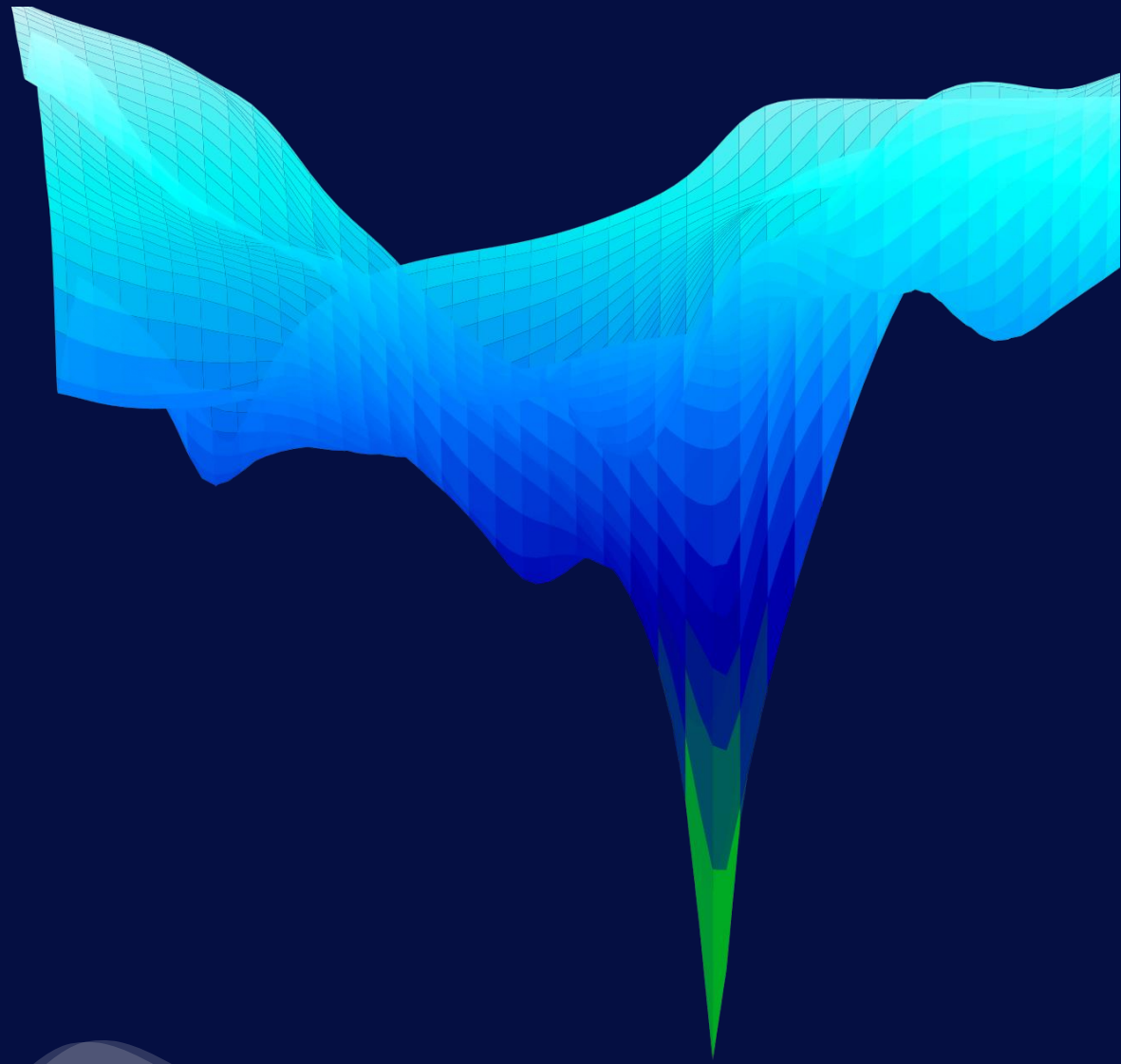
Scientific Machine Learning (SciML) for Solving PDEs

(Case Studies: two-phase flow in porous media)



University of Campinas (Nov. 2024)

A part of INTPART project, supported by NCS2030 - National Centre for Sustainable Subsurface Utilization of the Norwegian Continental Shelf.

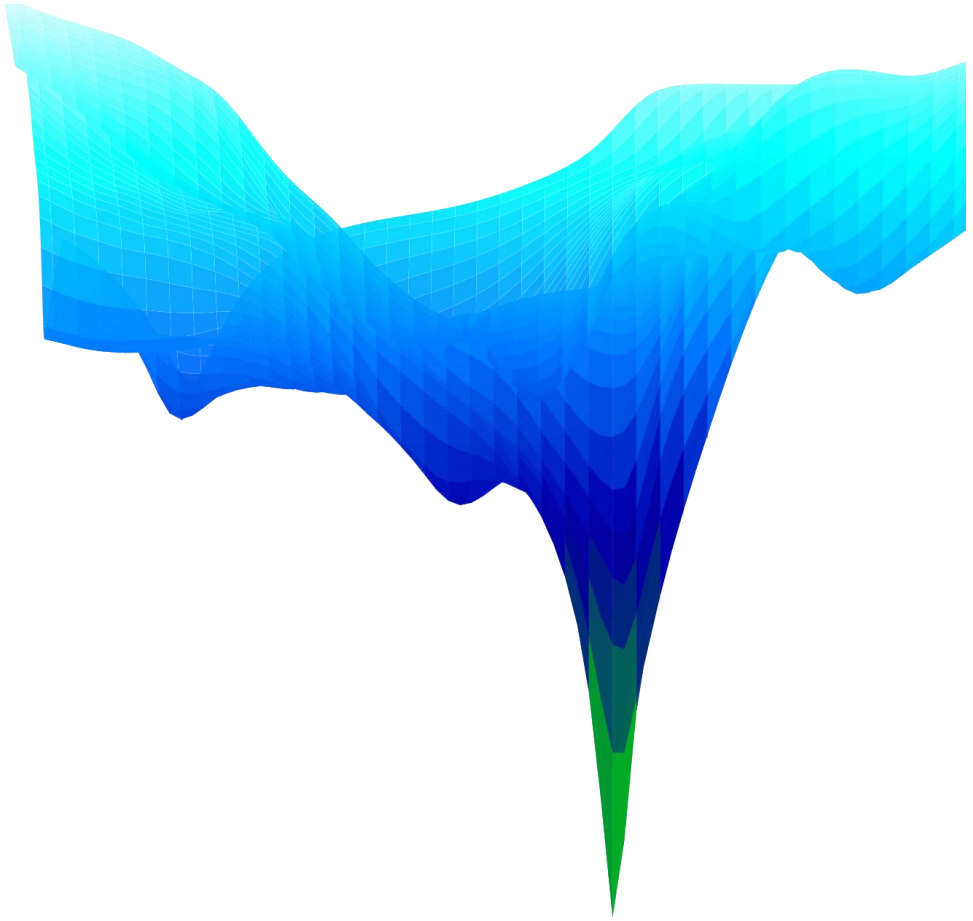
IFE NORCE UNIVERSITAS BERGENSIS

NCS 2030 National Centre for Sustainable Subsurface Utilization of the Norwegian Continental Shelf

University of Stavanger

University of Stavanger

Contents



Day 1

Part 1: Scientific Machine Learning (SciML): Introduction and Recent Advances (1 hour)

Part 2: Physics-Informed Neural Networks (PINNs) (1.5 hours)

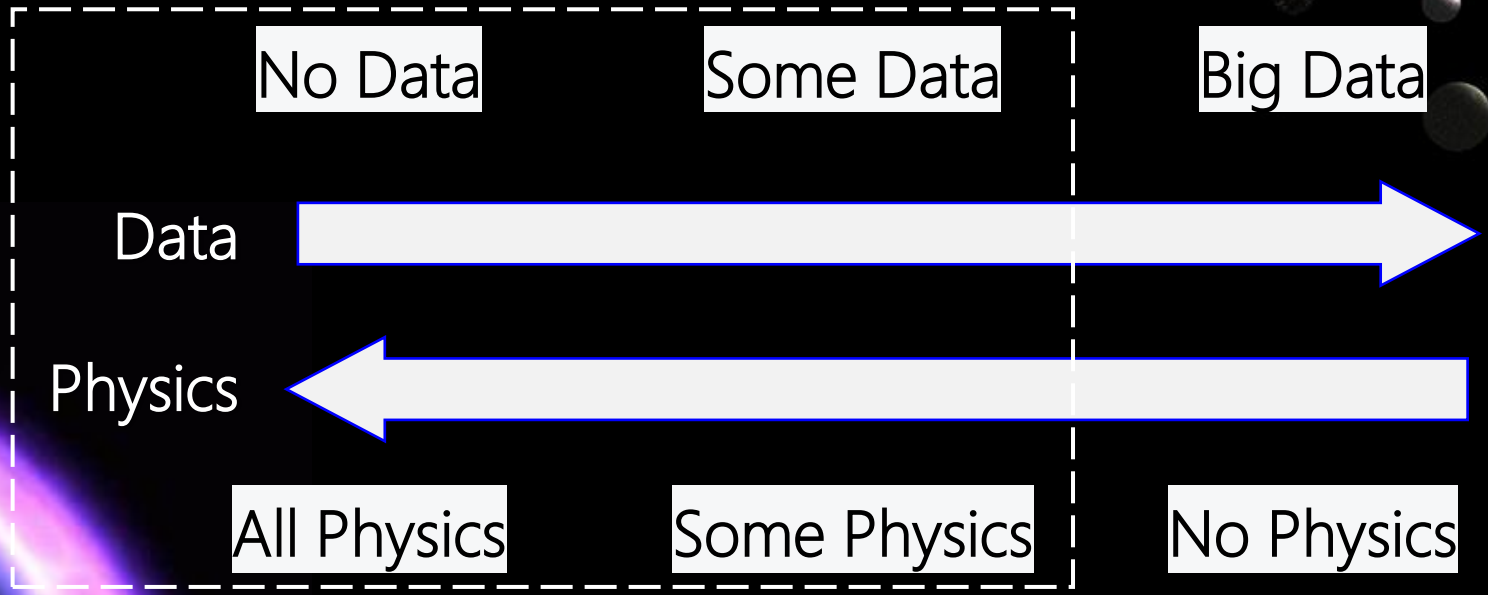
Day 2

Part 3: Hands-on Experience with PINNs (2.5 hours)

Data Driven Approaches

Physics-informed AI

Physics-informed Neural Networks (PINNs)



Karniadakis et al., 2021

Physics Informed Neural Networks

First Introduced by Raissi et al. (2019)

Reliable for mixed data types

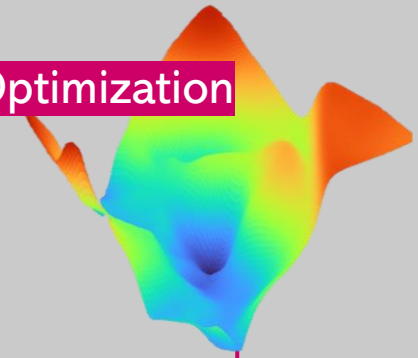
Flexible to noisy/incomplete data

Quicker history matching

More reliable sensitivity analysis

Live interpretation with low supervision

Optimization



Machine Learning

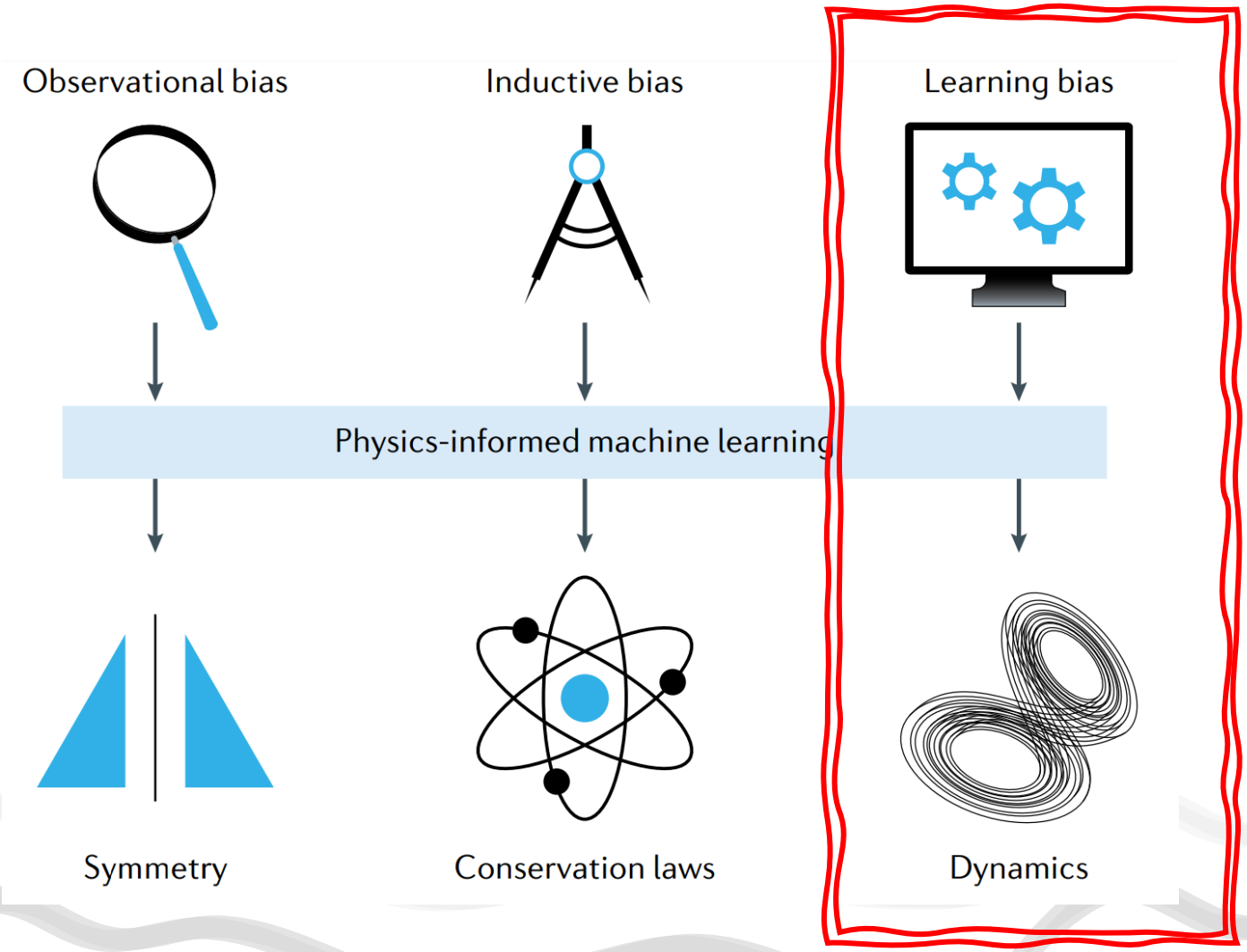
Physics

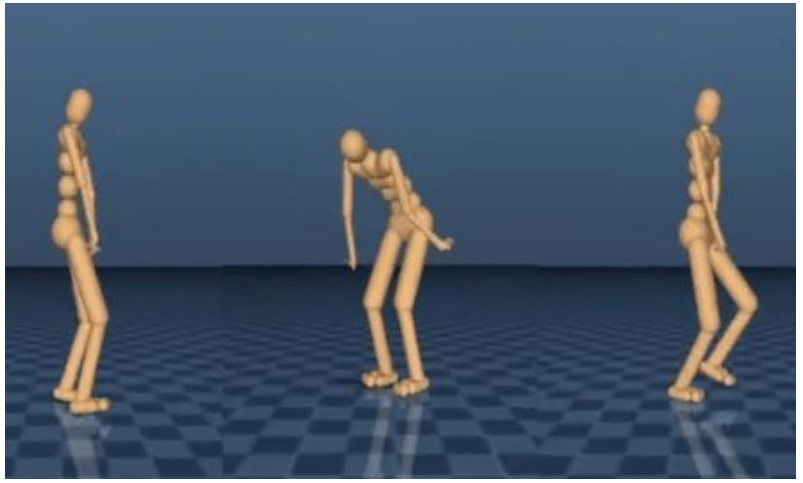
Numerical Methods

- Forward Simulation
- History Matching
- Uncertainty quantification
- Data-driven modeling

NCS 2030

Physics Informed Machine Learning

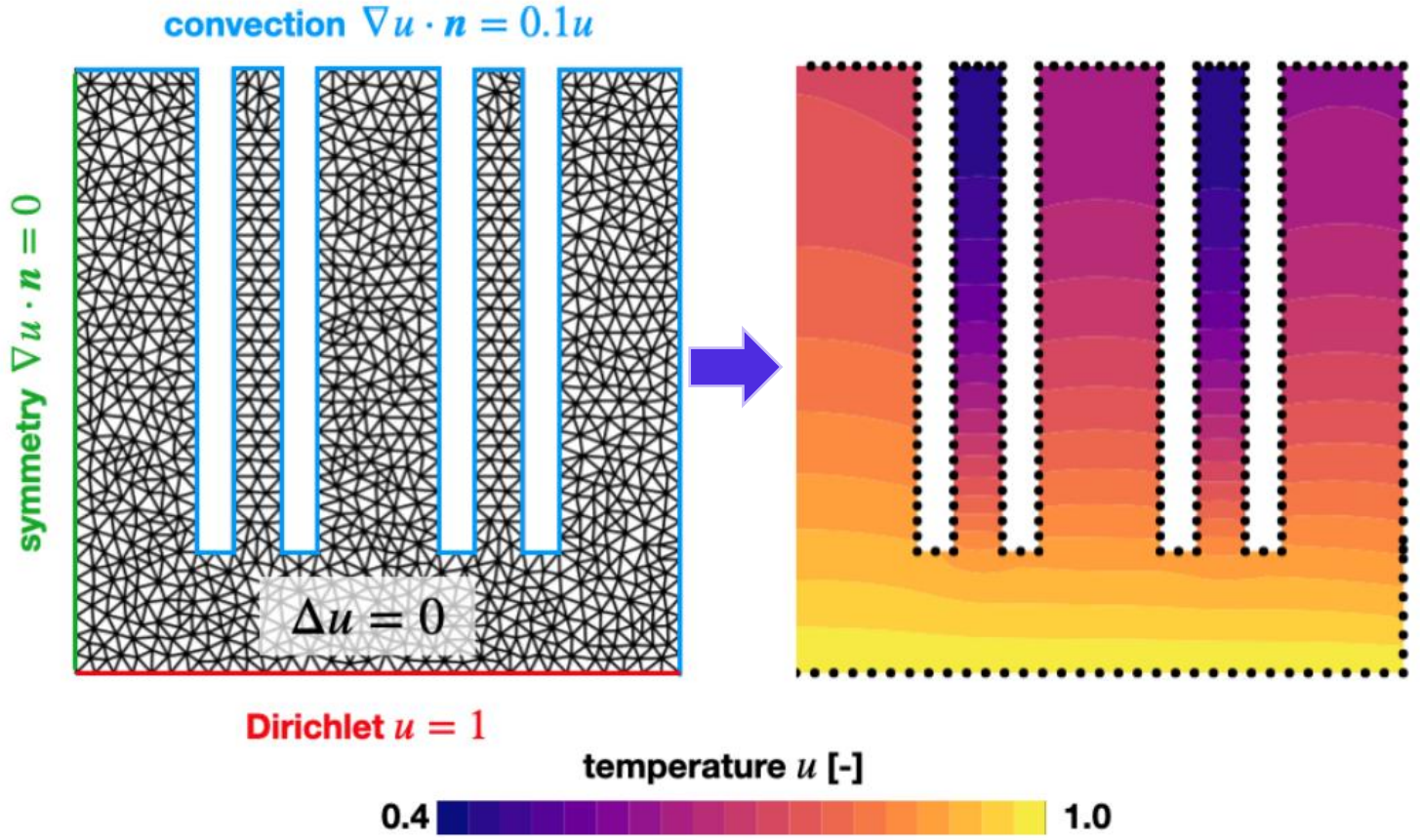




Karniadakis et al., 2021

Example: A Forward Problem

- Given boundary conditions and the heat equation, find the temperature inside the heat sink.



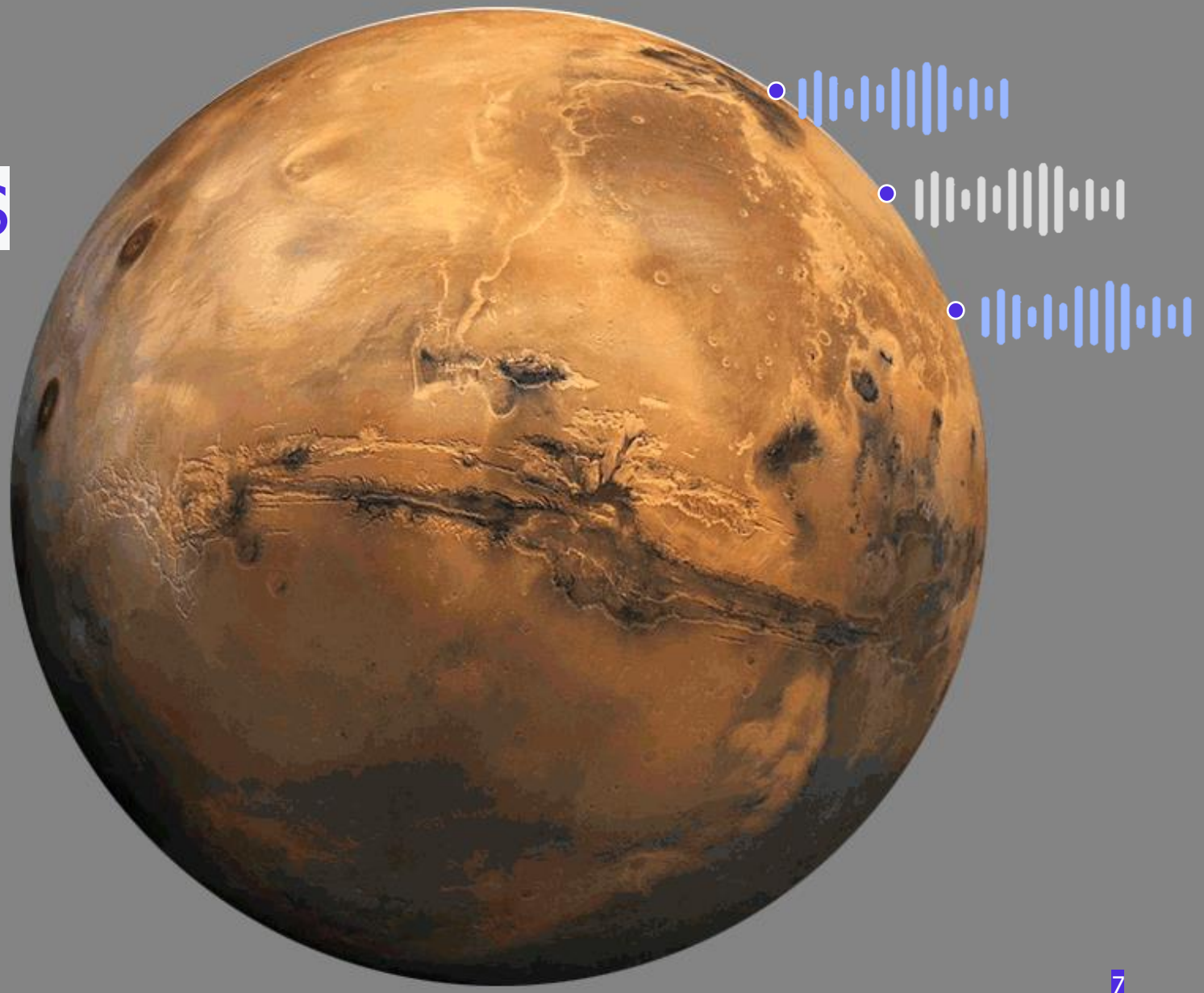
Inverse Problems

KEY CHARACTERISTICS

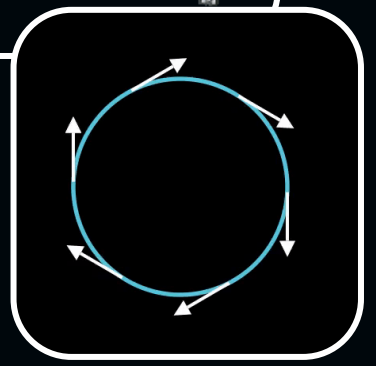
Ill-Posedness
Nonlinearity
Noisy Data

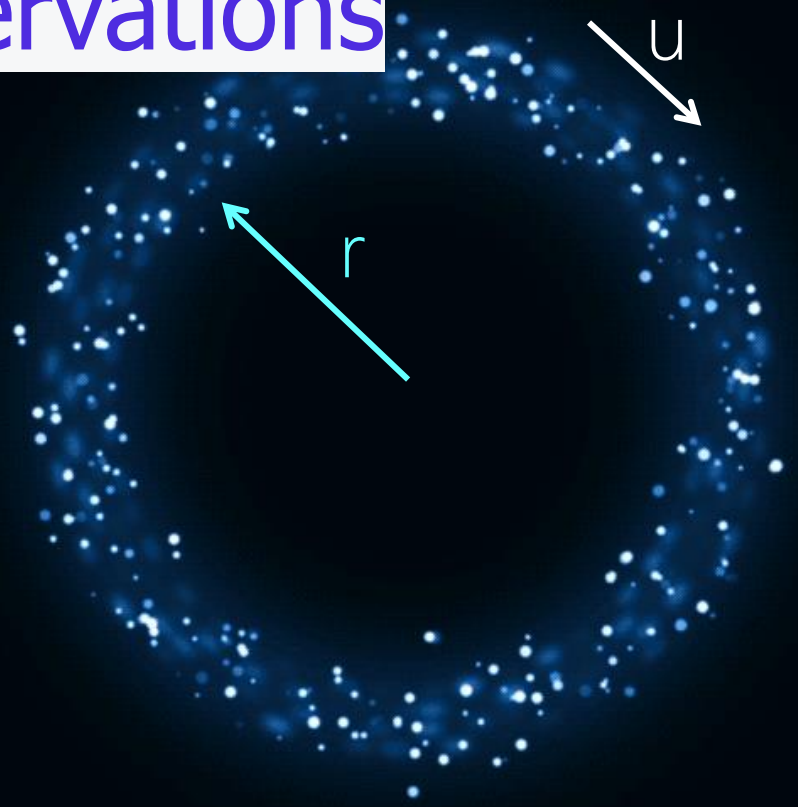
APPLICATIONS

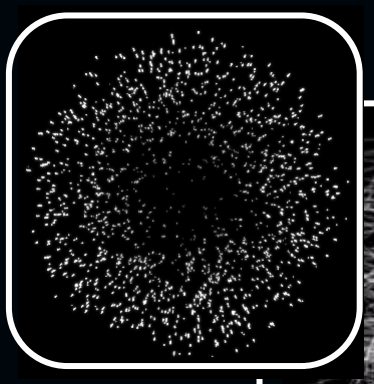
Medical Imaging
Geophysics
Engineering



Multi-Fidelity Observations
Multi-Physics



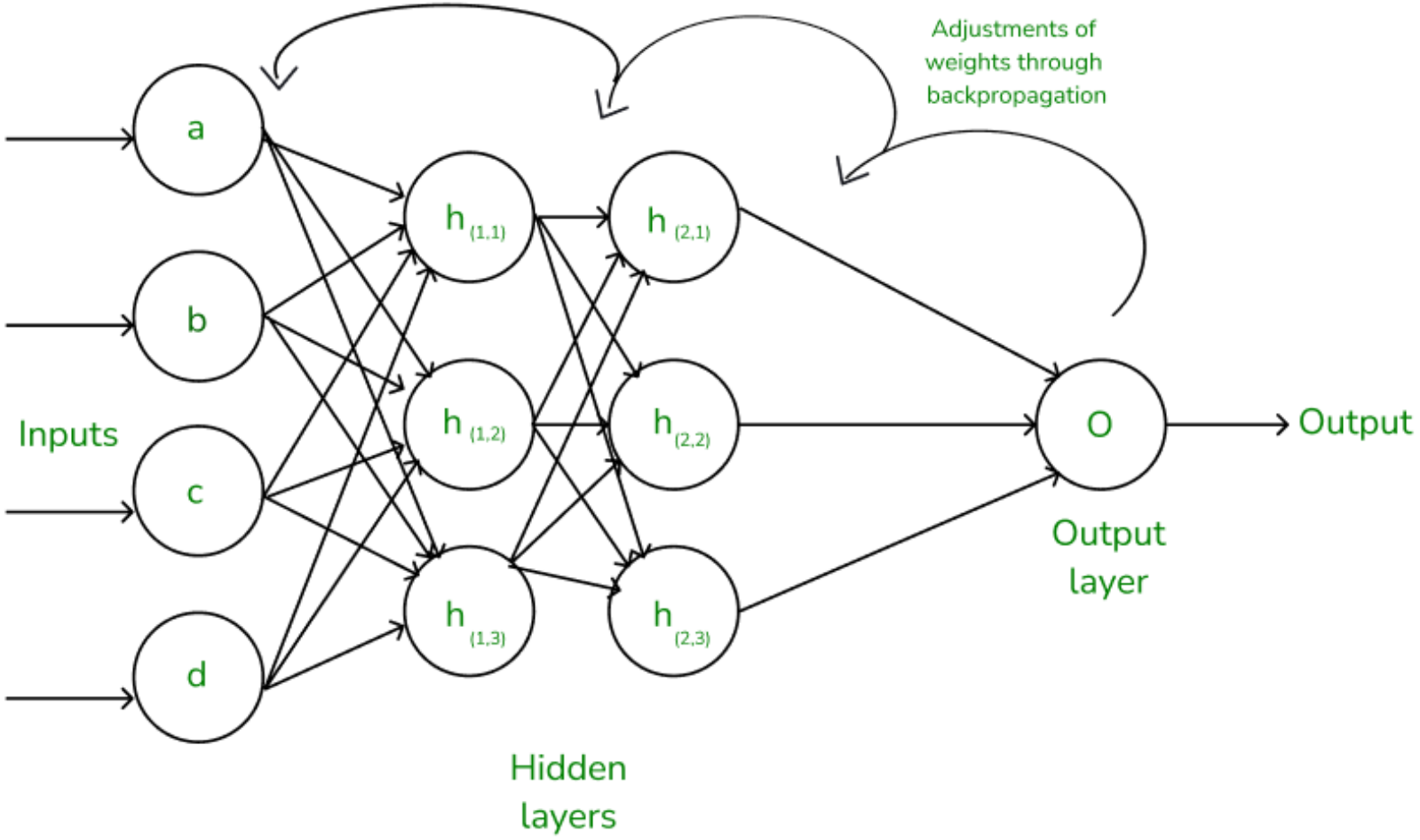


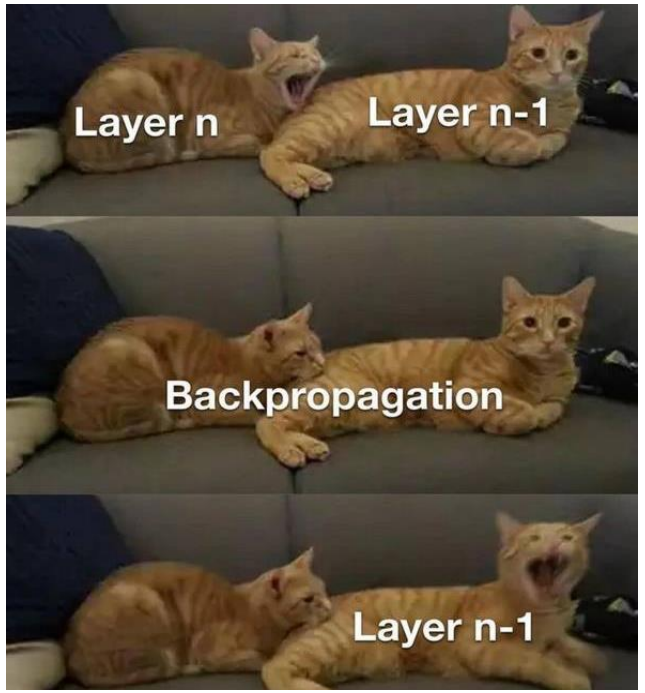


Domain Knowledge

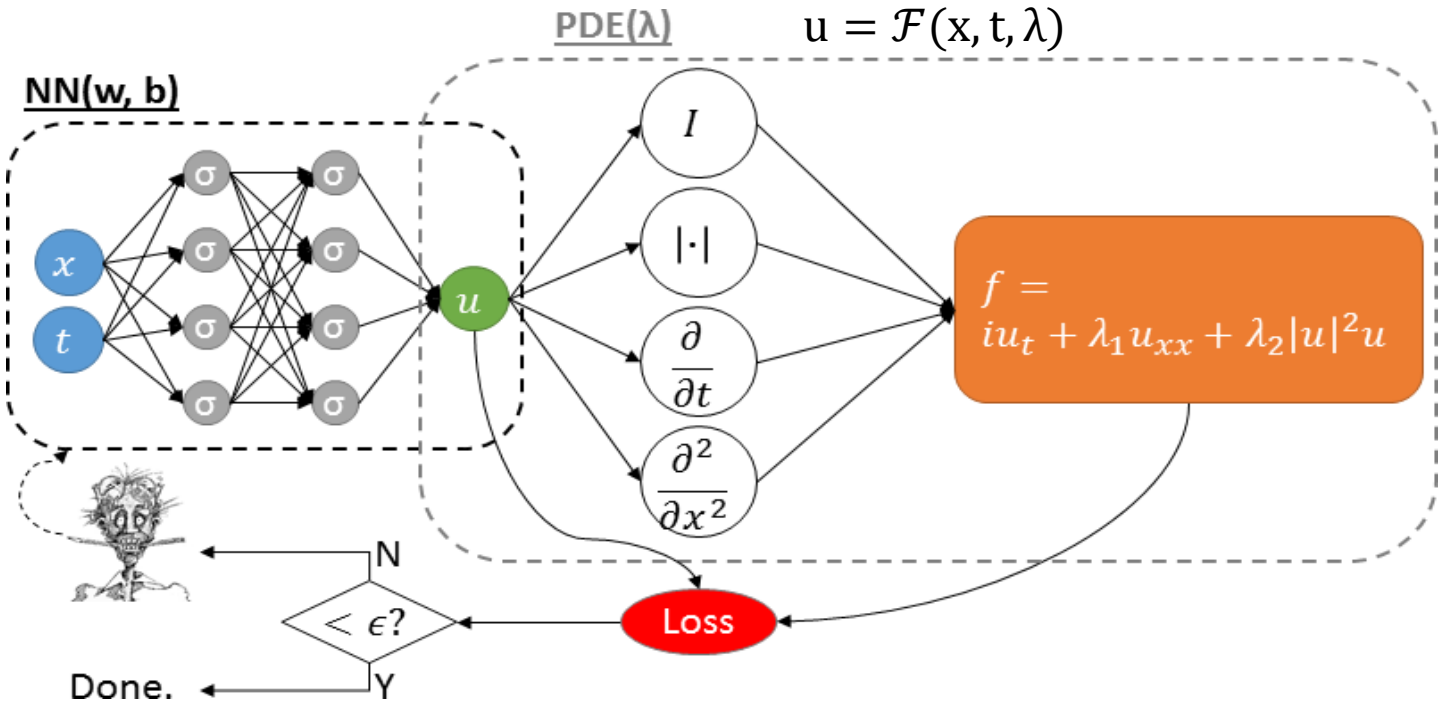
Multi-Fidelity Data

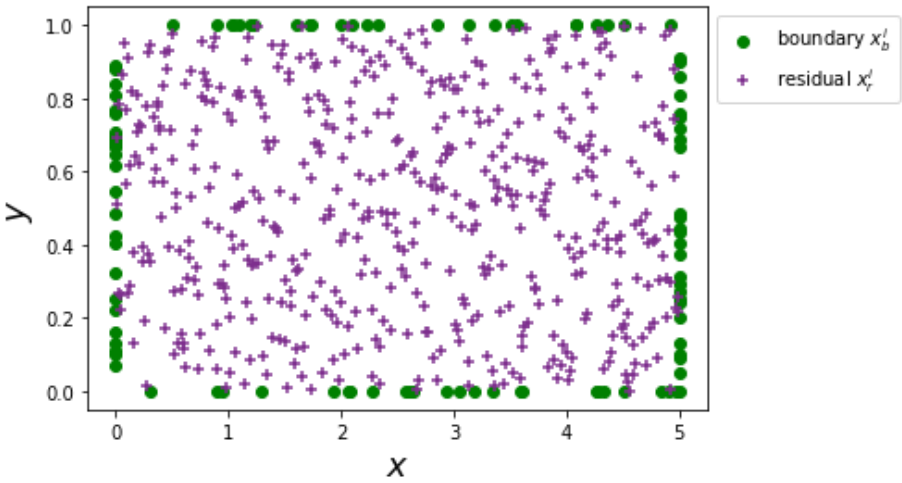
Backpropagation





What is a PINNs? Physics-Informed Neural Networks



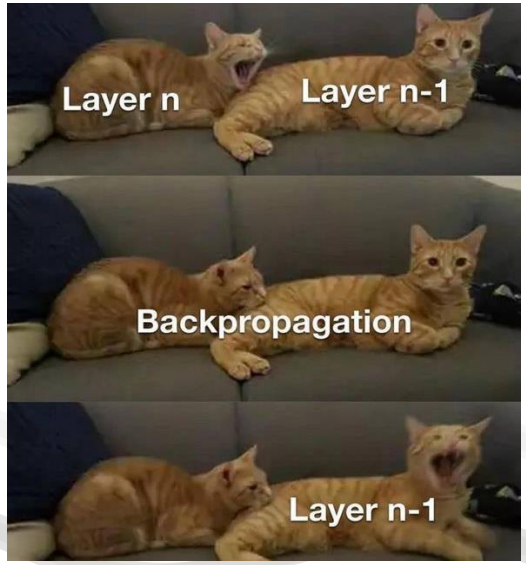


- Minimize:

$$MSE = MSE_u + MSE_f,$$

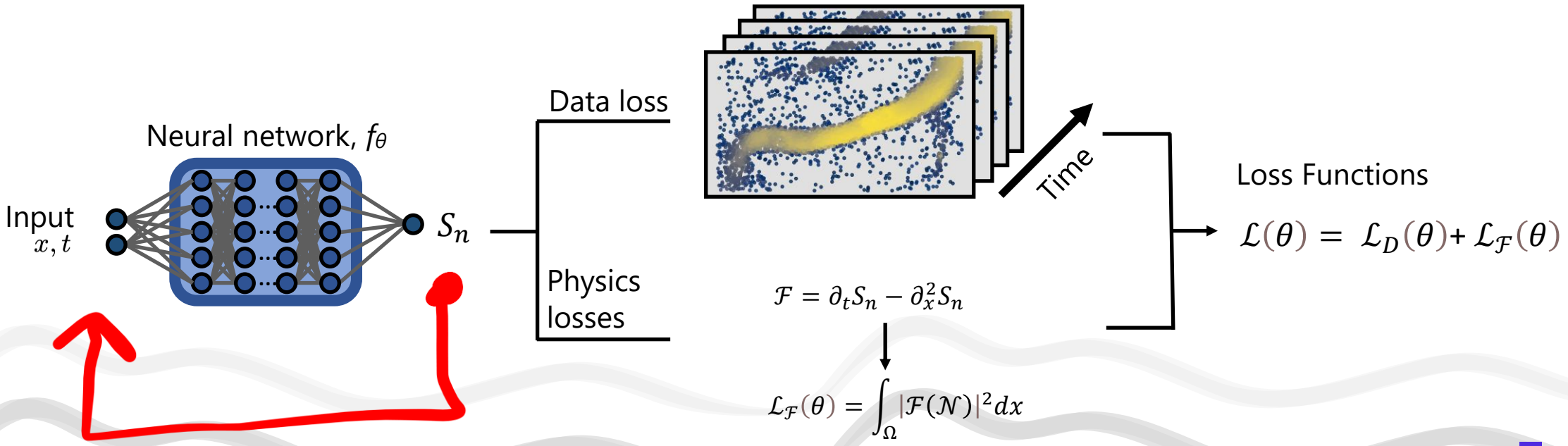
$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

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



Physics Informed Neural Networks(PINNs)

- Are a type of universal function approximators that can embed the knowledge of any physical laws
- The outputs are governed by partial differential equations (PDEs);

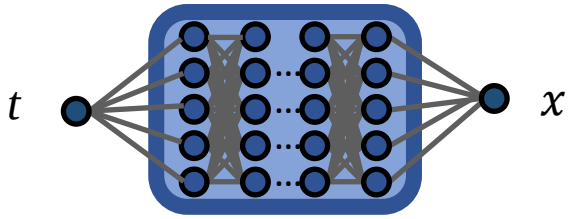


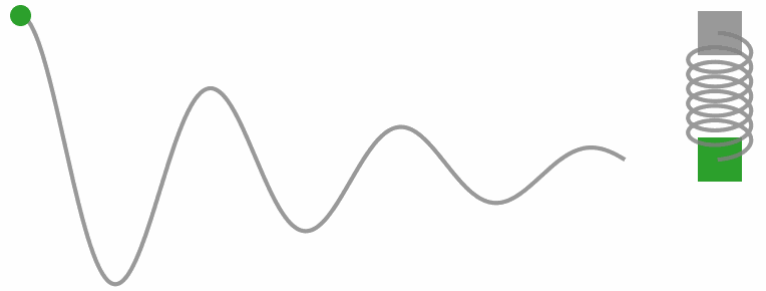
Riel et al., (2021)

Harmonic-Oscillator

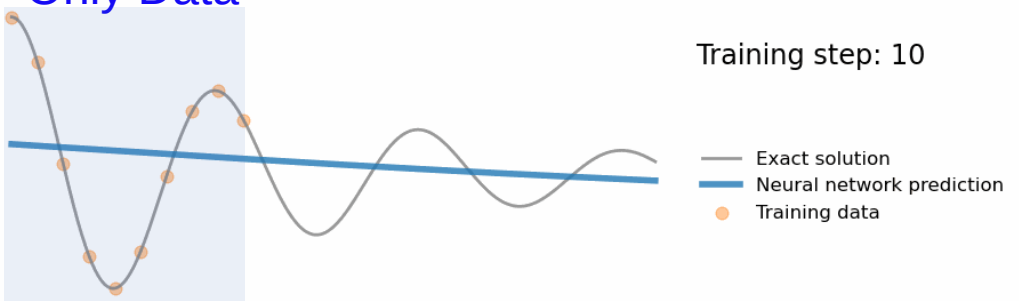
The dynamics of movements in mechanical oscillators such as spring or pendulum

$$f(x,t) = m\frac{\partial^2 x}{\partial t^2} + \mu\frac{\partial x}{\partial t} + kx = 0$$

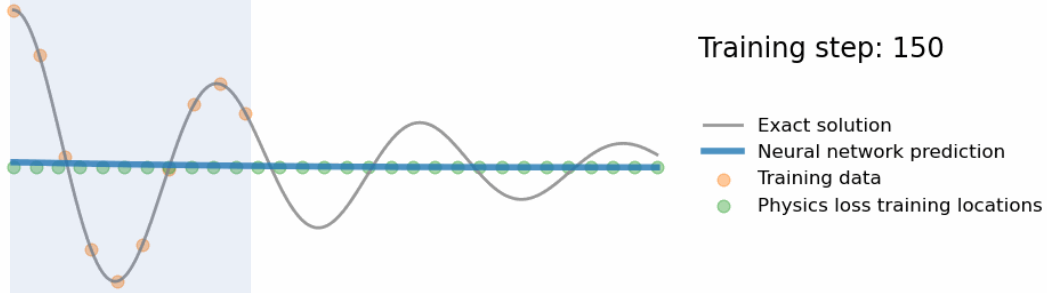




Only Data



Data + Physics



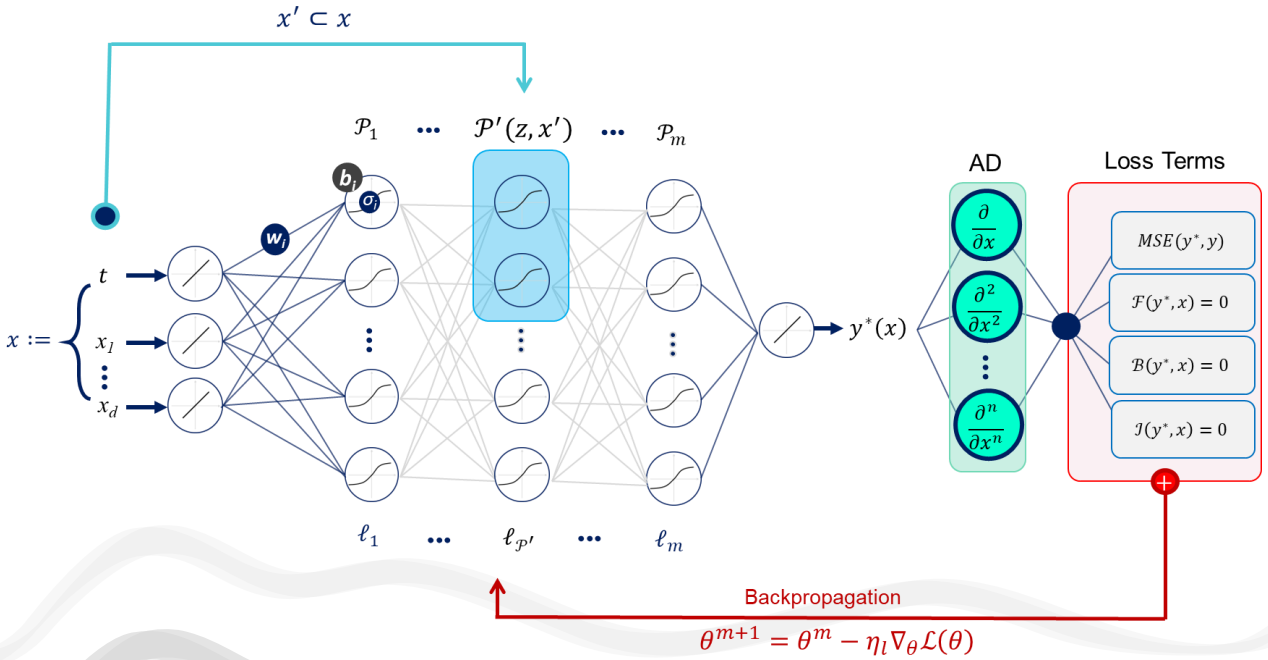
Reference: https://github.com/benmoseley/harmonic-oscillator-pinn

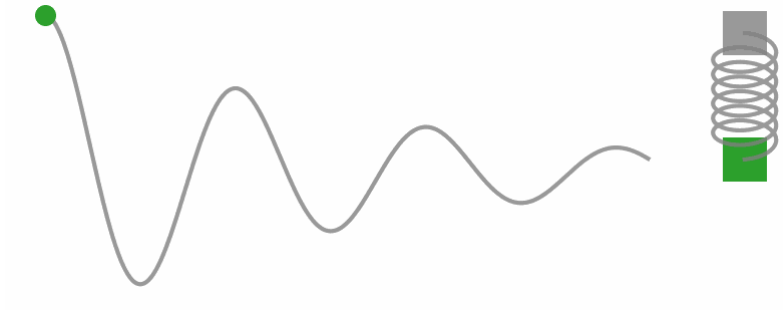
Harmonic-Oscillator

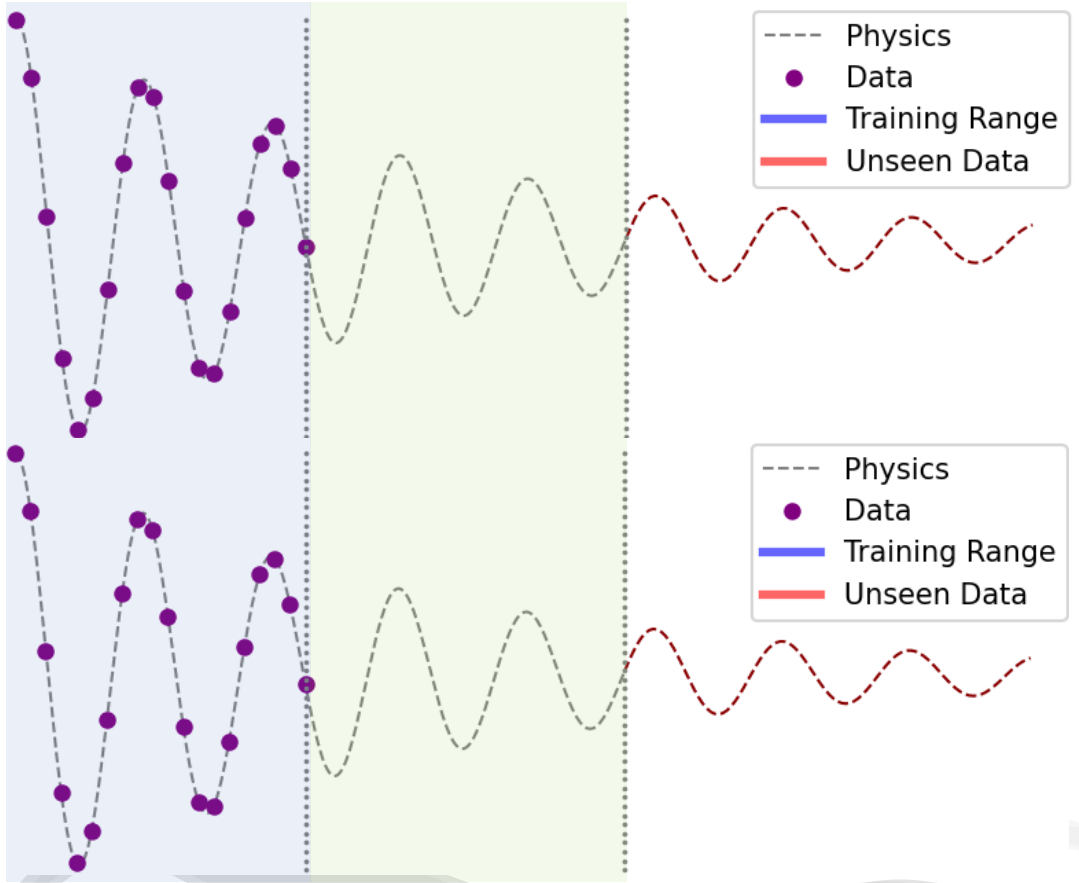
Activation Functions

The dynamics of movements in mechanical oscillators such as spring or pendulum

$$f(x,t) = m\frac{\partial^2 x}{\partial t^2} + \mu\frac{\partial x}{\partial t} + kx = 0$$





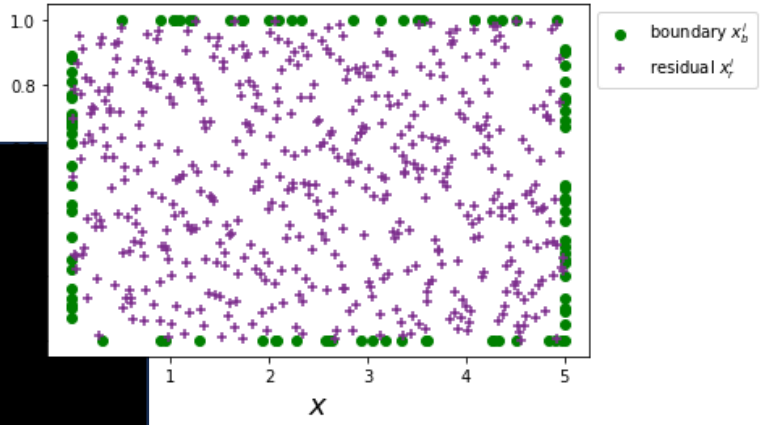


Abbasi and Andersen (2022) – Under Review

How do we calculate PDE residual loss?

$R := u_t + q u_x = 0$

at t=0: u=0; at x=0: u=1




```
u = model(x, t)

1   du_dt= torch.autograd.grad(u, t)[0]
    du_dx= torch.autograd.grad(u, x)[0]

2   residual = du_dt + q * du_dx

3   loss_physics = torch.mean(abs(residual))

ui = model(x, t=0)      >>   loss_ic = torch.mean(abs(ui - 0))
ub = model(x=0, t)      >>   loss_bc = torch.mean(abs(ub - 1))

loss_total = loss_physics + loss_ic + loss_bc
```

PINN for Burger's Equation: Implementation

Burger's Equation is defined as

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

1. Import Modules and Deep Learning Framework

```
import sys
sys.path.insert(0, 'Utilities/')
import os

from scipy.interpolate import griddata
from pyDOE import lhs
from plotting import newfig, savefig
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.gridspec as gridspec
from mpl_toolkits.axes_grid1 import make_axes_locatable
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time
import scipy.io
```

2a. Initialize Layers and NN Parameters

```
layers = [2, 20, 20, 20, 20, 20, 20, 20, 20, 1]
L = len(layers)
W = [hyper_initial([layers[l-1], layers[l]]) for l in range(1, L)]
b = [tf.Variable(tf.zeros([1, layers[l]])) for l in range(1, L)]
```

2b. Glorot Normal Initialization

```
def hyper_initial(size):
    in_dim = size[0]
    out_dim = size[1]
    std = np.sqrt(2.0/(in_dim + out_dim))
    return tf.Variable(tf.random.truncated_normal(shape=size, stddev = std))
```

PINN for Burger's Equation: Implementation

3a. Forward Pass

```
def net_u(x, t, w, b):
    u = DNN(tf.concat([x,t],1), w, b)
    return u
```

3b. DNN Function:

```
# Neural Network
def DNN(X, W, b):
    A = X
    L = len(W)
    for i in range(L-1):
        A = tf.tanh(tf.add(tf.matmul(A, W[i]), b[i]))
    Y = tf.add(tf.matmul(A, W[-1]), b[-1])
    return Y
```

4. Residual Computation

```
def net_f(x,t,W, b, nu):
    with tf.GradientTape(persistent=True) as tape1:
        tape1.watch([x, t])
        with tf.GradientTape(persistent=True) as tape2:
            tape2.watch([x, t])
            u=net_u(x,t, W, b)
        u_t = tape2.gradient(u, t)
        u_x = tape2.gradient(u, x)
    u_xx = tape1.gradient(u_x, x)
    del tape1
    f = u_t + u*u_x - nu*u_xx
    return f
```

PINN for Burger's Equation: Implementation

5. Backward Propagation

```
def train_step(W, b, X_u_train_tf, u_train_tf, X_f_train_tf, opt, nu):
    x_u = X_u_train_tf[:,0:1]
    t_u = X_u_train_tf[:,1:2]
    x_f = X_f_train_tf[:,0:1]
    t_f = X_f_train_tf[:,1:2]
    with tf.GradientTape() as tape:
        tape.watch([W,b])
        u_nn = net_u(x_u, t_u, W, b)
        f_nn = net_f(x_f,t_f, W, b, nu)
        loss =  tf.reduce_mean(tf.square(u_nn - u_train_tf)) + tf.reduce_mean(tf.square(f_nn))
    grads = tape.gradient(loss, train_vars(W,b))
    opt.apply_gradients(zip(grads, train_vars(W,b)))
    return loss
```

6. Predict

```
def predict(X_star_tf, w, b):
    x_star = X_star_tf[:,0:1]
    t_star = X_star_tf[:,1:2]
    u_pred = net_u(x, t, w, b)
    return u_pred
```

PINN for Burger's Equation: Implementation

7. Data Preparation and Training

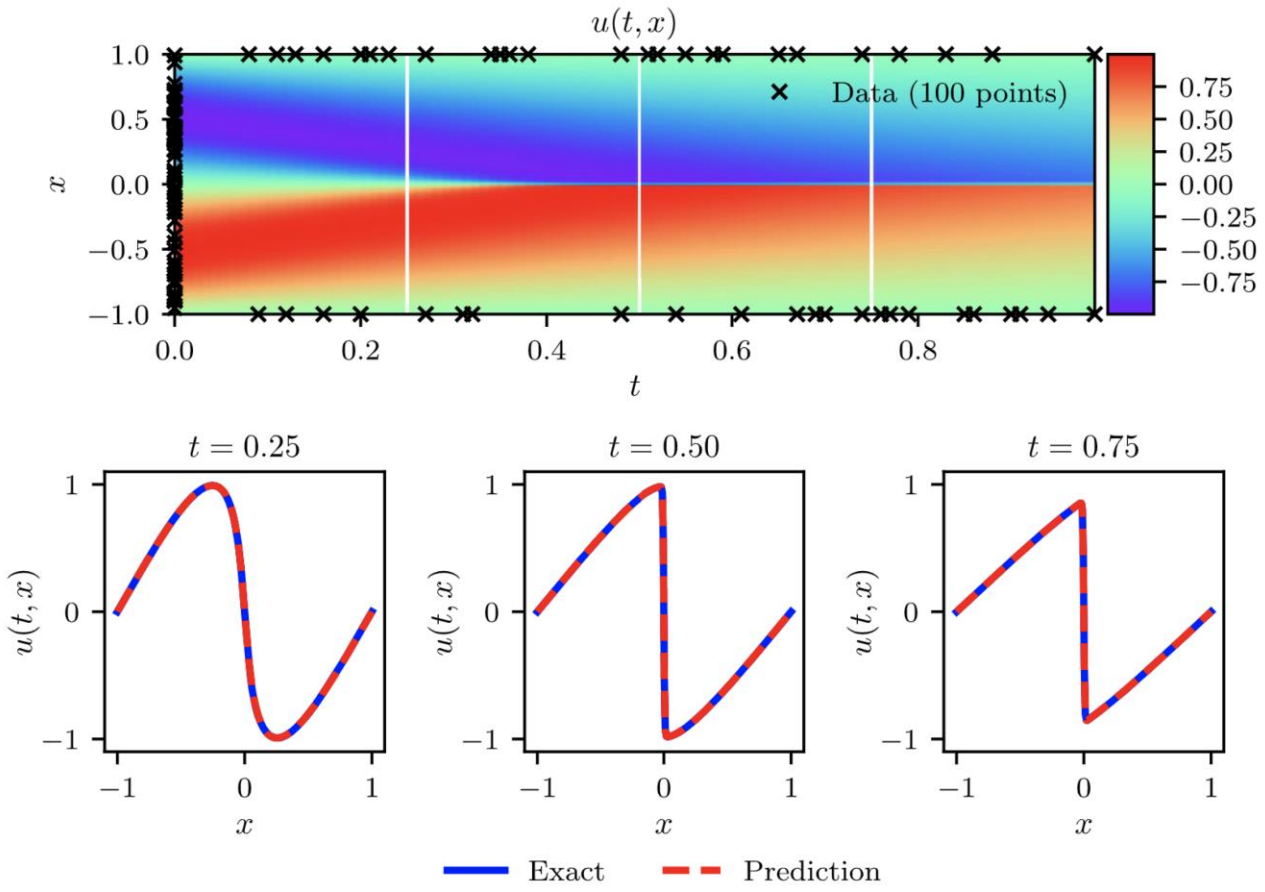
```
nu = 0.01/np.pi
noise = 0.0
N_u = 100
N_f = 10000
Nmax=10000

layers = [2, 20, 20, 20, 20, 20, 20, 20, 20, 1]
L = len(layers)
W = [hyper_initial([layers[l-1], layers[l]]) for l in range(1, L)]
b = [tf.Variable(tf.zeros([1, layers[l]])) for l in range(1, L)]

data = scipy.io.loadmat('./Data/burgers_shock.mat')

t = data['t'].flatten()[:,None]
x = data['x'].flatten()[:,None]
Exact = np.real(data['usol']).T
X, T = np.meshgrid(x,t)
X_star = np.hstack((X.flatten()[:,None], T.flatten()[:,None]))
u_star = Exact.flatten()[:,None]
# Doman bounds
lb = X_star.min(0)
ub = X_star.max(0)
xx1 = np.hstack((X[0:1,:].T, T[0:1,:].T))
uu1 = Exact[0:1,:].T
xx2 = np.hstack((X[:,0:1], T[:,0:1]))
uu2 = Exact[:,0:1]
xx3 = np.hstack((X[:,-1:], T[:,-1:]))
uu3 = Exact[:,-1:]

X_u_train = np.vstack([xx1, xx2, xx3])
X_f_train = lb + (ub-lb)*lhs(2, N_f)
X_f_train = np.vstack((X_f_train, X_u_train))
u_train = np.vstack([uu1, uu2, uu3])

idx = np.random.choice(X_u_train.shape[0], N_u, replace=False)

X_u_train = X_u_train[idx, :]
u_train = u_train[idx,:]

X_u_train_tf = tf.convert_to_tensor(X_u_train, dtype=tf.float32)
u_train_tf =   tf.convert_to_tensor(u_train, dtype=tf.float32)
X_f_train_tf = tf.convert_to_tensor(X_f_train, dtype=tf.float32)
```

8. Optimizer and PINN Training

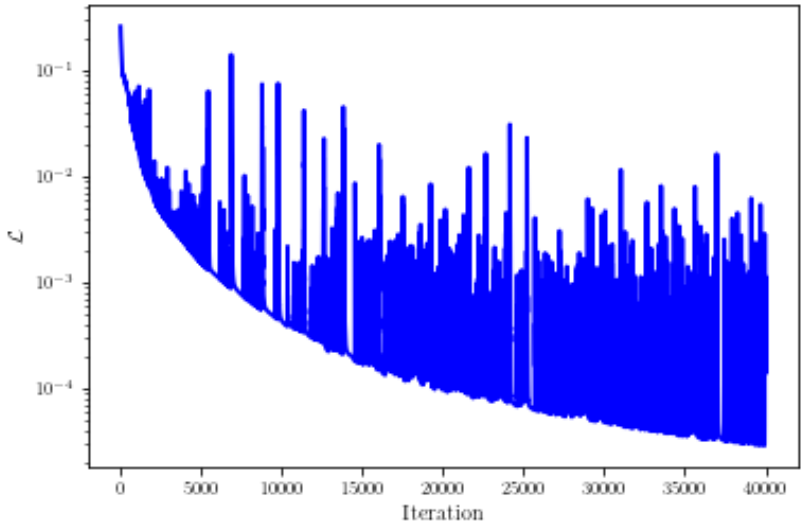
```
lr = 1e-3
optimizer = tf.optimizers.Adam(learning_rate=lr)

start_time = time.time()
n=0
loss = []
while n <= Nmax:
    loss_= train_step(W, b, X_u_train_tf, u_train_tf, X_f_train_tf, optimizer, nu)
```

PINN for Burger's Equation: Results

8. Actual vs Predicted Solutions at Different Times



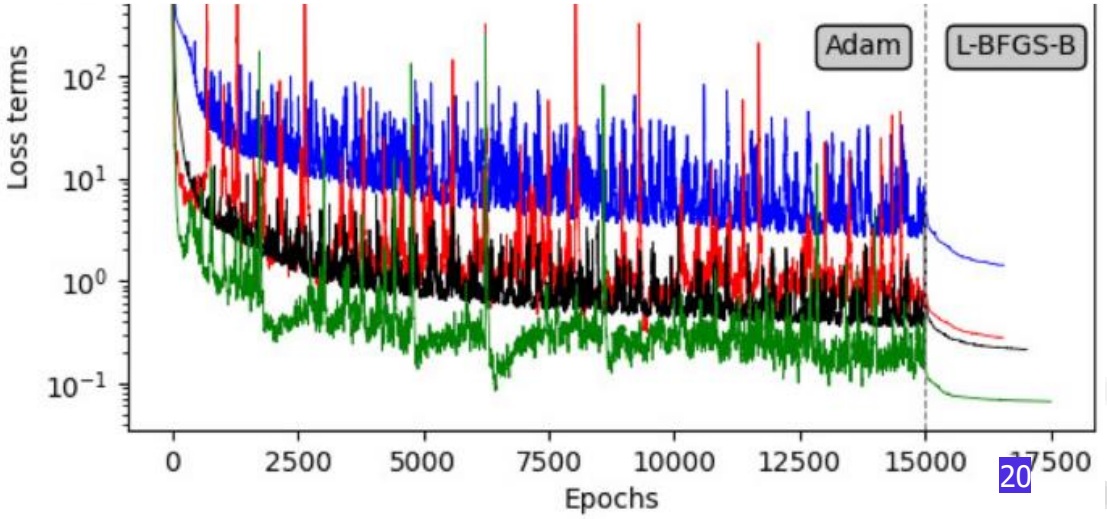
9. Loss Function



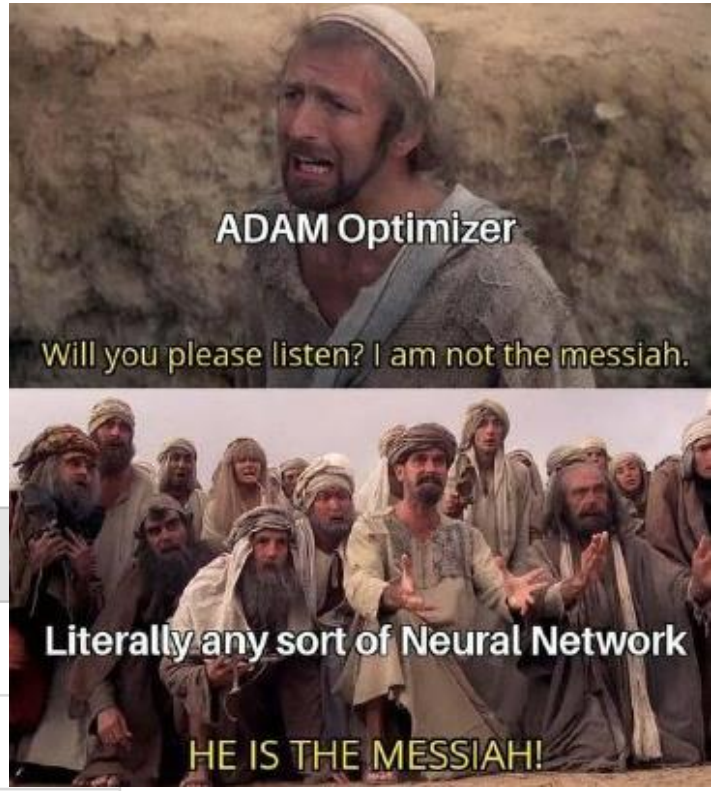
Optimization

When training PINNs, choosing the right optimizer is crucial.

- **Adam**: Commonly used for PINNs due to its adaptive learning rate and robustness across various tasks.
- **L-BFGS**: Second order optimizers like L-BFGS handle the high-dimensional, stiff optimization landscape in PINNs. They are often used in combination with Adam for final fine-tuning.



Optimization

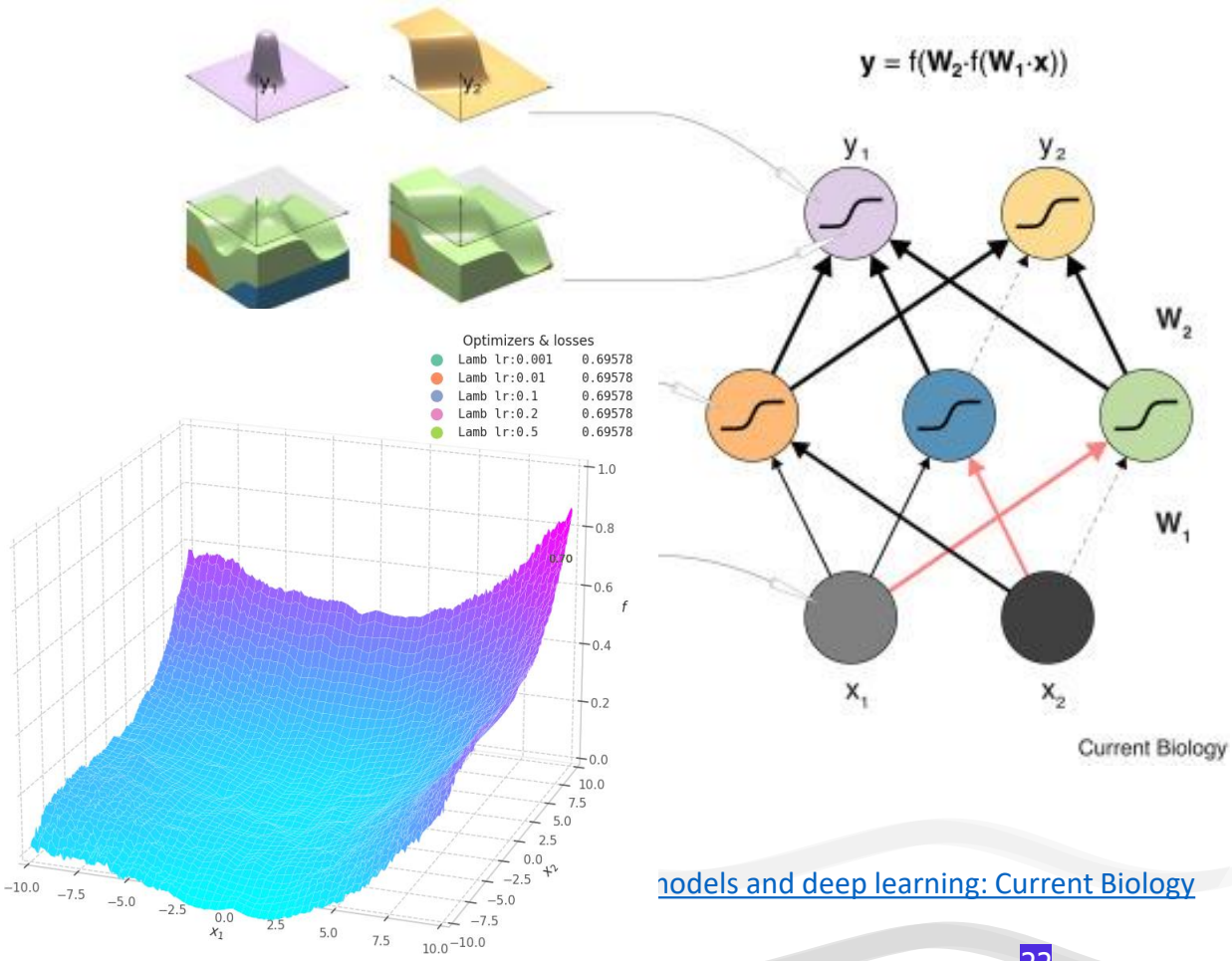


Summary of Runtime Differences

Aspect	Adam	L-BFGS
Iterations	More iterations needed	Fewer iterations
Per-Iteration Cost	Low	High
Gradient Evaluation	Mini-batch	Full-batch or large batches
Memory Usage	Low	Higher (stores history)
Convergence	Slower for smooth problems	Faster for smooth problems
Scalability	Highly scalable	Limited scalability

Why do we expect NNs to be helpful?

- MLPs with non-linear activations are universal function approximators
- However, they may require exponentially large number of neurons
- The missing piece in universal approximation theorem is that it only considers approximation error, and not trainability and/or generalizability of the NN.



[models and deep learning: Current Biology]

Error Decomposition

We can broadly characterize the accuracy of a NN into three main types:

Approximation Error: Model's inability to capture the true data distribution.

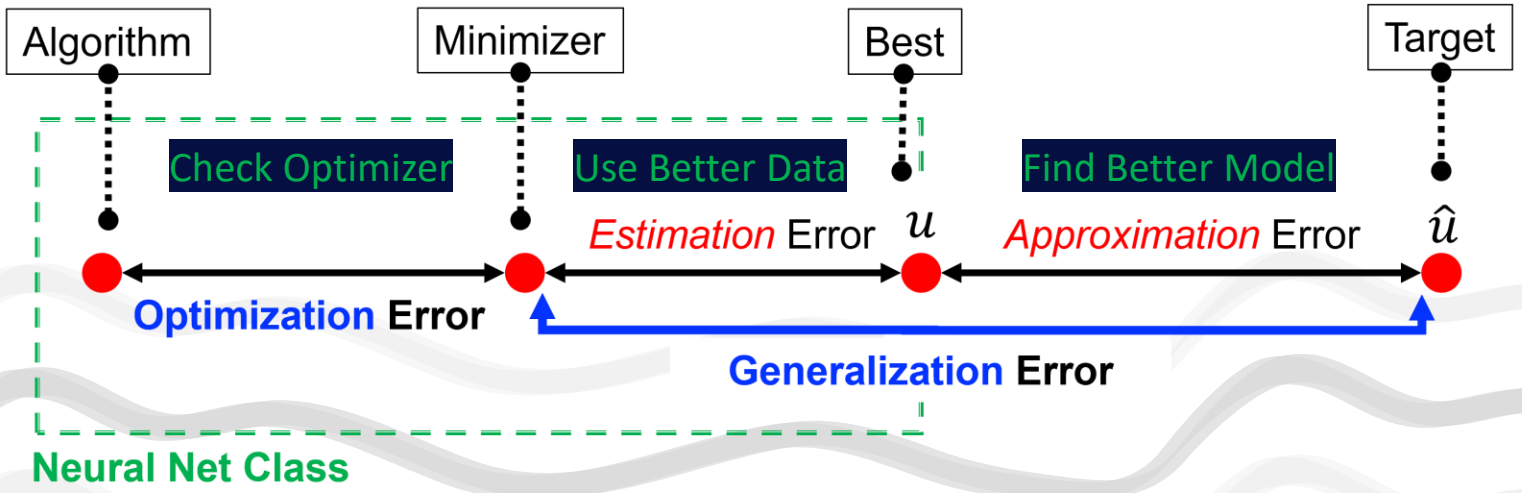
Optimization Error: Related to how well the optimization algorithm finds a suitable solution.

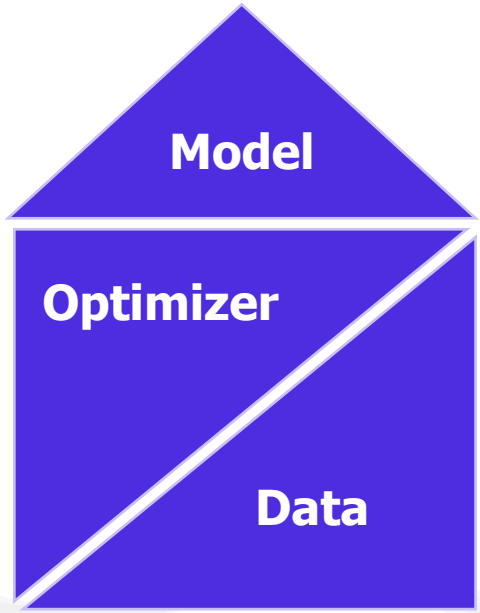
Estimation Error: Discrepancy between the true model and the learned model due to limited training data.

>> Generalization Error: Reflects model performance on unseen data, influenced by overfitting or underfitting.

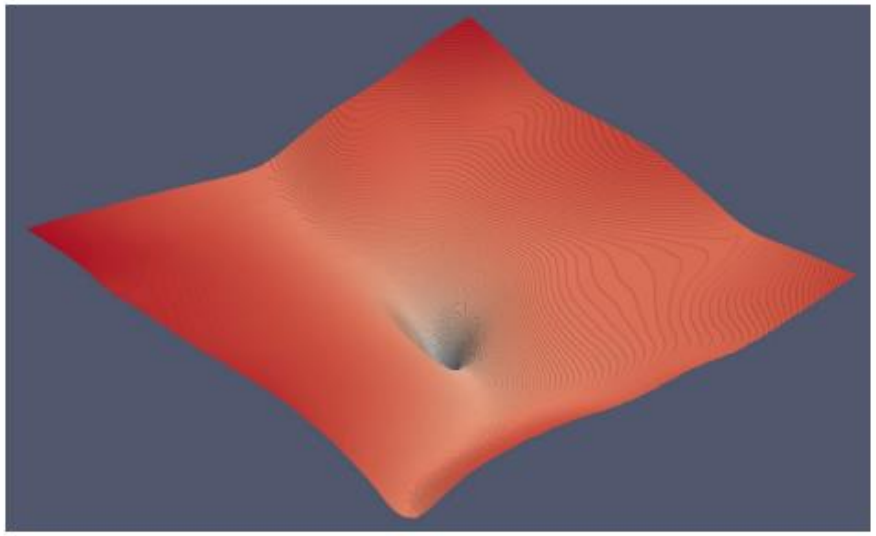
Universal approximation theorem only considers the first one

- It provides no method to train a model to get that approximation

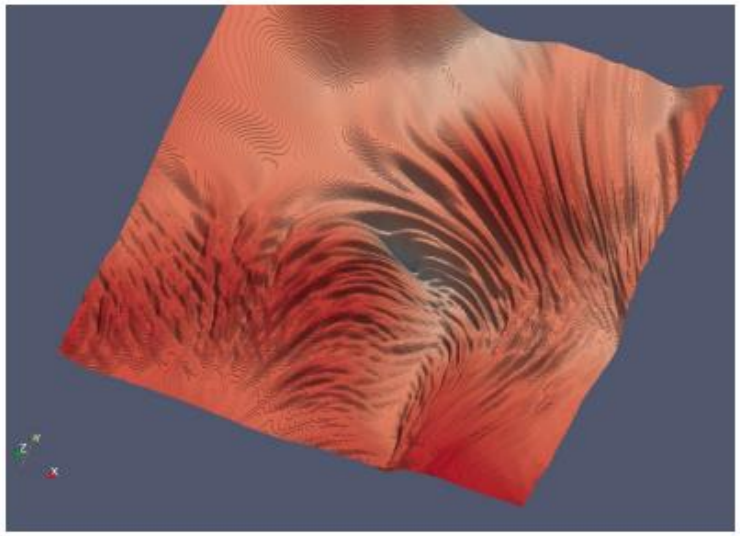




Training Challenges
Optimization with PINNs



Without Physics Loss



With Physics Loss

Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., & Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. *Advances in neural information processing systems*, *34*, 26548-26560.

Interesting graphics: https://losslandscape.com/

Advantages / limitations of PINNs

Advantages

- Mesh-free
- Easy to implement
- Can jointly solve forward and inverse problems
- Often performs well on “messy” problems (where some observational data is available)
- Tractable, analytical solution gradients (e.g. for sensitivity analysis)
- Mostly unsupervised (to data)

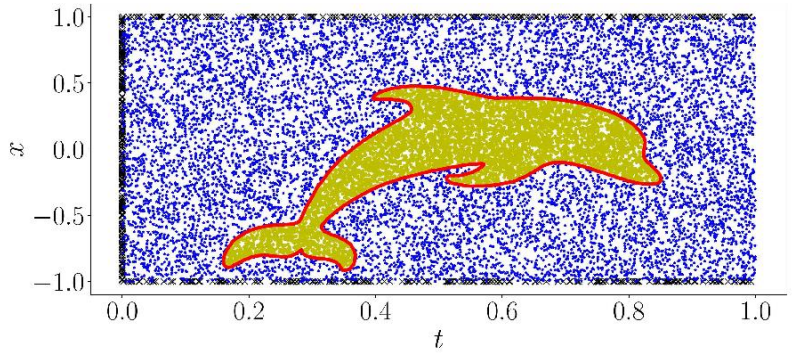
Limitations

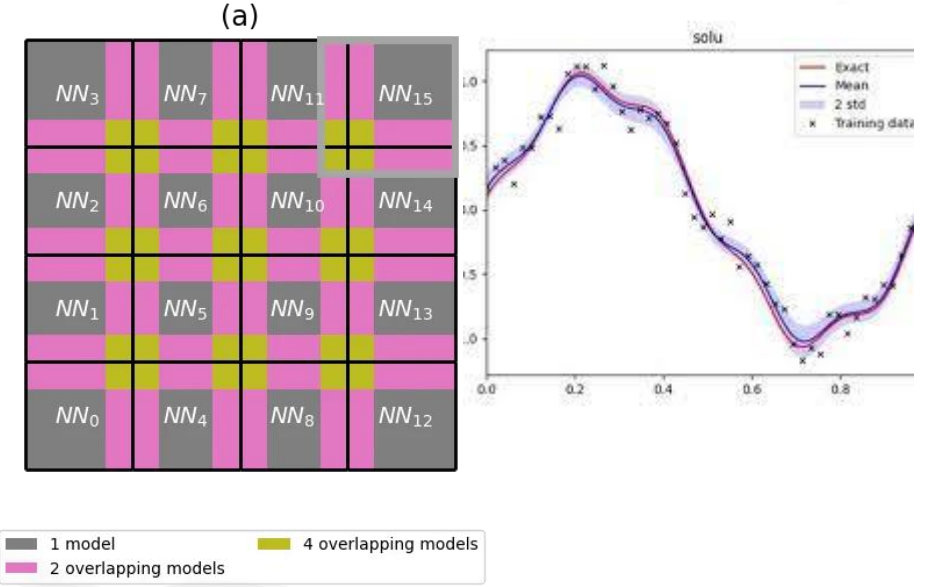
- Computational cost
- Poor convergence
- Scaling to more complex problems

- A short break

Variations of PINNs?

- Conservative PINNs (cPINNs)
- Extended PINNs (XPINNs)
- Variational PINNs (VPINNs)
- Stochastic PINNs (SPINNs)
- Adaptive PINNs (APINNs)
- Finite Basis PINNs (FBPINNs)
- Augmented PINNs (A-PINNs)
- Fourier PINNs (F-PINNs)
- Bayesian PINNs (B-PINNs)



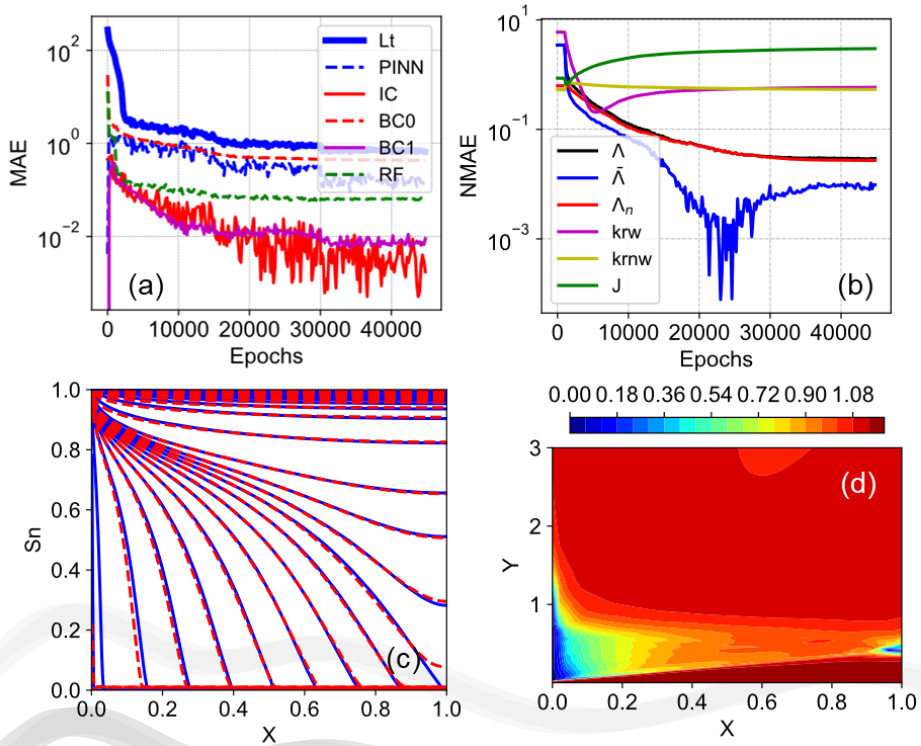


How to improve PINNs?
Adaptive Loss Weighting

- A technique designed to continuously adjust the weights of different terms in the loss function during the training.

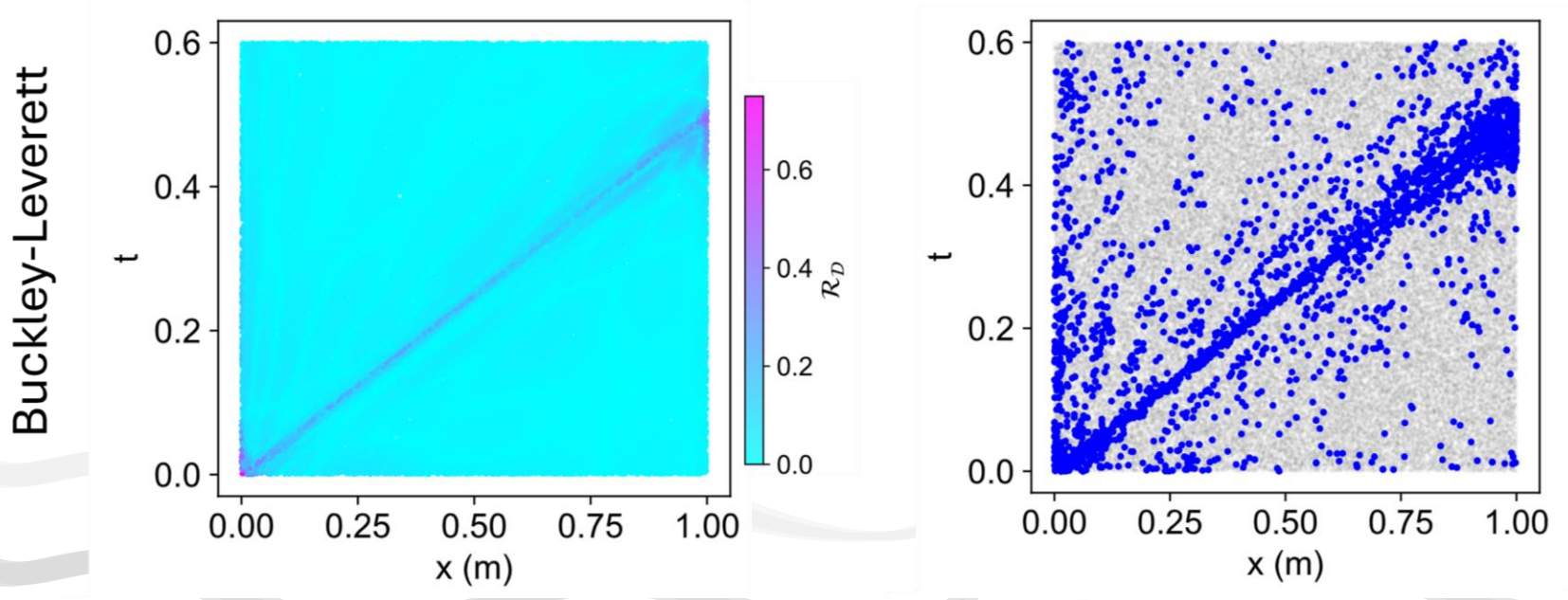
$$\mathcal{L}_{SA}\left(\theta_{\psi}\right)=\log\left(\prod_{\Omega}\theta_{\psi}\left(X,Y\right)\right)$$

$$\mathcal{L}_{\text{phys}}(\theta)=\frac{1}{M}\sum_{j=1}^{M}\mathcal{E}\left(e^{\theta_{\psi}}\mathcal{F}(\hat{u}(\mathbf{x}_j,t_j;\theta)),0\right),$$

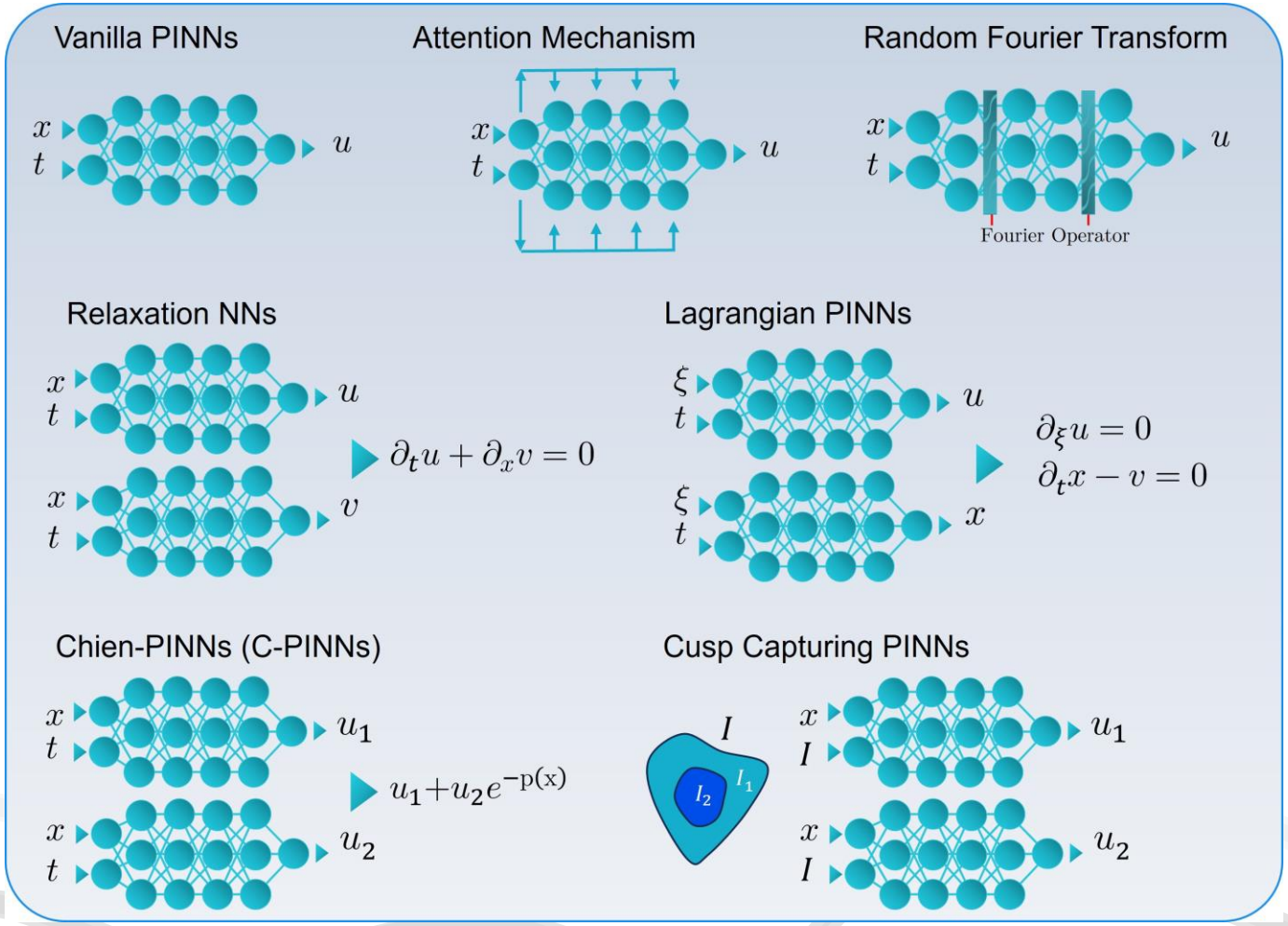


How to improve PINNs?
Adaptive Collocation Points

- Adaptive collocation points in PINNs dynamically modifies the spatiotemporal placement of collocation points
- It works either randomly, or based on the different criteria, such as solution's error estimates.

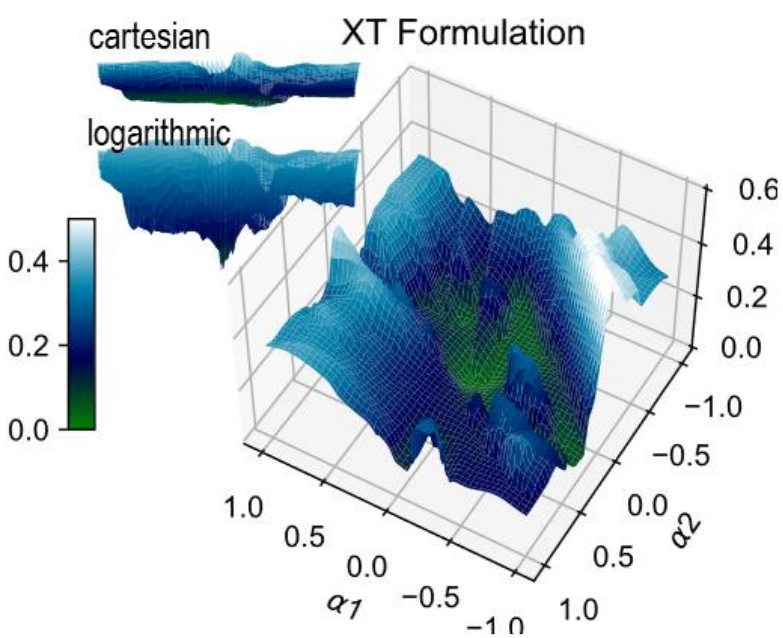


How to improve PINNs?
Improved Network Architecture

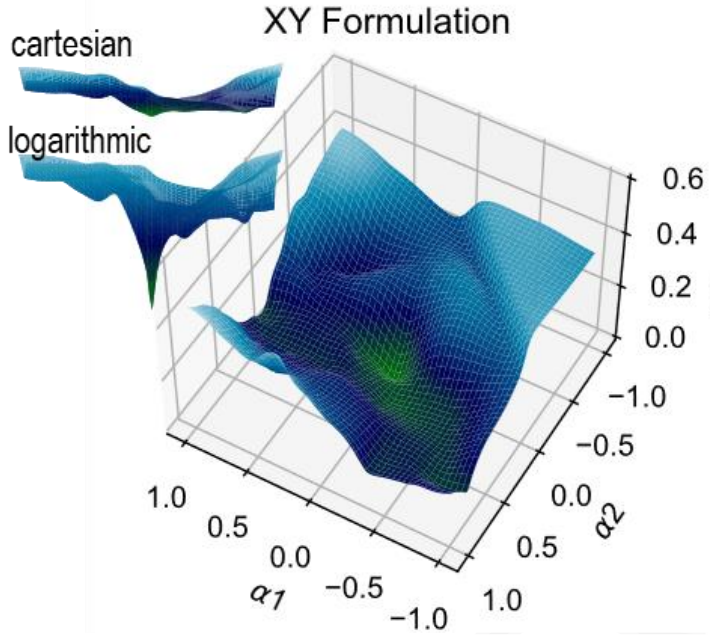


How to improve PINNs?
Normalized Mathematics / Changing Coordinates

- Different studies have proven the performance of PINNs can be significantly improved if the dimensionless forms of PDEs are applied to define residual loss of the system

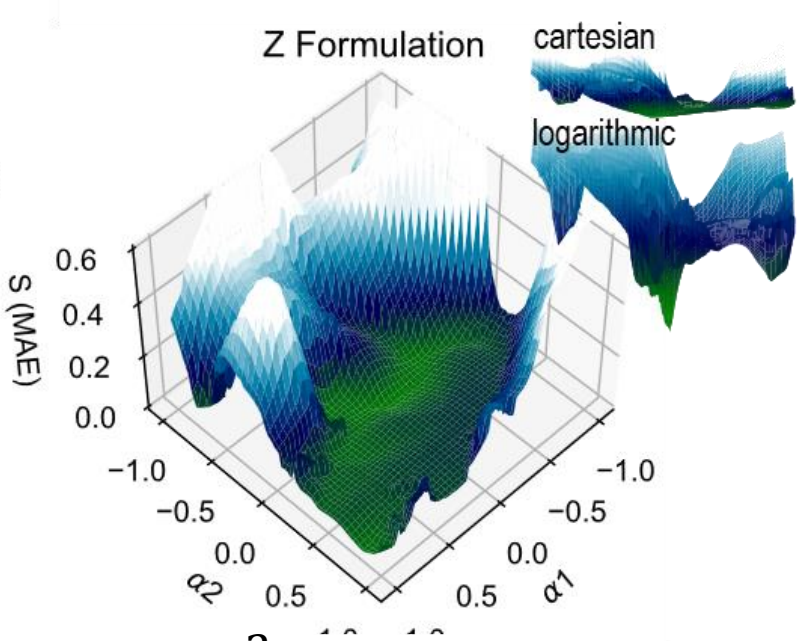


$$\partial_T S_n = \partial_X(\Lambda_n(S_n)\partial_X S_n)$$



$$\partial_Y S_n = 2\mathrm{Y}\partial_X(\Lambda_n(S_n)\partial_X S_n)$$

$$\mathrm{Y} = T^{0.5}$$



$$\partial_Z S_n = -\frac{2}{Z}\partial_Z(\Lambda_n(S_n)\partial_Z S_n)$$

$$Z = X/T^{0.5}$$

Abbasi et. al., 2023,
https://pubs.acs.org/doi/full/10.1021/acs.energyfuels.3c02271

How to improve PINNs?
Respecting Causality

- Causality in PINNs refers to the principle that the solution of time-dependent PDEs should respect the natural temporal order of events.
- In other words, the state of a system at a given time should preferably depend on its past states and not on future states.

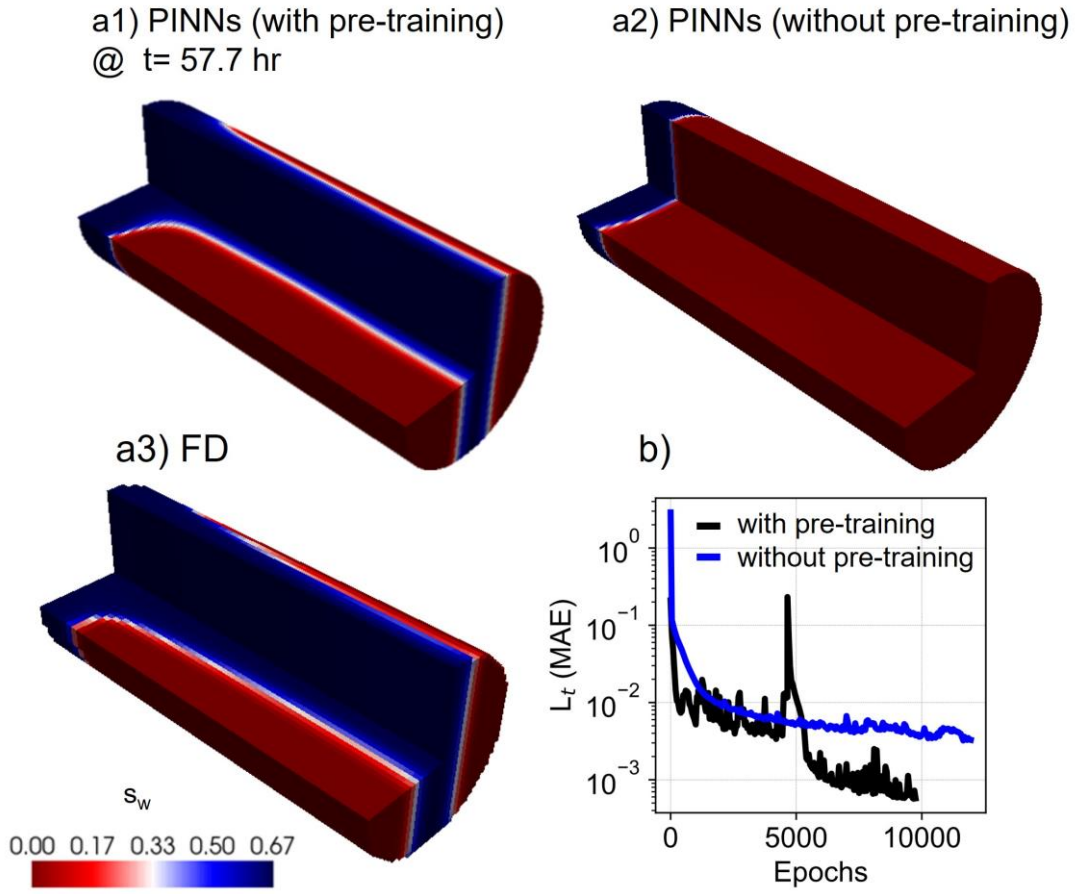
$$\mathcal{L}_t = \sum_{i=1}^{N_t} \beta(t_i)\mathcal{L}_{\text{phys}}(\theta, t_i) + \mathcal{L}_{\text{IC}}(\theta) + \mathcal{L}_{\text{BC}}(\theta)$$

$$\beta(t_i) = \exp(-\alpha t_i)$$

Wang et al., 2021, RESPECTING CAUSALITY IS ALL YOU NEED FOR TRAINING PHYSICS-INFORMED NEURAL NETWORKS

How to improve PINNs?
Curriculum Learning

- An advanced training strategy that aims to improve the performance and convergence of PINNs, especially when dealing with more complex systems of PDEs.
- The idea is to start training the network on easier version of the problem before gradually introducing more challenging aspects of the problem

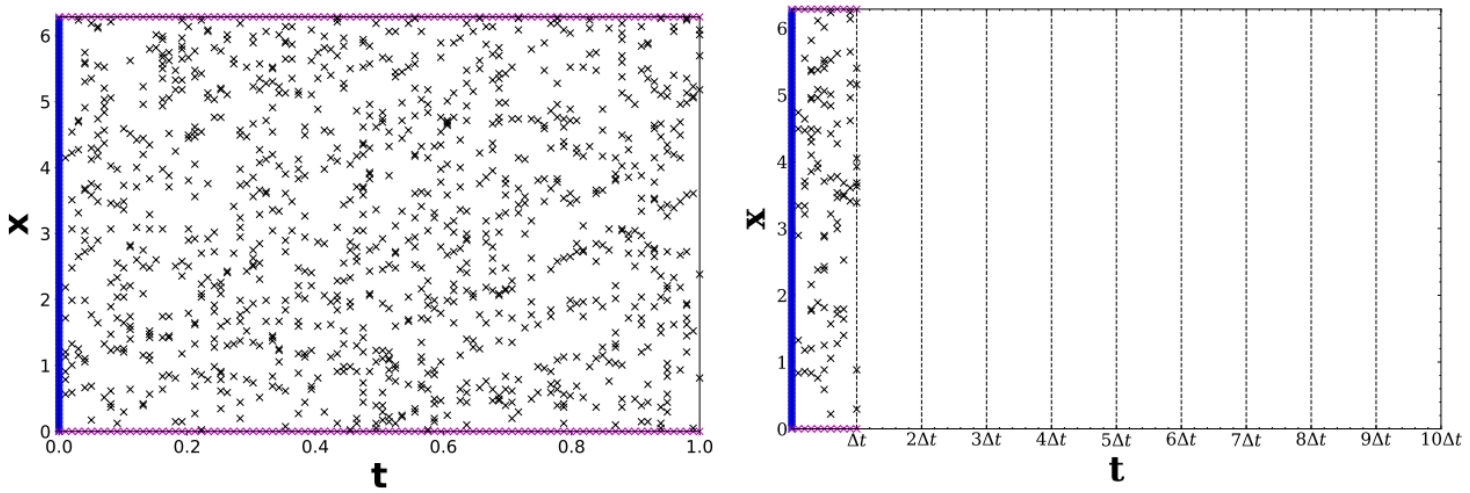


2109.01050

Abbasi et al., 2024, https://arxiv.org/abs/2410.20801

How to improve PINNs?
Sequence-to-sequence learning [seq2seq]

- In contrast to regular PINN training, the solution in seq2seq learning is predicted for only one Δt step at a time. Then, the predicted solution at $t = \Delta t$ is used as the initial condition for the next segment

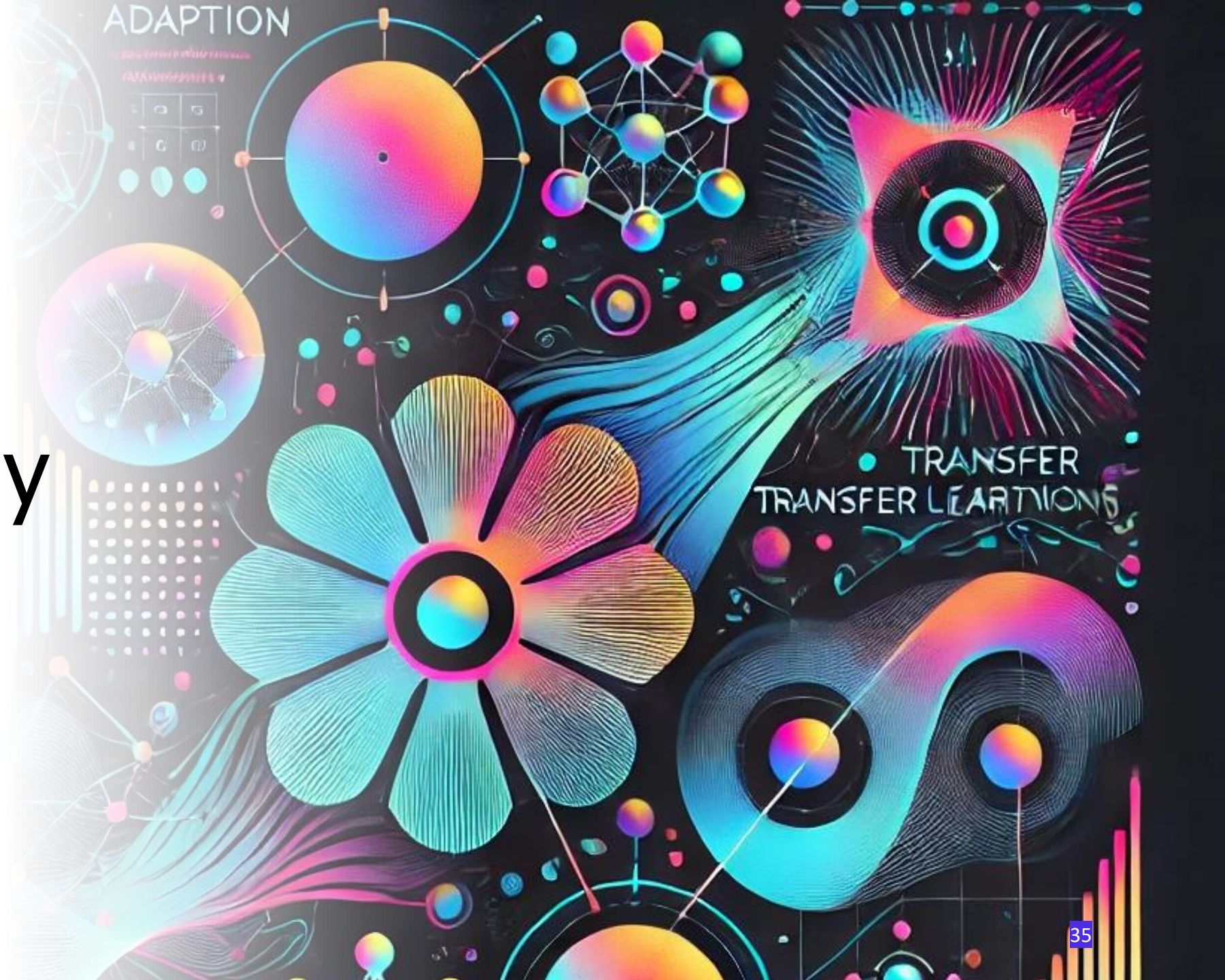


(a) *Regular PINN training*

(b) *Sequence-to-sequence learning (model trained every Δt)*

Characterizing possible failure modes in physics-informed neural networks, https://arxiv.org/abs/2109.01050

And many more...

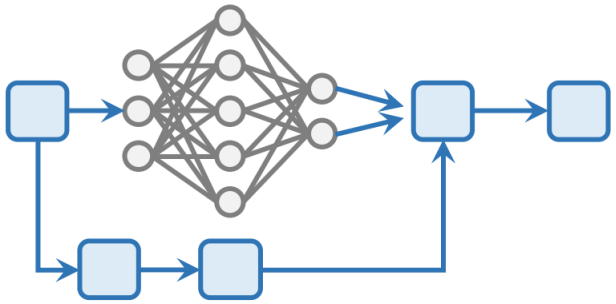


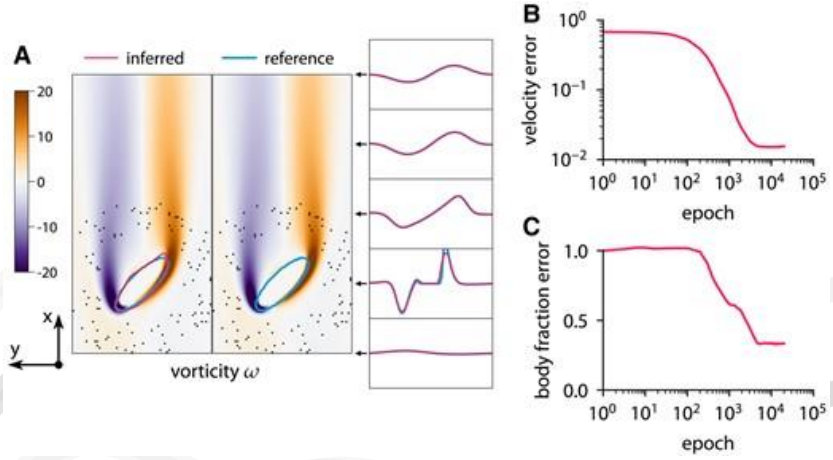
What is next?

- PINNs is just one of the first techniques for physics based neural calculations.

e.g.,

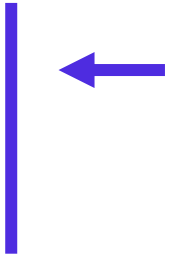
- **ODIL** (Optimizing a Discrete Loss) - https://github.com/cselab/odil
- Hybrid Methods (Neural-Numerical Integration)
- Physics-informed Convolutional Neural Networks (PICNN)



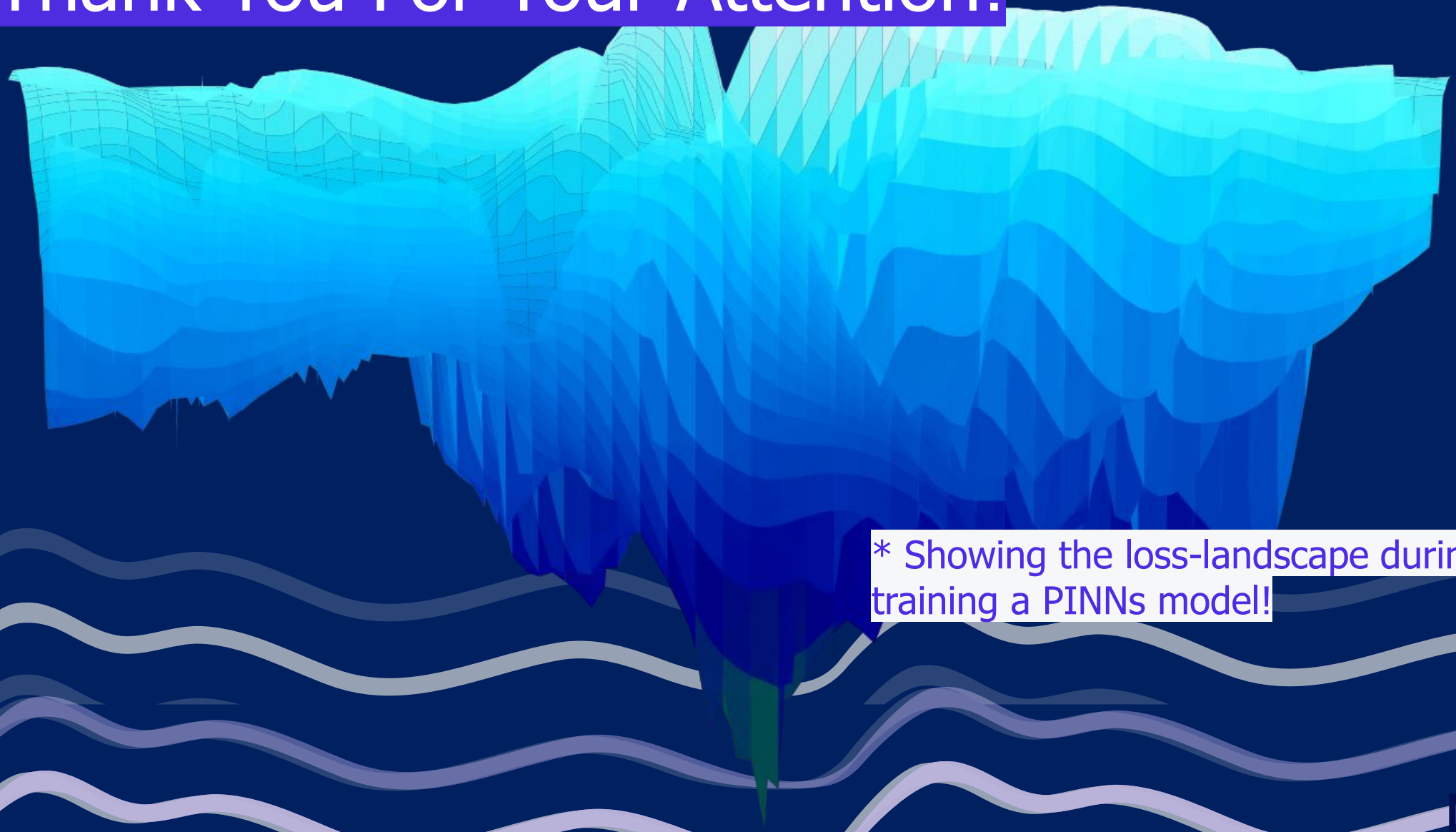


PINNs for Flow in Porous Media

- **"Simulation and Prediction of Countercurrent Spontaneous Imbibition at Early and Late Times Using Physics-Informed Neural Networks"** by Jassem Abbasi and Pål Østebø Andersen (2023)
- **" Application of Physics-Informed Neural Networks for Estimation of Saturation Functions from Countercurrent Spontaneous Imbibition Tests"** by Jassem Abbasi and Pål Østebø Andersen (2024)
- **"History-Matching of Imbibition Flow in Multiscale Fractured Porous Media Using Physics-Informed Neural Networks"** by J. Abbasi and P. Ø. Andersen, … (2024)



Thank You For Your Attention!



* Showing the loss-landscape during training a PINNs model!