Physics-based AI

Integrate physical laws and principles into ML

Mohammad Shafieeha, 1403 spring

(Inspired by the content of the 401-4656-21L course at the ETH)

Physics-based AI

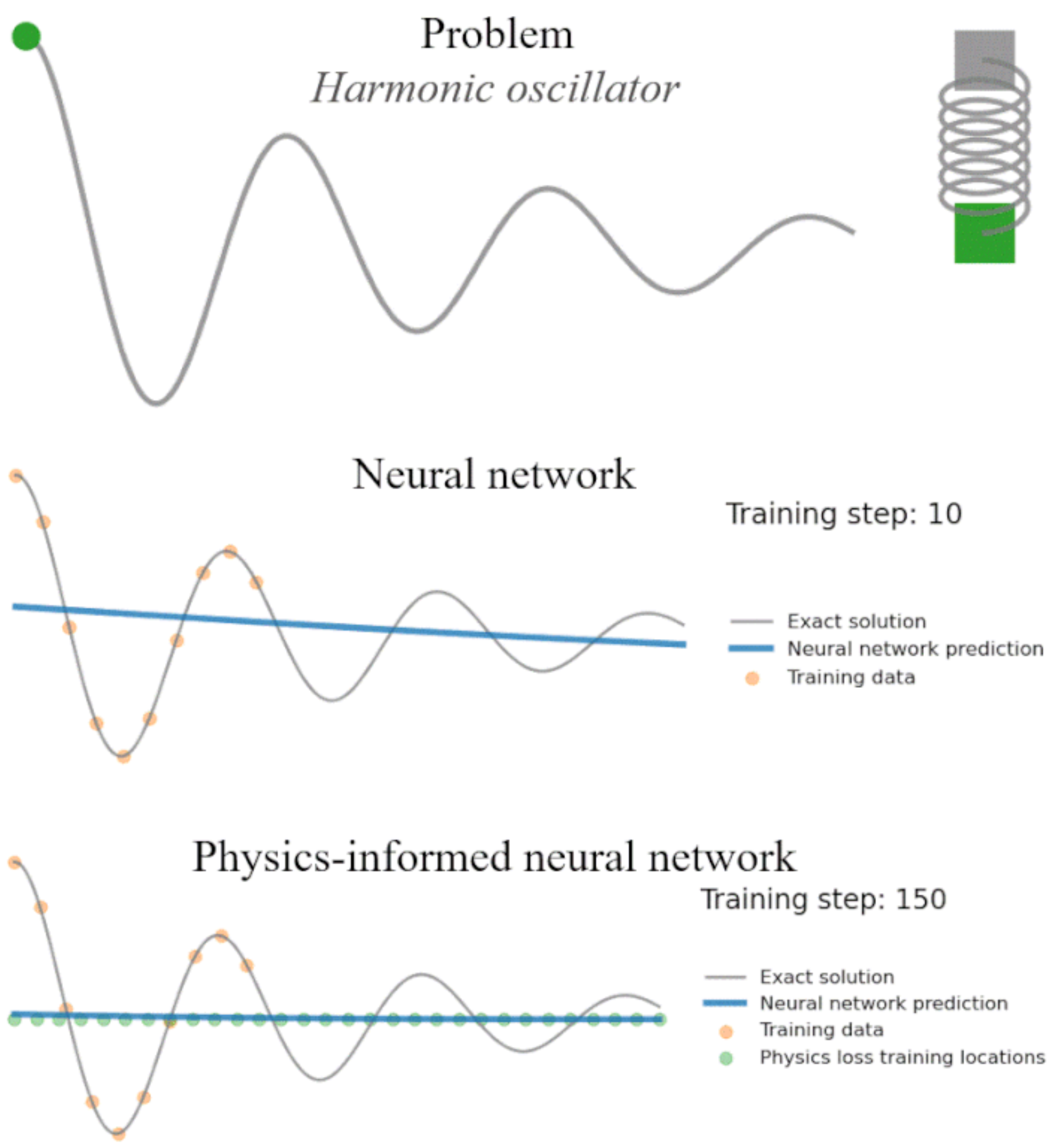
Physics-Informed Neural Networks

Physics-based Deep Learning

Physics-Informed Machine Learning

Physics Inspired

Neural Networks are trained to solve supervised tasks respecting given laws of physics described by general nonlinear partial differential equations.

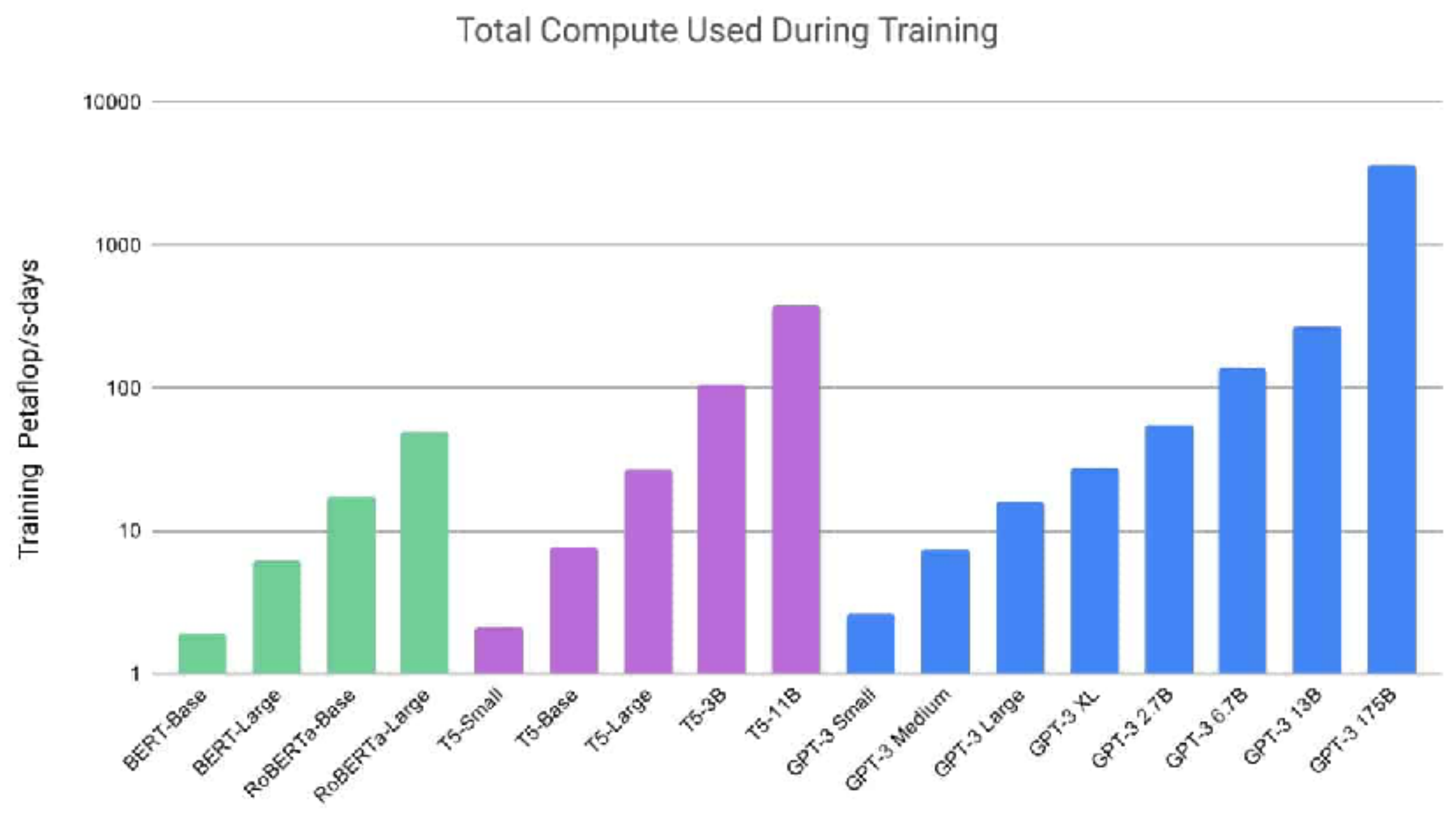

Problem
Harmonic oscillator
Neural network
Training step: 10
Exact solution
Neural network prediction
Training data
Physics-informed neural network
Training step: 150
Exact solution
Neural network prediction
Training data
Physics loss training locations

PROBLEMS

Problems

Flaws of Deep Learning

- Model and Architectures (Billion parameters)
- Training Dataset (Terabytes)
- Generalization
- Explainability



R. Merritt, "Beyond Words: Large Language Models Expand AI's Horizon," *NVIDIA Blog*, Oct. 10, 2022. https://blogs.nvidia.com/blog/llms-ai-horizon/ (accessed May 19, 2024).

Definitions

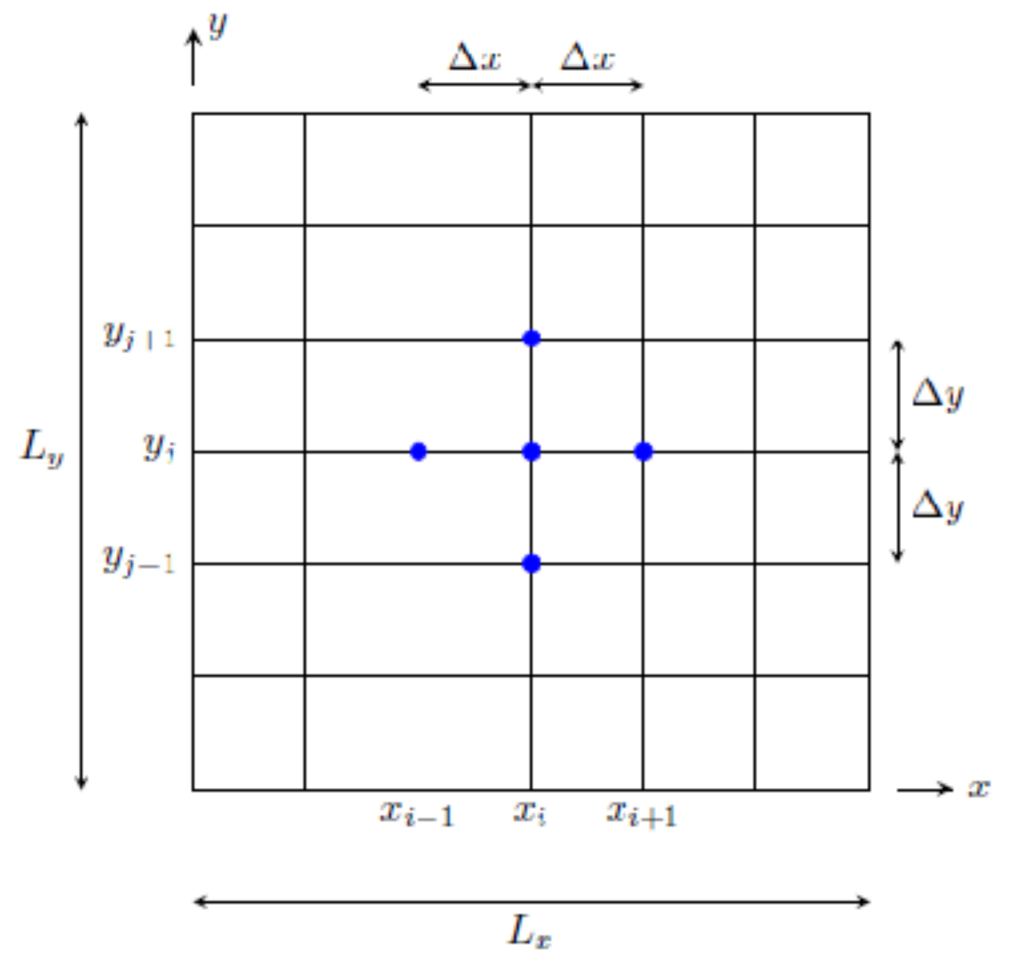
PDE

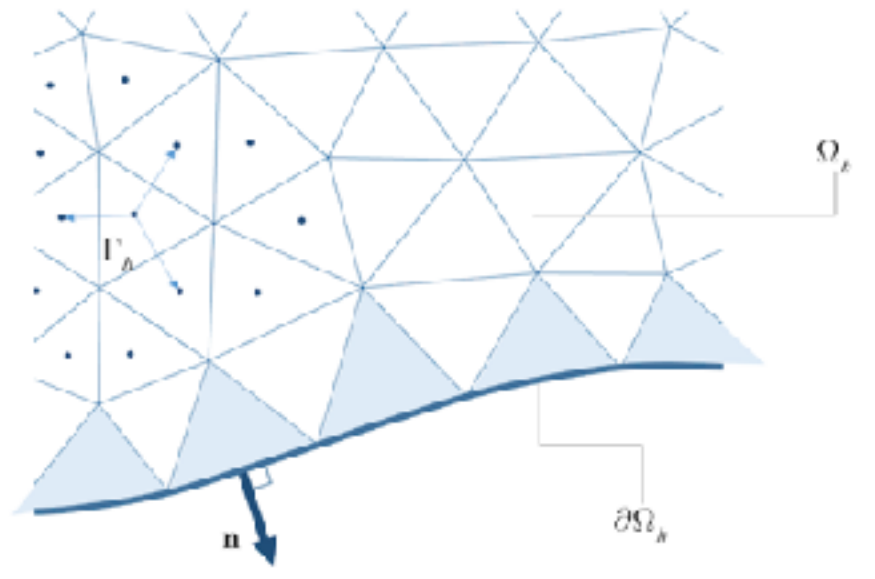
- When dealing with functions of multiple variables, if y depends on more than one variable, say x and z, then the partial derivative of y to x is denoted as $\frac{\partial x}{\partial y}$.
- This measures the rate of change of y to x while holding other variables constant. It treats y as a function of x while ignoring the other variables.
- Given a scientific problem, find a Partial Differential Equation (PDE) describing it. (e.g. Schrödinger equation, Einstein equation, …)

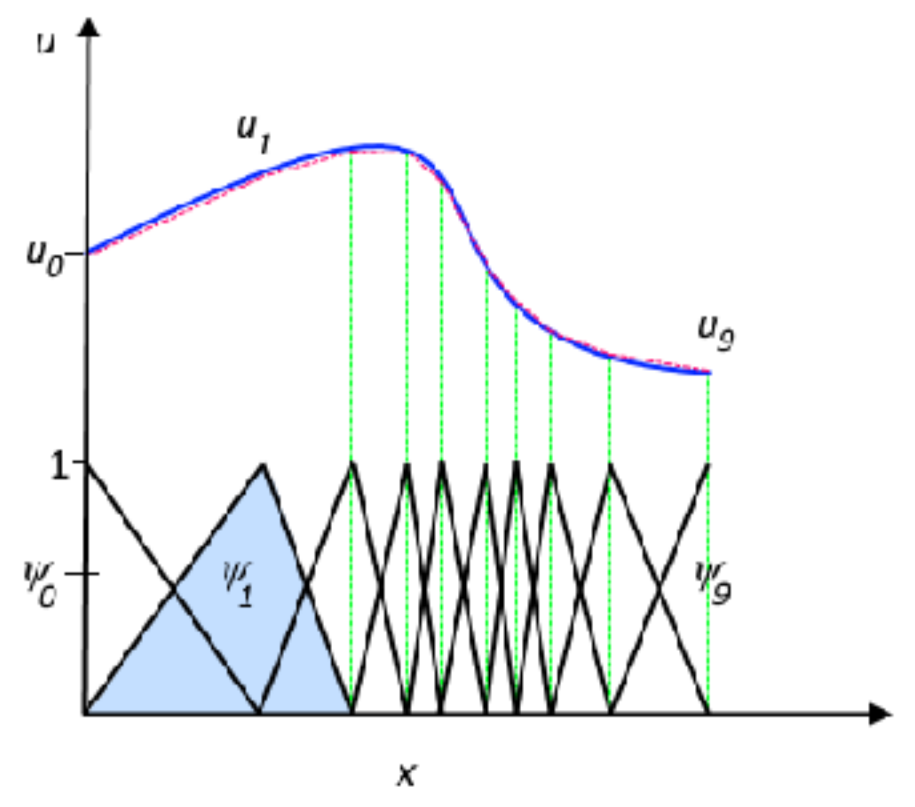
Definitions

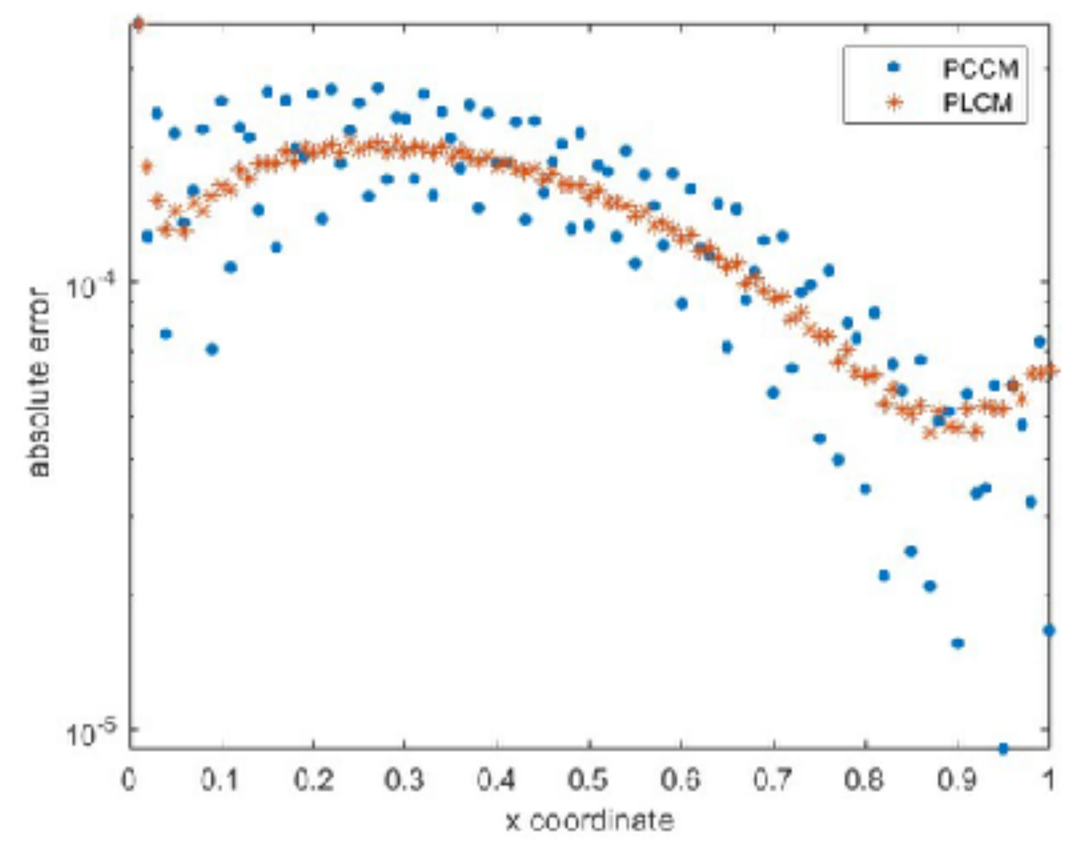
Traditional Numerical Methods

- PDEs don't get the exact answer, even the simple cases. In this scenario, we go for Traditional Numerical Methods. (e.g. Finite Difference, Finite Volume, Finite Element, Collocation)





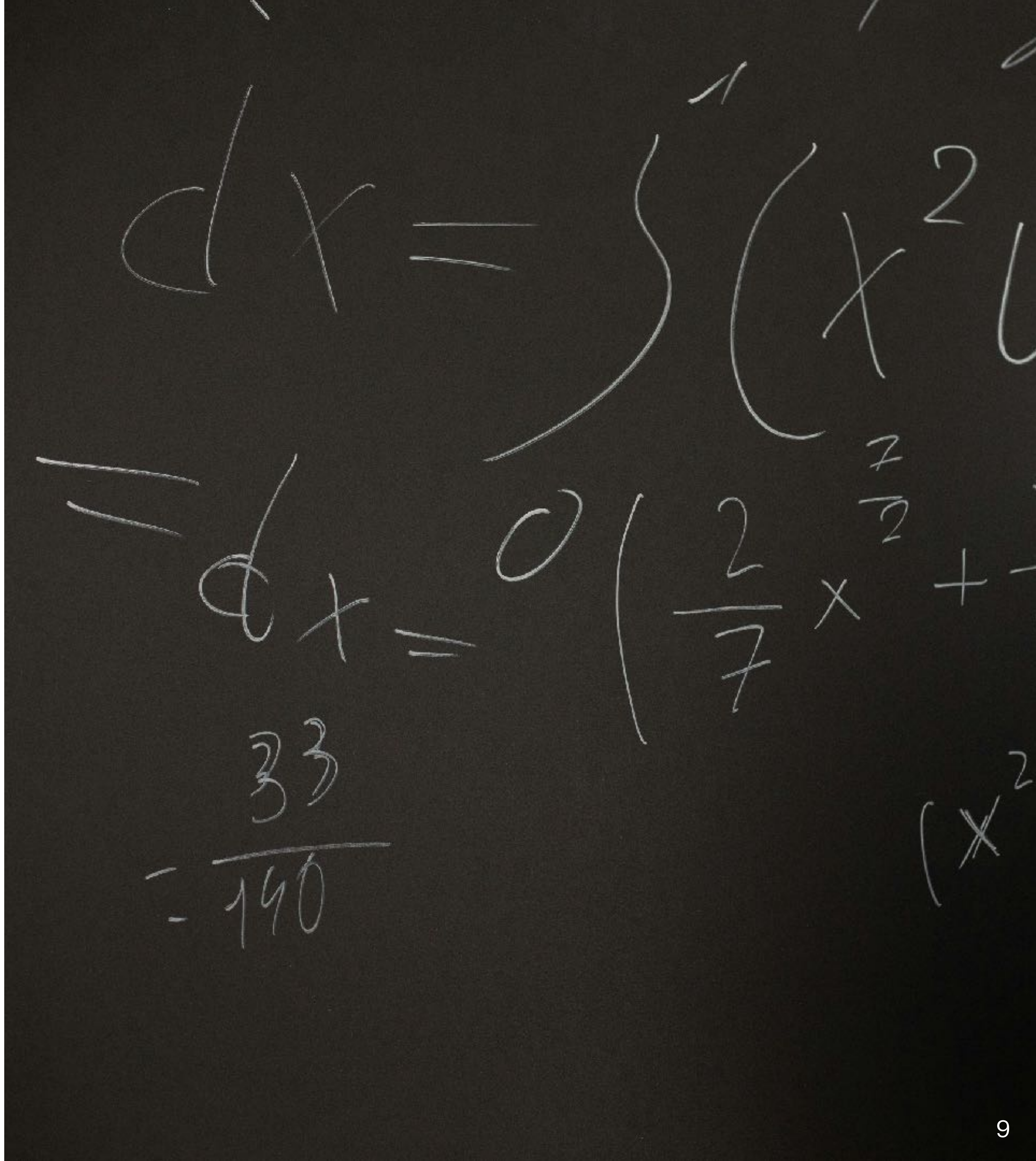




Problems

Flaws of PDE & Numerical Methods

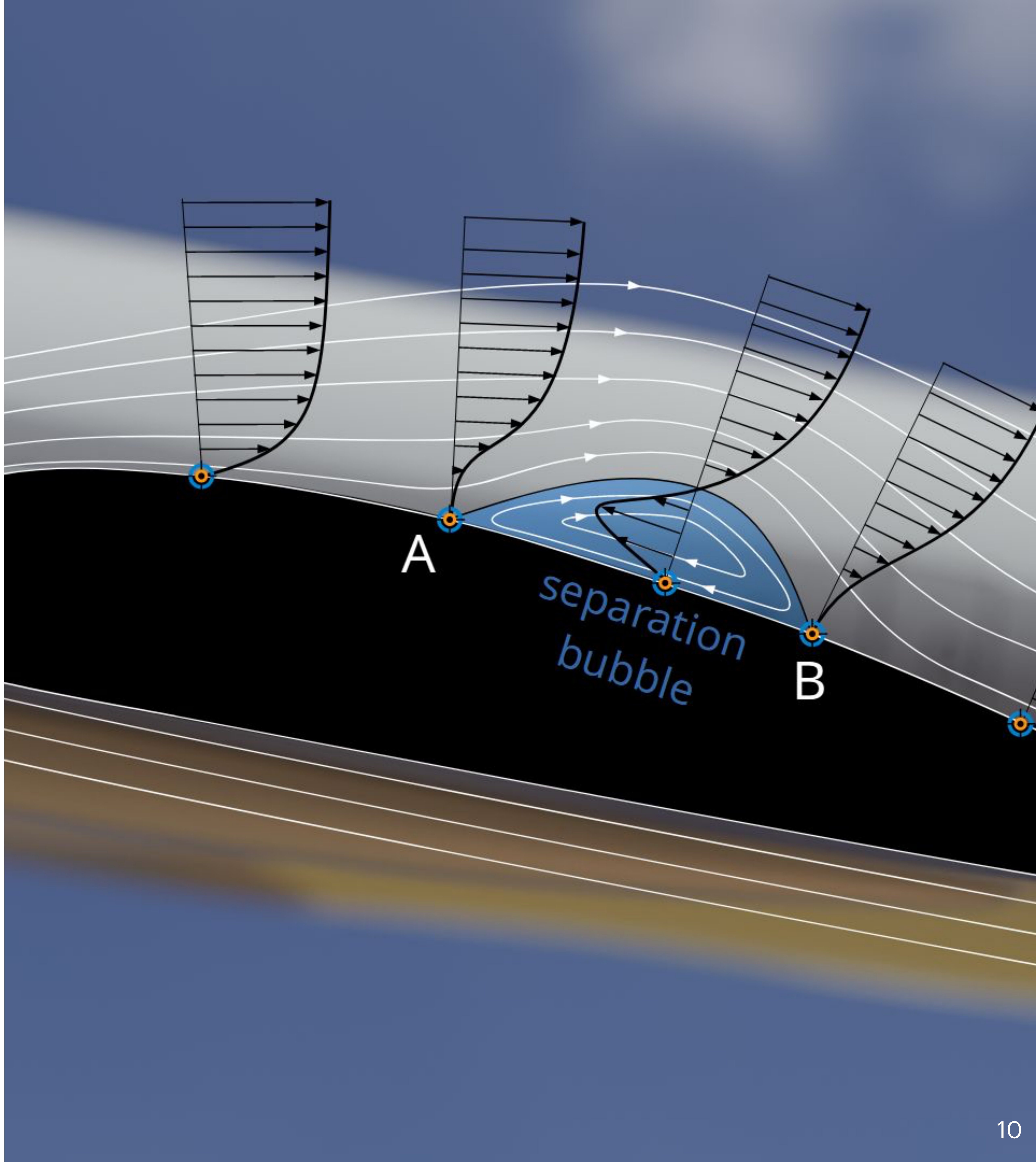
- Computational Cost (Many Query Problems)
- Feasibility (High-dimensional Problems)
- Systems with Incomplete Physics but lots of data.



Optimal Design

Flow past airfoils

- Observables: Lift, Drag
- Uncertain parameters: Incident Mach Number, Angle of Attack, Pressure, Shape defects.
- Design parameters: Shape via Hicks-Henne functions.
- Initial with 50 parameters, then need to optimize in thousands steps and every step 20′ to 30′.



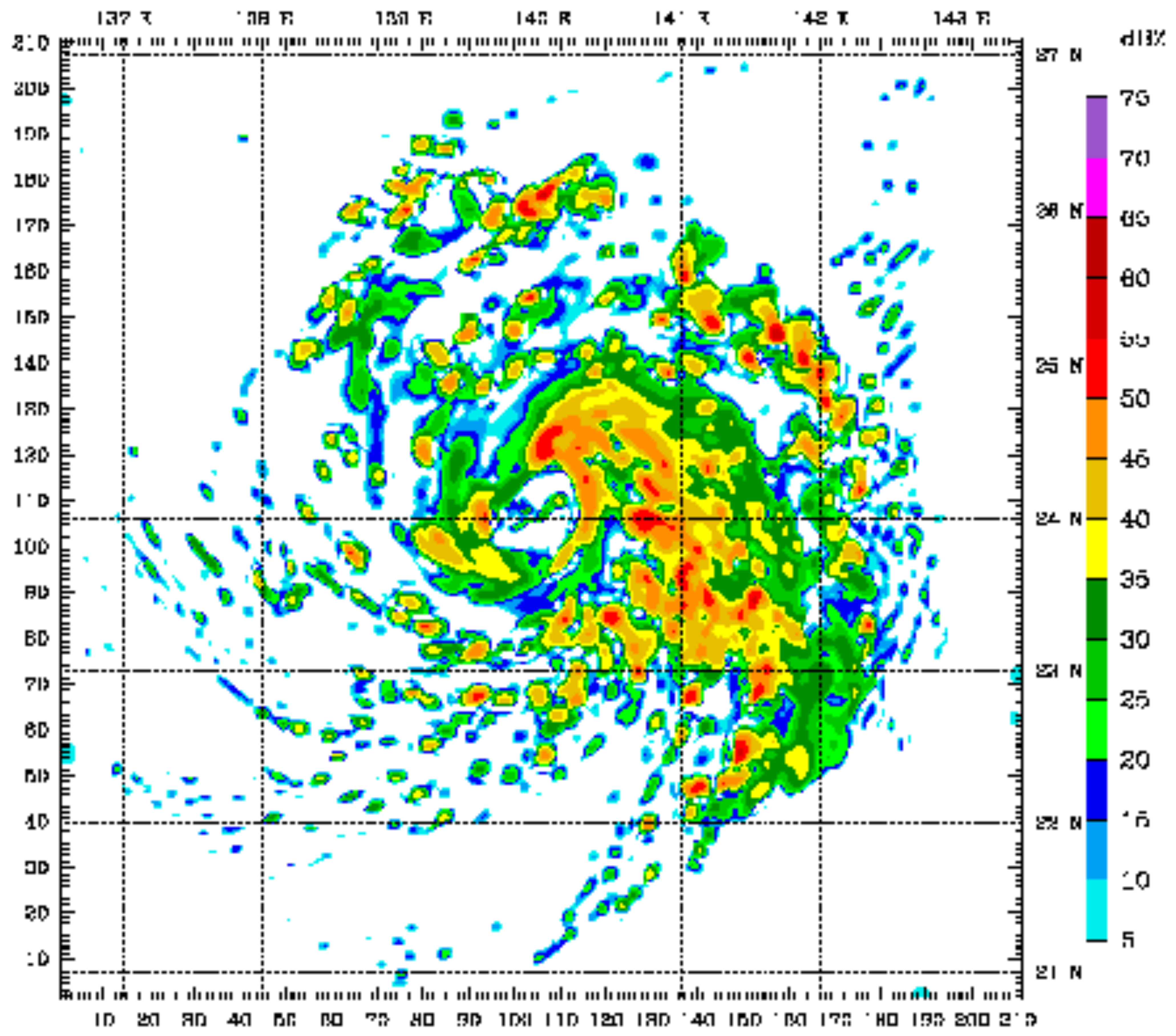
Can we forget the traditional numerical methods and solve PDEs only with DNNs?

The Main Question

APPLICATION

Simulation

Applications



"Computer simulation," Wikipedia, Apr. 25, 2020. https://en.wikipedia.org/wiki/Computer_simulation

Simulation

- Crucial for practically all domains of science
- Essential for understanding the behavior of complex phenomena
- Usually used as a starting point for other tasks (e.g. inverse, control, design)
- This setup encapsulates a wide range of problems in mathematical physics including conservation laws, diffusion processes, and kinetic equations.

$$\boldsymbol{b = F(a)}$$

a = set of input conditions

F = physical model of the system

b = resulting properties given F and a

Inverse Problems
Applications

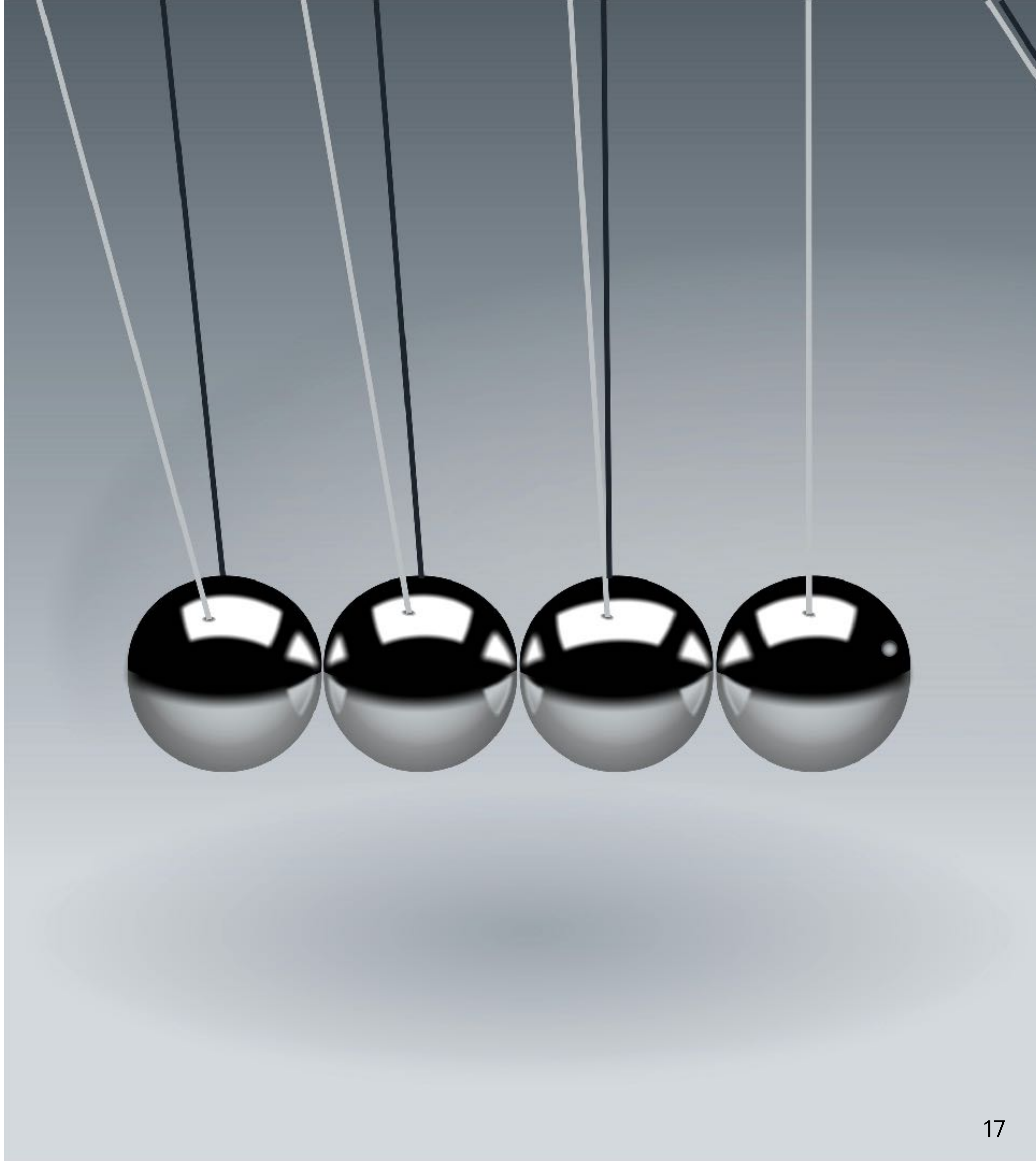
Inverse Problems

- Determine the causal factors from a set of observed effects. This is the opposite of a direct problem (effects are determined from the known causes).
- Examples of inverse problems include Seismic imaging, MRI, and Estimating infection rates, ….
- In seismic imaging, “F” represents the wave equation describing seismic waves, and “a” could represent the subsurface properties like density. The task is to infer these subsurface properties from recorded seismic data “b”.
- Challenges: Small changes in the data can lead to large variations in the solution + Non-linear relationships + Observations may be incomplete/noisy
- Fundamentally, inverse problems are search problems. If “F" is differentiable, one option is the gradient-based method. Otherwise, we can use a gradient-free method (e.g. Bayesian optimization, brute force, …)

$$b = F(a)$$

Equation Discovery

Applications



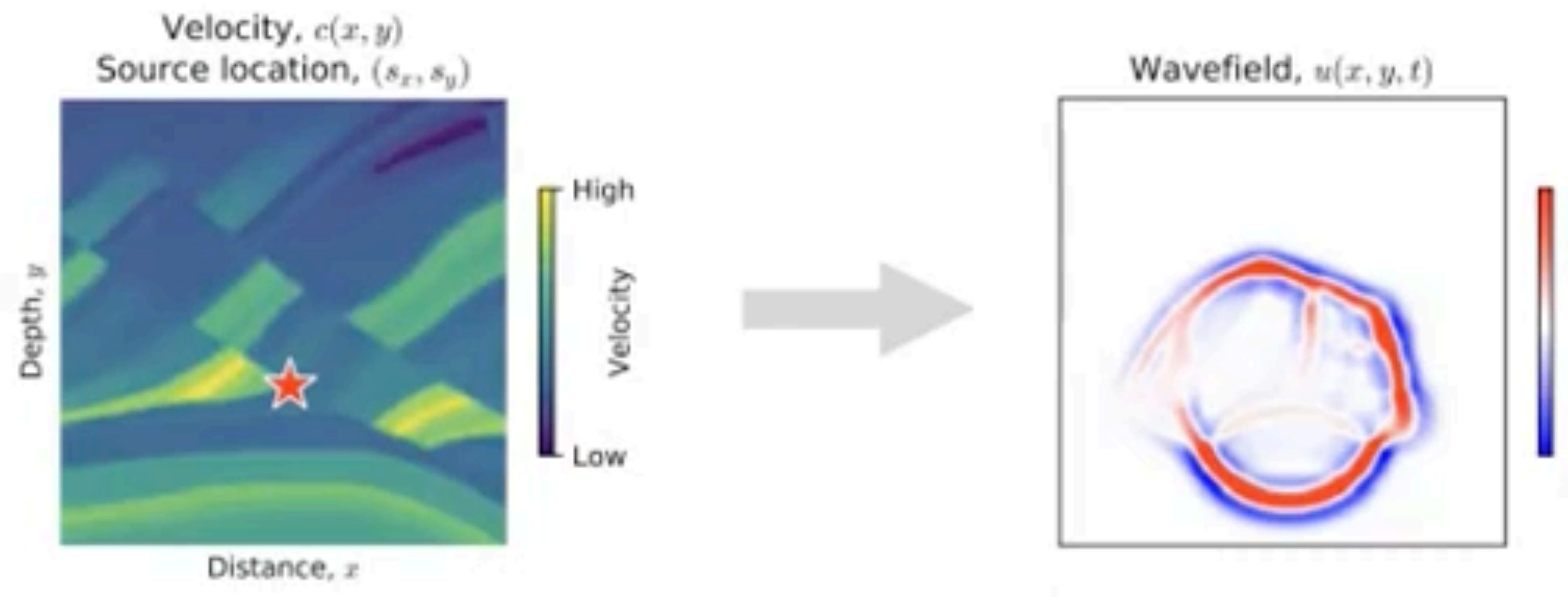
Equation Discovery

- Historically, F (= laws of physics) has been found through remarkable human intuition
- Constant interplay between theory and experiment
- From a computational standpoint, discovering physics is like solving an inverse problem (trying to fit a model to observed data)
- The model should be explainable, generalizable and make novel prediction

$$\boldsymbol{b = F(a)}$$

Applications

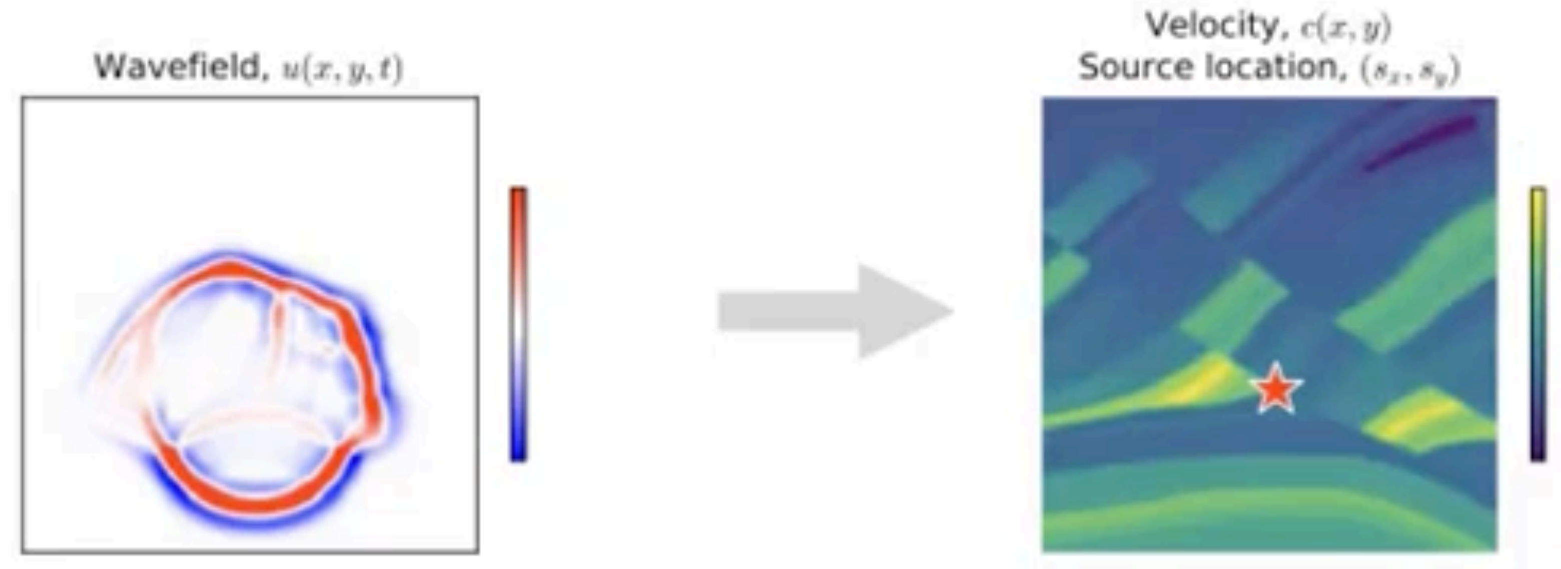
Recap



Forward simulation

Estimate wavefield $u(x,y,t)$
Given velocity $c(x,y)$
and source location (s_x, s_y)

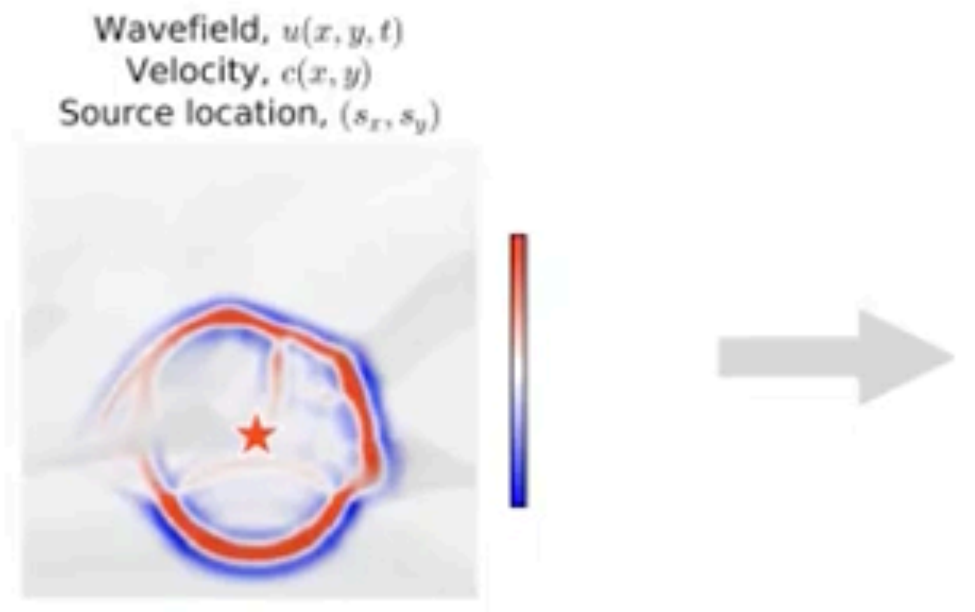
$$b = F(a)$$



Inversion

Estimate velocity $c(x,y)$
and source location (s_x, s_y)
Given wavefield $u(x,y,t)$

$$b = F(a)$$



Wave equation

$$\nabla^2 u - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = f$$

Equation discovery

Estimate governing equation
Given wavefield $u(x,y,t)$,
velocity $c(x,y)$,
and source location (s_x, s_y)

$$b = F(a)$$

SOLUTION

Differential Operator

In the context of solving a PDE using a neural network

- $D(u) = f$
- This non-linear function involves u's gradients and second derivatives, indicating a local non-linear function.
- This function can be evaluated through backpropagation, implying that the neural network can approximate the PDE's solution.
- All the inputs are given through the source term f, to represent the external influences terms that drive the system.
- To simplify the notation, temporarily set aside the boundary and initial conditions, which are essential components of a well-posed PDE problem.

Deep Neural Networks

- $u^*(y) = DNN$
- At the k-th Hidden layer:
  $y^{k+1} := \sigma(C_k y^k) = \sigma(W^k y^k + B^k)\ ,\ \theta = \{W_k, B_k\} \in \Theta$
- σ is a Scalar Activation Function (e.g. Sigmoid, Tanh).
- Don’t use ReLU, because it's not differentiable at certain points.

The key idea is to find a neural network that approximates the exact solution of the PDE.

Physics-Informed Neural Networks for the PDE

PDE Residual

- To do this, the network must be able to compute the residual, which is defined as the difference between the PDE and the neural network's output using backpropagation.

  $\theta \in \Theta$ such that $u_\theta \approx u_{true}$

  $R := R_\theta(y) = D(u_\theta(y)) - f(y)$

- The loss function is not just a supervised loss but also includes terms related to the PDE, which makes the training process more challenging. Set up the model, identify quadrature points, and minimize the loss (e.g. ADAM, L-BFGS, ...).

Physics-Informed Neural Networks for the PDE

Loss Function

- The residual loss ensures that the network's output satisfies the PDE:

$$\mathscr{L}_{PDE} = R(x)^2$$

$$R(x) = \|R\|_Y^{p=2} \sim \int |R|^p\, dy \longrightarrow S = \{y_i\}_{1\le i\le N}\ (\textit{quadrature points with weights } w_i)$$

- PINN for approximating PDE is defined as $u* = u_{\theta*}$

$$\theta* = arg \min_{\theta\in\Theta} \sum_{i=1}^{N} w_i |R_\theta(y_i)|^p$$

- A data loss (if available) ensuring the network's output fits any given training data:

$$\mathscr{L}_{data} = \sum (u^*_\theta - u_{true})^2$$

Physics-Informed Neural Networks for the PDE

Recap

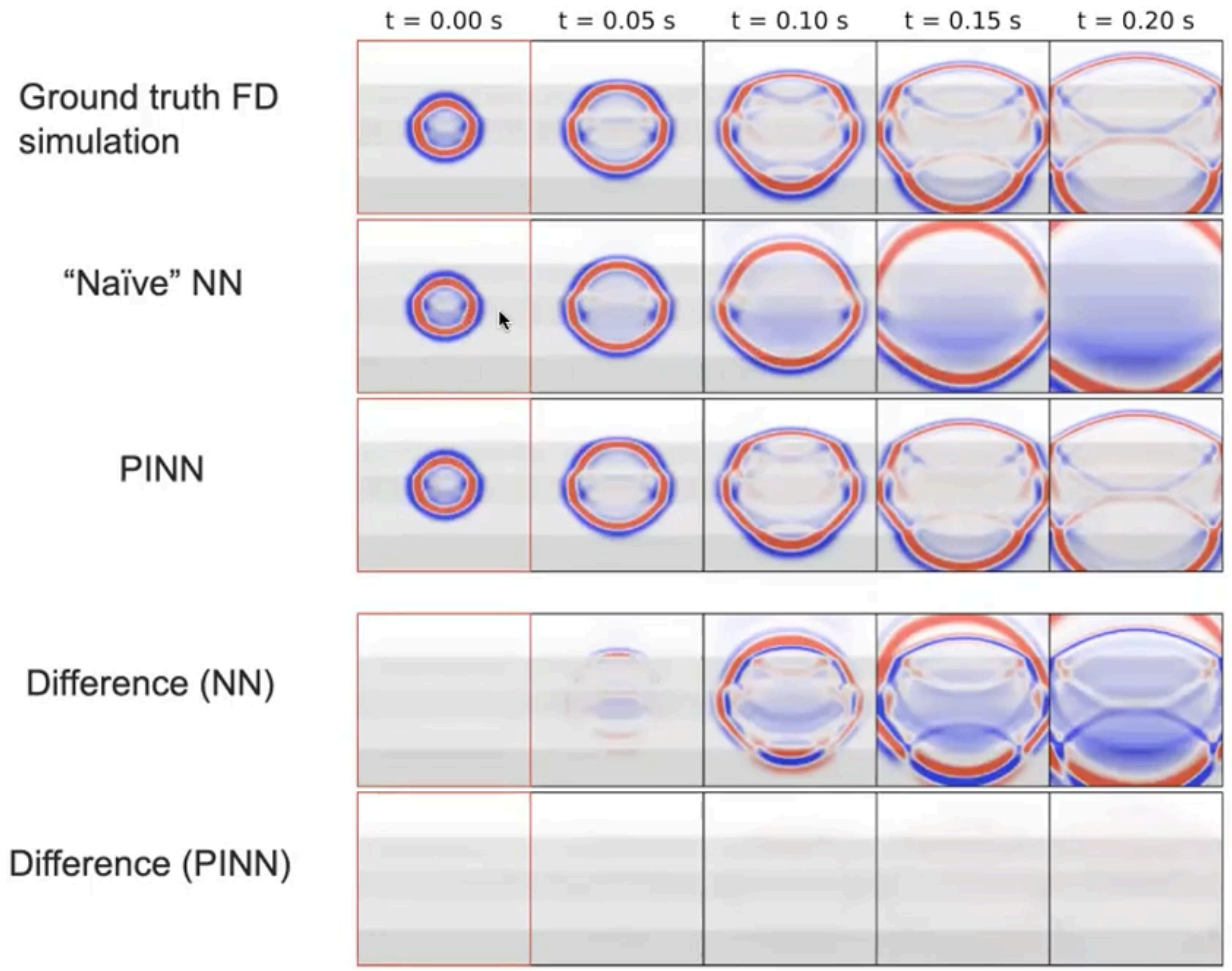
- The goal is the approximation u_θ is close to the true solution u. Quantified by minimizing the total error $\mathscr{L} = \|u - u_\theta\|$
- In practice, don't have access to the exact solution u at all points in the domain.
- R_θ is the residual of the PDE, which measures how well the NN approximation satisfies the PDE. PINNs aim to minimize the PDE residual $\|R_\theta\|$. A smaller residual indicates that the NN solution u_θ better satisfies the PDE.
  $\mathscr{L}_{Generalized} = \|R_\theta\| = \|D(u_\theta) - f\|$
- In practice, we only have access to a finite set of training points $\{y_i\}_{i=1}^N$ where we evaluate the residual.
- $\mathscr{L}_{Training} = (\sum_{i=1}^{N} w_i \,|\, R_\theta(y_i) \,|^p)^{\frac{1}{p}}$

While the algorithm is simple, the devil is in the details.
PINNs work in many cases, but not always.

Wave Equation

Example

- Fully connected with 10 layers
- 1024 hidden units
- Training time: ~1 hour



B. Moseley, A. Markham, and Tarje Nissen-Meyer, “Solving the wave equation with physics-informed deep learning,” arXiv (Cornell University), Jan. 2020, doi: https://doi.org/10.48550/arxiv.2006.11894.

1D Poisson equation

Example

- $\frac{d^2u}{dx^2} = f(x)$

- Neural Network represents $u(x)$ as $\hat{u}(x;\theta)$ and computes the second-order derivative of $\hat{u}$ with respect to x using backpropagation.

  $$R(x) = \frac{d^2\hat{u}}{dx^2} - f(x)$$

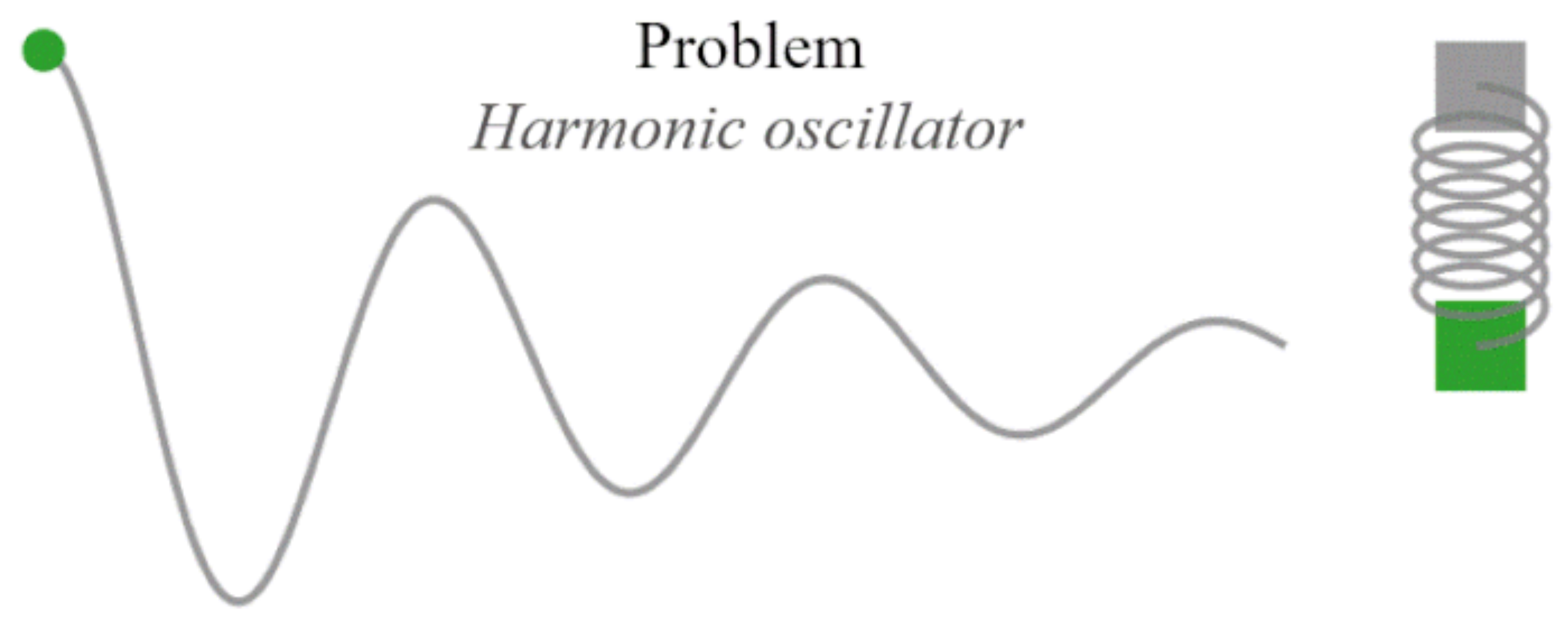
  $$\mathcal{L}_{PDE} = \sum R(x)^2$$

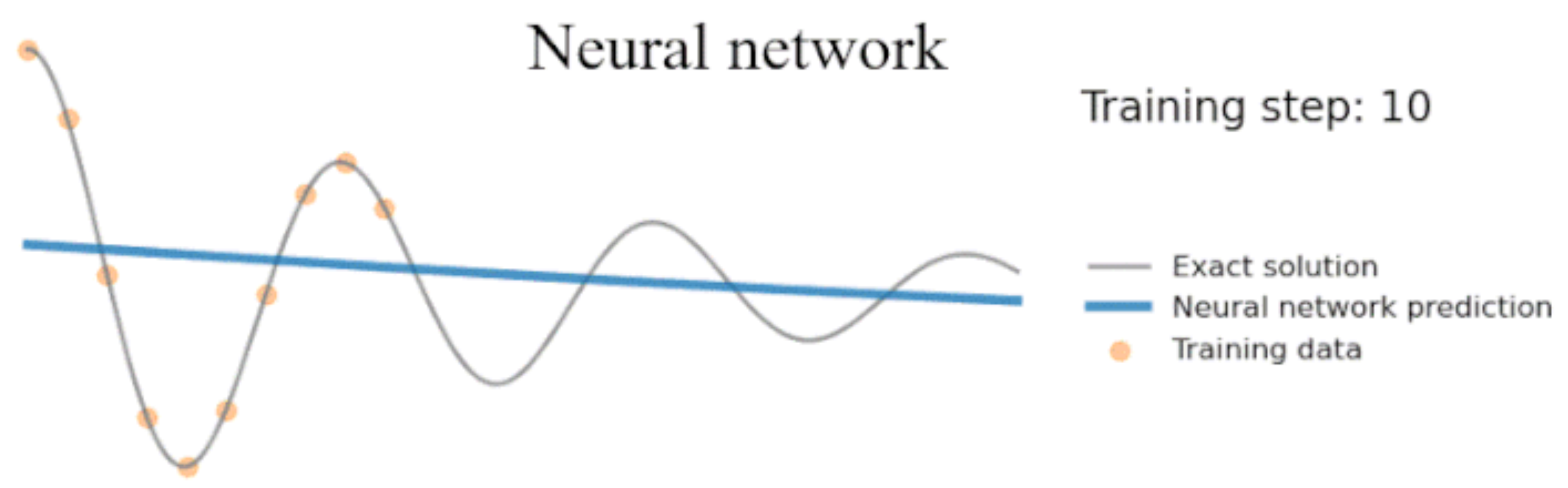
- $\mathcal{L}(\theta) = Supervised + Physics(aka\ PDE\ residual)$

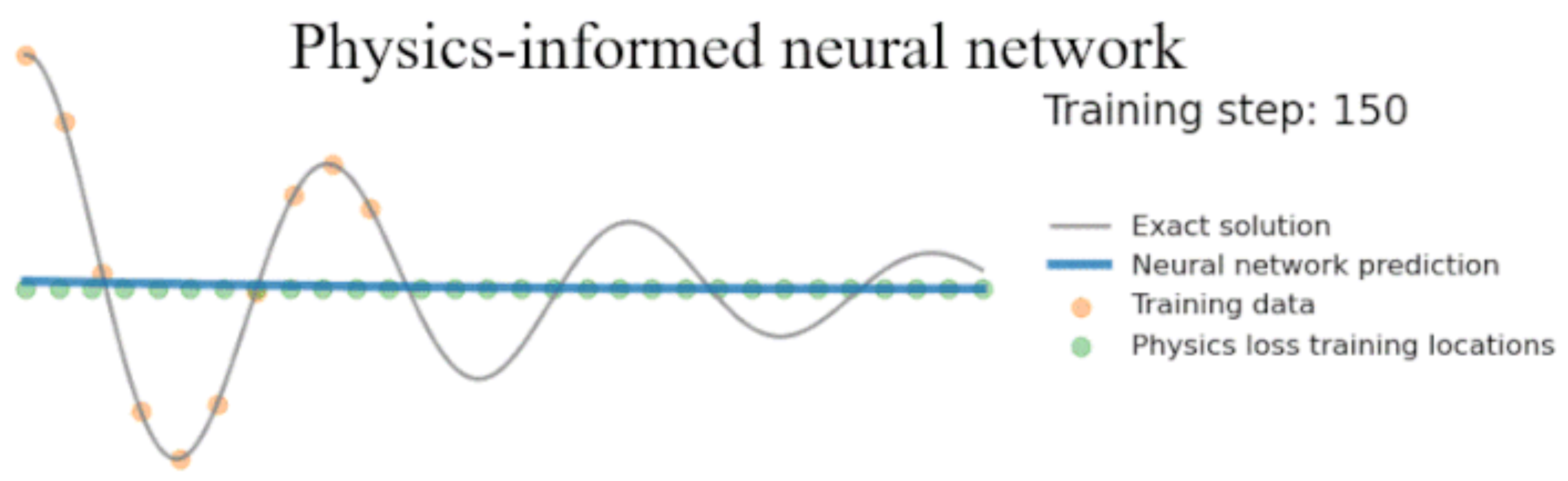
Harmonic oscillator

Example

- $m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0$
- $L(\theta) = \frac{1}{N}\sum (NN(t_i;\theta) - \underline{u_i})^2$
- $\mathscr{L}(\theta) = Supervised + Physics(aka\ PDE\ residual)$
- $L(\theta) = \frac{1}{N}\sum_i (NN(t_i;\theta) - \underline{u_i})^2 + \frac{\lambda}{M}\sum_j ([m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k](\underline{NN(t_j;\theta)})^2$



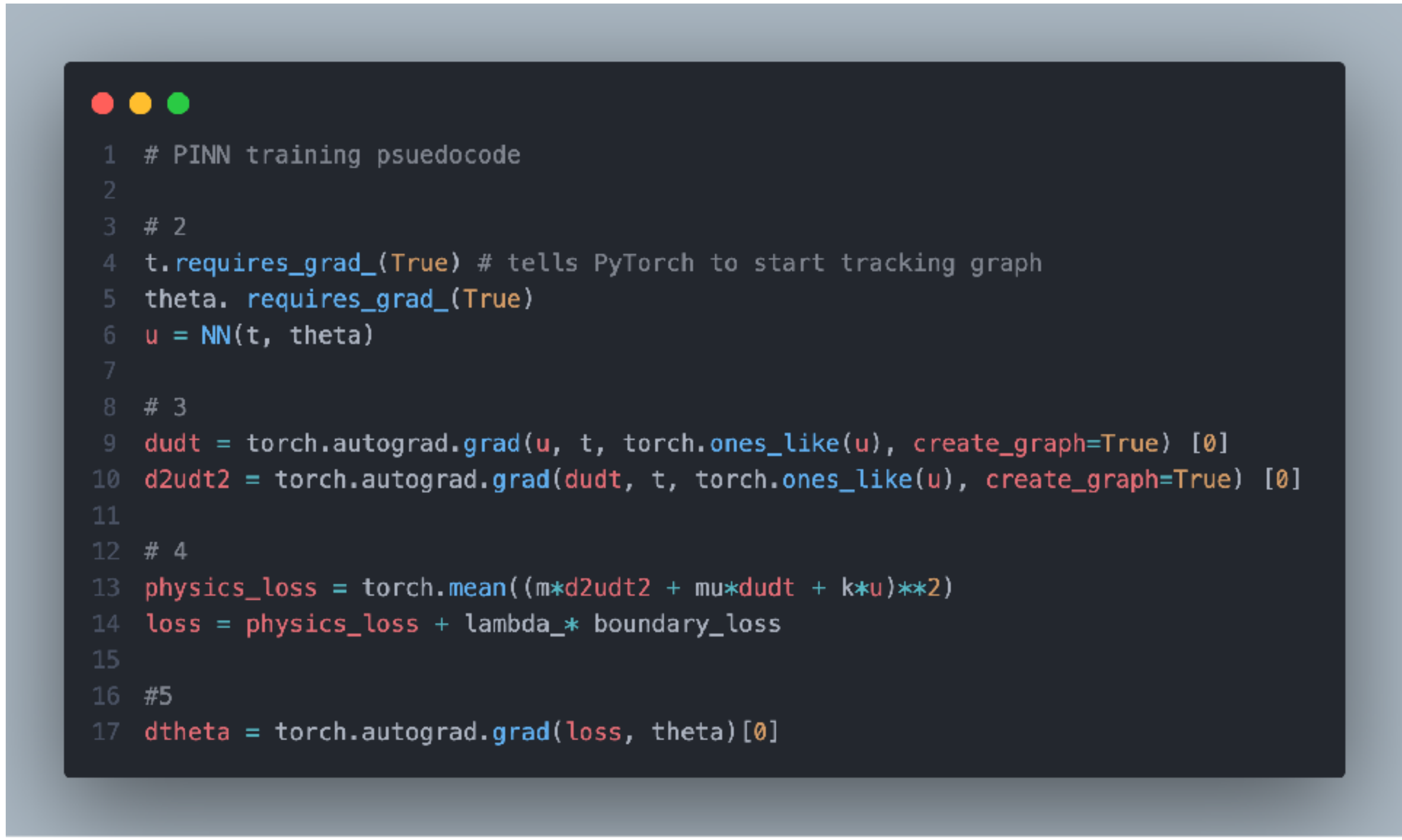




A CLOSER LOOK

PINN training loop

1. Sample boundary/ physics training points
2. Compute network outputs
3. Compute 1st and 2nd order gradient of network output with respect to network input
4. Compute loss
5. Compute gradient of loss function with respect to network parameters
6. Take the gradient descent step



```python
# PINN training psuedocode

# 2
t.requires_grad_(True) # tells PyTorch to start tracking graph
theta. requires_grad_(True)
u = NN(t, theta)

# 3
dudt = torch.autograd.grad(u, t, torch.ones_like(u), create_graph=True) [0]
d2udt2 = torch.autograd.grad(dudt, t, torch.ones_like(u), create_graph=True) [0]

# 4
physics_loss = torch.mean((m*d2udt2 + mu*dudt + k*u)**2)
loss = physics_loss + lambda_* boundary_loss

#5
dtheta = torch.autograd.grad(loss, theta)[0]
```

CONCLUSION

Physics-Informed Neural Networks for the PDE

Loss Function

- $\mathscr{L}(\theta) = Supervised + Physics(aka\ PDE\ residual)$

  $\mathscr{L}_{Pyisics} = (\sum_{i=1}^{N} w_i \,|\, R_\theta(y_i) \,|^p)^{\frac{1}{p}}$, $\|R_\theta\| = \|D(u_\theta) - f\|$, $\mathscr{L}_{data} = \sum (u_\theta^* - u_{true})^2$

- In case of Forward Simulation: $\mathscr{L}(\theta) = \mathscr{L}_p(\theta) + \mathscr{L}_b(\theta)$

  $$\mathscr{L}_b(\theta) = \sum_k \frac{\lambda_k}{N_{bk}} \sum_j^{N_{bk}} \|\mathscr{B}_k[NN(x_{kj};\theta)] - g_k(x_{kj})\|^2$$

- In case of Inverse Problems: $\mathscr{L}(\theta, \phi) = \mathscr{L}_p(\theta, \phi) + \mathscr{L}_d(\theta)$

  $\mathscr{L}_p(\theta, \phi) = \frac{1}{N_p} \sum_i^{N_p} \|\mathscr{D}[NN(x_i;\theta);\phi] - f(x_i)\|^2$, $\mathscr{L}_d(\theta) = \frac{\lambda_k}{N_d} \sum_j^{N_d} \|[NN(x_l;\theta)] - u_l\|^2$

Key Points

Concolusion

- Physics-informed neural networks (PINNs) incorporate physical laws into the learning process, usually as differential equations.
- The training of a PINN involves minimizing a loss function that includes terms for both data fidelity and adherence to the physical laws.
- PINNs have been applied successfully in fields such as material science, where they are used for tasks like simulating and modeling.

Questions
Generalization

How well do PINNs generalize across different types of PDEs, particularly those not included in the training set?

- PINNs can generalize across various types of PDEs if the underlying physical laws are similar.
- However, generalization to entirely novel PDEs, especially with different boundary conditions or non-linearities, may require retraining or adapting the network architecture.

Questions

Convergence Properties

Given that PINNs use gradient descent methods, can you discuss their convergence properties?

- The convergence of PINNs depends on the architecture, optimization, and the PDE.
- Convergence proofs are limited and generally focus on simpler, well-behaved PDEs.
- In practice, empirical testing is often necessary to evaluate convergence.

Questions

Handling Non-linear Systems

How effective are PINNs in handling highly non-linear PDEs?

- PINNs have shown promising results in handling non-linear PDEs, particularly where traditional methods struggle with stability or require fine discretization.

Pros

- **Mesh free**
- Can perform well for high-dimensional PDES
- Can be **extended** to solve inverse and discovery problems
- Often perform best on **"messy/mixed"** problems, where some noisy data is available, physics is perhaps not perfectly known, and traditional algorithms require many forward evaluations
- Tractable
- Mostly **unsupervised**

Cons

- **Computational costs** are often high, especially for forward-only problems (requires retraining each time)
- Not guaranteed to converge/ **convergence** properties less well understood
- Challenging to **scale** to more complex problems (larger domains, multi-scale, multi-physics)
- Often not obvious what appropriate neural network architecture is
- But.. many PINN extensions exist!

Thank You

M.shafieeha@gmail.com