Zero or Infinite Data? Knowledge Synchronized Machine Learning Emulation

Xihaier Luo
Brookhaven National Laboratory
xluo@bnl.gov

Wei Xu
Brookhaven National Laboratory
xuw@bnl.gov

Yihui Ren
Brookhaven National Laboratory
yren@bnl.gov

Shinjae Yoo
Brookhaven National Laboratory
sjyoo@bnl.gov

Balu Nadiga
Los Alamos National Laboratory
balu@lanl.gov

Ahsan Kareem
University of Notre Dame
Ahsan.Kareem.1@nd.edu

Abstract

Even when the mathematical model is known in many applications in computational science and engineering, uncertainties are unavoidable. They are caused by initial conditions, boundary conditions, and so on. As a result, repeated evaluations of a costly model governed by partial differential equations (PDEs) are required, making the computation prohibitively expensive. Recently, neural networks have been used as fast alternatives for propagating and quantifying uncertainties. Notably, a large amount of high-quality training data is required to train a reliable neural networks-based emulator. Such ground truth data is frequently gathered in advance by running the numerical solvers that these neural emulators are intended to replace. But, if the underlying PDEs' form is available, do we really need training data? In this paper, we present a principled training framework derived from rigorous and trustworthy scientific simulation schemes. Unlike traditional neural emulator approaches, the proposed emulator does not necessitate the use of a classical numerical solver to collect training data. Rather than emulating dynamics directly, it emulates how a specific numerical solver solves PDEs. The numerical case study demonstrates that the proposed emulator performed well in a variety of testing scenarios.

1 Introduction

Machine learning (ML) is a component of artificial intelligence and a computational technology that can be trained with data to augment or automate human skills. Across the science community, there has been a surge in interest in developing ML-based frameworks for climate prediction [27, 33], drug discovery [18, 44, 46], renewable energy development [17, 41], etc. However, if not well grounded, most ML models are purely data-driven. One impediment to data-driven modeling is that existing methods frequently do not meet the needs of scientific users. Consider the following:

Data Scarcity While ML has achieved remarkable success in *data-rich* applications, such as image recognition and natural language processing, modeling natural phenomena remains challenging as physical observations usually are limited and sparse. Worse yet, noiseless ground-truth data typically are unknown, and benchmark data sets are not always available for many scientific problems [13, 48].

NeurIPS 2022 AI for Science Workshop.

Discontinuity Current ML achievements are heavily reliant on *discretized* data, such as images of varying resolutions in computer vision. As a result, traditional model development has mainly centered on learning mappings between finite-dimensional Euclidean spaces or finite sets. On the other hand, most scientific application data are *continuous*. For example, air and water temperatures constantly influence atmospheric circulation patterns [11, 32].

Model Interpretability Human-interpretable models are required for gaining new scientific insights from data. However, modern ML models, particularly deep-learning-based models, are prone to over-parameterization. As a result, these models are frequently criticized for being opaque. Although important for scientific applications, interpreting these black box models remains challenging [34].

Attempts to answer these questions are commonly referred to as *physics-informed machine learning* [19, 35, 49]. Here, we rigorously ground ML models using physical principles via trustworthy scientific simulation procedures. This differs fundamentally from the existing body of research that attempts to combine ML and domain sciences, e.g., by using domain-specific knowledge in ML algorithms in simplistic ways, e.g., extended data loss. Specifically, this paper's main contributions are centered on answering the following two questions.

- *"Can we use explicit knowledge exclusively to train the ML model if the governing equation form is known but there are uncertainties in the initial condition, boundary conditions, or others?"* The proposed learning scheme enables the model to be trained without collecting training data beforehand. The training is derived from off-the-shelf numerical schemes and is performed in an on-the-fly learning mode.
- *"How can we determine and validate that the physics of interest is properly captured by the ML model?"* We use Explainable AI tools to examine the trained model, such as comparing the learned convolution filters with numerical integration coefficients.

2 Background and Related Work

In many scientific applications, it is reasonable to assume that the observed system follows a partial differential equation (PDE). Here, we assume the underlying PDE form is known but contains uncertainties in the initial condition, boundary conditions, or others. Without loss of generality, we are interested in PDEs with a single time dimension $t = [0, T]$ and possibly multiple spatial dimensions $\mathbf{x} = [x_1, x_2, \ldots, x_N]^\top \in \mathbb{X}$. These can be written in the form

$$\begin{aligned} &\partial_t \mathbf{u} = F\left(t, \mathbf{x}, \mathbf{u}, \partial_\mathbf{x}\mathbf{u}, \partial_{\mathbf{xx}}\mathbf{u}, \ldots\right) \\ &\mathbf{u}(0, \mathbf{x}) = \mathbf{u}^0(\mathbf{x}), \quad B[\mathbf{u}](t, x) = 0 \end{aligned} \tag{1}$$

where $\mathbf{u} : [0, T] \times \mathbb{X} \to \mathbb{R}^n$ is the solution, with initial condition $\mathbf{u}^0(\mathbf{x})$ at time $t = 0$ and boundary conditions $B[\mathbf{u}](t, x) = 0$ where $\mathbf{x}$ is on the boundary $\partial_\mathbf{x}$ of the domain $\mathbb{X}$. Specifically, the boundary operator B includes Dirichlet boundary conditions, where $B_\mathcal{D}[\mathbf{u}] = \mathbf{u} - \mathbf{b}_\mathcal{D}$ for fixed function $\mathbf{b}_\mathcal{D}$ and Neumann boundary conditions, where $B_\mathcal{N}[u] = \mathbf{n}^\top \partial_\mathbf{x} u - b_\mathcal{N}$ with $\mathbf{n}$ denoting an outward facing normal on $\partial\mathbb{X}$.

The question now is how to solve Eq. (1). Section 2.1 examines current efforts from a numerical *simulation* standpoint, while Section 2.2 reviews related work from an ML *emulation* standpoint.

2.1 Classic numerical simulation

The method of lines (MOL) is a common technique for solving PDEs stated in Eq. (1) [37]. It chooses N nodes in $\mathbb{X}$ and uses algebraic approximations to replace the spatial derivatives in F at these nodes. The discretization causes F to be approximated by $\hat{F}$, resulting in the following system of ordinary differential equations (ODEs):

$$\dot{\mathbf{u}}(t) = \begin{pmatrix} \dot{u}_1(t) \\ \vdots \\ \dot{u}_N(t) \end{pmatrix} = \begin{pmatrix} \frac{du(\mathbf{x}_1, t)}{dt} \\ \vdots \\ \frac{du(\mathbf{x}_N, t)}{dt} \end{pmatrix} \approx \begin{pmatrix} \hat{F}\left(\mathbf{x}_1, \mathbf{x}_{\mathcal{N}(1)}, u_1, u_{\mathcal{N}(1)}\right) \\ \vdots \\ \hat{F}\left(\mathbf{x}_N, \mathbf{x}_{\mathcal{N}(N)}, u_N, u_{\mathcal{N}(N)}\right) \end{pmatrix} \in \mathbb{R}^N \tag{2}$$

where $\mathcal{N}(i)$ is a collection of indices of neighboring nodes other than i that must be evaluated in order to compute $\hat{F}$ at $\mathbf{x}_i$ and $\mathbf{x}_{\mathcal{N}(i)}$ and $u_{\mathcal{N}(i)}$ are positions and states of nodes $\mathcal{N}(i)$. By doing so, MOL separates space discretization from time evolution. $\hat{F}$ can be formed in a variety of ways by approximating spatial derivatives on the grid and the ODE $\mathbf{u}(t) = \mathbf{u}(0) + \int_0^t \dot{\mathbf{u}}(\tau) d\tau$ for a fixed location $\mathbf{x}_i$ can be solved using standard ODE solvers. The following is a high-level overview of classical techniques for performing spatial differentiation and temporal integration.

Spatial Differentiation Traditionally, F is approximated by discretizing the space using traditional solvers such as finite difference methods (FDM) [43], finite volume methods (FVM) [22], and finite element methods (FEM) [15]. To represent the domain $\mathbb{X}$ by a union of N_{ele} mesh elements, a set of mesh points are introduced separately in the various space directions. Once the mesh has been established, it is common practice to fit a piecewise polynomial first and then use the derivatives of the fitted polynomial at any new off-mesh point to obtain spatial differentiation information. In this paper, we consider finite difference methods, where spatial derivative operators (e.g., ∂_x) are replaced with difference operators known as stencils in FDM. For example, $\partial_x u^k\big|_{x_i}$ can be approximated by $\left(u_{i+1}^k - u_i^k\right) / \left(x_{i+1} - x_i\right)$. *In our proposed learning scheme, such numerical approximation will be replaced by its deep learning counterpart, namely convolution filters.*

Temporal Integration To numerically integrate Eq. (2), the domain is typically discretized by a uniformly partitioned time mesh. Then, to approximate the exact solution at the mesh points, a time integration scheme is chosen. In general, we have a time step Δt and seek approximations, such as the forward Euler method $\mathbf{u}_{i=1} = \mathbf{u}_{i=0} + \Delta t \hat{F}\left(\mathbf{u}_{i=0}\right)$. The computational algorithm is then applied iteratively for $i = 1, 2, \ldots, N_K - 1$ and solves the ODEs. *We use autoregressive methods in our proposed learning scheme, which share the same recursive philosophy as such a numerical temporal integration scheme. A learnable neural surrogate function is used to approximate F.*

2.2 Deep learning emulation

Deep learning based models have been used to learn dynamical systems and PDEs in recent years, driven by the desire to reduce computational costs and/or learn new physical models. The variety of problem settings and applications has resulted in a diverse set of neural PDE emulators. Here, we review the current work on neural PDE emulators for the temporal PDEs stated in Eq. (1) from two perspectives: *Spatial Prediction* and *Temporal Projection*.

Spatial Prediction Inspired by the great success in computer vision, many neural PDE emulators are based on convolutional neural networks (CNN). Although CNN-based discrete learning can improve training efficiency significantly, CNNs struggle to handle irregular geometries with unstructured meshes and will require modifications and tuning for different resolutions and discretizations [2, 10, 20, 49]. Another approach is to train a fully-connected multilayer perceptron (MLP) to learn continuous functions (e.g., physics-informed neural networks (PINNs)). However, these neural PDE emulators are often instance-based and have limited generalizability. When applied to a new instance, a new neural network must be trained. To address this issue, advanced PINN variants have recently been proposed [19, 26, 31, 45]. In parallel, graph neural networks (GNNs) have shown excellent performance in dealing with unstructured simulation meshes [30, 36, 38]. In this paper, we use a GNN-based neural architecture to generate continuous spatial predictions [21, 24].

Temporal Projection Mapping from initial conditions to solutions at time t can be treated as an input-output mapping that can be learned through supervised learning (e.g., PINNs) [19, 24, 31, 45]. It is, however, especially difficult if the goal is to generate long-term trajectories. Most models, on the other hand, are next-step prediction models, which learn the next state $\mathbf{u}(t+1, \mathbf{x})$ in the trajectory from the current one $\mathbf{u}(t, \mathbf{x})$ [3, 8, 30, 36]. Nonetheless, attempting to fit a standard ML model, such as a recurrent neural network (RNN), directly on the long-time sequences causes the gradient to explode [28, 39]. One treatment method is to use neural ODEs. The model is trained using a continuous-time adjoint method, which necessitates numerical integration of the neural network used to capture the system dynamics, making training prohibitively expensive [6, 16]. We attempt to reformulate the training of a next-step prediction model in this paper. We use autoregressive methods with a chosen ML model to predict the next step and then constrain the prediction to satisfy the numerical integration schemes.

3 Methodology

3.1 Autoregressive forward prediction

In this work, we are focused on developing a ML surrogate model that can efficiently predict the given dynamical system for time-steps $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_N$ for a given initial state realization $\mathbf{u}_0$ and a set of boundary conditions. The forward prediction is performed in an autoregressive fashion. For example, the input-output relationship of the proposed autoregressive model for prediction $\hat{\mathbf{u}}$ at a given time index t_{n+1} is written as follows:

$$\hat{\mathbf{u}}_{\mathbf{n}+1} = \hat{F}\left(\mathbf{u}_{\mathbf{n}-\mathbf{k}}, \mathbf{u}_{\mathbf{n}-\mathbf{k}+1}, \ldots, \mathbf{u}_{\mathbf{n}}\right) \tag{3}$$

$\hat{F}$ is our learnable neural emulator. It is a function of the $k+1$ previous states. Many numerical time integration algorithms, such as Runge-Kutta methods, suggest that a model is more stable when the hyperparameter k, also known as the window size, is greater than 1. As a result, the forward prediction problem, particularly the first few steps, can be reparameterized as:

$$\begin{aligned}
\hat{\mathbf{u}}_1 &= \hat{F}\left(X_1\right), \quad X_1 = \{\mathbf{u}_0, \mathbf{u}_0, \ldots, \mathbf{u}_0\} \\
\hat{\mathbf{u}}_2 &= \hat{F}\left(X_2\right), \quad X_2 = \{\mathbf{u}_1, \mathbf{u}_0, \ldots, \mathbf{u}_0\} \\
&\cdots \\
\hat{\mathbf{u}}_i &= \hat{F}\left(X_i\right), \quad X_i = \{\mathbf{u}_{i-1}, \mathbf{u}_{i-2}, \ldots, \mathbf{u}_{i-1-k}\}
\end{aligned} \tag{4}$$

Combining Eq. (3) and Eq. (4), the state variable prediction $\hat{\mathbf{u}}$ at time t_n can be written as:

$$\hat{\mathbf{u}}_n = \underbrace{(\hat{F} \circ \cdots \circ \hat{F})}_{n \text{ times}}(X_1) \tag{5}$$

with $\circ$ denoting the composition operation. In comparison to the standard numerical time integration method, $\hat{F}$ is an easy-to-evaluate neural emulator. Once $\hat{F}$ has been trained, we can make inferences much faster. And the section that follows discusses how to train our neural emulator.

3.2 Random unrolling training

To train the network, we define the loss function as a sum of residuals at a sequence of time instances:

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{N_t} \sum_{i=1}^{N_t} (\mathbf{u}_i - \hat{\mathbf{u}}_i)^2 \tag{6}$$

where $\boldsymbol{\theta}$ denotes the learnable parameters of $\hat{F}$. In a departure from traditional data-driven deep learning approaches, the proposed training does not require a classical numerical solver to collect training data. Instead, we estimate the solution variable $\mathbf{u}$ on the fly using a pre-selected numerical time-integrator. Specifically, we can (1) use $\hat{F}$ to perform forward propagation and calculate $\hat{\mathbf{u}}_{i-1}$ and $\hat{\mathbf{u}}_i$; (2) compel $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_{i-1}$ to follow a time-stepping method and compute $\mathbf{u}_i$ using $\hat{\mathbf{u}}_i$ and $\hat{\mathbf{u}}_{i-1}$; and (3) minimize the difference between the model's predictions $\hat{\mathbf{u}}_i$ and a discrete numerical time-integrator's estimation $\mathbf{u}_i$.

To better illustrate the central idea, let's consider the standard forward Euler time integration scheme. Given the prediction at time t_{i-1}, the goal is to calculate the state variable at time t:

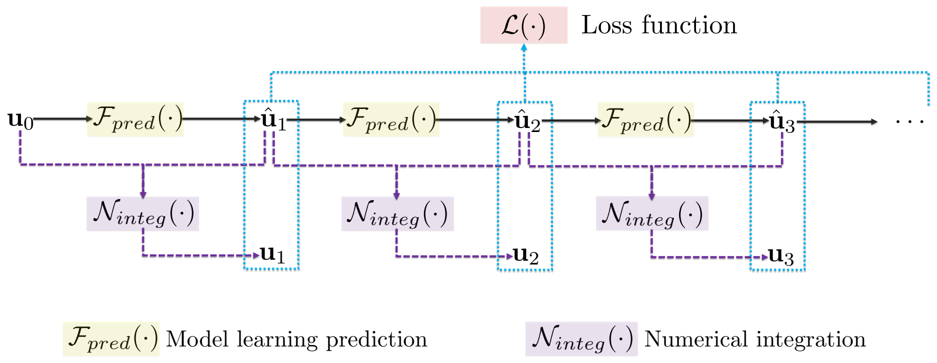
$$\frac{\mathbf{u}_i - \hat{\mathbf{u}}_{i-1}}{\Delta t} = S_{\Delta x}\left(\boldsymbol{x}, \hat{\mathbf{u}}_{i-1}\right) \tag{7}$$

where $S_{\Delta x}(\cdot)$ contains the spatial differentiation information. Substituting Eq. (7) into the loss function Eq. (6), we have

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{N_t} \sum_{i=1}^{N_t} \left(\hat{\mathbf{u}}_i - \hat{\mathbf{u}}_{i-1} - \Delta t S_{\Delta x}\left(\boldsymbol{x}, \hat{\mathbf{u}}_{i-1}\right)\right)^2 \tag{8}$$

From the standpoint of optimization, Eq. (8) is analogous to the L_2 minimization of the discretized PDE residual, where $\hat{\mathbf{u}}_i$ is provided by our neural emulator and $\hat{\mathbf{u}}_{i-1}$, which represents the previous state, is known. The only calculation left is for the spatial derivatives included in $S_{\Delta x}(\cdot)$. Because our emulator is a deep learning-based model, the spatial derivatives can be calculated using either auto-differentiation or numerical approximation [14, 29]. In this paper, we will use the second approach to speed the training up.

The emulator is trained by progressively incorporating loss terms estimated from different time steps to stabilize the training and offer reliable long-term predictions. If the aim is to forecast the dynamics for the next 20 steps $\mathbf{u}_1, \ldots, \mathbf{u}_{20}$ given the initial state $\mathbf{u}_0$, we will limit the model to gradually explore the system and learn the dynamics by gradually unrolling the number of time steps it predicts as training advances. We found that adding randomness to the unrolling number improves performance. The unrolling number, in particular, can be regarded as a uniformly distributed random number. We can also update the distribution, using a small number at first and allowing a larger number later. For example, if $N_t = 20$, a random realization of the unrolling number can be [2, 1, 3, 2, 4, 3, 5], implying that we only back-propagate 2 time-steps in the first step. Fig. 1 graphically demonstrates how the loss function is calculated.



$\mathcal{L}(\cdot)$ Loss function
$\mathbf{u}_0$
$\mathcal{F}_{pred}(\cdot)$
$\hat{\mathbf{u}}_1$
$\hat{\mathbf{u}}_2$
$\hat{\mathbf{u}}_3$
$\cdots$
$\mathcal{N}_{integ}(\cdot)$
$\mathbf{u}_1$
$\mathbf{u}_2$
$\mathbf{u}_3$
$\mathcal{F}_{pred}(\cdot)$ Model learning prediction
$\mathcal{N}_{integ}(\cdot)$ Numerical integration

Figure 1: Knowledge synchronized training of the proposed autoregressive model.

4 Experiments

4.1 Experiment setup

PDEs To highlight the benefits of the knowledge synchronized training strategy, we demonstrate its effectiveness on tasks of varying difficulty. PDEs considered in this paper can be written as

$$\left[\partial_t u + \partial_x \left(\alpha u^2 - \beta \partial_x u\right)\right](t, x) = f(t, x) \tag{9}$$

We have the heat equation if $\alpha \neq 0, \beta = 0$, and the Burgers' equation if $\alpha \neq 0, \beta \neq 0$. Specifically, the heat equation is used to validate the trained neural emulator's synchronized physics, and the Burgers' equation is used to demonstrate its computational and prediction performance. Appendix A contains detailed information on the set up of the PDEs.

Task The task is to emulate the underlying dynamics of the PDEs of interest (Eq. 9) using deep neural networks without using any simulation data. To make the emulation applicable to deep learning models, the continuous solution u is discretized in both the spatial and temporal domains, resulting in $U = \{u_0, u_1, u_2, \ldots, u_N\}$, where the initial state u_0 is random and the solution at other time instances must be inferred. As a result, we pose the emulation of a dynamical system as a forecasting problem. In other words, rather than learning a single solution, we are interested in learning the set of solutions for a distribution of random parameters, in our case, the random initial conditions.

Models The proposed training strategy is not constrained by the architecture of neural networks. Therefore, we experimented with two models: the standard convolutional neural network-based model (ResNet) [12] and the state-of-the-art Fourier Neural Operator (FNO) network [24]. After extensive hyperparameter optimization, we configure the two models as follows: *(a) ResNet* is a 27-layer residual convolutional neural network. The final dense layer is a convolutional layer with one output channel for the heat equation example and two output channels for the Burgers' equation example. *(b) FNO* has four layers and two fully connected networks (FCN). The first FCN lifts the dynamics from $\mathbb{R}^{d_{in}}$ to $\mathbb{R}^{20}$, and the last FCN projects the dynamics from $\mathbb{R}^{20}$ to $\mathbb{R}$ or $\mathbb{R}^2$. Also, each Fourier layer is parameterized with 12 truncated modes for the matrix-vector multiplication involved in the kernel computation.

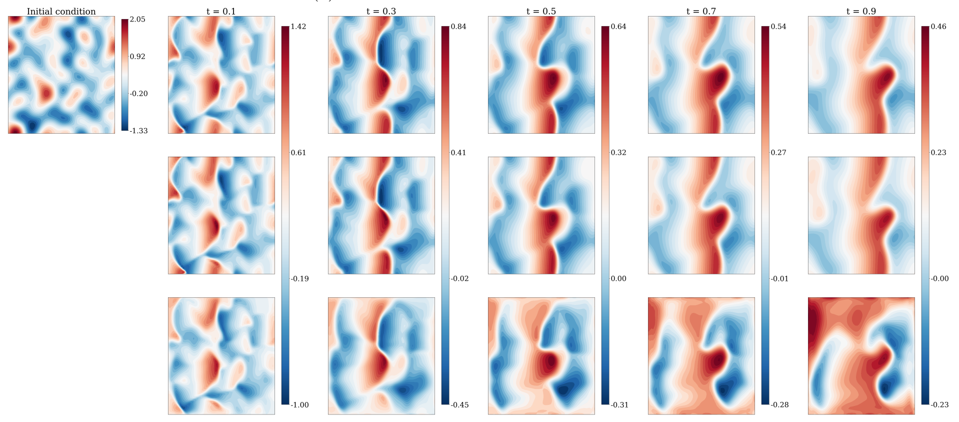
Training As discussed in section 3, emulating a numerical solver in solving a set of PDEs requires two key components: spatial derivative computation and time integration. Numerically, the finite difference can be used to approximate a derivative to an arbitrary order of accuracy for spatial derivatives. Lagrange interpolation is commonly used to determine the finite difference coefficients involved in the approximation. Here, we need to compute the first- and second-order derivatives given the heat/Burgers' equation. We use the Sobel and Laplace filters to approximate the first- and second-order derivatives with second-order accuracy to emulate the finite difference approximation [5, 42]. Next, we can quickly assemble the $S_{\Delta x}$ defined in Eq. 7 using calculated spatial derivatives. The loss function can then be crafted using a predetermined time integration method (e.g., the forward Euler scheme used in 8). Appendices B and C contain the detailed calculation procedures.

We generate $N_{train} = 8192$ training instances and another $N_{test} = 512$ testing instances for all experiments, where an instance denotes an initial condition. Note each experiment includes two regions, namely, the training $[0, 0.5]$ and the extrapolation $(0.5, 1]$. That is, N_t in Eq. 8 is 100, and the trained model is used to predict the next 100 time steps. Overall, the dynamics are projected 200 steps in time by the emulator. An Adam variant called *decoupled weight decay regularization* [25] is adopted as the optimizer with a weight decay $\lambda = 0.0001$. Both models are trained for 120 epochs with an initial learning rate of 0.001 that decays by $\gamma = 0.995$ every epoch.

4.2 Prediction performance

Qualitative assessment We begin by providing a qualitative assessment of the prediction performance. Fig. 2 shows the emulation results of a randomly chosen realization from the test dataset. Obviously, the adopted FNO model outperforms the state-of-the-art ResNet model trained on the same amount of initial condition realizations. In Fig. 2, it can be observed that both models are able to provide satisfactory predictions in the predefined training time period. However, ResNet's performance degrades significantly in the extrapolation region. ResNet's predictions deviate from the ground truth, while the prediction results of FNO still can maintain relatively high accuracy. These results reflect FNO's superior ability to absorb underlying physical principles, which supports better generalization of unknown scenarios [23, 30, 36].

(a) Simulation and emulation results of u



(b) Simulation and emulation results of v

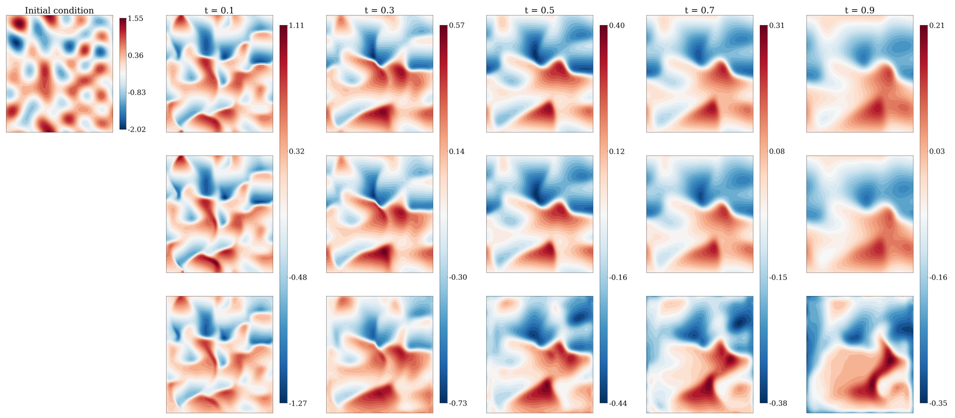
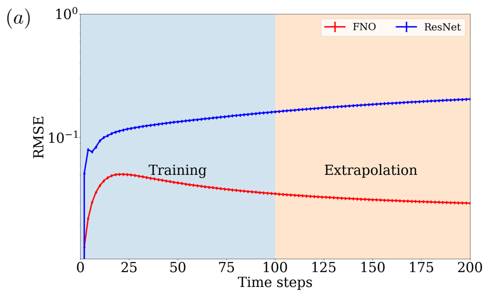


Figure 2: Burgers' equation emulation results. The first row in both subfigures shows the reference dynamics, the second row shows the FNO emulation results, and the third row shows the ResNet emulation results. Appendix 1 contains additional qualitative results for other PDEs.

Quantitative assessment Next, to quantitatively test the present scheme's precision, the root-mean-square errors (RMSEs) are reported. We tested 512 different samples of the fluid realizations. Fig.3 (a) shows the mean and standard deviation of the calculated RMSEs. It is observed that FNO outperforms ResNet by an order of magnitude when it comes to controlling error propagation. Although both models' accuracy decreases at a similar rate in the extrapolation region, the FNO model has a significantly lower mean value of RMSEs, particularly in the training region. Furthermore, the FNO model has a lower standard deviation than ResNet. This suggests that the FNO model is more reliable and robust in emulating Burgers' equation.

Computational efficiency Here, a study of the proposed model's computational efficiency is carried out. Two computation platforms are scheduled: (1) an Intel Xeon Processor E5-2698 v3 with a system memory of 187 GB and (2) a NVIDIA A100 GPU with 40 GB HBM2 memory. The total computational cost is classified into two broad categories: training and testing. First, the computational cost of DL-based emulators is compared against the standard numerical simulation. The computation time for a single simulation/emulation of the 2D coupled Burgers' equation on a 64×64 spatial gridded domain is shown in Fig.3 (b). Utilizing the same CPU, the speedup of the FNO model is approximately more than 100 times compared to the numerical solver for a single run. More significant saving on the computational effort is achieved if inferences are made using the GPU device. A speedup of three to four orders of magnitude is accomplished. These encouraging results show the potential of using a DL-based emulator for problems involving massive queries of the simulation model.



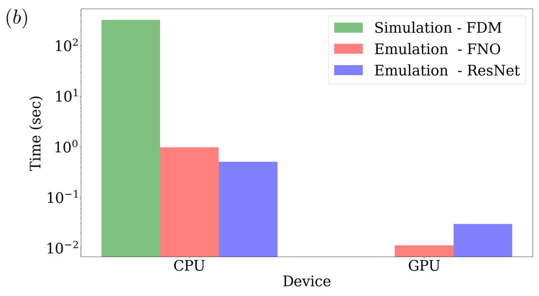


Figure 3: (a) Quantitative evaluation results of the Burgers' equation; (b) Computation time for different experiments. The time refers to the wall clock time for inferring one sample.

4.3 Generalization performance

Out-of-distribution Prediction Fig.4 (a) shows the mean RMSE of the emulation results. While both emulators are affected by *out-of-distribution* samples, the FNO model has lower errors. Notably, it is observed that FNO outperforms the ResNet in controlling the error propagation approximately by order of magnitude of 10. This effect gets more pronounced in the extrapolation region, where the mean RMSEs of the ResNet model increase as the system evolves. This finding is consistent with the intuitive expectation of using a surrogate model to forecast nonlinear dynamics. However, FNO is able to stabilize, even reduce, the prediction error within the same time region. In the meantime, it should be noted that FNO *out-of-distribution* prediction performance outperforms ResNet in-distribution test performance. These findings suggest that the FNO model possesses superior generalization performance than ResNet.

Super-resolution task We summarize the performance of the FNO and ResNet on super-resolution prediction tasks to test the model's ability to emulate a classic numerical solver [7]. The goal is to evaluate the generalization ability of these two emulators on various discretization meshes. ResNet and FNO are both trained on 64×64 meshes and tested at higher resolutions. Fig. 5 shows the results. Of note, the error is defined as the pixel-wise error averaged across all spatial grids of a given time instance. The emulation results show that because convolution kernels are spatial-agnostic and channel-specific, ResNet fails to provide accurate prediction if additional tuning for different resolutions is not available [9]. When evaluated at a higher resolution, the proposed FNO model, on the other hand, achieves consistent error. The promising results show that the FNO has greater generalization potential as the model is tailored to learn operators mapping between infinite-dimensional function spaces rather than standard finite-dimensional Euclidean spaces [36].

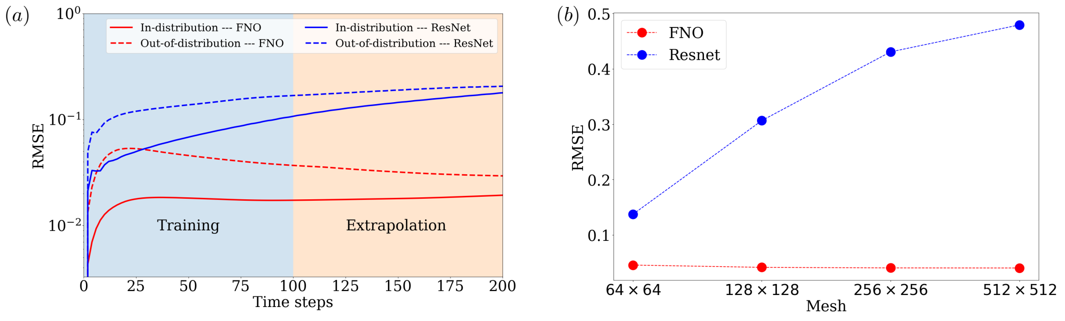


Figure 4: Generalization performance: (a) the mean of root-mean-square errors of *out-of-distribution* samples; (b) scaling performance on discretization from 64×64 up to 512×512.

4.4 Diagnosis of emulated physics

In order to diagnose the trained emulator and compare with the known physics, we employ one feature importance approach [47] for post-hoc local explanations of the behaviors of the emulator. This class of approach aims at highlighting the input features that contribute to the model prediction. Specifically, DeepLIFT [40] is adopted to depict the heatmaps showing both the spatial and temporal contributions to the prediction of an interested location. We demonstrate how the explanation works using the example of heat equation with the fixed $k = 3$ and a focused location in the center.

As illustrated in Fig.5 (a), three heatmaps represent contributing regions at corresponding timesteps t_{i-3}, t_{i-2}, and t_{i-1}, where blue shows the positive contribution while red shows the negative contribution. There are two observations about the spatial contributions: 1) the contributions decrease from the focused location to its periphery; and 2) the blue/red colors form a interweaving pattern that is consistent with the wave propagation. In Fig.5 (b), we aggregate the absolute values of the contribution scores for each temporal input (blue, red, and green representing t_{i-3}, t_{i-2}, and t_{i-1} respectively) and plot for all the test timesteps ($N = 100$) so as to compare which temporal channel has more impact to the prediction. We observe that the closer to the current timestep the more impact it brings, which complies with the physical principles.

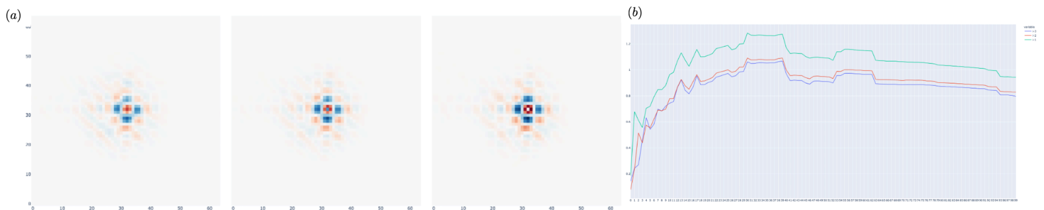


Figure 5: The (a) spatial and (b) temporal contributions of the input features.

5 Conclusion

In this work, we introduce a knowledge synchronized training strategy for building neural emulators for PDEs. There is no need to collect simulation data via a typical numerical solver. Instead, we cast the PDEs into a discretized form. The loss function is derived from variational principles. The explicit goal is to minimize the difference between the model's prediction and a pre-chosen numerical scheme. This is implicitly analogous to the L_2 minimization of the discretized PDE residual. The numerical case study shows the proposed emulator performed well in various testing scenarios.

References

[1] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[2] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, Karthik Duraisamy, and Shailendra Kaushik. Prediction of aerodynamic flow fields using convolutional neural networks. *Computational Mechanics*, 64(2):525–545, 2019.

[3] Johannes Brandstetter, Daniel E. Worrall, and Max Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022.

[4] Jian-Feng Cai, Bin Dong, Stanley Osher, and Zuowei Shen. Image restoration: total variation, wavelet frames, and beyond. *Journal of the American Mathematical Society*, 25(4):1033–1089, 2012.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6):679–698, 1986.

[6] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[7] Soheil Esmaeilzadeh, Kamyar Azizzadenesheli, Karthik Kashinath, Mustafa Mustafa, Hamdi A Tchelepi, Philip Marcus, Mr Prabhat, Anima Anandkumar, et al. Meshfreeflownet: a physics-constrained deep continuous space-time super-resolution framework. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE, 2020.

[8] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics*, 403:109056, 2020.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[10] Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.

[11] Yoo-Geun Ham, Jeong-Hwan Kim, and Jing-Jia Luo. Deep learning for multi-year enso forecasts. *Nature*, 573(7775):568–572, 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Tony Hey, Keith Butler, Sam Jackson, and Jeyarajan Thiyagalingam. Machine learning and big scientific data. *Philosophical Transactions of the Royal Society A*, 378(2166):20190054, 2020.

[14] Nicholas J Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[15] Thomas JR Hughes. *The finite element method: linear static and dynamic finite element analysis*. Courier Corporation, 2012.

[16] Valerii Iakovlev, Markus Heinonen, and Harri Lähdesmäki. Learning continuous-time {pde}s from sparse data with graph neural networks. In *International Conference on Learning Representations*, 2021.

[17] Sunil Kr Jha, Jasmin Bilalovic, Anju Jha, Nilesh Patel, and Han Zhang. Renewable energy: Present research and future scope of artificial intelligence. *Renewable and Sustainable Energy Reviews*, 77:297–317, 2017.

[18] Wengong Jin, Jeremy Wohlwend, Regina Barzilay, and Tommi S. Jaakkola. Iterative refinement graph neural network for antibody sequence-structure co-design. In *International Conference on Learning Representations*, 2022.

[19] George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

[20] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

[21] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces. *arXiv preprint arXiv:2108.08481*, 2021.

[22] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[23] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar, et al. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2020.

[24] Zongyi Li, Nikola Borislavov Kovachki, Kamyar Azizzadenesheli, Burigede liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. In *International Conference on Learning Representations*, 2021.

[25] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[26] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[27] Xihaier Luo, Balasubramanya T Nadiga, Yihui Ren, Ji Hwan Park, Wei Xu, and Shinjae Yoo. A bayesian deep learning approach to near-term climate prediction. *arXiv preprint arXiv:2202.11244*, 2022.

[28] Samuel E Otto and Clarence W Rowley. Linearly recurrent autoencoder networks for learning dynamics. *SIAM Journal on Applied Dynamical Systems*, 18(1):558–593, 2019.

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[30] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter Battaglia. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

[31] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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.

[32] Markus Reichstein, Gustau Camps-Valls, Bjorn Stevens, Martin Jung, Joachim Denzler, Nuno Carvalhais, et al. Deep learning and process understanding for data-driven earth system science. *Nature*, 566(7743):195–204, 2019.

[33] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2):1–96, 2022.

[34] Ribana Roscher, Bastian Bohn, Marco F Duarte, and Jochen Garcke. Explainable machine learning for scientific insights and discoveries. *Ieee Access*, 8:42200–42216, 2020.

[35] Esteban Samaniego, Cosmin Anitescu, Somdatta Goswami, Vien Minh Nguyen-Thanh, Hongwei Guo, Khader Hamdia, X Zhuang, and T Rabczuk. An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. *Computer Methods in Applied Mechanics and Engineering*, 362:112790, 2020.

[36] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[37] William E Schiesser. *The numerical method of lines: integration of partial differential equations*. Elsevier, 2012.

[38] Sungyong Seo*, Chuizheng Meng*, and Yan Liu. Physics-aware difference graph networks for sparsely-observed dynamics. In *International Conference on Learning Representations*, 2020.

[39] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[40] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences, 2019.

[41] Karen Stengel, Andrew Glaws, Dylan Hettinger, and Ryan N King. Adversarial super-resolution of climatological wind and solar data. *Proceedings of the National Academy of Sciences*, 117(29):16805–16815, 2020.

[42] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[43] James William Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

[44] Jessica Vamathevan, Dominic Clark, Paul Czodrowski, Ian Dunham, Edgardo Ferran, George Lee, Bin Li, Anant Madabhushi, Parantu Shah, Michaela Spitzer, et al. Applications of machine learning in drug discovery and development. *Nature reviews Drug discovery*, 18(6):463–477, 2019.

[45] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

[46] Minkai Xu, Lantao Yu, Yang Song, Chence Shi, Stefano Ermon, and Jian Tang. Geodiff: A geometric diffusion model for molecular conformation generation. In *International Conference on Learning Representations*, 2022.

[47] Wei Xu, Xihaier Luo, Yihui Ren, Ji Hwan Park, Shinjae Yoo, and Balasubramanya T. Nadiga. Feature importance in a deep learning climate emulator, 2021.

[48] Ying Zhang and Chen Ling. A strategy to apply machine learning to small datasets in materials science. *Npj Computational Materials*, 4(1):1–8, 2018.

[49] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

A Numerical Examples

Heat equation In the first example, a time-dependent diffusion equation on a bounded 2D Cartesian domain $\Omega = [0, L] \times [0, L]$ is considered,

$$\begin{aligned} \frac{\partial u}{\partial t} &= \alpha \nabla^2 u + f \quad \text{in} \quad \Omega \times (0, T] \\ u|_{t=0} &= u_0(x, y) \end{aligned} \tag{10}$$

with boundary conditions $\partial\Omega$,

$$\begin{aligned} u(0, y, t) &= u(L, y, t) \\ u(x, 0, t) &= u(x, L, t) \end{aligned} \tag{11}$$

where α is the diffusion coefficient which will be held at $\alpha = 0.0003$ and the domain size set to $L = 1$ and $t \in [0, 1]$. For this problem we will use homogeneous Dirichlet boundary conditions, i.e., $u = 0$ on $\partial\Omega$ and assume no external source, i.e., $f = 0$.

Burgers' equation The two-dimensional coupled viscous Burgers' equation is the second example we'll look at

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} &= \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \quad (x, y, t) \in D \times (0, T], \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} &= \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right), \quad (x, y, t) \in D \times (0, T], \end{aligned} \tag{12}$$

subject to the initial conditions

$$\begin{aligned} u(x, y, 0) &= u_0(x, y), \quad (x, y) \in D, \\ v(x, y, 0) &= u_0(x, y), \quad (x, y) \in D, \end{aligned} \tag{13}$$

and the boundary conditions

$$\begin{aligned} u(x, y, t) &= f(x, y, t), \quad (x, y, t) \in \partial D \times (0, T], \\ v(x, y, t) &= g(x, y, t), \quad (x, y, t) \in \partial D \times (0, T] \end{aligned} \tag{14}$$

B Calculation of spatial derivatives

spatial derivatives included in $S_{\Delta x}(\cdot)$ can be effectively estimated using either auto differentiation or numerical approximation. In this paper, we will take the second approach. In particular, the Sobel and Laplace filters are utilized to approximate the first- and second-order derivatives with second-order accuracy [5, 42]

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial x} &= \frac{1}{8\Delta x}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{u} \quad \text{and} \quad \frac{\partial \mathbf{u}}{\partial y} = \frac{1}{8\Delta y}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{u} \\ \frac{\partial^2 \mathbf{u}}{\partial x^2} &= \frac{1}{2\Delta x^2}\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \mathbf{u} \quad \text{and} \quad \frac{\partial^2 \mathbf{u}}{\partial y^2} = \frac{1}{2\Delta y^2}\begin{bmatrix} 1 & 0 & 1 \\ 0 & -4 & 0 \\ 1 & 0 & 1 \end{bmatrix} * \mathbf{u} \end{aligned} \tag{15}$$

It is worth noting that filter size determines the computational efficiency and accuracy. Large filters, in general, can approximate differential operators with higher approximation orders. On the other hand, large filters have more memory overhead and a higher computation cost. In practice, the trade-off must be balanced [4].

C Calculation of time integration

θ-Rule For the heat equation, we used the θ-rule for time discretization. The key feature of θ-rule time discretization scheme is that we can use one formula to generate a family of well-known. For example, the Forward Euler scheme, Backward Euler scheme, and Crank-Nicolson scheme are denoted by $\theta = 0$, $\theta = 1$, and $\theta = 1/2$, respectively. On the other hand, standard second-order accurate finite differences are used to compute the spatial derivatives. As a result, Eq. 10 can be written as:

$$\begin{aligned}\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} =&\theta \left(\alpha \left(\frac{u_{i-1,j}^{n+1} - 2_{i,j}^{n+1} + u_{i+1,j}^{n+1}}{\Delta x^2} + \frac{u_{i,j-1}^{n+1} - 2_{i,j}^{n+1} + u_{i,j+1}^{n+1}}{\Delta y^2} \right) + f_{i,j}^{n+1} \right) + \\ &(1-\theta) \left(\alpha \left(\frac{u_{i-1,j}^{n} - 2_{i,j}^{n} + u_{i+1,j}^{n}}{\Delta x^2} + \frac{u_{i,j-1}^{n} - 2_{i,j}^{n} + u_{i,j+1}^{n}}{\Delta y^2} \right) + f_{i,j}^{n} \right)\end{aligned} \tag{16}$$

Collecting the unknowns on the left-hand side, we have

$$\begin{gathered}u_{i,j}^{n+1} - \theta \left(F_x \left(u_{i-1,j}^{n+1} - 2_{i,j}^{n+1} + u_{i,j}^{n+1} \right) + F_y \left(u_{i,j-1}^{n+1} - 2_{i,j}^{n+1} + u_{i,j+1}^{n+1} \right) \right) = \\ (1-\theta) \left(F_x \left(u_{i-1,j}^{n} - 2_{i,j}^{n} + u_{i,j}^{n} \right) + F_y \left(u_{i,j-1}^{n} - 2_{i,j}^{n} + u_{i,j+1}^{n} \right) \right) + \theta \Delta t f_{i,j}^{n+1} + (1-\theta) \Delta t f_{i,j}^{n} + u_{i,j}^{n}\end{gathered} \tag{17}$$

where

$$F_x = \frac{\alpha \Delta t}{\Delta x^2}, \quad F_y = \frac{\alpha \Delta t}{\Delta y^2} \tag{18}$$

F_x and F_y are the Fourier numbers in x and y direction, respectively. To initialize the problem, we set $\Delta x = \Delta y = 1/64$ and $\Delta t = 0.01$.

Crank–Nicolson method To construct a numerical solution for the Burgers' equation, consider Eq. 12 on a bounded two-dimensional (2D) Cartesian domain $\Omega = [0, L_x] \times [0, L_y]$, where

$$\begin{aligned}&0 = x_0 < x_1, \ldots, x_{n_x-1} < x_{n_x} = L_x, \quad && x_{i+1} - x_i = \Delta x, \\ &0 = y_0 < y_1, \ldots, y_{n_y-1} < y_{n_y} = L_y, \quad && y_{j+1} - y_j = \Delta y, \\ &0 = t_0 < t_1, \ldots, t_{n_t-1} < t_{n_t} = T, \quad && t_{n+1} - t_n = \Delta t,\end{aligned} \tag{19}$$

Let nodal points $u_{i,j}^n$ and $v_{i,j}^n$ be the discrete approximation of u and v at the grid point $(i\Delta x, j\Delta y, n\Delta t)$, respectively. Because the Burgers' equations are time-dependent, a robust and precise numerical approach for temporal discretization is required. Compared to the explicit Euler method, the Crank-Nicolson finite-difference method that allows for the relaxation of strict constraints on the number of time steps and provides more accurate solutions is used. By applying Crank–Nicolson scheme to Eq. 12, we get

$$\begin{aligned}\frac{u_{i,j}^{n+1} - u_{i,j}^{n}}{\Delta t} &+ \frac{1}{2} \left[u_{i,j}^{n+1} \left(\frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} \right) + u_{i,j}^{n} \left(\frac{u_{i+1,j}^{n} - u_{i-1,j}^{n}}{2\Delta x} \right) \right] \\ &+ \frac{1}{2} \left[\mathrm{v}_{i,j}^{n+1} \left(\frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} \right) + v_{i,j}^{n} \left(\frac{u_{i,j+1}^{n} - u_{i,j-1}^{n}}{2\Delta y} \right) \right] \\ &- \frac{1}{\mathrm{Re}} \left[\frac{1}{2} \left\{ \left(\frac{u_{i+1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i-1,j}^{n+1}}{(\Delta x)^2} \right) + \left(\frac{u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n}}{(\Delta x)^2} \right) \right\} \right. \\ &\left. + \frac{1}{2} \left\{ \left(\frac{u_{i,j+1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j-1}^{n+1}}{(\Delta y)^2} + \left(\frac{u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n}}{(\Delta y)^2} \right) \right\} \right] = 0\right.\end{aligned} \tag{20}$$

and

$$
\begin{aligned}
\frac{v_{i,j}^{n+1}-v_{i,j}^{n}}{\Delta t} &+ \frac{1}{2}\left[u_{i,j}^{n+1}\left(\frac{v_{i+1,j}^{n+1}-v_{i-1,j}^{n+1}}{2\Delta x}\right)+u_{i,j}^{n}\left(\frac{v_{i+1,j}^{n}-v_{i-1,j}^{n}}{2\Delta x}\right)\right] \\
&+ \frac{1}{2}\left[\mathrm{v}_{i,j}^{n+1}\left(\frac{v_{i,j+1}^{n+1}-v_{i,j-1}^{n+1}}{2\Delta y}\right)+v_{i,j}^{n}\left(\frac{v_{i,j+1}^{n}-v_{i,j-1}^{n}}{2\Delta y}\right)\right] \\
&- \frac{1}{\mathrm{Re}}\left[\frac{1}{2}\left\{\left(\frac{v_{i+1,j}^{n+1}-2v_{i,j}^{n+1}+v_{i-1,j}^{n+1}}{(\Delta x)^2}\right)+\left(\frac{v_{i+1,j}^{n}-2v_{i,j}^{n}+v_{i-1,j}^{n}}{(\Delta x)^2}\right)\right\}\right. \\
&\left.+ \frac{1}{2}\left\{\left(\frac{v_{i,j+1}^{n+1}-2v_{i,j}^{n+1}+v_{i,j-1}^{n+1}}{(\Delta y)^2}+\left(\frac{v_{i,j+1}^{n}-2v_{i,j}^{n}+v_{i,j-1}^{n}}{(\Delta y)^2}\right)\right\}\right]=0
\end{aligned}
\tag{21}
$$

The truncation error of the adopted numerical scheme using the Taylor series expansion is of order $\mathcal{O}((\Delta x)^2+(\Delta y)^2+(\Delta t)^2)$.

D Qualitative emulation performance

We begin by applying the heat equation to the diffusion of a Gaussian hill. The initial value is set to

$$u_0(x,y)=e^{-a(x-\epsilon_1)^2-a(y-\epsilon_2)^2} \tag{22}$$

for $a=1$ on the domain $[0,1]\times[0,1]$. For this problem we will use homogeneous Dirichlet boundary conditions $u_D=0$. ϵ_1 and ϵ_2 are random variables generated from a normal distribution. A numerical solver generates the ground truth data. To solve time-dependent PDEs using the finite element method, first discretize the time derivative using a finite difference approximation, yielding a sequence of stationary problems, and then convert each stationary problem into a variational formulation. Fig. 6 shows the prediction results from the FNO and ResNet. ResNet with a hierarchical convolution layer provides better error distribution because the diffusion process can be very local, but FNO performs better in terms of magnitude.

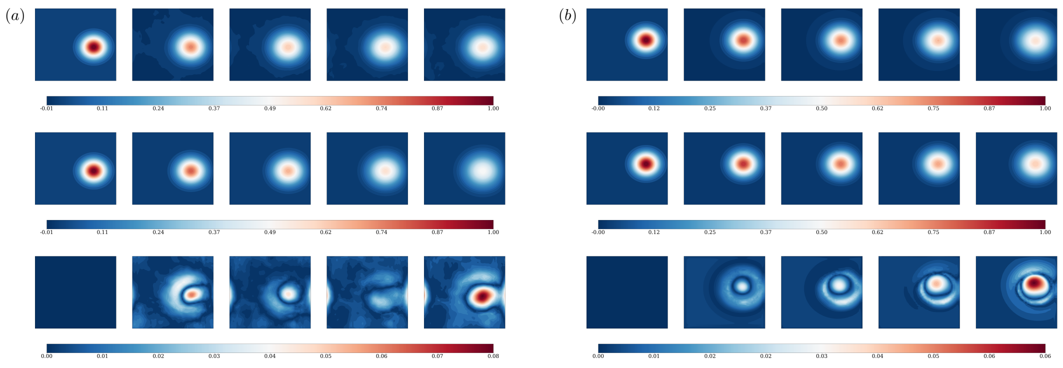


Figure 6: Heat equation: Gaussian hill (a) FNO and (b) ResNet.

To investigate the capabilities of FNO and ResNet further, we change the initial condition to a uniform distribution. Fig.7 shows the emulation results. The first row displays the predicted dynamics of the emulator, the second row displays the results of numerical simulation, and the third row displays the absolute error between numerical simulation and deep learning emulation. The results clearly show that FNO outperforms ResNet in simulating the diffusion process. For FNO, the error is smaller and grows gradually over time, whereas for ResNet, the error jumps to a higher range and stabilizes in most regions.

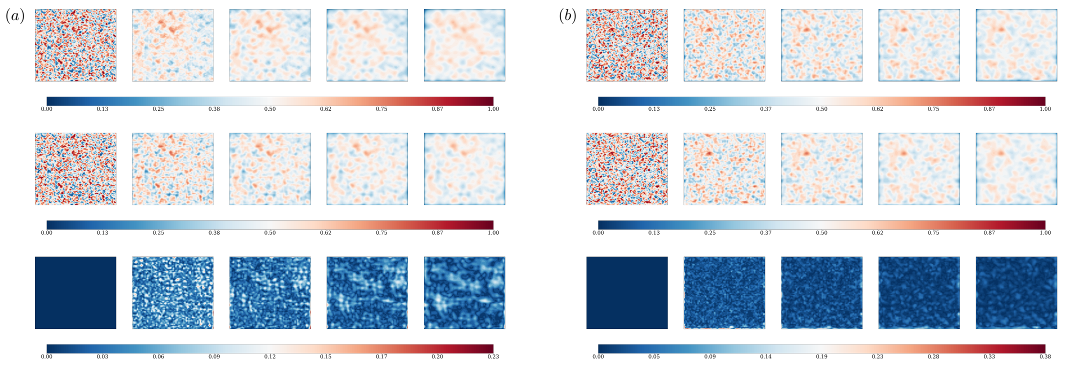


Figure 7: Heat equation: uniform distribution (a) FNO and (b) ResNet.

Fig.8 shows the FNO's prediction results of a randomly chosen realization from the Burgers' equation test dataset. The target velocity field was obtained by FEniCS [1], an open-source library for finite element simulations. The emulator was trained on 8192 realizations. To evaluate the results, the absolute error between the target and prediction is calculated and presented in the third column of two subfigures. Overall, the FNO model accurately predict the complex fluid structures. The predictions of FNO match the reference values well in all time instances.

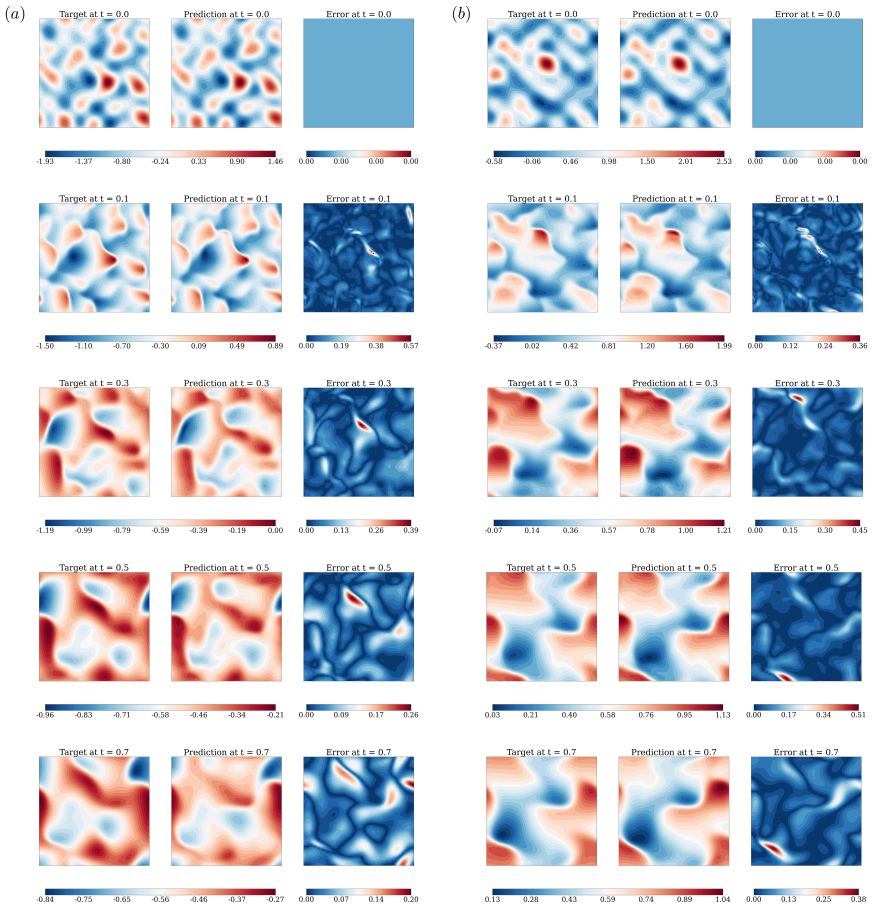


Figure 8: Comparisons of FNO emulation to the FEM simulation for a randomly chosen test example. Figure (a) and (b) summarize the x and y-velocity results, respectively.

Furthermore, the proposed FNO model outperforms the state-of-the-art ResNet model trained on the same amount of fluid realizations. In Fig.9, it can be observed that both models are able to provide satisfactory predictions in the predefined training time period, namely, $t \in [0, 0.5]$. However, the performance of ResNet degrades significantly in the extrapolation region, that is, $t \in (0.5, 1]$. The ResNet's predictions deviate from the ground truth while the prediction results of FNO can still maintain relatively high accuracy. Such comparison results reflect FNO's superior ability to absorb underlying physical principles, which supports a better generalization of unknown scenarios.

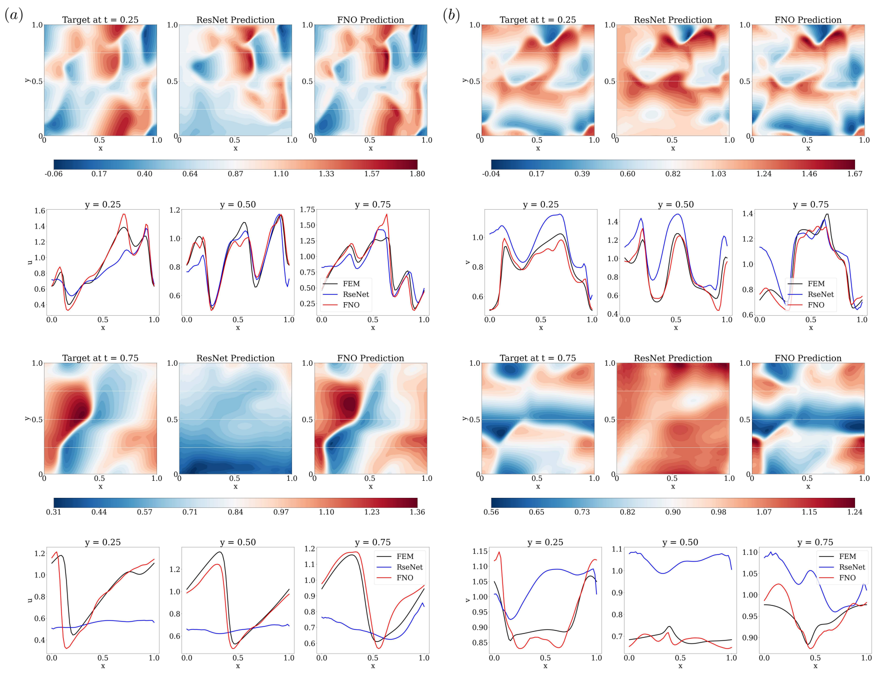


Figure 9: Prediction results of FNO and ResNet for (a) x and (b) y-velocity fields at time instances of 0.25 and 0.75.

E Quantitative emulation performance

Additionally, the quality of the ResNet/FNO predictions are quantitatively evaluated based on the following relative error metric:

$$Err\left(u_{\text{pred}}, u_{true}\right) = \frac{\left\|u_{\text{pred}} - u_{\text{true}}\right\|_2}{\left\|u_{\text{true}}\right\|_2} \tag{23}$$

and the coefficient of determination, also known as the R^2 score, which is defined as:

$$R^2 = 1 - \frac{\sum_{k=1}^{4096}\left(u_{pred}^k - u_{\text{true}}^k\right)^2}{\sum_{k=1}^{4096}\left(u_{\text{true}}^k - \bar{u}_{\text{true}}^k\right)^2} \tag{24}$$

where $\bar{u}_{\text{true}}$ is the mean of u_{true}^k. The 500 test realizations are statistically analyzed using these two metrics. Table 1 shows the mean and variance of the calculated relative L_2 error and R^2 score. Again, the FNO model produces a far less relative L_2 error than the ResNet model. Also, the R^2 score of the FNO model is closer to 1 than the ResNet model. All of these findings indicate that the predicted velocity fields from the FNO match the references better than those from the ResNet.

	Relative error		R^2 score	
	FNO	ResNet	FNO	ResNet
Mean	**0.15817**	0.30237	**0.92622**	0.74475
Variance	**0.00087**	0.00321	**0.00185**	0.00966

Table 1: Relative error and R2 score of test samples.

F Computational efficiency

Because the proposed FNO model is mesh-free, greater predictive efficiency is expected when applied to a more refined spatial discretization. To verify this, we further carried out the quanti- tatively analysis on resolutions of 128×128, 256×256, and 512×512. Note the ground truth solver (FEniCS) does not fully support GPUs and has varying options of linear solver and pre- conditioner, for example, conjugate gradient method vs. generalized minimal residual method. Interest here was to provide a hardware-specific performance comparison. The computation time of both FEM-based and FNO-based simulations is shown in Fig.10 (a). Compared to the classical FEM-based solver, the proposed FNO-based emulator scales very well. Evaluating our model on the same CPU hardware can still achieve speedups between 100x-4570x. Users with access to a GPU could further benefit from an approximate speedup of 60x. Meanwhile, we investigated FNO and ResNet scaling performance, which is defined as a function of the number of initial conditions generated for training. Fig.10 (b) depicts the results.

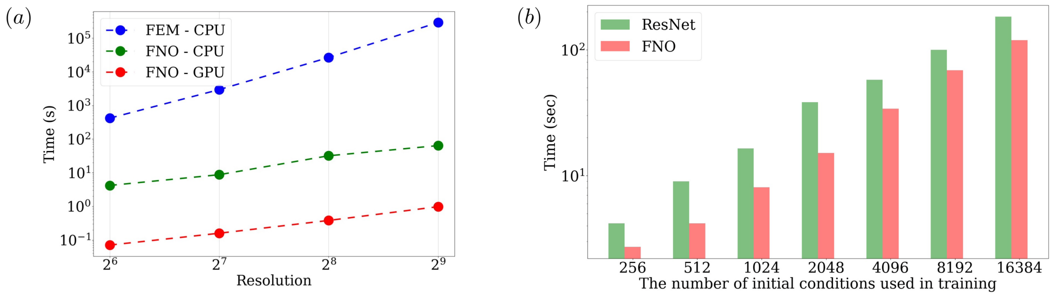


Figure 10: Computational cost: (a) computation time for different resolutions. The time refers to the wall clock time for inferring one sample. (b) cost as a function of samples.

G Out-of-distribution prediction

For the Burgers' equation, randomness is introduced to the initial conditions to better model the target system's stochastic nature, for instance, the presumed chaos in turbulence. Because Fourier series has been applied to approximate the physical turbulence in many studies, we formalize the random initial states using a truncated Fourier series with random coefficients $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c} \in \mathbb{R}^2$

$$\begin{aligned} \boldsymbol{w}(x,y) &= \sum_{i=-N_i}^{N_i} \sum_{j=-N_j}^{N_j} \boldsymbol{a}_{ij} \sin(2\pi(ix+jy)) + \boldsymbol{b}_{ij} \cos(2\pi(ix+jy)) \\ \boldsymbol{u}(x,y,0) &= \frac{2\boldsymbol{w}(x,y)}{\max_{\{x,y\}} \boldsymbol{w}(x,y)} + \boldsymbol{c} \end{aligned} \tag{25}$$

Originally, random coefficients are defined as:

$$\boldsymbol{a}_{ij}, \boldsymbol{b}_{ij} \sim \mathcal{N}(0, \boldsymbol{I}_2), \quad \boldsymbol{c} \sim \mathcal{U}(-1, 1) \in \mathbb{R}^2, \quad N_i = N_j = 4 \tag{26}$$

Here, we investigate the emulators' generalization performance to *out-of-distribution* initial conditions. We take the emulators trained on random inputs defined in Eq. 25 and test them on new input distributions:

$$\boldsymbol{a}_{ij}, \boldsymbol{b}_{ij} \sim \mathcal{U}(0, 1) \in \mathbb{R}^2, \quad \boldsymbol{c} \sim \mathcal{U}(-1, 1) \in \mathbb{R}^2, \quad N_i = N_j = 4 \tag{27}$$

Fig.11 depicts samples generated from the training distribution as well as the unknown testing distribution. It can be clearly seen that the newly generated initial conditions are very different from those used in model training.

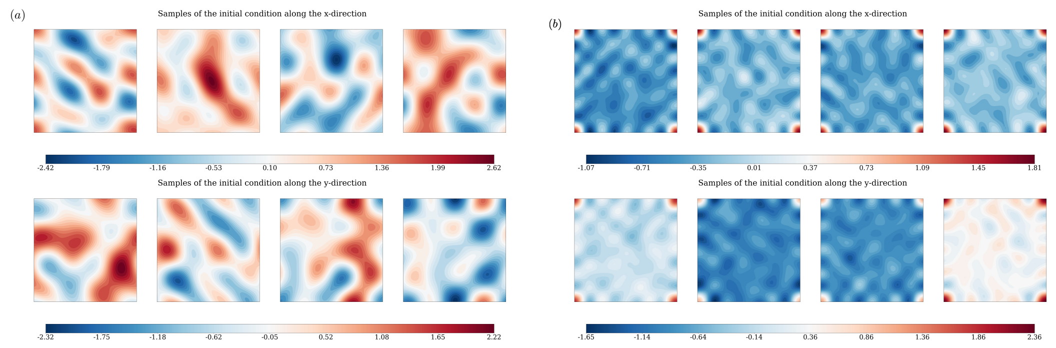


Figure 11: Initial conditions: (a) in-distribution samples; (b) *out-of-distribution* samples

Similar to Fig.4 (a), the error propagation process is presented in Fig.12. Overall, the FNO model has better generalization than the ResNet model. It generalizes well to new initial conditions, which is completely different from the training realizations. Though the accuracy of both models decreases at a very much similar rate in the extrapolation region, the FNO model has a significantly lower prediction error, particularly in the training region. This indicates that the FNO model has strong flexibility and robustness when emulating high-dimensional advection systems with strongly nonlinear characteristics.

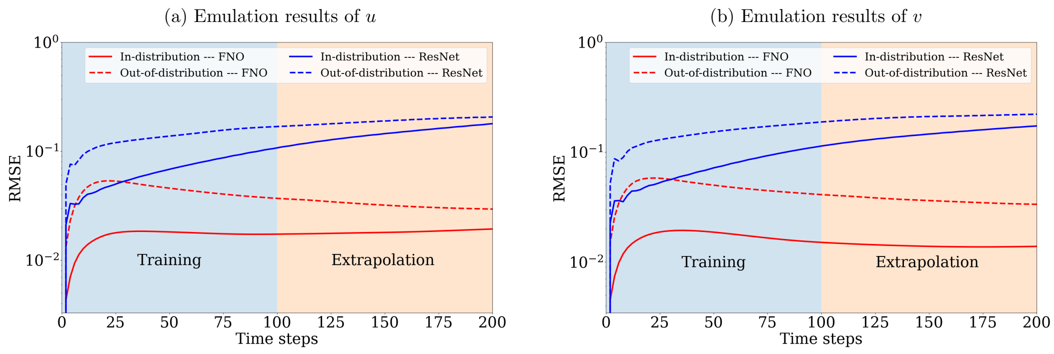


Figure 12: Quantitative evaluation results of the Burgers' equation.

Note each prediction experiment includes two regions, namely, the training $[0, 0.5]$ and the extrapolation $(0.5, 1]$. Fig.12 shows the prediction performance for a randomly chosen realization with time index specified to 0.25 for training and 0.75 for extrapolation. It is found that the predictions of FNO are almost unaffected by the *out-of-distribution* samples, while the ResNet model is seriously deteriorated and produces worse predictions, especially in the extrapolation region. Also, the error contour of the ResNet model shows the bulk of errors are clustered on the leading face of the waves, which is specifically where the extreme values are located. However, the errors of the FNO predictions are relatively small and evenly distributed, and there is no clear high-error area in the predictions. Such findings indicate that the FNO model possesses superior robustness to ResNet.

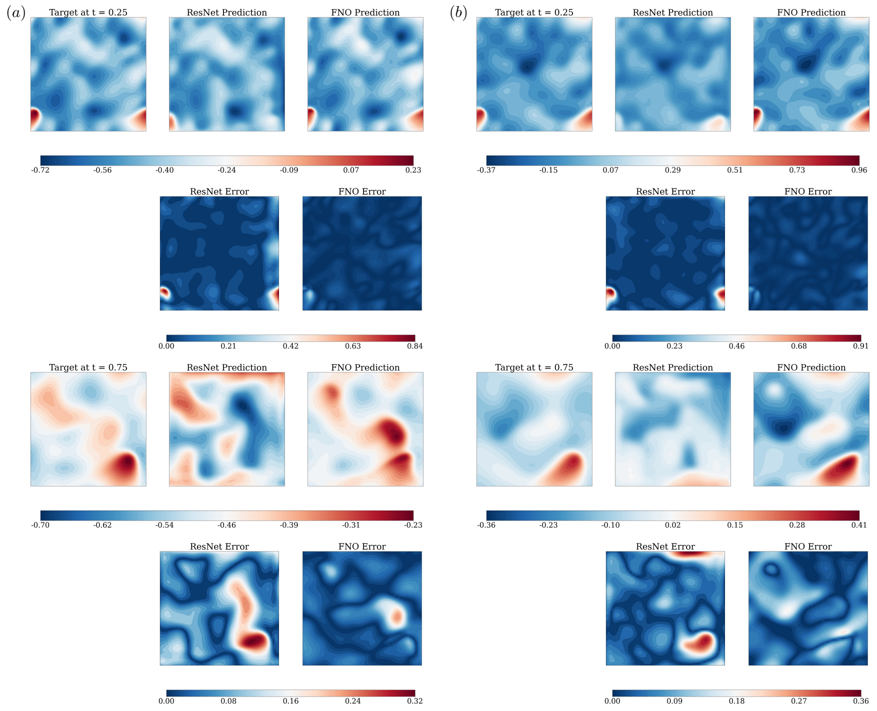


Figure 13: Predictions of the (a) x-velocity and (b) y-velocity of the selected out-of-distribution sample.

H Super resolution prediction

Here, we summarize the performance of the aforementioned two models on super-resolution prediction tasks. The purpose of this assessment is to demonstrate the unique ability of the proposed FNO model, that is, to generate continuous spatiotemporal solutions. Unlike many convolutional neural networks (CNNs) based approaches, which learn an end-to-end mapping between the low/high-resolution images, the FNO model is trained only with low-resolution data. Due to the inherent properties of convolution kernels, namely, spatial-agnostic and channel-specific, the ResNet fails to provide accurate prediction if additional tuning for different resolutions is not available. On the other hand, the proposed FNO model achieves consistent error when evaluated at a higher resolution. It is

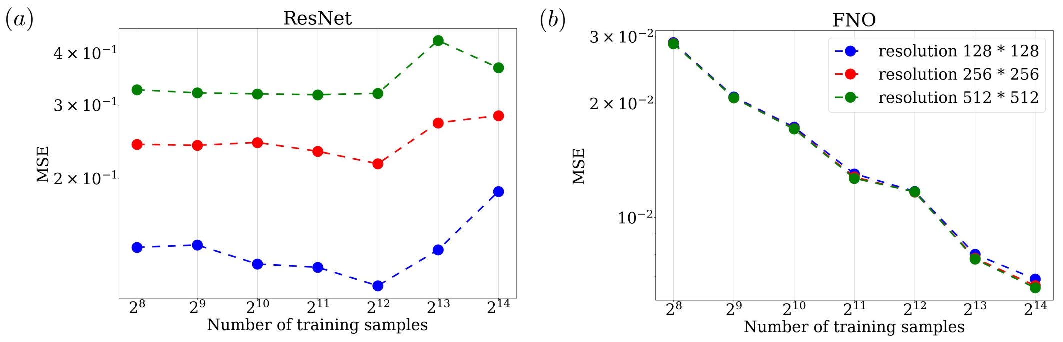


Figure 14: Scaling performance on super-resolution prediction tasks from 64×64 up to 512×512.

worth mentioning that the error is defined as the pixel-wise error, averaging over both the spatial and the temporal domain. The encouraging results (Fig.14) show great promise of the proposed model for fast emulating high-dimensional spatiotemporal dynamics.

I Notes on data saturation

Because we make direct use of the physics principles, i.e., governing equations in our case, both ResNet and FNO do not require a classical numerical solver such as a finite element simulator to generate data for training. Instead, the proposed knowledge synchronized learning framework begins with the initial conditions of the problem, which can be easily generated from a predefined distribution. Intuitively, one would expect the emulation quality of the model to improve when the number of the initial conditions increases. We can, in theory, generate infinite initial conditions and push the model to obey the constraints of the problem on all of them. However, we have to find a suitable number in practice considering constraints posed by the available time and computational resources. Viewing from the model capacity, another point worth mentioning is that the model may saturate rather than continuing to draw physics from the data. To verify this concern, the emulator was trained using sample sets with 256, 512, 1024, 2048, 4096, 8192, 16384, and 32768 realizations.

Fig.15 shows the computed results evaluated on a test set of 512 realizations. For both models, the error decreases as the amount of training data increases, indicating additional data is contributing to training. The improvement brought by the data is more evident at the beginning, where a small amount of information is used (e.g., 512 vs. 1024). Though the data size keeps increasing by a factor of 2, the performance enhancement slows down. In the case in which 32768 samples were utilized for training, the performance is almost the same and partially worse than the case where 16384 samples were used, implying the model may reach a saturation state. Also, it should be addressed that the FNO model can achieve the same accuracy using fewer data samples. After training of 120 epochs, the prediction performance measured by the mean MSE of different models shows apparent differences. The FNO model offers a more compelling ability to embedding physics to learn the underlying spatiotemporal dynamics.

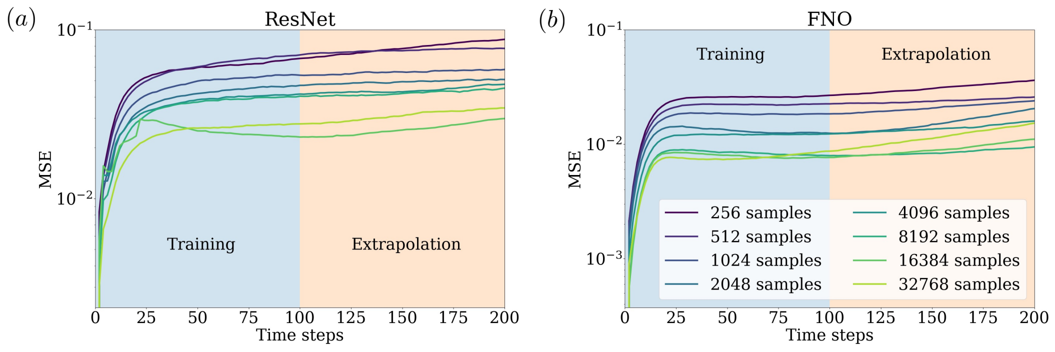


Figure 15: Predictions of the (a) x-velocity and (b) y-velocity of the selected out-of-distribution sample.