \forall x \in [x_1 - K\Delta x, x_2 + K\Delta x].$$

(1)

where $U_t$, $U_{t+\Delta t}$ are fields at time $t$ and $t + \Delta t$ respectively. $\kappa \in \mathbb{R}^{2K+1}$ is the kernel weight of the convolutional layer. Obviously, with a limited and constant $\Delta x$, the receptive range $r_{rcp} = K\Delta x$ is also limited. Thus, the convolutional operator and the convolutional neural networks, which are simply composed of convolutional layers and local nonlinear activation, are local neural operators.
3 Local Neural Operator - the architecture

3.1 Overall architecture

Figure 2 shows the overall architecture of a specific neural network we designed here as Local Neural Operator.

Firstly, the input $U_t(D; \mathbb{R}^{d_u})$ is processed by a local linear layer for channel lifting from $d_u$ to $d_v$. At the end of the network, the processed intermediate function of $d_v$ channels are projected back to $d_u$ with pointwise operations. $n$ blocks with independent weights compose the main body. In blocks, we design two paths for the networks to learn an approximation of the real PDE operator in two different function spaces, i.e., the physical space and the spectral space. In the physical path, we use a simple mini-network comprised of two stacked convolutional layers and an activation function sandwiched. About the spectral path, a more detailed description is in Section 3.2. The outputs of the two paths are added together and processed with an activation function $\sigma$ before sending out of the block.

3.2 Operations in the spectral path

Taking the conception of SEM [29] and Fourier integral operator in FNO [20], we design local spectral neural operations in the spectral path. These operations include a local spectral transform $T$, its inverse $T^{-1}$, and a linear weight $W \in \mathbb{R}^{d_v \times M \times M}$ for the interaction between $M$ modes. Here, $M$ is the number of spectral modes adopted. Note that in discretized cases, $M \leq N$, where $N$ is the number of discrete points. One can do frequency selection by adopting part of the modes, for example, adopting the $M$ modes at lower frequencies for low-pass filtering. The operations of the spectral path on a local domain $D' \in D$ are

$$\hat{V}_{cm} = (T(V_c))_m = \int_{D'} V_c(x) \varphi(m, x) dx, \quad m = 0, 1, ..., M - 1,$$

(2)
\[ \hat{V}_{cm'} = \sum_{m=0}^{M-1} w_{cm'm} \hat{V}_m, \quad w_{cm'm} \in W, \quad m' = 0, 1, ..., M - 1, \quad (3) \]

\[ V'_c(x) = (T^{-1}(\hat{V}_c))(x) = \sum_{m'=0}^{M-1} \hat{V}_{cm'} \psi(m', x), \quad x \in D', \quad (4) \]

where \( c = 1, ..., d_v \); \( V(D; \mathbb{R}^{d_v}) \) and \( V'(D; \mathbb{R}^{d_v}) \) are the input and output functions of the block, respectively; \( \hat{V}, \hat{V}' \in \mathbb{R}^{d_v \times M} \) are the local modal vectors in the spectral space before and after the linear operation, respectively. The subscript \( m \) and \( m' \) are both indicating the modal components; \( \varphi \) and \( \psi \) are the normalized basis of the spectral transform and its inverse, respectively. These local spectral operations are also illustrated in Figure 3. The function \( V \) on the local domain \( D' \) is decomposed, linear transformed, and reconstructed to the output function \( V' \). The spectral operations can be conducted on any local part of the computation domain.

![Fig. 3 Local operations of the spectral path.](image)

Actually, we can adopt different spectral transforms by using different \( \varphi \) and \( \psi \). The Legendre polynomials and Chebyshev polynomials are considered in this paper. A brief introduction of these two families of polynomials is in Appendix B. Note that since \( D' \) is usually aperiodic, the Fourier series is not suitable here. It introduces extra high-frequency components owing to the discontinuity at the boundary, which troubles the wave filtering and may blow up the recurrent process to predict the solution function.

Then, we introduce the decomposition, reconstruction of geometry, and an easy realization of them using convolutions and deconvolutions. For practice, we consider a specific 2-D case which adopts an equidistant grid discretization. As shown in Figure 4, the domain \( D \) is decomposed into square pieces. They are respectively sent to the spectral operations as input function. To ensure an invertible decomposition, we define a parameter, the number of repetitions \( k \), then the size of the window \( N \) should be an integer multiple of \( k \), and the stride \( s = \frac{N}{k} \). Correspondingly, the input function on the decomposed pieces should be scaled by factor \( \frac{1}{k} \) for normalization. With this geometry decomposition and
Local Neural Operators

Fig. 4 Discretized local operations of the spectral path. The term ‘corrosion of the domain’ is discussed in Section 3.3.

the discrete functions $\mathcal{V}, \mathcal{V}' \in \mathbb{R}^{d_v \times N_a \times N_b}, \hat{\mathcal{V}}, \hat{\mathcal{V}}' \in \mathbb{R}^{d_v \times M \times (N_a - 1) \times (N_b - 1)}$, where $N_a, N_b$ are the width and length of the computational domain, the local spectral transform coincides with the form of the convolution formula that

$$\hat{V}_{cmxy} = \frac{1}{k^2} \sum_{i=0}^{N_a-1} \sum_{j=0}^{N_b-1} V_{c(sx+i)(sy+j)} \varphi_2(m, i, j), \quad (5)$$

where $c = 1, \ldots, d_v$; $m = 0, \ldots, M - 1$; $x = 1, \ldots, \frac{N_a}{s} - 1$; $y = 1, \ldots, \frac{N_b}{s} - 1$; $\varphi_2$ is the 2-D discretized version of $\varphi$. This formula is equal to an ordinary convolution from 1 to $M$ channels using a constant filter weight $\kappa_{mij} = \frac{1}{k^2} \varphi_2(m, i, j)$. Conversely, the inverse transform coincides with the form of the deconvolution formula (also called fractionally strided convolution) from $M$ to 1 channel using a constant filter weight $\kappa'_{mij} = \psi_2(m, i, j)$ that

$$V'_{cpq} = \sum_{m=0}^{M-1} \sum_{i=0}^{N_a-1} \sum_{j=0}^{N_b-1} \hat{V}'_{cm(p+i)(q+j)} \psi_2(m, i, j), \quad (6)$$

where $p = 1, \ldots, N_a$; $q = 1, \ldots, N_b$; $\psi_2$ is the 2-D discretized version of $\psi$; $\hat{V}' \in \mathbb{R}^{d_v \times M \times (N_a + N - 1) \times (N_b + N - 1)}$ is the expanded and padded tensor of $\hat{V}'$ that, $s - 1$ zeros are inserted between every two elements and $N - 1$ zeros are padded two-side on the last two dimensions representing the geometry. Actually, this implementation is convenient to code using deep learning frameworks like Pytorch [34] and achieve great computational efficiency with GPUs.
3.3 Receptive range and corrosion of the domain

Here gives the method to calculate the receptive range of the local neural operators constructed. For point-wise local operators, e.g., the linear transformations and the nonlinear activations, the receptive range is 0. The receptive range of convolutional layers (in Eq. (1)) is $K \Delta x$. As for the local spectral operations, the receptive range depends on the size of the square window $N \Delta x$. Since the spectral transform and its inverse both use all values of the function on the local domain, any two points inside the window are related. The points on the boundary of the square window have the minimum related range $\frac{k-1}{k} N \Delta x$, which is the receptive range of the spectral operators. The receptive range of a complete local neural operator is the combination of all the components: for series connected subparts, sum their $r_{rcp}$ together; for parallel connected subparts, $r_{rcp}$ is the largest receptive range of them.

Corrosion of the computational domain will occur when using the local neural operators to recurrently predict the solution functions of the transient PDEs, see Figure 4. It means the networks do not predict the solution function in areas near the boundary. The size of the corrosion area is just equal to the receptive range of the local neural operator. Actually, the occurrence of the corrosion is not surprising, because the local neural operators are defined to predict the field values at time $t+\Delta t$ using the field values in the surrounding area at time $t$. For areas near the boundary, some input values are missing. In other words, the solution operator in these areas is closely related to the boundary conditions and quite different from the middle. Section 4.2 introduces how we specifically deal with the boundary in practices.

4 Experiments

Experiments of this work mainly consist of two parts, which respectively correspond to the two steps to apply the proposed framework for learning transient PDEs. The first part is for network training and validation. The samples are generated and collected in a specified square domain with periodic boundary. This case is quite suitable for training because the periodic boundary allows infinite expansion for dealing with the issue of corrosion of the domain. The second part is for application. The networks trained in the first step are directly applied in solving problems with quite different computational domains, initial conditions, and boundary conditions. These applications are significant because it is the first success to directly apply a trained neural network to completely different problems without any additional training or optimization.

The local neural operators constructed here are with a lifting layer (a convolutional layer from $d_u$ to $d_v$ channels, where the kernel size is 3, and $d_v$ is set to 40), 4 blocks comprised of the spatial path and the spectral path, and the projection layers (two pointwise linear layers of $40-128-d_u$ channels with a nonlinear activation sandwiched). The physical path includes two convolutional layers with the kernel size of 3, and with a nonlinear activation
sandwiched. We adopt GELU [35] for all the activation. As for other parameters, \( d_u \) is determined by the problem; \( M, N, k \) of the spectral path are specific for networks, so we show them together with the results.

We conduct the experiments on several typical PDEs, including the viscous Burgers equations, wave equation, and N-S equations. Details about the data generation of these problems are in Appendix C. The following results and discussions are focusing on the N-S equations, which are representative transient PDEs with strong nonlinearity, and with great prospects of application in scientific computation and engineering simulation. Results of other transient PDEs are in Appendix D.

We consider the 2-D N-S equations with no source term for a viscous, incompressible fluid flow as follows:

\[
\frac{\partial u(x,t)}{\partial t} + u(x,t) \cdot \nabla u(x,t) = -\nabla p(x,t) + \mu \nabla^2 u(x,t), \quad x \in \Omega, \ t > 0, \tag{7}
\]
\[
\nabla \cdot u(x,t) = 0, \quad x \in \Omega, \ t > 0, \tag{8}
\]
\[
u(x,0) = u_0(x), \quad x \in \Omega. \tag{9}
\]

Here \( u \in \mathbb{R}^2 \) is the field of velocity vector; \( u_0 \) is the arbitrary initial field of \( u \); \( \Omega \) is the computational domain with arbitrary boundary conditions. LNO predicts \( u_{t+\Delta t} \) using \( u_t \) as input, regarding the field of pressure \( p \) as an implicit variable inside the neural network.

### 4.1 Step I. Training and validation

We train LNO for learning Eqs. (7-9) on the benchmark problem in a square domain \([-1,1] \times [-1,1]\) with periodic boundary condition. We experiment on 3 viscosities of the equations, \( \mu = 0.01, 0.002, 0.001 \). Two networks based on the Legendre and Chebyshev polynomial are trained and validated on these three problems respectively. All the networks are trained following the same schedule of 200 epochs, which includes 500 iterations per epoch. In each iteration, the network is used recurrently to generate output of 10 steps. The loss function is defined as the sum of the \( L_2 \) loss of the 10 outputs. We adopt Adam with an initial learning rate of 0.001 as the optimizer, and the learning rate is multiplied by 0.7 every 20 epochs. Table 1 lists the results. A reproduction of the nonlocal neural operator FNO, which is also trained on the same data following the same schedule, is also included for comparison.

According to the results in Table 1, LNOs achieve comparable accuracy and get lower MSE in most scenarios (11 of 12 cases) compared to the FNO, especially, LNO-Legendre reserves the lowest MSE in 9 of 12 cases. Note that LNOs are trained based on the local learning problem definition, which means the network outputs are only according to the local values of the input functions. On the contrary, the nonlocal neural operator FNO adopts an infinite receptive range \( r_{rcp} \). Figure 5 shows a visualization of the case \( \mu = 0.002 \) comparing the velocities calculated via the implicit FEM (the ground truth) and the velocities predicted by LNO-Legendre. Figures of the other two viscosities
are in Appendix D. All these results and comparisons undoubtedly confirm our local-related hypothesis of transient PDEs and our definition of the learning problem (see Section 2). Meanwhile, the success of learning the transient PDEs as local neural operators allows maximum flexibility of the LNOs to adapt any computational domain and boundaries for application. We send our best network, LNO-Legendre, to the next step for application.

4.2 Step II. Application

This part is to directly apply the trained networks without any additional training or optimization for solving problems with various computational domains and boundaries. The specific adopted network is the LNO-Legendre with a viscosity of 0.01. It is necessary to declare the application scope of this trained model again that, according to the learning problem definition, the well-trained LNOs predict solutions of any problems without considering the computational domain, the initial condition, or the boundary condition, as long as the problem is governed by the same transient PDEs. On this basis, the only remaining issue for establishing a generic solver is to deal with the boundary properly, i.e., to deal with the corrosion of the domain (see Section 3.3). The following cases of application, for one thing, introduce two strategies to our neural solver for dealing with the boundaries, for another thing, show the flexibility of LNOs in different problems as an experimental proof of our learning problem definition.
| Viscosity | Model                        | Parameters                  | **Number of trainable weights | MSE error  |
|-----------|------------------------------|-----------------------------|-----------------------------|------------|
|           |                              |                             |                             | 0.5s | 1s   | 2s   | 5s   |
| 0.01      | *FNO                         | \( r_{rcp} = \infty \)     | 926326                      | 0.115  | 0.173 | 0.242 | 0.270 |
|           | LNO-Chebyshev (ours)         | \( N = 12, M = 6, k = 2, r_{rcp} = 25 \) | 328656                      | 0.102  | 0.135 | **0.178** | 0.269 |
|           | LNO-Legendre (ours)          |                             |                             | **0.097** | **0.120** | 0.181 | **0.200** |
| 0.002     | *FNO                         | \( r_{rcp} = \infty \)     | 926326                      | 0.266  | 0.504 | 0.943 | 1.440 |
|           | LNO-Chebyshev (ours)         | \( N = 16, M = 8, k = 2, r_{rcp} = 33 \) | 776656                      | 0.246  | **0.482** | 0.977 | 1.859 |
|           | LNO-Legendre (ours)          |                             |                             | **0.235** | 0.489 | **0.873** | **1.432** |
| 0.001     | *FNO                         | \( r_{rcp} = \infty \)     | 926326                      | **0.297** | 0.590 | 1.419 | 2.981 |
|           | LNO-Chebyshev (ours)         | \( N = 24, M = 8, k = 2, r_{rcp} = 49 \) | 776656                      | 0.350  | 0.704 | 1.590 | 1.982 |
|           | LNO-Legendre (ours)          |                             |                             | 0.302  | **0.579** | **1.232** | **1.937** |

* Reproduced, trained, and tested by us in our dataset.

** The complex weights of FNO are counted twice.
Fig. 6 Flow in a lid-driven cavity ($Re = 1000$). In (a), $\Omega_1$ and $\Omega_2$ respectively denote the regions computed by LNO-Legendre and FEM. The two regions are separated by the red dotted line which is also shown in (b) and (d). The reference solution is purely computed by the implicit FEM solver.

4.2.1 Internal flow: the lid-driven cavity

As shown in Figure 6(a), the fluid in a square cavity $\Omega = [0, 1] \times [0, 1]$ is driven by a constant velocity on the upper side. To deal with the corrosion of boundary (see Section 3.3), the cavity (computational domain) is split into two regions $\Omega_1$ (54.7% area of $\Omega$) and $\Omega_2$. LNO-Legendre and $P_2P_1$ FEM with implicit Euler scheme are used to respectively predict/compute the velocity fields in the two regions. This hybrid strategy takes three steps for recurrent computation.

1. Predict $u_{t+\Delta t}$ in $\Omega_1$ using $u_t$ in $\Omega$ as input by LNO-Legendre.
2. Calculate $u_{t+\Delta t}$ in $\Omega_2$ by the implicit FEM solver. The boundary condition used in the second step is provided by the output of LNO.
3. Combine the outputs of the two methods and obtain the complete field $u_{t+\Delta t}$ in $\Omega$.

Figure 6 depicts the results of velocity fields. As shown in Figure 6(d), vortex structure is captured correctly that, a main vortex occupies the center of the cavity and two small vortices are located at the lower left and right
corners. In Figure 6(b)-(c), the numerical values of the velocity are also in good agreement with the reference value. It costs 4.780 seconds (0.005/4.775 seconds spent by LNO-Legendre/FEM solver) for the hybrid strategy or 9.649 seconds for only using the implicit FEM solver to predict one step. This hybrid strategy speedup the computation in $\Omega_1$ by 912 times. All these results together demonstrate the effectiveness to directly apply the trained LNO-Legendre on the internal flow problems. The difference between the two solutions could be caused by the deviation from the incompressible condition (Eq. (8)) of LNO-Legendre, which could be reduced by improving the network architecture or adopting physics-informed fashion in network training [3, 22].

4.2.2 External flow around objects

External flow around objects is another kind of typical cases which is commonly appeared in engineering, e.g., the flow around aircrafts or wind turbines. In these cases, rigid obstacles are placed in an infinite plane with freestream comes from one side. The common boundary conditions in these cases, e.g., the far field boundary and the periodic boundary (for the cases of cascade), allows to extend the computational domain by padding (constant padding for the far field boundary and periodic padding for the periodic boundary). Concretely, the computational domain $\Omega$ is expanded to $\Omega \cup \Omega_1$, hence the trained LNO-Legendre is applied to predict $u_{t+\Delta t}$ in $\Omega$ with $u_t$ in $\Omega \cup \Omega_1$ as the input. To introduce the effect of the obstacle, we impose a velocity correction $\Delta u$ to points near the obstacle for the following two conditions: 1) the point is inside the obstacle or on the solid wall, set $\Delta u = -u$, i.e., set the velocities to zero; 2) the point is outside the obstacle but the shortest distance to the solid wall is smaller than the grid size $\Delta x$, $\Delta u$ is calculated by the immersed boundary method (IBM) [36, 37] (see Appendix E for more details). The second condition occurs when the obstacle is in irregular shapes, which is further demonstrated in Case 2.

- **Case 1. Flow around a square cylinder**

  For Case 1, as shown in Figure 7(a), the obstacle is a square cylinder with size 1, the inflow angle $\alpha = 0$. The initial condition is the freestream flow. Figure 7(b) shows the contour of velocity magnitude $\|u\|_2$. The reference value is calculated by characteristic-based split scheme [27] with unstructured triangular meshes (totally 5271 nodes). In general, LNO-Legendre predict most of the flow pattern accurately compared with the reference solution. Under the influence of the cylinder, the fluid is ‘squeezed’, resulting in the high-speed zones on the upper and lower sides. At the same time, the flow separation generates the low-speed zone behind the cylinder. In Figure 7(b), the evolving history predicted by LNO-Legendre from the initial condition to the steady state is also in good agreement with the reference solution. Remarkably, predicting the solution with LNO-Legendre is much faster (1523 times) than the conventional solver that the time cost to obtain the field of steady state is 6.50 seconds and 2.76 hours respectively.
• **Case 2. Flow across plane cascade of airfoils**

  Case 2 considers a series of NACA0012 airfoils with chord length 1 arranged at equal intervals. The periodic boundary condition is set on the upper and lower sides. The initial condition is a freestream flow with an angle of attack $\alpha = 10^\circ$. We introduce the effect of airfoils using a velocity correction $\Delta u$ calculated by IBM [36, 37] in the direct forcing form. The reference value is calculated by characteristic-based split scheme with body-fitted unstructured triangular meshes (totally 10691 nodes). Figure 8 shows the contour of the velocity magnitude. Similar to Case 1, LNO-Legendre predict most of the flow pattern accurately compared with the reference solution. The flow field is divided into the high-speed region on the both sides of the airfoil and the low-speed region behind the airfoil. The presence of the positive angle of attack makes the flow separation slightly biased towards the upper side of the airfoil. Also, LNO-Legendre captures the interaction between the periodic airfoils precisely. Remarkably, predicting the solution with LNO-Legendre is much faster (2191 times) than the conventional solver that the time cost to obtain the field of steady state is 2.68 seconds and 1.63 hours respectively.

5 **Conclusion**

This work proposed a practical framework to learn the transient PDEs with local neural operators. The highlight of this framework is the local learning
Local Neural Operators

(a) Schematic diagram

(b) Contours of the solution

Fig. 8 Flow across plane cascade of airfoils. The predicted solution is by LNO-Legendre without any additional modification.

problem definition based on the local-related hypothesis of transient PDEs, which solves the crux of neural networks towards wide applications. Without any retraining or optimization, our well-trained neural networks reserve maximum flexibility to solve problems with various computational domains, initial conditions, and boundary conditions. The experiments provided a paradigm to train, validate, and apply the local neural operators. Specifically, LNO-Legendre is constructed and trained using a spectral transform of Legendre polynomials. This neural network learned the Navier-Stokes equations well and successfully predicted solutions of fluid flow with completely different domains and boundaries, including the internal and external flows. Notably, LNO-Legendre shows a speedup over 1000 times compared to conventional finite element solvers. It could be significant for applications in scientific computations and engineering.
This study also obtained an encouraging success in training neural networks to summarize the laws of physics (the transient PDEs) by observation (the data samples), which is exactly what human researchers have been doing, because no prior knowledge is provided to the neural network in training besides the data samples.

Within the proposed framework, there are many valuable future directions. One is to handle the issue of corrosion of the computational domain. Besides the specific hybrid strategy used here, more efficient methods (especially neural methods) are expected. Another direction is to study the transferability of the well-trained LNOs between transient PDEs with different parameters (i.e., the Reynolds numbers for the N-S equations). Any improvement on this problem is meaningful for extending the range of extrapolation of LNOs.

**Fundings.** This work was supported by National Natural Science Foundation of China (Grant No.52176043) and National Science and Technology Major Project (2019-VII-0007-0147).

**References**

[1] Lagaris, I. E., Likas, A. & Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks* 9 (5), 987–1000 (1998). https://doi.org/10.1109/72.712178, https://arxiv.org/abs/9705023 [physics].

[2] Psichogios, D. C. & Ungar, L. H. A hybrid neural network-first principles approach to process modeling. *AIChE Journal* 38 (10), 1499–1511 (1992). URL https://onlinelibrary.wiley.com/doi/10.1002/aic.690381003. https://doi.org/10.1002/aic.690381003 .

[3] 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. *Journal of Computational Physics* 378, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045 .

[4] Raissi, M., Yazdani, A. & Karniadakis, G. E. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science* 367 (6481), 1026–1030 (2020). URL https://www.science.org/doi/10.1126/science.aaw4741. https://doi.org/10.1126/science.aaw4741 .

[5] Chen, Z., Liu, Y. & Sun, H. Physics-informed learning of governing equations from scarce data. *Nature Communications* 12 (1), 1–13 (2021). https://doi.org/10.1038/s41467-021-26434-1, https://arxiv.org/abs/2005.03448 .
[6] Meng, X. & Karniadakis, G. E. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *Journal of Computational Physics* **401** (1), 1–29 (2020). https://doi.org/10.1016/j.jcp.2019.109020, https://arxiv.org/abs/1903.00104.

[7] Yang, L., Meng, X. & Karniadakis, G. E. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics* **425**, 1–32 (2021). https://doi.org/10.1016/j.jcp.2020.109913, https://arxiv.org/abs/2003.06097.

[8] Jagtap, A. D., Kharazmi, E. & Karniadakis, G. E. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering* **365**, 113028 (2020). URL https://doi.org/10.1016/j.cma.2020.113028.

[9] Meng, X., Li, Z., Zhang, D. & Karniadakis, G. E. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering* **370**, 1–17 (2020). https://doi.org/10.1016/j.cma.2020.113250, https://arxiv.org/abs/1909.10145.

[10] Wang, S., Yu, X. & Perdikaris, P. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics* **449**, 1–29 (2022). https://doi.org/10.1016/j.jcp.2021.110768, https://arxiv.org/abs/2007.14527.

[11] Iten, R., Metger, T., Wilming, H., Del Rio, L. & Renner, R. Discovering Physical Concepts with Neural Networks. *Physical Review Letters* **124** (1), 10508 (2020). URL https://doi.org/10.1103/PhysRevLett.124.010508.

[12] Mao, Z., Jagtap, A. D. & Karniadakis, G. E. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering* **360**, 112789 (2020). URL https://doi.org/10.1016/j.cma.2019.112789.

[13] Goswami, S., Anitescu, C., Chakraborty, S. & Rabczuk, T. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics* **106**, 1–21 (2020). https://doi.org/10.1016/j.tafmec.2019.102447, https://arxiv.org/abs/1907.02531.
[14] Wang, R., Kashinath, K., Mustafa, M., Albert, A. & Yu, R. Towards Physics-informed Deep Learning for Turbulent Flow Prediction. Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1457–1466 (2020). https://doi.org/10.1145/3394486.3403198, https://arxiv.org/abs/1911.08655.

[15] Chen, Y., Lu, L., Karniadakis, G. E. & Dal Negro, L. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Optics Express 28 (8), 11618 (2020). https://doi.org/10.1364/oe.384875, https://arxiv.org/abs/1912.01085.

[16] Geneva, N. & Zabaras, N. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. Journal of Computational Physics 403 (2020). https://doi.org/10.1016/j.jcp.2019.109056, https://arxiv.org/abs/1906.05747.

[17] Sahli Costabal, F., Yang, Y., Perdikaris, P., Hurtado, D. E. & Kuhl, E. Physics-Informed Neural Networks for Cardiac Activation Mapping. Frontiers in Physics 8 (February), 1–12 (2020). https://doi.org/10.3389/fphy.2020.00042.

[18] Erichson, N. B., Muehlebach, M. & Mahoney, M. W. Physics-informed Autoencoders for Lyapunov-stable Fluid Flow Prediction 1–14 (2019). URL http://arxiv.org/abs/1905.10866, https://arxiv.org/abs/1905.10866.

[19] Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nature Machine Intelligence 3 (3), 218–229 (2021). https://doi.org/10.1038/s42256-021-00302-5.

[20] Li, Z. et al. Fourier Neural Operator for Parametric Partial Differential Equations (2020). URL http://arxiv.org/abs/2010.08895, https://arxiv.org/abs/2010.08895.

[21] Li, Z. et al. Markov Neural Operators for Learning Chaotic Systems 1–18 (2021). URL http://arxiv.org/abs/2106.06898, https://arxiv.org/abs/2106.06898.

[22] Li, Z. et al. Physics-Informed Neural Operator for Learning Partial Differential Equations (2021). URL http://arxiv.org/abs/2111.03794, https://arxiv.org/abs/2111.03794.

[23] Gupta, G., Xiao, X. & Bogdan, P. Multiwavelet-based Operator Learning for Differential Equations 1–31 (2021). URL http://arxiv.org/abs/2109.13459, https://arxiv.org/abs/2109.13459.
Local Neural Operators

[24] Jiang, P. et al. Digital Twin Earth – Coasts: Developing a fast and physics-informed surrogate model for coastal floods via neural operators 1–6 (2021). URL http://arxiv.org/abs/2110.07100. https://arxiv.org/abs/2110.07100.

[25] Pathak, J. et al. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators (2022). URL http://arxiv.org/abs/2202.11214. https://arxiv.org/abs/2202.11214.

[26] Zienkiewicz, O. C., Taylor, R. L. & Zhu, J. The Finite Element Method: Its Basis and Fundamentals Seventh ed edn (Elsevier Inc., 2013).

[27] Zienkiewicz, O. C., Taylor, R. L. & Nithiarasu, P. The Finite Element Method for Fluid Dynamics: Seventh Edition Seventh ed edn (Elsevier Inc., 2013).

[28] Shen, J., Tang, T. & Wang, L.-L. Spectral Methods: Algorithms, Analysis and Applications Vol. 41 of Springer Series in Computational Mathematics (Springer Berlin Heidelberg, Berlin, Heidelberg, 2011). URL https://www.springer.com/gp/book/9783540306634http://link.springer.com/10.1007/978-3-540-71041-7.

[29] Patera, A. T. A spectral element method for fluid dynamics: Laminar flow in a channel expansion. Journal of Computational Physics 54 (3), 468–488 (1984). https://doi.org/10.1016/0021-9991(84)90128-1.

[30] Ye, X., Qin, G. & Wang, Y. An accurate triangular spectral element method-based numerical simulation for acoustic problems in complex geometries. International Journal of Aeroacoustics 19 (3-5), 158–190 (2020). https://doi.org/10.1177/1475472X20930647.

[31] Huang, H., Liu, Y. & Yang, V. Neural Networks with Inputs Based on Domain of Dependence and A Converging Sequence for Solving Conservation Laws, Part I: 1D Riemann Problems 1–42 (2021). URL http://arxiv.org/abs/2109.09316. https://arxiv.org/abs/2109.09316.

[32] Zheng, J., Aono, T., Uchida, S., Zhang, J. & Honda, M. C. Distribution of Pu isotopes in marine sediments in the Pacific 30 km off Fukushima after the Fukushima daiichi Nuclear Power Plant accident. Geochemical Journal 46 (4), 361–369 (2012). https://doi.org/10.2343/geochemj.2.0209.

[33] Cheung, K. F., Bai, Y. & Yamazaki, Y. Surges around the Hawaiian Islands from the 2011 Tohoku Tsunami. Journal of Geophysical Research: Oceans 118 (10), 5703–5719 (2013). https://doi.org/10.1002/jgrc.20413.
[34] Paszke, A. et al. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems* **32** (NeurIPS) (2019). https://arxiv.org/abs/1912.01703.

[35] Hendrycks, D. & Gimpel, K. Gaussian Error Linear Units (GELUs) 1–9 (2016). URL http://arxiv.org/abs/1606.08415. https://arxiv.org/abs/1606.08415.

[36] Peskin, C. S. in *The immersed boundary method* 479–518 (Cambridge University Press, 2002). URL https://www.cambridge.org/core/product/identifier/CBO9780511550140A011/type/book_part.

[37] Uhlmann, M. An immersed boundary method with direct forcing for the simulation of particulate flows. *Journal of Computational Physics* **209** (2), 448–476 (2005). https://doi.org/10.1016/j.jcp.2005.03.017, https://arxiv.org/abs/1809.08170.

[38] Gladwell, I. & Thomas, R. Stability properties of the Newmark Houbolt and Wilson e methods. *International Journal for Numerical and Analytical Methods in Geomechanics* **4**, 143–158 (1980).

[39] Deville, M., Fischer, P. & Mund, E. *High-Order Methods for Incompressible Fluid Flow* (Cambridge University Press, 2002).
Appendix A  Local-related hypothesis in the view of Green’s function

Here, we take the wave equation and the convection-diffusion equation as examples.

a) The 1-D wave equation:
\[
\frac{\partial^2 p}{\partial t^2} - a_0^2 \frac{\partial^2 p}{\partial x^2} = 0 \quad (A1)
\]
describes the pressure wave propagates at the sound speed \( a_0 \). Green’s function of it is defined as the solution under a unit pulse initial condition: \( p(x, 0) = \delta(x), \dot{p}(x, 0) = 0 \) as:
\[
G(x, \Delta t) = \frac{1}{2}(\delta(x - a_0\Delta t) + \delta(x + a_0\Delta t)). \quad (A2)
\]

Eq. (A2) suggests that the pressure at \( t = 0, x = 0 \) makes an influence on \( x \in [-a_0\Delta t, a_0\Delta t] \). Conversely, the pressure value at \( t = \Delta t, x = 0 \) depends on \( p \) in \( x \in [-a_0\Delta t, a_0\Delta t] \), i.e., \( r_{\text{dep}} = a_0\Delta t \).

b) the 1-D convection-diffusion equation:
\[
\frac{\partial f}{\partial t} + U \frac{\partial f}{\partial x} = \mu \frac{\partial^2 f}{\partial x^2}, \quad (A3)
\]
describes the diffusion at viscosity \( \mu \) and convection at speed \( U \) of a field \( f \). Green’s function of it is defined as the solution under a unit pulse initial condition: \( f(x, 0) = \delta(x) \) as:
\[
G(x, \Delta t) = \frac{1}{\sqrt{4\pi \mu \Delta t}} \exp \left[ -\frac{(x - U\Delta t)^2}{4\mu \Delta t} \right], \quad \Delta t > 0. \quad (A4)
\]

Noticing that, though the velocity at \( x = 0 \) analytically affects everywhere inside the computational domain (\( G(x, \Delta t) > 0 \) when \( |x| < \infty \), i.e., \( r_{\text{dep}} \to \infty \)), \( G \) decreases exponentially as \( |x| \) grows, which enables a truncated range of dependence \( r_{\text{dep}}^\varepsilon \) numerically with the threshold \( \varepsilon \) satisfying
\[
G(r_{\text{dep}}^\varepsilon, t) = \varepsilon \max_x G(x, t), \quad (A5)
\]
so we have
\[
r_{\text{dep}}^\varepsilon = \sqrt{-\frac{4t}{\mu} \ln \varepsilon + Ut}. \quad (A6)
\]

For most transient PDEs that describe the physical laws of the real world, the range of dependence can be defined by the above-mentioned analytical way
or numerical way. Hence, the limited range of dependence enables our local-related hypothesis, and it leads to a tailored learning problem definition for transient PDEs.

Appendix B  An introduction to spectral transforms

Spectral transform is to map the function in the physical space to the modal components in the frequency space. A set of orthogonal polynomials and their corresponding Gauss quadrature nodes and weights are the most important elements to carry out the spectral transform. Here we introduce two widely used orthogonal polynomials, Legendre and Chebyshev polynomials.

B.1 Legendre polynomials

Let \( L_i \) denotes \( i^{th} \) Legendre polynomial:

\[
L_i(x) = \sum_{l=0}^{[i/2]} (-1)^l \frac{(2i - 2l)!}{(2l!)((i - l)!)(i - 2l)!} x^{i-2l},
\]

which are orthogonal with respect to \( \omega = 1 \) in \([-1, 1] \):

\[
\int_{-1}^{1} L_i L_j dx = \frac{2}{2i + 1} \delta_{ij}.
\]

There are no explicit expressions for the corresponding Gauss quadrature nodes \( x_i \) and weights \( \omega_i \). Usually, numerical approaches are applied to compute the nodes and weights.

B.2 Chebyshev polynomials

Let \( T_i \) denote \( i^{th} \) Chebyshev polynomial:

\[
T_i(x) = \cos \left( i \cos^{-1} x \right),
\]

which are orthogonal with respect to \( \omega = \frac{1}{\sqrt{1-x^2}} \) in \([-1, 1] \):

\[
\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} T_i T_j dx = \begin{cases} 
\pi \delta_{ij}, & i = 0, \\
\frac{\pi}{2} \delta_{ij}, & i > 0,
\end{cases}
\]

The corresponding \( N^{th} \)-order Gauss nodes and weights (in this paper the Gauss-Lobatto type quadrature is selected) are as follows:

\[
x_i = -\cos \frac{i\pi}{N}, \quad \omega_i = \frac{\pi}{ciN}, \quad 0 \leq i \leq N,
\]

A preprint (March 11, 2022)
where \( c_i = 1 \) except for \( c_0 = c_N = 2 \).

**B.3 Spectral transform**

A 1-D continuous function \( f(x) \) defined on \([-1, 1]\) can be approximated by the first \( N \) orthogonal polynomials as follows:

\[
    f(x) \approx \sum_{m=0}^{N-1} \hat{f}_m \phi_m(x),
\]

with

\[
    \hat{f}_m = \frac{\langle f, \phi_m \rangle_\omega}{\langle \phi_m, \phi_m \rangle_\omega},
\]

where \( \phi_m \) is the \( m \)-th order orthogonal polynomial; \( \hat{f}_m \) is the magnitude of the \( m \)-th mode; \( \langle \cdot, \cdot \rangle_\omega \) denotes the inner product with weight \( \omega \).

By comparing Eq. (5) and Eq. (B12), the normalized basis of the spectral transform is:

\[
    \varphi(m, x) = \frac{\omega \phi_m}{\langle \phi_m, \phi_m \rangle_\omega}.
\]

Similarly, the normalized basis of the reverse transform in Eq. (6) is as follows:

\[
    \psi(m, x) = \phi_m(x).
\]

**B.4 Discrete spectral transform**

When the 1-D function \( f(x) \) is given at a series of points \( \tilde{x}_i (i = 0, 1, 2, \ldots, N - 1) \), a spectral transform in the discrete form is required.

First, interpolate \( f(\tilde{x}_i) \) to the Gauss quadrature point:

\[
    f(x_k) = \sum_{i=0}^{N-1} a_{ki} f(\tilde{x}_i),
\]

here the coefficients \( a_{ki} \) depend on the interpolation order selected. For the spectral transform, replace the integration in Eq. (B13) to Gauss-Lobatto type quadrature:

\[
    \hat{f}_m = \frac{\sum_{k=0}^{N-1} \omega_k f(x_k) \phi_m(x_k)}{\sum_{k=0}^{N-1} \omega_k \phi_m^2(x_k)} = \frac{\sum_{k=0}^{N-1} a_{ki} f(\tilde{x}_i) \phi_m(x_k)}{\sum_{k=0}^{N-1} \omega_k \phi_m^2(x_k)},
\]

Then the discrete normalized basis can be found to be:

\[
    \varphi_1(m, \tilde{x}_i) = \frac{\sum_{k=0}^{N-1} \omega_k a_{ki} \phi_m(x_k)}{\sum_{k=0}^{N-1} \omega_k \phi_m^2(x_k)}.
\]
The discrete normalized basis of the inverse transform is:
\[ \psi_1 (m, \tilde{x}_i) = \phi_m (\tilde{x}_i). \] (B19)

For 2-D problems, the normalized basis \( \varphi_2, \psi_2 \) can be obtained by the product of \( \varphi_1, \psi_1 \) with respect to \( x \) and \( y \) axes:
\[
\begin{align*}
\varphi_2 (m, \tilde{x}_i, \tilde{y}_j) &= \varphi_1 (p, \tilde{x}_i) \varphi_1 (q, \tilde{y}_j), \\
\psi_2 (m, \tilde{x}_i, \tilde{y}_j) &= \psi_1 (p, \tilde{x}_i) \psi_1 (q, \tilde{y}_j),
\end{align*}
\] (B20) (B21)

where \( m = p \times (N + 1) + q - 1 \).

**Appendix C  Data generation**

**C.1 Governing equations**

Viscous Burgers equation:
\[
\frac{\partial u}{\partial t} + u \cdot \nabla u = \mu \Delta u.
\] (C22)

Wave equation:
\[
\frac{\partial^2 p}{\partial t^2} - a_0^2 \Delta p = 0.
\] (C23)

Incompressible N-S equation:
\[
\begin{align*}
\frac{\partial u}{\partial t} + u \cdot \nabla u &= -\nabla p + \mu \Delta u, \\
\nabla \cdot u &= 0.
\end{align*}
\] (C24) (C25)

**C.2 Discretization and sampling**

We generate the data for training and validation by conventional numerical schemes, the schemes for different PDEs are listed in Table C1. The problem is defined in a square domain \([-1, 1] \times [-1, 1]\) with periodic boundary condition. Equidistant mesh with spacing \( \Delta x = 1/64 \) is used, i.e., the total number of nodes is \( 128^2 \). The time step is \( \Delta t = 10^{-3} \), and the data is recorded every 50 steps. For each PDE, the dataset of training contains the physical field within 1000s (20000 time steps). We generate data samples for training using bootstrap, i.e., each sample is extracted according to a random \( t \) as the initial input. Additionally, 10 different random initial conditions are used to validate the trained network.

For viscous Burgers equation and N-S equation, the input and output are the velocity function \( u \) (\( u \) has 2 channels for 2-D cases). For wave equation, the input is \( p \) and \( \frac{\partial p}{\partial t} \), the output is \( p \). After one time step, \( \frac{\partial p^{n+1}}{\partial t} \) is approximated by \( \frac{p^{n+1} - p^n}{\Delta t} \) to serve as the input of the next step.
Table C1 The numerical schemes adopted for data generation.

| PDE                    | Temporal scheme     | Spatial scheme     |
|------------------------|---------------------|-------------------|
| Viscous Burgers equation | Backward Euler      | Linear FEM        |
| Wave equation          | Implicit Newmark [38]| Linear FEM        |
| Navier-Stokes equation | Backward Euler      | $P_2P_1$ FEM [39] |

C.3 Randomization of samples

We use random numbers to generate the initial conditions. For transient PDEs which have no restrictions on the initial condition, the initial condition is generated by:

\[
F(x, y) = [\sin \pi x \sin 2\pi x \cos \pi x \cos 2\pi x] \Theta [\sin \pi y \sin 2\pi y \cos \pi y \cos 2\pi y]^T,
\]

(C26)

where $\Theta = \{\theta_{ij}\} \ (i, j = 1 \sim 4)$ is a random number matrix with $\theta_{ij} \sim N(0, 1)$. Here $F$ denotes the variables required for the neural network training, $F = p$ for the wave equation (another initial condition $\frac{\partial p}{\partial t}$ is set as zero), $F = u$ for the viscous Burgers equation.

For transient PDEs which require the initial field to satisfy certain conditions, e.g., the incompressible N-S equation requires the velocity divergence being zero, the initial condition is generated by the source term. We start with zero initial condition and add a random source term Eq. (C26) to the right-hand side of Eq. (C24), then calculate for 0.05s. The results obtained are used as the random initial conditions for data generation.

C.4 Data augmentation

We augment the samples by coordinate transformations, including rotation and flip. There are 7 different transformations at most: rotation by $90^\circ, 180^\circ, 270^\circ$ and flip along $x = 0, y = 0, y = x, y = -x$.

Appendix D Performance of LNO-Legendre for different transient PDEs

Figures D1,D2 show the performance of the trained LNOs on N-S equation with $\mu = 0.01$ and 0.001. Different values of $\mu$ represent different proportion of convection and diffusion in the flow. Generally, smaller $\mu$ leads to more sophisticated flow patterns. By comparing Figures 5,D1 and D2, it can be found that LNO-Legendre is trained successfully to learn the laws of flow with different $\mu$.

LNOs is also trained for 1-D/2-D viscous Burgers equation and wave equation. The visualization of the validation is given in Figures D3-D5, and the accuracy is listed in Table D2. It is worth mentioning that there is significant gradient in the velocity contour of Burgers equation in Figure D4, which poses a serious challenge to the stability of the neural operator in recurrent
Table D2 Comparison between LNOs proposed herein and the nonlocal neural operator (FNO) in solving Burgers and wave equations.

| PDE           | Model            | Parameters          | **Number of trainable weights | MSE error     |
|---------------|------------------|---------------------|------------------------------|---------------|
|               |                  |                     |                              | 0.5s  | 1s   | 2s   | 5s  |
| 1-D Burgers   | *FNO             | \( r_{rcp} = \infty \) | 42957                        | 0.0030 | 0.0020 | 0.0013 | 0.0006 |
| \((\mu = 0.01)\) | LNO-Chebyshev(ours) | \( N = 12, M = 6, \) | 15228                        | 0.0025 | 0.0013 | 0.0009 | 0.0007 |
|               | LNO-Legendre(ours) | \( k = 2, r_{rcp} = 25 \) |                             | 0.0029 | 0.0013 | 0.0008 | 0.0005 |
| 2-D Burgers   | *FNO             | \( r_{rcp} = \infty \) | 926326                       | 0.141  | 0.115  | 0.098  | 0.099  |
| \((\mu = 0.01)\) | LNO-Chebyshev(ours) | \( N = 12, M = 6, \) | 328656                       | 0.080  | 0.071  | 0.070  | 0.100  |
|               | LNO-Legendre(ours) | \( k = 2, r_{rcp} = 25 \) |                             | 0.062  | 0.056  | 0.054  | 0.055  |
| Wave          | *FNO             | \( r_{rcp} = \infty \) | 926326                       | 0.049  | 0.047  | 0.063  | 0.035  |
| \((a_0 = 1)\)  | LNO-Chebyshev(ours) | \( N = 12, M = 4, \) | 162128                       | 0.050  | 0.048  | 0.065  | 0.038  |
|               | LNO-Legendre(ours) | \( k = 2, r_{rcp} = 25 \) |                             | 0.047  | 0.045  | 0.061  | 0.042  |

* Reproduced, trained, and tested by us in our dataset.
** The complex weights of FNO are counted twice.
Local Neural Operators

\[ t = 0.2 \] (4 iterations)

\[ t = 0.5 \] (10 iterations)

\[ t = 1 \] (20 iterations)

\[ t = 2 \] (40 iterations)

**Fig. D1** The predicted results of LNO-Legendre and the reference solution in solving 2-D incompressible N-S equations (\( \mu = 0.01 \)).

\[ t = 0.2 \] (4 iterations)

\[ t = 0.5 \] (10 iterations)

\[ t = 1 \] (20 iterations)

\[ t = 2 \] (40 iterations)

**Fig. D2** The predicted results of LNO-Legendre and the reference solution in solving 2-D incompressible N-S equations (\( \mu = 0.001 \)).
time marching process. However, LNO succeeds to predict the velocity field with high accuracy maintained.

Appendix E  Immersed boundary method (IBM)

IBM is an algorithm to deal with complex geometries in the computational domain, which allows for non-uniform Cartesian grids even in problems with curved boundaries [36, 37]. Figure E6 shows a sketch map for the mesh used in Section 4.2.2 where IBM is used, the grid points around the obstacle do not fall on the solid wall boundary.

The basic idea of IBM is to transform the effect of the boundary into an external volume force on the adjacent grid points. The main procedures of IBM in a basic direct forcing formulation are given as follows. First, calculate an intermediate velocity $u^*$ on the Cartesian grid points (named Euler points) using N-S equation without the solid wall boundary. Then, interpolate $u^*$ to the points on the airfoil curve (also called Lagrange points):

$$U^*(X) = \sum_{i=1}^{NG} u^*(x_i) \delta_h (x_i - X),$$  \hspace{1cm} (E27)

where $U^*$ denotes the intermediate velocity on the Lagrange points; $NG$ is the total number of Euler points; $\delta_h$ is an approximated delta function, in this paper the 4-point piecewise function is applied. Next, calculate the volume
Local Neural Operators

Fig. D4  The predicted results of LNO-Legendre and the reference solution in solving 2-D viscous Burgers equations ($\mu = 0.01$).

Fig. D5  The predicted results of LNO-Legendre and the reference solution in solving 2-D wave equations ($a_0 = 1$).

force $F$ using the boundary condition and interpolate back to the mesh points:

$$F (X) = \frac{U_{BC}(X) - U^*(X)}{\Delta t},$$  \hspace{1cm} (E28)

$$f (x) = \sum_{j=1}^{N_{\text{Lagrange}}} F (X_j) \delta_h (x - X_j),$$  \hspace{1cm} (E29)
Fig. E6  Euler points and Lagrange points of the immersed boundary method.

where $N^\text{Lagrange}_G$ is the total number of the Lagrange points. In the end, calculate the modified velocity considering the boundary:

$$u^{n+1} = u^* + f \Delta t.$$  \hspace{1cm} (E30)