Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 416 (2023) 116299
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Phase-Field DeepONet: Physics-informed deep operator neural network for fast simulations of pattern formation governed by gradient flows of free-energy functionals

Wei Li[a,b], Martin Z. Bazant[b,c], Juner Zhu[a,*]

[a] Department of Mechanical and Industrial Engineering, Northeastern University, United States of America
[b] Department of Chemical Engineering, Massachusetts Institute of Technology, United States of America
[c] Department of Mathematics, Massachusetts Institute of Technology, United States of America

Received 26 February 2023; received in revised form 13 July 2023; accepted 20 July 2023
Available online 11 August 2023

Abstract

Recent advances in scientific machine learning have shed light on the modeling of pattern-forming systems. However, simulations of real patterns still incur significant computational costs, which could be alleviated by leveraging large image datasets. Physics-informed machine learning and operator learning are two new emerging and promising concepts for this application. Here, we propose "Phase-Field DeepONet", a physics-informed operator neural network framework that predicts the dynamic responses of systems governed by gradient flows of free-energy functionals. Examples used to validate the feasibility and accuracy of the method include the Allen–Cahn and Cahn–Hilliard equations, as special cases of reactive phase-field models for nonequilibrium thermodynamics of chemical mixtures. This is achieved by incorporating the minimizing movement scheme into the framework, which optimizes and controls how the total free energy of a system evolves, instead of solving the governing equations directly. The trained operator neural networks can work as explicit time-steppers that take the current state as the input and output the next state. This could potentially facilitate fast real-time predictions of pattern-forming dynamical systems, such as phase-separating Li-ion batteries, emulsions, colloidal displays, or biological patterns.

Keywords: Physics-informed machine learning; Deep operator neural network; Phase-field method; Minimizing movement scheme; Allen–Cahn and Cahn–Hilliard equations

1. Introduction

Machine learning (ML) has achieved enormous success in many disciplines, especially computer vision [1] and natural language processing [2]. However, when it comes to scientific problems, ML algorithms are often thought of as black boxes that can hardly be interpreted and lack rigorous justifications by physical laws. Recently, a concept, scientific machine learning (SciML), emerged and has quickly attracted wide attention [3]. Its aim is to make the traditional ML domain-aware, interpretable, and robust. In some sense, SciML is now revolutionizing the area of computational science and has been applied in various scientific disciplines, such as ML-enhanced multiphysics and

* Corresponding author.
E-mail addresses: we.li@northeastern.edu (W. Li), bazant@mit.edu (M.Z. Bazant), j.zhu@northeastern.edu (J. Zhu).

https://doi.org/10.1016/j.cma.2023.116299
0045-7825/

multiscale modeling [4–6], ML-assisted fast online prediction and guided data acquisition [7], and optimal decisions for complex systems [7,8]. Among various SciML architectures, physics-informed machine learning (PIML) and operator learning are the two representative examples. PIML introduces known physics into an ML algorithm and as a result, requires a smaller dataset or sometimes even works without experimental data [9]. PIML is a general concept and can be implemented in various strategies [10]. Considering the three ingredients of a general ML algorithm, physical laws can be accordingly incorporated into (1) the training data, e.g., data generated from first-principle-based simulations, (2) the ML models, e.g., neural networks designed to reflect certain physical principles (symmetry, positive definiteness, hierarchical relations, etc.), and (3) the training strategies, e.g., loss functions formulated to include physical laws. Among these different approaches, one of the most extensively studied is the physics-informed neural networks (PINNs) [11–13], where the known physics, namely, the governing equations, initial conditions (ICs), and boundary conditions (BCs), are incorporated into the loss function in the form of residuals. It has also been demonstrated that PINNs are able to deal with both forward (solving equations) and inverse (identifying parameters) problems for fluid dynamics governed by Navier–Stokes equations [11]. Since proposed, PINNs have been widely adopted in various applications. Interested readers can refer to [9] for a comprehensive review. In addition to these successful applications, PINNs have also been extended to accommodate irregular [14] and multiple domains [15,16], enforce hard constraints with modified NNs [17], incorporate adaptive activation [18], introduce gradient-enhanced [19] or energy-based terms into loss function [20,21], to name a few.

Despite the initial successes, a prominent challenge of PINNs is to efficiently determine or optimize the hyperparameters in the loss function. Most existing studies applied a trial-and-error scheme which makes the training time-consuming. To tackle this challenge, Psaros et al. [22] recently proposed a meta-learning framework to optimize the loss function offline. Meanwhile, it is also possible to avoid this issue by reducing the total number of loss terms. One common way is to enforce the hard constraints [17], which entails modifying the neural networks such that the outputs always satisfy certain BCs and ICs. Another approach is to make use of the energy or variational principles that intrinsically include the PDEs and BCs. For example, solving the Laplacian equation $\nabla^2 u = 0$ with a Neumann-type BC $\partial_{\boldsymbol{n}} u = 0$ using PINN needs at least two loss terms. Alternatively, this problem is equivalent to finding the minimum of the functional $\mathcal{J} = \int_\Omega 0.5(\nabla u)^2 \,\mathrm{d}V$ (i.e. $\delta\mathcal{J} = 0$), which can be treated as the only loss term. In this way, the number of loss terms can be reduced. Mathematically, this example is a special case of the more general Euler–Lagrangian equation (see Appendix A.1), and note that the order of derivatives in the functional is lower than that in the PDEs, which further improves the training efficiency.

A few existing studies have explored the strategy of using energy as the loss function. E et al. [23] proposed a DeepRiz neural network to solve the Poisson's equation ($-\nabla^2 u(x) = f(x)$) with homogeneous essential BC ($u(x) = 0, x \in \partial\Omega$) by minimizing the functional $\mathcal{J} = \int_\Omega \left[0.5 \cdot (\nabla u)^2 - f(x) \cdot u(x)\right] \mathrm{d}x$, which is also a special case of the Euler–Lagrangian equation. Later, Wang et al. extended this framework to consider inhomogeneous essential BCs on complex boundary geometries and multiple domains [15]. Another case explored is the principle of minimal potential energy in solid mechanics, which states that the deformation of a solid domain will follow the path that minimizes the total potential energy under external loads [20,21]. For quasi-static elastic responses, the total potential energy $\mathcal{T}$ consists of the elastic strain energy $U = \int_\Omega 0.5\boldsymbol{\sigma} : \boldsymbol{\varepsilon} \,\mathrm{d}V$ and the work potential $W = -\int_{\partial\Omega} f \cdot u \,\mathrm{d}S$. Minimizing $\mathcal{T} = U + W$ is equivalent to solving the corresponding Euler–Lagrangian equation (force equilibrium) $\nabla \cdot \boldsymbol{\sigma} = 0$ with BC $\boldsymbol{\sigma} \cdot \boldsymbol{n} = 0$. In the authors' previous work, we implemented this principle by introducing the potential energy into the loss function and predicted the deformation of elastic plates [20]. We also compared this energy-based framework with the vanilla residual-based PINN and found that the energy-based one is more efficient in terms of training time due to the lower order derivatives and fewer hyperparameters in the loss function, though the accuracy of both is comparable. It should be mentioned that the residual-based PINN is a more universal framework, while the energy-based one is only limited to systems governed by energy or variational principles. To the authors' best knowledge, the above existing studies only explored simple linear systems under quasi-static conditions. In this study, we aim to look into the dynamics of highly nonlinear and coupled energy storage systems and make use of the variational principles to construct a PIML framework.

Operator learning is another concept that has emerged as a promising SciML technique; it learns the mapping from one function to another, such as the sequence-to-sequence and image-to-image mappings. Many widely-used network architectures, such as fully-connected neural networks (FNNs) and convolutional neural networks (CNNs), are finite-dimensional operators that map one discretized signal or image to another. Recently, some novel architectures have been proposed to learn the infinite-dimensional mappings, such as Deep Operator Networks

(DeepONets) [24] and Fourier Operator Networks (FNOs) [25]. A comprehensive review of the operator learning with neural networks could be found in [25]. In this study, we will focus on the DeepONet approach developed by Lu et al. [24]. So far, DeepONets have been proven to have better approximation and generalization abilities than FNNs and, therefore, has been used by many applications. One of the advantages of DeepONets is their ability to take the BCs or ICs as inputs, making it theoretically possible to train one network for all scenarios. This means that once the network is trained, it can be used to solve new problems with different boundary and/or initial conditions without additional training. This can be particularly useful in applications where the boundary and/or initial conditions may vary or come with significant uncertainty.

These new advances in SciML have shed light on the modeling of energy-storage systems (ESSs), especially Li-ion batteries. Due to the high-dimensional (e.g., multiple materials, scales, and physical fields) nature of this type of systems, it is cumbersome to develop a complete physics-based model or fully interpret a big dataset. Developing a unified physics-informed machine learning computational framework that can combine the partially-known physics and a small-size dataset is very appealing and necessary. The fundamental challenge here is the trade-off between the abundance of data and the adequacy of physical laws. At the electrode or cell level, experimental data is relatively easy to be obtained but physics is mostly hidden behind the data. Therefore, purely data-driven machine learning algorithms can be applied to predict the performance [7] and lifetime [26] of batteries. When data is expensive or limited, for example, battery degradation data during thousands of cycles taking years to complete, some studies proposed frameworks based on PINNs to estimate the states of battery cells [27,28], identify battery parameters [29], predict the lifetime [30], and recognize degradation patterns [31–33] at the electrode and cell levels. At the active particle level, experimental data is expensive and difficult to collect although many fundamental electro-chemo-mechanical physical theories have been developed at the micro-scales [34,35]. Physics-based models is often used to describe the single-particle pattern formation and extrapolate to porous electrodes [36–38], but it is often very time-consuming to solve the models. As explained previously, PIML has a potential advantage to produce efficient surrogates or reduced-order models due to the fast inference speed of machine learning algorithms after training. For example, several studies used PINNs to solve the two equations for the phase-field method, namely Allen–Cahn and Cahn–Hilliard [39–41], which will be elaborated on in Section 2. Another reason for applying PIML in ESSs is that the determination of constitutive relations and material constants is challenging. Many advanced algorithms have been developed for the interpretation of large datasets of full-field image data, in the context of phase-field models for electrochemical nonequilibrium thermodynamics [36]. For example, Zhao et al. used PDE-constrained optimization to learn the physics of driven phase separation in lithium iron phosphate nanoparticles from operando images of scanning tunneling X-ray microscopy [42]. Deng et al. used similar methods to learn the constitutive law of the eigen strain change with respect to lithium intercalation from micro X-ray tomography and diffraction images of active particles [43]. This optimization process is often time-consuming and PIML has the potential to achieve faster identification.

Nowadays, the greater scientific community has recognized the value of integrating physics and data into one unified framework as a high-level vision. The energy storage community is one of the pioneering areas. For example, the U.S. Department of Energy (DOE) is the first federal agency to propose the concept of SciML [3]. The current remaining challenge is to find realistic ways to implement them. It is always crucial to first understand the physics in order to construct a proper machine learning architecture for the studied system. The above-mentioned Allen–Cahn and Cahn–Hilliard equations are essential in chemical system modeling. They are able to describe the dynamics of non-conserved and conserved order parameters, respectively, in terms of variationally defined chemical potentials. Both equations can be derived through variational methods that have been well established to obtain the governing equations of complex coupled nonlinear systems [44]. More specifically, Allen–Cahn and Cahn–Hilliard equations are two special cases of gradient flows that entail finding and constructing an appropriate free energy and an inner product to incorporate the kinetics into a variational framework [44–46]. Gradient flows can be applied to a large variety of physics including diffusion, phase separation, microstructure evolution, etc. Therefore, constructing a machine learning framework for gradient flows can be beneficial to a wide range of applications. As we mentioned earlier, ML can be adopted naturally to solve variational problems, where we can approximate the solutions with ML models by minimizing the free energy functional as a loss function.

In this study, we propose the idea of “Phase-Field DeepONet” as a general neural network framework for dynamical systems governed by gradient flows of free energy functionals, taking advantage of the energy-based loss function, deep operator network, and physics-informed learning. The paper is organized as follows: Section 2

presents the theory of phase-field method and gradient flows; Section 3 describes the framework of Phase-Field DeepONet that incorporates the minimizing movement scheme into a physics-informed deep operator neural network; In Section 4, we investigate three different dynamical systems including the linear relaxation kinetics, Allen–Cahn, and Cahn–Hilliard dynamics to validate the proposed framework. The computational codeof this study is accessible online: https://github.com/weili101/Phase-Field_DeepONet

2. Phase-field method and gradient flows

2.1. Phase-field method

Phase-field methods are widely used in materials science because of their capability to track microstructure evolution, grain growth and coarsening, crack propagation [47,48], to name a few. Unlike other sharp interface models, phase-field models treat interfaces in a diffusive way with phase-field variables, which can then describe the domain and all interfaces continuously as a whole. There are two types of phase-field variables, namely conserved and non-conserved fields. The evolution (or dynamics) of both are governed by the total free energy $\mathcal{F}$ of a system and its variational derivatives with respect to the field variables, which can be viewed as diffusional chemical potentials.

For a single field variable ϕ, the standard free energy functional for an inhomogeneous system, proposed by Van der Waals [49] and Cahn and Hilliard [50], is defined as,

$$\mathcal{F} = \int_{\Omega} F(\phi) = \int_{\Omega} [f(\phi) + \frac{1}{2}\kappa_{\phi}(\nabla\phi)^2]\,\mathrm{d}x, \tag{1}$$

where $F(\phi)$ denotes the total free-energy density; $f(\phi)$ is the homogeneous free-energy density and the second term on the right-hand side represents the gradient energy at phase interfaces with the gradient coefficient κ_{ϕ}. Generally, the field variables evolve in the direction where the free energy continuously decreases. For a conserved field variable, the dynamics can be expressed as a conservation law for gradient-driven fluxes. The Cahn–Hilliard equation can be then obtained,

$$\frac{\partial\phi}{\partial t} = \nabla \cdot M\nabla\frac{\delta\mathcal{F}}{\delta\phi}, \tag{2}$$

where M is a transport coefficient (the product of the mobility and the concentration field variable [36]) and the functional derivative (diffusional chemical potential) is given by,

$$\frac{\delta\mathcal{F}}{\delta\phi} = \frac{\partial f}{\partial\phi} - \kappa_{\phi}\nabla\cdot\nabla\phi. \tag{3}$$

For a non-conserved field variable, we have the Allen–Cahn equation,

$$\frac{\partial\phi}{\partial t} = -M\frac{\delta\mathcal{F}}{\delta\phi}, \tag{4}$$

where the functional derivative is also given by Eq. (3). The Allen–Cahn equation can be viewed as a linearized model of a reaction producing the field variable, proportional to the affinity, or difference in diffusional chemical potential with respect to an external reservoir [36,51].

2.2. Mathematics of gradient flows

One common way to establish the theories or governing equations of a system is to start with constitutive relations based on experimental data. These theories need then to be checked for consistency with thermodynamic laws. The variational methods, on the other hand, start with thermodynamics so that the derived theories are always consistent with the thermodynamic laws. In addition, the variational methods can handle extreme anisotropy, non-differentiability, and nonlinearity more easily. In this section, we will review a general mathematical framework to derive phase-field models as gradient flows of free energy functionals.

The second law of thermodynamics states that the total free energy of a system always decreases. Therefore, the equations governing the evolution of field variables in a system should be constructed in an appropriate way

to guarantee a monotonic decrease in total free energy. One approach is the gradient flows which can be described by,

$$\frac{\mathrm{d}}{\mathrm{d}t}u(x,t) = -\nabla\mathcal{F}(u), \tag{5}$$

where u is a field variable that evolves with time (depending on space and time); $\mathcal{F}$ is a smooth and convex energy functional of u. $\nabla\mathcal{F}(u)$ indicates the functional gradient of $\mathcal{F}$ with respect to u. Physically, this equation states that the field variable u evolves in the direction where the free energy decreases fastest. The functional gradient is the driving force of the dynamic process. Mathematically, analogous to the directional derivative of a multi-variable function, the functional gradient $\nabla\mathcal{F}$ and functional derivative $\delta\mathcal{F}/\delta u$ are related by the inner product $\langle ., . \rangle$,

$$\langle \nabla\mathcal{F}, v\rangle = \frac{\mathrm{d}\mathcal{F}(u + v\cdot t)}{\mathrm{d}t}\,|_{t=0} = \int_\Omega \frac{\delta\mathcal{F}}{\delta u} v\,\mathrm{d}x, \tag{6}$$

where v is an arbitrary function and can be viewed as a flow field; t is a short time and $v \cdot t$ is also called the variation of u. Therefore, the functional gradient depends on not only the free energy functional (its functional derivative) but also the construction of the inner product (the measure of distance).

For a functional energy defined as $\mathcal{F} = \int_\Omega F(x, u(x), \nabla u(x))\,\mathrm{d}x$, the functional derivative can be determined by (see Appendix A.2)

$$\frac{\delta\mathcal{F}}{\delta u} = \frac{\partial F}{\partial u} - \nabla\cdot\frac{\partial F}{\partial \nabla u}. \tag{7}$$

In order to get the functional gradient, we still need to construct an inner product. In this study, we introduce two different inner products. The first one is the weighted L^2 inner product,

$$\langle f, g\rangle_{L^2,M} = \int_\Omega \frac{f(x)\cdot g(x)}{M}\,\mathrm{d}x, \tag{8}$$

where M is the weight; $f(x)$ and $g(x)$ are two arbitrary functions. The second one is the H^{-1} inner product with a weight, defined as

$$\langle f, g\rangle_{H^{-1},M} = \int_\Omega \nabla\phi_f \cdot M\nabla\phi_g\,\mathrm{d}x, \tag{9}$$

where ϕ_f is the solution of the Poisson's equation with Neumann boundary condition (see Appendix A.3).

We can then obtain the functional gradient. For L^2 inner product, with Eqs. (6) and (8) we have

$$\langle \nabla\mathcal{F}, v\rangle_{L^2,M} = \int_\Omega \frac{\nabla\mathcal{F} v}{M}\,\mathrm{d}x = \int_\Omega \frac{\delta\mathcal{F}}{\delta u} v\,\mathrm{d}x. \tag{10}$$

Since v is an arbitrary function, to ensure the equality is always satisfied, we have,

$$\nabla\mathcal{F} = M\frac{\delta\mathcal{F}}{\delta u}. \tag{11}$$

Similarly, with H^{-1} inner product (Eqs. (6) and (9)) we have,

$$\nabla\mathcal{F} = -\nabla\cdot M\nabla\frac{\delta\mathcal{F}}{\delta u}. \tag{12}$$

Substituting the above two functional gradients (Eqs. (11) and (12)) into Eq. (5), we get the Allen–Cahn and Cahn–Hilliard equations, respectively.

2.3. The minimizing movement scheme

We have demonstrated a mathematical way to derive the phase-field equations (i.e., Allen–Cahn and Cahn–Hilliard equations) with the gradient flows theory. In order to predict the dynamic response of a system governed by gradient flows, we can directly solve these equations with the finite element (FE) or finite difference (FD) method. Alternatively, we can make use of an important feature of gradient flows: given a fixed small time step $\tau > 0$ and

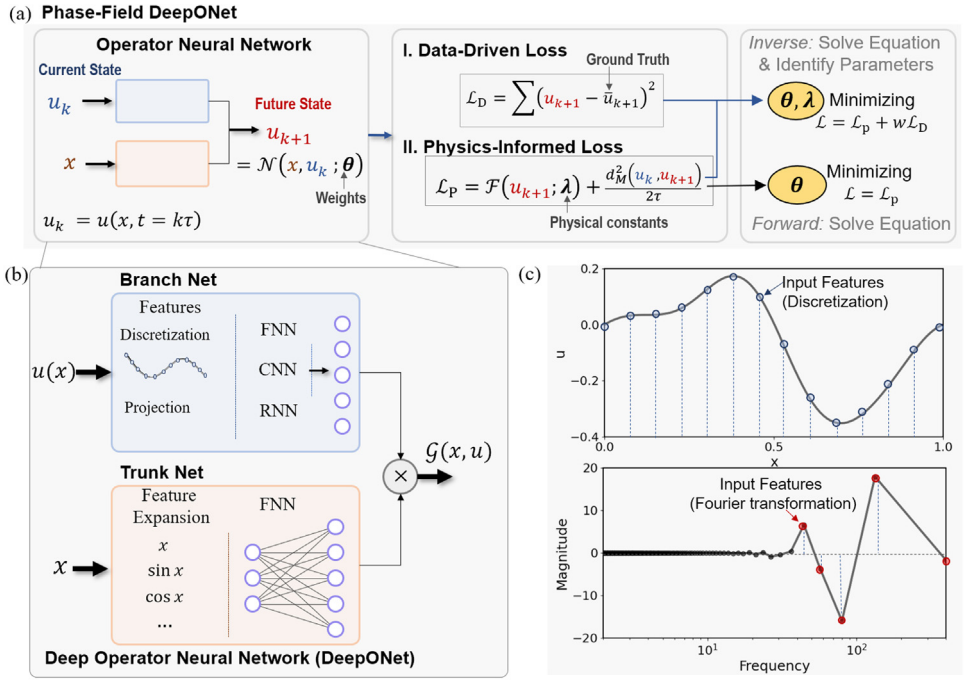

Fig. 1. Schematic illustration of (a) DeepONet and (b) Physics-informed DeepONet with energy-based loss function.

a smooth and convex functional $\mathcal{F}(u)$, we can find a time sequence of u, $[u_1, u_2, \ldots, u_n]$ from the initial condition u_0, through the following iterated scheme, which is called minimizing movement scheme,

$$u_{k+1} \in \underset{u}{\operatorname{argmin}} \left[\mathcal{F}(u) + \frac{d^2(u, u_k)}{2\tau} \right], \tag{13}$$

where $u_k = u(x, t = k\tau)$ and $u_{k+1} = u(x, (k+1)\tau)$ represent the field distributions at time step k and $k+1$; $d(.,.)$ indicates the distance between two functions. The time sequence obtained from this iterative minimization scheme can be proven to be the solution of the corresponding PDEs. For the above minimization problem, we know u_{k+1} is the solution of

$$\frac{\partial}{\partial u} \left[\mathcal{F}(u) + \frac{d(u, u_k)}{2\tau} \right] = \nabla \mathcal{F}(u) + \frac{d(u, u_k)}{\tau} = 0, \tag{14}$$

which gives

$$\frac{d(u, u_k)}{\tau} = -\nabla \mathcal{F}(u). \tag{15}$$

This equation is the discrete-time implicit Euler scheme for Eq. (5). Therefore, we can get the solution by the minimizing movement scheme instead of solving the PDEs directly.

3. Phase-field DeepONet: physics-informed deep operator neural network for gradient flows

In this section, we present a novel framework called "Phase-Field DeepONet", a physics-informed deep operator neural network framework incorporating the aforementioned minimizing movement scheme to solve the gradient flows of free energy functionals. The main structure of the proposed framework is depicted in Fig. 1a, which consists of two key components. The first one is an operator neural network that describes the temporal evolution of a field

variable. This network is responsible for learning the mapping between the distribution of the field variable at the current time step and its distribution at the subsequent time step. After training, it functions as an explicit "time-stepper", similar to a recurrent neural network (RNN), and can be used to predict the entire temporal evolution of the field variable. The second component is a physics-informed loss function that incorporates the aforementioned minimizing movement scheme (Eq. (13)). This loss function acts as a guiding principle, ensuring that the network adheres to the physics underlying the gradient flows. In the following sections, we will provide further elaboration on these two key components,

3.1. Deep operator neural network (DeepONet)

We first give a general introduction to DeepONet and explain later how to implement it in the current study. DeepONet was first proposed by Lu et al. [13]. The key idea behind it is to treat the neural network as an operator that approximates the unknown mapping between functions. DeepONet is a high-level network structure with two sub-networks, the "branch" network and the "trunk" network, as shown in Fig. 1b. The trunk network takes the coordinates x as the input, while the branch network takes the function u as the input. The final output is given by multiplying the outputs of both networks,

$$\mathcal{G}(x,u) = \sum_{m=1}^{p} b_m(u)t_m(x) + b_0, \tag{16}$$

where b_m and $t_m (m = 1, 2, \ldots, p)$ are the outputs of the branch network and trunk network, respectively; p is the number of outputs of both sub-networks; b_0 is a bias.

DeepONet has several noteworthy features. First, the trunk network typically takes the spatial coordinates as inputs. This enables continuous predictions at any location within the domain and evaluation of gradients (outputs with respect to spatial coordinates) by automatic differentiation, which is crucial to constructing the physics-informed loss function. Second, DeepONet exhibits high flexibility. The essence of this structure is to separate the vector input and function input into two sub-networks, making it easily adaptable to diverse applications. In the trunk network, we can perform feature expansion on the spatial coordinates inputs (e.g., $x \to (x, \sin x, \cos x)$). In the branch network, instead of utilizing a vector of discretized points of a function, it is possible to extract alternative features from the function, such as magnitude and phase in the frequency domain after Fourier-related transforms (see Fig. 1c). Additionally, the sub-networks can be any type of neural network such as the FNN, CNN, RNN, etc.

3.2. DeepONet as a time-stepper

We then implement DeepONet specifically for solving the gradient flows of free energy functionals. In real applications, the input of the branch net can be any function such as the initial or boundary condition and the output can be the solution at a random time. Here, we aim to capture the temporal evolution of the field variable and enable predictions of its future behavior based on its present state. To achieve this, we propose to take the current distribution of a field variable as the input and the distribution at the subsequent time step as the output (i.e. a mapping $u_k \to u_{k+1}$), which can be expressed by,

$$u_{k+1} = \mathcal{N}(x, \boldsymbol{u}_k; \theta) = \sum_{m=1}^{p} b_m(\boldsymbol{u}_k)t_m(x) + b_0, \tag{17}$$

where $\mathcal{N}$ denotes the whole network structure; θ represents the all the parameters (biases and weights) of the network to be trained. The boldface $\boldsymbol{u}_k \in R^D$ denotes the input vector of the branch net with D features (elements) extracted from u_k by processes such as discretization or Fourier transformation. The proposed network structure works like an explicit time-stepper. Given the input at the k^{th} time step u_k, we can get the output at the $(k+1)^{\text{th}}$ time step u_{k+1}, which can then be treated as the input to get u^{τ}_{k+2}. Following this iterative process, a sequence of distributions $[u_k, u_{k+1}, u_{k+2}, \ldots]$ at all following time steps can be obtained.

The input of the whole network will be $\{x_i, \boldsymbol{u}^j_k\}_{(i,j)}$, where $\{x_i\}^{N_t}_{i=1}$ represents the sampled inputs for the trunk net and $\{\boldsymbol{u}^j_k\}^{N_b}_{j=1}$ represents the sampled input features from a continuous function u^j_k for the branch net. We use boldface $\boldsymbol{u}^j_k$ and normal u^j_k to denote a discretized vector and a continuous function, respectively. Superscript j

indicates the j^{th} sample. Any random combinations of the elements in these two sets can generate the input set for the whole network. In this work, we assume a Cartesian product of the two sets to get the inputs for the whole network, i.e., $\{x_i, \boldsymbol{u}_k^j\}_{(i,j)} = \{x_i\}_{i=1}^{N_t} \times \{\boldsymbol{u}_k^j\}_{j=1}^{N_b}$. The total number of input samples is therefore $N = N_t \cdot N_b$.

3.3. Free energy as loss function

After constructing the neural network, the next step is to design an appropriate loss function. While traditional machine learning algorithms often rely on large datasets and construct loss functions based on error metrics like mean square error or mean absolute error, physics-informed machine learning offers a unique advantage. It can effectively work with small datasets or even without any data at all. This is achieved by incorporating physical laws, such as partial differential equations (PDEs), into the loss function. Physics-informed machine learning can address two common types of problems: the inverse problem and the forward problem. In the inverse problem, the physics is partially known with a small dataset available, and the aim is to learn the unknown physics; On the contrary, the forward problem assumes complete knowledge of the underlying physics, without any available data, and the goal is to solve the governing equations.

The proposed framework (Fig. 1) presents a general solution to both the forward and inverse problems by incorporating two types of loss functions. The first one, denoted as $\mathcal{L}_D$, is data-driven and requires experiment or simulation data. It is defined as the mean square error between the neural network predictions and the ground truth data,

$$\mathcal{L}_D = \frac{1}{N_b \cdot N_t} \sum_{j=1}^{N_b} \sum_{i=1}^{N_t} \left[u_{k+1}(x_i, \boldsymbol{u}_k^j) - \bar{u}_{k+1}(x_i, \boldsymbol{u}_k^j) \right]^2 . \tag{18}$$

where u_{k+1} and $\bar{u}_{k+1}$ are the neural network prediction and the corresponding ground truth, respectively. The second loss function is a physics-informed loss function that incorporates the minimizing movement scheme (Eq. (13)), which is based on the free energy functionals of gradient flows. It can be written as,

$$\mathcal{L}_{\text{P}} \approx \frac{A^2}{N_b \cdot N_t} \sum_{j=1}^{N_b} \sum_{i=1}^{N_t} \left[F(u_{k+1}(x_i, \boldsymbol{u}_k^j)) + \frac{d^2(u_{k+1}(x_i, \boldsymbol{u}_k^j), u_k(x_i))}{2\tau} \right], \tag{19}$$

where we use the Monte Carlo method to evaluate the integration and A is the area of the integration domain. Note that experiment or simulation data is not needed in this physics-informed loss function.

The training process will minimize the total loss $\mathcal{L}$ to update and optimize the weights and biases of the neural network. Depending on specific problems, we have different strategies to construct the total loss (see Fig. 1a). In the forward problem scenario, we select the physics-informed loss as the total loss ($\mathcal{L} = \mathcal{L}_P$). By training the neural network with this loss function, we effectively implement the minimizing movement scheme, enabling an approximation of the ground truth. Since no data is required, we are essentially solving the governing equations of the gradient flows. In contrast, the inverse problem requires a total loss that combines both the data-driven and physics-informed losses. The total loss is then constructed as $\mathcal{L} = \mathcal{L}_P + w\mathcal{L}_D$, where w is a weight used to balance the contributions of the two loss terms. By training with this total loss, we optimize the neural network to approximate the ground truth while simultaneously identifying unknown physical constants. For instance, if the free energy $\mathcal{F}(u_{k+1}; \lambda)$ relies on certain unknown physical constants λ, the network will learn to estimate those constants in addition to predicting the field variable distribution. Lastly, we can choose the data-driven loss as the total loss ($\mathcal{L} = \mathcal{L}_D$) if a large dataset is available. The trained network can also predict the field variable distribution and work as a surrogate model. We summarize the framework algorithmically in Algorithm 1.

It is worth noting that a PINN can be regarded as a special example of a physics-informed DeepONet, wherein the branch net is omitted or the input for the branch network is held constant as the initial condition. For example, if we only keep the trunk net in Fig. 1a and replace u_k in Eq. (19) with u_0, the resulting framework is a PINN to solve for the distribution of u in the next step.

4. Numerical examples

In this study, our focus will be on the forward problem, specifically solving governing equations of gradient flows using the proposed Phase-Field DeepONet framework. To illustrate its application and efficacy, three different examples will be explored in this section.

Algorithm 1 General Framework for Solving Gradient Flows with Phase-Field DeepONet

1. Neural Network Representation

(a) Discretize field $u(x,t)$ in time domain with a time interval τ:
$u_k = u(x, t = k\tau), k = 0, 1, 2, ...$

(b) Approximate u_{k+1} (subsequent time step) based on u_k (current time step) with am operator neural network:
$u_{k+1} = \mathcal{N}(x, \boldsymbol{u}_k; \theta)$.
x is the input spatial coordinate of the trunk net;
$\boldsymbol{u}_k$ denotes the input of the branch net (Vector or Matrix);
θ represents the weights and biases to be trained

2. Loss Function Construction

I. Data-driven loss function

$$\mathcal{L}_D = \frac{1}{N_b \cdot N_t} \sum_{j=1}^{N_b} \sum_{i=1}^{N_t} \left[u_{k+1}(x_i, \boldsymbol{u}_k^j) - \bar{u}_{k+1}(x_i, \boldsymbol{u}_k^j) \right]^2$$

II. Physics-informed loss function

$$\mathcal{L}_{\mathrm{P}} \approx \frac{A^2}{N_b \cdot N_t} \sum_{j=1}^{N_b} \sum_{i=1}^{N_t} \left[F(u_{k+1}(x_i, \boldsymbol{u}_k^j)) + \frac{d_{L^2}^2(u_{k+1}(x_i, \boldsymbol{u}_k^j), u_k(x_i))}{2\tau} \right]$$

N_b - number of sampled input vector $\boldsymbol{u}_k$
N_t - number of sampled input location x_i

3. Training Process

I. Forward Problem

(a) Training data: $\{x_i, \boldsymbol{u}_k^j\}_{(i,j)}$
x_i is sampled uniformly in the space domain;
$\boldsymbol{u}_k^j$ is sampled by Gaussian random process.

(b) Minimizing $\mathcal{L} = \mathcal{L}_P$ to train network

II. Inverse Problem

(a) Training data: $\{x_i, \boldsymbol{u}_k^j, \bar{\boldsymbol{u}}_{k+1}(\boldsymbol{x_i}, \boldsymbol{u}_k^j)\}_{(i,j)}$
$\bar{u}_{k+1}(x_i, \boldsymbol{u}_k^j)$ - output (data) at given inputs

(b) Minimizing $\mathcal{L} = \mathcal{L}_P + w\mathcal{L}_D$ to train network and optimize unknown physical constants

4.1. L^2 Gradient flow: relaxation kinetics

We start with the relaxation kinetics in 1D space governed by gradient flows. The governing free energy is

$$\mathcal{F}(u) = \int_\Omega \frac{1}{2} k \cdot u^2(x,t)\, \mathrm{d}x. \tag{20}$$

We consider a 1D domain within $[-1, 1]$. With the L^2 inner product, the corresponding PDE and boundary conditions can be derived with Eqs. (5), (7) and (11) ($M = 1$),

$$\begin{cases} \dfrac{\partial u}{\partial t} = -ku, \\ u(x = \pm 1) = 0. \end{cases} \tag{21}$$

This equation describes the relaxation phenomenon where the initial field will eventually deteriorate to zero ($u = 0$ at equilibrium state). Given an initial condition $u_0 = u(x, t = 0)$, the analytical solution can be found as $u = -e^{kt}u_0$. The constant k is set to 1 for the remaining discussion. In this example, we will compare the two different frameworks, the traditional PINN and the proposed Phase-Field DeepONet. The aim is to illustrate the feasibility of incorporating the minimizing movement scheme into a machine learning framework and highlight the advantages of the proposed framework.

Solving equations with PINN requires the initial condition (and boundary condition) to be given, which, in this case, is set as $u_0 = \sin(\pi x)$. Following the minimizing movement scheme, a straightforward approach is to discretize the time domain into many time steps with a fixed time interval τ. As shown in Fig. 2a, for each time step k, we construct one corresponding neural network $\mathcal{N}_k(x; \theta)$, where x denotes the input spatial coordinate and

θ represents the weights and biases to be optimized. The output of the sub-network is the current distribution of the field variable $u_k = u(x, t = k\tau)$. The sum of free energy and distance as in Eq. (13) is directly treated as the loss function, which can be written as

$$\begin{aligned}\mathcal{L} &= \mathcal{F}(u_k) + \frac{d_{L^2}^2\left(u_k, u_{k-1}\right)}{2\tau} \\ &= \int_{-1}^{1} u_k^2 \,\mathrm{d}x + \int_{-1}^{1} \frac{(u_k - u_{k-1})^2}{2\tau} \,\mathrm{d}x \\ &\approx \frac{2}{N}\sum_{i=1}^{N} u_k(x_i)^2 + \frac{2}{N}\sum_{i=1}^{N} \frac{[u_k(x_i) - u_{k-1}(x_i)]^2}{2\tau},\end{aligned} \tag{22}$$

where $u_k(x_i) = u(x = x_i, t = k\tau)$ represents the field value at locations x_i and N is the total number of training samples. The integration of the free energy term and distance term is estimated by Monte Carlo method.

The training process follows the minimizing movement scheme: (a) we first train the 1st sub-network $\mathcal{N}_1$ given the initial condition u_0; (b) after training, we can get the output u_1. We then train the 2nd sub-network $\mathcal{N}_2$ with u_1 as the input; (c) we repeat this process to train all the sub-networks sequentially.

Algorithm 2 Solving 1D Gradient Flows with PINN

1. Neural Network Representation

(a) Discretize field $u(x, t)$ in time domain with a time interval τ:
$u_k = u(x, t = k\tau), k = 1, 2, ...$

(b) Approximate each u_k with a separate neural network:
$u_k = \mathcal{N}_k(x; \theta)$. x is the input spatial coordinate; θ denotes the weights and biases to be trained

2. Loss Function Construction

$\mathcal{L} = \int_{-1}^{1} u_k^2 \,\mathrm{d}x + \int_{-1}^{1} \frac{(u_k - u_{k-1})^2}{2\tau} \,\mathrm{d}x$
$\approx \frac{2}{N}\sum_{i=1}^{N} u_k(x_i)^2 + \frac{2}{N}\sum_{i=1}^{N} \frac{[u_k(x_i) - u_{k-1}(x_i)]^2}{2\tau}$,
$u_k(x_i)$ is the output evaluated at i^{th} sampled input location x_i.
(Integration is performed using Monte Carlo method)

3. Training Process

(a) Training data: $\{x_i\}_{i=1}^{N}$ - randomly sampled N points following a uniform distribution in the space domain

(b) Given initial condition u_0, at k^{th} time step,
i. Evaluate loss $\mathcal{L}(u_k, u_{k-1})$
ii. Train network $\mathcal{N}_k$
iii. $k \to k + 1$, go to i.

All the sub-networks were constructed with two hidden layers with 20 nodes each. The ReLU activation function was adopted. We randomly sampled 100 uniformly distributed data points within the space domain ($N = 100$) as the training data. A learning rate of 0.001 and the Adam optimizer were used. Each sub-network was trained for 1500 epochs in sequence. The training took around 5 min on a single GPU (NVIDIA T400 4 GB) system. The predictions of the sub-networks at different time steps are compared with the analytical results and a good match can be observed (Fig. 2b). We can also see the decreasing free energy of the system (Fig. 2c). These successfully validate the feasibility to implement the minimizing movement scheme into physics-informed machine learning. However, this PINN-based framework is not very efficient. First, all the sub-networks have to be trained in sequence and the number of sub-networks will increase largely if predictions at a large time interval are required. Second, the PINNs need to be retrained once the initial condition is changed, which limits their applications.

Solving equations with DeepONet, as proposed in the previous section, could address the above-mentioned limits of PINN. A Phase-Field DeepONet is constructed specifically for dynamic systems in 1D, where we take the distribution of the current time step as the input and predicts the distribution of the next time step. As shown in Fig. 3a, both the branch net and the trunk net are fully connected neural networks. Note that a continuous function cannot be directly fed into the branch net; discretization is therefore performed here. We discretized u_k by sampling its values at equally spaced locations $\{x_s\}_{s=1}^{D_s}$ in the space domain, where D_s is the number of discretized points

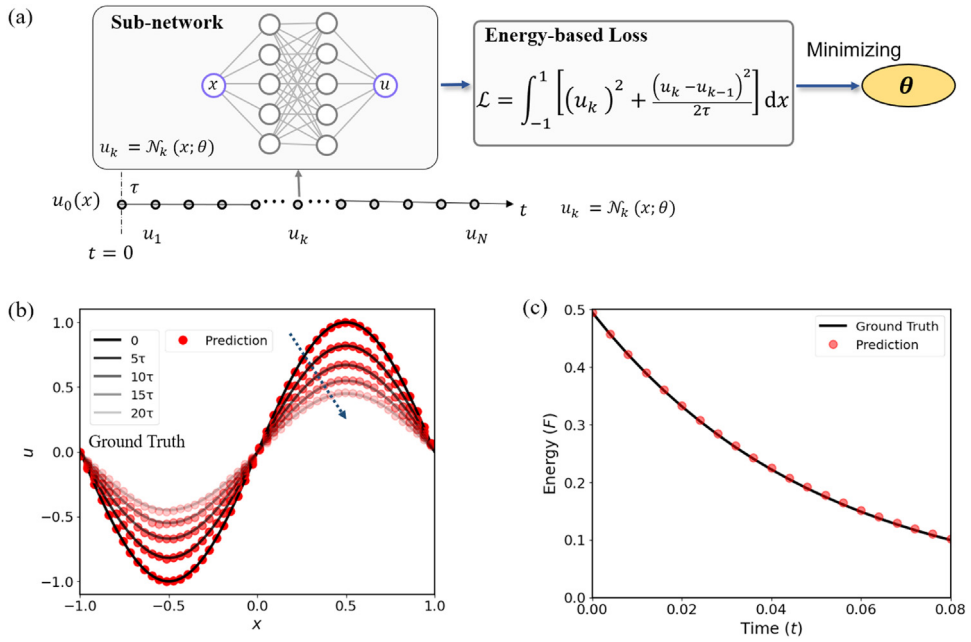

Fig. 2. (a) The multi-network structure for minimizing movement scheme; (b) Comparison between the ground truth and neural network predictions at different time steps; (c) Predicted free energy change with time.

(also referred as sensors). The resulting vector $\boldsymbol{u}_k^j = [u_k^{j,1}, u_k^{j,2}, \ldots, u_k^{j,D_s}]^T$ is then treated as the input of the branch net(see Fig. 3b), where superscript j indicates the j^{th} sampled input vector of branch net.

We use $\{x_i\}_{i=1}^{N_t}$ to denote a total of N_t sampled inputs for the trunk net and $\{\boldsymbol{u}_k^j\}_{j=1}^{N_b}$ to denote N_b sampled input vectors for the branch net. The input pair for the whole network is then denoted as $\{x_i, \boldsymbol{u}_k^{\tau,j}\}_l$, where subscript $l \in \{(i,j) \mid i \in [1, N_t], j \in [1, N_b]\}$. The total number of input samples is $N = N_t \cdot N_b$. The loss function is also constructed following Eq. (13). The loss function is defined as

$$\mathcal{L}_{\text{ONet}} = \frac{1}{N_b} \sum_{j=1}^{N_b} \left[\mathcal{F}(u_{k+1}) + \frac{d_{L^2}^2 \left(u_{k+1}, u_k\right)}{2\tau} \right] \tag{23}$$

where

$$\begin{aligned} \mathcal{F}(u_{k+1}) &\approx \frac{2}{N_t} \sum_{i=1}^{N_t} \left[u_{k+1}(x_i, \boldsymbol{u}_k^j) \right]^2 \\ d_{L^2}^2 \left(u_{k+1}, u_k\right) &\approx \frac{2}{N_t} \sum_{i=1}^{N_t} \left[u_{k+1}(x_i, \boldsymbol{u}_k^j) - u_k(x_i) \right]^2. \end{aligned} \tag{24}$$

The physical constant k is also set to 1; a given initial condition is not necessary for DeepONet. To train the network, we evenly sampled 100 points ($N_t = 100$) in the space domain $[-1, 1]$. We generated 5,000 random distributions ($N_b = 5000$) by Gaussian random process as the inputs of the branch net, 4,000 of which was used for training and the remaining 1,000 for testing. The size and structure of the DeepONet can be found in Table 1, which is determined by a grid search. The goal is to effectively minimize the training/testing error with a relatively small network size to avoid over-fitting. The network was trained for 2000 epochs with a learning rate of 0.001. The training time was around 10 min on the same single GPU system. Table 2 lists the training parameters and the R^2-value and L^2 relative error in the testing dataset. Fig. 3c shows the training and testing mean square error. An averaged coefficient of determination (R^2-value) of 0.99 and a L^2 relative error of 1.13% are achieved in the testing dataset. This implies that the trained DeepONet can accurately predict the relaxation kinetics.

Algorithm 3 Solving 1D Gradient Flows with Phase-Field DeepONet

1. Neural Network Representation

(a) Discretize field $u(x,t)$ in time domain with a time interval τ:
$u_k = u(x, t = k\tau), k = 1, 2, ...$

(b) Approximate u_{k+1} (subsequent time step) based on u_k (current time step) with an operator neural network:
$u_{k+1} = \mathcal{N}(x, \boldsymbol{u}_k; \theta)$.
x is the input spatial coordinate of the trunk net;
$\boldsymbol{u}_k$ denotes the input vector (sensors) of the branch net;
θ represents the weights and biases to be trained

2. Loss Function Construction

$$\mathcal{L}_{\text{ONet}} \approx \frac{2}{N_b}\sum_{j=1}^{N_b}\frac{2}{N_t}\sum_{i=1}^{N_t}\left\{\left[u_{k+1}(x_i, \boldsymbol{u}_k^{\tau,j})\right]^2 + \frac{\left[u_{k+1}^{\tau}(x_i, \boldsymbol{u}_k^{\tau,j}) - u_k^{\tau}(x_i)\right]^2}{2\tau}\right\}$$

N_b - number of sampled input vector $\boldsymbol{u}_k$
N_t - number of sampled input location x_i

3. Training Process

(a) Training data: $\{x_i, \boldsymbol{u}_k^{\tau,j}\}_{(i,j)}$
x_i is sampled uniformly in the space domain;
$\boldsymbol{u}_k^{\tau,j}$ is sampled by Gaussian random process.

(b) To facilitate numerical integration, $\{x_1, x_2, ..., x_{N_t}, \boldsymbol{u}_k^{\tau,j}\}$ is feed to the network as single sample.

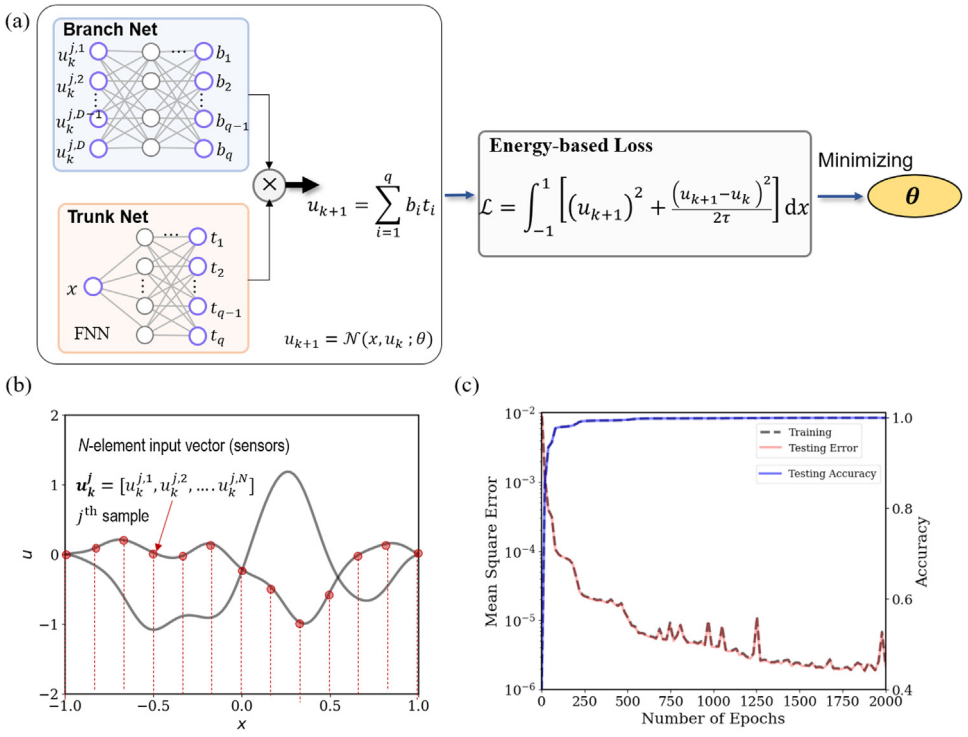

Fig. 3. (a) Illustration of physics-informed DeepONet incorporating minimizing movement scheme for 1D relaxation kinetics; (b) Random distribution of u generated by Gaussian Random Process and the discretized input vector; (c) Mean square error and R^2-value (accuracy) in training and testing sets.

Fig. 4a demonstrates the predicted distributions of u at different time steps, which agree well with the ground truth. Note that the sinusoidal input of u is outside of the training dataset, which indicates a good generality of the trained

Table 1
Parameters of the network structures for the three different cases.

Case	Trunk net		Branch net		Sensors #
	Depth	Width	Depth	Width	
Relaxation Kinetics (1D)	3	20	2	20	20
Cahn–Hilliard (1D)	3	100	2	100	20
Allen–Cahn (2D)	3	100	Filters: 32 Kernels: (3, 3) Strides: (1, 1)	6 (3, 3) (3, 3)	28 × 28

Table 2
Training parameters and results for the three different cases.

Case	#u Train	#u Test	# Epochs	Train time	R^2	L^2-error
Relax. Kinetics (1D)	4000	1000	2000	10 min	0.9991	1.13%
Allen–Cahn (2D)	4000	1000	1000	150 min	0.9990	3.18%
Cahn–Hilliard (1D) $\varepsilon = 0.25$	4000	1000	1000	180 min	0.9995	0.08%
$\varepsilon = 0.025$	4000	1000	1000	180 min	0.9989	3.15%

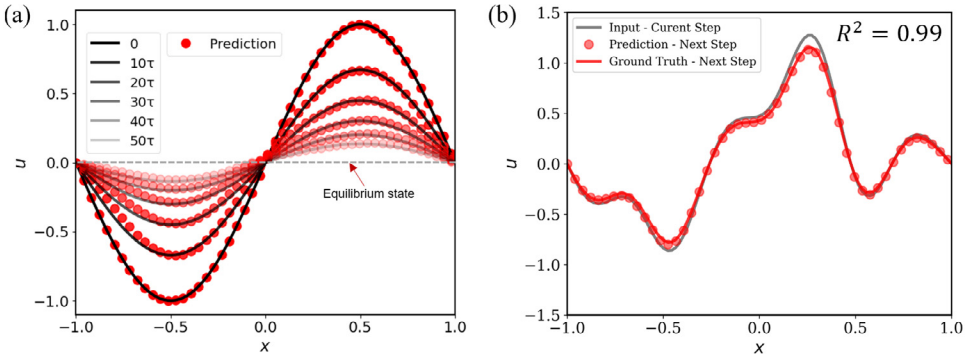

Fig. 4. Predictions of trained DeepONet: (a) predictions at multiple time steps; (b) prediction for a random input.

DeepONet. Besides, the trained Phase-Field DeepONet can easily predict a much longer time evolution, approaching the equilibrium state ($u = 0$). More importantly, given a random input of u at any time step, the trained network can accurately predict the distribution of u at the next step (Fig. 4b). In other words, it works for any given initial condition.

4.2. L^2 Gradient flow: Allen-cahn equation

In this example, the more complex Allen–Cahn equation in 2D domain is explored. The total free energy governing this equation is,

$$\begin{aligned} \mathcal{F}(u) &= \int_{\Omega} F(u)\,\mathrm{d}A \\ &= \int_{\Omega} \left[\frac{1}{\varepsilon^2} f(u) + 0.5(\nabla u)^2\right] \mathrm{d}A, \end{aligned} \tag{25}$$

where $F(u)$ denotes the total energy density; $f(u)$ is the bulk energy density and a common choice is $f(u) = \frac{(u^2-1)^2}{4}$; the second term represents the interfacial energy; ε is the physical constant (length scale). With the L^2 norm (or

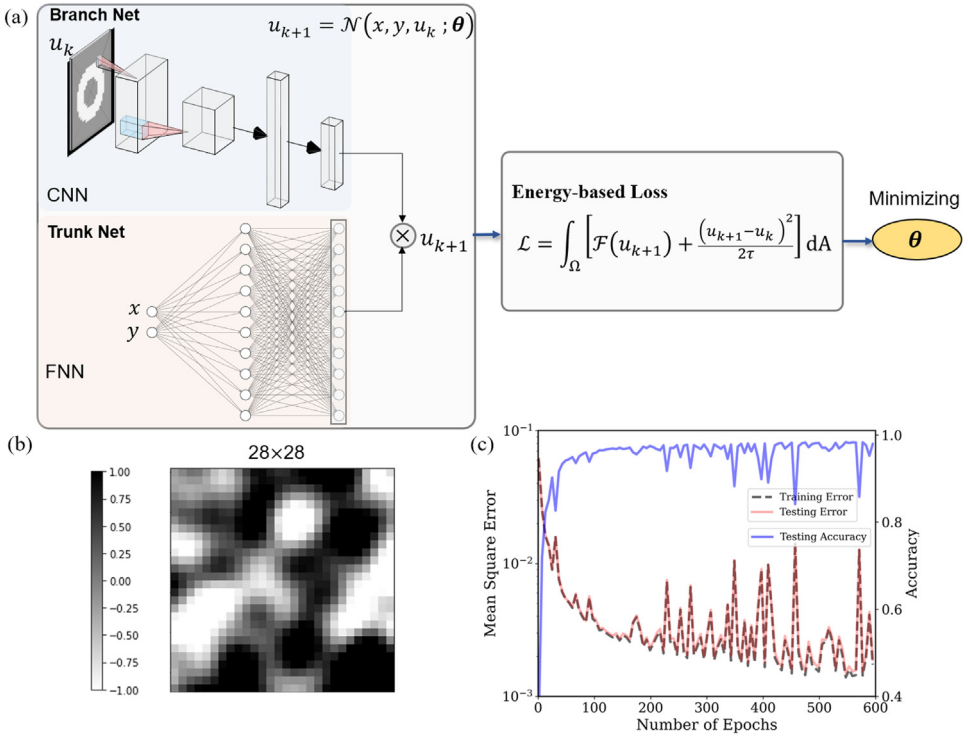

Fig. 5. (a) Illustration of physics-informed DeepONet incorporating minimizing movement scheme for 2D Allen–Cahn equation; (b) Image input of the branch net (28 × 28 grid of random distributed u generated by Gaussian Random Process); (c) Mean square error and R^2-value (accuracy) in training and testing sets.

inner product), the corresponding PDE and boundary conditions can then be derived with Eqs. (5), (7) and (11),

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla^2 u - \frac{1}{\varepsilon^2}\dfrac{\mathrm{d}f}{\mathrm{d}u}, \\ \nabla u \cdot \boldsymbol{n} = 0. \end{cases} \tag{26}$$

A 2D domain $[-1, 1] \times [-1, 1]$ is considered. The only physical constant, length scale ε, is set to 0.25. We constructed a DeepONet structure, as shown in Fig. 5a. The branch net takes the 2D distribution at the current time step as the input; a convolution neural network (CNN) is therefore adopted, which is then connected to a FNN. The trunk net is a FNN taking the spatial coordinates as the input. The final output is the distribution at the next time step. The loss function for this case is

$$\mathcal{L}_{\mathrm{ONet}} = \frac{1}{N_b}\sum_{j=1}^{N_b}\left[\mathcal{F}(u_{k+1}) + \frac{d^2_{L^2}\left(u_{k+1}, u_k\right)}{2\tau}\right], \tag{27}$$

where

$$\begin{aligned} \mathcal{F}(u_{k+1}) &\approx \frac{4}{N_t}\sum_{i=1}^{N_t} F(u_{k+1}(x_i, \boldsymbol{u}_k^j)) \\ d^2_{L^2}\left(u_{k+1}, u_k\right) &\approx \frac{4}{N_t}\sum_{i=1}^{N_t}\left[u_{k+1}(x_i, \boldsymbol{u}_k^j) - u_k(x_i)\right]^2. \end{aligned} \tag{28}$$

The network structure and training parameters can be found in Tables 1 and 2. The CNN in the branch net consists of two layers. The first layer has 32 filters with a kernel size of 3 and a stride of 1; the second layer has 6 filters with a kernel size of 3 and a stride of 3. The output of CNN is then flattened and fully connected to

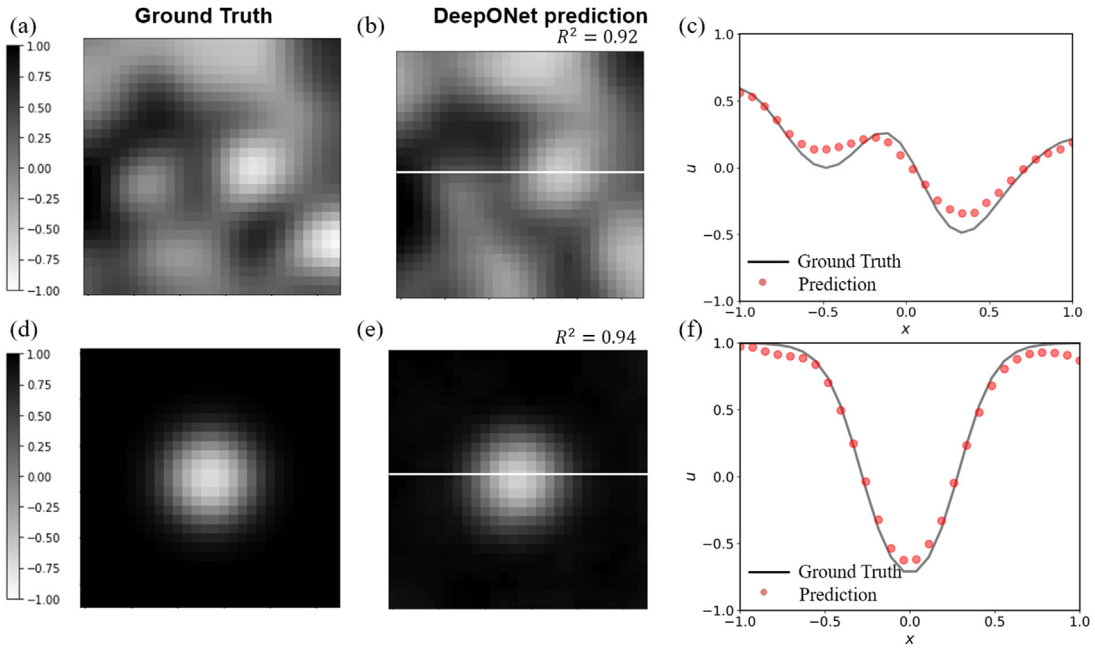

Fig. 6. Comparison between the predictions of the trained DeepONet and the ground truth for two representative cases: (a), (b), (c) for a case in the testing set; (d), (e), (f) for another distinct case outside the dataset. Data points in (c) and (f) were extracted from the diagonals as indicated in (b) and (e).

an output layer of 120 nodes. The trunk net consists of three hidden layers and each has 120 nodes. Random 2D distributions of u are generated by Gaussian random process on a uniform 28×28 grid as the input of the branch net (Fig. 5b). To simplify the training process, the corresponding spatial coordinates of the same uniform 28×28 grid ($N_t = 256$) were taken as the input of the trunk net. We set time step $\tau = 0.005$. The training process took around 150 min on the single GPU system due to the relatively large network and higher gradient terms in the loss function.

Fig. 5c shows the training and testing error. An R^2-value as high as 0.96 can be reached in the testing dataset. The predictions of the trained DeepONet are compared with the corresponding solution by the finite difference method, which is treated as the ground truth (Fig. 6). One representative sample in the testing set is compared in Fig. 6a, b, and c, where we can see the prediction of the neural network well matches the ground truth. We also compared a quite distinct case (Fig. 6d, e, and f), the distribution of which is not seen in the training set. A good match can still be observed. This further validates the reliability of the proposed framework. More importantly, the trained DeepONet can fast predict the long time field evolution. Fig. 7 shows a comparison over a long time evolution, where we see the DeepONet predictions agree well with the ground truth even approaching the equilibrium state ($u = 1$).

4.3. H^{-1} Gradient flow: Cahn–hilliard equation

An even more challenging case is the Cahn–Hilliard equation. Though the free energy is the same as in the previous case (Eq. (25)), a different inner product, the H^{-1} inner product, is used to describe the dynamics, which gives a higher order of PDE with boundary conditions,

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla^2 \left(\frac{1}{\varepsilon^2} \dfrac{\mathrm{d}f}{\mathrm{d}u} - \nabla^2 u \right), \\ \nabla \left(\frac{1}{\varepsilon^2} \dfrac{\mathrm{d}f}{\mathrm{d}u} - \nabla^2 u \right) \cdot \boldsymbol{n} = 0, \\ \nabla u \cdot \boldsymbol{n} = 0. \end{cases} \tag{29}$$

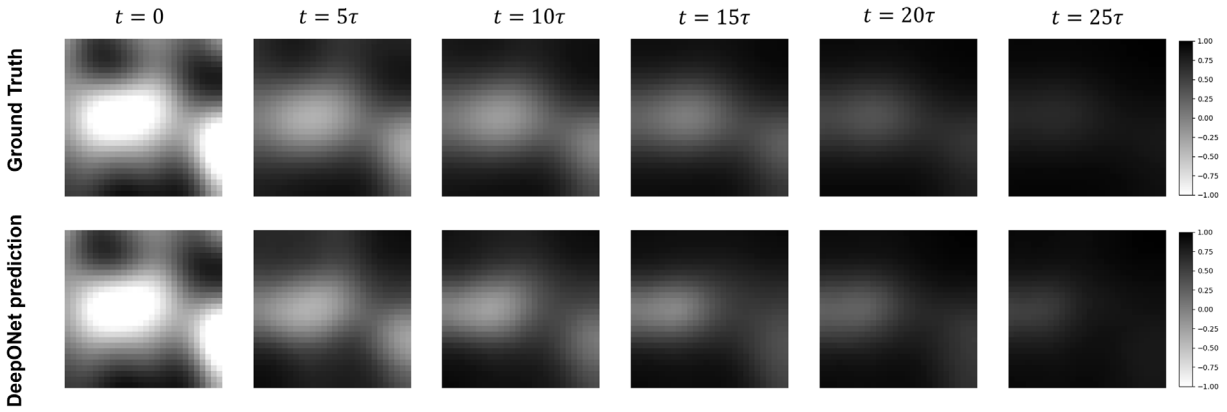

Fig. 7. Comparison between the predictions of the trained DeepONet and the ground truth for different time steps.

We consider a 1D domain [0, 1]. The loss function is,

$$\mathcal{L}_{\text{ONet}} = \frac{1}{N_b}\sum_{j=1}^{N_b}\left[\mathcal{F}(u_{k+1}) + \frac{d_{H^{-1}}^2\left(u_{k+1}, u_k\right)}{2\tau}\right], \tag{30}$$

where

$$\begin{aligned}\mathcal{F}(u_{k+1}) &\approx \frac{1}{N_t}\sum_{i=1}^{N_t} F(u_{k+1}(x_i, \boldsymbol{u}_k^j))\\ d_{H^{-1}}^2\left(u_{k+1}, u_k\right) &\approx \frac{1}{N_t}\sum_{i=1}^{N_t}\left[\frac{(u_{k+1}-u_k)(\phi_{k+1}-\phi_k)}{2\tau}\right]^2.\end{aligned} \tag{31}$$

ϕ_k is the solution of the Poisson's equation with a source term u_k. A linear mapping from discretized $\boldsymbol{u}_k^j$ to $\boldsymbol{\phi}_k^j$ can be obtained with the finite difference scheme,

$$\boldsymbol{\phi}_k^j = \boldsymbol{M}^{-1}\boldsymbol{u}_k^j, \tag{32}$$

where $\boldsymbol{\phi}_k = [\phi_k^{j,1}, \phi_k^{j,2}, \ldots, \phi_k^{j,N_d}]^T$, $\boldsymbol{u}_k = [u_k^{j,1}, u_k^{j,2}, \ldots, u_k^{j,N_d}]^T$, and

$$\boldsymbol{M} = \frac{1}{\Delta x}\begin{bmatrix} -1 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & 1 & -2 & 1 & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & 1 & -2 & 1 & \\ & & & & & 1 & -1 & \end{bmatrix}, \Delta x = \frac{1}{N_d - 1}. \tag{33}$$

Considering the complexity of this problem, we started with a relatively simple PINN-based framework to further validate the feasibility of incorporating minimizing movement scheme into physics-informed machine learning. An initial condition $u(x, t = 0) = \cos(4\pi x)$ was given. We used the same structure as described in the first example. Two different cases with ($\varepsilon = 0.025$) and without ($\varepsilon = 0.25$) apparent phase separation were explored by varying the length scale ε. The corresponding time steps τ are 5×10^{-4} and 5×10^{-5} for the two cases. The training process is the same as in the first example (see Algorithm 2) and the training results are shown in Fig. 8a, b. We can see a good agreement between the predictions of PINNs and the ground truth for both cases. This indicates that incorporating minimizing movement scheme into machine learning also works for this high 4th order equation.

We then trained a DeepONet, the structure of which can be found in Table 1. The training process is the same as in the 1D L^2 gradient flow (Algorithm 3) case, except that the distance is evaluated in a different way. Fig. 8c

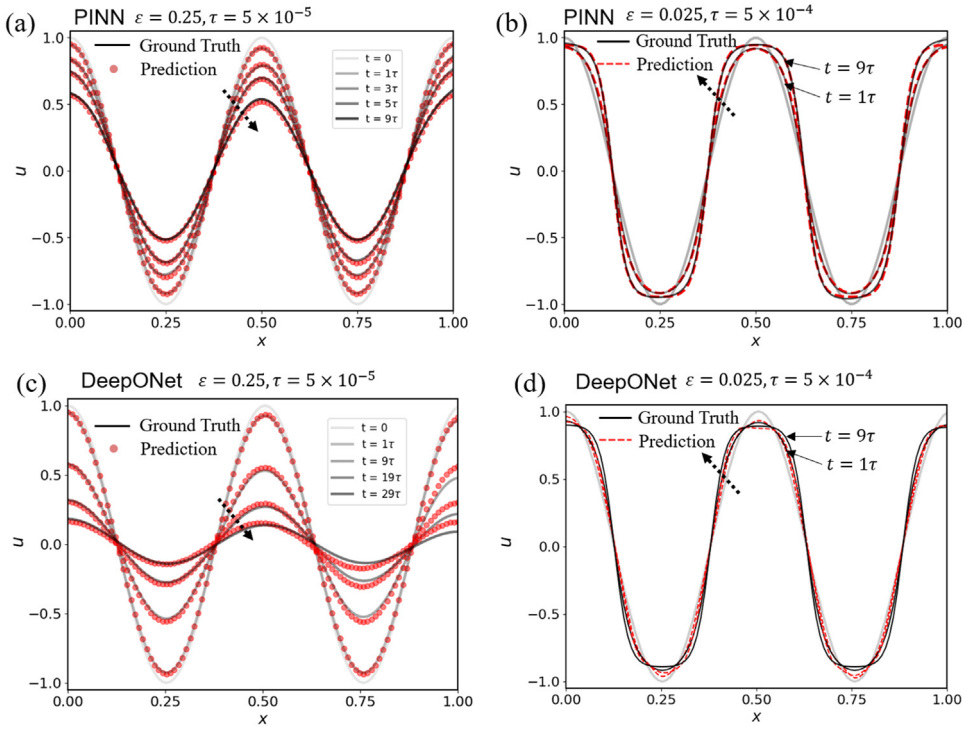

Fig. 8. Predictions of PINN for cases without phase separation (a) and with phase separation (b). (c) Prediction of DeepONet for the case without phase separation.

shows the results for the case without apparent phase separation. The prediction of trained DeepONet can also well capture the ground truth at multiple time steps. In addition, it can extend the predictions to longer time step and capture the trend reasonably even near the equilibrium state ($u = 0$). For the case with apparent phase separation (Fig. 8d), however, a noticeable deviation from ground truth can be observed, though the general trend and phase separation can be captured. Note that the prediction accuracy for single time step is still very high (see Table 2) in the testing set. This indicates that the deviation comes from the accumulated error after iterative inference of the trained neural network. It becomes a dominating error source for the case with phase separation, where the neural network cannot accurately capture the transition around the corners of the plateau and a slight error can then be magnified after a few iterations (see Section 5.3 for more details). We will further discuss the limitations and possible improvements of the proposed framework in the next section.

Even though we focus on validating the prediction accuracy of the proposed framework, we want to emphasis that the main benefit of a neural network-based surrogate is its fast inference speed after training. It is acceptable and also inevitable to sacrifice certain degree of accuracy for efficiency, considering machine learning is an optimization-based method. To demonstrate the computation efficiency of DeepONet, we compared the running time of DeepONet and the finite difference scheme when simulating the same equation (see Table 3). Since the time step of DeepONet is much larger than that required by finite difference scheme to ensure a converged solution, it demands less number of iteration to run the same time period simulation. As a result, DeepONet outperforms finite difference scheme in terms of computation efficiency. We can expect that this advantage will be more prominent when simulating complex systems.

5. Discussion

So far, we successfully developed a general physics-informed deep operator neural network with an energy-based loss function and have validated it with three different numerical examples. Here, we would like to elaborate on a few aspects to deepen the theoretical base of this approach and broaden its applicability.

Table 3
Comparison of running efficiency between neural network and finite difference method.

Case	dt	τ	Sim. time	# Iterations		Runtime	
				FD	NN	FD	NN
Allen–Cahn (2D)	1×10^{-3}	5×10^{-3}	0.5	500	100	2 s	0.4 s
Cahn–Hilliard (1D) $\varepsilon = 0.25$	2×10^{-8}	5×10^{-5}	0.01	5×10^{5}	200	5.9 s	0.7 s
$\varepsilon = 0.025$	1×10^{-6}	5×10^{-4}	0.01	1×10^{4}	20	2.7 s	0.5 s

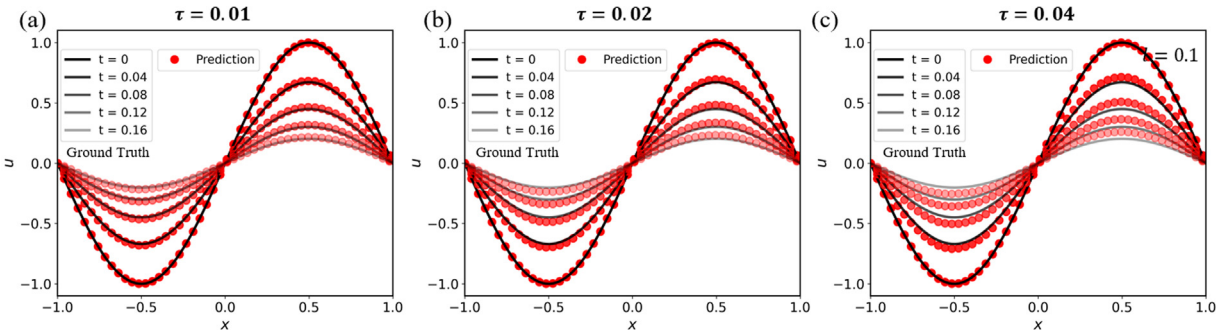

Fig. 9. Predictions of DeepONets for cases with different time steps τ: (a) $\tau = 0.01$, (b) $\tau = 0.02$, (c) $\tau = 0.04$.

5.1. Estimation of the time step τ

The time step τ needs to be small enough to achieve an accurate solution. However, to the authors' best knowledge, there is no direct method to estimate the upper limit of τ. In this study, we gradually decreased the time step to ensure the predictions converged to the ground truth. Fig. 9 shows the training results for three different values of τ in the kinetic relaxation example. As the value of τ increases from 0.01 (Fig. 9a) to 0.02 (Fig. 9b) and 0.04 (Fig. 9c), a growing discrepancy between DeepONet predictions and ground truth can be observed.

In practice, we have two suggestions for estimating the time step. First, select a time step that enables the detection of a noticeable change in the field variable if experiment data is available. Second, optimize this time step through a trial-and-error process with a small training dataset, which makes the training faster.

The potential for implementing an adaptive time step within this framework is another compelling topic to investigate. As in the phase separation case of the Cahn–Hilliard equation (Fig. 8b), before the onset of phase separation, the field variable undergoes minimal changes and this transitional period occurs over a relatively extended temporal duration. In contrast, during phase separation, the process unfolds rapidly within a short time frame. Therefore, the possible implementation of an adaptive time step, wherein the time step is larger during the pre-phase separation period and smaller during the phase separation period, has the potential to be advantageous in terms of computation efficiency.

5.2. Influence of training samples

The sampling method is another possible factor influencing the accuracy. The input function $u(x)$ of the field variable is generated by a mean-zero Gaussian random process,

$$\begin{aligned} u(x) &\sim \boldsymbol{G}(0, k_l(x_1, x_2)), \\ k_l(x_1, x_2) &= exp(-\|x_1 - x_2\|^2 / 2l^2), \end{aligned} \tag{34}$$

where $k_l(x_1, x_2)$ is the covariance kernel with a length-scale parameter l. It controls the smoothness of the generated distributions of u. A larger l results in a smoother u (Fig. 10a). In this study, the training and testing datasets are set to have the same "smoothness" for simplicity. For example, $l = 0.2$ is used for both training and testing datasets in the first numerical example. However, this may impact the accuracy when the input distribution of u has a different

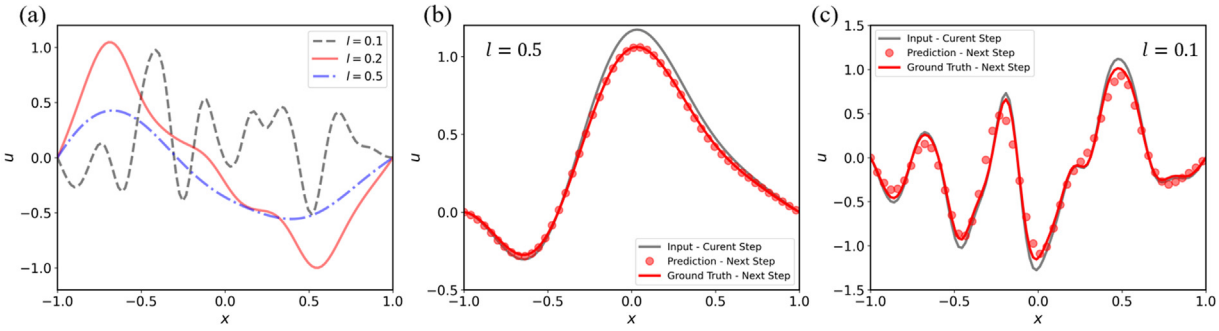

Fig. 10. (a) Random distributions generated with different length scales $l = 0.1, 0.2, and\ 0.5$; predictions of DeepONets for randomly generated inputs of u with length scales $l = 0.5$ (b) and $l = 0.5$ (c).

degree of smoothness. Fig. 10b and c show the predictions of the same DeepONet with a smoother ($l = 0.5$) and a sharper ($l = 0.1$) input of randomly disturbed u, respectively. A strong agreement can still be seen for the smoother input, however, discrepancies can be observed in the sharp corners for the sharper input. One potential solution for further enhancing the performance is to include training data with varying degrees of smoothness. Note that the overall trend is captured reasonably well even with the sharper input. This demonstrates the generality of this framework when applied to random inputs.

5.3. Error accumulation during multiple inferences

Multiple inferences of the trained DeepONet are needed to predict the dynamics of a system. Consequently, a small error from previous time steps can accumulate and propagate to later time steps, resulting in a substantial prediction error after an extended time period. This phenomenon was observed when solving the Cahn–Hilliard equation with strong phase separation (Fig. 8d). Fig. 11a demonstrates the prediction of the trained DeepONet over a time span of 40τ, approaching the equilibrium phase separation state. While the general trend of phase separation is accurately captured, the prediction error accumulates and becomes significant at later time steps, particularly inside the phase separating plateaus. As a comparison, we made the corresponding predictions using only one-time inference based on accurate inputs. For example, given the ground truth of the field variable u at $t = 9\tau$, we performed a one-time inference to obtain the output for the next time step, $t = 10\tau$. In this way, the input does not carry any error from previous steps. These single-step predictions exhibit negligible error (see Fig. 11b), indicating that the source of error stems from the accumulation of errors during multiple inferences. It remains to be further investigated to find methods mitigating this error and enhancing the robustness of the proposed framework, which will be crucial to improving the accuracy and reliability of long-time predictions using the DeepONet framework for strong phase separation problems.

5.4. Other potential improvements

Apart from optimizing the time step and sampling dataset, there are several other aspects that could be improved in the future for this framework:

a) Accounting for a wider range of physical constants. This will be beneficial to practical applications. Currently, a fixed set of physical constants is used for simplicity, but in reality, these values may vary across different systems. To address this issue, physical constants could be included as inputs by adding an extra branch net, allowing a single DeepONet to be trained for all possible physical constants.

b) Handling irregular-shaped domains. This would further enhance its applicability. DeepONet is capable of handling any fixed irregular domain by randomly sampling locations in a 2D or higher dimensional space and feeding them into the network as a vector. However, the challenge is how a DeepONet trained for a specific domain can be applied to domains with other geometric shapes.

c) Extending the framework to higher-dimensional problems. It would be interesting to investigate whether DeepONets trained at lower dimensions can be effectively applied to higher dimensions. Phase-field simulations

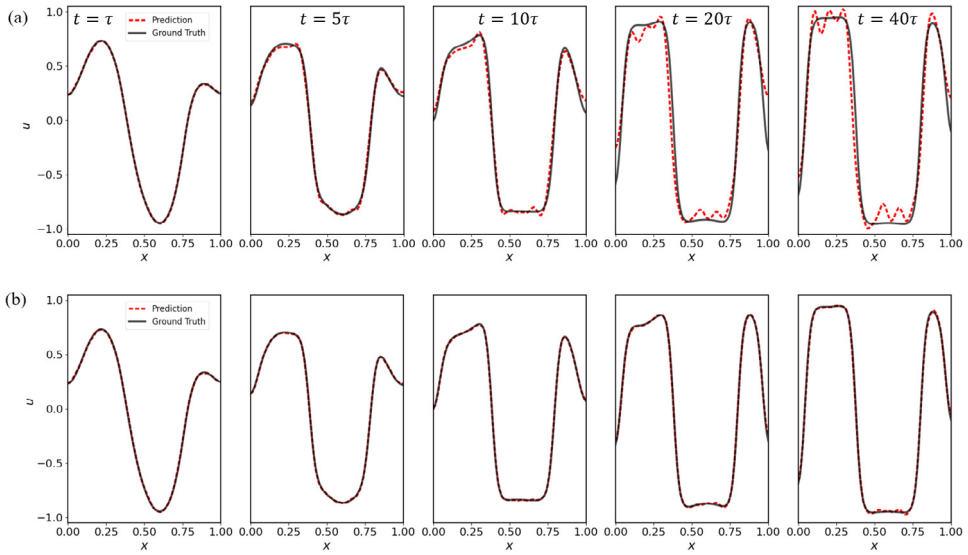

Fig. 11. DeepONet prediction of Cahn–Hilliard equation with strong phase separation: (a) Predictions with multiple inferences taking the output distribution from previous step as the input. Prediction errors accumulates with time steps (b) Predictions with one-time inference taking accurate current distribution as input.

at higher dimensions are notoriously computationally expensive, and scaling up the simulation using DeepONets could potentially overcome this issue.

d) Fast inverse learning from experiment data. The current framework focuses on the forward problem, where all physics and constants are known and the DeepONet approximates the solution with unsupervised learning (no experiment data needed). It will be even more meaningful if we could extend the framework to the inverse problem, as explained in Section 3.3 and Fig. 1a. The current framework can deal with the inverse problem. However, the training process can be time-consuming and is not applicable for fast or real-time identification. Developing a framework for fast identification based on DeepONets would require more effort.

6. Conclusion

We propose a physics-informed Phase-Field DeepONet framework for dynamical systems governed by gradient flows of free energy functionals. The minimizing movement scheme is incorporated into the framework to solve the system dynamics instead of directly solving the governing PDEs. Three different numerical examples validate the proposed framework, including the two major equations of the phase-field method, namely the Allen–Cahn and Cahn–Hilliard equations. Some major conclusions can be drawn from this work:

1. Variational principles, such as gradient flows, hold great potential to be seamlessly integrated into a physics-informed machine learning framework, providing a novel approach for the fusion of data and physics with wider practical implications.
2. The proposed framework's ability to generally solve both the Allen–Cahn and Cahn–Hilliard equations of the phase-field method, despite some error in strong phase-separation problems, showcases its promising effectiveness in simulating pattern formation in chemical systems.
3. The Phase-Field DeepONets trained in this study can serve as efficient explicit time-steppers, potentially enabling fast real-time predictions of dynamic systems.

This work raises the possibility of deep operator learning of more general phase-field models, including those of chemical nonequilibrium thermodynamics [51,52], which involve nonlinear dependencies of fluxes or reaction rates on diffusional chemical potentials. The minimizing movement scheme would need to be extended to go beyond the gradient flows approximation to account for nonlinear dynamics. In this way, the Phase-Field DeepONet framework

could enable data-driven learning and fast simulations of pattern formation from rich image datasets, going beyond PDE-constrained optimization [42].

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Juner Zhu reports financial support was provided by NASA Ames Research Center. Martin Z. Bazant reports financial support was provided by Toyota Research Institute.

Data availability

Data will be made available on request.

Acknowledgment

W.L. and J.Z. gratefully acknowledge the support of the present work through the NASA 19-TTT-0103 project (Award No. 80NSSC21M0114). They are also supported by the Northeastern University and College of Engineering startup funds. M.Z.B and W.L. are grateful for the support of Toyota Research Institute through the D3BATT Center on Data-Driven-Design of Rechargeable Batteries.

Appendix A. Basics of variational calculus

A.1. Euler-Lagrangian equation

Given a smooth manifold X and a smooth real-value function $f = f(x, u(x), u'(x))$, the functional $\mathcal{F}$ defined as

$$\mathcal{F} = \int_{x_a}^{x_b} f(x, u(x), u'(x))\,\mathrm{d}x \tag{A.1}$$

has a stationary value (maximum, minimum, or saddle point) if the Euler–Lagrangian equation is satisfied,

$$\frac{\partial f}{\partial u_i} - \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial f}{\partial (u_i)} = 0, \quad i = 1, 2, \ldots, n. \tag{A.2}$$

A.2. Derivation of functional derivative

Consider the functional derivative of a specific type of energy functional that only depends on the field variable and its first-order derivatives, i.e., $\mathcal{F} = \int_\Omega F(x, u(x), \nabla u(x))\,\mathrm{d}x$. We have

$$\begin{aligned}
\int_\Omega \frac{\delta \mathcal{F}}{\delta u} v\,\mathrm{d}x &= \left[\frac{\mathrm{d}}{\mathrm{d}t}\int_\Omega F(x, u + vt, \nabla u + \nabla v t)\,\mathrm{d}x\right]_{t=0} \\
&= \int_\Omega \left(\frac{\partial F}{\partial u} v + \frac{\partial F}{\partial \nabla u}\cdot \nabla v\right)\mathrm{d}x \\
&= \int_\Omega \left(\frac{\partial F}{\partial u} v - (\nabla \cdot \frac{\partial F}{\partial \nabla u}) v\right)\mathrm{d}x + \int_{\partial\Omega}\left(\frac{\partial F}{\partial \nabla u}\cdot \boldsymbol{n}\right) v\,\mathrm{d}x \\
&= \int_\Omega \left(\frac{\partial F}{\partial u} - \nabla \cdot \frac{\partial F}{\partial \nabla u}\right) v\,\mathrm{d}x.
\end{aligned} \tag{A.3}$$

The fourth line is valid when $v = 0$ or $\dfrac{\partial F}{\partial \nabla u}\cdot \boldsymbol{n} = 0$ on the boundary. Since v is an arbitrary function, we get the functional derivative

$$\frac{\delta \mathcal{F}}{\delta u} = \frac{\partial F}{\partial u} - \nabla \cdot \frac{\partial F}{\partial \nabla u}. \tag{A.4}$$

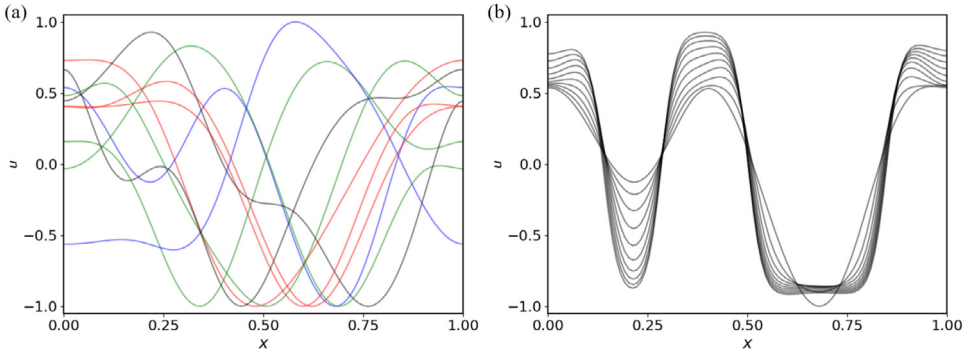

Fig. B.1. Training samples for solving Cahn–Hilliard equation with strong phase separation (a) Random distributions generated with Gaussian random process followed by a transformation to satisfy Neumann boundary condition (b) Samples extracted from time evolution of a given initial random distribution. These samples exhibit phase separation.

A.3. Inner products and norms

In this study, we focus on two different inner products, the weighted L^2 inner product (Eq. (8)) and weighted H^{-1} inner product (Eq. (9)).

The corresponding L^2 norm is

$$|f|_{L^2,M} = \sqrt{\int_\Omega \frac{f^2(x)}{M}\,\mathrm{d}x}. \tag{A.5}$$

The H^{-1} inner product requires the solution of Poisson's equation with Neumann boundary condition,

$$\begin{cases} \nabla^2\phi_f = f(x), & \text{in } \Omega, \\ \partial_{\boldsymbol{n}}\phi_f = 0, & \text{on } \partial\Omega. \end{cases} \tag{A.6}$$

Note that ϕ_f has an unique solution when $\int_\Omega f\,\mathrm{d}x = \int_\Omega \phi_f\,\mathrm{d}x = 0$. The corresponding H^{-1} norm can be obtained with

$$|f|_{H^{-1},M} = \sqrt{\int_\Omega \nabla\phi_f \cdot M\nabla\phi_f\,\mathrm{d}x}. \tag{A.7}$$

Appendix B. Process to generate training samples

We generate the training samples based on Gaussian random process. After getting the random distributions of u, we further manipulates the training data in order to satisfy certain boundary conditions. For example, in the 1D relaxation kinetics example, we did the following transformation to satisfy the homogeneous Dirichlet boundary conditions ($u(x = \pm 1) = 0$) at both ends,

$$u = 4u' \cdot (x+1)(x-1), \tag{B.1}$$

where u' denotes the random distribution generated by Gaussian random process and u represents the final samples used for training. Similarly, we did another transformation to satisfy Neumann boundary condition ($\frac{\partial u}{\partial x}(x = \pm 1) = 0$),

$$u = 16u' \cdot (x+1)^2(x-1)^2 + \epsilon_0, \tag{B.2}$$

where ϵ_0 is a random value to avoid enforcing $u(x = \pm 1) = 0$ at the same time (over-constraint).

As we discussed earlier, the training samples have a significant influence on the training results. When solving Cahn–Hilliard equation with strong phase separation, we particularly included extra samples exhibiting phase separation to increase the prediction accuracy. This is done by first generating 200 samples with Gaussian random process and the transformation mentioned above (see Fig. B.1a). For each of the sample, we further extract the time evolution of u at 24 different time steps $(2\tau, 4\tau, \ldots, 48\tau)$, as shown in Fig. B.1b. Therefore, we have a total of 5000 samples for training and testing.

References

[1] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: A brief review, Comput. Intell. Neurosci. 2018 (2018).

[2] T. Young, D. Hazarika, S. Poria, E. Cambria, Recent trends in deep learning based natural language processing, IEEE Comput. Intell. Mag. 13 (3) (2018) 55–75.

[3] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, S. Lee, Workshop Report on Basic Research Needs for Scientific Machine Learning: Core Technologies for Artificial Intelligence, Tech. Rep., USDOE Office of Science (SC), Washington, D.C. (United States), 2019, http://dx.doi.org/10.2172/1478744.

[4] S. Cai, Z. Wang, L. Lu, T.A. Zaki, G.E. Karniadakis, DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks, J. Comput. Phys. 436 (2021) 110296.

[5] C. Lin, Z. Li, L. Lu, S. Cai, M. Maxey, G.E. Karniadakis, Operator learning for predicting multiscale bubble growth dynamics, J. Chem. Phys. 154 (10) (2021) 104118.

[6] M. Yin, E. Zhang, Y. Yu, G.E. Karniadakis, Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems, Comput. Methods Appl. Mech. Engrg. (2022) 115027.

[7] B. Jiang, W.E. Gent, F. Mohr, S. Das, M.D. Berliner, M. Forsuelo, H. Zhao, P.M. Attia, A. Grover, P.K. Herring, et al., Bayesian learning for rapid prediction of lithium-ion battery-cycling protocols, Joule 5 (12) (2021) 3187–3203.

[8] B. Jiang, M.D. Berliner, K. Lai, P.A. Asinger, H. Zhao, P.K. Herring, M.Z. Bazant, R.D. Braatz, Fast charging design for Lithium-ion batteries via Bayesian optimization, Appl. Energy 307 (2022) 118244.

[9] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nat. Rev. Phys. 3 (6) (2021) 422–440, http://dx.doi.org/10.1038/s42254-021-00314-5.

[10] D.P. Finegan, J. Zhu, X. Feng, M. Keyser, M. Ulmefors, W. Li, M.Z. Bazant, S.J. Cooper, The application of data-driven methods and physics-based learning for improving battery safety, Joule 5 (2) (2021) 316–329, http://dx.doi.org/10.1016/j.joule.2020.11.018.

[11] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045.

[12] M. Raissi, G.E. Karniadakis, Hidden physics models: Machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141, http://dx.doi.org/10.1016/j.jcp.2017.11.039.

[13] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, 2019, pp. 1–21.

[14] H. Gao, L. Sun, J.-X. Wang, PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain, J. Comput. Phys. 428 (2021) 110079.

[15] Y. Wang, J. Sun, W. Li, Z. Lu, Y. Liu, CENN: Conservative energy method based on neural networks with subdomains for solving variational problems involving heterogeneous and complex geometries, Comput. Methods Appl. Mech. Engrg. 400 (2022) 115491, http://dx.doi.org/10.1016/j.cma.2022.115491.

[16] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Comput. Methods Appl. Mech. Engrg. 365 (2020) 113028.

[17] L. Lu, R. Pestourie, W. Yao, Z. Wang, F. Verdugo, S.G. Johnson, Physics-informed neural networks with hard constraints for inverse design, SIAM J. Sci. Comput. 43 (6) (2021) B1105–B1132.

[18] A.D. Jagtap, K. Kawaguchi, G.E. Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.

[19] J. Yu, L. Lu, X. Meng, G.E. Karniadakis, Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114823.

[20] W. Li, M.Z. Bazant, J. Zhu, A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches, Comput. Methods Appl. Mech. Engrg. 383 (2021) 113933, http://dx.doi.org/10.1016/j.cma.2021.113933.

[21] E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Comput. Methods Appl. Mech. Engrg. 362 (2020) 112790, http://dx.doi.org/10.1016/j.cma.2019.112790.

[22] A.F. Psaros, K. Kawaguchi, G.E. Karniadakis, Meta-learning PINN loss functions, J. Comput. Phys. 458 (2022) 111121.

[23] E. Weinan, B. Yu, The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, Commun. Math. Stat. 1 (6) (2018) 1–12.

[24] 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 (3) (2021) 218–229, http://dx.doi.org/10.1038/s42256-021-00302-5.

[25] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Learning maps between function spaces, 2021, arXiv:2108.08481 [cs, math], arXiv:2108.08481.

[26] K.A. Severson, P.M. Attia, N. Jin, N. Perkins, B. Jiang, Z. Yang, M.H. Chen, M. Aykol, P.K. Herring, D. Fraggedakis, M.Z. Bazant, S.J. Harris, W.C. Chueh, R.D. Braatz, Data-driven prediction of battery cycle life before capacity degradation, Nat. Energy 4 (5) (2019) 383–391, http://dx.doi.org/10.1038/s41560-019-0356-8.

[27] W. Li, J. Zhang, F. Ringbeck, D. Jöst, L. Zhang, Z. Wei, D.U. Sauer, Physics-informed neural networks for electrode-level state estimation in lithium-ion batteries, J. Power Sources 506 (2021) 230034.

[28] J. Tian, R. Xiong, J. Lu, C. Chen, W. Shen, Battery state-of-charge estimation amid dynamic usage with physics-informed deep learning, Energy Storage Mater. 50 (2022) 718–729.

[29] B. Wu, B. Zhang, C. Deng, W. Lu, Physics-encoded deep learning in identifying battery parameters without direct knowledge of ground truth, Appl. Energy 321 (2022) 119390.

[30] M. Aykol, C.B. Gopal, A. Anapolsky, P.K. Herring, B. van Vlijmen, M.D. Berliner, M.Z. Bazant, R.D. Braatz, W.C. Chueh, B.D. Storey, Perspective—combining physics and machine learning to predict battery lifetime, J. Electrochem. Soc. 168 (3) (2021) 030525.
[31] R.G. Nascimento, M. Corbetta, C.S. Kulkarni, F.A. Viana, Hybrid physics-informed neural networks for lithium-ion battery modeling and prognosis, J. Power Sources 513 (2021) 230526.
[32] A. Bills, S. Sripad, W.L. Fredericks, M. Guttenberg, D. Charles, E. Frank, V. Viswanathan, Universal battery performance and degradation model for electric aircraft, 2020, arXiv preprint arXiv:2008.01527.
[33] B.-R. Chen, M.R. Kunz, T.R. Tanim, E.J. Dufek, A machine learning framework for early detection of lithium plating combining multiple physics-based electrochemical signatures, Cell Rep. Phys. Sci. 2 (3) (2021) 100352.
[34] J. Lim, Y. Li, D.H. Alsem, H. So, S.C. Lee, P. Bai, D.A. Cogswell, X. Liu, N. Jin, Y.S. Yu, N.J. Salmon, D.A. Shapiro, M.Z. Bazant, T. Tyliszczak, W.C. Chueh, Origin and hysteresis of lithium compositional spatiodynamics within battery primary particles, Science 353 (2016) 566–571, http://dx.doi.org/10.1126/science.aaf4914.
[35] Y. Li, F. El Gabaly, T.R. Ferguson, R.B. Smith, N.C. Bartelt, J.D. Sugar, K.R. Fenton, D.A. Cogswell, A.L.D. Kilcoyne, T. Tyliszczak, M.Z. Bazant, W.C. Chueh, Current-induced transition from particle-by-particle to concurrent intercalation in phase-separating battery electrodes, Nature Mater. 13 (2014) 1149–1156, http://dx.doi.org/10.1038/nmat4084.
[36] M.Z. Bazant, Theory of chemical kinetics and charge transfer based on nonequilibrium thermodynamics, Accounts Chem. Res. 46 (5) (2013) 1144–1160.
[37] T.R. Ferguson, M.Z. Bazant, Nonequilibrium thermodynamics of porous electrodes, J. Electrochem. Soc. 159 (2012) A1967–A1985, http://dx.doi.org/10.1149/2.048212jes, URL https://iopscience.iop.org/article/10.1149/2.048212jes. NULL.
[38] R.B. Smith, M.Z. Bazant, Multiphase porous electrode theory, J. Electrochem. Soc. 164 (2017) E3291–E3310, http://dx.doi.org/10.1149/2.0171711jes, http://arxiv.org/abs/1702.08432. http://jes.ecsdl.org/lookup/doi/10.1149/2.0171711jes.
[39] C.L. Wight, J. Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, 2020, arXiv preprint arXiv:2007.04542.
[40] R. Mattey, S. Ghosh, A physics informed neural network for time-dependent nonlinear and higher order partial differential equations, 2021, arXiv preprint arXiv:2106.07606.
[41] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Engrg. 390 (2022) 114474.
[42] H. Zhao, H. Deng, A. Cohen, J. Lim, Y. Li, D. Fraggedakis, B. Jiang, B. Storey, W. Chueh, R. Braatz, M. Bazant, Learning heterogeneous reaction kinetics from X-ray movies pixel-by-pixel, 2022, http://dx.doi.org/10.21203/rs.3.rs-2320040/v1.
[43] H.D. Deng, H. Zhao, N. Jin, L. Hughes, B.H. Savitzky, C. Ophus, D. Fraggedakis, A. Borbély, Y.-S. Yu, E.G. Lomeli, et al., Correlative image learning of chemo-mechanics in phase-transforming solids, Nature Mater. 21 (5) (2022) 547–554.
[44] J.W. Cahn, J.E. Taylor, Overview no. 113 surface motion by surface diffusion, Acta Metall. Mater. 42 (4) (1994) 1045–1063, http://dx.doi.org/10.1016/0956-7151(94)90123-6.
[45] J.E. Taylor, J.W. Cahn, Linking anisotropic sharp and diffuse surface motion laws via gradient flows, J. Stat. Phys. 77 (1) (1994) 183–197, http://dx.doi.org/10.1007/BF02186838.
[46] F. Santambrogio, { Euclidean, Metric, and Wasserstein } gradient flows: An overview, 2016, arXiv:1609.03890 [math], arXiv:1609.03890.
[47] A. Singh, S. Pal, Coupled chemo-mechanical modeling of fracture in polycrystalline cathode for lithium-ion battery, Int. J. Plast. 127 (2020) 102636.
[48] A. Singh, S. Pal, Chemo-mechanical modeling of inter-and intra-granular fracture in heterogeneous cathode with polycrystalline particles for lithium-ion battery, J. Mech. Phys. Solids 163 (2022) 104839.
[49] J.S. Rowlinson, Translation of JD van der Waals'"the thermodynamik theory of capillarity under the hypothesis of a continuous variation of density", J. Stat. Phys. 20 (1979) 197–200.
[50] J.W. Cahn, J.E. Hilliard, Free energy of a nonuniform system. I. Interfacial free energy, J. Chem. Phys. 28 (2) (1958) 258–267, http://dx.doi.org/10.1063/1.1744102.
[51] M.Z. Bazant, Thermodynamic stability of driven open systems and control of phase separation by electro-autocatalysis, Faraday Discuss. 199 (2017) 423–463, http://dx.doi.org/10.1039/C7FD00037E, http://pubs.rsc.org/en/Content/ArticleLanding/2017/FD/C7FD00037E. http://xlink.rsc.org/?DOI=C7FD00037E.
[52] M.Z. Bazant, B.D. Storey, A.A. Kornyshev, Double layer in ionic liquids: Overscreening versus crowding, Phys. Rev. Lett. 106 (2011) 046102, http://dx.doi.org/10.1103/PhysRevLett.106.046102, URL https://link.aps.org/doi/10.1103/PhysRevLett.106.046102.