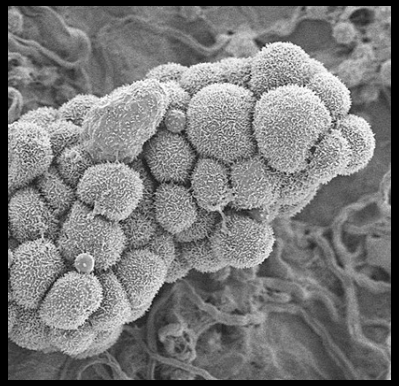
AI for Science

Juntao Yang, Solutions Architect, NVIDIA AI Technology Center (Singapore)

Deep Learning Everywhere

INTERNET & CLOUD

Image Classification
Speech Recognition
Language Translation
Language Processing
Sentiment Analysis
Recommendation



MEDICINE & BIOLOGY

Cancer Cell Detection
Diabetic Grading
Drug Discovery

MEDIA & ENTERTAINMENT

Video Captioning
Video Search
Real Time Translation

SECURITY & DEFENSE

Face Detection
Video Surveillance
Satellite Imagery

AUTONOMOUS MACHINES

Pedestrian Detection
Lane Tracking
Recognize Traffic Sign

OPTIMIZING QUALITY INSPECTIONS WITH AI

Due to the increasingly sophisticated design of its cars and the high-quality standards at Audi, the company inspects all components — doors, engine hoods, fenders, etc. — in its press shop.

In addition to visual inspection by Audi employees, several small deep learning-based cameras trained on NVIDIA GPUs installed directly in the presses detect the finest cracks in sheet metal with the utmost precision in a matter of seconds.

NVIDIA


PUMPING AI INTO THE OIL & GAS INDUSTRY
NVIDIA and Baker Hughes (BHGE) are using AI and GPU-accelerated computing to help companies distill oceans of data and reduce the cost of finding, extracting, processing and delivering oil.
BHGE's AI Factory and NVIDIA's end-to-end AI supercomputing solutions — from the NVIDIA DGX-1 in data centers, to the NVIDIA DGX Station deskside or at remote locations, to NVIDIA Jetson at the edge — can unlock insights from data that was previously as hidden as the oil underground.
NVIDIA
BAKER HUGHES
a GE company

CHASING 10^{60} CHEMICAL COMPOUNDS

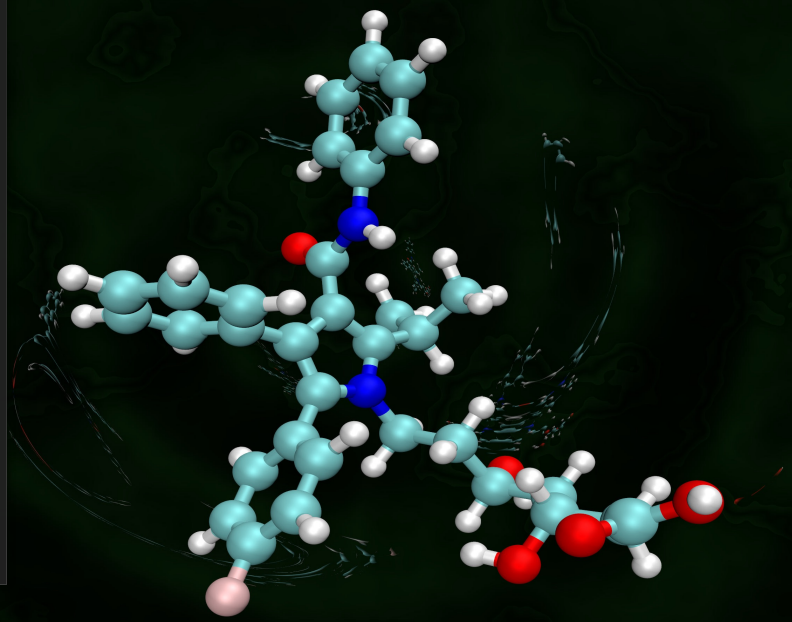
Identifying molecules with desirable chemical properties is central to many industries. In the chemical space of 10^{60} conceivable compounds, only 10^{8} have been synthesized.

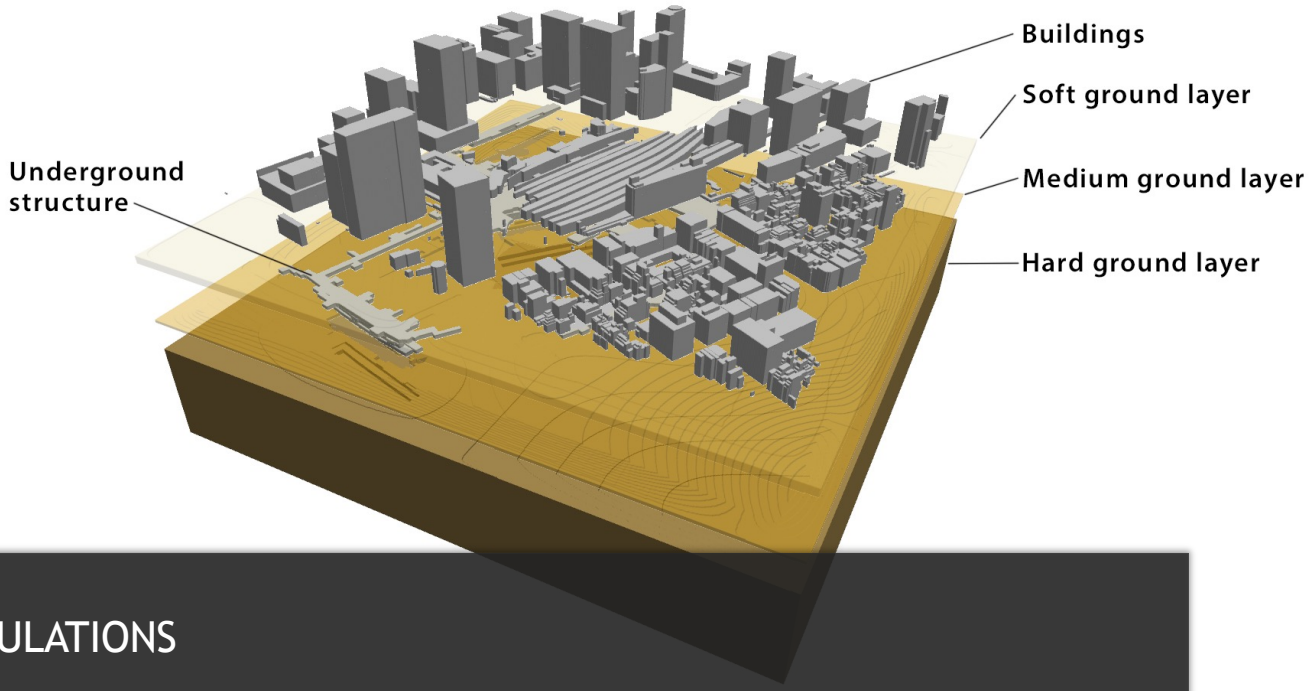
Screening even a small fraction of the remaining compounds with legacy methods would take 100 node-seconds per compound.

Researchers at Dow are using GPU-powered deep learning to deliver completely novel molecular structures with specific properties.

The AI produced 3M promising chemical leads in 1 day on an NVIDIA DGX.

NVIDIA Dow





IMPROVING EARTHQUAKE SIMULATIONS

As urban settlements grow, improved understanding of earthquakes and their effects on buildings and infrastructure is increasingly important.

To better manage cities vulnerable to these complex disasters, a team from the University of Tokyo's Earthquake Research Institute developed an earthquake simulation using deep learning on the Summit Supercomputer with NVIDIA V100 Tensor Core GPUs.

The team achieved a 4x speed-up of their existing algorithm while simulating both shaking ground and urban structures.

NVIDIA. 東京大学 THE UNIVERSITY OF TOKYO

Pictured: City model of Tokyo Station and the surrounding area with underground and building structures as well as two-layered ground. Image credit: University of Tokyo

AI4Science with Modulus

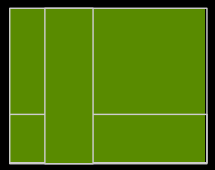
Yang Juntao, Nvidia AI Technology Center (Singapore)

Accelerated Computing

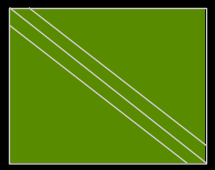
Linear Algebra, FFT, RNG and Basic Math

NVIDIA CUDA

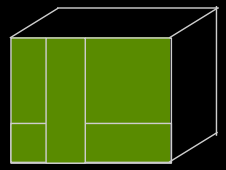
OpenACC
More Science, Less Programming



cuBLAS

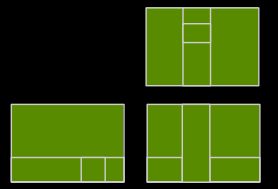


cuSPARSE

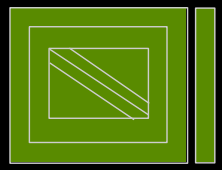


cuTENSOR

cuSOLVER

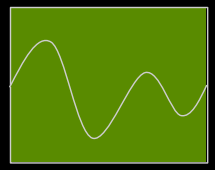


CUTLASS

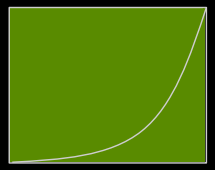


AMGX

cuRAND



cuFFT



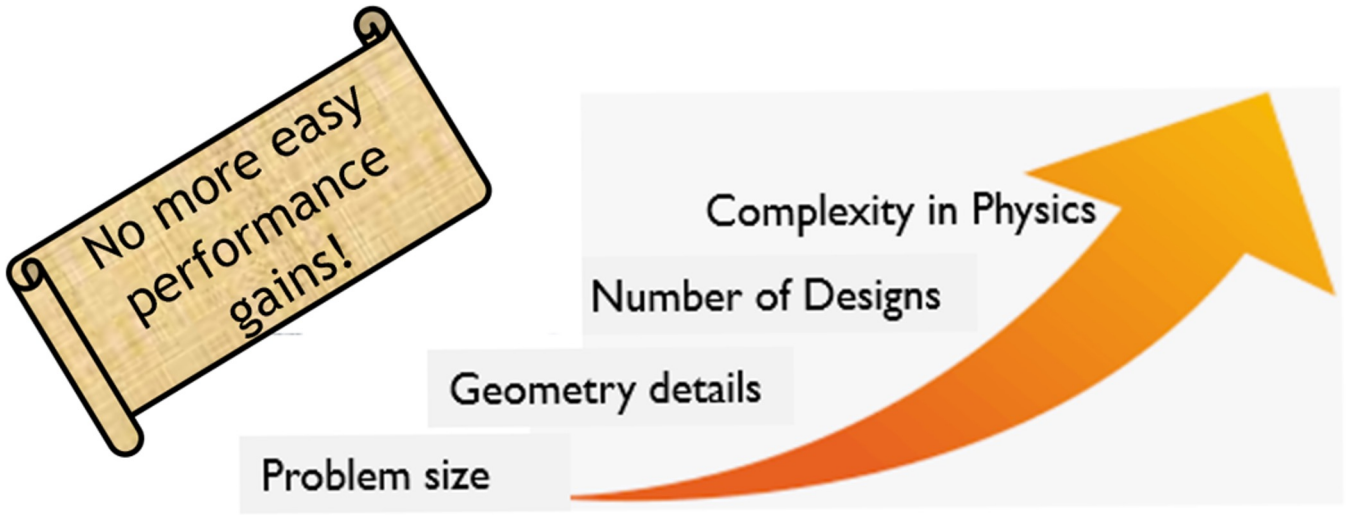
CUDA Math API

Saturating Performance in Traditional HPC

Simulations are getting larger and more complex

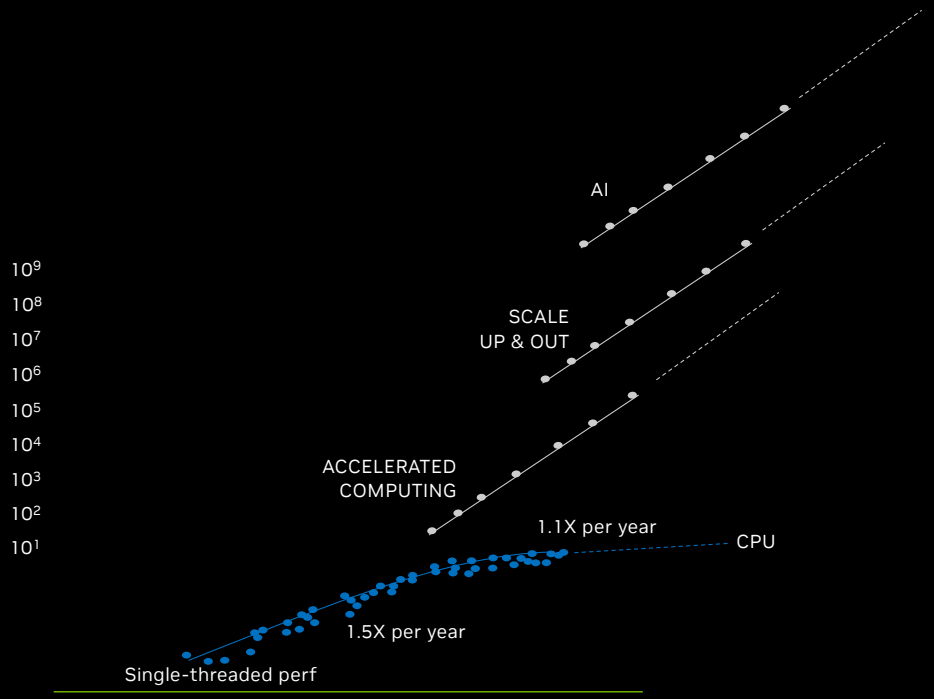
Traditional solution methods are:

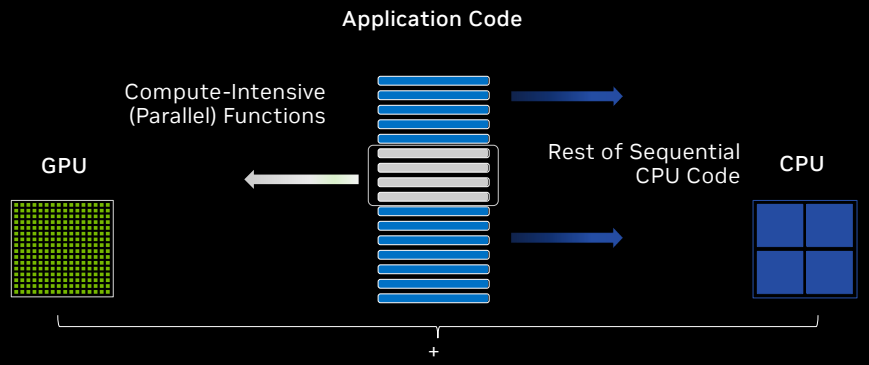
- Computationally Expensive
- Plagued by Domain Discretization Techniques
- Not suitable for Data-assimilation or Inverse problems



Getting Million-X Speedups to Power AI and Scientific Computing

Accelerated Computing + AI Provides the Compute Required



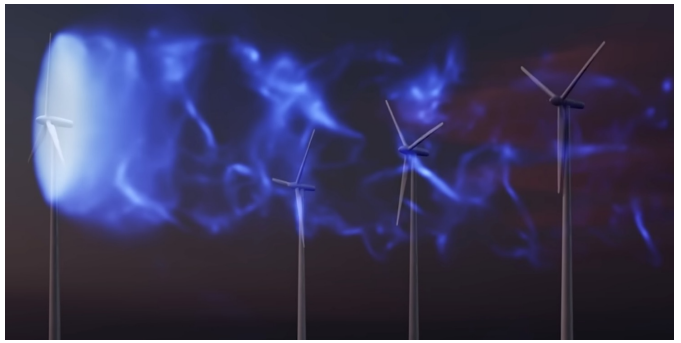


NVIDIA Modulus

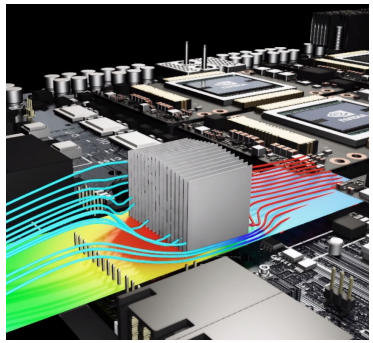
One Step Closer to Real-time Digital Twin

NVIDIA Modulus is a neural network framework that blends the power of physics in form of governing partial differential equations (PDEs) with data to build high-fidelity, parameterized surrogate models with near-real-time latency. It offers:

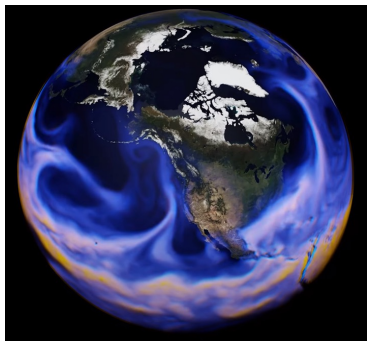
- **AI Toolkit**
  - Offers building blocks for developing physics-ML surrogate models as well as data driven neural operator models.
- **Scalable Performance**
  - Solves larger problems faster by scaling from single GPU to multi-node implementation
- **Near-Real-Time Inference**
  - Provides parameterized system representation that solves for multiple scenarios in near real time, trains once offline to infer in real time repeatedly
- **Easy Adoptability**
  - Includes APIs for domain experts to work at a higher level of abstraction. Extensible to new applications with reference applications serving as starting points



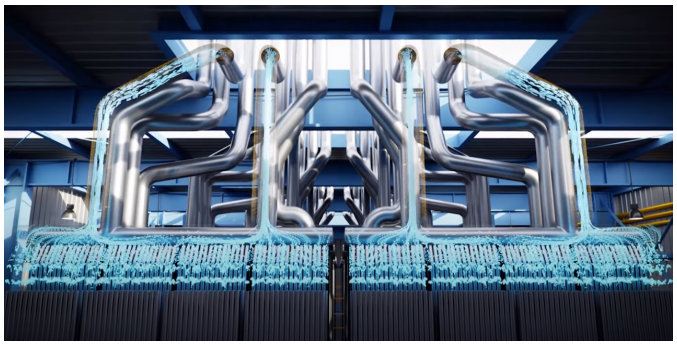
Wind Farm Super Resolution



FPGA Heatsink Design Optimization



Extreme Weather Prediction

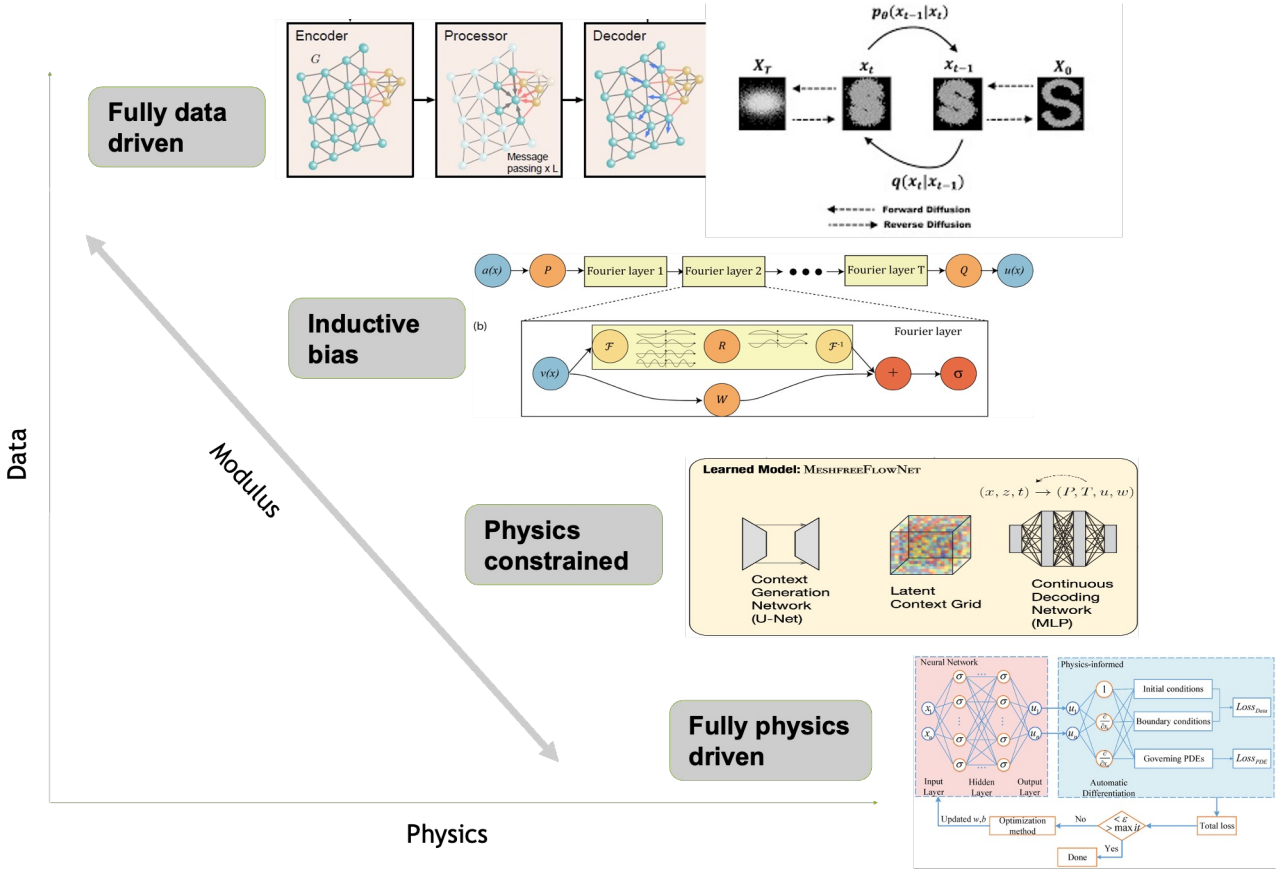


Industrial Digital Twin

NVIDIA

Open-Source AI Toolkit for Physics-based Machine Learning

https://github.com/NVIDIA/modulus



- Modulus Model Zoo - Diverse Physics-ML approaches:
  - fully Physics driven AI models
  - fully data driven AI models
  - hybrid (data + Physics) AI models

- **Neural Operators:**
  - Fourier Neural Operator family (FNO, AFNO, PINO)
  - DeepONet
- **GNNs:**
  - GraphCast
  - MeshGraphNet ..
- **Diffusion Models:**
  - DDPM++
  - NCSN++
  - ADM ..
- **PDE informed Neural Networks:**
  - Fourier Feature Network
  - Spatial-temporal Fourier Feature Networks
  - Super Resolution Net ...

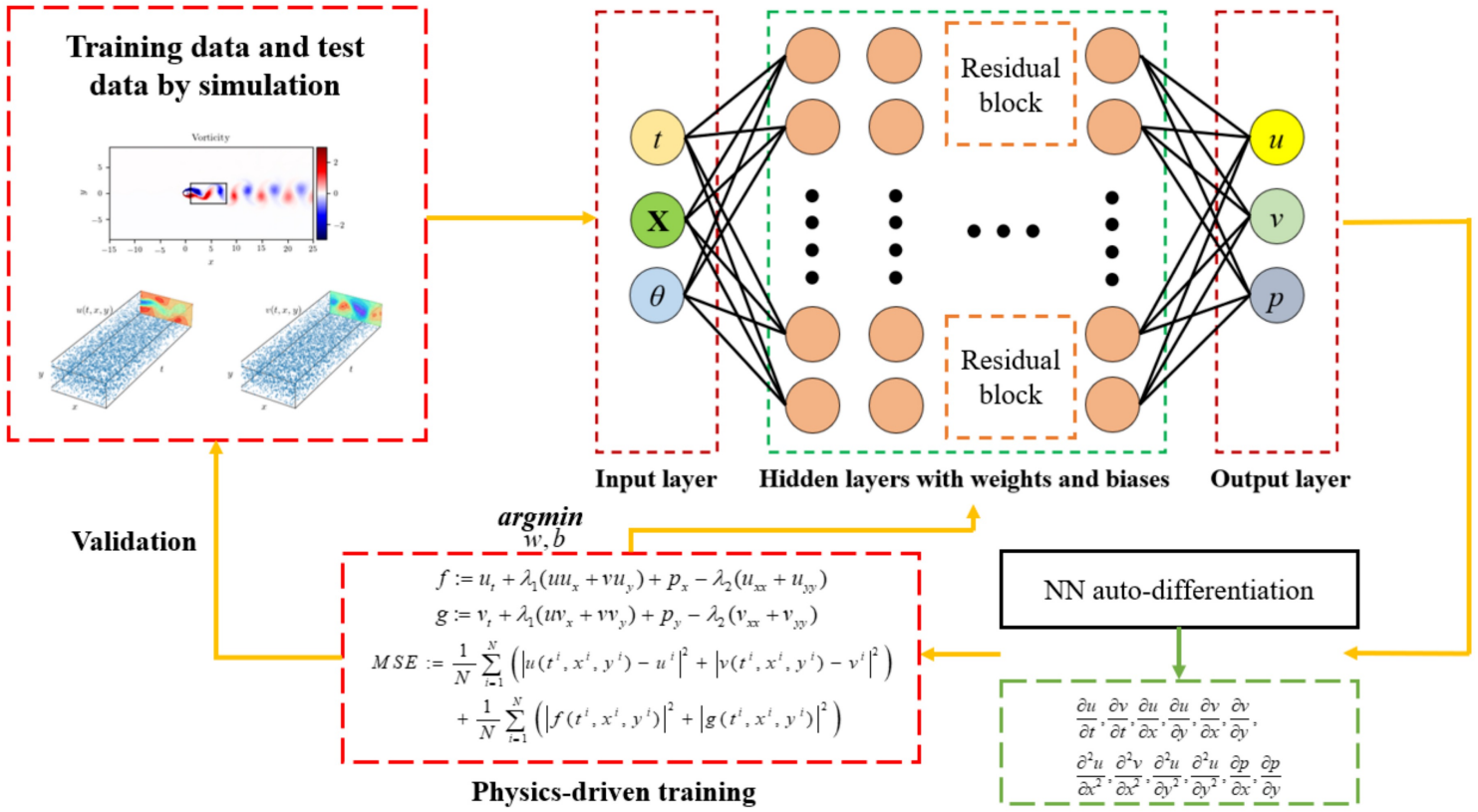
NVIDIA

Physics Informed Machine Learning

NVIDIA

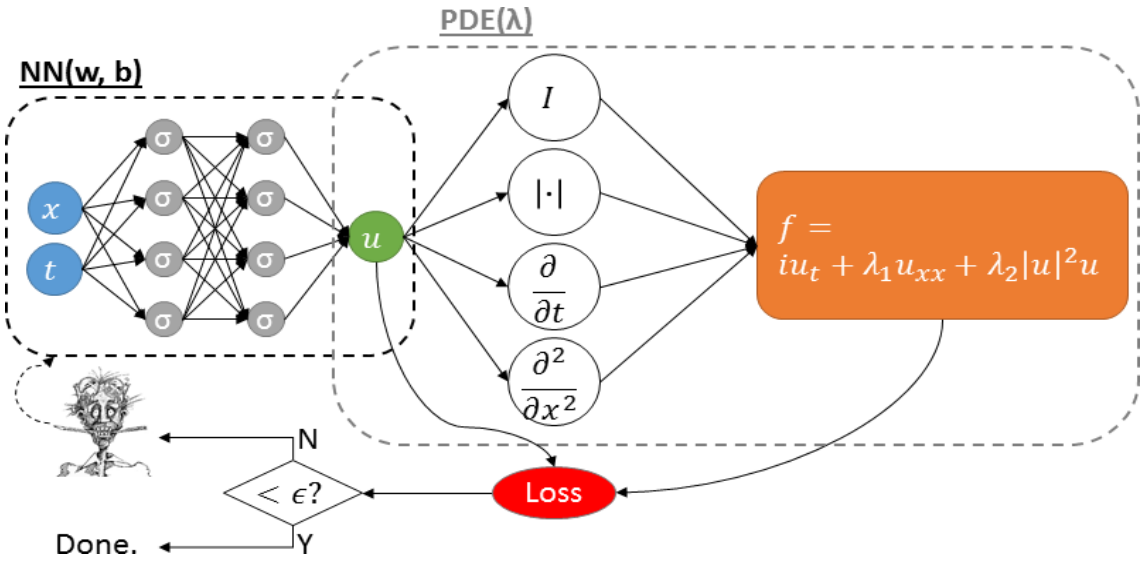
Physics Informed Machine Learning

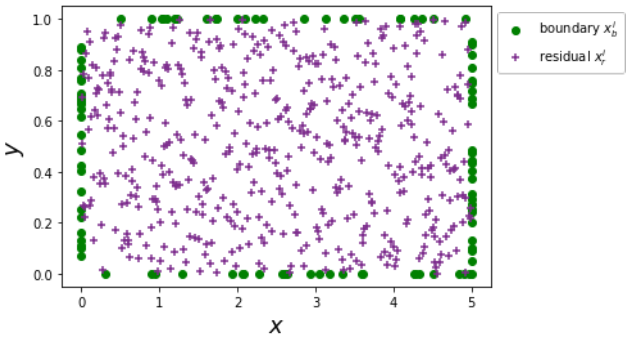
Using Partial Differential Equation Residual as Loss or Hybrid with Data Constraint



What is a PINN? Physics-Informed Neural Network

We employ two (or more) NNs that share the same parameters





- Minimize:

$$MSE = MSE_u + MSE_f,$$

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

$$MSE_f = \frac{1}{N_f}\sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

NVIDIA

Setup for PINNs (forward)

Consider the partial differential equations (PDEs):

$$\mathcal{L}[u](\mathbf{x}) = f(\mathbf{x}) \text{ in } U \subset \mathbb{R}^d$$
$$\mathcal{B}[u](\mathbf{x}) = g(\mathbf{x}) \text{ in } \partial U$$

Goal: Find a neural net $h_\theta(\mathbf{x})$ that approximates the PDE solution.

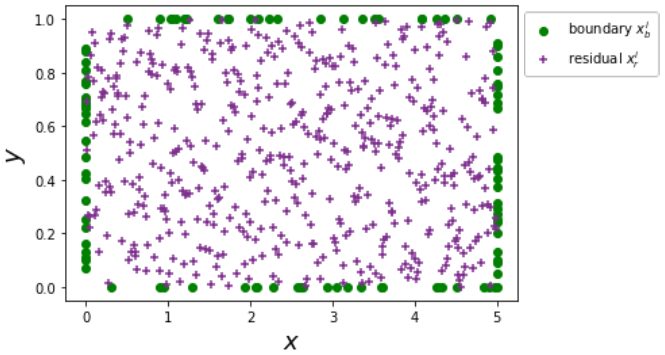
What are known to us?

Pointwise PDE data: i.i.d. samples

Residual data: $(\mathbf{x}_r^i, f(\mathbf{x}_r^i))$ where $\mathbf{x}_r^i \in U$

Boundary data: $(\mathbf{x}_b^i, g(\mathbf{x}_b^i))$ where $\mathbf{x}_b^i \in \partial U$

Physics Equations: $\mathcal{L}[u] - f = 0$ and $\mathcal{B}[u] - g = 0$



Physics Informed Neural Networks

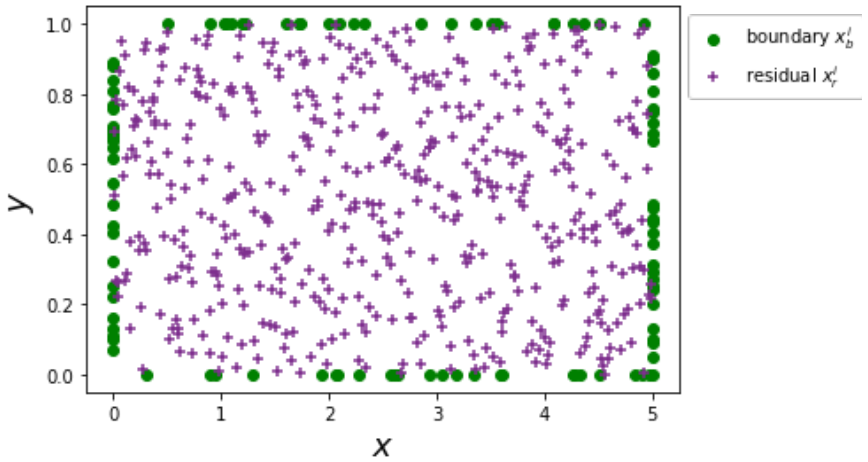
Idea: Encode "physics equations" and "data" into the neural net $h_\theta(\mathbf{x})$

DEF: Let $\boldsymbol{m} = (m_r, m_b)$ The prototype PINN loss is

$$\text{Loss}_{\boldsymbol{m}}^{\text{PINN}}(\theta) = \frac{1}{m_r}\sum_{i=1}^{m_r}\left(\mathcal{L}[h_\theta](\mathbf{x}_r^i) - f(\mathbf{x}_r^i)\right)^2 + \frac{1}{m_b}\sum_{i=1}^{m_b}\left(\mathcal{B}[h_\theta](\mathbf{x}_b^i) - g(\mathbf{x}_b^i)\right)^2$$

Method: $\underset{\theta}{\text{minimize}}\ \text{Loss}_{\boldsymbol{m}}^{\text{PINN}}(\theta) \implies \theta_{\boldsymbol{m}}^*$

$h_{\boldsymbol{m}} := h_{\theta_{\boldsymbol{m}}^*}$ is a **physics informed neural network**



Why PINNs?

Simple and easy to implement it

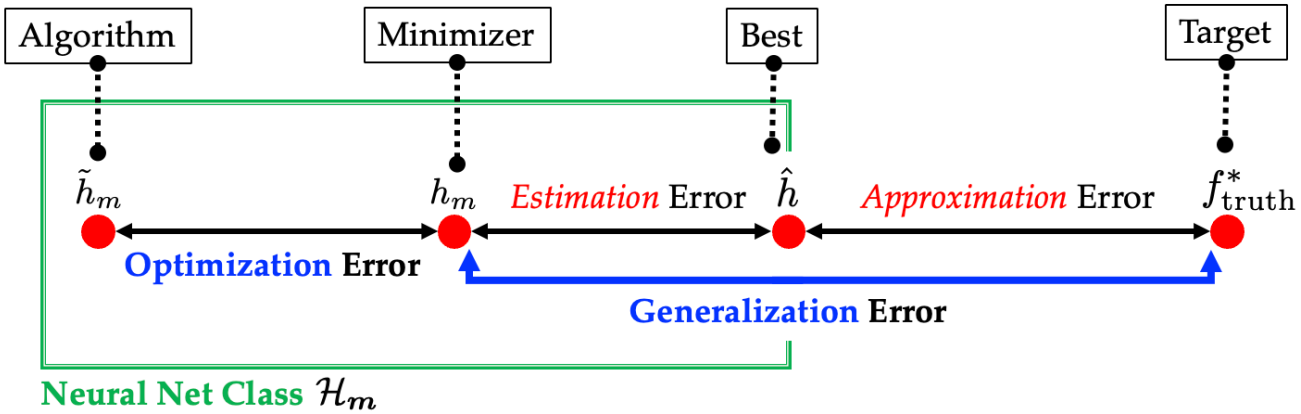
Solve PDEs without **meshes**

Great potential in **solving *inverse* problems**

Take advantage of rapid development of **deep learning**

Error Decomposition

$$\underset{\theta}{\text{minimize}}\ \text{Loss}_{\boldsymbol{m}}^{\text{Höld}}(\theta) \implies \theta_{\boldsymbol{m}}^* \implies h_{\boldsymbol{m}} := h_{\theta_{\boldsymbol{m}}^*}$$



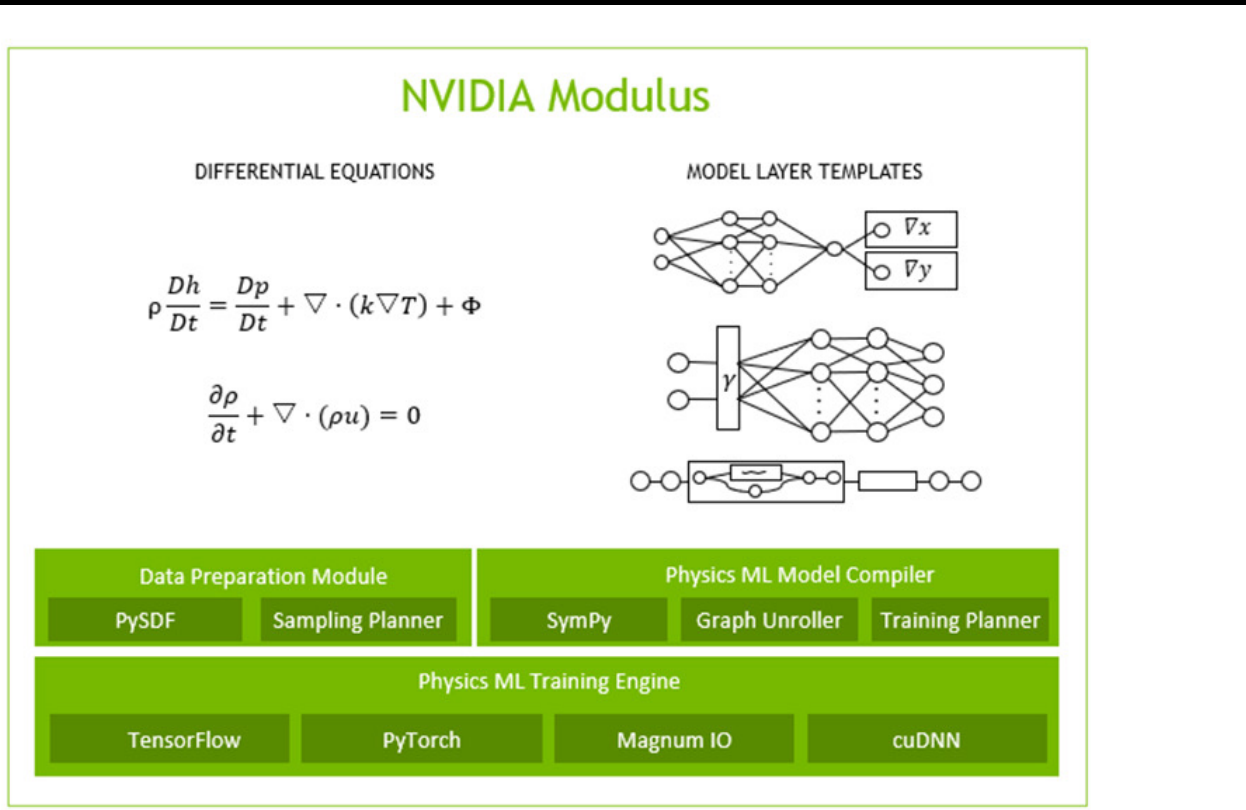
Neural net classes $\mathcal{H}_{\boldsymbol{m}}$ have to be properly chosen for the convergence!

Equivalently,

$$\min_{h \in \mathcal{H}_{\boldsymbol{m}}} \text{Loss}_{\boldsymbol{m}}^{\text{Höld}}(h)$$

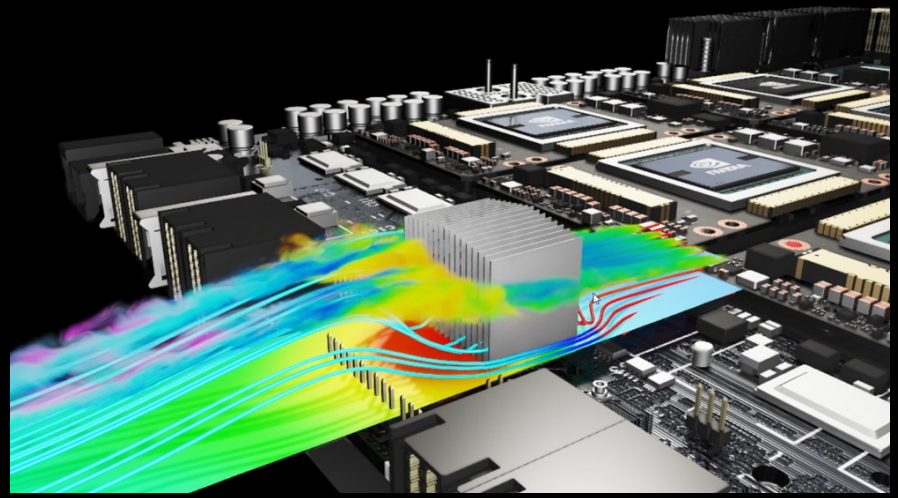
Physics Informed Neural Network

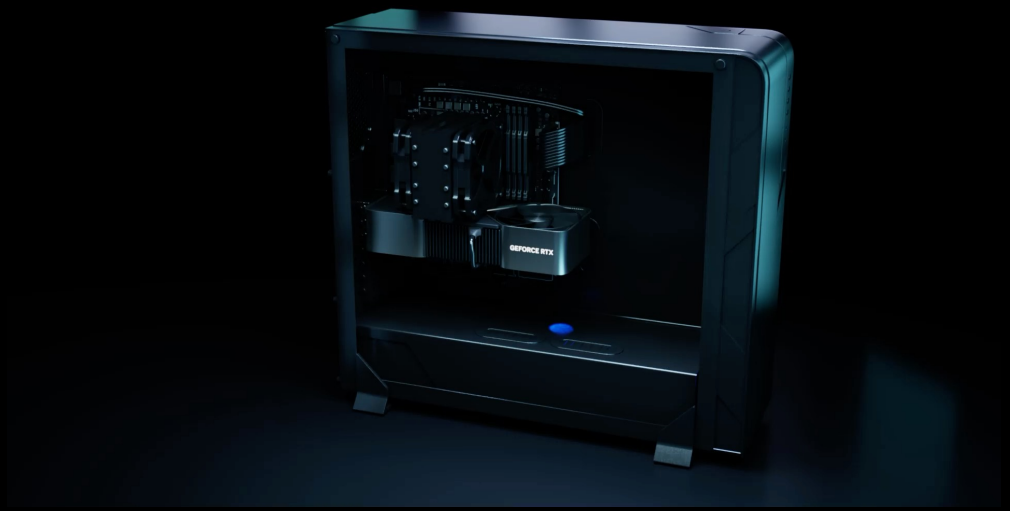
Using Partial Differential Equation Residual as Loss or Hybrid with Data Constraint



Cooling Design with Physics Informed Neural Network

Using Partial Differential Equation Residual as Loss or Hybrid with Data Constraint





HRSG FLUID ACCELERATED CORROSION SIMULATION — SIEMENS ENERGY

Use Case

- Detecting and predicting point of corrosion in heat recovery steam generators (HRSGs)

Challenges

- Using standard simulation to detect corrosion, it took SE at least couple of weeks, and the overall process took 14-16 weeks for every HRSG unit.

Solution

- Using NVIDIA Modulus Physics-Informed Neural Network, SE simulates the corrosive effects of heat, water and other conditions on metal over time to fine-tune maintenance needs.
- SE can replicate and deploy HRSG plant digital twins worldwide with NVIDIA Omniverse.

NVIDIA Solution Stack

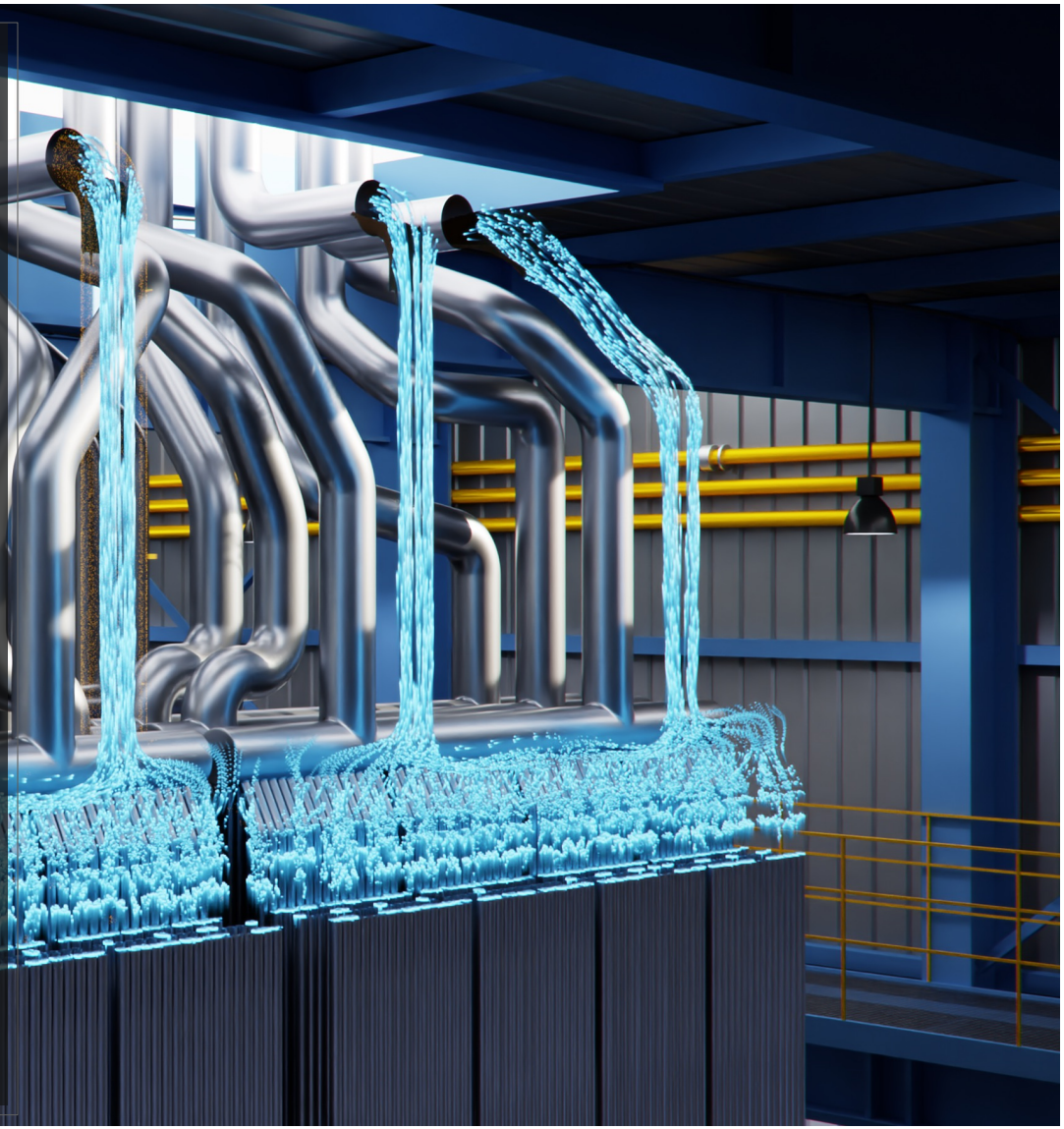
- Hardware: NVIDIA V100 & A100 Tensor Core GPUs
- Software: NVIDIA Modulus, NVIDIA Omniverse

Outcome

- 10,000X speed-up and inference in seconds can reduce downtime by 70%, saving the industry $1.7 billion annually

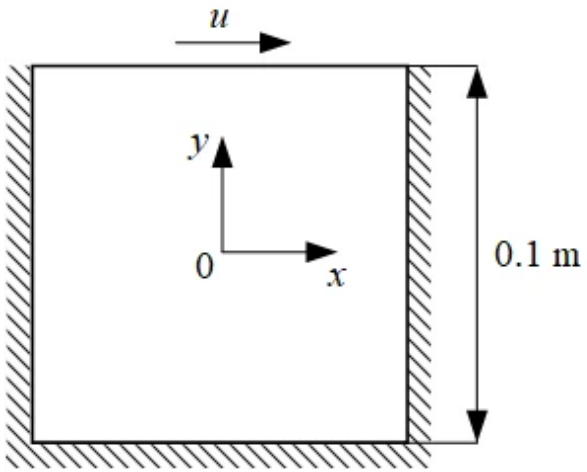
Link to Demo

NVIDIA SIEMENS energy



Example 1: Lid Driven Cavity

- How to generate a 2D geometry using Modulus Sym' geometry module;
- How to set up the boundary conditions
- How to select the flow equations to be solved
- How to impose suitable constraints
- How to post-process the results



Example 1: Lid Driven Cavity

Create a PDE Node

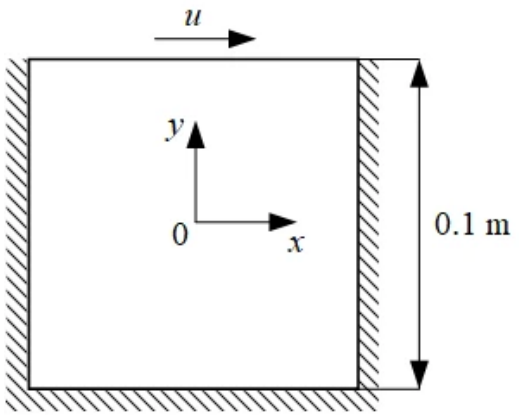
```python
from modulus.sym.eq.pdes.navier_stokes import NavierStokes

# make list of nodes to unroll graph on
ns = NavierStokes(nu=0.01, rho=1.0, dim=2, time=False)
```

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

$$u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = -\frac{\partial p}{\partial x} + \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

$$u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} = -\frac{\partial p}{\partial y} + \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right)$$

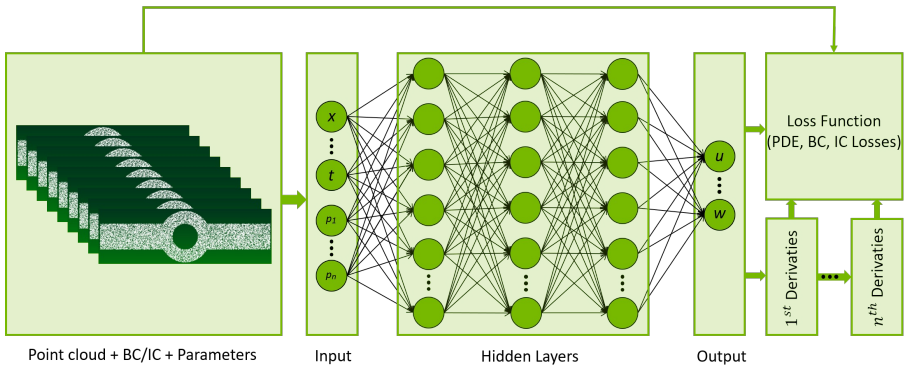
- Set the kinematic viscosity nu=0.01
- Set the density rho=1.0
- Set time=False as this is a steady-state problem
- Set dim=2 as this is a 2D problem



Example 1: Lid Driven Cavity

Create a Neural Network Node

```
from modulus.sym.hydra import instantiate_arch
from modulus.sym.key import Key

flow_net = instantiate_arch(
    input_keys=[Key("x"), Key("y")],
    output_keys=[Key("u"), Key("v"), Key("p")],
    cfg=cfg.arch.fully_connected,
)
nodes = ns.make_nodes() + [flow_net.make_node(name="flow_network")]
```

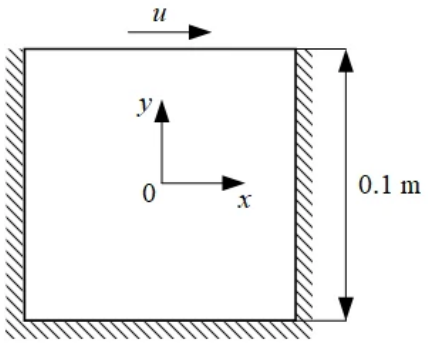


- Create a neural network to approximate the solution of the Navier-Stokes equations for the given boundary conditions. The neural network will have two inputs x, y and three outputs u,v,p.
- Once the PDE and network are defined, we create a list of nodes to pass different constraints that need to be satisfied for this problem.

Example 1: Lid Driven Cavity

Create Geometry

```python
from sympy import Symbol, Eq, Abs
from modulus.sym.geometry.primitives_2d import Rectangle

# make geometry
height = 0.1
width = 0.1
x, y = Symbol("x"), Symbol("y")
rec = Rectangle((-width / 2, -height / 2), (width / 2, height / 2))
```



- Modulus Sym allows geometry creation in various ways, including using the CSG module. The CSG module supports numerous primitive shapes, from 2D shapes like rectangles and circles, to 3D shapes like spheres and cones.
- In Modulus Sym, a Rectangle is defined by the coordinates of two opposite corners. The symbolic variable is later used for subsampling the geometry, which assists in defining various boundaries and interior regions.

Example 1: Lid Driven Cavity

Set up the Domain and add PDE Constraints

```python
from modulus.sym.domain import Domain

# make ldc domain
ldc_domain = Domain()
```

```python
from modulus.sym.domain.constraint import PointwiseInteriorConstraint

# interior
interior = PointwiseInteriorConstraint(
    nodes=nodes,
    geometry=rec,
    outvar={"continuity": 0, "momentum_x": 0, "momentum_y": 0},
    batch_size=cfg.batch_size.Interior,
    lambda_weighting={
        "continuity": Symbol("sdf"),
        "momentum_x": Symbol("sdf"),
        "momentum_y": Symbol("sdf"),
    },
)
ldc_domain.add_constraint(interior, "interior")
```

- Domain object will contain all needed information about **constraints**, **validators**, **inferencers**, and **monitors**.
- Use the **PointwiseInteriorConstraint** class to sample points in the interior of a geometry
- The **outvar** argument specifies the equations to solve as a dictionary.
- The **lambda_weighting** parameter is used to determine the weights for different losses.
- Each equation at each point is weighted by its distance from the boundary using the Signed Distance Field (SDF) of the geometry.

$$L_{continuity} = \frac{V}{N}\sum_{i=0}^{N}(0 - continuity(x_i, y_i))^2$$

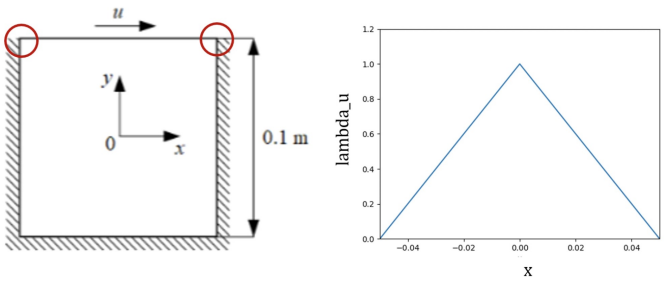
$$L_{momentum_x} = \frac{V}{N}\sum_{i=0}^{N}(0 - momentum_x(x_i, y_i))^2$$

$$L_{momentum_y} = \frac{V}{N}\sum_{i=1}^{n}(0 - momentum_y(x_i, y_i))^2$$

Example 1: Lid Driven Cavity

Add boundary Constraints

```python
from modulus.sym.domain.constraint import PointwiseBoundaryConstraint

# top wall
top_wall = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=rec,
    outvar={"u": 1.0, "v": 0},
    batch_size=cfg.batch_size.TopWall,
    lambda_weighting={"u": 1.0 - 20 * Abs(x), "v": 1.0},
    criteria=Eq(y, height / 2),
)
ldc_domain.add_constraint(top_wall, "top_wall")

# no slip
no_slip = PointwiseBoundaryConstraint(
    nodes=nodes,
    geometry=rec,
    outvar={"u": 0, "v": 0},
    batch_size=cfg.batch_size.NoSlip,
    criteria=y < height / 2,
)
ldc_domain.add_constraint(no_slip, "no_slip")
```



- Sampling a boundary uses a **PointwiseBoundaryConstraint** object, which samples the entire boundary of the specified geometry.
- A particular boundary of the geometry can be sub-sampled by using the **criteria** argument
- Avoid sharp discontinuity at the top corners by specifying the weighting for the top wall such that the weight of the loss varies continuously and is 0 on the corners.

Example 1: Lid Driven Cavity

Add Validator

```
from modulus.sym.hydra import to_absolute_path
from modulus.sym.domain.validator import PointwiseValidator
from modulus.sym.utils.io import (
    csv_to_dict,
    ValidatorPlotter,
    InferencerPlotter,
)

# add validator
mapping = {"Points:0": "x", "Points:1": "y", "U:0": "u", "U:1": "v", "p": "p"}
openfoam_var = csv_to_dict(
    to_absolute_path("openfoam/cavity_uniformVel0.csv"), mapping
)
openfoam_var["x"] += -width / 2  # center OpenFoam data
openfoam_var["y"] += -height / 2  # center OpenFoam data
openfoam_invar_numpy = {
    key: value for key, value in openfoam_var.items() if key in ["x", "y"]
}
openfoam_outvar_numpy = {
    key: value for key, value in openfoam_var.items() if key in ["u", "v"]
}
openfoam_validator = PointwiseValidator(
    nodes=nodes,
    invar=openfoam_invar_numpy,
    true_outvar=openfoam_outvar_numpy,
    batch_size=1024,
    plotter=ValidatorPlotter(),
)
ldc_domain.add_validator(openfoam_validator)
```

- Modulus Sym results can be validated by importing ground truth or simulation data and creating a validator. This will show validation error on tensorboard , and output .vtp files for comparing the Modulus and imported results in ParaView.
- Add a validator to the domain and compare the solution computed by Modulus Sym with results from OpenFOAM
- Convert the validation data into a dictionary of NumPy variables for input and output by **csv_to_dict**
- The validation data is incorporated into the domain using **PointwiseValidator**, with the dictionary of generated NumPy arrays for input and output variables as input.

Example 1: Lid Driven Cavity

Training and postprocessing

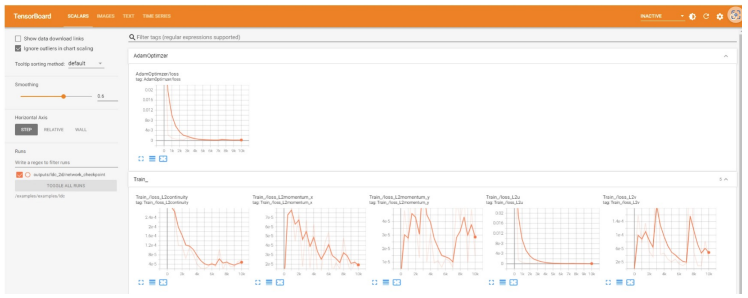
```python
import os
from modulus.solver import Solver

# make solver
slv = Solver(cfg, ldc_domain)

# start solver
slv.solve()
```

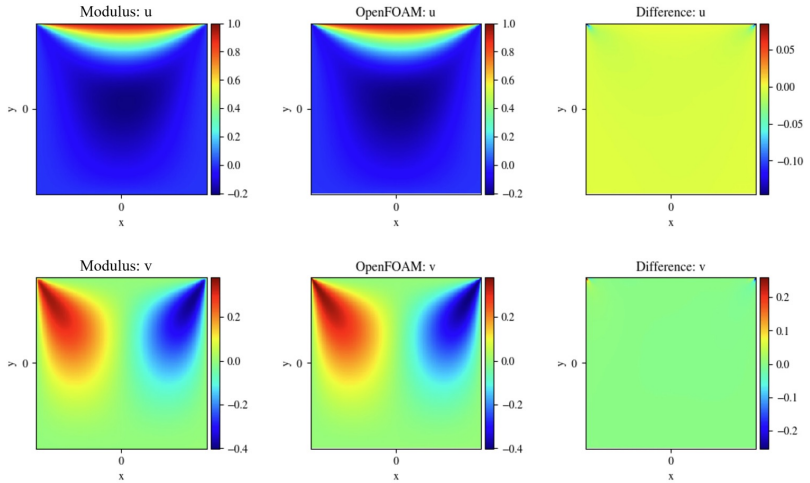
- Execute the Python script will train the neural network.

```
python ldc_2d.py
```

- Use tensorboard to visualize the loss curves and learning rate:

```
tensorboard --logdir=./ --port=7007
```

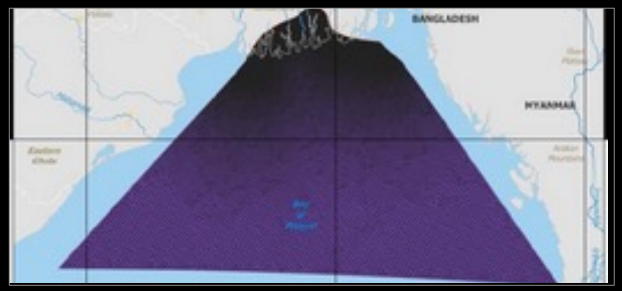




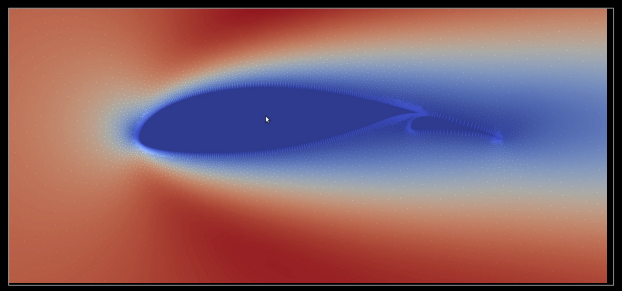
Refer to the Jupyter notebook for the full code.

NVAITC AI4Science

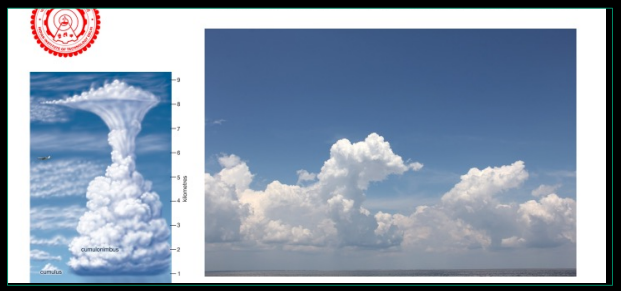
Solving scientific problem with AI, Improve AI with science



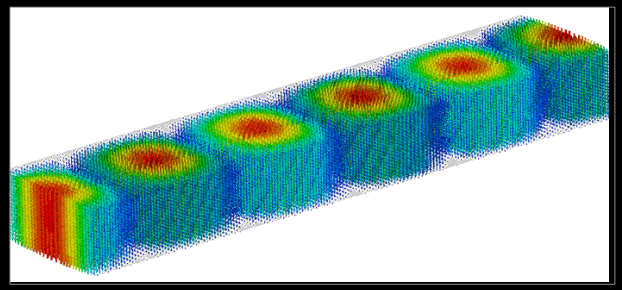
Physics informed neural network for hydro-morphodyanmics, Coventry University



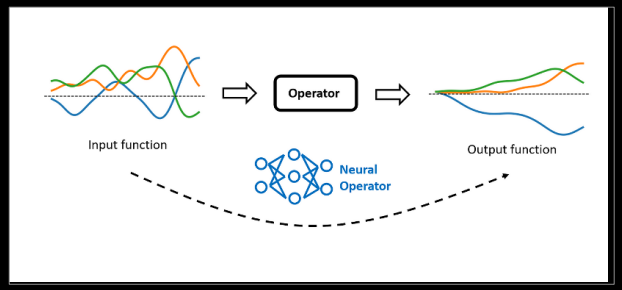
AI model based two element air flaps control, Temasek Lab Singapore



Physics Informed AI for cloud modeling, IIT



Physics Informed AI for EM Waveguide Design, Mahindra University



ReLU DNN expressivity analysis for two scale PDE
Nanyang Technological University

XAI for extreme weather prediction, National University of Singapore

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

Framework for optimization of two element airfoil using Nvidia Modulus, a physics informed neural network solver [1]

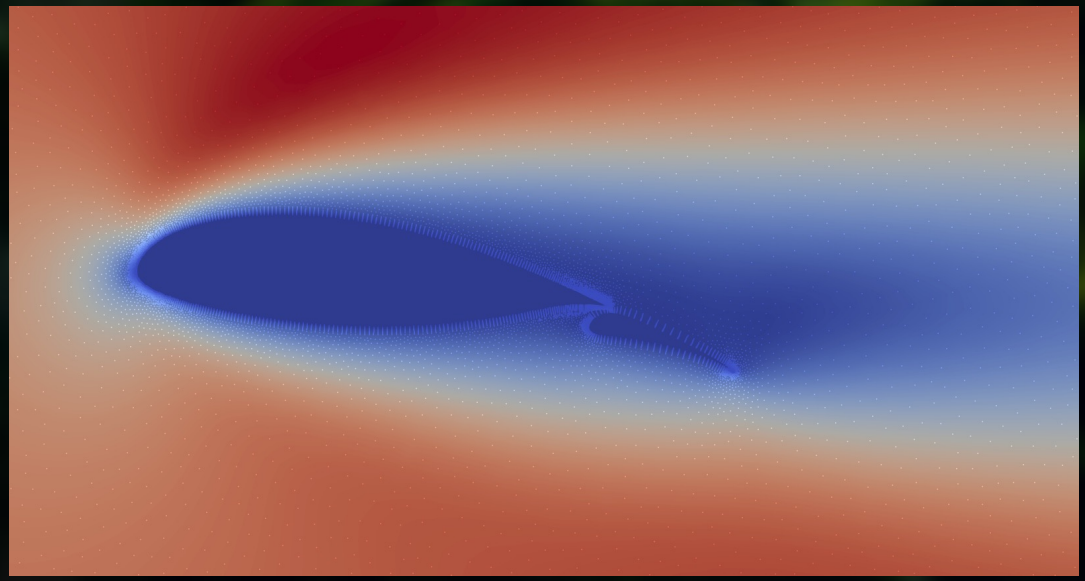
TAY Wee-Beng[1] , LIAO-YANG Tian-Chun[2], LUO Hong-Rui[2], CHEN Kai-Peng[2] , YANG Jun-Tao[3], See Simon[3] and KHOO Boo-Cheong[2]
Tayweebeng@nus.edu.sg
[1] Temasek Labs@NUS, National University of Singapore
[2] Mechanical Engineering, National University of Singapore
[3] Nvidia AI Technology Center, Singapore

AIAA SciTech Forum
8-12 January 2024, Orlando, FL
AIAA SCITECH 2024 Forum

Framework for optimization of two-element airfoil using Nvidia Modulus, a physics informed neural network solver

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

A framework has been designed to optimize the performance of two-element airfoils using an open-source physics informed neural network solver under certain aerodynamics constraints, The solver used is the Nvidia Modulus. The objective function is constructed such that with a single training, it can predict the flow fields and force coefficients for a range of angles of attack and airfoil thickness. Optimization is then performed using the trained network to improve the airfoil performance. Results show that this trained network can predict the flow fields and force coefficients with reasonable accuracy much faster than traditional computational fluid dynamics solvers. When coupled with an optimization routine, it can also predict maximum cl, cl/cd and endurance coefficient.



PINNs prediction of velocity field for two element air foil

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- Multi-element airfoil is more efficient
- But design variables and degree of control increases
- Traditional design optimization workflow become intractable
- Low fidelity solver e.g. XFOIL and MSES used
- But low accuracy

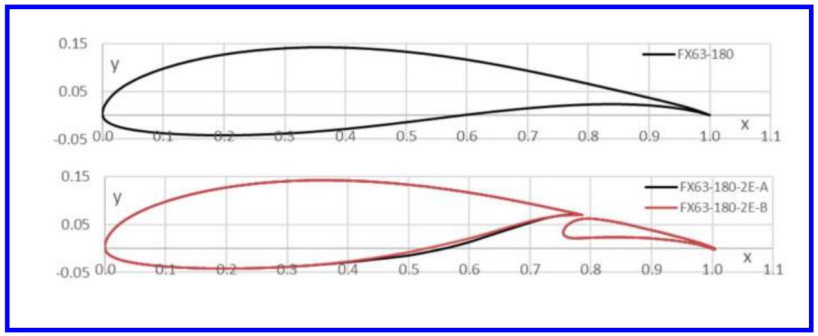


Fig. 1: A diagram showing the FX63-180 (top) and FX63-180-2E (bottom) airfoils.

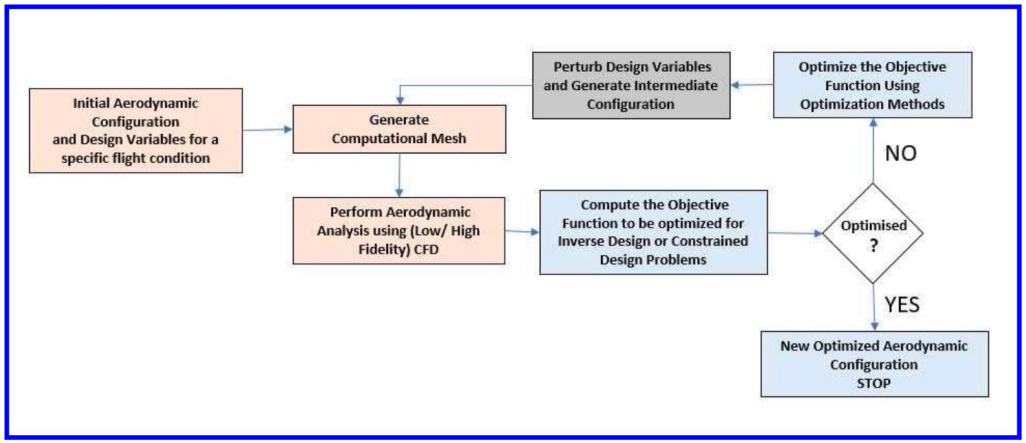
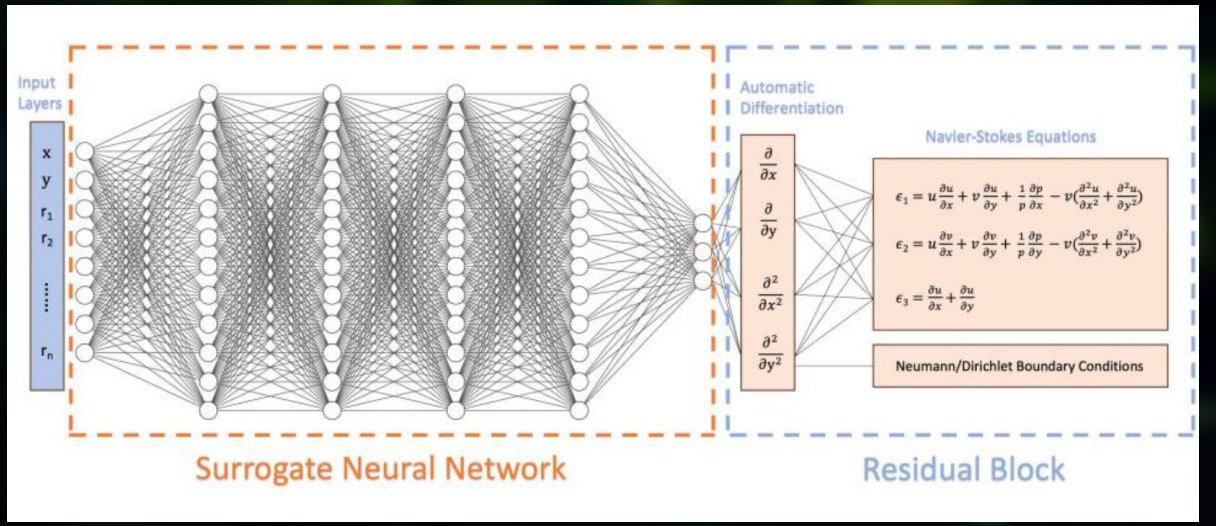


Fig. 2: Aerodynamic design optimization cycle based on low and high fidelity CFD and optimization routines.

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- Parameterized PINNs model has been found very useful for design optimization and control
- It saved the need to repeatedly compute the numerical model for the optimization loop
- This project couples parameterized PINNs model with an optimization solver for optimal drag/lift control.



NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- FX63-180-2E airfoil shape is the object of study in this investigation
- It is modified from FX63-167
- 10m/s inlet velocity with Re=100 and zero pressure outlet
- Red dot is the center of rotation
- The smallest mesh size is 5E-4m
- The mesh is used to run OpenFOAM simulation to provide reference

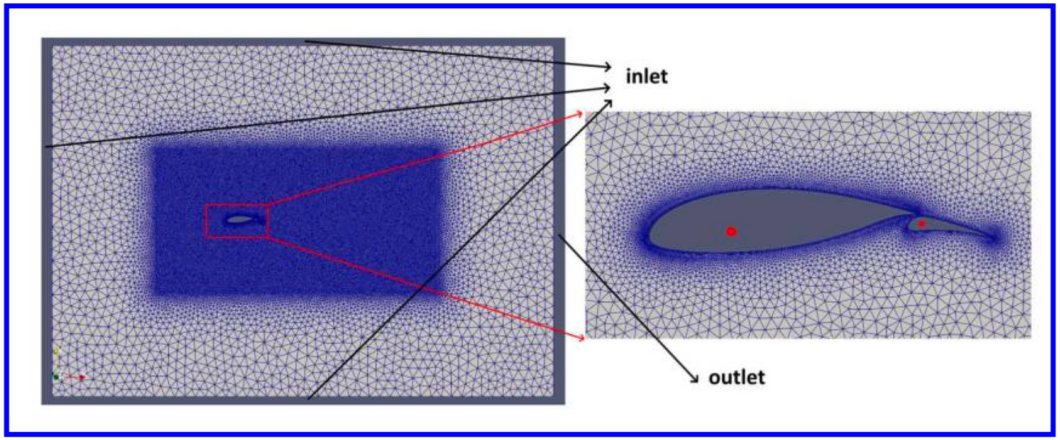


Fig. 4: Grid used for the OF simulation of flow past FX63-180-2E airfoil and its boundary conditions. The red dots denote the centers of rotation.

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- Parameterized geometry is used
- 6 hidden layer with 512 nodes each layer
- ADAM optimizer with 180,000 - 300,000 epoch
- There is a high-resolution collocation points near the boundary
- Training is done on 2 A100 gpus with Modulus build-in multi-GPU support

1) Define the aerodynamic properties and domain size
2) Import the coordinates of the airfoil
3) Define the governing equations. In this case, the steady incompressible Navier-Stokes equations are used
4) Define the input and output variables. In the current case, input variables are the spatial coordinates (x, y) while output variables are velocities (u, v) and pressure (p)
5) Selection of neural networks. Currently, the fully connected neural network is used
6) Define boundary conditions and initialize the domain and boundary with the collocation points
7) Load OpenFOAM data for validation. Create functions to calculate force coefficients

Procedures of training PINNs for the use case

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- Difference between PINNs output in comparison with OF reference
- The trailing edge has the largest error
- But the largest error is <10% which is better than low fidelity numerical solver
- Shows certain limitations of AI models
- Some motivations for a hybrid approach to be discussed later

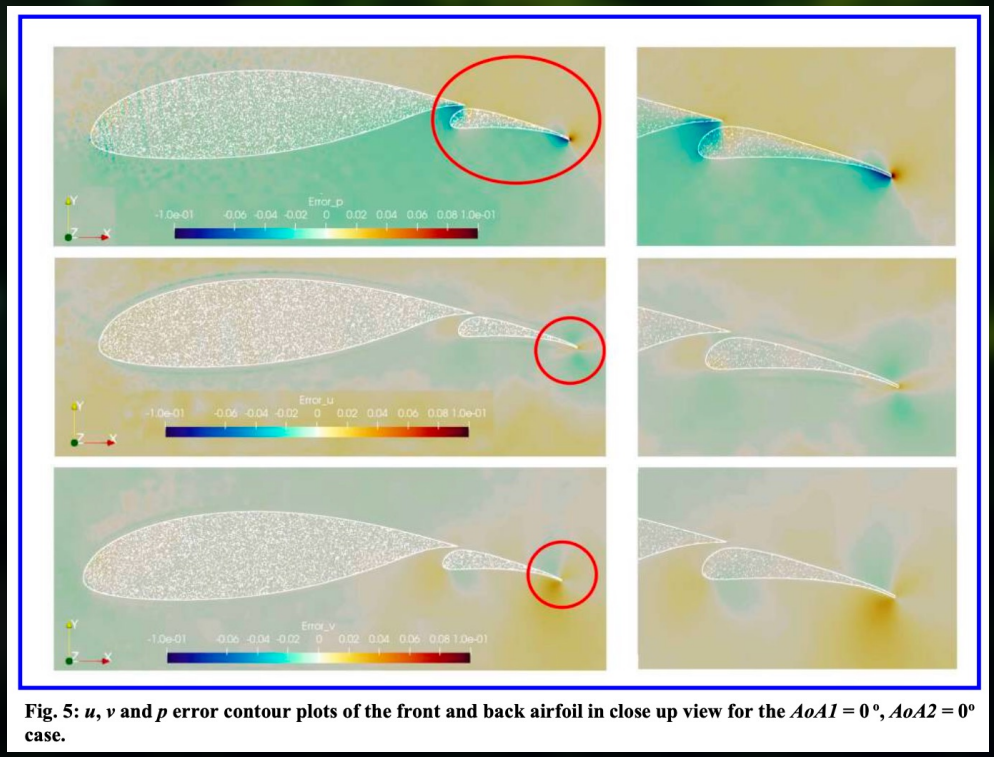


Fig. 5: *u*, *v* and *p* error contour plots of the front and back airfoil in close up view for the *AoA1* = 0 °, *AoA2* = 0° case.

NVAITC Projects (Collaboration with Temasek Lab@NUS)
PINNs based parameterized CFD model for two element airfoil control

- Integrate pressure over the surface to get lift and drag
- The force coefficient across the parameterized range has very good prediction results in comparison with OF references
- Good surrogate for optimization/control
- Speed up is more than 100X

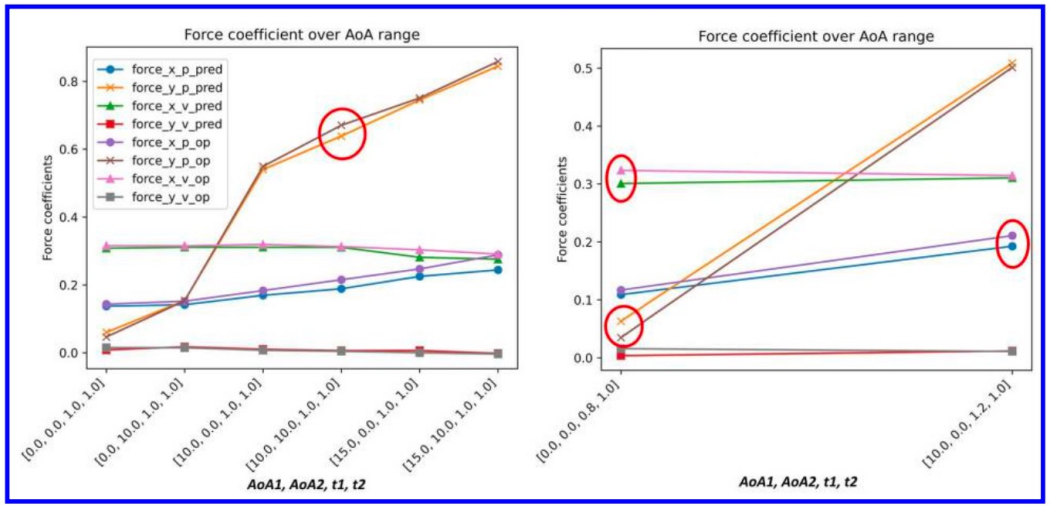


Fig. 13: Comparison between the true and predicted force coefficients for the 4 variables case.

NVAITC Projects – PINNs for Hydro-Morphodynamics collaboration with Coventry University

Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest

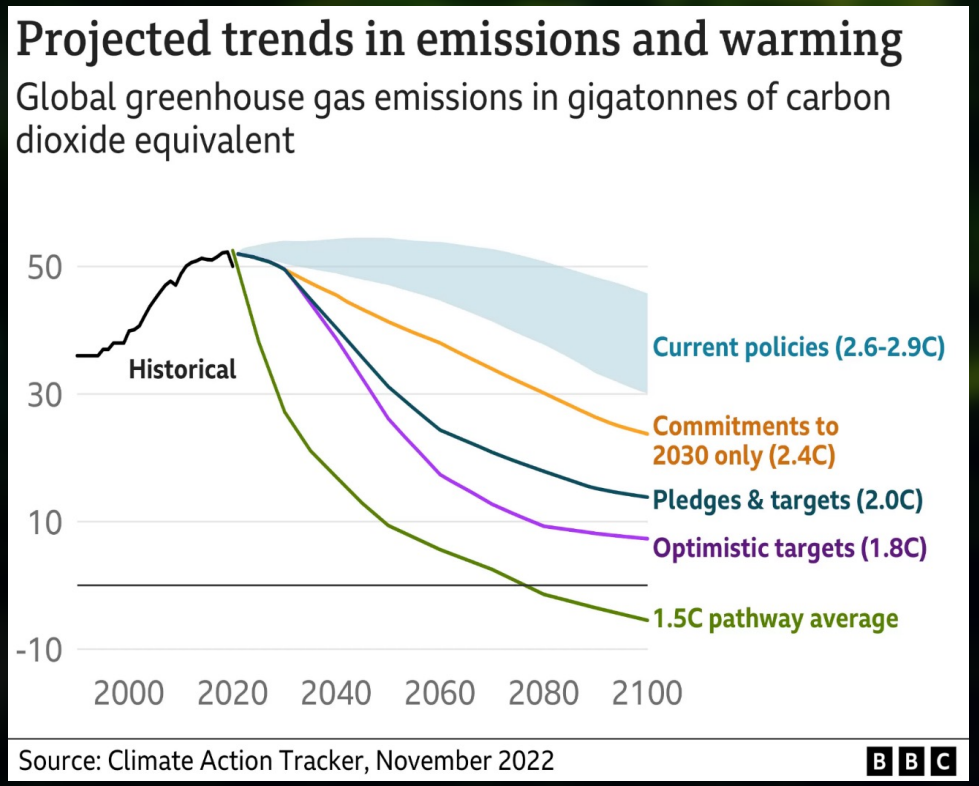
Majdi Fanous[a,*], Alireza Daneshkhah[a], Jonathan M. Eden[b], Juntao Yang[c], Simon See[a,d], Vasile Palade[a]

[a] *Centre for Computational Science & Mathematical Modelling, Coventry University, Priory Street, Coventry, CV1 5FB, United Kingdom*
[b] *Centre for Agroecology, Water and Resilience, Coventry University, Ryton Gardens, Wolston Lane, Warwickshire, CV8 3LG, United Kingdom*
[c] *NVIDIA AI Technology Center, Temasek Blvd, Cityhall District, 038988, Singapore*
[d] *NVIDIA AI Technology Center, San Tomas Expy, Santa Clara, 95051, United States of America*

***PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering**
*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

NVAITC Projects - PINNs for Hydro-Morphodynamics collaboration with Coventry University

- Climate mitigation and adaptation is crucially important for the upcoming decades
- IPCC6 report predicts 3 degrees warming at current projection, and 1.5 degree at the most optimistic case
- Sea rise and coastal corrosion is two major hazard from climate change
- Two solutions, 1. artificial barriers 2. ecosystem based adaptation



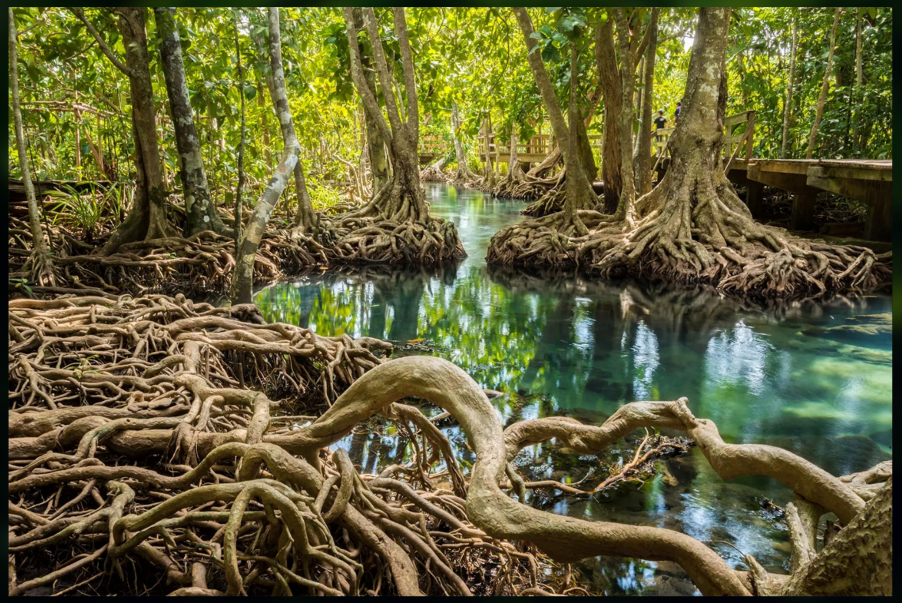
*PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering

*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

NVAITC Projects - PINNs for Hydro-Morphodynamics collaboration with Coventry University

Ecosystem-based adaptation

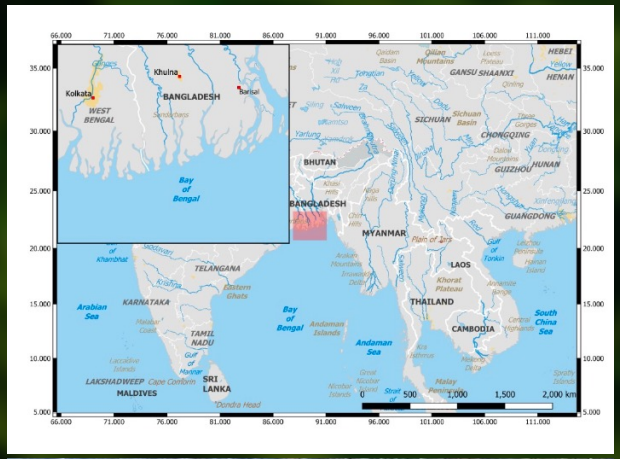
- Protects from sea-level rise
- Preserves natural habitats
- Cheap



*PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering
*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

NVAITC Projects – PINNs for Hydro-Morphodynamics collaboration with Coventry University

- We need good simulation models to predict the effects of grey or green
- The area of interest is bay of Bengal, Sundarbans mangrove forest
- One of the largest mangrove forest in the world



***PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering**

*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

NVAITC Projects – PINNs for Hydro-Morphodynamics collaboration with Coventry University

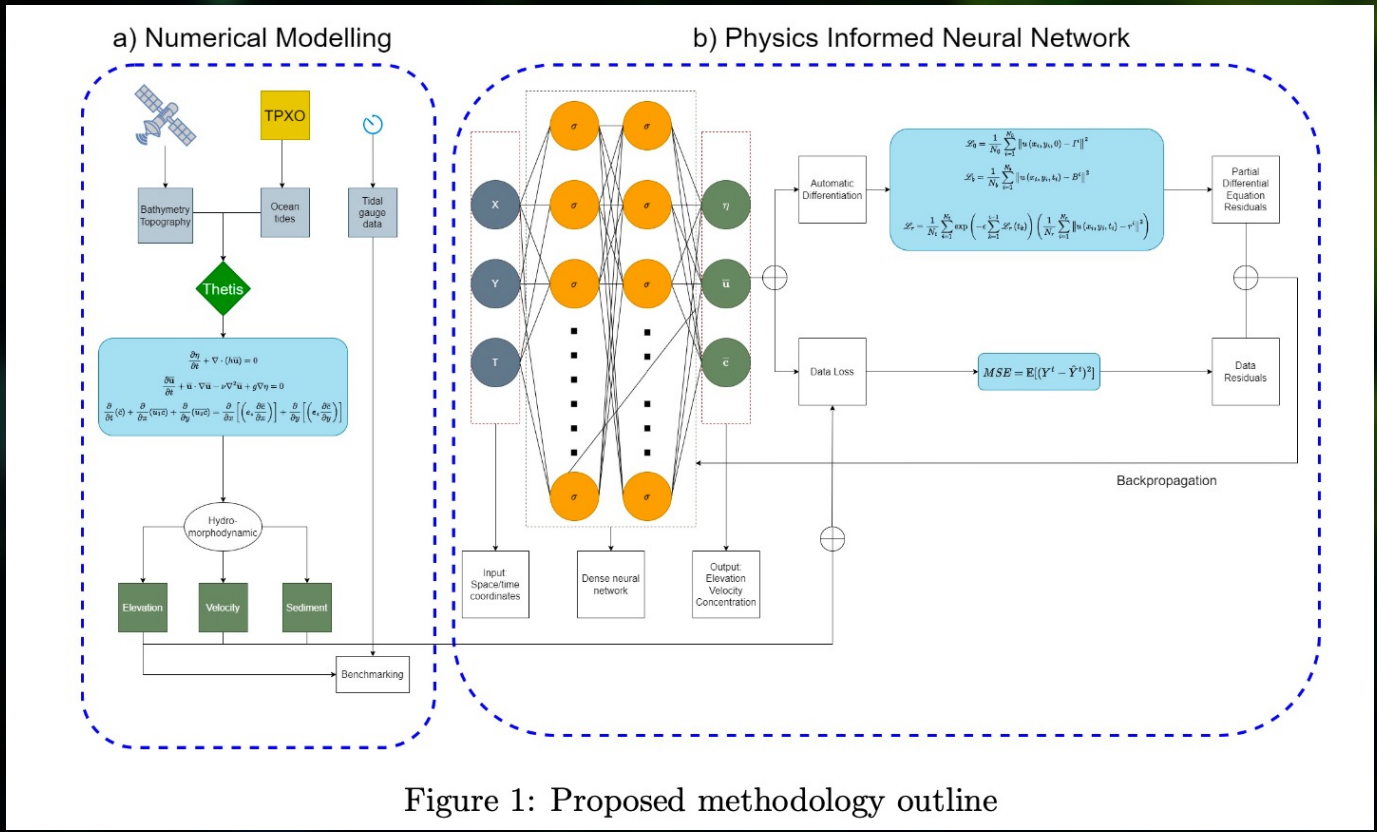


Figure 1: Proposed methodology outline

***PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering**

*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

NVAITC Projects - PINNs for Hydro-Morphodynamics collaboration with Coventry University

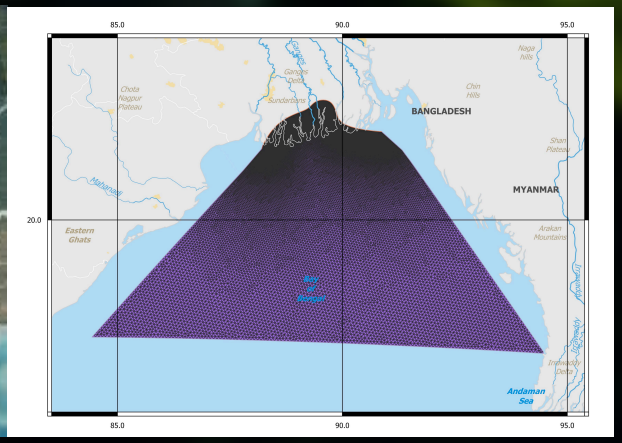
morphodynamics equations

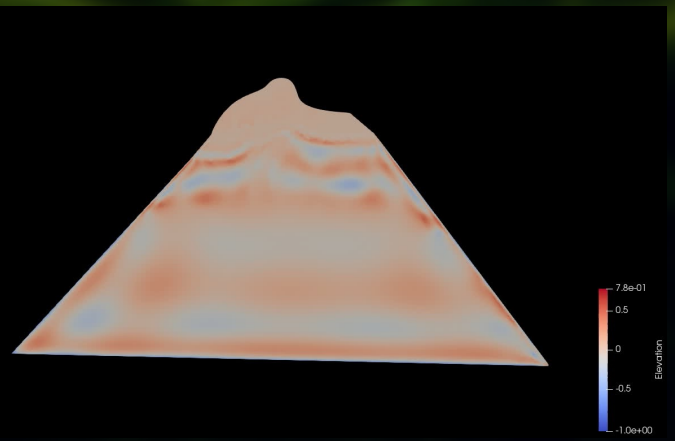
$$\frac{\partial \eta}{\partial t} + \nabla \cdot (h\overline{\mathbf{u}}) = 0,$$

$$\frac{\partial \overline{\mathbf{u}}}{\partial t} + \overline{\mathbf{u}} \cdot \nabla \overline{\mathbf{u}} - \nu \nabla^2 \overline{\mathbf{u}} + g \nabla \eta = 0,$$

$$\frac{\partial}{\partial t}(\bar{c}) + \frac{\partial}{\partial x}\left(\overline{u_1 c}\right) + \frac{\partial}{\partial y}\left(\overline{u_2 c}\right) = \frac{\partial}{\partial x}\left[\left(e_s \frac{\partial \bar{c}}{\partial x}\right)\right] + \frac{\partial}{\partial y}\left[\left(e_s \frac{\partial \bar{c}}{\partial y}\right)\right]$$

1. Solving the equation with satellite observations on mangrove
2. Causal loss with importance sampling used to deal with time-dependent problem
3. 100X speed up comparing to numerical model





***PHYSICS INFORMED NEURAL NETWORKS TO MODEL THE HYDRO-MORPHODYNAMICS OF MANGROVE ENVIRONMENTS, 5th International Conference on Uncertainty Quantification in Computational Sciences and Engineering**

*Leveraging Physics-Informed Neural Networks for Efficient Modelling of Coastal Ecosystems Dynamics: A Case Study of Sundarbans Mangrove Forest, under review , Journal of Expert Systems with Application

Neural Operator: AI as surrogate model

NVIDIA

Functions map data, Operators map functions

- Function: $\mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$

  e.g., image classification:  $\mapsto 5$

- Operator: function (∞-dim) $\mapsto$ function (∞-dim)
  e.g., derivative (local): $x(t) \mapsto x'(t)$
  e.g., integral (global): $x(t) \mapsto \int K(s,t)x(s)ds$

  e.g., dynamic system: $x(t)$ Input Signal → System T → Output Signal $y(t)$

  e.g., biological system
  e.g., social system

$\Rightarrow$ Can we learn operators via neural networks?
$\Rightarrow$ How?

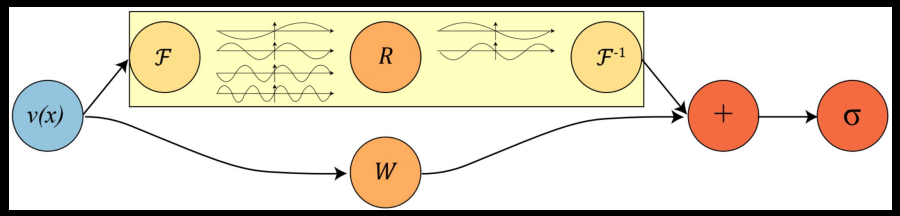
CRUNCH TIME
CRUNCH Group
Brown University

NVIDIA

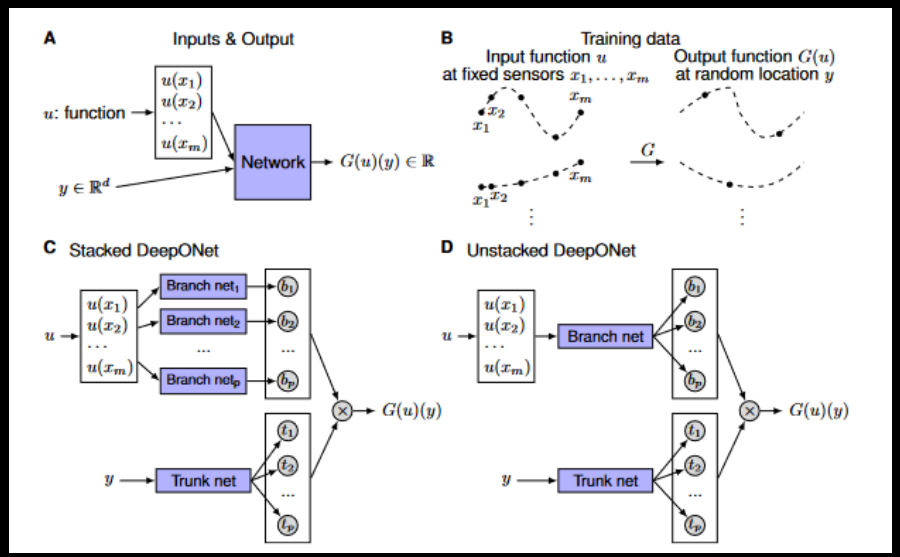
Neural Operator

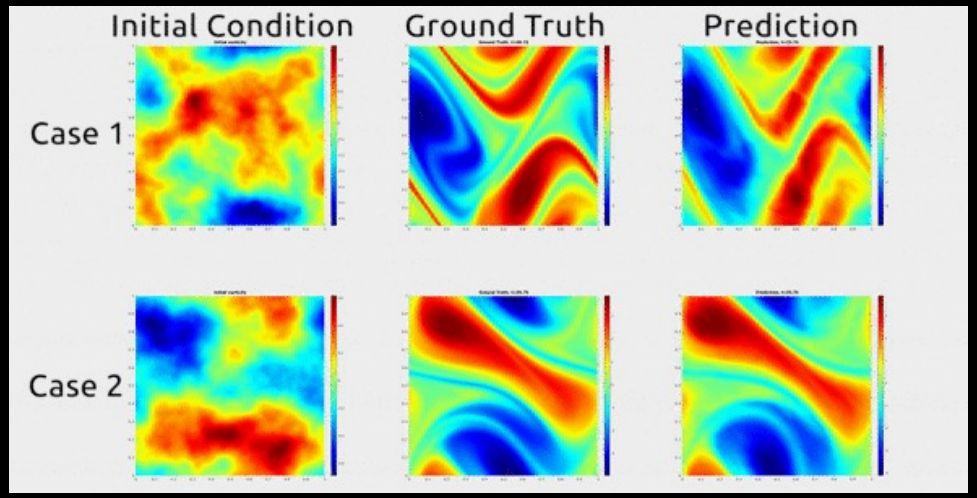
Learns mapping from functions to functions

Fourier Neural Operator



DeepONet





Universal Approximation Theorem for Operators

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

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

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

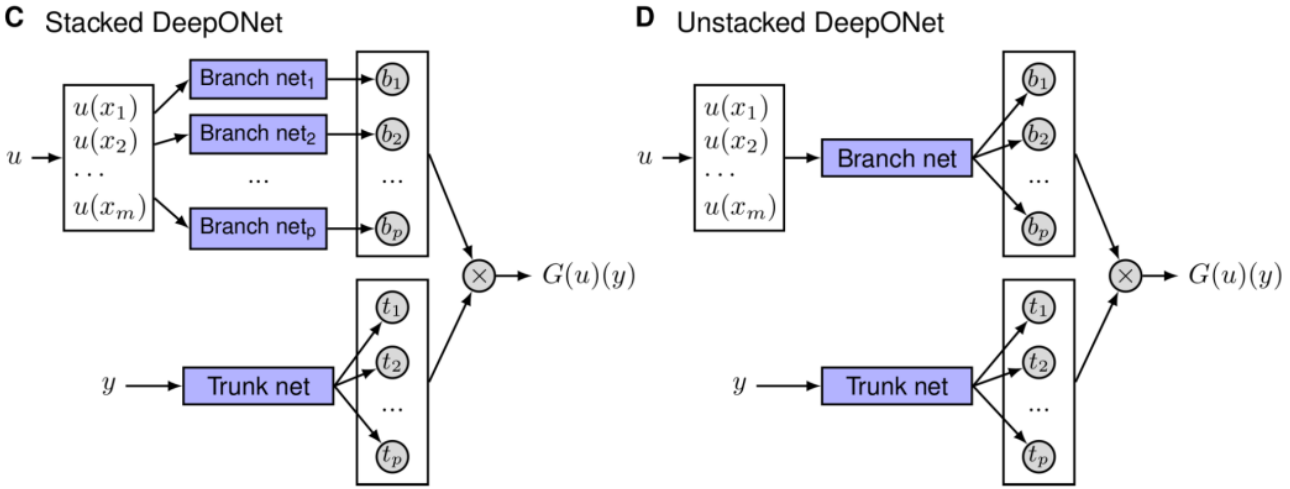
holds for all $u \in V$ and $y \in K_2$.

- T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks, 6(4):911–917, 1995.

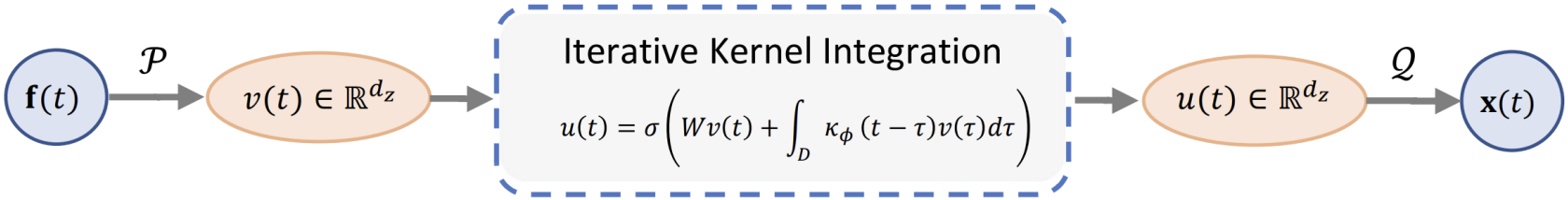
Deep Operator Network (DeepONet)

Recall the Theorem:

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



Basis-specific Neural Operators



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

[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 pdes.

Integral Kernel – Green's Functions

Considering the generic family of PDEs of the following form:

$$(\mathcal{L}_a u)(x) = f(x), \qquad x \in D \tag{1}$$
$$u(x) = 0, \qquad x \in \partial D$$

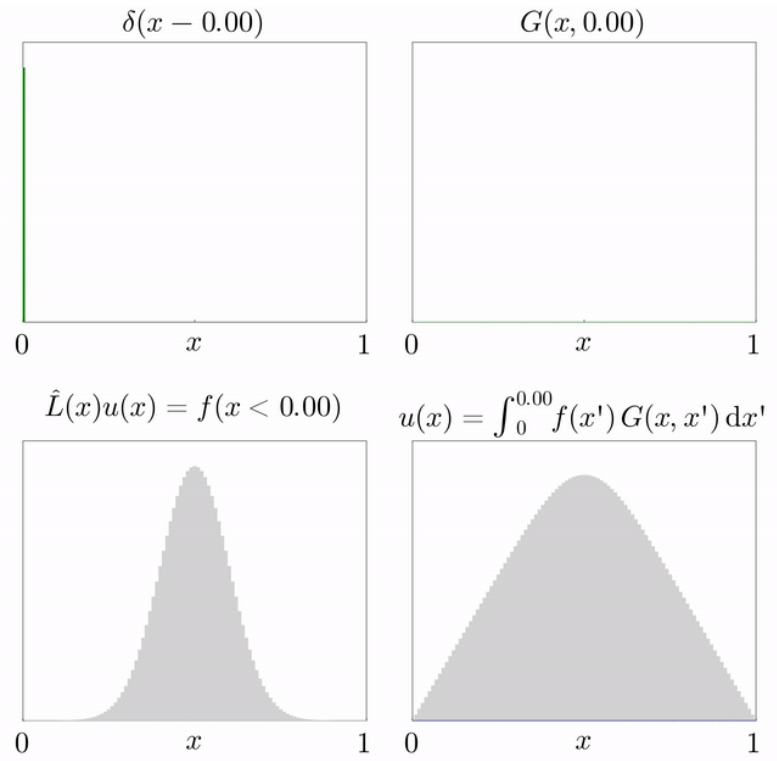
where $\mathcal{L}_a$ is a differential operator depending on a parameter a.

When the input is the delta measure centered at x, namely, $f(x) = \delta_x$, the unique solution of Eq. (1) is defined as the Green's function G_a, that is,

$$\mathcal{L}_a G(x, \cdot) = \delta_x \tag{2}$$

Since the source term $f(x)$ in Eq. (1) is a sum of delta functions, the solution of Eq. (1) can be represented as Eq. (3) by applying the superposition principle:

$$u(x) = \int_D G_a(x, y) f(y)\, dy \tag{3}$$



Building the solution for a general source based on Green's function [1]

[1] https://en.wikipedia.org/wiki/Green%27s_function

Neural Operators

$$u(x) = \int_D G_a(x, y) f(y) dy \tag{3}$$

Guided by Eq. (3), the following iterative architecture is proposed [2]:

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \int_D \kappa_\phi(x - y) v_t(y) dy\right) \tag{4}$$

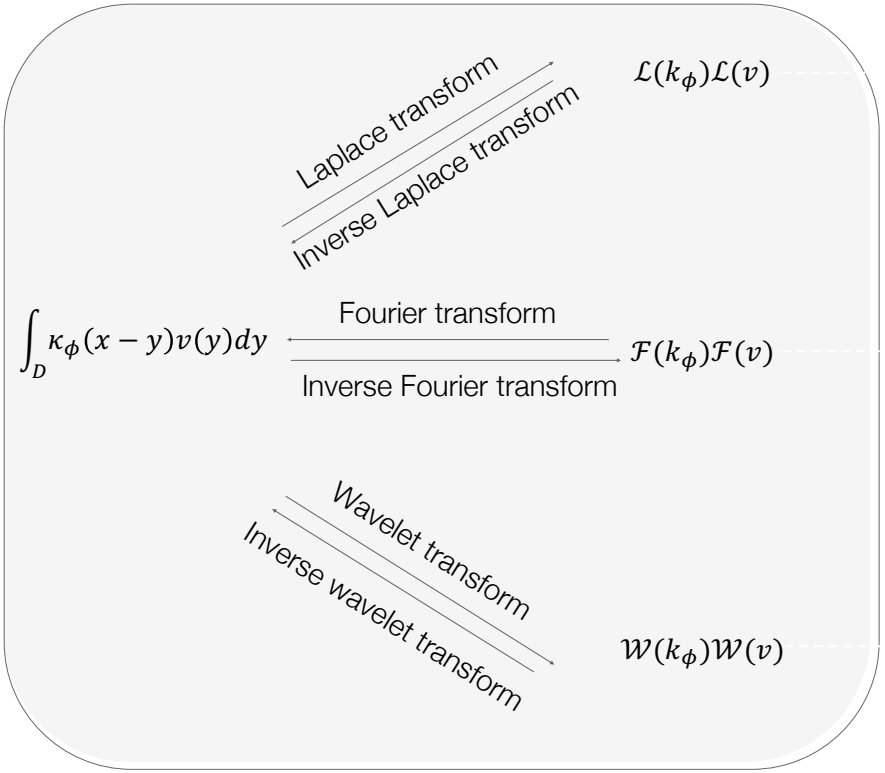
where σ is the activation function, the matrix W and the kernel κ_ϕ are modeled as neural networks, which are learned from data.

- Monte Carlo approximation
- Convolution theorem

- Expensive cost
- Multiplication in the other domain

[2] Li, Z. et al. (2020). Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*.

Convolution Theorem



- Laplace Neural Operator [3]

$$v_{t+1}(x) = \sigma\big(W v_t(x) + \mathcal{L}^{-1}(\mathcal{L}(k_\phi)\mathcal{L}(v))\big)$$

- Fourier Neural Operator [4]

$$v_{t+1}(x) = \sigma\big(W v_t(x) + \mathcal{F}^{-1}(\mathcal{F}(k_\phi)\mathcal{F}(v))\big)$$

- Wavelet Neural Operator [5]

$$v_{t+1}(x) = \sigma\big(W v_t(x) + \mathcal{W}^{-1}(\mathcal{W}(k_\phi)\mathcal{W}(v))\big)$$

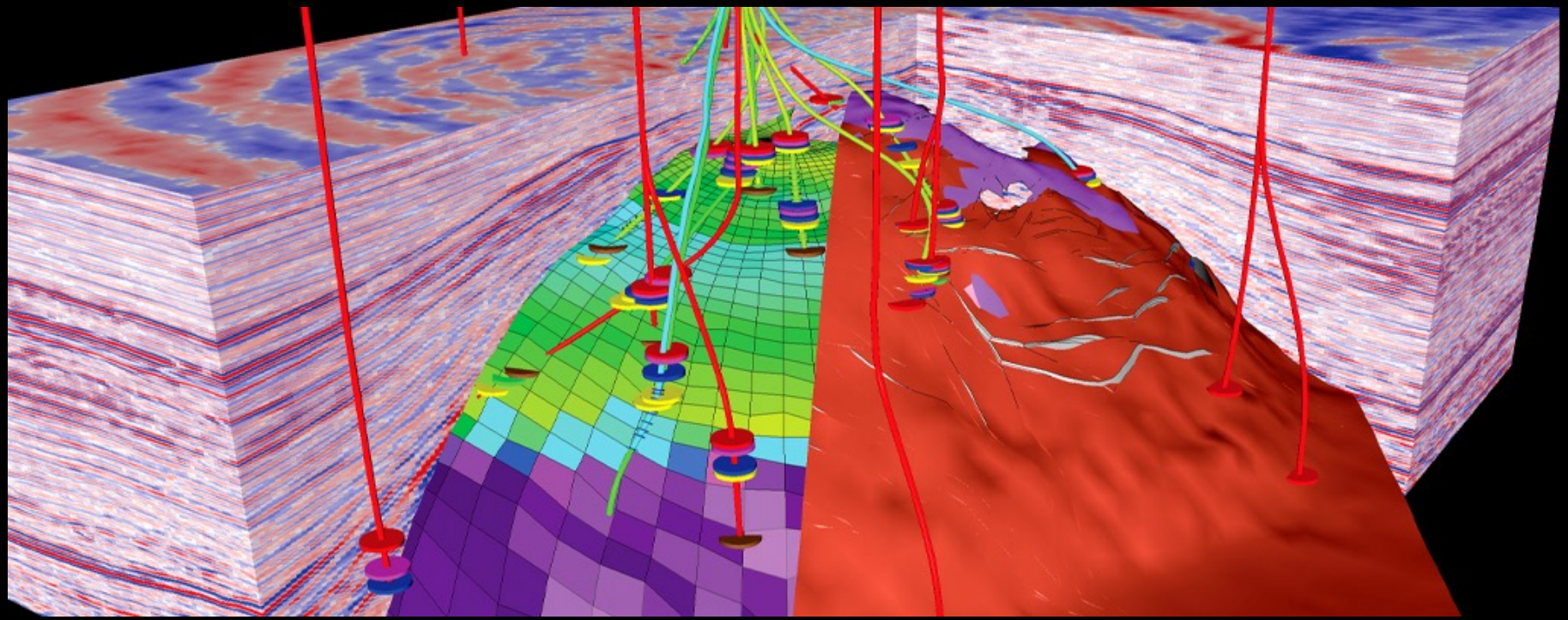
[3] Cao, Q., Goswami, S., & Karniadakis, G. E. (2023). LNO: Laplace neural operator for solving differential equations.

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

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

Neural Operator

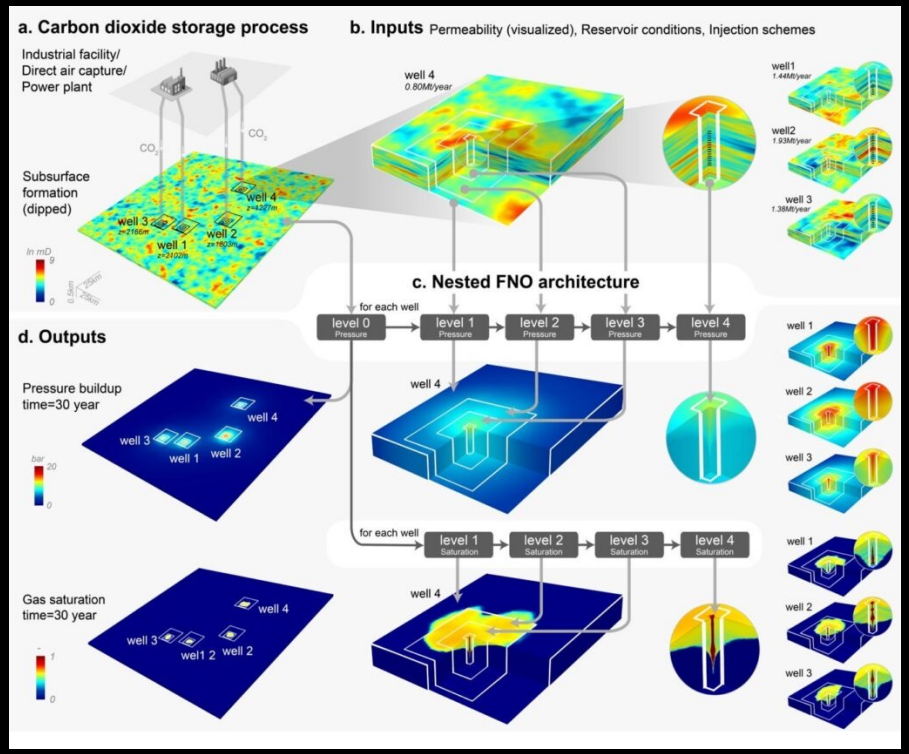
Subsurface modeling

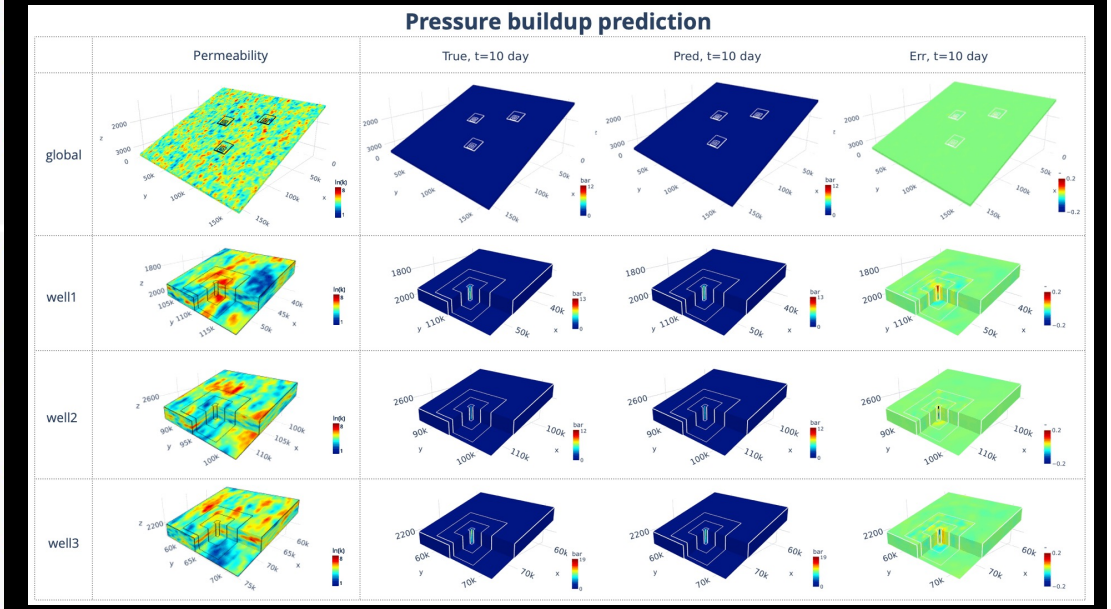


NVIDIA

Accelerating Carbon Capture and Storage Modeling Using Fourier Neural Operator

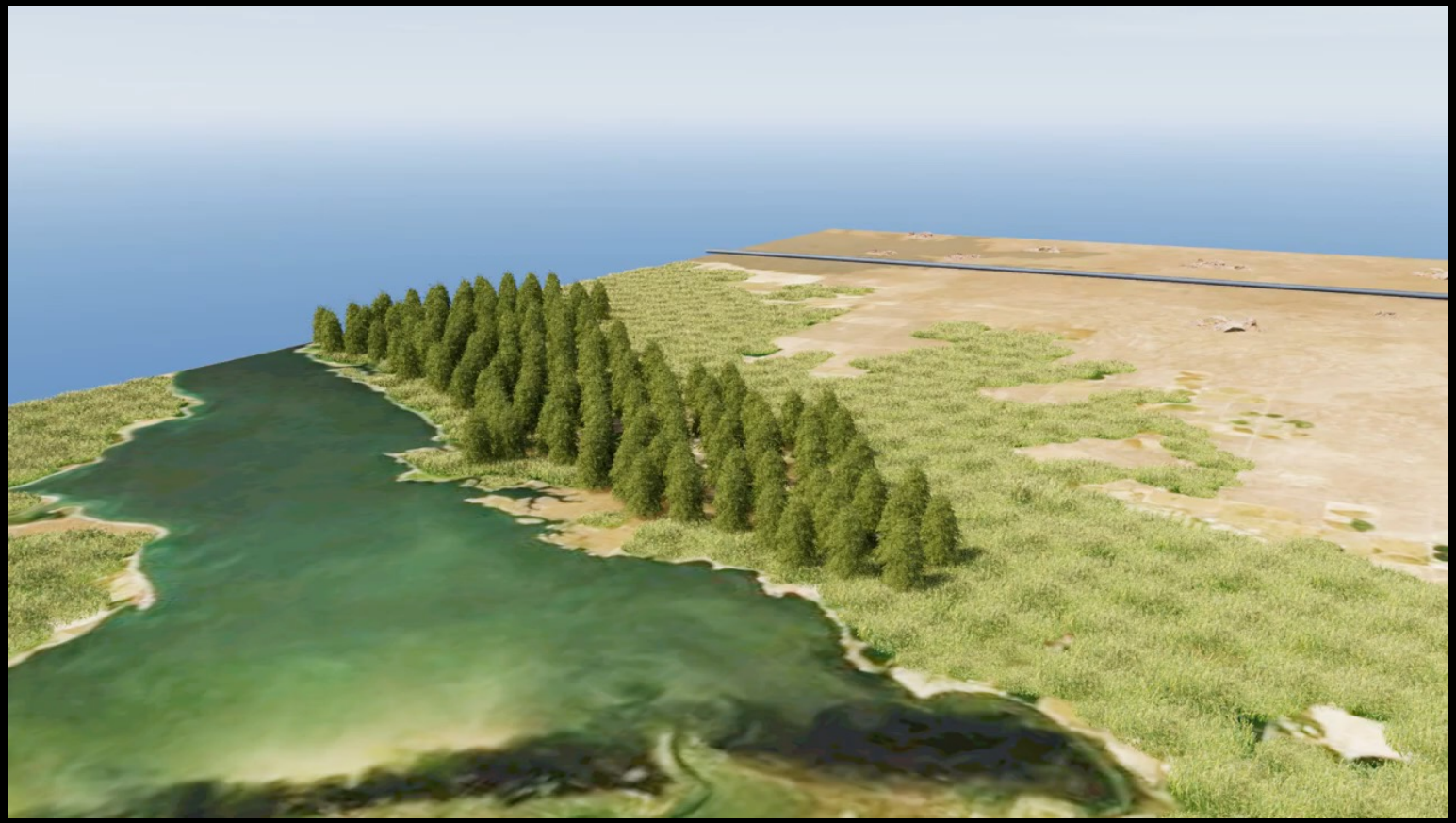
Nvidia and Stanford





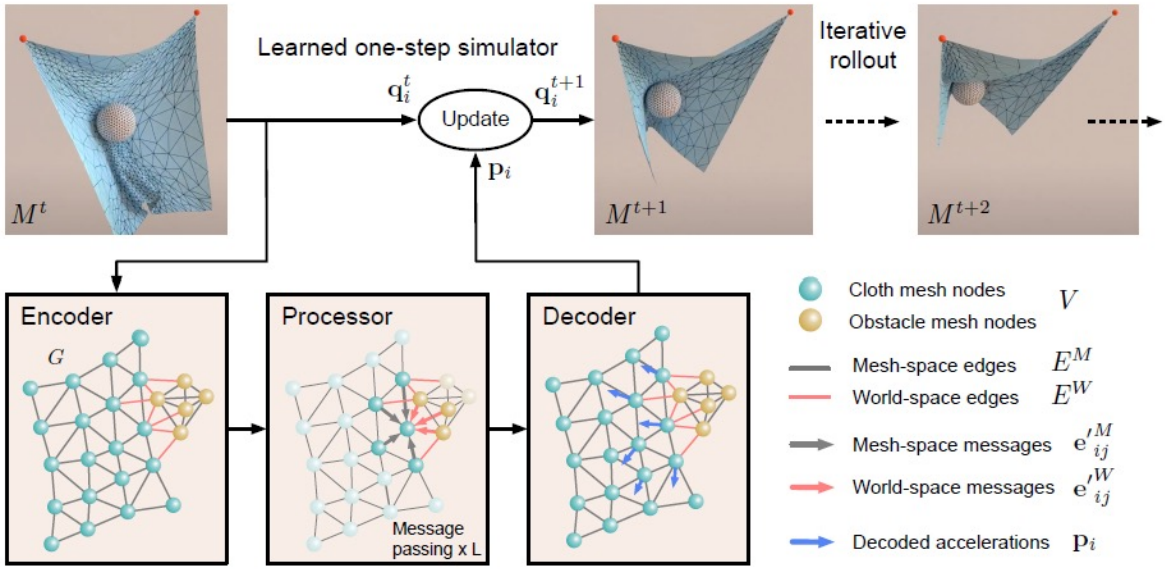
Physics Informed Neural Operator

FNO based surrogate model with Omniverse Visualization



https://developer.nvidia.com/blog/using-carbon-capture-and-storage-digital-twins-for-net-zero-strategies/

Leveraging the mesh



Mesh-based simulations are central to modeling in many disciplines across science and engineering.

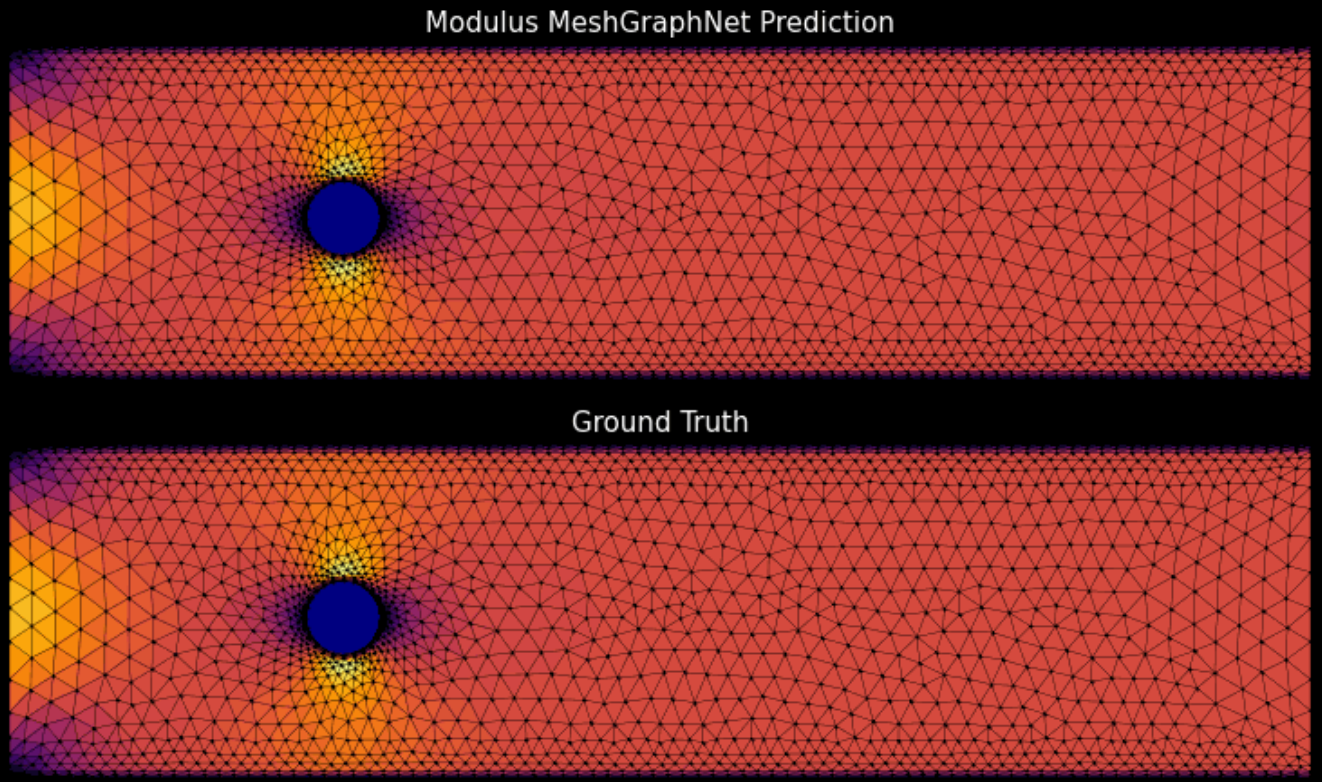
Mesh representations support powerful numerical integration methods

GNNs are a natural choice for data-driven approaches – its mesh-based and can cope with geometry irregularities and multi-scale physics.

Forward model – Predict variables of the mesh at time t+1 given current mesh at t and history of previous meshes.

MeshGraphNet

Develop Physics-Informed Machine Learning Models with Graph Neural Networks



https://developer.nvidia.com/blog/develop-physics-informed-machine-learning-models-with-graph-neural-networks/

Two level hybrid MCMC

From the best of both world, AI and numerical

- AI method is great in terms of speed up
- But many challenges still hinders the application of AI models
- Explainability, and no physical conservation are widely criticized by domain scientist
- Hybrid approach has been the transition to adopt current state of art AI models for real use cases
- There are two approaches in my opinion, deterministic approach and a statistical approach

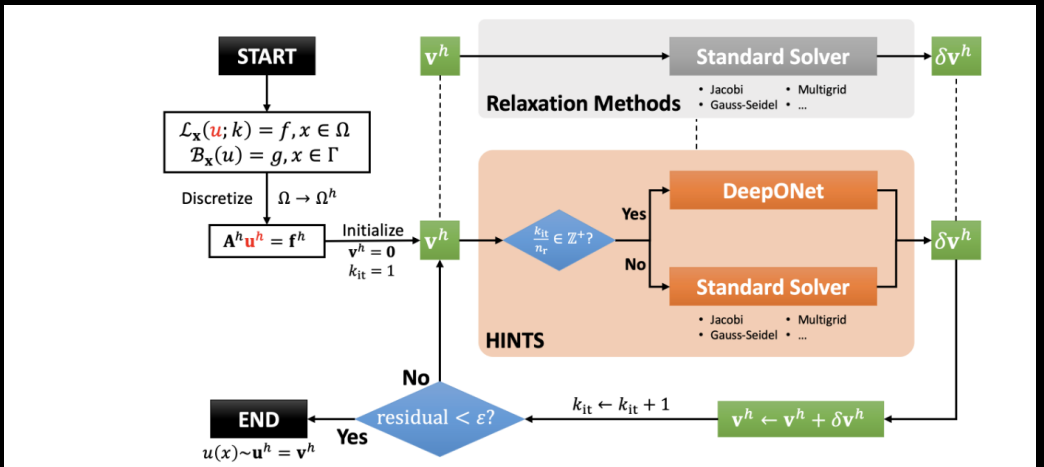


Figure 1: **Overview of the Hybrid Iterative Numerical Transferable Solver (HINTS).**

Two level hybrid MCMC

From the best of both world, AI and numerical

- AI surrogate model is the solution to large scale problems that was intractable for numerical models
- It has great potential in data inversion and data assimilation problems which requires repeated solution of the same forward problem
- But the limitations in AI surrogate models prevented its adoption in such problems
- Trust issues with AI surrogate model

Two level hybrid MCMC

From the best of both world, AI and numerical

forward map

We denote the probability space $U = [-1,1]^{\mathbb{N}}$, and sigma algebra on the domain U by $\Theta = \bigotimes_{j=1}^{n} \mathcal{B}([-1,1])$. we define the forward observation map $\mathcal{G} : U \to \mathbb{R}^k$ for all $z \in U$ as

$$\mathcal{G}(z) = (\mathcal{O}_1(u), \mathcal{O}_k(u), ..., \mathcal{O}_k(u)), \tag{1}$$

where $u \in V$ is the forward solution of the governing PDE equation and $\mathcal{O}_i \in V', i = 1, 2, ...k$ are k continous bounded linear functionals on V.

example

$$K(z) = \bar{K} + \sum_{j=1}^{n} z_j \psi_j.$$

$$\nabla \cdot K \nabla u = f$$

Two level hybrid MCMC

From the best of both world, AI and numerical

Noisy observations

Let ϑ be the observation noise. It is assumed Gaussian and independent of the parameters z. Thus the random variable ϑ has values in $\mathbb{R}^k$ and following normal distribution $N(0, \Sigma)$, where Σ is a known $k \times k$ symmetric positive covariance matrix. The noisy observation δ is

$$\delta = \mathcal{G} + \vartheta. \tag{2}$$

We denote the posterior distribution as γ^δ and the prior distribution as γ.

Two level hybrid MCMC

From the best of both world, AI and numerical

Assumption

Consider the numerical discretization approximation of the forward problem, there is a constant $C > 0$ such that for every $l \in \mathbb{N}$ and for every $z \in U$, the following error bound holds

$$\|u - u^l\|_V \leq C2^{-l}, \tag{3}$$

where l is the level of discretization and the mesh size $h = 2^{-l}$.

Assumption

We denote $\tilde{\mathcal{G}} : U \to \mathbb{R}^k$ as the map defined by a trained neural network. Given a neural network trained by data generated by a numerical model with mesh size $h = 2^{-L}$, we have

$$\|\tilde{G}(z) - u\|_V \leq 2^{-L+\epsilon},$$

where z is the input and we expect ϵ to be a small value for a well defined and trained neural network.

Two level hybrid MCMC

From the best of both world, AI and numerical

Plain MCMC with Numerical Model

To achieve 2^{-L} accuracy with a numerical model with mesh size $h = 2^{-L}$, we need to run the solver for $O(2^{2L})$ times.

Plain MCMC with AI surrogate model

To run the plain MCMC with AI surrogate model, with $O(2^{2L-2\epsilon})$, we can only achieve $2^{-L+\epsilon}$ accuracy. Even with longer chain, the accuracy will no improve.

Two level hybrid MCMC

From the best of both world, AI and numerical

Naive solutions

Generate more training data with finer resolution numerical model (e.g. 2^{-L-1}) to train the AI surrogate model with accuracy higher than 2^{-L}.

- Empirical results show that AI model accuracy increases with exponential growth of data. (that is $2^r, r > 1$ more data) [1]
- Numerical computation to generate each sample will be at least 4 times or more for 2D problems and 8 times or more for 3D problems
- Limitations on how good a given architecture can be trained. [2]

---

[1] Deep learning scaling in predictable empirically

[2] Residual-Based Error Correction for Neural Operator Accelerated Infinite-Dimensional Bayesian Inverse Problems

Two level hybrid MCMC

From the best of both world, AI and numerical

Plain MCMC with Numerical Model

To achieve 2^{-L} accuracy with a numerical model with mesh size $h = 2^{-L}$, we need to run the solver for $O(2^{2L})$ times.

Plain MCMC with AI surrogate model

To run the plain MCMC with AI surrogate model, with $O(2^{2L-2\epsilon})$, we can only achieve $2^{-L+\epsilon}$ accuracy. Even with longer chain, the accuracy will no improve.

Two level MCMC approach

With $M_{\mathrm{ML}} = C2^{2L}$ and $M_{\mathrm{num}} = C(1+2^{\epsilon})^2$, we get accuracy of $O(2^{-L})$.

Two level hybrid MCMC

From the best of both world, AI and numerical

We denote the Quantity of Interest as Q, the machine learning approximate posterior distribution γ^{ML} and the numerical approximate posterior distribution γ^{num}. With the target accuracy of $\mathcal{O}(2^{-L})$, γ^{ML} and γ^{num} is equivalent to $\gamma^{L,\delta}$ and $\gamma^{L-\epsilon,\delta}$. With the two level approach, we can rewrite the numerical approximation of the expected Quantity of Interest as

$$\begin{aligned}\mathbb{E}^{\gamma^{\mathrm{num}}}[Q] &= \mathbb{E}^{\gamma^{\mathrm{num}}}[Q] - \mathbb{E}^{\gamma^{\mathrm{ML}}}[Q] + \mathbb{E}^{\gamma^{ML}}[Q] \\ &= \left(\mathbb{E}^{\gamma^{\mathrm{num}}} - \mathbb{E}^{\gamma^{\mathrm{ML}}}\right)[Q] + \mathbb{E}^{\gamma^{ML}}[Q]. \qquad (4)\end{aligned}$$

Two level hybrid MCMC

From the best of both world, AI and numerical

Theorem

With $M_{\mathrm{ML}} = C2^{2L}$ *and* $M_{\mathrm{num}} = C(1+2^{\epsilon})^2$, *we have the following estimator error for the hybrid two level MCMC method,*

$$\mathbf{E}[|\mathbb{E}^{\gamma^{\delta}}[Q] - \mathbf{E}^{\mathrm{hybrid}}[Q]|] \leq C(\delta)2^{-L}. \tag{7}$$

Two level hybrid MCMC

From the best of both world, AI and numerical

Plain MCMC with Numerical Model

To achieve 2^{-L} accuracy with a numerical model with mesh size $h = 2^{-L}$, we need to run the solver for $O(2^{2L})$ times.

Plain MCMC with AI surrogate model

To run the plain MCMC with AI surrogate model, with $O(2^{2L-2\epsilon})$, we can only achieve $2^{-L+\epsilon}$ accuracy. Even with longer chain, the accuracy will no improve.

Two level MCMC approach

With $M_{\mathrm{ML}} = C2^{2L}$ and $M_{\mathrm{num}} = C(1+2^{\epsilon})^2$, we get accuracy of $O(2^{-L})$.

Two level hybrid MCMC

From the best of both world, AI and numerical

Elliptic Equation

$$\begin{cases} \nabla \cdot (K(x)\nabla u(x)) = \cos(2\pi x_1)\sin(2\pi x_2), \\ u(x_1 = 0) = 0, u(x_1 = 1) = 1, \\ \frac{\partial u}{\partial x_2}(x_2 = 0) = \frac{\partial u}{\partial x_2}(x_2 = 1) = 0. \end{cases}$$

Gaussian Prior

$$\mathcal{A}m = \begin{cases} \gamma\nabla \cdot (\Theta\nabla m) + \delta m & \text{in } \Omega \\ (\Theta\nabla m) \cdot \boldsymbol{n} + \beta m & \text{on } \partial\Omega \end{cases}$$

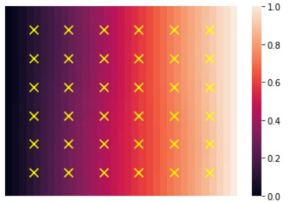
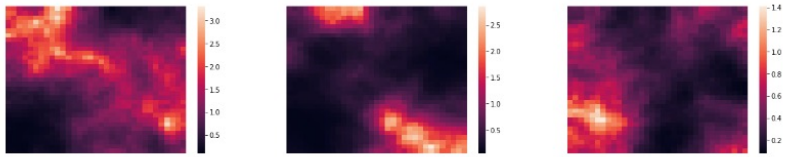
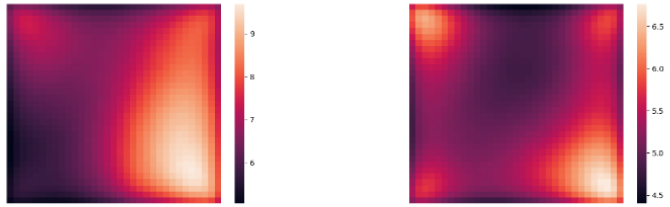


Figure 1: Eulerian observations of solution u

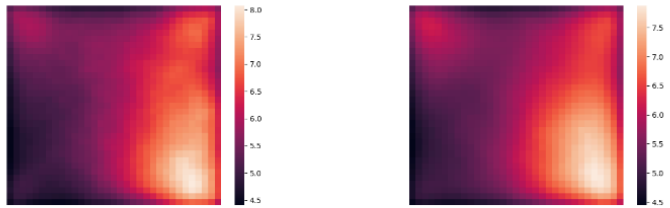


(a) sample 1 (b) sample 2 (c) sample 3

Figure 2: Samples from bi-laplacian random field



(a) numerical MCMC(100,000) (b) ML MCMC (100,000)



(c) Hybrid (100,000+1000) (d) Hybrid (100,000+4000)

Figure 3: Expected mean of $K(x)$ from 5 runs of MCMC