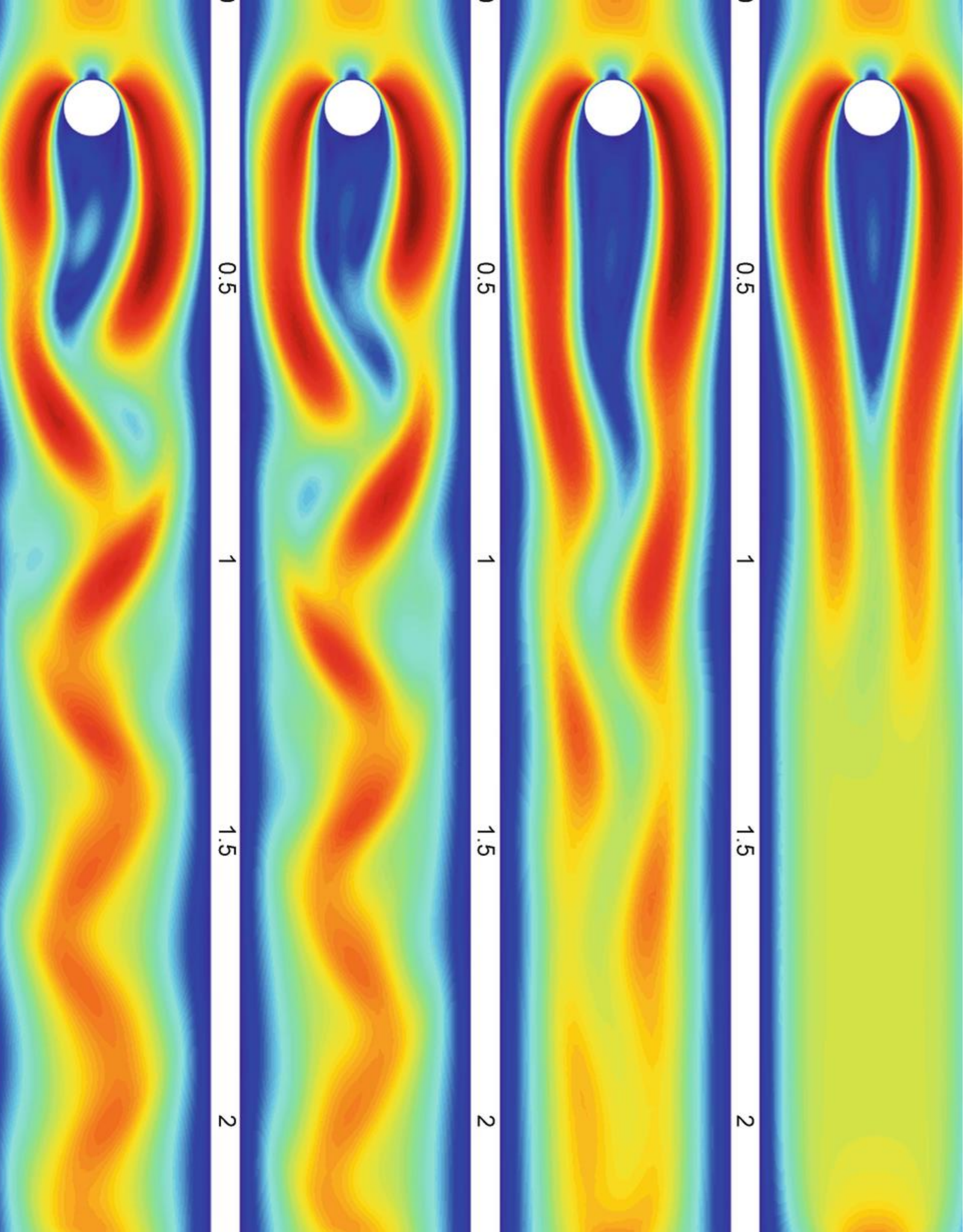
SESSION // 03B
PHYSICS INFORMED NEURAL NETWORKS

FACULTY OF
SCIENCE AND ENGINEERING
+++

MANCHESTER 1824
The University of Manchester

Diego Corona Lopez – AI Technical Specialist



AGENDA

Introduction to Physics-Informed Neural Networks

Navier-Stokes Equations and Fluid Flow

- Mathematical formulation of Navier-Stokes equations
- Momentum and continuity equations

PINN Architecture and Components

Loss Functions and Physics Constraints

- Computing derivatives via autograd
- First and second order derivatives implementation
- Residual formulation for Navier-Stokes equations

Training Process and Optimization

Results and Visualization

PINNS

What are PINNS:

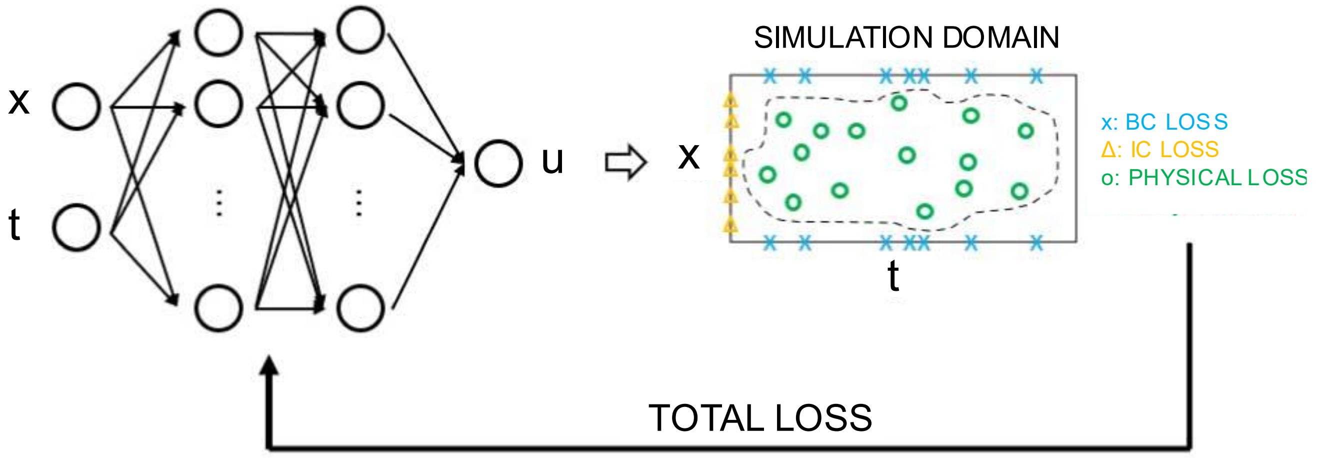
Neural networks that incorporate physical laws into training

Key insight:

Combine data-driven learning with physics constraints = **"Learning while respecting the laws of physics“**

Traditional numerical methods limitations:

- Computational cost
- Mesh requirements
- Limited data handling
- Curse of dimensionality
- Limited generalisation



NAVIER-STOKES EQUATIONS

Foundation of fluid dynamics

- Partial differential equations governing fluid motion
- Challenging to solve numerically

$u(x, y, t)$: horizontal velocity

$v(x, y, t)$: vertical velocity

$p(x, y, t)$: pressure

λ_1 : convection coefficient (usually 1)

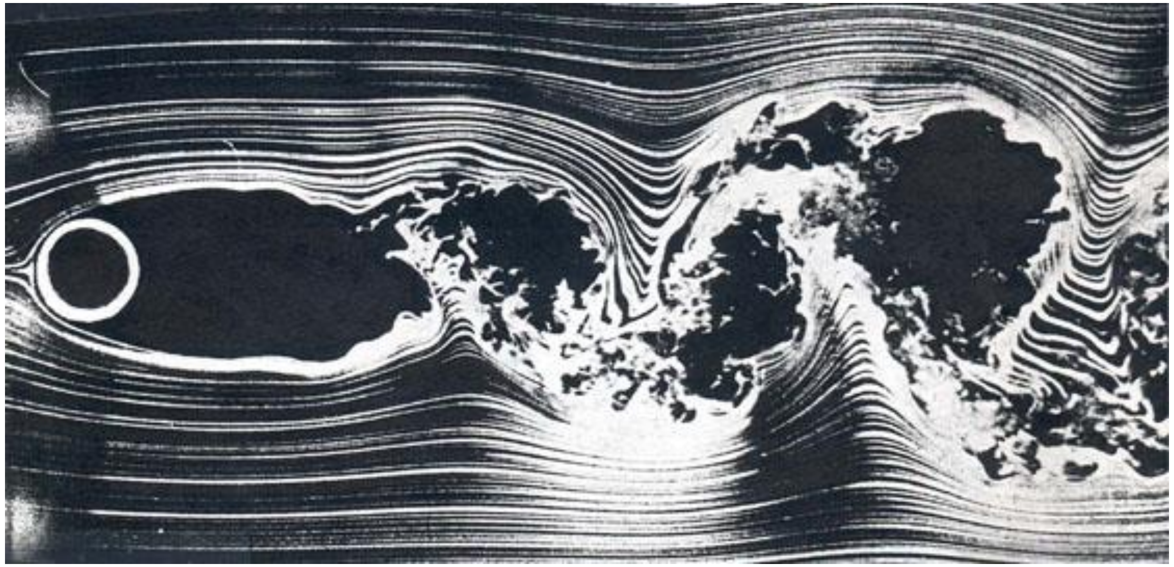
$\lambda_2 = \nu$: kinematic viscosity

$$\frac{\partial u}{\partial t} + \lambda_1\left(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \lambda_2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \quad \text{(momentum in x}-\text{direction)}$$

$$\frac{\partial v}{\partial t} + \lambda_1\left(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = -\frac{\partial p}{\partial y} + \lambda_2\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \quad \text{(momentum in y}-\text{direction)}$$

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

WAKE-CYLINDER DATA



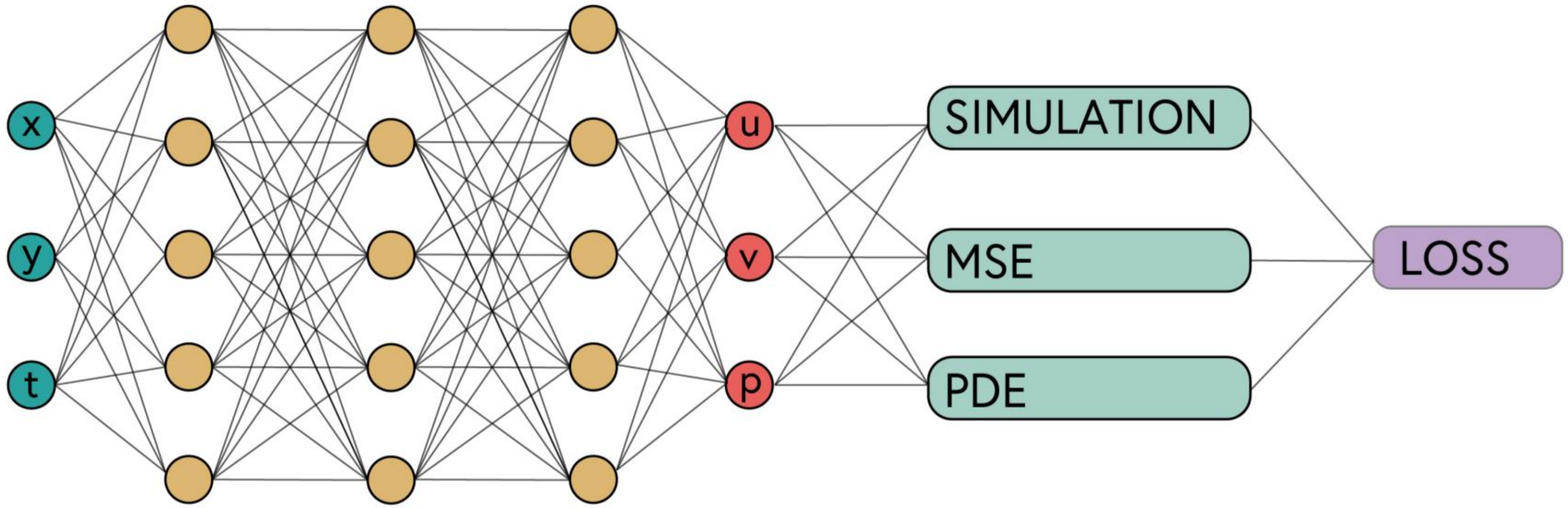
- Domain: [-15, 25] × [-8, 8]
- Reynolds number: 100
- Visualization of the flow simulation
- What we're predicting: λ_1, λ_2, and pressure field p

PINN ARCHITECTURE

$$\text{Loss} = \text{Data Loss} + \text{Physics Loss}$$

$$\text{Data Loss} = \frac{1}{N}\sum_{i=1}^{N}\left(u_i - u_{\text{pred},i}\right)^2 + \left(v_i - v_{\text{pred},i}\right)^2$$

$$\text{Physics Loss} = \frac{1}{M}\sum_{j=1}^{M}\left(f_{u,j}^2 + f_{v,j}^2 + f_{c,j}^2\right)$$



PHYSICS-CONSTRAINTS

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

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

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

AUTOGRAD DIFFERENTIATION

```
u_t = torch.autograd.grad(u, t,
                          grad_outputs=torch.ones_like(u),
                          create_graph=True)[0]
```

Parameter	Purpose	Description
`u`	Target tensor	The output we want to differentiate
`t`	Source tensor	The variable we're differentiating with respect to
`grad_outputs`	Scaling factor	Usually ones, for direct gradient computation
`create_graph`	Enable higher derivatives	Needed for second derivatives

TRAINING PROCESS

```python
optimizer = optim.LBFGS(model.parameters(),
                        lr=0.1,
                        max_iter=500,
                        max_eval=500,
                        tolerance_grad=1e-8,
                        tolerance_change=1e-8,
                        history_size=50,
                        line_search_fn="strong_wolfe")

# Define the closure function
def closure():
    optimizer.zero_grad()
    loss = loss_function(model)
    loss.backward()
    return loss

# Training loop
for epoch in range(num_epochs):
    optimizer.step(closure)
```

Two-phase optimization strategy

- Phase 1: Adam optimizer (warm-up)
- Phase 2: L-BFGS optimizer (fine-tuning)

tolerance_grad
Stops optimization when the maximum element in the gradient vector falls below this threshold
- Smaller values → More precise solutions

tolerance_change
Terminates when relative change in function value is below threshold
- Prevents wasting computation when progress stalls

history_size
Number of past iterations stored to approximate Hessian matrix
- Larger values → Better curvature information → More accurate steps

LOSS FUNCTION

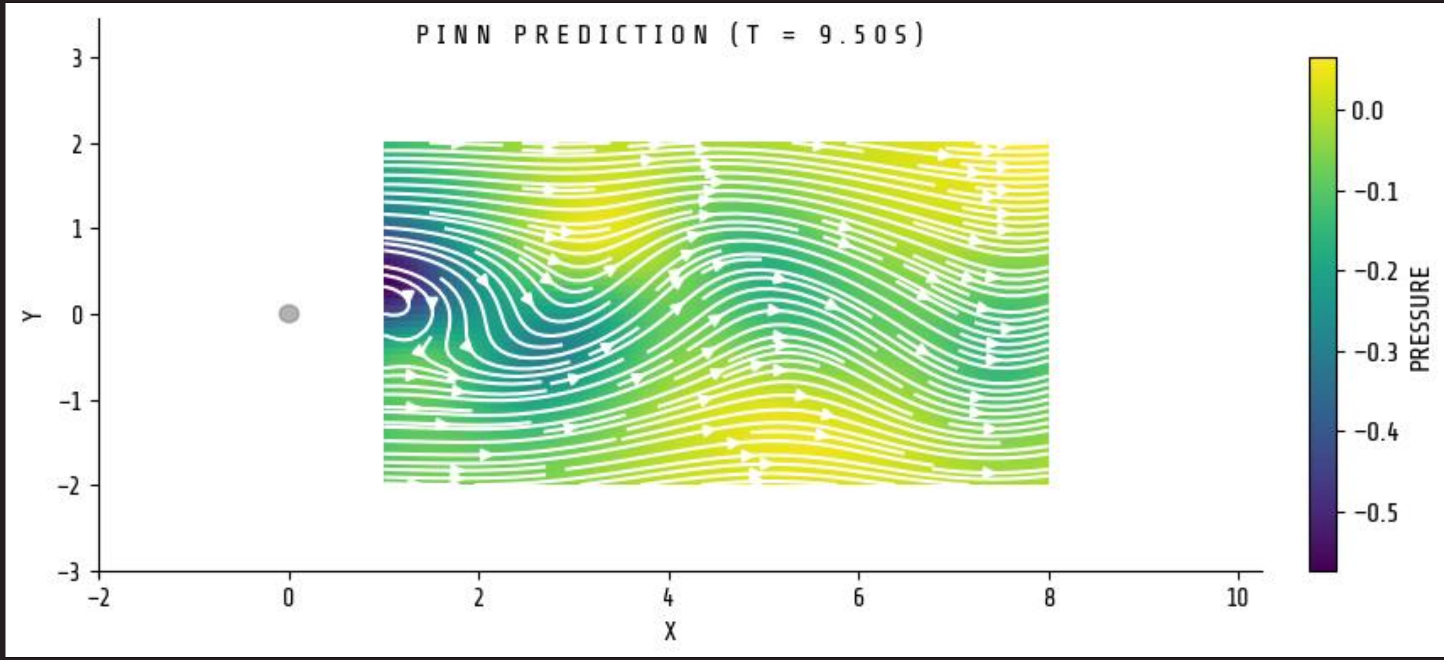
- Loss functions quantify prediction errors
- We use Mean Squared Error (MSE):

$$MSE = 1/n\,\Sigma(y - \hat{y})^2$$

- Provides direction for optimization

```
def mse_loss(predictions: torch.Tensor,
             targets: torch.Tensor) -> torch.Tensor:

    squared_diff = (predictions - targets) ** 2
    return squared_diff.mean()
```



ADVANTAGES AND LIMITATIONS

- Working with limited data (only 1% training data)
- No pressure data needed for training
- Physics-consistent prediction

- Computational cost of training
- Handling of boundary conditions
- Scaling to more complex geometries