



# **Solving Mathematical Models using ML: Application to Defects in Alloys**

DDP Stage 1 Presentation   by   Shivprasad Kathane   20/10/2022

# Outline



- Project Introduction & ML
- Deep Learning for Defect Detection
- Prediction of Microstructural Evolution
- Solving a simple ODE using a NN
- Physics-Informed Neural Networks
- Solving Burgers' Equation using PINN
- Solving the Linear Convection Diffusion
- Design of PINN & Tuning
- Conclusion & Next Steps



# Introduction

- Alloys form critical components of engineering structures
- Presence of defects may lead to component failure which can even be fatal
- Examples: interstitials, vacancies, dislocations, precipitates, cracks etc.
- Arise during processes: forming, casting, welding, additive manufacturing
- System traditionally modelled using numerical simulations given the PDE
- What do we do in the absence of a well-determined equation?

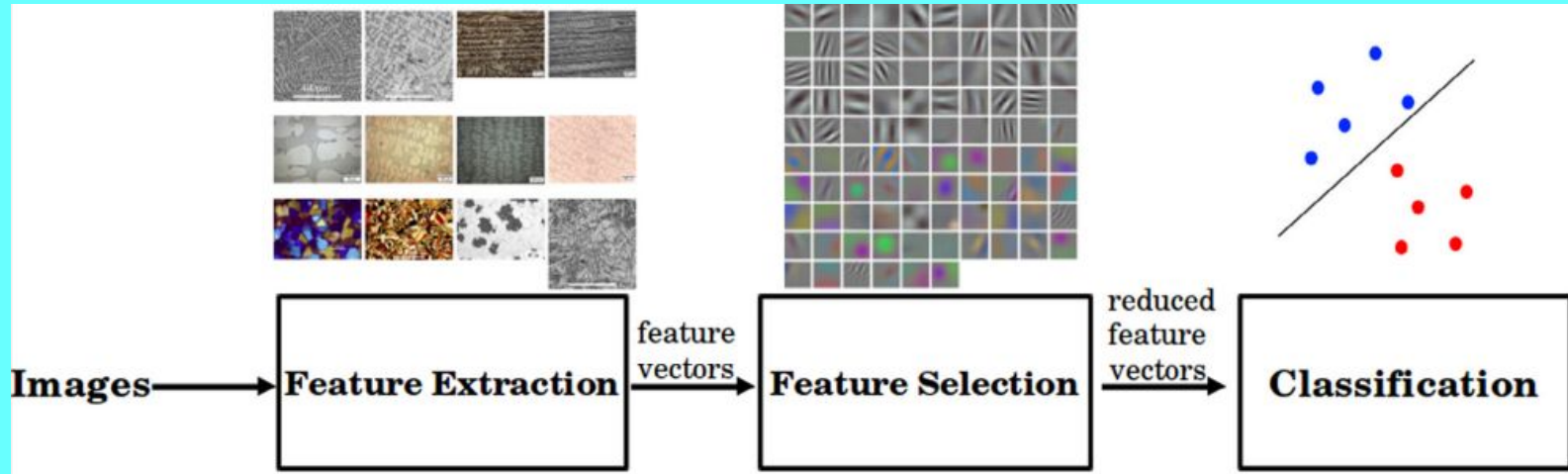


# DDP: Prediction of Defects in Metal Alloys via Combined Data and Physical Models

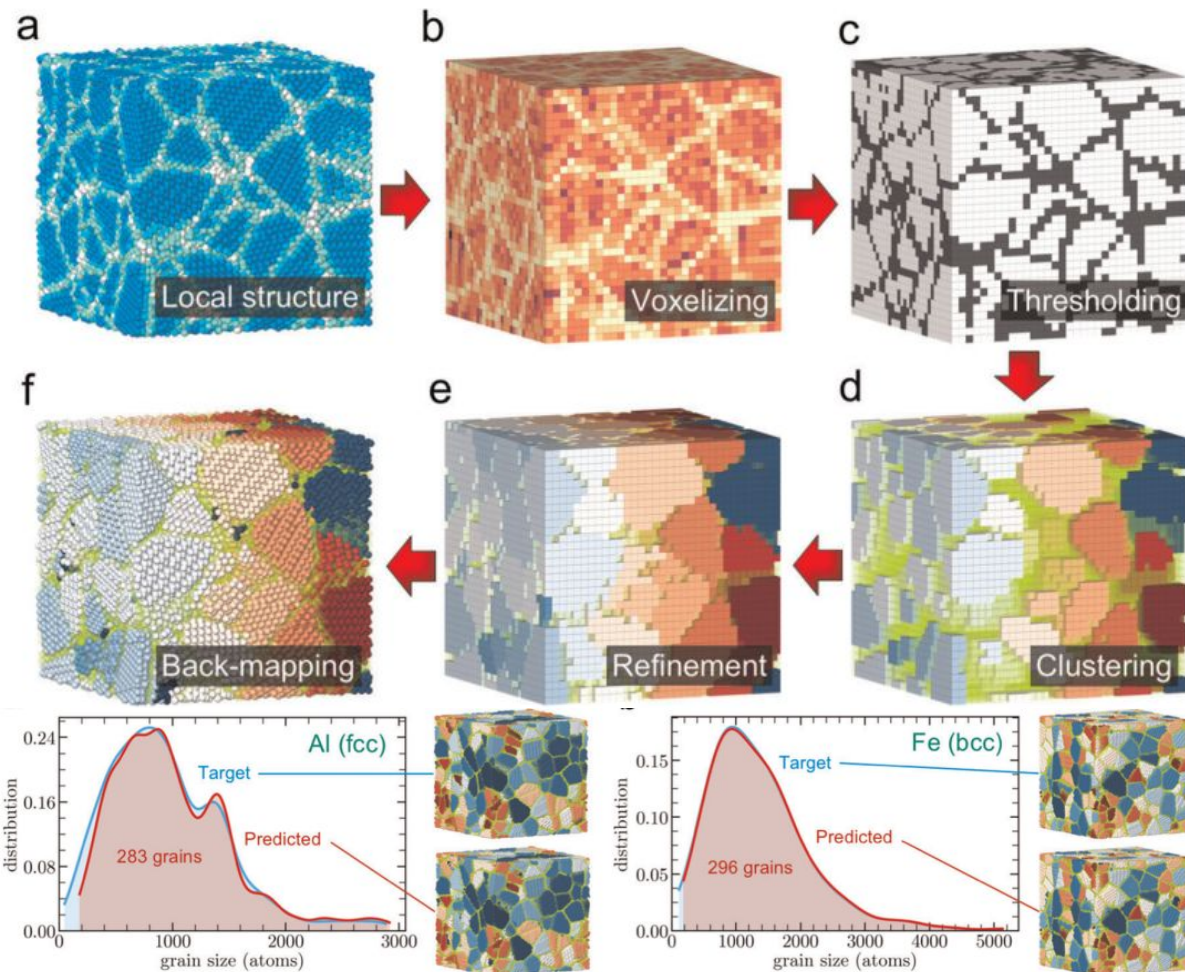
- Data:
    - Recorded Observations
    - Known Fixed Conditions
  - Physics:
    - Process Information
- Data:
    - Image Features
    - Numerical Values
  - Physics:
    - Differential Equation

# Machine Learning for Microstructural Studies

---



- 2 classification tasks:
  1. Detecting the presence of dendritic morphologies in micrographs
  2. Identifying the type of cross-sectional view
- Feature extraction using visual bag of words, texture & shape statistics, pre-trained CNNs
- Dimensionality reduction to obtain a relevant reduced feature set
- Classification using support vector machines, nearest neighbours, and random forest models
- Pre-trained CNNs + SVM yielded best results (accuracies > 90% for both tasks)

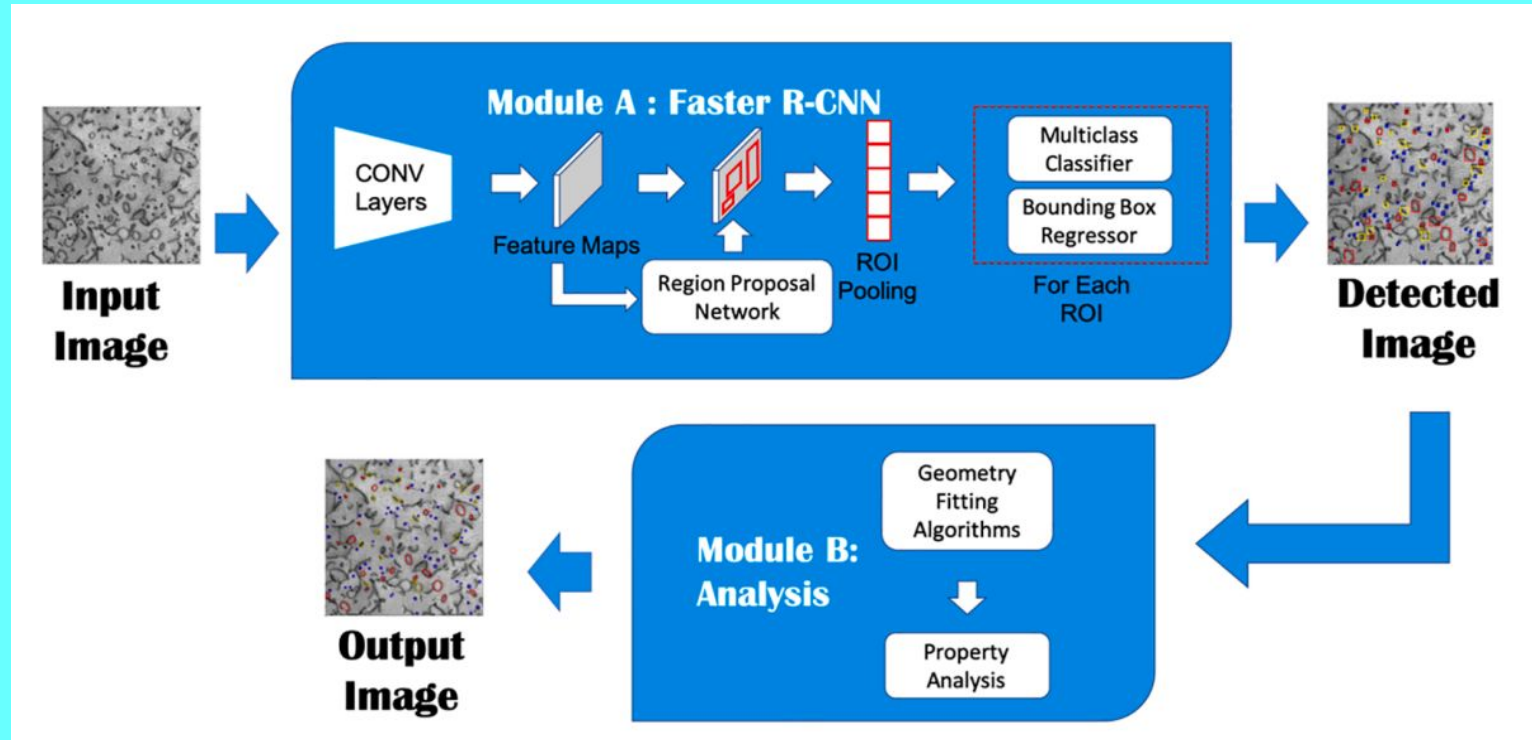


- Autonomous microstructural characterization in 3D samples
- Combines topology classification, image processing & clustering
- Unbiasedly performs quantification of grains and their size distributions
- Efficiently identifies and tracks features such as voids and porosity

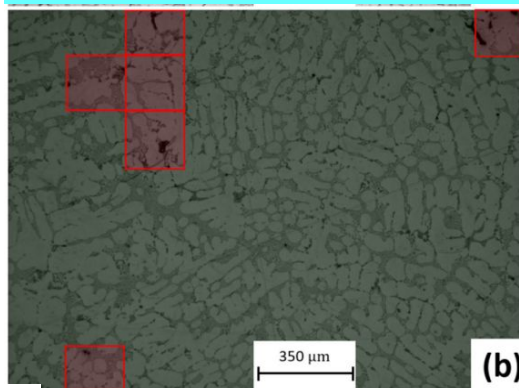
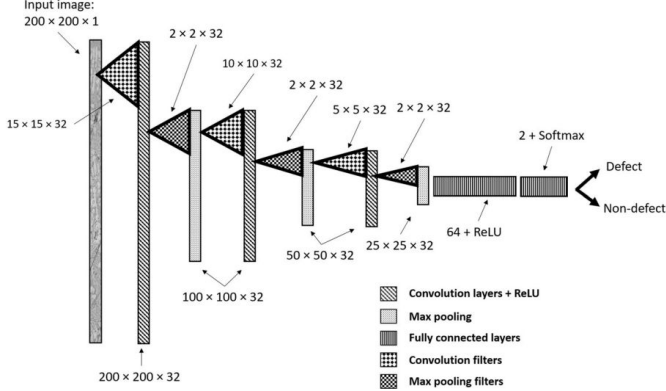
# Examples for Defect Detection

—

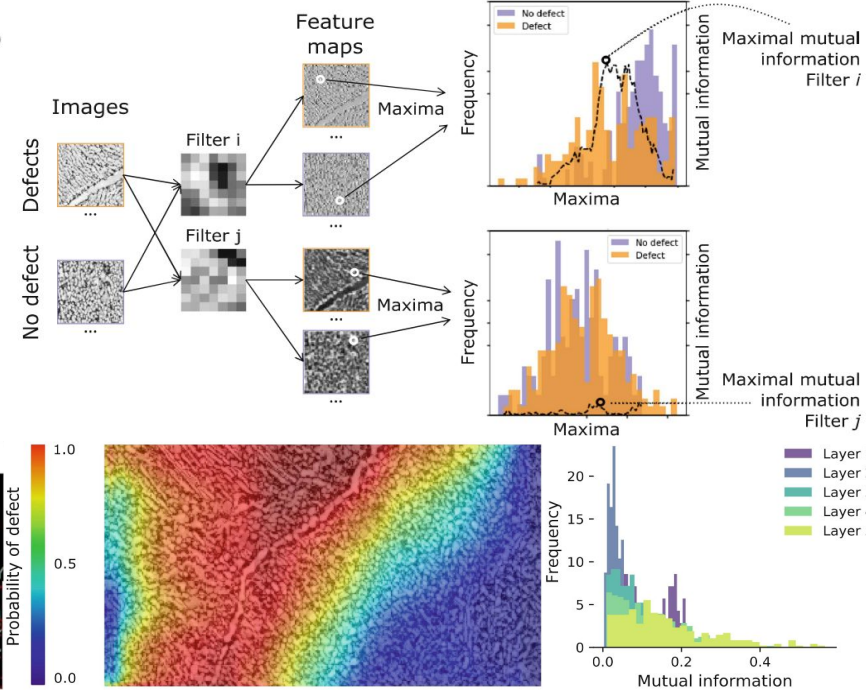
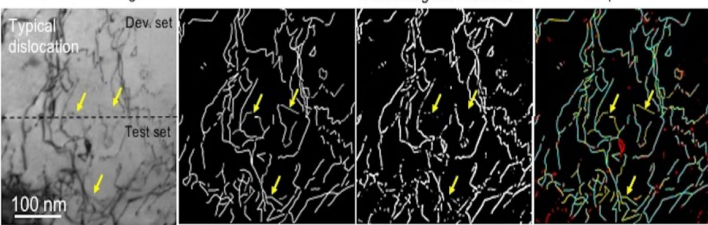
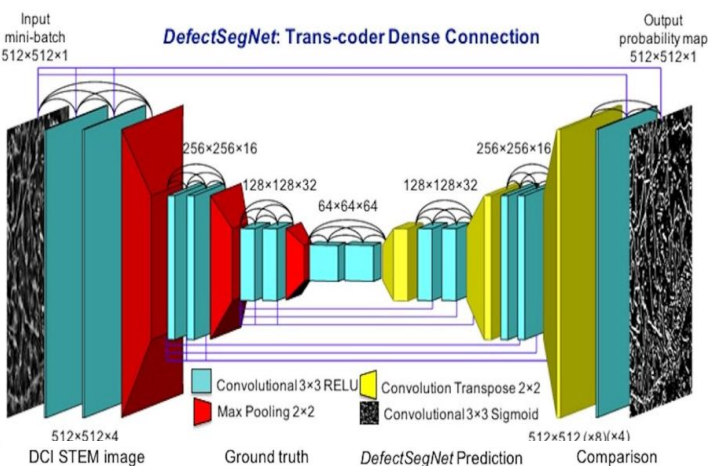




- CNN to extract feature maps from electron microscopy images of steel
- Localisation: Bounding Box Regressor to find the location of defect
- Classification: Multiclass classifier to detect the type of defect
- Geometry Fitting: To identify the shape and size of defect



- Porosity Detection in Al Alloys
- Semantic Segmentation of dislocations, pcpts., and voids
- Mutual Information between filter responses and defects



Nikolic et al., Inter Metalcast, 2022

Boyadjian et al., Lecture Notes in Computer Science, Springer, 2020

G Roberts et al., Sci Rep, 2019

# Prediction of Microstructure Evolution

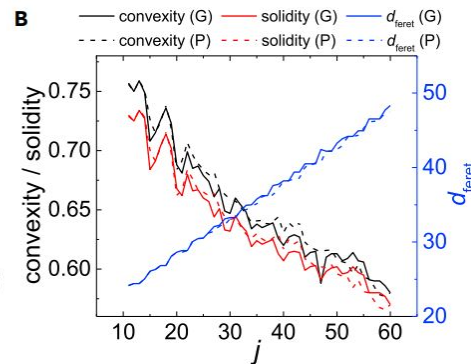
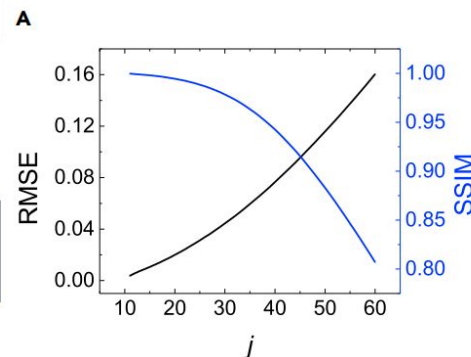
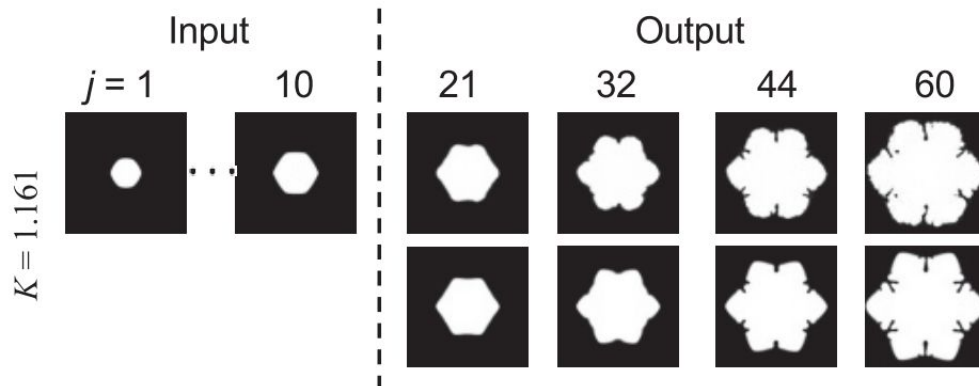
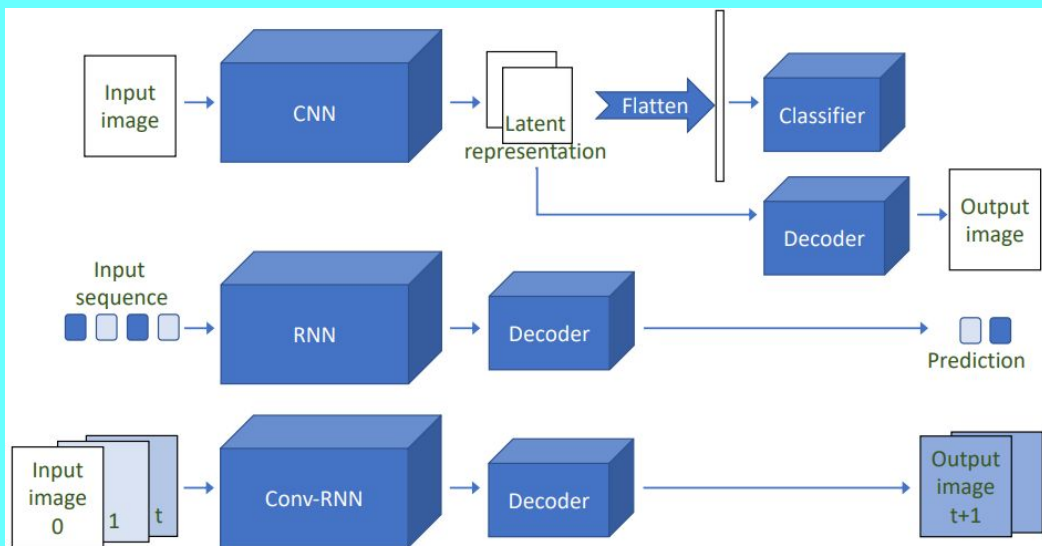
---

# Predicting Dendrite Growth using E3D-LSTM

$$\tau \frac{\partial \phi}{\partial t} = - \frac{\delta F}{\delta \phi}$$

K. Yang et al.,  
Patterns, 2021

$$\frac{\partial T}{\partial t} = \nabla^2 T + K \frac{\partial \phi}{\partial t}$$



# Solving a simple ODE using a NN

—

Let the differential equation be represented by  $G = 0$ . Here  $G$  will be a function of input  $x$ , the solution to be computed  $\psi$  and the derivatives of  $\psi$  (upto order  $n$  say 2):

$$G(x, \psi(x), \nabla\psi(x), \nabla^2\psi(x)) = 0 \quad x \in D, \text{ the domain} \quad (5)$$

If a neural network with weights  $p$  is used to model  $\psi$ , then we train the network (i.e. vary  $p$ ) to satisfy Eq. 5 subject to the given conditions. The trial solution is written as:

$$\psi(x) = A(x) + F(x, NN(x, p)) \quad (6)$$

Here, by construction,  $\psi$  satisfies the conditions given in  $A$  while the objective of neural network training is to minimise the sum of  $G$  values over data points  $x$ .



## Problem

Simple ODE  $u' = f(x) = 2x$

Initial Condition  $u(0) = 1$

Analytical Solution =  $x^2 + 1$

Domain =  $x \in [0, 1]$

## Approach

Define a trial solution  $g$  as follows:

$g(x) = u(0) + x * NN(x)$

$g$  is a solution to  $u$  satisfying IC

Train  $g'$  to be  $u'$  to satisfy ODE

Reference Blog: [Using Neural Networks to solve Ordinary Differential Equations](#)






## NN Architecture

- Input Layer = One Neuron for  $x$
- 2 Hidden Layers each with
  - Number of Neurons = 32
  - Activation Function = Sigmoid
- Output Layer = One Neuron for  $u$





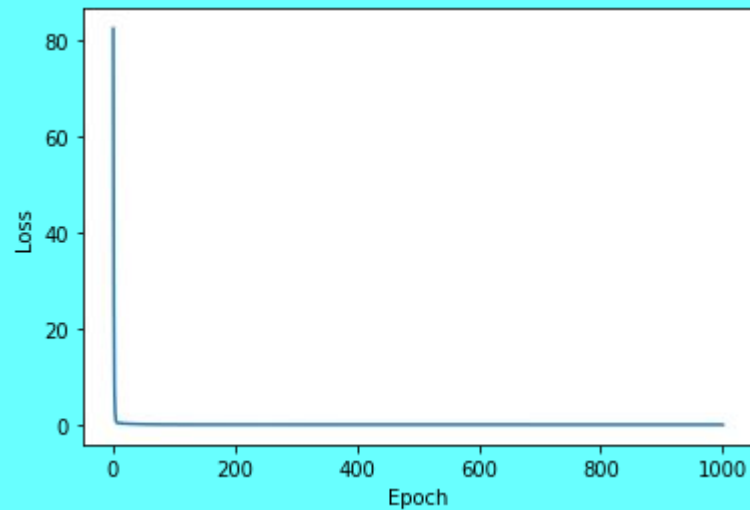
## Training

Data: 10 points (no labels)

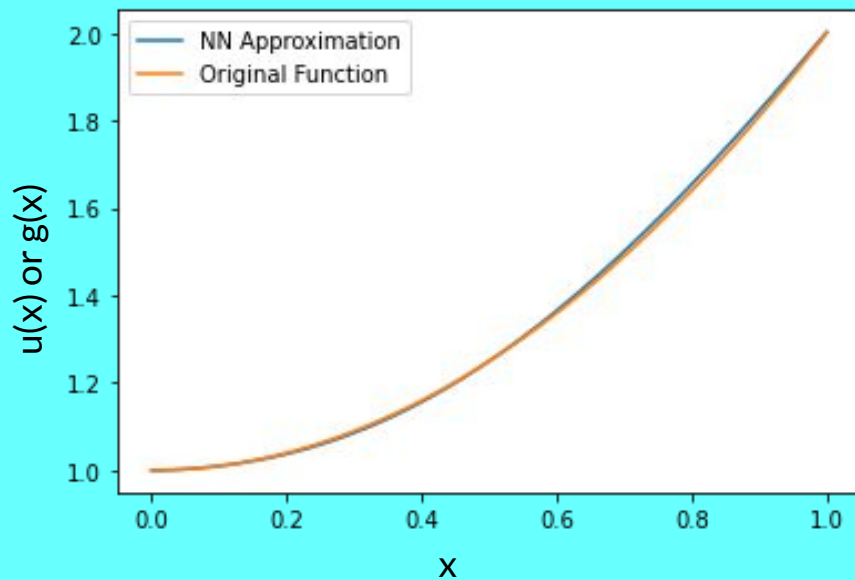
Repeat (for every iteration):

- For each data sample  $x$ 
  - Approximate  $u$  using NN
  - Calculate the gradient of  $g$
  - Calculate Error =  $g'(x) - f(x)$
- Loss = MSE over samples
- Use SGD to minimise loss
- Update NN model weights

# Plots



Final Loss = 0.0022



# Physics-Informed Neural Networks

---

## NSE+HE

$$\nabla \cdot u = 0$$

$$\partial_t u + (u \cdot \nabla) u = -\nabla p + (Re)^{-1} \nabla^2 u + (Ri) \theta$$

$$\partial_t \theta + (u \cdot \nabla) \theta = (Pe)^{-1} \nabla^2 \theta$$

2021



~1300 papers

## NSE

$$\nabla \cdot u = 0$$

$$\partial_t u + (u \cdot \nabla) u = -\nabla p + (Re)^{-1} \nabla^2 u$$

2020



~600 papers

## EE

$$\partial_t u + \beta \partial_x u = 0$$

2019



~100 papers

## SE

$$i \partial_t h + 0.5 \partial_{xx} h + |h|^2 h = 0$$

2018



~30 papers

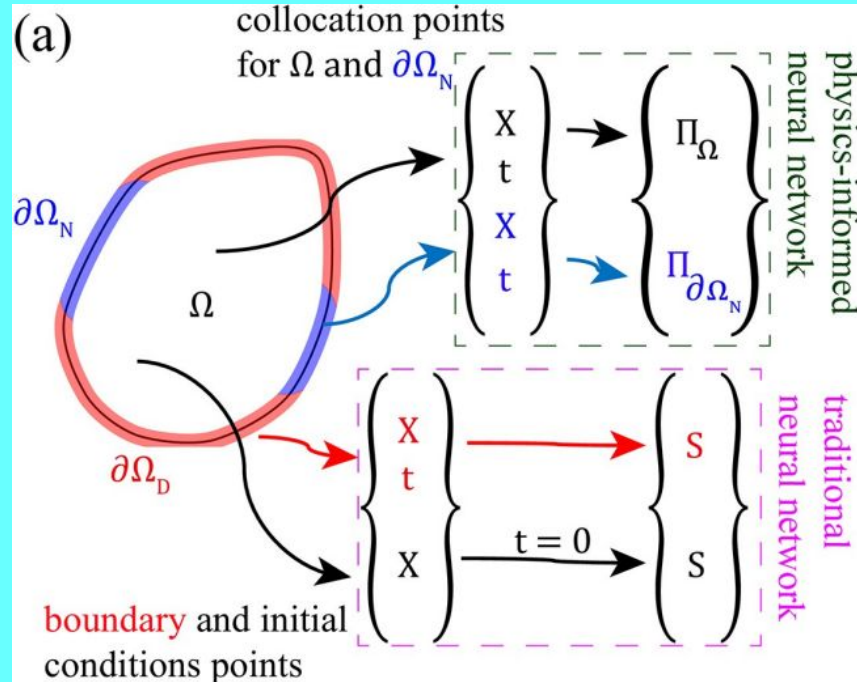
Original: M. Raissi et al., Journal of Computational Physics, 2018

“Given noisy measurements of the system, interest in the solution of two distinct problems:

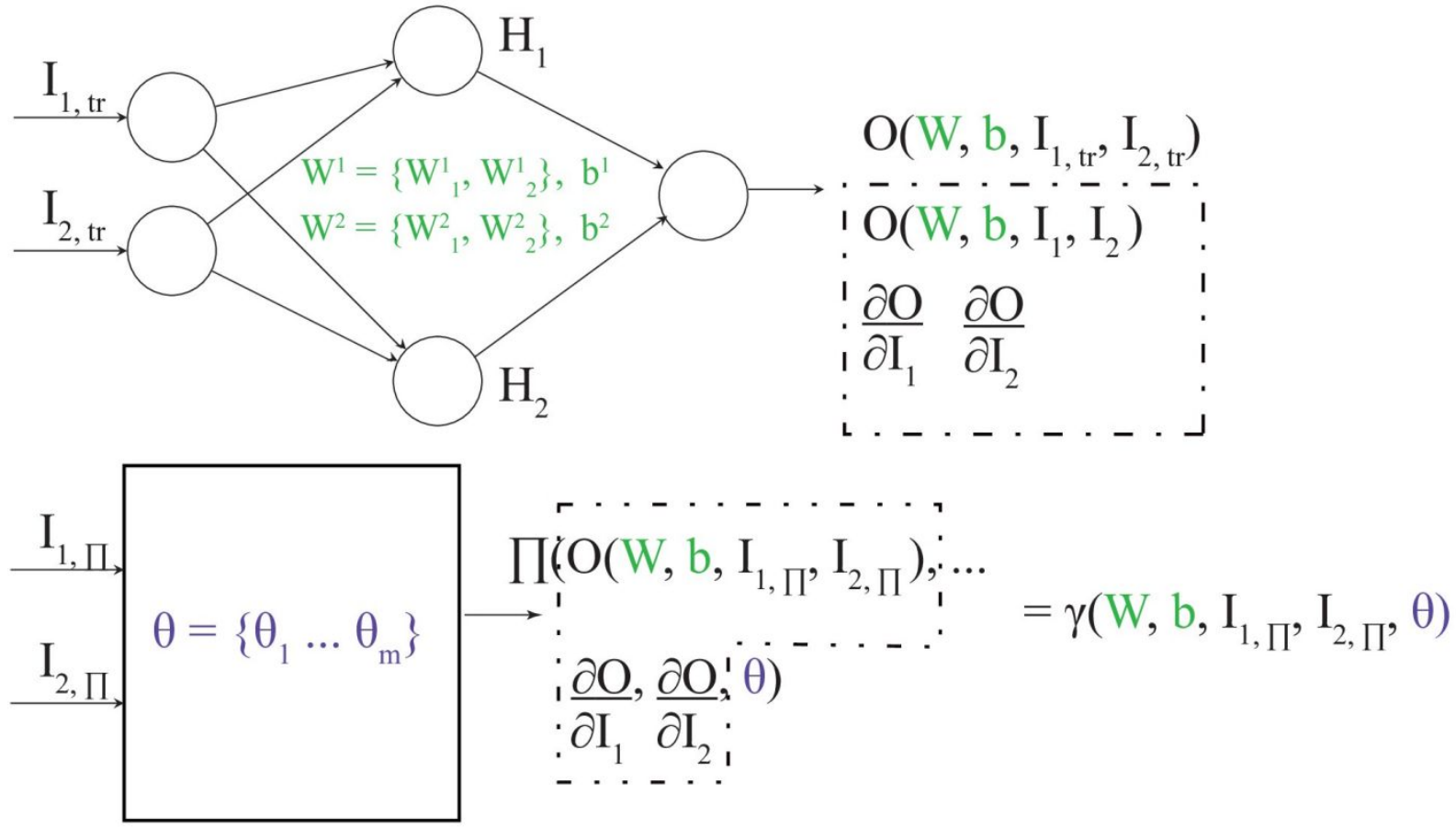
- The first problem is that of inference, filtering and smoothing, or data-driven solutions of partial differential equations which states: given fixed model parameters  $\lambda$  what can be said about the unknown hidden state  $u(t, x)$  of the system?
- The second problem is that of learning, system identification, or data-driven discovery of partial differential equations stating: what are the parameters  $\lambda$  that best describe the observed data?”

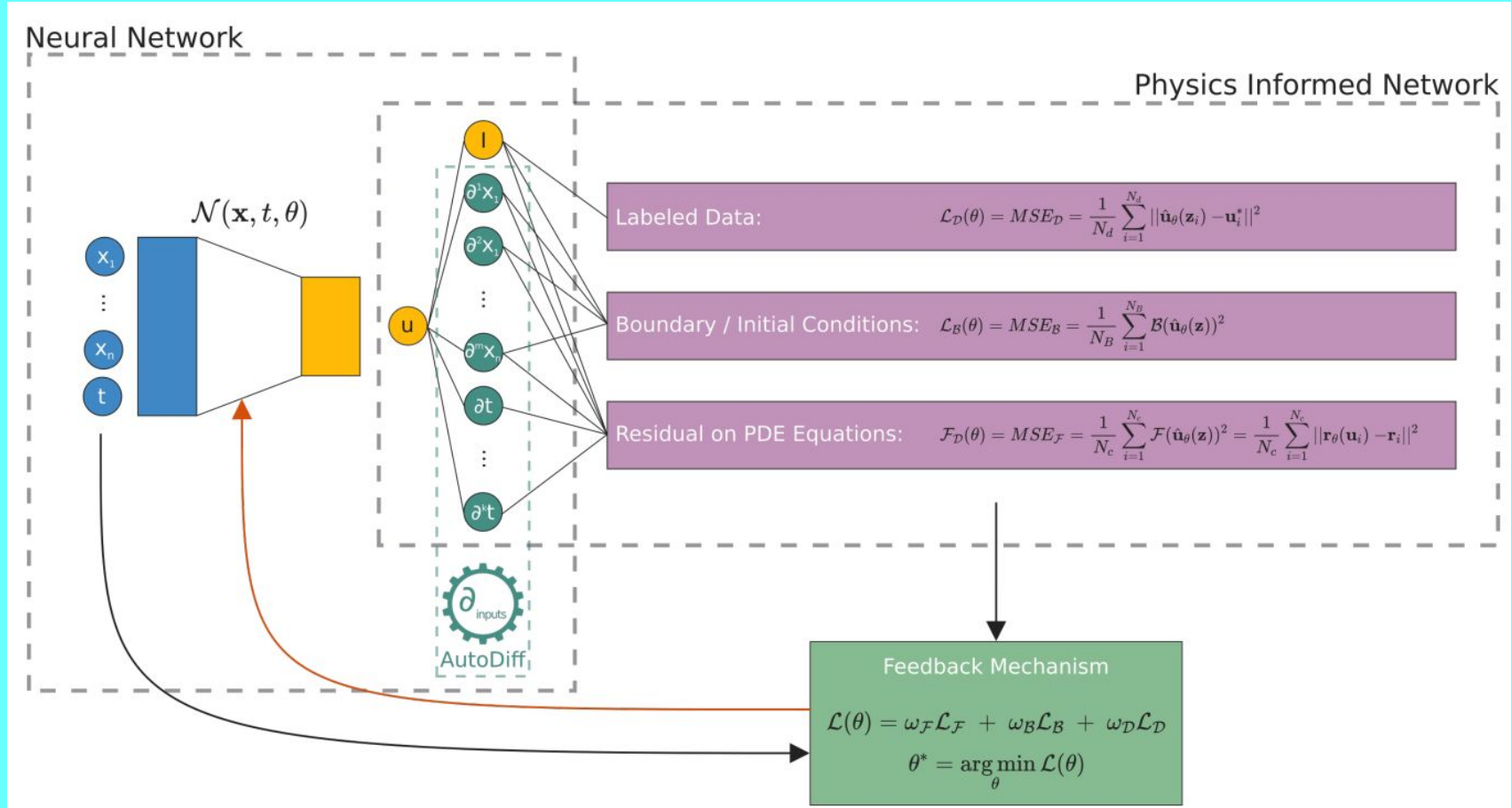
# Data Sampling

T. Kadeethum et al.,  
PLOS ONE, 2020



Random Uniform Sampling is used here





The PINN approach for the solution of the PDE (1) now proceeds by minimization of the loss functional

$$\phi_\theta(X) := \phi_\theta^r(X^r) + \phi_\theta^0(X^0) + \phi_\theta^b(X^b), \quad (3)$$

where  $X$  denotes the collection of training data and the loss function  $\phi_\theta$  contains the following terms:

- the mean squared residual

$$\phi_\theta^r(X^r) := \frac{1}{N_r} \sum_{i=1}^{N_r} \left| r_\theta(t_i^r, x_i^r) \right|^2$$

in a number of collocation points  $X^r := \{(t_i^r, x_i^r)\}_{i=1}^{N_r} \subset (0, T] \times D$ , where  $r_\theta$  is the physics-informed neural network (2),

- the mean squared misfit with respect to the initial and boundary conditions

$$\phi_\theta^0(X^0) := \frac{1}{N_0} \sum_{i=1}^{N_0} \left| u_\theta(t_i^0, x_i^0) - u_0(x_i^0) \right|^2 \quad \text{and} \quad \phi_\theta^b(X^b) := \frac{1}{N_b} \sum_{i=1}^{N_b} \left| u_\theta(t_i^b, x_i^b) - u_b(t_i^b, x_i^b) \right|^2$$

in a number of points  $X^0 := \{(t_i^0, x_i^0)\}_{i=1}^{N_0} \subset \{0\} \times D$  and  $X^b := \{(t_i^b, x_i^b)\}_{i=1}^{N_b} \subset (0, T] \times \partial D$ , where  $u_\theta$  is the neural network approximation of the solution  $u : [0, T] \times D \rightarrow \mathbb{R}$ .

We note that the training data  $X$  consists entirely of time-space coordinates. Moreover, individual weighting of each loss term in (3) may help improve the convergence of the scheme, see for example [163].



$$ih_t + 0.5h_{xx} + |h|^2h = 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2],$$

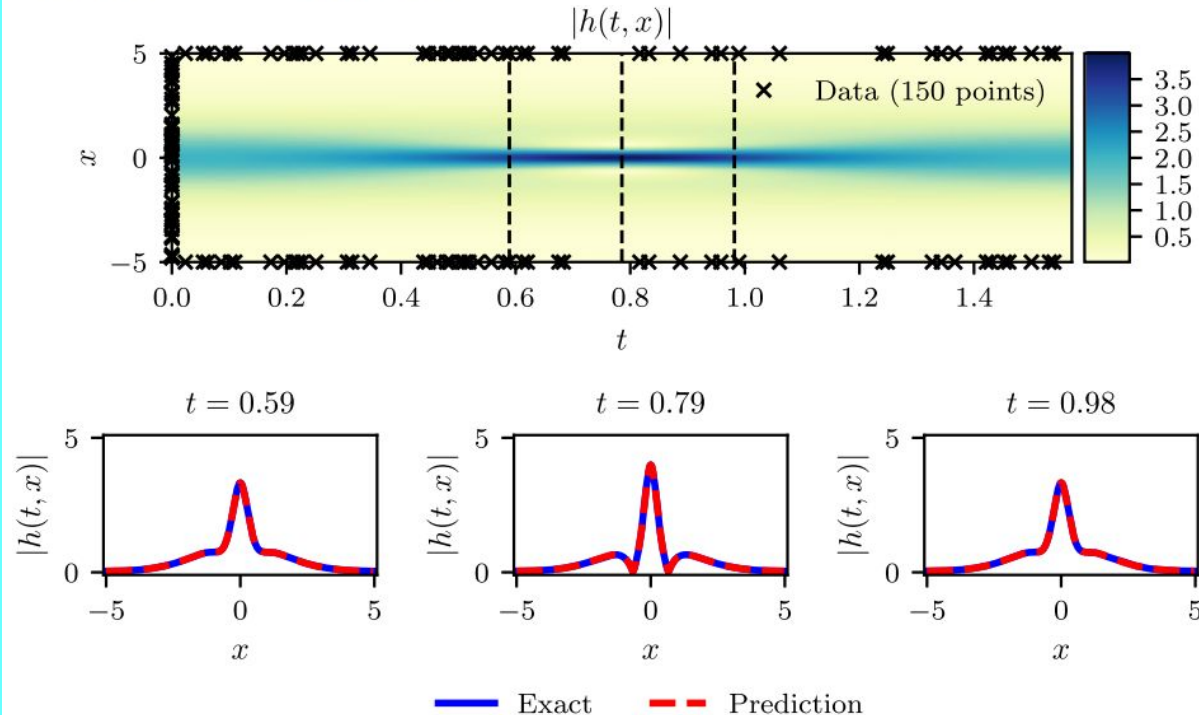
$$h(0, x) = 2 \operatorname{sech}(x),$$

$$h(t, -5) = h(t, 5),$$

$$h_x(t, -5) = h_x(t, 5),$$

## Schrodinger's Equation

M. Raissi et al., Journal of  
Computational Physics, 2018



# Burgers' Equation

---



## Problem

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

Valid in Domain:  $x \in [-1, 1]$  and  $t \in (0, 1]$

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

Boundary Condition:  $u(t, -1) = u(t, 1) = 0$

J. Blechschmidt & O.G. Ernst, *GAMM*, 2021



## NN Architecture

- Input Layer = 2 Neurons for  $x, t$
- 8 Hidden Layers each with
  - Number of Neurons = 20
  - Activation Function = Tanh
- Output Layer = One Neuron for  $u$



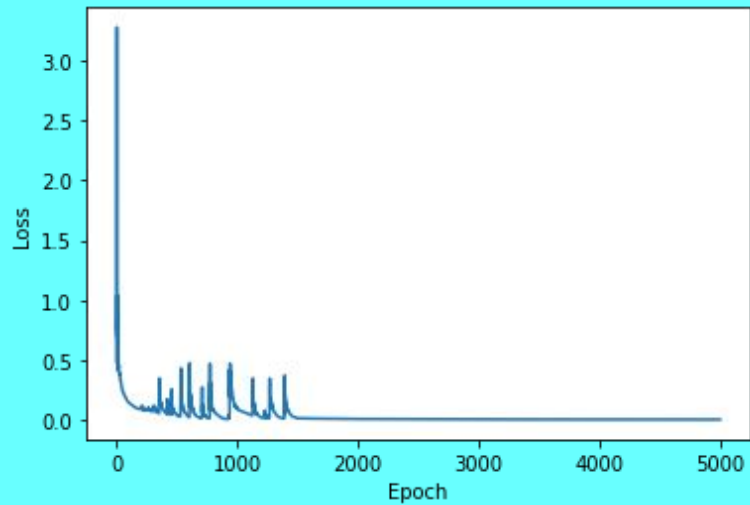
# Training

Repeat (for every iteration):

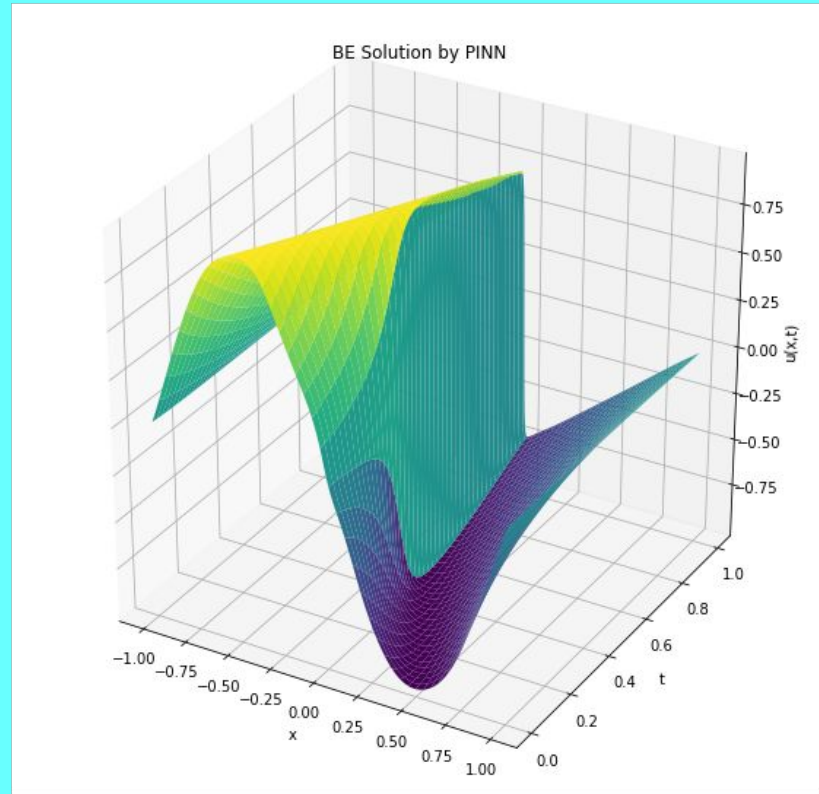
- For each data sample  $x$ , approximate  $u$  using PINN
- Loss = MSE over boundary & initial data + PDE residual
- Use Adam to minimise loss
- Update NN model weights

Data Points: 50 initial & 50 boundary (labeled) + 10,000 collocation (unlabeled)

# Plots



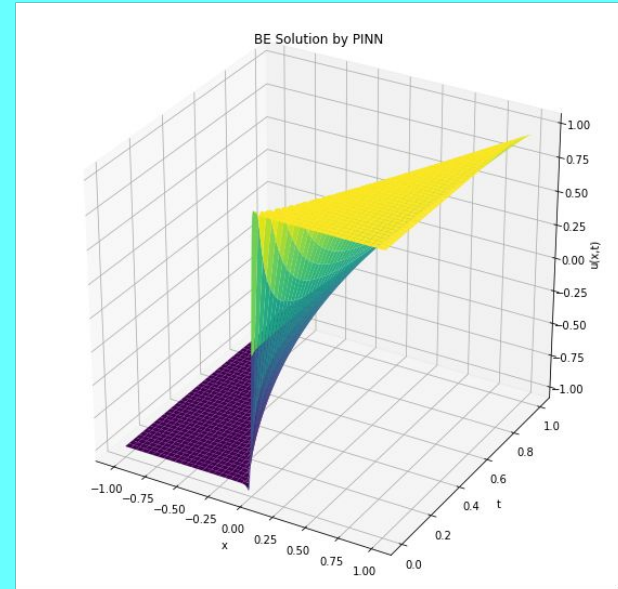
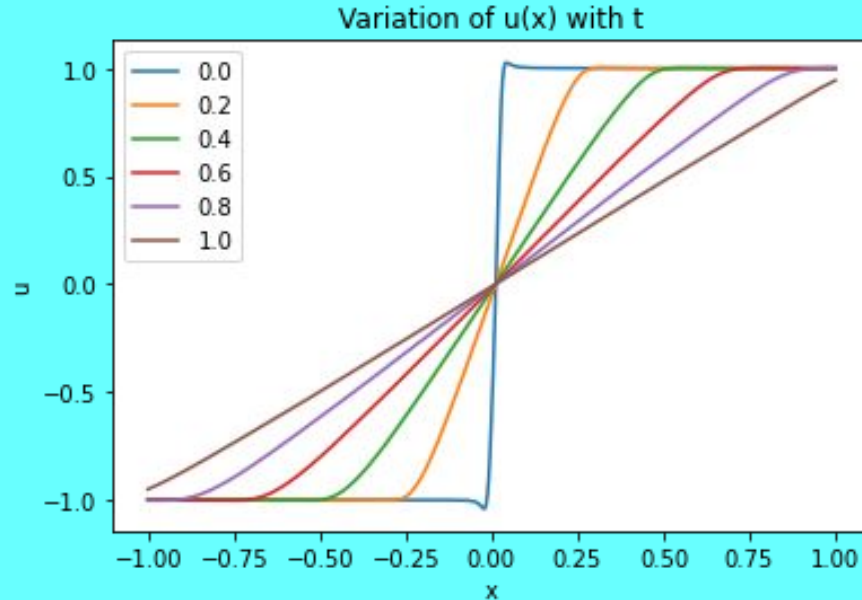
Final Loss = 0.00051



## Different Condition

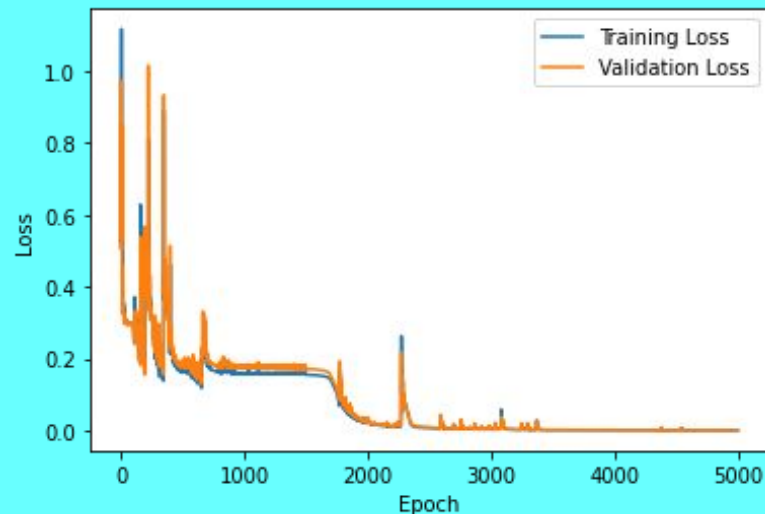
Initial Condition:  $u(0, x) = \text{sign}(x)$

Boundary Condition:  $u(t, \pm 1) = \pm 1$



## Modifications

- Added Regularization & Dropout
- Added Validation Data (1:5 ratio with train)
- Validation & training losses align with each other over the epochs (to end with  $6.57\text{E-}04$  &  $6.89\text{E-}04$ ): seems to suggest no overfitting





# Predicting Evolution of PINN Predicted $u$

—

# Data

Data		Input				Output		
		$u(x, t)$	$t = 1$	. . .	$t = 250$	$t = 251$	. . .	$t = 500$
Train	$x = 1$							
	$\vdots$							
	$x = 250$							
	$x = 251$							
Test	$\vdots$							
	$x = 500$							

NN Architecture:

- LSTM Layer = 32 LSTM units
- Dense Layer = 250 neurons

Training:

Epochs = 50

Batch size = 10

Validation split = 0.3

NN Architecture:

- LSTM Layer = 32 LSTM units
- Dense Layer = 250 neurons

Training:

Epochs = 50

Batch size = 10

Validation split = 0.3

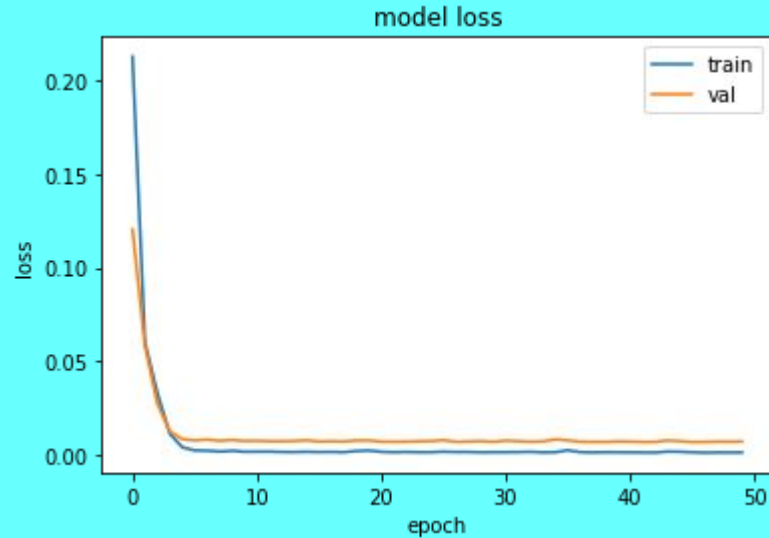


# Training

Repeat (for every iteration):

- For each data sample  $x$  in the batch
  - Use half of PINN  $u(x)$  time-series as input
  - Predict other half time-series using LSTM
- Calculate Losses = MSE between LSTM predictions and PINN labels
- Use Adam to minimise training loss
- Update LSTM model weights

## Loss vs Epochs



Final Metrics:

Train MSE Loss = 0.0011

Val MSE Loss = 0.0070

Test MSE = 0.0019

# Linear Convection Diffusion Equation

---



## Problem

PDE  $u_t + c * u_x - v * u_{xx} = 0$  with  $c = v = 1$

Valid in Domain:  $x \in [-1, 1]$  and  $t \in (0, 1]$

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

Boundary Condition:  $u(t, -1) = u(t, 1) = 0$

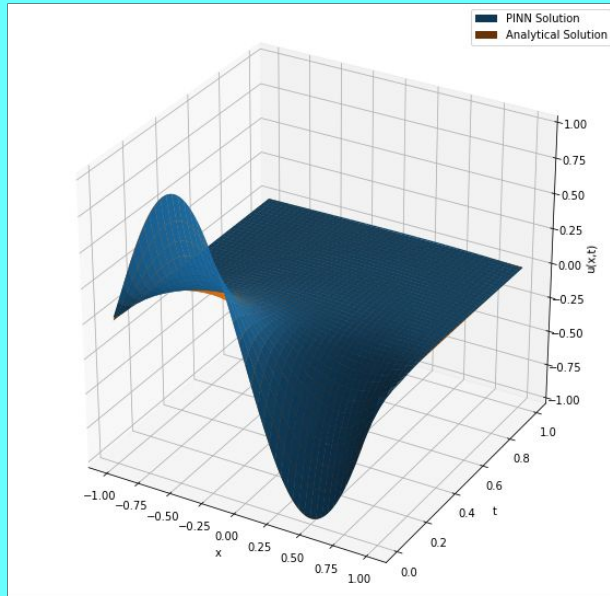
Analytical Solution:  $u(x, t) = -e^{-\pi^2 t} \sin(\pi x)$

Only PDE parameters are changed, rest approach same

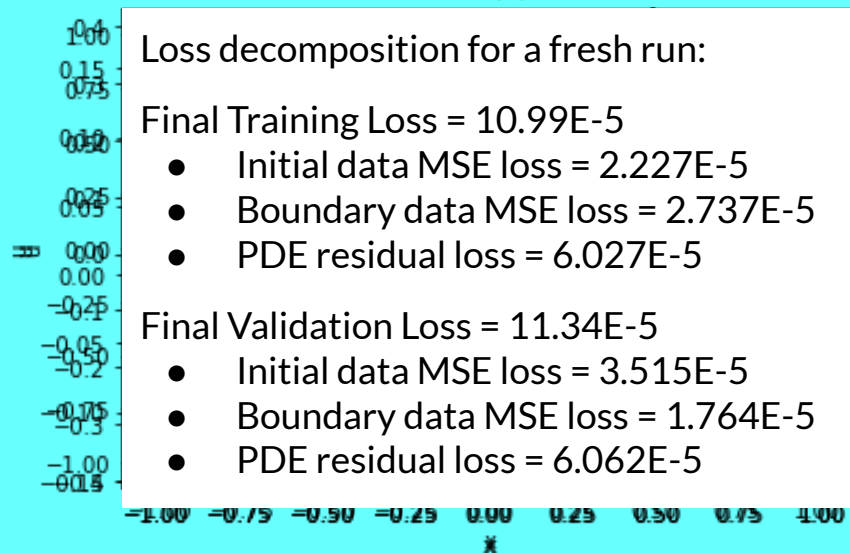
# Solution

Final PINN Loss = 0.00014

MAE wrt Analytical = 0.0015



Variation of  $u$  vs  $x$  every 0.1s



# Designing PINN

---



# Deciding the architecture

---

- Most PINNs are single feed-forward neural networks
- Wide networks (say 100 neurons per layer as in original)
- Deep networks (say  $>10$  hidden layers)
- Shallow networks (say  $<5$  hidden layers)
- Sparse networks (Ramabathiran & Ramachandran, JCP, 2021)
- Multiple fully-connected networks
- Regularization & Dropout to avoid overfitting
- Activations: Relu, Sigmoid, Swish, Tanh (popular)
- Single fully-connected feedforward networks considered for tuning



## Loss & Optimisation

- Soft-boundary constraints: NN fits data gradually, conditions are satisfied approximately
- Hard-boundary constraint: Conditions are satisfied exactly by encoding within NN
- Weights for different loss terms are hyperparameters, help vary fidelity of model
- Optimiser: Adam (easier functionality, faster & lower loss compared to), SGD, L-BFGS
- A decreasing learning rate is popular for faster convergence away from minima
- Stopping criteria: a fixed number of iterations (here) or a lower bound on the loss

# Error Decomposition

---

- **Approximation Error:**
  - Associated with the neural network approximation
  - Measured wrt the available exact values
  - Here, MSE loss on labelled data recorded during training
- **Optimization Error:**
  - Loss function is non-convex, so a local minimum is obtained
  - Error wrt the global minimum can't be determined
- **Generalisation Error:**
  - Measured in the predictions of the trained network on unseen test data
  - Here, MAE evaluated on test data (domain grid)
- **Bounds need to be established on these errors**



## Bayesian Optimisation

- A sequential design strategy for the global optimization of black-box functions
- Constructs a posterior distribution for the function, which is improved as the observations grow resulting in a restricted parameter search space each time
- Provides flexibility to the user to decide how to balance random exploration with the exploitation of learnings from the results of prior iterations
- Used for hyperparameter tuning of PINNs for solving Helmholtz problems where they concluded that wide shallow networks with sin activation perform better

# Hyperparameters & Tuning

Hyperparameter	Symbol	Range/Set of Values
No of Hidden Layers	h	Integer in [1, 20]
No of Neurons	n	Multiplier of 4 in [4, 256]
Dropout Fraction	d	{0, 0.1, 0.2, 0.3, 0.4, 0.5}
Activation Function	a	{relu, sigmoid, tanh, swish}
Weight Initialiser	wi	If ReLU then He else Glorot {normal, uniform}
L2 Weight Penalty	wr	{0, 10E-1, 10E-2, 10E-3, 10E-4, 10E-5, 10E-6}

PINN as the Black Box Function and Negative of Training Loss to be maximised

No of Iterations = 10 for Bayesian Optimization + 5 for random exploration

The least training loss for optimal hyperparameters combination = 3.354E-5

# Addressing: How to design NN given an equation?

Equation	Order	No of Hidden Layers	Activation	Dropout	Weight Initialiser	Weight Penalty
$u_t + u_x - u_{xx} = 0$	2	7	Swish	0.5	Glorot Normal	0
$u_t - u_{xx} = 0$	2	5	Swish	0	Glorot Normal	1.00E-06
$u_t = 0$	1	4	Swish	0	Glorot Uniform	0
$u_t + uu_x - u_{xx} = 0$	2	14	Swish	0.5	Glorot Normal	0
$u_t + uu_x = 0$	1	7	Swish	0.5	Glorot Normal	0

1. No of hidden layers increase with complexity
2. Swish activation results in a higher performance
3. Dropout gets enabled for deeper networks
4. Regularisation is applied for high number of weights

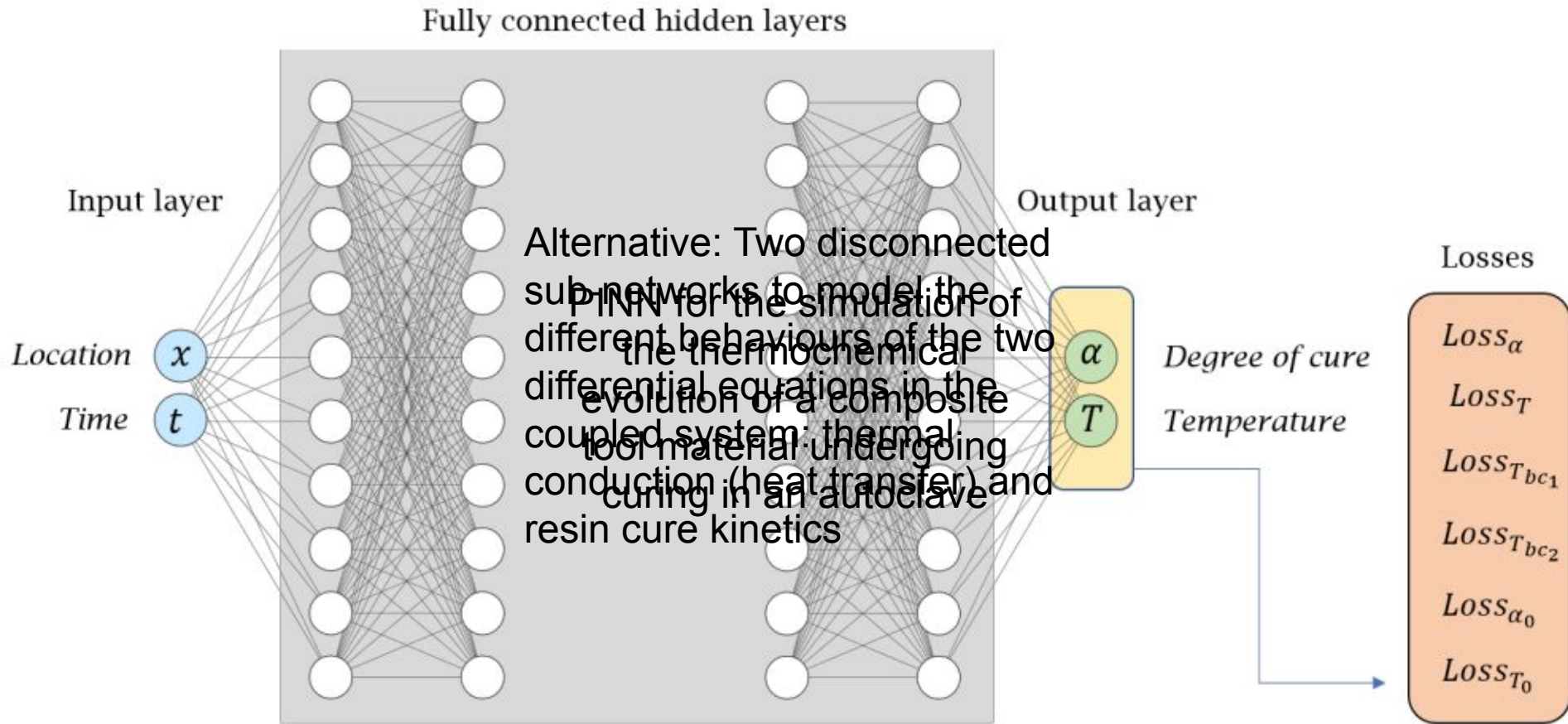
# Solving Coupled/Complex Equations



“PINNs for Solving Coupled Stokes-Darcy Equation”

## Strategies:

1. Loss terms corresponding to the different equations are simply added  
Doesn't work well for too small PDE parameters or discontinuity at interface
2. Designing a weighted loss function ensures that just because of the loss magnitudes, only certain specific terms aren't focussed upon for training
3. Use parallel network architecture where each network caters to a different sub-region of the solution: Helps improve interface discontinuity issues
4. Have a layer-wise locally defined activation function so that the layers have different activations to capture the different aspects of complex problems

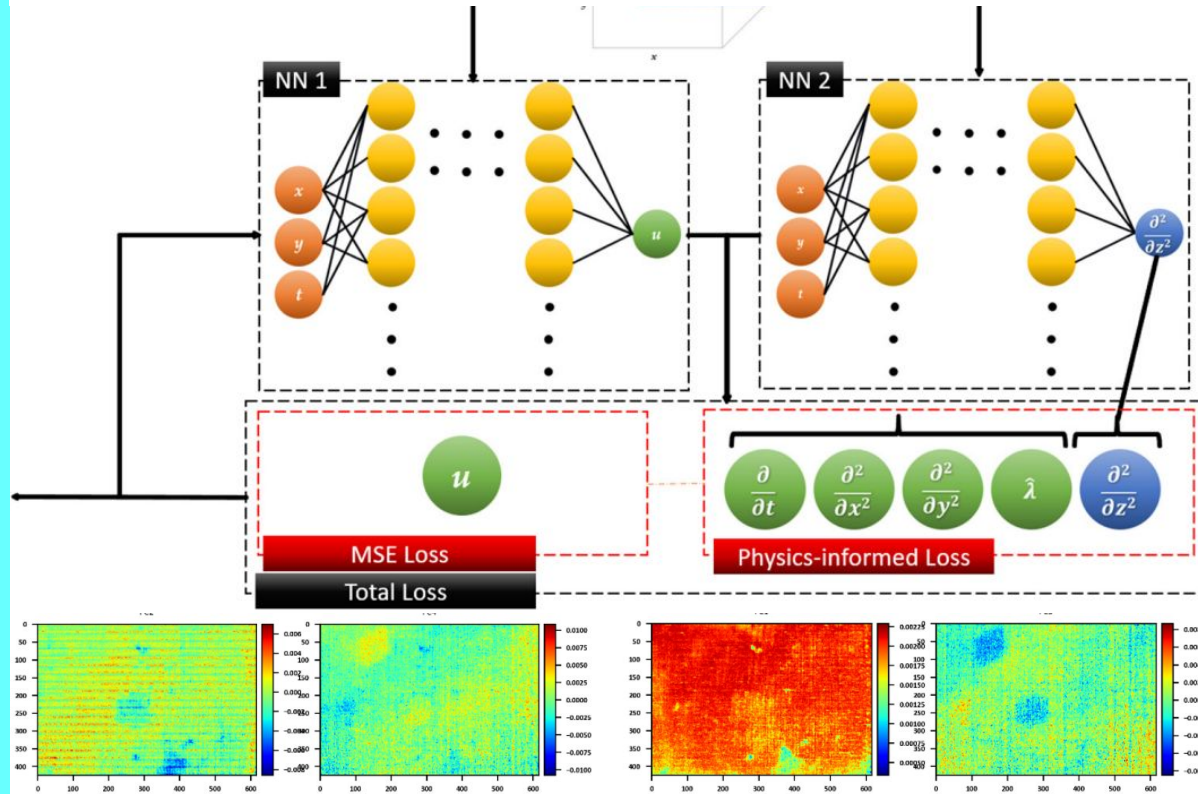


(\*) For prescribed temperature boundary condition

(\*\*) For convective boundary condition



# PINN Method for Defect Identification in Polymer Composites Based on Pulsed Thermography



# Solving Porosity Model as a Moving Interface Problem

$$\frac{\partial u_i}{\partial t} = k_i \frac{\partial^2 u_i}{\partial x^2}, \quad (x, t) \in \Omega_i, \quad i = 1, 2$$

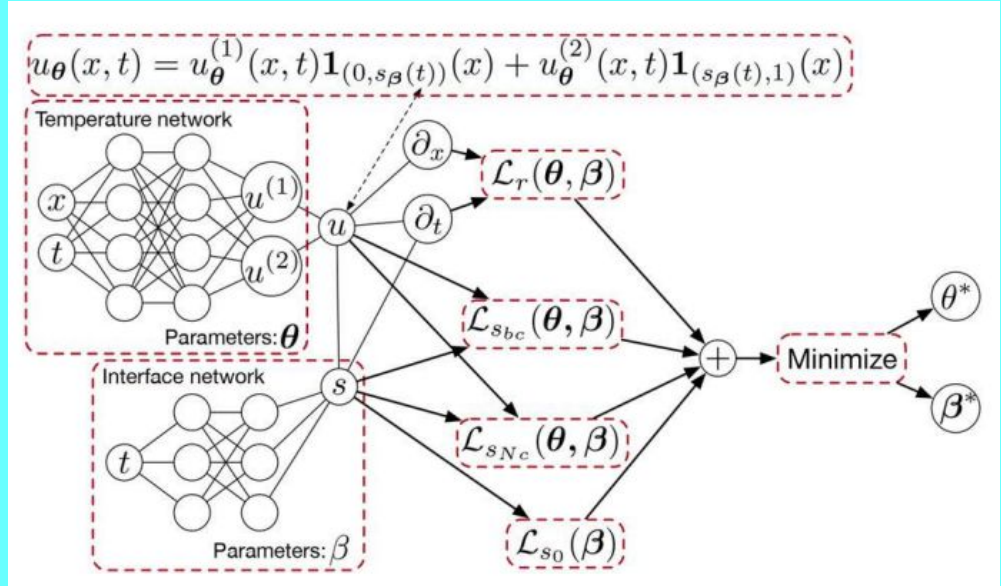
$$\Omega_1 = \{(x, t) \in \Omega : 0 < x < s(t), t \in (0, T]\}$$

$$\Omega_2 = \{(x, t) \in \Omega : s(t) < x < L, t \in (0, T]\}$$

$$u_1(s(t), t) = u_2(s(t), t) = u^*, \quad t \in [0, 1]$$

$$s'(t) = \alpha_1 \frac{\partial u_1}{\partial x}(s(t), t) + \alpha_2 \frac{\partial u_2}{\partial x}(s(t), t), \quad t \in [0, 1]$$

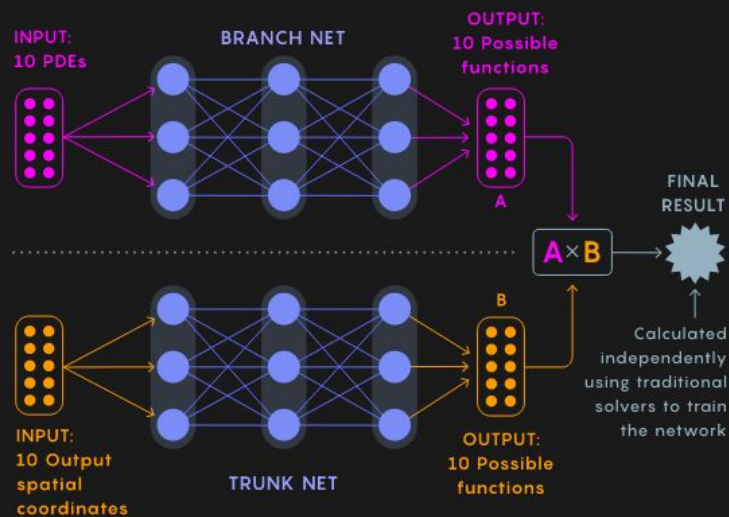
$$s(0) = s_0$$



# Generalisability

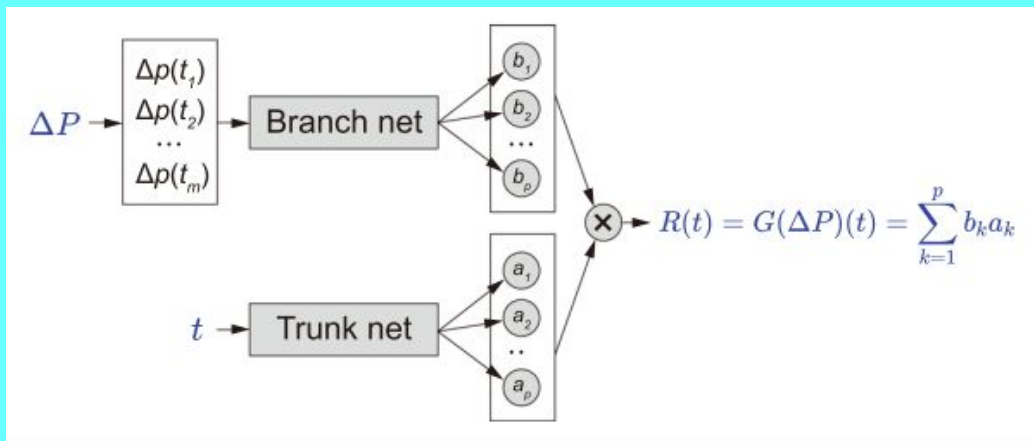
## DeepONet

The DeepONet architecture is split into two parallel networks: a **branch** that approximates functions related to input, and a **trunk** that does the same for the output. By combining the outputs from both branches, it can learn to approximate an operator that effectively solves partial differential equations (PDEs) much faster than traditional solvers.



Lin et al., Journal of Chemical Physics, 2021

Used for predicting multiscale bubble growth dynamics



**FIG. 1.** Schematic of the DeepONet architecture. The branch network takes the liquid pressure change at different times (from  $t_1$  to  $t_m$ ) as input and outputs  $[b_1, b_2, \dots, b_p]^T$ . The trunk network takes time  $t$  as input and outputs  $[a_1, a_2, \dots, a_p]^T$ . Then, the two vectors are merged together to give the prediction of bubble radius  $R(t) = G(\Delta P)(t) = \sum_{k=1}^p (b_k a_k)$ . Both the branch and trunk are feedforward neural networks (FNNs).



## Advantages of PINNs

- Highly fast and accurate prediction
- Highly stable and efficient solving of complex problems
- Work well with incomplete models and imperfect data
- Strong generalisation in small data regime
- Helpful in understanding deep learning



## Challenges

- Design of a PINN is nuanced: varies with the physical problem
- Needs to be retrained for any modifications in the underlying rules
- Establishing their convergence to the correct solution: approximation error has to tend to zero and generalisation error should be very low
- Tackling overflow issues during bayesian optimization
- Finding the right values for different hyperparameters
- Generalisation of its framework to solve a family of PDEs



## Next Steps

- Applying PINN to solve coupled equations and complex systems
- Extending the interpretation experiment to several equations
- Exploring frameworks to solve any general equation or a family of PDEs
- Quantifying the benefits of PINN over numerical approaches
- Eventually, use the best network to build the defect prediction model