Machine-learning-based data-driven discovery of nonlinear phase-field dynamics

Elham Kiyani,[1,2] Steven Silber,[2,3] Mahdi Kooshkbaghi,[4] and Mikko Karttunen[2,3,5]

[1]*Department of Mathematics, The University of Western Ontario, 1151 Richmond Street, London, Ontario, Canada N6A 5B7*

[2]*The Centre for Advanced Materials and Biomaterials (CAMBR), The University of Western Ontario, 1151 Richmond Street, London, Ontario, Canada N6A 5B7*

[3]*Department of Physics and Astronomy, The University of Western Ontario, 1151 Richmond Street, London, Ontario, Canada N6A 3K7*

[4]*Simons Center for Quantitative Biology, Cold Spring Harbor Laboratory, Cold Spring Harbor, New York 11724, USA*

[5]*Department of Chemistry, The University of Western Ontario, 1151 Richmond Street, London, Ontario, Canada N6A 5B7*

(Received 3 August 2022; accepted 15 November 2022; published 9 December 2022)

One of the main questions regarding complex systems at large scales concerns the effective interactions and driving forces that emerge from the detailed microscopic properties. Coarse-grained models aim to describe complex systems in terms of coarse-scale equations with a reduced number of degrees of freedom. Recent developments in machine-learning algorithms have significantly empowered the discovery process of governing equations directly from data. However, it remains difficult to discover partial differential equations (PDEs) with high-order derivatives. In this paper, we present data-driven architectures based on a multilayer perceptron, a convolutional neural network (CNN), and a combination of a CNN and long short-term memory structures for discovering the nonlinear equations of motion for phase-field models with nonconserved and conserved order parameters. The well-known Allen-Cahn, Cahn-Hilliard, and phase-field crystal models were used as test cases. Two conceptually different types of implementations were used: (a) guided by physical intuition (such as the local dependence of the derivatives) and (b) in the absence of any physical assumptions (black-box model). We show that not only can we effectively learn the time derivatives of the field in both scenarios, but we can also use the data-driven PDEs to propagate the field in time and achieve results in good agreement with the original PDEs.

DOI: 10.1103/PhysRevE.106.065303

I. INTRODUCTION

Partial differential equations (PDEs) are widely used in modeling of complex physical, chemical, and biological systems including fluid dynamics, chemical kinetics, population dynamics, and phase transitions. The study of PDEs in the context of machine learning (ML), broadly speaking, falls into two categories: (i) solving PDEs and (ii) predicting unknown PDEs from data [1–7]. In simulations of phase-field and reaction-diffusion models, commonly used numerical techniques are based on time and space discretization, such as finite-difference and finite-element methods. In recent years, a third approach based on ML has emerged with promising results for solving and even discovering unknown PDEs from data; see, for example, Ref. [8] and references therein.

The core idea for using ML algorithms to solve PDEs is representing the residuals of PDEs as a loss function of a neural network (NN) where the loss function is minimized; a loss function measures how far the predicted values are from their true values. This approach does not require discretization or meshing, which is beneficial when dealing with problems of high dimensions and/or complex geometries [1,2,9]. Since most deep learning frameworks are based on automatic differentiation, these methods are known as mesh-free approaches [10].

In the case of discovering unknown PDEs from data, the key idea of ML-based approaches is to estimate the time derivative of the desired (dependent) quantity. These approaches can be broadly categorized as follows:

(a) An ensemble of macroscopic observations is available, and there is knowledge about the physics of the governing coarse PDE(s). The typical knowledge is that the time evolution of the field of interest depends on the field and its derivatives (e.g., Navier-Stokes equations). One can design the ML algorithm to find that dependency. This relation can be formulated based on any of the following methods: (i) Linear dependence of the field evolution using a dictionary of spatial derivatives with unknown coefficients [11,12]; (ii) nonlinear dependence with black-box inference [7]; (iii) nonlinear dependence using a selective dictionary of spatial derivatives, which were found by other data-driven approaches [13]; (iv) nonlinear dependence where spatial derivatives are informed by the memory (history) of the system using a feedback loop, e.g., recurrent neural network (RNN) together with long short-term memory (LSTM) and gated recurrent unit (GRU) [14–16].

(b) An ensemble of microscopic observations is available, and the macroscopic field of interest is known. For example, the microscopic solutions of the Boltzmann equation are available and one is looking for the time evolution of coarse fields such as density, velocity, or temperature. Again, one can assume that the time evolution of the field depends on the spatial derivatives using physical intuition [13,17].

(c) An ensemble of microscopic observations is available, but the macroscopic field is unknown. Therefore, the first step is to discover the coarse-grained field, which is generally formulated as a model reduction problem [18]. The second step is to find the PDE(s) for the coarse variable(s). For example, Thiem *et al.* determined an order parameter for coupled oscillators using diffusion maps and the corresponding governing PDE using a Runge-Kutta network [19].

In this paper, we explore ML-based approaches which fall under the first category mentioned above. We assess two scenarios:

(i) There is an unknown relation between field evolution and its spatial derivatives.

(ii) The spatial derivatives, their orders, and combinations are unknown (there is no spatial derivative dictionary).

Afterwards, we also solve the predicted PDEs in time and space. We should note that by discovering PDEs, we are referring to finding the aforementioned unknown relation implicitly. For the first scenario, a flexible framework that can deal with large datasets and extract the unavailable PDE(s) from coarse-scale variables implicitly is developed. Two different approaches are presented for learning coarse-scale PDEs: (i) a multilayer perceptron (MLP) architecture, and (ii) a convolutional neural network and long short-term memory (CNN-LSTM). Since LSTM only passes time information to its layers and misses the spatial features of previous time steps, CNN can be used to learn and detect the spatial features of the inputs [20,21]. For the second scenario, a convolution operator is used to implicitly learn the dependence of the time derivative of the field on the spatial derivative(s) of unknown orders. The learned PDE is then marched in time with a time-integrator.

We demonstrate the capability of the above algorithms to learn PDEs using data obtained from phase-field simulations of the well-known Allen-Cahn [22], Cahn-Hilliard [23], and the phase-field crystal (PFC) [24,25] models using the open source software *SymPhas* [26].

The rest of this article is organized as follows. A brief summary of the phase-field approach and data preparation is presented in Sec. II. In Sec. III, an overview of MLP and CNN-LSTM networks as two data-driven approaches which learn PDEs with a spatial derivatives dictionary is presented. Finally, in Sec. IV, we introduce a CNN network that learns PDEs without any assumption regarding spatial derivatives. Then, the solutions of the original and data-driven PDEs are compared. The `tensorflow2` framework [27] was used to implement and train our networks throughout this paper.

II. PHASE-FIELD MODELING

A. Phase-field modeling in a nutshell

Phase-field modeling provides a theoretical and computational approach for simulating nonequilibrium processes in materials, typically with the objective of studying the dynamics and structural changes. For an excellent overview of phase-field modeling, see the book by Provatas and Elder [28]. In its essence, phase-field modeling is a coarse-grained approach that uses continuum fields to describe slow variables such as concentrations. The continuum fields are given by order parameters, which may be conserved or nonconserved. In this work, we consider only systems described by a single order parameter $U \equiv U(\vec{x}, t)$. In all of the descriptions below, we use the conventional dimensionless units [28].

The equations of motion for the nonconserved and conserved order parameters are (see Ref. [28] for a more detailed discussion) given as

$$\frac{\partial U}{\partial t} = -\Gamma \frac{\delta F}{\delta U} \quad \text{(nonconserved)}, \tag{1}$$

$$\frac{\partial U}{\partial t} = \Gamma \nabla^2 \frac{\delta F}{\delta U} \quad \text{(conserved)}, \tag{2}$$

where F is a free-energy functional and $\delta/\delta U$ is a functional derivative. We have neglected thermal noise from the above equations. The parameter Γ is a generalized mobility that is assumed to be constant, and is chosen based on a particular phase-field model. The free-energy functional typically has the form

$$F = \int d\vec{x}[|\vec{\nabla} U|^2 + f(U)], \tag{3}$$

where $f(U)$ is a bulk free energy with a double-well potential, which for this work we set to be

$$f(U) = \frac{a_4}{4} U^4 + \frac{a_2}{2} U^2. \tag{4}$$

It is also noteworthy that when no free-energy functional is available, the equations of motion are often postulated. This is the case with reaction-diffusion systems, for example the well-known Turing [29,30] and Gray-Scott models [30,31]. Phase-field models have been widely applied to various types of systems and phenomena, including dendritic and directional solidification [32–34], crystal growth [24,35–37], and magnetism [38], as well as for phenomena such as fracture propagation [39,40] to mention some examples.

B. Phase-field models used in the current work

We employed three different well-studied single order-parameter phase-field models: (i) the Allen-Cahn model [22] for the case of a nonconserved order parameter, (ii) the Cahn-Hilliard model [23] for the conserved order parameter, and (iii) the phase-field crystal (PFC) model [24] that has a conserved order parameter and generates a modulated field that describes atomistic length scales and diffusive times. In addition to being well-studied, these models were chosen because they contain differing orders of spatial derivatives: the Allen-Cahn model is described using a second-order derivative, the Cahn-Hilliard model is described using a fourth-order derivative, and the PFC model is described using a sixth-order derivative. Moreover, they each exhibit various spatial patterns that evolve according to different timescales.

1. The Allen-Cahn model

The Allen-Cahn model, a dynamical model for solidification originally developed by Allen and Cahn in 1972, has a single nonconserved order parameter corresponding to Eq. (1), and it is defined using the free energy of Eq. (3). The

TABLE I. The simulations were done on a uniformly discretized two-dimensional grid of size $n_x \times n_y$ and $\Delta x = \Delta y = 1$. The simulations use a time step of Δt and continue to time t. All models use periodic boundary conditions, and initial conditions are populated using a uniform random distribution with values between -1 and 1 generated using the Mersenne Twister 19937 generator from the C++ standard library [44]. Additionally, all constants in the equations of motion are set to 1 except for ε used in the PFC model, which is set to 0.1. Later, in our phase-field equations discovery, we used training sets with 60% (for MLP and CNN-LSTM) and 80% (for CNN) of the total snapshots ($n_k = 0.6n_t$ or $n_k = 0.8n_t$).

Phase-field model	$n_x \times n_y$	Δt	t	n_t	Equation parameters
Allen-Cahn, Eq. (5)	256×256	0.1	20	100	$M = a_2 = a_4 = 1$
Cahn-Hilliard, Eq. (6)	128×128	0.01	20	100	$D = a_2 = a_4 = 1$
PFC, Eq. (8)	128×128	0.05	100	200	$q_0 = 1$ and $\varepsilon = 0.1$

equation of motion is then given as

$$\frac{\partial U}{\partial t} = -M(\nabla^2 U + a_2 U - a_4 U^3), \tag{5}$$

where Γ is set to M [see Eq. (1)], a constant related to chemical mobility. The numerical values of the parameters M, a_2, and a_4 are given in Table I. Equation (5) represents a physical system that evolves purely due to a chemical potential. It is also called Model A according to the Hohenberg and Halperin classification of phase-field models [41].

2. The Cahn-Hilliard model

The Cahn-Hilliard model, formulated by Cahn and Hilliard in 1958, represents spinodal decomposition. It is a conserved order parameter model corresponding to Eq. (2). Applying the free-energy density of Eq. (3) then gives

$$\frac{\partial U}{\partial t} = D\nabla^2(\nabla^2 U + a_2 U + a_4 U^3), \tag{6}$$

where Γ is set to D [see Eq. (2)], a constant that represents the diffusion constant. The parameters D, a_2, and a_4 are given in Table I. Equation (6) is also known as Model B in the Hohenberg and Halperin classification [41].

3. Phase-field crystal model

A free-energy density to describe a crystal lattice at an atomistic scale incorporating elastic effects into a phase-field model was originally developed by Elder *et al.* [24,25]. The PFC free-energy functional is minimized by a hexagonal periodic lattice, and it can be defined [42] as

$$F(U) = \int d\vec{x}\left[\frac{U^3}{3} + \frac{U^4}{4} + U((q_0 + \nabla^2)^2 - \varepsilon)\frac{U}{2}\right], \tag{7}$$

where q_0 and ε are constants. The equation of motion with a conserved order parameter is then determined using Eq. (2) with $\Gamma = 1$ as

$$\frac{\partial U}{\partial t} = \nabla^2[U^2 + U^3 + ((q_0 + \nabla^2)^2 - \varepsilon)U]. \tag{8}$$

The order parameter, U, represents the mass density. The PFC model can be used to describe elastic and plastic deformations in isotropic materials, i.e., crystal structures. The lattice structure can assume any orientation (based on the initial conditions), and interactions between grains (individual crystal structures) can lead to defects and dislocations.

C. Simulation of phase-field models

The open-source *SymPhas* [26] software package was used to numerically simulate the above three systems. *SymPhas* allows the user to define phase-field models directly from their PDE formulations, and control simulation parameters from a single configuration file. All simulations were done using dimensionless units [28]. In terms of the numerical solution, *SymPhas* has the ability to simulate models using either explicit finite-difference methods or the semi-implicit Fourier spectral method. The latter was chosen. By virtue of its excellent error properties, the Fourier semi-implicit spectral method typically allows for larger time stepping than a finite difference solver [43]. For each of the models, five independent simulations with 100 frames of the field U were saved at equally spaced intervals. Parameters for the numerical simulations are summarized in Table I. For models A and B, the simulation is stopped after the fast growth regime, and the universal scaling of the system takes effect. The simulation of the PFC model was chosen to stop after approximately 10 diffusion times in order to be well within the regime where the number of defects is slowly decreasing [24]. To illustrate the different dynamics of each model, snapshots at the end of each simulation are provided in Fig. 1.

III. DATA-DRIVEN PDEs WITH A SPATIAL DERIVATIVES DICTIONARY

As already discussed above, we consider two distinct types of methods for discovering PDEs from data, assuming the network is informed by spatial derivatives either explicitly or implicitly. In the first method, an MLP network is used to learn a function F_{MLP} that can be formulated as

$$\begin{aligned} U_t(t,x,y) = F_{\mathrm{MLP}}(&U(t,x,y), U_x(t,x,y), U_{xx}(t,x,y), \\ &\times U_y(t,x,y), U_{yy}(t,x,y), \dots), \end{aligned} \tag{9}$$

where $U_t(t,x,y)$ is the time derivative, and $U_x(t,x,y)$, $U_{xx}(t,x,y)$, $U_y(t,x,y)$, and $U_{yy}(t,x,y)$ are the first and second spatial derivatives with respect to x and y, respectively.

In the second method, extending LSTM to a convolutional structure (CNN-LSTM) is used to learn an equation from local variables without giving spatial derivatives explicitly. Mathematically, the network learns the time derivative $U_t(t,x,y)$ as a function of local macroscopic variables on a small square

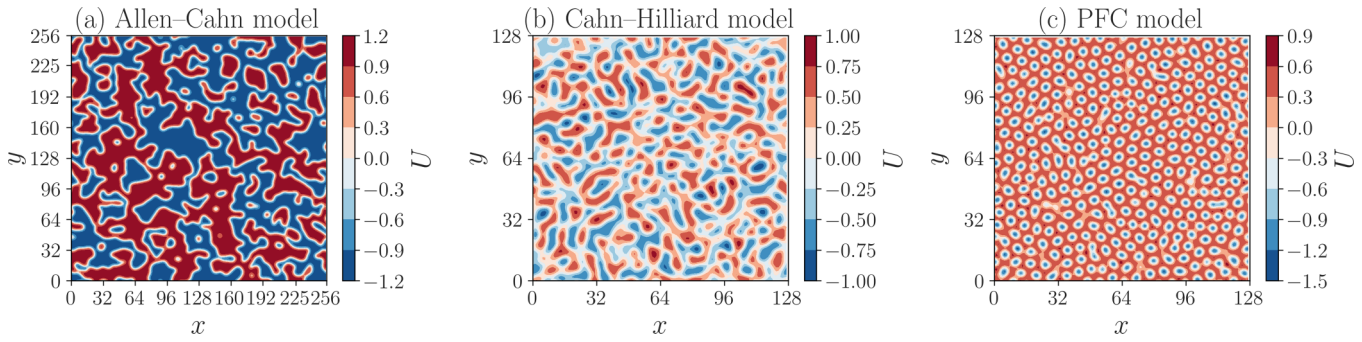


FIG. 1. Snapshots from the three phase-field models. Field solutions for the Allen-Cahn model [Eq. (5)] are shown on the left at $t = 20$, those for the Cahn-Hilliard model [Eq. (6)] are shown in the center at $t = 20$, and those for the PFC [Eq. (8)] are shown on the right at $t = 100$. The longer simulation time of the PFC model is required to allow the number of initial defects to decrease [24]. The parameters of the numerical simulations are presented in Table I. The vertical and horizontal axes display $x = n_x$ and $y = n_y$, respectively, and U represents the phase-field for the corresponding model, all in dimensionless units.

around each grid point,

$$U_t(t_k, x_i, y_j) = F_{\text{CNN-LSTM}}(U(t_k, x_{i-1}, y_j), U(t_k, x_i, y_j), \times U(t_k, x_{i+1}, y_j), U(t_k, x_i, y_{j-1}), \times U(t_k, x_i, y_{j+1})), \tag{10}$$

where $U(t_k, x_{i-1}, y_j)$, $U(t_k, x_i, y_j)$, $U(t_k, x_{i+1}, y_j)$, $U(t_k, x_i, y_{j-1})$, and $U(t_k, x_i, y_{j+1})$ are the field values at the positions x_{i-1}, x_i, x_{i+1}, y_{j-1}, and y_{i+1}, respectively, and t_k corresponds to the time of the snapshots used in the training set for $1 \leqslant k \leqslant n_k$.

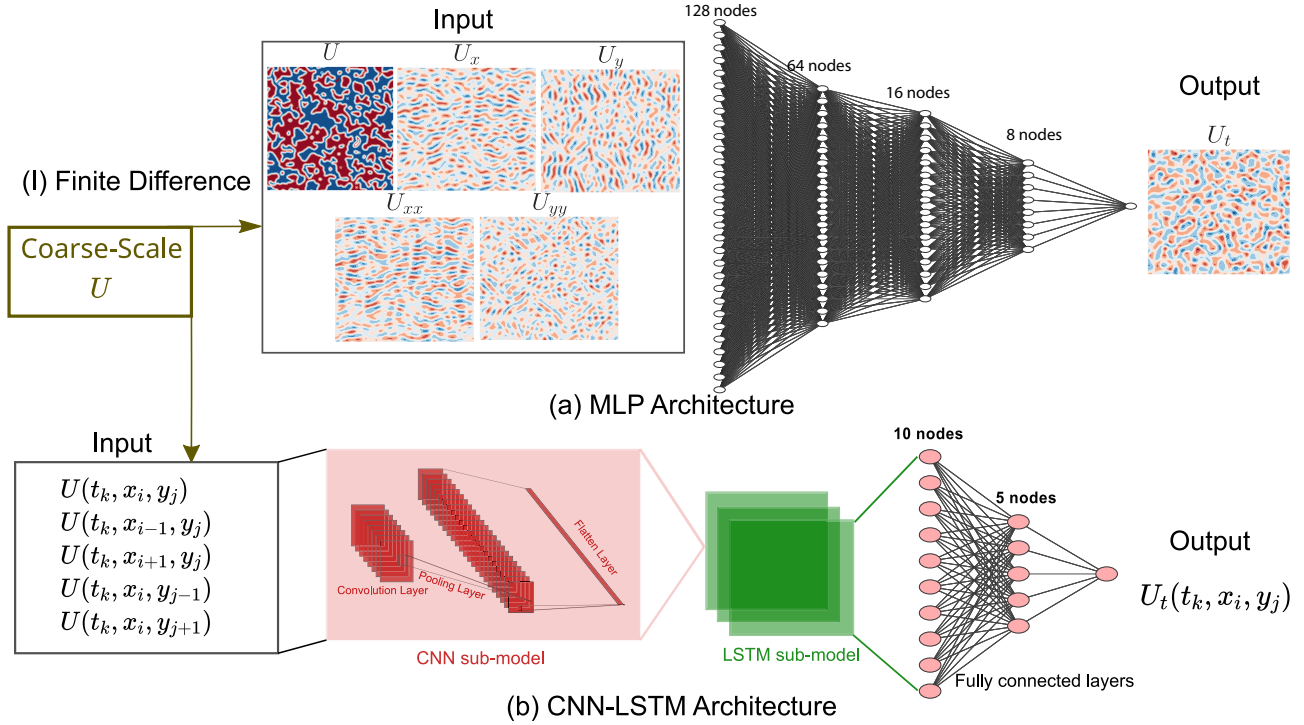


FIG. 2. Schematic of the general steps in discovery of PDEs with a spatial derivatives dictionary. Learning of PDEs from spatial derivatives and local values of coarse variables using two different approaches, (a) MLP and (b) CNN-LSTM. Coarse-scale variables are collected as snapshots from the phase-field simulations. We used a 60:20:20 ratio to randomly choose the training, validation, and test sets. Finite-difference methods are used to approximate the spatial derivatives, which are fed into panel (a) the MLP network according to Eq. (9). The network connecting the input layer consists of a list of input features (the field U and its spatial derivatives) to the output layer of a single neuron (time derivative U_t). The values of the macroscopic field U evaluated around each grid point are fed through the panel (b) CNN-LSTM network to learn PDEs of the form Eq. (10). CNN-LSTM network connecting the input layer consists of a list of input features [local variables $U(t_k, x_{i-1}, y_j), U(t_k, x_i, y_j), U(t_k, x_{i+1}, y_j), U(t_k, x_i, y_{j-1}), U(t_k, x_i, y_{j+1})$ for $1 \leqslant k \leqslant n_k$, $1 \leqslant i \leqslant n_x$, and $1 \leqslant j \leqslant n_y$] to the output layer of a single neuron U_t. Here n_k is the number of snapshots used for training, which is a random set of n_t with size $n_k = 0.6n_t$. The corresponding values for n_t, n_x, and n_y are summarized in Table I.

TABLE II. MLP architecture for discovering phase-fields given in Eqs. (5), (6), and (8) consists of four dense layers with 128/64/16/8 neurons in each layer. The network was trained with a learning rate of 10^{-3} for 2000 epochs. For each dataset, n_t snapshots were randomly split into training, validation, and test with a 60:20:20 ratio (the training set has n_k snapshots with a size of $0.6n_t$ for each dataset).

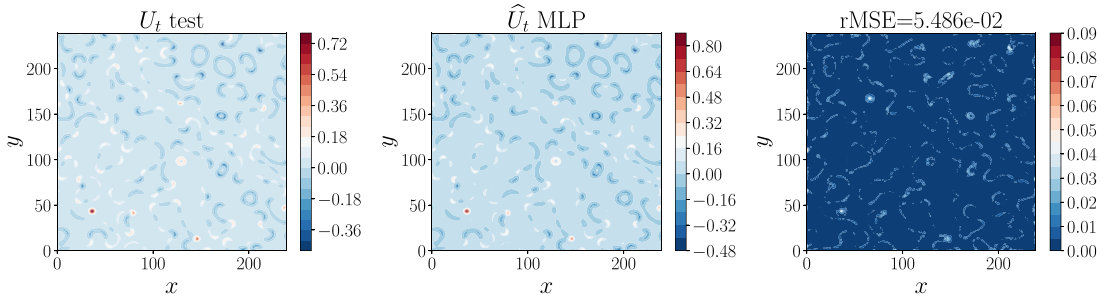
Networks	Layers	Neurons	Activation functions
MLP	four dense layers	128/64/16/8	ReLU

A schematic diagram of our framework for discovering PDEs with spatial derivatives dictionary is shown in Fig. 2. Specifically, it shows how the spatial derivatives $(U, U_x, U_y, U_{xx}, U_{yy}, \dots)$ and the local macroscopic variables $(U_{i-1,j}, U_{i,j}, U_{i+1,j}, U_{i,j-1}, U_{i,j+1})$ are fed through the MLP [Fig. 2(a)] and CNN-LSTM [Fig. 2(b)], respectively, to learn the time derivative $U_t(t, x, y)$.

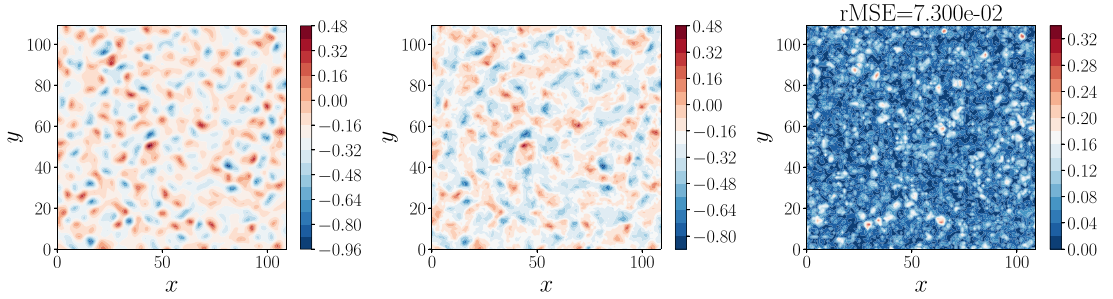
A. Multilayer perceptron network architecture and performance

An MLP is an example of a typical feedforward artificial neural network, consisting of a series of layers. Each layer calculates the weighted sum of its inputs and then applies an activation function to get a signal that is transferred to the next neuron [45].

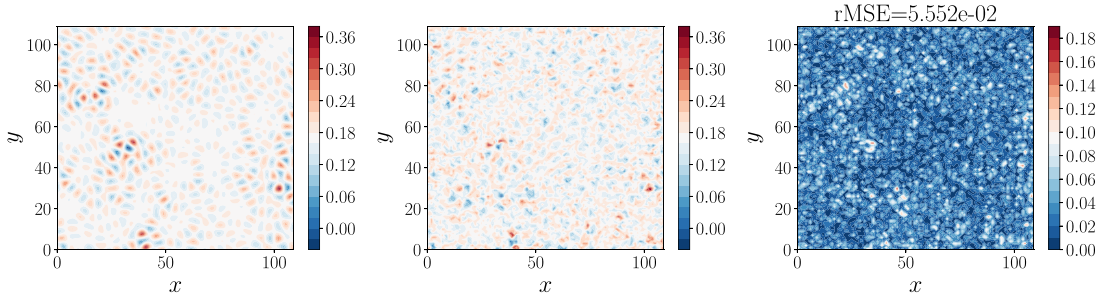
In our MLP network, the number of layers, neurons, and the type of activation functions for each phase-field model were found by trial and error. We approximate spatial derivatives of the coarse variable U by finite differences, and along with U itself, we feed this to the



(a) Allen–Cahn model



(b) Cahn–Hilliard model



(c) PFC model

FIG. 3. Performance of the MLP network for predicting time derivatives of phase-fields given by Eqs. (5), (6), and (8). Allen-Cahn, Eq. (5), as well as Cahn-Hilliard, Eq. (6), were plotted at $t = 20$, and PFC, Eq. (8), was drawn at $t = 100$. The left column shows U_t, the time derivative computed from the numerical solution generated by *SymPhas* [26], and the center column shows $\widehat{U}_t$, the learned time derivative. The right panel shows the difference between U_t and $\widehat{U}_t$, as well as the corresponding rMSE value for each phase-field model.

MLP network to learn the function F_{MLP} in Eq. (9). The MLP architecture for learning the Allen-Cahn model [Eq. (5)] is shown in Fig. 2(a) as an example. The five inputs $U(t,x,y)$, $U_x(t,x,y)$, $U_{xx}(t,x,y)$, $U_y(t,x,y)$, and $U_{yy}(t,x,y)$ are passed to the first hidden layer, which is connected to the layers with 128/64/16/8 neurons each. In the output layer, we use a dense layer with a single neuron to predict U_t. The network is trained for 2000 epochs using the ADAM optimizer [46], rectified linear unit (ReLU) activation function [47], and mean-squared error (MSE) as the loss function (see Table II). For all three phase-field models, the same architecture has been used. For the Cahn-Hilliard [Eq. (6)] and PFC [Eq. (8)] models, we used spatial derivatives up to fourth and sixth order for the input layers, respectively.The performance of the MLP network on learning the models is shown in Fig. 3. The root-mean-squared error (rMSE) is the square root of MSE calculated as

$$\mathrm{MSE} = \frac{1}{n_x \times n_y} \sum_{i=1}^{n_x \times n_y} \left(U_t^i - \widehat{U}_t^i\right)^2. \tag{11}$$

As shown in Fig. 3, the rMSE values are small ($\sim 10^{-2}$), indicating that the target time derivatives ($\widehat{U}_t$) learned by the proposed MLP network are close to the true ones for all three models.

B. Convolution and long short-term memory network architecture and performance

One of the main challenges in approximating coarse-scale PDEs is the estimation of spatial derivatives. While in previous studies PDEs have been successfully identified by learning time derivatives as a function of the estimated spatial derivatives, approximating derivatives remains challenging [48,49]. Generally, the choice of the grid size is one of the most important considerations in numerical differentiation. While large step sizes can increase simulation speed, steps that are too large can create instabilities. On the other hand, if the steps are too small, numerical errors can dominate and the derivatives are of no use. Accordingly, the question that arises in discovering PDEs is the accuracy of numerical differentiation that has been used for training.

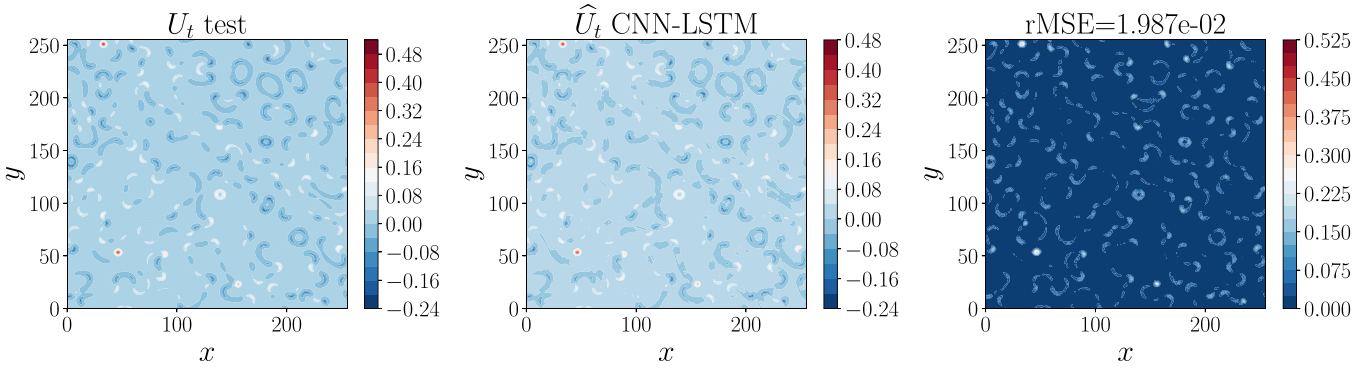
Unlike an MLP, CNN-LSTM is capable of automatically learning time derivatives from coarse-scale values. Using a combination of convolutional layers with other network structures for data-driven differential equations is an active field of research (see, for example, Refs. [50,51]). CNNs are widely used for image classification, and there have been several breakthroughs in image recognition with performance close to that of humans [52]. The CNN architecture can progressively extract higher level representations (color, shape, topology, etc.) of an input feature (image) and learn the dependency of the output (mostly a single class label) on those representations. The convolution operation sweeps a filter across the entire input field and extracts the global features and local (pixel-to-pixel) variations. The convolutional layer can be considered as an efficient implementation of the convolution operator, hence representing approximations of (potentially high order) derivatives of a scalar field. The relationship between the convolution-differentiation and derivatives-order of filters has been discussed in detail by Cai and Dong [53,54].

TABLE III. Details of the CNN-LSTM network used for field equation discovery. The network is trained for 2000 epochs with a learning rate 10^{-3}. n_t snapshots for each dataset are randomly split with 60:20:20 ratio for training, validation, and test (the training set has n_k snapshots with size $0.6n_t$ for each dataset).

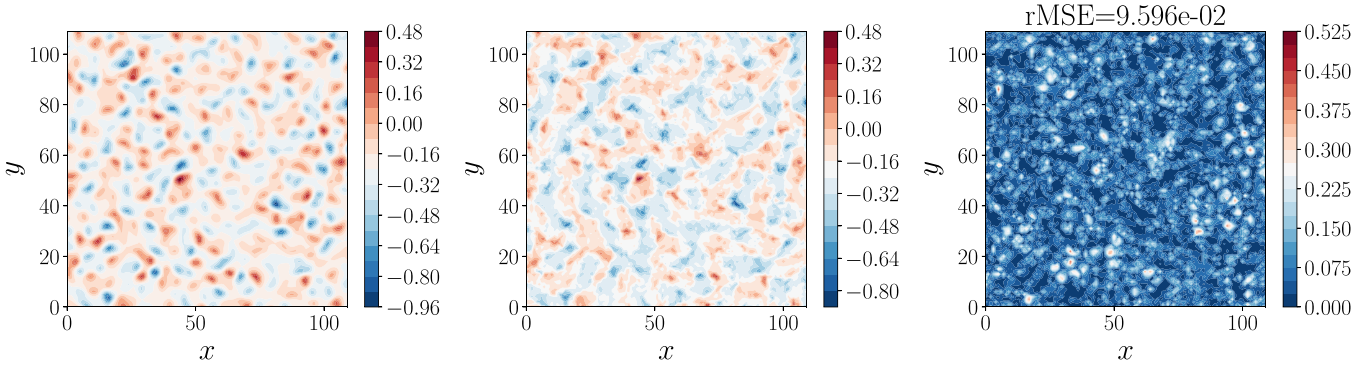
Layers	Structure	Units	Filter	Kernel size	Pool size	Activation
0	Input	1				
1	Conv1D		64	3		ReLU
2	Time distributed					
3	MaxPooling1D				2	
4	Time distributed					
5	LSTM	80				ReLU
6	Dense	10				ReLU
7	Dense	5				ReLU
8	Dense (output)	1				Linear

A schematic diagram of the proposed CNN-LSTM architecture is shown in Fig. 2(b). The architecture consists of two subnetworks: (i) A CNN subnetwork, including one-dimensional convolution and maxpooling layers for feature extraction from input data, and (ii) a LSTM subnetwork including sequential layers followed by one LSTM layer and two dense layers with ReLU activation. We feed the CNN-LSTM network (see Table III) with the five local coarse-scale variables, $U(t_k, x_{i-1}, y_j)$, $U(t_k, x_i, y_j)$, $U(t_k, x_{i+1}, y_j)$, $U(t_k, x_i, y_{j-1})$, $U(t_k, x_i, y_{j+1})$ tuple for $1 \leqslant k \leqslant n_k$, $1 \leqslant i \leqslant n_x$, and $1 \leqslant j \leqslant n_y$ for all phase-field models. Although we have second-, fourth-, and sixth-order equations, CNN-LSTM training can be performed with only five local points (mentioned above). Due to CNN's ability to extract spatial features from inputs, CNN-LSTM shows that increasing training of local points has no impact on performance. Here n_k correspond to 60% of the original datasets, which is randomly selected for training. These coarse-scale variables at each grid point are fed into the CNN subnetwork, and the output of the convolutional layer passes through the LSTM layer followed by a dense layer to provide the output. The output of the network is the single neuron approximating $U_t(t,x,y)$ at each grid point.

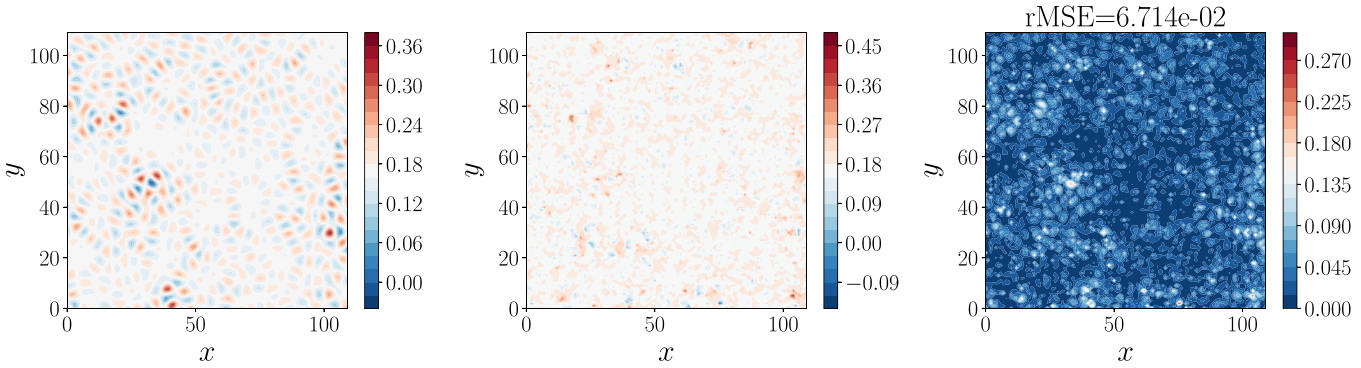
The LSTM network consists of a cell state that is the core concept of LSTM networks and memory blocks. Each block is composed of gates that can make decisions about which information passes through the cell state and which information can be removed. There are three kinds of gates: (i) input, (ii) output, and (iii) forget gate. Each memory block in an LSTM architecture has an input and an output gate which control information coming into the memory cell and information going out to the rest of the network, respectively. In addition, an LSTM architecture has a forget gate that contains an activation function and allows the LSTM to keep or forget information. Information from the previous hidden state and information from the current input is passed through the activation function. The output of each gate is a value between 0 (block the information) and 1 (pass the information) [55,56].



(a) Allen–Cahn model



(b) Cahn–Hilliard model



(c) PFC model

FIG. 4. CNN-LSTM predictions for (a) the Allen-Cahn Eq. (5) at $t = 20$, (b) the Cahn-Hilliard Eq. (6) at $t = 20$, and (c) the PFC Eq. (8) at $t = 100$. Actual and learned time derivatives U_t and $\widehat{U}_t$ are shown in the left two panels. The difference between the predicted and the actual time derivatives as well as rMSE are presented in the right panel.

In our setup, the network consists of a convolutional layer (Conv1D) with 64 filters before a pooling layer, kernel size 3 followed by an LSTM layer with 80 neurons. There are two dense layers (fully connected) with 10 and 5 neurons each. Data have been reshaped to one dimension before training the network. The performance of the trained CNN-LSTM network is shown in Fig. 4. In the left two panels, contours of U_t and $\widehat{U}_t$ for the test sets and the corresponding predictions by CNN-LSTM are compared. The snapshots of the differences between the true value and the predictions of the CNN-LSTM at time steps $t = 100$ are shown in the right panels, where rMSE is also reported for each phase-field model. A slight error can be observed along sharp boundaries in some isolated grid points, indicating CNN-LSTM does not identify smaller features in the field. Compared to true phase-field simulations, the spatial gradient of phase concentration is not as sharp. Similar behavior has been reported in other studies; see, e.g., Ref. [57]. The prediction errors from CNN-LSTM remain

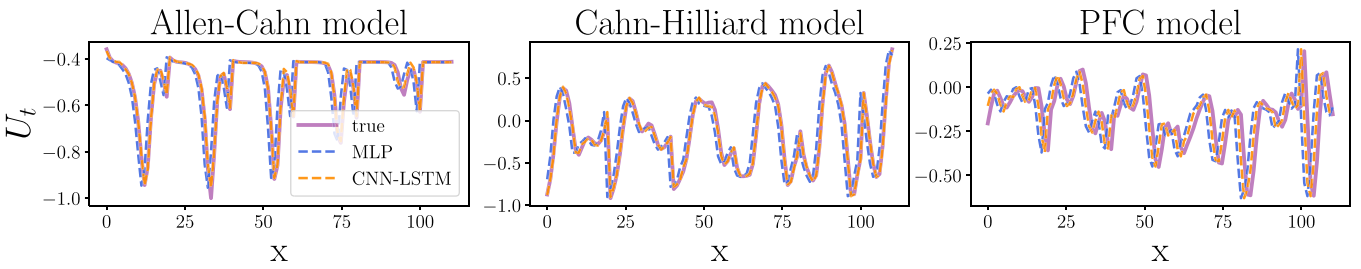


FIG. 5. Comparisons between MLP and CNN-LSTM performance using the phase-field Eqs. (5), (6), and (8). In each plot, the horizontal axis indicates $x = nx$ and the vertical axis represents the time derivative U_t and $\widehat{U}_t$ predicted by MLP and CNN-LSTM for each phase-field model. Two MLP and CNN-LSTM networks are trained and tested on the same datasets.

unchanged, and an rMSE $\sim 10^{-2}$ is obtained for all three models.

C. Hyperparameter study

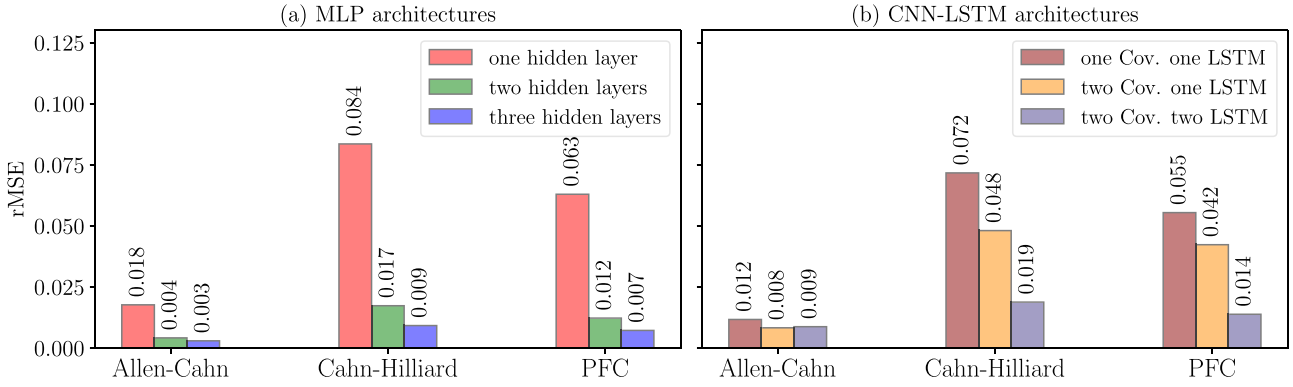
A comparison of regression results over the selected prediction period obtained by MLP and CNN-LSTM is shown in Fig. 5. One can clearly see the ability of both MLP and CNN-LSTM to accurately reproduce the original data and make predictions of the phase-field models.

We used the coefficient of determination, R^2, to compare the performance of the networks,

$$R^2 = 1 - \frac{\sum_{i=1}^{n_x \times n_y} \left(U_t^i - \widehat{U}_t^i\right)^2}{\sum_{i=1}^{n_x \times n_y} \left(U_t^i - \overline{U_t}\right)^2}, \tag{12}$$

where $\overline{U_t}$ is the mean value of the time derivative for a single snapshot. Root mean squares and R^2 scores can be affected by different hyperparameters such as learning rate, number of training epochs, and network depth and width. Here, we study the effect of adding/removing MLP and convolutional layers, while all the other parameters are fixed.

Figure 6(a) shows the effect of adding layers to our MLP architecture. An MLP network with one layer consisting of 64 hidden neurons is expanded to a network with two and three layers with 128, 64 and 256, 128, 64 hidden neurons, respectively. It can be seen that adding hidden layers reduces the rMSE and increases the performance of prediction. Figure 6(b) presents the effect of adding CNN and LSTM layers to the CNN-LSTM. Here, we use a single LSTM layer with two configurations for CNN layers: (i) a single CNN layer with output filters of size 64, (ii) two CNN layers with 128,



(c) R^2 for MLP and CNN-LSTM

	Allen-Cahn	Cahn-Hilliard	PFC
MLP with one hidden layer	0.96	0.85	0.51
MLP with two hidden layers	0.99	0.99	0.98
MLP with three hidden layers	0.99	0.99	0.99
CNN-LSTM with one Cov. one LSTM	0.98	0.99	0.95
CNN-LSTM with two Cov. one LSTM	0.99	0.99	0.98
CNN-LSTM with two Cov. two LSTM	0.98	0.99	0.97

FIG. 6. Effect of changing MLP and CNN-LSTM architectures on rMSE and R^2. (a) rMSE values obtained by three different MLP architectures. (b) rMSE values obtained by three different CNN-LSTM architectures. (c) R^2 values for the test set calculated by Eq. (12) reported for three different MLP and CNN-LSTM architectures.

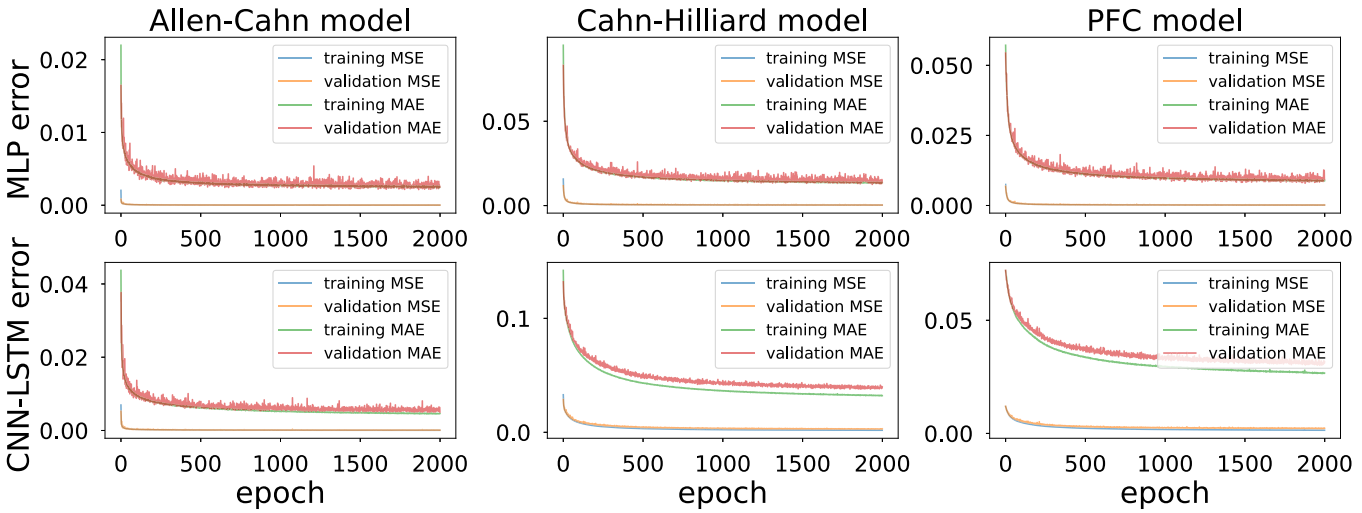


FIG. 7. Trace of MSE and MAE [see Eqs. (11) and (13)] errors for MLP and CNN-LSTM networks. The blue and green lines represent the errors on the training sets as a function of epochs, and the orange and red lines correspond to the errors on the validation sets. Learning curves show that the training and validation curves are very similar for both MSE and MAE errors and they decrease to a point of stability.

64 output shape as well as two LSTM layers with 128, 64 neurons followed by two CNN layers with 128, 64 output sizes. Adding convolutional layers increases the performance. However, MLP networks are more sensitive to the choice of architecture than the CNN-LSTM networks. Moreover, the computational cost of training a multilayer CNN-LSTM is huge compared to a single layer and should be taken into account for large-scale data. It can be roughly concluded that the optimal number of LSTM and CNN layers is 1 in our CNN-LSTM network. Conversely, the R^2 values show less sensitivity to the structural changes in our proposed neural networks, particularly in the CNN-LSTM network [see Fig. 6(c)].

To further study the dynamics of the optimization process (training models), the MSE and mean absolute error (MAE) as a function of epochs are given in Fig. 7. The MAE is the difference between the original and predicted values. This is calculated by averaging the absolute difference over the dataset, and it is expressed as

$$\mathrm{MAE} = \frac{1}{n_x \times n_y} \sum_{i=1}^{n_x \times n_y} \left| U_t^i - \widehat{U}_t^i \right|. \tag{13}$$

To achieve sufficiently small error, we trained networks for 2000 epochs with a batch size of 64. However, using approximately 500 epochs (e.g., early-stopping [58]) seems adequate for achieving optimal results, particularly for the Allen-Cahn and the Cahn-Hilliard models. Since training CNN-LSTM networks is computationally expensive, using smart early-stopping approaches can help in cases of large data PDE learning tasks.

IV. DATA-DRIVEN PDEs WITHOUT SPATIAL DERIVATIVES DICTIONARY

In this section, we reformulate the problem of learning PDEs as black-box supervised learning tasks, using convolutional neural network architecture where there is no selection of spatial derivatives and the field U is the only input to our deep learning model. The mathematical representation of a data-driven PDE learning task with the CNN is

$$U_t(t, x, y) = F_{\mathrm{CNN}}(U(t, x, y)),$$
$$\times\, F_{\mathrm{CNN}} : \mathbb{R}^{n_x \times n_y} \to \mathbb{R}^{n_x \times n_y}, \tag{14}$$

where n_x and n_y are the number of grid points in the x- and y-directions, respectively. We use U from our phase-field model simulations to train the CNN. After successful training of the

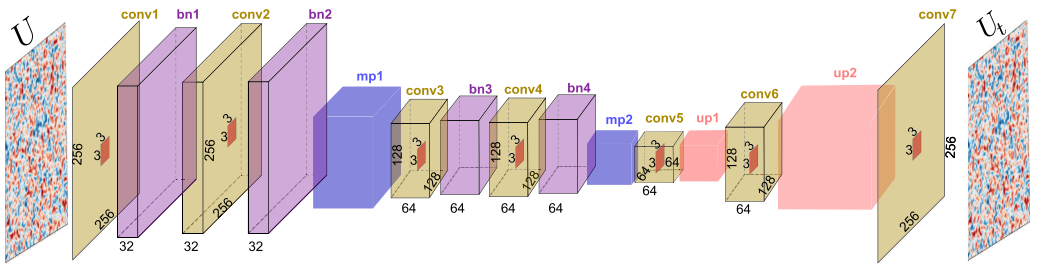


FIG. 8. The proposed CNN architecture. The input and output of the CNN are the U and U_t fields, respectively. Input passes through several convolution (`conv`), batch normalization (`bn`), max pooling (`mp`), and up-sampling (`up`) layers. All the relevant parameters of the network architecture are described in Sec. IV A.

TABLE IV. A CNN network used for discovering field equations without spatial derivatives. This network is trained for 20,000 epochs with the ADAM optimizer [46] with learning rate 10^{-4} and MAE loss function, Eq. (13). A random sampling of 80% of snapshots (n_t) was used as the training set. Validation was performed on 10% and testing was performed on the remainder.

Layers	Structure	Filter	Kernel size	Pool size	Activation	Padding
1	Conv2D	32	(3,3)		ReLU	same
2	BatchNormalization					
3	Conv2D	32	(3,3)		ReLU	same
4	BatchNormalization					
5	MaxPooling2D			(2,2)		valid
6	Conv2D	64	(3,3)		ReLU	same
7	BatchNormalization					
8	Conv2D	64	(3,3)		ReLU	same
9	BatchNormalization					
10	MaxPooling2D			(2,2)		valid
11	Conv2D	64	(3,3)		ReLU	same
12	UpSampling2D		(2,2)			
13	Conv2D	32	(3,3)		ReLU	same
14	UpSampling2D		(2,2)			
15	Conv2D (output)	1	(3,3)		linear	same

CNN networks, arbitrary initial conditions were chosen for the field U and it was evolved in time by solving $U_t = F_{\mathrm{CNN}}(U)$ numerically at each grid point.

A. Convolutional neural network architecture

The CNN network architecture is illustrated in Fig. 8. The full details of the mathematical operations and functionality of each layer are beyond the scope of this paper and can be found in reviews on CNNs such as the one by Rawar and Wang [59]. For particular applications where the desired outputs include localization (a class label is assigned to each pixel), a specific CNN architecture called "U-net" has been proposed [60]. Since in most engineering and physics applications the time evolution of the scalar field depends on the local spatial derivatives, the U-net architecture is a reasonable candidate for such a learning task. The U-net-inspired network has also been successfully used in subgrid flame surface density estimation for premixed turbulent combustion modeling [61].

The CNN structure proposed here, similar to the U-net [60,61], resembles the encoding-decoding (autoencoding)

TABLE V. R^2 values for CNN performance of predicting U_t for test (unseen) data.

2D model	Allen-Cahn Eq. (5)	Cahn-Hilliard Eq. (6)	PFC Eq. (8)
R^2	0.98	0.975	0.985

networks. The scalar field discretized on $n_x \times n_y$ grid points was fed as the input to the network. In the contracting path, two convolutional layers (`conv1, conv2` in Fig. 8) with 32 filters each followed by ReLU and batch normalization (`bn1, bn2`) were applied. The kernel size was 3×3 for all the convolutional layers. After the `bn2` layer, the 2D max pooling operation (`mp1`) with zero stride (for dimensionality reduction purposes) was applied. The pool size for all the max pooling layers was 2×2. The same substructure is repeated with 64 filters (`conv3, bn3, conv4, bn4`) up to the bottleneck unit (output of `mp2`). The expansion path consists of two convolutional layers (`conv5,conv6`) with ReLU units, each followed by an upsampling layer (`up1, up2`) with the expansion factor of (2,2). Finally, at the last convolutional layer (`conv7`), a linear activation function was used with a filter of size one resulting in an output of shape $n_x \times n_y$. All the parameters used for the network are summarized in Table IV. The ADAM optimization was applied to find the parameters of the network, where the cost function is the mean absolute error between the network output and U_t from the training set. In total, our CNN network consists of 121 057 trainable parameters.

B. CNN performance for learning PDEs

The phase-field models presented in Sec. II were used to evaluate the performance of the CNN network. For each model, the total of n_t two-dimensional U and U_t fields were used and randomly split 80:10:10 into training, validation, and test sets, respectively. The U and U_t fields from training sets were provided as an input and output to the CNN. All models were trained for 20 000 epochs, and the performance of the network to recover the U_t (learning the RHS of a PDE) on the test sets is summarized in Table V in terms of R^2 values. The values indicate that all the trained models performed outstandingly in recovering the PDEs. The contours of U_t and the prediction of the CNN [for the Cahn-Hilliard model, Eq. (6)] are compared in Fig. 9 in the left two panels. The figure shows a qualitative agreement between the original and the data-driven models. In addition, the true and CNN predicted

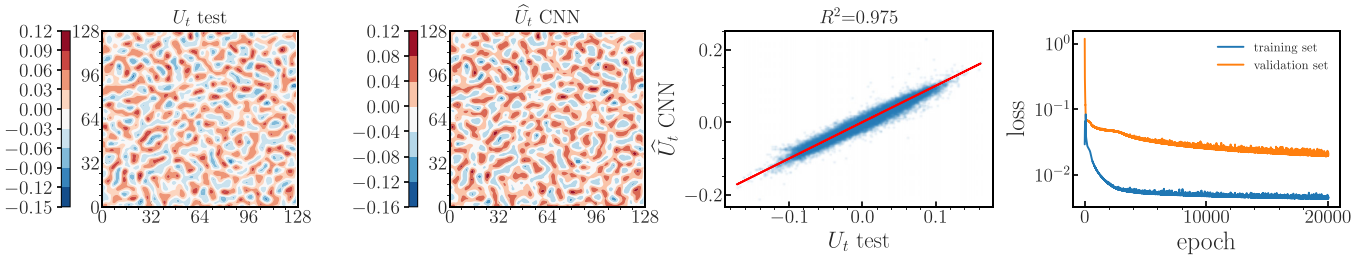


FIG. 9. Results using the CNN model trained on the Cahn-Hilliard [Eq. (6)] dataset. The left two panels show the color map of the U_t test set and the corresponding prediction by the CNN. The $\widehat{U}_t$ predictions for all test data as well as the traces of the loss functions are given in the right two panels.

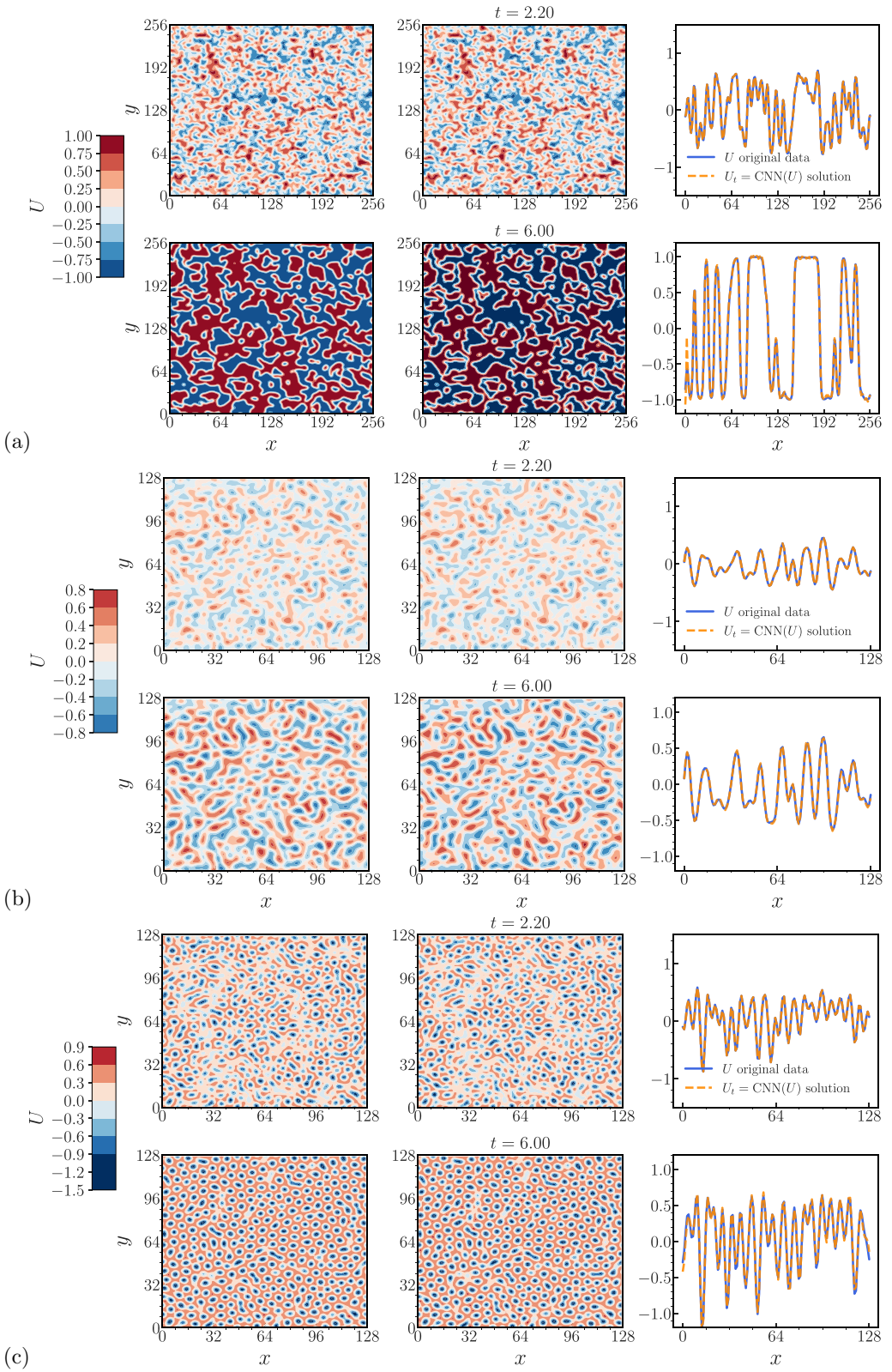


FIG. 10. Time integration results of the PDEs learned by CNN for (a) Allen-Cahn [Eq. (5)], (b) Cahn-Hilliard [Eq. (6)], and (c) PFC [Eq. (8)] at $t = 2.2$ and 6. Left panels: U field for original data. Middle panels: U field from simulations of the learned PDEs. Right panel: U values along the centerline $y = n_y/2$ for the original PDEs (solid lines) and from simulations of the learned PDEs (dashed lines).

values of U_t for all the grid points for the test set are compared in the third panel (correlation plot). The data lie mostly on the diagonal line indicating good performance. The traces of the loss/cost functions during the training phase are also given in the rightmost panel of Fig. 9. Similar results/plots were obtained for both the Allen-Cahn [Eq. (5)] and the PFC [Eq. (8)] models (data not shown here).

C. Simulation of data-driven PDEs

In this section, the potential of the proposed method to predict the field U in time and space based on a given initial condition U_0 is presented. For all three phase-field models (Sec. II), we are interested in solving a set of PDEs of the form

$$\frac{\partial U(t,x,y)}{\partial t} = F_{\mathrm{CNN}}(U(t,x,y)),$$
$$U(0,x,y) = U_0; \quad \text{initial condition}, \tag{15}$$

where the right-hand side is the output (prediction) of the trained CNN networks. In the following, we used the U fields at $t = 2$ (simulation time) as the initial condition (U_0) for all three models. The U field had $n_x \times n_y = 128 \times 128$ real values for the Cahn-Hilliard [Eq. (6)] and the PFC [Eq. (8)] models, and 256×256 for the Allen-Cahn model [Eq. (5)]. The different sizes were used to test if there is any size dependence. At each time t, the U_t values for every grid point were determined from our trained CNN models, and $n_x \times n_y$ ODEs (ordinary differential equations) were solved using the (stiff) integrator. We used the `scipy` Adams/BDF method with automatic stiffness detection and switching for time integration [62,63]. Those ODEs were solved up to $t = 6$ in our benchmark datasets.

Figure 10 shows the solutions of the original and the data-driven PDEs. The color maps for U are given for qualitative comparison as well as the U values along the centerline $y = n_y/2$ for two snapshots at times $t = 2.2$ and 6. The results in Fig. 10 showed that the data-driven PDEs learned by CNN approximate the original dynamics in both a quantitative and qualitative manner.

Finally, we would like to emphasize the following points: (i) The explicit forms of the data-driven PDEs are not known and there is no obvious relation between the functional form of the original and the learned PDEs. Therefore, unlike with the phase-field models, there is no guarantee of the existence and uniqueness for the learned PDEs. (ii) There are some isolated points in which the U_t predicted values are different from the original models. This discrepancy propagates in time and space and can lead to finite-time blow-up in simulations. This is a known issue in (almost all) machine-learning algorithms for time series forecasting where there is no periodicity in time [64,65]. In the case of no underlying periodicity, it may occur that the system trajectories do not span the whole phase space properly. Therefore, the observations do not properly represent the possible outcomes of the system, and hence models trained with those data may not be adequate. Such a situation may limit the applicability of the approach to short simulation times.

V. CONCLUSION

We have presented several data-driven methodologies for discovering PDEs from phase-field dynamics. The well-known Allen-Cahn, Cahn-Hilliard, and phase-field crystal models were used as the test cases to predict the underlying equations of motion.

First, we provided an MLP architecture to learn the PDEs where the spatial derivatives are explicitly computed by finite differences. Second, CNN-LSTMs were employed to learn the governing PDEs from coarse-scale local values. Third, we proposed a special CNN architecture for cases in which there is no information about spatial dependence. In addition, using numerical integration, we showed how the learned PDEs can be used to predict coarse-scale variables as a function of time and space, starting from given initial conditions. The evolution of the learned and original PDEs showed excellent agreement. We emphasize that all of the above algorithms yield a black-box-type discovery of PDEs with no obvious connection to the functional form of the physical models.

In general, MLP networks are extremely flexible with data, and PDEs can be learned from various types of data using these networks. More specifically, they can be used to learn a mapping from a coarse field and its spatial derivatives as the inputs. However, the performance of an MLP network is greatly affected by the choice of architecture as shown in Sec. III C. Along with approximating derivatives, we need to know the derivatives' orders, as that is required to train an MLP network.

In CNN networks, however, spatial derivatives are not required, and thus a CNN can be thought of as a finite-difference method capable of estimating derivatives in its first convolution layer. Moreover, one major advantage in using CNNs is their capability to extract spatial features from inputs. Since LSTMs pass only time information to the layers and keep the missing spatial information from the previous steps, a combination of CNNs and LSTMs can be applied more generally on data with spatial relationships, and, in the current case, to learn phase-field models. In spite of these advantages, CNN networks are memory-intensive and require a large amount of data and several iterations in order to be trained effectively, and LSTMs are computationally expensive. Despite the above limitations, we believe that the techniques introduced here offer approaches that are both general and systematic, and they provide a basis for future developments.

The study will be extended in two directions in the future: (a) predicting two-dimensional noisy phase-field models, and (b) predicting three-dimensional phase-field models. As a result of a limited amount of memory, it becomes increasingly challenging to train networks efficiently in the second scenario. As a consequence, we will use frameworks that can handle large datasets.

ACKNOWLEDGMENTS

M.K. was partially supported by NIH Grant No. GM133777. M.K. thanks the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Canada Research Chairs Program. Computing facilities were provided by SHARCNET [66] and Compute Canada [67].

[1] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. (USA) **115**, 8505 (2018).

[2] I. Lagaris, A. Likas, and D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. **9**, 987 (1998).

[3] J. Sirignano and K. Spiliopoulos, DGM: A deep learning algorithm for solving partial differential equations, J. Comput. Phys. **375**, 1339 (2018).

[4] C. Huré, H. Pham, and X. Warin, Some machine learning schemes for high-dimensional nonlinear PDEs, Math. Comput. **89**, 1547 (2020).

[5] W. E, J. Han, and A. Jentzen, Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning, Nonlinearity **35**, 278 (2022).

[6] R. Ranade, C. Hill, and J. Pathak, DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization, Comput. Methods Appl. Mech. Eng. **378**, 113722 (2021).

[7] M. Raissi, P. Perdikaris, and 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**, 686 (2019).

[8] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, Physics-informed machine learning, Nat. Rev. Phys. **3**, 422 (2021).

[9] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, Nat. Mach. Intell. **3**, 218 (2021).

[10] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, SIAM Rev. **63**, 208 (2021).

[11] S. L. Brunton, J. L. Proctor, and J. N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proc. Natl. Acad. Sci. (USA) **113**, 3932 (2016).

[12] H. Schaeffer, Learning partial differential equations via data discovery and sparse optimization, Proc. R. Soc. Math. Phys. Eng. Sci. **473**, 20160446 (2017).

[13] S. Lee, M. Kooshkbaghi, K. Spiliotis, C. I. Siettos, and I. G. Kevrekidis, Coarse-scale PDEs from fine-scale observations via machine learning, Chaos Interdiscip. J. Nonlin. Sci. **30**, 013141 (2020).

[14] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, and P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proc. R. Soc. Math. Phys. Eng. Sci. **474**, 20170844 (2018).

[15] F. A. Gers, D. Eck, and J. Schmidhuber, Applying LSTM to time series predictable through time-window approaches, in *Neural Nets WIRN Vietri-01* (Springer, Vienna, 2002), pp. 193–200.

[16] J. del Águila Ferrandis, M. S. Triantafyllou, C. Chryssostomidis, and G. E. Karniadakis, Learning functionals via LSTM neural networks for predicting vessel dynamics in extreme sea states, Proc. R. Soc. A **477**, 20190897 (2021).

[17] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Learning data-driven discretizations for partial differential equations, Proc. Natl. Acad. Sci. (USA) **116**, 15344 (2019).

[18] C. Theodoropoulos and E. Luna-Ortiz, A reduced input/output dynamic optimisation method for macroscopic and microscopic systems, in *Model Reduction and Coarse-graining Approaches for Multiscale Phenomena* (Springer, Berlin, 2006), pp. 535–560.

[19] T. N. Thiem, M. Kooshkbaghi, T. Bertalan, C. R. Laing, and I. G. Kevrekidis, Emergent spaces for coupled oscillators, Front. Comput. Neurosci. **14**, 36 (2020).

[20] Y. Kim, Convolutional neural networks for sentence classification, in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language* (Association for Computational Linguistics, Doha, Qatar, 2014), pp. 1746–1751.

[21] D. Qin, J. Yu, G. Zou, R. Yong, Q. Zhao, and B. Zhang, A novel combined prediction scheme based on CNN and LSTM for urban PM 2.5 concentration, IEEE Access **7**, 20050 (2019).

[22] S. M. Allen and J. W. Cahn, Ground state structures in ordered binary alloys with second neighbor interactions, Acta Metall. **20**, 423 (1972).

[23] J. W. Cahn and J. E. Hilliard, Free energy of a nonuniform system. I. Interfacial free energy, J. Chem. Phys. **28**, 258 (1958).

[24] K. R. Elder, M. Katakowski, M. Haataja, and M. Grant, Modeling Elasticity in Crystal Growth, Phys. Rev. Lett. **88**, 245701 (2002).

[25] K. R. Elder and M. Grant, Modeling elastic and plastic deformations in nonequilibrium processing using phase field crystals, Phys. Rev. E **70**, 051605 (2004).

[26] S. A. Silber and M. Karttunen, SymPhas–General purpose software for phase-field, phase-field crystal, and reaction-diffusion simulations, Adv. Theory Simul. **5**, 2100351 (2022).

[27] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. J. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Józefowicz, L. Kaiser, M. Kudlur, J. Levenberg *et al.*, TensorFlow: Large-scale machine learning on heterogeneous distributed systems, arXiv:1603.04467 (2016).

[28] N. Provatas and K. Elder, *Phase-Field Methods in Materials Science and Engineering* (Wiley, Weiheim, Germany, 2010), pp. 57–222.

[29] A. M. Turing, The chemical basis of morphogenesis, Bull. Math. Biol. **52**, 153 (1990).

[30] T. Leppänen, M. Karttunen, K. Kaski, R. A. Barrio, and L. Zhang, A new dimension to Turing patterns, Physica D **168-169**, 35 (2002).

[31] P. Gray and S. Scott, Sustained oscillations and other exotic patterns of behavior in isothermal reactions, J. Phys. Chem. **89**, 22 (1985).

[32] B. Grossmann, K. R. Elder, M. Grant, and J. M. Kosterlitz, Directional Solidification in Two and Three Dimensions, Phys. Rev. Lett. **71**, 3323 (1993).

[33] W. J. Boettinger, J. A. Warren, C. Beckermann, and A. Karma, Phase-field simulation of solidification, Annu. Rev. Mater. Sci. **32**, 163 (2002).

[34] B. Nestler, H. Garcke, and B. Stinner, Multicomponent alloy solidification: Phase-field modeling and simulations, Phys. Rev. E **71**, 041609 (2005).

[35] V. Heinonen, C. V. Achim, J. M. Kosterlitz, S.-C. Ying, J. Lowengrub, and T. Ala-Nissila, Consistent Hydrodynamics for Phase Field Crystals, Phys. Rev. Lett. **116**, 024303 (2016).

[36] E. Alster, K. R. Elder, and P. W. Voorhees, Displacive phase-field crystal model, Phys. Rev. Mater. **4**, 013802 (2020).

[37] N. Provatas, J. Dantzig, B. Athreya, P. Chan, P. Stefanovic, N. Goldenfeld, and K. Elder, Using the phase-field crystal method in the multi-scale modeling of microstructure evolution, JOM **59**, 83 (2007).

[38] N. Faghihi, S. Mkhonta, K. Elder, and M. Grant, Phase-field crystal for an antiferromagnet with elastic interactions, Phys. Rev. E **100**, 022128 (2019).

[39] I. S. Aranson, V. A. Kalatsky, and V. M. Vinokur, Continuum Field Description of Crack Propagation, Phys. Rev. Lett. **85**, 118 (2000).

[40] R. Spatschek, E. Brener, and A. Karma, Phase field modeling of crack propagation, Philos. Mag. **91**, 75 (2011).

[41] P. C. Hohenberg and B. I. Halperin, Theory of dynamic critical phenomena, Rev. Mod. Phys. **49**, 435 (1977).

[42] D.-H. Yeon, Z.-F. Huang, K. Elder, and K. Thornton, Density-amplitude formulation of the phase-field crystal model for two-phase coexistence in two and three dimensions, Philos. Mag. **90**, 237 (2010).

[43] L. Q. Chen and J. Shen, Applications of semi-implicit Fourier-spectral method to phase field equations, Comput. Phys. Commun. **108**, 147 (1998).

[44] ISO, *ISO/IEC 14882:2017: Programming languages—C++*, 5th ed. (International Organization for Standardization, Geneva, Switzerland, 2017), p. 1605.

[45] T. Barlow, Feed-forward neural networks for secondary structure prediction, J. Mol. Graph. **13**, 175 (1995).

[46] K. Diederik, B. Jimmy *et al.*, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).

[47] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, Understanding deep neural networks with rectified linear units, arXiv:1611.01491 (2016).

[48] W. L. Ziegler, Computational method to compute the derivative and antiderivative; with concern for terminating a converging iterative process and considering accuracy, roundoff error, approximation, and extrapolation, Math. Model. **8**, 77 (1987).

[49] W. H. Press and S. A. Teukolsky, Numerical calculation of derivatives, Comput. Phys. **5**, 68 (1991).

[50] H. Arbabi and I. G. Kevrekidis, Particles to partial differential equations parsimoniously, Chaos Interdiscip. J. Nonlin. Sci. **31**, 033137 (2021).

[51] H. Arbabi, J. E. Bunder, G. Samaey, A. J. Roberts, and I. G. Kevrekidis, Linking machine learning with multiscale numerics: Data-driven discovery of homogenized equations, JOM **72**, 4444 (2020).

[52] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, Going deeper with convolutions, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Boston, 2015), pp. 1–9.

[53] J.-F. Cai, B. Dong, S. Osher, and Z. Shen, Image restoration: Total variation, wavelet frames, and beyond, J. Am. Math. Soc. **25**, 1033 (2012).

[54] B. Dong, Q. Jiang, and Z. Shen, Image restoration: Wavelet frame shrinkage, nonlinear evolution pdes, and beyond, Multiscale Model. Simul. **15**, 606 (2017).

[55] S. Hochreiter and J. Schmidhuber, Long short-term memory, Neural Comput. **9**, 1735 (1997).

[56] A. Graves, Long short-term memory, in *Supervised Sequence Labelling with Recurrent Neural Networks* (Springer, Berlin, 2012), pp. 37–45.

[57] V. Oommen, K. Shukla, S. Goswami, R. Dingreville, and G. E. Karniadakis, Learning two-phase microstructure evolution using neural operators and autoencoder architectures, arXiv:2204.07230 (2022).

[58] L. Prechelt, Early stopping-but when?, in *Neural Networks: Tricks of the Trade* (Springer, Berlin, 1998), pp. 55–69.

[59] W. Rawat and Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural Comput. **29**, 2352 (2017).

[60] O. Ronneberger, P. Fischer, and T. Brox, U-net: Convolutional networks for biomedical image segmentation, in *International Conference on Medical Image Computing and Computer-Assisted Intervention* (Springer, Cham, 2015), pp. 234–241.

[61] C. J. Lapeyre, A. Misdariis, N. Cazard, D. Veynante, and T. Poinsot, Training convolutional neural networks to estimate turbulent sub-grid scale reaction rates, Combust. Flame **203**, 255 (2019).

[62] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, *et al.*, SciPy 1.0: Fundamental algorithms for scientific computing in python, Nat. Methods **17**, 261 (2020).

[63] A. C. Hindmarsh, ODEPACK, a systematized collection of ODE solvers, Sci. Comput. **1**, 55 (1983).

[64] R. Rico-Martinez, J. Anderson, and I. Kevrekidis, Continuous-time nonlinear signal processing: A neural network based approach for gray box identification, in *Proceedings of IEEE Workshop on Neural Networks for Signal Processing* (IEEE, Ermioni, Greece, 1994), pp. 596–605.

[65] J. Fan and Q. Yao, *Nonlinear Time Series: Nonparametric and Parametric Methods* (Springer, New York, 2003), pp. 1–553.

[66] www.sharcnet.ca.

[67] www.computecanada.ca.