Journal of Computational Physics 432 (2021) 110147

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

A comparative study of machine learning models for predicting the state of reactive mixing

B. Ahmmed [a,b,*,1], M.K. Mudunuru [c,2], S. Karra [a,3], S.C. James [b,d,4], V.V. Vesselinov [a,5]

[a] Computational Earth Science Group, Earth and Environmental Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545, United States of America
[b] Department of Geosciences, Baylor University, Waco, TX 76706, United States of America
[c] Watershed & Ecosystem Science, Pacific Northwest National Laboratory, Richland, WA 99352, United States of America
[d] Departments of Geosciences and Mechanical Engineering, Baylor University, Waco, TX 76706, United States of America

ARTICLE INFO

Article history:
Available online 19 January 2021

Dataset link:
https://github.com/bulbulahmmed/ML-to-reactive-mixing-data

Keywords:
Surrogate modeling
Machine learning
Reaction-diffusion equations
Random forests
Ensemble methods
Artificial neural networks

ABSTRACT

Mixing phenomena are important mechanisms controlling flow, species transport, and reaction processes in fluids and porous media. Accurate predictions of reactive mixing are critical for many Earth and environmental science problems such as contaminant fate and remediation, macroalgae growth, and plankton biomass growth. To investigate the evolution of mixing dynamics under different scenarios (e.g., anisotropy, fluctuating velocity fields), a finite-element-based numerical model was built to solve the fast, irreversible bimolecular reaction-diffusion equations to simulate a range of reactive-mixing scenarios. A total of 2,315 simulations were performed using different sets of model input parameters comprising various spatial scales of vortex structures in the velocity field, time-scales associated with velocity oscillations, the perturbation parameter for the vortex-based velocity, anisotropic dispersion contrast (i.e., ratio of longitudinal-to-transverse dispersion), and molecular diffusion. The outputs comprised concentration profiles of reactants and products. The inputs and outputs from these simulations were concatenated into feature and label matrices, respectively, to train 20 different machine learning (ML) models intended to emulate system behavior. These 20 ML emulators, based on linear methods, Bayesian methods, ensemble learning methods, and multilayer perceptrons (MLPs), were trained to classify the state of mixing and predict three quantities of interest (QoIs) characterizing species production, decay (i.e., average concentration, square of average concentration), and degree of mixing (i.e., variances of species concentration). Unsurprisingly, linear classifiers and regressors failed to reproduce the QoIs; however, ensemble methods (classifiers and regressors) and the MLP model accurately classified the state of reactive mixing and the QoIs. Among ensemble methods, random forest and decision-tree-based AdaBoost faithfully predicted the QoIs. At run time, trained ML emulators produced results $\approx 10^5$ times faster than the finite-element simulations. Due to their low computational expense and high accuracy, ensemble and MLP models are

* Corresponding author at: Department of Geosciences, Baylor University, Waco, TX 76706, United States of America.
E-mail address: bulbul_ahmmed@baylor.edu (B. Ahmmed).

[1] Developed the framework, ran the ML models, and drafted the original manuscript.
[2] Formulated the idea, generated data, supervised, and helped draft the manuscript.
[3] Wrote the FEM code, generated the data, and critically revised the manuscript.
[4] Participated in drafting and critically revising the manuscript.
[5] Critically revised the manuscript.

https://doi.org/10.1016/j.jcp.2021.110147

excellent emulators for these numerical simulations and great utilities in uncertainty quantification exercises, which can require 1,000s of forward model runs.

1. Introduction

Reactive-mixing phenomena dictate the distribution of chemical species in fluids (e.g., coastal waters) and subsurface porous media. Accurate quantification of species concentration is critical to applications such as contaminant remediation, plume characterization, algal-bloom forecasting, etc. [1–10]. Parameters that influence reactive mixing in fluids and subsurface porous media include the structure of the flow field (e.g., chaotic advection), fluid injection/extraction (i.e., location of wells, injection/extraction rates), subsurface heterogeneity, dispersion, and anisotropy [11,12]. These parameters have variable impacts on the important quantities of interest (QoIs) such as species production and decay (e.g., average and squared average species concentrations) and degree of mixing (i.e., variances of species concentrations). For the QoIs, nonlinear partial differential equations are solved using high-resolution numerical methods (e.g., finite-difference, finite-element, or finite-volume methods) that can take hours to days (for $\approx \mathcal{O}(10^6) - \mathcal{O}(10^9)$ degrees-of-freedom) on state-of-the-art, high-performance computing machines. Such computational times preclude real-time predictions, which can be critical to decision making for remediation activities. Hence, alternative faster approaches are needed and machine learning (ML)-based emulators show promise [13–17]. Here, we built and compared 20 ML emulators to predict the reactive-mixing QoIs. The ML emulators were trained and tested using data from finite-element numerical simulations, which expressly consider the underlying reaction-diffusion physics in anisotropic porous media.

Given sufficient data, ML models can successfully detect, quantify, and predict different types of phenomena in the geosciences [18,19]. Applications include remote sensing [17,20], ocean wave forecasting [21–23], seismology [13,14,24–27], hydrogeology [15,16,28], gene expression [29,30], and geochemistry [31–35]. ML emulators (also known as surrogate models or reduced-order models) can be fast, reliable, and robust when trained on large datasets [18,19,36]. The ML emulators are constructed using training data (e.g., features and labels), which include inputs and outputs either from field data, experimental data, high-resolution numerical simulations, or any combination of these [36,37]. In this paper, we compared emulators based on generalized linear methods [38,39], Bayesian methods [40,41], ensemble methods [42,43], and a multilayer perceptron (MLP) [44,45] to predict the various reactive-mixing QoIs.

Previous researchers used unsupervised and supervised ML methods to reproduce reactive-mixing QoIs. Vesselinov et al. [11] used non-negative tensor factorization with custom k-means clustering (unsupervised ML) to identify hidden features in the solutions to reaction-diffusion equations. They determined that anisotropic features (i.e., longitudinal and transverse dispersion) govern reactive mixing at early to middle times while molecular diffusion controls product formation at late times. They also quantified the effects of longitudinal and transverse dispersion and molecular diffusion on species production and decay over time. Mudunuru and Karra [12] ranked the importance of input parameters/features on reactive-mixing QoIs using support vector machine (SVM) and support vector regressor (SVR) emulators. Furthermore, they used SVM and SVR models to classify the degree of mixing and predict QoIs. However, SVM and SVR training times increase significantly with the size of the training dataset [12]. To obviate this problem, in this paper, we built ML emulators whose training times were $\approx 10^5$ times faster than SVM and SVR models without compromising accuracy.

Specifically, we compared one linear classifier, two Bayesian classifiers, an ensemble classifier, an MLP classifier, seven linear regressors, six ensemble regressors, and an MLP regressor. Emulator performance was assessed according to training and testing scores, training time, and R^2 score on the QoIs from a blind dataset. The blind dataset comprised of six solutions to the reaction-diffusion equation that used randomly selected input parameters and were not seen during either training or testing phases. *This study addressed the following questions*: (1) *Can ML emulators accurately classify the mixing state of the anisotropic reaction-diffusion system*? (2) *How accurately do they predict QoIs for reactive mixing*? (3) *How fast can they be trained*? (4) *How does each emulator rank overall*?

2. Governing equations for reactive mixing

Let $\Omega \subset \mathbb{R}^d$ be an open bounded domain, where d indicates the number of spatial dimensions. The boundary is denoted by $\partial\Omega$, which is assumed to be piece-wise smooth. Let $\overline{\Omega}$ be the set closure of Ω and let spatial point $\mathbf{x} \in \overline{\Omega}$. The divergence and gradient operators with respect to $\mathbf{x}$ are denoted by div[$\bullet$] and grad[$\bullet$], respectively. Let $\mathbf{n}(\mathbf{x})$ be the unit outward normal to $\partial\Omega$. Let $t \in]0, \mathcal{I}[$ denote time, where $\mathcal{I}$ is the length of time of interest. The governing equations are posed on $\Omega \times]0, \mathcal{I}[$ and the initial condition is specified on $\overline{\Omega}$. Consider the fast bimolecular reaction where species A and B react irreversibly to yield product C:

$$n_A\, A + n_B\, B \longrightarrow n_C\, C. \tag{1}$$

The governing equations for this fast bimolecular reaction without volumetric sources/sinks are:

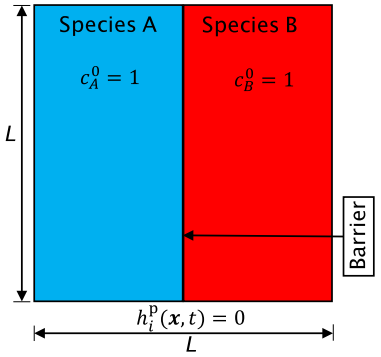


Fig. 1. Model domain for reactive-mixing: Schematic of the initial boundary value problem. L, $h_i^p(\mathbf{x},t)$, c_A^0, and c_B^0 are the length of the domain, diffusive flux on the boundary for ith chemical species, initial concentration of species A, and initial concentration of species B, respectively. Species A and B were initially on the left and right sides of the domain, respectively. Initial concentrations of A and B were 1.0 and mixing commenced at $t > 0$.

$$\frac{\partial c_A}{\partial t} - \text{div}[\mathbf{D}(\mathbf{x},t)\,\text{grad}[c_A]] = -n_A\,k_{AB}c_Ac_B \quad \text{in } \Omega\times]0,\mathcal{I}[, \tag{2a}$$

$$\frac{\partial c_B}{\partial t} - \text{div}[\mathbf{D}(\mathbf{x},t)\,\text{grad}[c_B]] = -n_B\,k_{AB}c_Ac_B \quad \text{in } \Omega\times]0,\mathcal{I}[, \tag{2b}$$

$$\frac{\partial c_C}{\partial t} - \text{div}[\mathbf{D}(\mathbf{x},t)\,\text{grad}[c_C]] = +n_C\,k_{AB}c_Ac_B \quad \text{in } \Omega\times]0,\mathcal{I}[, \tag{2c}$$

$$c_i(\mathbf{x},t) = c_i^p(\mathbf{x},t) \quad \text{on } \Gamma_i^D\times]0,\mathcal{I}[\quad (i = A,\ B,\ C), \tag{2d}$$

$$(-\mathbf{D}(\mathbf{x},t)\,\text{grad}[c_i])\cdot\mathbf{n}(\mathbf{x}) = h_i^p(\mathbf{x},t) \quad \text{on } \Gamma_i^N\times]0,\mathcal{I}[\quad (i = A,\ B,\ C), \tag{2e}$$

$$c_i(\mathbf{x},t=0) = c_i^0(\mathbf{x}) \quad \text{in } \overline{\Omega} \quad (i = A,\ B,\ C). \tag{2f}$$

See variable definitions in Appendix B. Traditional numerical formulations for Eqs. (2a)–(2f) can yield nonphysical solutions for chemical species concentration [46]. Also, when anisotropy dominates, the standard Galerkin formulation produces erroneous concentrations [46–49]. To overcome these problems, a non-negative, finite-element method was used to compute species concentrations [46]. This method ensured that concentrations were non-negative and satisfied the discrete maximum principle.

2.1. Reaction tank problem and associated QoIs

Fig. 1 depicts the initial boundary-value problem. The model domain is a square with $L = 1$. Zero-flux boundary conditions $h_i^p(\mathbf{x},t) = 0$ were enforced on all sides of the domain. For all chemical species, the non-reactive volumetric source $f_i(\mathbf{x},t)$ was equal to zero. Initially, species A and B were segregated (see Fig. 1) and stoichiometric coefficients were $n_A = 1$, $n_B = 1$, and $n_C = 1$. The total time of interest was $\mathcal{I} = 1$. The dispersion tensor is taken from the subsurface literature [46,50]:

$$\mathbf{D}_{\text{subsurface}}(\mathbf{x}) = D_m\mathbf{I} + \alpha_T\|\mathbf{v}\|\mathbf{I} + \frac{\alpha_L - \alpha_T}{\|\mathbf{v}\|}\mathbf{v}\otimes\mathbf{v}. \tag{3}$$

The preceding velocity field was used to define the dispersion tensor according to stream function [51–53]:

$$\psi(\mathbf{x},t) = \begin{cases} \frac{1}{2\pi\kappa_f}\left[\sin(2\pi\kappa_f x) - \sin(2\pi\kappa_f y) + v_0\cos(2\pi\kappa_f y)\right] & \text{if } \nu T \le t < \left(\nu+\frac{1}{2}\right)T, \\ \frac{1}{2\pi\kappa_f}\left[\sin(2\pi\kappa_f x) - \sin(2\pi\kappa_f y) - v_0\cos(2\pi\kappa_f y)\right] & \text{if } \left(\nu+\frac{1}{2}\right)T \le t < (\nu+1)\,T. \end{cases} \tag{4}$$

Using Eq. (4), the divergence-free velocity field components are:

$$v_x(\mathbf{x},t) = -\frac{\partial\psi}{\partial y} = \begin{cases} \cos(2\pi\kappa_f y) + v_0\sin(2\pi\kappa_f y) & \text{if } \nu T \le t < \left(\nu+\frac{1}{2}\right)T, \\ \cos(2\pi\kappa_f y) & \text{if } \left(\nu+\frac{1}{2}\right)T \le t < (\nu+1)\,T. \end{cases} \tag{5}$$

$$v_y(\mathbf{x},t) = +\frac{\partial\psi}{\partial x} = \begin{cases} \cos(2\pi\kappa_f x) & \text{if } \nu T \le t < \left(\nu+\frac{1}{2}\right)T, \\ \cos(2\pi\kappa_f x) + v_0\sin(2\pi\kappa_f x) & \text{if } \left(\nu+\frac{1}{2}\right)T \le t < (\nu+1)\,T. \end{cases} \tag{6}$$

In Eqs. (5)–(6), T controls the oscillation of the velocity field from clockwise to anti-clockwise. v_0 is the perturbation parameter of the underlying vortex-based flow field. Larger values of v_0 skew the vortices into ellipses while smaller values of v_0 yield circular vortex structures in the velocity field. $\frac{\alpha_L}{\alpha_T}$ controls the magnitude of the anisotropic dispersion contrast.

Smaller values of $\frac{\alpha_L}{\alpha_T}$ indicate less anisotropy and vice versa. The term $\kappa_f L$ governs the size of the vortex structures in the flow field [11,12]. Note that varying v_0 does not significantly alter vortex locations.

For reactive-mixing applications, the following QoIs were defined:

(1) Species production/decay, which can be analyzed by calculating normalized average concentrations, $\overline{c}_i$, and normalized average of squared concentrations, $\mathbb{c}$. $\mathbb{c}$ provided information on the species production/decay as a function of the eigenvalues of anisotropic dispersion. For example, Mudunuru et al. [12, Theorem 2.3] showed that $\mathbb{c}$ is bounded by the exponential functions representing the minimum and maximum eigenvalues of anisotropic dispersion. These quantities are:

$$\overline{c}_i = \frac{\langle c_i(t)\rangle}{\max\left[\langle c_i(t)\rangle\right]} \quad \text{where } \langle c_i(t)\rangle = \int_\Omega c_i(\mathbf{x},t)\,\mathrm{d}\Omega, \tag{7}$$

$$\mathbb{c} = \frac{\langle c_i^2\rangle}{\max\left[\langle c_i^2\rangle\right]} \quad \text{where } \left\langle c_i^2(t)\right\rangle = \int_\Omega c_i^2(\mathbf{x},t)\,\mathrm{d}\Omega. \tag{8}$$

(2) Degree of mixing is defined as the variance of concentration:

$$\sigma_{c_i}^2 = \frac{\langle c_i^2\rangle - \langle c_i\rangle^2}{\max\left[\langle c_i^2\rangle - \langle c_i\rangle^2\right]}. \tag{9}$$

Note that the values for $\overline{c}_i$, $\mathbb{c}$, and $\sigma_{c_i}^2$ are non-negative and range from 0 to 1 $\forall\ i = A, B, C$. Also, the selected QoIs (e.g., degree of mixing) are standard for reactive mixing [54,55].

2.2. Feature generation (numerical model inputs)

First, a two-dimensional non-negative numerical model was built using a first-order finite-element structured triangular mesh, which had 81 nodes on each side. The resulting finite element mesh was 81×81 with 6,561 degrees-of-freedom. There were 1,000 time steps to go from $t = 0$ to $t = 1$, so $\Delta t = 0.001$, and the backward Euler method was used to perform time marching. Mudunuru et al. [56–58] performed h-convergence studies in their previous work that solved a similar system of equations. Through grid- and time-step-refinement studies, they showed that 81×81 elements with $\Delta t = 0.001$ were sufficient to achieve accurate non-negative finite-element-method solutions with fine-scale spatio-temporal variation in dispersion. Next, a total of 2,500 high-fidelity numerical simulations were run for different sets of reaction-diffusion model input parameters, of which 2,315 ran to completion because the remaining (185) parameter combinations yielded unstable solutions. Each simulation comprised 1,000 time steps ($\mathcal{I} = 0.0$ to 1.0 with a uniform time step of 0.001). Features included: longitudinal-to-transverse anisotropic dispersion ratio $\frac{\alpha_L}{\alpha_T}$, molecular diffusion D_m, the perturbation parameter of the underlying vortex-based velocity field v_0, and velocity-field characteristic scales $\kappa_f L$ and T. Specifically, input parameters were: $v_0 = \left[1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\right]$, $\frac{\alpha_L}{\alpha_T} = \left[1, 10^{1}, 10^{2}, 10^{3}, 10^{4}\right]$, $D_m = \left[10^{-8}, 10^{-3}, 10^{-2}, 10^{-1}\right]$, $\kappa_f L = [1, 2, 3, 4, 5]$, and $T = [10^{-4}, 20^{-4}, 30^{-4}, 40^{-4}, 50^{-4}]$. α_T was varied with α_L held at 1.0. Five features for each of the 2,315 models with 1,000 time steps formed a feature matrix with the dimensions $2,315,000 \times 5$.

3. Machine learning emulators

3.1. Labels (QoIs) and preprocessing

Labels were the QoIs of the 2,315 simulations at each time step yielding label vectors. Features and labels were concatenated with the training and testing data, yielding a $2,315,000 \times 6$ matrix. For ML classification, the degree of mixing in the system was classified as: Class 1 (well mixed), Class 2 (moderately mixed), Class 3 (weakly mixed), and Class 4 (ultra-weak mixing). The corresponding σ_i^2 for these classes were 0.0–0.25, 0.25–0.5, 0.5–0.75, and 0.75–1.0, respectively. Of course, additional classes could be defined although this would necessitate re-training the ML emulators. These data were then partitioned into training and testing datasets during construction of the ML emulators and Table 2 lists the different partitions. Each emulator was trained using the three different data partitions and the performance of each was assessed. First, 0.9% of the data were used as training data to identify optimized hyperparameters and other adjustable parameters. Subsequently, emulators using the optimized hyperparameters were validated against the 63% and 81% data partitions. Later, validated emulators were tested on six blind datasets listed in Table 1.

Preprocessing is required for most ML model development. ML emulators that use the Euclidean norm (e.g., kernel-based methods) must have features/input parameters with a common scale to make accurate predictions [59–61]. Common preprocessors are standardization (recasting all feature data into the standard normal distribution $N(0,1)$), normalization (independently scaling each feature between 0 and 1), and max-abs scaling (scale and translate individual features such that the maximal absolute value of a feature is 1). In this study, no scaling was performed to data for random forests (RFs)

Table 1
Selections of model parameters used to develop the six blind datasets.

Parameter	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6
v_0	1	10^{-1}	10^{-3}	10^{-1}	10^{-2}	1
$\frac{\alpha_L}{\alpha_T}$	10^3	10^1	1	10^4	10^1	10^2
D_m	10^{-1}	10^{-3}	10^{-3}	10^{-2}	10^{-1}	10^{-2}
$\kappa_f L$	2	1	3	5	4	1
T	1×10^{-4}	1×10^{-4}	2×10^{-3}	4×10^{-4}	4×10^{-4}	5×10^{-4}

Table 2
Summary of training and testing data partitions used in ML emulator development and testing.

% of input data (No. of simulations)			Size of samples for QoIs	
Training data	Validation data	Testing data	Training	Testing
0.9% (20)	0.1% (3)	99% (2,292)	20,150	2,291,850
63% (1458)	7% (162)	30% (690)	1,458,500	694,500
81% (1875)	9% (208)	10% (230)	1,875,500	231,500

emulator because they are agnostic to feature scaling. However, because features are neither sparse nor skewed and do not have outliers, all data were standardized for the rest of the emulators. For polynomial regression, the data were quadratically transformed.

3.2. Optimization of hyperparameter and other adjustable parameters

ML emulators learn a function or a set of functions by comparing features and corresponding labels. Hyperparameters, different for each ML emulator, control the learning process. Common hyperparameters include regularization, learning rate, and the loss function. There are also additional adjustable parameters for each ML emulator that accelerate the learning process and make a more robust emulator, including the number of training iterations, kernel, truncation value, etc. Because hyperparameter optimization was the most time-consuming process of this exercise, 0.9% of the data (23 simulations) were used with the `gridsearch` algorithm in scikit-learn [62], a Python ML package. Tables 5 and 6 list the hyperparameters for each ML emulator. Next, 7% and 9% of the data were used for validation with 30% or 10% reserved as blind data for testing.

Because ML emulators can introduce bias during training, overfitting is a common phenomenon. To ameliorate this, k-fold cross-validation algorithm was used to avoid bias, to minimize overfitting, to ensure optimal computational times, and to calculate reliable variances [63,64]. In this work, 10-fold cross-validation was used. First, it subdivided training data into ten equal subsets. Then, it used nine sets for training with one set reserved for validation; this process was repeated leaving out each subset once. The average performance on the 10 withheld datasets are reported along with their variances.

3.3. ML emulators

This research developed 20 ML emulators to classify the state of reactive mixing and to predict the reactive-mixing QoIs. Among the 20 ML emulators, eight were linear, five Bayesian, six were ensemble, and there was one MLP model. The eight linear ML emulators were ordinary least-squares regression (LSQR), ridge regression (RR), lasso regression (LR), elastic-net regression (ER), Huber regression (HR), polynomial, logistic regression (LogR), and kernel ridge regression (KR). Among the linear emulators, only LogR was a classifier. The five Bayesian techniques were Bayesian ridge (BR), Gaussian process (GP), naïve Bayes (NB), linear discriminant analysis (LDA), and quadratic discriminant analysis (QDA) regression. Among these Bayesian emulators, LDA and QDA are classifiers and the remaining are regressors. The six ensemble ML emulators were bagging, decision tree (DT), random forest (RF), AdaBoost (AdaB), DT-based AdaB, and the gradient-boosting method (GBM). Among the six ensemble emulators, RF was used as both classifier and regressor. MLP was also used as both classifier and regressor. Each ML emulator is distinguished according to the formulation of its loss function. Linear regressors are distinguished according to the regularization component in the loss function. Each linear regressor has a unique loss function listed in Table 3. Nonlinear regressors and classifiers have standard/user defined loss functions that distinguish them. Table 4 lists the equation of each nonlinear emulator with loss functions.

3.4. Linear ML emulators

Linear ML emulators fit a line to the labels. The equations and corresponding loss functions for the linear emulators are listed in Table 3. A brief mathematical description of each linear ML emulator is provided in Appendix C. The equation for polynomial regression was not listed here because it applies the LSQR formula to quadratically scaled data. For LSQR and polynomial regression, the intercept was the hyperparameter. For RR, α_2 and ϵ (tolerance/threshold) were hyperparameters. For LR, α_1, ϵ, and the maximum number of iterations were examined. For ER, α_1, α_2, ϵ, l_1 ratio, and the maximum number of iterations were optimized. For HR, α_1, ϵ, and the maximum number of iterations were optimized. The optimized

Table 3
Equations and loss functions of the linear emulators. Variables are defined in Appendix B and equations are explained in Appendix C.

Emulator	Equation	Loss function
LSQR	$\hat{y}(\mathbb{w}, \mathbb{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = \mathbb{x} \cdot \mathbb{w}$	$L_{\text{LSQR}} = \min_{\mathbb{w}} \lVert \mathbb{X}\mathbb{w} - \mathbb{y} \rVert_2^2$
RR	Same as above	$L_{\text{RR}} = \min_{\mathbb{w}} \lVert \mathbb{X}\mathbb{w} - \mathbb{y} \rVert_2^2 + \alpha_2 \lVert \mathbb{w} \rVert_2^2$
LR	Same as above	$L_{\text{LR}} = \min_{\mathbb{w}} \lVert \mathbb{X}\mathbb{w} - \mathbb{y} \rVert_2^2 + \alpha_1 \lVert \mathbb{w} \rVert_1$
ER	Same as above	$L_{\text{ER}} = \min_{\mathbb{w}} \lVert \mathbb{X}\mathbb{w} - \mathbb{y} \rVert_2^2 + \alpha_1 \lVert \mathbb{w} \rVert_1 + \alpha_2 \lVert \mathbb{w} \rVert_2^2$
HR	Same as above	$L_{\text{HR}} = \min_{\mathbb{w}} \lVert \mathbb{X}\mathbb{w} - \mathbb{y} \rVert_2^2 + \alpha_1 \lVert \mathbb{w} \rVert_1 + \Sigma \sum_{i=1}^{m} \left[1 + H_\epsilon \left(\frac{\mathbb{x}_i \cdot \mathbb{w} - y_i}{\Sigma}\right)\right]$
LogR	$l = \log_b \left(\frac{p}{1-p}\right) = w_0 + w_1 x_1 + w_2 x_2$	$L_{\text{cross-entropy}} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i)\right]$
KR	$\mathcal{K}_{\text{RBF}}(\mathbb{x}_{\mathbb{1}}, \mathbb{x}_{\mathbb{2}}) = \exp\left(-\lambda \lVert \mathbb{x}_{\mathbb{1}} - \mathbb{x}_{\mathbb{2}} \rVert^2\right)$	$L_{\text{squared}} = \left(y - \hat{y}\right)^2$

Table 4
Equations and loss functions of the Bayesian, ensemble, and MLP emulators where $L_{\text{LML}} = -\frac{1}{2}\left[\log\lvert\omega^{-1}\mathbb{I} + \mathbf{\Phi}\mathbb{A}^{-1}\mathbf{\Phi}^{\intercal}\rvert + \mathbf{\Psi}^{\intercal}\left(\omega^{-1}\mathbb{I} + \mathbf{\Phi}\mathbb{A}^{-1}\mathbf{\Phi}^{\intercal}\right)^{-1}\mathbf{\Psi}\right] + \sum_{i=0}^{N}(o\log\beta_i - r\beta_i) + u\log\omega - w\omega$. Variables are defined in Appendix B and equations are explained in Appendix C.

Emulator	Equation	Loss function
BR	$p(\mathbb{w}\lvert\omega) = N\left(\mathbb{w}\lvert 0, \omega^{-1}\mathbb{I}\right)$	L_{LML}
GP	$p(\mathbb{y}\lvert\mathbb{X}, \mathbb{w}, \beta) = N(\mathbb{y}\lvert\mathbb{X}\mathbb{w}, \beta)$	L_{LML}
NB	$p(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left[-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right]$	Maximum $p(x)$
LDA	$p(y = k\lvert\mathbb{x}) = \frac{1}{(2\pi)^{J/2}\lvert\det(\sum_k)\rvert^{1/2}} \exp\left[-\frac{1}{2}(\mathbb{x} - \mu_k\mathbb{1}) \cdot (\sum_k)^{-1}(\mathbb{x} - \mu_k\mathbb{1})\right]$	$L_{\text{cross-entropy}}$
QDA	$p(y = k\lvert\mathbb{x})$	$L_{\text{cross-entropy}}$
DT	$G(Q, s) = \frac{n_{\text{left}}}{\Sigma_m} H\left[Q_{\text{left}}(s)\right] + \frac{n_{\text{right}}}{\Sigma_m} H\left[Q_{\text{right}}(s)\right]$	$L_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2$
Bagging	$\hat{f} = \sum_{i=1}^{M} f_i(\mathbb{x}_i)$	L_{MSE}
RF	$\hat{f}_{rf}^{M} = \frac{1}{M}\sum_{m=1}^{M} \mathfrak{T}(\mathbb{x})$	L_{MSE}
AdaB	$f = \inf\left[y \in \mathbb{y} : \sum_{m:h_m \le y} \log\left(\frac{1}{\theta_m}\right) \ge \frac{1}{2}\sum_m \log\left(\frac{1}{\theta_m}\right)\right]$	$L_{\text{square-loss}} = \frac{\lvert y_i(\mathbb{x}) - y_i\rvert^2}{J^2}$
DT-based AdaB	Use DT regression with AdaB procedure	$L_{\text{square-loss}}$
GBM	$f = \sum_{m=1}^{M} \gamma_m h_m(\mathbb{x}_m)$	L_{LSQR}
MLP	$a_{\mathcal{N}}^{(l)} = F\left(\sum_{\mathcal{K}=1}^{\mathcal{N}_{l-1}} w_{\mathcal{K},\mathcal{N}}^{(l)} a_{\mathcal{K}}^{(l-1)} + b_{\mathcal{N}}^{(l)}\right)$	L_{MSE}

hyperparameters and other adjustable parameters (bolded) for linear ML emulators are listed in Table 5. For LogR, multi-class (binary or multi-class), ϵ, and the maximum number of iterations were optimized and corresponding settings are presented in Table 5. Tested solvers include Newton's method, the limited-memory large-scale bound constrained (LBFGS) solver, and the stochastic average gradient (SAG) solver. For KR, α_1, λ, and kernels were optimized (see Table 5).

3.5. Bayesian ML emulators

Bayesian ML emulators apply Bayes' Law to learn a function from training features and labels to predict equivalent test labels using testing features. Equations for Bayesian ML emulators are listed in Table 4. Also, a brief mathematical description of each Bayesian ML emulator is provided in Appendix A. For BR, β, ω, maximum number of iterations, and ϵ were the hyperparameters and their optimized values are shown in bold in Table 6. For GP, the type of kernel was optimized with optimal listed in Table 6. In NB, only prior probability distributions and variance smoothing were hyperparameters. For LDA, the only hyperparameter was the solver, which can be singular value decomposition (SVD), LSQR, or eigenvalue decomposition. Among these three, SVD was fastest. For QDA, only the tolerance was a hyperparameter and its optimized value was 10^{-4}.

3.6. Ensemble emulators

If the relationship between features and labels is nonlinear, linear ML emulators are not expected to perform well. Instead, nonlinear ML emulators such as MLPs and ensemble methods should work better. Ensemble methods bootstrap (random sampling with replacement) data to develop different tree models/predictors. Each label is used with replacement as input for developing individual model or base learner; therefore, tree models have different labels based on the bootstrapping process. Because bootstrapping captures many uncorrelated base learners to develop a final model, it reduces variance; resulting in a reduced prediction error. Also, in ensemble models, many different trees learn the same target variable and make a final model by averaging them; therefore, they predict better than any single tree.

Ensemble techniques are further classified into bagging (bootstrapping aggregation) and boosting, which form many weak trees/learners into a strong tree. Weak learners are learnt during early stage of training that can provide a large prediction

Table 5
Hyperparameters and adjustable parameters for generalized linear ML emulators, LogR, and KR with the optimal parameters in bold.

Emulator	Hyperparameter and adjustable parameter	Interrogated values
LSQR	Fit intercept	**True**, False
RR	α_2	**1.0**, 100, 1,000
	Max. no. of iterations	10, 25, **50**, 300, 1,000
LR	α_1	10^{-1}, 10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
	ϵ	$\mathbf{10^{-3}}$, 10^{-4}
	Max. no. of iterations	50, 100, 300, $\mathbf{10^{3}}$, 10^{4}
ER	α_1 and α_2	10^{-1}, 10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
	ϵ	10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}
	l_1 ratio	0.1, **0.5**, 1.0
	Max. no. of iterations	10^{2}, 10^{3}, $\mathbf{10^{4}}$, 10^{5}
	Tolerance	10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
HR	α_1	10^{-1}, 10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
	ϵ	10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
	Max. no. of iterations	10, **50**, 100
LogR	Multi-class	OVR, **Multinomial**
	Solver	Newton-CG, LBFGS, **SAG**
	ϵ	10^{-3}, $\mathbf{10^{-4}}$, 10^{-5},
	Max. no. of iterations	10, **50**, 100, 200, 300
KR	α	10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
	λ	**1**, 2, 3
	Kernel	linear, polynomial, **radial basis function (RBF)**

Table 6
Hyperparameters for Bayesian emulators where optimal values are in bold. Exponential sine squared $(\text{sine}(x, x') = \sigma^2 \exp\left[-2\sin^2\left(\pi|x - x'|/\text{p}\right)/\ell^2\right])$ is parametrized by a length-scale, $\ell > 0$, and periodicity $\text{p} > 0$.

Emulator	Hyperparameter and adjustable parameter	Interrogated values
BR	No. of iterations	**100**, 200, 300
	ϵ	10^{-2}, 10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}
GP	Kernel	Exponential sine squared, **RBF**
NB	Priors	True, **None**
	Variance smoothing	10^{-7}, 10^{-8}, $\mathbf{10^{-9}}$, 10^{-10}
LDA	Solver	**SVD**, LSQR, Eigen
QDA	Tolerance	10^{-3}, $\mathbf{10^{-4}}$, 10^{-5}

error during test label prediction while strong trees are learnt during late stage of training that can predict test labels with low prediction error. While bagging emulators work best with strong and complex trees (e.g., fully developed decision trees), boosting emulators work best with weak tree (e.g., shallow decision trees). In this study, several averaging/bagging and boosting ensemble emulators were explored to classify and simulate reactive mixing. The averaging emulators include bagging and RF while boosting emulators include AdaB, DT-based AdaB, and GBM.

For DT, maximum tree depth, maximum number of features, and minimum sample splitting were optimized and optimized settings are in bold in Table 7. In bagging, number of trees, bootstrapping, and maximum number of features were optimized and their optimal settings are bold in Table 7. In RF, maximum depth of tree, number of trees in a forest, minimum sample splitting number, bootstrapping, and maximum feature number were optimized and their optimal settings listed in bold in Table 7. For AdaB and DT-based AdaB, number of trees, loss function, γ_{dt} (learning rate for decision tree), and γ_{dta} (learning rate for decision tree based AdaB) were optimized and the optimized settings are bolded in Table 7. In GBM, number of trees, sub-sampling, and γ_{m} were adjusted and the optimal settings are in bold in Table 7. For the MLP, the number of hidden layers, activation function, α, γ_{mlp}, solver, and the maximum number of iterations were hyperparameters with optimum values in bold in Table 8. Solvers in MLP were adaptive momentum (Adam), LBFGS, and stochastic gradient descent (SGD).

3.7. Performance metrics

Training time and R^2 score were the performance metrics for each emulator. The training time should be fast while R^2 measures the correlation between $\mathbb{y}$ and $\hat{\mathbb{y}}$ where $y_i \in \mathbb{y}$. For n pairs of data points, the R^2 score is:

Table 7
Hyperparameters and adjustable parameters for ensemble ML emulators with the optimal in bold.

Emulator	Hyperparameter and adjustable parameter	Interrogated values
DT	Maximum depth	2, 3, **None**
	Max. no. of features	3, 4, **5** (maximum # of features)
	Min. sample splits	3, 4, **5**
Bagging	No. of trees	50, **100**, 200, 500
	Bootstrap	**True**, False
	Max. no. of features	3, 4, **5**
RF	Maximum depth	2, 3, **None**
	No. of trees in the forest	250, **500**, 1,000
	Bootstrap	True, **False**
	Max. no. of features in a tree	3, **4**, 5
	Min. sample splits	**2**, 3, 4
AdaB	No. of trees	50, **100**, 200, 300
	Loss function type	linear, **square**, exponential
	γ_{dt}	0.1, 0.5, 0.75, **1.0**
DT-based AdaB	No. of trees	50, **100**, 200, 500
	Loss function type	linear, **square**, exponential
	γ_{dta}	0.1, 0.5, **1.0**
GBM	No. of trees	50, **100**, 200, 500
	Sub-sample	0.25, **0.5**, 0.7, 0.8
	γ_{m}	0.05, **0.1**, 0.25, 0.5

Table 8
Hyperparameters and adjustable parameters for MLP emulator with the optimal parameters in bold.

Emulator	Hyperparameter and adjustable parameter	Interrogated values
MLP	No. of hidden layers	25, 50, 100, **200**, 300, 400, 500
	Activation function	**ReLU**, tanh, logistic
	α	10^{-1}, 10^{-2}, $\mathbf{10^{-4}}$, 10^{-5}
	γ_{mlp}	10^{-1}, 10^{-2}, $\mathbf{10^{-3}}$, 10^{-4}
	Solver	**Adam**, LBFGS, SGD
	Max. no. of iterations	100, **200**, 500, 10^3, 5×10^3

$$R^2 = \frac{\sum_{i=1}^{n} \left(y_i - \overline{y}\right)^2 - \sum_{i=1}^{n} \left(y_i - \hat{y}_i\right)^2}{\sum_{i=1}^{n} \left(y_i - \overline{y}\right)^2}, \tag{10}$$

where $\overline{y}$ is the mean of a feature. For classification, the performance metric was defined as:

$$\text{Accuracy} = \frac{1}{n_{\text{samples}}} \sum_{i=1}^{n_{\text{samples}}} \mathbf{1}(y) \left(\hat{y}_i = y_i\right), \tag{11}$$

where $\mathbf{1}(y)$ is the indicator function [39].

4. Results

After time $t = 0$, reactants A and B were allowed to mix and form product C. The extent of mixing depended upon the reaction-diffusion inputs (features). Increasing the degree of mixing increased the yield of product C. Product C yields at normalized simulation times $t = 0$, 0.5, and 1.0 are shown in Figs. 2–4, revealing the significance of $k_f L$ on product formation at different times. The importance of $\frac{\alpha_L}{\alpha_T}$ on product formation at various times is also evident. For $k_f L = 2$ and $\frac{\alpha_L}{\alpha_T} = 10^3$ (see Fig. 2 (a-c)) at $t = 0.1$, there was little reaction at the center of the vortices. However, regions with zero concentration decreased as $k_f L$ increased. For example, at $k_f L = 3$ and $t = 1.0$, more product was formed and negligible regions where $C = 0$ were present in the model domain. At $k_f L = 5$ and $t = 1.0$, the system was well-mixed even at high anisotropy ratios. Because $k_f L$ increased the number of vortices, which enhanced reactant mixing, it increased product yield. Fig. 3 shows product C yield under moderate anisotropy. Reducing the anisotropy ratio from 1,000 to 100 improved product yield even under low $\kappa_f L$ (see Fig. 3(c)). Among $\frac{\alpha_L}{\alpha_T}$, $k_f L$, and D_m, $\frac{\alpha_L}{\alpha_T}$ controlled the reaction at early times while $k_f L$ and D_m controlled the reaction at late times. Higher values of $\frac{\alpha_L}{\alpha_T}$ decreased product yield while higher values of $k_f L$ and D_m increased product yield.

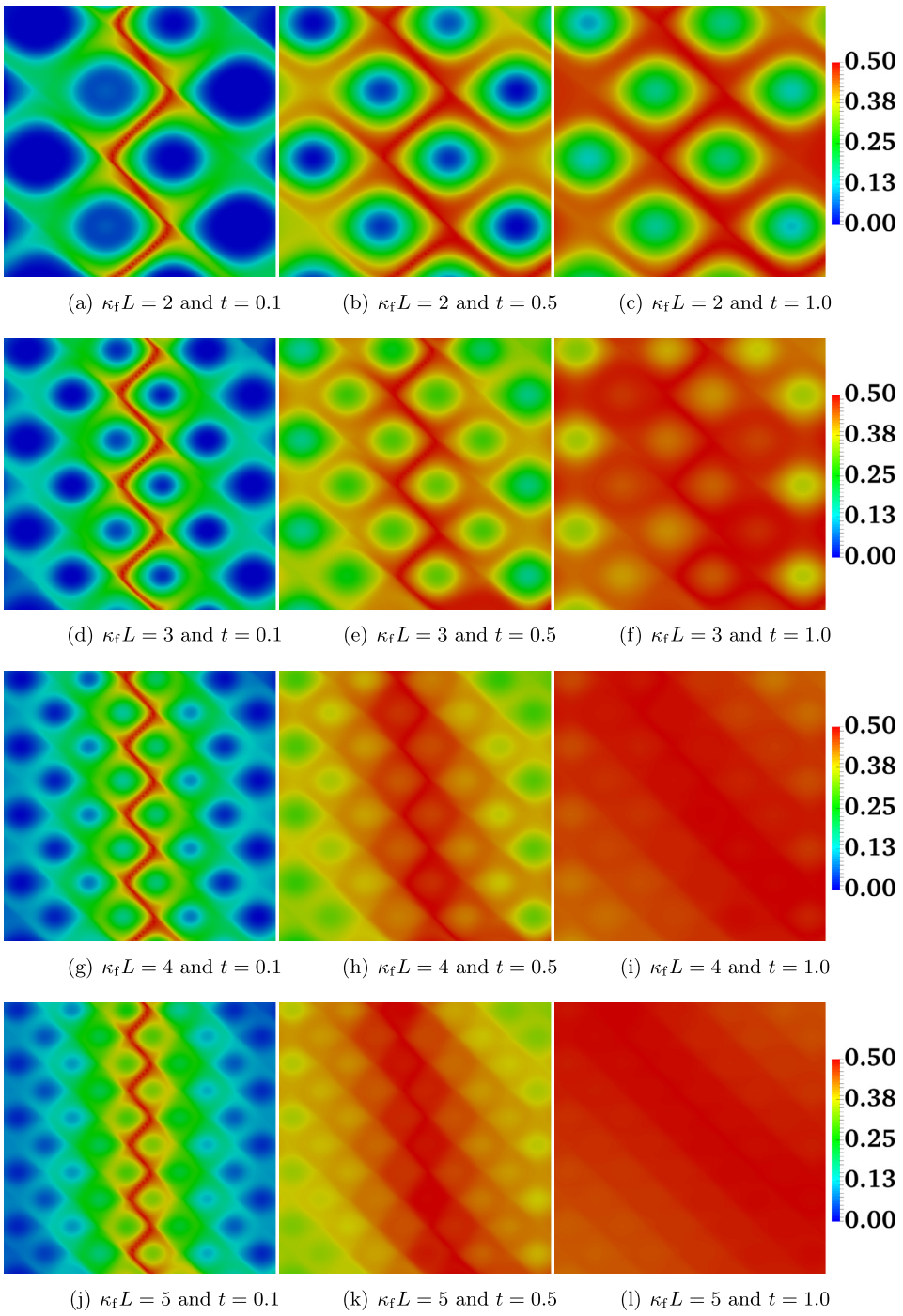
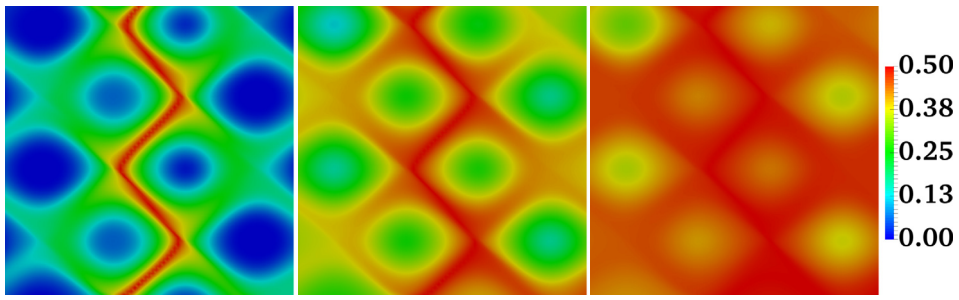
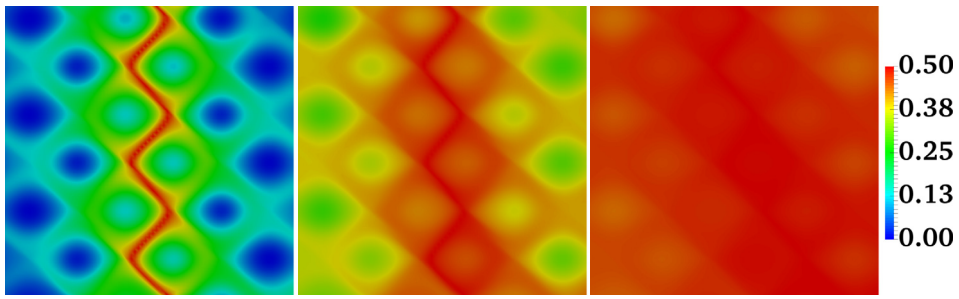


Fig. 2. Concentration contours of species C under high anisotropy: Concentration of product C at times $t = 0.1$, 0.5, and 1.0. Other input parameters were $\frac{\alpha_L}{\alpha_T} = 10^3$ (high anisotropy), $v_0 = 1$, $T = 0.1$, and $D_m = 10^{-3}$. Increased $\kappa_f L$ increases C production, especially at later times. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

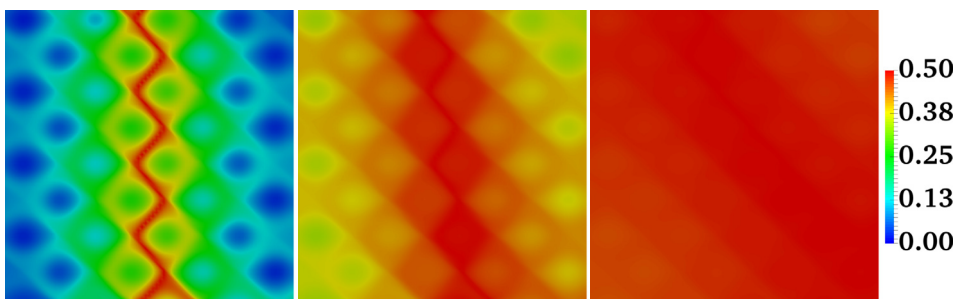
ML emulators were also used to classify the mixing state of the system. Out of 20 ML emulators, only LogR, LDA, QDA, RF, and MLP were used for classification. Table 9 shows the training score, testing score, sample sizes, and training time for each linear ML emulator. Because the progress of reactive-mixing is nonlinear, linear ML emulators (e.g., LogR, LDA, QDA) failed to learn an accurate function for the state of mixing. Mixing state classification by linear classifiers on training and testing data had accuracies <80%. Nonlinear classifiers such as RF and MLP were quite accurate at >95%. Results from RF and MLP were used to plot the confusion matrix of Fig. 5. The confusion matrix for RF and MLP was constructed using approximately 1% of data (23 simulations as training data) while the remaining 99% (2,292 simulations) were used as testing data. In the



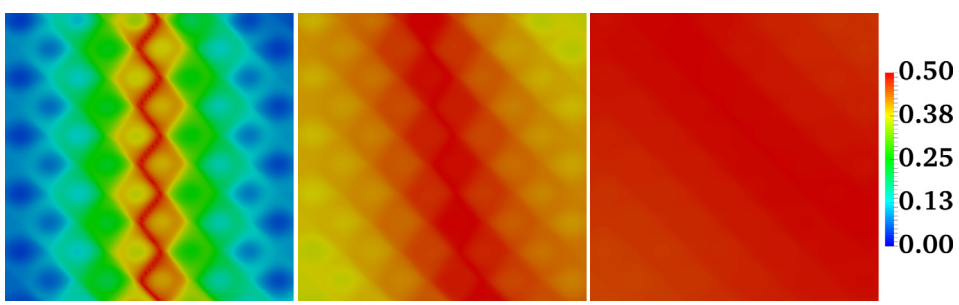
(a) $\kappa_f L = 2$ and $t = 0.1$ (b) $\kappa_f L = 2$ and $t = 0.5$ (c) $\kappa_f L = 2$ and $t = 1.0$



(d) $\kappa_f L = 3$ and $t = 0.1$ (e) $\kappa_f L = 3$ and $t = 0.5$ (f) $\kappa_f L = 3$ and $t = 1.0$



(g) $\kappa_f L = 4$ and $t = 0.1$ (h) $\kappa_f L = 4$ and $t = 0.5$ (i) $\kappa_f L = 4$ and $t = 1.0$



(j) $\kappa_f L = 5$ and $t = 0.1$ (k) $\kappa_f L = 5$ and $t = 0.5$ (l) $\kappa_f L = 5$ and $t = 1.0$

Fig. 3. Concentration contours of species C under moderate anisotropy: Concentration of product C at times $t = 0.1$, 0.5, and 1.0. Other input parameters were $\frac{\alpha_L}{\alpha_T} = 100$ (moderate anisotropy), $v_0 = 1$, $T = 0.1$, and $D_m = 10^{-3}$. Lower anisotropy increased C production than higher anisotropy in Fig. 2.

confusion matrix, the diagonal and off-diagonal elements indicate true and false predictions, respectively. The RF and MLP emulators false-prediction scores were less than 2% and 10%, respectively. Similar trends were observed for species A and B, hence the confusion matrices were not necessary.

Table 10 shows the training and testing scores for the six linear ML emulators. Although the training times were short (always <20 minutes), the training and testing R^2 scores never exceeded 73%. Also, the three Bayesian ML emulators (i.e., BR, GP, NB) predicted QoIs with accuracies on par with the linear emulators. Training and testing scores of BR and NB were <75%. GPs failed to converge for large datasets; however, GP trained on a smaller sample size scored >99%. The better

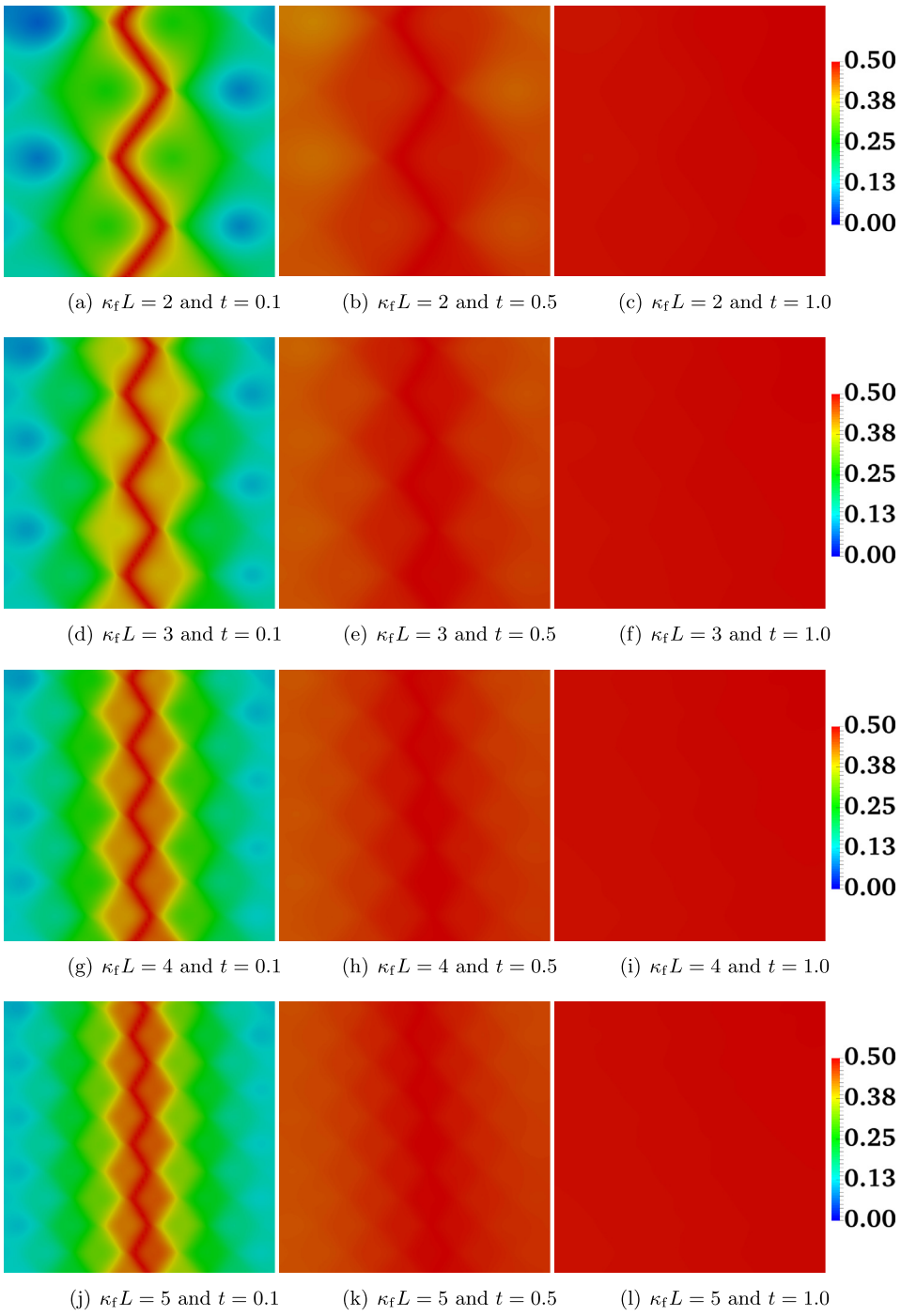


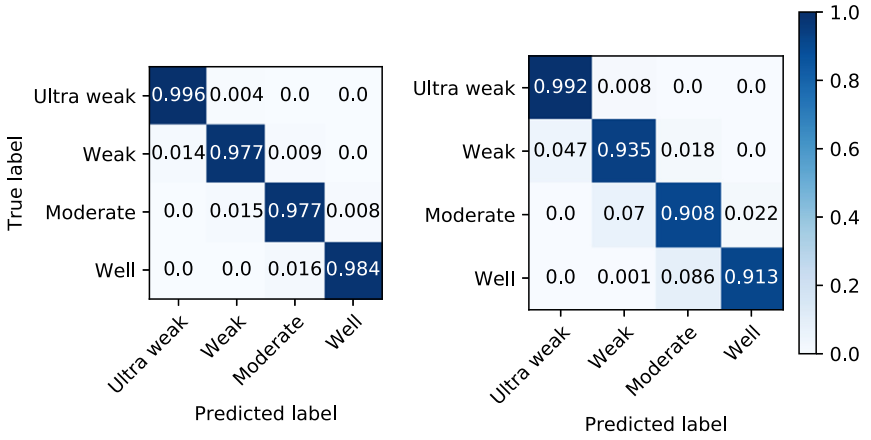
Fig. 4. Concentration contours of species C under low anisotropy: Concentration of product C at times $t = 0.1$, 0.5, and 1.0. Other input parameters were $\frac{\alpha_L}{\alpha_T} = 10$ (low anisotropy), $v_0 = 1$, $T = 0.1$, and $D_m = 10^{-3}$. At low anisotropy, production of C increased. During late times (e.g., $t = 0.5$ and 1.0), diffusion dominates C production while $\kappa_f L$ and $\frac{\alpha_L}{\alpha_T}$ minimally affect C production.

predictive capability of GP compared to other Bayesian ML emulators was attributed to the RBF kernels. As species A and B decayed while product C increased, the RBF kernels used by GP emulators were better suited to model this reactive-mixing system. Hence, GP emulators trained on small (0.25% of the data) datasets performed best.

Table 11 compares the training and testing scores for ensemble and MLP emulators. The R^2 scores for training and testing datasets were >90% (i.e., bagging, DT, RF, MLP). For the six blind realizations, bagging, DT, RF, AdaB, DT-based AdaB, and GBM showed near perfect performance. Here, only figures for RF and GBM emulators (Figs. 6–7) are shown because the

Table 9
Performance metrics of ML emulators on the training and test datasets for classifying the mixing state of the reaction-diffusion system.

Emulator	Training size (%)	Testing size (%)	Training score (%)	Testing score (%)	Training time (s)
LogR	0.9	99	75	75	31
	63	30	75	75	138
	81	10	75	75	174
LDA	0.9	99	72	72	28
	63	30	72	72	93
	81	10	72	72	102
QDA	0.9	99	77	77	66
	63	30	77	77	128
	81	10	77	77	133
RF	0.9	99	100	98	6,527
	63	30	100	99	22,161
	81	10	100	99	24,015
MLP	0.9	99	97	96	3,384
	63	30	99	99	50,397
	81	10	99	99	66,381



(a) Species C: RF (b) Species C: MLP

Fig. 5. Confusion matrices classifying the degree of mixing for the RF (left) and MLP (right) emulators.

remaining ensemble emulators had similar trends. These results indicated that tree-based methods outperformed linear ML methods in capturing QoIs of this reactive-mixing system. Also, Fig. 8 shows the QoI predictions by the MLP emulator for the six blind realizations. The consistently high test scores ($R^2 > 99\%$) indicated that overfitting was not a problem. As the size of the training dataset increased, the ensemble and MLP emulator development time increased.

Finally, the computational costs of running the numerical model and the ML emulators were compared. Tables 9–11 compare the computational costs for developing the various ML emulators on an Intel(R) Xeon(R) CPU E5-2695 v4 2.10 GHz. A single numerical simulation required approximately 1500 s on a single core. Testing an ML emulator (e.g., RF, MLP) took only $0.01 - 0.1$ s about 1/100,000th of the time to complete a high-fidelity numerical simulation.

5. Discussion

A suite of linear, Bayesian, and nonlinear ML emulators were trained to classify and replicate QoIs from high-fidelity, anisotropic, bi-linear diffusion numerical simulations. For this highly nonlinear system, linear and Bayesian ML emulators never exceeded 70% classification accuracy while LogR and QDA achieved only 75% and 77% classification accuracies, respectively. On the other hand, nonlinear emulators performed well (95% classification accuracies for RF and MLP). For the regression problem (predicting the three QoIs for each chemical species), as expected, linear regressors predicted QoIs at only $R^2 = 69\%$, but DT-based ensembles and the MLP neural network performed remarkably well. DTs (with and without AdaB), RFs, and the MLP all had $R^2 \geq 99\%$ with GBM (98%), bagging (95%), and AdaB (85%) performing somewhat

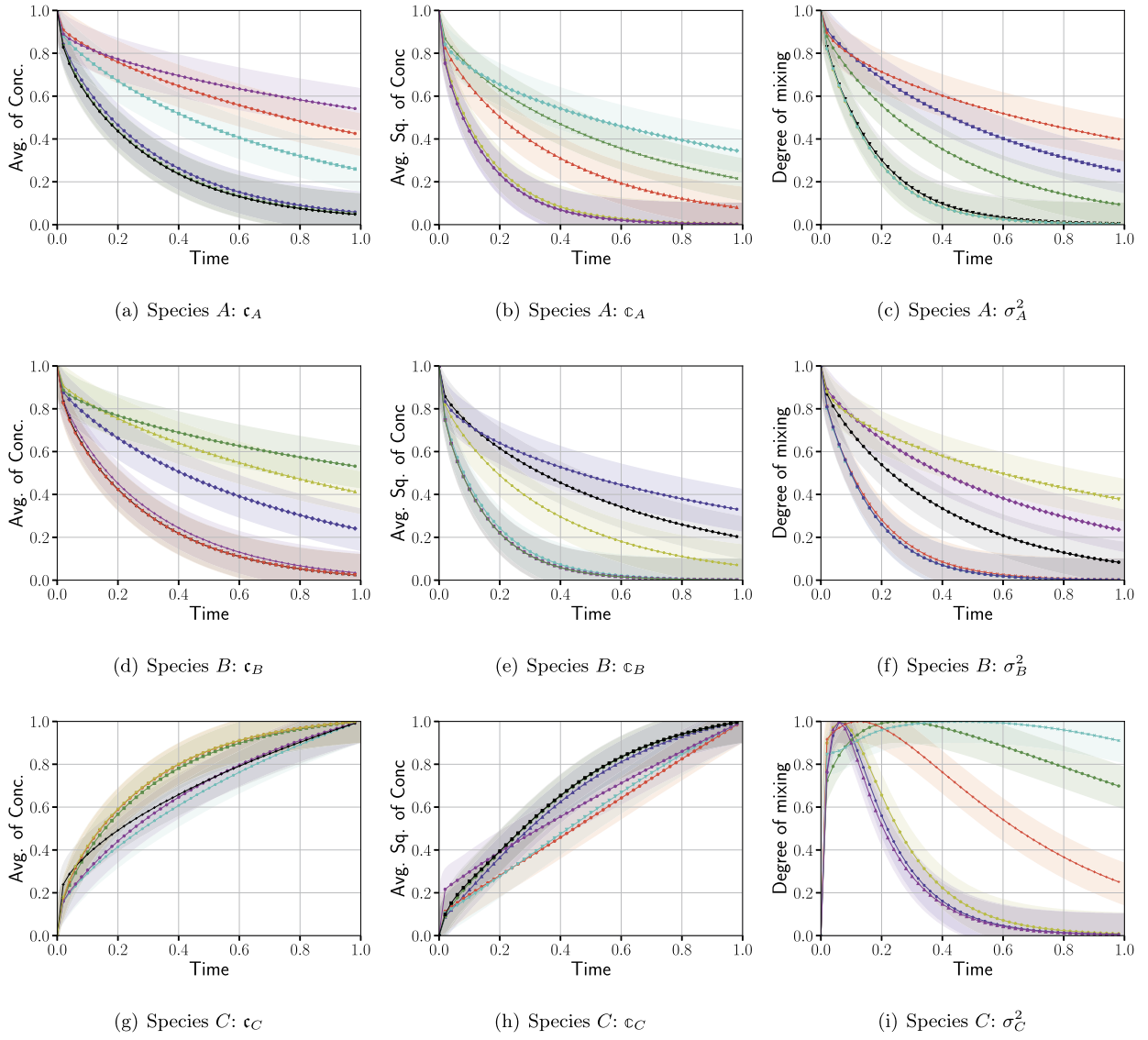
Table 10
Performance metrics of linear and Bayesian ML emulators (regressors). Note, GP and KR failed to converge even when trained using only 1% of training data because of a memory leak.

Emulator	Training size (%)	Testing size (%)	Training score (%)	Testing score (%)	Training time (s)
	0.9	99	69	69	12
LSQR	63	30	69	69	52
	81	10	69	69	57
	0.9	99	69	69	10
RR	63	30	69	69	42
	81	10	69	69	50
	0.9	99	69	69	95
LR	63	30	69	69	330
	81	10	69	69	368
	0.9	99	69	69	121
ER	63	30	69	69	1,077
	81	10	69	69	1,227
	0.9	99	69	69	14
HR	63	30	69	69	185
	81	10	69	69	195
	0.9	99	89	89	79
Polynomial	63	30	89	99	143
	81	10	89	89	164
	0.9	99	69	69	12
BR	63	30	69	69	62
	81	10	69	69	69
	0.9	99	73	73	69
NB	63	30	73	73	73
	81	10	73	73	91

Table 11
Performance metrics of ensemble and MLP emulators.

Emulator	Training size (%)	Testing size (%)	Training score (%)	Testing score (%)	Training time (s)
	0.9	99	100	99	42
DT	63	30	99	99	100
	81	10	99	99	110
	0.9	99	98	95	42
Bagging	63	30	98	95	110
	81	10	98	95	100
	0.9	99	100	99	1,435
RF	63	30	100	99	5,468
	81	10	100	99	6,044
	0.9	99	90	90	72
AdaB	63	30	89	89	1,378
	81	10	89	89	1,585
	0.9	99	99	99	103
DT-based AdaB	63	30	99	99	1,648
	81	10	99	99	1,778
	0.9	99	98	98	133
GBM	63	30	98	98	1,533
	81	10	98	98	2,048
	0.9	99	99	99	688
MLP	63	30	99	99	4,678
	81	10	99	99	9,691

worse. These results indicated that ensemble emulators outperformed other ML emulators in predicting the progress of reactive mixing on unseen data. However, they did not performed equally well. For example, RF outperformed other averaging ensembles (i.e., bagging and DT) while DT-based AdaB outperformed other boosting methods (i.e., AdaB and GBM). Bagging/averaging ensemble methods introduced randomness through voting-based evaluation metrics; therefore, their per-



(a) Species A: $\mathfrak{c}_A$ (b) Species A: $\mathbb{c}_A$ (c) Species A: σ_A^2

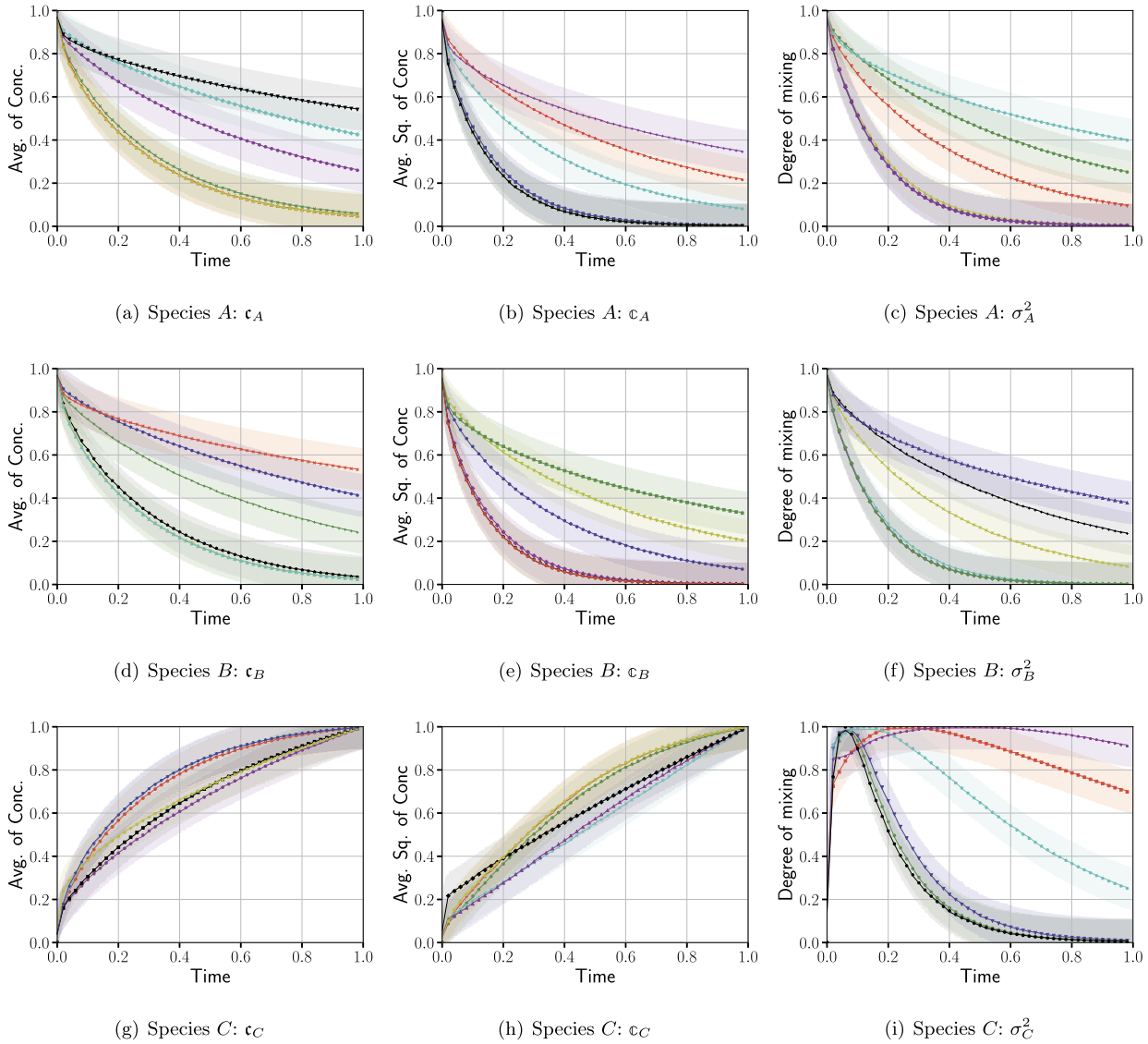
(d) Species B: $\mathfrak{c}_B$ (e) Species B: $\mathbb{c}_B$ (f) Species B: σ_B^2

(g) Species C: $\mathfrak{c}_C$ (h) Species C: $\mathbb{c}_C$ (i) Species C: σ_C^2

Fig. 6. Predictions of QoIs by Random Forest emulators for the six blind datasets: This figure shows the true (symbols) and RF emulator predictions (solid curves) of average concentrations, squared of average concentrations, and degree of mixing (a)–(c) of species A; (d)–(f) of species B, and (g)–(i) of species C. The color band denotes the confidence/prediction intervals. Specifically, it represents the upper and lower prediction estimates obtained from ML-emulator ensembles.

formances differed. For example, DTs often used the first feature to split, hence the order of the variables in the training data impacted results. Also, trees were pruned and not full grown in DTs, while RFs have unpruned, fully grown trees that are insensitive to feature order. Moreover, each tree in an RF learned using random sampling, and at each node, a random set of features was considered for splitting. This random sampling and splitting introduces tree diversity in a forest. After randomly selecting features, RF built a number of regression trees before averaging (bagging) them. Given enough trees, combinations of randomly selected features, and averaging (voting), RF emulators reduced the variance of predictions and minimized overfitting; their performances were best among averaging ensemble emulators.

Among boosting methods, DT-based AdaB outperformed AdaB and GBM because it combines DT and boosting estimates to predict QoIs. In this study, the DT-based AdaB used 100 trees as a base estimator to build the DT-based AdaB emulator. Two base estimators reduced the uncertainties in QoI predictions, and as a result, the DT-based AdaB emulator scored better than other two boosting approaches. Based on the analyses presented in Sec. 4, linear and Bayesian ML emulators (e.g., NB, BR, GP) performed poorly when classifying and predicting reactive-mixing QoIs. Overall, RF, DT-based AdaB, GBM, and MLP emulators accurately predicted unseen realizations with average accuracies >90%. Regarding computational cost, generalized linear and Bayesian ML emulators were faster to train than ensemble and MLP emulators. Among ensemble and boosting methods, RF and GBM emulators took the longest to train. Also, MLP emulators were more expensive to develop than other



(a) Species A: $\mathfrak{c}_A$ (b) Species A: $\mathbb{c}_A$ (c) Species A: σ_A^2

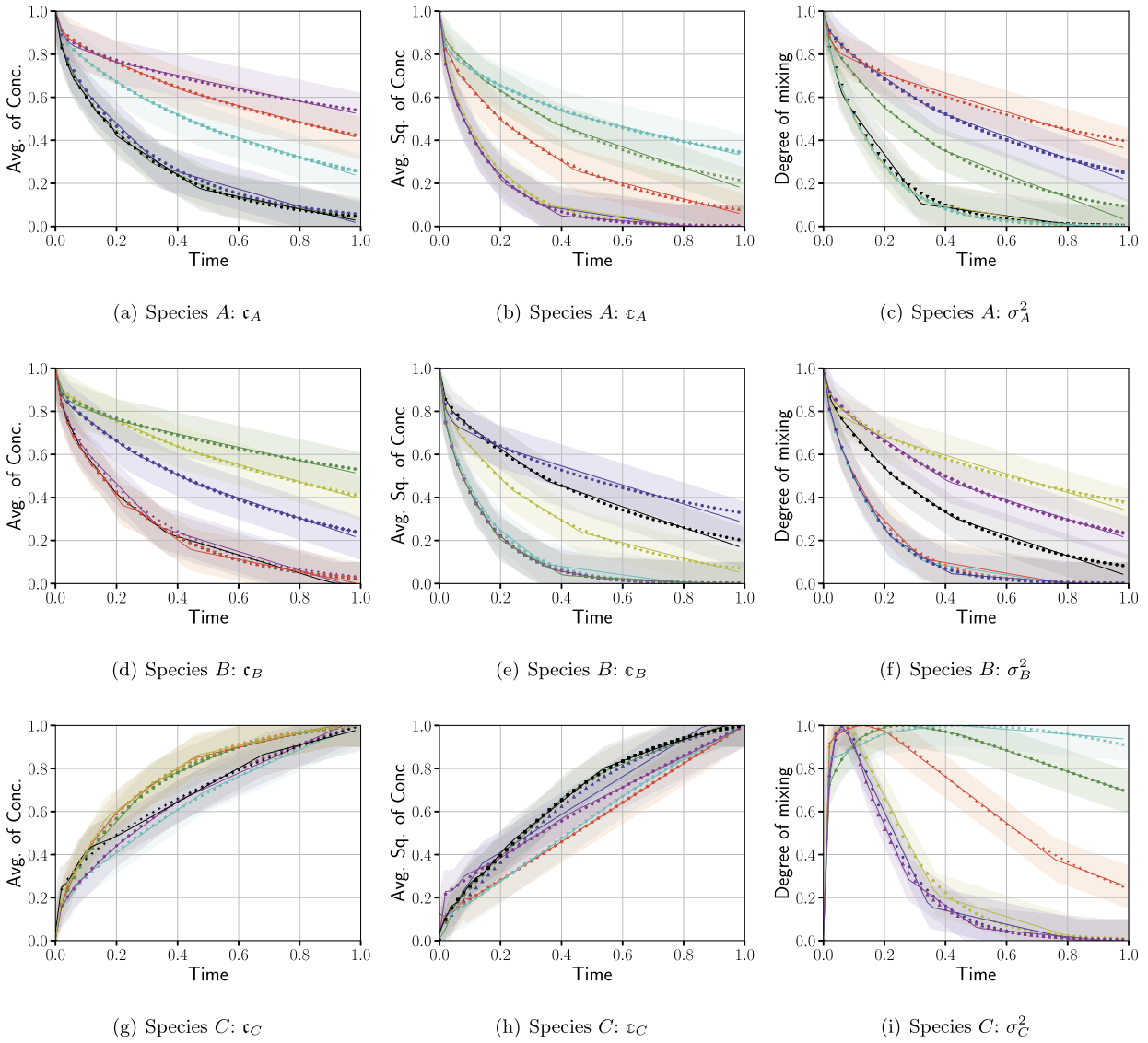
(d) Species B: $\mathfrak{c}_B$ (e) Species B: $\mathbb{c}_B$ (f) Species B: σ_B^2

(g) Species C: $\mathfrak{c}_C$ (h) Species C: $\mathbb{c}_C$ (i) Species C: σ_C^2

Fig. 7. Predictions of QoIs by GBM emulators for the six blind datasets: This figure shows the true (symbols) and GBM emulator predictions (solid curves) of average concentrations, squared of average concentrations, and degree of mixing (a)–(c) of species A; (d)–(f) of species B, and (g)–(i) of species C. The color band denotes the confidence/prediction intervals. Specifically, it represents the upper and lower prediction estimates obtained from ML-emulator ensembles.

ML emulators. However, both RFs and MLPs can be parallelized on both CPUs [65,66] and GPUs [67,68] to greatly reduce training times. Although training ensemble and MLP emulators is computationally expensive but they took 1/100,000th of the time required for a high-fidelity simulation to predict equivalent QoIs.

This study demonstrated how ML models can be used as surrogates to predict reactive-mixing QoIs without needing to run high-fidelity numerical simulations. Now that we have interrogated the various architectures available to develop emulators, our results could be extended to solving for QoIs from partial differential equations used to study turbulent momentum transfer [69], enthalpy mixing with entropy generation [70], and turbulent combustion [71]. Of course, there is no free lunch, and these problems still require running a sufficient number of numerical simulations to generate the training matrix and then interrogating the hyperparameter space before training the ML emulator. But, because of this work, a modeler should feel confident that using an RF or MLP model is the way to go. The effort required to generate the data needed to retrain these emulators is minimal as evident in Table 11 (e.g., using only 1% of available data when training resulted in >95% accuracy). The re-trained ML-emulators could then be used to predict new QoIs without requiring solution of computationally expensive numerical models. And finally, based on these results the range of hyperparameters interrogated could be significantly shortened.



(a) Species A: $\mathfrak{c}_A$ (b) Species A: $\mathfrak{c}_A$ (c) Species A: σ_A^2

(d) Species B: $\mathfrak{c}_B$ (e) Species B: $\mathfrak{c}_B$ (f) Species B: σ_B^2

(g) Species C: $\mathfrak{c}_C$ (h) Species C: $\mathfrak{c}_C$ (i) Species C: σ_C^2

Fig. 8. Predictions of QoIs by the MLP emulator for the six blind datasets: This figure shows the true (symbols) and MLP emulator predictions (solid curves) of average concentrations, squared of average concentrations, and degree of mixing (a)–(c) of species A; (d)–(f) of species B, and (g)–(i) of species C. The color band denotes the confidence/prediction intervals. Specifically, it represents the upper and lower prediction estimates obtained from ML-emulator ensembles.

6. Conclusions

Our primary purpose was to accurately quantify reactive-mixing state and expedite predictions of species concentrations (QoIs) due to reactive mixing. A suite of linear, Bayesian, ensemble, and MLP ML emulators was developed to predict species concentrations and to classify the state of reactive mixing. All ML emulators were trained to reproduce numerical simulations. A total of 2,315 simulations were completed to generate data to train and test the emulators. Data were generated by solving the anisotropic reaction-diffusion equations. Because of the highly nonlinear reactive-mixing system, linear and Bayesian (except GP) ML emulators performed poorly when classifying and predicting the state of reactive mixing (e.g., $R^2 \approx 70\%$). Among Bayesian ML emulators, GP showed promise for accurate prediction of QoIs for small datasets. On the other hand, ensemble and MLP emulators accurately classified the state of reactive mixing and predicted associated QoIs. For example, RF and MLP emulators classified the state of reactive mixing with accuracies >90% while predicting the progress of reactive mixing with accuracies >95% on training, testing, and unseen data. Among bagging ensemble methods, RF emulators outperformed bagging and DT emulators. Similarly, among boosting ensemble methods, DT-based AdaB emulators yielded better predictions than AdaB and GBM emulators. Computationally, for QoI predictions, ML emulators were approximately 10^5 faster than the corresponding numerical simulation. Finally, ensemble ML and MLP emulators proved

good classifiers and predictors for simulating reactive mixing. Our future work will investigate temporal QoIs by estimating time derivatives. This can be achieved by applying entirely different ML architectures such as recurrent neural networks or long short-term memory networks.

Declaration of competing interest

The authors declare that they do not have conflict of interest.

Computer code availability

Codes for machine learning implementation are available from the public Github repository https://github.com/bulbulahmmed/ML-to-reactive-mixing-data. Additional information regarding the simulation datasets can be obtained from Bulbul Ahmmed (Email: bulbul_ahmmed@baylor.edu) and Maruti Kumar Mudunuru (Email: maruti@pnnl.gov).

Acknowledgements

BA is grateful for the support from the Mickey Leland Energy Fellowship (MLEF) awarded by U.S. Department of Energy (DOE) Office of Fossil Energy (FE). MKM and SK also thank the support of the LANL Laboratory Directed Research and Development (LDRD) Early Career Award 20150693ECR. VVV thanks the support of LANL LDRD-DR Grant 20190020DR. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001). MKM was partially supported by ExaSheds project during the paper writing process. ExaSheds is supported by the U.S. Department of Energy, Office of Science, Office of Biological and Environmental Research, Earth and Environmental Systems Sciences Division, Data Management Program, under Award Number DE-AC02-05CH11231. Additional information regarding the simulation datasets and codes can be obtained from Bulbul Ahmmed (E-mail: ahmmedb@lanl.gov) and Maruti Kumar Mudunuru (E-mail: maruti@pnnl.gov). The authors thank the reviewers whose feedback helped in substantially improving the manuscript.

Appendix A. Rationale for the selected parameter ranges

The parameter range described in Subsection 2.2 corresponds to a wide range of spatio-temporal scales in the velocity field and anisotropic diffusion tensor. Specifically:

- $v_0 = \left[1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\right]$. $v_0 = 1$ yields a velocity field with elliptical vortex structures. The minimum value of $v_0 = 10^{-4}$ yields a velocity field with circular vortex structures. Physically, elliptical vortex structures afford increased reactive mixing (e.g., in the direction of major axis of the ellipse) compared to circular vortex structures. This range of values for $v_0 = 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$ was selected to account for transition from symmetric and asymmetric mixing scenarios.
- $\frac{\alpha_L}{\alpha_T} = \left[1, 10^1, 10^2, 10^3, 10^4\right]$. This range of anisotropy ratios begins with the isotropic case, $\frac{\alpha_L}{\alpha_T} = 1$, and ends in a strongly anisotropic regime, $\frac{\alpha_L}{\alpha_T} = 10^4$, where longitudinal dispersion (α_L along the streamlines) is 10^4 times higher than transverse dispersion (α_T across streamlines). This work and our previous research [12, Section 4] demonstrated that anisotropy ratio is one of the most sensitive parameters using RF classification in the system of PDEs, so we investigated an especially broad range of this ratio.
- $D_m = \left[10^{-8}, 10^{-3}, 10^{-2}, 10^{-1}\right]$. Molecular diffusion scenarios were constrained with D_m between 10^{-8} and 10^{-1}, representing low- and high-molecular diffusion scenarios, respectively.
- $\kappa_f L$ = $[1, 2, 3, 4, 5]$. $\kappa_f L = 1$ and $\kappa_f L = 5$ represent large-scale and small-scale vortex structures in the velocity field, respectively. Higher values of $\kappa_f L$ enhance mixing, which is evident from Figs. 2–4. To be specific, please compare Fig. 2(a), (d), (g), and (j).
- $T = [1 \times 10^{-4},\ 2 \times 10^{-4}, 3 \times 10^{-4}, 4 \times 10^{-4}, 5 \times 10^{-4}]$. $T = 1 \times 10^{-4}$ and $T = 5 \times 10^{-4}$ equate to velocity fields oscillating at 10 and 2 kHz, respectively.

These range of values for each parameter allow investigation of the entire domain of mixing scenarios (e.g., uniform mixing under low $\frac{\alpha_L}{\alpha_T}$ and/or high D_m, preferential mixing under high $\frac{\alpha_L}{\alpha_T}$, or incomplete mixing due to large-scale vortex structures, $k_f L = 2$). The system always starts in the unmixed state, but depending on the choice of model parameters, it may or may not end up in a fully mixed state. Under low anisotropy, $\frac{\alpha_L}{\alpha_T} \leq 10$, the system always evolves into a fully mixed state. Under high-anisotropy, $\frac{\alpha_L}{\alpha_T} > 100$, with large-scale vortex structures, $k_f L \leq 3$, there was incomplete mixing. Under high anisotropy, $\frac{\alpha_L}{\alpha_T} \geq 100$, with small-scale vortex structures, $k_f L \geq 4$, there was preferential mixing. This broad parameter ranges facilitated a thorough interrogation of the behaviors of this system. Starting from the unmixed state, the choice of parameters changes the progress of the reaction allowing incomplete mixing, preferential mixing, or full mixing.

Appendix B. Nomenclature

B.1. Variables

$\mathbb{A}$ = Diagonal matrix
$a^l_{\mathscr{N}}$ = Output from the activation function of neuron $\mathscr{N}$ at layer l
b = Bias of linear ML emulators
c_i = Molar concentration of chemical species i, [mol m^{-3}]
$c_i^0(\mathbf{x})$ = Initial concentration of chemical species i
$c_i^{\mathrm{p}}(\mathbf{x}, t)$ = Prescribed molar concentration, [mol m^{-3}]
$\mathbf{D}(\mathbf{x}, t)$ = Anisotropic dispersion tensor, [m^2 s^{-1}]
D_{m} = Molecular diffusivity, [m^3 s^{-1}]
$\mathbb{E}$ = Expectation
F = MLP activation function
e = Error
f = Function
f_m = mth regressor in AdaB regression
G = Impurity measure in DTs
H = Gini impurity function in DTs
h_m = Base learner/tree
H_ϵ = Truncation value for Huber loss or loss function in ML emulator
$h_i^{\mathrm{p}}(\mathbf{x}, t)$ = Flux, [m s^{-1}]
i = Index
$\mathbf{I}$ = Identity tensor
j = Feature index
k = Class variable index
k_{AB} = Bi-linear reaction rate coefficient, [m^{-1}]
L = Loss function
L_1 = L_1 norm
L_2 = L_2 norm
l = Layer index of neural networks
M = Number of training data
m = Tree node index
N = Gaussian or normal distribution of data
n = Index
n_A = Stoichiometric coefficient for species A
n_B = Stoichiometric coefficient for species B
n_C = Stoichiometric coefficient for species C
n_{left} = Number of training samples to the left
n_{right} = Number of training samples to the right
o = Random coefficient
p = Periodicity
p = Probability, [%]
p_{mk} = Probability at mkth leaf, [%]
Q = Data at tree node m of a decision tree
Q_{left} = Number of trees to the left
Q_{right} = Number of trees to the right
q = Batch or subsample of a dataset
r = Random coefficient
s = Splitting of leaves in a DT
t_m = Threshold at which trees split
u = Random coefficient
$\mathbf{v}$ = Velocity vector field, [m s^{-1}]
v_0 = Perturbation parameter
w = Random coefficient
w = Coefficient or weight
$\mathbb{w}$ = Coefficient or weight vector
w_0 = Intercept of a linear equation
$\mathbb{X}$ = Feature / input matrix
$\mathbb{x}$ = Feature / input vector
y = Label / output

$\mathbb{y}$ = Label / output vector
$\hat{y}$ = Approximation to y
z = Dummy variable

B.2. Greek symbols

α = Penalty/regularization parameter
α_1 = Regularization parameter used to reduce the magnitudes of w through the L_1 norm to reduce the chance for overfitting
α_2 = Regularization parameter used to reduce the magnitudes of w through the L_2 norm to reduce the chance for overfitting
α_{L} = Longitudinal dispersion, [$\mathrm{m}^2\,\mathrm{s}^{-1}$]
α_{T} = Transverse dispersion, [$\mathrm{m}^2\,\mathrm{s}^{-1}$]
β = Regularization parameter
Γ_i^{D} = Dirichlet boundary condition
Γ_i^{N} = Neumann boundary condition
θ_m = Confidence in the prediction for the mth datum
$\kappa_{\mathrm{f}} L$ and T = The characteristic spatial and temporal scales of the flow field, [–]
γ_{dt} = Learning rate for the DT
γ_{dta} = Learning rate for the DT-based AdaB
γ_{m} = Learning rate for the GBM
γ_{mlp} = Learning rate for the MLP
ϵ = Truncation value under which no penalty is associated with the training loss
η = Noise
θ = Confidence function for prediction
λ = Spread of kernel
μ = Mean
ν = Integers
π = $\cos^{-1}(-1)$
Σ = Covariance matrix
$\mathbf{\Phi}$ = Design matrix of $\mathcal{N} \times (\mathcal{N}+1)$ size
ω = A regularization parameter

B.3. Mathematical operators, vectors, tensors, and acronyms

$\otimes$ = The tensor product
$\mathbb{I}$ = Identity matrix
$\mathscr{K}$ = Node number
$\mathcal{K}$ = Kernel
ℓ = Wiggle length in sine function
$\mathcal{N}$ = Number of rows of design matrix
$\mathscr{N}$ = Number of neuron
$\mathfrak{T}$ = Number of regression tree
arg min = Argument of the minimum or smallest value of a function or a set of numbers
det = Determinant of a matrix
div = Divergence operator
grad = Gradient operator
inf = Greatest upper bound
mean = Calculated central value of a set of numbers
sup = Least upper bound

Appendix C. Brief mathematical description of ML emulators

C.1. Linear emulators

C.1.1. Generalized linear emulators

Suppose there are n features x_1 through x_n that correspond to a label y. LSQR calculates the closest $\hat{y}$ by finding the best linear combination of features as:

$$\hat{y}\,(\mathbb{w}, \mathbb{x}) = w_0 + w_1 x_1 + \cdots + w_{\mathrm{n}} x_{\mathrm{n}} = \mathbb{x} \cdot \mathbb{w}. \tag{12}$$

Linear regressors minimize a loss (or cost) function. The loss function in this case is the residual sum of squares between the set of training feature vectors $\mathbb{x}_1, \mathbb{x}_2, \cdots, \mathbb{x}_{\mathrm{n}}$ and predicted targets $y_1, y_2, \cdots, y_{\mathrm{n}}$ of the form:

$$L_{\text{linear}} = \min_{\mathbb{w}} \|\mathbb{X}\mathbb{w} - \mathbb{y}\|_2^2 + \alpha_1 \|\mathbb{w}\|_1 + \alpha_2 \|\mathbb{w}\|_2^2 + \Sigma \sum_{i=1}^{n} \left[1 + H_\epsilon \left(\frac{\mathbb{x}_i \cdot \mathbb{w} - y_i}{\Sigma} \right) \right]. \tag{13}$$

For LSQR, $\alpha_1 = \alpha_2 = \Sigma = 0$, for RR, $\alpha_1 = \Sigma = 0$, for LR, $\alpha_2 = \Sigma = 0$, for ER, $\Sigma = 0$, and for HR, $\alpha_2 = 0$.

$$H_\epsilon(e) = \begin{cases} e^2 & \text{if } e < \epsilon \\ 2\epsilon|e| - \epsilon^2 & \text{otherwise.} \end{cases} \tag{14}$$

The LSQR method minimizes L_{linear} without regularization. RR uses the L_2 norm, which does not use sparsity constraints. However, it includes a penalty, α_2, on weights, which is known as the ridge coefficient. This prevents weights from getting too large, which can lead to overfitting. LR is another linear regressor that penalizes the L_1 norm. The penalty α_1 on the absolute value of weights may result in fewer non-zero coefficients, which may reduce the number of features of the given solution. With increasing α_1, bias increases, but variances decrease and vice versa. ER is another linear regressor that combines the L_1 and L_2 penalties of RR and LR. It is useful for data with multiple, correlated features. LR likely picks one of these correlated features at random, but ER picks all the correlated features. HR is a generalized linear regression method that treats a sample as an inlier if the absolute error of that sample is less than the specified threshold. HR treats a sample as an outlier if the absolute error exceeds the specified threshold. Polynomial regression applies the LSQR formula on quadratically scaled data.

C.1.2. Logistic regression

Despite its name, logistic regression is a classifier; it uses the linear regression scheme to correlate a probability for each class. Logistic regression predicts the outcome in terms of probability and provides a meaningful threshold where distinguishing between classes is possible [72]. Multi-class classification is achieved through either a One-vs-One or a One-vs-Rest strategy [60]. A simple linear ML emulator fails to provide multi-class outputs as probabilities. But logistic regression provides these probabilities through the logistic function. Consider an ML model with two features x_1 and x_2 with one label y, which is classified with probability p. If we assume a linear relationship between predictor variables and the log-odds of the event:

$$\ln \frac{p}{1-p} = w_0 + w_1 x_1 + w_2 x_2. \tag{15}$$

With simple algebraic manipulation, the probability, p, of classifying the predictor variable can be recast as:

$$p = \frac{1}{1 + \mathrm{e}^{-(w_0 + w_1 x_1 + w_2 x_2)}}. \tag{16}$$

Here, the loss function is the cross-entropy loss:

$$L_{\text{cross-entropy}} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(p_i) + (1 - y_i) \log(1 - p_i) \right]. \tag{17}$$

In most cases, ML classifiers use the cross-entropy cost function, $L_{\text{cross-entropy}}$, which ranges between zero and one (zero indicates the occurrence of an event while one indicates that the occurrence of an event is impossible). Leveraging the mathematical properties of $L_{\text{cross-entropy}}$ for classification problems results in faster training with better generalization [73] than other cost functions such as L_{MSE}.

C.1.3. Kernel Ridge (KR) regression

KR regression combines RR with kernel tricks [40] to learn a linear function induced by both the kernel and the data. These kernel tricks enable a linear ML emulator to learn nonlinear functions without explicitly mapping a linear learning algorithm. The kernel function is applied to each label to map the original nonlinear observations into a higher dimensional space. In this work, the stationary radial basis function (RBF) kernel was the optimized kernel. The RBF kernel on two different feature vectors, $\mathbb{x}_1$ and $\mathbb{x}_2$, is:

$$\mathcal{K}_{\text{RBF}}(\mathbb{x}_1, \mathbb{x}_2) = \exp\left(-\lambda \|\mathbb{x}_1 - \mathbb{x}_2\|^2 \right), \tag{18}$$

where λ sets the spread of the kernel. If the kernel is Gaussian, then large λ shrinks the spread of the Gaussian distribution and vice versa. The squared-loss function is used to learn the linear mapping function:

$$L_{\text{squared}} = \left(y - \hat{y} \right)^2. \tag{19}$$

C.2. Bayesian emulators

C.2.1. Bayesian Ridge (BR) regression

Using Bayes' Rule, BR formulates a probabilistic model of the regression problem. BR assumes that labels, $\mathbb{y}$, are normally distributed around $\mathbb{X}\mathbb{w}$ and obtains a probabilistic model by:

$$p\left(\mathbb{y}|\mathbb{X},\mathbb{w},\beta\right)=N\left(\mathbb{y}|\mathbb{X}\mathbb{w},\beta\right), \tag{20}$$

where β is the regularization parameter and N is the Gaussian or normal distribution. The prior for the coefficient vector $\mathbb{w}$ is given by a spherical Gaussian distribution:

$$p\left(\mathbb{w}|\omega\right)=N\left(\mathbb{w}|0,\omega^{-1}\mathbb{I}\right). \tag{21}$$

The β and ω are selected to be conjugate priors (prior and posterior distributions are in the same probability distribution family) and gamma distributions. The parameters β and ω are estimated by maximizing the log-marginal likelihood [41,74] as:

$$L_{\text{lml}}=-\frac{1}{2}\left[\log_{10}\left|\omega^{-1}\mathbb{I}+\mathbf{\Phi}\mathbb{A}^{-1}\mathbf{\Phi}^{\intercal}\right|+\mathbf{\Psi}^{\intercal}\left(\omega^{-1}\mathbb{I}+\mathbf{\Phi}\mathbb{A}^{-1}\mathbf{\Phi}^{\intercal}\right)^{-1}\mathbf{\Psi}\right]+\sum_{i=0}^{n}\left(o\log\beta_i-r\beta_i\right)+u\log\omega-w\omega. \tag{22}$$

Bayesian emulators maximize probability of an estimate. This is the primary reason to use a log-marginal likelihood (L_{lml}) loss function − to maximize the occurrence of a certain event. where $\mathbb{A}=\text{diag}\left(\beta_0,\beta_1,\cdots,\beta_n\right)$, $\mathbf{\Phi}$ is $\mathcal{N}\times\left(\mathcal{N}+1\right)$ "design" matrix, $\mathbf{\Psi}=\left[y\left(\mathbb{x}_1;\mathbb{w}\right)+\eta_1,\cdots,y\left(\mathbb{x}_n;\mathbb{w}\right)+\eta_n\right]^{\intercal}$ is a vector of size $\mathcal{N}\times 1$, and η_i is the noise of ith sample. o, r, u, and w are random parameters.

C.2.2. Gaussian Process (GP)

GPs are generic supervised learning methods for prediction and probabilistic classification that use properties inherited from the normal distribution. GP has the capability of using kernel mathematics, which differentiate GP from BR. GP emulators are not sparse; as a result they are computationally inefficient when developing models in high-dimensional spaces. That is, GP emulators are difficult to implement if features exceed a few dozens [75,76] in scikit-learn and training data are $>\mathcal{O}(10^4)$ (the training sample size in this work was $\mathcal{O}(10^5)$). In this study, the RBF kernel (Eq. (18)) was used to obtain GP emulators by maximizing the first term on the right of Eq. (22) to predict $\hat{\mathbb{y}}$.

C.2.3. Naïve Bayes (NB)

NB emulators are supervised ML methods that also apply Bayes' Theorem with the naïve assumption of conditional independence between every pair of features given the label value [77–80]. NB maximizes $p\left(x_i\,|\,y\right)$ and $p\left(y\right)$ by maximizing the *a posteriori* function [60]. Various naïve Bayes regressions differ by the assumptions they make regarding the distribution of $p\left(x_i\,|\,y\right)$ [81]. Herein, we use the Gaussian-naïve Bayes emulator:

$$p\left(x_i\,|\,y\right)=\frac{1}{\sqrt{2\pi\sigma_y^2}}\exp\left[-\frac{\left(x_i-\mu_y\right)^2}{2\sigma_y^2}\right], \tag{23}$$

where μ_y and σ_y^2 are the mean and variance of regressor/label y. NB updates model parameters such as feature means and variances using different batch sizes, which makes NB computationally efficient [82].

C.2.4. Linear and quadratic discriminant analyses (LDA/QDA)

LDA and QDA are classifiers that use Bayes' Rule. They compute the class conditional distribution of data $p\left(\mathbb{x}|y=k\right)$ for each class k. Based on $p\left(\mathbb{x}|y=k\right)$, for partition $y=q$ of sample space, predictions are made using Bayes' Rule:

$$p\left(y=k|\mathbb{x}\right)=\frac{p\left(\mathbb{x}|y=k\right)p\left(y=k\right)}{p\left(\mathbb{x}\right)}=\frac{p\left(\mathbb{x}|y=k\right)p\left(y=k\right)}{\sum\limits_{q}p\left(\mathbb{x}|y=q\right)p\left(y=q\right)}. \tag{24}$$

Later, class k is selected to maximize the conditional probability. Specifically, $p\left(\mathbb{x}|y\right)$ is modeled using a multivariate Gaussian distribution with density:

$$p\left(y=k|\mathbb{x}\right)=\frac{1}{\left(2\pi\right)^{j/2}\left|\det\left(\sum_k\right)\right|^{1/2}}\exp\left[-\frac{1}{2}\left(\mathbb{x}-\mu_k\mathbb{1}\right)\cdot\left(\sum\nolimits_k\right)^{-1}\left(\mathbb{x}-\mu_k\mathbb{1}\right)\right], \tag{25}$$

where j is the feature number, $\mathbb{1}$ is the one vector. Using training data, it estimates the class priors $p\left(y=k\right)$, class means $\left(\mu_k\right)$, and covariance matrices $\left(\sum_k\right)$ either by the empirical sample class covariance matrices or by a regularized estimator. In LDA, each class shares the same covariance matrix (i.e., $\sum_k=\sum$), which leads to a linear decision surface [39]:

$$\log\left[\frac{p\left(y=k|\mathbb{x}\right)}{p\left(y=q|\mathbb{x}\right)}\right]=\log\left[\frac{p\left(\mathbb{x}|y=k\right)p\left(y=k\right)}{p\left(\mathbb{x}|y=q\right)p\left(y=q\right)}\right]=0 \iff$$
$$\left(\mu_k-\mu_q\right)\mathbb{1}\cdot\sum^{-1}\mathbb{x}=\frac{1}{2}\left(\mu_k\mathbb{1}\cdot\sum^{-1}\mu_k\mathbb{1}-\mu_q\mathbb{1}\cdot\sum^{-1}\mu_q\mathbb{1}\right)-\log\frac{p\left(y=k\right)}{p\left(y=q\right)}. \tag{26}$$

However, QDA does not assume covariance matrices of the Gaussian's, which leads to a quadratic decision surface [39]. Both LDA and QDA use the cross-entropy loss function (Eq. (17)).

C.3. Ensemble ML emulators

C.3.1. Decision Tree (DT)

DT is considered a weak ML classifier and regressor. DT splits leaves in a tree and finds the best or optimal split s^* that increases the purity/accuracy of the resulting tree [31,83–85]. A single tree reduces error in a locally optimal way during feature-space splitting while a regression tree minimizes the residual squared error. For n pairs of training samples, the DT recursively partitions the space to bring the same labels under the same group. Let data at node m be represented by Q. For each candidate split $s=(j,\epsilon_m)$ consisting of feature j and threshold ϵ_m, DT splits data into $Q_{\text{left}}(s)$ and $Q_{\text{right}}(s)$ subsets. For regression, the impurity at m is computed using the Gini impurity function $H\left(\mathbb{x}_m\right)=\frac{1}{\mathfrak{T}_m}\sum\limits_{i\in\mathfrak{T}_m}\left(y_i-\hat{y}_i\right)^2$ using:

$$G\left(Q,s\right)=\frac{n_{\text{left}}}{\mathfrak{T}_m}H\left(Q_{\text{left}}(s)\right)+\frac{n_{\text{right}}}{\mathfrak{T}_m}H\left(Q_{\text{right}}(s)\right), \tag{27}$$

where $\mathfrak{T}_m\leq\text{minimum}_{\text{samples}}$ or $\mathfrak{T}_m=1$. Then, the DT selects the parameters that minimize the impurity:

$$s^*=\arg\min_s G\left(Q,s\right). \tag{28}$$

DT recursively find $Q_{\text{left}}\left(s^*\right)$ and $Q_{\text{right}}\left(s^*\right)$ until the maximum allowable depth is reached, $\mathfrak{T}_m<\text{minimum}_{\text{samples}}$ or $\mathfrak{T}_m=1$. For regression, the loss function, L_{MSE}, is defined as the MSE between the high-fidelity simulations and the results form the ML emulator:

$$L_{\text{MSE}}=\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{y}_i\right)^2. \tag{29}$$

Many ML emulators use L_{MSE} for the cost function during training and cross-validation. Subject to minimizing the MSE, predictions far from their corresponding labels are more aggressively penalized (due to squaring) than estimates close to the expected results.

C.3.2. Bagging emulator

Bagging is a simple ensemble technique that builds many independent tree/predictors and combines them using various model averaging techniques such as a weighted average, majority vote, or arithmetic average. For n pairs of training samples, bagging (bootstrap aggregating) selects M set of samples from n with replacement. Based on each sample, it trains functions $f_1\left(\mathbb{x}_1\right),\ldots,f_M\left(\mathbb{x}_M\right)$. Then, these individual functions or trees are aggregated for regression as:

$$\hat{f}=\sum_{i=1}^{M}f_i(\mathbb{x}_i). \tag{30}$$

The optimized regression criteria, or loss function, to select locations for splits is L_{MSE} (see Eq. (29)).

C.3.3. Random Forest (RF)

RFs are model-free ensemble emulators that provide good accuracy by combining the performance of numerous DTs to classify or predict the value of a variable [83,84]. For given input data (e.g., feature vector $\mathbb{x}$), an RF builds a number of regression trees (M) and averages the results. For each tree, $\mathfrak{T}_m(\mathbb{x})$, for all $m=1,2,\cdots,M$, the RF prediction is:

$$\hat{f}_{\text{RF}}^{M}=\frac{1}{M}\sum_{m=1}^{M}\mathfrak{T}\left(\mathbb{x}\right). \tag{31}$$

For classification, the Gini impurity function is used for the loss function and for k class variables, the Gini impurity is:

$$H\left(\mathbb{x}_m\right)=\sum_k p_{mk}\left(1-p_{mk}\right), \tag{32}$$

where p_{mk} is the probability for the mth tree of the kth variable. For regression, L_{MSE} (Eq. (29)) is used for the loss function.

C.3.4. AdaBoost (AdaB)

AdaB (Adaptive Boosting) converts weak learners into a strong learners [43,86–89]. Weak learners are DTs with a single split that are also known as decision stumps. AdaB is a forward stage-wise additive model (adding up multiple models to create a composite model) with an exponential loss function that iteratively fits a weak classifier to improve the current estimator. AdaB puts more weight on difficult-to-learn labels and less on others. AdaB construct a tree regressor, f_m, from training data so that $f_m : \mathbb{x} \rightarrow y$. Every pair of training data is passed through f_m. Then, f_m calculates a loss for each training datum using the square-loss function:

$$L_i = \frac{\left(\hat{y}_i - y_i\right)^2}{J^2}, \tag{33}$$

where $J = \sup\left|\hat{y}_i - y_i\right|$ and sup is supremum or the least upper bound. Then, L_i is averaged by $\hat{L} = \sum_{i=1}^{n} L_i p_i$ to measure confidence in the prediction as:

$$\theta = \frac{\hat{L}}{1 - \hat{L}}, \tag{34}$$

where low θ indicates high confidence in the prediction. The resulting θ is used to update weights: $w_i \rightarrow w_i \theta \exp\left(1 - L_i\right)$. For $\mathbb{x}_i$, each of M trees/regressors makes a prediction, h_m, $m = 1, \cdots, \mathfrak{T}$, to form a cumulative function:

$$f = \inf\left[y \in \mathbb{y} : \sum_{m:h_m \leq y} \log\left(\frac{1}{\theta_m}\right) \geq \frac{1}{2} \sum_{m} \log\left(\frac{1}{\theta_m}\right) \right]. \tag{35}$$

DT-based AdaB is a heterogeneous emulator that applies both DT and boosting base estimators to learn a prediction function.

C.3.5. Gradient Boosting Method (GBM)

GBM learns function like AdaB, but it generalizes the model by allowing optimization of an arbitrary differentiable loss function. GBM builds learning function f for M trees as:

$$f = \sum_{m=1}^{M} \gamma_m h_m\left(\mathbb{x}_m\right), \tag{36}$$

where h_m and γ_m are the weak learner and the learning rate for mth weak learner. f is a weighted combination of M weak learners. After learning each weak model, the additive model (f_m) is built in a greedy fashion (choose locally optimal solution at each iteration):

$$f_m = f_{m-1} + \gamma_m h_m, \tag{37}$$

where the newly added tree minimizes the least-squared function, L_{LSQR}, for previous model f_{m-1}. The new learner is:

$$h_m = \arg\min_{h} \sum_{1=1}^{n} L_{\text{LSQR}}\left[y_i, f_{m-1}\left(\mathbb{x}_i\right) + h\left(\mathbb{x}_i\right)\right]. \tag{38}$$

GBM minimizes L_{LSQR} (optimal loss function for this work) using the steepest descent where the steepest descent direction is the negative gradient of L_{LSQR} determined at f_{m-1}. The steepest gradient direction and rate is calculated as:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=n}^{n} L_{\text{LSQR}} \left\{ y_i, f_{m-1}\left(\mathbb{x}_i\right) - \gamma \frac{\partial L_{\text{LSQR}}\left[y_i, f_{m-1}\left(\mathbb{x}_i\right)\right]}{\partial f_{m-1}\left(\mathbb{x}_i\right)} \right\}. \tag{39}$$

C.4. Multi-Layer Perceptron (MLP)

An MLP is a supervised ML method for classification and prediction. MLPs are feed-forward neural networks [59] that consist of numerous, simple, connected computation elements called neurons arranged in layers. Neuron output is calculated as the result of a nonlinear activation function whose input is the sum of weighted inputs from all neurons in the preceding layer. The output from neuron n in layer l is:

$$a_{\mathscr{N}}^{(l)} = F\left(\sum_{\mathscr{K}=1}^{\mathscr{N}_{l-1}} w_{\mathscr{K},\mathscr{N}}^{(l)} a_{\mathscr{K}}^{(l-1)} + b_{\mathscr{N}}^{(l)} \right), \tag{40}$$

where F is the activation function, $\mathscr{N}_{l-1}$ is the number of neurons in layer $l-1$, $w^{(l)}_{\mathscr{K},\mathscr{N}}$ is the weight projecting from node $\mathscr{K}$ in layer $l-1$ to node $\mathscr{N}$ in layer l, $a^{(l-1)}_{\mathscr{K}}$ is the activation of neuron $\mathscr{K}$ in hidden layer $l-1$, and $b^{(l)}_{\mathscr{N}}$ is the bias added to hidden layer l contributing to subsequent layer. The rectified linear unit (ReLU) [90] outperformed hyperbolic tangent and logistic functions in this application:

$$F(z) = \max(0, z), \tag{41}$$

where z is the dummy variable or sum of weighted upstream data [91]. The MSE (see Equation (29)) and cross-entropy function (see Equation (17)) are appropriate loss functions for regression and classification, respectively, but any loss function can be used as dictated by the problem.

References

[1] V. Lagneau, O. Regnault, M. Descostes, Industrial deployment of reactive transport simulation: an application to uranium in situ recovery, Rev. Mineral. Geochem. 85 (2019) 499–528, 09.
[2] J. Cama, J.M. Soler, C. Ayora, Acid water-rock-cement interaction and multicomponent reactive transport modeling, Rev. Mineral. Geochem. 85 (09 2019) 459–498.
[3] M. Rolle, T. Le Borgne, Mixing and reactive fronts in the subsurface, Rev. Mineral. Geochem. 85 (09 2019) 111–142.
[4] I. Sin, J. Corvisier, Multiphase multicomponent reactive transport and flow modeling, Rev. Mineral. Geochem. 85 (09 2019) 143–195.
[5] S. Molins, P. Knabner, Multiscale approaches in reactive transport modeling, Rev. Mineral. Geochem. 85 (09 2019) 27–48.
[6] P.C. Lichtner, C.I. Steefel, E.H. Oelkers, Reactive Transport in Porous Media, vol. 34, Walter de Gruyter GmbH & Co KG, 2019.
[7] P.C. Lichtner, G.E. Hammond, C. Lu, S. Karra, G. Bisht, B. Andre, R.T. Mills, J. Kumar, PFLOTRAN user manual: a massively parallel reactive flow and transport model for describing surface and subsurface processes, Technical report (Report No.: LA-UR-15-20403), Los Alamos National Laboratory, 2015.
[8] L. Chen, M. Wang, Q. Kang, W. Tao, Pore-scale study of multiphase multicomponent reactive transport during CO_2 dissolution trapping, Adv. Water Resour. 116 (2018) 208–218.
[9] M.A. Öztürk, M. Ashraf, A. Aksoy, M.S.A. Ahmad, K.R. Hakeem, Plants, Pollutants and Remediation, Springer, 2015.
[10] B. Ahmmed, Numerical modeling of CO_2-water-rock interactions in the Farnsworth, Texas hydrocarbon unit, USA, Master's thesis, University of Missouri, 2015.
[11] V.V. Vesselinov, M.K. Mudunuru, S. Karra, D. O'Malley, B.S. Alexandrov, Unsupervised machine learning based on non-negative tensor factorization for analyzing reactive mixing, J. Comput. Phys. 395 (2019) 85–104.
[12] M.K. Mudunuru, S. Karra, Physics-informed machine learning models for predicting the progress of reactive mixing, arXiv preprint arXiv:1908.10929v1, 2019.
[13] Y. Wu, Y. Lin, Z. Zhou, A. Delorey, Seismic-net: a deep densely connected neural network to detect seismic events, arXiv preprint arXiv:1802.02241, 2018.
[14] C. Hulbert, B. Rouet-Leduc, P.A. Johnson, C.X. Ren, J. Riviére, D.C. Bolton, C. Marone, Similarity of fast and slow earthquakes illuminated by machine learning, Nat. Geosci. 12 (2019) 69.
[15] H.S. Viswanathan, J.D. Hyman, S. Karra, D. O'Malley, S. Srinivasan, A. Hagberg, G. Srinivasan, Advancing graph-based algorithms for predicting flow and transport in fractured rock, Water Resour. Res. 54 (2018) 6085–6099.
[16] S. Srinivasan, J. Hyman, S. Karra, D. O'Malley, H.S. Viswanathan, G. Srinivasan, Robust system size reduction of discrete fracture networks: a multi-fidelity method that preserves transport characteristics, Comput. Geosci. 22 (2018) 1515–1526.
[17] G. Camps-Valls, L. Martino, D.H. Svendsen, M. Campos-Taberner, J. Muñoz-Marí, V. Laparra, D. Luengo, F.J. García-Haro, Physics-aware gaussian processes in remote sensing, Appl. Soft Comput. 68 (2018) 69–82.
[18] K.J. Bergen, P.A. Johnson, V. Maarten, G.C. Beroza, Machine learning for data-driven discovery in solid Earth geoscience, Science 363 (2019) eaau0323.
[19] M. Reichstein, G. Camps-Valls, B. Stevens, M. Jung, J. Denzler, N. Carvalhais, Prabhat, Deep learning and process understanding for data-driven Earth system science, Nature 566 (2019) 195.
[20] C. R.-Mesa, M. Reichstein, M. Mahecha, B. Kraft, J. Denzler, Predicting landscapes as seen from space from environmental conditions, in: IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2018, pp. 1768–1771.
[21] S.C. James, Y. Zhang, F. O'Donncha, A machine learning framework to forecast wave conditions, Coast. Eng. 137 (2018) 1–10.
[22] F. O'Donncha, Y. Zhang, B. Chen, S.C. James, Ensemble model aggregation using a computationally lightweight machine-learning model to forecast ocean waves, J. Mar. Syst. 199 (2019) 103206.
[23] F. O'Donncha, Y. Zhang, B. Chen, S.C. James, An integrated framework that combines machine learning and numerical models to improve wave-condition forecasts, J. Mar. Syst. 186 (2018) 29–36.
[24] B. R.-Leduc, C. Hulbert, N. Lubbers, K. Barros, C.J. Humphreys, P.A. Johnson, Machine learning predicts laboratory earthquakes, Geophys. Res. Lett. 44 (2017) 9276–9282.
[25] A. Reynen, P. Audet, Supervised machine learning on a network scale: application to seismic event classification and detection, Geophys. J. Int. 210 (2017) 1394–1409.
[26] S.A. M.-Zook, S.D. Ruppert, Explosion monitoring with machine learning: a LSTM approach to seismic event discrimination, in: AGU Fall Meeting Abstracts, 2017.
[27] B. Yuan, Y.J. Tan, M.K. Mudunuru, O.E. Marcillo, A.A. Delorey, P.M. Roberts, J.D. Webster, C.N.L. Gammans, S. Karra, G.D. Guthrie, P.A. Johnson, Using machine learning to discern eruption in noisy environments: a case study using CO_2-driven cold-water geyser in Chimayó, New Mexico, Seismol. Res. Lett. 90 (2019) 591–603.
[28] R. Barzegar, A.A. Moghaddam, R. Deo, E. Fijani, E. Tziritis, Mapping groundwater contamination risk of multiple aquifers using multi-model ensemble of machine learning algorithms, Sci. Total Environ. 621 (2018) 697–712.
[29] C. Ferreira, Gene expression programming: a new adaptive algorithm for solving problems, arXiv preprint, arXiv:cs/0102027, 2001.
[30] C. Ferreira, Designing neural networks using gene expression programming, in: Applied Soft Computing Technologies: the Challenge of Complexity, Springer, 2006, pp. 517–535.
[31] V. R.-Galiano, M. S.-Castillo, M. C.-Olmo, M. C.-Rivas, Machine learning predictive models for mineral prospectivity: an evaluation of neural networks, random forest, regression trees and support vector machines, Ore Geol. Rev. 71 (2015) 804–818.
[32] C. Kirkwood, M. Cave, D. Beamish, S. Grebby, A. Ferreira, A machine learning approach to geochemical mapping, J. Geochem. Explor. 167 (2016) 49–61.
[33] R. Zuo, Machine learning of mineralization-related geochemical anomalies: a review of potential methods, Nat. Resour. Res. 26 (10 2017) 457–464.
[34] S. Oonk, J. Spijker, A supervised machine-learning approach towards geochemical predictive modelling in archaeology, J. Archaeol. Sci. 59 (2015) 80–88.

[35] M.J. Cracknell, A.M. Reading, A.W. McNeill, Mapping geology and volcanic-hosted massive sulfide alteration in the Hellyer–mt charter region, Tasmania, using random forests and self-organising maps, Aust. J. Earth Sci. 61 (2014) 287–304.
[36] M.K. Salah, Machine Learning for Model Order Reduction, Springer, 2018.
[37] S.L. Brunton, J.N. Kutz, Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, Cambridge University Press, 2019.
[38] D.W. Marquaridt, Generalized inverses, ridge regression, biased linear estimation, and nonlinear estimation, Technometrics 12 (1970) 591–612.
[39] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Springer Series in Statistics, Springer, New York, 2009.
[40] K.P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.
[41] M.E. Tipping, Sparse bayesian learning and the relevance vector machine, J. Mach. Learn. Res. 1 (2001) 211–244.
[42] L. Breiman, Bagging predictors, Mach. Learn. 24 (1996) 123–140.
[43] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1997) 119–139.
[44] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Cognitive modeling, 5:1, 1988.
[45] G. Montavon, W. Samek, K. Müller, Methods for interpreting and understanding deep neural networks, Digit. Signal Process. 73 (2018) 1–15.
[46] K.B. Nakshatrala, M.K. Mudunuru, A.J. Valocchi, A numerical framework for diffusion-controlled bimolecular-reactive systems to enforce maximum principles and the non-negative constraint, J. Comput. Phys. 253 (2013) 278–307.
[47] M.K. Mudunuru, M. Shabouei, K.B. Nakshatrala, On local and global species conservation errors for nonlinear ecological models and chemical reacting flows, in: Proceedings of ASME 2015 International Mechanical Engineering Congress and Exposition, 2015, V009T12A018.
[48] M.K. Mudunuru, K.B. Nakshatrala, A framework for coupled deformation-diffusion analysis with application to degradation/healing, Int. J. Numer. Methods Eng. 89 (2012) 1144–1170.
[49] M.K. Mudunuru, K.B. Nakshatrala, On mesh restrictions to satisfy comparison principles, maximum principles, and the non-negative constraint: recent developments and new results, Mech. Adv. Mat. Struct. 24 (2017) 556–590.
[50] G.F. Pinder, M.A. Celia, Subsurface Hydrology, John Wiley & Sons, Inc., New Jersey, USA, 2006.
[51] A. Adrover, S. Cerbelli, M. Giona, A spectral approach to reaction/diffusion kinetics in chaotic flows, Comput. Chem. Eng. 26 (2002) 125–139.
[52] Y.K. Tsang, Predicting the evolution of fast chemical reactions in chaotic flows, Phys. Rev. E 80 (2009) 026305.
[53] M.K. Mudunuru, K.B. Nakshatrala, On enforcing maximum principles and achieving element-wise species balance for advection–diffusion–reaction equations under the finite element method, J. Comput. Phys. 305 (2016) 448–493.
[54] M. Dentz, T. Le Borgne, A. Englert, B. Bijeljic, Mixing, spreading and reaction in heterogeneous media: a brief review, J. Contam. Hydrol. 120 (2011) 1–17.
[55] P.K. Kitanidis, P.L. McCarty, Delivery and Mixing in the Subsurface: Processes and Design Principles for in Situ Remediation, vol. 4, Springer Science & Business Media, 2012.
[56] K.B. Nakshatrala, M.K. Mudunuru, A.J. Valocchi, A numerical framework for diffusion-controlled bimolecular-reactive systems to enforce maximum principles and non-negative constraint, J. Comput. Phys. 253 (2013) 278–307.
[57] M.K. Mudunuru, K.B. Nakshatrala, On enforcing maximum principles and achieving element-wise species balance for advection-diffusion-reaction equations under the finite element method, J. Comput. Phys. 305 (2016) 448–493.
[58] M.K. Mudunuru, K.B. Nakshatrala, On mesh restrictions to satisfy comparison principles, maximum principles, and the non-negative constraint: recent developments and new results, Mech. Adv. Mat. Struct. 24 (2017) 556–590.
[59] A.C. Müller, S. Guido, Introduction to Machine Learning with Python: A Guide for Data Scientists, O'Reilly Media, Inc., 2016.
[60] 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.
[61] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, B. Holt, G. Varoquaux, API design for machine learning software: experiences from the scikit-learn project, in: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, 2013, pp. 108–122.
[62] 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.
[63] R. Kohavi, A study of cross-validation and bootstrap for accuracy estimation and model selection, in: IJCAI, vol. 14, 1995, pp. 1137–1145.
[64] J.S. Chou, C.F. Tsai, A.D. Pham, Y.H. Lu, Machine learning in concrete strength simulations: multi-nation data analytics, Constr. Build. Mater. 73 (2014) 771–780.
[65] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, K. Li, A parallel random forest algorithm for big data in a spark cloud computing environment, IEEE Trans. Parallel Distrib. Syst. 28 (4) (2016) 919–933.
[66] V. Turchenko, Parallel batch pattern training algorithm for mlp with two hidden layers on many-core system, in: Distributed Computing and Artificial Intelligence, 11th International Conference, Springer International Publishing, Cham, 2014, pp. 537–544.
[67] B. Van Essen, C. Macaraeg, M. Gokhale, R. Prenger, Accelerating a random forest classifier: multi-core, gp-gpu, or fpga?, in: 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines, IEEE, 2012, pp. 232–239.
[68] K. Oh, K. Jung, Gpu implementation of neural networks, Pattern Recognit. 37 (6) (2004) 1311–1314.
[69] H. Iacovides, B.E. Launder, Turbulent momentum and heat transport in square-sectioned ducts rotating in orthogonal mode, Numer. Heat Transf., Part A, Appl. 12 (4) (1987) 475–491.
[70] Y. Lu, H. Jiang, S. Guo, T. Wang, Z. Cao, T. Li, A new strategy to design eutectic high-entropy alloys using mixing enthalpy, Intermetallics 91 (2017) 124–128.
[71] G.P. Merker, C. Schwarz, G. Stiesch, F. Otto, Simulating Combustion: Simulation of Combustion and Pollutant Formation for Engine-Development, Springer Science & Business Media, 2005.
[72] C. Molnar, Interpretable Machine Learning: A Guide for Making Black Box Models Explainable, 2019.
[73] C.M. Bishop, Pattern Recognition and Machine Learning, Springer, 2006.
[74] D.J.C. MacKay, Bayesian interpolation, Neural Comput. 4 (1992) 415–447.
[75] C.E. Rasmussen, Gaussian processes in machine learning, in: Summer School on Machine Learning, Springer, 2003, pp. 63–71.
[76] T.J. Santner, B.J. Williams, W. Notz, The Design and Analysis of Computer Experiments, vol. 1, Springer, 2018.
[77] J.D. Rennie, L. Shih, J. Teevan, D.R. Karger, Tackling the poor assumptions of naïve Bayes text classifiers, in: Proceedings of the 20th International Conference on Machine Learning, ICML-03, 2003, pp. 616–623.
[78] C. Manning, P. Raghavan, H. Schütze, Introduction to information retrieval, Nat. Lang. Eng. 16 (2010) 100–103.
[79] A. McCallum, K. Nigam, A comparison of event models for naïve Bayes text classification, in: AAAI-98 Workshop on Learning for Text Categorization, vol. 752, Citeseer, 1998, pp. 41–48.
[80] V. Metsis, I. Androutsopoulos, G. Paliouras, Spam filtering with naïve Bayes-which naïve Bayes?, in: CEAS, vol. 17, Mountain View, CA, 2006, pp. 28–69.
[81] H. Zhang, The optimality of naïve Bayes, AA, 1:3, 2004.
[82] F.C. Tony, H.G. Gene, J.L. Randall, Algorithms for computing the sample variance: analysis and recommendations, Am. Stat. 37 (3) (1983) 242–247.
[83] L. Breiman, Random forests, Mach. Learn. 45 (2001) 5–32.

[84] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Taylor & Francis Group, New York, 2017.
[85] P. Geurts, D. Ernst, L. Wehenkel, Extremely randomized trees, Mach. Learn. 63 (2006) 3–42.
[86] Y. Freund, Boosting a weak learning algorithm by majority, Inf. Comput. 121 (1995) 256–285.
[87] Y. Freund, E.S. Robert, Experiments with a New Boosting Algorithm, in: ICML, vol. 96, Citeseer, 1996, pp. 148–156.
[88] Y. Freund, R.E. Schapire, Game theory, on-line prediction and boosting, in: COLT, vol. 96, Citeseer, 1996, pp. 325–332.
[89] R.E. Schapire, Y. Freund, A decision-theoretic generalization of on-line learning and an application to boosting, in: Second European Conference on Computational Learning Theory, 1995, pp. 23–37.
[90] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning, ICML-10, 2010, pp. 807–814.
[91] S. Garavaglia, A. Sharma, A smart guide to dummy variables: four applications and a macro, in: Proceedings of the Northeast SAS Users Group Conference, vol. 43, 1998.