Journal of Computational Physics 420 (2020) 109719

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Mesoscale informed parameter estimation through machine learning: A case-study in fracture modeling

Nishant Panda [a,*], Dave Osthus [b], Gowri Srinivasan [c], Daniel O'Malley [d], Viet Chau [e], Diane Oyen [f], Humberto Godinez [a]

[a] T-5, Applied Mathematics and Plasma Physics, Los Alamos National Lab, Los Alamos, NM 87545, United States of America
[b] CCS-6, Statistical Sciences, Los Alamos National Lab, Los Alamos, NM 87545, United States of America
[c] XCP-8, Computational Physics, Los Alamos National Lab, Los Alamos, NM 87545, United States of America
[d] EES-16, Computational Earth Science, Los Alamos National Lab, Los Alamos, NM 87545, United States of America
[e] EES-17, Earth and Environmental Sciences Division, Los Alamos National Lab, Los Alamos, NM 87545, United States of America
[f] CCS-3, Information Sciences, Los Alamos National Lab, Los Alamos, NM 87545, United States of America

ARTICLE INFO

Article history:
Received 22 October 2019
Received in revised form 29 May 2020
Accepted 12 July 2020
Available online 20 July 2020

Keywords:
Uncertainty quantification
Reduced order model
Machine learning
Data driven upscaling
Probabilistic emulator
Fracture propagation

ABSTRACT

Scale bridging is a critical need in computational sciences, where the modeling community has developed accurate physics models from first principles, of processes at lower length and time scales that influence the behavior at the higher scales of interest. However, it is not computationally feasible to incorporate all of the lower length scale physics directly into upscaled models. This is an area where machine learning has shown promise in building emulators of the lower length scale models, which incur a mere fraction of the computational cost of the original higher fidelity models. We demonstrate the use of machine learning using an example in materials science estimating continuum scale parameters by emulating, with uncertainties, complicated mesoscale physics. We describe a new framework to emulate the fine scale physics, especially in the presence of microstructures, using machine learning, and showcase its usefulness by providing an example from modeling fracture propagation. Our approach can be thought of as a data-driven dimension reduction technique that yields probabilistic emulators. Our results show well-calibrated predictions for the quantities of interests in a low-strain simulation of fracture propagation at the mesoscale level. On average, we achieve ∼10% relative errors on time-varying quantities like total damage and maximum stresses. Successfully replicating mesoscale scale physics within the continuum models is a crucial step towards predictive capability in multi-scale problems.

Published by Elsevier Inc.

1. Introduction

Many observable physical phenomena are often determined by interactions of physical processes occurring across a wide range of scales. Most often, we are interested in the macro-scale behavior of the physical system, and we upscale the finer scale effects, which appear in the PDE as a heterogeneous parameter through some constitutive relation. However, practical real-time simulations/experiments of fine-scale physics are intensive. Thus, the main question we address in this paper

* Corresponding author.
E-mail address: nishant.panda@gmail.com (N. Panda).

https://doi.org/10.1016/j.jcp.2020.109719
0021-9991/Published by Elsevier Inc.

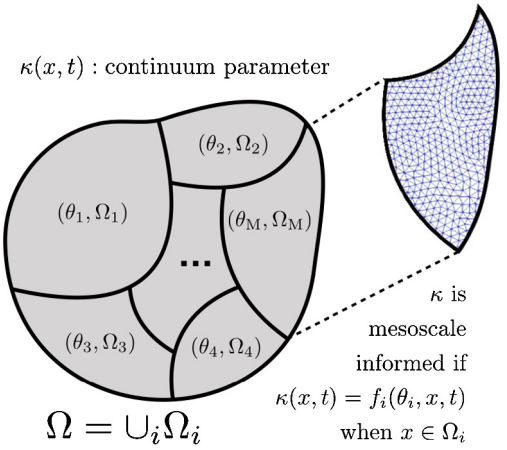

Fig. 1. Shown here is the domain, Ω, of a continuum scale simulation/experiment decomposed into the union of mesoscale domains Ω_i. A continuum parameter $\kappa(\mathbf{x}, t)$ can be mesoscale informed if $\kappa|_{\Omega_i}$ is learnt using a mesoscale simulation. That is we say that $\kappa(\mathbf{x}, t)$ is mesoscale informed if Ω_i is a mesoscale domain with some parameters θ_i such that $\kappa|_{\Omega_i} = f_i(\theta_i)$. The inset to the right of the figure shows that θ_i comes from a high dimensional mesoscale simulation or experiment. For example κ could be a stress strain relation and Ω_i could depict a mesoscale domain with pre-existing fractures or cracks. Thus, $\kappa|_{\Omega_i}$ would be the stress strain relationship in the presence of fractures.

can be stated as follows: How can fine-scale effects be emulated in a data-driven framework to inform continuum scale parameters efficiently while also accounting for uncertainty? While this is a general upscaling framework, we focus on the particular case of upscaling mesoscale information to the continuum or macroscale. This central question is illustrated in Fig. 1, where we show a continuum scale domain composed of many mesoscale domains. A continuum parameter $\kappa(x, t)$ is mesoscale informed if in each mesoscale domain Ω_i, $\kappa(x, t) = f_i(\theta_i)$ where θ_i are mesoscale parameters.

Multi-scale modeling is a topic of great interest to the simulation and modeling community. Researchers have developed accurate physics models from first principles, in many cases, of processes at lower length and time scales that influence behavior at the higher scales of interest. The challenges of including lower length and time scale information at higher scales are primarily driven by computational cost considerations. While this study focuses on the interactions and propagation of fractures in brittle materials, the principles of multi-scale modeling developed and discussed here are widely applicable. Micro-structural information plays a key role in governing the dominant physics for fracture propagation leading to damage in brittle materials. Brittle materials are materials in which cracks of atomic sharpness propagate by bond ruptures. Failure in brittle materials is a key concern for several applications including in airplane wings, ceramics, and shale formations in the sub-surface. Unlike ductile materials, which fail through the process of necking, brittle solids fail with little warning along grain boundaries, or in the case of geo-materials, fractures propagate along interstitials. Once fracturing has begun, the fractures have a strong tendency to propagate since they are driven by internally stored elastic energy. Accurately predicting the time evolution of fractures leading to failure at the macroscale requires knowledge of the dynamically evolving microstructure, including features such as grain boundaries and interstitials, and also accounting for fracture interactions with the microstructure. See [1–4] and references therein for information on brittle material fracture. Refer to [5] for information about fracture mechanics. Many approaches have been taken to address the failure of a material at different length scales, including analytical and numerical methods. See, e.g., [6] and references therein.

Modeling micro- and meso-scale fracture mechanics is computationally expensive and cannot be directly applied to large components or systems crucial for many applications. Instead, the physics on these scales must be averaged or scaled up and incorporated into continuum models used at the macroscale. Traditional approaches either ignore or idealize micro-scale information at the larger scales because we lack a framework that is capable of efficiently utilizing it in its entirety. As a result, predictions using simulations often fail to match experimental data collected at the higher length scales. This is an area where machine learning (ML) methods offer the promise of several orders of magnitude speedup in mimicking the lower length scale physics accurately.

The use of ML in upscaling is being widely considered with promising preliminary results in modeling continuum processes such as material strength, damage, equation of state for metals and gases, turbulence, material informatics etc [7–9] ML is a field that is rapidly evolving and enables computers to learn based on data. The data can come from various sources, including physical experiments, computer models, or a combination of both. There have been successful applications of ML across many fields. Here we list some relevant articles that have used ML to emulate or discover physics [10–16]. Note that, reduced order methods (ROM) [17–20] that are not considered as traditional ML methods are also being used increasingly as surrogate models that mimic the relevant aspects of the physics. As such, our method here could be thought of as a probabilistic reduced order method. We use ML, ROM, emulator, and surrogate interchangeably in this paper.

In this study, we take advantage of recent advances in machine learning to develop a computationally efficient ROM of a high fidelity model that accounts for the evolution of discrete fractures. We describe a general data-driven framework for parameter estimation in continuum physics model. We then illustrate the framework by emulating a high dimensional

mesoscale fracture simulation. Our approach efficiently tackles a broad set of problems in fractured systems, where structure and topology are critical, while seamlessly lending itself to an uncertainty quantification (UQ) framework that requires a fraction of the computational resources. We demonstrate that our ROM mimics the high fidelity model for a given set of initial and boundary conditions with a computational speed up of approximately 4 orders of magnitude in the log scale.

It is extremely challenging to build emulators of fine scale physics with micro-structures that also provide uncertainty information. The key contribution of this paper is to highlight the important areas in building a data-driven surrogate for such models that also accounts for uncertainty. While we note that the novelty of our method is not in the approach of discovering the underlying physics as described in recent ML methods [12,14] or in discovering relevant projections of the dynamics as in recent reduced order methods [21,13]; our method describes a unified framework to upscale fine-scale physics through emulation (either through experiments or simulation). In Section 2, we describe this framework in the abstract, illustrating the four main components of the framework. Then in Section 3, we illustrate the ideas described in the previous section through the application of emulating mesoscale fracture propagation in a brittle material. In Section 4, we present the results of our approach to the case of emulating mesoscale fracture in a low-strain setting in the presence of random mesoscale cracks in a brittle material. We showcase four metrics to validate our approach. Finally, we end with a conclusion of our method and present areas of future work.

2. A data driven framework

In this section, we describe a general framework for emulating mesoscale physics-based models that will be used to estimate parameters in the continuum scale. Typically, physics-based models are dynamical systems that describe the evolution of some *state variables* in a distinctive spatiotemporal scale. The parameter space for physical models consists of parameters that describe field quantities, boundary conditions, initial conditions, and forcing conditions, while functions of the state variables are quantities of interests (QoI) of the model. There are many different ways to construct an emulator. One such approach is to fit a functional approximation from the parameter space to the QoIs of the model. These emulators are quite useful when the QoIs are fixed or known *a priori*, and the parameter space is not large. A different approach is to build a ROM for the underlying dynamical system itself. Such surrogates capture a lower-dimensional subspace where the dynamical system evolves. The QoIs do not have to be known *a priori*, and the surrogate can be used as a proxy for the model. As alluded to in the introduction, we employ this second approach wherein we emulate the dynamical system (with data from experiments or numerical simulation) representing the mesoscale model by constructing a *probabilistic ROM*. While being entirely data-driven, the framework has enough flexibility to incorporate physics (e.g., loss functions can be developed to enforce physics constraints like conservation laws, symmetric structure, etc.). While existing literature regarding the probabilistic description of the dynamical system is described in fields such as state space modeling, Markov models etc. (see for example [22,23]), we outline a general framework for emulating fine scale physics for multi-physics and multi-scale simulation that have been solved using numerical methods. We also note that there has been recent work regarding machine learning methods that are physics agnostic. While our framework allows for physics to be incorporated, it is not a requirement.

Let $\mathbf{s} = \{s_1, s_2, \ldots, s_m\}$ denote the set of states of the dynamical system defined on some domain Ω where each s_i is a time varying field defined on the spatial domain Ω. Thus, if $\vec{x}$ and t denote the space and time variables, our set of state $\mathbf{s}$ is $\{s_1(\vec{x}, t), s_2(\vec{x}, t), \ldots, s_m(\vec{x}, t)\}$. For example, in a three dimensional Navier-Stokes equations, the three components of velocity u, v, w could be the state of the system s_1, s_2, s_3 respectively. Similarly in fractured systems, the damage of the material, the stress and strain tensor could be the state of the system. We will consider a discrete dynamical system from initial time t_0 to some final time t_N, given by

$$\mathbf{s}_t = \mathcal{M}_t(\mathbf{s}_{t-1}, \theta), \tag{1}$$

where $\mathcal{M}_t$ represents the model evolution operator at the (discrete) time t,

$$\mathbf{s}_t = \{s_{1,t}(\vec{x}), s_{2,t}(\vec{x}), \ldots, s_{m,t}(\vec{x})\}$$

denotes the set of states at time t and θ represents all the parameters of the model like boundary conditions, forcing terms, initial conditions, constitutive parameters etc. In constructing an emulator for equation (1), we will view the state and parameters as random variables in some probabilistic space. In the following paragraphs we will elaborate on this central idea.

Most mesoscale models are either solved through numerical methods (like finite element, finite volume, etc.) or correspond to experiments where the state is approximated by a high dimensional function. In the context of emulating such state evolution, the high dimensional structure can be quite restricting. Moreover, we often do not need the full state evolution but only functionals on the state space. With this in mind, we outline the important ingredients for a data-driven approach to emulate high-fidelity mesoscale models below.

2.1. Mesoscale coarsening

As alluded to, the state space is typically very large that makes the emulation process challenging. Here, we outline a coarsening process that reduces the dimension of the state to be emulated. Let $\mathbf{s_t} = \{s_{1,t}, s_{2,t}, \ldots, s_{m,t}\}$ be the state of a

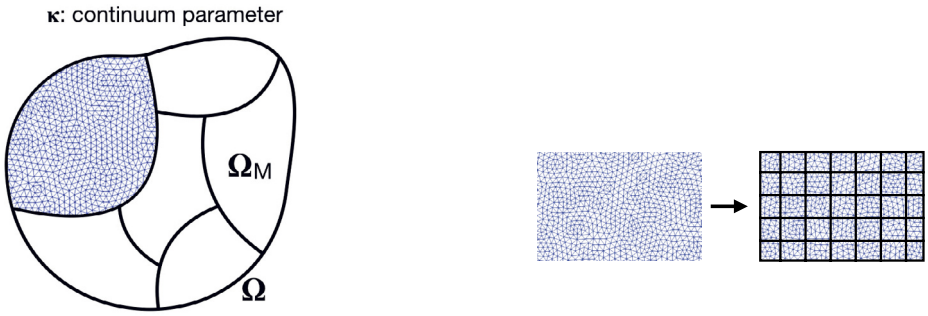

Fig. 2. An illustration of mesoscale coarsening. The blue mesh in each figure on the left and right represents the high-dimensional (d-dimensional) state space while the regions (R_c) outlined in black are the coarse (C-dimensional) regions. The coarse state is approximated as piecewise constant within each cell of the coarse partition. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

mesoscale model defined on a spatial domain Ω at time t where each $s_{i,t}$ can be identified with vector in $\mathbb{R}^d$. Consider a partition of Ω given by,

$$\Omega = \bigcup_{c=1}^{C} R_c,$$

where the number of partition $C \ll d$. For example, C can be chosen such that each partition R_i contains approximately 100 finite elements. This process is illustrated in Fig. 2.

A coarsened state on Ω denoted by $\widehat{\mathbf{s_t}} = \{\widehat{s_{1,t}}, \widehat{s_{2,t}}, \ldots, \widehat{s_{m,t}}\}$ is given by the following simple function approximation

$$\widehat{s_{k,t}}(\vec{x}) = \sum_{c=1}^{C} g_{k,c}(s_{k,t}) \mathbb{I}_{R_c}(\vec{x}),$$

where $g_{k,c}(s_{k,t})$ is a coarsening function of the state $s_{k,t}$ and $\mathbb{I}_{R_c}(\vec{x})$ is the indicator or characteristic function of the partitioned cell R_c, i.e., $\mathbb{I}_{R_c}(\vec{x}) = 1$ if $\vec{x}$ is inside the cell R_i, otherwise $\mathbb{I}_{R_c}(\vec{x}) = 0$. Some examples of the coarsening functions $g_{k,c}$ could be the average, maximum, minimum etc. of all $s_{k,t}$ that are in a partition R_c. Thus, while our true state $s_{k,t}$ is identified as a vector in $\mathbb{R}^d$, our coarsened state $\widehat{s_{k,n}}$ is identified by a vector in $\mathbb{R}^C$ where $C \ll d$. The choice of coarsening function as well the number of divisions will affect both the construction and performance of the emulator. These choices should come from knowledge of the underlying micro-mesoscale physics. While we have stuck to a simple rectangular partition rather than a more geometrically consistent shape, we want to note that in real time predictions we will usually approximate mesoscale domains by a structured representation like a rectangle or cuboid even though they can take any shape conceptually.

2.2. Transition model

Once we have coarsened our state space, we need to prescribe a transition model for our emulator that describes how the coarsened state at time t evolves from past states. While machine learning methods are being developed that also learn the transitional structure [11][24], in our experience, imposing a constrained transition structure is a way to bring physics into the learning process. Since our ultimate goal is to emulate, this is not necessarily a restriction on the algorithm.

Fig. 3 shows the graphical representation of the coarsened state space, with $K = 4$ rows (states) and $N = 3$ columns (discrete time points). When defining transition models on the coarsened state space, we restrict our attention to transition models of the following form:

$$\widehat{s_{k,t}} = f_k(\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}) \tag{2}$$

where $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}} \subseteq \{\widehat{s_{k',t'}} | t' \leq t \text{ and } 1 \leq k' \leq K\}$ is a subset of the set of coarsened states $\widehat{s_{k',t'}}$ that $\widehat{s_{k,t}}$ depends on through the transition function $f_k()$ that will be learned through machine learning. Furthermore, when Fig. 3 is viewed as a directed graph $\mathcal{G}(V, E)$ where $V = \{\widehat{s_{1,1}}, \widehat{s_{1,2}}, \ldots, \widehat{s_{K,t_N}}\}$ is the set of vertices and E is the set of directed edges, we require $\mathcal{G}(V, E)$ to be a directed, acyclic graph (DAG). The correspondence between $\widehat{s_{k,t}}$ and $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}$ is that the set of coarsened states $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}$ equals the parents of vertex $\widehat{s_{k,t}}$. The DAG structure is crucial since it enables us to encode temporal dependency in a fashion consistent with the physical process. Also, by encoding a DAG structure, we can do Bayesian inference on the full state variables, for e.g. see [23] and the references therein.

The simplest example of a transition model would represent a state-independent, first-order Markov transition model where

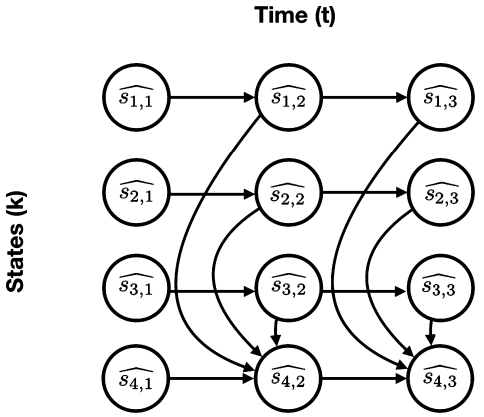

Fig. 3. An example graphical representation of the coarsened state space, with $K = 4$ states and $N = 3$ discrete time points is shown. The transition model is depicted by directed arrows i.e. transition models can be described as a directed acyclic graph (DAG) where $\widehat{s_{k,t}}$ only depends on states $\{\widehat{s_{k',t'}} | t' \leq t$ and $1 \leq k' \leq K\}$.

$$\widehat{s_{k,t}} = f_k(\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}) = f_k(\widehat{s_{k,t-1}}) \tag{3}$$

indicating that $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}} = \widehat{s_{k,t-1}}$ for $t > 1$. This is a common transition model assumption if information is not shared across states k and the state $\widehat{s_{k,t}}$ is assumed to only depend on its immediately preceding state in time. Our framework, due to its generality, can accommodate much more complex conditioning sets in the transition model. For clarity, Fig. 3 shows an example of transition model conditioning sets for $K = 4$ states and $N = 3$ distinct time points. For $t > 1$,

$$\widehat{\mathbf{u}_{\mathbf{1},\mathbf{t}}} = \{\widehat{s_{1,t-1}}\},$$
$$\widehat{\mathbf{u}_{\mathbf{2},\mathbf{t}}} = \{\widehat{s_{2,t-1}}\},$$
$$\widehat{\mathbf{u}_{\mathbf{3},\mathbf{t}}} = \{\widehat{s_{3,t-1}}\},$$
$$\widehat{\mathbf{u}_{\mathbf{4},\mathbf{t}}} = \{\widehat{s_{4,t-1}}, \widehat{s_{1,t}}, \widehat{s_{2,t}}, \widehat{s_{3,t}}\}.$$

Note that the graph formed in this example is a DAG (i.e., there are no directed cycles in the graph in Fig. 3) and the conditioning set for $\widehat{s_{k,t}}$, namely $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}$, only depends on coarsened states at earlier or concurrent times. The transition functions are highly dependent on the time scales at which data is gathered for emulation. If shorter time steps are employed, one increases the number of pairs $(t-1, t)$ to learn a transition function, while increasing the time interval we may miss out on important transition physics. The choice of what time step to use for data emulation is thus informed with the physical process involved.

There are many ML techniques to learn this transition model. The basic approach to learning the transition function decomposes into two techniques - 1) projecting the data into a lower-dimensional manifold and 2) evolving the manifold in time. In neural networks, this is done through an encoder-decoder architecture for manifold learning and a temporal architecture (like LSTMs, RNNs) for the temporal evolution [25]. Other ML methods include non-linear PCA for manifold learning and sparse regression for temporal learning [26]. To keep the ideas simple, we employ this later approach to demonstrate machine learning techniques for the transition function.

2.3. Probabilistic graphical model

Our framework allows for full sate UQ if desired. Notationally, we make a few changes. We denote by $S_{k,t}$, the true kth state at time t defined as a random variable on some probabilistic space while $s_{k,t}$ denotes a particular sample of this state. Similarly, we denote by $\widehat{S_{k,t}}$ the coarsened random variable where $\widehat{s_{k,t}}$ denotes a sample of this random variable. By restricting our attention to transition models that correspond to DAGs, there is a straightforward way to write the joint distribution of all coarsened random variables. Namely,

$$[\widehat{S_{1,1}}, \widehat{S_{1,2}}, \ldots, \widehat{S_{K,t}}] = \prod_{k,t} [\widehat{S_{k,t}} | pa(\widehat{S_{k,t}})], \tag{4}$$

where $pa(\widehat{S_{k,t}})$ are the parents of $\widehat{S_{k,t}}$ which equals $\widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}$. Thus the joint distribution over all coarsened random variables can be written as

$$[\widehat{S_{1,1}}, \widehat{S_{1,2}}, \ldots, \widehat{S_{K,t}}] = \prod_{k,t} [\widehat{S_{k,t}} | \widehat{\mathbf{u}_{\mathbf{k},\mathbf{t}}}]. \tag{5}$$

Refer to [23] for a detailed description of constructing a probabilistic graphical model. A common distribution choice for $[\widehat{S_{k,t}}|\widehat{\mathbf{u_{k,t}}}]$ is a Normal distribution, where $\mathbb{E}(\widehat{S_{k,t}}|\widehat{\mathbf{u_{k,t}}})$ is given by $f_k(\widehat{\mathbf{u_{k,t}}})$. That is, the expectation of the random variable $\widehat{S_{k,t}}$ given the conditioning set $\widehat{\mathbf{u_{k,t}}}$ equals the transition model $f_k()$ applied to the conditioning set $\widehat{\mathbf{u_{k,t}}}$. The function $f_k()$ must still be learned using machine learning techniques. For instance, the simplest learning technique is the (multivariate)linear regression, which assumes $f_k(\widehat{\mathbf{u_{k,t}}}) \approx \beta_0 + \sum_{j=1}^{J} \beta_j h_j(\widehat{\mathbf{u_{k,t}}})$ and estimates the unknown parameters β_j for chosen functions $h_j()$ (e.g., polynomials, interactions, etc.). Of course, more expressive machine learning models can be used when needed.

2.4. Quantities of interest

In most cases, full state estimation is not required. The quantities of interest (QoIs) are typically not the full state but functionals on the state. If $s_{k,t}$ is a true state, then

$$y_{k,t} = q_k(s_{k,t})$$

is a quantity of interest where $q_k : \mathbb{R}^d \to \mathbb{R}$ is a functional on the state space. Correspondingly, the coarsened quantity of interest is given by

$$\widehat{y_{k,t}} = q_k\left(\widehat{s_{k,t}}\right),$$

where we employ the same notation for the function q_k. We prescribe a probabilistic model on the quantity of interest

$$Y_{k,t}\left|\widehat{Y_{k,t}} \sim GP\left(\widehat{Y_{k,t}}, K(t,t')\right)\right.,$$

where GP is a Gaussian Process (GP) whose parameters are to be learnt. The GP models the mismatch between the emulated quantity of interest $\widehat{Y_{k,t}}$, where $\widehat{Y_{k,t}} = q_k(\widehat{S_{k,t}}) = q_k(f_k(\widehat{\mathbf{u_{k,t}}}))$ and the true quantity of interest $Y_{k,n}$. The interested reader can consult [27] and the references therein for a detailed explanation on constructing GPs and using data to estimate its parameters. A general GP can also be used where deep neural networks are used to learn a complex kernel. Refer to [28] for details in fitting a flexible kernel. To keep ideas simple we will prescribe a parametric form of the kernel and use data to learn the parameters of the kernel as in [27].

3. Emulating mesoscale fracture propagation

In this section, we illustrate the ideas presented earlier by constructing an emulator of a complex fracture propagation system.

3.1. The high fidelity model: HOSS

The Hybrid Optimization Software Suite (HOSS) [29–31] is a multi-physics software package developed at Los Alamos National Laboratory based on the combined Finite-Discrete Element Method (FDEM). The HOSS model is used to simulate deformation and failure (i.e., fracture and fragmentation) of different materials based on the FDEM [32]. In HOSS, cracks form along the boundaries of the finite elements. To capture fine mechanisms, such as crack nucleation, propagation, branching, reorientation, etc., the crack networks must be finely resolved spatially, with dozens to hundreds of finite elements along the length of each crack [33]. The problem of interest for this work (described in more detail in the next section) is a 2D sample under tension loading. For fracture modeling, HOSS considers two primary modes of failure in 2D: Mode I, which is opening due to tensile load, and Mode II, which is crack growth due to shear loading conditions [32,34]. Since the problem of interest is dominated by Mode I crack growth, we focus this discussion on the key details as to how HOSS accounts for Mode I crack growth. However, it must be pointed out that the Mode II crack growth is handled in a similar way to the Mode I case, except that different sets of parameters are applied. It is worth noting that, although globally Mode I failure dominates this problem, both shear and opening can occur at a local mesh element scale. Between the interface of any two finite elements, there lies a user-specified number of cohesive points (four are used in all simulations presented and discussed here), which are modeled as non-linear springs, as shown in Fig. 4. As the two elements shown in the figure undergo tensile load and are pulled apart, the springs within the interface are strained, resulting in a small space opening between the elements. Similarly, for shear, or Mode II, deformation, four cohesive points describe the shear stresses generated when one element slides relative to its neighbor. A typical behavior of the springs as a function of the opening (displacement) created between the edges of two finite elements is shown in Fig. 4. These cohesive points between elements are responsible for representing the material strength response both in tension and in shear. It is worth noting that the area below the softening portion of the curve, indicated in Fig. 4, represents the specific energy (measured in J/m^2) density dissipated during the fracture process. Fig. 4 only presents a schematic representation of this curve. The actual shape of the curve is found through fitting to experimental results. For more information on FDEM and HOSS, the interested reader is referred to [29,32,34,33].

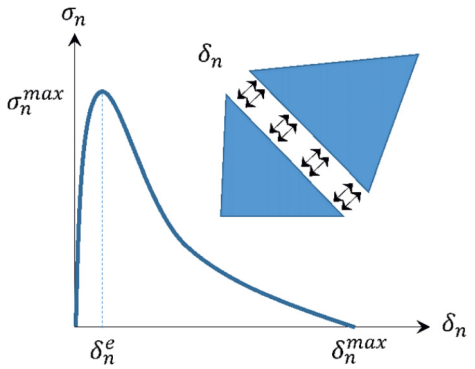

Fig. 4. The cohesive points between elements shown in the right inset are responsible for representing the material strength response both in tension and in shear. An illustration of the softening curve is shown in the left.

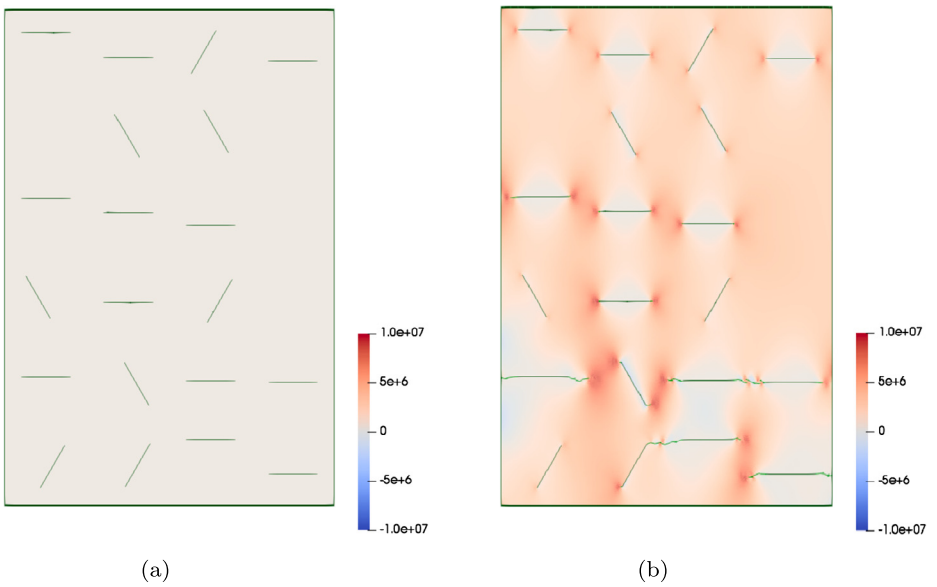

Fig. 5. A typical HOSS simulation at (a) the initiation of loading and (b) after loading has induced crack propagation. The cracks are depicted as white lines. The color palette is used to denote the stress field σ_{22} in Pa units. This particular simulation depicts a rectangular slab of a brittle material (concrete) with 20 pre-existing cracks present that is pulled vertically (tensile loading). Cracks tend to elongate and merge near regions of higher stress.

3.2. Emulating stress, damage, fractures and QoIs

For fractured systems, the main QoIs that we are concerned with from the mesoscale simulation to inform continuum scale models are maximum tensile stress, damage per unit volume, fracture (crack) lengths, and the number of distinct fractures (cracks). While HOSS simulates these quantities, it is computationally expensive to use in real-time prediction (a typical simulation takes 4 hrs with ≥ 400 processors). A typical HOSS simulation at some time t is shown in Fig. 5a and 5b.

We detail all the ingredients of the data-driven framework described in Section 2: the set of states we are interested in emulating, the mesoscale coarsening, the transition model for each state and the quantities of interest. Let $\mathbf{s} = \{\boldsymbol{\sigma}, \mathrm{d}\}$ denote the set of states where $\boldsymbol{\sigma} = \{\sigma_{ij}\}$ is the stress tensor in cartesian co-ordinates and d represents damage. Damage occurs once the strain in Fig. 4 goes beyond the peak in the stress curve (i.e., when the strain exceeds δ_n^e). Beyond this point, the material has a diminished ability to support a load. In $2D$ we will have 3 distinct stress components $\sigma_{11}, \sigma_{12}, \sigma_{22}$ of the Cauchy stress tensor. Note that this a subset of the state variables that HOSS simulates. However, this set is sufficient to inform the continuum scale parameter. Thus, we will have $K = 4$ states in $2D$ that we will emulate. We employ the following coarsening function for stress at time t, $\sigma_{ij,t}$,

$$\widehat{\sigma_{ij,t}}(\vec{x}) = \sum_{c=1}^{C} \max_{R_c}(\sigma_{ij,t}) \mathbb{I}_{R_c}(\vec{x}), \tag{6}$$

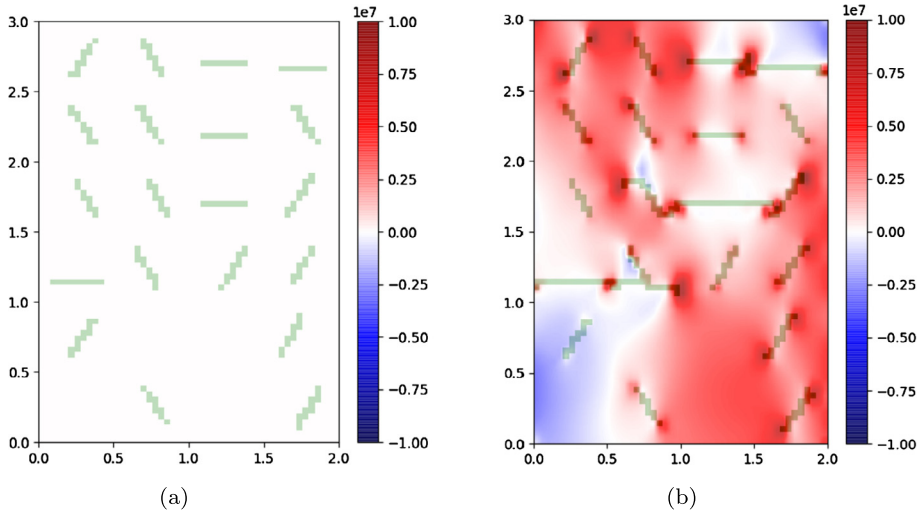

Fig. 6. Coarsened state at two different times. The green color denotes the coarsened damage field while the red/blue color palette is used to denote the coarsened stress ($\widehat{\sigma_{22}}$) component in Pa units. The coarsening for stress is described by equation (6) and for damage by equation (7). (a) Initial coarsened state; (b) Coarsened state at some later time in the simulation.

while for damage we employ the following,

$$\widehat{\mathrm{d}}_t(\vec{x}) = \sum_{c=1}^{C} d_{c,t}\mathbb{I}_{R_c}(\vec{x}), \tag{7}$$

where $d_{c,t} = 1$ if the cell R_c is damaged (i.e., if any damage has accumulated in the cell) at time t, otherwise $d_{c,t}$ is 0. Thus, we model damage as a binary field with values 0 (not damaged) and 1 (damaged). A coarsened state is shown in Figs. 6a and 6b.

Our transitional model is the same transitional model shown in Fig. 3. For each stress component $\widehat{\sigma_{ij}}$, the transitional model is given by

$$\widehat{\sigma_{ij,t}} = f_{ij}(\widehat{\mathbf{u}_{\mathbf{ij},\mathbf{t}}}), \tag{8}$$

where $\widehat{\mathbf{u}_{\mathbf{ij},\mathbf{t}}} = \{\widehat{\sigma_{ij,t-1}}\}$ and the functions f_{ij} are to be learned. The transition model for damage is given by

$$\widehat{\mathrm{d}}_t = f_d(\widehat{\mathbf{u}_{\mathbf{d},\mathbf{t}}}), \tag{9}$$

where $\widehat{\mathbf{u}_{\mathbf{d},\mathbf{t}}} = \{\widehat{d_{t-1}}, \widehat{\sigma_{11,t}}, \widehat{\sigma_{12,t}}, \widehat{\sigma_{22,t}}\}$ and the function f_d is also to be learned. Thus, to predict stress at time t, we use stress from time $t-1$, whereas to predict damage at time t, we use stress at time t and damage from time $t-1$. Hence, for a given time t, stress is predicted first and then used to predict damage. This transition model is guided from knowledge of the physical system describing damage evolution as given by HOSS.

The quantities of interests are univariate time series that are functions of the state. Typically we are interested in the maximum stress at time t given by $\max\sigma_{ij,t}$, the normalized damage at time t denoted by $\mathbb{D}_t$ and given by

$$\mathbb{D}_t = \frac{1}{C}\sum_{c=1}^{C} d_c; \tag{10}$$

the number of distinct fractures at time t and the length of each fracture.

In this paper, we will only focus on the two quantities of interests $\max\sigma_{ij,t}$ and $\mathbb{D}_t$. However, the latter two are important and will be considered in future work. Here, we only describe one way to define them that can be used in any model that predicts fracture propagation at the mesoscale level. We need a model to describe fractures from the stress and damage. For the problem considered here, we construct a graph $\mathcal{G}_F(V, E_t)$ that creates a dynamic network structure on our binary damage field $\widehat{\mathrm{d}}_t$. The vertex set V of the graph $\mathcal{G}_F(V, E_t)$ is the set of all partition cells R_c for $1 \le c \le C$, whereas the edges E_t evolve dynamically as follows: if, v_k, v_l are two vertices in V such that v_k and v_l are neighbors (in the sense of the mesh), then the edge e_{kl} between v_k and v_l is in the edge set E_t if at time t, the cells represented by v_k and v_l are damaged (i.e. $\widehat{\mathrm{d}}_t = 1$ at these cells). The number of distinct fractures at time t is then modeled by the number of distinct connected components of the fracture graph $\mathcal{G}_F(V, E_t)$, whereas the length of each fracture is given by the area (or volume) of the cells in that connected component. This process is illustrated in Figs. 7a and 7b.

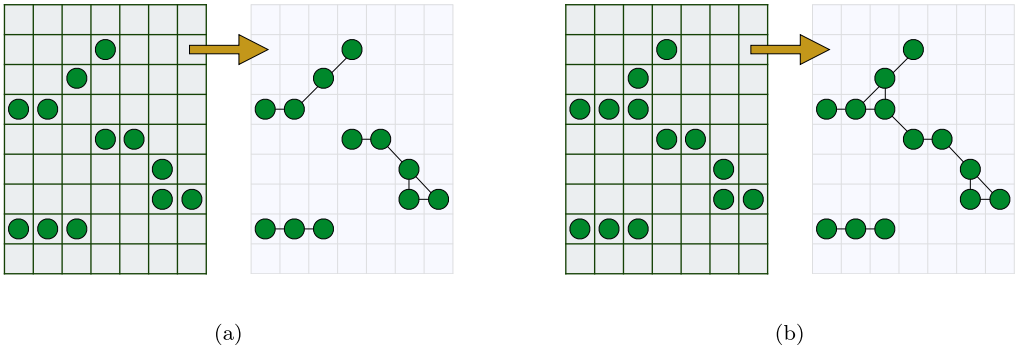

Fig. 7. Illustration of identifying cracks with the connected components of a graph. The green color denotes cells that have been damaged while the arrow indicates the process of obtaining individual cracks from this field: (a) Fracture graph corresponding to a damage field. Here we have 3 distinct fractures; (b) Fracture graph when a new cell is damaged. Here we have 2 distinct fractures.

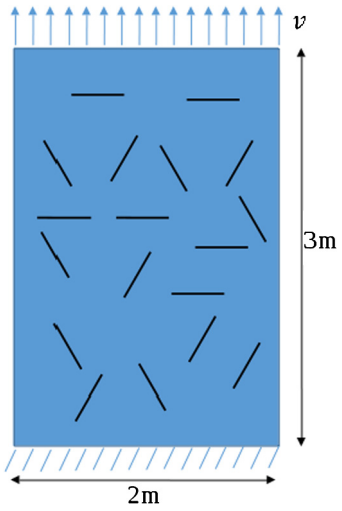

Fig. 8. Initial configuration of a HOSS simulation of a 2 meter by 3 meter concrete sample loaded in uniaxial tension. Black line segments represent cracks.

4. Results

Here, we illustrate the framework on a mesoscale damage evolution problem that helps justify the full framework while illustrating the main theme. Though this problem is an illustration, it presents significant challenges for emulation. The data are sparse (since damage occurs at about 10% of the region) and high dimensional (since stresses are functions approximated by high dimensional subspace). The spatiotemporal aspect of the problem is highly non-linear, and we have interactions between stresses and cracks, the stresses can range from $-1e^7$ to $1e^7$ Pa while the damage is treated as binary.

The mesoscale HOSS simulations considered here represents a system composed of a 2 meter $\times$ 3 meter concrete sample loaded in uniaxial tension. To impose this loading condition, the bottom boundary of the sample is kept fixed while the top boundary is moved with a constant speed of $v = 0.3$ m/s, which results in a strain rate of 0.1 s^{-1}. The material is assumed to be elastically isotropic for all cases. The following elastic material properties were used: density of 2500 kg/m^3, Young's modulus of 22.6 GPa, and shear modulus of 9.1 GPa.

Each specimen contains cracks or defects. In this example, we have 20 initial cracks that are each 30 cm long. These cracks are randomly arranged in space (satisfying a weak uniform distribution), while the initial orientation is chosen uniform randomly from one of the three angles: 0, 60, and 120 degrees. An initial configuration is shown in Fig. 8. The system described above is simulated by HOSS using approximately 158, 000 triangular elements per specimen. Each vertex of a triangle can move independently. Fractures are characterized by a damage value associated with each edge, ranging from 0 (completely intact) to 1 (completely broken). When the damage exceeds a fixed threshold (0.1 in this case), that edge is considered to be fractured. The simulations end when the material fails or after 7 ms, whichever comes first.

For this problem, we do not use $\sigma_{12,t}$ as part of the state to be learned. In keeping with the central theme of this paper, our main motivation in doing so was to guide the development of the ML methodology using the underlying physics. Since the mechanism of failure is Mode I, where shear forces play an insignificant role compared to tensile and compressive forces, we neglected the $\sigma_{12,t}$ terms while considering input features. This observation was confirmed by a preliminary correlation

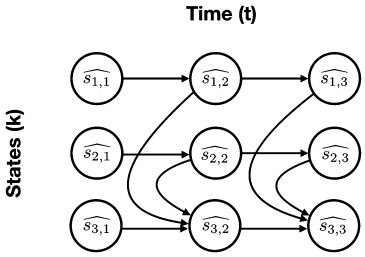

Fig. 9. Graphical representation of the transition model for the coarsened state space of this problem, with $K = 3$ states and $N = 3$ discrete time points. The transition model is depicted by directed arrows i.e. transition models can be described as a directed acyclic graph (DAG) where $\widehat{s_{k,t}}$ only depends on states $\{\widehat{s_{k',t'}} | t' \leq t \text{ and } 1 \leq k' \leq K\}$. Here $\widehat{s_{1,t}} = \widehat{\sigma_{11,t}}$, $\widehat{s_{2,t}} = \widehat{\sigma_{22,t}}$ and $\widehat{s_{3,t}} = \widehat{\mathrm{d}_t}$.

study which found that damage at time t was highly correlated with only $\sigma_{11,t}$ and $\sigma_{22,t}$. This is, in fact, the power of combining physics insight with the ML algorithms, so that we move away from the typical black-box approach that ignores any of our physics understanding. However, in general, feature engineering is part of the process of designing an emulator. Thus, we have a set of $K = 3$ states at some time t given by $\mathbf{s_t} = \{\sigma_{11,t}, \sigma_{22,t}, \mathrm{d}_t\}$. Each state is identified with a vector in a ∼158,000-dimensional space (i.e., the dimensionality of the vector scales with the number of elements on the mesh). In our separate work [35] we devised a spatio-temporal Neural Network and let the Network figure out the dependencies and do automatic feature selection based on a physics informed loss function.

For the mesoscale coarsening, we divide the domain $\Omega = [0, 2] \times [0, 3]$ into $C = 3750$ rectangles that corresponds to a partition of the horizontal dimension into 50 intervals and the vertical dimension into 75 intervals. Our coarsened set of states at time t is $\widehat{\mathbf{s_t}} = \{\widehat{\sigma_{11,t}}, \widehat{\sigma_{22,t}}, \widehat{\mathrm{d}_t}\}$, where each state is identified with a vector in $C = \mathbb{R}^{3750}$. The transition model is same as in Fig. 3 with *row* 3 removed. This is shown in Fig. 9. A coarsened state is shown in Fig. 6a and 6b for times $t = 0$ and $t = 4$ μs respectively. We take 225 timesteps to learn the transition function from $t - 1$ to t, where each timestep corresponds to an increment of 0.01 μs. This time step is a standard output for low strain damage simulation considered here.

We split the process of learning the transition into two steps. The first step is to learn a lower-dimensional representation of the state, and this lower-dimensional representation is the inputs to the transition function. The second step is to learn how the state evolves to the next step from the lower-dimensional representation. For learning the transition function for stresses, we use kernel PCA to accomplish the dimension reduction task. We then use regularized regression to learn how the state at the next time evolves. Thus a multi-output, kernel-PCA regression [26][36], gives us our learned function $\widehat{f_{ij}}$ to yield a prediction for the coarsened state at each time t,

$$\widehat{\sigma_{ij,t}} = \widehat{f_{ij}}\left(\widehat{\sigma_{ij,t-1}}\right).$$

Note that a kernel-PCA regression is part of a technique that falls under unsupervised learning. The coarsened stress $\widehat{\sigma_{ij,t}}$ is an element of $\mathbb{R}^C$ where $C = 3750$. To predict one cell (component) of this vector at time t, we use the entire field at time $t - 1$. By employing a higher-order kernel, the machine learning detects a smaller subspace in which our state is embedded. The parameters of this kernel (number of PCA components and degree of the polynomial in the kernel) are obtained by doing a grid search over an interval for both the PCA components and polynomial degree, and this is cross-validated.

Similarly to predict damage at time t, $\widehat{\mathrm{d}_t}$ we learn the function $\widehat{f_d}$,

$$\widehat{\mathrm{d}_t} = \widehat{f_d}\left(\widehat{\sigma_{ij,t}}, \widehat{\mathrm{d}_{t-1}}\right).$$

Since, $\widehat{\mathrm{d}_t}$ is a binary field, damage prediction is akin to a binary classification problem. In this particular case of low strain damage propagation, we impose our knowledge of the mesoscale physics to determine $\widehat{f_d}$ in the following way:

1. A cell that was damaged at time $t - 1$, will remain damaged at time t. Thus, at a cell c at time t, $\widehat{f_d} = 1$ if the corresponding $\widehat{\mathrm{d}_{t-1}}(\mathbf{x}) = 1$.
2. If a cell, c, and all of its nearest neighbors are undamaged at time $t - 1$ then c remains undamaged at time t. This is effectively a constraint that no new fractures are nucleated and is reasonable for the low strain rate problem we consider.
3. If a cell, c, is undamaged at time $t - 1$ but at least one of its neighbors is damaged at time $t - 1$, then $\widehat{f_d}$ is a function of the stresses at t, $\widehat{\sigma_{ij,t}}$.

Note that the function described above is highly non-linear and by using information from the physics of damage propagation, we break this non-linear function into manageable pieces. The separation of the function into different pieces follows a physics-based approach, whereas the function learned in the third condition is purely data-driven. For the nearest neighbors $\mathcal{NN}_c$, we take a step of size 1 around each column and row of the cell c. This corresponds to 8 nearest neighbors of the cell c.

Finally, to showcase this framework we ran $N_{\text{sims}} = 61$ simulations. Each simulation is the time evolution of an initial state by HOSS and coarsened as described in the paragraphs above. The difference between these simulations is the initial state resulting from the random placement and orientation of the cracks.

The choice of 50×75 divisions was taken to resolve the crack lengths in the discrete structure. At this resolution, an individual crack, which was 30 cm long, was sufficiently well resolved in our coarsened state (occupied $6-7$ cells). The aspect ratio taken was to maintain the aspect ratio of the experiment where the height of the slab (y dimension) was 1.5 times the width of the slab (x dimension). To show the dependency on partition size, we have run the simulations for sizes (10×15, 20×30, 40×60) along with the original (50×75) and presented the average width of our prediction interval of our emulated QoIs in Table 4. Larger width indicates larger uncertainty in our prediction. We notice that the errors are large in smaller partitions, and the error width for partition (40×60) is extremely close to that of our original partition ($\pm 1\%$). We want to note that taking even larger partitions corresponding to smaller length scales may also lead to poor performance since we are increasing the dimension of the feature space while also making damage prediction a highly imbalanced classification task since only 10% of our cells get damaged, and we obtain a larger number of undamaged cells.

4.1. Prediction

A prediction of the state is given by iteratively applying the learned functions $\widehat{f_{ij}}$ and $\widehat{f_d}$. We start with an initial state $\sigma_{ij,0}, \mathrm{d}_0$. The algorithm then iteratively produces a time evolution of the coarsened state from an initial state at time n_0 to a final state at time n_f described below:

Data: $\sigma_{ij,0}, \mathrm{d}_0$
Result: predicted $\widehat{\sigma_{ij,t}}, \widehat{\mathrm{d}_t}$, for $n_0 \le t \le n_f$.
coarsening and initialization;
$\widehat{\sigma_{ij,0}} \leftarrow \sigma_{ij,0}$;
$\widehat{\mathrm{d}_0} \leftarrow \mathrm{d}_0$;
for *t from* n_0 *to* n_f **do**
  $\widehat{\sigma_{ij,t}^{ML}} = \widehat{f_{ij}}\left(\widehat{\sigma_{ij,t-1}^{ML}}\right)$;
  $\widehat{\mathrm{d}_t^{ML}} = \widehat{f_d}\left(\widehat{\sigma_{ij,t}^{ML}}, \widehat{\mathrm{d}_{t-1}^{ML}}\right)$
end

Algorithm 1: Emulate state through machine learning. Once the transition model is learnt it is iteratively applied starting from some initial conditions.

Our quantities of interest are $y_{ij,t} = \max \sigma_{ij,t}$ and $y_{d,t} = \mathbb{D}_t$, while the corresponding emulated quantities are $\widehat{y_{ij,t}} = \max \widehat{\sigma_{ij,t}^{ML}}$ and $\widehat{y_{d,t}} = \widehat{\mathbb{D}_t^{ML}}$. The discrepancy errors $e_{ij,t} = y_{ij,t} - \widehat{y_{ij,t}}$ and $e_{d,t} = y_{d,t} - \widehat{y_{d,t}}$ are modeled as Gaussian processes with 0 prior mean and a parametric kernel (GPs (see subsection 2.4) whose parameters are learned from data. To get predictions for the QoI, we sample error from the GP and correct the mean by adding the corresponding emulated quantities. Thus, if y is the true QoI, the predicted QoI $\widehat{Y}$ is a random variable. The next subsection describes how we obtain $\widehat{f_{ij}}$ and $\widehat{f_d}$ from data and fit a Gaussian process to the errors $e_{ij,t}$ and $e_{d,t}$.

We split the simulations $N_{\text{sims}} = 61$ randomly into 2 groups N_{train} and N_{test} of sizes 50 and 11 respectively corresponding roughly to a $80\% - 20\%$ split for training and testing. We further split the training dataset N_{train} into 2 groups N_T, N_{GP} of size 30 and 20 respectively to learn the transition function and the discrepancy error for the QoIs. This split corresponds to a $60\% - 40\%$ split. The goal is to have enough samples to represent the underlying distribution. Our feature space was of size ($50 \times 75 = 3750$) and taking 50 simulations we were able to get around $11,250 (50 \times 225)$ pairs of times steps $(t-1, t)$ which is about 3 times our feature space (sufficient to find a non-linear lower dimensional manifold using K-PCA).

4.2. Training

A leave one (simulation) out cross-validation procedure is performed on N_T to fit the parameters of the machine-learned transition functions $\widehat{f_d}$ and $\widehat{f_{ij}}$. To elaborate, we take 1 full simulation out of N_T and fit our ML model on the remaining simulation data and compute the cross-validation error on the left out simulation. This error metric is used to determine the ML parameters, which include the choice of kernel, the number of kernel PCA components, and the regularization penalty parameter for regression.

To learn the discrepancy between the emulated and the true quantity of interest we fit the Gaussian processes described in the subsection 2.4 on the N_{GP} data. That is, on this data set, $\widehat{f_d}$ and $\widehat{f_{ij}}$ gives a prediction times series starting from the initial conditions as described in Algorithm 1 and the discrepancy between the true state is used as data to fit the Gaussian process. Our model for stress error $e_{ij} \sim GP(\mu_{\text{prior}}, K_{ij})$ where K_{ij} is a radial basis function (RBF) kernel and $\mu_{\text{prior}} = 0$. For normalized damage error, our model is $e_d \sim GP(\mu_{\text{prior}}, K_d)$, where K_d is a combination of an RBF kernel and white noise and $\mu_{\text{prior}} = 0$. Since normalized damage is monotonic, we fit isotonic regressions to the output of the Gaussian process to ensure monotonicity in the predictive distribution.

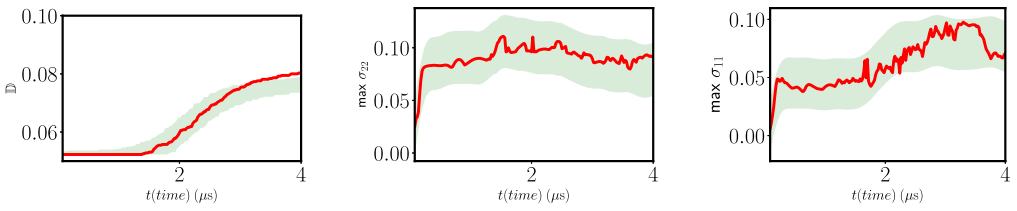

Fig. 10. 95% prediction intervals are shown for (a) Normalized damage, (b) $\max \sigma_{22,t}$, and (c) $\max \sigma_{11,t}$ for one simulation. Red is the true value.

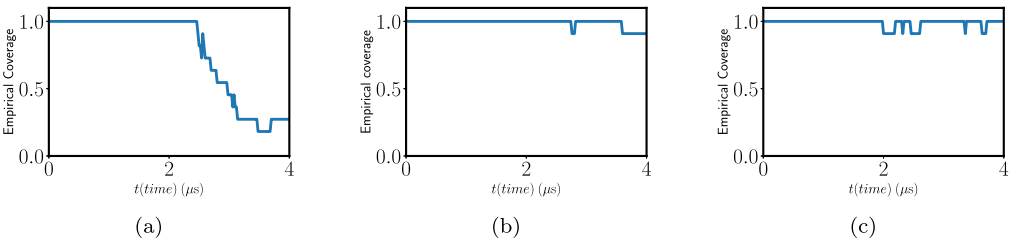

Fig. 11. 95% Empirical coverage for (a) Normalized damage; (b) $\max \sigma_{22,t}$; (c) $\max \sigma_{11,t}$.

4.3. Testing

The 11 simulations in N_{test} are used for testing our framework. Our time interval of interest from $n_0 = 4$ to $n_f = 225$ corresponds to the time interval $[0.08, 4]$ µs, and we denote the length of this interval by $T = n_f - n_0$. Let y_t denote the true quantity of interest and let $\widehat{Y}_t$ denote the predicted quantity of interest. (Note that $\widehat{Y}_t$ is a random variable). We use four validation metrics to assess our framework:

1. Empirical coverage: We estimate a 95% prediction interval (PI) for the quantity of interest by sampling from the Gaussian process that has been fit during our training. Let y_t be a quantity of interest at time t, and let $y_{j,t}$ denote the quantity of interest from the jth simulation in N_{test} and let $(Y_L^{(t)}, Y_U^{(t)})_j$ denote the 95% PI around $y_{j,t}$. The empirical coverage is the (estimated) probability that our emulation captures the true quantity of interest, and is given by:

$$\mathfrak{p}(t) = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} \mathbb{I}_{y_{j,t} \in (Y_L^{(t)}, Y_U^{(t)})_j}, \tag{11}$$

   where $\mathbb{I}_{y_{j,t} \in (Y_L^{(t)}, Y_U^{(t)})_j}$ is 1 if $y_{j,t} \in (Y_L^{(t)}, Y_U^{(t)})_j$ (i.e. our prediction interval captures the true quantity of interest) and 0 otherwise.
2. Relative l_∞, l_1 and l_2 errors. These are defined below:

$$e_k = \frac{\left\| y - \mathbb{E}(\hat{Y}) \right\|_k}{\left\| y \right\|_k}, \tag{12}$$

   where $\| \cdot \|_k$ denotes the standard norms for $k = 1, 2, \infty$. For example,

$$\| y \|_2 = \frac{1}{T} \sum_{t=t_0}^{t_f} (y_t)^2 .$$

The empirical coverage provides one measure of the confidence of our framework to predict out of sample simulations. The Figs. 11a, 11b and 11c show the empirical coverage of damage, max stress σ_{22} and max stress σ_{11} respectively for one out of sample prediction. Note that we do well for the stress prediction as our empirical coverage is 95%. Note that we also do well predicting damage for the first 2.5 µs as seen in Fig. 10a with an empirical coverage of 80% in that time interval. We have a tighter band for damage. However, we slightly miss damage prediction after 3 µs. Even a slight miss counts as non-coverage, and this is confirmed in the Fig. 11a where our confidence in predicting damage drops drastically from 60% at 2.5 µs to 40% at 4 µs. We provide relative errors between the expected prediction and the truth in the Tables 1, 2 and 3. Most of our relative errors are within $5 - 15\%$.

Table 1
Relative errors for normalize damage prediction.

N_{test} id	$e_2(\%)$	$e_1(\%)$	$e_\infty(\%)$
0	5.20	4.04	8.03
1	8.27	6.41	11.45
2	7.77	5.74	10.70
3	3.29	2.40	5.26
4	6.24	4.55	8.84
5	3.14	2.42	5.47
6	5.24	4.22	7.37
7	2.75	2.39	4.51
8	2.19	1.79	4.50
9	11.91	8.69	16.85
10	1.21	1.02	2.39

Table 2
Relative errors for $\max\sigma_{22}$ prediction.

N_{test} id	$e_2(\%)$	$e_1(\%)$	$e_\infty(\%)$
0	4.47	3.56	9.86
1	8.94	7.72	15.89
2	8.10	6.70	17.11
3	7.51	5.90	17.36
4	10.38	6.95	31.89
5	8.40	6.95	16.41
6	16.81	11.37	36.97
7	5.79	5.03	12.69
8	7.48	6.19	16.46
9	6.61	5.45	13.45
10	9.01	7.42	18.84

Table 3
Relative errors for $\max\sigma_{11}$ prediction.

N_{test} id	$e_2(\%)$	$e_1(\%)$	$e_\infty(\%)$
0	9.16	6.78	25.20
1	9.96	7.62	19.90
2	13.25	10.43	26.08
3	12.05	9.70	22.48
4	9.33	7.37	18.79
5	14.53	11.13	26.54
6	7.89	6.36	18.43
7	9.70	7.52	18.85
8	10.59	7.76	22.42
9	13.99	10.58	34.03
10	9.74	7.85	20.99

Table 4
Average prediction width corresponding to the 95% confidence interval for the quantities of interest: $2 \times$ [1.96 std. error] with respect to partition sizes. A larger width corresponds to more uncertainty.

Partition size	*dmg*	σ_{22}	σ_{11}
10×15	0.098	0.033	0.027
20×30	0.017	0.0031	0.0080
40×60	0.012	0.0071	0.0058
50×75	0.012	0.0064	0.0060

While our predictions yield sub 15% relative errors, certain deficiencies remain. The empirical coverage provides a good indication of this. Looking at the empirical coverage for damage, as shown in Fig. 10a, we notice that our method does well in predicting the onset of damage and the growth of damage, as evidenced by an 80% empirical coverage in the time interval $(0, 2.5\ \mu s)$. However, the propagation of damage when nearly 7% of the material has been damaged is not well captured. There are a couple of reasons for it. First, after 2.5 μs, there is much variability in the simulations as some simulations fail around 3.5 μs while some last till 7 μs. Thus, there is a heteroscedasticity (time-dependent variability) that is not captured by a simple Gaussian process model. Second, our physics informed damage prediction is based on a nearest-neighbor partition and logistic regression of stress values. This hints at the idea that a simple nearest-neighbor partition is not sufficient when more than 7% of the material is damaged. Also, while stress values are a strong indicator of damage, any error in learning stresses influences the damage prediction. Thus a better stress emulator could improve the damage

predictions. We have selected a Gaussian QoI model with a constant variance that does not change in time. This is done for two reasons – 1) designing heteroscedastic Gaussian Processes is challenging, and 2) our ML algorithm is designed to get the trends right. Our current model does not do well predicting damage at later times and with a constant variance QoI model, some test cases are not captured in the prediction interval giving us higher uncertainty. We have shown in a recent work ([35]) that by using a complex transition model (Spatio-temporal Neural Network) we can get better predictions.

5. Conclusion

We have presented a framework to emulate meso-scale physics in the context of parameter estimation for macro-scale models. The framework can be viewed as a probabilistic reduced-order model. In our example concerning fracture propagation, the model was trained on simulated data, but in other scenarios, it could be trained with data from other sources such as experimental data. While our transition models used relatively simple techniques from the machine learning literature, the application we have illustrated is fairly complex and demonstrates all the aspects of our framework. Notably, it involves the simulation of multiple spatially variable fields (two for stress and one for damage) that involve coupled physics – damage impacts stress and stress impacts damage.

Our results highlight that, when emulating fine scale physics with microstructures, all aspects of statistical modeling (data representation through coarsening, transition model, stochastic model) need to be addressed. By choosing the appropriate metrics, we have been able to validate our models and learn where they can be improved. The main area where there could be improvements is in predicting damage at late times. The typical relative l_2 errors are around 10%, and the empirical coverage is high for the stress at all times and high for the damage except at late times. This work could be extended by exploring more complex transition models to improve the empirical coverage of damage at late times.

The overarching goal of this work is to enable scale bridging by using machine learning emulators that accurately inform a macro-scale model. The computational speed and efficiency of the machine learning emulators are critical to the framework because traditional models are often too costly to feasibly inform the macro-scale model. Ongoing work shows promise in effectively utilizing this framework to inform macro-scale models. We have used [37] for all our computations.

CRediT authorship contribution statement

Nishant Panda: Conceptualization, Methodology, Software, Writing - original draft. **Dave Osthus:** Conceptualization, Methodology, Writing - review & editing. **Gowri Srinivasan:** Project administration, Supervision, Writing - review & editing. **Daniel O'Malley:** Writing - review & editing. **Viet Chau:** Data curation. **Diane Oyen:** Software, Validation. **Humberto Godinez:** Supervision.

Declaration of competing interest

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

Acknowledgements

This work was supported by the Laboratory Directed Research and Development (LDRD) program at Los Alamos National Laboratory. All the authors thank Grant No. 20170103DR for support. Nishant Panda also acknowledges support from the Los Alamos National Laboratory – Advanced Simulation and Computing (ASC) program.

References

[1] M.F. Ashby, D. Cebon, Materials selection in mechanical design, J. Phys. IV 3 (C7) (1993) C7.
[2] R.L. Kranz, Microcracks in rocks: a review, Tectonophysics 100 (1–3) (1983) 449–480.
[3] M.H. Anders, S.E. Laubach, C.H. Scholz, Microfractures: a review, J. Struct. Geol. 69 (2014) 377–394.
[4] S.P. Shah, S.E. Swartz, C. Ouyang, Fracture Mechanics of Concrete: Applications of Fracture Mechanics to Concrete, Rock and Other Quasi-Brittle Materials, John Wiley & Sons, 1995.
[5] T.L. Anderson, T.L. Anderson, Fracture Mechanics: Fundamentals and Applications, CRC Press, 2005.
[6] A. Hunter, B.A. Moore, M. Mudunuru, V. Chau, R. Tchoua, C. Nyshadham, S. Karra, D. O'Malley, E. Rougier, H. Viswanathan, et al., Reduced-order modeling through machine learning and graph-theoretic approaches for brittle fracture applications, Comput. Mater. Sci. 157 (2019) 87–98.
[7] P. Larrañaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J.A. Lozano, R. Armañanzas, G. Santafé, A. Pérez, V. Robles, Machine learning in bioinformatics, Brief. Bioinform. 7 (1) (2006) 86–112, https://doi.org/10.1093/bib/bbk007, arXiv:1412.3919v1.
[8] K. Rajan, Materials informatics, Mater. Today 8 (10) (2005) 38–45, https://doi.org/10.1016/S1369-7021(05)71123-8.
[9] K. Takahashi, Y. Tanaka, Materials informatics: a journey towards material design and synthesis, Dalton Trans. 45 (26) (2016) 10497–10499, https://doi.org/10.1039/C6DT01501H.
[10] M. Raissi, G.E. Karniadakis, Hidden physics models: machine learning of nonlinear partial differential equations, J. Comput. Phys. 357 (2018) 125–141.
[11] S. Wiewel, M. Becher, N. Thuerey, Latent-space physics: towards learning the temporal evolution of fluid flow, preprint, arXiv:1802.10123.
[12] M. Raissi, P. Perdikaris, G.E. Karniadakis, Machine learning of linear differential equations using Gaussian processes, J. Comput. Phys. 348 (2017) 683–693.
[13] N.M. Mangan, J.N. Kutz, S.L. Brunton, J.L. Proctor, Model selection for dynamical systems via sparse regression and information criteria, Proc. R. Soc. A, Math. Phys. Eng. Sci. 473 (2204) (2017) 20170009.

[14] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part II): data-driven discovery of nonlinear partial differential equations, preprint, arXiv:1711.10566.
[15] J.-X. Wang, J.-L. Wu, H. Xiao, Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data, Phys. Rev. Fluids 2 (3) (2017) 034603.
[16] L.J. Nelson, G.L. Hart, F. Zhou, V. Ozoliņš, et al., Compressive sensing as a paradigm for building physics models, Phys. Rev. B 87 (3) (2013) 035125, https://doi.org/10.1103/PhysRevB.87.035125.
[17] B. Peherstorfer, K. Willcox, M. Gunzburger, Survey of multifidelity methods in uncertainty propagation, inference, and optimization, SIAM Rev. 60 (3) (2018) 550–591, https://doi.org/10.1137/16M1082469.
[18] D.J. Lucia, P.S. Beran, W.A. Silva, Reduced-order modeling: new approaches for computational physics, Prog. Aerosp. Sci. 40 (1–2) (2004) 51–117.
[19] I. Kevrekidis, C. Rowley, M. Williams, A kernel-based method for data-driven Koopman spectral analysis, J. Comput. Dyn. 2 (2) (2015) 247–265.
[20] J. Roychowdhury, Reduced-order modeling of time-varying systems, IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process. 46 (10) (1999) 1273–1288.
[21] P. Benner, S. Gugercin, K. Willcox, A survey of projection-based model reduction methods for parametric dynamical systems, SIAM Rev. 57 (4) (2015) 483–531, https://doi.org/10.1137/130932715.
[22] A. Gelman, H.S. Stern, J.B. Carlin, D.B. Dunson, A. Vehtari, D.B. Rubin, Bayesian Data Analysis, Chapman and Hall/CRC, 2013.
[23] D. Koller, N. Friedman, F. Bach, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[24] M. Kamb, E. Kaiser, S.L. Brunton, J.N. Kutz, Time-delay observables for Koopman: theory and applications, preprint, arXiv:1810.01479.
[25] Y. Long, X. She, S. Mukhopadhyay, Hybridnet: integrating model-based and data-driven learning to predict evolution of dynamical systems, preprint, arXiv:1806.07439.
[26] J. Friedman, T. Hastie, R. Tibshirani, The Elements of Statistical Learning, vol. 1, Springer Series in Statistics, Springer-Verlag, New York, 2001.
[27] C.E. Rasmussen, Gaussian processes in machine learning, in: Summer School on Machine Learning, Springer, 2003, pp. 63–71.
[28] A.G. Wilson, Z. Hu, R. Salakhutdinov, E.P. Xing, Deep kernel learning, in: Artificial Intelligence and Statistics, 2016, pp. 370–378.
[29] E.E. Knight, E. Rougier, Z. Lei, Hybrid optimization software suite (HOSS) - educational version, Tech. rep., Los Alamos National Laboratory, 2015, lA-UR-15-27013.
[30] E. Rougier, E.E. Knight, A. Munjiza, LANL-CSM: HOSS - MUNROU technology overview, presentation, LA-UR-13-23422, 2013.
[31] E.E. Knight, E. Rougier, A. Munjiza, LANL-CSM: consortium proposal for the advancement of HOSS, presentation, LA-UR-13-23409, 2013.
[32] A. Munjiza, The Combined Finite-Discrete Element Method, John Wiley and Sons LTD, New York, 2004.
[33] A. Munjiza, E. Rougier, E. Knight, Large Strain Finite Element Method: A Practical Course, John Wiley and Sons, New York, 2015.
[34] A. Munjiza, E. Knight, E. Rougier, Computational Mechanics of Discontinua, John Wiley and Sons LTD, Chichester, 2011.
[35] A. Mehta, C. Scott, D. Oyen, N. Panda, G. Srinivasan, Physics-informed spatiotemporal deep learning for emulating coupled dynamical systems.
[36] R. Rosipal, L.J. Trejo, Kernel partial least squares regression in reproducing kernel Hilbert space, J. Mach. Learn. Res. 2 (Dec 2001) 97–123.
[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.