Multi-Modal Data Driven and Physics-Informed Machine Learning with Uncertainty for Materials Applications

Nat Trask[1], Somdatta Goswami[2], Michael D. Shields[2]

[1]University of Pennsylvania, Division of Mechanical Engineering and Applied Mechanics

[2]Johns Hopkins University, Department of Civil and Systems Engineering

SIAM Conference on Mathematical Aspects of Materials Science (SIAM -MS24)

21 May 2024

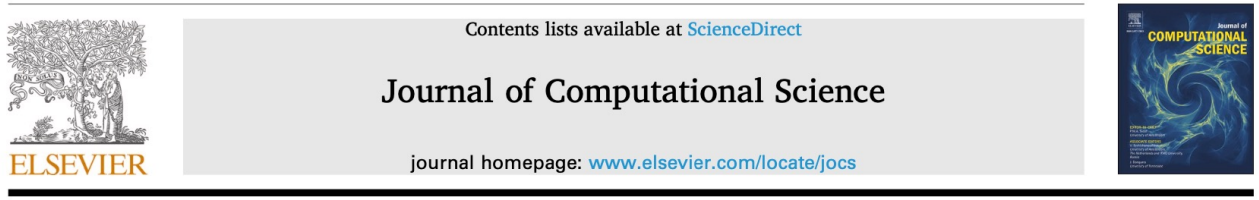
SIAM MS24

UQpy

- A collection of Python modules used for uncertainty quantification and propagation
  - Includes commonly applied methods and new developments
  - Serves as a UQ toolbox and a Python development environment
- Developed collaboratively by members of SURG
  - Contributors: Ponkrshnan Thiagarajan, Connor Krill, Georgios Pasparakis, Dimitris Tsapetis, Dimitris Giovanis, Audrey Olivier, Aakash Bangalore Satish, Lohit Vandanapu, Mohit Chauhan, Katiana Kontolati, Ketson R.M. dos Santos
- Contributions from researchers around the world
  - Dr. Dimitris Loukrezis (Siemens), Prof. Michael Gardner (University of Nevada, Reno), Dr. Lukas Novak (Brno University of Technology), Ulrich Römer, Julius Schultz (TU Braunschweig)

UQpy

- Version control through git (requires Python 3)
  - Version 4.1.1 available for download/installation via GitHub (https://github.com/SURGroup/UQpy)
- Available on the Python Package Index (PyPI) and Conda
- Thorough documentation and examples:



UQpy: A general purpose Python package and development environment for uncertainty quantification

Original Software Publication

UQpy v4.1: Uncertainty quantification with Python

Dimitrios Tsapetis [a], Michael D. Shields [a,*], Dimitris G. Giovanis [a], Audrey Olivier [b], Lukas Novak [c], Promit Chakroborty [a], Himanshu Sharma [a], Mohit Chauhan [a], Katiana Kontolati [a], Lohit Vandanapu [a], Dimitrios Loukrezis [d], Michael Gardner [e]

UQpy-ML

- New module on UQ for Scientific Machine Learning (SciML) under development
- Module will add capabilities for:
  - Neural Operators
  - Physics Informed Neural Networks (PINNs)
  - Physics Informed Neural Operators
  - Bayesian Neural Networks
- Methods being built on the PyTorch library

In cooperation with
Society for Industrial and Applied Mathematics

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Multi-Modal Data Driven and Physics-Informed Machine Learning with Uncertainty for Materials Applications

Nat Trask[1], Somdatta Goswami[2], Michael D. Shields[2]

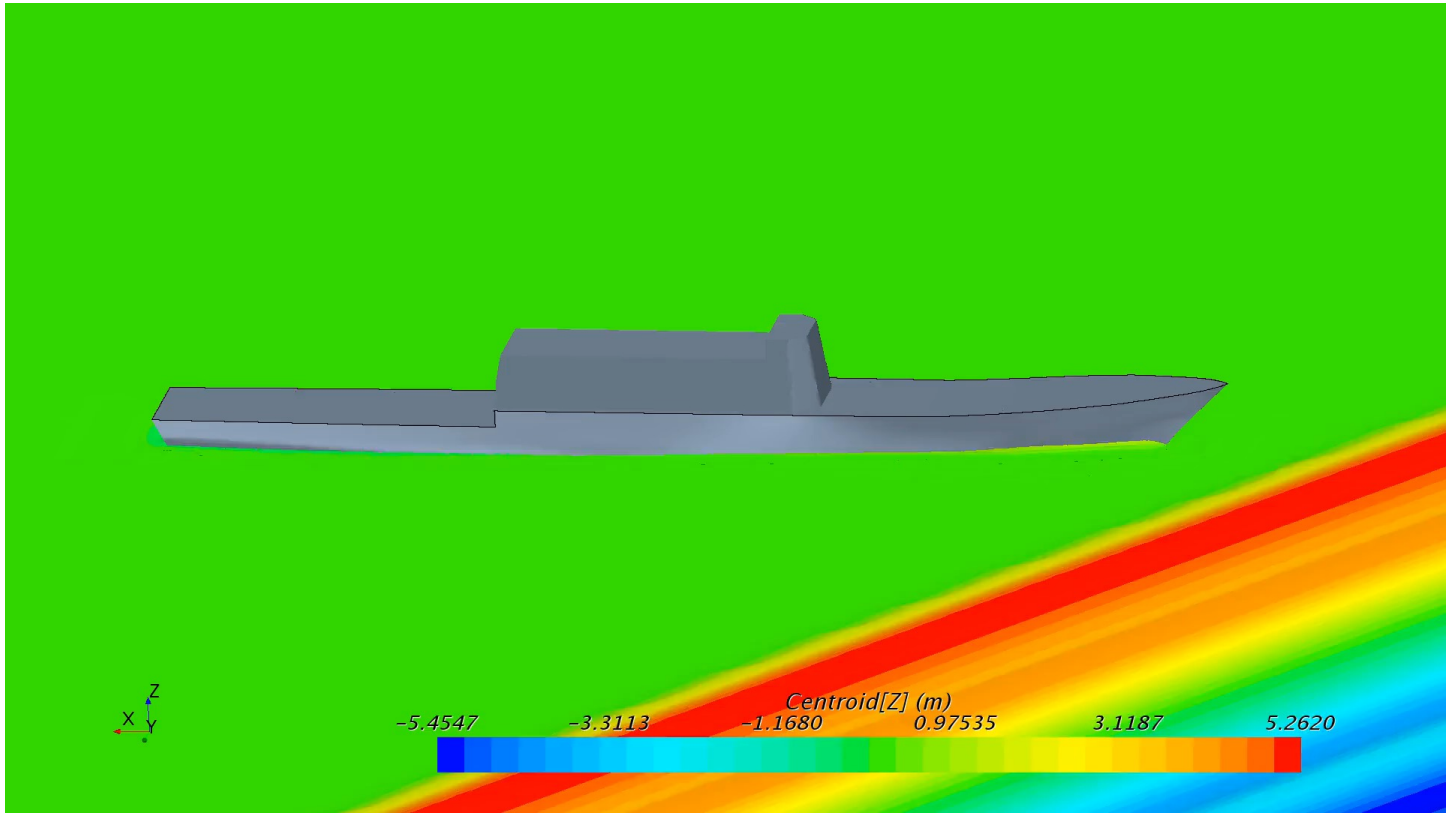
[1]University of Pennsylvania, Division of Mechanical Engineering and Applied Mechanics

[2]Johns Hopkins University, Department of Civil and Systems Engineering

SIAM Conference on Mathematical Aspects of Materials Science (SIAM -MS24)

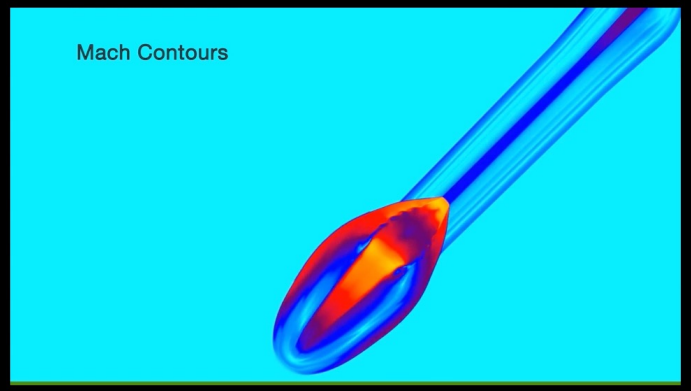
May 21, 2024

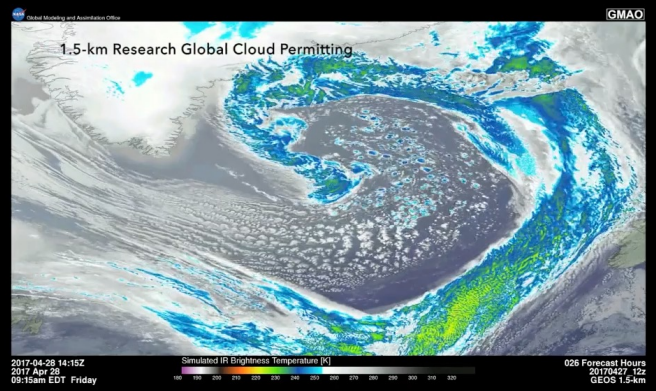
Can Neural Networks Predict in Real-Time (Autonomy)?

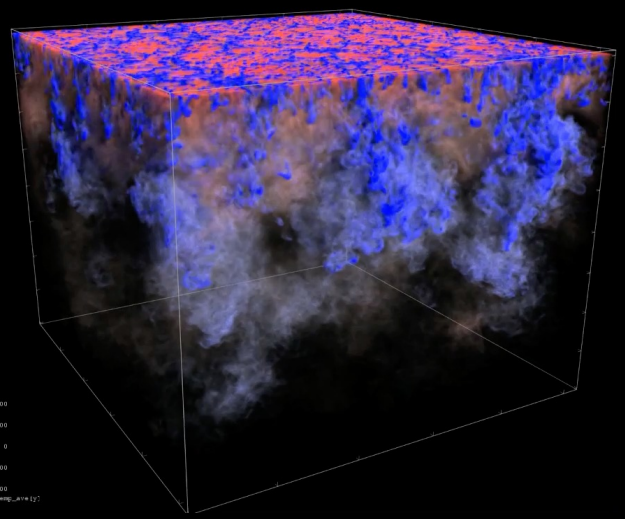


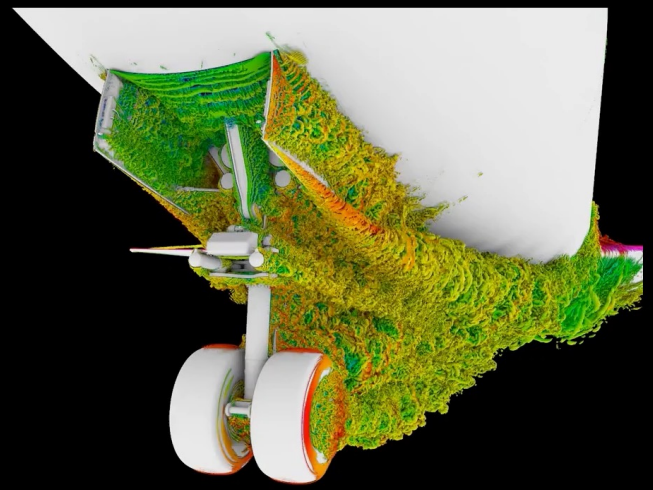
Courtesy: MIT, Michael Triantafyllou

Surrogate modeling for nonlinear PDEs









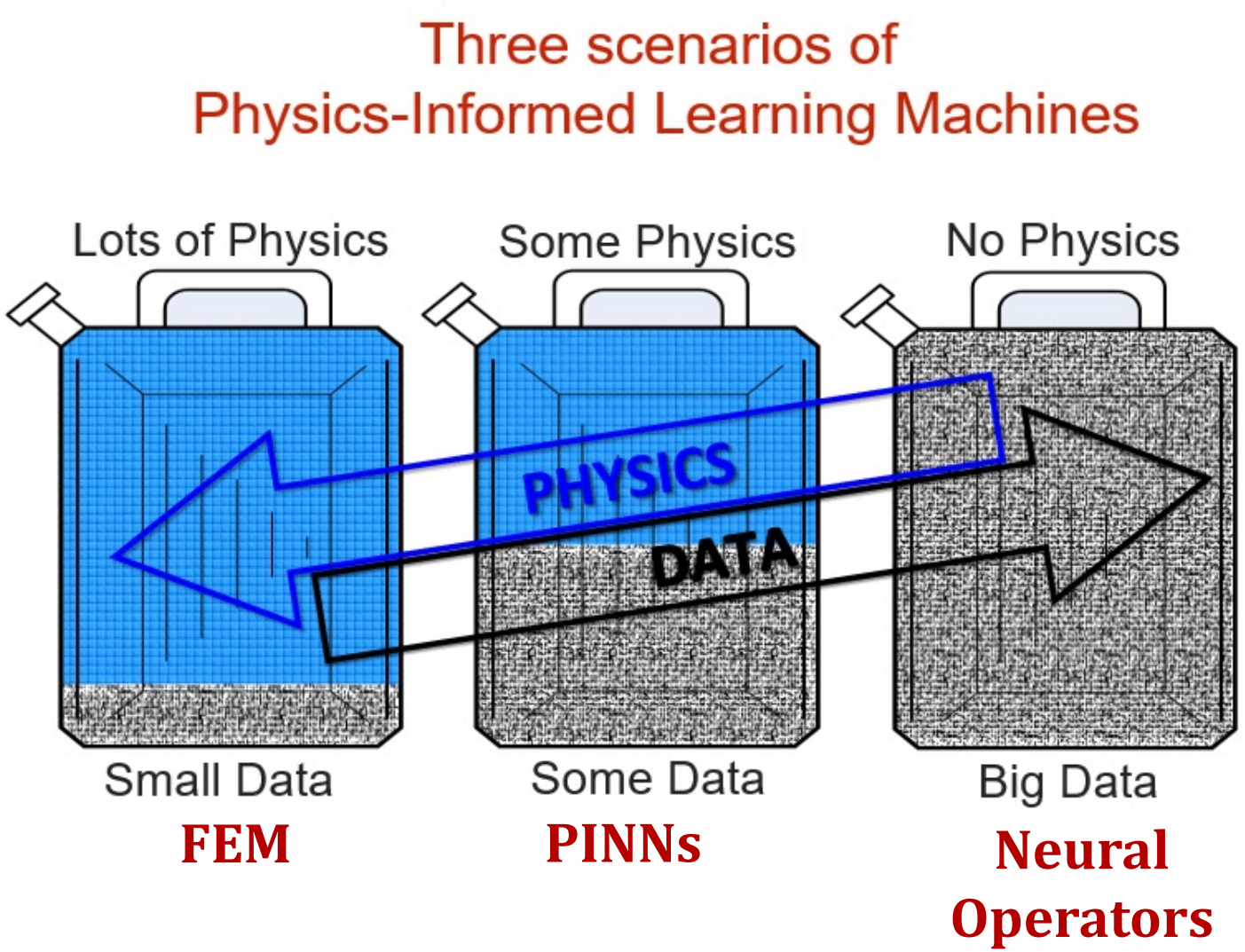
Upper left: Simulation of airburst generated by a meteoroid with 20k/m entry speed
Upper right: Evolution of an Icelandic Low in the North Atlantic Ocean over three days
Lower left: Temperature variations in a star 1.35 times as massive as the Sun (Kelvin)
Lower right: Flow field around the nose landing gear of a Boeing 777
Source: https://www.nas.nasa.gov/SC17/ (NASA)

- Physics behind complex systems are approximated via mathematical models (ODEs, PDEs)

$$\frac{\partial v}{\partial t} = \mathcal{L}(t, x, v, \nabla u, \nabla\nabla v, \dots)$$

- Surrogate models provide a fast-to-evaluate alternative

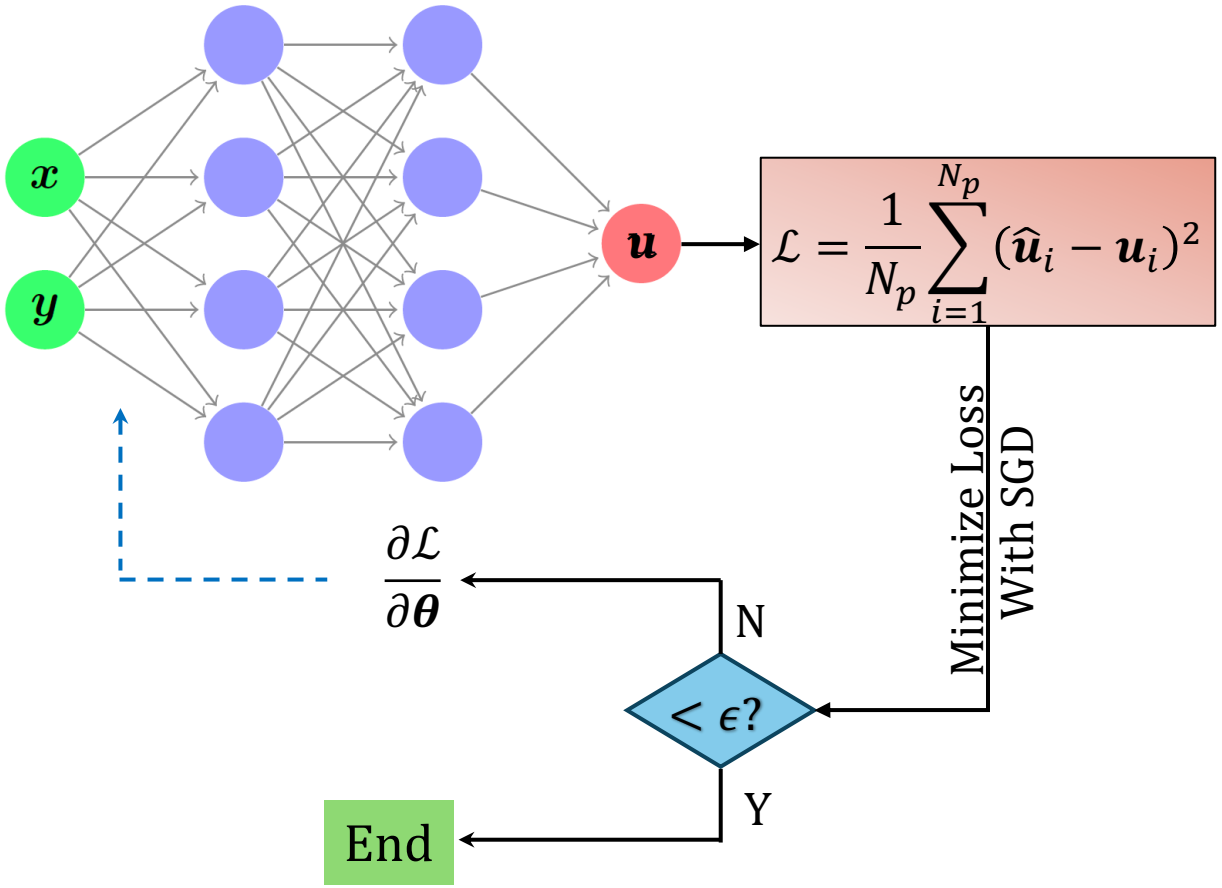
Data + Laws of Physics



Training data-driven NNs

$$\mathbb{L}(\mathcal{X}, \lambda, \boldsymbol{u}(\boldsymbol{x}); \boldsymbol{\theta}) = g$$
$$\boldsymbol{x} = (x, y) \in \Omega, \lambda \in D, \lambda = 1$$

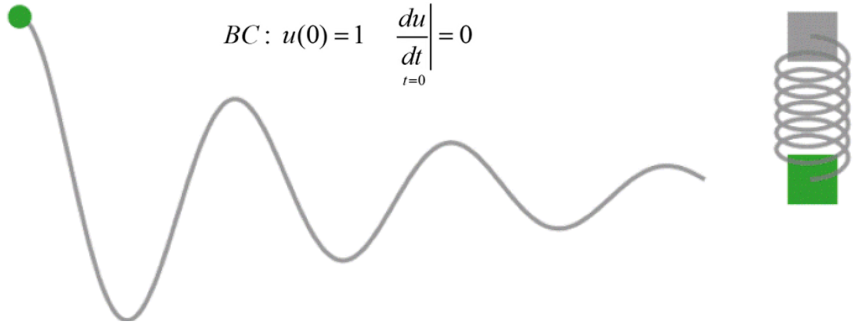
$\hat{\boldsymbol{u}}$: Predicted solution
$\boldsymbol{\theta}$: Weights and biases
N_p: # of collocation points
$(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}))$: Labelled dataset



What is the problem with data-driven?

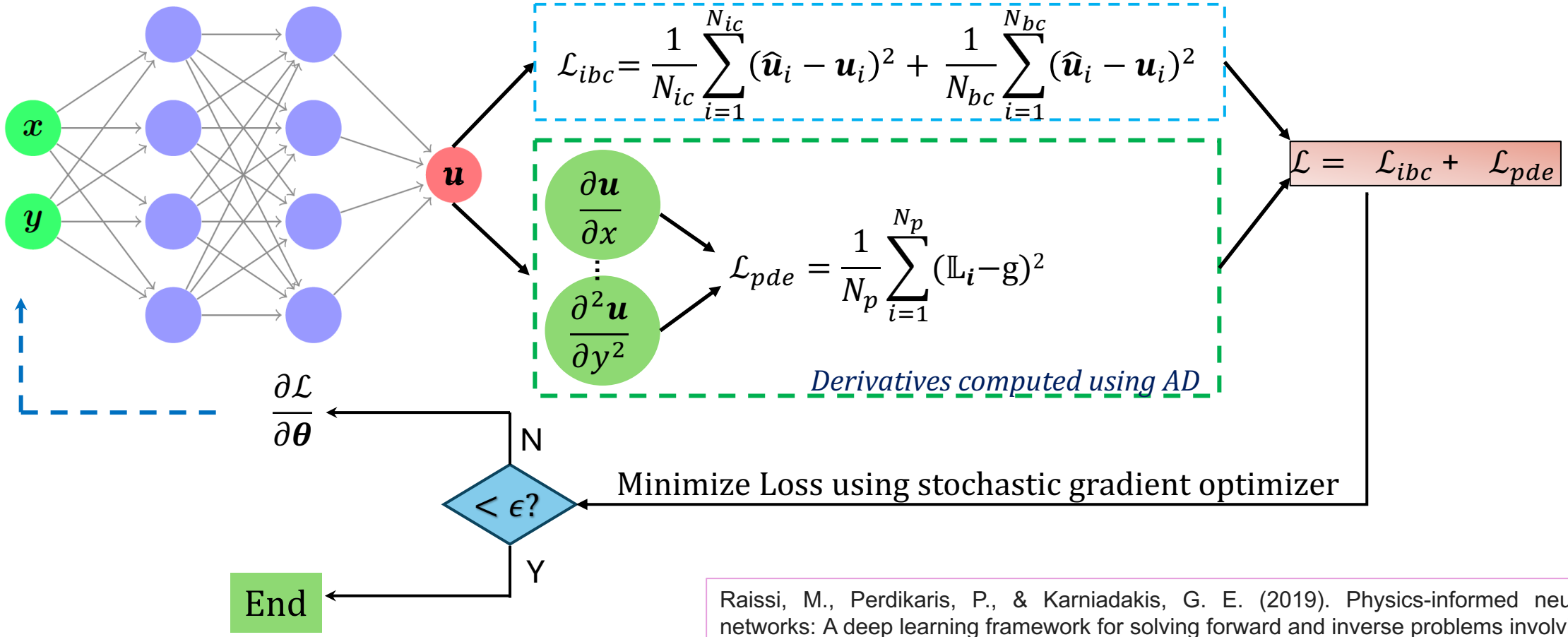
$$m\frac{d^2u}{dt^2}+\mu\frac{du}{dt}+ku=0 \quad 0\le x\le 1$$

$$BC:\ u(0)=1 \qquad \left.\frac{du}{dt}\right|_{t=0}=0$$



Inspired by Ben Moley blog 2021

Physics-Informed Neural Networks (PINNs)



Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, *378*, 686-707.

$\mathbb{L}(\mathcal{X}, \lambda, \boldsymbol{u}(\boldsymbol{x}); \boldsymbol{\theta}) = g$

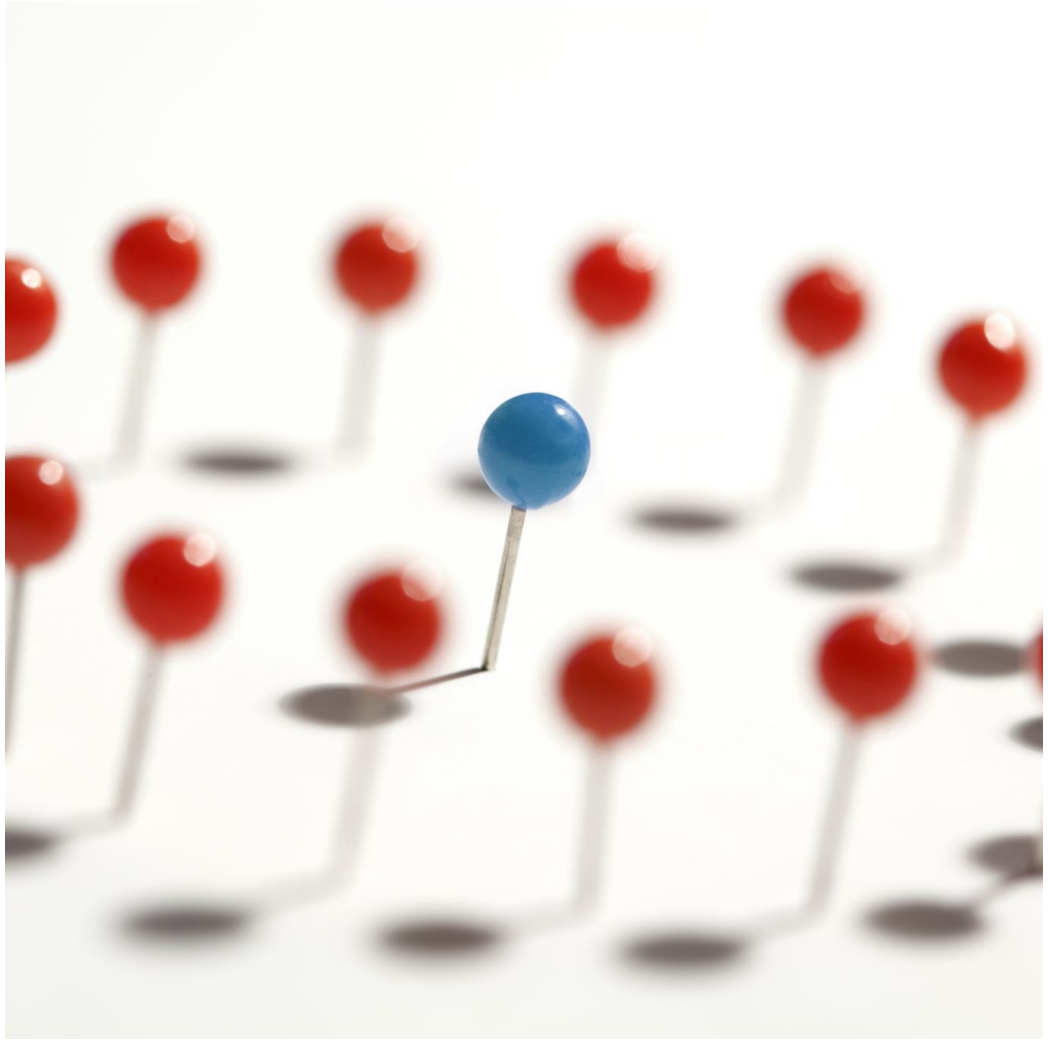
$\boldsymbol{x} = (x, y) \in \Omega,\ \lambda \in D, \lambda = 1$

$\hat{\boldsymbol{u}}$: Predicted solution

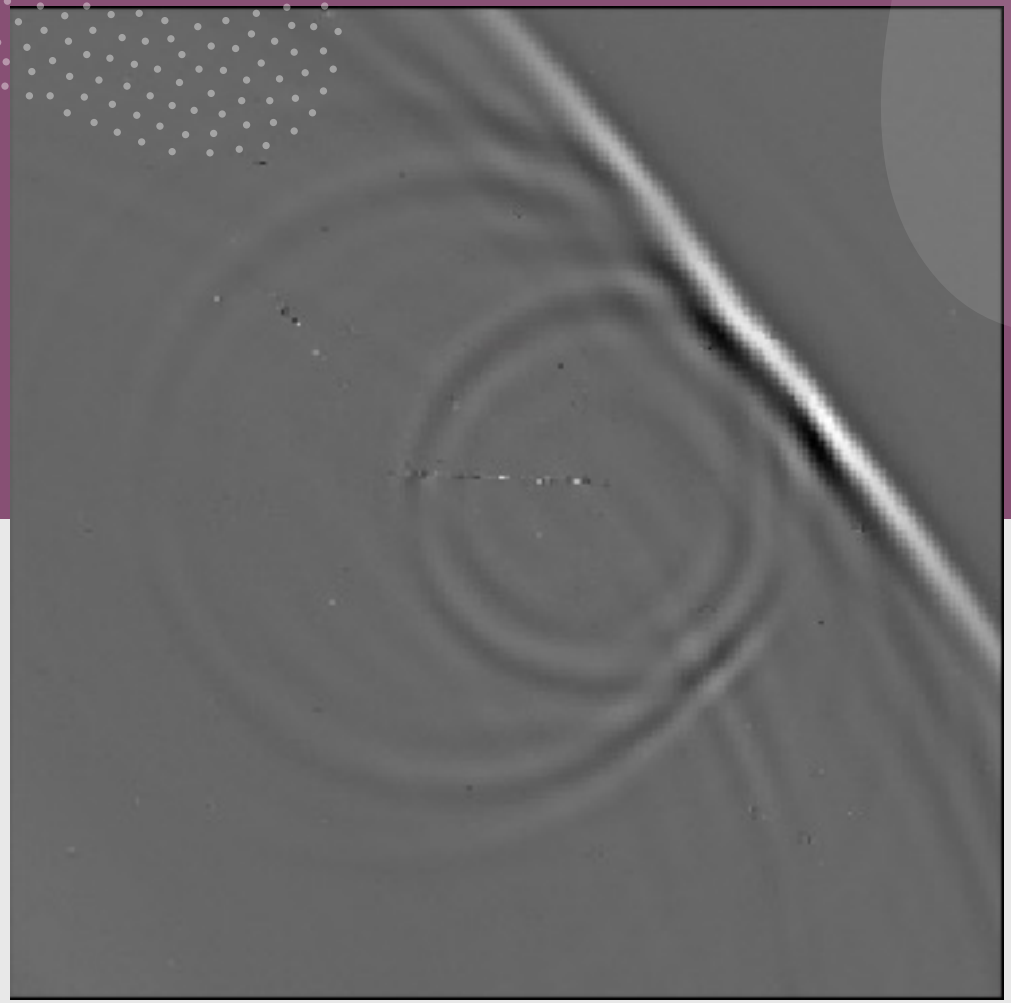
θ : Weights and biases

$N_{ic/bc}$: Initial conditions and boundary conditions, $N_{ic} + N_{bc} \ll N_p$

In the last 5 years, PINNs have been successful for *Inverse* problems.



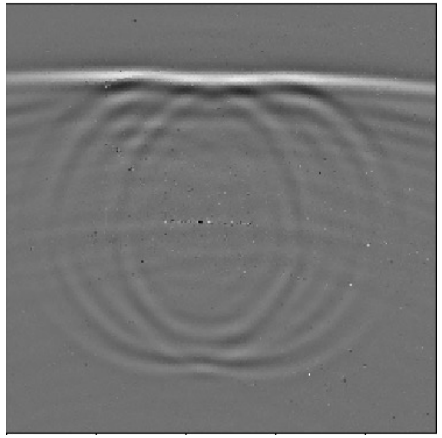
Example for Inverse PINNs: Where is the Crack?



Courtesy: Air Force Lab, WPAFL,
J. Blackshire & D. Sparkman

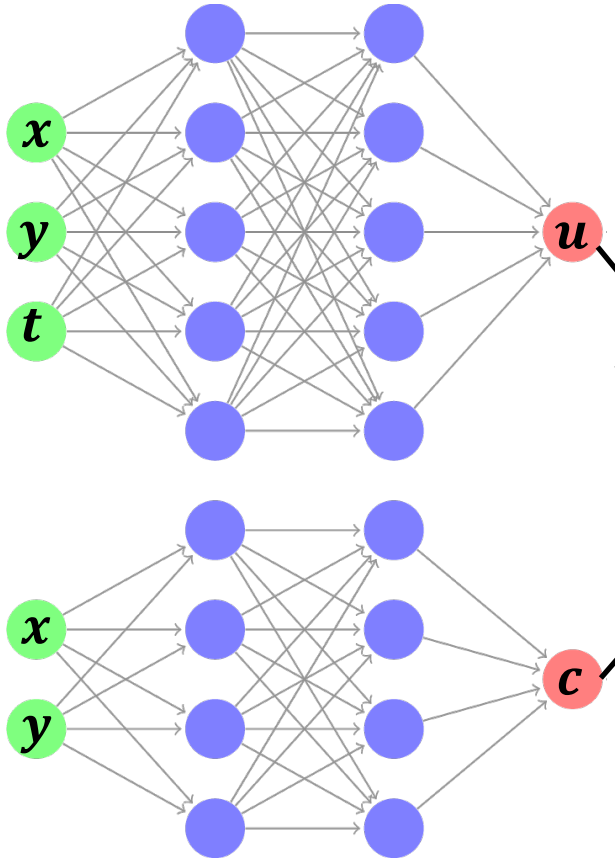
Crack characterization in aluminum alloy using PINNs

- Objective: Identifying and characterizing surface breaking crack in 7075-T651 aluminum alloy metal plate.
- Dataset: Particle displacement induced by surface acoustic waves of 5 MHz at three incidence angle 0°, 45°, and 90°



0° incidence

$$f = \frac{\partial^2 u}{\partial t^2} - c(x,y)^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

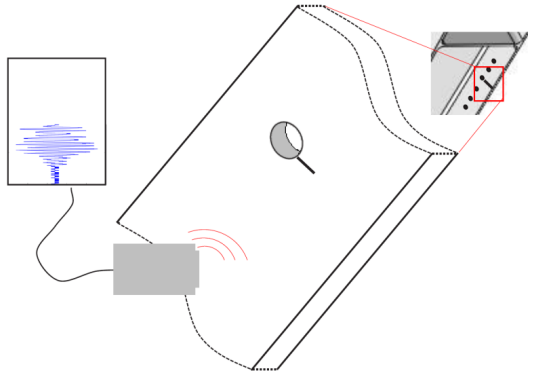


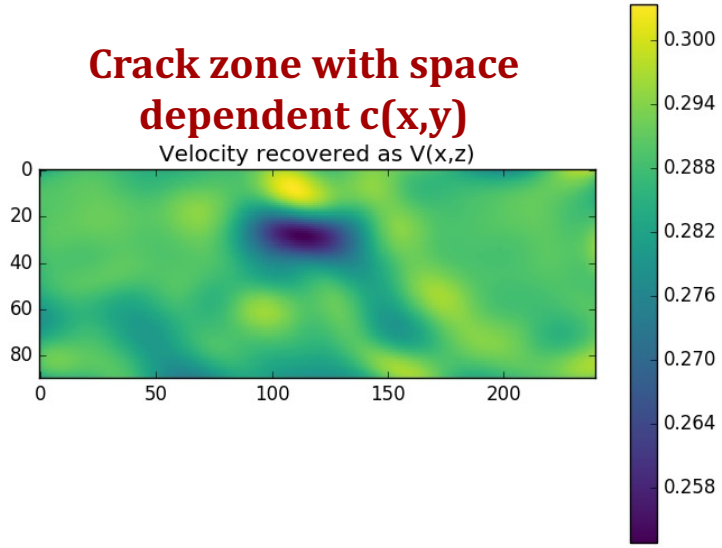
$$\mathcal{L}_{data} = |u_{NN}(x,y,t) - u(x,y,t)|^2$$

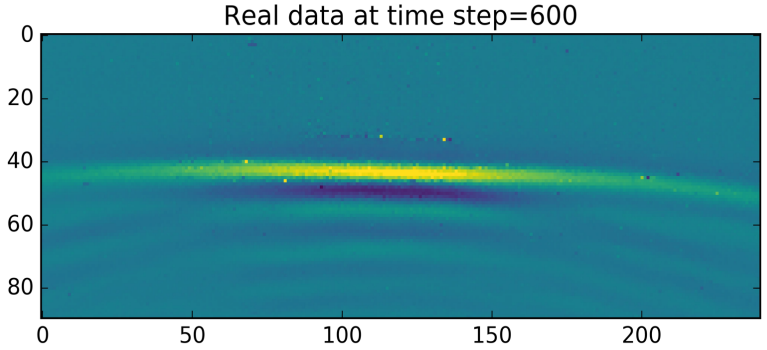
$$\mathcal{L}_{pde} = |\frac{\partial^2 u_{NN}}{\partial t^2} - c_{NN}(x,y)^2\left(\frac{\partial^2 u_{NN}}{\partial x^2} + \frac{\partial^2 u_{NN}}{\partial y^2}\right)$$

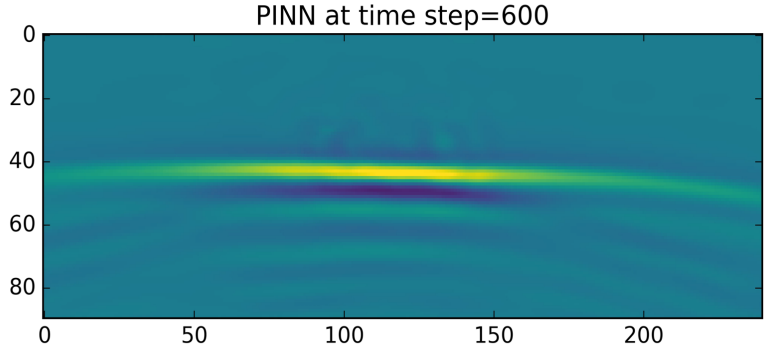
$$\mathcal{L} = \gamma\mathcal{L}_{data} + \mathcal{L}_{pde}$$

Crack characterization in aluminum alloy using PINNs









K. Shukla, P. Clark Di Leoni, J. Blackshire, D. Sparkman, G. E. Karniadakis,“ Physics-informed Neural Network for Ultrasound Nondestructive Quantification of Surface Breaking Cracks. J Nondestruct Eval 39, 61 (2020).

Burger's Equation

Burger's Equation is defined as

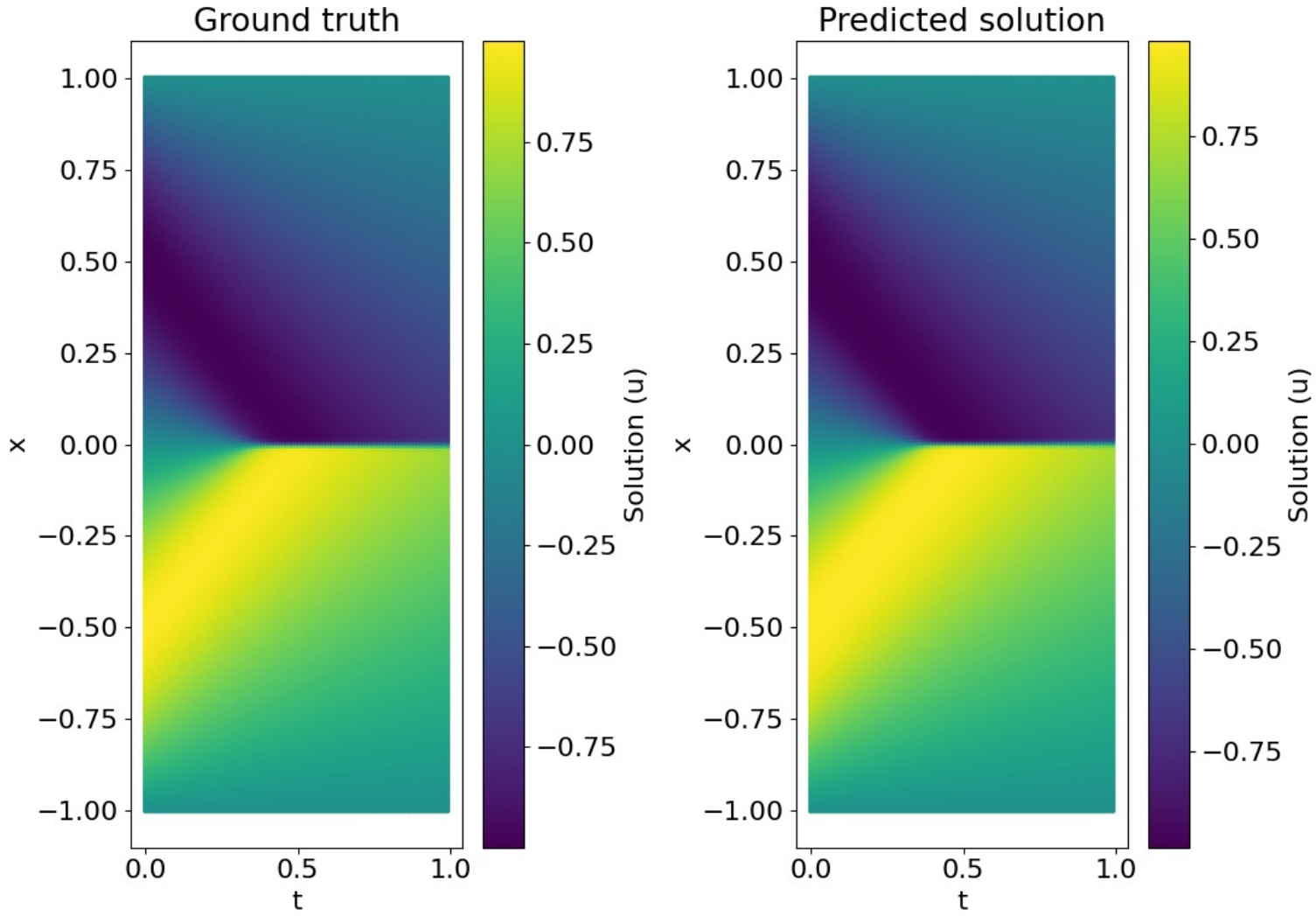
$$u_t + uu_x - (0.01/\pi)u_{xx} = 0$$

$$u(0, x) = -\sin(\pi x),$$
$$u(t, -1) = u(t, 1) = 0.$$

Let us define $f(t, x)$ to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

Burgers Equation: Results using PINNs



Variants of PINNs

- Variational PINNs (V-PINNs): Instead of residual minimization, the total energy of the system is minimized. Good for problems in solid mechanics. Requires lower lower derivatives than in residual.
  Goswami, S., Anitescu, C., Chakraborty, S., & Rabczuk, T. (2020). Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theoretical and Applied Fracture Mechanics, 106, 102447.

- hp-PINNs: Writing the loss function on the weak formulation with assumptions of the testing function. Taking advantage of Finite element approach.
  Kharazmi, Ehsan, Zhongqiang Zhang, and George Em Karniadakis. "hp-VPINNs: Variational physics-informed neural networks with domain decomposition." Computer Methods in Applied Mechanics and Engineering 374 (2021): 113547.

- Extended PINNs (X-PINNs): Generalized space-time domain decomposition approach to solve nonlinear PDEs on arbitrary complex-geometry domains.
  Shukla, Khemraj, Ameya D. Jagtap, and George Em Karniadakis. "Parallel physics-informed neural networks via domain decomposition." Journal of Computational Physics 447 (2021): 110683.

- Conservative PINNs (C-PINNs): Enforcing the flux continuity in the strong form along the sub-domain interfaces of XPINNs.
  Jagtap, Ameya D., Ehsan Kharazmi, and George Em Karniadakis. "Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems." Computer Methods in Applied Mechanics and Engineering 365 (2020): 113028.

Variants of PINNs

- Gradient-enhanced PINNs: Introducing the gradient pf the loss function in the residual. Done by a 17 years old high-school student.

  Yu, Jeremy, Lu Lu, Xuhui Meng, and George Em Karniadakis. "Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems." Computer Methods in Applied Mechanics and Engineering 393 (2022): 114823

- Seperable PINNs: Introduced to mitigate the curse of dimensionality using forward mode automatic differentiation and decomposed input spaces using smaller MLPs.

  Cho, Junwoo, Seungtae Nam, Hyunmo Yang, Seok-Bae Yun, Youngjoon Hong, and Eunbyung Park. "Separable physics-informed neural networks." Advances in Neural Information Processing Systems 36 (2024).

..... A-Z PINNs

Can PINNs be a Surrogate model?

ν

Governing Equation $u_t + uu_x - (0.01/\pi)u_{xx} = 0$

Initial conditions $u(0, x) = -\sin(\pi x),$

Boundary conditions $u(t, -1) = u(t, 1) = 0.$

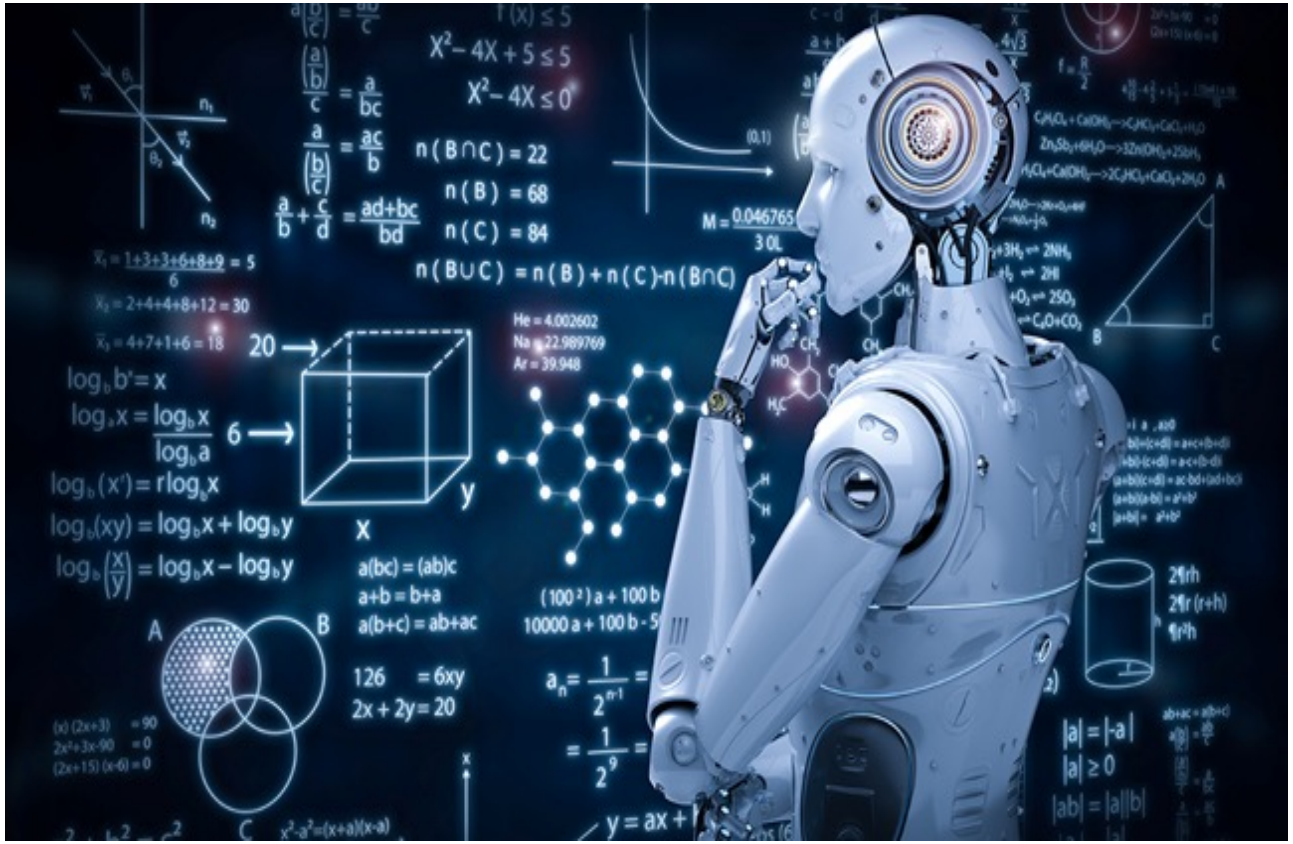
Given the above information, PINNs is trained for $\nu = 0.01$

Wow my surrogate model is made. Let's check the results for $\nu = 100$.

My results are all wrong.

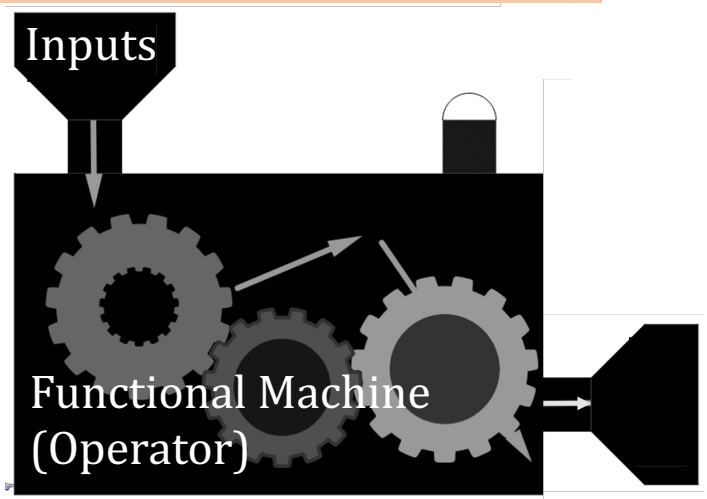
Let me train a robot to solve this for me automatically.

Do we need to teach calculus to the robot?



Learning a Neural Operator

$$\mathcal{F} = \{f(\boldsymbol{x}, \lambda_1), f(\boldsymbol{x}, \lambda_2), \dots, f(\boldsymbol{x}, \lambda_n)\}$$



$$\Phi : \mathcal{F} \to \mathcal{S}$$

$\mathcal{F}, \mathcal{S}$ are infinite dimensional function space

Outputs

$$\mathcal{S} = \{u(\boldsymbol{x}, \lambda_1), u(\boldsymbol{x}, \lambda_2), \dots, u(\boldsymbol{x}, \lambda_n)\}$$

What we know

$$\{\mathcal{F}_n, \mathcal{S}_n\}_{n=1}^{N}$$

$$\mathcal{S}_n = \Phi(\mathcal{F}_n)\,, \mathcal{F}_n \sim \mu \; i.i.d$$

Task

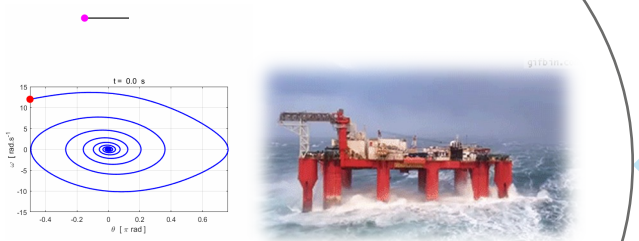
Operator, $\Psi : \mathcal{F} \times \Theta \to \mathcal{S}$ such that $\Psi(., \theta^*) \approx \Phi$

Training $\theta^* = \text{argmin}_\theta l(\{\mathcal{F}_n, \Psi(\mathcal{S}_n, \theta)\})$

Deep Neural Operator

- Learning the mapping between infinite-dimensional spaces employing continuous nonlinear operators developed with DNN.
- The neural operators can learn an entire family of equations and have enhanced generalization ability.

$$G: \mathcal{F} \times \Theta \rightarrow \mathcal{S}$$



Type 1
Universal Approximation

Type 2
Green's Integral

Universal Approximation Theorem for Operators

Universal Approximation Theorem for Operator

$G : u \mapsto G(u),\ G(u) : y \in \mathbb{R}^d \to \mathbb{R}$

Theorem (Chen & Chen, IEEE Trans. Neural Netw., 1995)

Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, such that

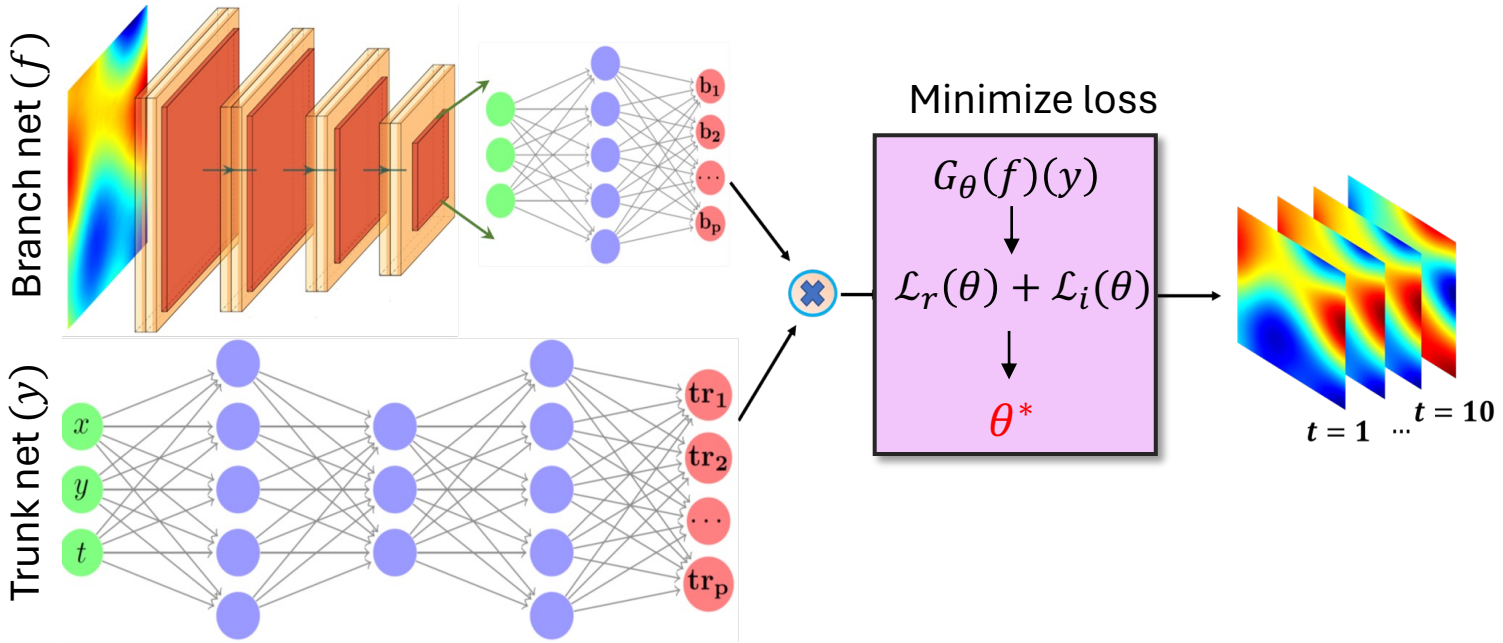
$$\left| G(u)(y) - \underbrace{\sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon$$

holds for all $u \in V$ and $y \in K_2$.

Deep Operator Network (DeepONet)

- Generalized Universal Approximation Theorem for Operator [Chen '95, Lu et al. '19]
- **Branch net**: Input $\{f(x_i)\}_{i=1}^{m}$, output: $[b_1, b_2, .., b_p]^T \in \mathbb{R}^p$
- **Trunk net**: Input y, output: $[t_1, t_2, .., t_p]^T \in \mathbb{R}^p$
- Input f is evaluated at locations $\{y_i\}_{i=1}^{m}$

$$\hat{u} = G_\theta(f)(y) = \underbrace{\sum_{i=1}^{p} b_i(f(x_1), f(x_2), \dots, f(x_m))}_{\text{branch net}} \cdot \underbrace{tr_i(y)}_{\text{trunk net}}$$



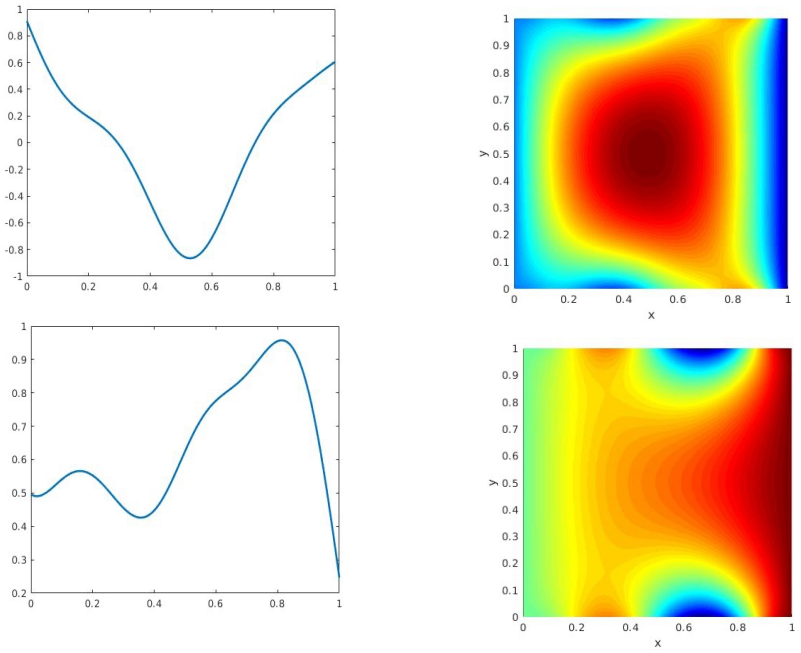
Lu, Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators." Nature machine intelligence

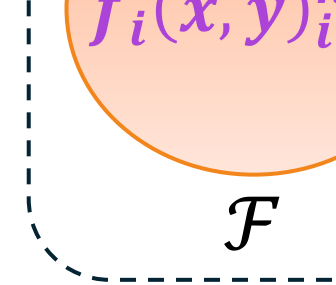
Motivation of Operator Networks

$$-\left(\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2}\right) = 1 \qquad u_{\partial\Omega}(1,y) = f(x,y) \qquad \Omega = [0,1]\times[0,1]$$

Domain

$f(x,y) \longrightarrow u(x,y)$

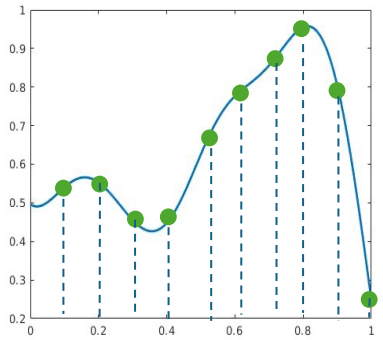


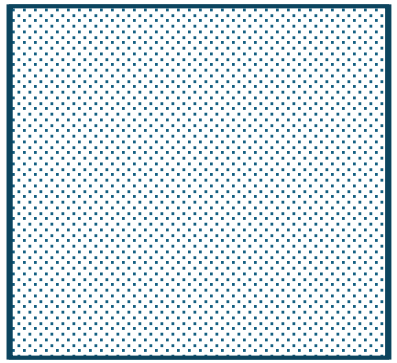


Building a DeepONet Model

Testing Boundary conditions

u on $\partial\Omega$

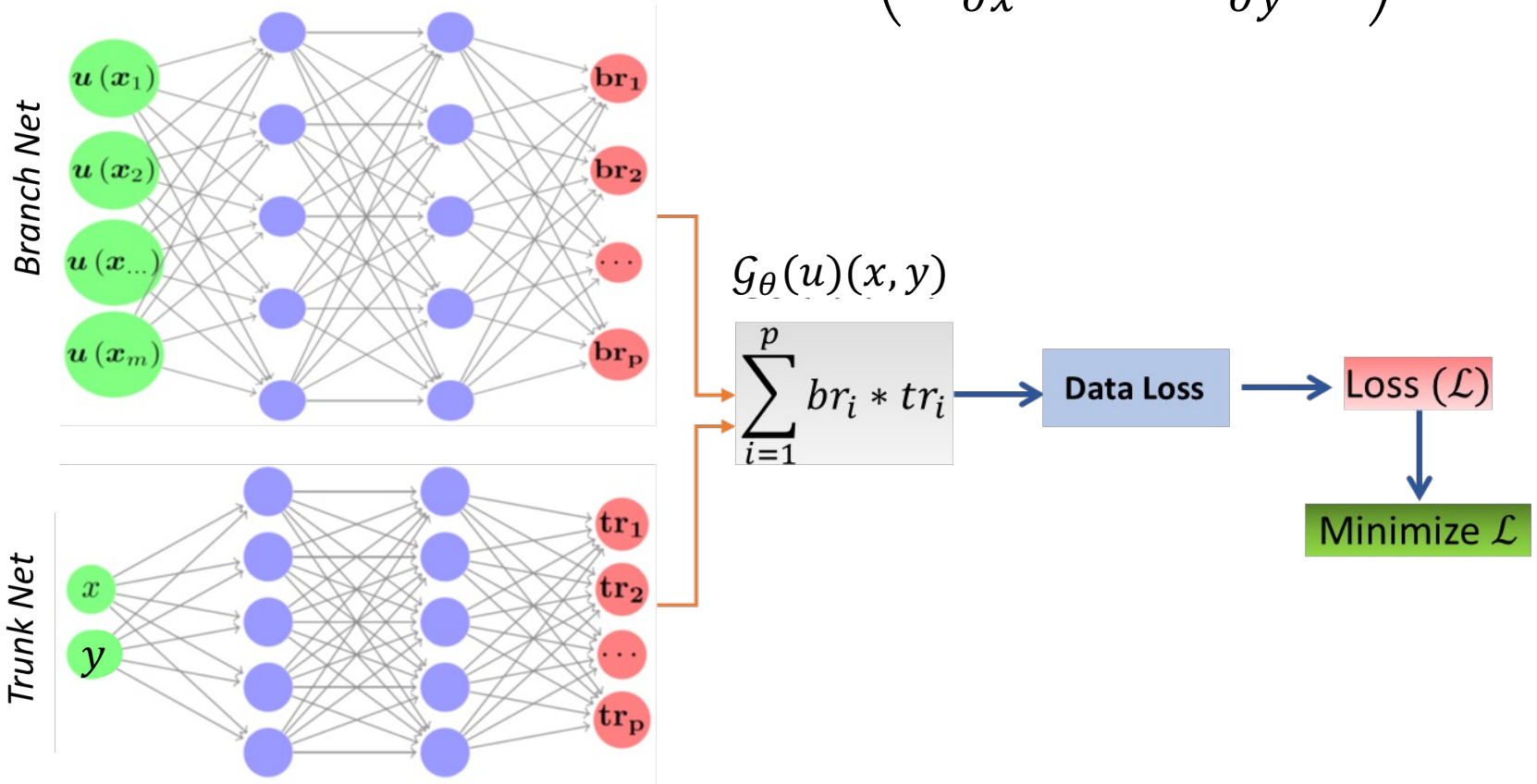




Query Points during testing

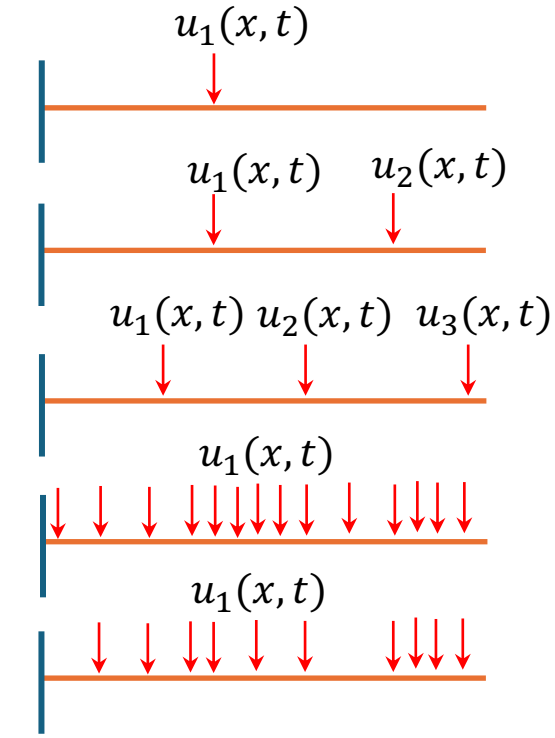
$$\mathcal{G}_\theta : u_{\partial\Omega}(0, y) \rightarrow u(x, y)$$

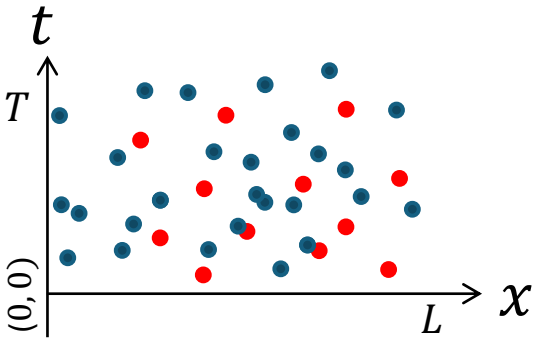
$$-\left(\frac{\partial^2 u(x,y)}{\partial x^2} + \frac{\partial^2 u(x,y)}{\partial y^2}\right) = 1$$



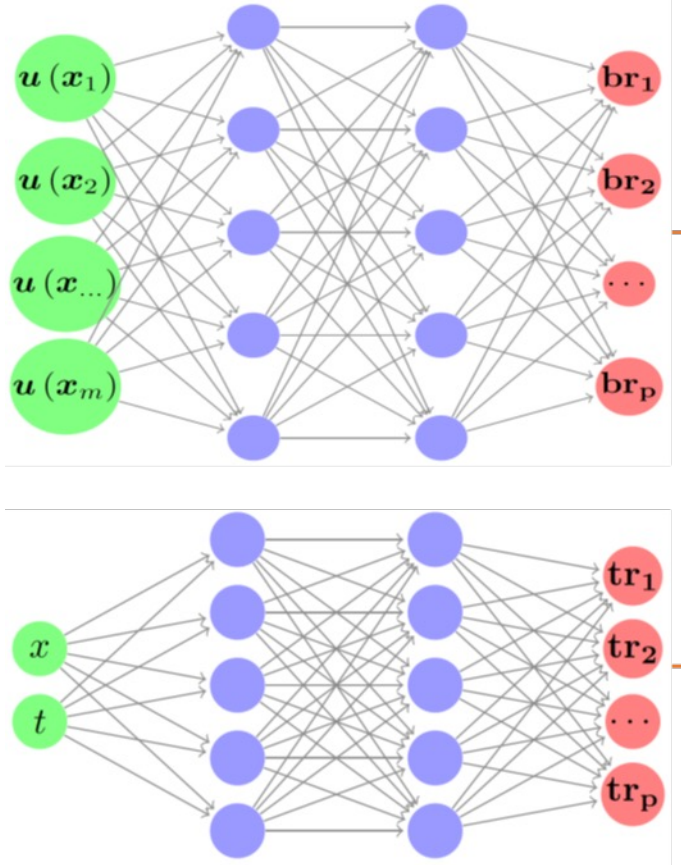
- Sensors $(m) = 10$

Building a DeepONet Model





$$\mathcal{G}_\theta : f(x,t) \rightarrow w(x,t)$$



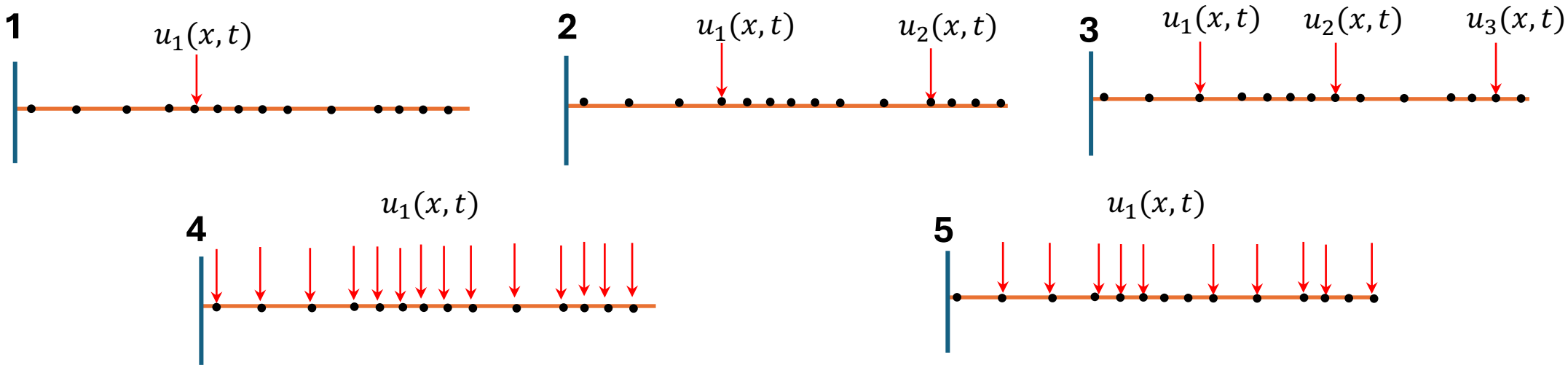
$$\mathcal{G}_\theta(u)(x,t)$$

$$\sum_{i=1}^{p} br_i * tr_i$$

Data Loss → Loss ($\mathcal{L}$) → Minimize $\mathcal{L}$

- Sensors$(m) = 15$
- Query Points

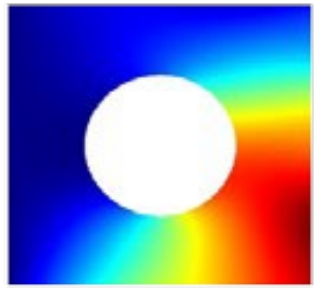
Preparing Inputs to the Branch Network



	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15
1	0	0	0	0	u_1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	u_1	0	0	0	0	0	0	0	u_2	0	0	0
3	0	0	u_1	0	0	0	0	u_2	0	0	0	0	0	u_3	0
4	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1	u_1
5	0	u_1	u_1	u_1	u_1	u_1	0	0	u_1	u_1	u_1	u_1	u_1	0	u_1

DeepONet to learn Linear Elastic solution on square domain

$E_s = 300e5,\ \nu_s = 0.3$



Random RHS

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(0, \mathcal{K}(\boldsymbol{x}, \boldsymbol{x}')\right)$$

$$\mathcal{K}(\boldsymbol{x}, \boldsymbol{x}') = \exp\left[-\frac{(\boldsymbol{x}-\boldsymbol{x}')^2}{2l^2}\right]$$

$$l = 0.12, \boldsymbol{x}, \boldsymbol{x}' \in [0,1]^2$$

Governing Equation

$$\nabla . \sigma + f(\boldsymbol{x}) = 0$$

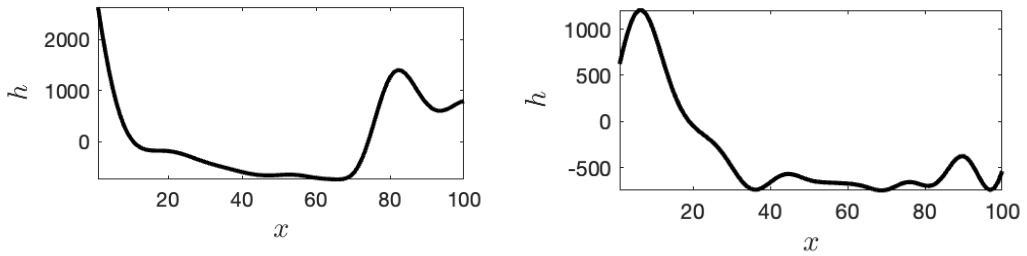
$$u(\boldsymbol{x}) = v(\boldsymbol{x}) = 0 \ \ \forall\ x = 0$$

$$u(\boldsymbol{x}) = f(\boldsymbol{x}) \ \ \forall\ x = 1$$

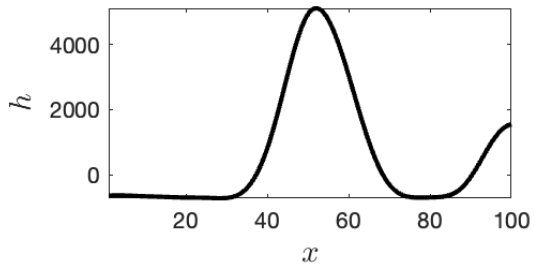
Objective:

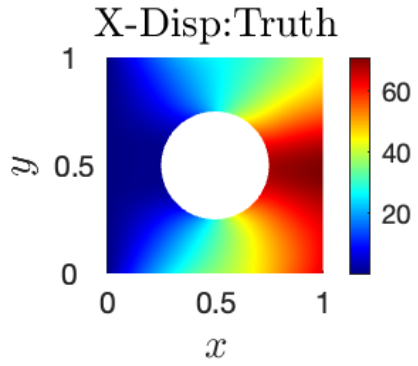
$$\mathcal{G}: f(\boldsymbol{x}) \rightarrow [u(\boldsymbol{x}), v(\boldsymbol{x})]$$

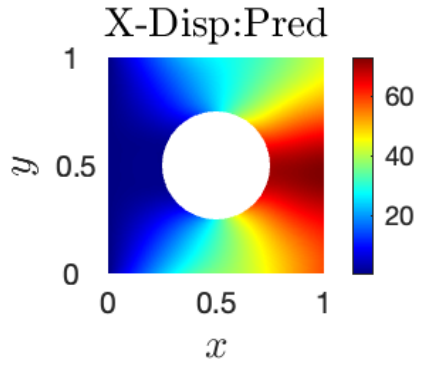
Sample RHS Function

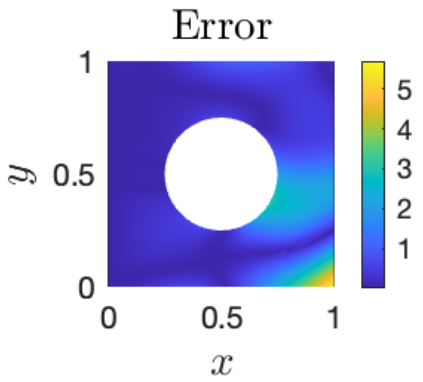


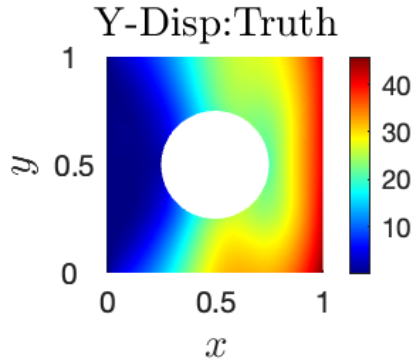
DeepONet to learn Linear Elastic solution on square domain

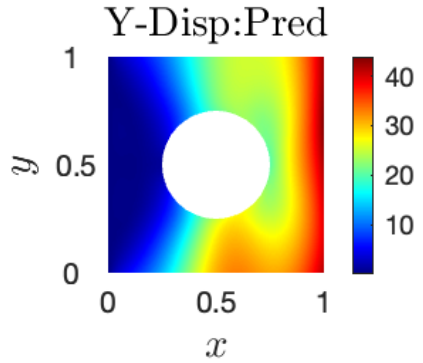


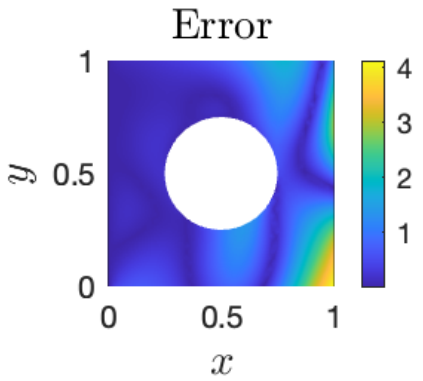




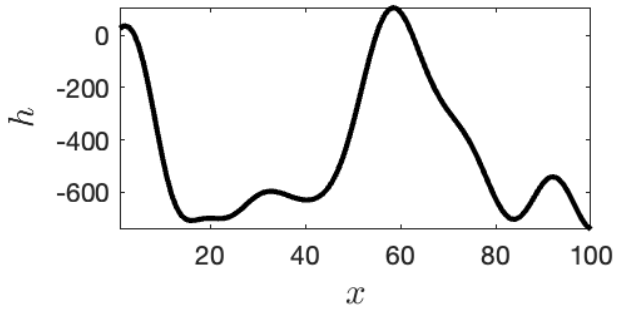


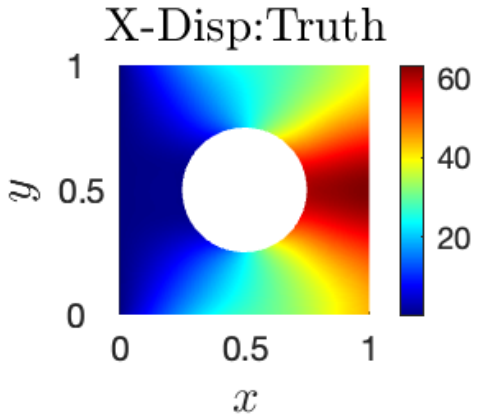


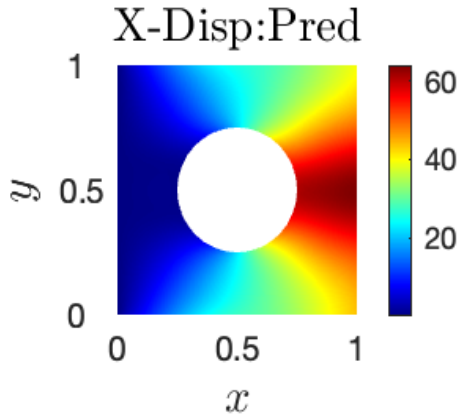


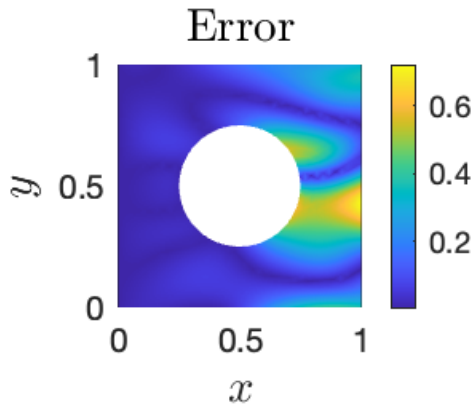


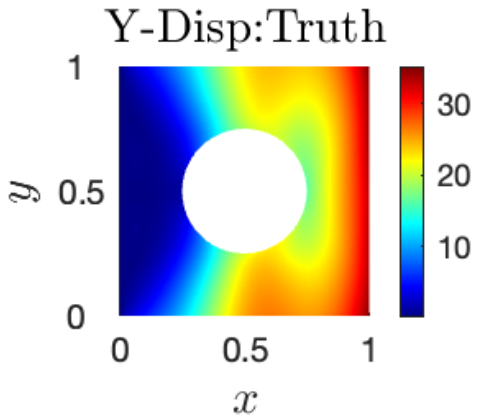
DeepONet to learn Linear Elastic solution on square domain

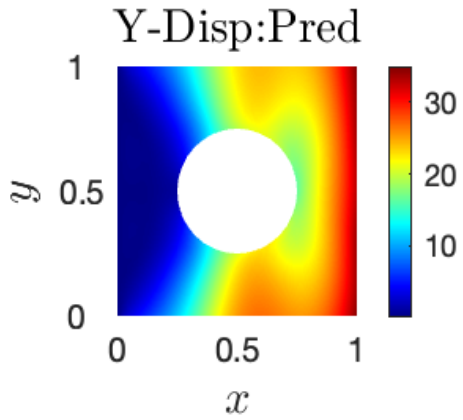


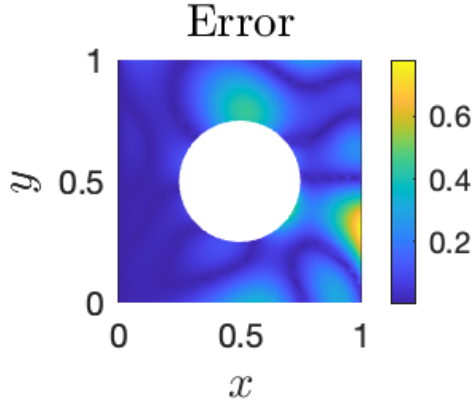








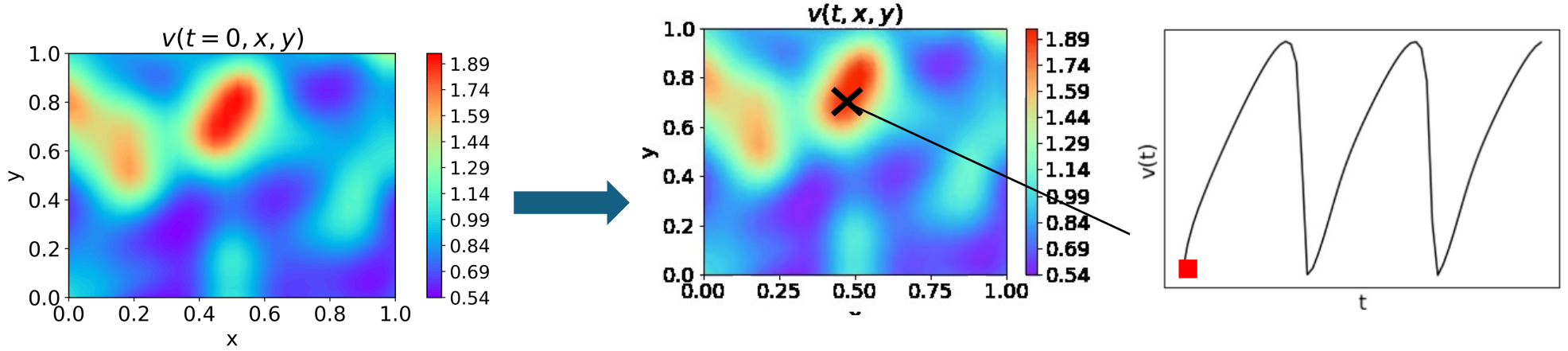




Variants of DeepONet

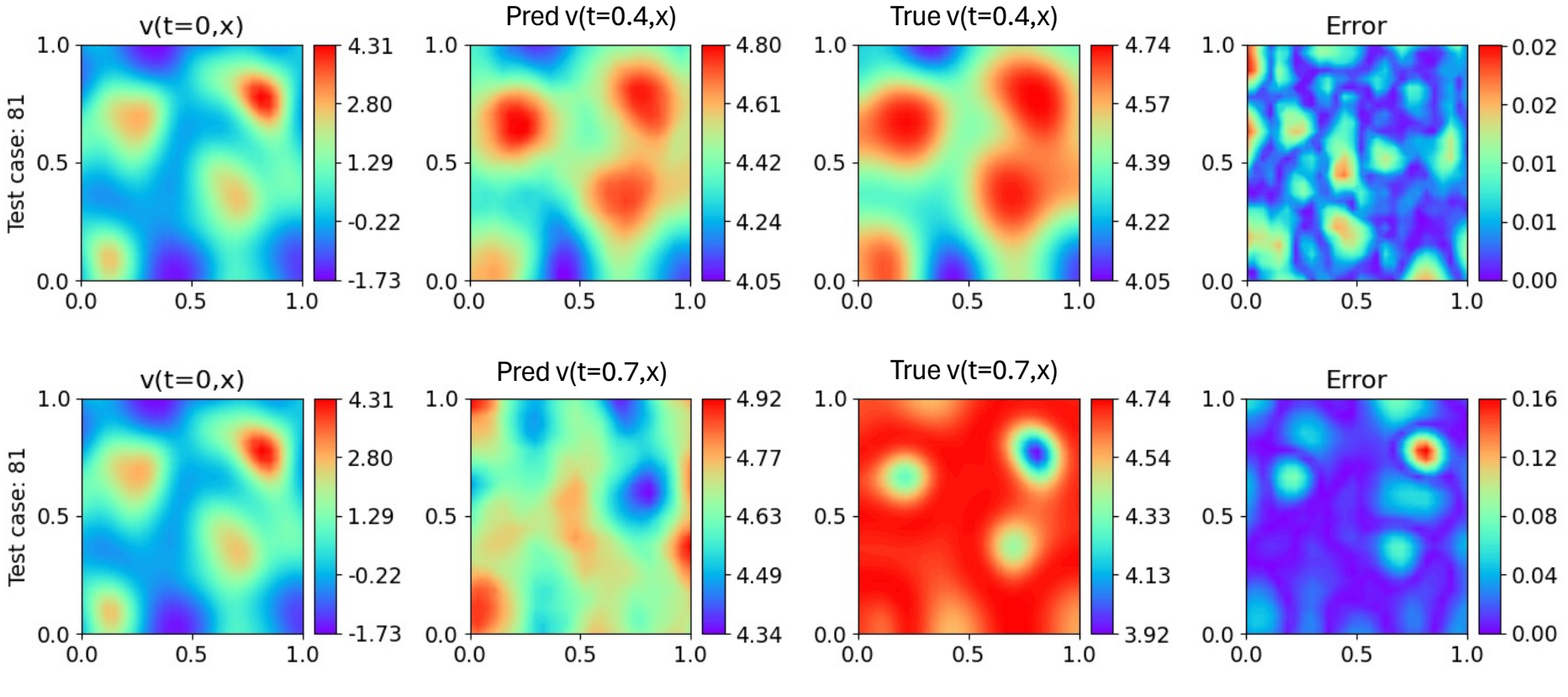
- POD-DeepONet: Employs the POD modes of the solutions of the training samples as inputs to the trunk net. The trunk net is non-trainable.

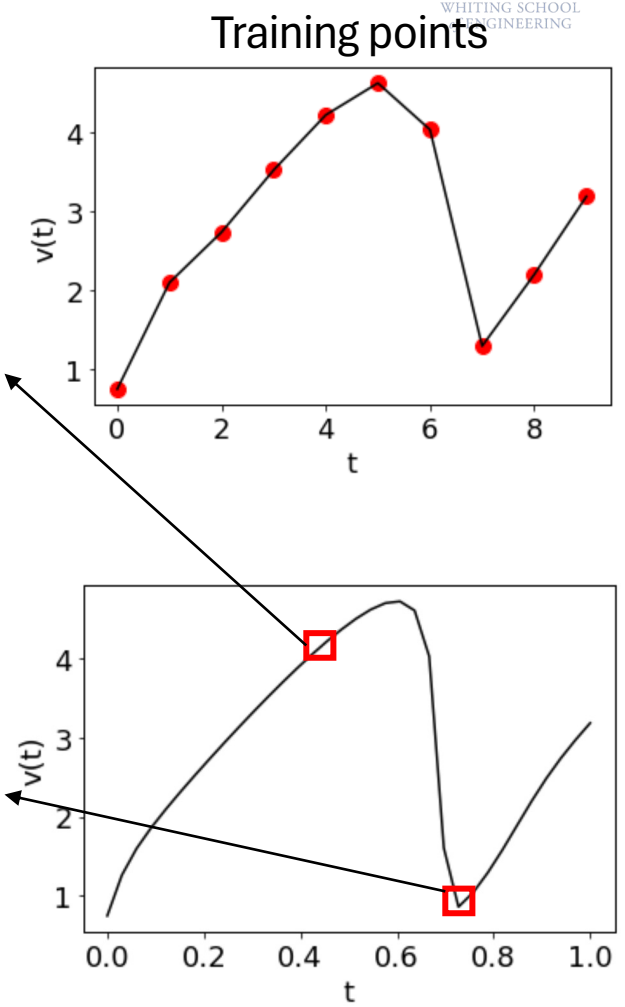
- SA-DeepONet: To balance difference components (boundary conditions and inner domain) of a loss function, a weighing term (λ) is introduced such that: ($\mathcal{L} = \mathcal{L}_b + \lambda\mathcal{L}_d$). SA-DeepONet considers λ as a variables and updates it during optimization.



Kontolati, Katiana, Somdatta Goswami, Michael D. Shields, and George Em Karniadakis. "On the influence of over-parameterization in manifold based surrogates and deep neural operators." Journal of Computational Physics 479 (2023): 112008.

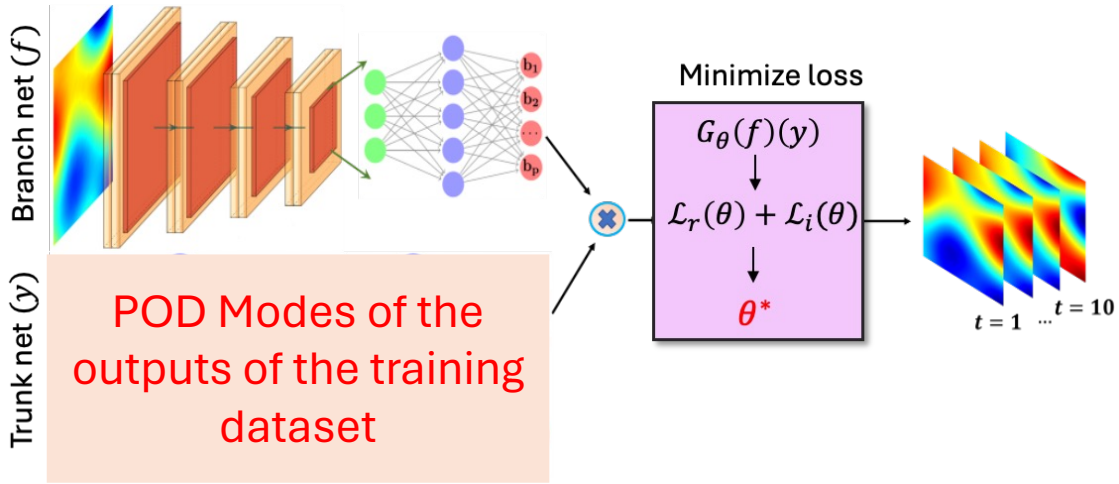
SA-DeepONet

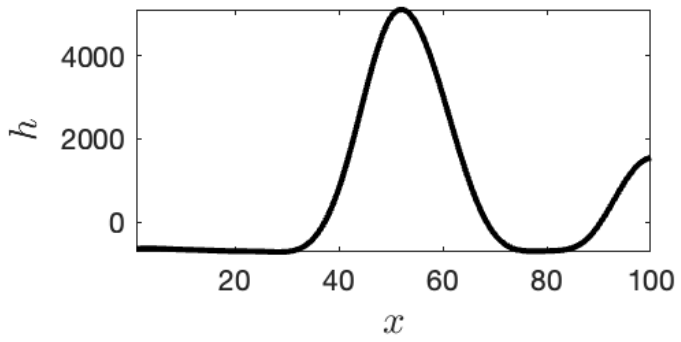

v(t=0,x)
Pred v(t=0.4,x)
True v(t=0.4,x)
Error
Test case: 81
v(t=0,x)
Pred v(t=0.7,x)
True v(t=0.7,x)
Error
Test case: 81

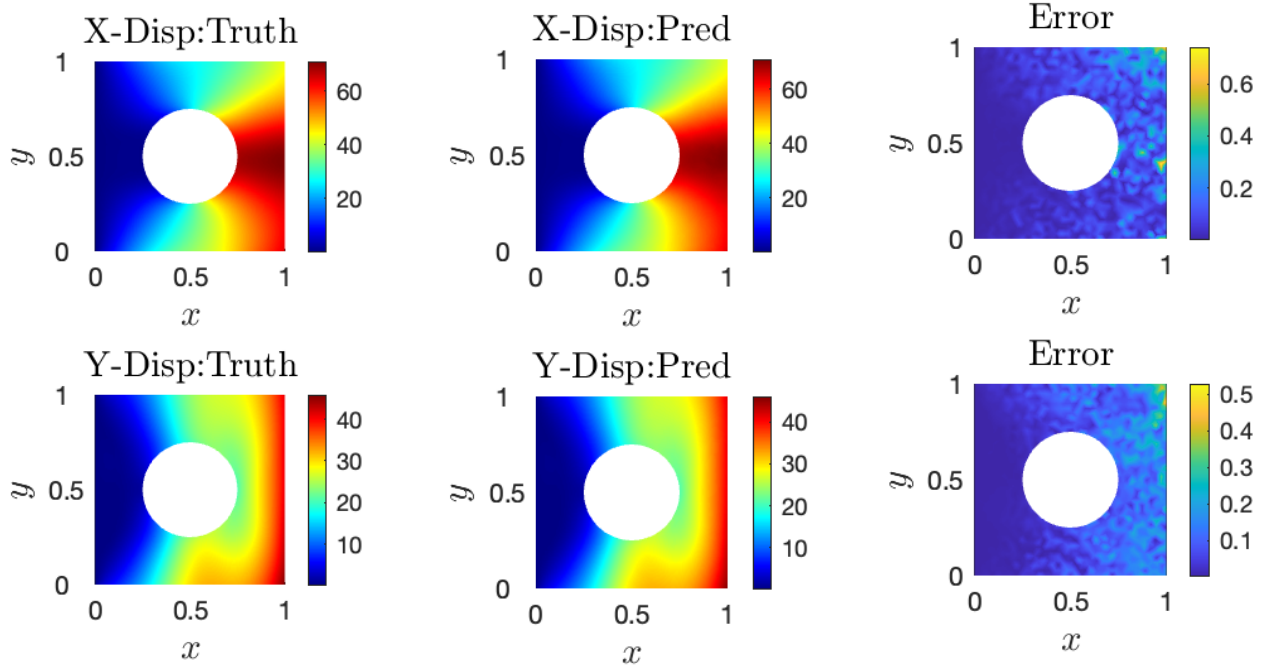

Training points
v(t)
t
v(t)
t

POD-DeepONet

1. Compute the POD modes of the outputs of the training dataset.
2. Train just the branch network.
   The trunk network is non-trainable.







SA-DeepONet

Original loss function:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(u_i(\theta) - \mathcal{G}_\theta(\mathbf{v}_i)(\theta)\right)^2$$

Weighted loss function:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N} w_j\left(u_i(\theta) - \mathcal{G}_\theta(\mathbf{v}_i)(\theta)\right)^2 \Longrightarrow$$

$$\mathcal{L}(\theta, \boldsymbol{\lambda}) = \sum_{i=1}^{N} g(\boldsymbol{\lambda})\left(u_i(\theta) - \mathcal{G}_\theta(\mathbf{v}_i)(\theta)\right)^2$$

Masking function

Optimization problem:

$$\min_{\theta} \max_{\boldsymbol{\lambda}} \mathcal{L}(\theta, \boldsymbol{\lambda})$$

Updating the masking function:

$$\boldsymbol{\lambda}^{k+1} = \boldsymbol{\lambda}^{k} + \eta_{\lambda}\nabla_{\lambda}\mathcal{L}(\theta, \boldsymbol{\lambda})$$

Gradient ascent

Constraints on the masking function:

$$g(\boldsymbol{\lambda}_i) > \mathbf{0}$$

Variants of DeepONet

- DeepONet with history: For Explicit time-step problems, to predict the $u_{t+\Delta t}$, $[u_t, u_{t-1}, \ldots u_0]$ is required. In DeepONet with History, the branch network is autoregressively update with predecided number of history terms.

- Wavelet DeepONet (W-DeepONet): Learning the operator network on wavelet decomposed input and output functions improves the optimization process.

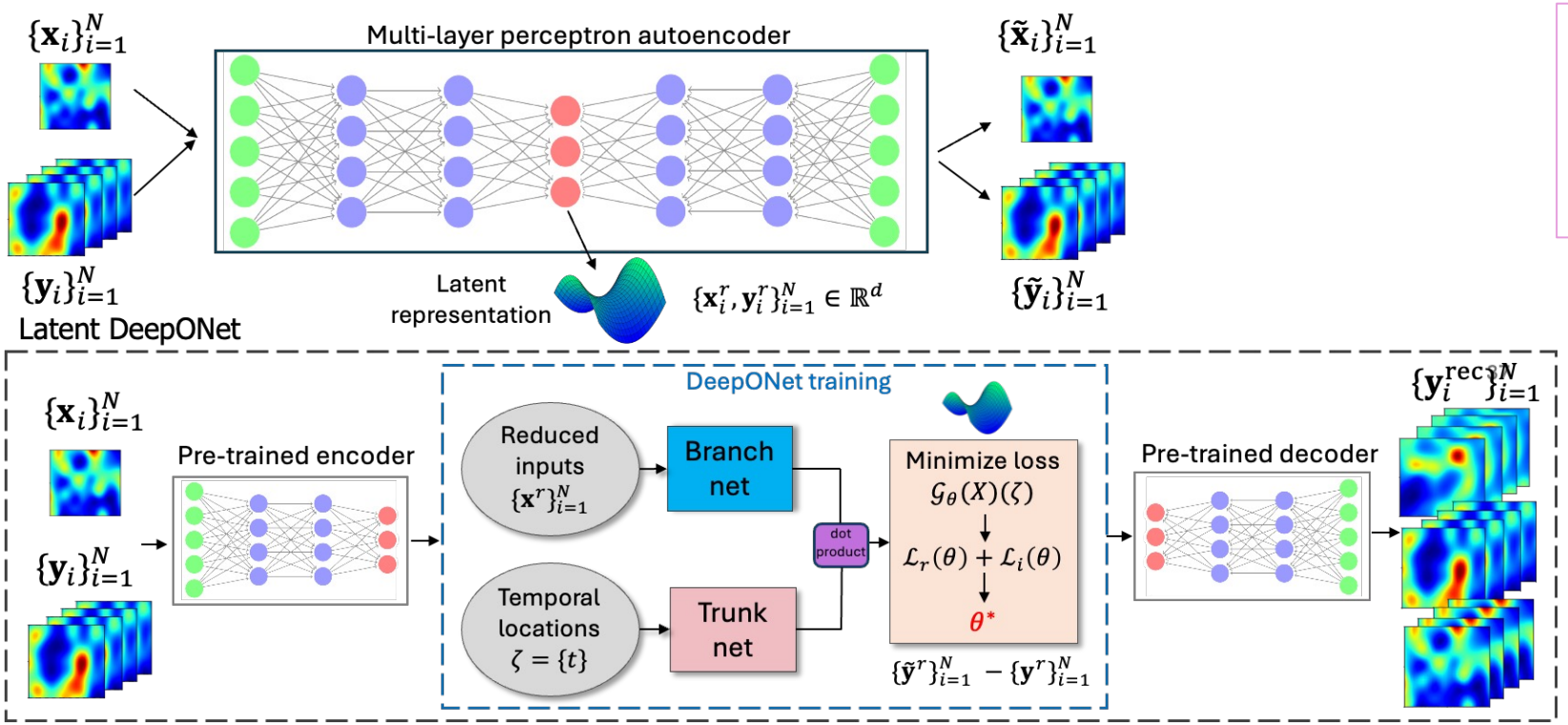
  Cao, Qianying, Somdatta Goswami, Tapas Tripura, Souvik Chakraborty, and George Em Karniadakis. "Deep neural operators can predict the real-time response of floating offshore structures under irregular waves." Computers & Structures 291 (2024): 107228.

- Physics-Informed DeepONet (PI-DeepONet): To reduce the over-reliance on data in training Nos, PI-DeepONet llows to embed the known physics into the loss function with little or no labelled dataset during training.

1. Goswami, Somdatta, Minglang Yin, Yue Yu, and George Em Karniadakis. "A physics-informed variational DeepONet for predicting crack path in quasi-brittle materials." Computer Methods in Applied Mechanics and Engineering 391 (2022): 114587.
2. Wang, Sifan, Hanwen Wang, and Paris Perdikaris. "Learning the solution operator of parametric partial differential equations with physics-informed DeepONets." Science advances 7, no. 40 (2021): eabi8605.

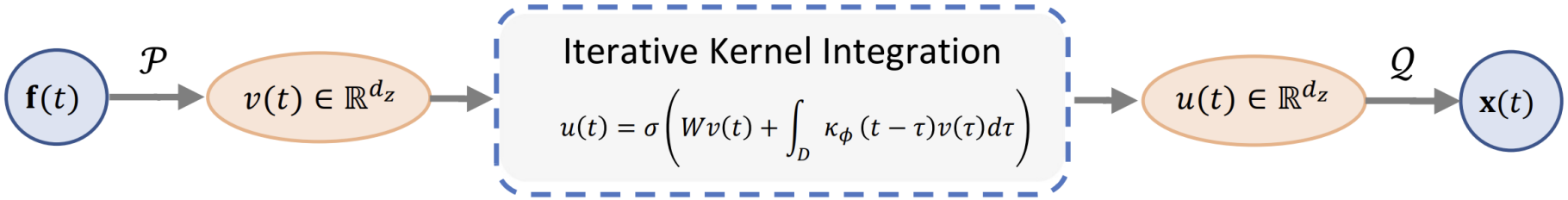
Variants of DeepONet

- Latent DeepONet (L-DeepONet) : Learning the operator latent spaces.



Kontolati, Katiana, Somdatta Goswami, George Em Karniadakis, and Michael D. Shields. "Learning in latent spaces improves the predictive accuracy of deep neural operators." Nature Communications

Integral Operators



- Graph Neural Operator [1]
- Low-rank Neural Operator
- Multipole Graph Neural Operator
- Fourier Neural Operator [2]
- Wavelet Neural Operator [3]
- Laplace Neural Operator [4]

[1] Kovachki, N. et al. (2021). Neural operator: Learning maps between function spaces.

[2] Li, Z. et al. (2020). Fourier neural operator for parametric partial differential equations.

[3] Tripura, T. and Chakraborty, S. (2022). Wavelet neural operator: a neural operator for parametric partial differential equations.

[4] Cao, Q. Goswami, S. and Karniadakis, G.E. (2024). Laplace Neural Operator for PDEs

Fourier neural operator (FNO)

Iterative algorithm:

$$v_{t+1}(x) = \sigma\big(W v_t(x) + (\mathcal{K}(\alpha;\varphi)v_t)(x)\big)$$

for $t = 1, \dots, T-1$

Fourier integral operator:

$$(\mathcal{K}(\alpha;\varphi)v_t)(x) = \mathcal{F}^{-1}\left(R_\varphi \cdot (\mathcal{F}v_t)\right)(x)$$

$\forall x \in \Omega$

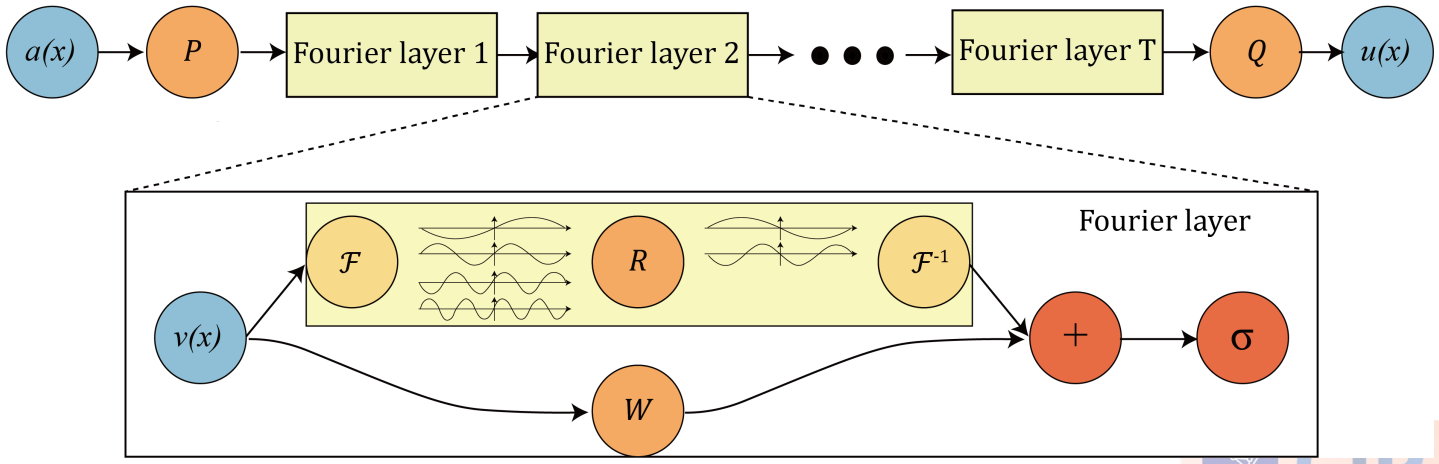
Fourier transform of function $f: \Omega \rightarrow \mathbb{R}^{d_v}$

$$(\mathcal{F}f)_j(k) = \int_\Omega f_j(x) e^{-2i\pi\langle x,k\rangle} dx$$

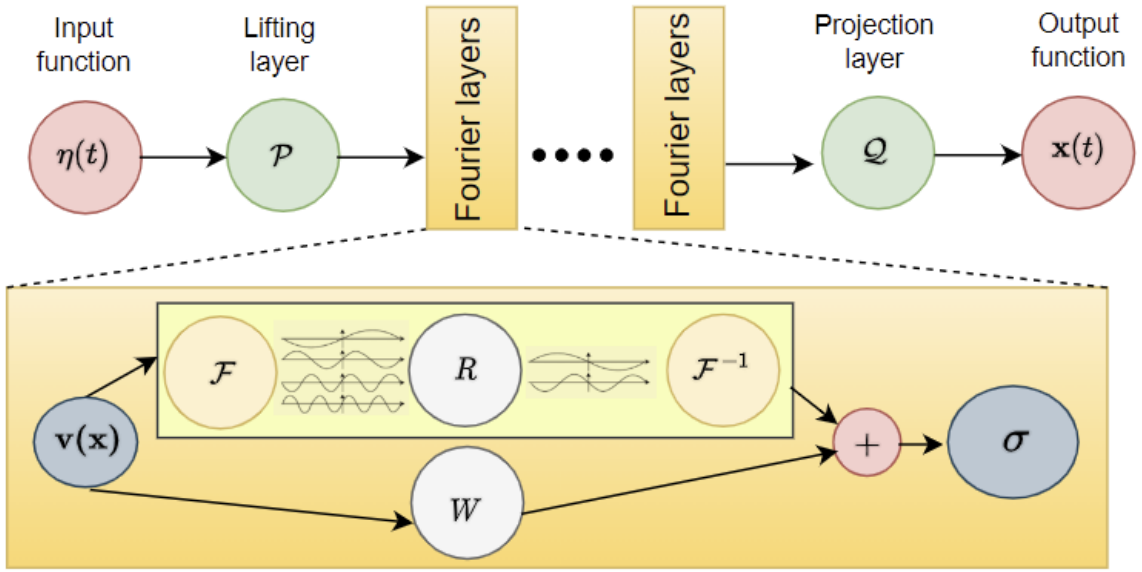
$$(\mathcal{F}^{-1}f)_j(x) = \int_\Omega f_j(k) e^{2i\pi\langle x,k\rangle} dk$$

for $j = 1, \dots, d_v$ k: FFT modes

- Employs convolution via fast Fourier transform (FFT)
- The Fourier layer just consists of three steps:
  - ✓ Fourier transform $\mathcal{F}$
  - ✓ Linear transform on the lower Fourier modes, $\mathcal{R}$
  - ✓ Inverse Fourier transform $\mathcal{F}^{-1}$
- For time-dependent PDEs (2D in space, 1D in time):
  1. **FNO-3D**: 3D → 3D (3D convolutions)
  2. **FNO-2D**: 2D → 2D (2D + RNN)



Fourier Neural Operator



Advantage

- Cheap computational cost due to FFT.
- A small size of training samples.

Limitation:

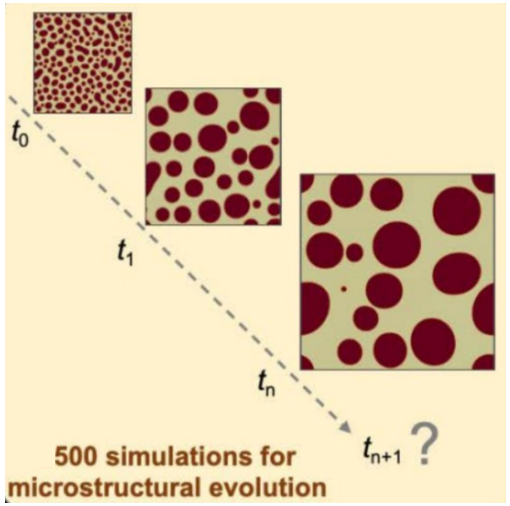
- For FFT, the input is assumed to be periodic.
- Only capture steady-state responses.

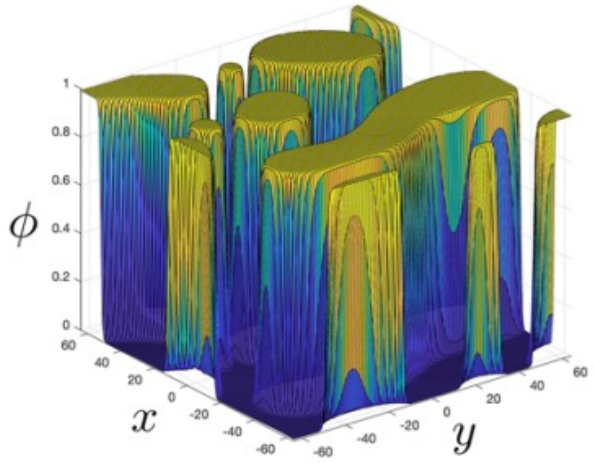
Accelerating traditional methods

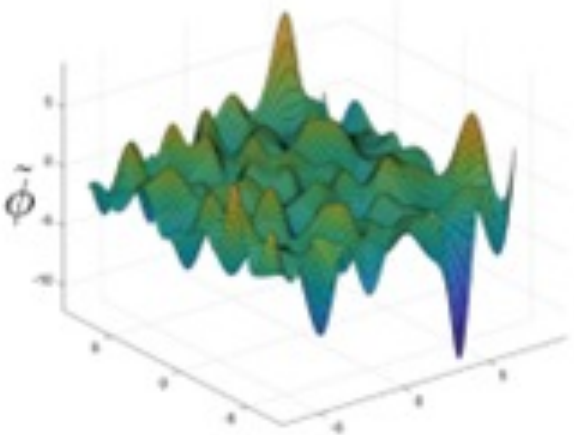
Microstructure evolution of a two-phase mixture during Spinodal decomposition

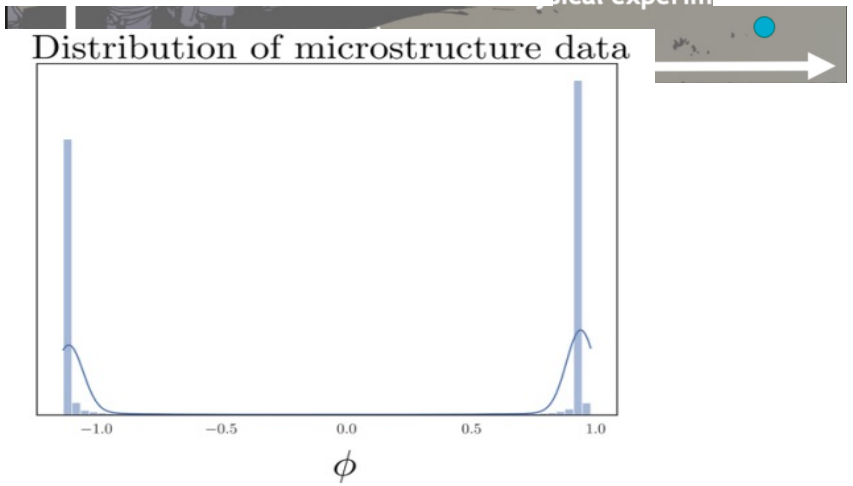
$$\frac{\partial \phi_i}{\partial t} = \nabla \cdot (M_{ij} \nabla \frac{\delta F}{\delta \phi_j})$$

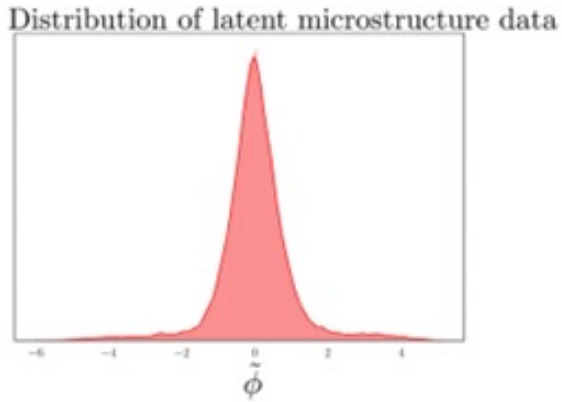
High-Fidelity Simulations









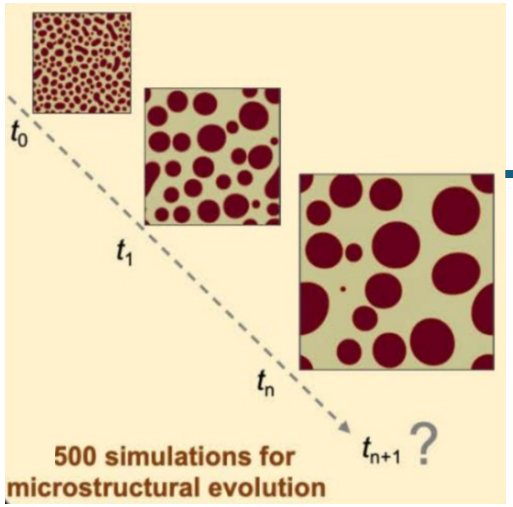


Accelerating traditional methods

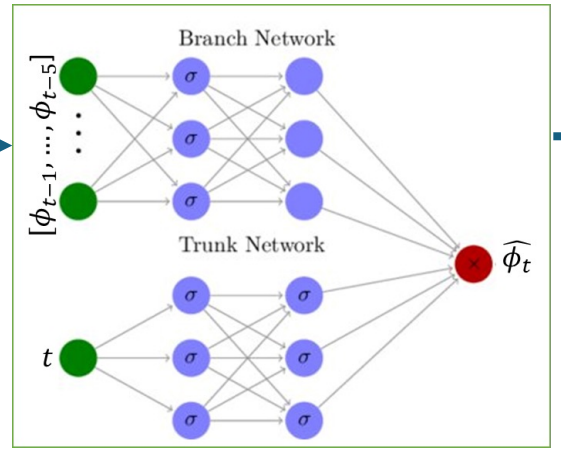
Non-linear microstructure evolution of a two-phase mixture during Spinodal decomposition

$$\frac{\partial \phi_i}{\partial t} = \nabla \cdot (M_{ij} \nabla \frac{\delta F}{\delta \phi_j})$$

High-Fidelity Simulations

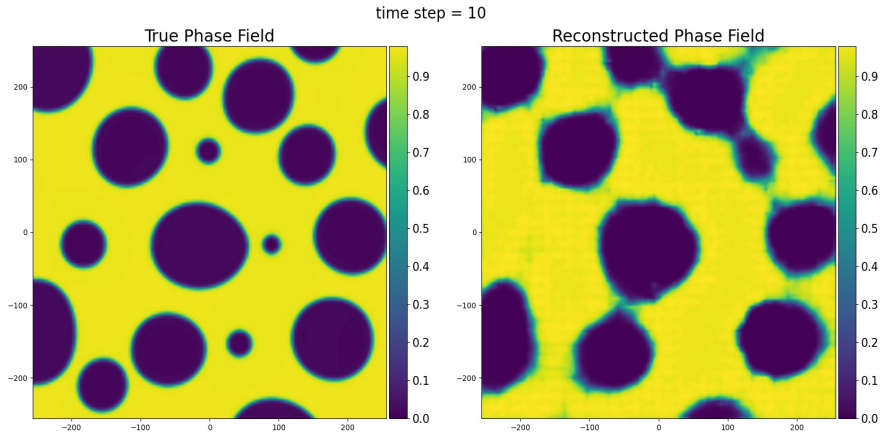


Convolutional Autoencoders (Reduced dimension = 100)



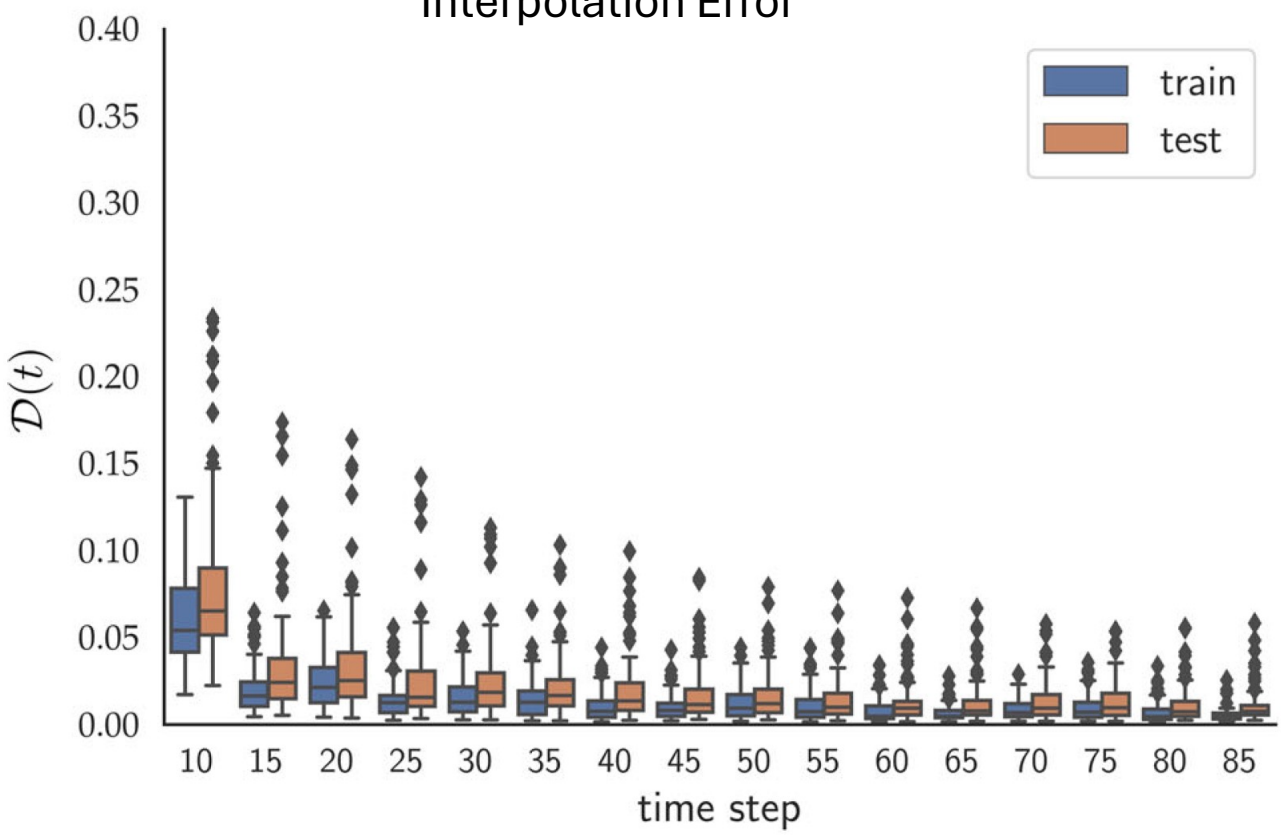
Pre-trained Transposed Convolutional decoder

ϕ_t

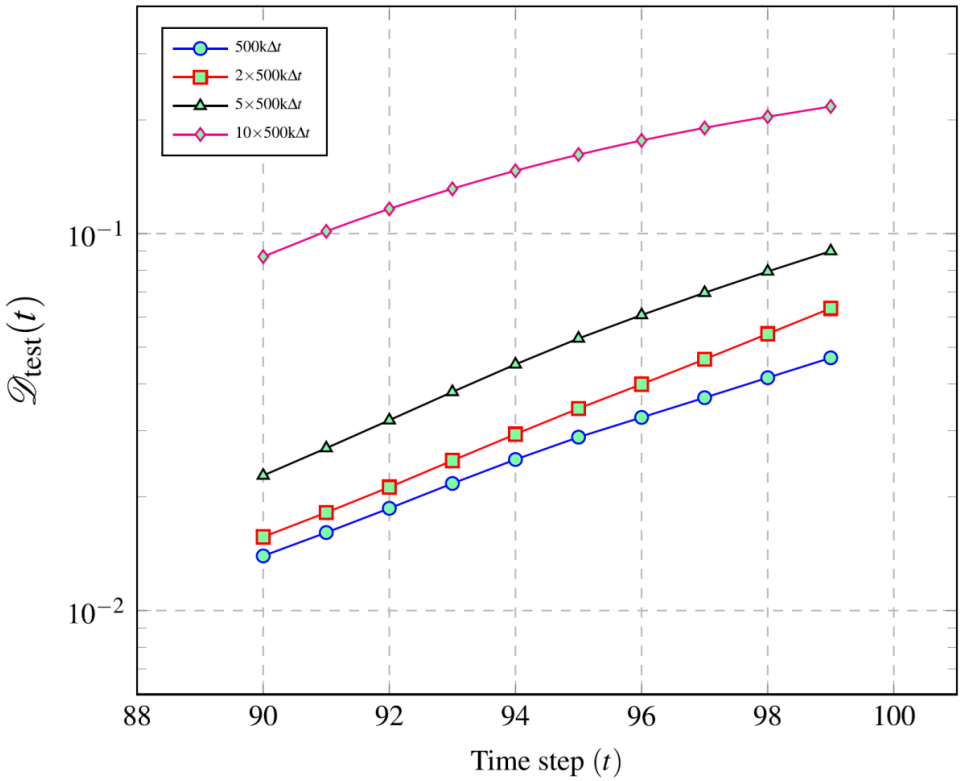


Accelerating traditional methods

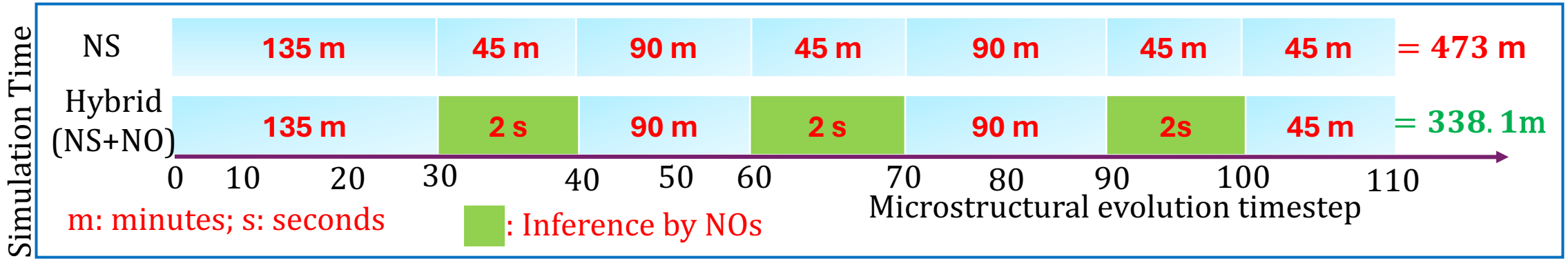
Interpolation Error

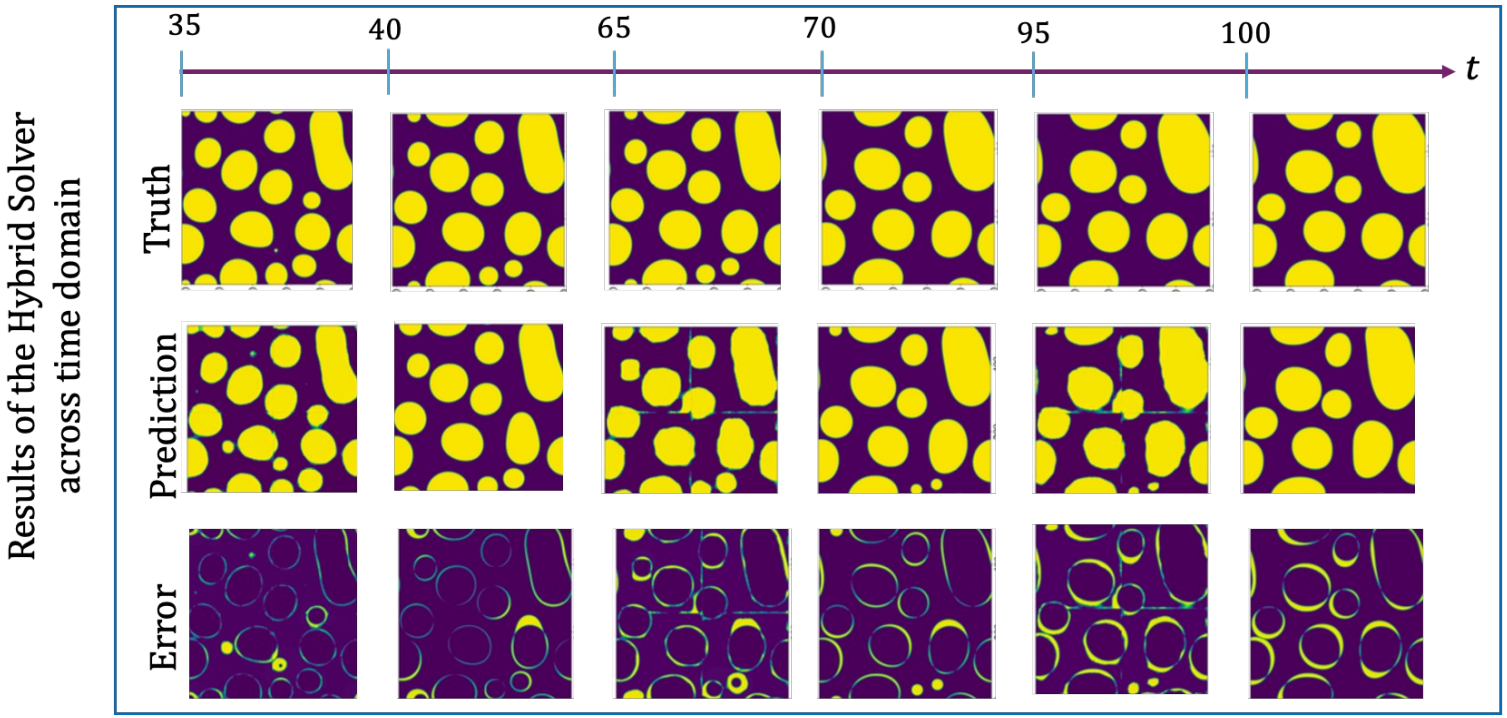


Extrapolation Error



Designing a Hybrid Solver





Oommen, Vivek, Khemraj Shukla, Somdatta Goswami, Rémi Dingreville, and George Em Karniadakis. "Learning two-phase microstructure evolution using neural operators and autoencoder architectures." npj Computational Materials 8, no. 1 (2022): 190.

Acknowledgement


Ponkrshnan Thiagarajan


Connor Krill


Georgios Pasparakis

Work in Collaboration with:

- Prof. George Karniadakis, Brown University
- Dr. Khemraj Shukla, Brown University
- Vivek Oommen, Brown University
- Dr. Remi Dingreville, Sandia National Lab

Github Repo

Thank you!

Multi-Modal Data Driven and Physics-Informed Machine Learning with Uncertainty for Materials Applications

Nat Trask[1], Somdatta Goswami[2], Michael D. Shields[2]

[1]University of Pennsylvania, Division of Mechanical Engineering and Applied Mechanics

[2]Johns Hopkins University, Department of Civil and Systems Engineering

SIAM Conference on Mathematical Aspects of Materials Science (SIAM -MS24)

21 May 2024

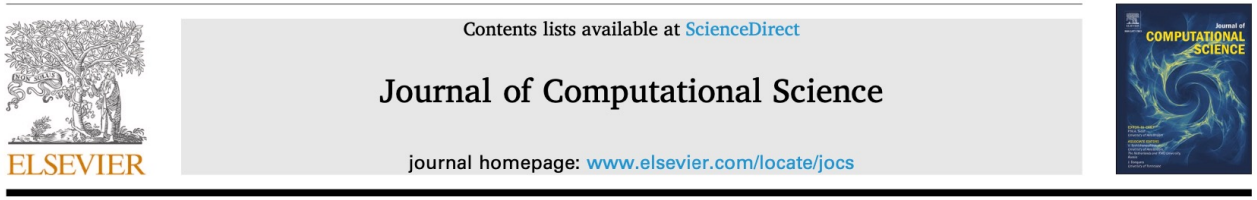
SIAM MS24

UQpy

- A collection of Python modules used for uncertainty quantification and propagation
  - Includes commonly applied methods and new developments
  - Serves as a UQ toolbox and a Python development environment
- Developed collaboratively by members of SURG
  - Contributors: Ponkrshnan Thiagarajan, Connor Krill, Georgios Pasparakis, Dimitris Tsapetis, Dimitris Giovanis, Audrey Olivier, Aakash Bangalore Satish, Lohit Vandanapu, Mohit Chauhan, Katiana Kontolati, Ketson R.M. dos Santos
- Contributions from researchers around the world
  - Dr. Dimitris Loukrezis (Siemens), Prof. Michael Gardner (University of Nevada, Reno), Dr. Lukas Novak (Brno University of Technology), Ulrich Römer, Julius Schultz (TU Braunschweig)

UQpy

- Version control through git (requires Python 3)
  - Version 4.1.1 available for download/installation via GitHub (https://github.com/SURGroup/UQpy)
- Available on the Python Package Index (PyPI) and Conda
- Thorough documentation and examples:



UQpy: A general purpose Python package and development environment for uncertainty quantification

Original Software Publication

UQpy v4.1: Uncertainty quantification with Python

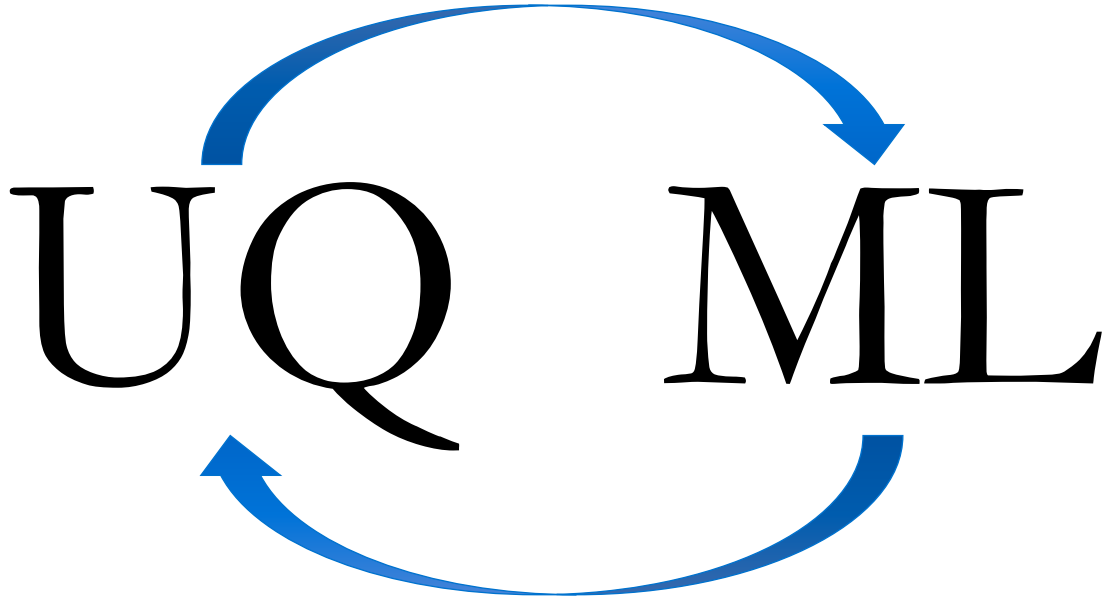
Dimitrios Tsapetis [a], Michael D. Shields [a,*], Dimitris G. Giovanis [a], Audrey Olivier [b], Lukas Novak [c], Promit Chakroborty [a], Himanshu Sharma [a], Mohit Chauhan [a], Katiana Kontolati [a], Lohit Vandanapu [a], Dimitrios Loukrezis [d], Michael Gardner [e]

UQpy-ML

- New module on UQ for Scientific Machine Learning (SciML) under development
- Module will add capabilities for:
  - Neural Operators
  - Physics Informed Neural Networks (PINNs)
  - Physics Informed Neural Operators
  - Bayesian Neural Networks
- Methods being built on the PyTorch library

- Code from this Mini-tutorial can be found at: https://github.com/ponkrshnan/SIAM_MS24_tutorial or

UQ for ML



ML for UQ

UQ for ML:

- Aim to quantify uncertainty in ML predictions
  - Regression uncertainty
  - Classification uncertainty
  - Etc.
- Bayesian Neural Networks
  - Basic construction
  - Applications to Operator Networks, PINNs, etc.
- Gaussian Process Regression Models
  - Basic construction
  - Including physical constraints for materials applications

Bayesian Neural Networks

$$\mathbb{L}(\mathcal{X}, \lambda, \boldsymbol{u}(\boldsymbol{x}); \boldsymbol{\theta}) = g$$
$$\boldsymbol{x} = (x, y) \in \Omega, \lambda \in D, \lambda = 1$$

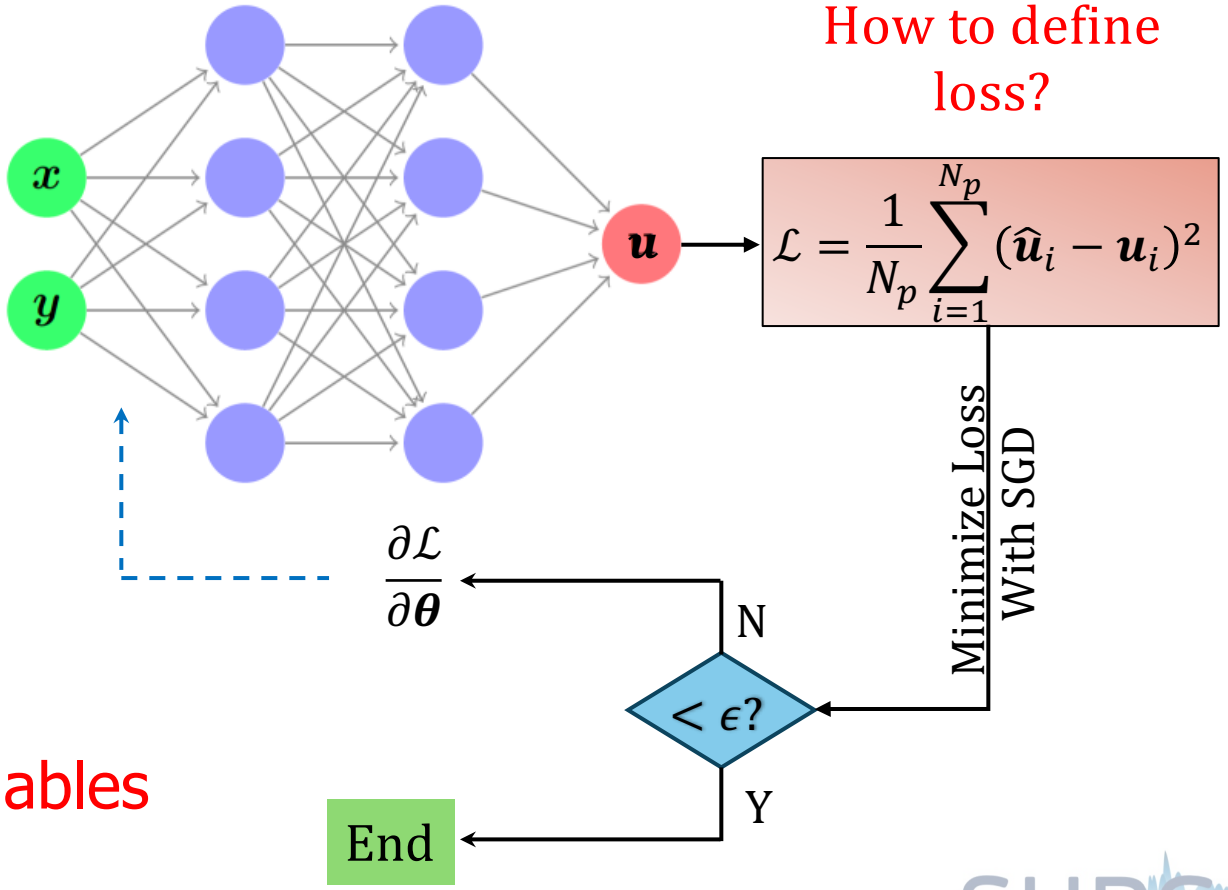
$\hat{\boldsymbol{u}}$: Predicted solution

$\boldsymbol{\theta}$: Weights and biases

N_p: # of collocation points

$(\boldsymbol{x}, \boldsymbol{u}(\boldsymbol{x}))$: Labelled dataset

Make $\boldsymbol{\theta}$ random variables



Bayesian Neural Networks

Bayes' Rule:

$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)}$$

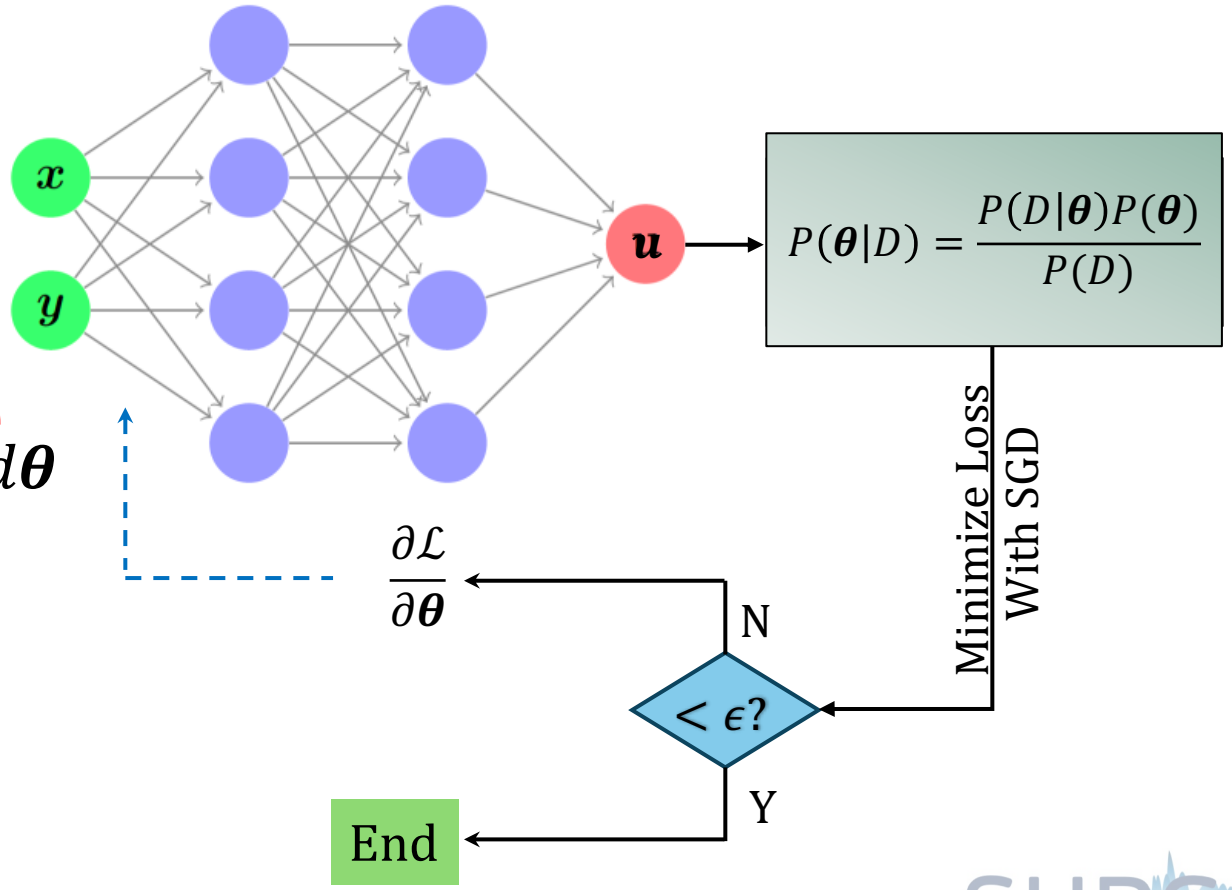
Predictive Probability of Output, $\mathbf{u}$:

$$P(\mathbf{u}^*|\boldsymbol{x}^*, D) = \int P(\mathbf{u}^*|\boldsymbol{x}^*, \boldsymbol{\theta})P(\boldsymbol{\theta}|D)d\boldsymbol{\theta}$$

Mean Prediction:

$$E[\mathbf{u}^*|\boldsymbol{x}^*, D] = E_{p(\boldsymbol{\theta},D)}\left[E[\mathbf{u}^*|\boldsymbol{x}^*, \theta]\right]$$
$$= E_{p(\boldsymbol{\theta},D)}\left[f^{\boldsymbol{\theta}}(x^*)\right]$$

NN with weights $\boldsymbol{\theta}$



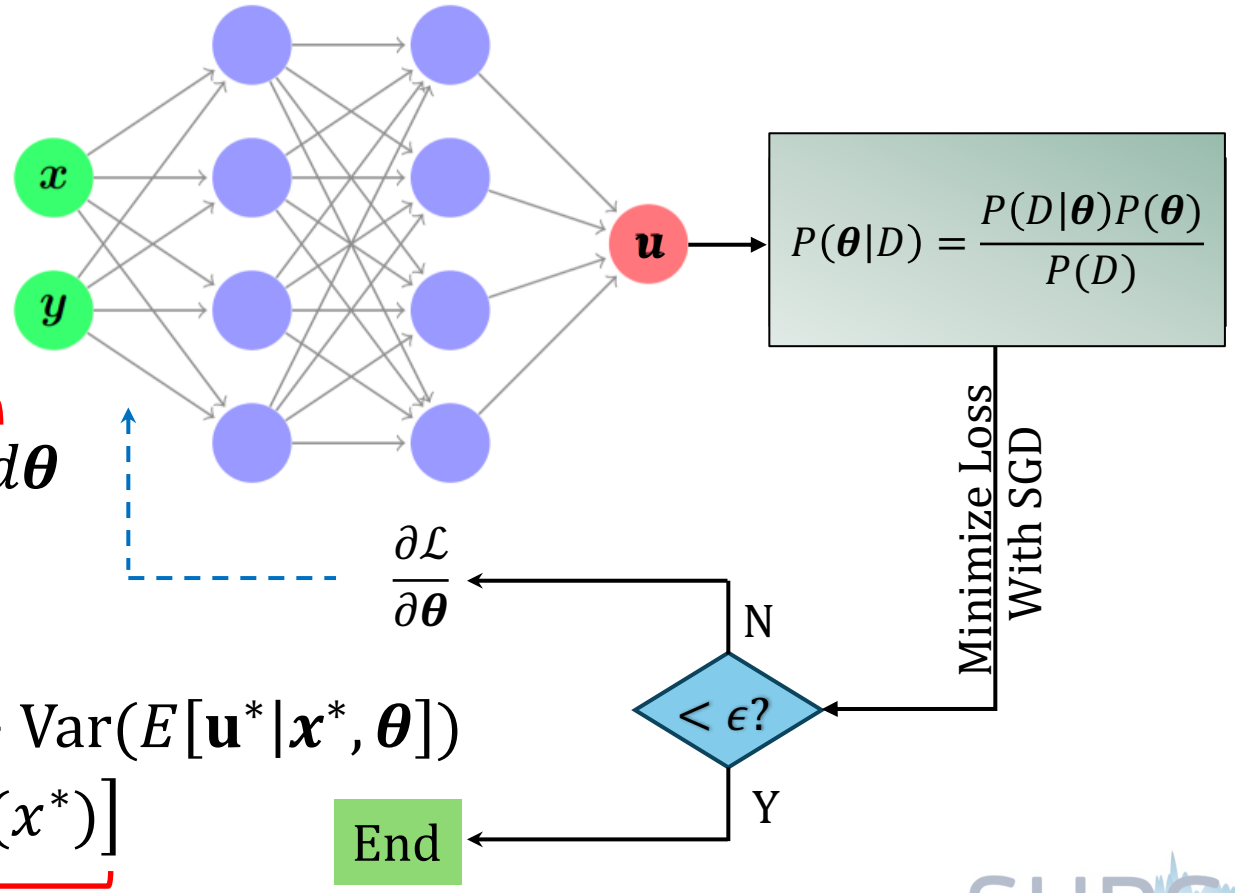
Bayesian Neural Networks

Bayes' Rule:

$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)}$$

Predictive Probability of Output, $\mathbf{u}$:

$$P(\mathbf{u}^*|\boldsymbol{x}^*, D) = \int P(\mathbf{u}^*|\boldsymbol{x}^*, \boldsymbol{\theta})P(\boldsymbol{\theta}|D)d\boldsymbol{\theta}$$



Variance of Prediction:

$$\begin{aligned}\mathrm{Var}[\mathbf{u}^*|\boldsymbol{x}^*, D] &= E_{p(\boldsymbol{\theta},D)}\left[\mathrm{Var}[\mathbf{u}^*|\boldsymbol{x}^*, \boldsymbol{\theta}]\right] + \mathrm{Var}(E[\mathbf{u}^*|\boldsymbol{x}^*, \boldsymbol{\theta}]) \\ &= \sigma_n(\boldsymbol{x}^*)^2 + \mathrm{Var}_{p(\boldsymbol{\theta},D)}\left[f^{\boldsymbol{\theta}}(x^*)\right]\end{aligned}$$

Aleatory Epistemic

Bayesian Neural Networks

Bayes' Rule:

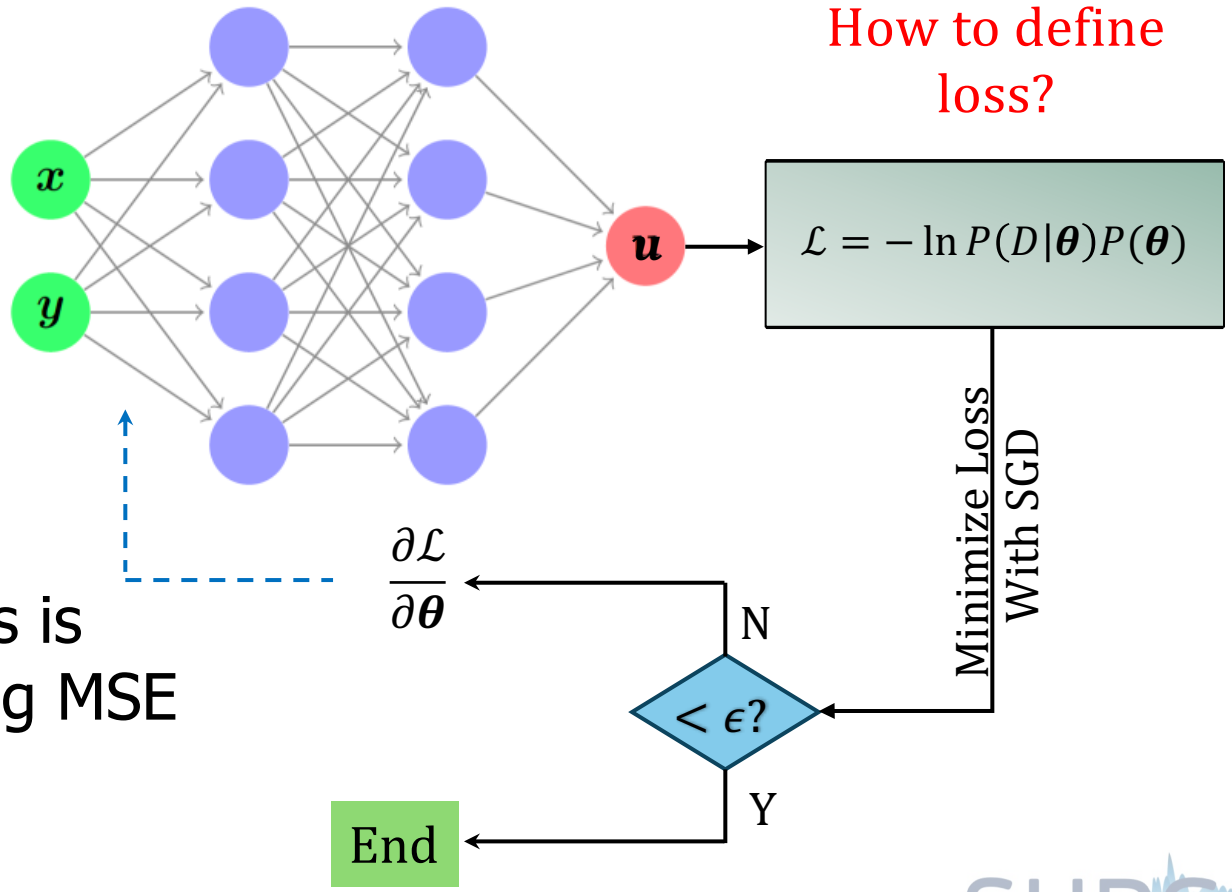
$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)}$$

Maximize log of posterior:

- Maximum a posteriori (MAP) estimate
- Equivalent to standard NN training

- Placing a Gaussian prior on the weights is mathematically equivalent to minimizing MSE with weight decay regularization[1]

Deterministic Point Estimate



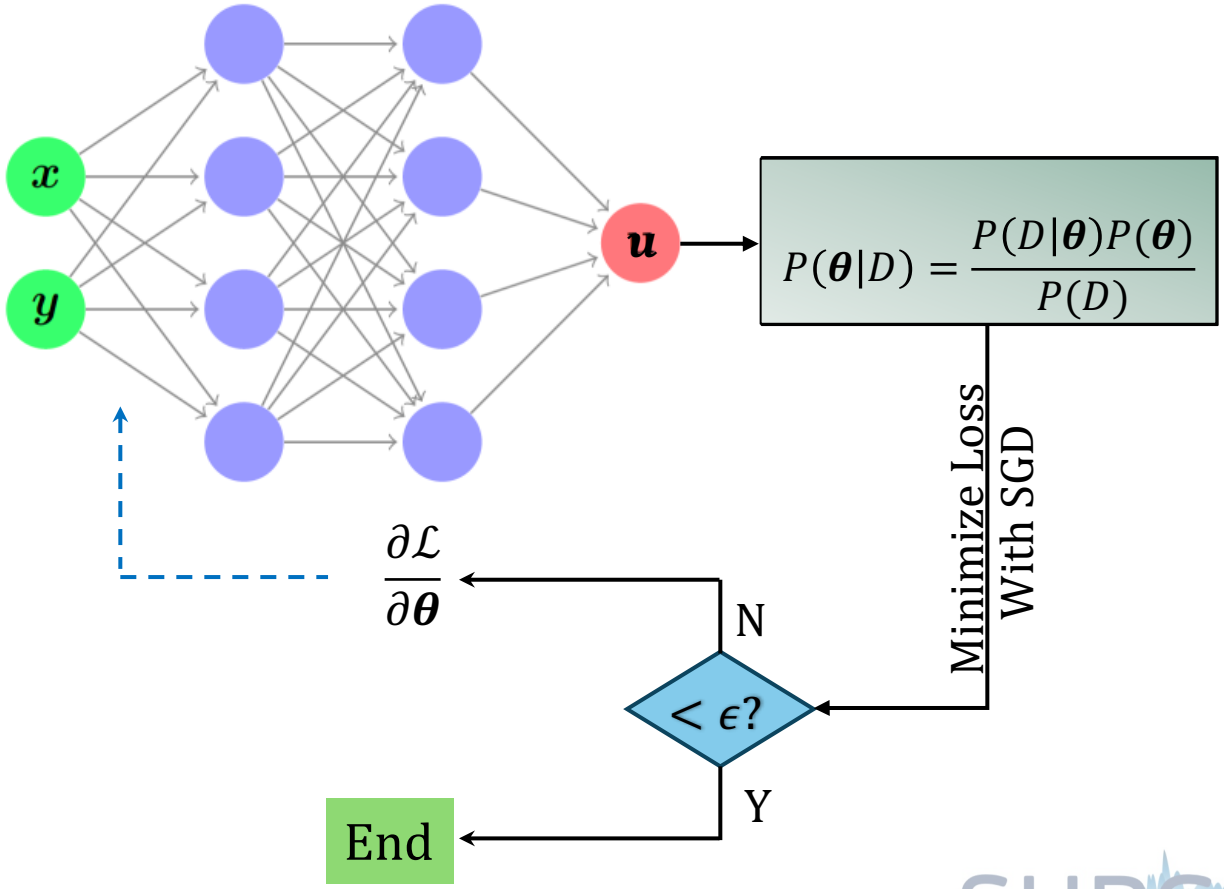
Posterior Sampling with MCMC

Bayes' Rule:

$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)}$$

Draw Samples from $P(\boldsymbol{\theta}|D)$:

- Define a Markov chain with stationary distribution $P(\boldsymbol{\theta}|D)$
- Random walk Metropolis Hastings
  - Very long run times
  - Very high rejection rate
  - Impractical for very high-dimensional distributions



Hamiltonian Monte Carlo[2]

- Define the Hamiltonian (total energy) as $H(\boldsymbol{q}, \boldsymbol{p}) = U(\boldsymbol{q}) + K(\boldsymbol{p})$
- The positions and momenta of the system evolve according to Hamilton's equations

$$\frac{d\boldsymbol{q}}{dt} = \frac{\partial H}{\partial \boldsymbol{p}}$$
$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H}{\partial \boldsymbol{q}}$$

- Define the probability distribution as $\pi(\boldsymbol{p}, \boldsymbol{q}) \propto e^{-H(\boldsymbol{p},\boldsymbol{q})} = e^{-U(\boldsymbol{q})} e^{-K(\boldsymbol{p})}$ with Potential and Kinetic Energy given by

$$U(\boldsymbol{q}) = -\log \pi(\boldsymbol{q}), \qquad K(\boldsymbol{p}) = \frac{1}{2}\boldsymbol{p}\boldsymbol{M}^{-1}\boldsymbol{p}$$

- We can solve this system by numerical time integration

$$\mathbf{q}(t+\Delta t) = \mathbf{q}(t) + \frac{\Delta t}{m_i}\,\mathbf{p}(t) + \frac{\Delta t^2}{2m_i}\,\dot{\mathbf{p}}(t) = \mathbf{q}(t) + \frac{\Delta t}{m_i}\,\mathbf{p}(t) - \frac{\Delta t^2}{2m_i}\,\frac{\partial H}{\partial \mathbf{q}(t)}\,,$$
$$\mathbf{p}(t+\Delta t) = \mathbf{p}(t) + \frac{\Delta t}{2}\left(\dot{\mathbf{p}}(t) + \dot{\mathbf{p}}(t+\Delta t)\right) = \mathbf{p}(t) - \frac{\Delta t}{2}\left(\frac{\partial H}{\partial \mathbf{q}(t)} + \frac{\partial H}{\partial \mathbf{q}(t+\Delta t)}\right),$$

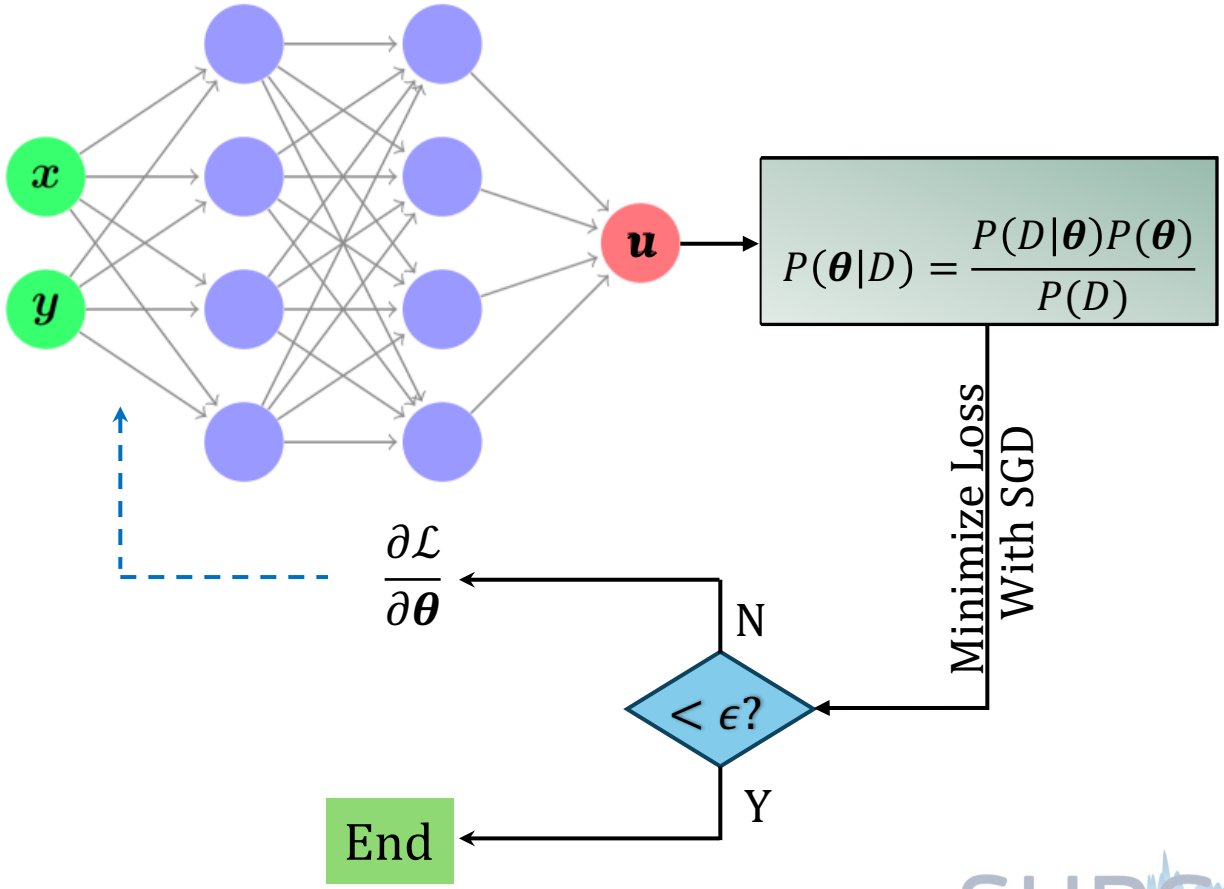
Posterior Sampling with MCMC

Bayes' Rule:

$$P(\boldsymbol{\theta}|D) = \frac{P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(D)}$$

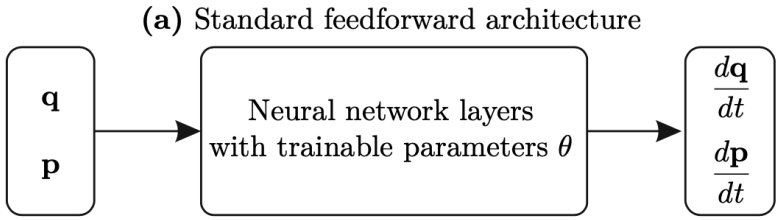
Draw Samples from $P(\boldsymbol{\theta}|D)$:

- Hamiltonian Monte Carlo
  - High acceptance rate
  - Requires time integration at every step
  - Need to compute gradients of the likelihood function for time integration

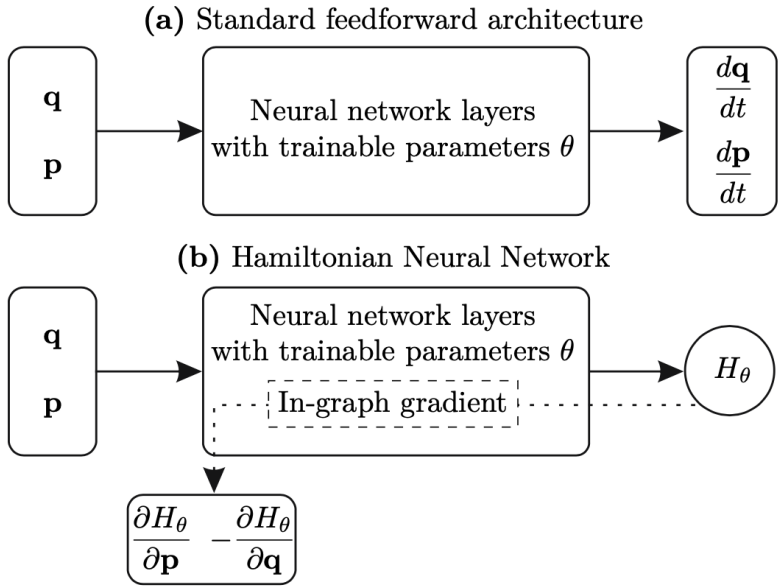


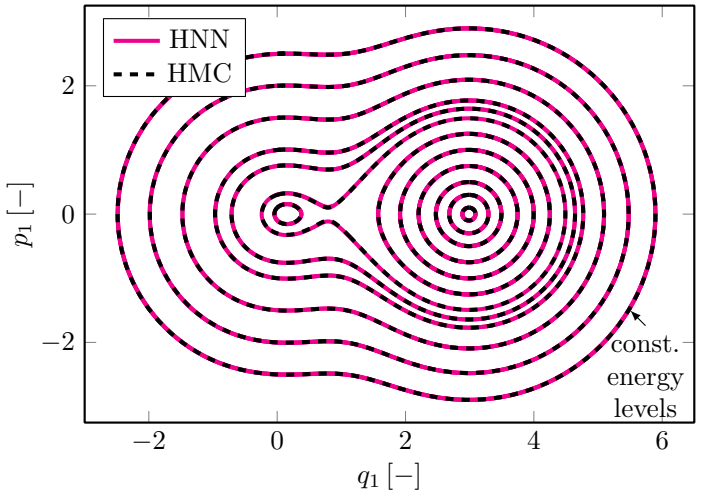
HMC Illustration

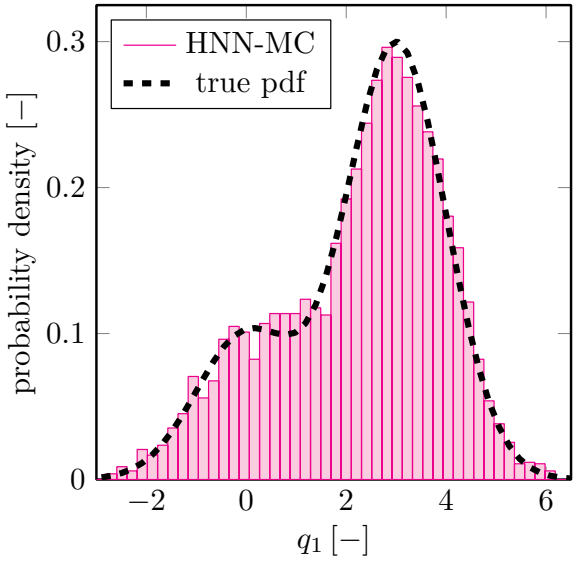
Hamiltonian Neural Networks



Hamiltonian Neural Networks

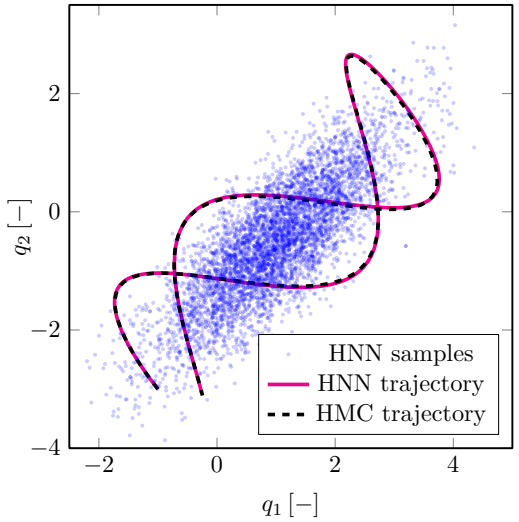


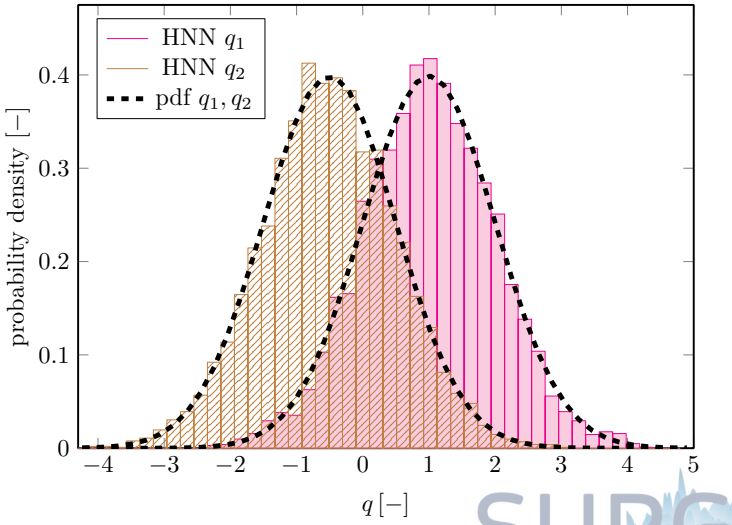




- Neural Network constrained to satisfy[3,4]

$$\frac{d\boldsymbol{q}}{dt} = \frac{\partial H}{\partial \boldsymbol{p}}$$
$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H}{\partial \boldsymbol{q}}$$





Variational Inference

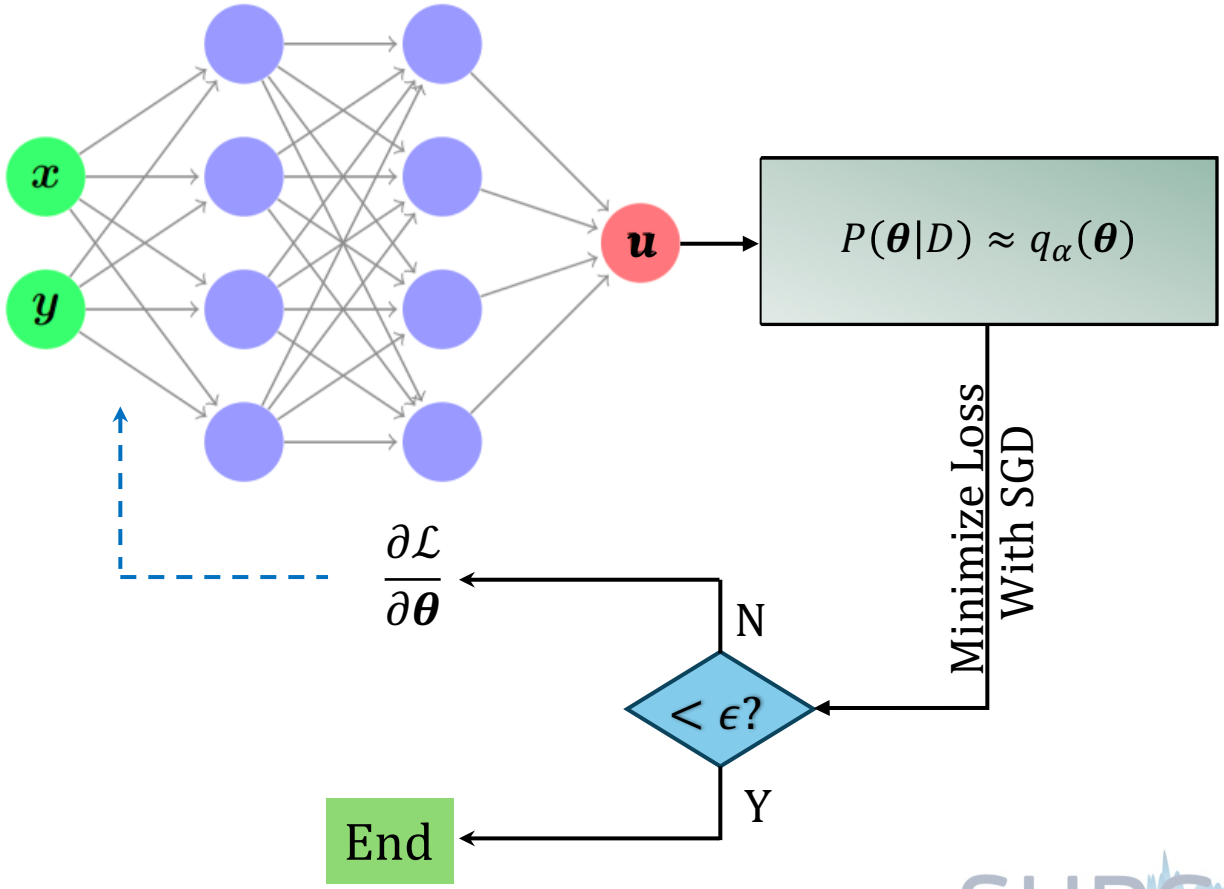
Approximate Posterior:

$$P(\boldsymbol{\theta}|D) \approx q_\alpha(\boldsymbol{\theta})$$

Intractable tractable, simpler, known family

Minimize difference between $P(\boldsymbol{\theta}|D)$ and $q_\alpha(\boldsymbol{\theta})$.

- Typically defined through a divergence
  - Kullback-Leibler (KL) divergence
    - Bayes By Backprop[5]
  - Jensen Shannon (JS) divergence[6]
  - α divergence
    - α Black Box[7]



KL Divergence

$$KL\,(q||p) = E[\ln q(\boldsymbol{\theta}|\alpha) - \ln p(\boldsymbol{\theta}|D)]$$
$$= \int_{\Theta} q(\boldsymbol{\theta}|\alpha) \ln \frac{q(\boldsymbol{\theta}|\alpha)}{p(\boldsymbol{\theta}|D)} d\boldsymbol{\theta}$$

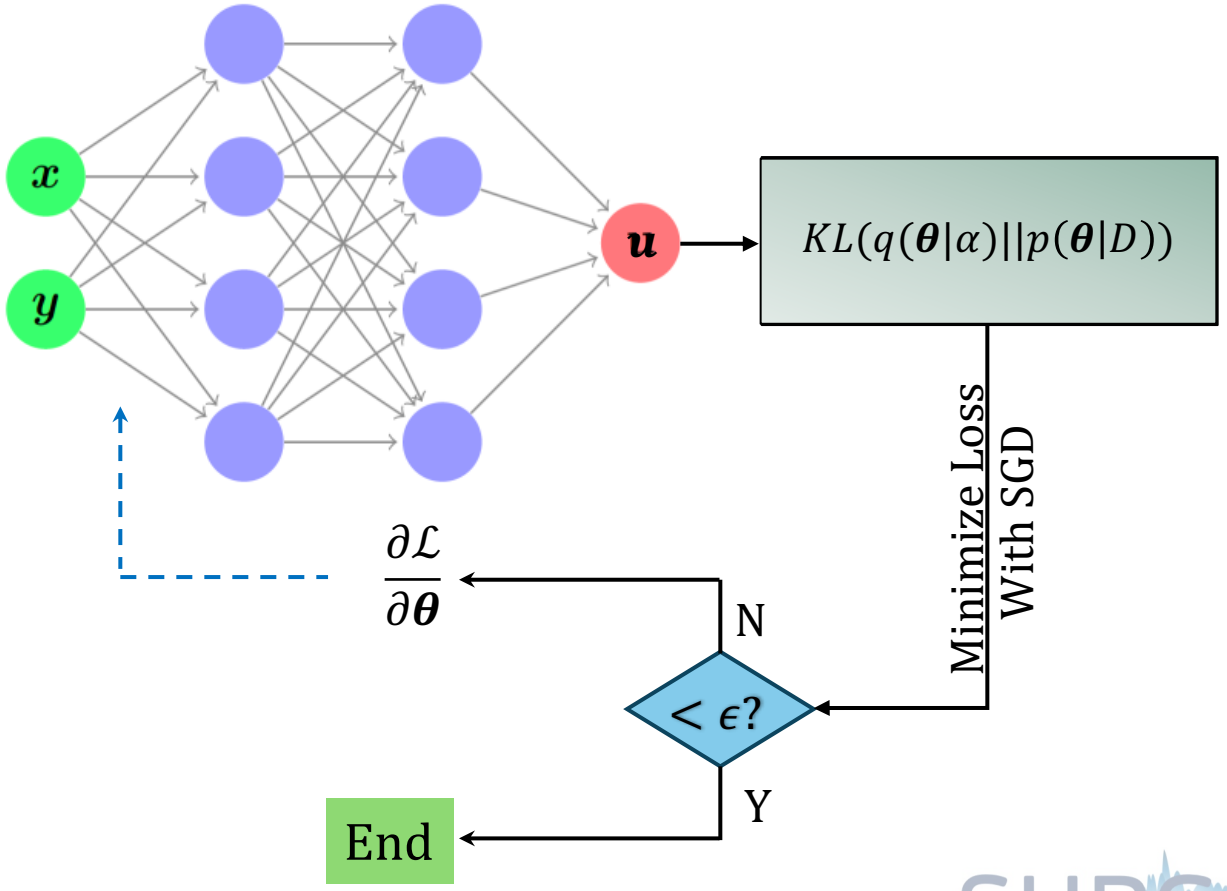
Define Loss Function

$$KL(q(\boldsymbol{\theta}|\alpha)||p(\boldsymbol{\theta}|D))$$

Applying Bayes Rule, this loss can be expressed

$$KL(q(\boldsymbol{\theta}|\alpha)||p(\boldsymbol{\theta})) - E_{q(\boldsymbol{\theta}|\alpha)}[\ln p(D|\boldsymbol{\theta})]$$

Evidence Lower Bound (ELBO)[8]



KL Divergence

$$KL\,(q||p) = E[\ln q(\boldsymbol{\theta}|\alpha) - \ln p(\boldsymbol{\theta}|D)]$$
$$= \int_{\Theta} q(\boldsymbol{\theta}|\alpha) \ln \frac{q(\boldsymbol{\theta}|\alpha)}{p(\boldsymbol{\theta}|D)} d\boldsymbol{\theta}$$

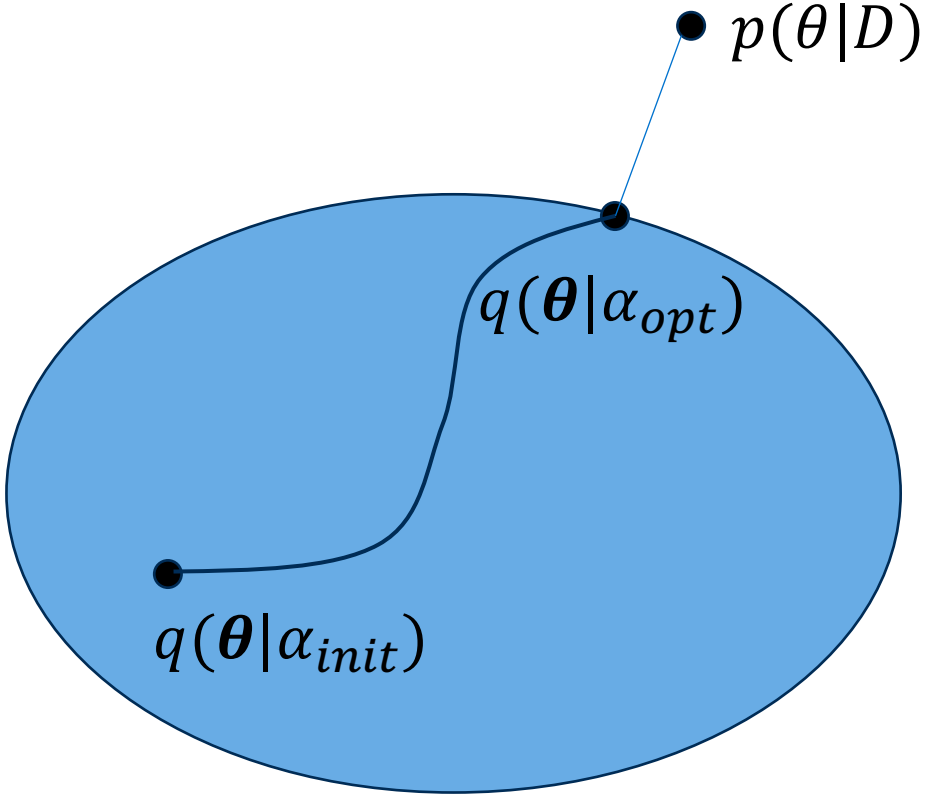
Define Loss Function

$$KL(q(\boldsymbol{\theta}|\alpha)||p(\boldsymbol{\theta}|D))$$

Applying Bayes Rule, this loss can be expressed

$$KL(q(\boldsymbol{\theta}|\alpha)||p(\boldsymbol{\theta})) - E_{q(\boldsymbol{\theta}|\alpha)}[\ln p(D|\boldsymbol{\theta})]$$

Evidence Lower Bound (ELBO)[8]



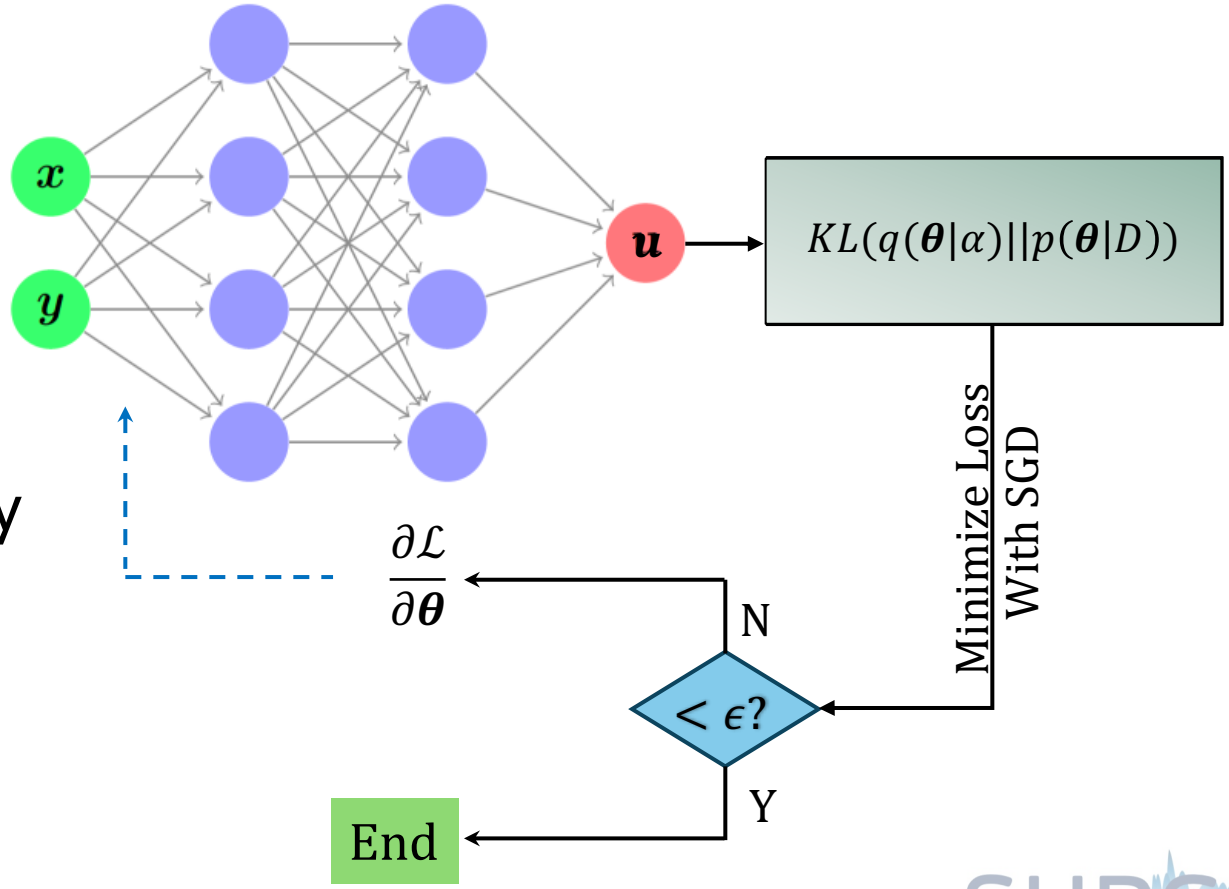
Bayes By Backprop[5]

$$KL(q(\boldsymbol{\theta}|\alpha)||p(\boldsymbol{\theta})) - E_{q(\boldsymbol{\theta}|\alpha)}[\ln p(D|\boldsymbol{\theta})]$$

Let

$$q(\theta|\alpha) = \prod_{i=1}^{n} \mathcal{N}(\theta_i; \mu_i, \sigma_i)$$

- First term can be computed analytically
- Monte Carlo simulation can be used to compute the second term
- Gradients for backpropagation can be readily computed



BBB Illustration

Monte Carlo Dropout[9]

Let

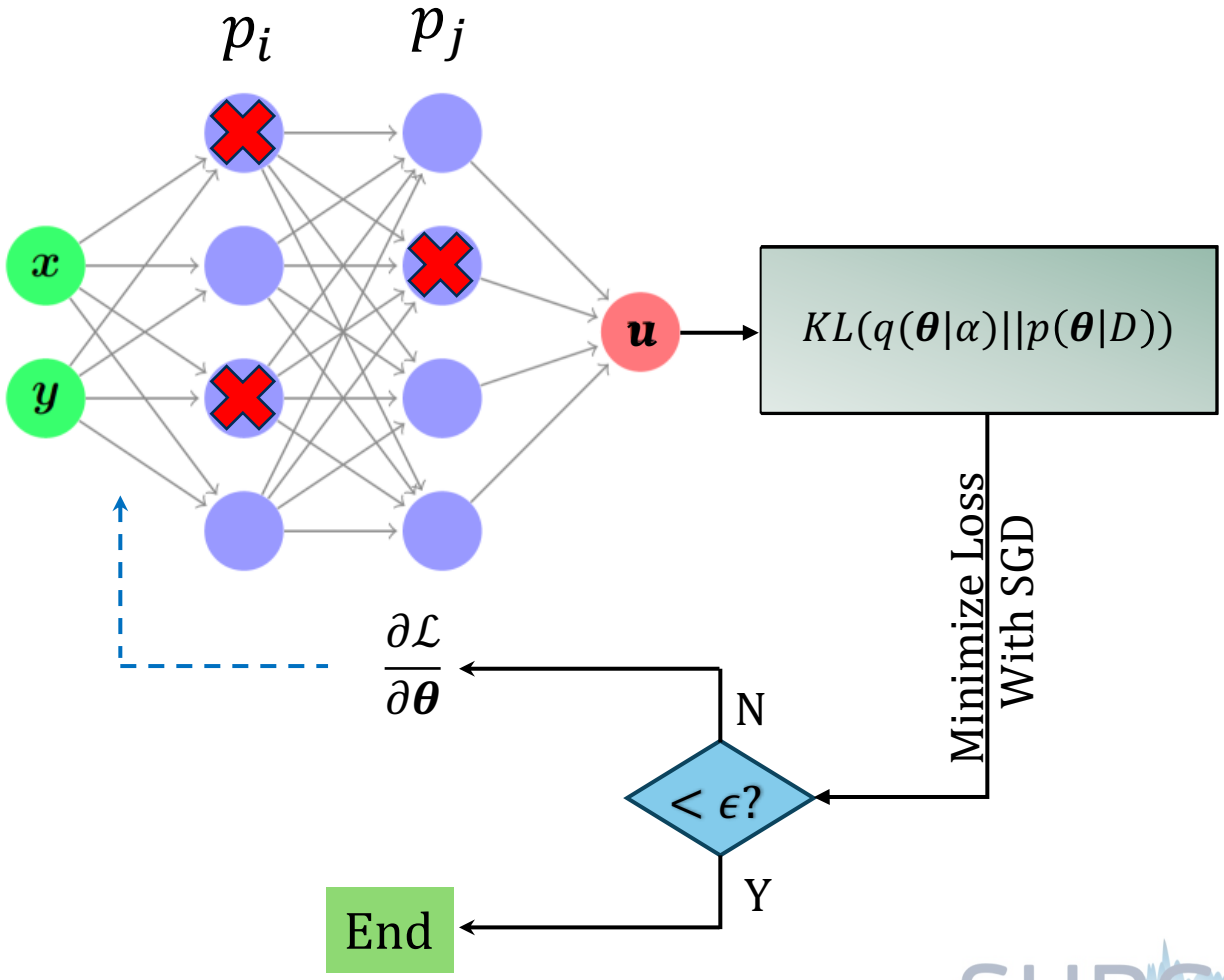
$$\boldsymbol{\theta}_i = \mathcal{M}_i \cdot \mathrm{diag}\left(\left[z_{i,j}\right]_{j=1}^{K_i}\right)$$

where

$$z_{i,j} \sim \mathrm{Bernoulli}(p_i)$$

be the matrix of weights for layer i.

The variational posterior is parameterized by $\mathcal{M}_i, p_i$.



Monte Carlo Dropout[9]

Let

$$\boldsymbol{\theta}_i = \mathcal{M}_i \cdot \mathrm{diag}\left(\left[z_{i,j}\right]_{j=1}^{K_i}\right)$$

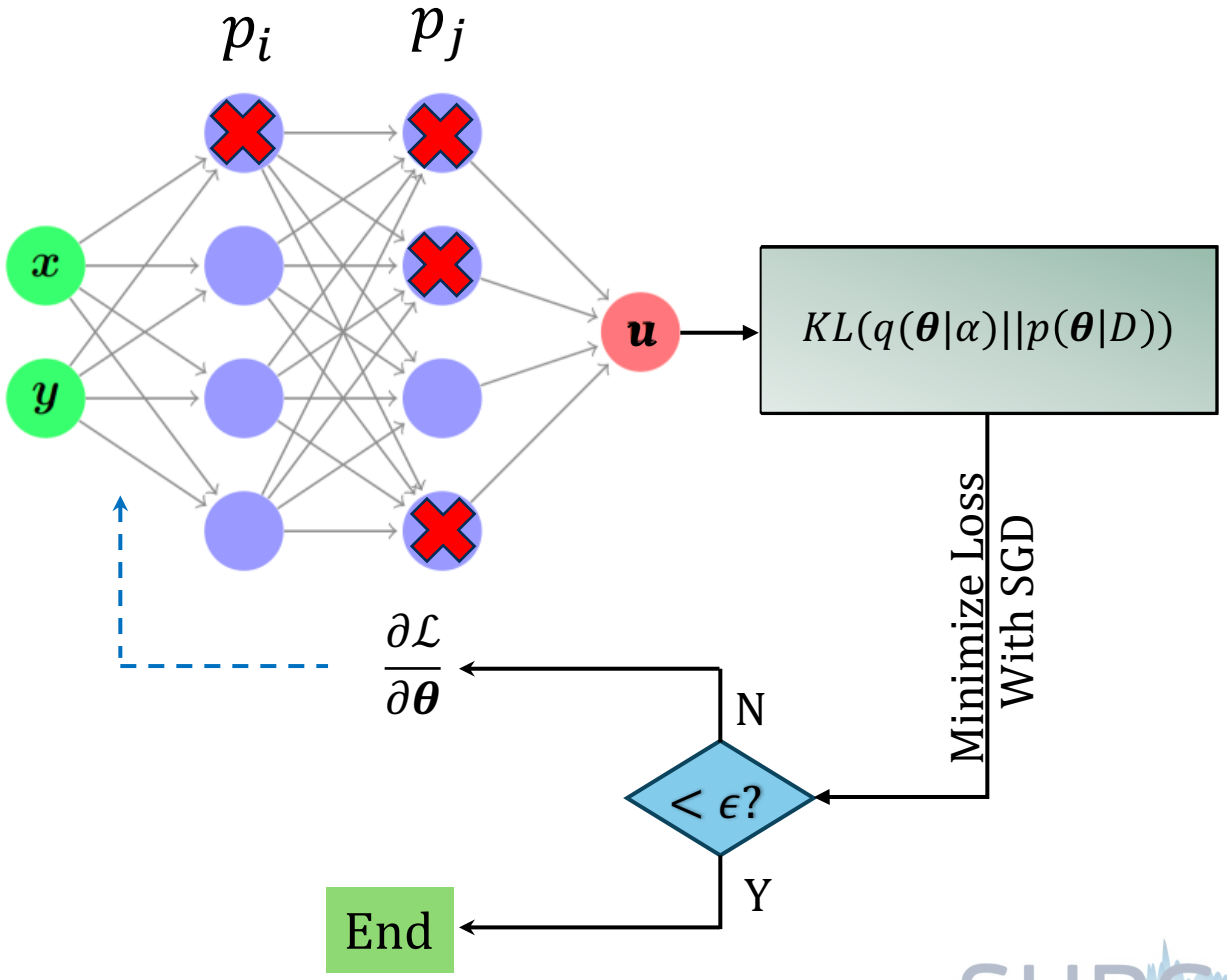
where

$$z_{i,j} \sim \mathrm{Bernoulli}(p_i)$$

be the matrix of weights for layer i.

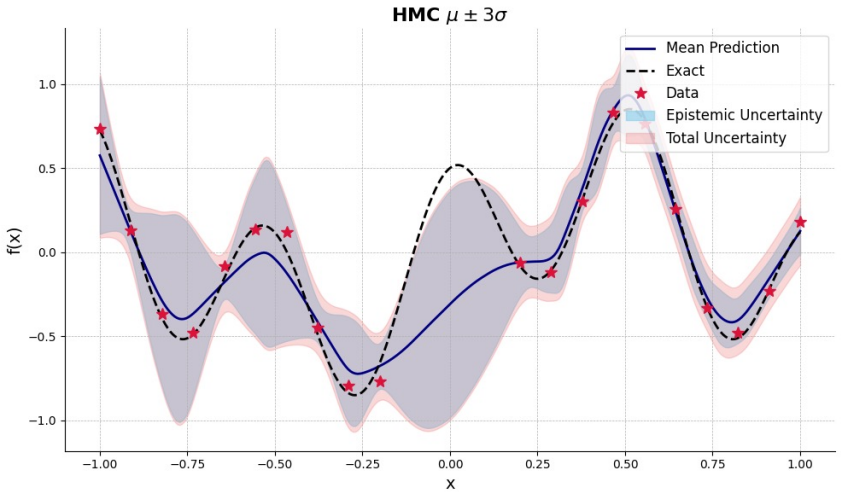
The variational posterior is parameterized by $\mathcal{M}_i, p_i$.

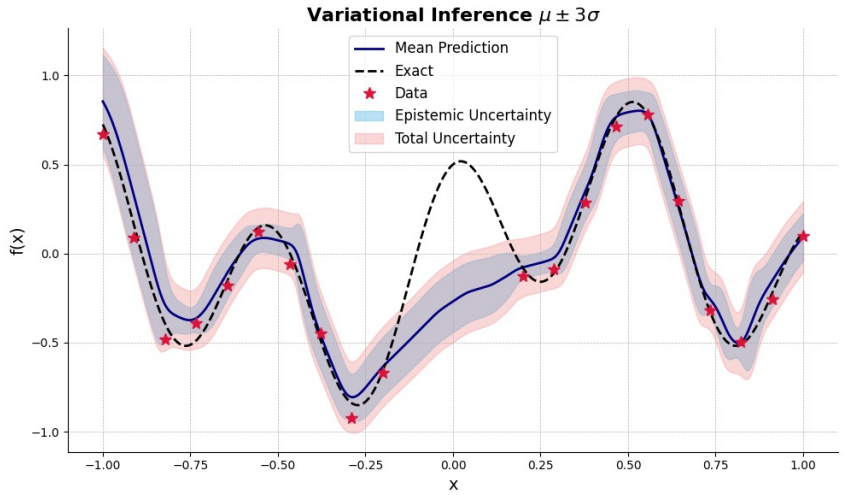
Mean and variance can be computed by Monte Carlo simulations on the trained network.

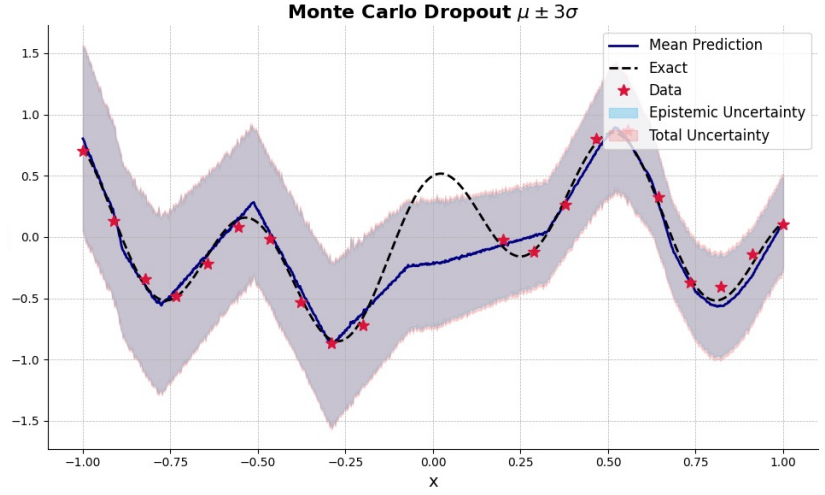


Bayesian NN Comparison

- Collect a small data set of 20 points from the function $y = 0.4 \sin 4x + 0.5 \sin 12x$
- Apply a small Gaussian noise to the data to introduce aleatory uncertainty
- Differentiate aleatory and epistemic uncertainty using HMC, BBB, and MCD
- Scripts provided via Github**







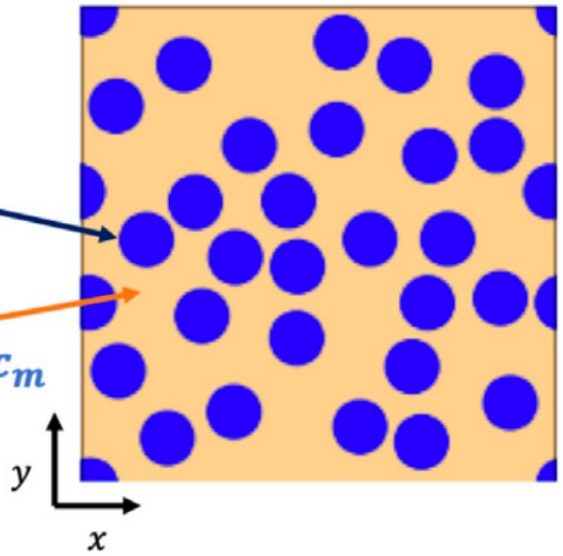
Materials Application[10]

Volume fraction $\mathbf{vf}$

Fibers are linear-elastic
Young modulus $\mathbf{E_f}$

Matrix plasticity law

$$\sigma = a_m + b_m(\varepsilon_{pl})^{c_m}$$



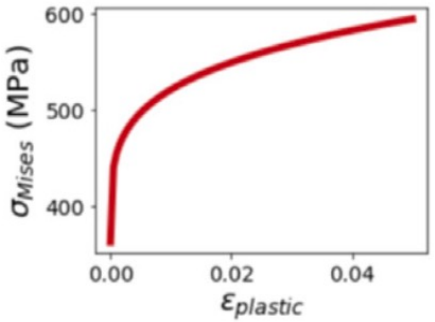
Periodic boundary conditions
Prescribed $\varepsilon_x^0 \neq 0$

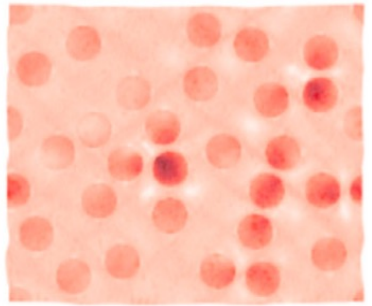
FE simulation
Post-processing

- Homogenized outputs
  - Linear elasticity: $\boldsymbol{E_{eff}, \nu_{eff}}$
  - Plasticity: $\boldsymbol{b_{eff}, c_{eff}}$
- Localized output

$$\mathbf{p_\sigma} = \frac{\text{vol. fibers: } \{\sigma_{Mises} > \sigma\}}{\text{total vol. fibers}}$$

Effective stress-strain law

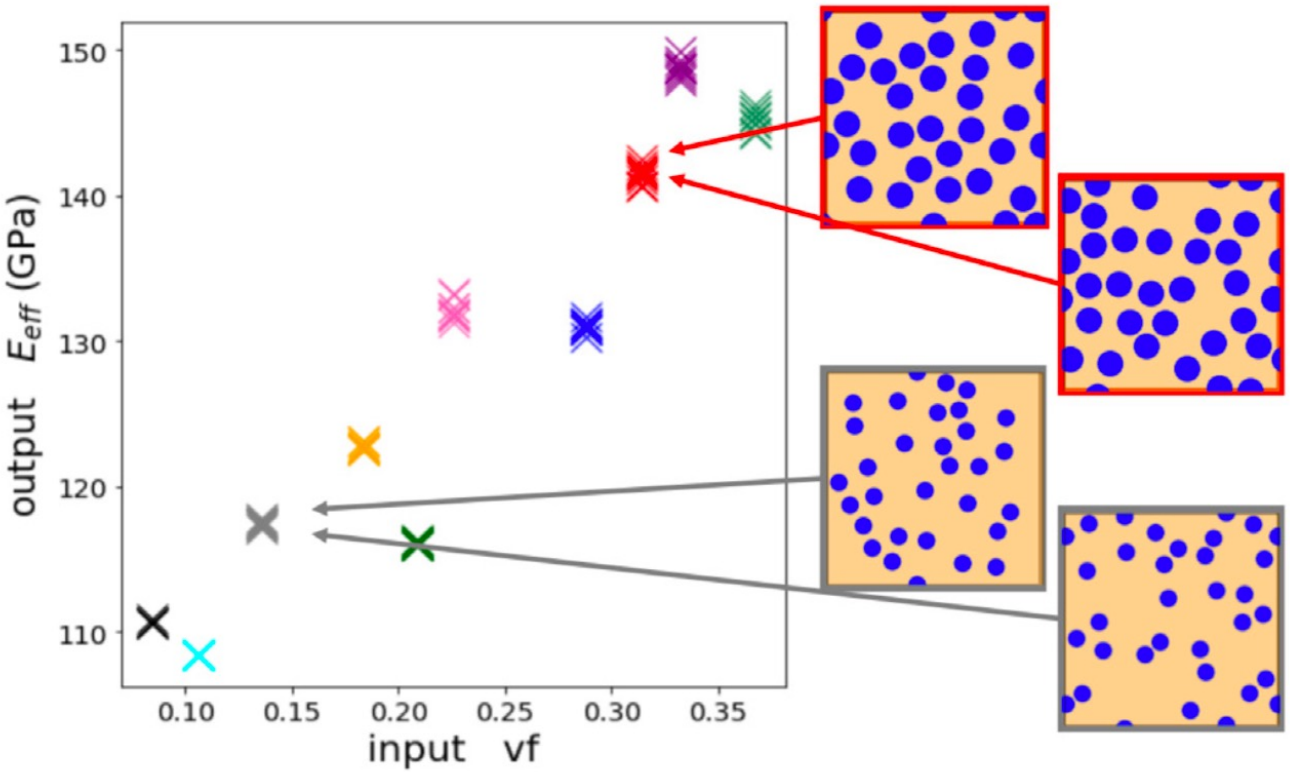


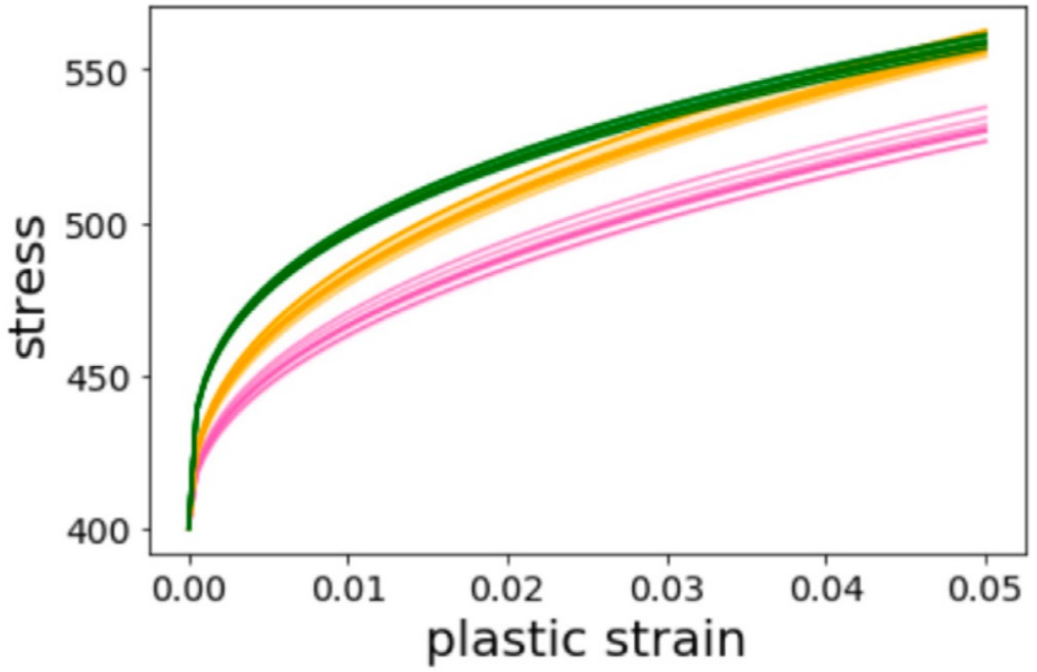


Mises stresses on deformed microstructure

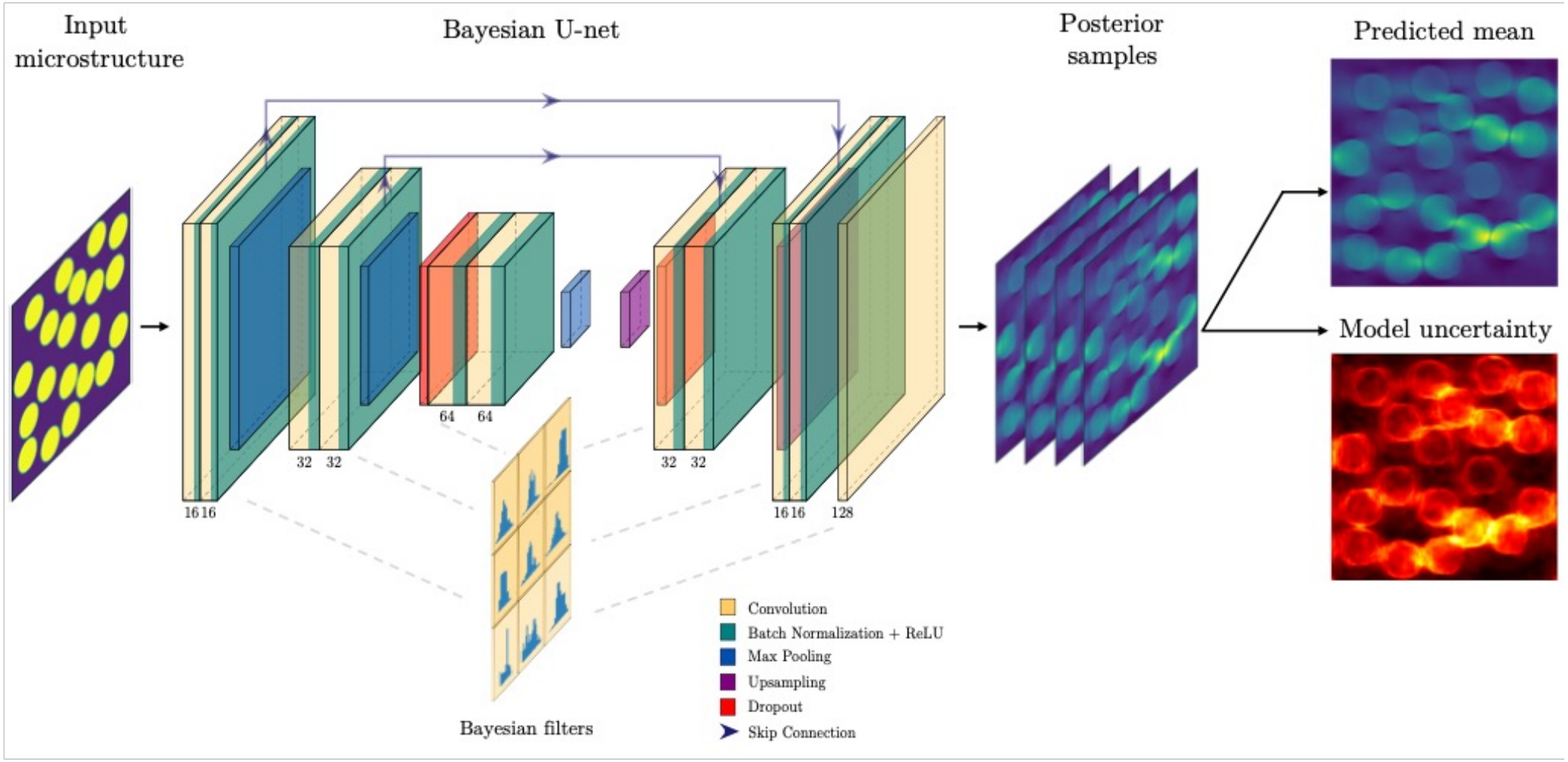
Materials Application[10]

- Predicts both aleatory and epistemic uncertainty in stochastic material properties

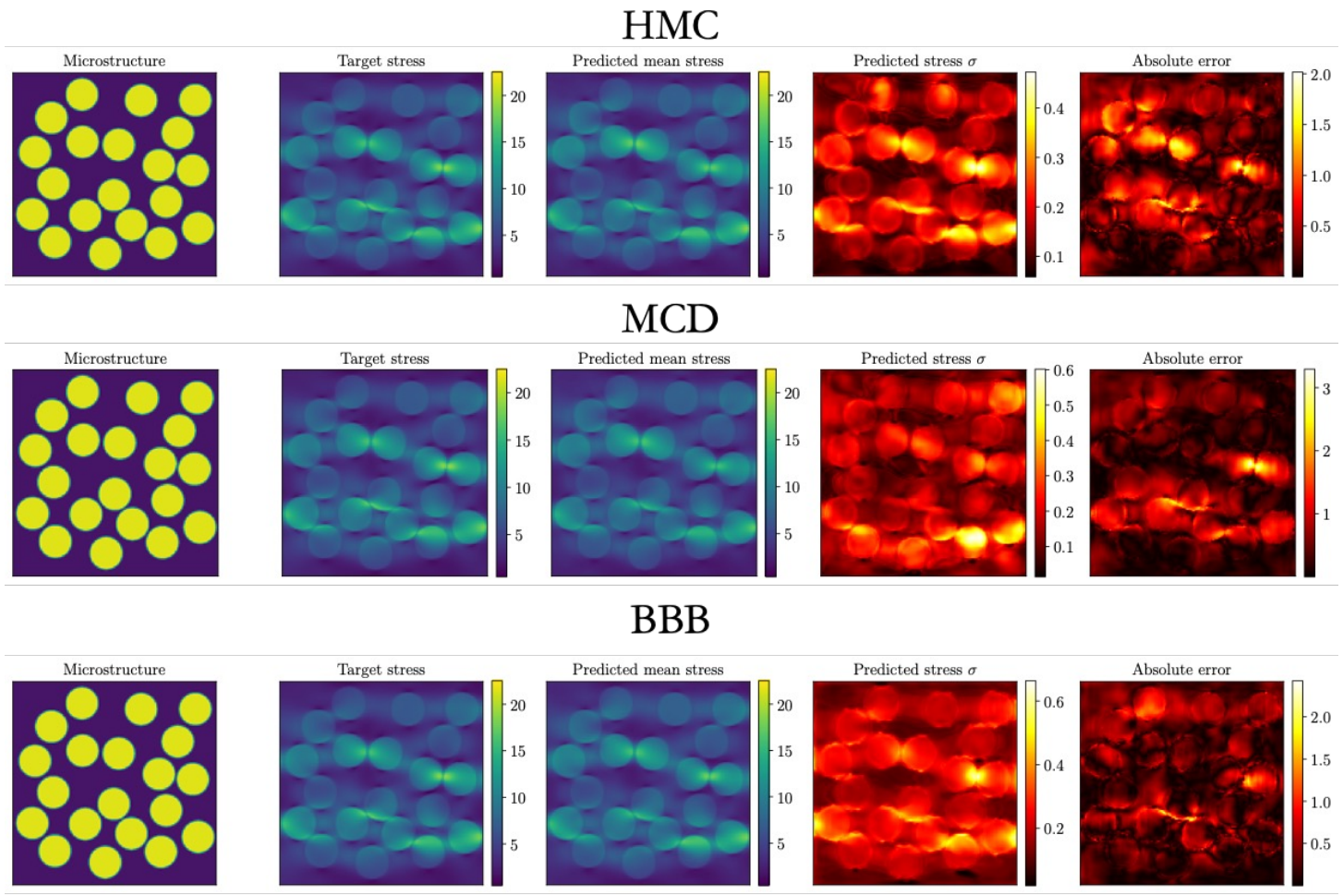


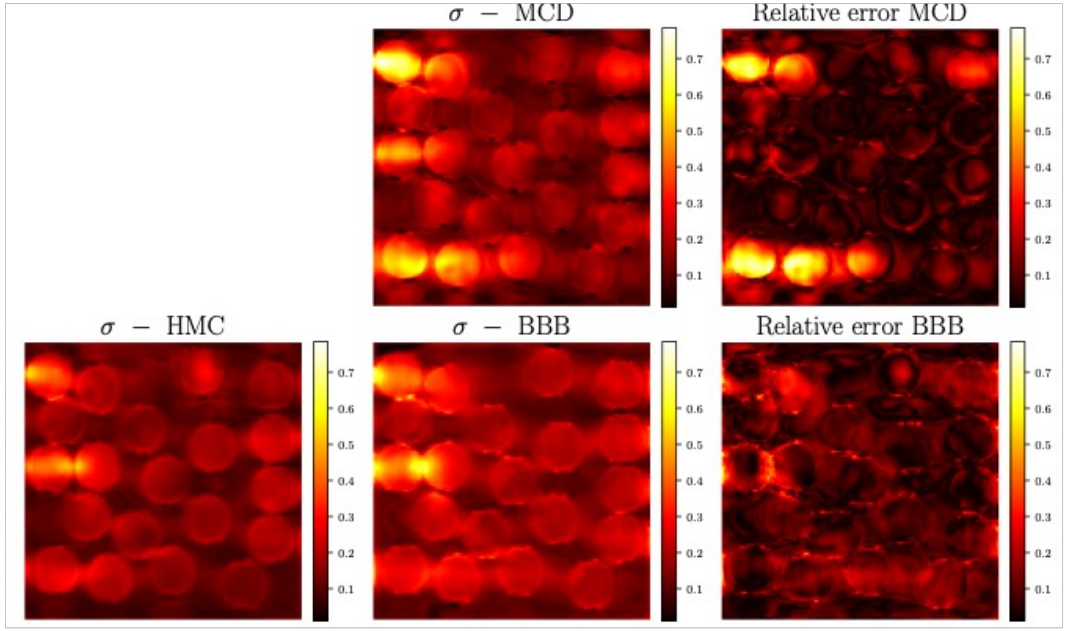


Materials Application

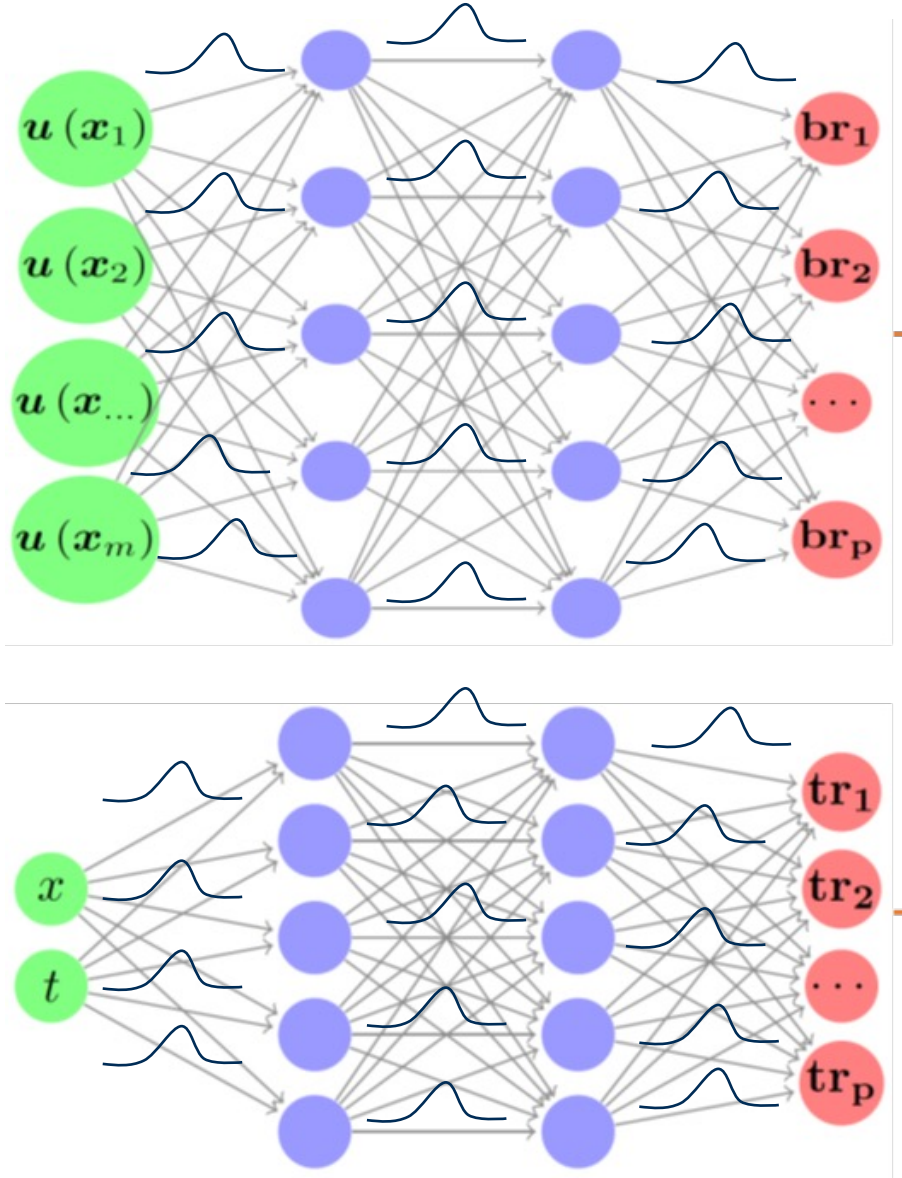


Materials Application





Bayesian DeepONet[11,12]



$$\mathcal{G}_\theta(u)(x,t)$$

$$\sum_{i=1}^{p} br_i * tr_i$$

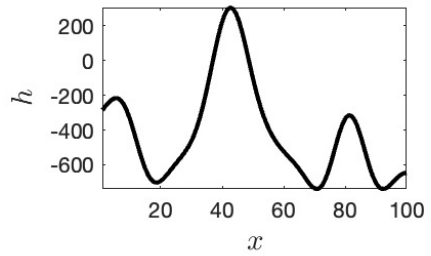
Data Loss $\mathcal{L}_D$ (MSE)

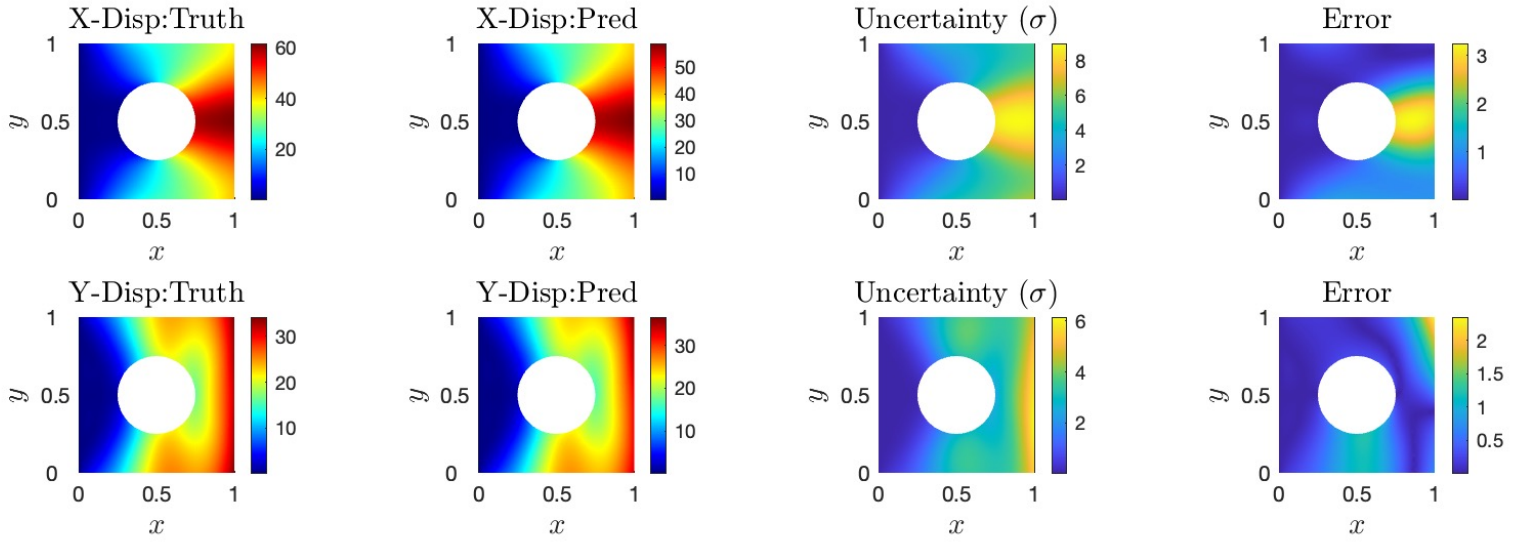
Prob. Loss $\mathcal{L}_P$ (KL div)

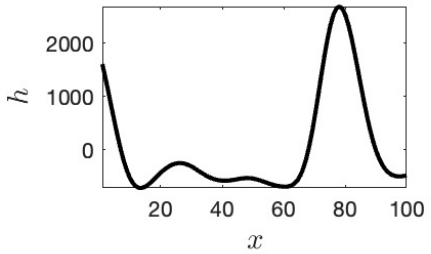
Minimize Total Loss $\mathcal{L} = \mathcal{L}_D + \lambda\mathcal{L}_P$

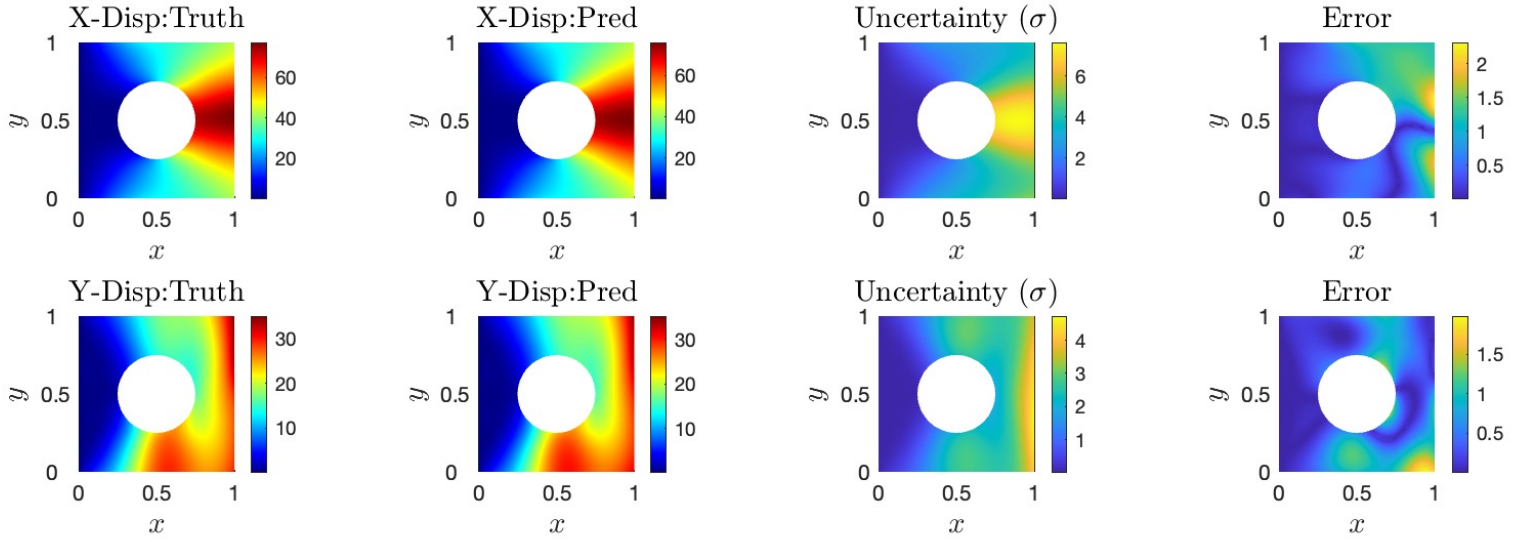
Bayesian DeepONet Illustration

Bayesian DeepONet Results

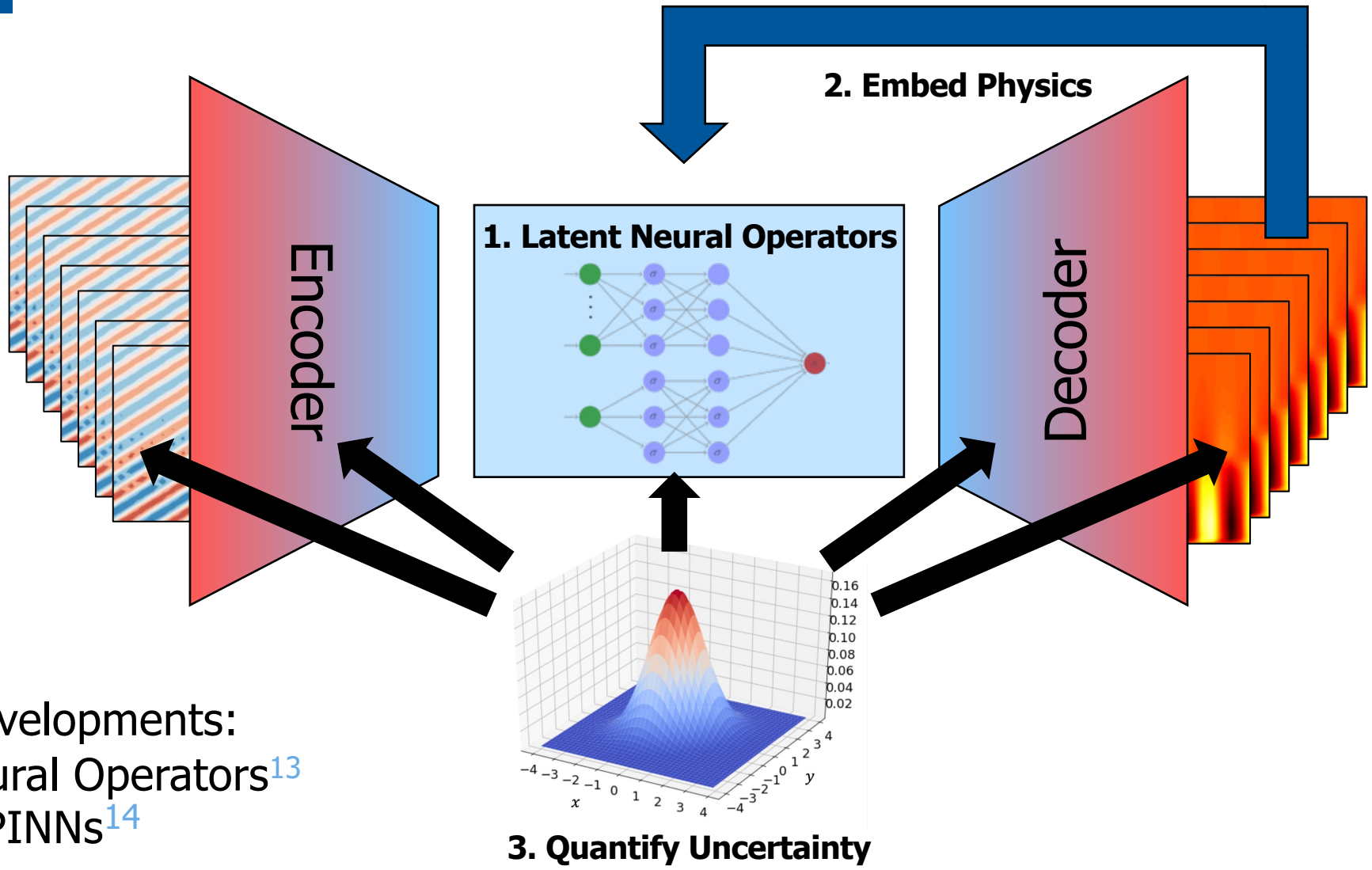








Physics-Informed Bayesian L-DeepONet



Important Developments:

- Latent Neural Operators[13]
- Bayesian PINNs[14]

Bayesian Neural Network References

1. [MAP Estimate] H. Zou and T. Hastie. (2005). "Regularization and variable selection via the elastic net". *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2, pp. 301–320.
2. [Hamiltonian Monte Carlo] R. M. Neal et al. (2011). "MCMC using Hamiltonian dynamics". In: *Handbook of Markov Chain Monte Carlo.* 2.11, p. 2.
3. [Hamiltonian Neural Networks] S. Greydanus, M. Dzamba, & J. Yosinski. (2019). Hamiltonian neural networks. *Advances in neural information processing systems*, 32.
4. [HNNs for Bayesian Inference] S.L. Dhulipala, Y. Che, & M.D. Shields. (2023). Efficient Bayesian inference with latent Hamiltonian neural networks in No-U-Turn Sampling. *Journal of Computational Physics*, 492, 112425.
5. [Bayes By Backprop] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. (2015). "Weight uncertainty in neural network". *International conference on machine learning. PMLR.*, pp. 1613–1622.
6. [Jensen-Shannon Divergence] P. Thiagarajan, & S. Ghosh. (2022). Jensen-Shannon Divergence Based Novel Loss Functions for Bayesian Neural Networks. *arXiv preprint arXiv:2209.11366*.
7. [Alpha Black-Box] J. Hernández-Lobato, Y. Li, M. Rowland, T. Bui, D. Hernández-Lobato, R. Turner. (2016). Black-box alpha-divergence minimization, *Proceedings of the 33rd International Conference on Machine Learning*, Vol. 48, PMLR, pp. 1511–1520.

Bayesian Neural Network References

8. [Evidence Lower Bound (ELBO)] D.P. Kingma, & M. Welling. (2013). Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*.
9. [Monte Carlo Dropout] Y. Gal and Z. Ghahramani. "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning". *International conference on machine learning*. PMLR. 2016, pp. 1050–1059.
10. [BNNs for Materials] A. Olivier, M.D. Shields, & L. Graham-Brady. (2021). Bayesian neural networks for uncertainty quantification in data-driven materials modeling. *Computer methods in applied mechanics and engineering*, 386, 114079.
11. [Bayesian DeepONet] G. Lin, C. Moya, Z. & Zhang. (2023). B-DeepONet: An enhanced Bayesian DeepONet for solving noisy parametric PDEs using accelerated replica exchange SGLD. *Journal of Computational Physics*, 473, 111713.
12. [Variational Bayesian DeepONet] S. Garg, S. & Chakraborty. (2023). VB-DeepONet: A Bayesian operator learning framework for uncertainty quantification. *Engineering Applications of Artificial Intelligence*, 118, 105685.

Bayesian Neural Network References

13. [Latent Neural Operators] K. Kontolati, S. Goswami, G.E. Karniadakis, M.D. Shields. (2024). Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems. *Nature Communications.* Accepted for publication.
14. [Bayesian PINNs] L. Yang, X. Meng, and G. E. Karniadakis. (2021). "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data". *Journal of Computational Physics.* 425, p. 109913.

Gaussian Process Regression[1]

- Consider a computational model $y(\boldsymbol{x})$ and a set of n training data

$$\boldsymbol{X}, \boldsymbol{Y} = \left\{\boldsymbol{x}^{(i)}, y^{(i)}\right\}_{i=1}^{n}$$

- We approximate the model by

$$\mathcal{Y}(\boldsymbol{x}, \omega) = \mathcal{F}(x) + Z(\boldsymbol{x}, \omega)$$

where

$$\mathcal{F}(\boldsymbol{x}) = \boldsymbol{\beta}^T f(\boldsymbol{x})$$

$Z(x)$ is a zero-mean Gaussian random process having covariance

$$E[Z(\boldsymbol{x}_1)Z(\boldsymbol{x}_2)] = \sigma_z^2 \mathcal{R}(\boldsymbol{x}_1, \boldsymbol{x}_2|\boldsymbol{\theta})$$

and $\mathcal{R}(\boldsymbol{x}_1, \boldsymbol{x}_2|\boldsymbol{\theta})$ is the autocorrelation function having hyperparameters $\boldsymbol{\theta}$.

Gaussian Process Regression[1]

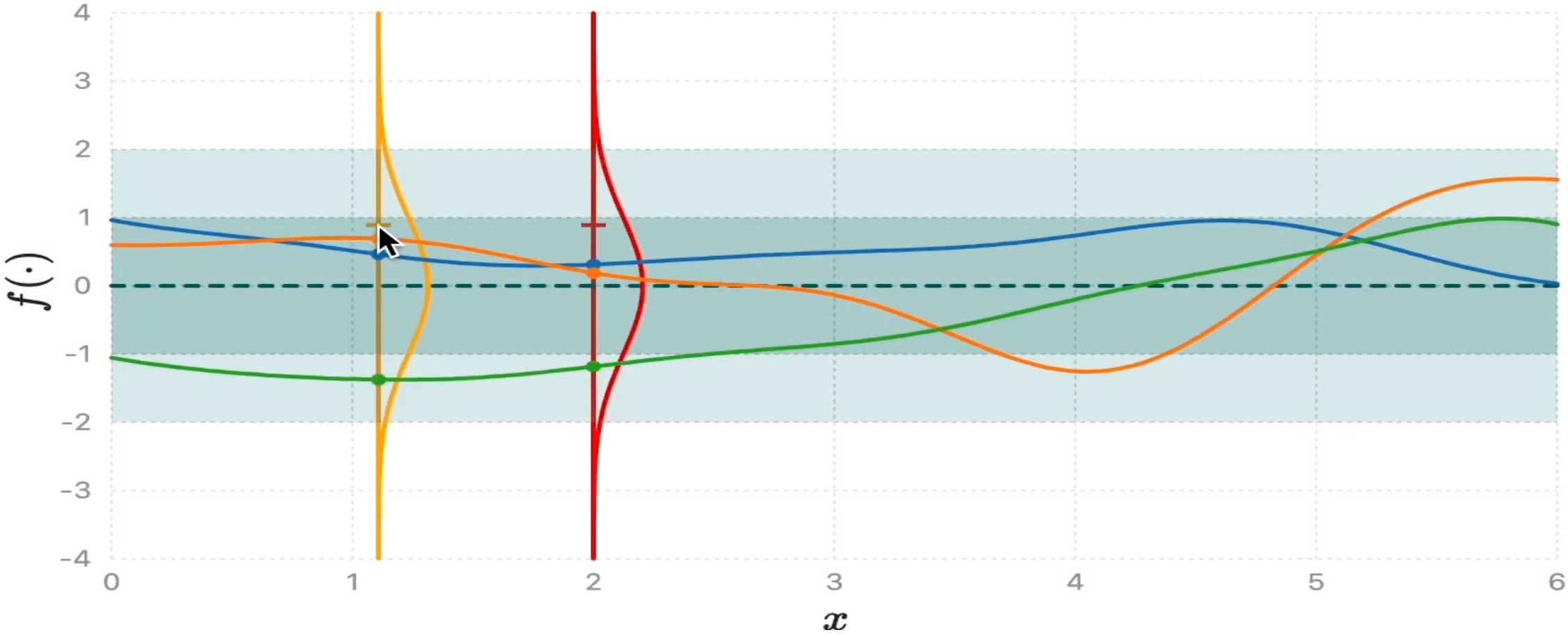
- The joint distribution of the predictions $\mathcal{Y}(\boldsymbol{x})$ and the training data Y is given by

$$\begin{Bmatrix} \mathcal{Y}(\boldsymbol{x}) \\ \boldsymbol{Y} \end{Bmatrix} \sim \mathcal{N}\left(\begin{Bmatrix} \boldsymbol{f}(\boldsymbol{x})^{\boldsymbol{T}}\boldsymbol{\beta} \\ \boldsymbol{F\beta} \end{Bmatrix}, \sigma_z^2 \begin{Bmatrix} 1 & \boldsymbol{r}^T(\boldsymbol{x}) \\ \boldsymbol{r}(\boldsymbol{x}) & \boldsymbol{R} \end{Bmatrix} \right)$$

where

- $\boldsymbol{F}$ is a matrix of basis functions evaluated at the training points
- $\boldsymbol{r}(\boldsymbol{x})$ is a vector of correlations between the prediction point $\boldsymbol{x}$ and training points
- $\boldsymbol{R}$ is the matrix of correlations between the training points

Gaussian Process Regression[1]



Gaussian Process Regression[1]

- The parameters of the model $\{\boldsymbol{\theta}, \sigma_z, \boldsymbol{\beta}\}$ are determined by maximizing the likelihood function

$$\mathcal{L}(\boldsymbol{\theta}, \sigma_z, \boldsymbol{\beta}) = \frac{\sqrt{\det \boldsymbol{R}}}{(2\pi\sigma_z^2)^{\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma_z^2}(\boldsymbol{Y} - \boldsymbol{F}\boldsymbol{\beta})^T \boldsymbol{R}^{-1}(\boldsymbol{Y} - \boldsymbol{F}\boldsymbol{\beta})\right]$$

- This yields a regression model with predictor mean and variance:

$$\hat{y}(\boldsymbol{x}) = f(\boldsymbol{x})\boldsymbol{\beta}^T + \boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{R}^{-1}(\boldsymbol{Y} - \boldsymbol{F}\boldsymbol{\beta})$$

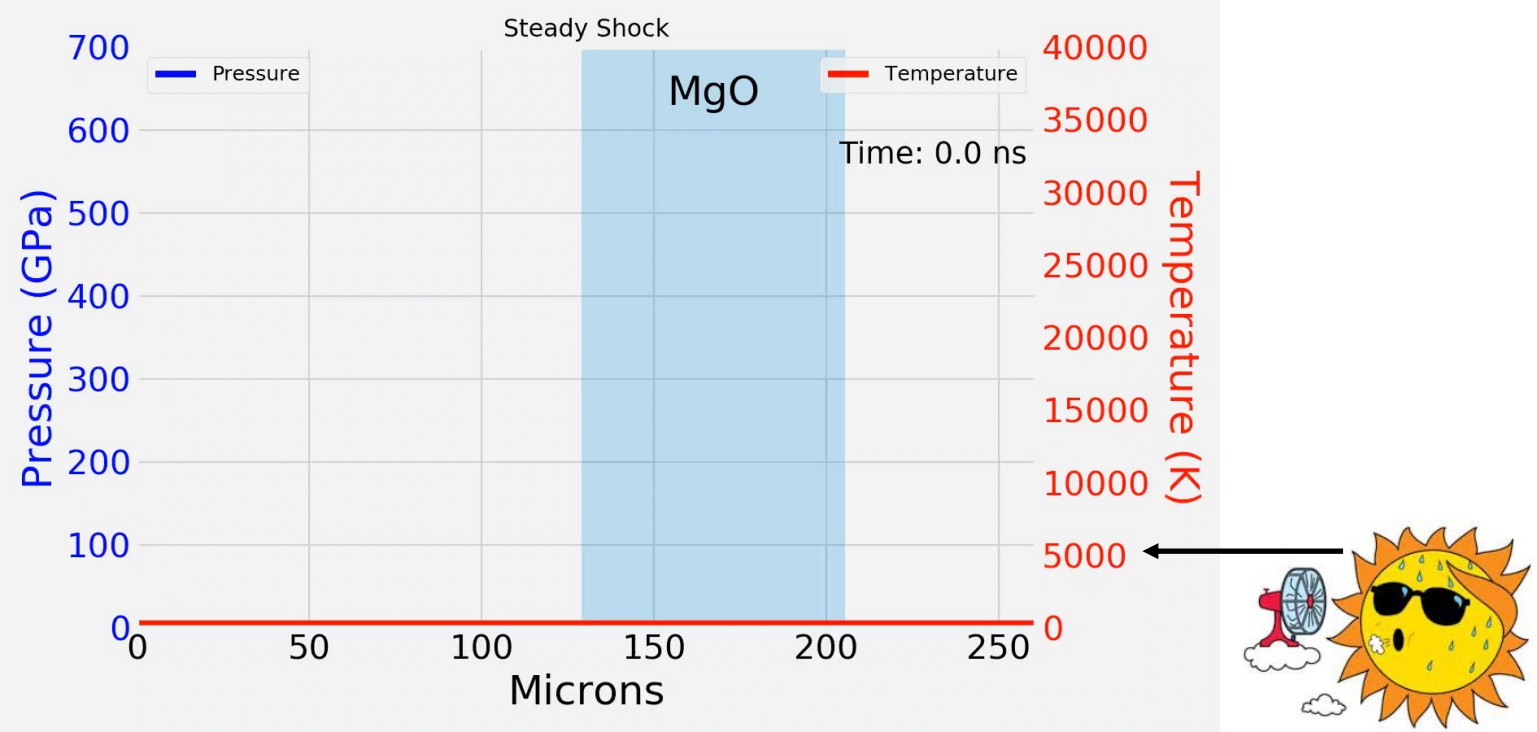
$$\sigma_{\hat{y}}^2(\boldsymbol{x}) = \sigma_z^2\left(1 - \boldsymbol{r}(\boldsymbol{x})^T \boldsymbol{R}^{-1}\boldsymbol{r}(\boldsymbol{x}) + \boldsymbol{t}(\boldsymbol{x})^T (\boldsymbol{F}^T \boldsymbol{R}^{-1}\boldsymbol{F})^{-1}\boldsymbol{t}(\boldsymbol{x})\right)$$

where

$$\boldsymbol{t}(\boldsymbol{x}) = \boldsymbol{F}^T \boldsymbol{R}^{-1} r(\boldsymbol{x}) - f(\boldsymbol{x})$$

Materials Application

- Equations of State (EOS) relate thermodynamic state variables in a system
- These are needed, e.g. to estimate the state of a system under extreme T and P



Equation of State

- Historically the EOS has been
  - Semi-empirically derived
  - Following an assumed physically consistent form
  - Deterministic
- Mie-Grüneisen-Debye model[2]

$$P(V,T) = P_V(V,T=0) + P_{TH\ \mathrm{Debye}}(V,T)$$

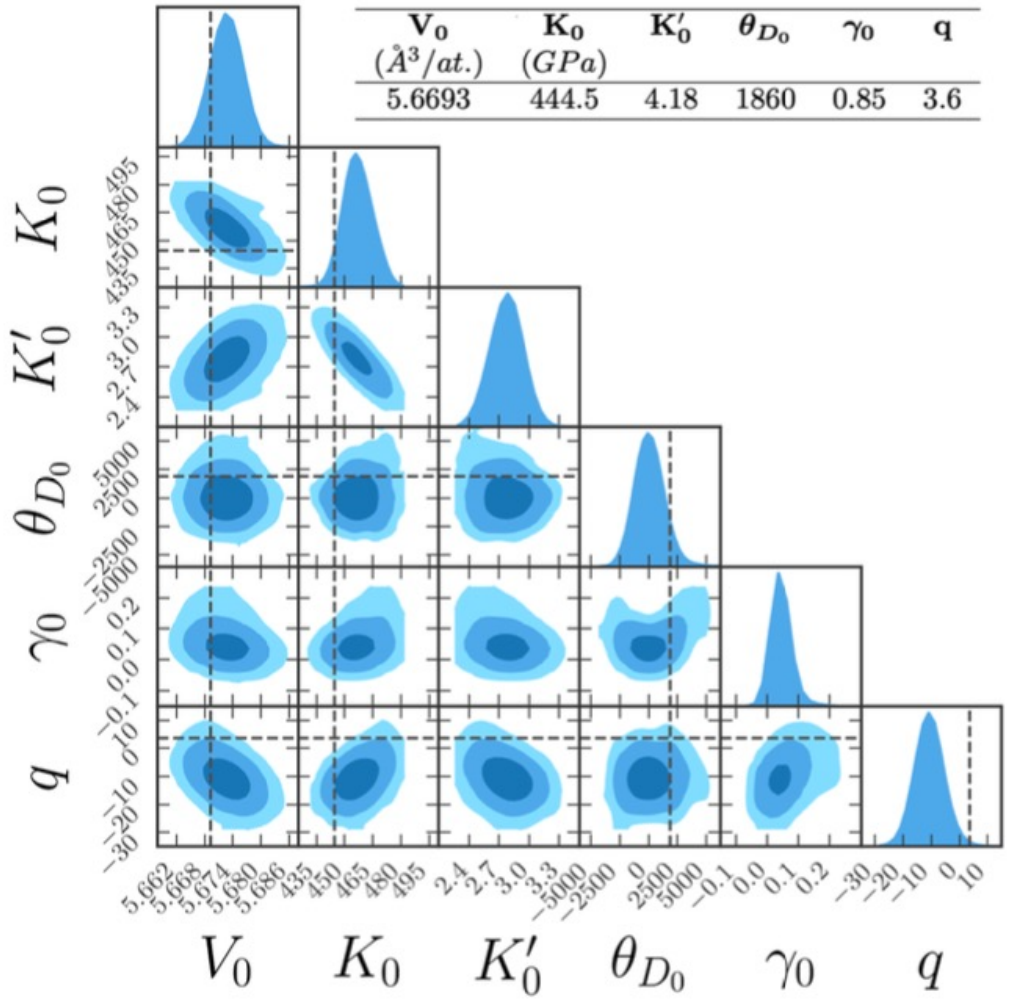
$$P_V = 3K_0 x^{-2}(1-x)\exp[(1.5K_0' - 1.5)(1-x)]$$

$$P_{TH\ \mathrm{Debye}}(V,T) = \frac{9RT\gamma_D}{V}\left(\frac{T}{\Theta_D}\right)^3 \int_0^{\theta_D/T} \frac{z^3}{e^z - 1} dz.$$

$$\theta_D = \theta_0 x^{-1.5}\exp[\gamma_1(1-x^{3q})/q] \qquad \gamma_D = -\frac{d\ln\theta_D}{d\ln V} = \gamma_1 x^{3q} + 1/2.$$

Equation of State: UQ

- This model has 6 parameters
- These parameters are calibrated from experimental data (deterministically)[3]
- For UQ, we can estimate the parameters using Bayesian Inference
- Scarce data + large number of parameters make this task challenging
- Model form is **assumed**



Gaussian Process Equation of State

- Let’s use GP regression to learn a non-parametric EOS. What’s the problem?

Gaussian Process Equation of State

- Let's use GP regression to learn a non-parametric EOS. What's the problem?
- **EOS MUST SATISFY THE LAWS OF THERMODYNAMICS**

- Helmholtz Free Energy: $F = E - TS$

- 1st and 2nd Laws of Thermodynamics: $dF = -SdT - PdV$ implies $P = -\left.\frac{\partial F}{\partial V}\right|_T$, $E = F - T\left.\frac{\partial F}{\partial T}\right|_V$

- Taking derivatives $\frac{\partial P}{\partial T}$ and $\frac{\partial E}{\partial V}$ yields $\underbrace{P = T\frac{\partial P}{\partial T} + \frac{\partial E}{\partial V}.}$

Thermodynamic Consistency Constraint

Constrained Gaussian Process Regression

- Let's use ML to fit a non-parametric model to the data. What's the problem?
- **EOS MUST SATISFY THE LAWS OF THERMODYNAMICS**

- Helmholtz Free Energy: $F = E - TS$

- F must be a convex function

$$\left(\frac{\partial^2 F}{\partial V^2}\right)_T = -\left(\frac{\partial P}{\partial V}\right)_T = \frac{1}{V\kappa_T} \geq 0 \Longleftrightarrow \kappa_T > 0 \Longleftrightarrow \left(\frac{\partial P}{\partial V}\right)_T \leqslant 0,$$

$$\left(\frac{\partial^2 S}{\partial E^2}\right)_V = -\frac{1}{T^2}\left(\frac{\partial T}{\partial E}\right)_V = -\frac{1}{T^2 c_v} \leqslant 0 \Longleftrightarrow c_V > 0 \Longleftrightarrow \left(\frac{\partial E}{\partial T}\right)_V \geqslant 0$$

Thermodynamic Stability Constraints

Linear Operations of Gaussian Processes

- Let's start by modeling the Helmholtz Free Energy as a GP, $F \sim GP\left[\mu_F(\mathbf{X}), k_{FF}\left(\mathbf{X},\mathbf{X}'\right)\right]$

- Applying the linear operator $\mathscr{L}_{(V,T)} = \begin{pmatrix} -\frac{\partial}{\partial V} \\ 1 - T\frac{\partial}{\partial T} \end{pmatrix}$

$$\begin{bmatrix} P \\ E \end{bmatrix} | \mathbf{X} \sim GP\left(\begin{bmatrix} \frac{\partial \mu_F(\mathbf{X})}{\partial V} \\ \mu_F - T\frac{\partial \mu_F(\mathbf{X})}{\partial T} \end{bmatrix}, \begin{bmatrix} \frac{\partial^2}{\partial V^2} k_{FF}(\mathbf{X},\mathbf{X}) & -\frac{\partial}{\partial V}\left(1 - T\frac{\partial}{\partial T}\right) k_{FF}(\mathbf{X},\mathbf{X}) \\ -\frac{\partial}{\partial V}\left(1 - T\frac{\partial}{\partial T}\right) k_{FF}(\mathbf{X},\mathbf{X}) & \left(1 - T\frac{\partial}{\partial T}\right)\left(1 - T\frac{\partial}{\partial T}\right) k_{FF}(\mathbf{X},\mathbf{X}) \end{bmatrix} \right)$$

- This new GP satisfies the *Thermodynamic Consistency Constraint* with log likelihood given by[4]

$$-\log p(P,E \mid \mathbf{X},\boldsymbol{\theta}) = \frac{1}{2}\left(\begin{bmatrix} P \\ E \end{bmatrix} - \begin{bmatrix} \mu_P(\mathbf{X}) \\ \mu_E(\mathbf{X}) \end{bmatrix} \right)^T \mathbf{K}_{11}^{-1} \left(\begin{bmatrix} P \\ E \end{bmatrix} - \begin{bmatrix} \mu_P \\ \mu_E \end{bmatrix} \right) + \frac{1}{2}\log|\mathbf{K}_{11}| + \frac{N}{2}\log(2\pi)$$

Model Form Uncertainty

- To impose *Thermodynamic Stability Constraints* we have

$$P\left[\left(\frac{\partial P(X_v)}{\partial V} \mid \mathbf{X}_v, P, X\right) < 0\right] \leq \eta, \quad \text{for all } \mathbf{X}_v \in \mathscr{X}$$

$$P\left[\left(\frac{\partial E(\mathbf{X}_v)}{\partial T} \mid X_v, E, \mathbf{X}\right) > 0\right] \leq \eta, \quad \text{for all } \mathbf{X}_v \in \mathscr{X}$$

$$0 < \eta \ll 1$$

- Because these probabilities are Gaussian, we have

$$\mu_{\frac{\partial P}{\partial V}} + \Phi^{-1}(\eta)\, \sigma_{\frac{\partial P}{\partial V}} \leq 0 \qquad \mu_{\frac{\partial E}{\partial T}} - \Phi^{-1}(\eta)\, \sigma_{\frac{\partial E}{\partial T}} \geq 0$$

Constrained Gaussian Process Regression[5]

- Maximize the likelihood function

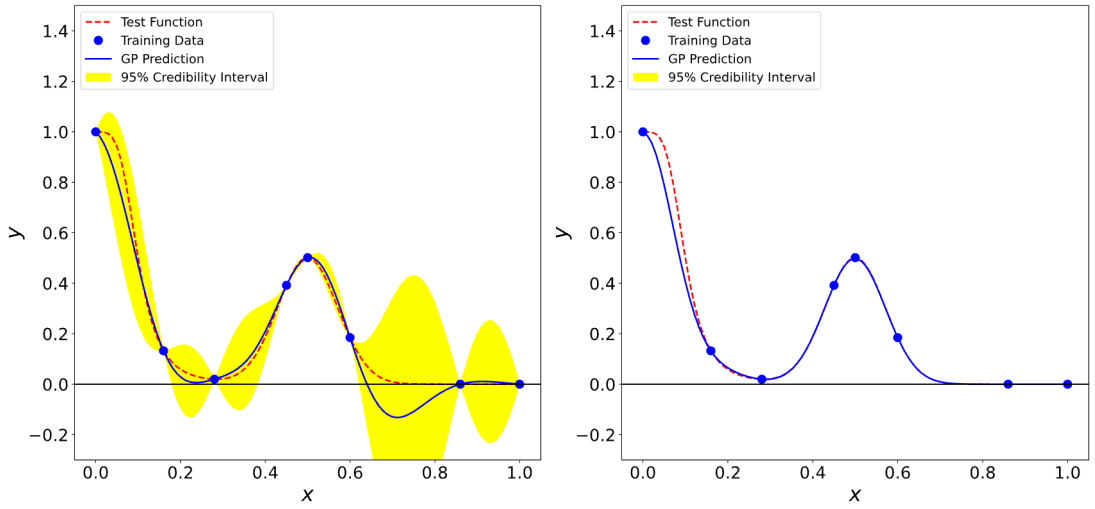
$$\mathcal{L}(\boldsymbol{\theta}, \sigma_z, \boldsymbol{\beta}) = \frac{\sqrt{\det \boldsymbol{R}}}{(2\pi\sigma_z^2)^{\frac{n}{2}}} \exp\left[-\frac{1}{2\sigma_z^2}(\boldsymbol{Y} - \boldsymbol{F\beta})^T \boldsymbol{R}^{-1}(\boldsymbol{Y} - \boldsymbol{F\beta})\right]$$

subject to:

$$\mu_{\frac{\partial P}{\partial V}} + \Phi^{-1}(\eta)\, \sigma_{\frac{\partial P}{\partial V}} \leq 0 \qquad \mu_{\frac{\partial E}{\partial T}} - \Phi^{-1}(\eta)\, \sigma_{\frac{\partial E}{\partial T}} \geq 0$$

or other constraints:[5]

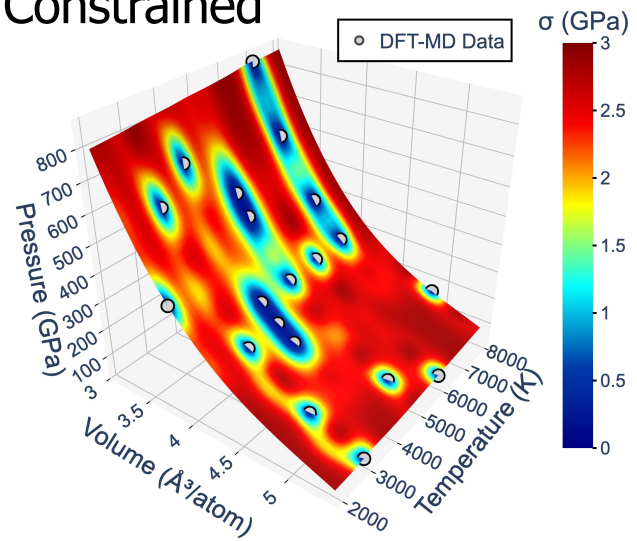
- **Boundedness**
- Monotonicity
- Convexity

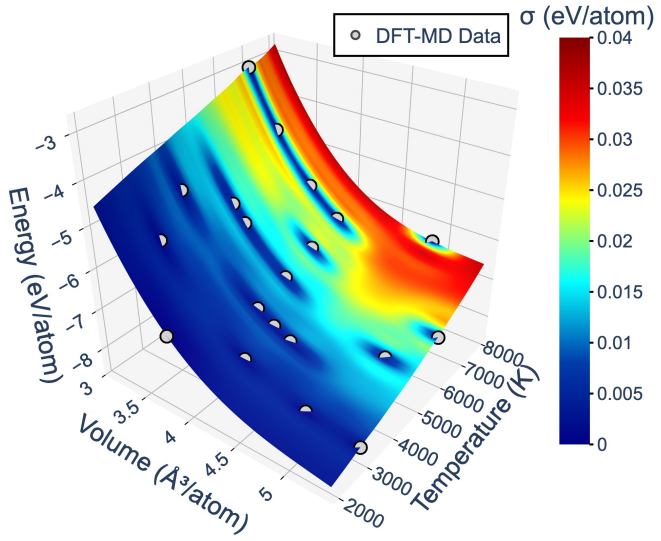


Constrained GP Equation of State[6]

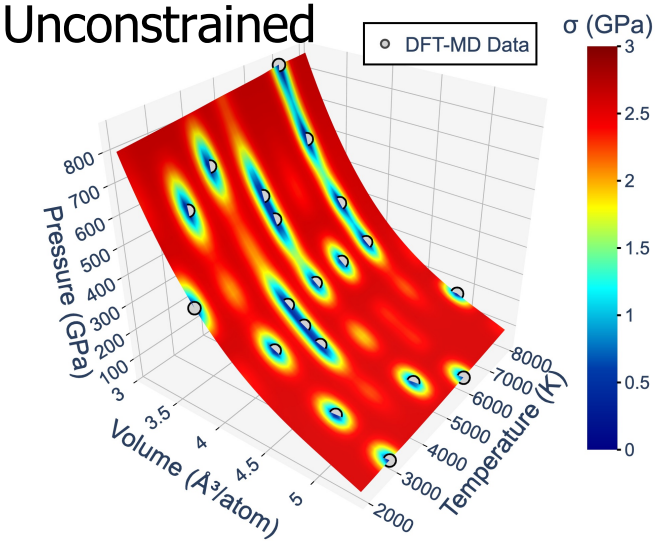
JOHNS HOPKINS WHITING SCHOOL of ENGINEERING

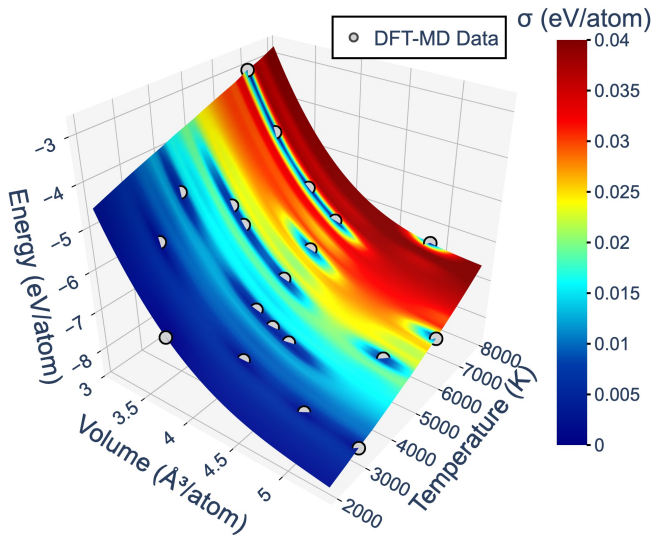
Constrained



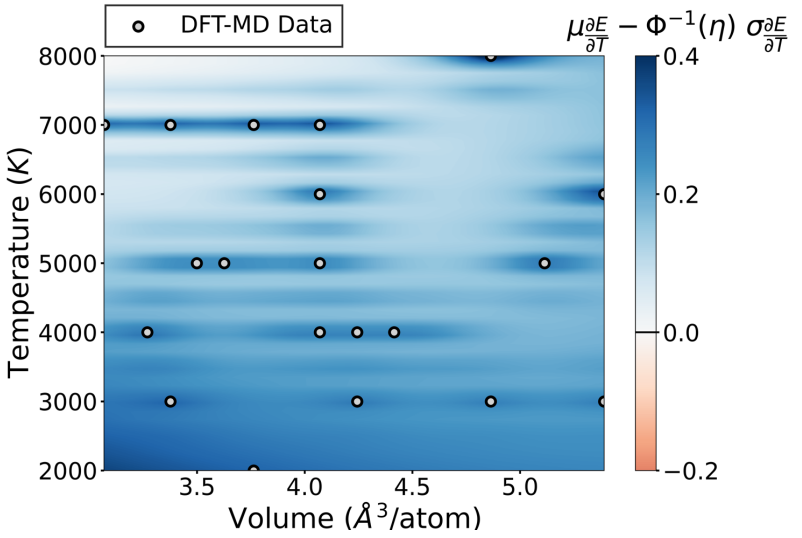


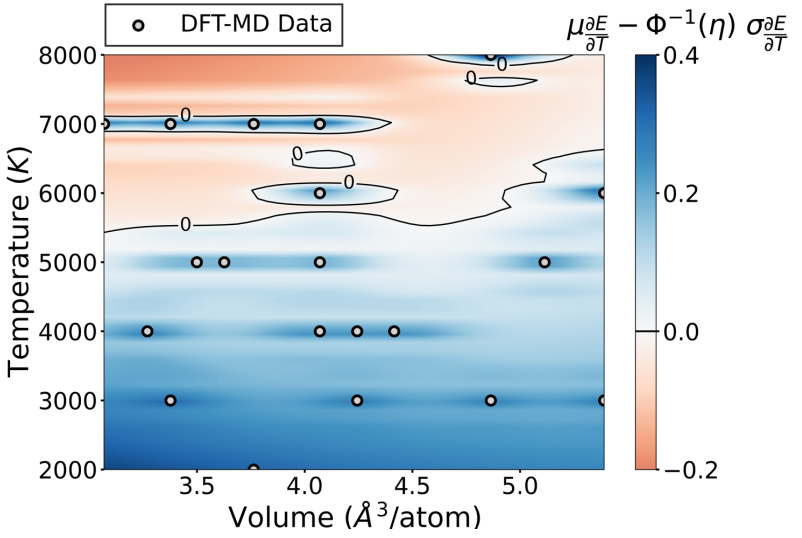
Unconstrained





Energy Stability Constraint





Gaussian Process References

1. [GP Fundamentals] Williams, C. K., & Rasmussen, C. E. (2006). *Gaussian processes for machine learning.* Cambridge, MA: MIT press.
2. [Mie-Grüneisen-Debye EOS] Y. Fei, A. Ricolleau, M. Frank, K. Mibe, G. Shen, & V. Prakapenka. (2007). Toward an internally consistent pressure scale. *Proceedings of the National Academy of Sciences*, *104*(22), 9182-9186.
3. [Diamond EOS data] F. Occelli, P. Loubeyre, & R. LeToullec, (2003). Properties of diamond under hydrostatic pressures up to 140 GPa. *Nature materials*, *2*(3), 151-154.
4. [Thermodynamically Consistent GP] J.A. Gaffney, L. Yang, & S. Ali. (2022). Constraining Model Uncertainty in Plasma Equation-of-State Models with a Physics-Constrained Gaussian Process. *arXiv preprint arXiv:2207.00668*.
5. [Constrained GP] L.P. Swiler, M. Gulian, A.L. Frankel, C. Safta, & J.D. Jakeman. (2020). A survey of constrained Gaussian process regression: Approaches and implementation challenges. *Journal of Machine Learning for Modeling and Computing*, 1(2).
6. [Constrained GP EOS] H. Sharma, J.A. Gaffney, D. Tsapetis, & M.D. Shields. (2024). Learning thermodynamically constrained equations of state with uncertainty. *APL Machine Learning*, 2(1).

Acknowledgments

- Work presented here was sponsored by the Defense Threat Reduction Agency, Department of Energy, and Army Research Laboratory.
- Code developments in UQpy have been supported by the Johns Hopkins Open-Source Programs Office.

Ponkrshnan Thiagarajan

Connor Krill

Georgios Pasparakis

MSEE MATERIALS SCIENCE IN EXTREME ENVIRONMENTS UNIVERSITY RESEARCH ALLIANCE

Github Repo