INL/JOU-25-83717-Revision-0

MOOSE ProbML: Parallelized Probabilistic Machine Learning and Uncertainty Quantification for Computational Energy Applications

December 2025

Som LakshmiNarasimha Dhulipala, Peter German, Yifeng Che, Zachary M Prince, Xianjian Xie, Pierre-Clement A Simon, Hao Yan

INL/JOU-25-83717-Revision-0

MOOSE ProbML: Parallelized Probabilistic Machine Learning and Uncertainty Quantification for Computational Energy Applications

Som LakshmiNarasimha Dhulipala, Peter German, Yifeng Che, Zachary M Prince, Xianjian Xie, Pierre-Clement A Simon, Hao Yan

December 2025

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517, DE-AC07-05ID14517, DE-AC07-05ID14517, DE-AC07-05ID14517, DE-AC07-05ID14517, DE-AC07-05ID14517

MOOSE ProbML: Parallelized Probabilistic Machine Learning and Uncertainty Quantification for Computational Energy Applications

Somayajulu L. N. Dhulipala[a,*], Peter German[b], Yifeng Che[c], Zachary M. Prince[b], Xianjian Xie[d], Pierre-Clément A. Simon[a], Vincent M. Labouré[e], Hao Yan[d]

[a] *Computational Mechanics and Materials Department, Idaho National Laboratory, Idaho Falls, 83415, ID, USA*
[b] *Computational Frameworks Department, Idaho National Laboratory, Idaho Falls, 83415, ID, USA*
[c] *Woodruff School of Mechanical Engineering, Georgia Institute of Technology, Atlanta, GA, 30332, USA*
[d] *School of Computing and Augmented Intelligence, Arizona State University, Tempe, AZ, 85287, USA*
[e] *Reactor Physics Methods and Analysis Department, Idaho National Laboratory, Idaho Falls, 83415, ID, USA*

Abstract

This paper presents the development and demonstration of massively parallel probabilistic machine learning (ML) and uncertainty quantification (UQ) capabilities within the Multiphysics Object-Oriented Simulation Environment (MOOSE), an open-source computational platform for parallel finite element and finite volume analyses. In addressing the computational expense and uncertainties inherent in complex multiphysics simulations, this paper integrates Gaussian process (GP) variants, active learning, Bayesian inverse UQ, adaptive forward UQ, Bayesian optimization, evolutionary optimization, and Markov chain Monte Carlo (MCMC) within MOOSE. It also elaborates on the interaction among key MOOSE systems—`Sampler`, `MultiApp`, `Reporter`, and `Surrogate`—in enabling these capabilities. The modularity offered by these systems enables development of a multitude of probabilistic ML and UQ algorithms in MOOSE. Example code demonstrations include parallel active learning and parallel Bayesian inference via active learning. The impact of these developments is illustrated through five applications relevant to computational energy applications: UQ of nuclear fuel fission product release, using parallel active learning Bayesian inference; very rare events analysis in nuclear microreactors using active learning; advanced manufacturing process modeling using multi-output GPs (MOGPs) and dimensionality reduction; fluid flow using deep GPs (DGPs); and tritium transport model parameter optimization for fusion energy, using batch Bayesian optimization. These capabilities are part of the MOOSE framework.

Keywords: Active learning, Gaussian processes, Bayesian inference, Bayesian optimization, Finite element models, Nuclear fission and fusion energy

1. Introduction

The Multiphysics Object-Oriented Simulation Environment (MOOSE), an open-source computational platform for parallel finite element and finite volume analyses, is being developed and maintained primarily at Idaho National Laboratory, and has a wide user and developer base spanning academia, industry, and national laboratories [1]. It is easy to install, offers extensive tutorials, comes with built-in physics modules, and naturally lends itself to multiscale and multiphysics simulations. MOOSE supports a vibrant community of computational scientists and engineers via a highly active discussions forum, and its code base receives tens of pull requests each month (https://github.com/idaholab/moose). MOOSE has traditionally supported computational simulations intended to advance energy solutions such as nuclear fission energy, geothermal energy, and, more recently, nuclear fusion energy. Several applications were built by

*Corresponding author
Email address: `Som.Dhulipala@inl.gov` (Somayajulu L. N. Dhulipala)

using MOOSE to tackle specific problems such as nuclear fuel performance (BISON [2]), structural materials aging (Grizzly [3]), medium-fidelity thermal hydraulics (Pronghorn [4]), radiation transport (Griffin [5]), seismic analysis (Mastodon [6]), mesoscale materials simulations (Marmot [7]), high-fidelity thermal-hydraulics and/or radiation transport (Cardinal [8]), tritium transport for fusion energy (TMAP8 [9]), thermal-hydraulic-mechanical-chemical processes in geothermal systems (Falcon [10]), etc. MOOSE also provides a stochastic tools module to support uncertainty quantification (UQ) and propagation, as well as surrogate model development for multiphysics simulations [11]. This paper presents the development and demonstration of massively parallel probabilistic machine learning (ML) and UQ in the MOOSE stochastic tools module to support capabilities such as Gaussian process (GP) ML, active learning, Bayesian inference, rare events analysis, Bayesian optimization, and evolutionary optimization. These capabilities in the native MOOSE framework are motivated by the following: (1) complex multiphysics simulations, when validated with experimental data, are subject to different sources of uncertainties (i.e., model parameters, model inadequacy, and experimental noise) that must be quantified and propagated to the outputs; (2) complex multiphysics models are computationally expensive to run, especially in a UQ setting, and surrogate models that quantify their prediction uncertainties (i.e., probabilistic ML models such as GPs) will support their efficient and accurate execution by leveraging active learning principles; and (3) probabilistic ML and UQ capabilities could be leveraged by MOOSE's extensive user base.

Probabilistic ML deals with the development of surrogate models that can quantify complex multiphysics model prediction uncertainties. UQ deals with all aspects of identifying and inversely quantifying different sources of uncertainties, then forward propagating them to the model predictions. Probabilistic ML and UQ go hand-in-hand, leading to efficient approaches for active learning, Bayesian inference, Bayesian optimization, etc. Among the existing software for performing various aspects of probabilistic ML and UQ are UQPy [12], CUQIPy [13], MUQ [14], and PyApprox [15], as discussed in Seelinger et al. [16]. Most of these software programs were written in Python. The development and demonstration of probabilistic ML and UQ capabilities presented herein is oriented toward the extensive user/developer community of MOOSE, which is written in C++. Moreover, MOOSE inherently supports massive parallelism, meaning that the probabilistic ML and UQ approaches can be scaled to use thousands of processors, thus leading to high levels of efficiency when dealing with complex multiphysics models. Ultimately, the right software tools can significantly enhance various stages of the research, development, and deployment processes for energy solutions, with different tools being better suited to specific scenarios.

Massively parallel probabilistic ML and UQ in MOOSE is achieved through its `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` systems. `Sampler` proposes new input parameter samples from the underlying probability distributions, `MultiApp` facilitates evaluation of the MOOSE computational model while handling massive parallelism, `Reporter` facilitates post-model-evaluation decision making, and `Surrogate` handles the training, evaluation, and retraining of probabilistic surrogates. These systems and their interaction are key to the development of GP variants, active learning, Bayesian inverse UQ, adaptive forward UQ, Bayesian optimization, evolutionary optimization, and Markov chain Monte Carlo (MCMC) in MOOSE. The modularity offered by these systems enables development of a multitude of probabilistic ML and UQ algorithms. These aspects will be discussed in detail later in this paper. Besides discussing the software implementation, this paper also demonstrates its application to five different types of computational problems: (1) Bayesian inverse UQ of fission product release from nuclear fuel, using parallel active learning; (2) very rare events analysis of a heat pipe (HP) nuclear microreactor, using active learning; (3) acceleration of advanced manufacturing process simulations, using multi-output GPs (MOGPs) and dimensionality reduction; (4) prediction of lid-driven cavity flow, using with deep GPs (DGPs); and (5) model parameter optimization of tritium diffusion for nuclear fusion, using batch Bayesian optimization. Figure 1 presents an overview of the probabilistic ML, forward/inverse UQ, active learning, optimization, and dimensionality reduction capabilities in MOOSE and the core MOOSE systems that are utilized for achieving these capabilities.

This paper is organized as follows. Section 2 provides a theoretical review of the active learning, Bayesian inverse UQ, adaptive forward UQ, Bayesian optimization, evolutionary optimization, and MCMC methods relevant to MOOSE. Section 3 details the MOOSE code implementations. Section 4 discusses the impact to the five aforementioned energy applications. Lastly, Section 6 summarizes the paper and presents the conclusions.

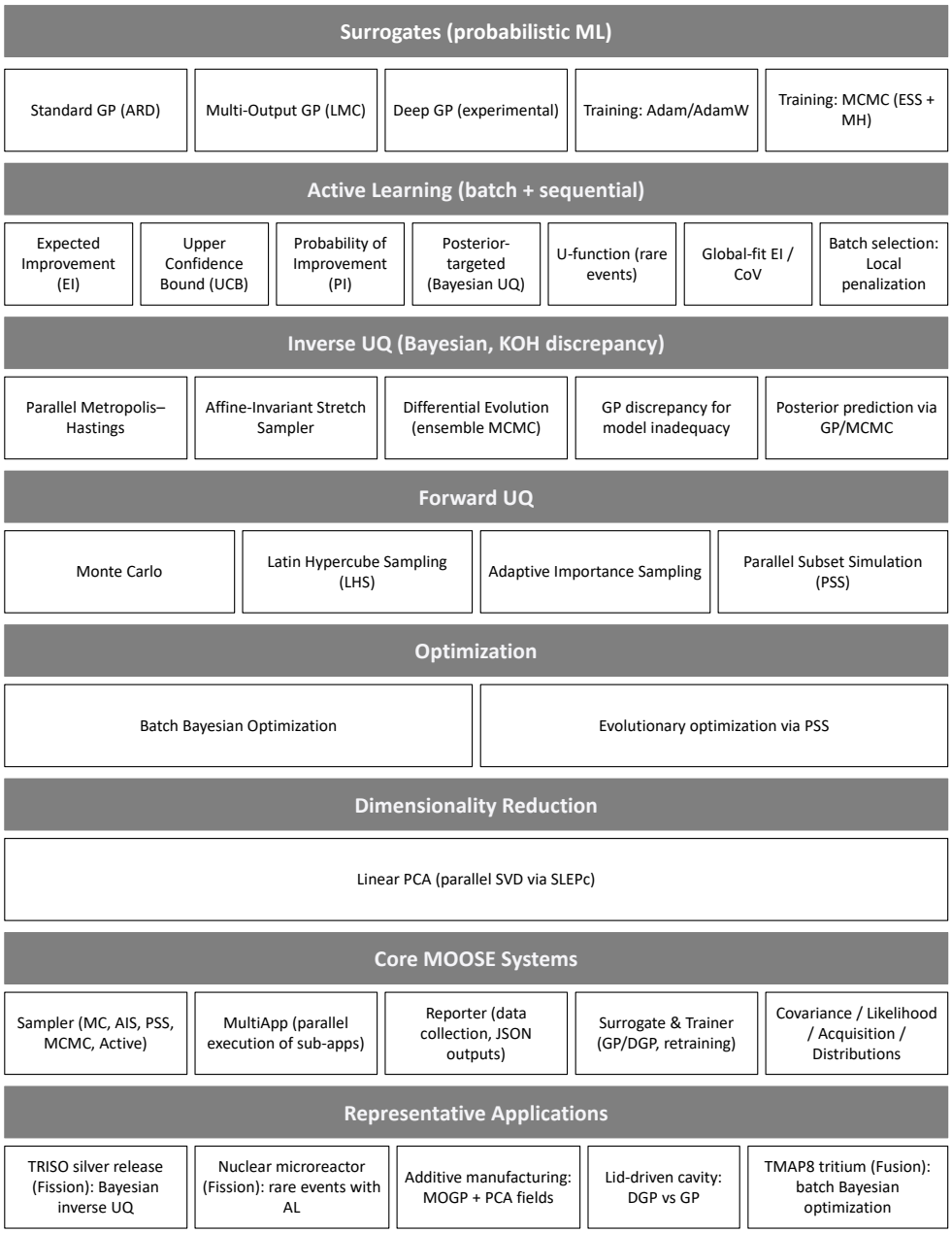


Figure 1: An overview of the probabilistic ML, forward/inverse UQ, active learning, optimization, and dimensionality reduction capabilities in MOOSE and the core MOOSE systems that are utilized for achieving these capabilities. These capabilities are part of the MOOSE framework.

2. Methodology Overview

This section provides a theoretical overview of the probabilistic ML and UQ methods relevant to the MOOSE implementation.

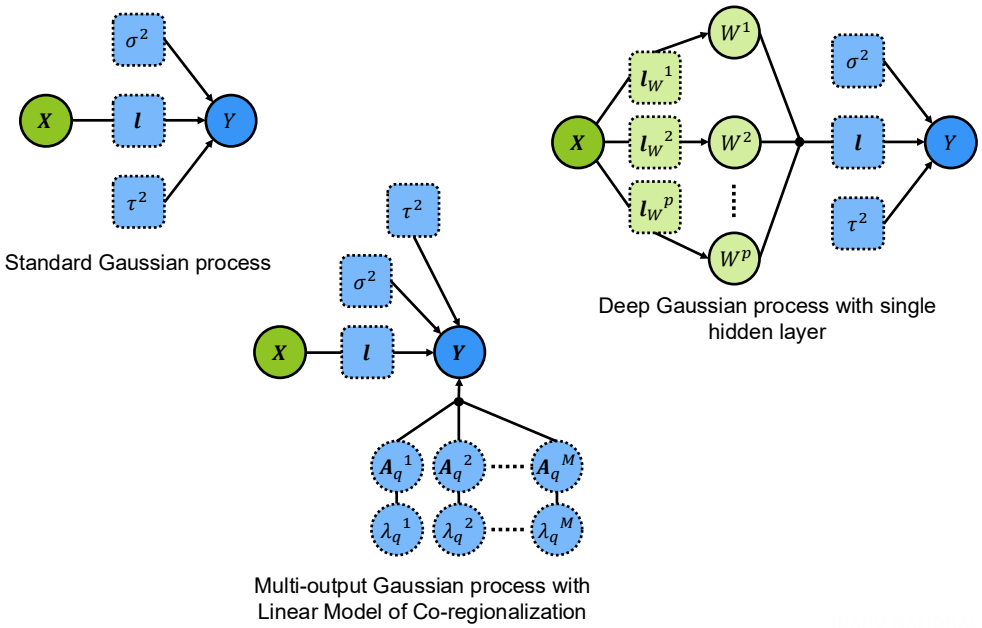


Figure 2: Graphical representation of the input ($\boldsymbol{X}$) and output ($\boldsymbol{Y}$) mapping of the three GP variants in MOOSE: standard GP, MOGP, and DGP. σ^2, $\boldsymbol{l}$, and τ^2, respectively, represent the amplitude scale, length scales, and noise variance hyperparameters. $\boldsymbol{A}_q^i$ and λ_q^i are the additional hyperparameters for an MOGP, and $\boldsymbol{l}_W^i$ is the additional hyperparameters for a DGP. In MOOSE, these GP variants can be trained via either adaptive moment estimation (Adam) optimization (gradient-based) or MCMC sampling (gradient-free). Here, "gradients" refers to gradients of the log-likelihood objective function.

2.1. Gaussian process variants

Figure 2 presents a graphical representation of the different GP variants in MOOSE. The theoretical details are briefly discussed below. The GP capabilities are used for Bayesian analysis of fission product release in an advanced nuclear fuel (Section 4.1), rare events analysis of a nuclear reactor (Section 4.2), advanced manufacturing process modeling (Section 4.3), predicting fluid flow (Section 4.4), and the optimization of a computational model for nuclear fusion (Section 4.5), as discussed later in this paper.

2.1.1. Standard Gaussian process

A standard GP is a stochastic process in which any finite collection of random variables follows a Gaussian distribution. Essentially, a GP describes a probability distribution over a function space and is discretized at certain points in the input space. A zero-mean GP is described as [17]:

$$\boldsymbol{y} \sim \mathcal{N}\Big(\boldsymbol{0},\ k(\boldsymbol{X},\boldsymbol{X}')\Big) \tag{1}$$

where $\boldsymbol{y}$ is the output vector of size N, $k(.,.)$ is the covariance function, and $\boldsymbol{X}$ is the input matrix of size $N \times D$ (D being the dimensionality of the inputs). As shown in Figure 2, given input vectors $\boldsymbol{x}$ and $\boldsymbol{x}'$, the scalar kernel function is described as:

$$k(\boldsymbol{x},\boldsymbol{x}') = \sigma^2\ \exp\left(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x_d - x_d')^2}{l_d^2} + \tau^2 \mathbb{1}_{\boldsymbol{x}=\boldsymbol{x}'}\right) \tag{2}$$

where $\boldsymbol{l} = (l_1, \cdots l_D)$ is the vector of length scales, σ^2 is the amplitude, and τ^2 is the noise term. When each input dimension is associated with its own length scale, the GP fitting procedure is referred to as automatic relevance determination (ARD) [17], which is often used to implicitly determine the relevance of input variables. Note that $\boldsymbol{x}$ is an input vector and $\boldsymbol{X}$ is the input matrix at N points. As such, $k(\boldsymbol{x}, \boldsymbol{x'})$ is a scalar kernel function and $k(\boldsymbol{X}, \boldsymbol{X'})$ is a covariance matrix of size $N \times N$. The parameters $\{\boldsymbol{l}, \sigma^2, \tau^2\}$ are the hyperparameters to be optimized by maximizing the log-likelihood function:

$$\ln\ p(\boldsymbol{y} \mid \boldsymbol{X}, \sigma^2, \boldsymbol{l}, \tau^2) \propto -\frac{1}{2}\ \ln|k(\boldsymbol{X}, \boldsymbol{X})| - \frac{1}{2}\ \boldsymbol{y}^T\ k(\boldsymbol{X}, \boldsymbol{X})^{-1}\ \boldsymbol{y} \tag{3}$$

where $\boldsymbol{X}$ and $\boldsymbol{y}$ are the training inputs and outputs, respectively. Upon optimizing the hyperparameters, as discussed in Section 2.1.4, the predictions of the GP on testing inputs $\boldsymbol{X}_*$ constitute a Gaussian distribution:

$$\begin{aligned} p(\boldsymbol{y}_* \mid \boldsymbol{X}, \boldsymbol{X}_*, \boldsymbol{y}) \sim \mathcal{N}\Big(\ & k(\boldsymbol{X}_*, \boldsymbol{X})\ k(\boldsymbol{X}, \boldsymbol{X})^{-1}\ \boldsymbol{y}, \\ & k(\boldsymbol{X}_*, \boldsymbol{X}_*) - k(\boldsymbol{X}_*, \boldsymbol{X})\ k(\boldsymbol{X}, \boldsymbol{X})^{-1}\ k(\boldsymbol{X}, \boldsymbol{X}_*)\ \Big) \end{aligned} \tag{4}$$

where $p(\boldsymbol{y}_* \mid .)$ is the probabilistic prediction of the GP with mean vector $k(\boldsymbol{X}_*, \boldsymbol{X})\ k(\boldsymbol{X}, \boldsymbol{X})^{-1}\ \boldsymbol{y}$ and covariance $k(\boldsymbol{X}_*, \boldsymbol{X}_*) - k(\boldsymbol{X}_*, \boldsymbol{X})\ k(\boldsymbol{X}, \boldsymbol{X})^{-1}\ k(\boldsymbol{X}, \boldsymbol{X}_*)$.

2.1.2. Multi-output Gaussian processes (MOGP)

MOGPs model and predict vector outputs of size M. For any input matrix $\boldsymbol{X}$, let the matrix of outputs be denoted by $\bar{\boldsymbol{Y}} = [\boldsymbol{y}_1,\ \boldsymbol{y}_2, \ldots,\ \boldsymbol{y}_N]^\intercal$. Note that $\boldsymbol{y}_i$ is of size $M \times 1$ and $\bar{\boldsymbol{Y}}$ is of size $N \times M$. The matrix $\bar{\boldsymbol{Y}}$ is vectorized and represented as $\hat{\boldsymbol{y}}$ with size $NM \times 1$. $\hat{\boldsymbol{y}}$ is modeled with a zero-mean Gaussian distribution prior, defined as:

$$\hat{\boldsymbol{y}} \sim \mathcal{N}\Big(\hat{\boldsymbol{0}},\ \bar{\boldsymbol{K}}\Big) \tag{5}$$

where $\hat{\boldsymbol{0}}$ is the mean vector and $\bar{\boldsymbol{K}}$ is the full covariance matrix. $\bar{\boldsymbol{K}}$ captures covariances across the input variables and the vector of outputs, and thus has a size of $NM \times NM$. $\bar{\boldsymbol{K}}$ can be modeled in several different ways, as discussed in [18, 19]. As shown in Figure 2, we will follow the linear model of co-regionalization (LMC), which distinctly models the covariances between the N inputs and the M outputs. Mathematically, the LMC is defined as [18, 20]:

$$\bar{\boldsymbol{K}} = \sum_{q=1}^{Q} \bar{\boldsymbol{B}}_q \otimes \boldsymbol{K}_q \tag{6}$$

where q denotes the basis index, $\bar{\boldsymbol{B}}_q$ is the outputs covariance matrix of size $M \times M$ for the q^{th} covariate, $\boldsymbol{K}_q$ is the inputs covariance matrix of size $N \times N$ for the q^{th} covariate, Q is the total number of bases, and $\otimes$ denotes the Kronecker product. $\bar{\boldsymbol{B}}_q$ is further defined as the sum of two matrices of weight [20]:

$$\bar{\boldsymbol{B}}_q = \boldsymbol{A}_q \boldsymbol{A}_q^\intercal + \text{diag}\Big(\boldsymbol{\lambda}_q\Big) \tag{7}$$

where $\boldsymbol{A}_q$ and $\boldsymbol{\lambda}_q$ are, respectively, the matrix (size $M \times R$) and vector (size $M \times 1$) of hyperparameters, both for the q^{th} basis. The size R is user defined and can be greater than or equal to 1. The larger the R, the more sophisticated the MOGP in modeling complex outputs. Furthermore, the size of Q can also be greater than or equal to 1. Again, the larger the Q, the more sophisticated the MOGP in modeling complex outputs. In total, the MOGP with the LMC output covariance and the squared exponential input covariance kernel will have $Q\ (D+1)\ (M+1)\ R$ hyperparameters to be optimized arising from Q basis. If $Q = 1$, the LMC reduces to the intrinsic co-regionalization model, with $(D+1)\ (M+1)\ R$ hyperparameters to be optimized. The MOGP log-likelihood function has a form similar to that of a scalar GP:

$$\mathcal{L} = -\frac{1}{2}\ \ln|\bar{\boldsymbol{K}}| - \frac{1}{2}\ \hat{\boldsymbol{y}}^T\ \bar{\boldsymbol{K}}^{-1}\ \hat{\boldsymbol{y}} - \frac{1}{2}\ N\ \ln(2\pi) \tag{8}$$

Once the MOGP hyperparameters are optimized, as discussed in Section 2.1.4, probabilistic predictions of the vector quantities of interest can be made. Given a prediction input $\boldsymbol{x}_*$, the probability distribution of the vector outputs is given by:

$$p(\hat{\boldsymbol{y}}_*|\boldsymbol{x}_*, \hat{\boldsymbol{y}}, \bar{\boldsymbol{x}}, \boldsymbol{\theta}) = \mathcal{N}(\hat{\boldsymbol{\mu}}_*,\ \bar{\boldsymbol{\Sigma}}_*) \tag{9}$$

where $\bar{\boldsymbol{x}}$ is the matrix of training inputs, $\hat{\boldsymbol{\mu}}_*$ is the mean vector, and $\bar{\boldsymbol{\Sigma}}_*$ is the covariance matrix. The mean vector is defined as:

$$\hat{\boldsymbol{\mu}}_* = \bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}}\ (\bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}},\hat{\boldsymbol{y}}})^{-1}\ \hat{\boldsymbol{y}} \tag{10}$$

where $\bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}}$ is the full covariance matrix of the training inputs and prediction inputs, and $\bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}},\hat{\boldsymbol{y}}}$ is the full covariance matrix of the training inputs. The covariance matrix $\bar{\boldsymbol{\Sigma}}_*$ is defined as:

$$\bar{\boldsymbol{\Sigma}}_* = \bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}_*} - \bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}}\ (\bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}},\hat{\boldsymbol{y}}})^{-1}\ \bar{\boldsymbol{K}}^{\intercal}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}} \tag{11}$$

where $\bar{\boldsymbol{K}}_{\hat{\boldsymbol{y}}_*,\hat{\boldsymbol{y}}_*}$ is the full covariance matrix of the prediction inputs.

2.1.3. Deep Gaussian process

Standard GPs entail the stationarity assumption, potentially limiting the GP's predictive performance (e.g., under regime changes in the input/output space). A stationary GP implies that the covariance between any two points depends only on the distance between them, not on their absolute locations. A DGP was first introduced by Damianou et al. [21] and Damianou et al. [22] as a means of overcoming this stationarity assumption. By moving the inputs through hidden Gaussian layers, a DGP achieves non-stationarity even while using standard kernel functions (e.g., a squared exponential kernel) [23]. Several DGP variants were proposed based on the optimization procedures used for determining the hyperparameters [24, 25]. Herein, we rely on the DGP formulation of Sauer et al. [23], who used MCMC for hyperparameter optimization. Considering a single-hidden-layer DGP (see Figure 2), output y is modeled as GPs over the hidden layer latents $\boldsymbol{w}$, which are themselves modeled as a GP over the input $\boldsymbol{x}$. The prior is mathematically described as:

$$\begin{aligned} y|\boldsymbol{w} &\sim \mathcal{N}\Big(0,\ k(\boldsymbol{w},\boldsymbol{w}')\Big) \\ \boldsymbol{w} &\sim \mathcal{N}\Big(\boldsymbol{0},\ k(\boldsymbol{x},\boldsymbol{x}')\Big) \end{aligned} \tag{12}$$

Note that, for convenience, the prior is described for a scalar value of the output y corresponding to the input vector $\boldsymbol{x}$. In this case, the latents $\boldsymbol{w}$ are a vector of size p. Sauer et al. [23] recommends that p be equal to the size of the input vector. The log-likelihood function is the summation of log-likelihoods describing the mapping from y to $\boldsymbol{w}$ and from $\boldsymbol{w}$ to $\boldsymbol{x}$. Given N training inputs, $\boldsymbol{X}$, $\boldsymbol{y}$, and $\boldsymbol{W}$ have sizes of $N \times D$, N, and $N \times p$, respectively. $\boldsymbol{W}^i$ is the vector of latents for the i^{th} node in the hidden layer, and has dimensionality N. The compound log-likelihood function is given by:

$$\begin{aligned} \ln\ p(\boldsymbol{y}\ |\ \boldsymbol{W}, \sigma^2, \boldsymbol{l}, \tau^2) &\propto -\frac{1}{2}\ \ln|k(\boldsymbol{W},\boldsymbol{W})| - \frac{1}{2}\ \boldsymbol{y}^T\ k(\boldsymbol{W},\boldsymbol{W})^{-1}\ \boldsymbol{y} \\ \ln\ p(\boldsymbol{W}\ |\ \boldsymbol{X}, \boldsymbol{l}_W) &\propto \sum_{i=1}^{p} -\frac{1}{2}\ \ln|k^i(\boldsymbol{X},\boldsymbol{X})| - \frac{1}{2}\ (\boldsymbol{W}^i)^T\ k^i(\boldsymbol{X},\boldsymbol{X})^{-1}\ \boldsymbol{W}^i \\ \ln\ p(\boldsymbol{y}\ |\ \boldsymbol{W}, \sigma^2,\ \boldsymbol{X}, \boldsymbol{l},\ \boldsymbol{l}_W, \tau^2) &= \ln\ p(\boldsymbol{y}\ |\ \boldsymbol{W}, \sigma^2, \boldsymbol{l}, \tau^2) + \ln\ p(\boldsymbol{W}\ |\ \boldsymbol{X}, \boldsymbol{l}_W) \end{aligned} \tag{13}$$

The DGP hyperparameters are optimized with respect to the log-likelihood function above, as discussed in Section 2.1.4. For the testing inputs $\boldsymbol{X}_*$, the latents are first predicted per:

$$\begin{aligned} \mu_{w^i}(\boldsymbol{X}_*) &= k^i(\boldsymbol{X}_*,\boldsymbol{X})\ k^i(\boldsymbol{X},\boldsymbol{X})^{-1}\ \boldsymbol{W}^i \\ \Sigma_{w^i}(\boldsymbol{X}_*) &= k^i(\boldsymbol{X}_*,\boldsymbol{X}_*) - k^i(\boldsymbol{X}_*,\boldsymbol{X})\ k^i(\boldsymbol{X},\boldsymbol{X})^{-1}\ k^i(\boldsymbol{X},\boldsymbol{X}_*) \end{aligned} \tag{14}$$

Note that the index i denotes the node in the hidden layer. Using these latents, the output mean and covariance matrix are predicted per:

$$\begin{aligned}\boldsymbol{\mu}_* &= k(\boldsymbol{W}_*, \boldsymbol{W})\ k(\boldsymbol{W}, \boldsymbol{W})^{-1}\ \boldsymbol{y} \\ \boldsymbol{\Sigma}_* &= k(\boldsymbol{W}_*, \boldsymbol{W}_*) - k(\boldsymbol{W}_*, \boldsymbol{W})\ k(\boldsymbol{W}, \boldsymbol{W})^{-1}\ k(\boldsymbol{W}, \boldsymbol{W}_*)\end{aligned} \tag{15}$$

2.1.4. Gradient-based and gradient-free optimization methods for hyperparameter tuning

For gradient-based optimization of the hyperparameters of the GP variants, MOOSE employs adaptive moment estimation (Adam) [26]. Adam is a stochastic optimization algorithm that permits mini-batch sampling during the optimization iterations. In traditional Adam with regularization, the gradient update and hyperparameter update steps are defined as [26]:

$$\begin{aligned}\boldsymbol{g}_t &\leftarrow \nabla \mathcal{L}_t(\boldsymbol{\theta}_{t-1}) + \lambda\ \boldsymbol{\theta}_{t-1} \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \Big(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{\nu}}_t} + \varepsilon) \Big)\end{aligned} \tag{16}$$

where t is the iteration, $\boldsymbol{\theta}$ represents the optimizable hyperparameters, $\boldsymbol{g}$ is the gradient update, λ is the regularization weight, α and ε are internal parameters of the algorithm, $\hat{\boldsymbol{m}}$ is the corrected first moment update, $\hat{\boldsymbol{\nu}}$ is the corrected second moment update, and η is the schedule multiplier. Loshchilov et al. [27] proposed the AdamW algorithm, which modifies how the regularization is performed in Adam, thereby increasing its optimization performance. AdamW modifies the gradient update and hyperparameter update steps as follows [27]:

$$\begin{aligned}\boldsymbol{g}_t &\leftarrow \nabla \mathcal{L}_t(\boldsymbol{\theta}_{t-1}) \\ \boldsymbol{\theta}_t &\leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \Big(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{\nu}}_t} + \varepsilon) + \lambda\ \boldsymbol{\theta}_{t-1} \Big)\end{aligned} \tag{17}$$

wherein we see that the regularization is decoupled from the gradient update step and instead added to the hyperparameter update step. Loshchilov et al. [27] found that this decoupling generally enhanced the Adam algorithm's performance across the suite of case studies considered.

In MOOSE, gradient-free optimization is also available for tuning the GP hyperparameters, particularly the DGP. This is based on MCMC sampling via the elliptical slice sampler (ESS) and Metropolis-Hastings (MH) sampler. ESS is particularly well suited for fields $\boldsymbol{f}$ with Gaussian priors $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ [28]. A random angle $\gamma \sim \mathcal{U}(0,\ 2\pi)$ is drawn with the bounds set to $\gamma_{\min} = \gamma - 2\pi$ and $\gamma_{\max} = \gamma$. A new proposal for $\boldsymbol{f}$ is then made with the acceptance rate α, as shown below [28]:

$$\begin{aligned}\boldsymbol{f}^* &= \boldsymbol{f}^{t-1} \cos\gamma + \boldsymbol{f}^{\text{prior}} \sin\gamma \\ \alpha &= \min\left(1,\ \frac{\mathcal{L}(\boldsymbol{f}^*)}{\mathcal{L}(\boldsymbol{f}^{t-1})}\right)\end{aligned} \tag{18}$$

where t is the MCMC iteration index and $\mathcal{L}$ denotes the likelihood function. Crucially, in contrast to the MH sampler, if the proposal $\boldsymbol{f}^*$ is rejected, the bounds on γ are shrunken to $\gamma_{\min} = \gamma$ (if $\gamma < 0$) and $\gamma_{\max} = \gamma$ (O.W.). A new proposal for γ is then made using $\mathcal{U}(\gamma_{\min},\ \gamma_{\max})$. The procedure is repeated until the new proposal $\boldsymbol{f}^*$ is accepted in the current iteration t. For DGPs in particular, Sauer et al. [23] proposed a hybrid version of ESS and the MH sampler in order to improve hyperparameter inference, and this version is implemented in MOOSE. At each MCMC iteration t, the MH sampler is first used to update the parameters $\boldsymbol{l}$, σ^2, τ^2, and $\boldsymbol{l}_W^i$ in sequence, such as in a Gibbs sampling scheme. Then, by conditioning on these new values, the latents $\boldsymbol{W}$ are updated using ESS. The updating for iteration t is given by:

$$\begin{aligned}\sigma^2[t], \tau^2[t] &\text{ via } \texttt{MH} \text{ with } p(\boldsymbol{y} \mid \boldsymbol{W}, \sigma^2, \boldsymbol{l}, \tau^2) \\ \boldsymbol{l}[t] &\text{ via } \texttt{MH} \text{ with } p(\boldsymbol{y} \mid \boldsymbol{W}, \sigma^2, \boldsymbol{l}, \tau^2) \\ \boldsymbol{l}_W^i[t] &\text{ via } \texttt{MH} \text{ with } p(\boldsymbol{W} \mid \boldsymbol{X}, \boldsymbol{l}_W)\ \forall\ i \in \{1, \ldots, p\} \\ \boldsymbol{W}^i[t] &\text{ via } \texttt{ESS} \text{ with } p(\boldsymbol{y} \mid \boldsymbol{W}, \sigma^2, \boldsymbol{l}, \tau^2)\ \forall\ i \in \{1, \ldots, p\}\end{aligned} \tag{19}$$

Note that the combination of MH and ESS for updating at each MCMC iteration resembles a Gibbs sampling scheme. Also, $p(.)$ in Equation (19) is used for decision making in either the MH sampler or ESS to accept/reject a proposed sample.

2.2. Batch acquisition functions for parallelized active learning

MOOSE currently features several acquisition functions for a variety of tasks such as Bayesian optimization, Bayesian inverse UQ, and global surrogate fitting. These acquisition functions are dependent on the mean prediction ($\hat{\mu}$) and standard deviation ($\hat{\sigma}$) of the GP variant. Table 1 presents these acquisition functions and also lists their usage. Note that some of them have a tuning parameter λ whose functionality depends on the usage. For example, λ serves to boost either exploratory or exploitative behavior for Bayesian optimization and Bayesian inverse UQ tasks. In contrast, λ is the failure threshold for a rare events analysis task. Also, for some GP variants such as MOGP, the mean prediction and standard deviation are vector quantities. In such a case, the computed acquisition function will also be a vector quantity that must be reduced to a scalar by using operations such as sum, average, maximum, minimum, or product.

Table 1: Acquisition functions in MOOSE for active learning for tasks such as optimization, Bayesian inverse UQ, and global surrogate fitting.

Acquisition function $a(\boldsymbol{x})$	**Mathematical form**	**Usage**
Expected Improvement [29]	$z\Phi(z/\hat{\sigma}) + \hat{\sigma}\phi(z/\hat{\sigma})$	Bayesian optimization
Upper Confidence Bound [30]	$\lambda\hat{\sigma} + \hat{\mu}$	Bayesian optimization
Probability of Improvement [29]	$\Phi\big((\hat{\mu} - \mathcal{M}(\boldsymbol{x}^*))/\hat{\sigma}\big)$	Bayesian optimization
Bayesian posterior targeted [31]	$\exp(2\lambda\hat{\mu})\big(\exp(\hat{\sigma}) - 1\big)$	Bayesian inverse UQ
U-function [32, 33]	$(\hat{\mu} - \lambda)/\hat{\sigma}$	Rare events analysis
Expected Improvement for Global Fit [34]	$\big(\hat{\mu} - \mathcal{M}(\boldsymbol{x}^*)\big)^2 + \hat{\sigma}^2$	Global fitting
Coefficient of variation	$\hat{\sigma}/\hat{\mu}$	Global fitting

ϕ : Gaussian probability density function (PDF), Φ : Gaussian cumulative distribution function (CDF), $\hat{\mu}$: GP variant mean, $\hat{\sigma}$: GP variant standard deviation, $\mathcal{M}$: Computational model, $\boldsymbol{x}^*$: current best point, λ : acquisition function parameter, and $z = \hat{\mu} - \lambda - \mathcal{M}(\boldsymbol{x}^*)$

The acquisition functions listed in Table 1 permit sequential active learning, with one optimal location $\boldsymbol{x}$ being specified to run the full-fidelity MOOSE model. However, sequential active learning can incur significant computational cost, as running the full-fidelity MOOSE model several times in sequence is expensive. To alleviate this, we used batch versions of the acquisition functions, where b (a user-defined parameter) optimal locations of the inputs are specified to run the MOOSE model in parallel. For simplicity, we adopted the local penalization approach proposed by Zhan et al. [35]. In it, a correlation function between two inputs is first defined as:

$$Corr(\boldsymbol{x}, \boldsymbol{x'}) = 1 - \exp\Big(-\frac{1}{2}\sum_{d=1}^{D}\frac{(x - x')^2}{l_d^2}\Big) \tag{20}$$

where $\boldsymbol{l}$ represents the length scales, as obtained through GP hyperparameter optimization. The b optimal points for running the MOOSE model are defined as:

$$\begin{aligned}
\boldsymbol{x}^1 &= \arg\max_{\boldsymbol{x}} \; a(\boldsymbol{x}) \\
\boldsymbol{x}^2 &= \arg\max_{\boldsymbol{x}} \; a(\boldsymbol{x})\; Corr(\boldsymbol{x}, \boldsymbol{x}^1) \\
\boldsymbol{x}^b &= \arg\max_{\boldsymbol{x}} \; a(\boldsymbol{x}) \prod_{i=1}^{b-1} Corr(\boldsymbol{x}, \boldsymbol{x}^i)
\end{aligned} \tag{21}$$

In this manner, we can select b optimal points within each iteration of active learning by performing local penalization to mitigate any clustering of those points. These b points can be evaluated in parallel by

using a MOOSE model, and the GP variant is retrained by appending the input/output data with the new points. These active learning capabilities are used for Bayesian analysis of fission product release in an advanced nuclear fuel (Section 4.1), rare events analysis of a nuclear reactor (Section 4.2), and optimizing a computational model in nuclear fusion (Section 4.5), as discussed later in this paper.

2.3. Inverse sampling and Bayesian inference

For inverse UQ, it is often of interest to calibrate computational models given the experimental data while quantifying the uncertainties associated with model parameters, model inadequacy (i.e., model structural error), and experimental noise. Following the Kennedy and O'Hagan framework [36], the experimental data are defined to have originated from a generative model of the following form assuming independent and identically distributed experiments:

$$\begin{aligned} &\mathcal{D}(\Theta_i) = \mathcal{M}(\boldsymbol{\theta},\ \Theta_i) + \delta(\Theta_i) + \varepsilon \\ &\text{where, } \varepsilon \sim \mathcal{L}(\sigma_\varepsilon) \end{aligned} \tag{22}$$

where the i^{th} experimental observation is indicated to be the model prediction plus a model inadequacy term (δ), plus a correction factor (ε) to account for noise in the experimental data. In Equation (22), $\mathcal{M}$ is the computational model, $\boldsymbol{\theta}$ are the model parameters, and Θ is the experimental configuration. The model inadequacy term is traditionally modeled with a standard GP, as further discussed in Section 2.1.1. The correction factor is treated as a random variable that follows a probability distribution generically defined as $\mathcal{L}$, and whose scale is σ_ε and mean is 0. $\mathcal{L}$ is the likelihood function that evaluates the adequacy of the model predictions against the experimental data for a given $\boldsymbol{\theta}$ and σ_ε:

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_\varepsilon|\boldsymbol{\Theta}, \mathcal{M}, \boldsymbol{\mathcal{D}}) = \prod_{i=1}^{N} \mathcal{L}(\boldsymbol{\theta}, \sigma_\varepsilon|\Theta_i, \mathcal{M}, \mathcal{D}_i) \tag{23}$$

where the term within the product sign is specific to a given experimental configuration, and the product sign itself indicates that the experiments are independent and identically distributed. Specifically, under the Gaussian assumption, the likelihood function becomes:

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_\varepsilon|\boldsymbol{\Theta}, \mathcal{M}, \boldsymbol{\mathcal{D}}) = \prod_{i=1}^{N} \mathcal{N}\big(\mathcal{D}(\Theta_i) - \mathcal{M}(\boldsymbol{\theta},\ \Theta_i) - \delta(\Theta_i),\ \sigma_\varepsilon\big) \tag{24}$$

With the likelihood function defined, the Bayesian inference problem entails quantifying the posterior distribution of $\{\boldsymbol{\theta}, \sigma_\varepsilon\}$ [36, 37, 38, 39, 40]:

$$f(\boldsymbol{\theta}, \sigma_\varepsilon|\boldsymbol{\Theta}, \mathcal{M}, \boldsymbol{\mathcal{D}}) \propto \mathcal{L}(\boldsymbol{\theta}, \sigma_\varepsilon|\boldsymbol{\Theta}, \mathcal{M}, \boldsymbol{\mathcal{D}})\ f(\boldsymbol{\theta}, \sigma_\varepsilon) \tag{25}$$

where $f(\boldsymbol{\theta}, \sigma_\varepsilon)$ defines the prior distribution before observing new experimental data. The proportionality constant in Equation (25) is a multidimensional integration over $\{\boldsymbol{\theta}, \sigma_\varepsilon\}$ and is typically unknown. Thus, MCMC techniques are traditionally used to solve the Bayesian inverse problem.

MCMC techniques, widely regarded as the gold standard for solving the Bayesian inference problem, involve drawing samples from the posterior distribution described by Equation (25). Use of an MCMC sampler in practice is presented in Figure 3a. We start from an arbitrary realization of $\{\boldsymbol{\theta},\ \sigma\}$ and propose a new sample. The proposal can rely on the proposal distribution if the MCMC sampler falls under the MH class. Otherwise, it can be implicitly defined without requiring a proposal distribution, as in the case of an ensemble MCMC sampler [41, 42]. In any case, the computational model is then evaluated for the newly proposed $\{\boldsymbol{\theta},\ \sigma\}$. Using the computational model output, the likelihood function is evaluated and the transition probability with respect to the old sample is computed. The new proposal is accepted with probability t_{xy}. Repeating the process of making a new proposal, evaluating the computational model and the likelihood function, and accepting/rejecting the proposal a sufficient number of times will give us the samples from the required posterior distribution.

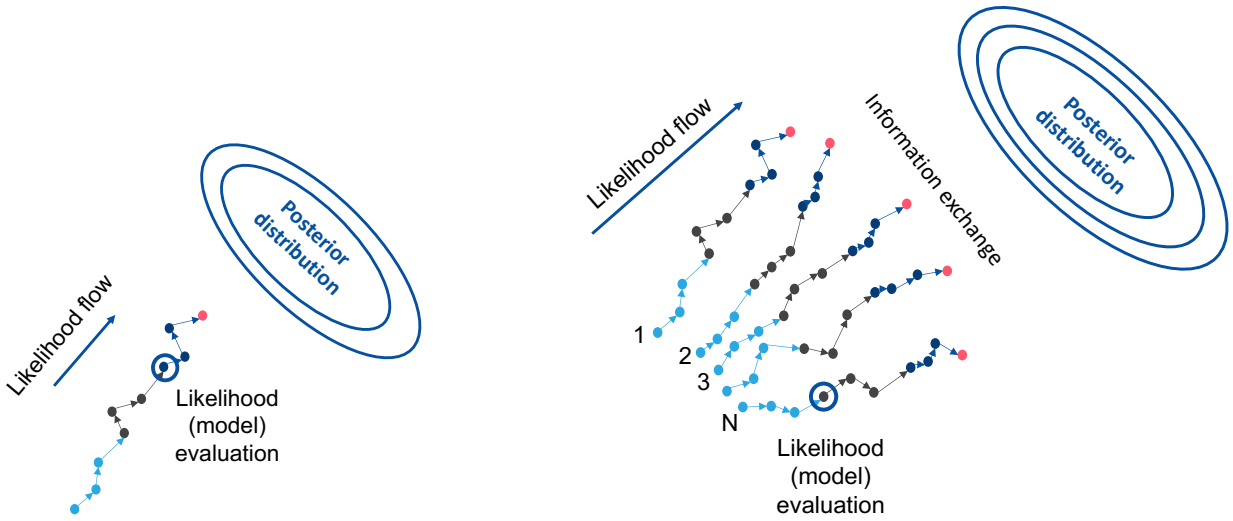


Figure 3: (a) Serial and (b) parallel/ensemble MCMC methods for obtaining samples from the posterior distribution. In comparison to serial MCMC samplers, parallel/ensemble MCMC samplers usually accelerate convergence to the posterior distribution.

This version of the MCMC sampler is serial in nature. Thus, it can take a significant number of serial steps to reach convergence, entailing many serial evaluations of the computational model. As this can be very expensive in practice, we will discuss parallelizable MCMC samplers that have multiple parallel Markov chains. Figure 3b presents the working principle behind parallel MCMC samplers, which is similar to that of a serial MCMC sampler. At each step, P parallel proposals are made, then the computational model corresponding to each proposal is evaluated. Since these model evaluations are independent of each other, they can be parallelized. The outputs are then used to compute the likelihood functions, and the Markov chains exchange information with each other to determine the next-best set of P parallel proposals. The manner in which information exchange between chains is formulated differentiates the parallel MCMC samplers. Calderhead [43] proposed a parallelized version of the MH class of samplers. Goodman and Weare [42] proposed a version of ensemble MCMC based on the affine invariance property, whereas Braak [41] proposed one based on differential evolution optimization [44]. All these parallel MCMC variants are available in MOOSE. Interested readers are referred to [43, 42, 41, 45] for the corresponding mathematical details.

In addition to being massively parallelizable, parallel/ensemble samplers have been shown to accelerate convergence to the posterior, in comparison to the serial MCMC samplers. Studies such as Laloy and Vrugt [46], Foreman-Mackey et al. [47], and Opara and Arabas [48] discuss the convergence of MCMC samplers with the aid of metrics such as the Gelman-Rubin diagnostic [49] and the effective sample size.

For any new experimental configuration $\hat{\Theta}$, the posterior predictive distribution is:

$$f\big(\mathcal{M}(\hat{\Theta},\ \boldsymbol{\theta})|\boldsymbol{\Theta},\boldsymbol{\mathcal{D}}\big) = \int_{\sigma_\varepsilon}\int_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta},\sigma_\varepsilon|\boldsymbol{\Theta},\mathcal{M},\boldsymbol{\mathcal{D}})\ f(\boldsymbol{\theta},\sigma_\varepsilon|\boldsymbol{\Theta},\mathcal{M},\boldsymbol{\mathcal{D}})\ d\boldsymbol{\theta}\ d\sigma_\varepsilon \tag{26}$$

where $\mathcal{L}(\boldsymbol{\theta},\sigma_\varepsilon|\boldsymbol{\Theta},\mathcal{M},\boldsymbol{\mathcal{D}})$ has the same form as in Equation (23). From the probability distribution of the model prediction described in Equation (26), statistics such as the median prediction and confidence bands can be inferred. This requires forward sampling techniques, discussed next. The inverse UQ capabilities are used for Bayesian analysis of fission product release in an advanced nuclear fuel (Section 4.1), as discussed later in this paper.

2.4. Forward sampling

Forward sampling methods sample from a known probability distribution $q(\boldsymbol{x})$. Traditional methods such as Monte Carlo sampling and Latin hypercube sampling (LHS) are available in MOOSE. When estimating certain statistics, Monte Carlo and LHS may require numerous evaluations of the model $\mathcal{M}$, thus becoming computationally intractable. There may also be cases in which directly drawing samples from the distribution $q(\boldsymbol{x})$ is infeasible. Importance sampling addresses these concerns by sampling from an importance density $f(\boldsymbol{x})$. The mean estimator of the quantity of interest $\mathcal{Q}\big(\mathcal{M}(\boldsymbol{x})\big)$ is then computed via the modified equation [50]:

$$\hat{\mathcal{Q}} = \frac{1}{S} \sum_{i=1}^{S} \mathcal{Q}\big(\mathcal{M}(\boldsymbol{x}_i)\big) \ \frac{q(\boldsymbol{x}_i)}{f(\boldsymbol{x}_i)} \tag{27}$$

where S is the number of samples drawn from the importance density $f(\boldsymbol{x})$. The variance of the estimator is computed per [50]:

$$\mathrm{Var}(\hat{\mathcal{Q}}) = \frac{1}{S} \left\{ \frac{1}{S} \sum_{i=1}^{S} \left[\mathcal{Q}\big(\mathcal{M}(\boldsymbol{x}_i)\big) \ \frac{q(\boldsymbol{x}_i)}{f(\boldsymbol{x}_i)} \right]^2 - \hat{\mathcal{Q}}^2 \right\} \tag{28}$$

A crucial component of importance sampling is the creation of importance density $f(\boldsymbol{x})$. To estimate rare events, MCMC is a popular approach for creating $f(\boldsymbol{x})$ by using an adaptive importance sampling scheme [51, 52, 53]. For other applications, methods that use control variates [54], multilevel Monte Carlo [55], and multifidelity modeling [56] have also been proposed to create $f(\boldsymbol{x})$.

For more complex forward UQ applications such as global optimization and very rare events analysis, MOOSE also features a parallel subset simulation sampler [57, 58]. This is a variant of the sequential Monte Carlo sampler [59], with the goal being to sample from the failure or the optimal region. Subset simulation creates a series of intermediate thresholds—representing the suboptimal regions—that incrementally draw nearer to the optimal region. The method begins with regular Monte Carlo sampling for N samples. The top $p_o \in [0,\ 1]$ samples are then selected in light of the quantity of interest $\mathcal{Q}\big(\mathcal{M}(\boldsymbol{x})\big)$. Using these p_o samples, Markov chains are initiated such that they propagate toward the optimal region and not in the other direction. If there are N_M Markov chains, each is evaluated $\mathrm{int}(N/N_M)$ times to obtain N samples from this intermediate suboptimal region. The process of selecting the top p_o samples from this intermediate region and initiating the Markov chains is repeated until convergence is achieved. As tens or hundreds of Markov chains are propagated in each subset, these and the corresponding MOOSE model evaluations can be massively parallelized. Note that parallelization can only be achieved across all the Markov chains, and not within the individual chains. More advanced versions of subset simulation have been proposed with respect to aspects such as the dynamic/adaptive intermediate thresholds [59, 60] and the MCMC samplers [61, 62, 63]. Building on the subset simulation sampler, other variants of this method—or of sequential Monte Carlo samplers in general—can be implemented in MOOSE at some point in the future. The forward UQ capabilities are used for rare events analysis of a nuclear reactor (Section 4.2), as discussed later in this paper.

2.5. Dimensionality reduction

MOOSE stochastic tools module supports linear principal component analysis (PCA), a dimensionality reduction technique widely used across multiple scientific disciplines [64]. Linear PCA can be used to determine a lower-dimensional space (latent space) that is closest to the given data in a discrete L^2 norm. Let $\boldsymbol{s} \in \mathbb{R}^N$ be a high-dimensional vector (N is large) representing the high-dimensional solution fields from numerical solvers in MOOSE. To discover a low-dimensional latent space by using PCA, we collect snapshots of the solution fields and organize them into a snapshot matrix $\boldsymbol{S} = [\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_{N_s}]$. For discrete problems such as the one presented here, singular value decomposition (SVD) is performed for a linear PCA analysis. Therefore, we can obtain the principal components of the snapshots (basis functions of the latent space) by computing the SVD of the snapshot matrix:

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \tag{29}$$

where matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary and contain the left and right singular vectors, respectively, whereas diagonal matrix $\boldsymbol{\Sigma}$ contains the singular values. MOOSE relies on the parallel SVD solvers through the aid of SLEPc [65], enabling it to efficiently compress very high-dimensional output fields. The columns of $\boldsymbol{U}$ are also called principal components, and can be used to approximate the high-dimensional snapshots per:

$$\boldsymbol{s} \approx \boldsymbol{U}_r \boldsymbol{c}_r \tag{30}$$

where $\boldsymbol{c}_r \in \mathbb{R}^r$ contains the expansion coefficients or coordinates in the lower-dimensional latent space, while matrix $\boldsymbol{U}_r$ contains the first r principal components. The columns of $\boldsymbol{U}_r$ span the closest r-dimensional subspace to the snapshots in $\boldsymbol{S}$. Based on this expression and the fact that the principal components are orthonormal, we can map the snapshots to the latent space via the following operation:

$$\boldsymbol{c}_r = \boldsymbol{U}_r^T \boldsymbol{s} \tag{31}$$

To determine the necessary number of principal components, (i.e., r) an explained variation-based approach is utilized that relies on the the singular values (σ_i) located on the diagonal of matrix $\boldsymbol{\Sigma}$:

$$r = \underset{1 \leq r \leq N_s}{\arg\min} \left(1 - \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{N_s} \sigma_i^2} \right) < \tau \tag{32}$$

The above metric selects r so that the relative sum of the squared singular values from r to N_s is lower than a given number $\tau \in (0, 1]$. The dimensionality reduction capabilities are used for advanced manufacturing process modeling (Section 4.3), as discussed later in this paper.

3. MOOSE Code Implementations

3.1. Background on the MOOSE Stochastic Tools Module

The MOOSE stochastic tools module aims to efficiently and scalably sample parameters, run multiphysics models, and perform stochastic analyses, including UQ, sensitivity analysis, and surrogate model generation. In Slaughter et al. [11], a more comprehensive and general overview of the module is presented. The following subsections describe the MOOSE systems relevant to the probabilistic ML and UQ techniques focused on in this paper.

3.1.1. `Samplers` system

The `Samplers` system represents a class of objects responsible for generating random samples. MOOSE provides a variety of objects for specific sampling strategies, including `MonteCarlo` and `LatinHypercube` for basic random sampling, `Quadrature` for sparse quadrature sampling, `AdaptiveImportance` and `ParallelSubsetSimulation` for MC-based forward-UQ sampling, and various objects for MC-based inverse-UQ sampling. For adaptive sampling schemes (e.g., MC-based sampling), these objects can gather data from associated objects so as to determine subsequent sets of samples—for instance, gathering whether or not a sample was rejected or accepted in the chain. `Samplers` also define how the multiphysics runs are parallelized. Typically, the number of parallel runs and the number of processors needed for each run are determined programmatically, though there are input parameters that allow for user control.

3.1.2. `MultiApps` system

MultiApps is a framework-level system in MOOSE that enables instantiation of independent simulations [66]. MOOSE utilizes this system to run multiphysics simulations during stochastic sampling and to gather the results. In particular, it leverages the flexibility in distributing simulations across processors, making the stochastic simulations both extremely scalable and memory efficient. This parallelism works on two fronts: sample parallelism and model parallelism. Sample parallelism involves distributing the concurrent simulations evenly across the available processors—possibly leaving multiple processors per simulation. Model parallelism is supported by distributed memory parallelism (with MPI) and shared memory parallelism (with OpenMP). This interplay between sample and model parallelism is customizable within MOOSE to help with memory consumption for larger models. Further details on the distribution of MultiApps for MOOSE are presented in Slaughter et al. [11]. GPU-based parallelism for model parallelism is currently under development. However, MOOSE can be configured and compiled with Libtorch, the C++ frontend of PyTorch [67, 68]. Libtorch modules within MOOSE can already harness the GPU acceleration for Libtorch model evaluations, model training, and tensor manipulations.

3.1.3. `Reporters` system

The MOOSE `Reporters` system provides an interface for declaring, manipulating, and gathering global data in a given application. MOOSE primarily utilizes this system to store data from `MultiApps` runs during the stochastic simulation. `Reporter` objects also handle heterogeneous storage of the data, keeping data distributed for memory efficiency and homogenizing them when necessary. `Reporters` is also the primary strategy for outputting data such as UQ results, typically in the form of JSON files.

3.1.4. `Surrogates` system

The `Surrogates` system in MOOSE provides the capability to train and evaluate meta-models. `Trainers` are responsible for gathering parameter values from `Samplers` and responses from `Reporters` to compute the necessary data for model generation. These data can be declared globally or output for later use. `Surrogates` then takes the trained model and provides an interface for evaluating it. Specified `Trainers` and `Surrogates` are accessible from any MOOSE object in order to either evaluate the model based on specific parameters or retrain them on-the-fly. All the GP variants are built using the `Surrogates` system. While not directly relevant to probabilistic ML, MOOSE also has support for other types of surrogates such as polynomial chaos, polynomial, and proper orthogonal decomposition reduced basis. Importantly, MOOSE can be configured and compiled with Libtorch, the C++ frontend of PyTorch [67, 68]. This means more complex surrogates like neural networks can either be trained natively in MOOSE or be imported from Python/PyTorch via `TorchScript`.

3.2. Modularity: understanding the `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` interaction

The `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` systems in MOOSE afford extensive modularity and enable development of many variants of active learning, forward/inverse UQ, and Bayesian optimization algorithms. Moreover, these algorithms can be implemented in an inherently parallel manner by calling several instances of the computational MOOSE model in parallel, using the `MultiApp` system. Understanding how the `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` systems interact with each other—as well as their order of execution within MOOSE—is key to implementing these algorithms. This section discusses the interaction between these systems.

For the sake of simplicity, the interaction among `Sampler`, `MultiApp`, and `Reporter` is discussed first. `Sampler` proposes new samples from the underlying probability distributions, using objects in the `Distributions` system. These proposed samples are stored in a global array, with the rows containing the samples to be executed in parallel and the columns representing the parameters to the computational model. The numerical simulations corresponding to the proposed samples are automatically executed in parallel, if the user desires, via the `MultiApp` system. Upon execution, the simulation outputs are received by the `Reporter` system and stored in a JSON file. Under simple schemes such as Monte Carlo or LHS, the `Reporter` system only outputs the simulation results and the `Sampler` system then moves on to

propose the next batch of samples, without any influence from the previously proposed samples or their simulation outcomes. In schemes such as adaptive Monte Carlo and MCMC, the `Reporter` system plays a more crucial role of influencing the next batch of samples proposed by the `Sampler` system, depending upon the simulation outcomes of the previously proposed batch of samples. Several adaptive Monte Carlo and MCMC algorithms such as adaptive importance sampling and parallel subset simulation for forward UQ and parallel MH, and ensemble MCMC for inverse UQ, fit well within the `Sampler`, `MultiApp`, and `Reporter` interaction scheme in MOOSE. For Bayesian inverse UQ problems, the `Sampler` system performs the additional function of collecting the user-supplied experimental configuration data and combining them with the proposed samples of model parameters by creating combinations of these parameters and experimental configurations. Owing to the inherent parallelization via the `MultiApp` system, algorithms such as parallel subset simulation, parallel MH, and ensemble MCMC, which rely on multiple Markov chains, can be massively parallelized in terms of the computational model calls. Figure 4 presents the `Sampler`, `MultiApp`, and `Reporter` system interaction flowchart, along with several objects available in MOOSE for forward and inverse UQ applications.

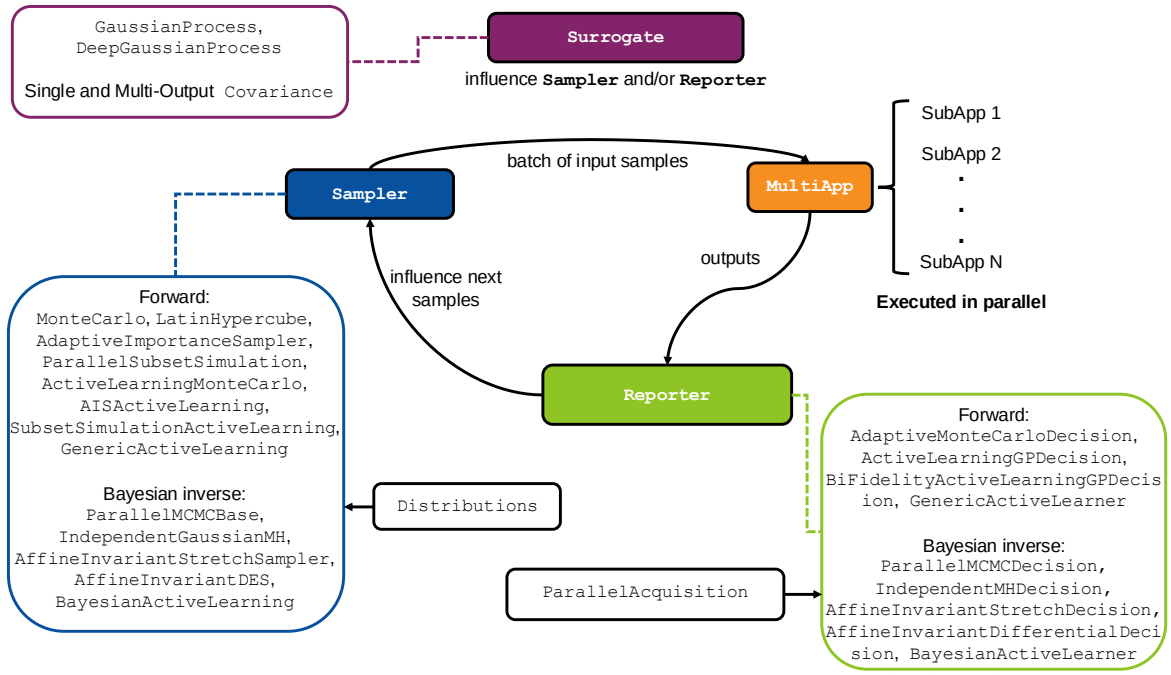


Figure 4: `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` system interaction in MOOSE for performing parallel active learning. The available objects deriving off of `Sampler` and `Reporter` are also shown in regard to supporting tasks such as forward/inverse UQ, Bayesian optimization, and active learning with different GP variants.

Next, we will discuss the `Surrogate` system's influence on the interaction among the `Sampler`, `MultiApp`, and `Reporter` systems. Training, evaluation, and active/online learning of the GP variants in MOOSE are handled by `Surrogate` and `Trainer`. The `Surrogate` system can be easily coupled to the `Reporter` system to influence its behavior and/or that of the `Sampler` system. For example, in parallel active learning tasks such as forward/inverse UQ and Bayesian optimization, the GP surrogate variant, based on its predictive uncertainties and the acquisition function values, tells the `Sampler` system the best sets of input parameters under which to call the MOOSE computational model during the next iteration. After evaluating the computational model, in parallel, the outputs will be obtained by the `Reporter` system, which retrains the GP variant with the appended new data. The `Reporter` system will then query the acquisition function about the next-best sets of input parameters, and this process repeats until reaching a user-specified number of outer iterations. `GaussianProcess` and `DeepGaussianProcess` surrogates

are currently derivable off of the `Surrogate` system. Both rely on the `Covariance` system to set up the training data input/output covariances (output covariances are only required for the MOGP surrogate). They also rely on the `GaussianProcess` class, which handles the training and retraining by using the gradient-based Adam algorithm or gradient-free MCMC sampling. Here, "gradients" refers to gradients of the log-likelihood function of the GP variant. Figure 4 indicates how the `Surrogate` system influences the interaction among `Sampler`, `MultiApp`, and `Reporter`, and supports parallelized active learning. Moreover, a pre-trained GP surrogate variant saved as an .rd (restartable data) file can be loaded and evaluated by using a combination of user-specified `Sampler` and `Reporter` objects, without calling the MOOSE computational model.

3.3. Example implementation of parallelized active learning

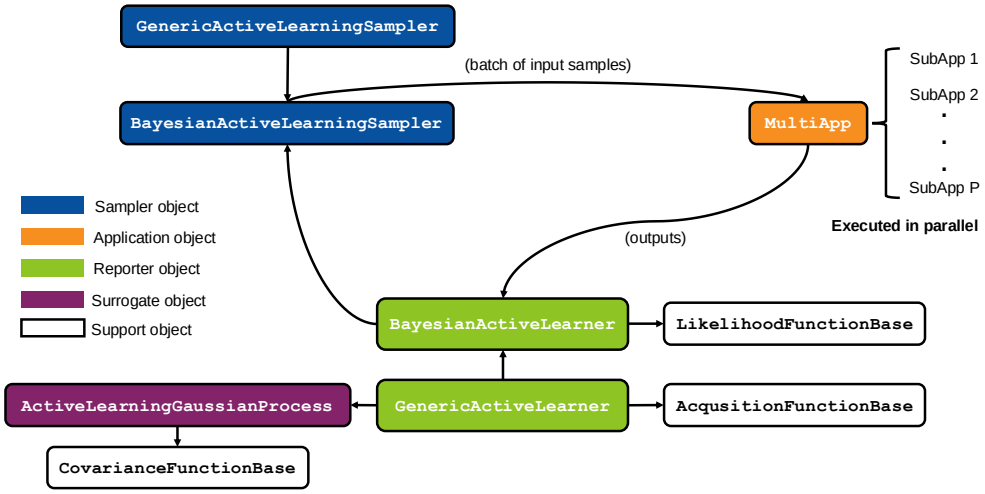
An example implementation of parallel active learning capabilities in MOOSE—via leveraging the `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` interaction—will now be discussed for Bayesian UQ and Bayesian optimization applications. Figure 5a presents the MOOSE objects and their dependencies. This schematic is comprised of the following main components:

- **`GenericActiveLearningSampler`/`BayesianActiveLearningSampler`:** `GenericActiveLearningSampler` creates a large population of input samples at each iteration, and this is retrieved by the `Reporter` object to facilitate optimization of the acquisition function. Importantly, this object also facilitates evaluation of the computational model via the `MultiApp` system for a best batch of inputs, as informed by the GP model. `BayesianActiveLearningSampler` derives from `GenericActiveLearningSampler` and is tailored for Bayesian UQ applications such that it considers the experimental configurations. Specifically, before sending the inputs to the `MultiApp` system, `BayesianActiveLearningSampler` combines them with the experimental configurations.

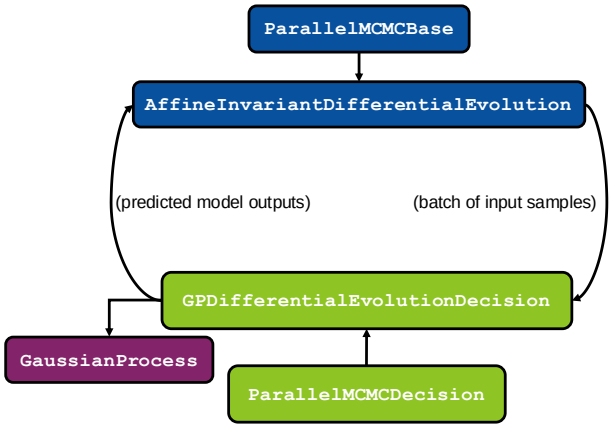
- **`GenericActiveLearner`/`BayesianActiveLearner`:** `GenericActiveLearner` optimizes the acquisition function via the `GaussianProcess` surrogate and selects the next-best set of inputs to the `Sampler` object. The acquisition function is optimized by selecting the best P inputs from among the large population of samples created earlier in the iteration by the `GenericActiveLearningSampler`. `BayesianActiveLearner` derives from `GenericActiveLearner` to compute the log-likelihood function, which serves as the training/retraining data for the GP for Bayesian UQ applications.

- **Support objects:** `CovarianceFunctionBase` constructs covariances for the GP object, based on the kernel specified by the user. `LikelihoodFunctionBase` evaluates the likelihood function, given inputs and model outputs based on the user-specified distribution. `AcquisitionFunctionBase` computes the acquisition function specified by the user and performs local penalization when selecting the best P input samples.

`GenericActiveLearningSampler` and `GenericActiveLearner` can readily perform batch Bayesian optimization for maximizing a user-defined objective evaluated via a MOOSE computational model. For Bayesian UQ, `BayesianActiveLearningSampler` and `BayesianActiveLearner` train a GP model by prioritizing regions of high log-likelihood via the Bayesian posterior targeted acquisition function detailed in Table 1. The trained GP model is saved as an .rd file. This will be used in conjunction with MCMC objects such as the `AffineInvariantDifferentialEvolution` sampler `GPDifferentialEvolutionDecision` reporter for sampling from the posterior distribution. Doing so circumvents evaluation of the MOOSE computational model, since the trained GP model will directly predict the log-likelihood values during forward evaluation. The flowchart in Figure 5b details the use of an actively trained GP model for sampling from the posterior distribution.



(a)



(b)

Figure 5: (a) MOOSE objects and their dependencies for performing parallel active learning for Bayesian optimization and Bayesian UQ applications by leveraging the `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` interaction. Note that the combination of `GenericActiveLearningSampler` and `GenericActiveLearner` performs Bayesian optimization. `BayesianActiveLearningSampler` and `BayesianActiveLearner` are derived objects for Bayesian UQ, and they consider the experimental configurations and likelihood functions supplied by the user. For Bayesian UQ, the actively trained GP prioritizes regions of high log-likelihood and is saved as an .rd file. (b) Evaluation phase of the actively trained GP for Bayesian UQ by leveraging the MCMC sampling objects; specifically, the `AffineInvariantDifferentialEvolution` for proposing new samples and the `GPDifferentialEvolutionDecision` for decision-making. Here, the GP model directly predicts the log-likelihood values under different input parameters and experimental configurations, thus circumventing evaluation of the computational MOOSE model.

3.4. A note on parallel scalability

Many of the discussed capabilities in MOOSE rely on generating samples of model parameters from distributions and executing the sub-applications (i.e., model evaluations) in parallel. To this end, MOOSE features three modes for parallelization: (1) `normal` mode creates one sub-application per row of data (i.e., one realization of the parameters) supplied by the `Sampler` object; (2) `batch-reset` mode creates N sub-applications, where the sub-applications are destroyed and re-created (on the same existing MPI communicator) for each row of data supplied by the `Sampler` object; and (3) `batch-restore` mode creates N sub-applications, where the sub-application is backed up after initialization and for each row of

data supplied by the `Sampler` object the sub-application is restored to the initial state prior to execution. Here, N is $\min\left(n_{\text{rows}}, \text{floor}(\frac{n_{\text{proc}}}{n_{\text{min}}})\right)$, where n_{rows} is the number of rows in the `Sampler` object, n_{proc} is the number of processors, and n_{min} is the minimum number of processors per sub-application specified by the user. More information can be found here: MOOSE Stochastic Tools Batch Mode.

Embarrassing parallelizability depends on two factors, the computational overhead and the parallelizability of the algorithms themselves. The computational overhead indeed increases with the number of parallel calls to the model and this overhead depends upon the model and the physics involved. However, we found for our practical applications that this overhead is minimal compared to the time it takes to evaluate the model itself. Sampling methods like Monte Carlo and Latin Hypercube are embarrassingly parallel by principle. Adaptive methods like the PSS and parallel MCMC can be parallelized to a certain extent. For example, the PSS method can only be parallelized across the Markov chains but not within each chain. This creates a limit for parallelization after which adding more processors can only further speed up the model evaluations but not the PSS method itself; for more information, interested readers are referred to Figure 8 in Dhulipala et al. [50]. MCMC methods for Bayesian inversion are parallelizable across the Markov chains and also the experimental configurations. However, after each iteration, all the MCMC chains in the implemented methods have to exchange information to decide their flow in the next iteration. Interested readers are referred to Figure 23 in Dhulipala et al. [69]. Parallelizability of the parallel active and batch Bayesian optimization methods is controlled by the user specified batch size which controls the number of model evaluations at each outer iteration.

4. Application Demonstrations

Most of the application demonstrations are run on the Sawtooth high-performance computing cluster at the Idaho National Laboratory. This cluster has 2052 compute nodes each with 2 Intel Xeon 8268 CPUs, Cascade Lake Platinum chipset, 24 cores per CPU, 192GB of RAM, and Mellanox Infiniband EDR for inter-node communication. It also features 27 nodes each with 2 Intel Xeon 8268 CPUs, Cascade Lake Platinum chipset, 24 cores per CPU, 384GB of RAM, Mellanox Infiniband EDR for inter-node communication, and 4 NVidia Tesla V100 32 GPUs.

4.1. Parallel active learning for Bayesian inverse UQ of TRISO nuclear fuel fission product release

This section uses the active learning with GP and forward UQ capabilities discussed in Sections 2.2 and 2.3, respectively.

Tristructural isotropic (TRISO) particle fuel is proposed for use in advanced reactors because of its high temperature resistance. Its protective layers are intended to encapsulate the fission products, which these reactor designs are based on. Thus, it critical to assess the predictive uncertainties in the TRISO fission product release model. To this end, inverse UQ of the TRISO fission product release model is necessary to quantify the uncertainties due to model parameters, model inadequacy, and experimental noise. A 25-mm-long, 6-mm-radius cylindrical fuel compact can contain approximately 10,000–15,000 TRISO particles, each with a radius of around 375–430 μm [70]. Each TRISO particle has several protective layers around the fuel kernel—namely, the buffer, inner pyrolitic carbon (PyC), silicon carbide, and outer PyC layers.

Fission products, particularly silver release, are modeled using the BISON fuel performance code [2, 71], which is a MOOSE-based application. The diffusion process of fission products in TRISO particles requires computation of the fuel temperature (if not prescribed), temperature-dependent diffusion coefficients, source rates for the fission products, and the particle geometry. Material models were developed in BISON for each type of material in the TRISO particles: the buffer, the PyC layers, the silicon carbide layer, and the fuel kernel. Fission product diffusion is governed by the Fickian diffusion equation, wherein the diffusivity of the fission products is in units of m^2/s, and is normally estimated via an effective diffusivity defined per an Arrhenius law. See [2, 71] for further details on the modeling using BISON. The values for the pre-exponential factor D_i and activation energy Q_i in the Arrhenius equation for the different TRISO layers are usually calibrated from existing experimental data. A sensitivity analysis conducted in Dhulipala et al. [69] concluded that the pre-exponential factors of the fuel kernel and PyC layer are, in comparison to the

other model parameters, unimportant in predicting fractional silver release, which is the fission product of interest herein. Hence, the parameter space of interest is $\boldsymbol{\theta} = \{Q_{kernel},\ Q_{ipyc},\ D_{sic},\ Q_{sic}\}$ when considering the Arrhenius equation for silver diffusivity.

Experimental datasets on the observed silver release from TRISO particles are available from the Department of Energy Advanced Gas Reactor program. This enables inverse calibration and UQ of the TRISO model parameter space. At the same time, it is also of interest to quantify the predictive uncertainty associated with model inadequacy and experimental noise. We used the massively parallel MCMC samplers and parallelizable active learning in MOOSE to inversely quantify the model parameters $\boldsymbol{\theta}$ and the sigma term (model inadequacy plus experimental noise). Thanks to the Advanced Gas Reactor program, 32 experimental data points on the observed silver release have been made available, and were used for the inverse UQ process [72].

The approaches to inversely assess the uncertainties in the model parameters and model inadequacy plus experimental noise are detailed below.

- *Parallel MCMC:* The TRISO fractional silver release predictions and observations are bounded between 0 and 1. So, we used a truncated normal likelihood function to assess the model predictions against the experimental data. The inversely calibrated parameters were $\{\boldsymbol{\theta}, \sigma\}$, and the prior distributions for all the parameters were uniformly distributed. We used the differential evolution sampler [41] in MOOSE to inversely quantify the uncertainties in $\{\boldsymbol{\theta}, \sigma\}$. For this purpose, we used 50 parallel chains, each executing the MOOSE model 32 times (i.e., the number of experimental data points), in parallel, to evaluate the likelihood function. As a result, $1,600$ (i.e., 50×32) processors were employed to perform inverse UQ for a total of 500 serial iterations in the differential evolution sampler.

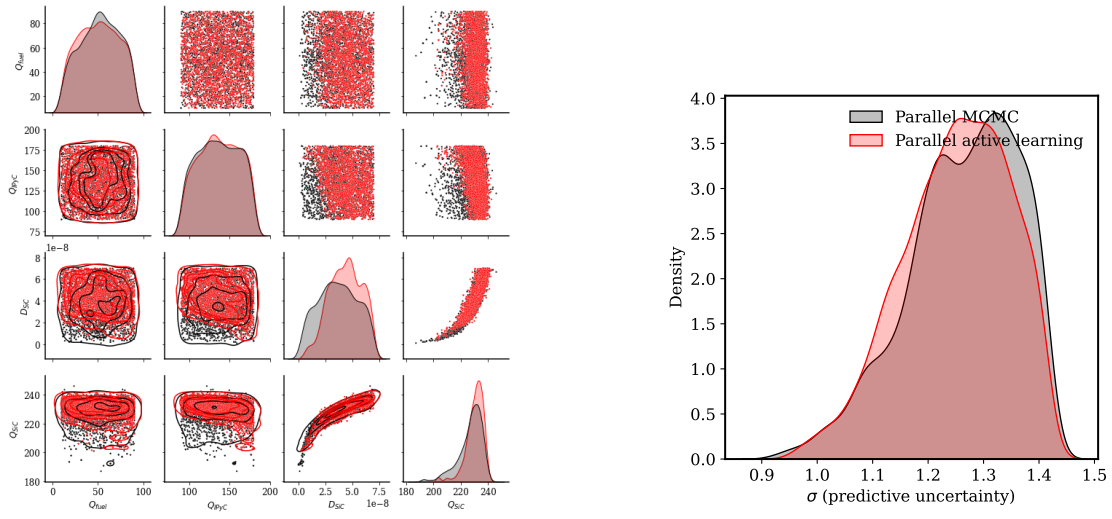
- *Parallel active learning:* For this, we used the same likelihood formulation and priors as before. We used a standard GP to predict the fractional silver release of the MOOSE model. For active learning, we relied on the Bayesian posterior targeted acquisition function from Table 1 to actively acquire new training data by running the MOOSE model. We also combined this acquisition function with the local penalization approach (Equations (20)–(21)) to acquire a batch of new training data. We set the batch size to 10 and performed 80 serial iterations of active learning. At the end of the 80 iterations, we observed that a convergence metric had sufficiently stabilized. Then, using the actively trained standard GP, we performed differential evolution sampling, just as before, by replacing the MOOSE model evaluations. This led to an approximated posterior distribution of $\{\boldsymbol{\theta}, \sigma\}$.

Figure 6a presents the inversely quantified posterior distributions of $\boldsymbol{\theta}$, comparing the parallel MCMC and parallel active learning approaches. Note that, in general, parallel active learning gives posterior distributions consistent with parallel MCMC, which is considered to be the reference solution. Between the model parameters D_{SiC} and Q_{sic}, we see a strong non-linear correlation, as shown in the subplots located in the third row, fourth column and the fourth row, third column. Parallel active learning is able to capture this non-linear correlation, though it struggles near the bottom left tip, where there is a small concentration of probability density. Figure 6b presents the posterior distribution of the sigma (σ) term, which captures the model inadequacy plus the experimental noise. Again, parallel MCMC and parallel active learning produce highly consistent results.

Figure 7 compares the computational cost of inverse UQ in regard to parallel active learning and parallel MCMC. Computational cost is measured as the product of the number of processors required times the elapsed time necessary to solve the inverse UQ problem. Parallel active learning has shown to have a computational cost at least three orders of magnitude smaller than parallel MCMC, which is considered the reference solution, while still delivering satisfactory posterior uncertainties. Capturing features in the posterior distribution like sharp tails can be accomplished by increasing the number of iteration or using a better acquisition function.

4.2. Active learning variance reduction for very rare events analysis of a heat-pipe nuclear microreactor

This section uses the active learning with GP and forward UQ capabilities discussed in Sections 2.2 and 2.4, respectively.



(a) (b)

Figure 6: Comparison of the posterior distributions of (a) the model parameters $\boldsymbol{\theta}$ and (b) the sigma term (model inadequacy plus noise) in regard to parallel MCMC and parallel active learning approaches for the TRISO fuel silver release case.

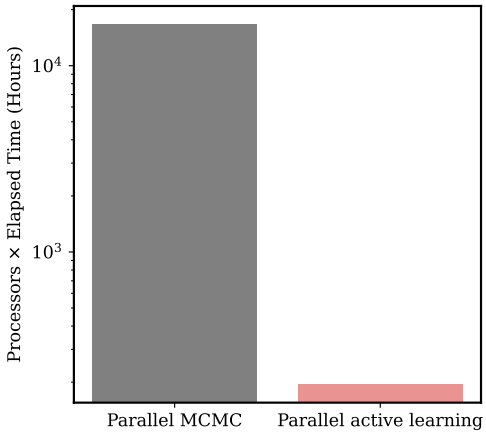


Figure 7: Comparison of the computational cost of performing inverse UQ in regard to the parallel active learning and parallel MCMC approaches for the TRISO fuel silver release case. (Computational cost is measured as the product of the number of processors required times the elapsed time necessary to solve the inverse UQ problem.)

This section demonstrates the use of MOOSE ProbML capabilities for estimating very rare events, based on an HP nuclear microreactor model. Very rare events correspond to low failure probabilities on the order of 10^{-6} or lower. Unlike other types of nuclear reactors, HP-cooled microreactors must consider additional failure modes stemming from heat transfer limitations governing HP operability. These bounding limits constrain how much heat can be removed by the HPs, depending mainly on the HP design parameters and its temperature. Failure limits are computed using the MOOSE-based application Sockeye [73], based on the design parameters specified in Terlizzi et al. [74], with the pore radius increased to 45 μm (to lower the capillary limit). Even though the sonic and viscous limits are not catastrophic—in that the HPs can recover after reaching them—for the purpose of this demonstration, all these limits are considered when determining failure probability. As manufacturing and thermal property uncertainties are very much design specific, and because the model considered herein is a prototypical design, this demonstration only serves as a proof-of-concept of MOOSE's methodological implementations for computing very low failure probabilities.

As such, the reported values of the failure probabilities should not be directly applied to assess the safety of HP reactors.

The MOOSE computational model consists of a single HP. It employs the effective heat conduction model in Sockeye [73], with the HP vapor core being represented as a material with extremely high thermal conductivity, as described in Matthews et al. [75]. Four uncertain parameters are considered: (1) Q_{evap}: the power removed by (or heating rate of) the HP; (2) T_{sink}: the sink temperature on the HP condenser; (3) htc_{sink}: the corresponding heat transfer coefficient; and (4) R_{pore}: the pore radius in the HP wick. Each of these parameters was assumed to follow normal distributions, with the means being defined consistently with what was used in Terlizzi et al. [74] (i.e., 1821 W, 900 K, 10^3 W/K/m^2, and 45 μm, respectively). The standard deviation for each parameter was arbitrarily chosen to be equal to 10% of the mean.

We used three forward UQ approaches in MOOSE to quantify the low probability of HP failure: (1) Monte Carlo, which serves as the reference solution but is computationally expensive; (2) standard subset simulation executed in a massively parallel fashion; and (3) subset simulation with active learning via a standard GP. These approaches are detailed below.

- *Monte Carlo:* We used 10^9 MOOSE model evaluations to compute the HP failure probability.
- *Standard subset simulation executed in parallel:* We used seven subsets and $20,000$ MOOSE model evaluations per subset. In each subset, we used 40 independent Markov chains, each evaluating the MOOSE model 500 times in serial. These 40 Markov chains were launched in parallel fashion. Intermediate thresholds were computed, corresponding to a probability of 0.1. In total, the MOOSE model was evaluated $140,000$ times to compute the failure probability.
- *Active learning subset simulation:* We used seven subsets and $2,000$ samples per subset. The input samples were first evaluated by using a standard GP to predict the MOOSE model output. If the GP prediction, as deemed by the U-function (see Table 1), is inadequate, only then is the MOOSE model evaluation performed. Intermediate thresholds were computed, corresponding to a probability of 0.1. Note that the number of actual MOOSE model evaluations depends on the adequacy of the GP model for each input sample. This is discussed in detail next.

Table 2 presents the failure probabilities computed using the three different approaches, along with the corresponding coefficient of variation, the total number of MOOSE model evaluations, and the required number of processors. First, note that all three methods return similar failure probability values. As the failure probability is extremely small, Monte Carlo requires an enormous number of MOOSE model evaluations. Subset simulation reduces this number by a factor of $7,000$ as compared to Monte Carlo. Active learning subset simulation reduces this number even further, by a factor of 7.7×10^6 and $1,000$ in comparison to Monte Carlo and subset simulation, respectively. Figure 8 presents the distributions of input parameters for failed HPs so as to enable further comparison of the three approaches. Note that all three return similar input parameter distributions for the failed HPs.

Table 2: Comparison of the statistics for the three forward UQ approaches in MOOSE when evaluating the failure of an HP microreactor model. Shown for reference are the number of MOOSE model evaluations and the number of required processors utilized when computing the failure probabilities.

Method	**Failure probability**	**Coefficient of variation**	**MOOSE model evaluations**	**Processors used**
Monte Carlo	7×10^{-8}	0.12	10^9	192
Parallelized subset simulation	5.1×10^{-8}	0.06	$140,000$	40
Active learning subset simulation	4.75×10^{-8}	0.192	130	1

The "MOOSE model evaluations" column represents the total number of model evaluations required.

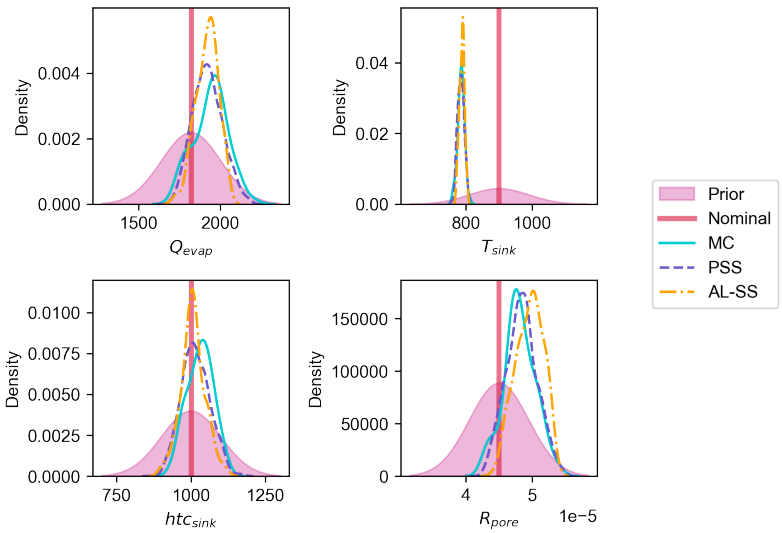


Figure 8: Distributions of the input parameters for failed HPs when comparing the three approaches: Monte Carlo (MC), parallelized subset simulation (PSS), and active learning subset simulation (AL-SS). Also shown for reference are the nominal input parameter distributions to the MOOSE HP model.

4.3. Multi-output Gaussian processes and dimensionality reduction for advanced manufacturing simulations

This section uses the MOGP and dimensionality reduction capabilities discussed in Sections 2.1.2 and 2.5, respectively.

Several advanced manufacturing techniques, including direct energy deposition and laser powder bed fusion, rely on the melting of metals with the help of a laser. The quality of the final product depends on the process parameters employed (e.g. laser power and beam radius). However, simulation of laser melt pools is challenging due to the multiple physics involved, including melting and solidification along with fluid dynamics and heat transfer in the melt pool. This is why development of surrogate models for such simulations carries high potential for accelerating parametric studies that aim to explore the relationship between process parameters and product quality. We trained an MOGP-based surrogate model combined with dimensionality reduction, using linear PCA within MOOSE to predict full temperature fields during the advanced manufacturing process [76]. The high-fidelity MOOSE model was run to gather temperature fields with different process parameters—namely, effective laser power and effective laser beam radius. The MOOSE model relied on the Arbitrary Lagrangian-Eulerian method for capturing deformations caused by the vapor pressure on the melt pool surface. Figure 9 presents the MOOSE model setup, together with the temperature distribution for a specific combination of the two process parameters.

In this work, the temperature field at a given time step was the primary quantity of interest. In total, 120 snapshots of temperature fields were collected from the high-fidelity model by varying the process parameters. LHS was employed to randomize the process parameters, using $\mathcal{U}(70, 83)$ [W] and $\mathcal{U}(125, 200)$ [μm] for the effective laser power and beam radius, respectively. Then linear PCA was applied to the temperature field snapshots for data compression. The decay of the squared singular values and the relative variance content are presented in Figure 10a. We see rapid decay in the explained variance, indicating that a few PCA components are sufficient to describe the thermal behavior of the system. Based on this information, a latent space of 10 dimensions was selected, and the temperature snapshot fields were mapped onto this space by using the first 10 components of linear PCA.

The 10 latent space components across 120 random realizations of the process parameters served as the training samples for the MOGP. The MOGP was trained via Adam optimization with 1,000 epochs, at a learning rate of 5×10^{-4}. The trained MOGP was then evaluated on a test set, using 200 samples of the process parameters. The MOGP-predicted latent quantities, which have 10 dimensions, were projected back

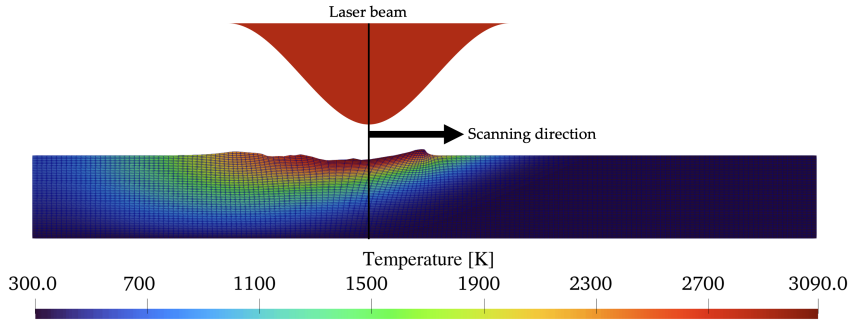
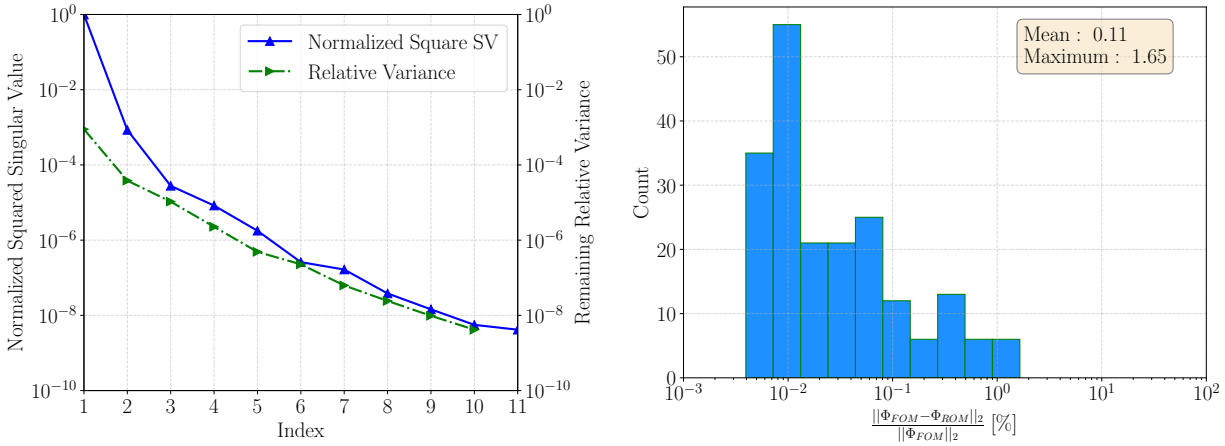


Figure 9: Temperature field output from the high-fidelity MOOSE model, which simulated the advanced manufacturing process by considering an effective laser power of $P = 81.97\ W$ and a laser radius of $R = 125.8\ \mu m$. The model relied on the Arbitrary Lagrangian-Eulerian method for capturing deformations caused by the vapor pressure on the melt pool surface.



(a) (b)

Figure 10: (a) Decay in the squared singular values of the temperature fields upon performing linear PCA. The remaining relative variance is also shown, as computed by excluding the variance of the modes up the given index. (b) Histogram of the relative L^2 errors (in %) of the temperature field between the high-fidelity model and the reconstructed solution from the MOGP by considering the testing set of 200 samples.

to the original space by using an inverse PCA. The reconstructed temperature fields were then compared against the reference temperature fields obtained by evaluating the high-fidelity MOOSE model. Figure 10b presents a histogram of the relative full-field errors (in percentages) for the testing set. Generally, these relative errors are quite small, with a mean relative error of around 0.1% and a maximum relative error of 1.65%. The maximum error occurs near the boundary of the parameter domain, which was not properly covered by the training set, thus leading to minor inaccuracies in the MOGP prediction.

The reference temperature field, along with the space-dependent absolute error between the reference and the MOGP solutions, is presented in Figure 11 for those process parameters with the highest relative error in Figure 10b. We see that the highest space-wise error is approximately 2.5%, which is acceptable for the given use case. Evaluation of the MOGP occurred 4–5 orders of magnitude faster than the solving of the transient melt pool simulation. This further reflects the high potential for accelerating parameter studies related to the product quality's dependence on process parameters, in addition to permitting active learning based on the uncertainty estimates of the MOGP.

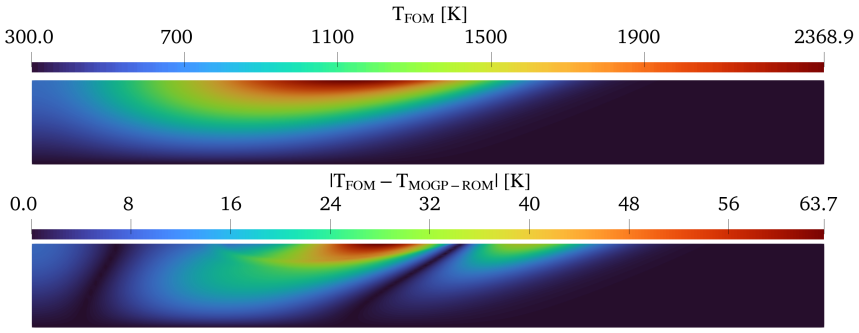


Figure 11: Comparison of solutions from the high-fidelity MOOSE model and reduced-order models at the least accurate sample in the testing set. Top: temperature profile computed using the high-fidelity MOOSE model. Bottom: absolute difference between the MOOSE model and the reconstructed MOGP solutions.

4.4. Comparing deep and standard Gaussian processes for a lid-driven cavity flow

Note that the deep Gaussian process capabilities in MOOSE are still in the experimental phase.

This section uses the GP and deep GP capabilities discussed in Sections 2.1.1 and 2.1.3, respectively.

In this section, we compare a DGP trained using MCMC against a standard GP trained using either MCMC or Adam optimization for a four-sided lid-driven cavity flow problem. The fluid domain, a 2D square region defined by viscosity and density, is subjected to velocity boundary conditions on all four sides. The pressure is set to zero at the lower-left corner. More details on the problem setup can be found in Dhulipala et al. [33]. We are interested in predicting the resultant velocity at the domain's center as a function of the viscosity and density of the fluid and of the four boundary conditions. We used the MOOSE Navier-Stokes Module to generate training and testing data under random values for the viscosity and density of the fluid and the four boundary conditions [77].

The training data were comprised of 30 points, and the testing data were comprised of 100. We first trained a standard GP by using MCMC. There were seven hyperparameters to optimize (i.e., six length scales and one amplitude scale), and we used 10,000 samples in the MCMC algorithm in order to estimate the posterior distributions of the hyperparameters. We then trained a DGP with one hidden layer using MCMC. This time, there were 43 hyperparameters; that is, six for each of six nodes in the hidden layer, plus an additional seven for the output layer. We again used 10,000 samples in the MCMC algorithm so as to estimate the posterior distributions of the hyperparameters. Finally, we trained a standard GP by using Adam optimization (giving us seven hyperparameters to optimize). The Adam optimization entailed 1,000 iterations, a learning rate of 0.005, and a batch size of 20.

We compared the three approaches for predicting the resultant velocity—namely, GP using MCMC, DGP using MCMC, and GP using Adam optimization—based on diagnostics such as parity plots, calibration curves, uncertainty distributions, and error bars, as detailed in Tran et al. [78] and Kuleshov et al. [79]. The parity plots assessed the accuracy of the predictions and presented metrics such as median absolute error, root mean squared error, mean absolute error, and mean absolute relative percent difference. Calibration curves "use the standard deviation predictions to create Gaussian random variables for each test point and then test how well the residuals followed their respective Gaussian random variables" [78]. In other words, the model is said to be well calibrated if the expected-vs.-observed cumulative distribution of the testing points follows a straight line. A well-calibrated model could still have large uncertainty estimates that are less useful in practice [78]. Thus, from the uncertainty distributions, metrics such as sharpness and coefficient of variation (C_v) are derived. Large uncertainty estimates are less desirable than small values, and sharpness assesses this by taking the root mean of the predicted variances. The model should not predict constant uncertainty estimates outside the training bounds, and C_v assesses this by computing the coefficient of variation of the predictive variances. While smaller values of the accuracy metrics, miscalibration area, and sharpness are preferred, a larger value of C_v is desirable.

Figure 12 compares the three approaches—GP using MCMC, DGP using MCMC, and GP using Adam optimization—in light of the aforementioned diagnostics. The first row corresponds to GP using MCMC, the second row to DGP using MCMC, and the third row to GP using Adam optimization. In comparing GP using MCMC against DGP using MCMC, the latter generally outperforms the former in almost every metric. DGP using MCMC has better accuracy, lower sharpness, and a larger C_v than GP using MCMC, showing the power of DGP method compared to GP. Although GP using MCMC has a smaller miscalibration area, this is likely due to it predicting constant wider uncertainty bands (as observed by comparing Figure 12d to Figure 12h) than does DGP using MCMC. As such, hidden layers help a DGP model with more expressivity and better uncertainty quality than a standard GP when trained using MCMC. In comparing DGP using MCMC against GP using Adam optimization, the latter outperforms the former in every metric. We suspect that this is largely due to the inefficiency of MCMC in high-dimensional parameter spaces in DGP and the optimization algorithm plays a big role in the predictive performance (including accuracy and uncertainty quality) of the GP models. In the future, DGP will be implemented with a more efficient variational inference and gradient-based solvers coupled with MOOSE's libtorch capabilities [68].

4.5. Batch Bayesian optimization of tritium diffusion experiment in beryllium

This section uses the active learning with GP capabilities discussed in Section 2.2.

The Tritium Migration Analysis Program, Version 8 (TMAP8) is a state-of-the-art, open-source, MOOSE-based application designed for multiscale tritium transport. TMAP8 incorporates multispecies, multiphysics, multiscale simulation capabilities on complex geometries. These capabilities make it an essential tool for the fusion energy community, particularly for addressing the challenges of tritium tracking, fusion system safety, and fuel sustainability. Validation case study val-2b in TMAP8's test suite validates against implantation and thermal absorption/desorption experiments on wafers of polished beryllium from Macaulay et al. [80]. The beryllium was exposed to 13.3 kPa of deuterium at 773 K for 50 hours, cooled down to 300 K in vacuum, and then heated back up to 1073 K at a rate of 3 K/min to desorb the deuterium. Further details are available in Simon et al. [9]. The modeled deuterium flux during desorption was compared against experimental data, as shown in Figure 13a. Herein, batch Bayesian optimization was applied to calibrate the diffusivities and solubilities of the TMAP8 model in order to improve agreement with the experimental data.

The deuterium flux model had 10 parameters, comprised of the diffusivities and solubilities that must be calibrated against the experimental data. Prior to the calibration, the model predictions and the experimental data resulted in a root mean squared percent error of 22.72%. Two approaches in MOOSE were used to achieve this calibration: parallel subset simulation, which is an evolutionary approach, and batch Bayesian optimization. The aim of the optimization with respect to the model parameters was to minimize the mean squared percentage error between the experimental data and the model predictions regarding the deuterium flux during desorption. Details on the usage of these approaches are as follows:

- **Parallel subset simulation:** Run for five subsets, with 1,000 samples per subset. Ten processors were used, simultaneously simulating five parallel chains for 1,000 serial model evaluations.
- **Batch Bayesian optimization:** Run for 80 serial iterations, with a batch of five optimal points selected in parallel in each iteration by using the expected improvement acquisition function. A standard GP with squared exponent covariance matrix was trained using Adam optimization, in which 2,000 iterations were performed using a learning rate of 0.01. The five selected optimal points were used for evaluating the computational model in parallel.

Figure 13a presents the model output against the experimental data following the parameter calibration. We see that both the parallel subset simulation and batch Bayesian optimization have similar root mean squared percent error values, and both substantially reduce this error metric in comparison to the uncalibrated model. Figure 13b presents the computational burden of the two approaches, as assessed based on the product of the number of processors and the elapsed time in hours. Ultimately, batch Bayesian optimization is revealed to be substantially lower in computational cost than parallel subset simulation.

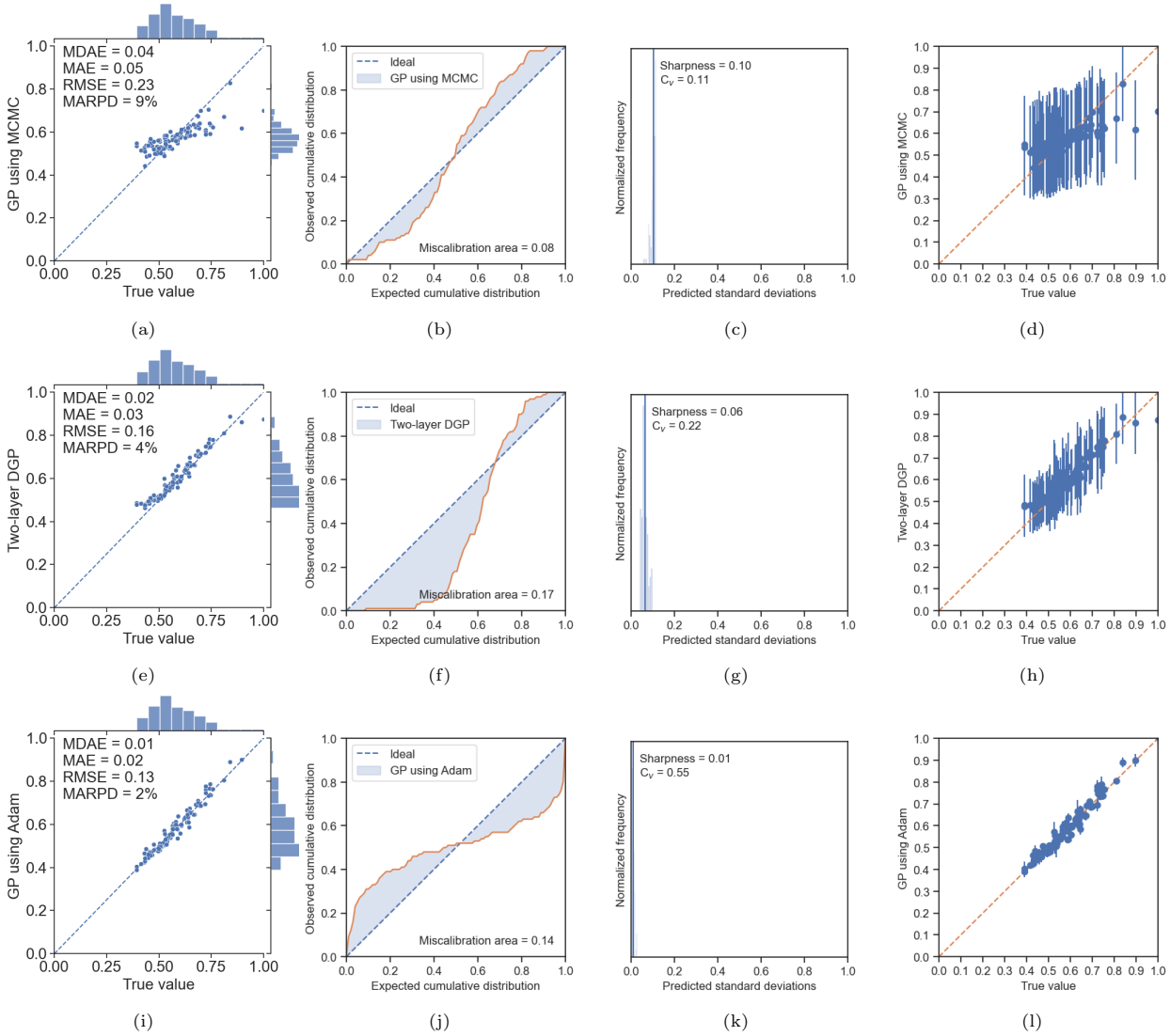
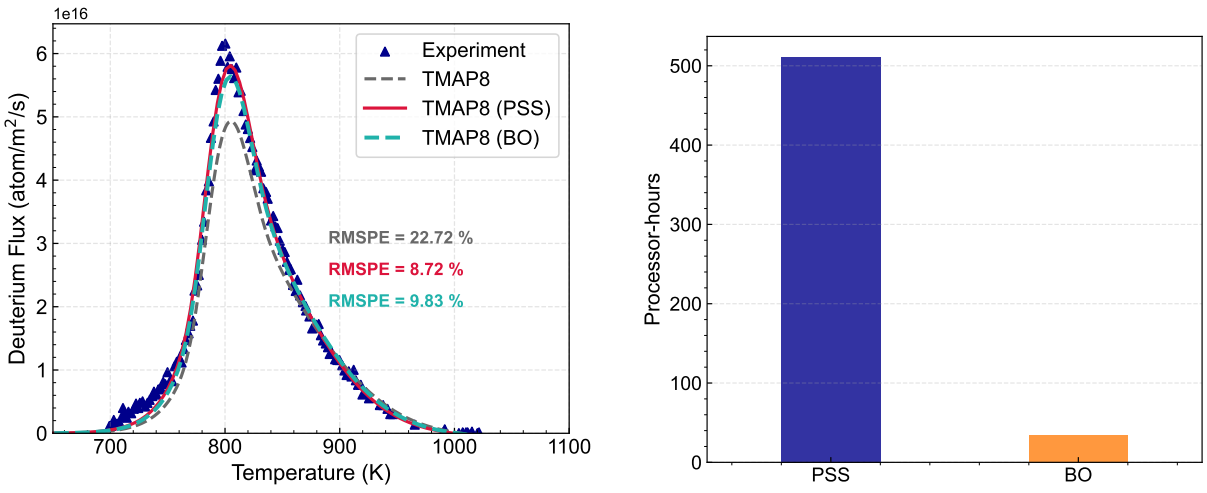


Figure 12: Predictive performance of the three GP variants in terms of both accuracy and uncertainty quality. Top row [(a)–(d)]: GP trained using MCMC. Middle row [(e)–(h)]: DGP trained using MCMC. Bottom row [(i)–(l)]: GP trained using Adam optimization. (a), (e), and (i) show the parity plots and present accuracy metrics such as median absolute error, root mean squared error, mean absolute error, and mean absolute relative percent difference. (b), (f), and (j) show the calibration plots and miscalibration area metric for uncertainty quality. (c), (g), and (k) show histograms of the predictive standard deviations, the metric sharpness, and the C_v. (d), (h), and (l) show the error bars.

5. Discussion of Future Implementations

There are several avenues for future implementations which can be made possible through the modularity of the MOOSE framework. Specifically, the modularity offered by the `Sampler`, `MultiApp`, `Reporter`, and `Surrogate` objects interaction. For example, the parallel active learning and batch Bayesian optimization are currently dependent on the local penalization approach (Equation (20)). Other approaches for batch (parallel) selection of the optimal points are also available, such as the Kriging Believer algorithm proposed by Ginsbourger et al. [81]. Wang et al. [82] provide a review of the recent developments in batch selection. These approaches will be pursued in MOOSE in the future. Similarly, optimization-based Bayesian inference approaches like variational inference can be implemented to mitigate the high computational cost of



(a) (b)

Figure 13: (a) Modeled deuterium flux during desorption, compared against experimental data. Before calibrating the model parameters, the model had a root mean squared percent error of 22.72% when examined against the experimental data. Upon calibration based on parallel subset simulation (an evolutionary approach) and batch Bayesian optimization, the root mean squared percent error reduced to 8.72% and 9.83%, respectively. (b) The computational cost of calibrating the model parameters via parallel subset simulation and Bayesian optimization was measured as the product of the number of processors and the elapsed time (in hours).

MCMC methods, but with a trade off for accuracy [83]. Methods like black-box variational inference are particularly attractive given that they do not require gradient estimations of the model outputs [84]. Also, while MOOSE currently supports linear dimensionality reduction via PCA, nonlinear methods like kernel PCA, diffusion maps, and manifold learning methods can be pursued in the future [85].

One significant capability of MOOSE which can be expanded upon from a ML/UQ standpoint is its `libtorch` integration [67, 68]. The `libtorch` library provides PyTorch like functionalities from within C++ and supports the training and evaluation of more sophisticated ML models like deep neural networks and operator learning networks. These expressive ML models can be integrated into active learning frameworks to mitigate the limitations of GPs, especially when dealing with high-dimensional data. Furthermore, the `TorchScript` capability in MOOSE also allows importing of ML models trained in PyTorch into C++ code.

6. Summary and Conclusions

MOOSE, an open-source computational platform for parallel numerical analysis, is being actively developed and is maintained at Idaho National Laboratory. MOOSE has an extensive user base in varied scientific and engineering fields. Complex multiphysics simulations, when validated against experimental data, are subject to different sources of uncertainties that must be quantified and propagated to the outputs. They are also computationally expensive to run, especially in a UQ setting, and surrogate models for quantifying their prediction uncertainties will foster their efficient and accurate execution by leveraging active learning principles. In this context, the present paper covered the development and demonstration of massive parallel probabilistic ML and UQ capabilities in MOOSE. Among these capabilities are active learning, Bayesian inverse UQ, adaptive forward UQ, Bayesian optimization, evolutionary optimization, and MCMC. The MOOSE systems `Sampler`, `MultiApp`, `Reporter`, and `Surrogate`, as well as the modularity thereof, were discussed in detail in regard to successfully developing a multitude of probabilistic ML and UQ algorithms. Example code demonstrations include parallel active learning and parallel Bayesian inference via active learning. Finally, the impacts of these code developments were discussed in regard to five

different applications: nuclear fuel fission product release, using parallel active learning Bayesian inference; nuclear microreactor very rare events analysis, using active learning; advanced manufacturing process modeling, using MOGP and dimensionality reduction; lid-driven cavity flow, using DGPs; and tritium transport for fusion energy, using batch Bayesian optimization. These capabilities are part of the MOOSE framework.

Acknowledgements

The forward UQ capability developments, including active learning and multifidelity modeling for forward problems, are supported through Idaho National Laboratory (INL)'s Laboratory Directed Research & Development (LDRD) Program under U.S. Department of Energy (DOE) Idaho Operations Office Contract DE-AC07-05ID14517.

The Bayesian inverse UQ capability developments, including active learning for inverse problems, are supported through Battelle Energy Alliance, LLC under contract no. DE-AC07-05ID14517 with DOE, along with funding from the Nuclear Energy Advanced Modeling and Simulation (NEAMS) program within the DOE Office of Nuclear Energy (DOE-NE).

The multi-output Gaussian processes and dimensionality reduction capability developments are supported through Battelle Energy Alliance, LLC under contract no. DE-AC07-05ID14517 with DOE, with funding from the Advanced Materials and Manufacturing Technologies (AMMT) program within DOE-NE.

The deep Gaussian processes and Bayesian optimization capability developments are supported through Battelle Energy Alliance, LLC under contract no. DE-AC07-05ID14517 with DOE, along with funding from the Nuclear Energy University Partnerships (NEUP) program within DOE-NE.

This research made use of resources of the High-Performance Computing Center at INL, which is supported by DOE-NE and the Nuclear Science User Facilities under contract no. DE-AC07-05ID14517.

We thank the following individuals for their support in developing the capabilities of the MOOSE Stochastic Tools Module: Stephen R. Novascone, Sudipta Biswas, Benjamin W. Spencer, Jason D. Hales, and Daniel Schwen from Idaho National Laboratory; Michael D. Shields and Promit Chakroborty from Johns Hopkins University; and Andi Wang from the University of Wisconsin-Madison. We thank John Shaver at INL for his technical edit of this paper.

Declaration of generative AI and AI-assisted technologies in the writing process

The authors did not use generative AI technologies during the initial writing process or editing of this paper. During the revisions stage, the authors used the ChatGPT 5 Thinking model in order to improve the readability and language of the manuscript. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

References

[1] G. Giudicelli, A. Lindsay, L. Harbour, C. Icenhour, M. Li, J. E. Hansel, P. German, P. Behne, O. Marin, R. H. Stogner, J. Miller, 3.0-MOOSE: Enabling massively parallel multiphysics simulations, SoftwareX 26 (2024) 101690. doi:10.1016/j.softx.2024.101690.

[2] R. L. Williamson, J. D. Hales, S. R. Novascone, G. Pastore, K. A. Gamble, B. W. Spencer, W. Jiang, S. A. Pitts, A. Casagranda, D. Schwen, A. X. Zabriskie, BISON: A flexible code for advanced simulation of the performance of multiple nuclear fuel forms, Nuclear Technology 207 (7) (2021) 954–980. doi:10.1080/00295450.2020.1836940.

[3] B. W. Spencer, W. M. Hoffman, S. Biswas, W. Jiang, A. Giorla, M. A. Backman, Grizzly and BlackBear: Structural component aging simulation codes, Nuclear Technology 207 (2021) 981–1003. doi:10.1080/00295450.2020.1868278.

[4] A. J. Novak, R. W. Carlsen, S. Schunert, P. Balestra, D. Reger, R. N. Slaybaugh, R. C. Martineau, Pronghorn: A multidimensional coarse-mesh application for advanced reactor thermal hydraulics, Nuclear Technology 207 (2021) 1015–1046. doi:10.1080/00295450.2020.1825307.

[5] Y. Wang, Z. M. Prince, H. Park, O. W. Calvin, N. Choi, Y. S. Jung, S. Schunert, S. Kumar, J. T. Hanophy, V. M. Labouré, C. Lee, Griffin: A MOOSE-based reactor physics application for multiphysics simulation of advanced nuclear reactors, Annals of Nuclear Energy 211 (2025) 110917. doi:10.1016/j.anucene.2024.110917.

[6] S. Veeraraghavan, C. Bolisetti, A. Slaughter, J. Coleman, S. L. N. Dhulipala, W. Hoffman, K. Kim, E. Kurt, R. Spears, L. Munday, MASTODON: an open-source software for seismic analysis and risk assessment of critical infrastructure, Nuclear Technology 207 (2021) 1073–1095. doi:10.1080/00295450.2020.1807282.

[7] M. R. Tonks, D. Gaston, P. C. Millett, D. Andrs, P. Talbot, An object-oriented finite element framework for multiphysics phase field simulations, Computational Materials Science 51 (2012) 20–29. doi:10.1016/j.commatsci.2011.07.028.
[8] A. J. Novak, D. Andrs, P. Shriwise, J. Fang, H. Yuan, D. Shaver, E. Merzari, P. K. Romano, R. Martineau, Coupled Monte Carlo and thermal-fluid modeling of high temperature gas reactors using Cardinal, Annals of Nuclear Energy 177 (2022) 109310. doi:10.1016/j.anucene.2022.109310.
[9] P. C. A. Simon, C. T. Icenhour, G. Singh, A. D. Lindsay, C. Bhave, L. Yang, A. Riet, Y. Che, P. Humrickhouse, P. Calderoni, M. Shimada, MOOSE-based tritium migration analysis program, version 8 (TMAP8) for advanced open-source tritium transport and fuel cycle modeling, Fusion Engineering and Design 214 (2025) 114874. doi:10.1016/J.FUSENGDES.2025.114874.
[10] P. R., A. Finnila, S. Simmons, J. McLennan, A Reference Thermal-Hydrologic-Mechanical Native State Model of the Utah FORGE Enhanced Geothermal Site, Energies 14 (2021) 4758. doi:10.3390/en14164758.
[11] A. E. Slaughter, Z. M. Prince, P. German, I. Halvic, W. Jiang, B. W. Spencer, S. L. N. Dhulipala, D. R. Gaston, MOOSE Stochastic Tools: A module for performing parallel, memory-efficient in situ stochastic simulations, SoftwareX 22 (2023) 101345. doi:10.1016/j.softx.2023.101345.
[12] D. Tsapetis, M. D. Shields, D. G. Giovanis, A. Olivier, L. Novak, P. Chakroborty, H. Sharma, M. Chauhan, K. Kontolati, L. Vandanapu, D. Loukrezis, Uqpy v4. 1: Uncertainty quantification with Python, SoftwareX 24 (2023) 101561. doi:10.1016/j.softx.2023.101561.
[13] N. A. Riis, A. M. Alghamdi, F. Uribe, S. L. Christensen, B. M. Afkham, P. C. Hansen, J. S. Jørgensen, CUQIpy: I. Computational uncertainty quantification for inverse problems in Python, Inverse Problems 40 (2024) 045009. doi:10.1088/1361-6420/ad22e7.
[14] M. Parno, A. Davis, L. Seelinger, MUQ: The MIT uncertainty quantification library, Journal of Open Source Software 6 (2021) 3076. doi:10.21105/joss.03076.
[15] J. D. Jakeman, PyApprox: A software package for sensitivity analysis, Bayesian inference, optimal experimental design, and multi-fidelity uncertainty quantification and surrogate modeling, Environmental Modelling & Software 170 (2023) 105825. doi:10.1016/j.envsoft.2023.105825.
[16] L. Seelinger, A. Reinarz, M. B. Lykkegaard, R. Akers, A. M. Alghamdi, D. Aristoff, W. Bangerth, J. Bénézech, M. Diez, K. Frey, J. D. Jakeman, Democratizing uncertainty quantification, Journal of Computational Physics 521 (2025) 113542. doi:10.1016/j.jcp.2024.113542.
[17] C. K. Williams, C. E. Rasmussen, Gaussian processes for machine learning, 2nd Edition, MIT press, 2006.
[18] H. Liu, J. Cai, Y. S. Ong, Remarks on multi-output Gaussian process regression, Knowledge-Based Systems 144 (2018) 102–112. doi:10.1016/j.knosys.2017.12.034.
[19] M. A. Alvarez, L. Rosasco, N. D. Lawrence, Kernels for vector-valued functions: A review, Foundations and Trends in Machine Learning 4 (3) (2012) 195–266. doi:10.1561/2200000036.
[20] L. F. Cheng, B. Dumitrascu, G. Darnell, C. Chivers, M. Draugelis, K. Li, B. E. Engelhardt, Sparse multi-output Gaussian processes for online medical time series prediction, BMC medical informatics and decision making 20 (1) (2020) 1–23. doi:10.1186/s12911-020-1069-4.
[21] A. Damianou, N. D. Lawrence, Deep Gaussian processes, in: Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, Scottsdale, AZ United States, 2013, pp. 207–215.
[22] A. Damianou, Deep Gaussian processes and variational propagation of uncertainty, Doctoral dissertation, University of Sheffield (2015).
[23] A. Sauer, R. B. Gramacy, D. Higdon, Active learning for deep Gaussian process surrogates, Technometrics 65 (1) (2023) 4–18. doi:10.1080/00401706.2021.2008505.
[24] H. Salimbeni, M. Deisenroth, Doubly stochastic variational inference for deep Gaussian processes, in: Advances in Neural Information Processing Systems, Long Beach, CA United States, 2017, pp. 1–12.
[25] Z. Dai, A. Damianou, J. Hensman, N. Lawrence, Gaussian process models with parallelization and GPU acceleration, arXiv:1410.4984 (2014). arXiv:1410.4984.
[26] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv:1711.05101 (2014). arXiv:1412.6980.
[27] I. Loshchilov, F. Hutter, Decoupled weight decay regularization, arXiv:1711.05101 (2017). arXiv:1711.05101.
[28] I. Murray, R. Adams, D. MacKay, Elliptical slice sampling, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 2010, pp. 541–548.
[29] C. Chen, J. Liu, P. Xu, Comparison of parallel infill sampling criteria based on kriging surrogate model, Scientific Reports 12 (1) (2022) 678. doi:10.1137/16M1082469.
[30] E. Contal, D. Buffoni, A. Robicquet, N. Vayatis, Parallel Gaussian process optimization with upper confidence bound and pure exploration, Springer Berlin Heidelberg, 2013.
[31] J. El Gammal, N. Schöneberg, J. Torrado, C. Fidler, Fast and robust Bayesian inference using Gaussian processes with GPry, Journal of Cosmology and Astroparticle Physics 2023 (2023) 021. doi:10.1088/1475-7516/2023/10/021.
[32] B. Echard, N. Gayton, M. Lemaire, AK-MCS: an active learning reliability method combining Kriging and Monte Carlo simulation, Structural Safety 33 (2) (2011) 145–154. doi:10.1016/j.strusafe.2011.01.002.
[33] S. L. N. Dhulipala, M. D. Shields, B. W. Spencer, C. Bolisetti, A. E. Slaughter, V. M. Labouré, P. Chakroborty, Active learning with multifidelity modeling for efficient rare event simulation, Journal of Computational Physics 468 (2022) 111506. doi:10.1016/j.jcp.2022.111506.
[34] C. Q. Lam, Sequential adaptive designs in computer experiments for response surface model fit, Doctoral dissertation, The Ohio State University (2008).
[35] D. Zhan, J. Qian, Y. Cheng, Pseudo expected improvement criterion for parallel EGO algorithm, Journal of Global Optimization 68 (2017) 641–662. doi:10.1007/s10898-016-0484-7.

[36] M. C. Kennedy, A. O'Hagan, Bayesian calibration of computer models, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 63 (3) (2001) 425–464. doi:10.1111/1467-9868.00294.
[37] P. D. Arendt, D. W. Apley, W. Chen, Quantification of Model Uncertainty: Calibration, Model Discrepancy, and Identifiability, ASME Journal of Mechanical Design 134 (10) (2012) 100908. doi:10.1115/1.4007390.
[38] X. Wu, T. Kozlowski, H. Meidani, K. Shirvan, Inverse uncertainty quantification using the modular Bayesian approach based on Gaussian process, Part 1: Theory, Nuclear Engineering and Design 335 (2018) 339–355. doi:10.1016/j.nucengdes.2018.06.004.
[39] M. I. Radaideh, K. Borowiec, T. Kozlowski, Integrated framework for model assessment and advanced uncertainty quantification of nuclear computer codes under bayesian statistics, Reliability Engineering & System Safety 189 (2019) 357–377. doi:10.1016/j.ress.2019.04.020.
[40] P. Robbe, D. Andersson, L. Bonnet, T. A. Casey, M. D. Cooper, C. Matthews, K. Sargsyan, H. N. Najm, Bayesian calibration with summary statistics for the prediction of xenon diffusion in UO2 nuclear fuel, Computational Materials Science 225 (2023) 112184. doi:10.1016/j.commatsci.2023.112184.
[41] C. J. T. Braak, A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces, Statistics and Computing 16 (2006) 239–249. doi:10.1007/s11222-006-8769-1.
[42] J. Goodman, J. Weare, Ensemble samplers with affine invariance, Communications in applied mathematics and computational science 5 (1) (2010) 65–80. doi:10.2140/camcos.2010.5.65.
[43] B. Calderhead, A general construction for parallelizing Metropolis-Hastings algorithms, Proceedings of the National Academy of Sciences 111 (49) (2014) 17408–17413. doi:10.1073/pnas.1408184111.
[44] B. Nelson, E. B. Ford, M. J. Payne, Run DMC: an efficient, parallel code for analyzing radial velocity observations using n-body integrations and differential evolution Markov chain Monte Carlo, The Astrophysical Journal Supplement Series 11 (2013) 11–25. doi:10.1088/0067-0049/210/1/11.
[45] S. Dhulipala, D. Schwen, Y. Che, R. Sweet, A. Toptan, Z. M. Prince, P. German, S. R. Novascone, Massively parallel Bayesian model calibration and uncertainty quantification with applications to nuclear fuels and materials, Tech. Rep. INL/RPT-23-73383-Rev000, Idaho National Laboratory, Idaho Falls, ID United States (2023).
[46] E. Laloy, J. A. Vrugt, High-dimensional posterior exploration of hydrologic models using multiple-try DREAM (ZS) and high-performance computing, Water Resources Research 48 (1). doi:10.1029/2011WR010608.
[47] D. Foreman-Mackey, W. M. Farr, M. Sinha, A. M. Archibald, D. W. Hogg, J. S. Sanders, J. Zuntz, P. K. Williams, A. R. Nelson, M. de Val-Borro, T. Erhardt, emcee v3: A Python ensemble sampling toolkit for affine-invariant MCMC, arXiv:1911.07688 (2019). arXiv:1911.07688.
[48] K. R. Opara, J. Arabas, Differential Evolution: A survey of theoretical analyses, Swarm and Evolutionary Computation 44 (2019) 546–558. doi:10.1016/j.swevo.2018.06.010.
[49] D. Vats, C. Knudson, Revisiting the Gelman–Rubin diagnostic, Statistical Science 36 (4) (2021) 518–529. doi:10.1214/20-STS812.
[50] S. L. N. Dhulipala, W. Jiang, B. W. Spencer, J. D. Hales, M. D. Shields, A. E. Slaughter, Z. M. Prince, V. M. Labouré, C. Bolisetti, P. Chakroborty, Accelerated statistical failure analysis of multifidelity TRISO fuel models, Journal of Nuclear Materials 563 (2022) 153604. doi:10.1016/j.jnucmat.2022.153604.
[51] S. K. Au, J. L. Beck, A new adaptive importance sampling scheme for reliability calculations, Structural safety 21 (2) (1999) 135–158. doi:10.1016/S0167-4730(99)00014-4.
[52] H. Zhao, Z. Yue, Y. Liu, Z. Gao, Y. Zhang, An efficient reliability method combining adaptive importance sampling and Kriging metamodel, Applied Mathematical Modelling 39 (7) (2015) 1853–1866. doi:10.1016/j.apm.2014.10.015.
[53] J. Zhang, M. Xiao, L. Gao, S. Chu, A combined projection-outline-based active learning Kriging and adaptive importance sampling method for hybrid reliability analysis with small failure probabilities, Computer Methods in Applied Mechanics and Engineering 344 (2019) 13–33. doi:10.1016/j.cma.2018.10.003.
[54] R. Kawai, Adaptive importance sampling and control variates, Journal of Mathematical Analysis and Applications 483 (1) (2020) 123608. doi:10.1016/j.jmaa.2019.123608.
[55] A. Kebaier, J. Lelong, Coupling importance sampling and multilevel Monte Carlo using sample average approximation, Methodology and Computing in Applied Probability 20 (2018) 611–641. doi:10.1007/s11009-017-9579-y.
[56] B. Peherstorfer, T. Cui, Y. Marzouk, K. Willcox, Multifidelity importance sampling, Computer Methods in Applied Mechanics and Engineering 300 (2016) 490–509. doi:10.1016/j.cma.2015.12.002.
[57] S. K. Au, J. L. Beck, Estimation of small failure probabilities in high dimensions by subset simulation, Probabilistic Eng. Mech. 16 (4) (2001) 263–277. doi:10.1016/S0266-8920(01)00019-4.
[58] H. S. Li, S. K. Au, Design optimization using subset simulation algorithm, Structural Safety 32 (6) (2010) 384–392. doi:10.1016/j.strusafe.2010.03.001.
[59] J. Bect, L. Li, E. Vazquez, Bayesian subset simulation, SIAM/ASA Journal on Uncertainty Quantification 5 (2017) 762–786. doi:10.1137/16M1078276.
[60] Y. Zhao, Z. Wang, Subset simulation with adaptable intermediate failure probability for robust reliability analysis: An unsupervised learning-based approach, Structural and Multidisciplinary Optimization 65 (2022) 172. doi:10.1007/s00158-022-03260-7.
[61] I. Papaioannou, W. Betz, K. Zwirglmaier, D. Straub, MCMC algorithms for subset simulation, Probabilistic Engineering Mechanics 41 (2015) 89–103. doi:10.1016/j.probengmech.2015.06.006.
[62] Z. Wang, M. Broccardo, J. Song, Hamiltonian Monte Carlo methods for subset simulation in reliability analysis, Structural Safety 76 (2019) 51–67. doi:10.1016/j.strusafe.2018.05.005.
[63] M. D. Shields, D. G. Giovanis, V. S. Sundar, Subset simulation for problems with strongly non-Gaussian, highly anisotropic, and degenerate distributions, Computers & Structures 245 (2021) 106431. doi:10.1016/j.compstruc.2020.106431.

[64] S. Wold, K. Esbensen, P. Geladi, Principal component analysis, Chemometrics and Intelligent Laboratory Systems 2 (1-3) (1987) 37–52. doi:10.1016/0169-7439(87)80084-9.
[65] V. Hernandez, J. E. Roman, V. Vidal, Slepc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software (TOMS) 31 (3) (2005) 351–362.
[66] D. R. Gaston, C. J. Permann, J. W. Peterson, A. E. Slaughter, D. Andrš, Y. Wang, M. P. Short, D. M. Perez, M. R. Tonks, J. Ortensi, L. Zou, R. C. Martineau, Physics-based multiscale coupling for full core nuclear reactor simulation, Annals of Nuclear Energy 84 (2015) 45–54. doi:10.1016/j.anucene.2014.09.060.
[67] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, Automatic differentiation in PyTorch, in: Proceedings of the 31st Conference on Neural Information Processing Systems, Long Beach, CA, 2017, pp. 1–4.
[68] P. German, D. Yushu, Enabling scientific machine learning in MOOSE using Libtorch, SoftwareX 23 (2023) 101489. doi:10.1016/j.softx.2023.101489.
[69] S. L. N. Dhulipala, A. Toptan, Y. Che, D. Schwen, R. T. Sweet, J. D. Hales, S. R. Novascone, Bayesian uncertainty quantification of tristructural isotropic particle fuel silver release: Decomposing model inadequacy plus experimental noise and parametric uncertainties, Journal of Nuclear Materials 588 (2024) 154790. doi:10.1016/j.jnucmat.2023.154790.
[70] D. A. Petti, P. A. Demkowicz, J. T. Maki, Triso-coated particle fuel performance, Comprehensive Nuclear Materials 3 (2012) 151–213. doi:10.1016/B978-0-08-056033-5.00055-0.
[71] J. D. Hales, W. Jiang, A. Toptan, K. A. Gamble, Modeling fission product diffusion in triso fuel particles with bison, Journal of Nuclear Materials 548 (2021) 152840. doi:10.1016/j.jnucmat.2021.152840.
[72] J. D. Stempien, R. N. Morris, T. J. Gerczak, P. A. Demkowicz, AGR-2 TRISO fuel post-irradiation examination final report, Tech. rep., Idaho National Laboratory, INL/EXT-21-64279 (2021).
URL https://www.osti.gov/biblio/1822447
[73] J. E. Hansel, R. A. Berry, D. Andrs, M. S. Kunick, R. C. Martineau, Sockeye: A one-dimensional, two-phase, compressible flow heat pipe application, Nuclear Technology 207 (7) (2021) 1096–1117. doi:10.1080/00295450.2020.1861879.
[74] S. Terlizzi, V. Labouré, Asymptotic hydrogen redistribution analysis in Yttrium-Hydride-moderated heat-pipe-cooled microreactors using DireWolf, Annals of Nuclear Energy 186. doi:10.1016/j.anucene.2023.109735.
[75] C. Matthews, V. Laboure, M. DeHart, J. Hansel, D. Andrs, Y. Wang, J. Ortensi, R. C. Martineau, Coupled multiphysics simulations of heat pipe microreactors using DireWolf, Nuclear Technology 207 (7) (2021) 1142–1162. doi:10.1080/00295450.2021.1906474.
[76] S. Biswas, S. L. N. Dhulipala, P. German, A. M. Jokisaari, D. Yushu, M. D. McMurtrey, Multiscale And Machine Learning Modeling For Process-informed Microstructure Prediction In Additively Manufactured Materials Using MALAMUTE, Tech. Rep. INL/RPT-24-80418, Idaho National Laboratory, Idaho Falls, ID United States (2024).
[77] J. W. Peterson, A. D. Lindsay, F. Kong, Overview of the incompressible navier–stokes simulation capabilities in the moose framework, Advances in Engineering Software 119 (2018) 68–92. doi:10.1016/j.advengsoft.2018.02.004.
[78] K. Tran, W. Neiswanger, J. Yoon, Q. Zhang, E. Xing, Z. W. Ulissi, Methods for comparing uncertainty quantifications for material property predictions, Machine Learning: Science and Technology 1 (2) (2020) 025006. doi:10.1088/2632-2153/ab7e1a.
[79] V. Kuleshov, N. Fenner, S. Ermon, Elliptical slice sampling, in: Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 2018, pp. 2796–2804.
[80] R. G. Macaulay-Newcombe, D. A. Thompson, W. W. Smeltzer, Deuterium diffusion, trapping and release in ion-implanted beryllium, Fusion Engineering and Design 18 (1991) 419–424. doi:10.1016/0920-3796(91)90158-M.
[81] D. Ginsbourger, R. Le Riche, L. Carraro, Kriging is well-suited to parallelize optimization, Springer Berlin Heidelberg, 2010.
[82] X. Wang, Y. Jin, S. Schmitt, M. Olhofer, Recent advances in Bayesian optimization, ACM Computing Surveys 55 (13s) (2023) 1–36. doi:10.1145/3582078.
[83] D. M. Blei, A. Kucukelbir, J. D. McAuliffe, Variational inference: A review for statisticians, Journal of the American statistical Association 112 (518) (2017) 859–877. doi:10.1080/01621459.2017.1285773.
[84] R. Ranganath, S. Gerrish, D. Blei, Black box variational inference, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, Reykjavik, Iceland, 2014, pp. 814–822.
[85] K. Kontolati, D. Loukrezis, D. G. Giovanis, L. Vandanapu, M. D. Shields, A survey of unsupervised learning methods for high-dimensional uncertainty quantification in black-box-type problems, Journal of Computational Physics 464 (2022) 111313. doi:10.1016/j.jcp.2022.111313.