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

The Mathematics of Scientific Machine Learning and Digital Twins
November 20–24, 2025
Erice, Italy

Friedrich-Alexander-Universität
DYNAMICS, CONTROL, MACHINE LEARNING AND NUMERICS

Outline

Outline

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

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

- **Traditional Numerical Methods:**
  - Finite Element Method (FEM),
  - Finite Difference Method (FDM),
  - Finite Volumn Method (FVM).
- **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, 2017]
- Deep Galerkin method [Sirignano and Spiliopoulos, 2017]
- Physics-informed neural networks (PINNs) [Raissi, Perdikaris, and Karniadakis, 2017]
- Weak adversarial networks [Zang, Bao, Ye, and Zhou, 2019]
- 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 , which maps v to the solution u, by a neural-network–based functional Ψ_θ with trainable parameters θ.

Scientific Machine Learning for PDEs - operator learning

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

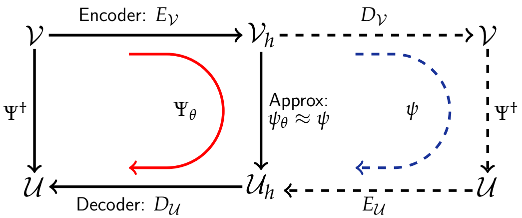
- Deep Operator Networks (DeepONets) [Lu, Jin, Pang, Zhang, and Karniadakis, 2019],
- Physics-Informed DeepONets [Wang, Wang, and Perdikaris, 2021]
- Fourier Neural Operator, Graph Neural Operator, [Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, Stuart, and Anandkumar,2021]
- The Random Feature Approach [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 $\Psi^\dagger : \mathcal{V} \to \mathcal{U}$.



- An encoder-decoder pair $(E_\mathcal{V}, D_\mathcal{V})$ on $\mathcal{V}$, i.e., $E_\mathcal{V} : \mathcal{V} \to \mathbb{R}^{d_\mathcal{V}}$, $D_\mathcal{V} : \mathbb{R}^{d_\mathcal{V}} \to \mathcal{V}$,
- An encoder-decoder pair $(E_\mathcal{U}, D_\mathcal{U})$ on $\mathcal{U}$, i.e., $E_\mathcal{U} : \mathcal{U} \to \mathbb{R}^{d_\mathcal{U}}$, $D_\mathcal{U} : \mathbb{R}^{d_\mathcal{U}} \to \mathcal{U}$.
- These encoder/decoder pairs imply a finite-dimensional function

$$\psi : \mathcal{V}_h \to \mathcal{U}_h, \quad \psi(\zeta) = E_\mathcal{U} \circ \Psi^\dagger \circ D_\mathcal{V}(\zeta), \quad \forall \zeta \in \mathcal{V}_h$$

and then $D_\mathcal{U} \circ \psi \circ E_\mathcal{V} \approx \Psi^\dagger$.

- Using a neural network $\psi_\theta : \mathcal{V}_h \to \mathcal{U}_h$ to approximate ψ, we obtain an encoder-decoder network $\Psi_\theta : \mathcal{V} \to \mathcal{U}$:

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

Outline

A concrete example: Heat equation

We consider the 1D heat equation

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

The obective is to use a neural-network–based functional Ψ_θ 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: $(x_1, x_2, \ldots, x_{N_x})$
- Time: $[0,1] \mapsto N_T$ points: $(t_1, t_2, \ldots, t_{N_T})$

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

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

where $U_{0,h} = \big(u_0(x_i)\big)_{i=1}^{N_x} \in \mathbb{R}^{N_x}$, $U_h^k = \big(u(t_j, x_i)\big)_{i=1,j=1}^{N_x,N_T} \in \mathbb{R}^{N_x \times N_T}$. 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_θ 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:

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

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

Step 3: Decoding (Reconstructe PDE solution)

After the latent dynamics are learned, the PDE solution u is reconstructed from the NODE outputs u_θ

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

where α_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:

$$\textbf{Architecture} \begin{cases} \text{Encoding:} \quad E_{\mathcal{V}}(v) := \{v_\ell(t)\}_{\ell=1}^{d_{\mathcal{V}}} \in \mathbb{R}^{d_{\mathcal{V}}} \text{ for any } t \in [0,T]; \\ \text{Physics-encoded NODE surrogate:} \quad \begin{cases} \dot{\boldsymbol{\psi}}(t) = \mathcal{N}_{\theta_\psi}(\boldsymbol{\psi}(t), \mathcal{P}_v \boldsymbol{v}(t), t), \\ \boldsymbol{\psi}(0) = \mathcal{P}_u \boldsymbol{u}_0 \in \mathbb{R}^{d_{\mathcal{U}}}, \end{cases} \\ \text{Decoding:} \quad \Psi_{\text{NODE-ONet}}(v;\theta)(t,x) = D_{\mathcal{U}}(t,\alpha) = \sum_{j=1}^{d_{\mathcal{U}}} \alpha_j(x)\psi_j(t), \end{cases}$$

with $\{\alpha_j\}_{j=1}^{d_{\mathcal{U}}}$ a set of spatial basis functions.

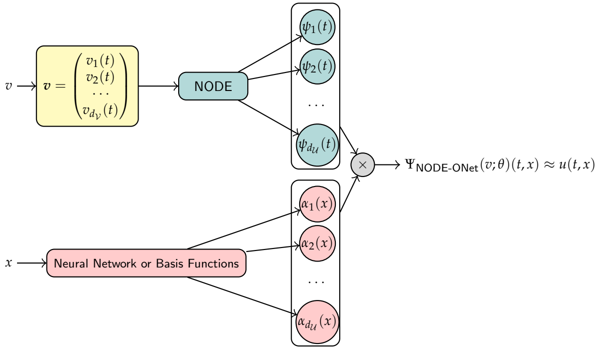
General framework for $v \mapsto u$

The training setting is summarized as follows:

Training:

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

The generic architure of NODE-ONet:



Outline

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:

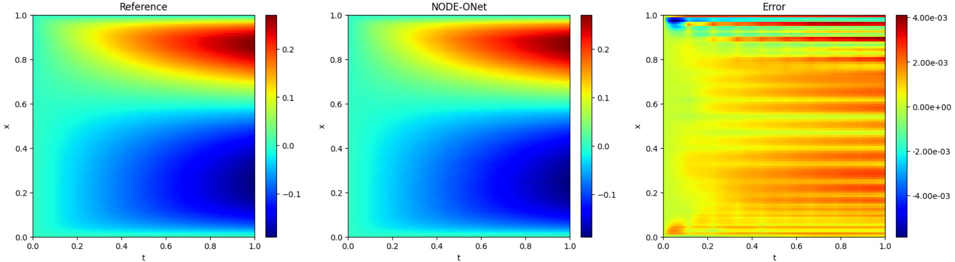
- Source-to-solution operator: $f \mapsto u$. (Comparison with DeepONet)
- Diffusion-to-solution operator: $D \mapsto u$.
- Solution operator with multi-inputs: $\{D, f \mapsto u\}$. (Comparison with MIONet)
- Prediction beyond the training time. (Comparison with DeepONet and MIONet)

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

- D=0.01.
- Physics-encoded NODE:

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

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



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

Table: Comparisons of the NODE-ONet with the unstacked DeepONet for learning the source-to-solution operator $\Psi_f^\dagger : f(x) \mapsto u(t,x)$.

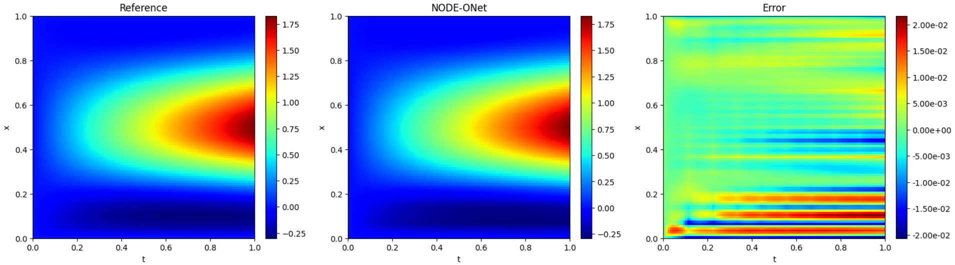
	Training epochs	Training resolutions	#Trainable parameters	#Training input f	Test resolutions	#Test input f	Absolute error	Relative error
NODE-ONet	ADAM 5×10^5	$N_x = 10$ $N_t = 5$ $d_\mathcal{V} = 20$	27,550	100	$N_x = 100$ $N_t = 100$	10,000	4.248×10^{-3}	7.370×10^{-3}
NODE-ONet	ADAM 5×10^5	$N_x =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

1D diffusion-reaction - Multi-inputs operator

- Physics-encoded NODE:

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

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



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: $\Psi_m^\dagger : \{D(x), f(x)\} \mapsto u(t,x)$.

	Training epochs	Training resolutions	#Trainable parameters	#Training $\{D,f\}$	Test resolutions	#Test $\{D,f\}$	Absolute error	Relative error
NODE-ONet	ADAM 1×10^5	$N_x = 50$ $N_t = 10$	28,550	100	$N_x = 100$ $N_t = 100$	5,000	2.362×10^{-2}	5.297×10^{-2}
NODE-ONet	ADAM 1×10^5	$N_x = 100$ $N_t = 10$	28,550	1,000	$N_x = 100$ $N_t = 100$	5,000	4.626×10^{-3}	1.032×10^{-2}
MIONet	ADAM 1×10^5	$N_x = 100$ $N_t = 100$	161,600	100	$N_x = 100$ $N_t = 100$	5,000	1.212×10^{-1}	2.661×10^{-1}
MIONet	ADAM 1×10^5	$N_x = 100$ $N_t = 100$	161,600	1,000	$N_x = 100$ $N_t = 100$	5,000	9.491×10^{-3}	2.072×10^{-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.

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

1D diffusion-reaction - Prediction - Compare with DeepONet

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

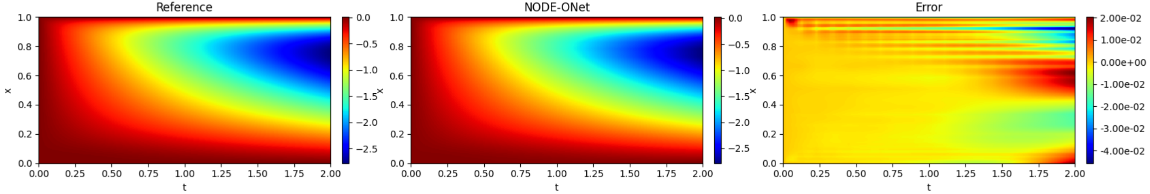
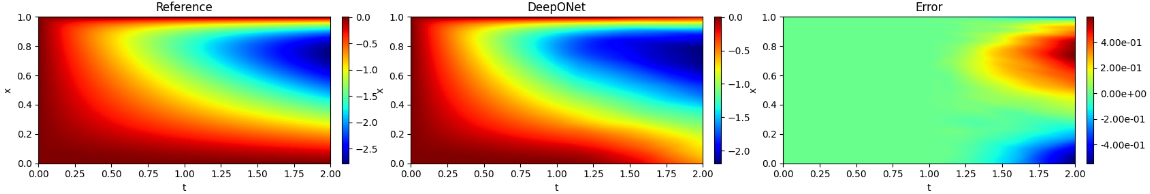


Figure: Prediction of $\Psi_f^* : f \to u$ beyond $t \in [0,1]$ by DeepONet.



1D diffusion-reaction - Prediction - Compare with MIONet

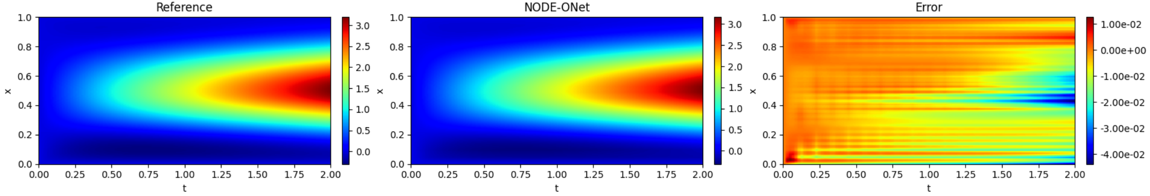


Figure: Prediction of $\Psi_m^* : \{D, f\} \to u$ beyond $t \in [0,1]$ by NODE-ONet.

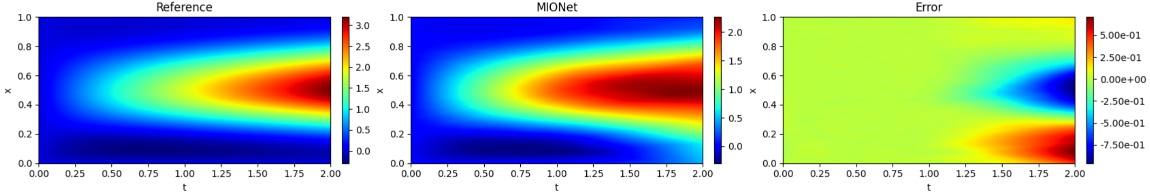


Figure: Prediction of $\Psi_m^* : \{D, f\} \to u$ beyond $t \in [0,1]$ by MIONet.

2D Navier-Stokes - Setup

We consider the following Navier-Stokes equation

$$\begin{cases} \partial_t u(t,x) + \boldsymbol{V}(t,x) \cdot \nabla u(t,x) = \nu \Delta u(t,x) + f(t,x), & \forall (t,x) \in [0,T] \times \Omega, \\ u(t,x) = \nabla \times \boldsymbol{V}(t,x) := \partial_{x_1} \boldsymbol{V}_2 - \partial_{x_2} \boldsymbol{V}_1, & \forall (t,x) \in [0,T] \times \Omega, \\ \nabla \cdot \boldsymbol{V}(t,x) = 0, & \forall (t,x) \in [0,T] \times \Omega, \\ u(0,x) = u_0(x), & \forall x \in \Omega, \end{cases}$$

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

Conclusions

- **Theoretical Foundation:** A general error analysis for encoder-decoder networks is established, providing mathematical insights on operator approximation errors and guiding the design of NODE-ONets.
- **Physics-Encoded NODEs:** By enforcing explicit time dependence in trainable parameters and embedding PDE-specific knowledge (e.g., nonlinear dependencies and effects of known PDE parameters), these NODEs achieve superior generalization while maintaining low model complexity.
- **Numerical Efficiency:** The NODE-ONets outperform state-of-the-art methods (e.g., DeepONets, MIONet) in terms of numerical accuracy, model complexity, and training cost, especially for learning operators with multi-input functions.
- **Generalization:** Trained encoders/decoders can be transferred to related PDEs without retraining, and predictions remain satisfactory beyond the training time horizon.
- **Flexibility:** The framework accommodates various encoders/decoders (e.g., neural networks, Fourier bases) and adapts to stationary and non-stationary PDEs.

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. arXiv:2510.15651, 2025.


Danke
Merci
Cảm ơn bạn
Teşekkür ederiz
Grazie
谢谢
Obrigado
謝謝
Gracias
Thank you
شكرا
Terima kasih
धन्यवाद
ధన్యవాదాలు
Спасибо
شكرا
நன்றி
धन्यवाद
ありがとう
ขอบคุณ
감사합니다