

# Deep Neural ODE Operator Networks for PDEs

**Ziqian Li**

joint work with K. Liu (UBE), Y. Song (NTU), H. Yue (Nankai), and E. Zuazua (FAU)

Chair for Dynamics, Control, Machine Learning, and Numerics  
Alexander von Humboldt-Professorship  
Friedrich-Alexander-Universität Erlangen-Nürnberg

EZ 65: Control, PDEs and Machine Learning  
June 11, 2026  
Bilbao, Spain

## EZ 65: Happy Birthday!



Figure: September 16th, 2023, Changchun, China.

# Outline

- 1 Introduction on operator learning
- 2 Deep Neural ODE Operator Networks
- 3 Numerical Results
- 4 Perspectives

# Outline

- 1 Introduction on operator learning
- 2 Deep Neural ODE Operator Networks
- 3 Numerical Results
- 4 Perspectives

# PDEs and Traditional Numerical solvers

- We consider a generic class of PDEs modeled by

$$\begin{cases} \partial_t u(t, x) + \mathcal{L}[a](u)(t, x) = f(t, x) & \forall (t, x) \in [0, T] \times \Omega, \\ u(0, x) = u_0(x) & \forall x \in \Omega, \\ \mathcal{B}u(t, x) = u_b(t, x) & \forall (t, x) \in [0, T] \times \partial\Omega. \end{cases}$$

A numerical solver of the PDE tries to find the numerical approximation of

$$(t, x) \rightarrow u \approx u_\theta.$$

- **Traditional Numerical Methods:** FEM, FDM, FVM, Spectral methods,...
- **Challenges of Traditional Methods:**
  - ▶ PDEs in **high-dimensional spaces** and **complex domains**.
  - ▶ Problems requiring **repeated but expensive simulations**, e.g., inverse problems and optimal control of PDEs.

# Scientific Machine Learning for PDEs - function learning

## Solution learning methods: using NNs to approximate the solution function of a PDE

- Deep Ritz method [E and Yu, 2018]
- Deep Galerkin method [Sirignano and Spiliopoulos, 2018]
- Physics-informed neural networks (PINNs) [Raissi, Perdikaris, and Karniadakis, 2018]
- Weak adversarial networks [Zang, Bao, Ye, and Zhou, 2020]
- Random feature methods [Chen, Chi, E, and Yang, 2022]
- ... ..

## Typical applications:

- High-dimensional PDEs.
- Complex geometries.
- ... ..

# Operator Learning

We consider a class of non-stationary PDEs modeled by

$$\begin{cases} \partial_t u(t, x) + \mathcal{L}[a](u)(t, x) = f(t, x) & \forall (t, x) \in [0, T] \times \Omega, \\ u(0, x) = u_0(x) & \forall x \in \Omega, \\ \mathcal{B}u(t, x) = u_b(t, x) & \forall (t, x) \in [0, T] \times \partial\Omega. \end{cases}$$

- **Parameters:**  $v \subset \{f, a, u_0, u_b\}$ .
- **Goal of operator learning:**

$$\Psi_\theta \approx \Psi^\dagger : v \mapsto u,$$

where  $\Psi^\dagger$  is the solution operator (infinite dimension), which maps  $v$  to the solution  $u$ , by a neural-network-based functional  $\Psi_\theta$  with trainable parameters  $\theta$ .

# Scientific Machine Learning for PDEs - operator learning

## Operator learning methods: using NNs to approximate the solution operator of a PDE

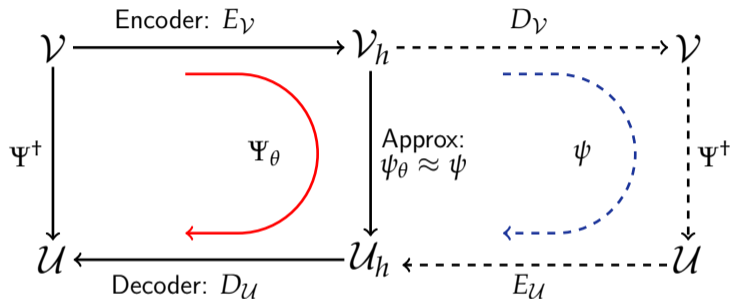
- Deep Operator Networks (DeepONets) [Lu, Jin, Pang, Zhang, and Karniadakis, 2019],
- Physics-Informed DeepONets [Wang, Wang, and Perdikaris, 2021]
- Fourier Neural Operator, Graph Neural Operator, [Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, Stuart, and Anandkumar, 2021]
- The Random Feature Approach [Nelsen and Stuart, 2021]
- In-Context Operator Networks [Yang, Liu, Meng, and Osher, 2024]
- ... ..

## Typical applications:

- PDEs discovery: model and predict unknown physics through data.
- Acceleration: speed-up computationally expensive simulations.
- ... ..

## Encoder-decoder-net architectures

- Learning an infinite-dimensional operator  $\Psi^\dagger : \mathcal{V} \rightarrow \mathcal{U}$ .



- An encoder-decoder pair  $(E_{\mathcal{V}}, D_{\mathcal{V}})$  on  $\mathcal{V}$ , i.e.,  $E_{\mathcal{V}} : \mathcal{V} \rightarrow \mathbb{R}^{d_{\mathcal{V}}}$ ,  $D_{\mathcal{V}} : \mathbb{R}^{d_{\mathcal{V}}} \rightarrow \mathcal{V}$ ,
- An encoder-decoder pair  $(E_{\mathcal{U}}, D_{\mathcal{U}})$  on  $\mathcal{U}$ , i.e.,  $E_{\mathcal{U}} : \mathcal{U} \rightarrow \mathbb{R}^{d_{\mathcal{U}}}$ ,  $D_{\mathcal{U}} : \mathbb{R}^{d_{\mathcal{U}}} \rightarrow \mathcal{U}$ .
- Using a neural network  $\psi_\theta : \mathcal{V}_h \rightarrow \mathcal{U}_h$  to approximate  $\psi$ , we obtain an encoder-decoder network  $\Psi_\theta : \mathcal{V} \rightarrow \mathcal{U}$ :

$$\Psi_\theta := D_{\mathcal{U}} \circ \psi_\theta \circ E_{\mathcal{V}} \approx \Psi^\dagger.$$

# Outline

1 Introduction on operator learning

**2 Deep Neural ODE Operator Networks**

3 Numerical Results

4 Perspectives

## Motivation: Approximation theory of SA-NODEs

The semi-autonomous neural ODEs (SA-NODEs) [Li, Liu, Liverani, Zuazua, 2026] read

$$\begin{cases} \dot{\mathbf{x}} = \sum_{i=1}^P W_i \circ \sigma(A_i^1 \mathbf{x} + A_i^2 t + B_i), & t \in [0, T], \mathbf{x} \in \mathbb{R}^d \\ \mathbf{x}(0) = \mathbf{x}_0 \in \mathbb{R}^d. \end{cases}$$

Here the parameters  $W_i$ ,  $A_i^1$ ,  $A_i^2$  and  $B_i$  are **constant matrixes**.

For an ODE system

$$\begin{cases} \dot{\mathbf{z}}(t) = f(t, \mathbf{z}(t)), & t \in [0, T], \\ \mathbf{z}(0) = \mathbf{z}_0, \end{cases}$$

the following approximation property holds:

$$\|\mathbf{z}_{z_0}(\cdot) - \mathbf{x}_{z_0}(\cdot)\|_{\mathbf{L}^\infty([0, T]; \mathbb{R}^d)} \leq \frac{C_{T, K, f}}{\sqrt{P}},$$

where  $C_{T, K, f}$  is a constant independent of  $P$ .

## A concrete example: Heat equation

We consider the 1D heat equation

$$\begin{cases} \partial_t u(t, x) - \Delta u(t, x) = 0, & \forall (t, x) \in [0, 1] \times [0, 1], \\ u(t, 0) = u(t, 1) = 0, & \forall t \in [0, 1], \\ u(0, x) = u_0(x), & \forall x \in [0, 1]. \end{cases}$$

The objective is to use a neural-network-based functional  $\Psi_\theta$  to learn the mapping:

$$\Psi_\theta \approx \Psi^\dagger : u_0 \mapsto u.$$

Learn from numerical solutions with different initial conditions. (need discretization)

## Step 1: Encoding (Discretization and preparing training dataset)

We discretize space and time into uniform mesh:

- Space:  $[0, 1] \mapsto N_x$  points:  $(x_1, x_2, \dots, x_{N_x})$
- Time:  $[0, 1] \mapsto N_T$  points:  $(t_1, t_2, \dots, t_{N_T})$

Let  $N_{u_0}$  denote the number of samples of  $u_0$ , then the **training dataset** is:

$$\{U_{0,h}^k, U_h^k\}, \quad k = 1, 2, \dots, N_{u_0}$$

where  $U_{0,h} = (u_0(x_i))_{i=1}^{N_x} \in \mathbb{R}^{N_x}$ ,  $U_h^k = (u(t_j, x_i))_{i=1, j=1}^{N_x, N_T} \in \mathbb{R}^{N_x \times N_T}$ . All the training data is computed by the Finite Difference Method (FDM).

## Step 2: NODE surrogate (Approximation and training)

Suppose that  $U_h^k$  can be seen as values at different times of the solutions of an ODE with initial data  $U_{0,h}^k$ . We use  $U_\theta$  to approximate  $U_h$  and we present this ODE by the approximation of the following NODE:

$$\begin{cases} \frac{d}{dt}U_\theta = \sum_{p=1}^P w_p \circ \sigma(A_p^1 U_\theta + A_p^2 t + B_p), \\ U_\theta(0, x) = U_{0,h} \in \mathbb{R}^n. \end{cases}$$

Then the loss function to train the NODE is:

$$L(\theta) = \frac{1}{N_v N_x N_T} \sum_{k=1}^{N_v} \sum_{i=1}^{N_x} \sum_{j=1}^{N_T} \left| (U_\theta^k(t_j))_i - U_h^k(x_i, t_j) \right|^2 + \mathcal{R}(\theta).$$

Here,  $\mathcal{R}(\theta)$  is a general regularization term.

### Step 3: Decoding (Reconstruct PDE solution)

After the latent dynamics are learned, the PDE solution  $u$  is reconstructed from the NODE outputs  $u_\theta$

$$u(t, x) \approx u_\theta(t, x) = \sum_{i=1}^{N_x} (U_\theta(t))_i \alpha_i(x), \quad \forall (t, x) \in [0, 1] \times [0, 1],$$

where  $\alpha_i$  is the  $P_1$ -FEM basis centered at  $x_i$ .

## General framework for $v \mapsto u$

For a class of non-stationary PDEs modeled by

$$\begin{cases} \partial_t u(t, x) + \mathcal{L}[a](u)(t, x) = f(t, x) & \forall (t, x) \in [0, T] \times \Omega, \\ u(0, x) = u_0(x) & \forall x \in \Omega, \\ \mathcal{B}u(t, x) = u_b(t, x) & \forall (t, x) \in [0, T] \times \partial\Omega. \end{cases}$$

The parameters of the PDE is defined by  $v \subset \{f, a, u_0, u_b\}$ . The architecture of the **NODE-ONet**:

$$\text{Architecture} \left\{ \begin{array}{l} \text{Encoding: } E_{\mathcal{V}}(v) := \{v_{\ell}(t)\}_{\ell=1}^{d_{\mathcal{V}}} \in \mathbb{R}^{d_{\mathcal{V}}} \text{ for any } t \in [0, T]; \\ \text{Physics-encoded NODE surrogate: } \begin{cases} \dot{\psi}(t) = \mathcal{N}_{\theta_{\psi}}(\psi(t), \mathcal{P}_v v(t), t), \\ \psi(0) = \mathcal{P}_u u_0 \in \mathbb{R}^{d_{\mathcal{U}}}, \end{cases} \\ \text{Decoding: } \Psi_{\text{NODE-ONet}}(v; \theta)(t, x) = D_{\mathcal{U}}(t, \alpha) = \sum_{j=1}^{d_{\mathcal{U}}} \alpha_j(x) \psi_j(t), \end{array} \right.$$

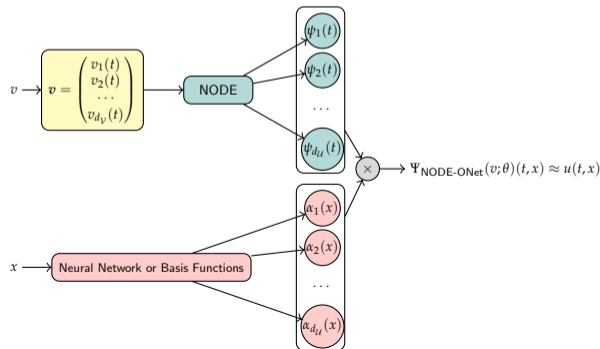
with  $\{\alpha_j\}_{j=1}^{d_{\mathcal{U}}}$  a set of spatial basis functions (FEM, spectral, NN, ...).

## General framework for $v \mapsto u$

The training setting is summarized as follows:

$$\text{Training: } \begin{cases} \text{Dataset: } \{v_i, x_j, \Psi^\dagger(v_i)(t_k, x_j)\}_{1 \leq i \leq N_v, 1 \leq k \leq N_t, 1 \leq j \leq N_x} \\ \text{Loss function: } \mathcal{L}(\theta) = \frac{1}{N_v N_x N_t} \sum_{i=1}^{N_v} \sum_{j=1}^{N_x} \sum_{k=1}^{N_t} \|\Psi_{\text{NODE-ONet}}(v_i)(t_k, x_j) - \Psi^\dagger(v_i)(t_k, x_j)\|_2^2. \end{cases}$$

The generic architecture of NODE-ONet:



# Outline

1 Introduction on operator learning

2 Deep Neural ODE Operator Networks

**3 Numerical Results**

4 Perspectives

# 1D diffusion-reaction - Setup

We consider the following diffusion-reaction equation

$$\begin{cases} \partial_t u(t, x) - \nabla \cdot (D(t, x) \nabla u(t, x)) + R(t, x) u^2(t, x) = f(t, x) & \forall (t, x) \in [0, T] \times \Omega, \\ u(0, x) = u_0(x) & \forall x \in \Omega, \\ u(t, x) = u_b(t, x) & \forall (t, x) \in [0, T] \times \partial\Omega, \end{cases}$$

In the following experiments, we validate the efficiency of NODE-ONets to approximate the following operators:

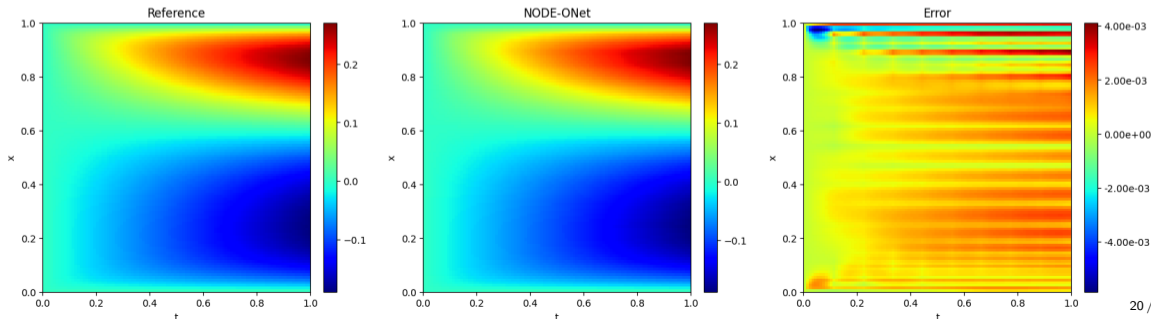
- Source-to-solution operator:  $f \mapsto u$ . (Comparison with DeepONet)
- Solution operator with multi-inputs:  $\{D, f \mapsto u\}$ . (Comparison with MIONet)
- Prediction beyond the training time. (Comparison with DeepONet and MIONet)

# 1D diffusion-reaction - Source-to-solution operator

Physics-encoded NODE:

$$\begin{cases} \psi(t) = \sum_{i=1}^P W_i \odot \sigma(A_i \odot \psi + \mathbf{a}_i^1 t + B_i) + \mathcal{P}_f f, \\ \psi(0) = \mathbf{0} \in \mathbb{R}^{d_U}. \end{cases}$$

**Figure:** Test results of the deep NODE operator network for the learned source-to-solution operator  $\Psi_f^* : f(x) \rightarrow u(x, t)$  with one random  $f(x)$ .



# 1D diffusion-reaction - Source-to-solution operator - Comparison

**Table:** Comparisons of the NODE-ONet with DeepONet for learning the source-to-solution operator  $\Psi_f^\dagger : f(x) \mapsto u(t, x)$ . Testing resolution:  $N_x = 100$ ,  $N_t = 100$ .

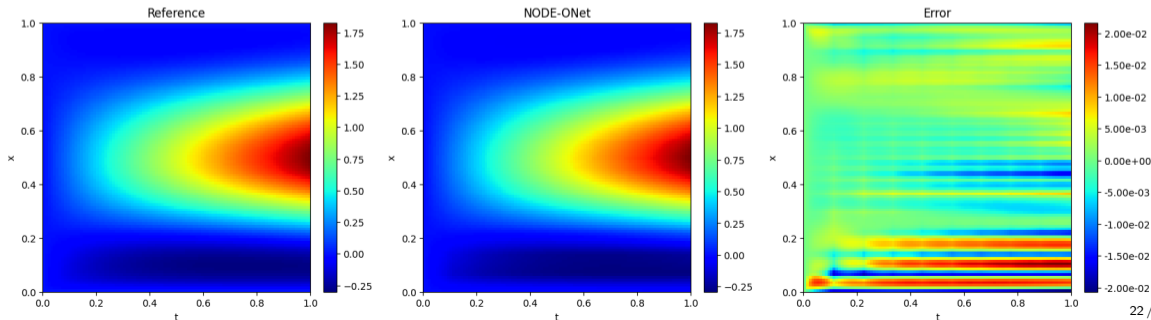
	Training resolutions	#Trainable parameters	#Training input $f$	Absolute error	Relative error
NODE-ONet	$N_x = 10$ $N_t = 5$ $d_{\mathcal{Y}} = 20$	27,550	100	$4.248 \times 10^{-3}$	$7.370 \times 10^{-3}$
NODE-ONet	$N_x = 100$ $N_t = 10$ $d_{\mathcal{Y}} = 20$	27,550	500	$1.368 \times 10^{-3}$	$2.675 \times 10^{-3}$
DeepONet	$N_x = 100$ $N_t = 10$ $K = 50$ $d_{\mathcal{Y}} = 100$	40,600	100	$6.352 \times 10^{-3}$	$1.230 \times 10^{-2}$
DeepONet	$N_x = 100$ $N_t = 100$ $K = 1,000$ $d_{\mathcal{Y}} = 100$	40,600	500	$1.313 \times 10^{-3}$	$2.582 \times 10^{-3}$

# 1D diffusion-reaction - Multi-inputs operator

Physics-encoded NODE:

$$\begin{cases} \dot{\psi}(t) = \sum_{i=1}^P W_i \odot \sigma(A_i \odot [\mathcal{P}_D \mathbf{D}] \odot \psi + \mathbf{a}_i^1 t + B_i) + \mathcal{P}_f \mathbf{f}, \\ \psi(0) = \mathbf{0} \in \mathbb{R}^{d_u}. \end{cases}$$

Figure: Test results of the deep NODE operator network for the learned solution operator  $\Psi_m^*$  with one random multi-input function:  $\{D(x), f(x)\} \rightarrow u(x, t)$ .



# 1D diffusion-reaction - Multi-inputs operator - Comparison

**Table:** Comparisons of the NODE-ONet with the MIONet for learning the solution operator with multi-input functions:  $\Psi_m^+ : \{D(x), f(x)\} \mapsto u(t, x)$ . Testing resolution:  $N_x = 100, N_t = 100$ .

	Training resolutions	#Trainable parameters	#Training $\{D, f\}$	Absolute error	Relative error
NODE-ONet	$N_x = 50$ $N_t = 10$	28,550	100	$2.362 \times 10^{-2}$	$5.297 \times 10^{-2}$
NODE-ONet	$N_x = 100$ $N_t = 10$	28,550	1,000	$4.626 \times 10^{-3}$	$1.032 \times 10^{-2}$
MIONet	$N_x = 100$ $N_t = 100$	161,600	100	$1.212 \times 10^{-1}$	$2.661 \times 10^{-1}$
MIONet	$N_x = 100$ $N_t = 100$	161,600	1,000	$9.491 \times 10^{-3}$	$2.072 \times 10^{-2}$

# 1D diffusion-reaction - Prediction

**Table:** Prediction results of for  $t \in [0, 2]$ .  $\Psi_f^* : f(x) \mapsto u(x, t)$ : the learned source-to-solution operator  
,  $\Psi_m^* : \{D(x), f(x)\} \mapsto u(x, t)$ : the learned solution operator with multi-input functions. Testing resolution:  
 $N_x = 100$ ,  $N_t = 100$ .

		#Training input functions	Training time frame	Test time frame	Absolute error	Relative error
$\Psi_f^*$	NODE-ONet	500	$t \in [0, 1]$	$t \in [0, 2]$	$6.839 \times 10^{-3}$	$7.113 \times 10^{-3}$
	DeepONet				$2.302 \times 10^{-1}$	$2.360 \times 10^{-1}$
$\Psi_m^*$	NODE-ONet	1,000	$t \in [0, 1]$	$t \in [0, 2]$	$1.392 \times 10^{-2}$	$1.732 \times 10^{-2}$
	MIONet				$1.012 \times 10^{-1}$	$1.251 \times 10^{-1}$

# 1D diffusion-reaction - Prediction - Compare with DeepONet

Figure: Prediction of  $\Psi_f^* : f \rightarrow u$  beyond  $t \in [0, 1]$  by NODE-ONet.

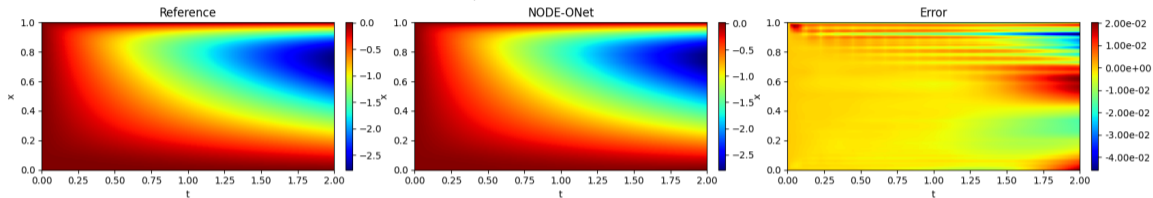
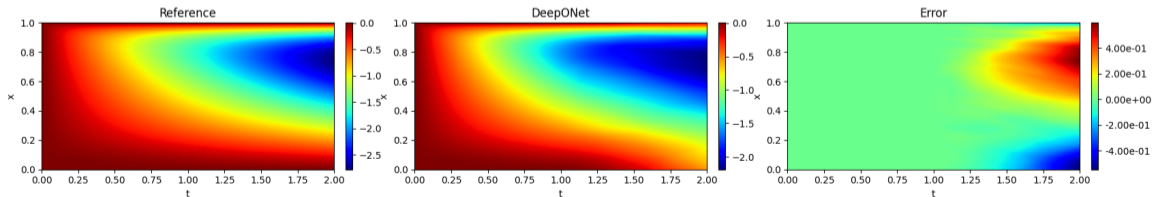


Figure: Prediction of  $\Psi_f^* : f \rightarrow u$  beyond  $t \in [0, 1]$  by DeepONet.



# 1D diffusion-reaction - Prediction - Compare with MIONet

Figure: Prediction of  $\Psi_m^* : \{D, f\} \rightarrow u$  beyond  $t \in [0, 1]$  by NODE-ONet.

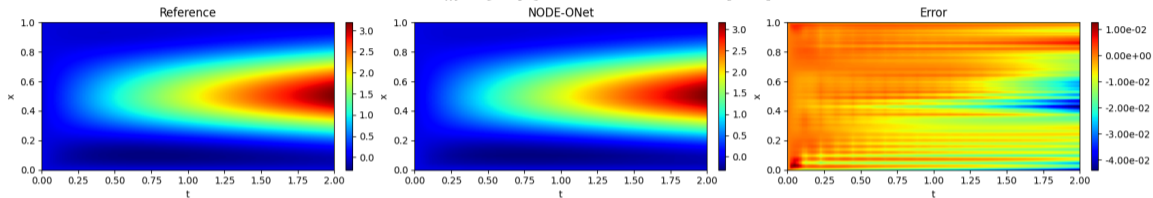
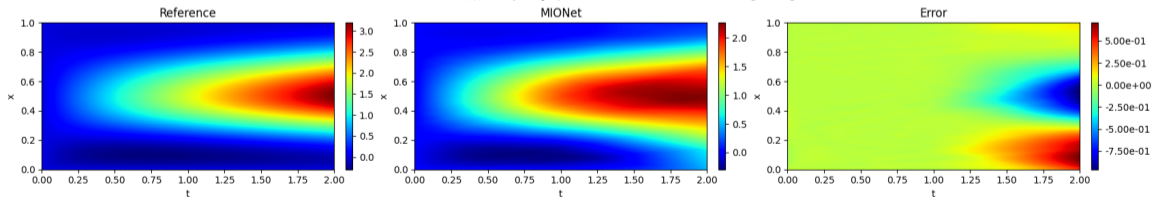


Figure: Prediction of  $\Psi_m^* : \{D, f\} \rightarrow u$  beyond  $t \in [0, 1]$  by MIONet.



## 2D Navier-Stokes - Setup

We consider the following Navier-Stokes equation

$$\begin{cases} \partial_t u(t, x) + \mathbf{V}(t, x) \cdot \nabla u(t, x) = \nu \Delta u(t, x) + f(t, x), & \forall (t, x) \in [0, T] \times \Omega, \\ u(t, x) = \nabla \times \mathbf{V}(t, x) := \partial_{x_1} V_2 - \partial_{x_2} V_1, & \forall (t, x) \in [0, T] \times \Omega, \\ \nabla \cdot \mathbf{V}(t, x) = 0, & \forall (t, x) \in [0, T] \times \Omega, \\ u(0, x) = u_0(x), & \forall x \in \Omega, \end{cases}$$

with proper boundary conditions. In the following experiments, we validate the efficiency of NODE-ONets to approximate the following operators:

- Initial-to-solution operator:  $u_0 \mapsto u$ .
- Source-to-solution operator:  $f \mapsto u$ .
- Multi-inputs operator:  $\{u_0, f \mapsto u\}$ .

All the NODE-ONets are trained on  $t \in [0, 10]$  and tested on  $t \in [0, 20]$ .

## 2D Navier-Stokes - Initial-to-solution operator

## 2D Navier-Stokes - Source-to-solution operator

## 2D Navier-Stokes - Multi-inputs operator

# Outline

1 Introduction on operator learning

2 Deep Neural ODE Operator Networks

3 Numerical Results

4 Perspectives

# Perspectives

- **Further Error Analysis.** The error analysis in this work is mainly devoted to the generic encoder-decoder architecture. It is relevant to analyze the (approximate and generalization) errors for the NODE-ONet framework, which depends intricately on the specific PDE under consideration and is technically involved.
- **Optimal NODEs.** Note that the physics-encoded NODEs used in our experiments are not unique. Hence, it is of great theoretical and practical significance to establish a mathematical principle to determine the optimal physics-encoded NODE for a specific PDE solution operator.
- **Extensions.**
  - ▶ Extend the NODE-ONet framework to address optimal control and inverse problems involving PDEs;
  - ▶ Our current algorithm focus is on parabolic equations, developing NODE-ONets for hyperbolic equations remains crucial.

---

Z. Li, K. Liu, Y. Song, H. Yue, E. Zuazua. Deep neural ODE operator networks for PDEs. *Math. Models Methods Appl. Sci.*, 36(8), 2026.

Thanks!  
Happy Birthday, Enrique!