mechanoChemML: A software library for machine learning in computational materials physics

X. Zhang[1], G.H. Teichert[1], Z. Wang[1], M. Duschenes[2], S. Srivastava[1],
E. Livingston[1], J. Holber[2], M. Faghih Shojaei[1], A. Sundararajan[1] and K. Garikipati[1,2,3,4*]
[1]Department of Mechanical Engineering, University of Michigan, United States
[2]Applied Physics Program, University of Michigan, United States
[3]Department of Mathematics, University of Michigan, United States
[4]Michigan Institute for Computational Discovery & Engineering, University of Michigan, United States

Abstract

We present mechanoChemML, a machine learning software library for computational materials physics. mechanoChemML is designed to function as an interface between platforms that are widely used for machine learning on one hand, and others for solution of partial differential equations-based models of physics. Of special interest here, and the focus of mechanoChemML, are applications to computational materials physics. These typically feature the coupled solution of material transport, reaction, phase transformation, mechanics, heat transport and electrochemistry. Central to the organization of mechanoChemML are machine learning workflows that arise in the context of data-driven computational materials physics. The mechanoChemML code structure is described, the machine learning workflows are laid out and their application to the solution of several problems in materials physics is outlined.

Keywords machine learning software library · machine learning workflows · computational materials physics · partial differential equation solvers · scientific software

1 Introduction

Until roughly a decade ago, the dominant theme in computational materials physics was the forward solution of a very wide array of problems by using methods that ranged from electronic structure calculations, through molecular and statistical mechanics computations, to partial differential equations (PDEs) describing continuum phenomena. Data, while used, was rarely a part of the computational workflow. Much has changed since then with the rise of of data-driven modeling and machine learning (ML) in particular. It is now routine for computations with each of the above classes of methods to ingest data, produce and employ them as the packets of communication with other simulations, or experimental platforms. Thus, Density Functional Theory (DFT), long the workhorse of electronic structure calculations for materials applications, may now be based on full field *ab initio* solutions from Configuration Interaction methods to inform or learn the exchange correlation functional [1]. Formation energy data computed by DFT parameterize interatomic potentials for molecular dynamics [2] or, through cluster Hamiltonians, drive Monte Carlo (MC)-based statistical mechanics simulations [3]. And, all of these methods generate data for continuum scale PDEs [4, 5, 6]. The data and information flow also travels down the scales as, for example, strain fields drive deformation potentials in DFT.

Notably, the interaction between the model formalisms at different levels in this outline of scale bridging in materials physics is not a trivial matter. Efficient representations are needed, often in the form of high-dimensional, or conversely, reduced dimensionality functions or functionals. For instance, MC simulations of phase or order-disorder transitions yield millions of configurations from which an $\mathcal{O}(10)$-dimensional free energy surface needs to be parameterized for continuum phase field or elasticity computations [6, 7]. This is an obvious entry point for neural network-based ML methods. The continuum simulations now commonly entail billions of degrees of freedom evolving over time scales ranging from microseconds to hours. The imperative of high-throughput design and material optimization problems needs reduced-order models ranging from homogenization methods through neural networks of various flavors [8, 9, 10, 11] and modern optimization methods to graph theoretic representations [12]. Also of very recent interest are physics-constrained machine learning techniques that ensure fidelity of fast solutions of continuum PDEs [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25].

*Corresponding author. E-mail address: krishna@umich.edu

The reliance on data ingestion, transfer, high-dimensional as well as reduced-order function representations naturally rests upon the impressive, robust and rapidly growing ecosystem for machine learning, and more broadly artificial intelligence and even all of data science. However, there are aspects specific to data-driven computational materials physics that have led to a sub-ecosystem of mathematical techniques, algorithms and frameworks. This sub-ecosystem is now spawning software that interfaces platforms such as `TensorFlow` [26], `Keras` [27], `PyTorch` [28] and their ilk with the foundational software for scientific computing such as `PETSc` [29], `Trilinos` [30] and yet others built on top of them such as `deal.ii` [31], `FEniCS` [32] and even their derivatives. The `mechanoChemML` library belongs in this sub-ecosystem.

2 Background to the methods

`mechanoChemML` leverages PDE solver libraries such as `deal.ii, FEniCS` [31, 32] [33] and others built on them, such as `mechanoChem`, also developed by the authors. These libraries present frameworks with varying degrees of abstraction for the solution of PDEs. They exploit the range of high-performance computing methods including geometry and mesh generation, vectorization, automatic differentiation, linear and nonlinear solvers, threading and parallelism, among others. With the availability of cluster computing resources, both fixed and in the cloud, this has made the large scale solution of multiphysics problems commonplace and widely accessible. As one natural next step, this ability to compute PDE solutions, almost at will, has led to thinking about enhancing the fidelity and speed of modeling as well as further exploiting the solutions. We introduce examples of such thinking that has led to `mechanoChemML`.

Phase field modeling is centered on a parabolic PDE for the evolution of a scalar or vector variable that parameterizes a free energy density function. The dissipative dynamics is governed by minimization of the free energy. In many situations, the final (near-)equilibrium state is of greater interest than the dynamics itself. This leads to alternate views of the problem as one of non-convex optimization over the free energy density landscape. Since the optimization entails computation of feasible solutions in large numbers, it suggests neural network surrogates for the free energy density in terms of reduced-order representations. Multi-fidelity learning, sensitivity analysis and a wrapper of active learning help improve the efficiency of the surrogate optimization [4].

Scale bridging computational frameworks have long been of interest in materials physics. A peculiarity of this problem is that the model description changes in a fundamental manner across scales: from electronic structure computations, typically DFT, to a variety of Monte Carlo simulations of molecular configurations to recreate the statistical mechanics, through continuum PDEs of elastic or inelastic behavior. Central quantities across these scales are the free energy, or its density and derivatives (e.g., chemical potentials and stresses). Their fundamental parameterization comes from sparse DFT computations. Their upscaling through Monte Carlo is dependent on surrogate representations via machine learning methods. The next step in bridging scales–up to the continuum–brings special considerations of integrability as well as high-dimensional representation, both of which we have found convenient to satisfy with a breed of specially developed neural networks. Monte Carlo sampling in regions of interest also naturally has invoked active learning techniques [5, 6].

At the continuum scale, a class of PDE models that combine phase field and non-convex elasticity underlies a wide range of problems that develop finely detailed microstructure. The effective mechanical response of these microstructures is obtained by homogenization methods, which are reduced-order models of a type. It has been natural to turn to deep and convolutional neural networks (DNNs and CNNs) for this problem. While the former require the insightful selection of features with the proper invariances, CNNs with an auto encoder-decoder structure afford the flexibility of learning from microstructures as images. This is the first example in which the imposition of physical constraints–of invariances and simpler symmetries–has arisen in our work [11].

The high-throughput solution of PDEs for inverse modeling, design and optimization leads to requirements of very fast solutions that are largely beyond the capability of traditional PDE solver libraries, such as those introduced at the beginning of this section. The development of PDE solvers is one of the most rapidly advancing fields in scientific machine learning. Neural networks have been a natural choice, with the added imperative of embedding the physics of the PDE as constraints. With field solutions regarded as images, CNNs in auto encoder-decoder structures are a natural choice, and the extension to uncertainty quantification made possible by Bayesian neural networks. This is the first instance in which learning from label-free data has arisen in our work [13].

Inference of PDEs from data–inverse modeling of not just parameters but of the entire set of algebraic and differential operators is feasible with the availability of extensive data, regression, and more generally, nonlinear optimization methods. Along with the development of neural network PDE solvers, system inference enjoys advantages when the variational setting of the weak form, as well as discretization structures such as finite elements and finite differences

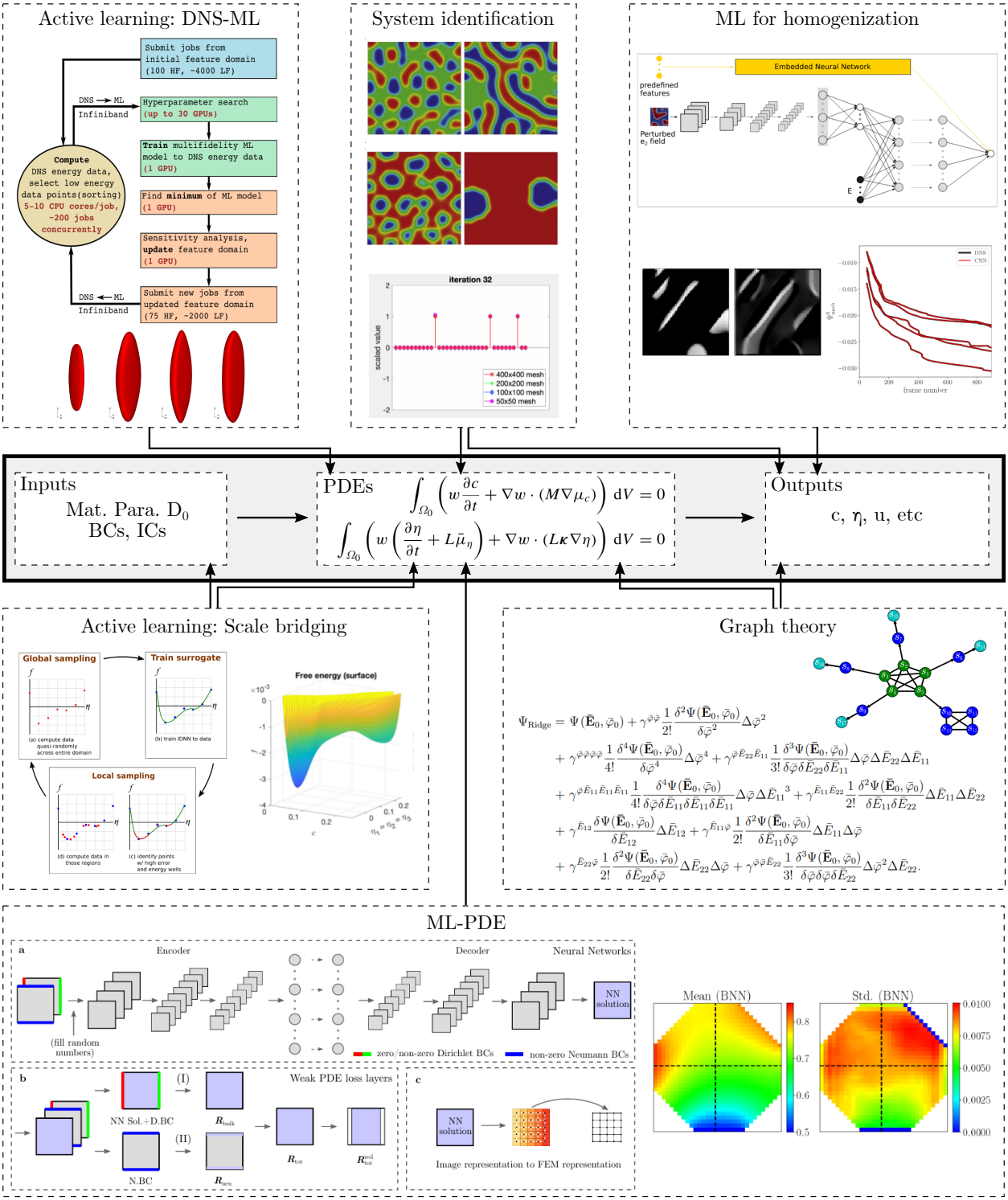


Figure 1: A schematic illustrating the range of ML methods comprising `mechanoChemML` for data-driven computational material physics.

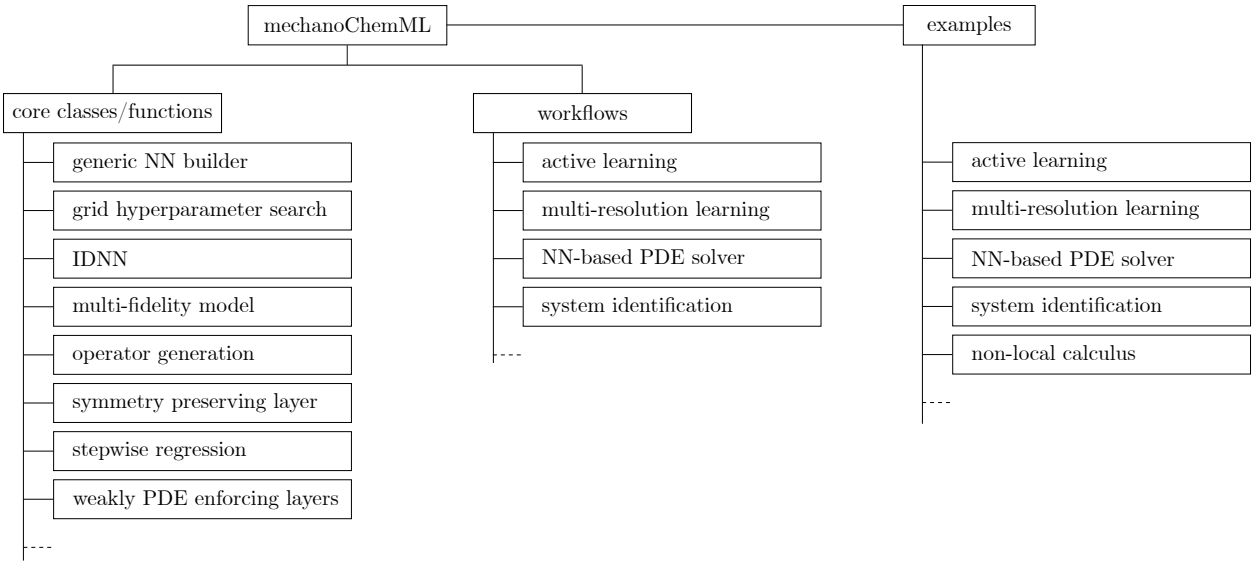


Figure 2: Illustration of the structure and components of the `mechanoChemML` library and the provided examples.

are exploited. A number of supporting methods become of relevance for this problem, including those for sparse and noisy data, and gradient optimization [34, 35, 36, 37, 38].

Finally, reduced-order modeling on a physical systems scale is feasible with large datasets of computed solution states. We have exploited a correspondence between the properties dictated by PDEs on solution states of physical systems and graph theory [39]. In addition to representation and analysis on the systems scale by exploiting the machinery of graph theory, this opens the door to reduced-order modeling by exploiting a nonlocal calculus [12]. It leverages methods developed for system inference, mainly stepwise regression to choose the reduced-order model.

`mechanoChemML` addresses the above ecosystem of machine learning techniques, as illustrated in Fig. 1. Section 3 discusses the structure of the library. Some of the above scientific applications are presented as examples using the `mechanoChemML` library in Section 4. These two sections are the mainstay of the paper. Installation, documentation, and contributing is discussed in Section 5, and closing remarks appear in Section 6.

3 Structure of the `mechanoChemML` library

The `mechanoChemML` library can be divided into two parts: the machine learning (ML) class library and example workflows (see Figure 2). Scientific examples using the `mechanoChemML` library are discussed in Section 4.

3.1 The machine learning (ML) class library

The ML class library is a collection of classes and functions that are related to neural networks or other machine learning methods and are useful in data-driven modeling of mechanochemistry but do not already exist in well-used machine learning libraries. These classes and functions are in most cases, however, built on existing libraries. A list of the neural network architectures/layers and machine learning related algorithms appears in Figure 2 and examples of their use in select applications are given in section 3.2.

3.1.1 Integrable deep neural network

Integrable deep neural networks (IDNNs) [5, 6] are a type of neural network structure that allows training to derivative data and analytic integration to recover a neural network representation of the antiderivative. A schematic is shown in Figure 3. The IDNN is constructed as the derivative of a standard DNN so that after an IDNN is trained to the derivative data, its antiderivative is simply the standard DNN with the IDNN's trained weights and bias.

A standard DNN with n hidden layers, where $\boldsymbol{W}_\ell$, $\boldsymbol{b}_\ell$ are the weight matrix and bias vector of hidden layer ℓ, g is the activation function, a_ℓ and z_ℓ are intermediate vector values at each layer, and Y is the DNN output, can be represented

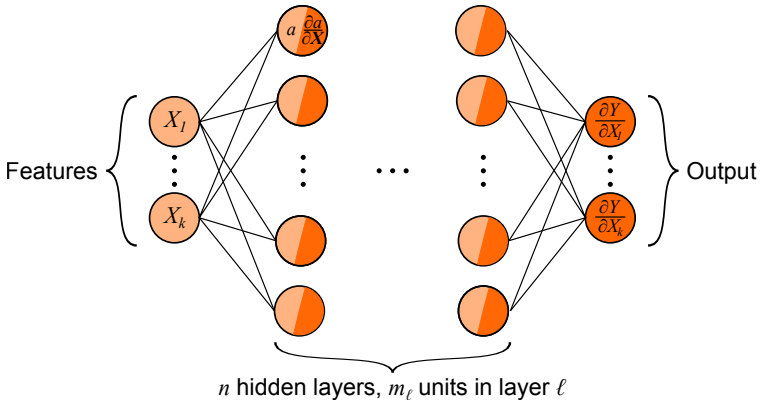


Figure 3: A schematic of an IDNN, which is trained to derivative data and can be analytically integrated to recover the antiderivative function.

with the following:

$$\begin{aligned} \boldsymbol{z}_\ell &= \boldsymbol{b}_\ell + \boldsymbol{W}_\ell \boldsymbol{a}_{\ell-1} \\ \boldsymbol{a}_\ell &= g(\boldsymbol{z}_\ell) \\ Y &= \boldsymbol{b}_{n+1} + \boldsymbol{W}_{n+1}\boldsymbol{a}_n \end{aligned} \tag{1}$$

Differentiation of Y leads to the following additional equations that describe the IDNN, which is represented by $\partial Y/\partial X_k$:

$$\begin{aligned} \frac{\partial \boldsymbol{a}_\ell}{\partial X_k} &= g'(\boldsymbol{z}_\ell) \odot \left(\boldsymbol{W}_\ell \frac{\partial \boldsymbol{a}_{\ell-1}}{\partial X_k} \right) \\ \frac{\partial Y}{\partial X_k} &= \boldsymbol{W}_{n+1} \frac{\partial \boldsymbol{a}_n}{\partial X_k} \end{aligned} \tag{2}$$

where the operator $\odot$ denotes element-wise multiplication. Note that both the activation function and its derivative are used in the IDNN.

Instead of directly implementing Eq. (2) to train an IDNN, however, we take advantage of the ability of deep learning libraries to apply a gradient operator to a standard neural network structure. To illustrate this approach, we consider a standard DNN to be represented as a function $\boldsymbol{Y}(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{b})$ of inputs $\boldsymbol{X}$, weights $\boldsymbol{W}$, and biases $\boldsymbol{b}$. Then, the training of an IDNN to a set of first derivative data $\{(\hat{\boldsymbol{X}}_\theta, \hat{\boldsymbol{y}}_{k_\theta})\}$ is the minimization of the mean square error of the gradient of a standard DNN (i.e. the IDNN) and the derivative data over the space of weights and biases:

$$\hat{\boldsymbol{W}}, \hat{\boldsymbol{b}} = \underset{\boldsymbol{W},\boldsymbol{b}}{\arg\min} \sum_{k=1}^{n} \mathrm{MSE}\left(\frac{\partial \boldsymbol{Y}(\boldsymbol{X}, \boldsymbol{W}, \boldsymbol{b})}{\partial X_k}\Big|_{\hat{\boldsymbol{X}}_\theta}, \hat{y}_{k_\theta} \right) \tag{3}$$

The `mechanoChemML` library provides an `idnn` module that returns an IDNN implemented in `Keras` and capable of being trained on first derivative (gradient) data, second derivative (Hessian) data, and/or data from the function itself, provided at

```
mechanoChemML.src.idnn.IDNN
```

The following example code shows how an IDNN with four inputs and two hidden layers of 20 neurons each would be initialized and trained to first derivative data:

```
idnn = IDNN(4,[20,20])

idnn.compile(loss=[None,'mse',None],
optimizer=keras.optimizers.RMSprop(lr=0.01))

idnn.fit(c_train,
mu_train,
```

```
epochs=1000,
batch_size=20)
```

A test function for the IDNN model is included in the test for one of the utility functions and is briefly described in Section 3.1.2. The IDNN is used in an example active learning workflow in Section 4.1.1 to train a free energy density function where the training data are derivatives of the free energy.

3.1.2 IDNN convexity

Since the IDNN was created to represent energy data, it is useful to determine if a point or set of points lies in a well (i.e., convex region) of the IDNN. The library includes functions that report the convexity of one or more points by checking if the Hessian, returned by the IDNN model, is positive definite. These functions are included at

```
mechanoChemML.src.idnn
```

An example using the IDNN convexity functions is shown in Figure 5 of Section 3.2.1, which describes active learning workflows. If desired, additional characteristics of the IDNN could also be evaluated for use in an active learning or other workflow.

A test function was written to confirm that the convexity functions and the IDNN are performing the differentiation and positive definiteness check correctly. This is done by comparing the code's result on an IDNN with known weights to the correct results. The test is located at

```
mechanoChemML.testing.idnn_convexity_test
```

3.1.3 Transform layer

Sometimes it is desirable to apply a transform to the inputs of a neural network. In addition to the relatively simple tasks of shifting and scaling data, this is one way to enforce some types of symmetry. For example, if a neural network should be symmetric about an input x, the network can use the transform x^2 as the input instead of x itself. Higher dimensional symmetries, such as rotational symmetries, would include symmetry preserving functions of multiple input variables. We provide a custom transform layer using `Keras` that allows the user to specify a function, `transforms`, that takes an input array `x` and returns a list of the transformed outputs, as in the following example snippet:

```
def transforms(x):
return [x[0],x[1]**2]

y = Transform(transforms)(x)
```

In this example, the first variable is taken as is, but the second variable is squared to enforce a symmetry about $x_1 = 0$. The transform layer is used in the construction of the `IDNN` model (see Section 3.1.1) to allow symmetry enforcement and is provided at the following location:

```
mechanoChemML.src.transform_layer.Transform
```

3.1.4 General neural network creation

Data-driven modeling research often requires a hyperparameter study, which involves the exploration of different NN architectures. To facilitate it, a collection of general NN classes, such as `NN_user_general`, `BNN_user_general`, and `BNN_user_weak_pde_general`, are provided in `mechanoChemML.src.nn_models` to construct NNs based on inputs from a configuration file. Such classes offer the flexibility to explore different NN architectures, particularly complex ones. They also allow non-expert users to quickly setup different NNs. For example, the following input will construct a deterministic NN with an encoder-decoder structure.

```
NNArchitecture =
type=PDERandom;
type=Conv2D | filters=8 | kernel_size=5 | activation=relu | padding=same;
type=MaxPooling2D | pool_size=(2,2) | padding=same;
type=Conv2D | filters=8 | kernel_size=5 | activation=relu | padding=same;
type=MaxPooling2D | pool_size=(2,2) | padding=same;
type=Flatten;
type=Dense | units=64 | activation=relu;
```

```
type=Reshape | target_shape=[4,4,4];
type=Conv2D | filters=8 | kernel_size=5 | activation=relu | padding=same;
type=UpSampling2D | size=(2,2);
type=Conv2D | filters=8 | kernel_size=5 | activation=relu | padding=same;
type=Conv2D | filters=1 | kernel_size=5 | activation=linear | padding=same;
```

One can also easily change the layer types in the configuration file to names of their corresponding probabilistic implementation in the `TensorFlow Probability` library to construct a probabilistic NN with identical architecture.

3.1.5 Weak PDE enforcing layers

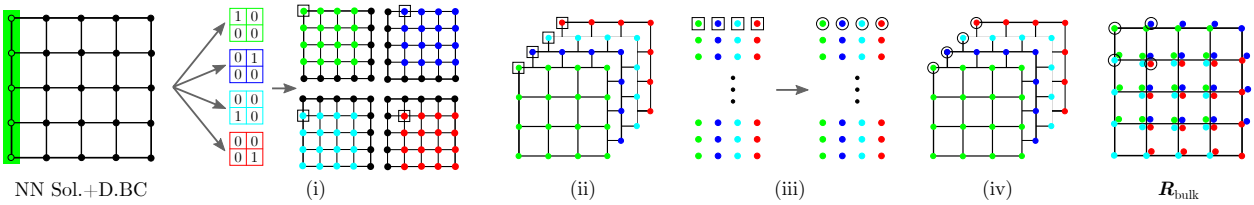


Figure 4: Illustration of steps to compute the bulk residual $\boldsymbol{R}_{\text{bulk}}$ based on the NN inputs (BC info) and NN outputs through different convolutional operator-based, and vectorized calculations.

A major thrust in using machine learning techniques of interest in the computational mechanics, materials, and physics communities is to solve PDEs with little or no pre-labeled data. In the `mechanoChemML` library, a collection of classes, layers, and functions, such as,

```
mechanoChemML.src.pde_layers.GetElementResidualMask
mechanoChemML.src.pde_layers.ComputeBoundaryMaskNodalData
mechanoChemML.src.pde_layers.ComputeNeumannBoundaryResidualNodalData
mechanoChemML.src.pde_layers.Get1DGaussPointInfo
mechanoChemML.src.pde_layers.Get2DGaussPointInfo
mechanoChemML.src.pde_layers.GetNodalInfoFromElementInfo
mechanoChemML.src.pde_layers.LayerBulkResidual
```

are provided in `mechanoChemML.src.pde_layers` to compute the discretized residual, as illustrated in Fig. 4, through an efficient, convolutional operator-based, and vectorized residual calculation implementation based on the framework proposed in [13]. The listed classes, layers, and functions will be used in the workflow related to the NN-based PDE solver that is discussed in Section 3.2.4.

3.1.6 Non-local calculus on graphs

Graph-based representation methods have shown great success in reduced-order modeling for a range of physical phenomena [40], that often lend themselves to a PDE form. Non-local calculus on graphs (see [41]) provide a mathematical framework for defining and evaluating these PDE models. A graph intrinsically allows representation of unstructured data, making it suitable for data-driven system identification when the data is sparse and non-uniformly distributed. For this reason, it has been applied to a variety of problems [41, 42, 43].

A graph, denoted by $G = (V, E)$, consists of a set of vertices, $i \in V$, with $|V| = n$. The vertices are connected by a set of edges, $e \in E$, where each edge $e = (i, j)$ is a pair of vertices. A data set provided in the form, $\{(\boldsymbol{x}_1, u_1), \cdots, (\boldsymbol{x}_n, u_n)\}$, with independent variables, $\boldsymbol{x} \equiv [x^1, \cdots x^p] \in \Omega \subset \mathbb{R}^p$, and dependent variables, $u : \Omega \to \mathbb{R}$, can be represented as a graph, (V, E) by placing a vertex of the graph at $\boldsymbol{x}_i$ for each data point. Furthermore, the function(al) values, $u_i \equiv u(\boldsymbol{x}_i)$ are treated as the states of the of the vertex. The edges on the graph can then be formed between points that lie in each other's local neighborhoods, $\mathcal{N}(\boldsymbol{x})$.

The graph can also have global and local attributes on the vertices and edges, as we describe below. Of importance are edge weights w, that are generally functions of the local vertex attributes, and admit a notion of distance between states on the graph. Scalars, $u(\boldsymbol{x}_i)$ correspond to function(al)s at the i^{th} vertex. Vectors $v(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are functions on vertex pairs i, j or the edge $e = (i, j)$. Gilboa *et al.* define a discrete calculus, consisting of non-local operators, based on differences between states of vertices $i, j \in V$, and an edge weight $w(\boldsymbol{x}_i, \boldsymbol{x}_j)$ [41];

$$\frac{\delta u}{\delta x^\mu}(\widetilde{\boldsymbol{x}}) = \sum_{\boldsymbol{x}\in\mathcal{N}(\widetilde{\boldsymbol{x}})} \frac{u(\boldsymbol{x}) - u(\widetilde{\boldsymbol{x}})}{z^\mu} w(\boldsymbol{x}, \widetilde{\boldsymbol{x}}), \qquad z = \boldsymbol{x} - \widetilde{\boldsymbol{x}} \tag{4}$$

We recently developed a methodology to choose an optimal set of edges, E and corresponding weights, w, to achieve arbitrary accuracy in the computation of these derivatives. The theoretical grounds for understanding the convergence of derivatives in non-local calculus to the standard definitions of local derivatives in terms of h^p-convergence in grid-based graphs are presented in Ref. [12]. The non-local derivatives are computed in the `mechanoChemML` library's `graph` module. The user can access the main functionality of the class by calling the following function, that executes the various functions involved in the library.

```
mechanoChemML.src.graph_main.main
```

The library handles Input/Output using *comma-separated values (CSV)* files. The main function takes only one argument: the *settings* dictionary, where the user defines the paths to input/output files, the model settings (e.g. dimension, p, of Ω) and the set of operations on data (e.g. calculation of partial derivatives). The user can manipulate the accuracy and the neighborhood selection strategy for derivatives using nested dictionaries in settings.

The following example code shows how to read data, $(x^1, x^2, x^3, u^1, u^2)$, formatted with column names `['x_1','x_2','x_3','u_1','u_2']` located in the file, `'./data/func_val.csv'`. First, the code demonstrates computation of an algebraic term, $u^3 = x^1 + x^2 + x^3$. Then estimation of a partial derivative $\delta u^3/\delta x^1$ is demonstrated. After the main function is executed, the new data is saved in `'./result/data.csv'`.

```
settings = {
#Path settings
'cwd': '.',
'directories_load':'data',
'directories_dump':'result',
'data_filename':'func_val.csv',

#Model settings
'model_order':2,
'model_p':3,

#Operations settings
'algebraic_operations': [[
{'func':lambda df: df['x_1'] + df['x_2'] + df['x_3'], 'labels':'u_3'} ]],

'differential_operations':[[
{'function':'u_3',
'variable': ['x_1'],
'weight':['stencil'],
'adjacency':['nearest'],
'manifold':[['x_1', 'x_2', 'x_3']],
'accuracy': [2],
'dimension':[0],
'order':1,
'operation':['partial']}]]}
mechanoChemML.src.graph_main.main(settings = settings)
```

3.2 ML workflows

The bulk of the library consists of several example workflows for specific applications. Each of these workflows combines various elements of the ML class library with additional algorithms and methods to solve a problem of interest in computational materials physics. Introductory descriptions of the physics and the library components used in the workflows are presented in the following section.

3.2.1 Active learning

Deep learning can be used to create surrogate models for accurate-yet-expensive simulations. In these cases, the data used for training comes from computations. This provides a challenge and an opportunity: the challenge is that we

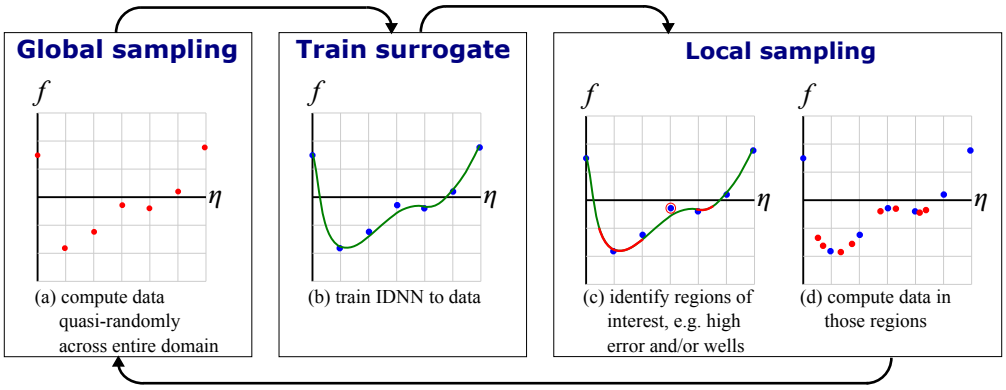


Figure 5: Flowchart describing the active learning process with hypothetical 1D data.

must first compute the data we use to train, and the opportunity is that we can choose what data to compute. Since it would be infeasible, particularly in the case of a high-dimensional input space, to densely sample data uniformly across the entire feature space, we turn to active learning. Active learning methods interweave machine learning training with data sampling in a way that lets the machine learning model choose the data that will be most informative [44]. This allows us to reduce the amount of data needed to train a sufficiently representative surrogate model.

The active learning workflows in the `mechanoChemML` library follow a general cycle of global sampling or exploration, training, and local sampling or exploitation. These are defined in the following functions

- `global_sampling`
- `surrogate_training`
- `local_sampling`

from the active learning examples provided at

```
mechanoChemML.workflows.active_learning
```

Global sampling or exploration allows data to be found in potentially interesting yet unexplored regions of the feature space, and it is also necessary to generate an initial training set. Training is performed using data sampled up to that point in the workflow, potentially down-selected based on a desired criterion. Active learning takes place during local sampling or exploitation, in which the currently trained network is queried to determine regions of interest where more data could be useful. This might simply determine areas where there is still a high pointwise error between the data and the neural network prediction. Local sampling might also be guided by the landscape of the DNN, such as convexity or high gradients. Other criteria can be used for local sampling, based on the important physics in the application. A flowchart describing the active learning process is shown in Figure 5.

The active learning workflow is demonstrated through an example scientific application in Section 4.1.1, which describes learning a high-dimensional free energy density function from first-principles data.

Testing Since an active learning workflow is largely application specific, there is no generic test function for the workflow as a whole. However, it is recommended that the user confirm that a modified workflow is running correctly by using a set of smaller workflow parameters (e.g., fewer training iterations, reduced data generation, etc.) before running the full workflow.

3.2.2 Stepwise regression for system identification

We have developed a class of inverse modeling techniques, referred as as Variational System Identification (VSI) to allow the identification of physics from data [34, 35, 45]. It aims to infer the physics governing observed phenomena by identifying the minimal set of operators whose combination into PDEs completely describes the data. VSI has been applied to various real-world problems such as discovering physics from dynamical systems[34, 35], identifying constitutive models of biological tissues[38], and inferring governing mechanisms in developmental biology and disease propagation[36, 37, 46]. VSI leverages the weak form unlike most other methods that use the strong form for identification of governing equations [1, 47, 48, 49, 50, 51, 52, 53]. It inherits many advantages from the weak form,

including (a) the natural identification of boundary conditions, (b) introduction of basis functions to interpolate the data, which can be chosen to have high regularity, (c) transfer of derivatives to variations, thus lowering the regularity required in the data, and (d) isolating operators on the data by judicious choice of variations.

As a first step of data collection, users can import from their own data set. However, the `mechanoChemML` library also provides an example DNS interface `mechanoChemML.third_party.dns_wrapper` which can be extended to link with PDE solver packages such as `deal.ii` and `FEniCS`. With this interface, users can build a full pipeline of data generation to system identification. An example of the forward model to generate the patterns in Figure 12 can be found at

```
mechanoChemML.workflows.systemID.forward_model
```

The `mechanoChemML` library provides an operator construction module for constructing operators in weak form with the basis function from two families: polynomial basis functions traditional to finite element analysis (FEA), and the Non-Uniform Rational B-Splines (NURBS) used widely in Isogeomeric Analysis (IGA). A example of constructing operators in weak form as shown in Equations (24) and (25) can be found at

```
mechanoChemML.workflows.systemID.generate_basis
```

The approach to inference combines system identification by stepwise regression [34, 45] with a statistical criterion called the F-test for eliminating basis terms. This forms the main component in the stepwise regression module:

```
mechanoChemML.src.stepwiseRegression.stepwiseR
```

The stepwise regression module provides two basis elimination strategies and a variety of regularization schemes for regression. Users may adopt the stepwise regression module and build their own system identification framework. However we have found that in most cases, users can set up their inference problems using a higher level module for system identification:

```
mechanoChemML.workflows.systemID.systemID
```

using the minimal coding as follows:

```
problem = systemID()
problem.identifying(data)
```

The benefit of this approach is that users can easily set up problems with minimal coding, and control them with a configuration file. A typical configuration file contains two part: controls for VSI such as “identify_strategy" and controls for stepwise regression such as using ridge regression, setting “F_criteria" and others. A representative configuration file is presented below.

Example of configuration file

```
[VSI]
data_dir=N/A
identify_strategy= specified_target
target_index= 0

[StepwiseRegression]
basis_drop_strategy = most_inignificant
regression_method = ridge
alpha_ridge = 1.0e-5
F_criteria=1
```

We refer readers to our previous work [34, 35, 38] and online documentation for the detailed explanations.

The full script for discovering the pattern forming physics problem described in Section 4.4.1 is then presented as following:

```
mechanoChemML.examples.systemID.Example1_pattern_forming.main
```

The VSI workflow is also illustrated in Figure 6.

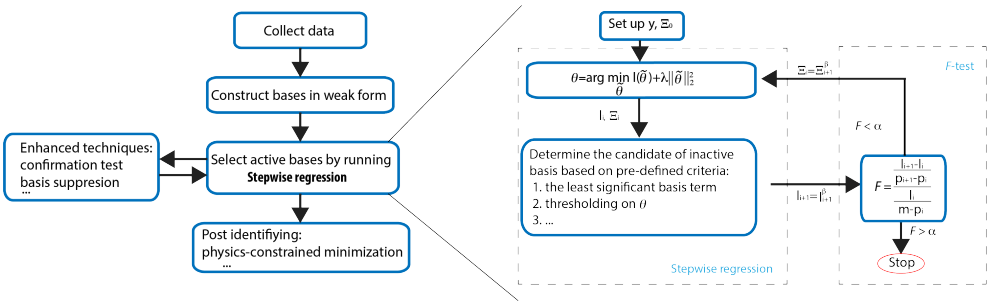


Figure 6: Flowchart of Variational System Identification as a workflow.

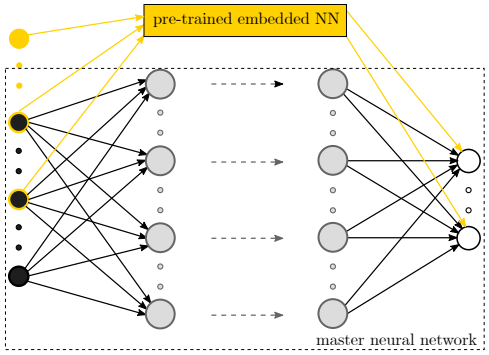


Figure 7: Illustration of the NN architecture used for multi-resolution learning.

3.2.3 Multi-resolution learning

In materials physics problems, it is not uncommon to encounter data that possesses a hierarchical structure. For example, when studying the homogenized stress-strain response of a family of multi-component crystalline microstructures, the free energy of each microstructure has a multi-resolution structure with a dominant trajectory arising from phase transformations that drive evolution of the microstructure, and small-scale fluctuations from strains that explore the effective elastic response of a given microstructure [11]. The dominant trajectory strongly depends on the microstructural information, such as the volume fraction, the location and orientation of each crystalline phase, and the interfaces, whereas the small-scale fluctuations are related to the applied loading. Multi-resolution learning, as illustrated in Fig. 7, can be used to capture the details in the data, which are not well-delineated by the pre-trained model. Taking the homogenization of microstrutures problem presented in Section 4.2.1 as an example, a multi-resolution neural network (MRNN) can be built upon pre-trained fully connected NNs or convolutional NNs, which describe the dominant part of the free energy, to learn the small-scale fluctuations of free energy and predict homogenized stresses. The loss function for such a problem therefore is written as

$$\text{MSE} = \frac{1}{m}\sum_i (\mathbf{Y}-\mathbf{Z})_i^2 \quad \text{with} \quad \mathbf{Y} = \Psi_{\text{mech}} - \Psi^0_{\text{mech,NN}} \tag{5}$$

where $\mathbf{Y}$ is the label, $\mathbf{Z}$ is the MRNN predicted value, Ψ_{mech} is the hierarchical quantity to be learned, and $\Psi^0_{\text{mech,NN}}$ is the pre-trained NN learned dominant information in the data. If additional physics-based constraints need to be applied, the loss function could be updated as

$$\text{MSE} = \frac{1}{m}\sum_i \left[(\mathbf{Y}-\mathbf{Z})_i^2 + \beta \left\|\boldsymbol{P}_{\text{MRNN}} - \boldsymbol{P}_{\text{DNS}}\right\|_i^2\right] \quad \text{with} \quad \mathbf{Y} = \Psi_{\text{mech}} - \Psi^0_{\text{mech,NN}} \tag{6}$$

where β is a parameter to penalize the constraints expressed in terms of the Frobenius norm $\|\bullet\|$. Multi-resolution learning is a two-step learning process, which can be constructed based on classes and functions provided in the `mechanoChemML` library, such as

```
mechanoChemML.workflows.mr_learning.mrnn_models
mechanoChemML.workflows.mr_learning.mrnn_utility.
```

See the homogenization of microstrutures problem presented in Section 4.2.1 for details of constructing a MRNN.

3.2.4 NN-based solver for PDEs

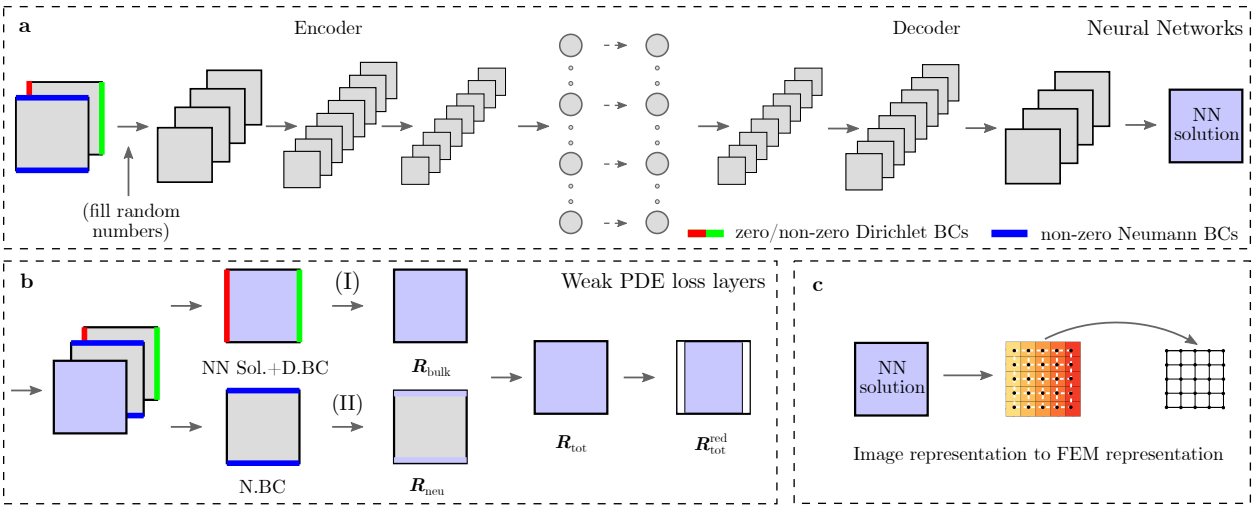


Figure 8: Illustration of the NN architecture and steps in the residual calculation for a NN-based PDE solver.

We have developed NN-based PDE solvers that compute full field solutions orders of magnitude faster than the traditional PDE solvers [13]. These solvers work for both small datasets, which only contain a few boundary value problems (BVPs), and large datasets, which could contain hundreds of thousands of BVPs. The workflow to construct a NN-based PDE solver consists of (i) implementing the discretized residual of PDEs (ii) building PDE constrained NNs, (iii) preparing BCs-encoded NN input data, (iv) training NNs, and (v) testing NNs. The NN inputs are multi-dimensional `numpy` arrays, which can be easily generated. One can use the `mechanoChemML.workflows.pde_solver.geometry` to generate BCs-encoded NN inputs for arbitrary polygons. The PDE constrained NNs are constructed as subclasses of the `tf.keras.Model`. This allows us to use methods such as `train_on_batch` and `test_on_batch` from the Keras model to train and test the PDE constrained NNs.

To construct NNs to solve PDEs, we use `mechanoChemML.src.nn_models.BNN_user_weak_pde_general`, whose output also contains the NN input information, including the BCs and domain, to construct the PDE constrained loss.

A templated workflow for solving general elliptic PDEs is provided at

```
mechanoChemML.workflows.pde_solver.pde_workflow_steady_state,
```

which utilizes classes and functions provided at

```
mechanoChemML.src.pde_layers
mechanoChemML.src.nn_models,
```

with the detailed steps to compute the discretized residual being provided at

```
mechanoChemML.workflows.pde_solver.pde_workflow_steady_state._compute_residual.
```

The discretized residual can then be used to construct either the loss for deterministic NNs or the likelihood function that is used in the total loss of probabilistic NNs. Ref. [13] has details on the theoretical background to the construction of different loss functions.

For BNNs, the model parameters $\boldsymbol{\Theta}$ are stochastic and sampled from a posterior distribution $P(\boldsymbol{\Theta}|\mathcal{D})$, which is computed based on Bayes' theorem

$$P(\boldsymbol{\Theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\Theta})P(\boldsymbol{\Theta})}{P(\mathcal{D})}, \tag{7}$$

where $\mathcal{D}$ denotes the i.i.d. observations (training data) and P represents the probability density function. In (7), $P(\mathcal{D}|\boldsymbol{\Theta})$ is the likelihood, $P(\boldsymbol{\Theta})$ is the prior probability, and $P(\mathcal{D})$ is the evidence, respectively. The likelihood is the probability of the observed data $\mathcal{D}$ given parameters $\boldsymbol{\Theta}$.

To compute the posterior distributions of $\boldsymbol{\Theta}$, we use variational inference, which approximates the exact posterior distribution $P(\boldsymbol{\Theta}|\mathcal{D})$ with a more tractable surrogate distribution $Q(\boldsymbol{\Theta})$ by minimizing the Kullback-Leibler (KL) divergence

$$Q^* = \arg\min D_{\mathrm{KL}}(Q(\boldsymbol{\Theta})||P(\boldsymbol{\Theta}|\mathcal{D})). \tag{8}$$

with the KL divergence being computed as

$$D_{\mathrm{KL}}(Q(\boldsymbol{\Theta})||P(\boldsymbol{\Theta}|\mathcal{D})) = \mathbb{E}_Q[\log Q(\boldsymbol{\Theta})] - \mathbb{E}_Q[\log P(\boldsymbol{\Theta}, \mathcal{D})] + \log P(\mathcal{D}). \tag{9}$$

Since the evidence $P(\mathcal{D})$ is very expensive to compute, and in general challenging to obtain, we optimize the so-called evidence lower bound (ELBO), which is equivalent to the KL-divergence up to an added constant, with

$$\mathrm{ELBO}(Q) = \mathbb{E}_Q[\log P(\mathcal{D}|\boldsymbol{\Theta})] - D_{\mathrm{KL}}\left(Q(\boldsymbol{\Theta})||P(\boldsymbol{\Theta})\right). \tag{10}$$

The loss function for the BNN is written as

$$\mathcal{L} = D_{\mathrm{KL}}\left(Q(\boldsymbol{\Theta})||P(\boldsymbol{\Theta})\right) - \mathbb{E}_Q[\log P(\mathcal{D}|\boldsymbol{\Theta})], \tag{11}$$

which consists of a prior-dependent part and a data-dependent part, with the former being the KL-divergence of the surrogate posterior distribution $Q(\boldsymbol{\Theta})$ and the prior $P(\boldsymbol{\Theta})$, and the latter being the negative log-likelihood cost, which is related to the discretized residual of a specific PDE system. In the `mechanoChemML` library, one can provide the detailed implementation of a specific PDE system in the method `_bulk_residual`. The bulk residual for the specific PDE system can be constructed by utilizing the methods defined in the class `mechanoChemML.src.pde_layers.LayerBulkResidual`. An example of the use of the NN-based PDE solver for steady-state diffusion is provided in Section 4.3.1.

4 Scientific application examples

4.1 Active learning

4.1.1 Learning free energy density functions for scale bridging

The formation and evolution of material microstructures can be simulated at the continuum scale using phase field modeling. These techniques rely on a free energy density function for the material, which, among other physics, encapsulates a description of the stable phases of the materials. Atomistic models, including DFT and statistical mechanics calculations, can be used to obtain the gradient of the free energy, i.e. the chemical potential, for discrete values of chemical composition and/or other parameters describing the ordered/disordered state of the atomic structure, in terms of order parameters. Learning a free energy density function from these data that can be used in phase field models is a powerful tool in bridging scales.

Since the chemical potential training data that are produced are gradients of the free energy, we employ IDNNs, which were described in Section 3.1.1, to train to the data. With order parameters η_k and chemical potentials $\mu_k := \partial f/\partial \eta_k$, $k = 1, \ldots, n$, the IDNN training involves the following optimization:

$$\hat{\boldsymbol{W}}, \hat{\boldsymbol{b}} = \underset{\boldsymbol{W},\boldsymbol{b}}{\arg\min} \sum_{k=1}^{n} \mathrm{MSE}\left(\frac{\partial f(\boldsymbol{\eta}; \boldsymbol{W}, \boldsymbol{b})}{\partial X_k}\Big|_{\hat{\boldsymbol{\eta}}_\theta}, \hat{\mu}_{k_\theta}\right) \tag{12}$$

This results in the following representation of the free energy density, $\hat{f}$ as a function of the order parameters:

$$\hat{f}(\boldsymbol{\eta}) = f(\boldsymbol{\eta}; \hat{\boldsymbol{W}}, \hat{\boldsymbol{b}}) \tag{13}$$

where f has the form of a standard, fully connected deep neural network.

Additionally, in some situations the number of order parameters may increase the dimensionality of the input space to the point where it is unfeasible to uniformly and densely sample the entire space. In these cases, it is helpful to employ active learning, as described in Section 3.2.1, to guide the sampling of data. An example is the Ni-Al system, described by us in previous work [6]. $\mathrm{Ni}_{1-x}\mathrm{Al}_x$ forms an ordered structure a $x = 1/4$, while a disordered structure is formed for lower values of x. Additionally, four different translational variants of the $x = 1/4$ ordering exist, which affect the formation of precipitates in the Ni-Al system. By defining the input space of the free energy using composition, η_0 and three additional order parameters, η_1, η_2, η_3, it is possible to model the order-disorder transition and track the formation of variants. The stable disordered phase at low x and each of the four ordered variants at $x = 1/4$ correspond to wells in the free energy density. Because of the importance is capturing these energy wells, we used pointwise error and local convexity as criteria to guide the local sampling.

As implemented in the `mechanochemML` library, the IDNN was considered sufficiently converged after twelve iterations of the active learning workflow. The evolution of a two-dimensional slice of the free energy is shown in Figure 9, including energy wells corresponding to the disordered phase and one of the ordered variants. The following function from the `Active_learning` class encapsulates the active learning workflow:

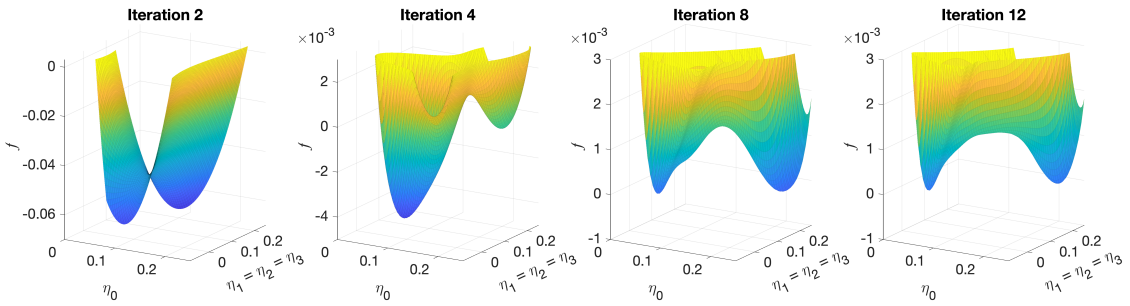


Figure 9: Plots of the free energy density (i.e., the integrated IDNN) in a two-dimensional subspace at various iterations of the active learning workflow. (A linear function of η_0 has been added to each free energy surface to "rotate" the surface and enhance visualization of the convex regions.)

```
def main_workflow(self):
for rnd in range(self.N_rnds):
self.global_sampling(2*rnd)

if rnd==1:
self.hyperparameter_search(rnd)
custom_objects = {'Transform': Transform(self.IDNN_transforms())}

unique_inputs = self.idnn.unique_inputs
self.idnn = keras.models.load_model('idnn_1',
custom_objects=custom_objects)
self.idnn.unique_inputs = unique_inputs

self.surrogate_training(rnd)
self.local_sampling(2*rnd+1)
```

4.2 Multi-resolution learning

In this section, we illustrate how to use `mechanoChemML` library to perform multi-resolution learning to study the homogenized behavior of microstructures. Readers are referred to [11] for more details.

4.2.1 Microstructure homogenization

In this example, we illustrate how to use multi-resolution NNs provided in the `mechanoChemML` library to construct surrogate NN models to predict the homogenized, macroscopic, mechanical free energy and stress fields arising in a family of multi-component crystalline solids that develop microstructure. The physics is driven by a non-convex free energy density function ψ,

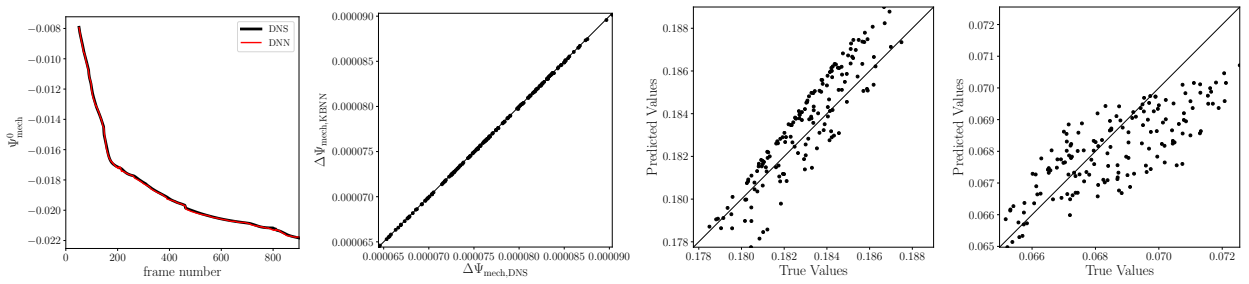
$$\psi(c, \boldsymbol{e}, \nabla c, \nabla \boldsymbol{e}) = \mathscr{F}(c, \boldsymbol{e}) + \mathscr{G}(\nabla c, \nabla \boldsymbol{e}), \tag{14}$$

with

$$\mathscr{F}(c, \boldsymbol{e}) = \underbrace{16 d_c c^4 - 32 d_c c^3 + 16 d_c c^2}_{\text{chemical}} + \underbrace{\frac{2d_e}{s_e^2}(e_1^2 + e_3^2) + \frac{d_e}{s_e^4} e_2^4}_{\text{mechanical}} + \underbrace{(1 - 2c)\frac{2d_e}{s_e^2} e_2^2}_{\text{mechanochemical}} \tag{15a}$$

$$\mathscr{G}(\nabla c, \nabla \boldsymbol{e}) = \underbrace{\frac{1}{2}\nabla c \cdot \kappa \nabla c}_{\text{chemical}} + \underbrace{\frac{1}{2}\nabla e_2 \cdot \lambda_e \nabla e_2}_{\text{mechanical}} \tag{15b}$$

where c is the compositional parameter, $\boldsymbol{e}$ is the mechanical strain vector with e_1, e_2, and e_3 as its components, and $\{d_c, d_e, s_e, \kappa, \lambda_e\}$ are material parameters. Here, $\mathscr{F}$ represents a homogeneous contribution from both composition and strain, and $\mathscr{G}$ being a gradient-dependent, inhomogeneous contribution to regularize the free energy density. The resulting microstructure, driven by this free energy with its perturbation, has a hierarchical–or multi-resolution–structure, which can be resolved by an MRNN. The main steps to build the surrogate homogenization model workflow include: (i) synthetic training data generation, (ii) training NNs to describe the dominant characteristic of the data with



(a) DNN predicted Ψ^0_{mech} (b) MRNN predicted $\Delta\Psi_{\text{mech}}$ (c) MRNN predicted P_{11} (d) MRNN predicted P_{22}

Figure 10: Illustration of the NN predicted solutions. (a) NN predicted base mechanical free energy Ψ^0_{mech} vs. DNS solutions. (b) the MRNN predicted $\Delta\Psi_{\text{mech}}$ vs. DNS solutions. (c-d) the components of $\boldsymbol{P}_{\text{MRNN}}$ vs. DNS solutions, where the MRNN learns a reasonable derivative representation for P_{11} and P_{22}.

a hyper-parameter search, (iii) constructing the MRNN to fully represent the hierarchically evolving free energy with a hyper-parameter search. To setup the workflow, we use the following classes in the `mechanoChemML` library

```
mechanoChemML.src.kfold_train
mechanoChemML.src.hparameters_cnn_grid
mechanoChemML.src.hparameters_dnn_grid
mechanoChemML.workflows.mr_learning.mrnn_models
mechanoChemML.workflows.mr_learning.mrnn_utility
mechanoChemML.third_party.dns_wrapper.dns_wrapper
```

Synthetic data generation is a tedious process and is not the focus of the `mechanoChemMl` library, although a user could potentially use the example Python DNS interface class provided at `mechanoChemML.third_party.dns_wrapper.dns_wrapper` to run, the spinodal-decomposition initial boundary value problem (IBVP) that generated the data for the example in this section. A small fraction of preprocessed data that was thus generated and used in Ref. [11] is provided with this example in `mechanoChemML` to allow readers to skip the data generation process and directly learn the workflow.

As discussed in [11], both DNNs and CNNs can be used to learn the dominant characteristic of the data. In this example, we focus on DNNs in the interest of simplicity. Furthermore, we only focus on predicting the free energy for microstructures generated from one DNS, and the homogenized stress for a single microstructure. However, more complex examples studied in [11] are also included in the `mechanoChemML` library. Readers may refer to the code documentation for more details. The MRNN constructed based on DNNs for microstructures from one DNS resides at

```
examples/mr_learning/Example1_single_microstructure_dnn
```

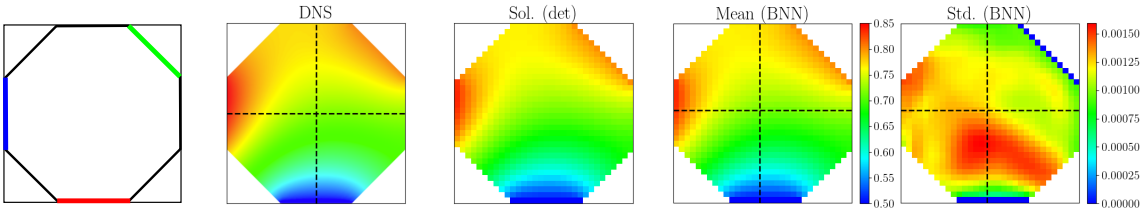
with four steps

```
step1_hp_search_main
step2_final_dnn_main
step3_hp_search_mrnn_detail
step4_final_mrnn_no_penalize_P.
```

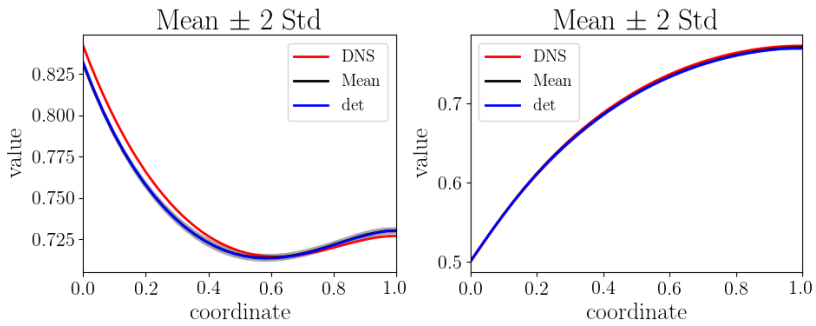
The first two steps perform a hyper-parameter search to identify the best DNN structure and subsequently train the best model to capture the dominant characteristic of the data. The MRNN is constructed by subtracting the dominant characteristic predicted by the pre-trained DNNs from the data. The third step performs another hyper-parameter search to identify the best MRNN structure to capture the detailed characteristic, and the fourth step trains the best found model. The resulting solutions from both the DNN and MRNN are presented in Fig. 10. See Ref. [11] for detailed interpretation of the results.

4.3 NN-based PDE solver

In this section, we provide an example to illustrate the use of the `mechanoChemML` library to solve PDEs with NNs. See Ref. [13] for further details.



(a) setup (b) results (32×32)



(c) UQ horizontal (32×32) (d) UQ vertical (32×32)

Figure 11: Steady-state diffusion BVP on an octagonal domain with mixed BCs. (a) Simulation setup, red: zero Dirichlet BC, green: non-zero Dirichlet BC, blue: non-zero Neumann BC. (b) Solutions from DNS, deterministic (det) NNs, and BNNs (Mean, Std.) for an output resolution of 32×32. (c-d) Quantitative comparison of the solution distribution between DNS and BNNs along the horizontal and vertical dashed lines in (b).

4.3.1 Solving steady-state diffusion

Here, we construct the NN-based solver to solve single species steady-state diffusion, whose strong form is written as

$$\begin{aligned} \nabla \cdot \boldsymbol{H} &= \boldsymbol{0} \quad \text{on} \quad \Omega, \\ c(\boldsymbol{X}) &= \bar{c}(\boldsymbol{X}) \quad \text{on} \quad \Gamma^c, \\ H &= \bar{H}(\boldsymbol{X}) \quad \text{on} \quad \Gamma^H, \end{aligned} \tag{16}$$

In (16), c represents the compositional order parameter, $\boldsymbol{H}$ is the diffusive flux term defined as

$$\boldsymbol{H} = -D\nabla c, \tag{17}$$

with D as the diffusivity, and H is the outward surface flux in the normal direction. The discretized residual function for steady-state diffusion is written as

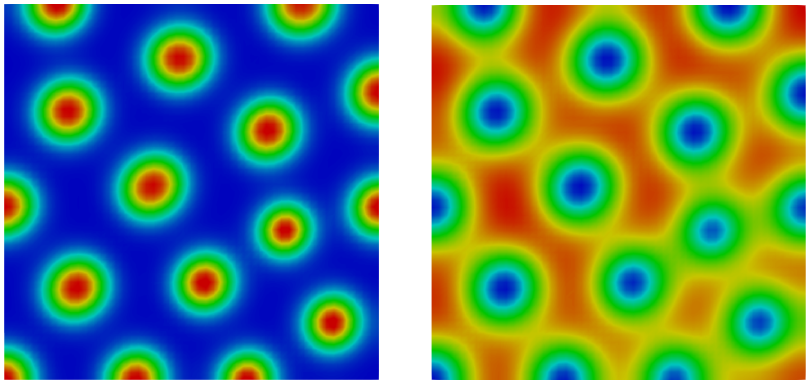
$$\boldsymbol{R} = \sum_{e=1}^{n_{\text{elem}}} \left\{ \int_{\Omega^e} \boldsymbol{B}^T \boldsymbol{H} dV - \int_{\Gamma^{e,H}} \boldsymbol{N}^T \bar{H}\, dS \right\}. \tag{18}$$

We consider a diffusivity of $D = 1.0$. The detailed Python implementation of this discretized residual of this PDE system is provided in the tutorial example in the library documentation. We use the NN-based PDE solver to solve a boundary value problem (BVP) on an octagonal domain with mixed applied BCs. The BVP setup and the associated results are illustrated in Fig. 11. The details of this example, including the uncertainty quantification, can be found in Ref. [13].

In the `mechanoChemML` library, we have provided implementations of steady-state diffusion, linear elasticity, and nonlinear elasticity at

```
mechanoChemML.workflows.pde_solver.pde_system_diffusion_steady_state
mechanoChemML.workflows.pde_solver.pde_system_elasticity_linear
mechanoChemML.workflows.pde_solver.pde_system_elasticity_nonlinear
```

For the steady-state diffusion example, input data and configuration files are provided at



(a) c_1 (b) c_2

Figure 12: c_1 and c_2 at steady state, simulated using the pre-defined Schnakenberg kinetics model.

```
examples/pde_solver/Example1_diffusion_steady_state
```

To generate DNS solutions for comparison, users can follow an example Python DNS interface class provided at `mechanoChemML.third_party.dns_wrapper.dns_wrapper` to run the steady-sate diffusion boundary value problem (BVP) by specifying the corresponding BVP name via the command line as:

```
python mechanoChemML/third_party/dns_wrapper/dns_wrapper.py -e ↘
    steady_state_diffusion
```

We note that the Python DNS wrapper only provides an interface to interact with different physics-based simulation tools. Users need to install the actual software in order to run the physics-based simulation. Furthermore, the physics-based simulation tools need to be compiled to a Python dynamic library. In this example, the installation of `mechanoChemFEM` is needed. The results from the NN-based PDE solver are presented in Fig. 11.

4.4 System identification

In this section, we demonstrate system inference in the `mechanoChemML` library with an example of pattern formation in material microstructures using stepwise regression introduced in Section 3.2.2 within the VSI framework.

4.4.1 Pattern formation of material microstructures

In the case of dynamic pattern formation in materials, the physics is governed by first-order (in time) PDEs–a class that includes the time-dependent reaction-diffusion and phase field equations. For demonstration, consider the following model form in $[0, T] \times \Omega$:

$$\frac{\partial c_1}{\partial t} = D_{11}\nabla^2 c_1 + D_{12}\nabla^2 c_2 + R_{10} + R_{11}c_1 + R_{12}c_2 + R_{13}c_1^2 c_2 \tag{19}$$

$$\frac{\partial c_2}{\partial t} = D_{21}\nabla^2 c_1 + D_{22}\nabla^2 c_2 + R_{20} + R_{21}c_1 + R_{22}c_2 + R_{23}c_1^2 c_2 \tag{20}$$

$$\text{with} \quad \nabla c_1 \cdot \mathbf{n} = 0, \quad \nabla c_2 \cdot \mathbf{n} = 0 \text{ on } \Gamma = \partial\Omega \tag{21}$$

$$\text{and } c_1\boldsymbol{x}, 0) = c_{1_0}(\boldsymbol{x}), \quad c_2(\boldsymbol{x}, 0) = c_{2_0}(\boldsymbol{x}). \tag{22}$$

Here, $c_1(\boldsymbol{x}, t)$ and $c_2(\boldsymbol{x}, t)$ are the compositions, with diffusivities $D_{11}, \ldots, D_{22}$ and reaction rates $R_{10}, \ldots, R_{23}$ assumed constant in space and time. This model represents the coupled diffusion-reaction equations for two species following Schnakenberg kinetics [54]. For an activator-inhibitor species pair having auto-inhibition with cross-activation of a short range species, and auto-activation with cross-inhibition of a long range species these equations form so-called Turing patterns [55]. See Fig. 12.

For infinite-dimensional problems with Dirichlet boundary conditions on Γ^c, the weak form corresponding to the strong form in Equation (19) or (20) is, $\forall\, w \in \mathscr{V} = \{w |\, w = 0 \text{ on } \Gamma^c\}$, find c such that

$$\int_\Omega w \frac{\partial c}{\partial t} \mathrm{d}v = \boldsymbol{\omega} \cdot \boldsymbol{\chi} \tag{23}$$

where Ω is the domain, $\boldsymbol{\chi}$ is the vector containing all possible independent operators in weak form:

$$\boldsymbol{\chi} = \left[-\int_\Omega \nabla w \cdot \nabla c_1 \mathrm{d}v, -\int_\Omega \nabla w \cdot \nabla c_2 \mathrm{d}v, \int_\Omega w \mathrm{d}v, \int_\Omega w c_1 \mathrm{d}v, \int_\Omega w c_2 \mathrm{d}v, \int_\Omega w c_1^2 c_2 \mathrm{d}v \right] \tag{24}$$

and $\boldsymbol{\omega}$ is the vector of operator prefactors. Using this notation, $\boldsymbol{\omega} = [D_{11}, \ldots, R_{13}]$ for Equation (19) and $\boldsymbol{\omega} = [D_{21}, \ldots, R_{23}]$ for Equation (20). Upon integration by parts, application of appropriate boundary conditions, and accounting for the arbitrariness of w, the finite-dimensionality leads to a vector system of residual equations: $\mathscr{R} = \boldsymbol{y} - \boldsymbol{\chi}\boldsymbol{\omega}$,

where $\boldsymbol{y}$ is the time derivative term and may be represented via a backward difference approximation

$$y^i = \int_\Omega N^i \sum_{a=1}^{n_\mathrm{b}} \frac{c_n^a - c_{n-1}^a}{\Delta t} N^a \mathrm{d}v \tag{25}$$

with N^i denoting the basis function corresponding to degree of freedom (DOF) i, and $\Delta t = t_n - t_{n-1}$ the time step. The other operators in $\boldsymbol{\chi}$ are constructed similarly and grouped together into the matrix $\boldsymbol{\chi}$. Minimizing the residual norm towards $||\mathscr{R}|| = 0$ then yields the linear regression problem

$$\boldsymbol{y} = \boldsymbol{\chi}\boldsymbol{\omega}. \tag{26}$$

Solving Equation (26) via standard regression, especially with noisy data, will lead to a non-sparse $\boldsymbol{\omega}$. Such a result will not sharply delineate the relevant bases for parsimonious identification of the governing system. We therefore use stepwise regression coupled with the statistical F-test for parsimonious inference of a minimal set of operators [34, 35]. Performing stepwise regression with the F-test also is a key step in the workflow of system identification (Fig. 6).

In another scenario, steady state, or near-steady state data with high spatial resolution may be obtained from modern microscopy methods. In fact the data satisfying the steady state equation:

$$\boldsymbol{\chi} \cdot \boldsymbol{\theta} = 0, \tag{27}$$

already provide rich information about the spatial operators (that is, other than time derivatives) in the system. However, in the absence of prior knowledge about the system, it may be challenging to choose a proper "target" operator, e.g. the left hand side of Equation (26). The confirmation test developed in [35] can provide a sharp condition for acceptance of the inferred operators. The confirmation test also has been used in the example of pattern formation in material microstructures demonstrated in this section.

4.5 Graph-theoretic system identification

In this section, we provide an example to illustrate the use of the graph theory based non-local calculus in the `mechanoChemML` library to generate operators for Variational System Identification problems. Specifically, a 1D phase separation problem is introduced via the Allen-Cahn equations for generation of high-dimensional data. These data are used to train a reduced order model. The readers are directed to Ref. [12] for an in-depth discussion on this example.

4.5.1 Allen-Cahn dynamics

Consider a field $\phi = \phi(x,t) : \Omega \times [0,T] \mapsto \mathbb{R}$, governed by first order dynamics driven by gradient flow:

$$\frac{\partial \phi}{\partial t} = -M_\phi \frac{\delta \psi}{\delta \phi}, \qquad \text{in } \Omega \times [0,T] \tag{28}$$

$$\nabla \phi \cdot \mathbf{n} = 0, \qquad \text{on } \partial\Omega \tag{29}$$

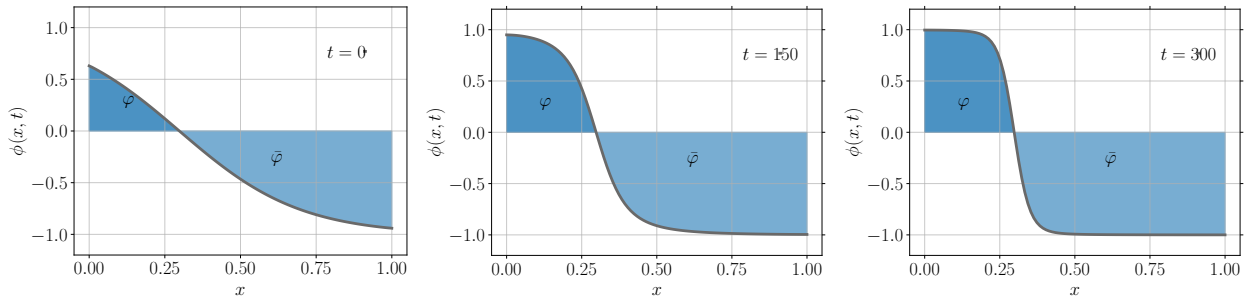
$$\phi(x,0) = \phi_0(x) \tag{30}$$

with the free energy density, ψ including f, an algebraic Landau energy density of the form

$$\psi = f(\phi) + \frac{\lambda}{2}|\nabla\phi|^2, \qquad f(\phi) = (\phi^2 - 1)^2 \tag{31}$$

and f having wells at $\phi = \pm 1$. The gradient energy $\lambda|\nabla\phi|^2$, with $\lambda > 0$ penalizes sharp transitions between the positive and negative phases $\Omega_\pm \subset \Omega$, which are defined by

$$x \in \begin{cases} \Omega_+ & \text{if } \phi(x) \geq 0 \\ \Omega_- & \text{if } \phi(x) < 0 \end{cases} \tag{32}$$



(a) Initial condition. (b) Solution at intermediate t. (c) Near equilibrium solution.

Figure 13: Field evolution of 1D Allen-Cahn dynamics with $M_\phi = 1e-3, \lambda = 1$ at 0, 150, and 300 time steps. A Backward-Euler scheme is used with a time step of $\Delta t = 0.01$

.

The kinetics are controlled by the local mobility, $M_\phi \geq 0$. The Eqs (28) and (31) together constitute the Allen-Cahn equation [56]. An example of the system dynamics in 1D appears in Fig 13.

The local variations in the field, ϕ, can be studied in terms of global observables, for instance, the total energy of the system, Ψ:

$$\Psi\left[\phi\right] = \int_\Omega \psi(x,t)\mathrm{d}\Omega \tag{33}$$

A rich set of global observables can be constructed by computing phase average quantities as different functions of ϕ as:

$$\varphi_{g(\phi)_\pm} = \frac{1}{|\Omega|}\int_\Omega g(\phi)I(\pm\phi)\mathrm{d}\Omega, \qquad I(\phi) = \begin{cases} 1 & \text{if } \phi \geq 0 \\ 0 & \text{if } \phi < 0, \end{cases}$$

The following functions, $g(\phi)$ are chosen in this study:

$$g(\phi) \in \mathcal{G} = \left\{\phi, \phi^2, \phi^3, \phi^4, \phi^5, f(\phi), f'(\phi), \Delta\phi, |\nabla\phi|^2\right\}$$

Similarly, the total energy of the system in the positive phase, Ψ_+ can be estimated as:

$$\Psi_+\left[\phi\right] = \int_\Omega \psi(x,t)I(\phi)\mathrm{d}\Omega \tag{34}$$

A code to simulate Allen-Cahn dynamics and estimate the global observables is provided in

```
examples.non_local_calculus.Example2_Allen_Cahn.dns
```

4.5.2 Reduced order model for Global observables

We are interested in developing a parsimonious reduced order model for φ_{ϕ_+}. We consider a model for first order kinetics of the observable as follows:

$$\frac{d\varphi_{\phi_+}}{dt} = \sum_\alpha \gamma^\alpha v_\alpha, \qquad v_\alpha \in \mathcal{B} \tag{35}$$

Here, $\mathcal{B}$ is the basis set of operators. For demonstration, we choose the following 3 successively expanded basis sets:

$$\mathcal{B}_1 = \left\{\frac{\delta\Psi_+}{\delta\varphi_{g(\phi)_+}} \cup \frac{\delta\Psi_+}{\delta\varphi_{g(\phi)_-}} | g(\phi) \in \mathcal{G}\right\} \tag{36}$$

$$\mathcal{B}_2 = \mathcal{B}_1 \cup \left\{\frac{\delta\Psi}{\delta\varphi_{g(\phi)_+}} \cup \frac{\delta\Psi}{\delta\varphi_{g(\phi)_-}} | g(\phi) \in \mathcal{G}\right\} \tag{37}$$

$$\mathcal{B}_3 = \mathcal{B}_2 \cup \left\{\varphi_{g(\phi)_+} \cup \varphi_{g(\phi)_-} | g(\phi) \in \mathcal{G}\right\} \tag{38}$$

Our choice of basis sets, $\mathcal{B}_1$ and $\mathcal{B}_2$, is guided by the gradient flow form of the field (Eq.(28)). The derived quantities like $\frac{\delta\Psi}{\delta\varphi_{\phi_+}}$ are estimated using the non-local calculus via the code:

Basis	Iteration	Loss	Model coefficients				
			$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi_+}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi_-}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{f(\phi)_-}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi^3_-}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi^4_-}}}$
$\mathcal{B}_1$	14	1.56e-4	-2.81e-1	3.01e-1	-5.84e-2	1.39e-2	4.45e-4
	15	1.82e-4	-2.66e-1	2.48e-1	-3.54e-2	4.62e-3	0
	16	2.48e-4	-2.68e-1	1.68e-1	-1.18e-2	0	0
	17	3.32e-4	-2.90e-1	1.04e-1	0	0	0
	18	8.94e-4	-4.36e-1	0	0	0	0
			$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi_+}}}$	$\gamma^{\frac{\delta\Psi}{\delta\varphi_{\phi^2_-}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{f(\phi)_+}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\phi^2_-}}}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{f(\phi)_-}}}$
$\mathcal{B}_2$	32	9.79e-5	-4.99e-1	-4.48e-1	-2.40e-1	8.49e-1	2.02e-1
	33	5.39e-4	-5.61e-1	-6.14e-2	-6.33e-2	4.34e-2	0
	34	5.54e-4	-5.51e-1	-2.15e-2	-5.04e-2	0	0
	35	6.83e-4	-3.80e-1	-1.79e-2	0	0	0
	36	8.94e-4	-4.36e-1	0	0	0	0
			$\varphi_{f'(\phi)_+}$	$\varphi_{\Delta\phi_-}$	φ_{ϕ_-}	$\varphi_{\phi^3_+}$	$\gamma^{\frac{\delta\Psi_+}{\delta\varphi_{\Delta\phi_-}}}$
$\mathcal{B}_3$	50	6.33e-10	-1.00e+0	-9.90e-4	1.67e-4	-2.10e-4	-1.04e-3
	51	6.52e-10	-1.00e+0	-9.90e-4	1.28e-4	-1.50e-4	0
	52	8.32e-10	-1.00e+0	-9.90e-4	5.31e-5	0	0
	53	8.94e-10	-1.00e+0	-9.90e-4	0	0	0
	54	1.05e-4	-9.51e-1	0	0	0	0

Table 1: The last 5 iterations of stepwise regression in system identfication of Allen Cahn dynamics.

```
examples.non_local_calculus.Example2_Allen_Cahn.estimate_derivative.py
```

Basis set $\mathcal{B}_3$ is further expanded to include the global observables. For each choice of basis set, a parsimonious model is identified using the stepwise regression strategy via the code:

```
examples.non_local_calculus.Example2_Allen_Cahn.train_model.py
```

The training data is provided in terms of 100 trajectories, each estimated with respect to a different initial condition with parameters, $M_\phi = 1e-3$, and $\lambda = 1$. The resulting coefficients of the trained model are presented in Table 1 with the loss curves provided in Fig. 14. We observe that a familiar gradient flow model is recovered as a 1-term model in the case of basis sets, $\mathcal{B}_1$ and $\mathcal{B}_2$. However, in the case of $\mathcal{B}_3$, the loss is considerably lower for models with 2 and more terms. It is also observed that the 2-term model, in case of $\mathcal{B}_3$, approximates an analytical model for this global variable given as [12]:

$$\frac{d\varphi_{\phi_+}}{dt} = -M_\phi \varphi_{f'(\phi)_+} \pm \lambda M_\phi \varphi_{\Delta\phi_\pm} \equiv 4M_\phi \varphi_{\phi_+} - 4M_\phi \varphi_{\phi^3_+} \pm \lambda M_\phi \varphi_{\Delta\phi_\pm} \tag{39}$$

5 Installation, documentation, and contributing

In this section, we briefly discuss the installation steps and documentation, as well as suggest how users may contribute to the library. Users can refer to specific pages of our online documentation for detailed instructions.

5.1 Installation

`mechanoChemML` is publicly available on GitHub [57] and is hosted on the Python Package Index (PyPI, https://pypi.org/). To install `mechanoChemML`, we recommend the use of `Anaconda`. One Can create a virtual Python package installation environment that is isolated from the Python environment of the operating system with `Anaconda`.

```
$(base) conda create --name mechanochemml python==3.7
$(base) conda activate mechanochemml
```

One can use the following command to install `mechanoChemML` and its required libraries.

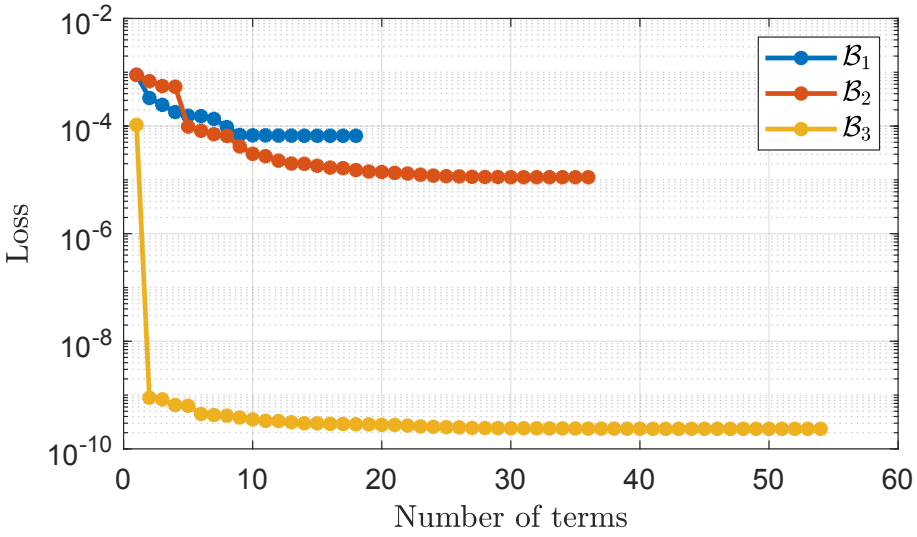


Figure 14: Loss during stepwise regression in system identfication of Allen Cahn dynamics.

```
$(mechanochemml) pip install mechanoChemML
```

To install the proper `TensorFlow` and `TensorFlow Probability` version that is compatible with the CUDA version on the user's system, one needs to download the examples provided by the `mechanoChemML` library via

```
$(mechanochemml) svn export https://github.com/mechanoChem/mechanoChemML/trunk/
    examples ./examples
```

or download the whole library from GitHub via

```
$(mechanochemml) git clone https://github.com/mechanoChem/mechanoChemML.git
    mechanoChemML-master
```

and run the TensorFlow installation script as

```
$(mechanochemml) python3 examples/install_tensorflow.py
```

For developers, one can compile the `mechanoChemML` library and install it locally to reflect the latest GitHub changes that are not available in the released version on the PyPI by the following commands

```
$(mechanochemml) git clone https://github.com/mechanoChem/mechanoChemML.git
    mechanoChemML-master
$(mechanochemml) cd mechanoChemML-master/
$(mechanochemml) python3 setup.py bdist_wheel sdist
$(mechanochemml) pip3 install -e .
```

The newly compiled `mechanoChemML` library will overwrite the old installed version.

5.2 Documentation

The detailed documentation of the `mechanoChemML` library is provided at `https://mechanochemml.readthedocs.io/en/latest/index.html`. One can use the following commands to further compile a local copy of the documentation files.

```
$(mechanochemml) cd mechanoChemML-master/docs
$(mechanochemml) make html
```

5.3 Contributing

Instructions for contributing to the mechanoChemML library, such as bug reports, code contribution, documentation contribution, workflow contribution, etc., is discussed at https://mechanochemml.readthedocs.io/en/latest/contribute.html.

6 Conclusion

Our goal with this communication is to motivate the niche that exists for scientific software between traditional PDE solver libraries and machine learning platforms, and to describe, with an appropriate degree of detail, how the framework of mechanoChemML occupies this position. In addition to code for the machine learning classes, an important idea here is that of machine learning workflows. Laid out with the proper abstraction, these workflows can accommodate a reasonable range of applications in computational materials physics. Moving forward, this library structure will undergo continuous development driven by both: the applications and the evolving understanding of machine learning workflows.

7 Acknowledgments

We gratefully acknowledge the support of Toyota Research Institute, Award #849910: "Computational framework for data-driven, predictive, multi-scale and multi-physics modeling of battery materials". This work has also been supported in part by National Science Foundation DMREF grant #1729166, "Integrated Framework for Design of Alloy-Oxide Structures". Additional support was provided by Defense Advanced Research Projects Agency (DARPA) under Agreement No. HR0011199002, "Artificial Intelligence guided multi-scale multi-physics framework for discovering complex emergent materials phenomena". Computing resources were provided in part by the National Science Foundation, United States via grant 1531752 MRI: Acquisition of Conflux, A Novel Platform for Data-Driven Computational Physics (Tech. Monitor: Ed Walker). This work also used the Extreme Science and Engineering Discovery Environment (XSEDE) Comet at the San Diego Supercomputer Center and Stampede2 at The University of Texas at Austin's Texas Advanced Computing Center through allocation TGMSS160003 and TG-DMR180072.

Data availability

The raw data required to reproduce these findings are available to download from https://github.com/mechanoChem/mechanoChemML. The processed data required to reproduce these findings are available to download from https://github.com/mechanoChem/mechanoChemML.

References

[1] Bikash Kanungo, Paul M Zimmerman, and Vikram Gavini. Exact exchange-correlation potentials from ground-state electron densities. *Nature communications*, 10(1):1–9, 2019.

[2] Wujie Wang and Rafael Gómez-Bombarelli. Coarse-graining auto-encoders for molecular dynamics. *npj Computational Materials*, 5(1):1–9, 2019.

[3] Anirudh Raju Natarajan, John C. Thomas, Brian Puchala, and Anton Van der Ven. Symmetry-adapted order parameters and free energies for solids undergoing order-disorder phase transitions. *Phys. Rev. B*, 96:134204, Oct 2017. doi: 10.1103/PhysRevB.96.134204. URL https://link.aps.org/doi/10.1103/PhysRevB.96.134204.

[4] Gregory H. Teichert and Krishna Garikipati. Machine learning materials physics: Surrogate optimization and multi-fidelity algorithms predict precipitate morphology in an alternative to phase field dynamics. *Computer Methods in Applied Mechanics and Engineering*, 344:666 – 693, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2018.10.025. URL http://www.sciencedirect.com/science/article/pii/S0045782518305292.

[5] G.H. Teichert, A.R. Natarajan, A. Van der Ven, and K. Garikipati. Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. *Computer Methods in Applied Mechanics and Engineering*, 353:201 – 216, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.05.019. URL http://www.sciencedirect.com/science/article/pii/S0045782519302889.

[6] G.H. Teichert, A.R. Natarajan, A. Van der Ven, and K. Garikipati. Scale bridging materials physics: Active learning workflows and integrable deep neural networks for free energy function representations in alloys. *Computer Methods in Applied Mechanics and Engineering*, 371:113281, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113281. URL `http://www.sciencedirect.com/science/article/pii/S0045782520304667`.

[7] Gregory H Teichert, Sambit Das, Muratahan Aykol, Chirranjeevi Gopal, Vikram Gavini, and Krishna Garikipati. Li _x coo _2 phase stability studied by machine learning-enabled scale bridging between electronic structure, statistical mechanics and phase field theories. *arXiv preprint arXiv:2104.08318*, 2021.

[8] Ari L Frankel, Reese E Jones, Coleman Alleman, and Jeremy A Templeton. Predicting the mechanical response of oligocrystals with deep learning. *Computational Materials Science*, 169:109099, 2019.

[9] Kun Wang and WaiChing Sun. A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. *Computer Methods in Applied Mechanics and Engineering*, 334:337–380, 2018.

[10] BA Le, Julien Yvonnet, and Q-C He. Computational homogenization of nonlinear elastic materials using neural networks. *International Journal for Numerical Methods in Engineering*, 104(12):1061–1084, 2015.

[11] Xiaoxuan Zhang and Krishna Garikipati. Machine learning materials physics: Multi-resolution neural networks learn the free energy and nonlinear elastic response of evolving microstructures. *Computer Methods in Applied Mechanics and Engineering*, 372:113362, 2020. ISSN 00457825. doi: 10.1016/j.cma.2020.113362. URL `http://arxiv.org/abs/2001.01575https://doi.org/10.1016/j.cma.2020.113362`.

[12] Matthew Duschenes and Krishna Garikipati. Reduced order models from computed states of physical systems using non-local calculus on finite weighted graphs, 2021.

[13] Xiaoxuan Zhang and Krishna Garikipati. Bayesian neural networks for weak solution of PDEs with uncertainty quantification. *arXiv:2101.04879*, pages 1–37, 2021. URL `http://arxiv.org/abs/2101.04879`.

[14] Yinhao Zhu and Nicholas Zabaras. Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. *J. Comput. Phys.*, 366:415–447, 2018.

[15] Yinhao Zhu, Nicholas Zabaras, Phaedon Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *J. Comput. Phys.*, 394:56–81, 2019.

[16] Nick Winovich, Karthik Ramani, and Guang Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *J. Comput. Phys.*, 394: 263–279, 2019.

[17] Saakaar Bhatnagar, Yaser Afshar, Shaowu Pan, and Karthik Duraisamy. Prediciton of Aerodynamic Flow Fields Using Convolutional Neural Networks. *Comput. Mech.*, 5:1–30, 2019.

[18] Angran Li, Ruijia Chen, Amir Barati Farimani, and Yongjie Jessica Zhang. Reaction diffusion system prediction based on convolutional neural network. *Sci. Rep.*, 10:1–9, 2020.

[19] Isaac Lagaris, Aristidis Likas, and Dimitrios I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9:987–1000, 1998.

[20] Jiequn Han, Arnulf Jentzen, Weinan E, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115:8505–8510, 2018.

[21] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[22] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

[23] Nicholas Geneva and Nicholas Zabaras. Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks. *J. Comput. Phys.*, 403:109056, 2020.

[24] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comput. Phys.*, 425:109913, 2021.

[25] Jens Berg and Kaj Nystroem. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[26] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[27] Francois Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[29] Satish Balay, Shrirang Abhyankar, Mark Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, W Gropp, et al. Petsc users manual, 2019.

[30] Michael A Heroux, Roscoe A Bartlett, Vicki E Howle, Robert J Hoekstra, Jonathan J Hu, Tamara G Kolda, Richard B Lehoucq, Kevin R Long, Roger P Pawlowski, Eric T Phipps, et al. An overview of the trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423, 2005.

[31] Giovanni Alzetta, Daniel Arndt, Wolfgang Bangerth, Vishal Boddu, Benjamin Brands, Denis Davydov, Rene Gassmöller, Timo Heister, Luca Heltai, Katharina Kormann, et al. The deal. ii library, version 9.0. *Journal of Numerical Mathematics*, 26(4):173–183, 2018.

[32] Martin Alnæs, Jan Blechta, Johan Hake, August Johansson, Benjamin Kehlet, Anders Logg, Chris Richardson, Johannes Ring, Marie E Rognes, and Garth N Wells. The fenics project version 1.5. *Archive of Numerical Software*, 3(100), 2015.

[33] G.H. Teichert, S. Rudraraju, Z. Wang, X. Zhang, and K. Garikipati. mechanochem software package. `https://github.com/mechanoChem`, 2020. URL `https://github.com/mechanoChem`. (accessed December 3, 2021).

[34] Z. Wang, X. Huan, and K. Garikipati. Variational system identification of the partial differential equations governing the physics of pattern-formation: Inference under varying fidelity and noise. *Computer Methods in Applied Mechanics and Engineering*, 356:44 – 74, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2019.07.007.

[35] Z. Wang, X. Huan, and K. Garikipati. Variational system identification of the partial differential equations governing microstructure evolution in materials: Inference over sparse and spatially unrelated data. *Computer Methods in Applied Mechanics and Engineering*, 377:113706, 2021. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.113706. URL `https://www.sciencedirect.com/science/article/pii/S0045782521000426`.

[36] Z. Wang, X. Zhang, G.H. Teichert, M. Carrasco-Teja, and K. Garikipati. System inference for the spatio-temporal evolution of infectious diseases: Michigan in the time of covid-19. *Computational Mechanics*, 66:1177, 2020.

[37] Zhenlin Wang, Mariana Carrasco-Teja, Xiaoxuan Zhang, Gregory H. Teichert, and Krishna Garikipati. System Inference Via Field Inversion for the Spatio-Temporal Progression of Infectious Diseases: Studies of COVID-19 in Michigan and Mexico. *Archives of Computational Methods in Engineering*, 28(6):4283–4295, 2021. doi: 10.1007/s11831-021-09643-1. URL `https://doi.org/10.1007/s11831-021-09643-1`.

[38] Z. Wang, J.B. Estrada, E.M. Arruda, and K. Garikipati. Discovery of deformation mechanisms and constitutive response of soft material surrogates of biological tissue by full-field characterization and data-driven variational system identification. *bioRxiv*, 2020. doi: 10.1101/2020.10.13.337964.

[39] R. Banerjee, K. Sagiyama, G. Teichert, and K. Garikipati. A graph theoretic framework for representation, exploration and analysis on computed states of physical systems. *Comput. Methods Appl. Mech. Eng.*, 351: 501–530, jul 2019. ISSN 0045-7825. doi: 10.1016/J.CMA.2019.03.053.

[40] R Banerjee, K Sagiyama, GH Teichert, and K Garikipati. A graph theoretic framework for representation, exploration and analysis on computed states of physical systems. *Computer Methods in Applied Mechanics and Engineering*, 351:501–530, 2019.

[41] G. Gilboa and S. Osher. Nonlocal operators with applications to image processing. *Multiscale Model. Simul.*, 7 (3):1005–1028, 2008. ISSN 15403459. doi: 10.1137/070698592.

[42] X. Desquesnes, A. Elmoataz, and O. Lézoray. Eikonal Equation Adaptation on Weighted Graphs: Fast Geometric Diffusion Process for Local and Non-local Image and Data Processing. *J Math Imaging Vis*, 46:238–257, 2013. doi: 10.1007/s10851-012-0380-9.

[43] M. Hein, J. Audibert, and U. von Luxburg. Graph Laplacians and their Convergence on Random Neighborhood Graphs. *J. Mach. Learn. Res.*, 8:1325–1368, 2007.

[44] Burr Settles. *Active learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[45] Z. Wang, B. Wu, K. Garikipati, and X. Huan. A perspective on regression and bayesian approaches for system identification of pattern formation dynamics. *Theoretical & Applied Mechanics Letters*, 10(3):188, 2020. ISSN 2095-0349. doi: 10.1016/j.taml.2020.01.028.

[46] Z. Wang, B. Martin, J. Weickenmeier, and K. Garikipati. An inverse modelling study on the local volume changes during early morphoelastic growth of the fetal human brain. *Brain Multiphysics*, 2:100023, 2021. ISSN 2666-5220. doi: https://doi.org/10.1016/j.brain.2021.100023. URL `https://www.sciencedirect.com/science/article/pii/S2666522021000034`.

[47] 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.*, 113, 2016.

[48] N. M. Mangan, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Inferring biological networks by sparse identification of nonlinear dynamics. *IEEE Trans. Mol. Biol. Multi-Scale Commun.*, 2, 2016.

[49] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Sci. Adv.*, 3, 2017.

[50] N. M. Mangan, T. Askham, S. L. Brunton, J. N. Kutz, and J. L. Proctor. Model selection for hybrid dynamical systems via sparse regression. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475(2223):20180534, 2019. doi: 10.1098/rspa.2018.0534.

[51] M. Quade, M. Abel, J. N. Kutz, and S. L. Brunton. Sparse identification of nonlinear dynamics for rapid model recovery. *Chaos*, 28, 2018.

[52] E. Kaiser, J. N. Kutz, and S. L. Brunton. Discovering conservation laws from data for control. *IEEE Conference on Decision and Control*, 2018.

[53] K. P. Champion, S. L. Brunton, and J. N. Kutz. Discovery of nonlinear multiscale systems: Sampling strategies and embeddings. *SIAM J. Appl. Dyn. Syst.*, 18, 2019.

[54] J. Schnakenberg. Network theory of microscopic and macroscopic behavior of master equation systems. *Rev. Mod. Phys.*, 48, 1976.

[55] A. M. Turing. The chemical basis of morphogenesis. *Phil. Trans. Roy. Soc. Lond. Ser. B.*, 237, 1952.

[56] S. M. Allen and J. W. Cahn. A microscopic theory for antiphase boundary motion and its application to antiphase boundary coarsening. *Acta Metallurgica*, 27:1085–1091, 1979.

[57] X. Zhang, G.H. Teichert, Z. Wang, M. Duschenes, S. Srivastava, A. Sunderarajan, E. Livingston, J. Holber, M. Shojaei, and K. Garikipati. mechanochemml software package. `https://github.com/mechanoChem/mechanoChemML`, 2022. URL `https://github.com/mechanoChem/mechanoChemML`. (accessed April 29, 2022).