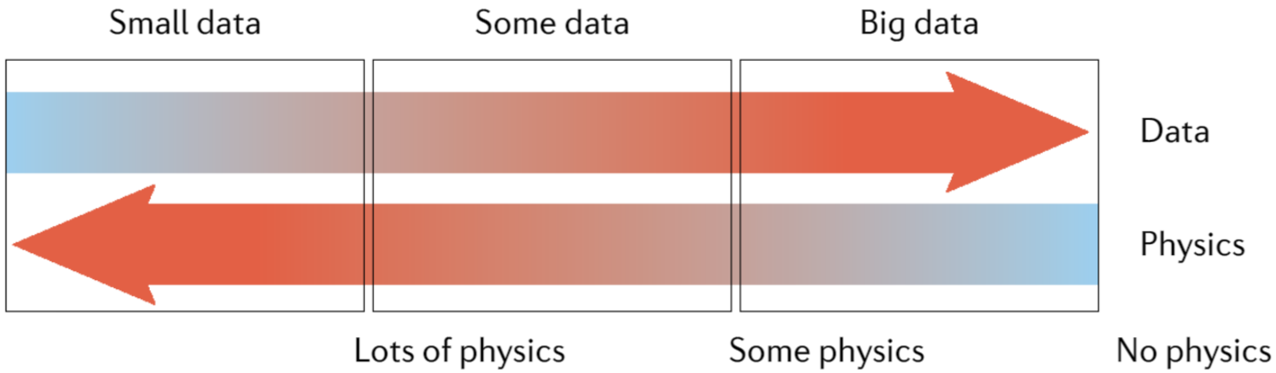
Physics Inspired AI

Shawn Rosofsky
UIUC Department of Physics
NCSA Gravity group
ALCF AI for Science Training Series

Introduction

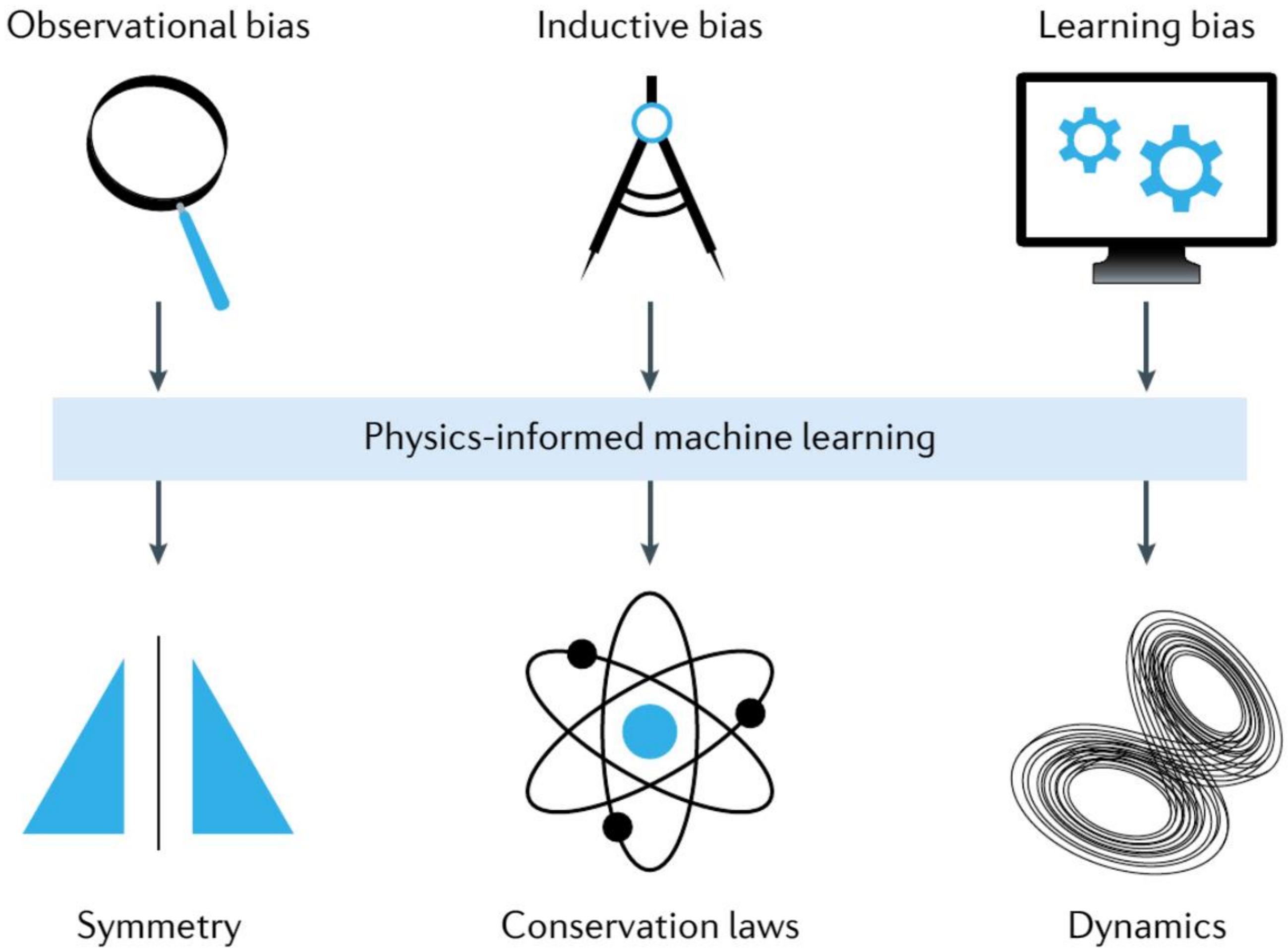
- Deep learning has expanded in recent years
    - Availability of big data
    - Improvements in hardware (GPU/TPU)
    - Open source libraries
- Data is not always available in all cases
    - Expensive
    - Time consuming
- But may have good physical understanding of system
    - Scientific experiments
    - Hardware design
- **Solution: Include physics of problem into neural networks to train with much less data**



Karniadakis et al 2021

How Can Physics Help?

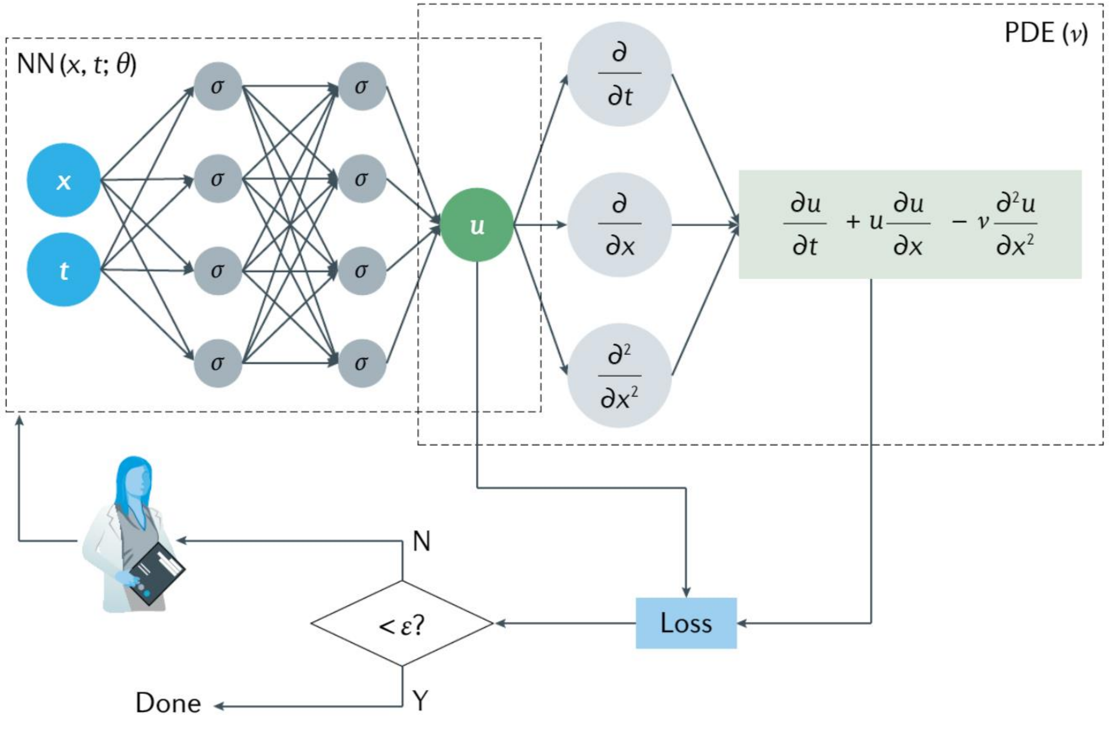
- Neural networks retain bias of its training data
    - Gender bias in NLP ([Sun et al 2019](#))
    - Racial and age bias in image recognition ([Nagpal et al 2019](#))
- Some bias can be removed
    - Data augmentation
    - Increase data quantity
- Physics knowledge can remove data biases system with well understood physics
    - Symmetries
    - Conservation laws
    - Partial differential equations (PDEs)



Karniadakis et al 2021

How to Encode Physics into Neural Networks

- Add known physical laws into loss function
  - Introduces soft constraints
  - Improves with more training
- Encode derivatives by employing automatic differentiation
  - Accurate
  - Fast
- Weight data and physical laws to improve training
- May need second derivatives → No ReLU activation function
- Normalize equations



$$\mathcal{L} = w_{\text{data}}\mathcal{L}_{\text{data}} + w_{\text{PDE}}\mathcal{L}_{\text{PDE}},$$

where

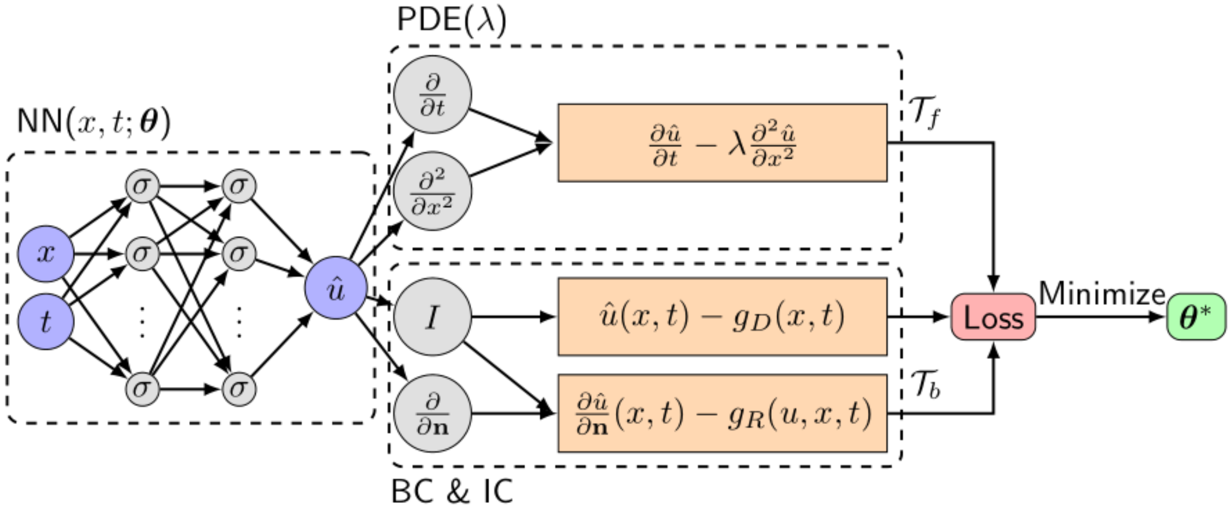
$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}}(u(x_i, t_i) - u_i)^2 \quad \text{and}$$

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}}\sum_{j=1}^{N_{\text{PDE}}}\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}\right)^2\Big|_{(x_j,t_j)}$$

[Karniadakis et al 2021]

Traditional Physics Informed Neural Networks (PINNs)

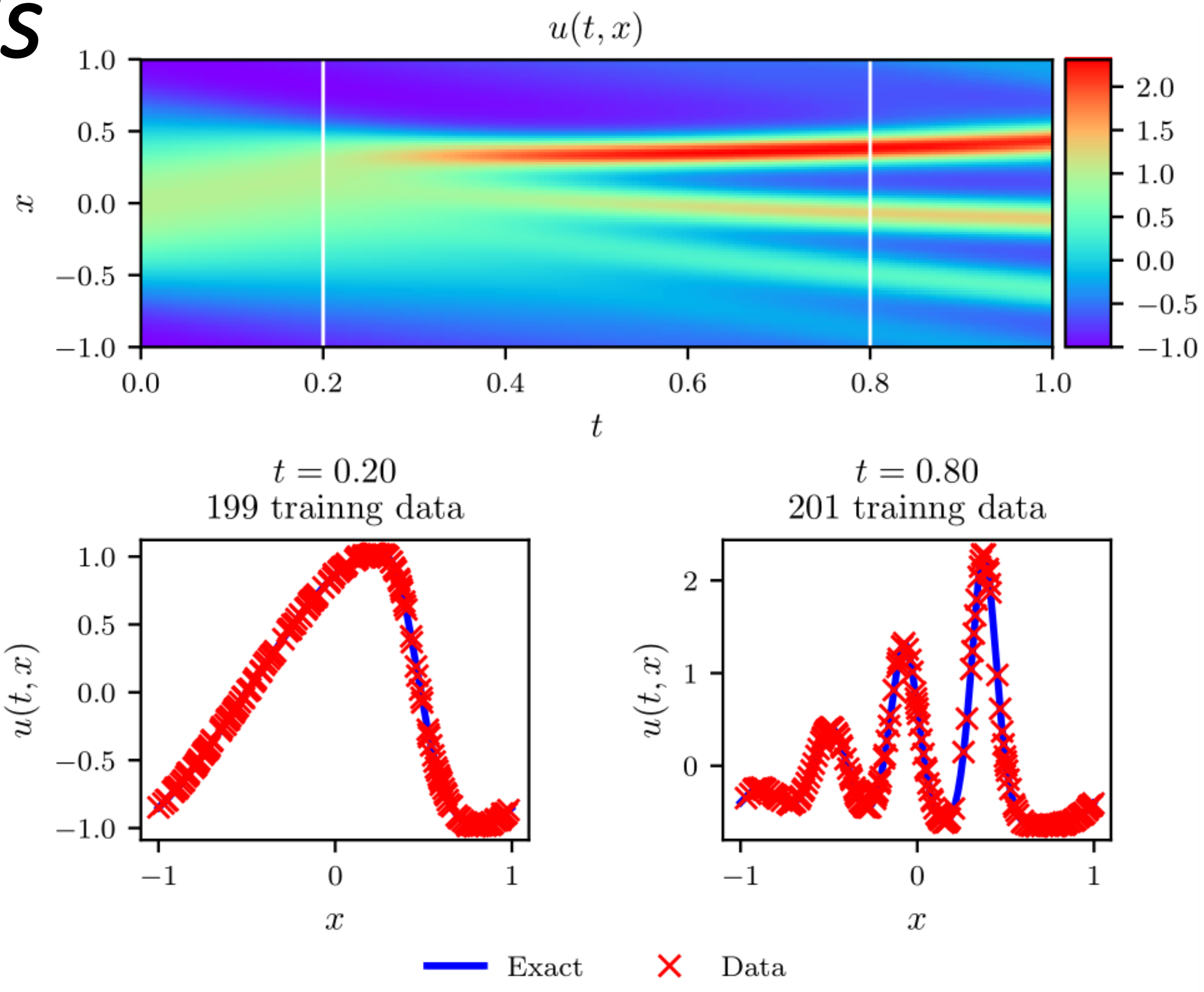
- PINNs are the most well known type of physics informed deep learning models
- Inputs
  - Coordinates (space and/or time)
  - May add auxiliary variables to input
- Outputs
  - PDE solution fields
  - May add other outputs (inverse problems)
- Train by constraining encoded physics
  - Randomly sample domain
  - May add known data
- **Trained for a single case**
  - 1 set of ICs/BCs
  - 1 set of PDEs → cannot modify source terms



Lu et al 2019

Types of Problems for PINNs

- Forward problems
  - Solve PDEs within specified domain
  - We will look at using PINNs to solve various forward problems
- Inverse problems
  - Given data that obeys a known (or partially known) PDE
  - Compute quantities of interest
    - Flow field from sensors at a few locations
    - Unknown PDE coefficients from data
  - We will look at finding unknown coefficients for a Lorenz system

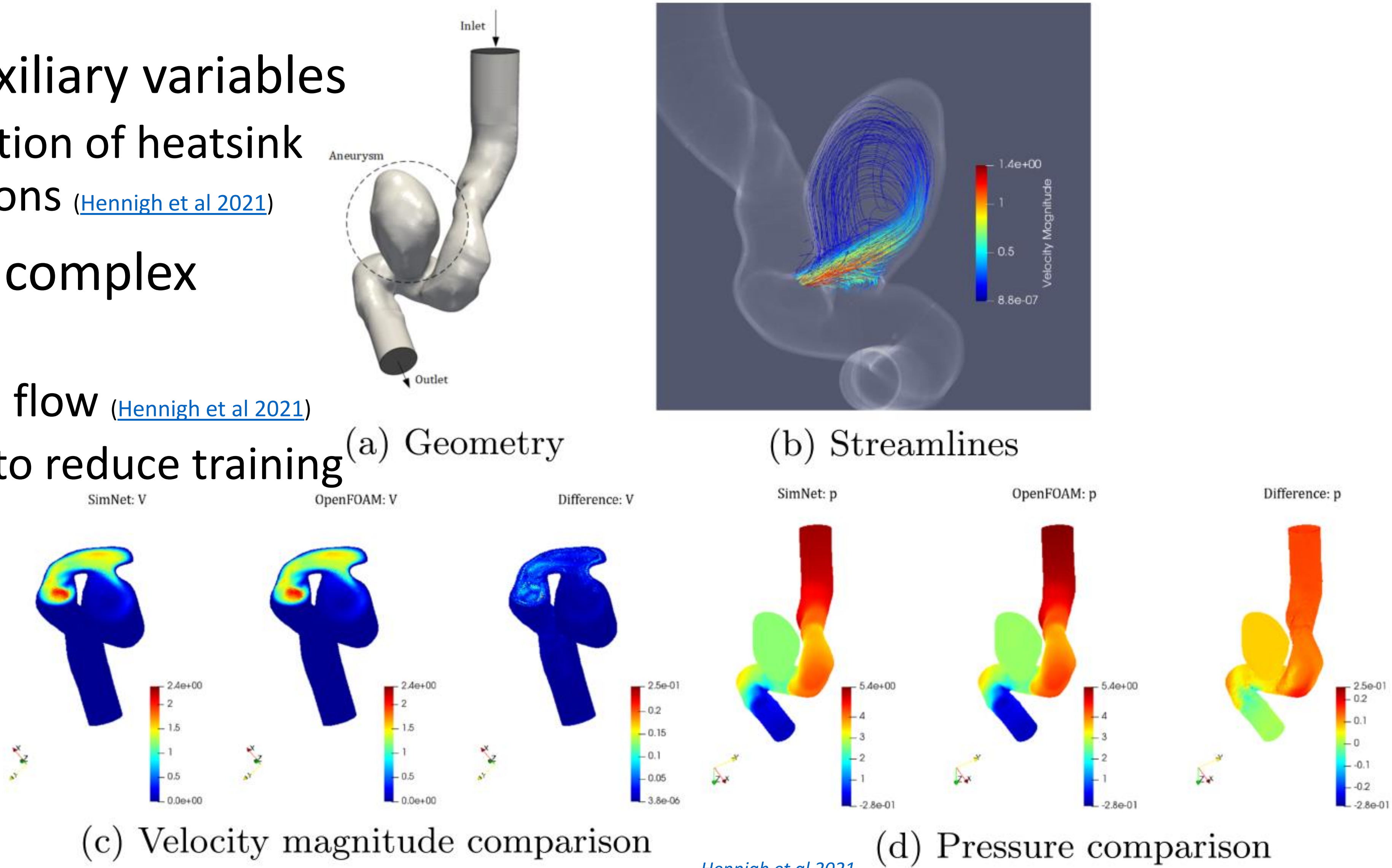


Correct PDE	$u_t + uu_x + 0.0025u_{xxx} = 0$
Identified PDE (clean data)	$u_t + 1.000uu_x + 0.0025002u_{xxx} = 0$
Identified PDE (1% noise)	$u_t + 0.999uu_x + 0.0024996u_{xxx} = 0$

Raissi et al 2019

Applications of PINNs Forward Problems

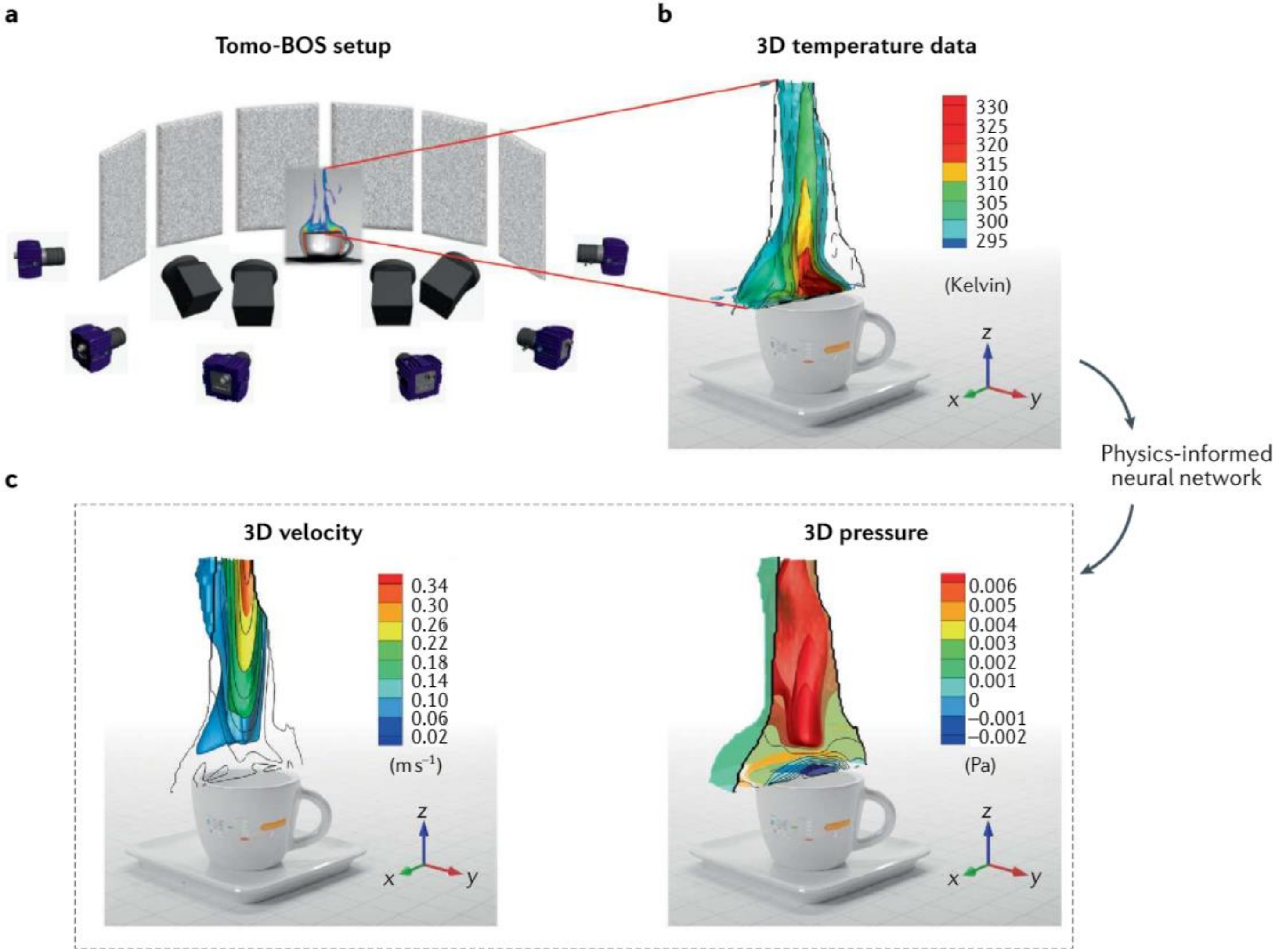
- Optimize PDE over auxiliary variables
  - FPGA design optimization of heatsink geometric configurations ([Hennigh et al 2021](#))
- Simulations over very complex geometries
  - Brain aneurysm blood flow ([Hennigh et al 2021](#))
  - Use transfer learning to reduce training time



[Hennigh et al 2021](#)

Applications of PINNs Inverse Problems

- Reconstruct fields from limited sensor data in ill-posed problems
  - Construct fluid flow from a coffee cup (Cai et al. 2021)
  - Used temperature measurements to construct velocity and pressure data
- Analysis of scientific experiments
  - Well understood models
  - Controlled environments



Karniadakis et al 2021

PINN Software

- **Deepxde (https://github.com/lululxvi/deepxde) → Will use deepxde for our tutorials**
- NVIDIA Modulus/SimNet (Modulus | NVIDIA Developer)
- SciANN (https://github.com/sciann/sciann)
- Elvet (https://gitlab.com/elvet/elvet)
- TensorDiffEq (https://github.com/tensordiffeq/TensorDiffEq)
- NeuroDiffEq (https://github.com/analysiscenter/pydens)
- NeuralPDE (https://github.com/SciML/NeuralPDE.jl)
- Universal Differential Equations for Scientific Machine Learning (https://github.com/ChrisRackauckas/universal_differential_equations)
- IDRLnet (https://github.com/idrl-lab/idrlnet)

Limitations of PINNs

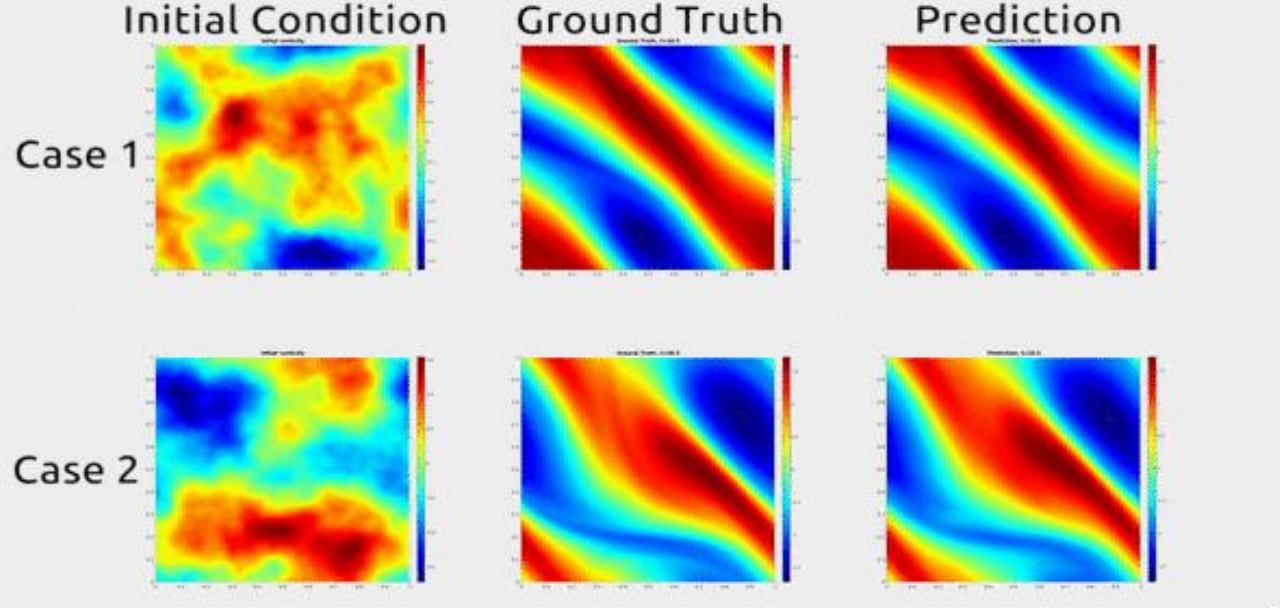
- Only trained for a single set of ICs/BCs/source terms → need to retrain for each new configuration
- Pure PINNs make poor surrogate models
- Will look at operator networks for solving PDEs with variable input fields

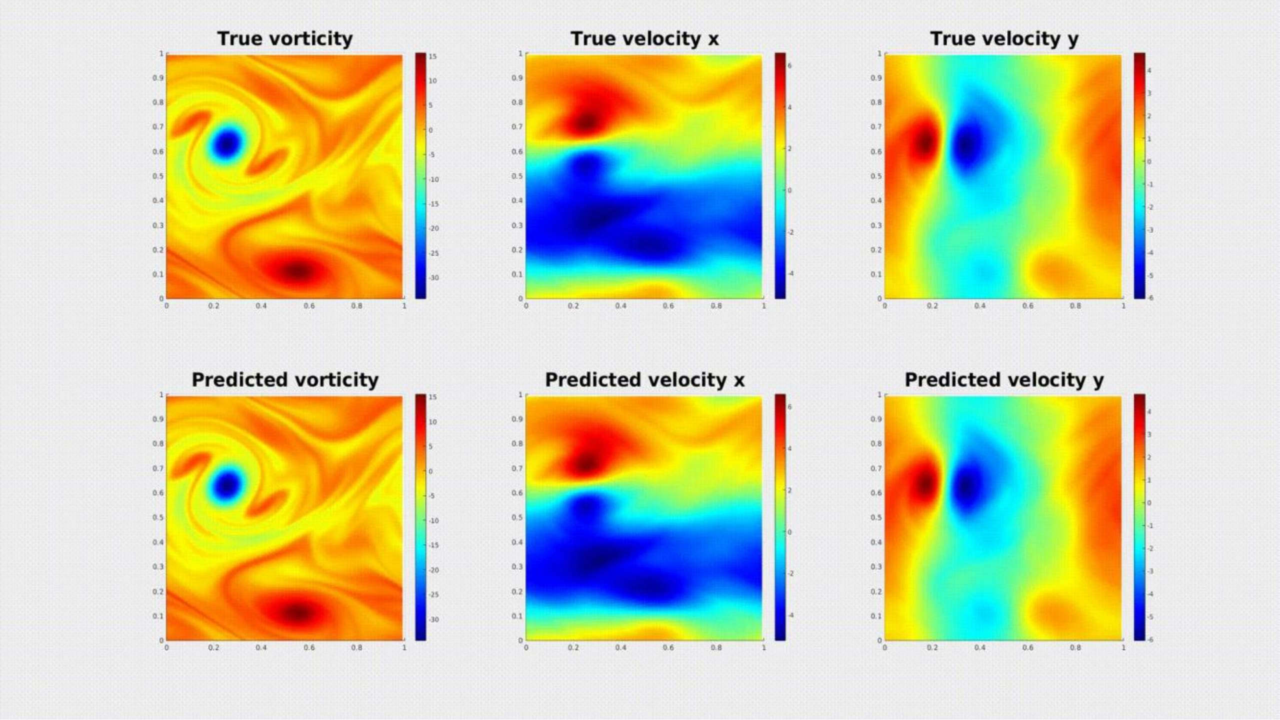
PINNs Exercises

- 07_physics_inpiredAI
  - Burgers Equation
  - Poisson Lshape
  - Complex Geometry
  - Lorenz Inverse System

Operator Networks

- Learn output field for a given input field
- Can learn variable ICs, BCs, and/or source terms
- Need to generate data for many input fields
- May use physics information to improve performance
- Examples
  - DeepONets ([Lu et al 2021]())
  - **Physics Informed DeepONets** ([Wang et al 2021]())
  - Graph Operator Networks ([Li et al 2020]())
  - Fourier Operator Networks ([Li et al 2020]())
  - PINOs [(Li et al 2021)]()

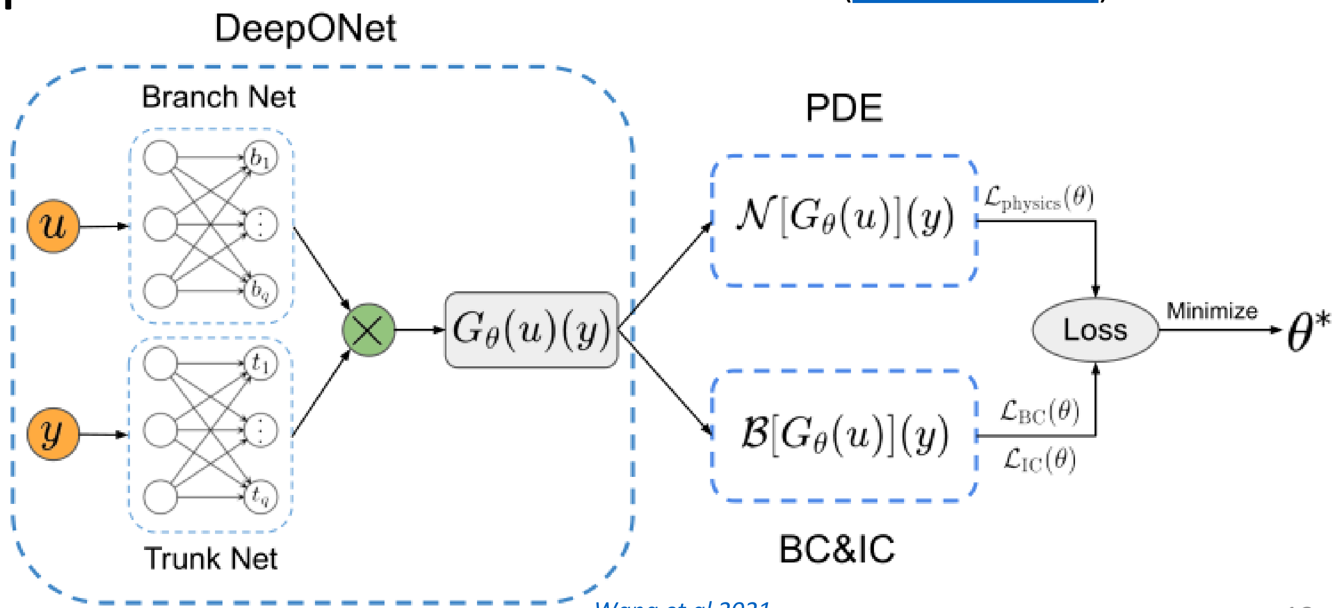




[Neural Operator (zongyi-li.github.io)]()

Physics Informed DeepONets

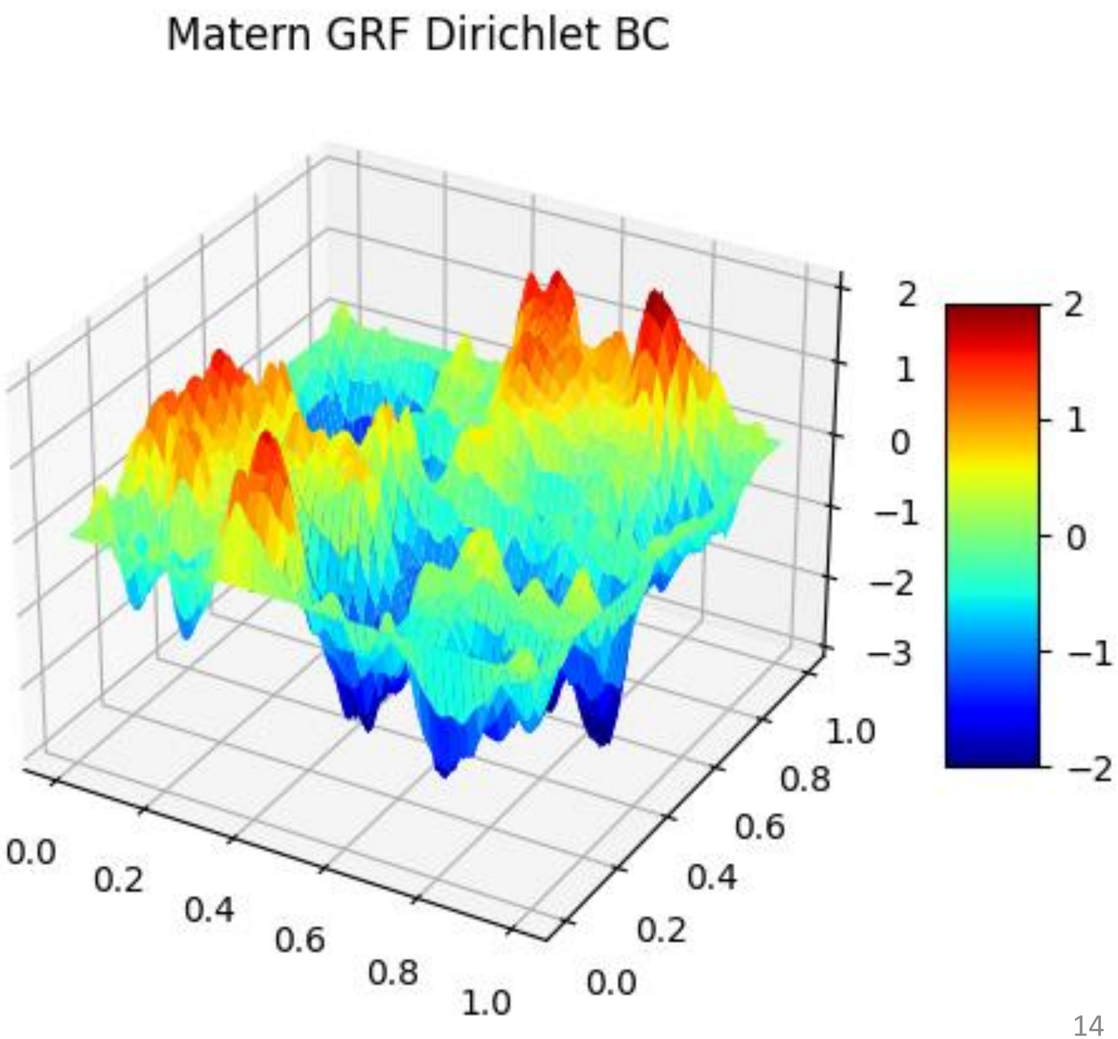
- DeepONets can generalize PDE solutions (Lu et al 2021)
  - Input field u → Initial conditions, source terms, and/or boundary conditions
  - Input coordinate y → space and time
  - Output operator $G(u)(y)$ → PDE solution
  - Difference between data s and operator $G(u)(y)$ is our loss $\mathcal{L}_{data}$
- Physics informed DeepONets improve performance with less data (Wang et al 2021)
  - Incorporate PDE into loss $\mathcal{L}_{physics}$
  - Incorporate ICs into loss $\mathcal{L}_{IC}$
  - Incorporate BCs into loss $\mathcal{L}_{BC}$



Wang et al 2021

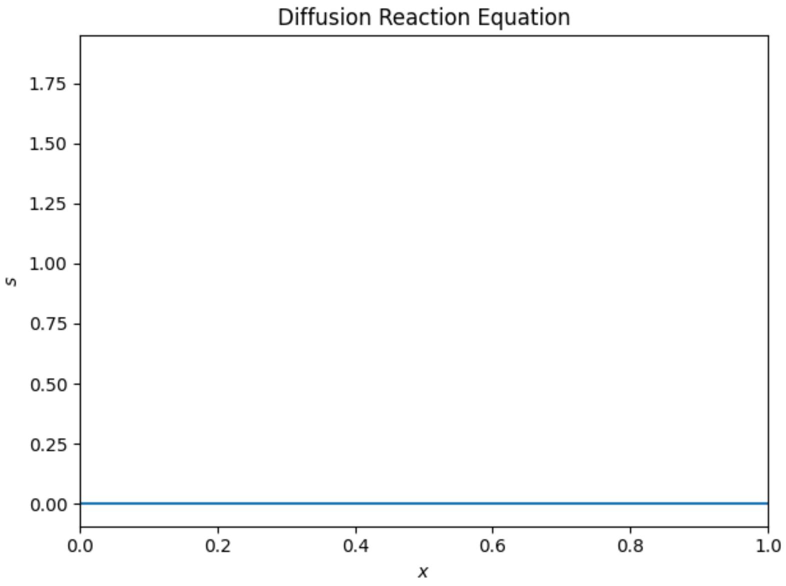
Training Physics Informed DeepONets

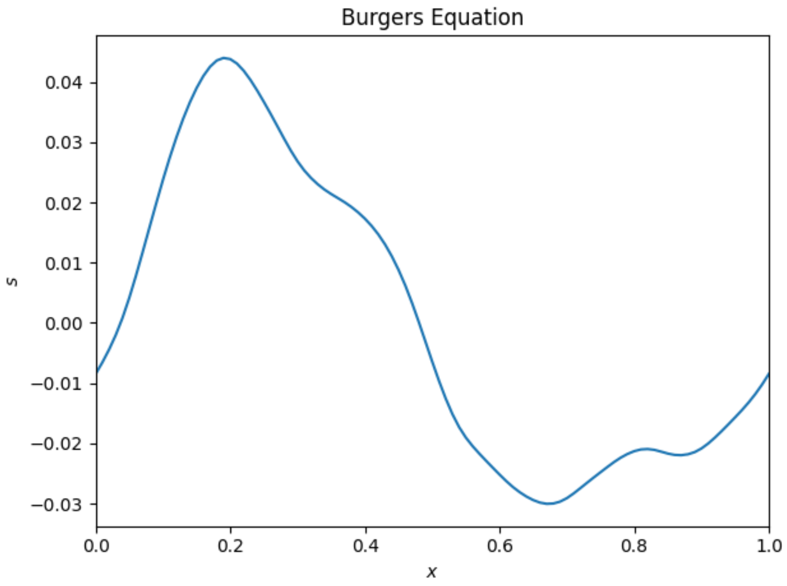
- Generate u using Gaussian random fields (GRF)
  - Use RBF or Matérn kernel to obtain spatially correlated random data
  - Apply length scale l associated with typical spatial deviations
  - Expand in Fourier components to obey boundary conditions
- Run simulations for each u to generate training data
- Sample the solution space during training

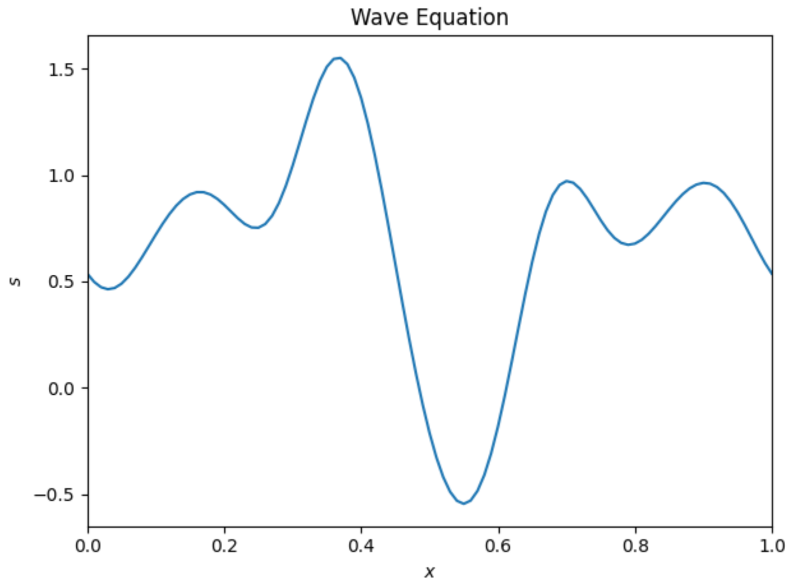


Physics Informed DeepONet Tests

- 1D Diffusion Reaction Equation
  - $\partial_t s = D\, \partial_{xx} s + k\, s^2 + u(x)$
  - u is a source term
  - Homogenous Dirichlet BC
  - Zero IC → $s(x, 0) = 0$
  - $k = D = 0.01$
- 1D Viscous Burgers Equation
  - $\partial_t s + s\, \partial_x s - \nu\, \partial_{xx}^2 s = 0$
  - u is the IC
  - Periodic BC
  - $\nu = 0.01$
- 1D Wave Equation
  - $\partial_{tt} s - c^2 \partial_{xx}^2 s = 0$
  - u is the IC
  - Periodic BC
  - $c = 1$

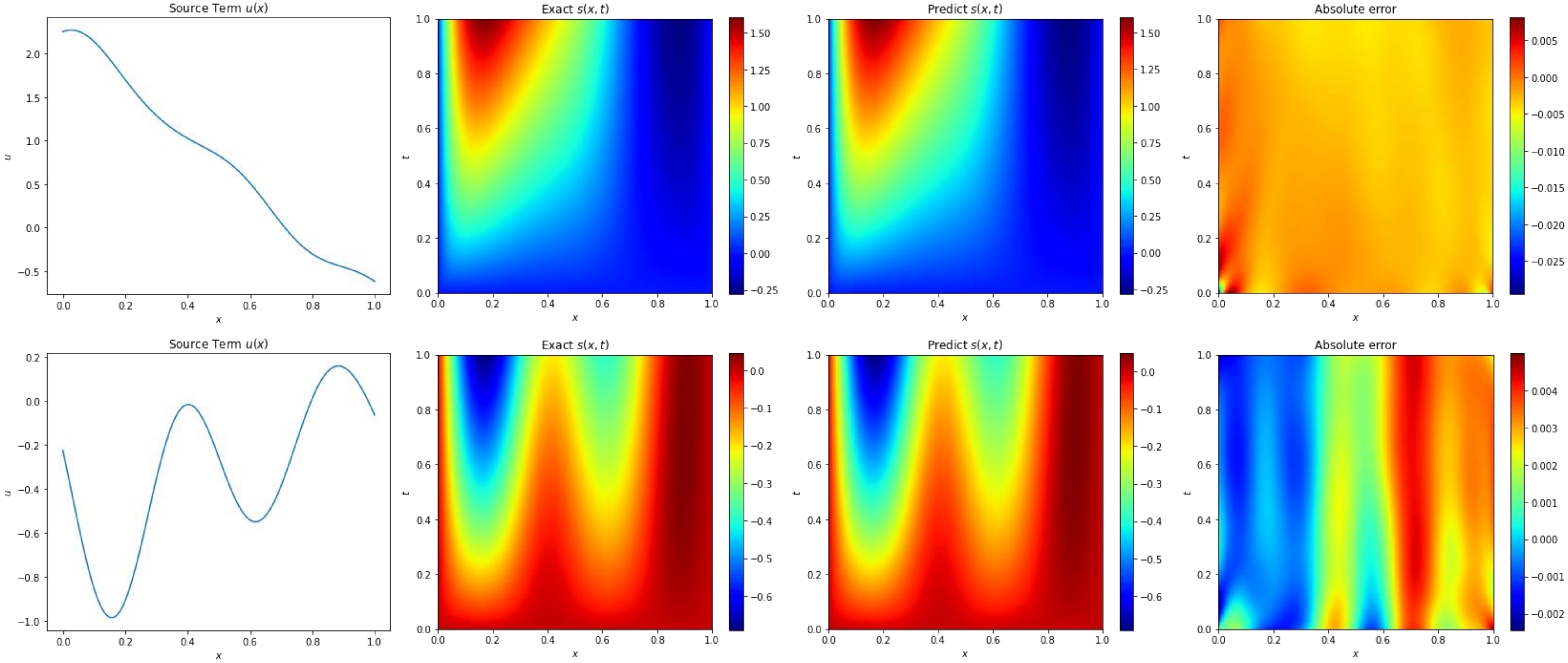






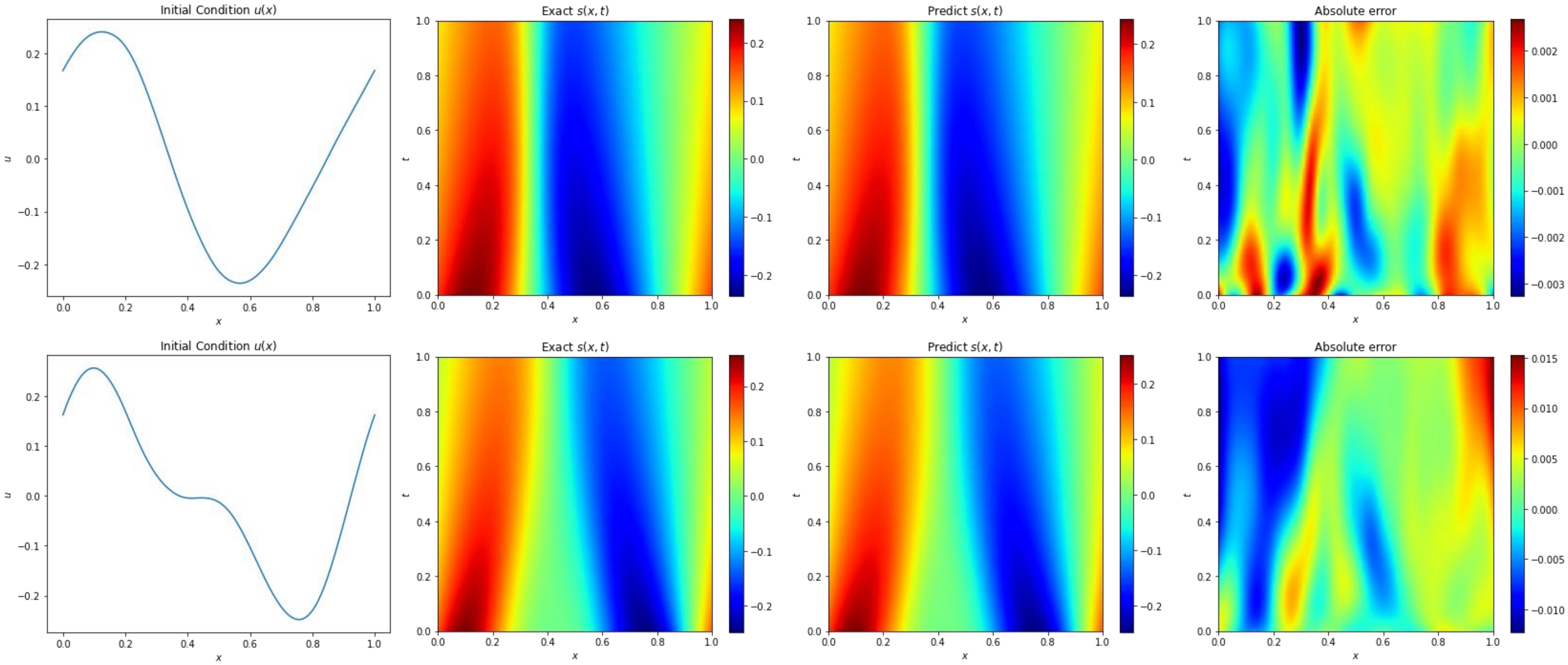
Diffusion Reaction Equation Results on Test Data:

$$\partial_t s = D\ \partial_{xx} s + k\ s^2 + u(x)$$



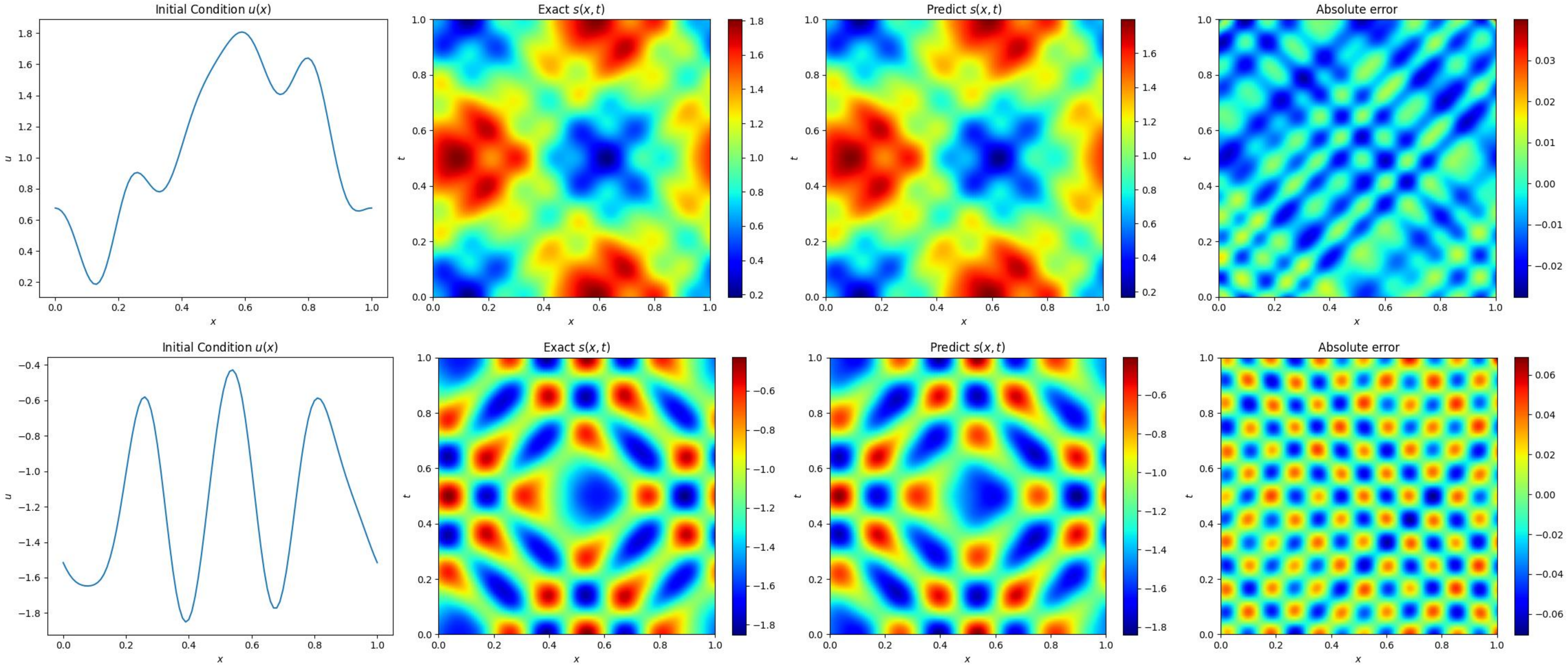
Viscous Burgers Equation Results on Test Data

$$\partial_t s + s\,\partial_x s - \nu\,\partial_{xx}^2 s = 0$$



Wave Equation Results on Test Data

$$\partial_{tt}s - c^2\partial_{xx}^2 s = 0$$



Physics Informed DeepONet Excercise

- [Physics Informed DeepONet Diffusion Reaction]