

MOOSE Stochastic Tools Module

Stochastic Sampling, Surrogate modeling, and Optimization for MOOSE-
Based Modeling and Simulation

Zachary M. Prince

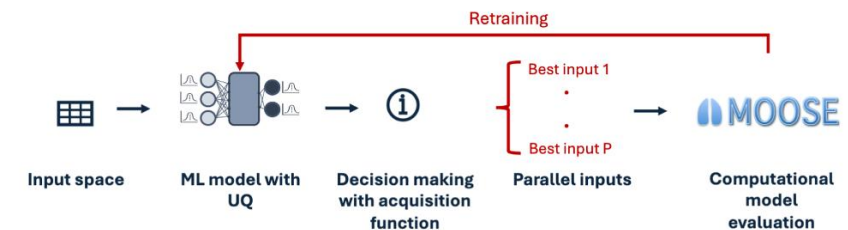
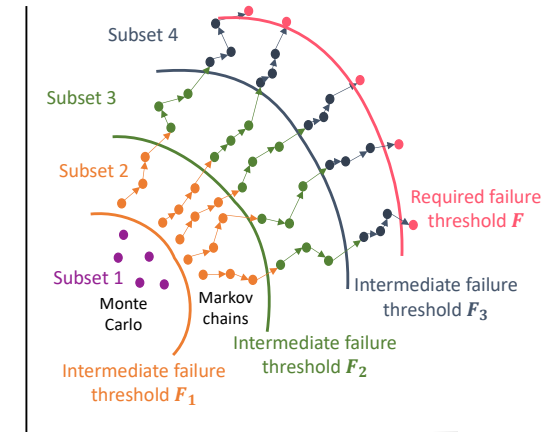
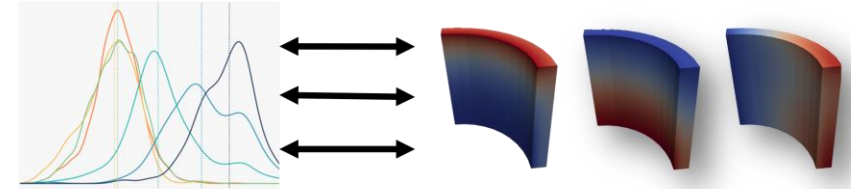


U.S. DEPARTMENT
of **ENERGY**

Office of
Nuclear Energy

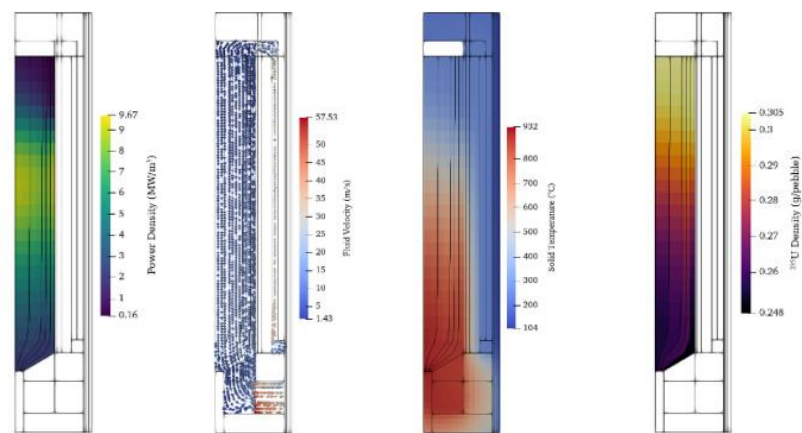
What is the MOOSE-STM?

- An open-source MOOSE module available to all MOOSE-based applications.
- Provide a **MOOSE-like interface** for performing stochastic analysis on MOOSE-based models.
- Sample parameters, run applications, and gather data that is both **efficient** (memory and runtime) and **scalable**.
- Perform UQ and sensitivity analysis with **distributed data** and leveraging **advanced variance reduction** methods.
- Train meta-models to build fast-evaluating **surrogates** of the high-fidelity multiphysics model and provide a **pluggable** interface for these surrogates.
- Harness machine learning capabilities through the C++ front end of PyTorch, i.e. **LibTorch**.
- Use **active learning** models for building surrogates.



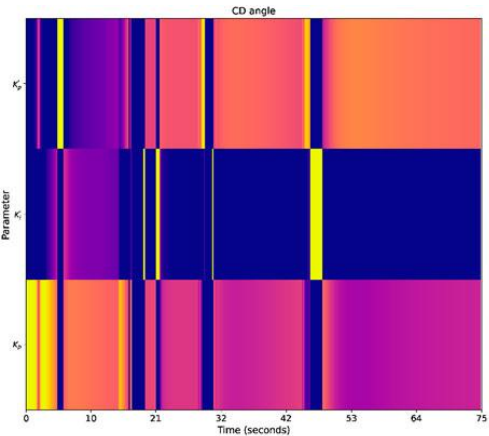
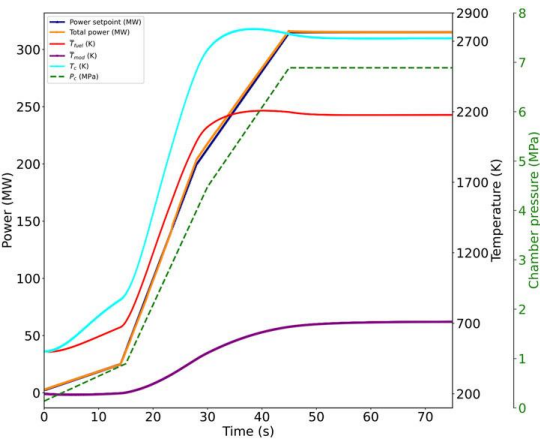
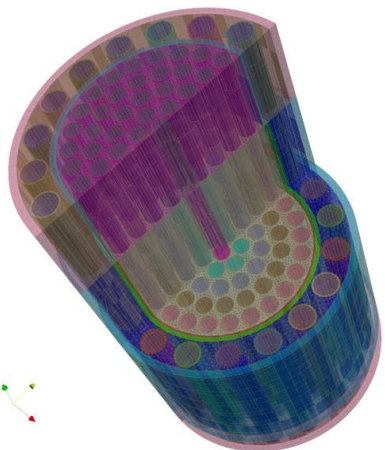
Applications of the MOOSE-STM

Design optimization of PBRs using surrogates

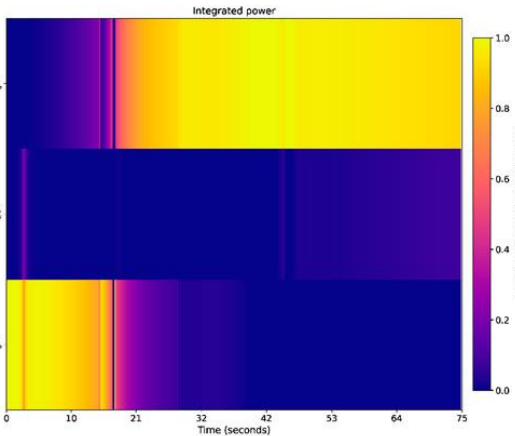


Parameter	Nominal	Full model	PR	GP	ANN
Kernel radius	0.2125	0.28117	0.17307	0.22652	0.17988
Filling factor	9.344	5.9473	12.707	8.5034	10.389
Enrichment	15.5	16.505	17.554	16.565	16.973
Feed rate	1.5	1.758	2.1557	2.955	1.1119
Burnup limit	147.6	157.64	163.97	163.75	163.86
Total power	200	219.97	219.95	219.53	219.56
Core radius	1.2	1.2754	1.2512	1.2635	1.2873
Core height	8.93	9.166	9.9708	9.1133	9.1428
Quantity	Full order evaluation		Surrogate model evaluation with relative error		
k_{eff}	0.99961	1.0281	1.0146 + 0.0172%	1.0081 - 0.296%	1.004 - 0.514%
Max pebble power	2.6742	2.6981	2.5166 + 0.0887%	2.6054 + 0.718%	2.6764 + 1.63%
Peaking factor	2.0145	1.9804	2.1305 - 1.03%	1.9303 + 1.57%	2.0864 + 0.493%
Fissile Pu fraction	63.158	61.012	65.267 - 0.0865%	63.69 - 0.66%	60.569 - 0.429%
^{235}U utilization	984.18	970.77	980.57 - 1.23%	1020.8 - 0.497%	1028.2 - 0.223%
Max operating fuel temp	1006.2	1099.5	1041.2 + 0.23%	1063.3 + 1.79%	1080 + 0.695%
Max DLOFC fuel temp	1438.4	1474.9	1422.3 + 0.301%	1454.5 - 0.247%	1515.5 + 0.357%
Max DLOFC RPV temp	321.97	337.84	327.06 + 0.121%	334.46 - 0.118%	347.86 - 0.217%
Cost metric	Value (million USD)				
Revenue	2994	3293	3292	3286	3287
Fabrication	250.4	262.2	244	247.9	240.2
Enrichment	98.97	111.2	110.8	105.6	105.1
Conversion	8.941	9.977	9.884	9.47	9.405
Source	59.15	66	65.39	62.65	62.22
Total cost	417.4	449.3	430.1 + 1.24%	425.6 + 0.499%	417 + 0.223%
Profit	2576	2843	2862 - 0.187%	2861 - 0.0743%	2870 - 0.0324%

Sensitivity analysis of NTP startup



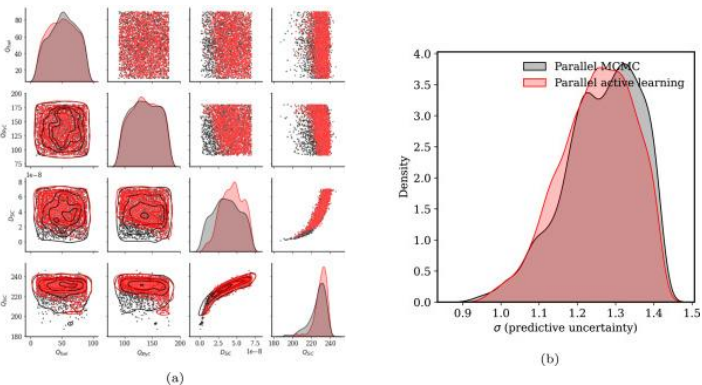
(a)



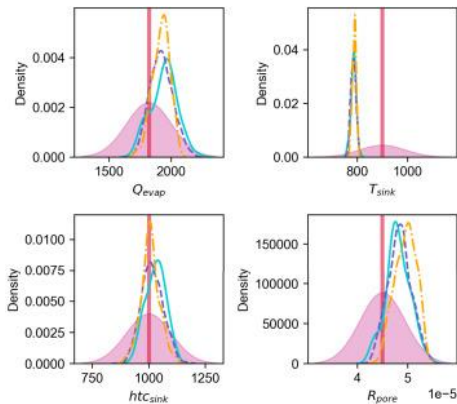
(b)

Applications of the MOOSE-STM (cont.)

Inverse UQ of TRISO fission product release with BISON

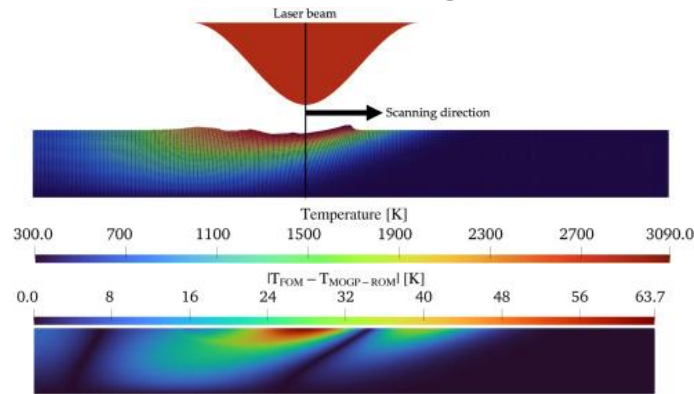


Active learning variance reduction for very rare events analysis with Sockeye

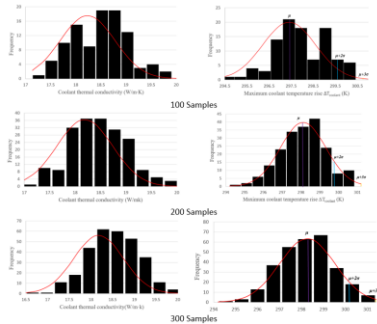
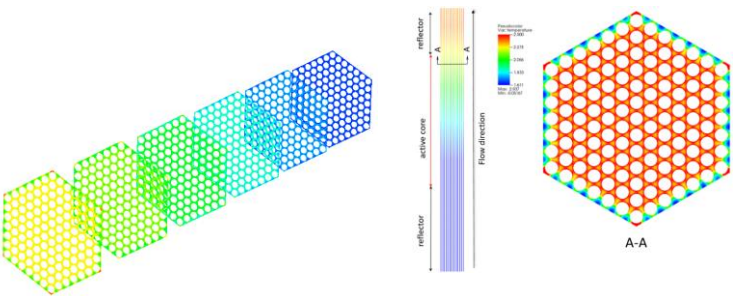


Method	Failure probability	Coefficient of variation	MOOSE model evaluations
Monte Carlo	7×10^{-8}	0.12	10^9
Parallelized subset simulation	5.1×10^{-8}	0.06	140,000
Active learning subset simulation	4.75×10^{-8}	0.192	130

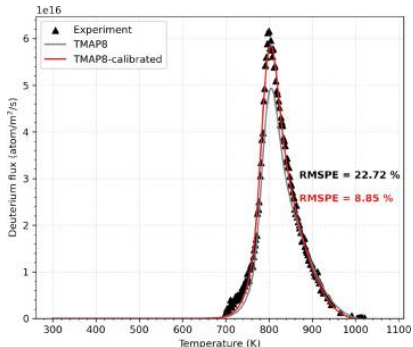
Dimensionality reduction for additive manufacturing simulations



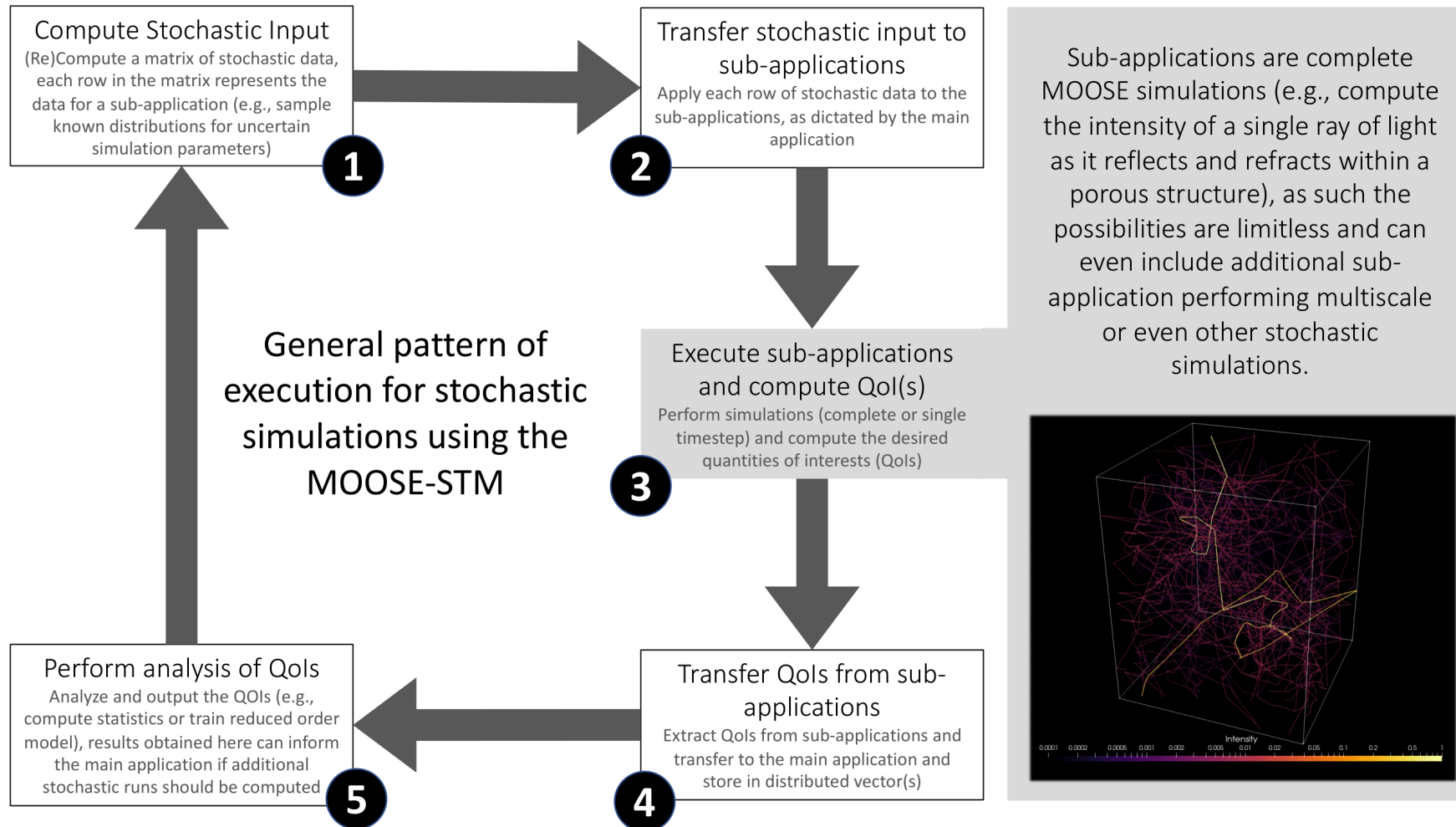
Hot channel factor calculation of LFRs with Griffin and Cardinal



Model calibration of Tritium migration with TMAP8



Workflow for Common Use-Cases of the MOOSE-STM



Sampling Methodology

- Sampling matrix

$$\mathbf{X} \equiv \begin{bmatrix} \mathbf{x}_{1,1} & \mathbf{x}_{1,2} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \mathbf{x}_{2,2} & \dots & \mathbf{x}_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_{N,1} & \mathbf{x}_{N,2} & \dots & \mathbf{x}_{N,M} \end{bmatrix}$$

- Random sampling using distributions and Monte Carlo or Latin Hypercube
- Tensor product or sparse quadrature
- Explicitly defined matrices with CSV or input syntax

- Parallelism

5 rows / 3 Ranks

$$\begin{bmatrix} \mathbf{x}_{1,1} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \dots & \mathbf{x}_{2,M} \\ \mathbf{x}_{3,1} & \dots & \mathbf{x}_{3,M} \\ \mathbf{x}_{4,1} & \dots & \mathbf{x}_{4,M} \\ \mathbf{x}_{5,1} & \dots & \mathbf{x}_{5,M} \end{bmatrix} \rightarrow \begin{array}{l} \text{CPU 0} \\ \text{CPU 0} \\ \text{CPU 1} \\ \text{CPU 1} \\ \text{CPU 2} \end{array}$$

5 rows / 8 Ranks

$$\begin{bmatrix} \mathbf{x}_{1,1} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \dots & \mathbf{x}_{2,M} \\ \mathbf{x}_{3,1} & \dots & \mathbf{x}_{3,M} \\ \mathbf{x}_{4,1} & \dots & \mathbf{x}_{4,M} \\ \mathbf{x}_{5,1} & \dots & \mathbf{x}_{5,M} \end{bmatrix} \rightarrow \begin{array}{l} \text{CPU 0/1} \\ \text{CPU 2/3} \\ \text{CPU 4/5} \\ \text{CPU 6} \\ \text{CPU 7} \end{array}$$

5 rows / 8 Ranks
Every row ≥ 2 ranks

$$\begin{bmatrix} \mathbf{x}_{1,1} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \dots & \mathbf{x}_{2,M} \\ \mathbf{x}_{3,1} & \dots & \mathbf{x}_{3,M} \\ \mathbf{x}_{4,1} & \dots & \mathbf{x}_{4,M} \\ \mathbf{x}_{5,1} & \dots & \mathbf{x}_{5,M} \end{bmatrix} \rightarrow \begin{array}{l} \text{CPU 0/1} \\ \text{CPU 0/1} \\ \text{CPU 2/3} \\ \text{CPU 4/5} \\ \text{CPU 6/7} \end{array}$$

- Mode of execution

Normal mode

$$\begin{bmatrix} \mathbf{x}_{1,1} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \dots & \mathbf{x}_{2,M} \\ \mathbf{x}_{3,1} & \dots & \mathbf{x}_{3,M} \\ \mathbf{x}_{4,1} & \dots & \mathbf{x}_{4,M} \\ \mathbf{x}_{5,1} & \dots & \mathbf{x}_{5,M} \end{bmatrix} \rightarrow \begin{array}{ll} \text{CPU 0} & \text{Sub-app 0} \\ \text{CPU 0} & \text{Sub-app 1} \\ \text{CPU 1} & \text{Sub-app 2} \\ \text{CPU 1} & \text{Sub-app 3} \\ \text{CPU 2} & \text{Sub-app 4} \end{array}$$

Batch mode

$$\begin{bmatrix} \mathbf{x}_{1,1} & \dots & \mathbf{x}_{1,M} \\ \mathbf{x}_{2,1} & \dots & \mathbf{x}_{2,M} \\ \mathbf{x}_{3,1} & \dots & \mathbf{x}_{3,M} \\ \mathbf{x}_{4,1} & \dots & \mathbf{x}_{4,M} \\ \mathbf{x}_{5,1} & \dots & \mathbf{x}_{5,M} \end{bmatrix} \rightarrow \begin{array}{ll} \text{CPU 0} & \text{Sub-app 0} \\ \text{CPU 0} & \text{Sub-app 0} \\ \text{CPU 1} & \text{Sub-app 1} \\ \text{CPU 1} & \text{Sub-app 1} \\ \text{CPU 2} & \text{Sub-app 2} \end{array}$$



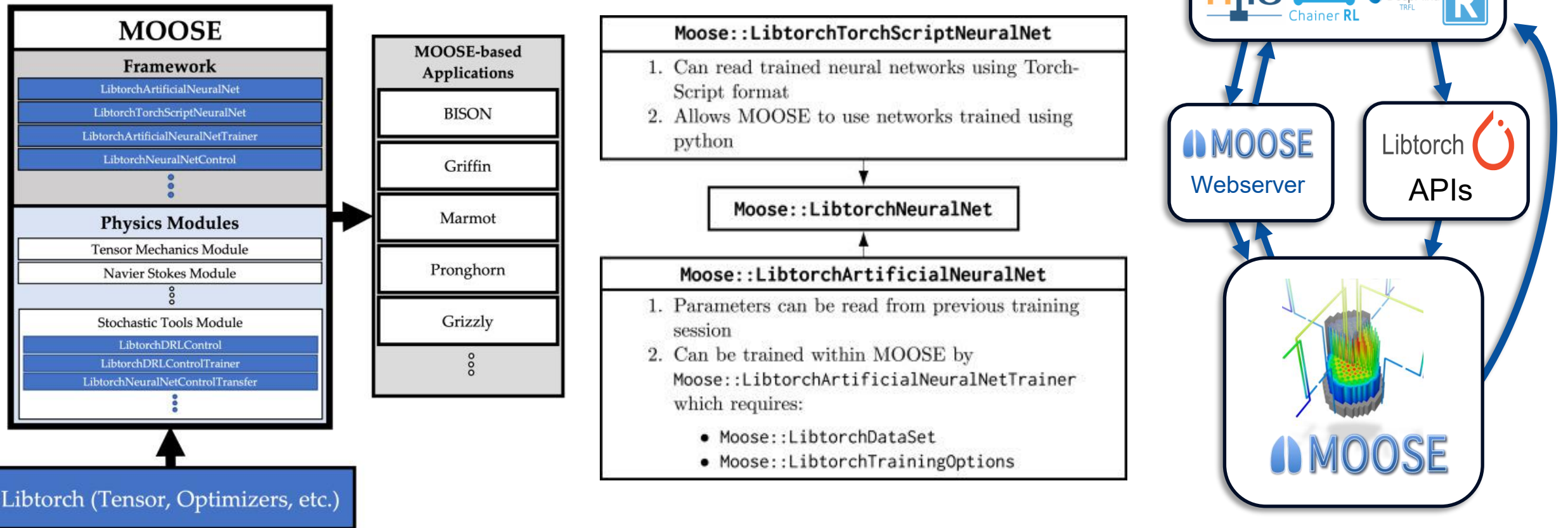
Surrogate Modeling

- STM Trainers/Surrogates provide a way of building reduced-order models that emulate high-fidelity multiphysics models.
- These surrogates are meant to be much faster to evaluate than the high-fidelity model.
- Usages for these surrogates:
 - Uncertainty quantification
 - Sensitivity analysis
 - Parameter optimization
 - Multi-scale modeling
 - Controller design
 - Digital twins
- Process:
 1. Produce response data using sampling/multiapp strategy
 2. Use Trainer object to compute meta data from predictor and response data
 3. Load trainer data to create Surrogate object and perform cross validation.
 4. Pick up Surrogate object to evaluate at inputted predictor values.
- Available surrogates:
 - Polynomial regression
 - Polynomial chaos
 - Proper orthogonal decomposition
 - **Gaussian process regression**
 - **Libtorch artificial neural network**



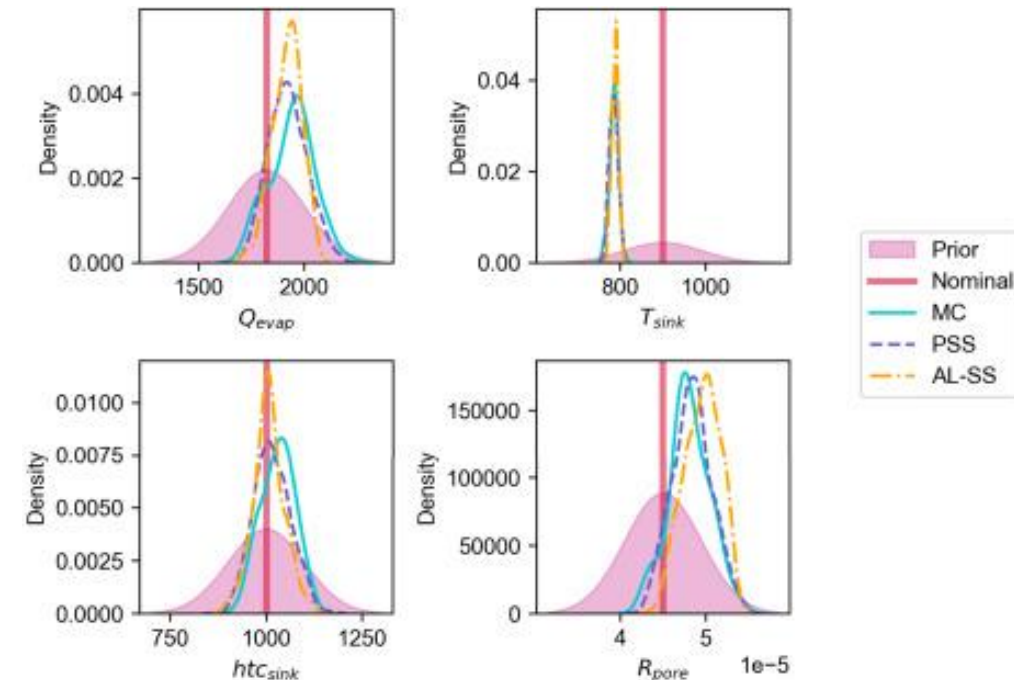
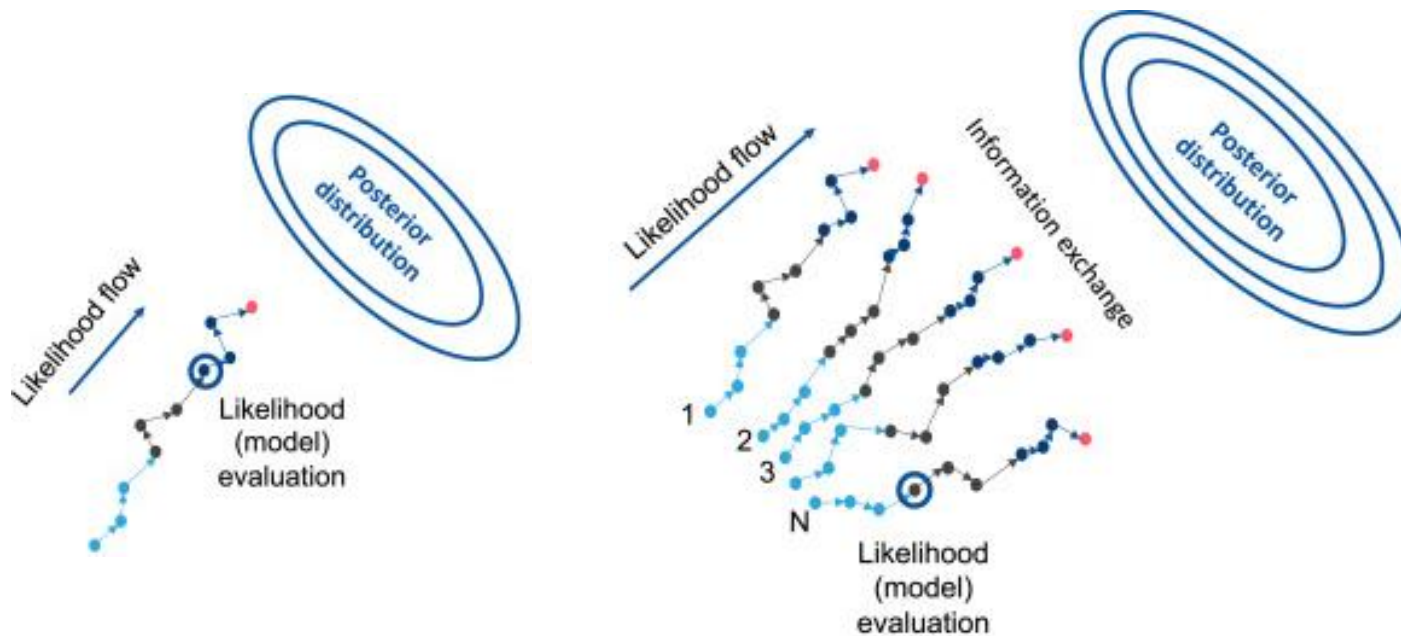
Torch Integration

- LibTorch is the C++ backend for PyTorch, enabling GPU-capable machine learning within MOOSE



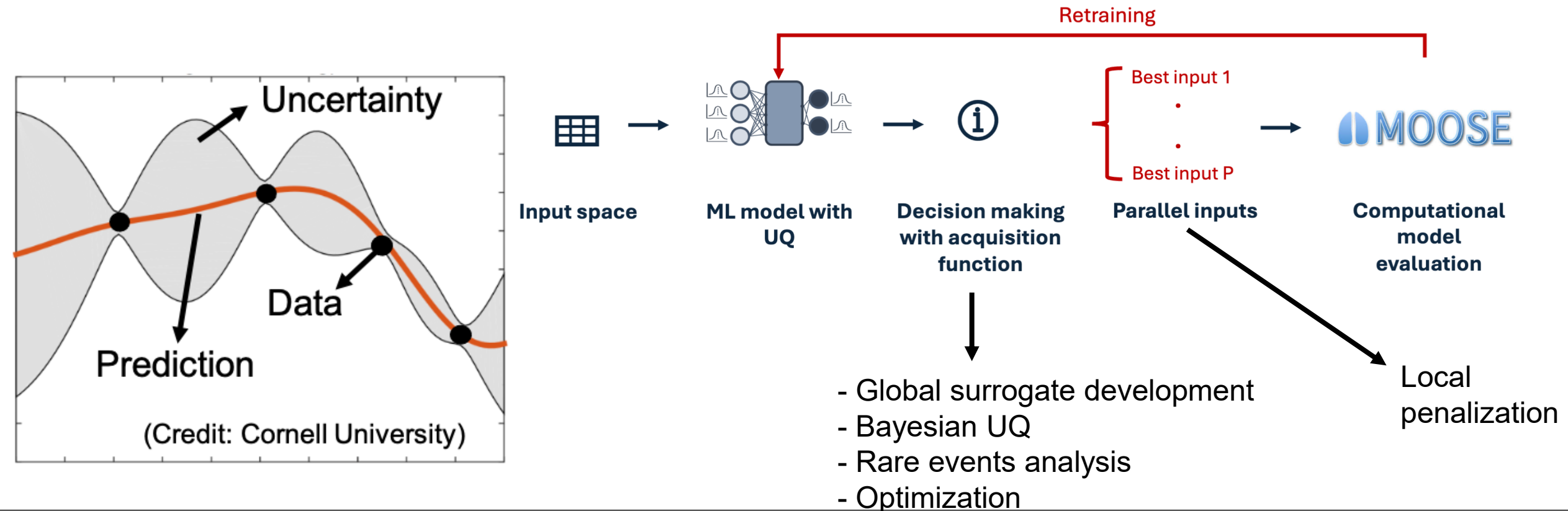
Bayesian Uncertainty Quantification

- Bayesian UQ is used to calibrate model parameters based on experimental results
- MOOSE-STM implements massively parallel algorithms



Active Learning

- Active learning is the process of training a surrogate while performing sampling
- Gaussian processes are used to determine the surrogate's uncertainty of a given sample



Python Interface with StochasticControl

- Utilize the massive parallelism of MOOSE-STM within Python

```
import numpy as np
from stochastic import StochasticControl, StochasticRunOptions

executable = "/path/to/moose-app-opt"      # MOOSE app with STM
physics_input = "physics.i"                # Your base physics input
parameters = ["Materials/K/value",         # Parameters to sample
              "Kernels/convection/alpha",
              "beta"]
qois = ["pp/value"]                        # Reporter(s) to collect as QoIs

# Configure how the stochastic run is executed
opts = StochasticRunOptions(
    num_procs=4,
    multiapp_mode=StochasticRunOptions.MultiAppMode.BATCH_RESET,
)

with StochasticControl(executable, physics_input, parameters, qois,
                       options=opts) as runner:
    # x: N-by-P matrix (N samples, P parameters)
    x = np.array([
        [1.0, 0.1, 0.2],
        [1.1, 0.1, 0.25],
        [0.9, 0.12, 0.2],
    ])
    y = runner(x)    # -> shape (N, Q) if multiple QoIs, vector if one QoI
```

```
from scipy.optimize import shgo
from stochastic import StochasticControl

with StochasticControl(executable, input_file, params, qois, opts) as runner:
    runner.configCache() # avoid recomputation of identical inputs
    # Bounds for inner radius and thickness
    bounds = [(0.1, 10), (0.01, 10)]
    # Create lambdas since there needs to be a separation of QoIs
    # for the objective and constraint
    objective = lambda x: runner(x)[: , 0] # max temperature
    eq_constraint = lambda x: runner(x)[: , 1] # volume

    result = shgo(
        objective,
        bounds=bounds,
        constraints=[{"type": "eq", "fun": eq_constraint}],
        # StochasticRunner provides a callback to interface with
        # SciPy's parallel workers feature
        workers=runner.parallelWorker
    )
```

NEAMS-Funded Goals

- “...increased robustness and speed of stochastic tools”
- FY26
 - Enable stateless sample generation
 - Make stochastic runs “recoverable”
 - Improve handling of failed runs
- And beyond
 - Enable GPU-acceleration with libTorch
 - Expand usability of batch-restore
 - Create verification and validation cases





U.S. DEPARTMENT
of **ENERGY**

Office of
Nuclear Energy

Sampling Methodology

- Samplers define a “sampling matrix” used for stochastic analysis

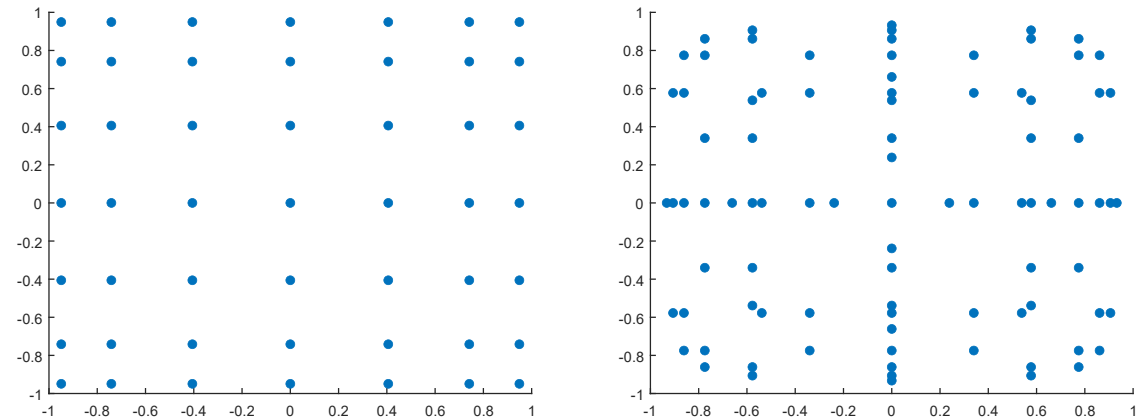
$$\mathbf{X} \equiv \begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,M} \\ x_{2,1} & x_{2,2} & \dots & x_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,M} \end{bmatrix}$$

- Random sampling

```
[Distributions]
[uniform]
type = Uniform
lower_bound = 1
upper_bound = 9
[]
[normal]
type = Normal
mean = 1
standard_deviation = 0.1
[]
[]
```

```
[Samplers]
[mc]
type = MonteCarlo
distributions = 'uniform normal'
num_rows = 10
[]
[lhs]
type = LatinHypercube
distributions = 'uniform normal'
num_rows = 10
[]
[]
```

- Product samplers



- Explicitly defined samplers

```
[Samplers]
[csv]
type = CSVSampler
samples_file = data.csv
[]
[matrix]
type = InputMatrix
matrix = '11 12 13; 21 22 23'
[]
[]
```

