Journal of Computational Physics 498 (2024) 112638

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Energy-dissipative evolutionary deep operator neural networks

Jiahao Zhang [a,1], Shiheng Zhang [a,1], Jie Shen [b,*], Guang Lin [a,c,*]

[a] Department of Mathematics, Purdue University, 150 N. University Street, West Lafayette, IN 47907-2067, USA
[b] School of Mathematical Science, Eastern Institute of Technology, Ningbo, Zhejiang 315200, PR China
[c] School of Mechanical Engineering, Purdue University, 585 Purdue Mall, West Lafayette, IN 47907-2067, USA

ARTICLE INFO

Keywords:
Operator learning
Evolutionary neural networks
Energy dissipative
Parametric equation
Scalar auxiliary variable
Deep learning

ABSTRACT

Energy-Dissipative Evolutionary Deep Operator Neural Network is an operator learning neural network. It is designed to seek numerical solutions for a class of partial differential equations instead of a single partial differential equation, such as partial differential equations with different parameters or different initial conditions. The network consists of two sub-networks, the Branch net and the Trunk net. For an objective operator $\mathcal{G}$, the Branch net encodes different input functions f at a fixed number of sensors $x_i, i = 1, 2, \cdots, m$, and the Trunk net evaluates the output function at any location. By minimizing the error between the evaluated output q and the expected output $\mathcal{G}(f)(y)$ at the test point y, DeepONet generates a good approximation of the operator $\mathcal{G}$. A key distinction in our methodology is the utilization of DeepONet for the training of the initial state, which operates as a multi-parametric operator. Further, the evolution of parameters in our model is facilitated by the Scalar Auxiliary Variable (SAV) method, leading to a formulated iterative process for the parameter's progression over time. The SAV approach is adopted to preserve essential physical properties of PDEs, particularly the Energy Dissipation Law. It introduces a kind of modified energy and establishes unconditional energy dissipation law in the discrete level. By treating the parameters of the well-trained DeepONet as a representation of the initial operator and evolving them by a dynamic system, our network can predict the accurate solution at any further time, while the training data is only the initial state. In order to validate the accuracy and efficiency of our neural networks, we provide numerical simulations of several partial differential equations, including heat equations, parametric heat equations, Allen-Cahn equations and a reaction–diffusion equation in three dimensions.

1. Introduction

Operator learning is a popular and challenging problem with potential applications across various disciplines. The opportunity to learn an operator over a domain in Euclidean spaces [1] and Banach spaces [2] opens a new class of problems in neural network design with generalized applicability. In application to solve partial differential equations (PDEs), operator learning has the potential to predict accurate solutions for the PDEs by acquiring extensive prior knowledge [3–11]. In a recent paper [12], Lu, Jin, and Karniadakis proposed an operator learning method with some deep operator networks, named as DeepONets. It is based on the universal approximation theorem [13–15]. The goal of this neural network is to learn an operator instead of a single function. By

* Corresponding authors.
E-mail addresses: jshen@eitech.edu.cn (J. Shen), guanglin@purdue.edu (G. Lin).
[1] These two authors contributed equally to this work.

https://doi.org/10.1016/j.jcp.2023.112638
Received 2 April 2023; Received in revised form 26 September 2023; Accepted 12 November 2023

Available online 22 November 2023

denoting $\Omega = \mathbb{R}^l \times [0,T]$ and $f = f(x,t)$, we can define an operator $\mathcal{G}$ by taking f as the input function, $\mathcal{G}(f)$ as the output function, such that for any (x,t) in Ω, $\mathcal{G}(f)(x,t) \in \mathbb{R}$. DeepONet takes the discrete input function and approximates the operator by a network.

Before delving into our network, it is pertinent to revisit the architecture of DeepONet. In DeepONet, the term "sensors" refers to specific finite locations within Ω, exemplified as $\{(x_1,t_1),(x_2,t_2),\cdots,(x_m,t_m)\}$. The DeepONet architecture is bifurcated into the Branch net and the Trunk net. The Branch net undertakes the task of encoding the input function f at these fixed sensors. The resultant output from the Branch net comprises p neurons, where each neuron is conceptualized as a scalar of the form $b_k = b_k(f(x_1,t_1), f(x_2,t_2),\cdots,f(x_m,t_m))$ for $k = 1,2,\cdots,p$. Conversely, the Trunk net encodes evaluation points denoted as $\{y_j \in \Omega | j = 1,\cdots,n\}$. Its output is also structured into p neurons, with each neuron represented as a scalar: $g_k = g_k(y_1,y_2,\cdots,y_n)$ for $k = 1,2,\cdots,p$. It's crucial to note that the neuron count in the final layers of both the Trunk and Branch nets is identical. This permits the DeepONet output to be articulated as an inner product between the vectors $(b_1,b_2,\cdots,b_p)^T$ and $(g_1,g_2,\cdots,g_p)^T$. Hence, the relation linking the expected and evaluated outputs is delineated as $\mathcal{G}(f)(y) \approx \sum_{k=1}^{p} b_k g_k$. DeepONet essentially stands as an embodiment of the Universal Approximation Theorem for Operators, as expounded by Chen & Chen [16].

For any time-dependent PDE, the training data is the form of $(f, y, \mathcal{G}(f)(y))$, where f in the discrete form can be represented as $\left(f(x_1,t_1), f(x_2,t_2),\cdots,f(x_m,t_m)\right)^T$ in the neural network. In the original paper, they used the classic FNN [17] as the baseline model. For dynamic systems, various network architectures are used, including residual networks [18], convolutional NNs(CNNs) [19, 20], recurrent NNs(RNNs) [21], neural jump stochastic differential equations [22] and neural ordinary differential equations [23]. DeepONet effectively predicts solutions for a wide array of nonlinear ODEs and PDEs, encompassing systems such as simple dynamics, gravity pendulum, and diffusion-reaction. Nonetheless, a significant challenge arises due to the need for generating training data at each time step, making the network training computationally expensive. In many initial value problems, information about $f(x,t)$ is available only at $t = 0$. This naturally prompts the question: *Is it possible to learn an operator for a time-dependent PDE using solely the initial input functions?*

Inspired by the Evolutionary Deep Neural Network (EDNN) approach [24], it becomes more streamlined to learn an operator at a fixed time rather than an operator governed by both spatial and time variables. Without loss of generality, in the context of initial value problems, the time variable t can be set to 0. Upon acquiring the operator corresponding to the initial time, a plethora of traditional numerical methods become available for solution updating. Specifically, once the initial condition operator is proficiently trained, the parameters of both the Branch net and the Trunk net can be interpreted as functions of the time variable, as depicted in Fig. 1. Elaborating further for a specified initial value problem,

$$\begin{cases} \dfrac{\partial u}{\partial t} = s(u), \\ u(x,0) = u_0(x). \quad x \in X \end{cases} \tag{1}$$

If we denote an operator input f which defines the dynamic of the initial condition u_0, our objective is to approximate an operator $\mathcal{G} : f \mapsto \mathcal{G}(f)$, where $\mathcal{G}(f)(y) = u_0(y)$ for any $y \in X$. Given sensors located at $\{x_1, x_2, \cdots, x_m\}$, the output from the Branch net can be expressed as:

$$\boldsymbol{b} = \boldsymbol{b}(f(x_1), f(x_2), \ldots, f(x_m)) = \boldsymbol{b}(x_1, x_2, \ldots, x_{m_1}; W_1),$$

where W_1 denotes the parameters within the Branch net. Similarly, the output from the Trunk net can be articulated as:

$$\boldsymbol{g} = \boldsymbol{g}(y; W_2),$$

with W_2 being the parameters of the Trunk net. Upon satisfactory training, the parameters are interpreted as a function of t. Here, W_1 and W_2 from the initial DeepONet serve as the initial conditions for $W_1(t)$ and $W_2(t)$ respectively. Leveraging the architecture of the unstacked DeepONet, the solution at the initial time $t_0 = 0$ can be written as:

$$u(x,t_0) \approx \sum_{k=1}^{p} b_k g_k = \boldsymbol{b}^T \boldsymbol{g}. \tag{2}$$

To approximate $u(x,t_1)$, no additional data is required. The values of $u(x,t_1)$ should align with parameters of $W_1(t_1)$ and $W_2(t_1)$. Drawing inspiration from numerical solvers for PDEs, $W_1(t_1)$ and $W_2(t_1)$ can be readily obtained provided that $\frac{\partial W_1}{\partial t}$ and $\frac{\partial W_2}{\partial t}$ are known. Using the chain rule, the time derivative of the solution u can be expressed as:

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial W}\frac{\partial W}{\partial t}, \tag{3}$$

where W encompasses both W_1 and W_2. The term $\frac{\partial W}{\partial t}$ can be determined through a least squares problem. Upon obtaining $\frac{\partial W}{\partial t}$, traditional time discretization schemes can be employed to compute W^{n+1} given W^n.

The selection of the time discretization scheme varies based on the specific problem at hand. Common choices in evolutionary networks include the Euler or Runge–Kutta methods. In the following, we will present the Energy-Dissipative Evolutionary Deep Operator Neural Network (EDE-DeepONet) as Fig. 1, along with an unconditional energy dissipative approach for gradient flow problems.

Many PDEs originate from fundamental physical principles, such as Newton's Law, the Conservation Law, and the Energy Dissipation Law. These principles are pivotal in various scientific and engineering domains, especially within materials science, where

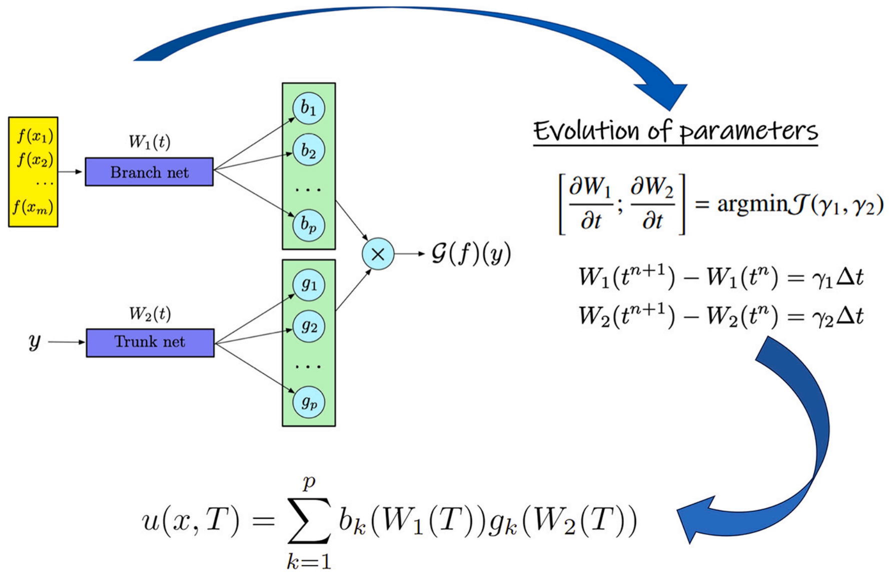

$f(x_1)$
$f(x_2)$
$\dots$
$f(x_m)$
$W_1(t)$
Branch net
b_1
b_2
$\dots$
b_p
$\times$
$\mathcal{G}(f)(y)$
y
$W_2(t)$
Trunk net
g_1
g_2
$\dots$
g_p
Evolution of parameters
$\left[\frac{\partial W_1}{\partial t};\frac{\partial W_2}{\partial t}\right] = \operatorname{argmin}\mathcal{J}(\gamma_1,\gamma_2)$
$W_1(t^{n+1}) - W_1(t^n) = \gamma_1 \Delta t$
$W_2(t^{n+1}) - W_2(t^n) = \gamma_2 \Delta t$
$u(x,T) = \sum_{k=1}^{p} b_k(W_1(T))g_k(W_2(T))$

Fig. 1. Energy-Dissipative Evolutionary Deep Operator Neural Network. The yellow block represents input at sensors, and the blue block represents subnetworks. The green blocks represent the output of the subnetworks and also the last layer of the DeepONet. The parameters in the well-trained DeepONet are used as the initial state of the parameter evolution. The update formula can be obtained by solving the minimization problem. The solution $u(x,T)$ is determined by the parameters at time T. In the right minimization problem, the energy term r^2 can be shown to be dissipative, i.e., $(r^{n+1})^2 \leq (r^n)^2$, where $\mathcal{J}(\gamma_1,\gamma_2) = \frac{1}{2}\left\| \sum_{k=1}^{p} \frac{\partial b_k(W_1^n)}{\partial W_1^n}\gamma_1 g_k(W_2^n) + \sum_{k=1}^{p} b_k(W_1^n)\frac{\partial g_k(W_2^n)}{\partial W_2^n}\gamma_2 + \frac{r^{n+1}}{\sqrt{E(u^n)}}\mathcal{N}(u^n)\right\|_2^2$.

gradient flows are frequently incorporated in mathematical modeling [25–32]. When seeking an approximation for the solution of a specific PDE, ensuring adherence to these laws is crucial. We now turn our attention to a gradient flow problem,

$$\frac{\partial u}{\partial t} = -\frac{\delta E}{\delta u}, \tag{4}$$

where E is a specific free energy functional and $\frac{\delta E}{\delta u}$ is the functional derivative of E. In order to keep the energy dissipative of the above gradient flow problem, we employed the scalar auxiliary variable (SAV) method [33] to formulate a necessary least squares problem. This method introduces a kind of modified energy, ensuring that the unconditionally dissipative modified energy dissipation law holds at each iteration. The SAV method, recognized for its robustness, ease of implementation, and accuracy, has been effectively used to solve numerous PDEs. Integrating this method with neural networks paves the way for a harmonious fusion of neural network models and foundational physical laws.

The objectives of this article are:

- Designing an operator learning neural network without data except the given information.
- Predicting solutions of parametric PDEs at any further time.
- Keeping energy dissipative property of a dynamic system.

Our main contributions are:

- Constructing an evolutionary operator learning neural network to solve PDEs.
- Solving a kind of PDEs with different parameters in a single neural network.
- Introducing the modified energy in the neural network and applying SAV algorithm to keep the unconditionally modified energy dissipation law.
- Introducing an adaptive time stepping strategy and restart strategy in order to speed the training process.

The organization of this paper is as follows: In Section 2, we introduce the Evolutionary Deep Operator Neural Network for a given PDE problem. In Section 3, we consider the physics law behind the gradient flow problem and apply the SAV method to obtain the energy dissipation law. We propose the architecture of EDE-DeepONet. In Section 4, we present two adaptive time stepping strategies, where the second one is called restart in some cases. In Section 5, we generally review the architecture of the EDE-DeepONet. In Section 6, we implement our neural network to predict solutions of heat equations, parametric heat equations, Allen-Cahn equations and a reaction–diffusion equation in three dimensions to show the numerical results. In Section 7, we give some concluding remarks.

2. Evolutionary deep operator neural network

Consider the general form of a gradient flow problem given by

$$\begin{aligned} &\frac{\partial u}{\partial t} + \mathcal{N}(u) = 0, \\ &u(x,0) = u_0(x), x \in X \subseteq \mathbb{R}^l, \end{aligned} \tag{5}$$

where there is a differentiable function F, such that $\mathcal{N}(u) = F'(u)$. By denoting the free energy functional $E[u(x,t)] = \int_X F(u(x,t))\mathrm{d}x$, we obtain $\mathcal{N}(u) = \frac{\delta E}{\delta u}$.

The first step is to represent the initial condition operator using DeepONet.

2.1. Operator learning

For the gradient flow (5), we can define the initial condition operator $\mathcal{G}$ with the input function f, such that

$$\mathcal{G}(f) : y \mapsto u_0(y), \text{ for any } y \in X. \tag{6}$$

Sensors can be selected at the points $\{x_1, x_2, \cdots, x_m\}$. The data input to the DeepONet has the format $(f, y, \mathcal{G}(f)(y))$. The Branch net accepts the vector $\left(f(x_1), f(x_2), \cdots, f(x_m)\right)^T$ as input, which numerically represents f, and produces the output vector $(b_1, b_2, \cdots, b_p)^T$. Meanwhile, the Trunk net receives y as input and outputs $(g_1, g_2, \cdots, g_p)^T$. The DeepONet's output at the test point y can be articulated as:

$$\mathcal{G}(f)(y) \approx \sum_{k=1}^{p} b_k g_k. \tag{7}$$

As previously noted, we operate under the assumption that the initial condition operator is expertly trained. Our next step is to determine the update rule for the parameters to facilitate the evolution of the neural network.

2.2. Parameters evolution within the neural network

Let the parameters of the Branch net be represented by W_1 and those of the Trunk net by W_2. Given that these parameters evolve over time, both W_1 and W_2 can be considered functions of t. Using the chain rule of differentiation, we get

$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial W_1}\frac{\partial W_1}{\partial t} + \frac{\partial u}{\partial W_2}\frac{\partial W_2}{\partial t}. \tag{8}$$

Considering that $u = \sum_{k=1}^{p} b_k g_k = \sum_{k=1}^{p} b_k(W_1(t)) g_k(W_2(t))$, we can differentiate it with respect to t to obtain:

$$\frac{\partial u}{\partial t} = \sum_{k=1}^{p}\left(\frac{\partial b_k(W_1(t))}{\partial W_1(t)}\frac{\partial W_1(t)}{\partial t} g_k(W_2(t)) + b_k(W_1(t))\frac{\partial g_k(W_2(t))}{\partial W_2(t)}\frac{\partial W_2(t)}{\partial t}\right).$$

Our goal is to determine $\frac{\partial W_1(t)}{\partial t}$ and $\frac{\partial W_2(t)}{\partial t}$, which will guide the evolution of the network parameters. This can be achieved by framing it as a minimization problem:

$$\left[\frac{\partial W_1(t)}{\partial t}; \frac{\partial W_2(t)}{\partial t}\right] = \text{ argmin } \mathcal{J}(\gamma_1, \gamma_2), \tag{9}$$

where

$$\mathcal{J}(\gamma_1, \gamma_2) = \frac{1}{2}\left\|\sum_{k=1}^{p}\frac{\partial b_k(W_1(t))}{\partial W_1(t)}\gamma_1 g_k(W_2(t)) + \sum_{k=1}^{p} b_k(W_1(t))\frac{\partial g_k(W_2(t))}{\partial W_2(t)}\gamma_2 + \mathcal{N}(u)\right\|_2^2. \tag{10}$$

In this article, the inner product (a, b) is defined in the integral sense, $(a, b) = \int_X a(x)b(x)\,\mathrm{d}x$ and the L_2 norm is defined as $\|a\|_2^2 = \int_X |a(x)|^2\,\mathrm{d}x$.

The minimization problem can be transformed into a linear system by the first-order optimal condition:

$$\frac{\partial \mathcal{J}}{\partial \gamma_1} = \int_X \left(\sum_{k=1}^{p}\frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t))\right)^T \left(\gamma_1 \sum_{k=1}^{p}\frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t)) + \sum_{k=1}^{p} b_k(W_1(t))\frac{\partial g_k(W_2(t))}{\partial W_2(t)}\gamma_2 + \mathcal{N}(u)\right)\mathrm{d}x = 0$$

$$\frac{\partial \mathcal{J}}{\partial \gamma_2} = \int_X \left(\sum_{k=1}^{p} b_k(W_1(t))\frac{\partial g_k(W_2(t))}{\partial W_2(t)}\right)^T \left(\gamma_1 \sum_{k=1}^{p}\frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t)) + \sum_{k=1}^{p} b_k(W_1(t))\frac{\partial g_k(W_2(t))}{\partial W_2(t)}\gamma_2 + \mathcal{N}(u)\right)\mathrm{d}x = 0$$

In this framework, the gradients with respect to $W_1(t)$ and $W_2(t)$ can be efficiently determined using automatic differentiation for each time increment. Let's denote the following:

$$
\begin{aligned}
(\mathbf{J_1})_{ij_1} &= \sum_{k=1}^{p} \frac{\partial b_k(W_1(t))}{\partial W_1^{j_1}(t)} g_k^i(W_2(t)),\\
(\mathbf{J_2})_{ij_2} &= \sum_{k=1}^{p} b_k(W_1(t)) \frac{\partial g_k^i(W_2(t))}{\partial W_2^{j_2}(t)},\\
(\mathbf{N})_i &= \mathcal{N}\left(u^i\right),
\end{aligned}
$$

where i ranges from 1 to l, j_1 from 1 to N^b_{para}, and j_2 from 1 to N^t_{para}. Here, N^b_{para} denotes the total number of parameters in the Branch net, and N^t_{para} represents the total number of parameters in the Trunk net. $\mathbf{N}$ is the output of the DeepONet and is therefore computationally accessible for any spatial location. The integrals mentioned earlier can be numerically approximated:

$$
\begin{aligned}
&\frac{1}{|X|}\int_X \left(\sum_{k=1}^{p} \frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t))\right)^T \left(\sum_{k=1}^{p} \frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t))\right) \mathrm{d}x = \lim_{N_x\to\infty} \frac{1}{N_x}\mathbf{J_1^T J_1},\\
&\frac{1}{|X|}\int_X \left(\sum_{k=1}^{p} b_k(W_1(t)) \frac{\partial g_k(W_2(t))}{\partial W_2(t)}\right)^T \left(\sum_{k=1}^{p} b_k(W_1(t)) \frac{\partial g_k(W_2(t))}{\partial W_2(t)}\right) \mathrm{d}x = \lim_{N_x\to\infty} \frac{1}{N_x}\mathbf{J_2^T J_2},\\
&\frac{1}{|X|}\int_X \left(\sum_{k=1}^{p} \frac{\partial b_k(W_1(t))}{\partial W_1(t)} g_k(W_2(t))\right)^T \mathcal{N}(u)\mathrm{d}x = \lim_{N_x\to\infty} \frac{1}{N_x}\mathbf{J_1^T N},
\end{aligned}
$$

where N_x is the number of collocation points for numerical integrals. By denoting γ_i^{opt} as the optimal values of γ_i, $i = 1, 2$, the above system can be reduced to

$$
\begin{aligned}
\mathbf{J_1^T}\left(\gamma_1^{opt}\mathbf{J_1} + \gamma_2^{opt}\mathbf{J_2} + \mathbf{N}\right) &= 0,\\
\mathbf{J_2^T}\left(\gamma_1^{opt}\mathbf{J_1} + \gamma_2^{opt}\mathbf{J_2} + \mathbf{N}\right) &= 0.
\end{aligned}
$$

The feasible solutions of the above equations are the approximated time derivatives of $W_1(t)$ and $W_2(t)$.

$$
\begin{aligned}
\frac{dW_1(t)}{dt} &= \gamma_1^{opt},\\
\frac{dW_2(t)}{dt} &= \gamma_2^{opt},
\end{aligned}
$$

where the initial conditions W_1^0 and W_2^0 can be determined by DeepONets for initial condition operators. The decoupled ODEs represent the update rules within the neural networks. One straightforward approach to solving these ODEs is to use the explicit Euler method:

$$
\begin{aligned}
\frac{W_1^{n+1} - W_1^n}{\Delta t} &= \gamma_1^{opt},\\
\frac{W_2^{n+1} - W_2^n}{\Delta t} &= \gamma_2^{opt}.
\end{aligned}
$$

Consequently, the solution at the subsequent time step, $u(x, t^{n+1})$, is given by:

$$
u(x, t^{n+1}) = \sum_{k=1}^{p} b_k(W_1^{n+1}) g_k(W_2^{n+1}). \tag{11}
$$

3. Energy dissipative evolutionary deep operator neural network

Let's revisit the problem at hand. The gradient flow problem is given by:

$$
\begin{aligned}
&\frac{\partial u}{\partial t} + \mathcal{N}(u) = 0,\\
&u(x, 0) = u_0(x),
\end{aligned} \tag{12}
$$

where the energy functional is given by

$$
E[u(x,t)] = \int_X F(u(x,t))\mathrm{d}x.
$$

This energy functional is assumed to have a positive lower bound. Additionally, $\mathcal{N}(u)$ can be expressed in terms of the variational derivative of $E(u)$, that is, $\mathcal{N}(u) = \frac{\delta E}{\delta u}$.

Taking the inner product of the first equation of (12) with $\mathcal{N}(u)$, we can derive the energy dissipation property of the system:

$$\frac{dE(u)}{dt} = \left(\frac{\delta E}{\delta u}, \frac{\partial u}{\partial t}\right) = \left(\mathcal{N}(u), \frac{\partial u}{\partial t}\right) = -\left(\mathcal{N}(u), \mathcal{N}(u)\right) \le 0. \tag{13}$$

However, it is usually hard for a numerical algorithm to be efficient as well as energy dissipative. Recently, the SAV approach [33] was introduced to construct numerical schemes which is energy dissipative (with a modified energy), accurate, robust and easy to implement. More precisely, assuming $E[u(x)] > 0$, it introduces a $r(t) = \sqrt{E[u(x,t)]}$, and expands the gradient flow problem as

$$\begin{aligned} \frac{\partial u}{\partial t} &= -\frac{r}{\sqrt{E(u)}}\mathcal{N}(u), \\ \frac{dr}{dt} &= \frac{1}{2\sqrt{E(u)}}\left(\mathcal{N}(u), \frac{\partial u}{\partial t}\right). \end{aligned} \tag{14}$$

With $r(0) = \sqrt{E[u(x,0)]}$, the above system has a solution $r(t) \equiv \sqrt{E[u(x,t)]}$ and u being the solution of the original problem.

By setting $u^n = \sum_{k=1}^{p} g_k b_k$, a first order scheme can be constructed as

$$\begin{aligned} \frac{u^{n+1}-u^n}{\Delta t} &= -\frac{r^{n+1}}{\sqrt{E(u^n)}}\mathcal{N}(u^n) \\ \frac{r^{n+1}-r^n}{\Delta t} &= \frac{1}{2\sqrt{E(u^n)}}\int_X \mathcal{N}(u^n)\frac{u^{n+1}-u^n}{\Delta t}dx. \end{aligned} \tag{15}$$

This is a coupled system of equations for (r^{n+1}, u^{n+1}). But it can be easily decoupled as follows. Plugging the first equation into the second one, we obtain:

$$\frac{r^{n+1}-r^n}{\Delta t} = -\frac{r^{n+1}}{2E(u^n)}\left\|\mathcal{N}(u^n)\right\|^2, \tag{16}$$

which implies

$$r^{n+1} = \left(1 + \frac{\Delta t}{2E(u^n)}\left\|\mathcal{N}(u^n)\right\|^2\right)^{-1} r^n \tag{17}$$

Theorem 3.1 *(Discrete Energy Dissipation Law). With the modified energy define above, the scheme is unconditionally energy stable, i.e.*

$$(r^{n+1})^2 - (r^n)^2 \le 0. \tag{18}$$

Proof 3.1. *Taking the inner product of the first equation with* $\frac{r^{n+1}}{\sqrt{E(u^n)}}\mathcal{N}(u^n)$ *and the second equation with* $2r^{n+1}$

$$\begin{aligned} (r^{n+1})^2 - (r^n)^2 &= 2r^{n+1}(r^{n+1}-r^n) - (r^{n+1}-r^n)^2 \\ &= \frac{\Delta t r^{n+1}}{\sqrt{E(u^n)}}\int_X \mathcal{N}(u^n)\frac{u^{n+1}-u^n}{\Delta t}dx - (r^{n+1}-r^n)^2 \\ &= -\Delta t\left(\frac{r^{n+1}}{\sqrt{E(u^n)}}\right)^2\int_X \mathcal{N}(u^n)\mathcal{N}(u^n)dx - (r^{n+1}-r^n)^2 \\ &\le 0 \end{aligned} \tag{19}$$

In order to maintain the modified energy dissipation law in the evolution neural network, we only need to replace $\mathcal{N}(u^n)$ by $\frac{r^{n+1}}{\sqrt{E(u^n)}}\mathcal{N}(u^n)$ in section 2. The update rule of the neural network is

$$\left[\frac{\partial W_1(t)}{\partial t}; \frac{\partial W_2(t)}{\partial t}\right] = \operatorname{argmin}\mathcal{J}(\gamma_1, \gamma_2), \tag{20}$$

where

$$\mathcal{J}(\gamma_1,\gamma_2) = \frac{1}{2}\left\|\sum_{k=1}^{p}\frac{\partial b_k(W_1^n(t))}{\partial W_1^n(t)}\gamma_1 g_k(W_2^n(t)) + \sum_{k=1}^{p} b_k(W_1^n(t))\frac{\partial g_k(W_2^n(t))}{\partial W_2^n(t)}\gamma_2 + \frac{r^{n+1}}{\sqrt{E(u^n)}}\mathcal{N}(u^n)\right\|_2^2. \tag{21}$$

The corresponding linear system of the first order optimal condition is

$$\begin{aligned}&\mathbf{J}_1^{\mathbf{T}}\left(\gamma_1^{opt}\mathbf{J}_1+\gamma_2^{opt}\mathbf{J}_2+\frac{r^{n+1}}{\sqrt{E(u^n)}}\mathbf{N}\right)=0,\\&\mathbf{J}_2^{\mathbf{T}}\left(\gamma_1^{opt}\mathbf{J}_1+\gamma_2^{opt}\mathbf{J}_2+\frac{r^{n+1}}{\sqrt{E(u^n)}}\mathbf{N}\right)=0.\end{aligned}\tag{22}$$

After getting γ_1^{opt} and γ_2^{opt}, W^{n+1} can be obtained by the Forward Euler method:

$$\begin{aligned}&W_1^{n+1}=W_1^n+\gamma_1^{opt}\Delta t,\\&W_2^{n+1}=W_2^n+\gamma_2^{opt}\Delta t.\end{aligned}\tag{23}$$

Therefore, the solution for the next time step, $u(x,t^{n+1})$, can be expressed as:

$$u(x,t^{n+1})=\sum_{k=1}^{p}b_k(W_1^{n+1})g_k(W_2^{n+1}).\tag{24}$$

4. Adaptive time stepping strategy and restart strategy

A key benefit of an unconditionally stable scheme is the flexibility to employ an adaptive time step. Given that the coefficient of N, represented by $\frac{r^{n+1}}{\sqrt{E(u^n)}}$, should approximate 1, we can define $\xi^{n+1}=\frac{r^{n+1}}{\sqrt{E(u^n)}}$. If ξ is near 1, a larger Δt is permissible, whereas if ξ diverges significantly from 1, a smaller Δt is more appropriate. This leads us to the following straightforward adaptive time-stepping approach:

Algorithm 1 Adaptive time stepping strategy.

```
1. Set the tolerance for $\xi$ as $\epsilon_0$ and $\epsilon_1$, the initial time step $\Delta t$, the maximum time step $\Delta t_{max}$ and the minimum time step $\Delta t_{min}$
2. Compute $u^{n+1}$.
3. Compute $\xi^{n+1}=\frac{r^{n+1}}{\sqrt{E(u^n)}}$.
4. **If** $|1-\xi^{n+1}|>\epsilon_0$,
      **Then** $\Delta t=\max(\Delta t_{min},\Delta t/2)$;
   **Else if** $|1-\xi^{n+1}|<\epsilon_1$,
      **Then** $\Delta t=\min(\Delta t_{max},2\Delta t)$.
   **Go to Step 2**.
5. Update time step $\Delta t$.
```

Another widely adopted approach to ensure that r^{n+1} closely mirrors the original energy $E(u^{n+1})$ involves resetting $r^{n+1}=E(u^{n+1})$ under certain conditions. The detailed algorithm is outlined below:

Algorithm 2 Restart strategy.

```
1. Set the tolerance for $\xi$ as $\epsilon_2$.
2. Compute $u^{n+1}$.
3. Compute $\xi^{n+1}=\frac{r^{n+1}}{\sqrt{E(u^n)}}$.
4. **If** $|1-\xi^{n+1}|>\epsilon_2$,
      **Then** $r^{n+1}=\sqrt{E(u^{n+1})}$ and **Go to Step 2**.
5. Go to next iteration.
```

The choice for ϵ_0, ϵ_1 should be some small tolerance, usually 10^{-1} and 10^{-3}. The choices for Δt_{max} and Δt_{min} are quite dependent on Δt, usually $\Delta t_{max}=10^3\times\Delta t$ and $\Delta t_{min}=10^{-3}\times\Delta t$. In Algorithm 2, we usually take ϵ_2 as 2×10^{-2}.

5. Algorithm for EDE-DeepONet

A methodical approach to solving time-dependent PDEs with EDE-DeepONet is delineated in Algorithm 3.

6. Numerical experiments

In this section, we utilize EDE-DeepONet to solve heat equations, parametric heat equations, Allen-Cahn equations, and reaction-diffusion equations, demonstrating its efficacy and accuracy.

Algorithm 3 Energy-Dissipative Evolutionary Deep Operator Neural Networks (EDE-DeepONet).

```
1. Initialization: Prepare input data samples in the form of $(f, y, \mathcal{G}(f)(y))$, where $f$ is the operator input defining the dynamic of the initial condition. Each input function $f$ is crafted using the same sensor locations $\{x_1, x_2, \cdots, x_m\}$.
2. DeepONet Training:
a. Input $(f(x_1), f(x_2), \cdots, f(x_m))^T$ into the Branch net and $y$ into the Trunk net.
b. Compute the output of the DeepONet and denote it as $q$.
c. Adjust the parameters in the DeepONet by minimizing the mean squared error: $\frac{1}{|Y|}\sum_{y\in Y}\|\mathcal{G}(f)(y)-q\|^2$. Denote the parameters in the well-trained Branch net and Trunk net as $W_1^0$ and $W_2^0$.
3. Parameter Evolution:
a. Using the well-trained DeepONet, solve the system of equations (22) to derive $\left[\frac{\partial W_1(t)}{\partial t}; \frac{\partial W_2(t)}{\partial t}\right]$.
b. Given that the parameters $W_1^0$ and $W_2^0$ are known, compute new parameters $W_1^n$ and $W_2^n$ using equations (23) iteratively.
4. Output: Compute the solution at time $t_n$ using DeepONet by $u(x, t_n) = \sum_{k=1}^{p} b_k(W_1^n) g_k(W_2^n)$.
```

Table 1
The mean squared error of the heat equation: The initial input function is $u_0(x) = a\sin(\pi x)$.

Error	T = 0.025	T = 0.05	T = 0.075	T = 0.1
a = 1.0	1.47×10^{-5}	1.33×10^{-5}	1.32×10^{-5}	1.29×10^{-5}
a = 1.5	5.11×10^{-6}	7.05×10^{-6}	8.48×10^{-6}	9.81×10^{-6}
a = 1.8	1.46×10^{-5}	1.69×10^{-5}	1.79×10^{-5}	1.83×10^{-5}
a = 2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	1.72×10^{-5}

6.1. Example 1: simple heat equations

To demonstrate the accuracy of EDE-DeepONet, we begin by considering the simple heat equation with various initial conditions for which we already possess exact solutions.

A one-dimensional heat equation system can be described by the following partial differential equation (PDE):

$$u_t = u_{xx},$$
$$u(x,0) = u_0,$$
$$u(0,t) = u(2,t) = 0.$$

Using the method of separation of variables, we can derive the solution to the heat equation. If we set $u_0(x) = a\sin(\pi x)$, the solution is given by $u(x,t) = a\sin(\pi x)e^{-\pi^2 t}$, where $a \in [1,2]$. The corresponding energy can be calculated as $E(u) = \int_0^2 \frac{1}{2}|u_x|^2 dx \approx \Delta x(\sum_{i=1}^{n} \frac{1}{2}|u_x(x_i)|^2)$.

To generate initial data samples, we choose 51 points uniformly from the interval $[0,2)$ for x and 50 random values of a from $[1,2]$. The time step for updating the parameters in the neural network is 2.5×10^4, and the number of iteration steps is 400. We compare the solutions for different values of a, where $a = 1.0$, 1.5, 1.8, and 2.5. Even though $a = 2.5$ is outside the range of training data, the model performs well. We calculate the mean squared error for different a as shown in Table 1. In this article, the mean squared error is defined as $e(u,\hat{u}) = \frac{1}{N}\sum_{k=1}^{N}(u(x_k) - \hat{u}(x_k))^2$, where u is the solution obtained by EDE-DeepONet, and $\hat{u}$ is the reference solution.

To illustrate the relationship between the modified energy and the original energy, we compare r^2 and E at each step (see Fig. 2). Both energies are dissipative in EDE-DeepONet, except when the restart strategy is applied. The restart strategy is used to make r^2 approach E. The modified energy is initialized when the restart strategy is applied. The restart strategy was triggered on the 370th step to realign the modified energy with the real energy. After that, they followed the same trajectory again. It is evident that the modified energy approaches the original energy both before and after the restart strategy being applied.

In Fig. 3, we provide a comparison between the exact solution and the solution obtained by EDE-DeepONet. From this analysis of the simple heat equation, we conclude that EDE-DeepONet accurately predicts the solution of the PDE. Most importantly, EDE-DeepONet can predict the solution not only within the training subset range but also beyond it. For instance, we tested with $a = 2.5$ while a was in the range $[1,2]$ during training. EDE-DeepONet demonstrates good accuracy compared to the exact solution, as shown in Fig. 3 and Table 1.

6.2. Example 2: parametric heat equations

In Example 1, we considered different initial conditions as our inputs. In Example 2, we will tackle parametric heat equations.

A one-dimensional parametric heat equation can be described as:

$$u_t = cu_{xx},$$
$$u(x,0) = \sin(\pi x),$$
$$u(0,t) = u(2,t) = 0.$$

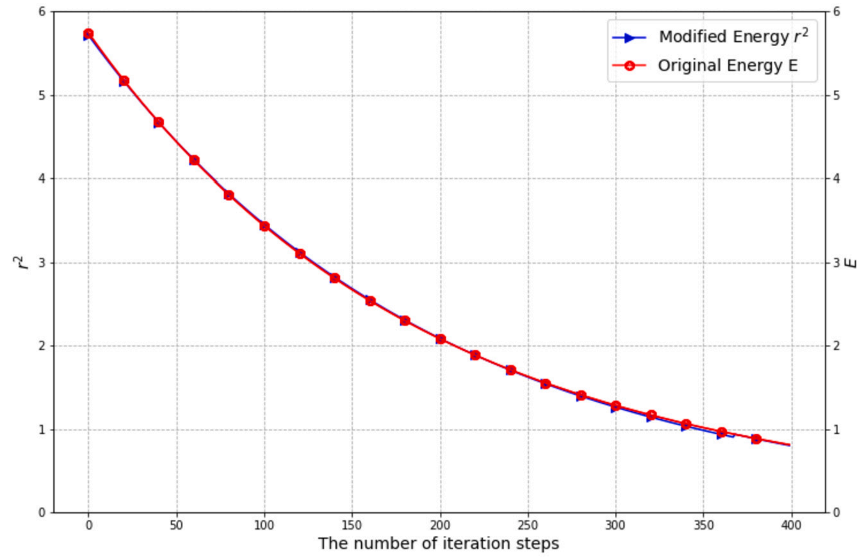
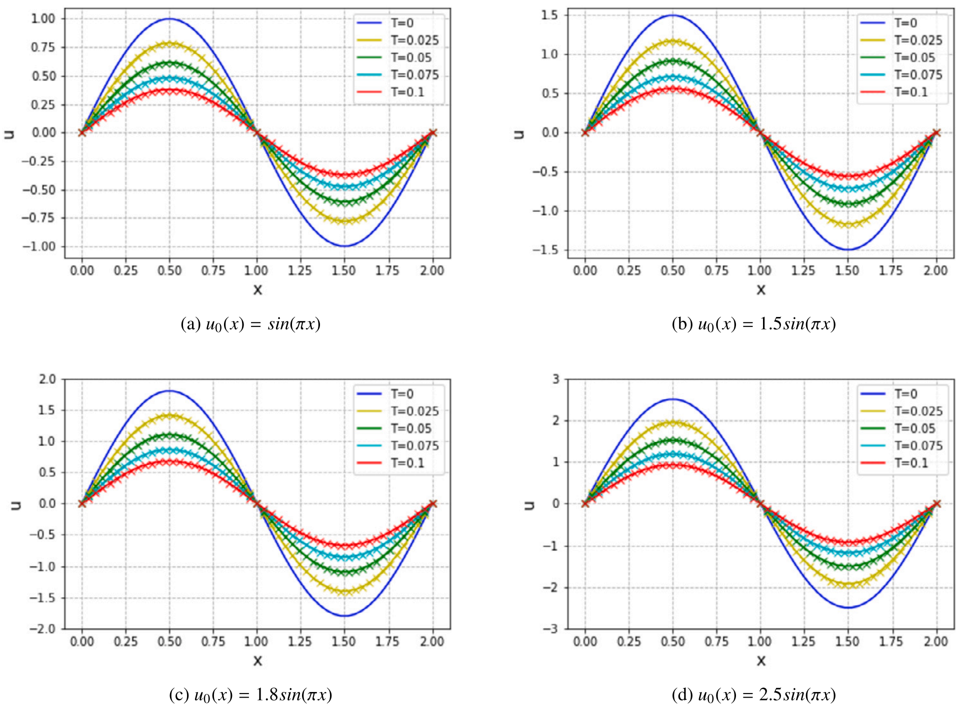


Fig. 2. Energy comparison for the heat equation: This figure showcases the comparison between the modified and original energies during the training of EDE-DeepONet. Each iteration corresponds to one forward step of the PDE's numerical solution with a timestep, $\Delta t = 2.5 \times 10^{-4}$. Notably, in EDE-DeepONet, both energy forms are dissipative, with the sole exception of the restart strategy implemented. This strategy ensures that r^2 converges towards E. Specifically, the modified energy is reset at the 370th step. It's crucial to observe the alignment of the modified and original energies along the same trajectory both before and after this pivotal 370th step.



(a) $u_0(x) = sin(\pi x)$

(b) $u_0(x) = 1.5sin(\pi x)$

(c) $u_0(x) = 1.8sin(\pi x)$

(d) $u_0(x) = 2.5sin(\pi x)$

Fig. 3. The heat equation: The solution with 4 different initial input functions $u_0(x) = a\sin(\pi x)$. The curve represents the solution obtained by the EDE-DeepONet, and xxx represents the reference solution. The training parameter a is in the range of $[1, 2)$, so we give three examples in this range. We also present the case out of the range. It also shows accuracy in Fig. 3d.

This PDE is more complex than the one in Example 1 because the parameter c is embedded within the equation. Traditional numerical schemes require multiple runs to handle cases with different parameters, as they essentially represent distinct equations. In contrast, EDE-DeepONet only needs to be trained once. We choose the training range of c as $[1, 2)$ and select 51 points for x and c in a manner similar to Example 1. To incorporate the parameter c into the initial state, we can compute the solution after a small time step using traditional methods. This approach allows us to obtain various initial states corresponding to different values of c.

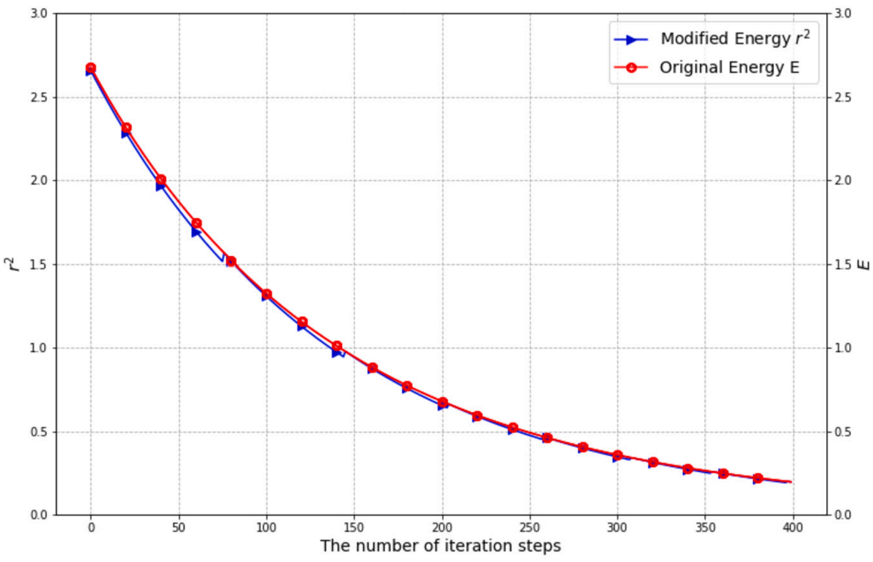


Fig. 4. Energy comparison for the parametric heat equation: We present the modified energy and original energy when training EDE-DeepONet. Each iteration step represents one forward step of the PDE's numerical solution with $\Delta t = 2.5 \times 10^{-4}$. These types of PDEs are inherently complex, necessitating multiple restarts during the training phase. While the original energy consistently diminishes, the modified energy effectively approximates it.

Table 2
The mean squared error of the parametric heat equation.

Error	T = 0.025	T = 0.05	T = 0.075	T = 0.1
c = 1.2	1.30×10^{-5}	1.43×10^{-5}	1.35×10^{-5}	1.20×10^{-5}
c = 1.5	1.35×10^{-5}	1.27×10^{-5}	9.80×10^{-6}	7.80×10^{-6}
c = 1.8	1.17×10^{-5}	1.03×10^{-5}	7.88×10^{-5}	1.83×10^{-5}
c = 2.5	2.20×10^{-4}	1.34×10^{-4}	6.02×10^{-5}	7.08×10^{-6}

First, we compare the modified energy with the original energy, as shown in Fig. 4. The energy differs from the first example because it depends on the parameter c. To represent the energy of the system, we compute the average energy across different values of c. This case is more complex than the first one, requiring more restarts during training. Although the modified energy oscillates when the restart strategy is used, it consistently decreases after each restart.

Second, we present the mean squared error between the solution obtained by EDE-DeepONet and the reference solution in Table 2. The reference solution can be obtained explicitly using the variable separation method, and the error is defined in the same way as in Example 1.

Third, we provide a comparison between our solution and the reference solution in Fig. 5. Similar to Example 1, we predict the solution for values of c outside the range of $[1, 2]$, and all results demonstrate excellent accuracy. Therefore, EDE-DeepONet is capable of solving parametric PDEs effectively.

6.3. Example 3: Allen-Cahn equations

We will now demonstrate results for a PDE with a more complex energy. The Allen-Cahn equation is developed to describe phase separation processes. Originally formulated for materials science, it has found applications in representing moving interfaces within phase-field models in fluid dynamics. The Allen-Cahn equation can be considered as a gradient flow in L^2 with a specific energy. We will discuss both the one-dimensional case and two-dimensional case as follows:

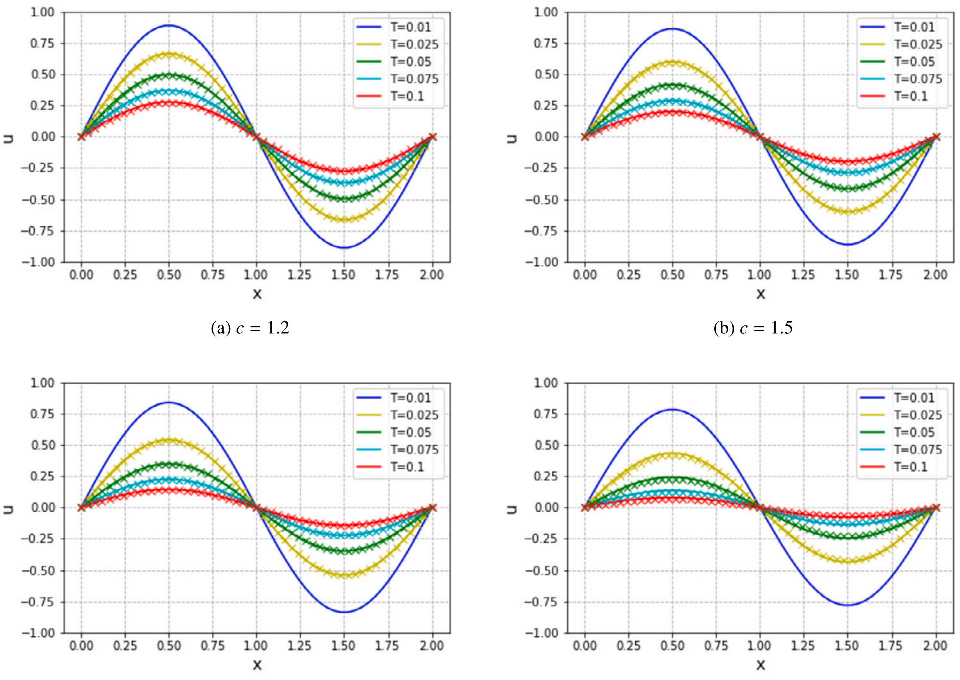
6.3.1. One-dimensional case

(a) Various initial conditions:

We will begin with the one-dimensional Allen-Cahn equation, which can be described by the following equations:

$$u_t = u_{xx} - g(u),$$
$$u(x, 0) = \sum_{j=1}^{3} a_j \sin(j\pi x) + \sum_{j=1}^{3} b_j \cos(j\pi x)$$

with periodic boundary conditions. The corresponding Ginzburg–Landau free energy is defined as:



(a) $c = 1.2$ (b) $c = 1.5$

(c) $c = 1.8$ (d) $c = 2.5$

Fig. 5. The parametric heat equation: The solution with 4 different parameters c. The curve represents the solution obtained by the EDE-DeepONet and xxx represents the reference solution. The training parameter c is in the range of $[1,2)$, so we give 3 examples in this range. We also present the case out of the range in Fig. 5d.

Table 3
The mean squared error of the one-dimensional Allen-Cahn equation with the initial input function $u_0(x) = \sum_{j=1}^{3} a_j sin(j\pi x) + \sum_{j=1}^{3} b_j cos(j\pi x)$.

$a_1, a_2, a_3, b_1, b_2, b_3$	T = 0.02	T = 0.04	T = 0.06
0.07, 0.03, 0.06, 0.13, 0.03, 0.09	2.93×10^{-4}	1.67×10^{-3}	2.11×10^{-3}
0.03, 0.06, 0.08, 0.06, 0.08, 0.08	2.90×10^{-4}	4.62×10^{-4}	1.59×10^{-4}
0.18, 0.01, 0.01, 0.02, 0.04, 0.14	2.36×10^{-4}	1.02×10^{-3}	1.09×10^{-3}

$$E[u] = \int_0^1 \frac{1}{2}|u_x|^2 dx + \int_{x=0}^{x=1} G(u) dx.$$

Here, we have $G(u) = \frac{1}{4\epsilon^2}(u^2-1)^2$ and $g(u) = G'(u) = \frac{1}{\epsilon^2}u(u^2-1)$, with $\epsilon = 0.1$. The parameter ϵ influences the width of the jump in the steady state.

In our EDE-DeepONet configuration, we define Δt as 5×10^{-4} and choose a spatial point count, N_x, of 201. The initial input functions are crafted as linear combinations of $\sin(j\pi x)$ and $\cos(j\pi x)$, where $j = 1,2,3$. The parameters a_j and b_j are drawn from a uniform distribution spanning $[0,1]^6$. They are then normalized such that $\sum a_i + \sum b_i = 0.4$, a measure to prevent the initial input function's amplitude from becoming excessively large. A comparison between the modified and original energies can be observed in Fig. 6. Notably, the modified energy closely follows the trajectory of the original, even when presented in this intricate form.

Subsequently, we compared the solutions derived using EDE-DeepONet, incorporating three distinct sets of randomly generated a_j, b_j parameters, with the benchmark solution ascertained through the traditional numerical SAV method. This comparison is illustrated in Fig. 7. The associated errors are tabulated in Table 3. In this table, the first column showcases the specific a_j values and the second column showcases the b_j values. Both a_j and b_j are rounded to two decimal places for clarity.

This example underscores the versatility of EDE-DeepONet in managing PDEs equipped with a plethora of initial input functions anchored to a given basis function. Such flexibility offers the potential to probe into any initial state, granted an ample selection of basis functions is at our disposal.

(b) Various thickness of the interface:

Heuristically, ϵ represents the thickness of the interface in the phase separation process. Obtaining a sharp interface as $\epsilon \to 0$ with time evolution is a key aspect. The theoretical and numerical analysis of this limit plays a crucial role in understanding the equation, as discussed in [34,35].

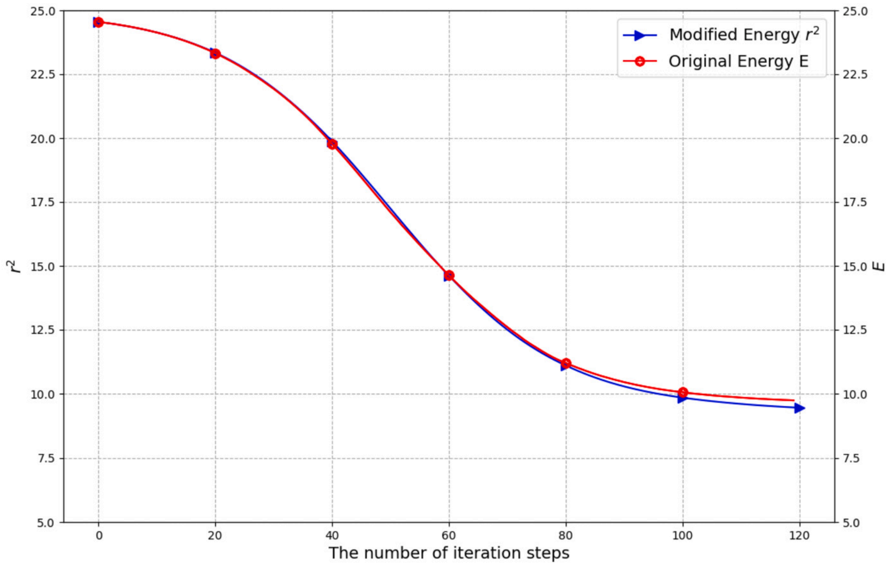
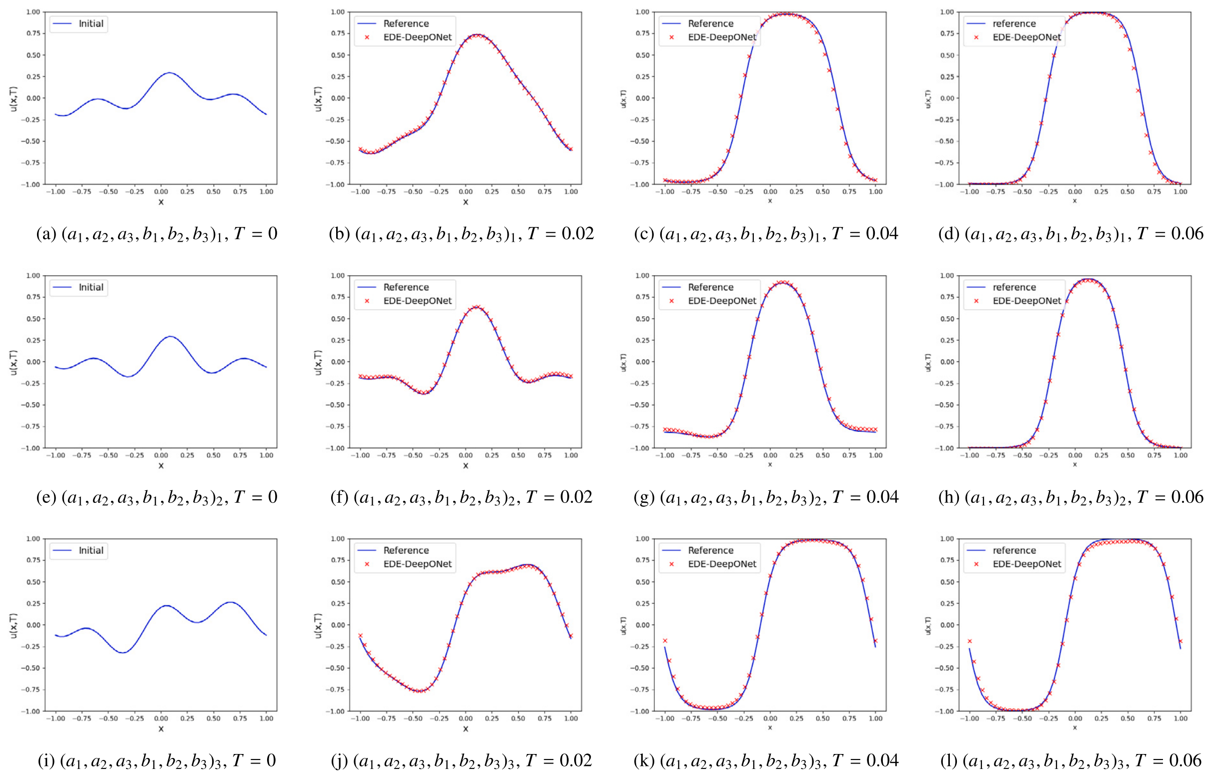


Fig. 6. The one-dimensional Allen-Cahn Equation: The figure displays both the modified and original energies during network training. Each iteration corresponds to a single forward step in the numerical solution of the PDE, with a time increment of $\Delta t = 5 \times 10^{-4}$. Notably, the modified energy closely mirrors the trends observed in the original energy.



(a) $(a_1, a_2, a_3, b_1, b_2, b_3)_1$, $T = 0$ (b) $(a_1, a_2, a_3, b_1, b_2, b_3)_1$, $T = 0.02$ (c) $(a_1, a_2, a_3, b_1, b_2, b_3)_1$, $T = 0.04$ (d) $(a_1, a_2, a_3, b_1, b_2, b_3)_1$, $T = 0.06$

(e) $(a_1, a_2, a_3, b_1, b_2, b_3)_2$, $T = 0$ (f) $(a_1, a_2, a_3, b_1, b_2, b_3)_2$, $T = 0.02$ (g) $(a_1, a_2, a_3, b_1, b_2, b_3)_2$, $T = 0.04$ (h) $(a_1, a_2, a_3, b_1, b_2, b_3)_2$, $T = 0.06$

(i) $(a_1, a_2, a_3, b_1, b_2, b_3)_3$, $T = 0$ (j) $(a_1, a_2, a_3, b_1, b_2, b_3)_3$, $T = 0.02$ (k) $(a_1, a_2, a_3, b_1, b_2, b_3)_3$, $T = 0.04$ (l) $(a_1, a_2, a_3, b_1, b_2, b_3)_3$, $T = 0.06$

Fig. 7. One-dimensional Allen-Cahn equation with different initial states $u_0(x) = \sum_{j=1}^{3} a_j sin(j\pi x) + \sum_{j=1}^{3} b_j cos(j\pi x)$: The coefficients are randomly chosen as $(a_1, a_2, a_3, b_1, b_2, b_3)_1 = (0.07, 0.03, 0.06, 0.13, 0.03, 0.09)$, $(a_1, a_2, a_3, b_1, b_2, b_3)_2 = (0.03, 0.06, 0.08, 0.06, 0.08, 0.08)$, $(a_1, a_2, a_3, b_1, b_2, b_3)_3 = (0.18, 0.01, 0.01, 0.02, 0.04, 0.14)$. The curve represents the reference solutions obtained by SAV method, and xxx represents the solutions generated by EDE-DeepONet. We compared the solutions at $T = 0.02$, $T = 0.04$, $T = 0.06$.

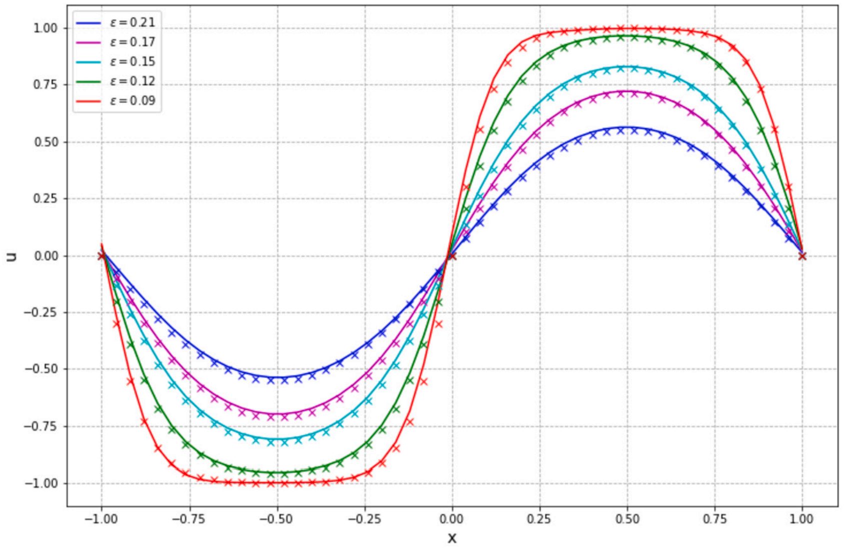


Fig. 8. One dimensional Allen-Cahn equation: Solutions with different thickness of the interface at the same final time. The curve represents the solution obtained by EDE-DeepONet. xxx represents the reference solution.

In this scenario, we treat ϵ as a training parameter, and the problem can be described as follows:

$$u_t = u_{xx} - \frac{1}{\epsilon^2}(u^3 - u),$$
$$u(-1,t) = u(1,t) = 0.$$

The initial states $u(x, t_0; \epsilon)$ are obtained by spectral methods for a few steps with the initial input function $u(x,0) = 0.4\sin(\pi x)$. The training sample is generated based on the numerical solution of $u(x, 0.02; \epsilon)$. We randomly select 50 different values of ϵ from the range $[0.1, 0.2]$. The learning rate is set to $\Delta t = 10^{-4}$, and an adaptive time-stepping strategy is applied. We obtain the predicted solution after 400 iterations for different ϵ. The rest of the settings remain the same as in the previous example.

The solutions obtained for different ϵ values are presented in Fig. 8. As ϵ decreases, the interface becomes sharper. Additionally, it's important to note that the training parameter range is limited to $(0.1, 0.2)$, yet EDE-DeepONet is capable of predicting solutions beyond this range. EDE-DeepONet demonstrates its ability to track the limit of ϵ in a single training process, a task that traditional numerical methods often struggle with.

6.3.2. Two-dimensional case

The two-dimensional case of the Allen-Cahn equation introduces even greater complexity. The problem can be described as follows:

$$u_t = \Delta u - g(u),$$
$$u(x,y,0) = a\sin(\pi x)\sin(\pi y),$$
$$u(-1,y,t) = u(1,y,t) = u(x,-1,t) = u(x,1,t) = 0.$$

The corresponding Ginzburg–Landau free energy $E[u]$ can be defined as:

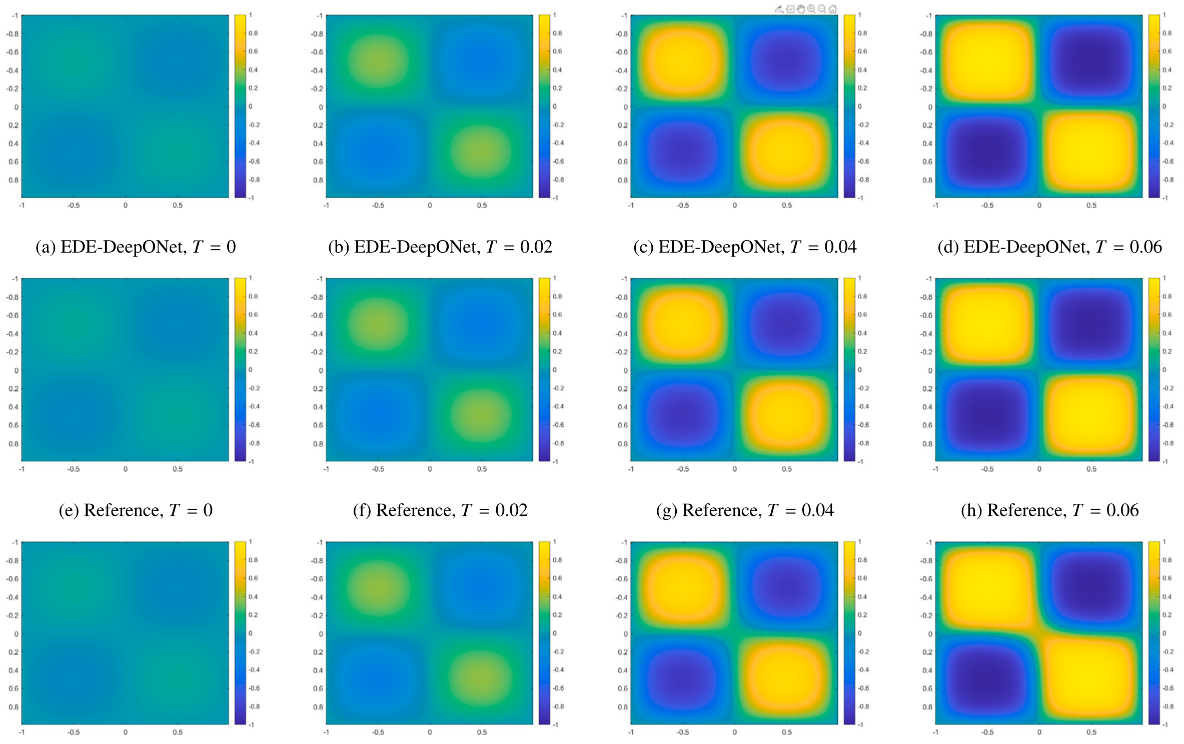
$$E[u] = \int_{-1}^{1}\int_{-1}^{1} \frac{1}{2}(|u_x|^2 + |u_y|^2)dxdy + \int_{-1}^{1}\int_{-1}^{1} G(u)dxdy.$$

Here, $G(u) = \frac{1}{4\epsilon^2}(u^2-1)^2$ and $g(u) = G'(u) = \frac{1}{\epsilon^2}u(u^2-1)$. Typically, ϵ is set to 0.1. During the training process, we use $\Delta t = 1\times 10^{-4}$, employ a grid with 201×201 spatial points, and select 20 training parameters a in the range of $(0.05, 0.35)$. The selection of a and x is performed in the same manner as in example 1. We compare the solutions with $a = 0.08, 0.2, 0.32$ and 0.5, where $a = 0.5$ is out of the training range. The mean squared error is shown in the Table 4.

In Fig. 9, we assess the performance of both EDE-DeepONet and E-DeepONet. The latter is identical to EDE-DeepONet but lacks the support of SAV. The figure provides a comparative visual analysis of the solutions. In the figure, the top panel (Figs. 9a-9d) depicts the solutions produced via the EDE-DeepONet algorithm. The middle panel (Figs. 9e-9h) showcases the canonical reference solutions. The bottom panel (Figs. 9i-9l) offers the solutions produced via the E-DeepONet algorithm. Across all panels, the initial input function is consistently $u_0(x,y) = 0.08\sin(\pi x)\sin(\pi y)$, facilitating a standardized comparison at $T = 0.02$, 0.04, and 0.06.

Table 4
The mean squared error of the two-dimensional Allen-Cahn equation: The initial input function is $u_0(x,y) = a\sin(\pi x)\sin(\pi y)$.

Error	T = 0.02	T = 0.04	T = 0.06
a = 0.08	2.37×10^{-5}	2.46×10^{-4}	1.06×10^{-3}
a = 0.2	5.71×10^{-5}	3.41×10^{-4}	1.34×10^{-3}
a = 0.32	9.57×10^{-5}	3.27×10^{-4}	1.32×10^{-3}
a = 0.5	4.46×10^{-4}	8.62×10^{-4}	1.53×10^{-3}



(a) EDE-DeepONet, $T = 0$ (b) EDE-DeepONet, $T = 0.02$ (c) EDE-DeepONet, $T = 0.04$ (d) EDE-DeepONet, $T = 0.06$

(e) Reference, $T = 0$ (f) Reference, $T = 0.02$ (g) Reference, $T = 0.04$ (h) Reference, $T = 0.06$

(i) E-DeepONet, $T = 0$ (j) E-DeepONet, $T = 0.02$ (k) E-DeepONet, $T = 0.04$ (l) E-DeepONet, $T = 0.06$

Fig. 9. Two-dimensional Allen-Cahn equation: Figs. 9a-9d display the solutions obtained by the EDE-DeepONet. Figs. 9e-9h showcase the reference solutions, while Figs. 9i-9l represent another set of reference solutions. The initial input function is given by $u_0(x,y) = 0.08\sin(\pi x)\sin(\pi y)$. Solutions are compared at $T = 0.02$, 0.04, and 0.06. EDE-DeepONet approximates the reference solution closely, but the E-DeepONet, without the aid of SAV, struggles to approximate the reference solution over extended periods.

As evident from the visual data, the EDE-DeepONet provides a commendable approximation to the reference solution. In contrast, E-DeepONet, when bereft of the support from SAV, visibly lags in its approximation prowess, especially over more extended periods.

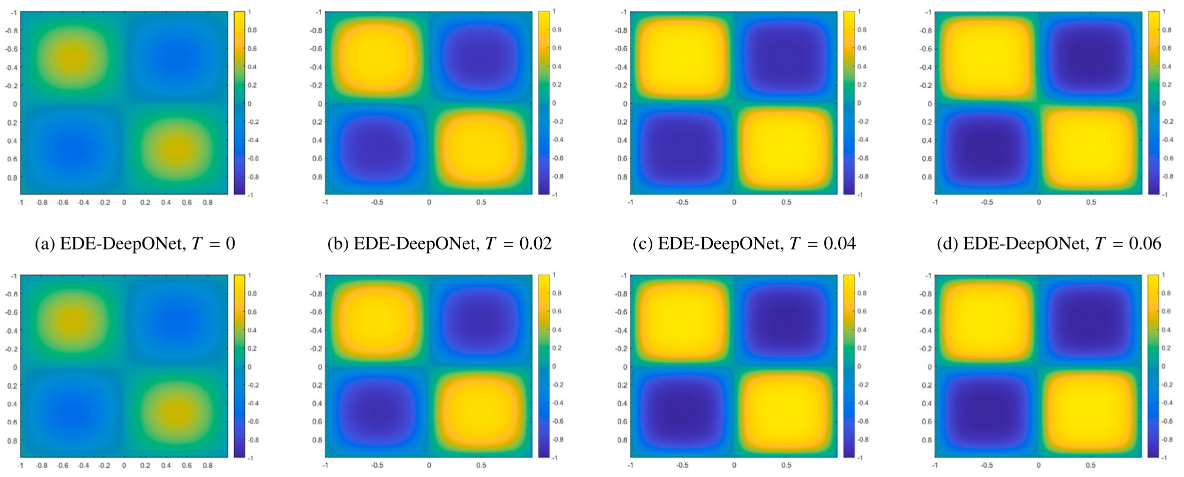
In Fig. 10, we present the solution derived from EDE-DeepONet for $a = 0.5$ alongside its corresponding reference solution. Intriguingly, while $a = 0.5$ falls outside the training domain, EDE-DeepONet retains its adeptness in closely mirroring the exact solution. This performance highlights its strong generalization, making the training process more efficient.

6.4. Example 4: reaction-diffusion equations in three dimensions

Consider a three-dimensional reaction-diffusion system described by:

$$u_t = \Delta u - g(u),$$
$$u(x,y,z,0) = a\sin(\pi x)\sin(\pi y)\sin(\pi z),$$
$$u = 0 \text{ on } \partial X \quad \text{(Dirichlet boundary conditions)},$$

where $X = [-1,1]^3$.



(a) EDE-DeepONet, $T = 0$ (b) EDE-DeepONet, $T = 0.02$ (c) EDE-DeepONet, $T = 0.04$ (d) EDE-DeepONet, $T = 0.06$

(e) Reference, $T = 0$ (f) Reference, $T = 0.02$ (g) Reference, $T = 0.04$ (h) Reference, $T = 0.06$

Fig. 10. Two-dimensional Allen-Cahn equation: Fig. 10a-Fig. 10d is the solution obtained by the EDE-DeepONet. Fig. 10f-Fig. 10h represents the reference solution of the two-dimensional Allen-Cahn equation with initial input function $u_0(x, y) = 0.5\sin(\pi x)\sin(\pi y)$. We compare the solutions at $T = 0.02, 0.04, 0.06$.

Table 5
The mean squared error of the three-dimensional reaction-diffusion equations.

Error	T = 0.02	T = 0.04	T = 0.06
a = 0.4	1.39×10^{-7}	2.13×10^{-7}	2.21×10^{-7}
a = 0.6	3.05×10^{-9}	6.24×10^{-8}	1.47×10^{-7}
a = 0.8	7.10×10^{-9}	1.21×10^{-7}	2.58×10^{-7}

For testing purposes, we adopt an exact solution given by:

$$u(x, y, z, t) = ae^{-10t}\sin(\pi x)\sin(\pi y)\sin(\pi z),$$

and the nonlinear term can be deduced as:

$$g(u) = (10 - 3\pi^2)u.$$

For our simulations, we uniformly select 30 training samples of a in the interval (0.5, 1) and utilize a computational grid comprising $51 \times 51 \times 51$ spatial points. Leveraging the exact solution, we generate an error table for values $a = 0.4, 0.6, 0.8$ as presented in Table 5. The visual representation of the predicted solution and the exact solution with $a = 0.8$ at $T = 0.06$ is depicted in Fig. 11. Furthermore, EDE-DeepONet demonstrates its accuracy and capability in accurately solving PDEs even in complex three-dimensional scenarios. Its efficiency stands out, especially when handling higher-dimensional problems. The visualization of these solutions, as seen in the slices at varying z values, provides insight into the spatial distribution and behavior of the underlying system. The ability to achieve such accuracy in three-dimensional environments underscores the potential of EDE-DeepONet as a powerful tool for solving intricate reaction-diffusion equations.

7. Concluding remarks

In this paper, we present a novel neural network architecture, EDE-DeepONet, designed for solving parametric partial differential equations (PDEs) under varying initial conditions, all while preserving the energy dissipative nature of dynamic systems. Our approach incorporates the energy dissipative law of dynamic systems into the DeepONet framework. We also introduce two critical strategies: an adaptive time-stepping strategy and a restart strategy. Through our experiments, we demonstrate that both strategies contribute significantly to maintaining the modified energy's proximity to the original energy during the evolution of the network.

To mitigate the computational cost associated with training the DeepONet and keep physical properties of the dynamic system, we employ the Scalar Auxiliary Variable (SAV) method to evolve the EDE-DeepONet. This success paves the way for more exploration. For instance, extending our approach to address general Wasserstein gradient flow problems is a promising direction.

While our work currently employs a basic DeepONet architecture, more advanced neural network architectures are compatible with our methodology. These advanced architectures hold the potential to further enhance the accuracy and capabilities of EDE-DeepONet.

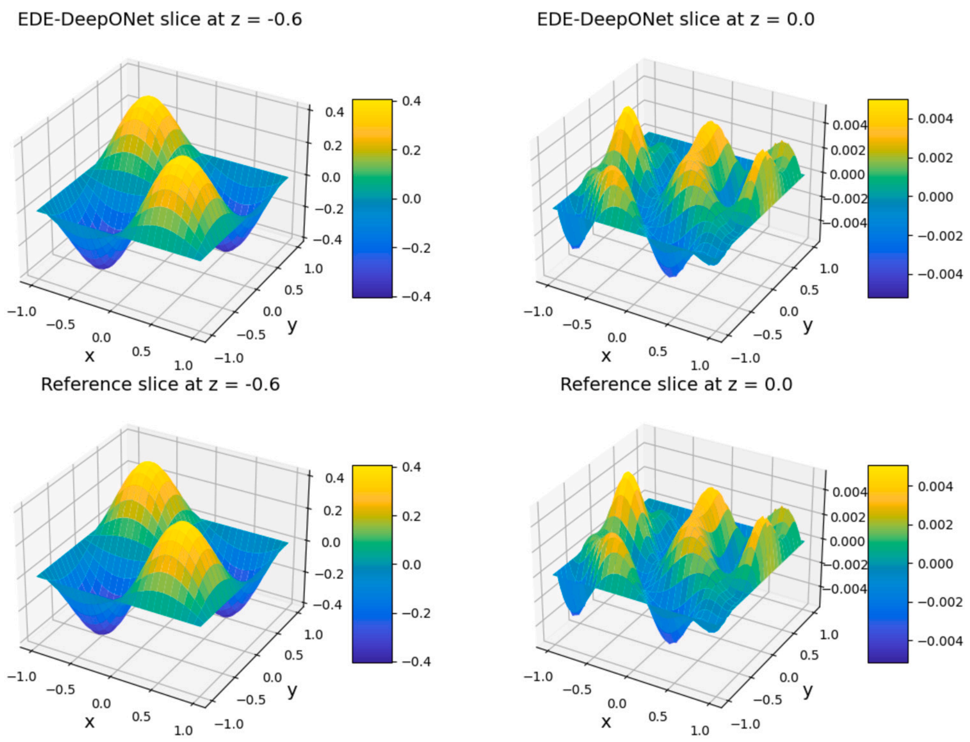


Fig. 11. Three-dimensional Reaction-Diffusion Equations: The figure displays the solution obtained using EDE-DeepONet and the reference solution at $T = 0.06$ with $a = 0.8$. Two cross-sectional slices are depicted at $z = -0.6$ and $z = 0.0$.

CRediT authorship contribution statement

J. Z. and S. Z. conceived the mathematical models, implemented the methods, designed the numerical experiments, interpreted the results, and wrote the paper. G.L. and J. S. edited and reviewed the final manuscript. All the authors gave their final approval for publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgements

SJ and SZ gratefully acknowledge the support of NSF DMS-1720442, NSFC 11971407 and AFOSR FA9550-20-1-0309. GL and ZZ gratefully acknowledge the support of the National Science Foundation (DMS-2053746, DMS-2134209, ECCS-2328241, and OAC-2311848), and U.S. Department of Energy (DOE) Office of Science Advanced Scientific Computing Research program DE-SC0021142, DE-SC0023161, and the Uncertainty Quantification for Multifidelity Operator Learning (MOLUcQ) project (Project No. 81739), and DOE – Fusion Energy Science, under grant number: DE-SC0024583.

References

[1] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: learning maps between function spaces, arXiv preprint, arXiv:2108.08481, 2021.
[2] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: graph kernel network for partial differential equations, arXiv preprint, arXiv:2003.03485, 2020.
[3] Y. Khoo, J. Lu, L. Ying, Solving parametric pde problems with artificial neural networks, Eur. J. Appl. Math. 32 (2021) 421–435.
[4] K. Bhattacharya, B. Hosseini, N.B. Kovachki, A.M. Stuart, Model reduction and neural networks for parametric pdes, arXiv preprint, arXiv:2005.03180, 2020.
[5] N.H. Nelsen, A.M. Stuart, The random feature model for input-output maps between banach spaces, SIAM J. Sci. Comput. 43 (2021) A3212–A3243.

[6] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier neural operator for parametric partial differential equations, arXiv preprint, arXiv:2010.08895, 2020.
[7] R.G. Patel, N.A. Trask, M.A. Wood, E.C. Cyr, A physics-informed operator regression framework for extracting data-driven continuum models, Comput. Methods Appl. Mech. Eng. 373 (2021) 113500.
[8] J.A. Opschoor, C. Schwab, J. Zech, Deep learning in high dimension: Relu network expression rates for bayesian pde inversion, SAM Research Report 2020, 2020.
[9] C. Schwab, J. Zech, Deep learning in high dimension: neural network expression rates for generalized polynomial chaos expansions in uq, Anal. Appl. 17 (2019) 19–55.
[10] T. O'Leary-Roseberry, U. Villa, P. Chen, O. Ghattas, Derivative-informed projected neural networks for high-dimensional parametric maps governed by pdes, Comput. Methods Appl. Mech. Eng. 388 (2022) 114199.
[11] K. Wu, D. Xiu, Data-driven deep learning of partial differential equations in modal space, J. Comput. Phys. 408 (2020) 109307.
[12] L. Lu, P. Jin, G. Pang, Z. Zhang, G.E. Karniadakis, Learning nonlinear operators via deeponet based on the universal approximation theorem of operators, Nat. Mach. Intell. 3 (2021) 218–229.
[13] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[14] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Netw. 4 (1991) 251–257.
[15] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Netw. 2 (1989) 359–366.
[16] T. Chen, H. Chen, Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems, IEEE Trans. Neural Netw. 6 (1995) 911–917.
[17] M. Raissi, P. Perdikaris, G.E. Karniadakis, Multistep neural networks for data-driven discovery of nonlinear dynamical systems, arXiv preprint, arXiv:1801.01236, 2018.
[18] T. Qin, Z. Chen, J.D. Jakeman, D. Xiu, Deep learning of parameterized equations with applications to uncertainty quantification, Int. J. Uncertain. Quantificat. 11 (2021).
[19] N. Winovich, K. Ramani, G. Lin, Convpde-uq: convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains, J. Comput. Phys. 394 (2019) 263–279.
[20] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81.
[21] J. del Águila Ferrandis, M.S. Triantafyllou, C. Chryssostomidis, G.E. Karniadakis, Learning functionals via lstm neural networks for predicting vessel dynamics in extreme sea states, Proc. R. Soc. A 477 (2021) 20190897.
[22] J. Jia, A.R. Benson, Neural jump stochastic differential equations, Adv. Neural Inf. Process. Syst. 32 (2019).
[23] T. Chen, Y. Rubanova, J. Bettencourt, D. Duvenaud, Neural ordinary differential equations, in: Advances in Neural Information Processing Systems, La Jolla, 2018.
[24] Y. Du, T.A. Zaki, Evolutional deep neural network, Phys. Rev. E 104 (2021) 045303.
[25] S.M. Allen, J.W. Cahn, A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening, Acta Metall. 27 (1979) 1085–1095.
[26] D.M. Anderson, G.B. McFadden, A.A. Wheeler, Diffuse-interface methods in fluid mechanics, Annu. Rev. Fluid Mech. 30 (1998) 139–165.
[27] J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. I. Interfacial free energy, J. Chem. Phys. 28 (1958) 258–267.
[28] M. Doi, S.F. Edwards, S.F. Edwards, The Theory of Polymer Dynamics, vol. 73, Oxford University Press, 1988.
[29] K. Elder, M. Katakowski, M. Haataja, M. Grant, Modeling elasticity in crystal growth, Phys. Rev. Lett. 88 (2002) 245701.
[30] M.E. Gurtin, D. Polignone, J. Vinals, Two-phase binary fluids and immiscible fluids described by an order parameter, Math. Models Methods Appl. Sci. 6 (1996) 815–831.
[31] F.M. Leslie, Theory of Flow Phenomena in Liquid Crystals, Advances in Liquid Crystals, vol. 4, Elsevier, 1979, pp. 1–81.
[32] P. Yue, J.J. Feng, C. Liu, J. Shen, A diffuse-interface method for simulating two-phase flows of complex fluids, J. Fluid Mech. 515 (2004) 293–317.
[33] J. Shen, J. Xu, J. Yang, The scalar auxiliary variable (sav) approach for gradient flows, J. Comput. Phys. 353 (2018) 407–416.
[34] G. Caginalp, X. Chen, Convergence of the phase field model to its sharp interface limits, Eur. J. Appl. Math. 9 (1998) 417–445.
[35] X. Chen, G. Caginalp, C. Eck, A rapidly converging phase field model, Discrete Contin. Dyn. Syst. 15 (2006) 1017.