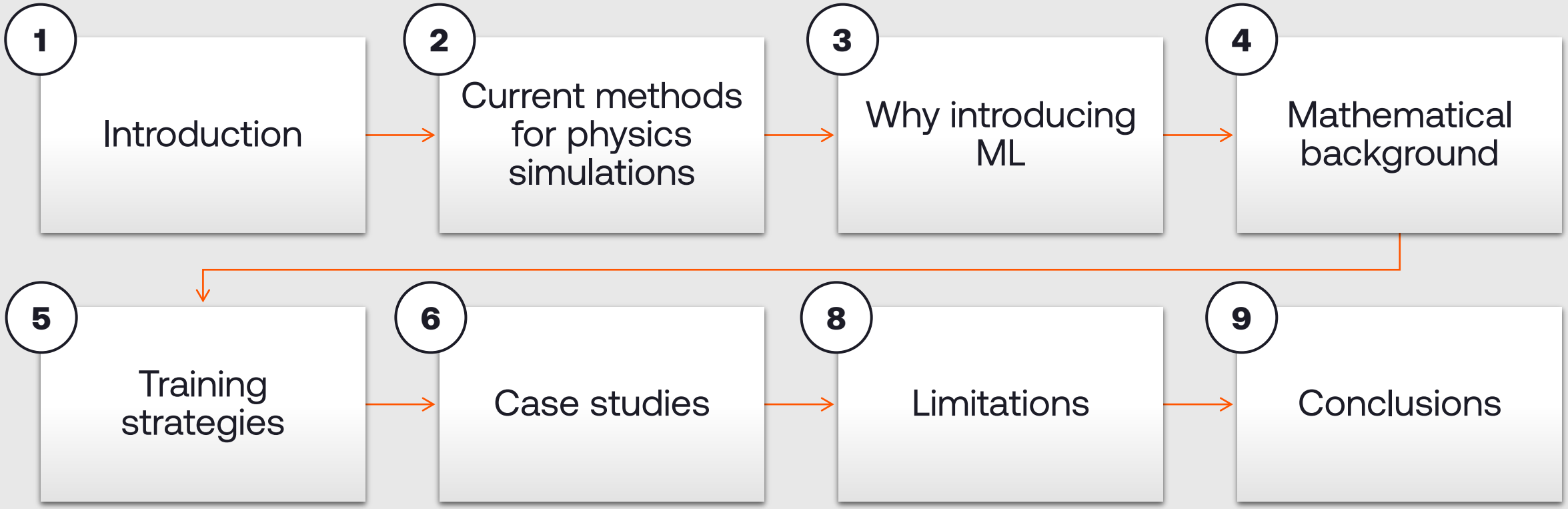
BRAID

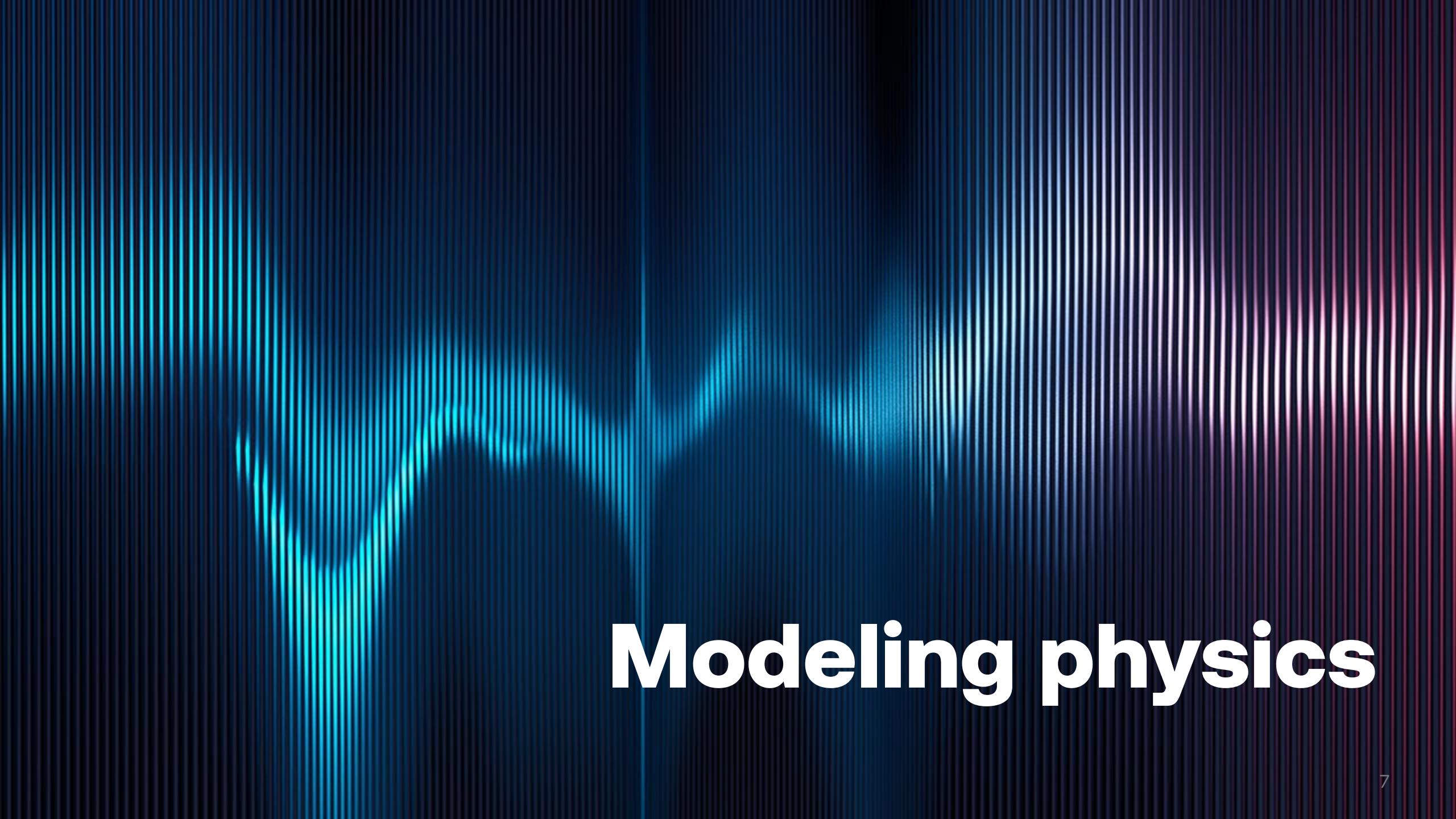
Guido Cossu PhD
Technology & Creative Director

Agenda


1
Introduction
2
Current methods for physics simulations
3
Why introducing ML
4
Mathematical background
5
Training strategies
6
Case studies
8
Limitations
9
Conclusions

What you will learn

- **Panoramic view** of physics problems and current solution methods
- How can ML be **integrated** and be **helpful** in *modeling of physical phenomena*, in an age of abundant data
- **Physics Informed Machine Learning, a** relatively new area of ML:
  - What is it?
  - What are its key insights?
  - Its basic components
  - How you can setup your first problems yourself

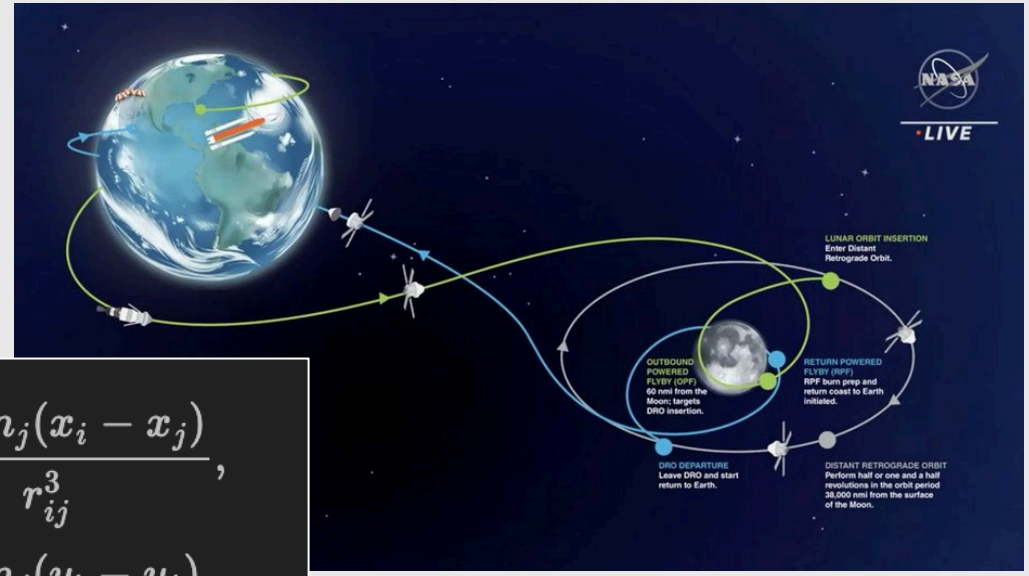

Modeling physics

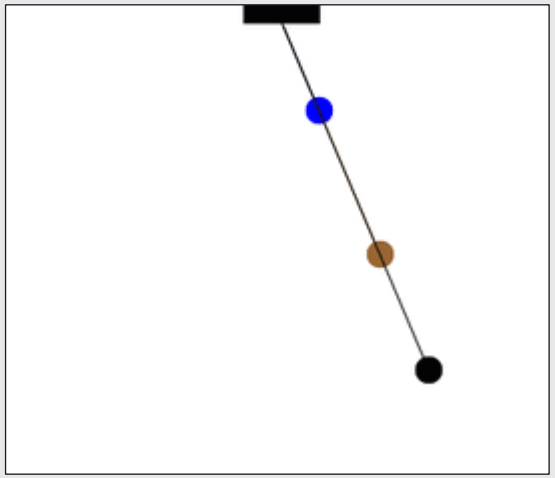
Simulating physics - ODE

Ordinary Differential Equations (ODEs)

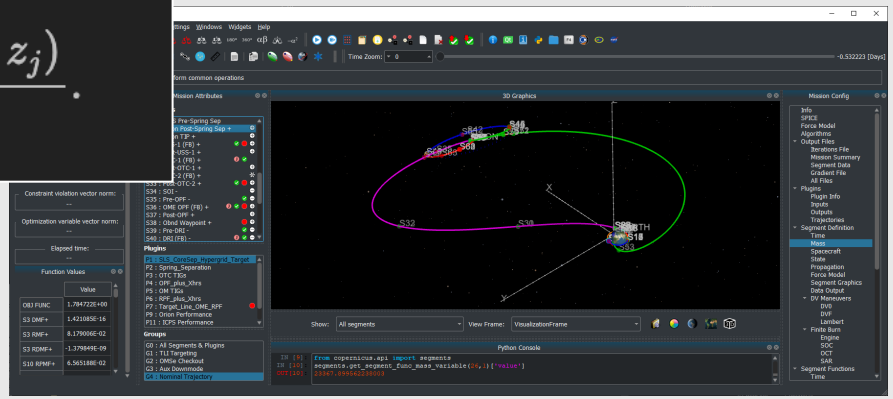
Examples: Planetary motion (Newton's laws), chemical kinetics, population growth.

Required: Initial conditions





$$m_i \frac{dv_{ix}}{dt} = \sum_{j=1}^{N} -\frac{Gm_i m_j (x_i - x_j)}{r_{ij}^3},$$
$$m_i \frac{dv_{iy}}{dt} = \sum_{j=1}^{N} -\frac{Gm_i m_j (y_i - y_j)}{r_{ij}^3},$$
$$m_i \frac{dv_{iz}}{dt} = \sum_{j=1}^{N} -\frac{Gm_i m_j (z_i - z_j)}{r_{ij}^3}.$$



$$\frac{d^2\theta(t)}{dt} + \frac{g}{l}\theta(t) = 0 \qquad \theta(t) = \theta_0 \cos\left(\sqrt{\frac{g}{l}}t\right)$$

Pendulum eq. (small angles approximation) $\qquad \theta(t) \ll 1$

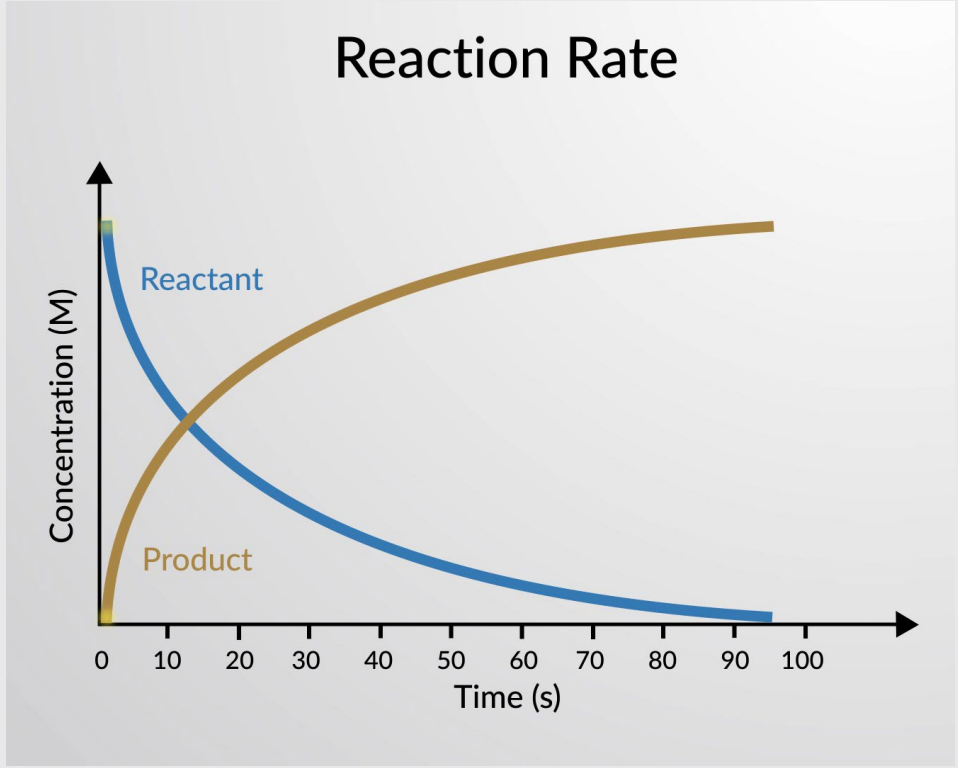
Simulating physics - ODE

Ordinary Differential Equations (ODEs)

Examples: Planetary motion (Newton's laws), chemical kinetics, population growth.

Required: Initial conditions

	Zero-Order	First-Order	Second-Order	*n*th-Order
Rate Law	$-\frac{d[A]}{dt} = k$	$-\frac{d[A]}{dt} = k[A]$	$-\frac{d[A]}{dt} = k[A]^2$[8]	$-\frac{d[A]}{dt} = k[A]^n$
Integrated Rate Law	$[A] = [A]_0 - kt$	$[A] = [A]_0 e^{-kt}$	$\frac{1}{[A]} = \frac{1}{[A]_0} + kt$[8]	$\frac{1}{[A]^{n-1}} = \frac{1}{[A]_0{}^{n-1}} + (n-1)kt$ [Except first order]
Units of Rate Constant (*k*)	$\frac{\mathrm{M}}{\mathrm{s}}$	$\frac{1}{\mathrm{s}}$	$\frac{1}{\mathrm{M}\cdot\mathrm{s}}$	$\frac{1}{\mathrm{M}^{n-1}\cdot\mathrm{s}}$
Linear Plot to determine *k*	$[A]$ vs. t	$\ln([A])$ vs. t	$\frac{1}{[A]}$ vs. t	$\frac{1}{[A]^{n-1}}$ vs. t [Except first order]
Half-life	$t_{1/2} = \frac{[A]_0}{2k}$	$t_{1/2} = \frac{\ln(2)}{k}$	$t_{1/2} = \frac{1}{k[A]_0}$[8]	$t_{1/2} = \frac{2^{n-1}-1}{(n-1)k[A]_0{}^{n-1}}$ [Except first order]



Simulating physics - PDE

Partial Differential Equations (PDEs)

Examples: Heat diffusion, wave propagation, electrodynamics, fluid dynamics (Navier-Stokes equation).

Required: boundary conditions, initial conditions

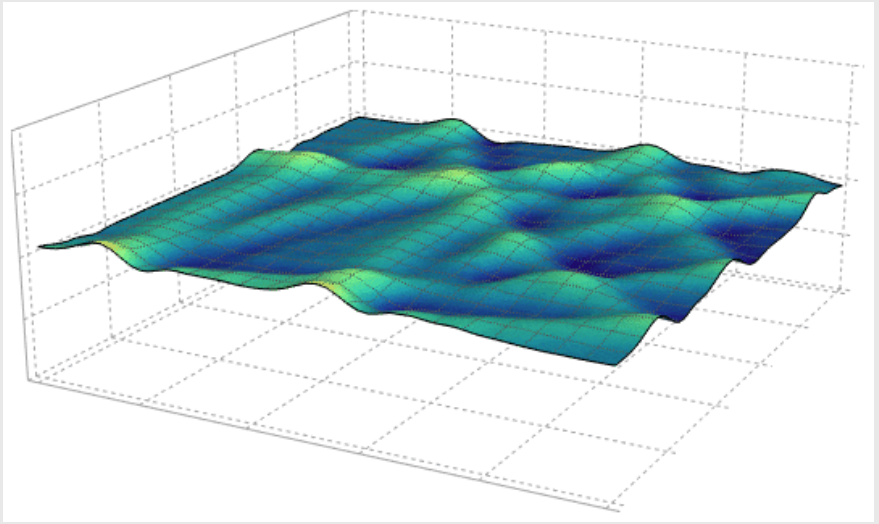
$$\frac{\partial^2 u}{\partial t^2} = c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right)$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \qquad \text{Gauss's Law}$$

$$\nabla \cdot \mathbf{B} = 0 \qquad \text{Gauss's Law for Magnetism}$$

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \qquad \text{Faraday's Law of Induction}$$

$$\nabla \times \mathbf{B} = \mu_0 \left(\epsilon_0 \frac{\partial \mathbf{E}}{\partial t} + \mathbf{J} \right) \qquad \text{Ampere's Circuital Law}$$



Simulating physics - PDE

Partial Differential Equations (PDEs)

Examples: Heat diffusion, wave propagation, electrodynamics, fluid dynamics (Navier-Stokes equation).

Required: boundary conditions, initial conditions

Continuity Equation

$$\nabla \cdot \vec{V} = 0$$

Momentum Equations

$$\rho \frac{D\vec{V}}{Dt} = -\nabla p + \rho \vec{g} + \mu \nabla^2 \vec{V}$$

Total derivative = $\rho[\frac{\partial V}{\partial t} + (V.\nabla)V]$

Change of velocity with time

Convective term

Pressure gradient

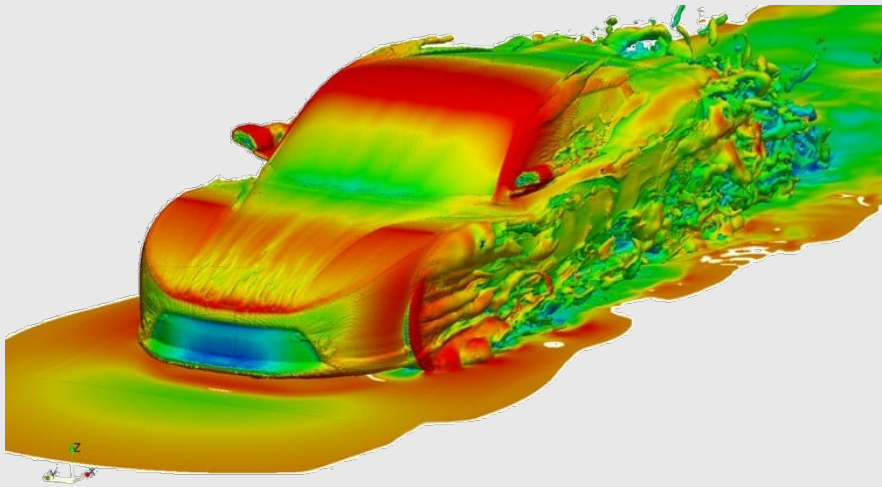
Fluid flows in the direction of largest change in pressure.

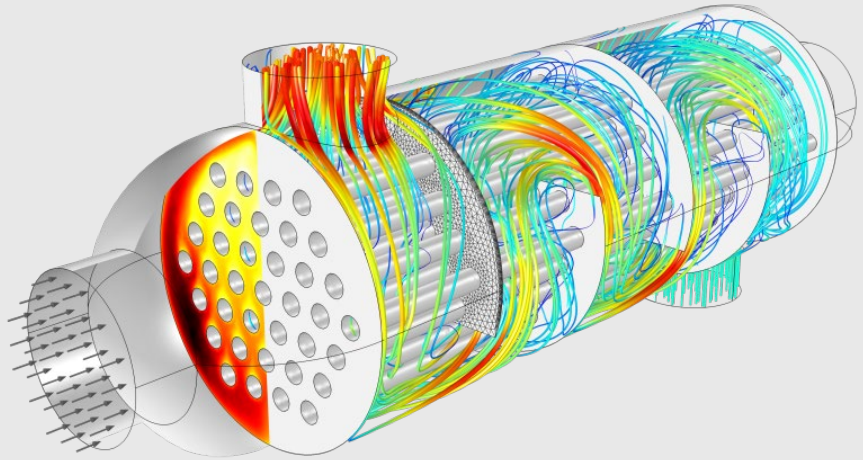
Body force term

External forces, that act on the fluid (gravitational force or electromegnetic).

Diffusion term

For a Newtonian fluid, viscosity operates as a diffusion of momentum.



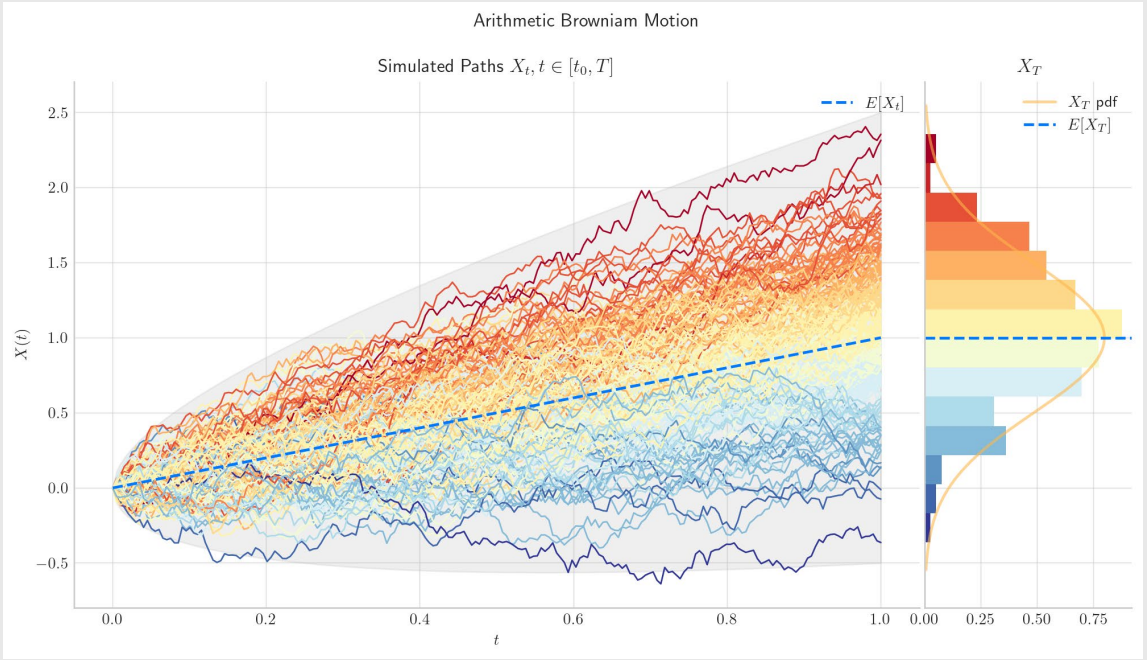


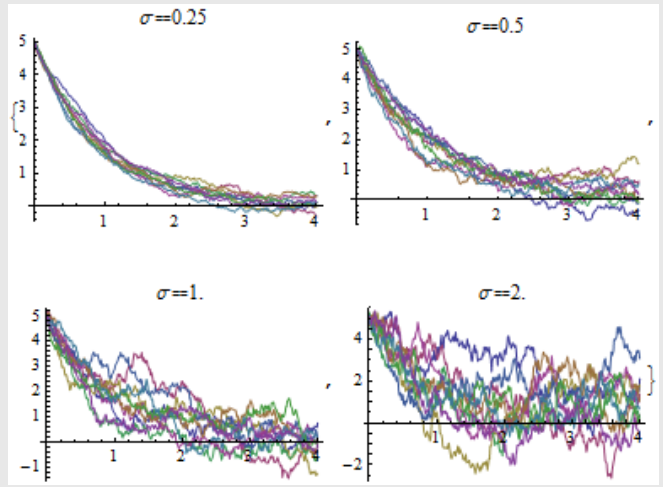
Simulating physics - SDE

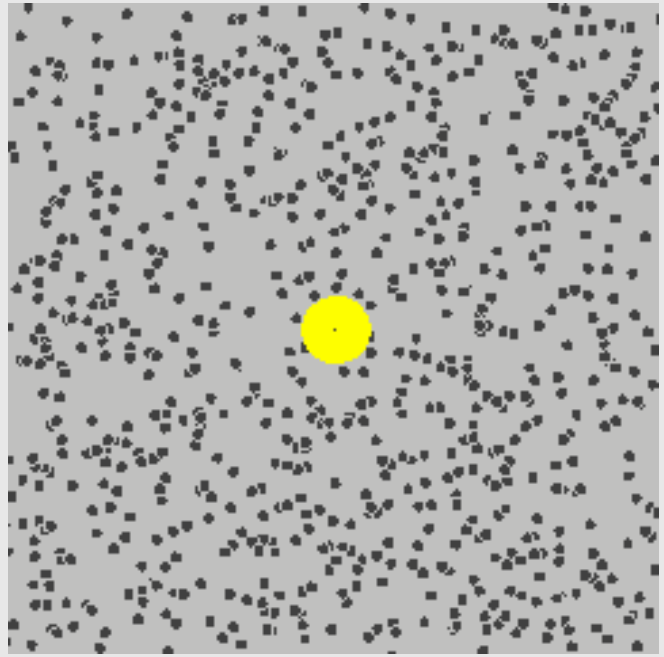
Stochastic Differential Equations (PDEs)

Examples: Stock price models (e.g., Black–Scholes), Brownian motion in physics, population models with random effects.

$$\mathrm{d}X_t = \mu(X_t, t)\,\mathrm{d}t + \sigma(X_t, t)\,\mathrm{d}B_t$$



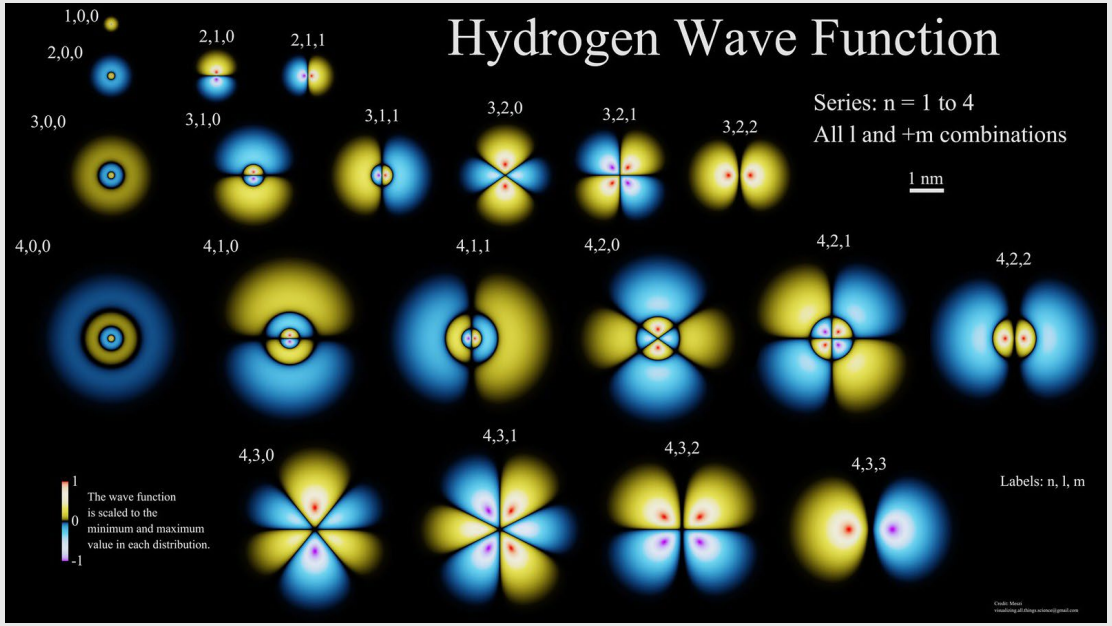




Simulating physics – QM

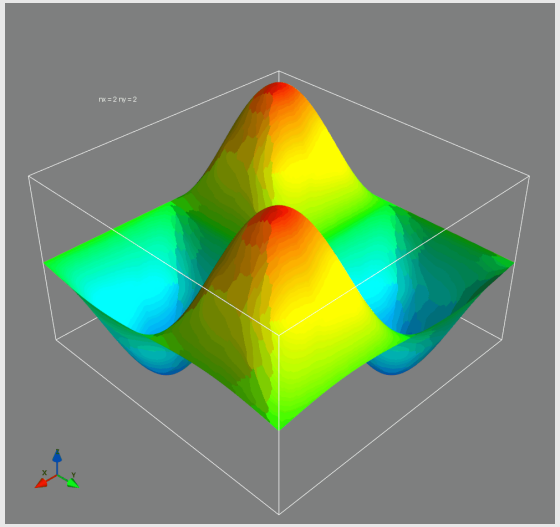
Quantum Mechanics

Examples: Time-dependent Schrödinger equation for electron wavefunctions, quantum harmonic oscillator, quantum field approximations.



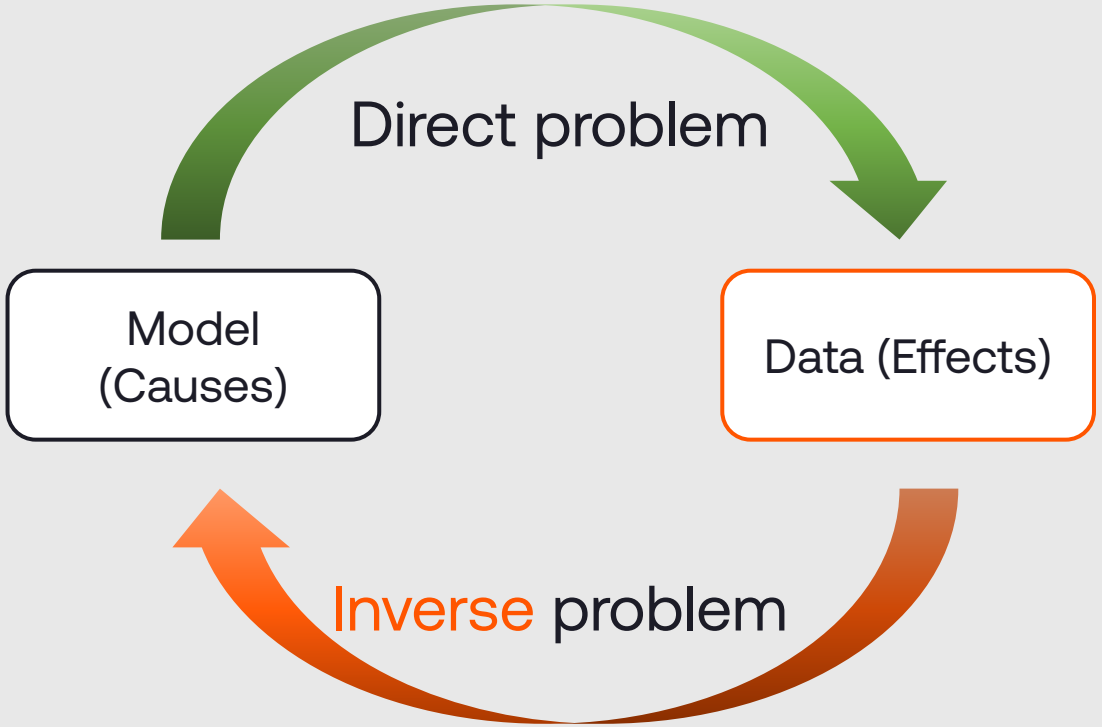
$$i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r},t)=\left[-\frac{\hbar^2}{2m}\nabla^2+V(\mathbf{r})\right]\Psi(\mathbf{r},t)$$

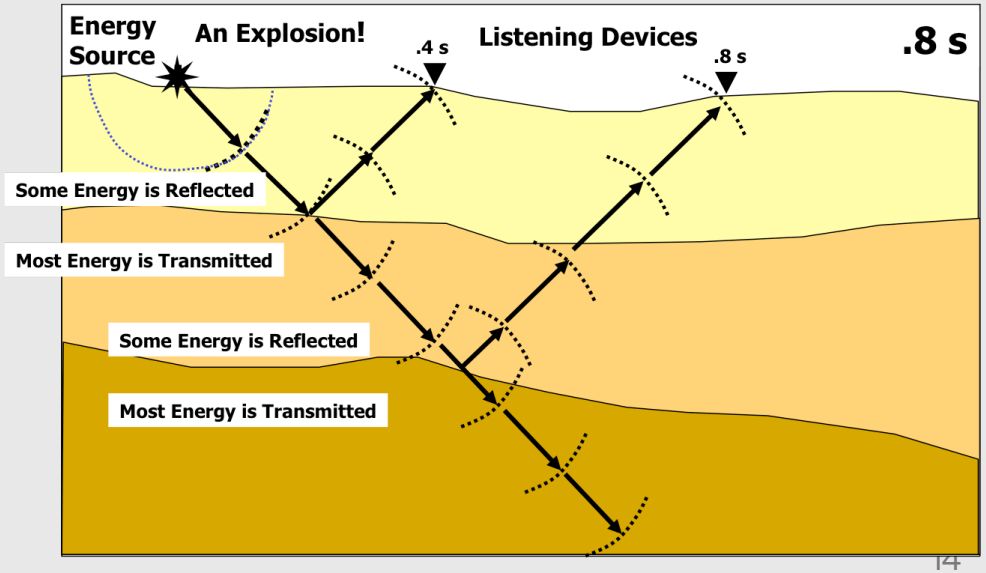
Time-dependent Schrödinger equation



Simulating physics – Inverse problems

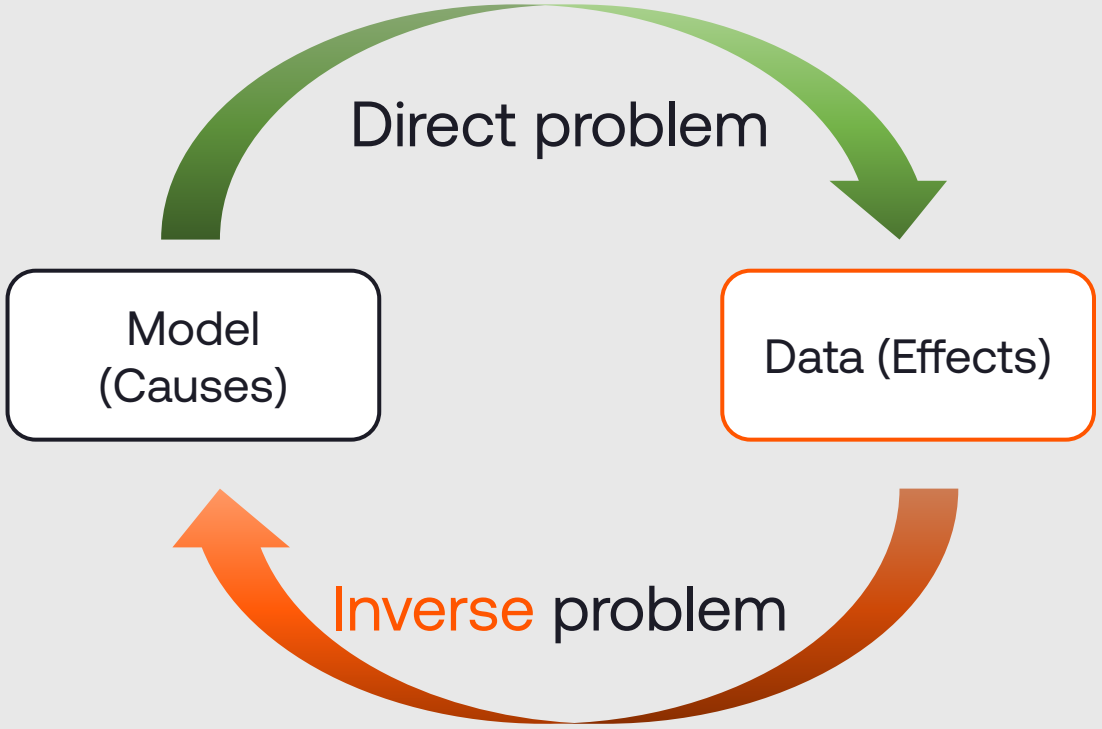
Examples: Parameter estimation (e.g., finding diffusion coefficients from observed data), medical imaging (MRI), geophysics (subsurface structure from seismic data).



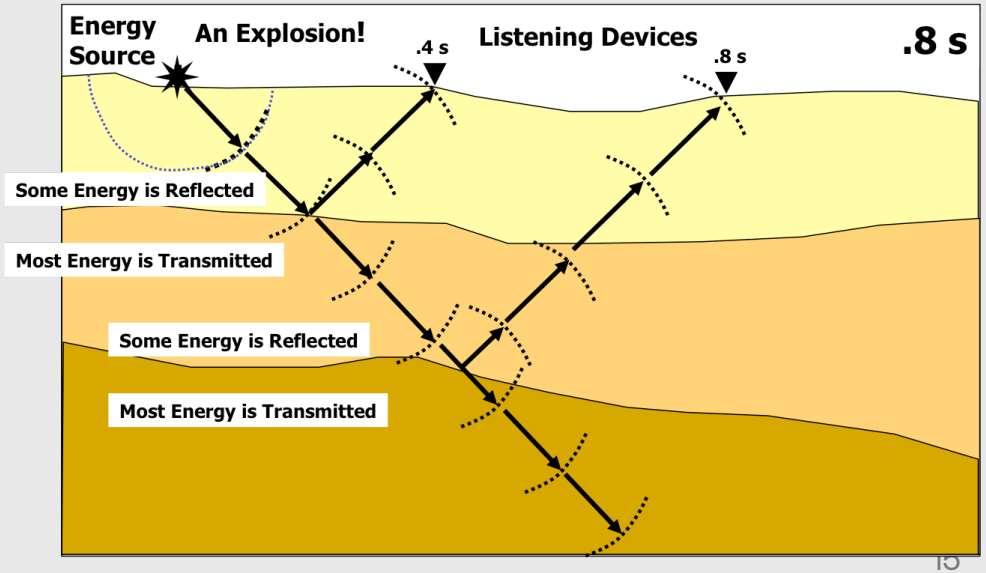


Simulating physics – Inverse problems

Examples: Parameter estimation (e.g., finding diffusion coefficients from observed data), medical imaging (MRI), geophysics (subsurface structure from seismic data).



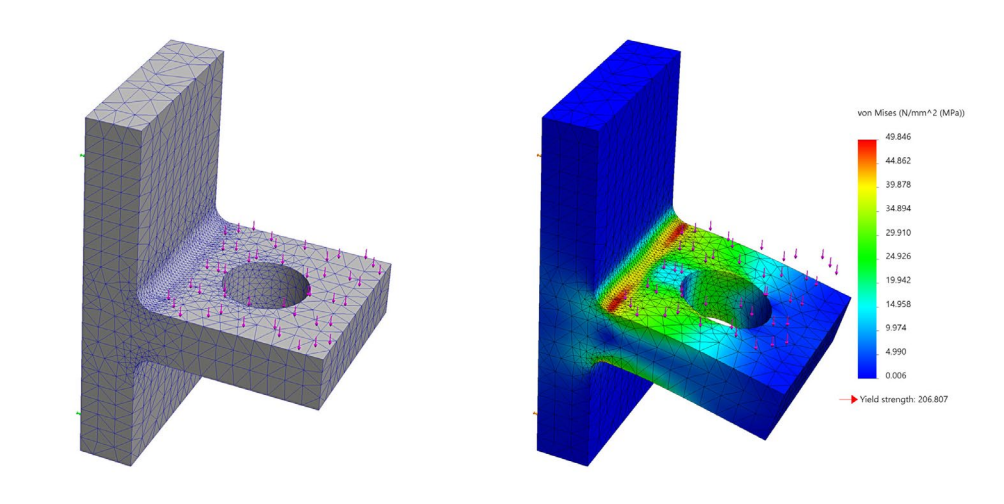
THIS IS DESIGN!

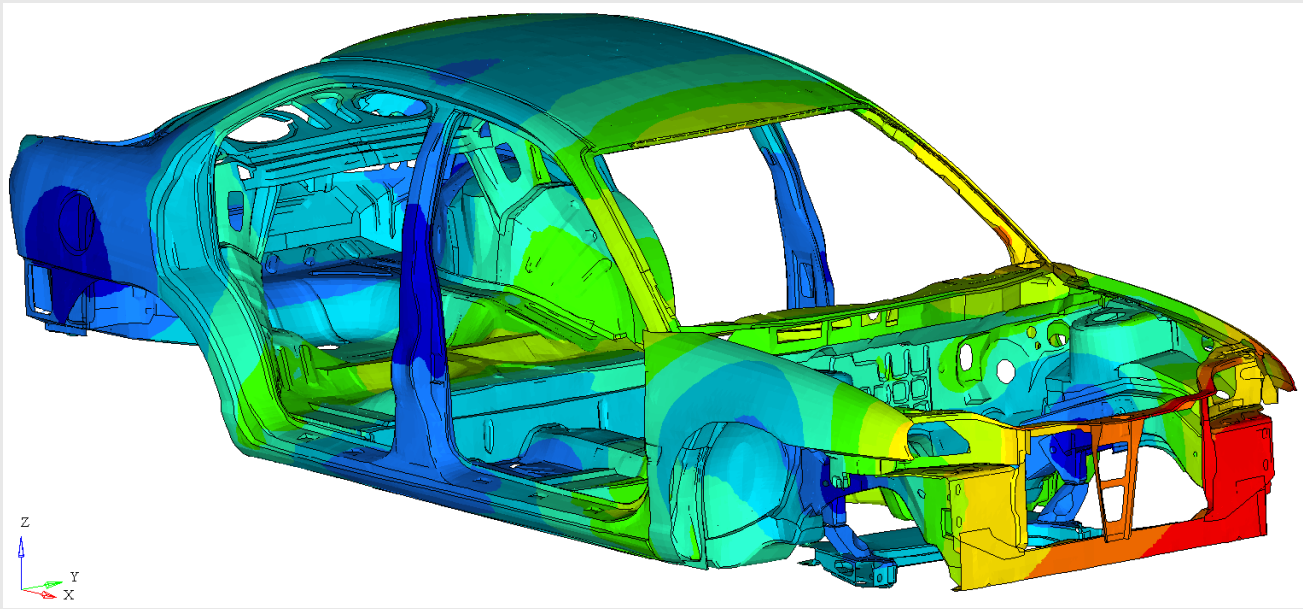


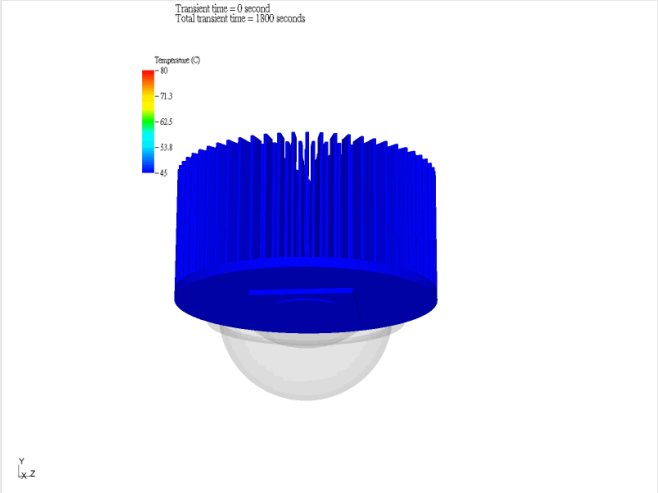
Simulations

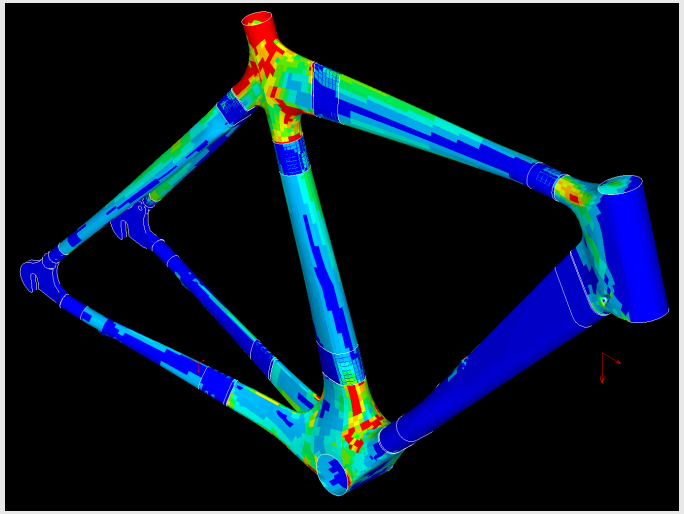
Current methods

Finite Element Method (FEM)







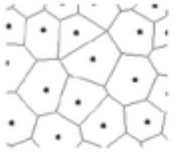


Current methods

Finite Volume Method (FVM) and **Finite Differences** (FDM)

Differential equation of flow problem

FVM



$$\frac{\partial U}{\partial x}$$

Integral form

$$\int_{CV} \frac{\partial U}{\partial x} dV$$

Divergence theorem

$$\int_{A} n \cdot U \, dA$$

FDM

$$\frac{\partial U}{\partial x}$$

Difference Equations

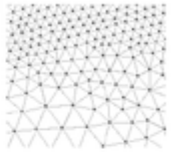
$$U_{i+1} = U_i + \Delta x \cdot \frac{\partial U}{\partial x}$$

$$U_{i-1} = U_i - \Delta x \cdot \frac{\partial U}{\partial x}$$

Taylor series

$$\frac{U_{i+1} - U_{i-1}}{2\Delta x}$$

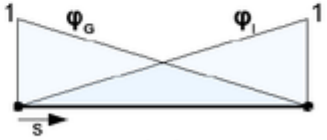
FEM



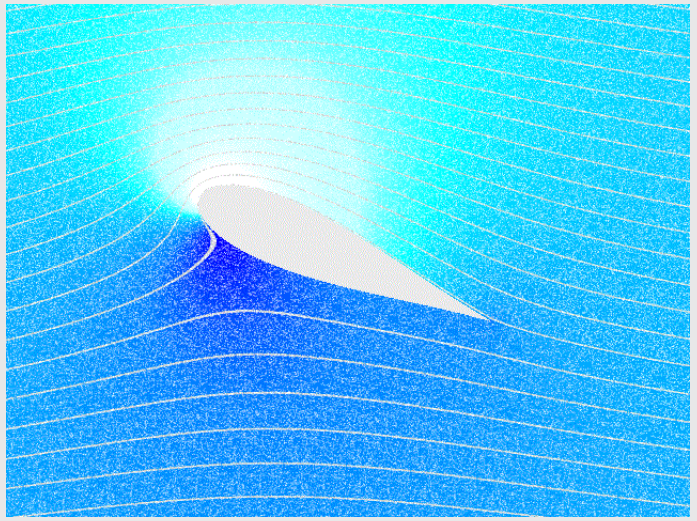
$$\frac{\partial U}{\partial x}$$

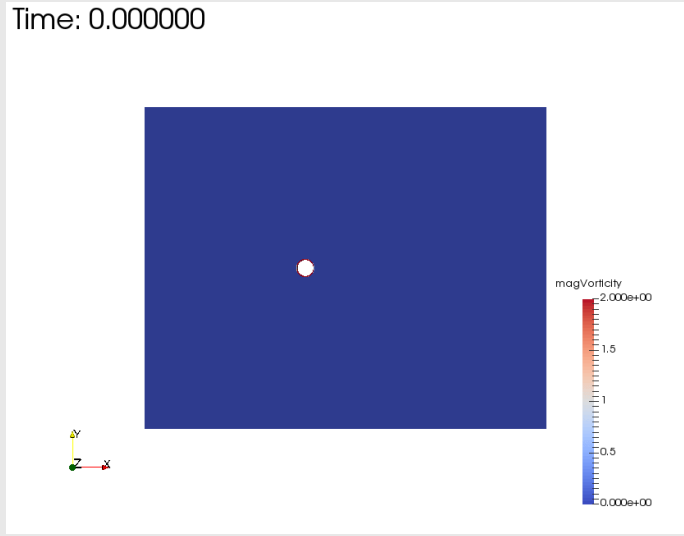
Using basis function

$$U = U_G \cdot \phi_G + U_I \cdot \phi_I$$



$$\frac{\partial U}{\partial x} = U_G \cdot \frac{\partial \phi_G}{\partial x} + U_I \cdot \frac{\partial \phi_I}{\partial x}$$

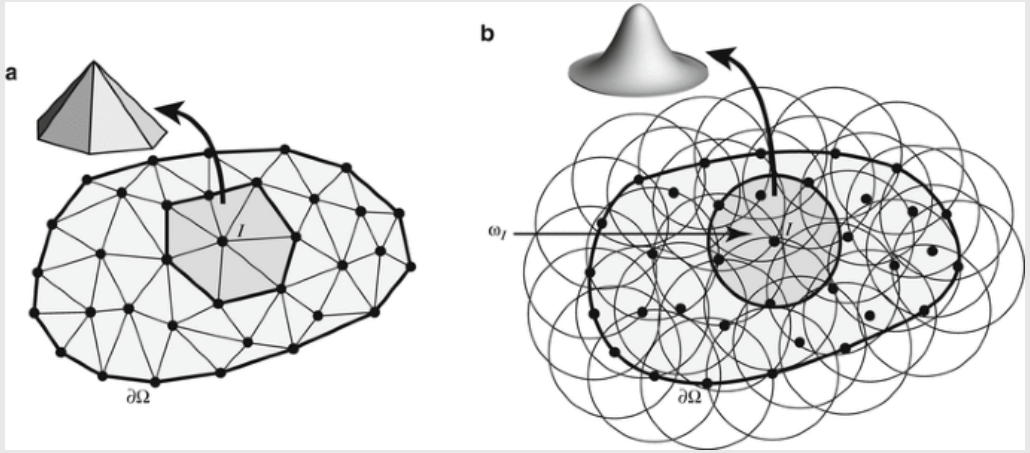




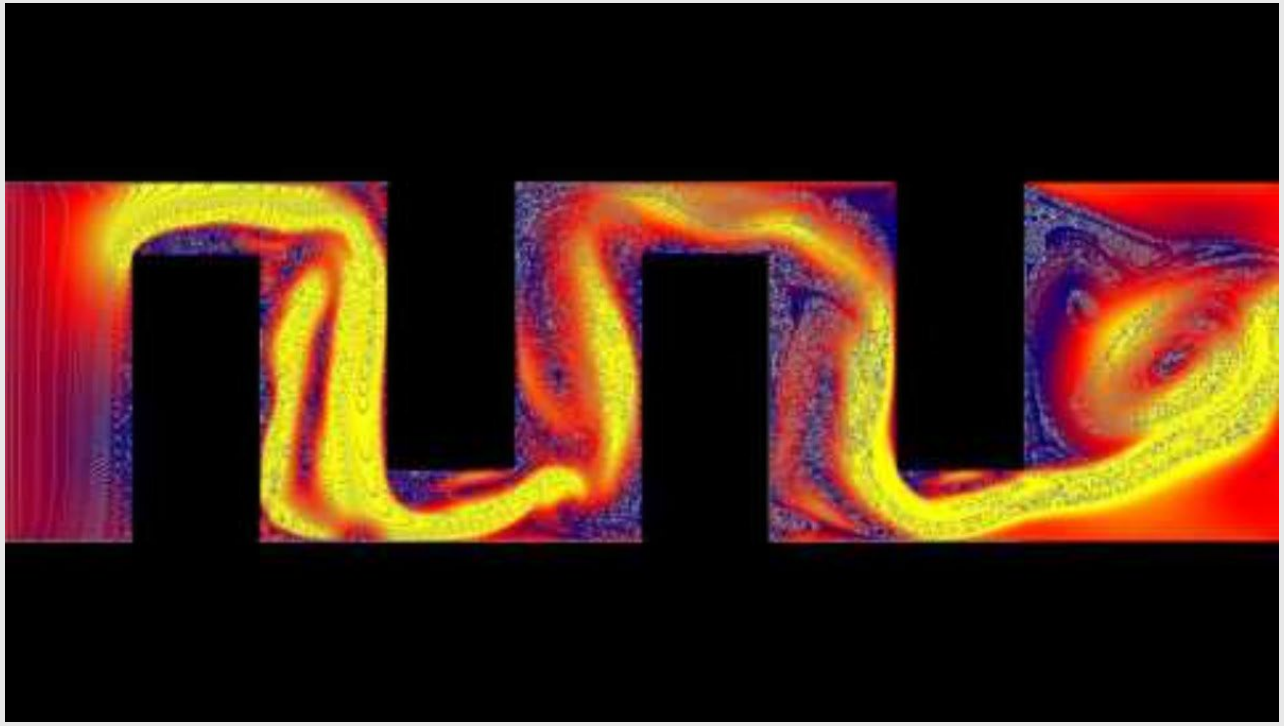
Current methods

Mesh-free methods

Radial Basis Function method



Lattice Boltzmann (particle method)



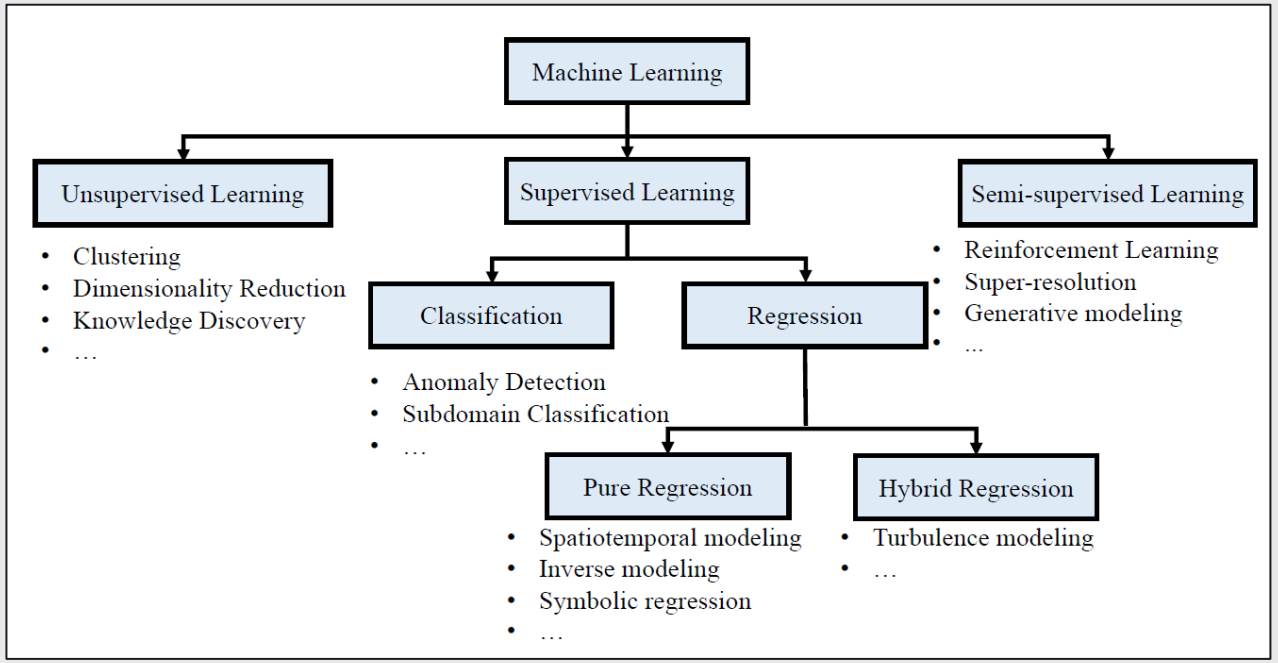
ML + Physics

ML for physics - Why

Why ML would be useful for physics modeling?

Scientific computing	**Neural Networks**
No data	Lots of data
Theoretical model	No theory

- ❑ Growing role of data-driven approaches in scientific discovery.
  - ❑ Data based model creation and inference
- ❑ Creation of surrogates for physical phenomena (surrogate modeling)
- ❑ NN are an ideal tool for data fitting
  - ❑ Variety of techniques developed



ML for physics

Pros

- ❑ Does **not require** deep knowledge of the physics domain
- ❑ Can make use of **all available collected data**
- ❑ **Low** computational cost once trained
- ❑ NN naturally model **non-linearities**

Cons

- ❑ May output results that are **physically inconsistent**
- ❑ Easy to learn spurious relationships that look good only on training and test data
  - ❑ Can lead to **poor generalization** outside the available data (out-of-sample prediction tasks)
- ❑ Interpretability is **absent**
  - ❑ Discovering the mechanism of an underlying process is crucial for scientific advancements

ML for physics

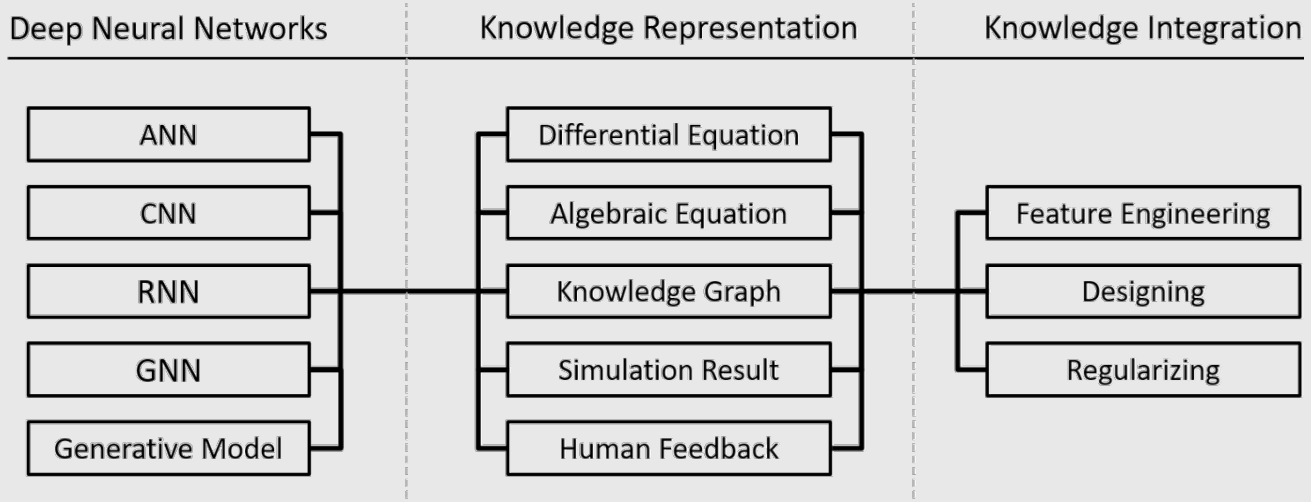
Combine physical knowledge with data-driven models

Do not relearn from data scientific knowledge we already have

Physics ∩ AI/ML

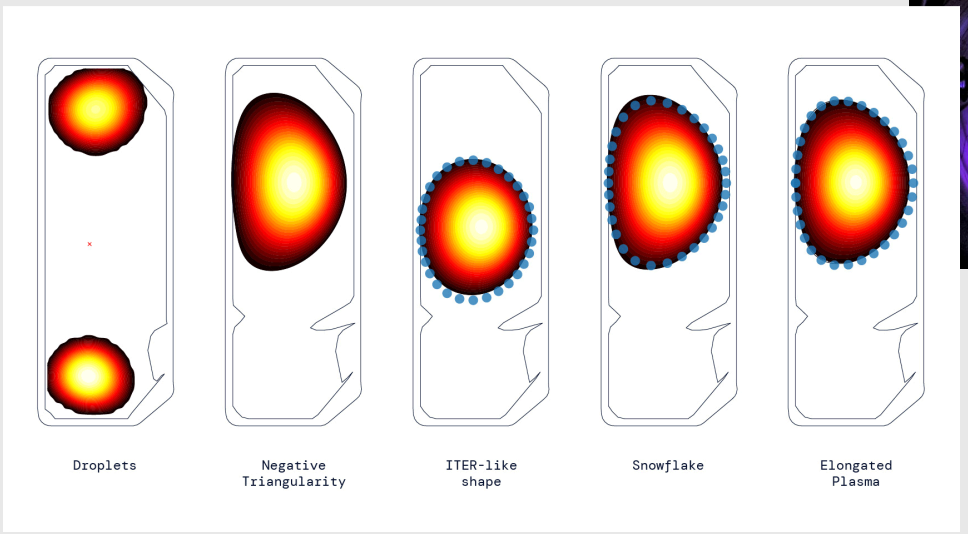
Scientific computing		**Neural Networks**
No data	?	Lots of data
Theoretical model		No theory

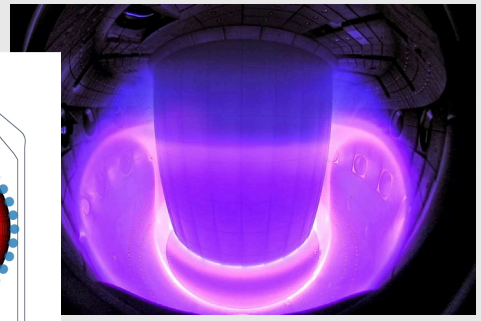
Informed Deep Learning (DL with inductive biases)

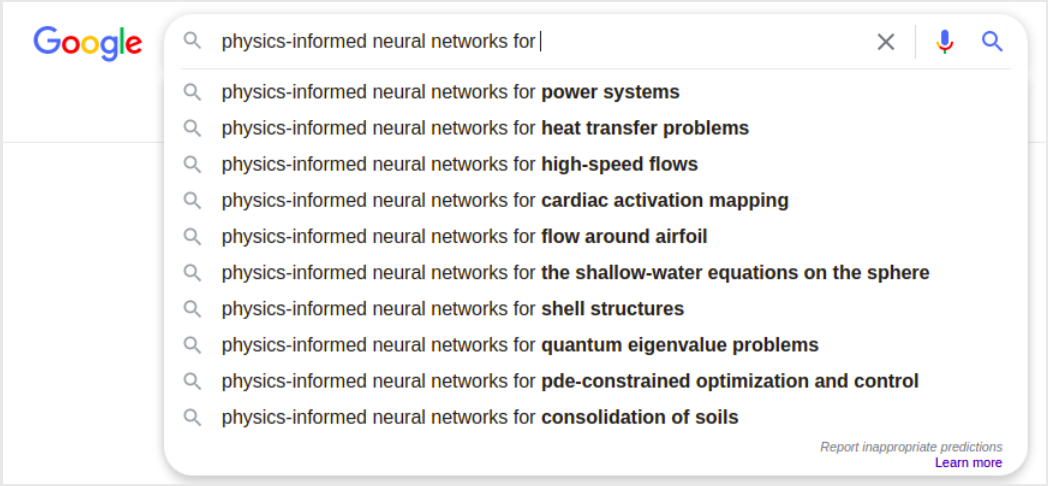


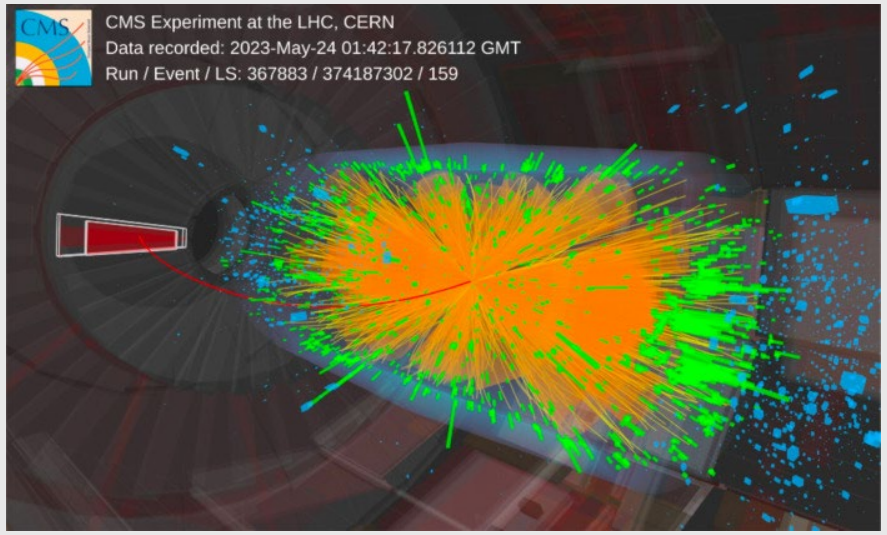
Data scarce vs data rich: amount of data available is a very important consideration

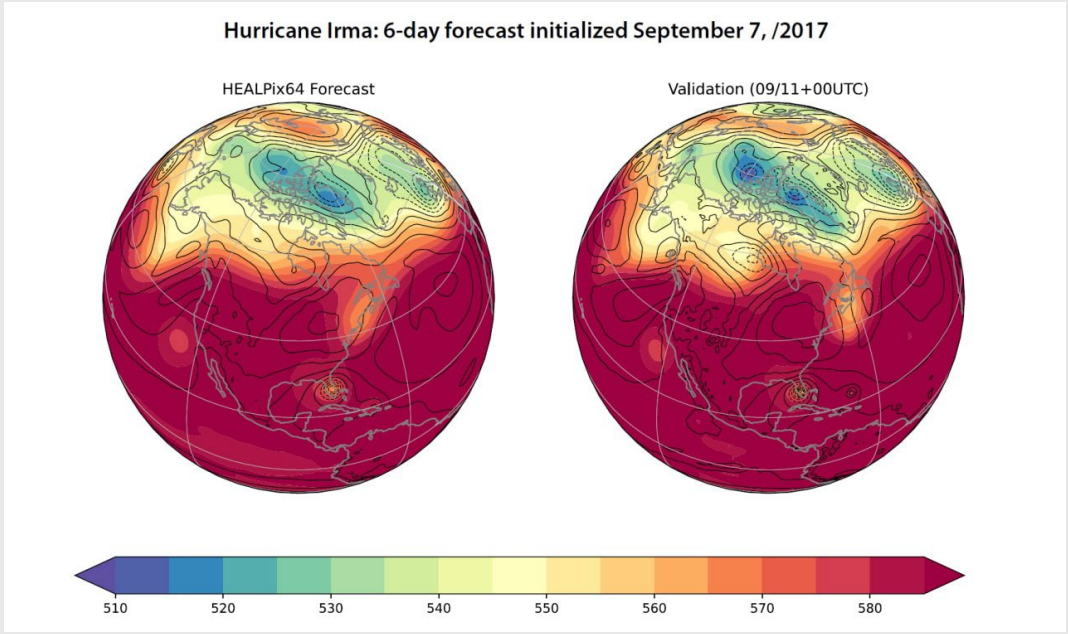
Applications



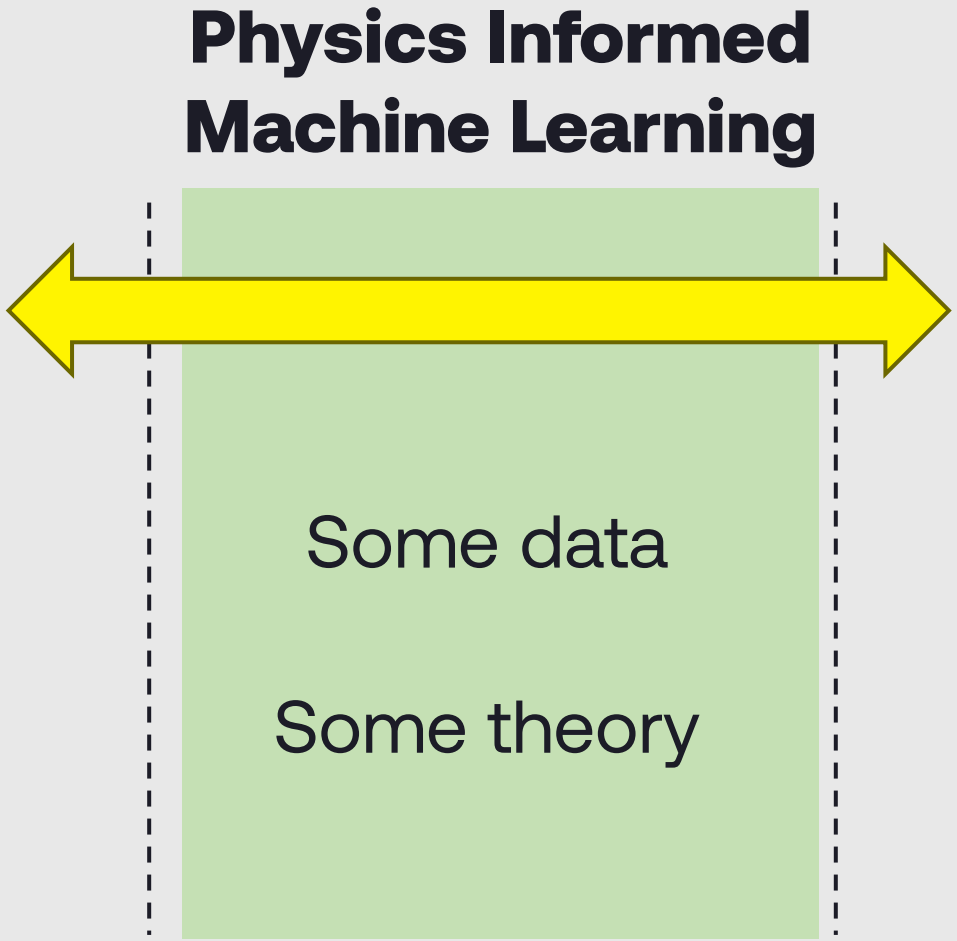






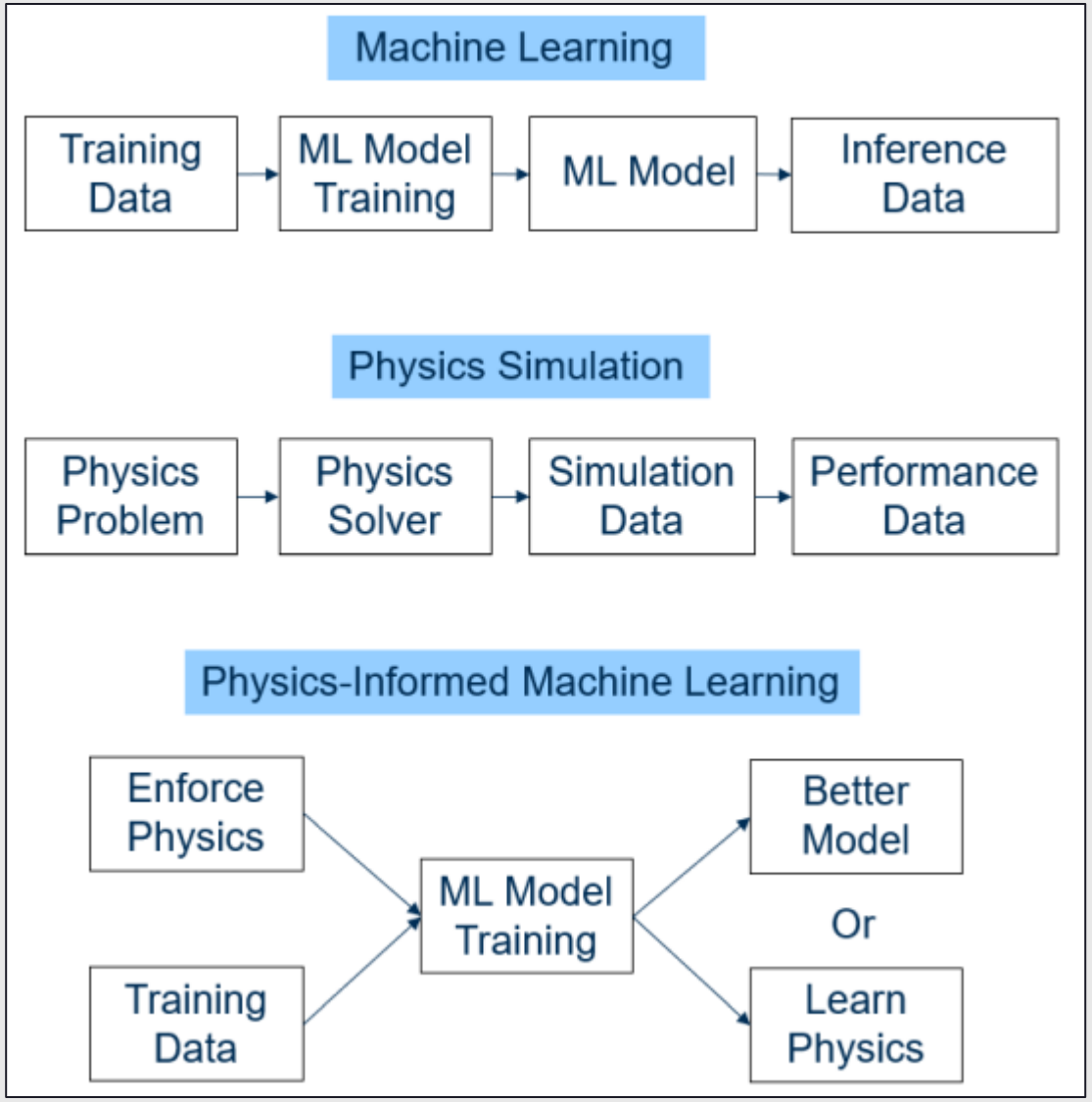


PIML

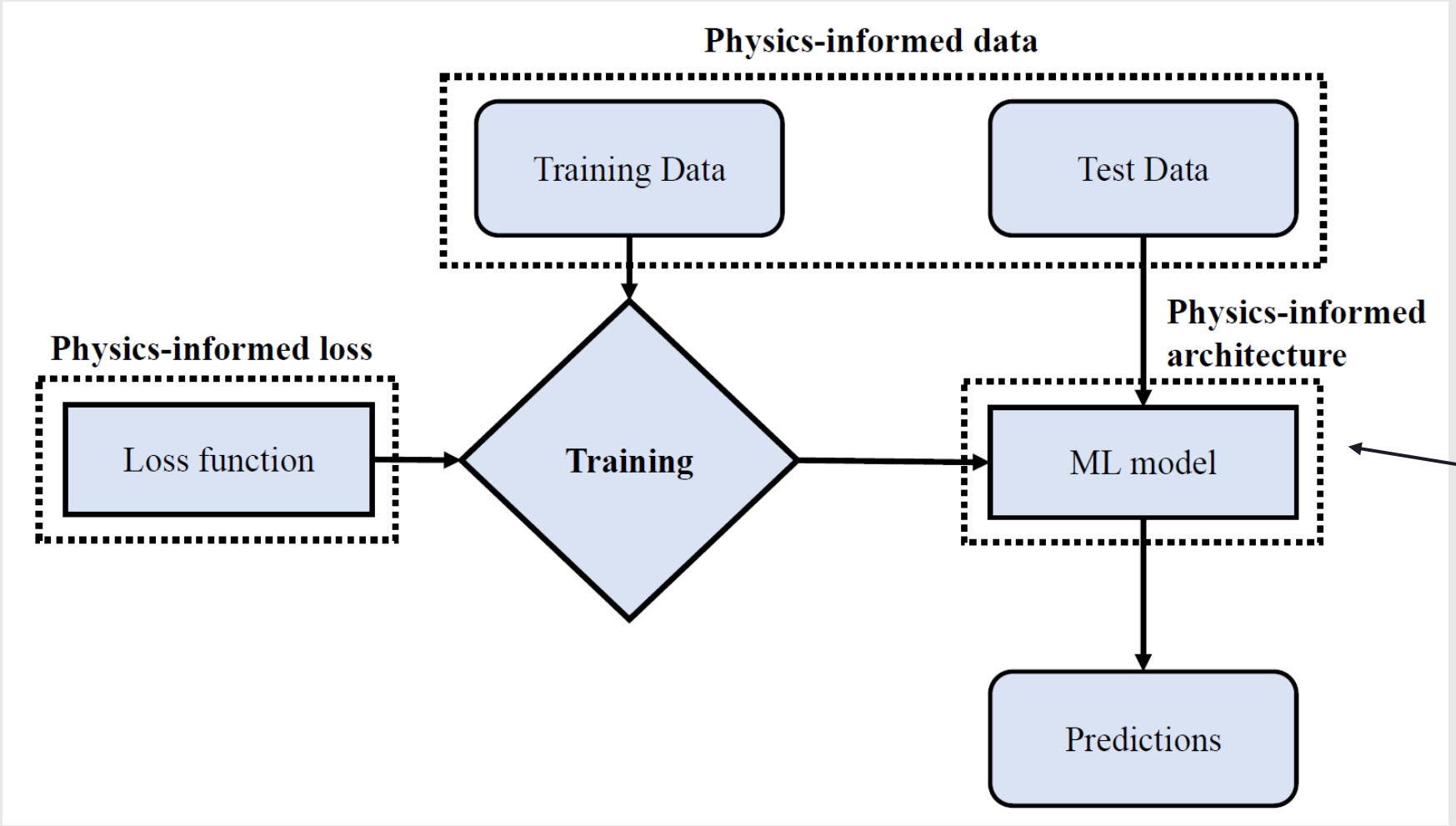

Physics Informed Machine Learning
Scientific computing
Neural Networks
No data
Theoretical model
Some data
Some theory
Lots of data
No theory

PIML

- **Definition**
  - A neural network that encodes physical laws (e.g., through differential equations) into the loss function.
- **High-Level Mechanics**
  - Standard neural network training with an extra penalty for deviation from known physical laws
  - Forward and inverse problem settings.
- **Why PIMLs over just ML?**
  - faster simulations
  - better generalization
  - reduced data requirements
  - interpretability improvements
  - automatic differentiation for inverse problems

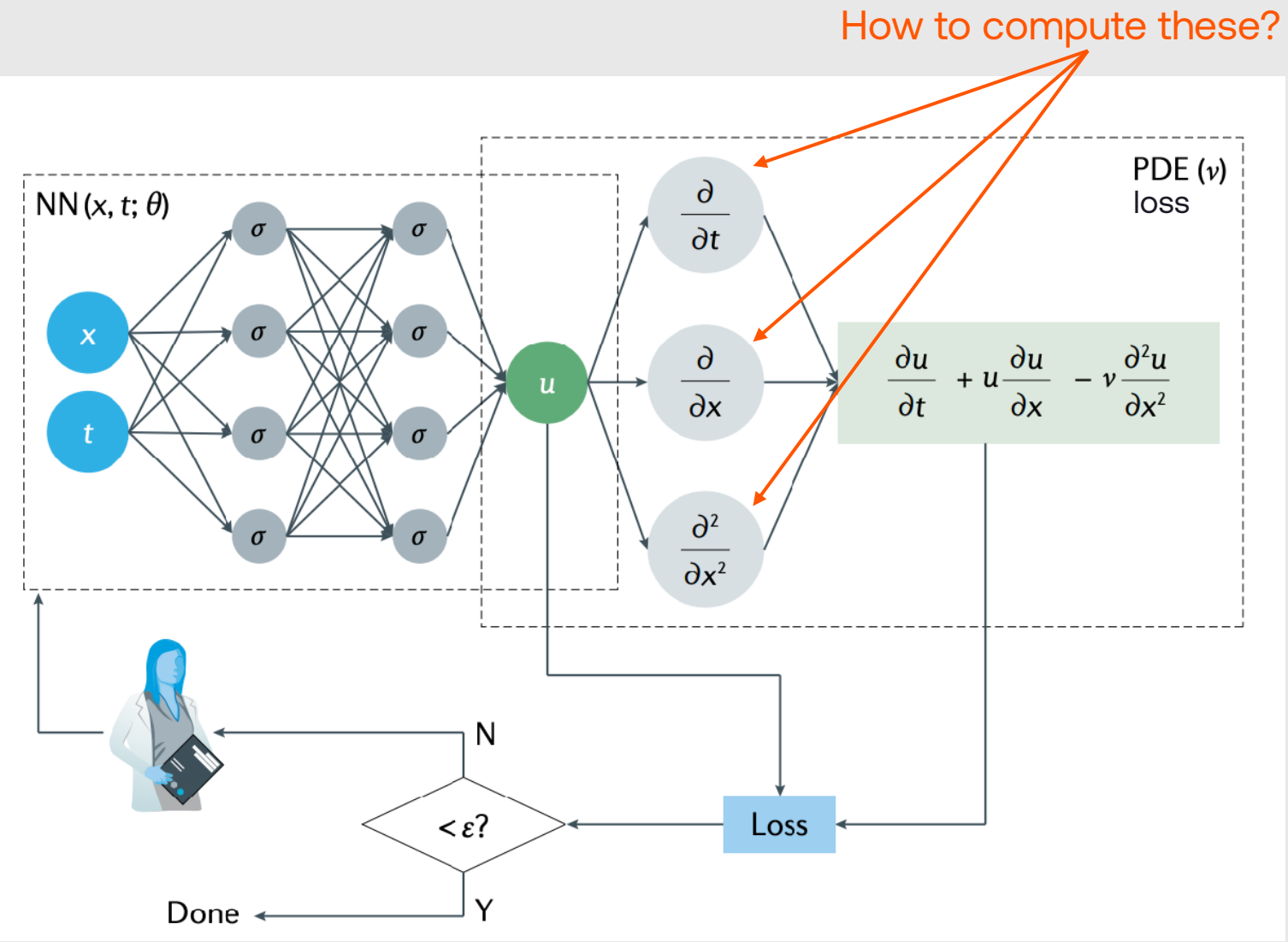


Overview



Typically, an ANN, but CNNs or GNNs are also common

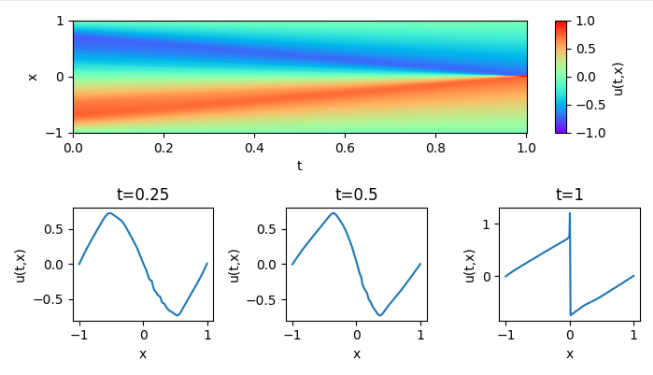
PINN Basics



Burgers' equation in 1D

- fluid dynamics, shock waves
- Viscous form
- Non-linear
- ODE

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$



Mathematical formulation

$$\begin{cases} \text{Domain PDEs residual: } \boldsymbol{r}_d(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) = \boldsymbol{L}(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) - \boldsymbol{f}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \Omega \times (0,T] \\ \text{Boundary condition residual: } \boldsymbol{r}_b(\boldsymbol{x},t) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t) - \boldsymbol{h}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \partial\Omega \times (0,T] \\ \text{Initial condition residual: } \boldsymbol{r}_{ini}(\boldsymbol{x}) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t=0) - \boldsymbol{g}(\boldsymbol{x}) & \forall(\boldsymbol{x},t) \in \Omega \times \{t=0\} \end{cases}$$

Network $\tilde{u}(x,t,\mathbf{W})$

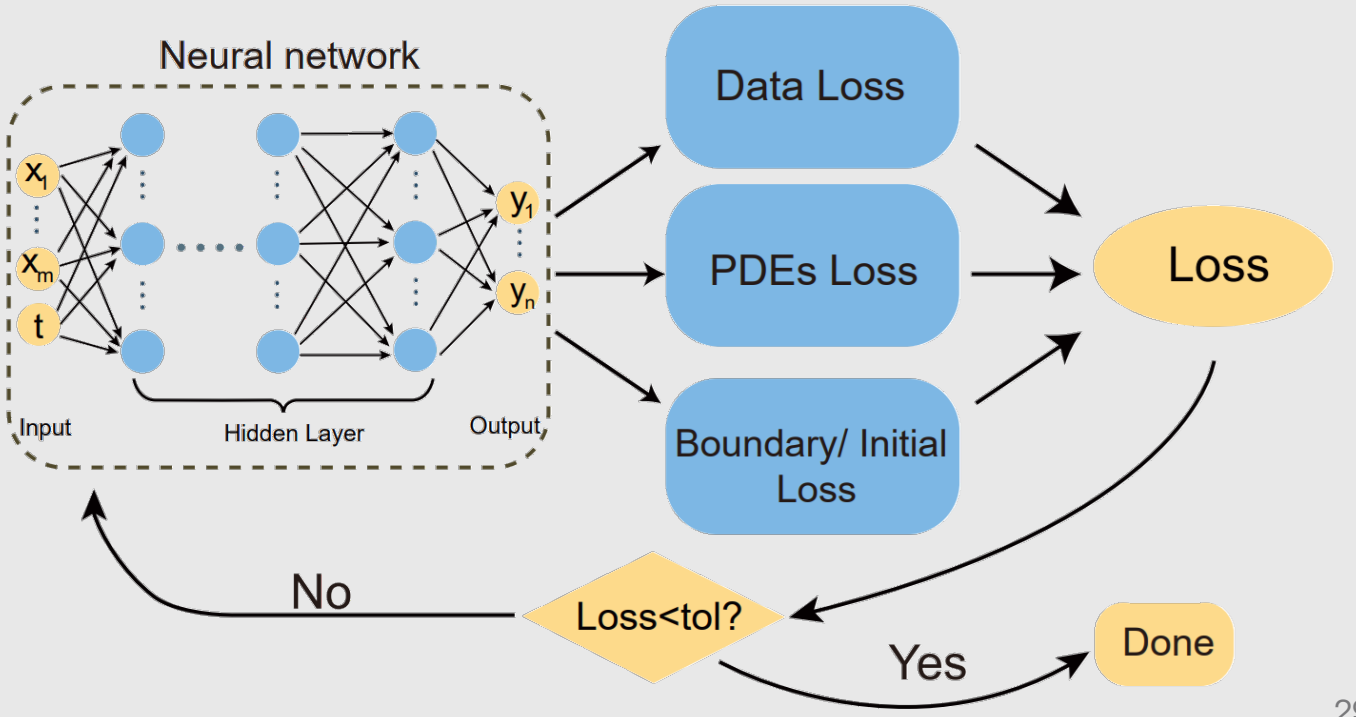
Loss functions

$$\mathcal{L} = \lambda_{pdes}\mathcal{L}_{pdes} + \lambda_b\mathcal{L}_b + \lambda_{data}\mathcal{L}_{data}$$

$$\mathcal{L}_{pdes} = \sum_{j=1}^{S^d} \boldsymbol{\beta}_j^d \cdot (\boldsymbol{r}_j^d)^2$$

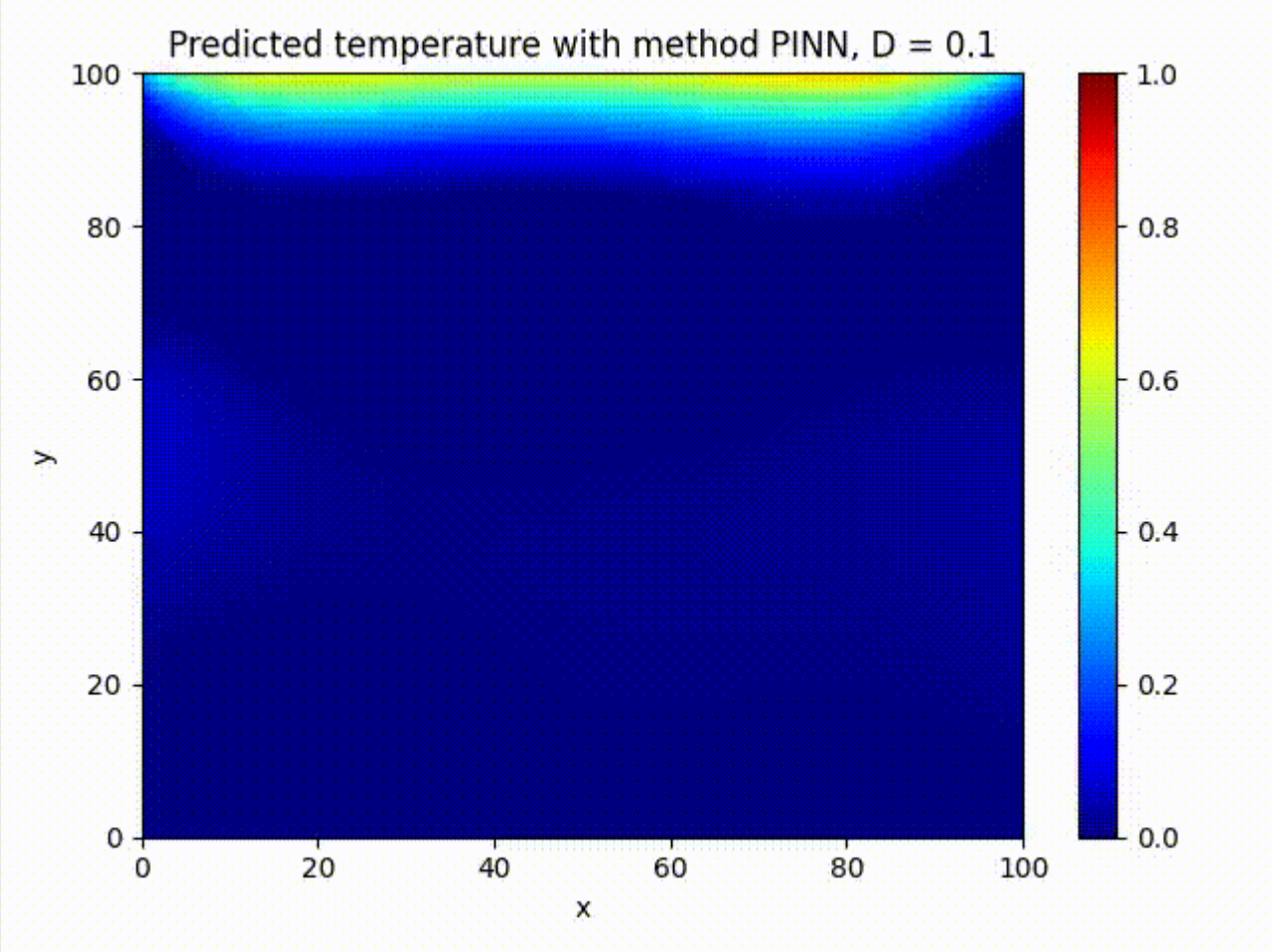
$$\mathcal{L}_b = [\sum_{j=1}^{S^b} \boldsymbol{\beta}_j^b \cdot (\boldsymbol{r}_j^b)^2 + \sum_{j=1}^{S^{ini}} \boldsymbol{\beta}_j^{ini} \cdot (\boldsymbol{r}_j^i)^2]$$

$$\mathcal{L}_{data} = \sum_{I=1}^{N_{data}} [\tilde{\boldsymbol{u}}(\boldsymbol{x}_I, t_I) - \bar{\boldsymbol{u}}(\boldsymbol{x}_I, t_I)]^2,$$

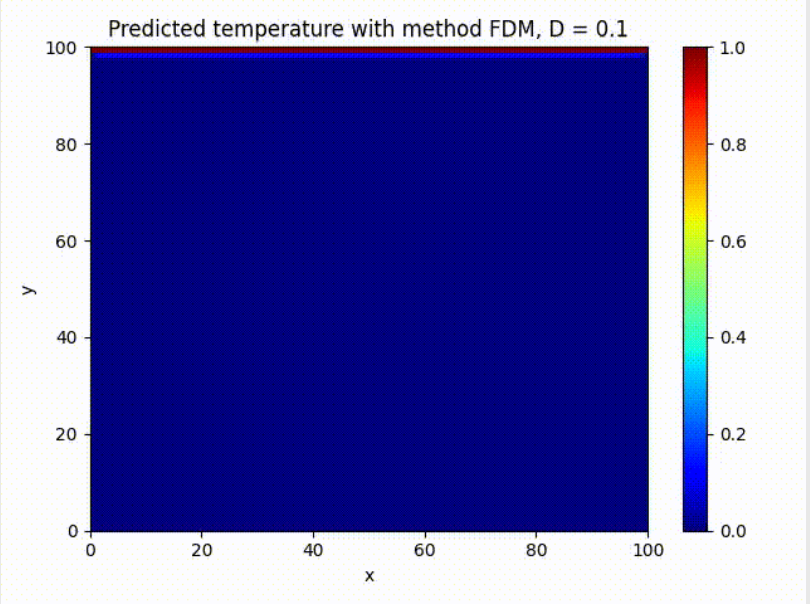


Preview before an interval

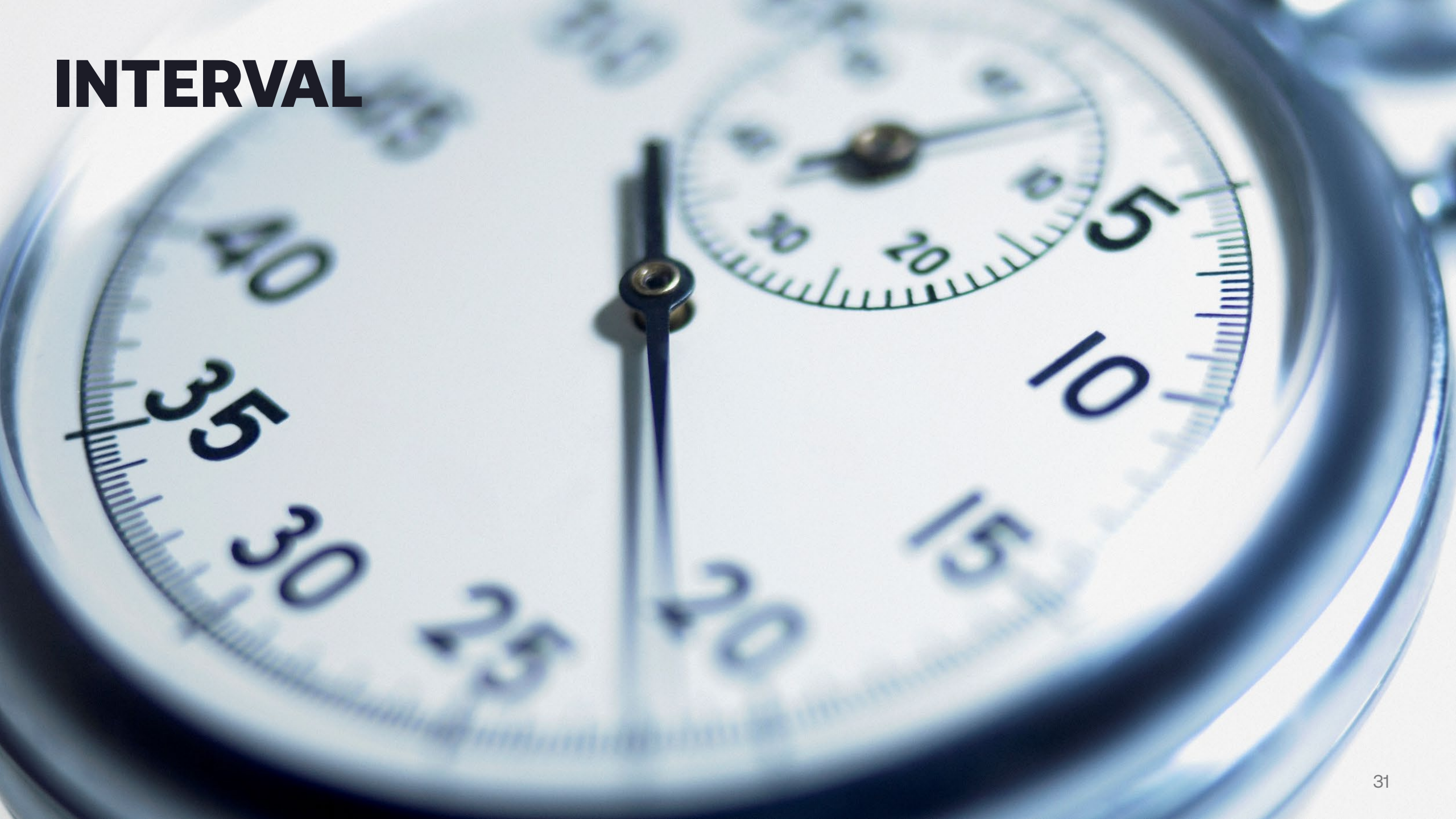
Solution with PINNs



Solution with Finite Differences



INTERVAL



Mathematical formulation

$$\begin{cases} \text{Domain PDEs residual: } \boldsymbol{r}_d(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) = \boldsymbol{L}(\tilde{\boldsymbol{u}}(\boldsymbol{x},t)) - \boldsymbol{f}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \Omega \times (0,T] \\ \text{Boundary condition residual: } \boldsymbol{r}_b(\boldsymbol{x},t) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t) - \boldsymbol{h}(\boldsymbol{x},t) & \forall(\boldsymbol{x},t) \in \partial\Omega \times (0,T] \\ \text{Initial condition residual: } \boldsymbol{r}_{ini}(\boldsymbol{x}) = \tilde{\boldsymbol{u}}(\boldsymbol{x},t=0) - \boldsymbol{g}(\boldsymbol{x}) & \forall(\boldsymbol{x},t) \in \Omega \times \{t=0\} \end{cases}$$

Network $\tilde{u}(x,t,\mathbf{W})$

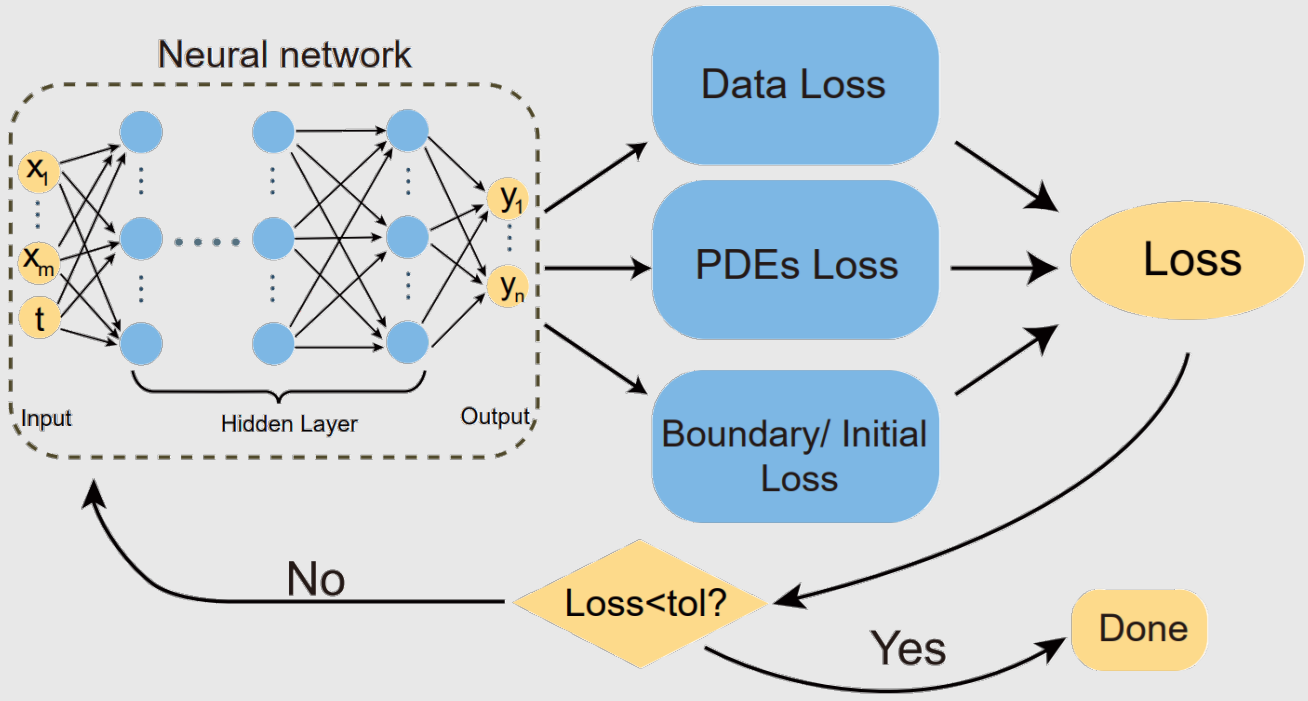
Loss functions

$$\mathcal{L} = \lambda_{pdes}\mathcal{L}_{pdes} + \lambda_b\mathcal{L}_b + \lambda_{data}\mathcal{L}_{data}$$

$$\mathcal{L}_{pdes} = \sum_{j=1}^{S^d} \boldsymbol{\beta}_j^d \cdot (\boldsymbol{r}_j^d)^2$$

$$\mathcal{L}_b = [\sum_{j=1}^{S^b} \boldsymbol{\beta}_j^b \cdot (\boldsymbol{r}_j^b)^2 + \sum_{j=1}^{S^{ini}} \boldsymbol{\beta}_j^{ini} \cdot (\boldsymbol{r}_j^i)^2]$$

$$\mathcal{L}_{data} = \sum_{I=1}^{N_{data}} [\tilde{\boldsymbol{u}}(\boldsymbol{x}_I,t_I) - \bar{\boldsymbol{u}}(\boldsymbol{x}_I,t_I)]^2,$$



Training strategies

- **Optimization Challenges**
  - Balancing data and physics loss terms.
  - Gradient pathologies and potential vanishing or exploding gradients during backprop for the derivative terms

- **Strategies**
  - Adaptive weighting of loss terms (e.g., dynamic weighting, curriculum learning).
  - Stochastic gradient descent vs. more sophisticated optimizers (Adam, L-BFGS).
  - Techniques to handle multi-scale problems (e.g., domain decomposition).
  - Adequate point collocation

$$\mathcal{L} = \lambda_{pdes}\mathcal{L}_{pdes} + \lambda_b\mathcal{L}_b + \lambda_{data}\mathcal{L}_{data}$$

$$\mathcal{L}_{pdes} = \sum_{j=1}^{S^d} \boldsymbol{\beta}_j^d \cdot (\boldsymbol{r}_j^d)^2$$

$$\mathcal{L}_b = [\sum_{j=1}^{S^b} \boldsymbol{\beta}_j^b \cdot (\boldsymbol{r}_j^b)^2 + \sum_{j=1}^{S^{ini}} \boldsymbol{\beta}_j^{ini} \cdot (\boldsymbol{r}_j^i)^2]$$

$$\mathcal{L}_{data} = \sum_{I=1}^{N_{data}} [\tilde{\boldsymbol{u}}(\boldsymbol{x}_I, t_I) - \bar{\boldsymbol{u}}(\boldsymbol{x}_I, t_I)]^2,$$

Case study - I

Burgers' equation

First let's write a simple
Finite Differences Solver

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

$$y(x,0) = \sin(\pi x)$$ Initial condition

$$y(-1,t) = 0$$
$$y(1,t) = 0$$ Boundary conditions

$$x \in [0,2]$$
$$t \in [0,0.48]$$
$$\nu = 0.01/\pi$$

```python
def solve_burgers_fd():
    # Initialize arrays
    x = np.linspace(x_min, x_max, total_points_x)
    t = np.linspace(t_min, t_max, total_points_t)

    # Initial condition: u(x,0) = sin(πx)
    u = np.sin(np.pi * x)
    u_history = np.zeros((total_points_x, total_points_t))

    # Store initial condition
    u_history[:, 0] = u

    # Time stepping
    for j in range(total_points_t-1):
        un = u.copy()  # Copy current solution to calculate spatial derivatives

        # Apply finite difference scheme for interior points
        for i in range(1, total_points_x-1):
            # Forward time, central space for diffusion, upwind for convection
            u[i] = un[i] - un[i] * (dt/dx) * (un[i] - un[i-1]) + \
                   viscosity * (dt/(dx**2)) * (un[i+1] - 2*un[i] + un[i-1])

        # Apply boundary conditions
        u[0] = 0
        u[-1] = 0

        # Store solution
        u_history[:, j+1] = u

    return x, t, u_history
```

Case study - I

Burgers' equation

PINN Solver

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

$$y(x,0) = \sin(\pi x)$$ Initial condition

$$y(-1,t) = 0$$
$$y(1,t) = 0$$ Boundary conditions

$$x \in [0,2]$$
$$t \in [0,0.48]$$
$$\nu = 0.01/\pi$$

Reference

```python
def compute_pde_residual(self, x_colloc):
    x_colloc.requires_grad = True
    u = self.network(x_colloc)

    # Calculate derivatives
    u_grad = torch.autograd.grad(u.sum(), x_colloc, create_graph=True)[0]
    u_t = u_grad[:, 1:2]
    u_x = u_grad[:, 0:1]

    u_xx = torch.autograd.grad(u_x.sum(), x_colloc, create_graph=True)[0][:, 0:1]

    # PDE residual: du/dt + u*du/dx - v*d²u/dx²
    residual = u_t + u * u_x - viscosity * u_xx
```

```python
def train(self, epochs=20000, learning_rate=0.001):
    optimizer = torch.optim.Adam(self.network.parameters(), lr=learning_rate)
    scheduler = ExponentialLR(optimizer, gamma=0.9999)

    # Training history
    history = {'total_loss': [], 'ic_loss': [], 'bc_loss': [], 'pde_loss': []}

    for epoch in range(epochs):
        optimizer.zero_grad()

        # Compute losses
        ic_loss = self.loss_ic(self.X_ic, self.u_ic)
        bc_loss = self.loss_bc(self.X_bc_left) + self.loss_bc(self.X_bc_right)
        pde_loss = self.loss_pde(self.X_colloc)

        # Total loss
        total_loss = ic_loss + bc_loss + pde_loss

        # Backpropagation
        total_loss.backward()
        optimizer.step()
        scheduler.step()
```

Case study - II

Heat equation

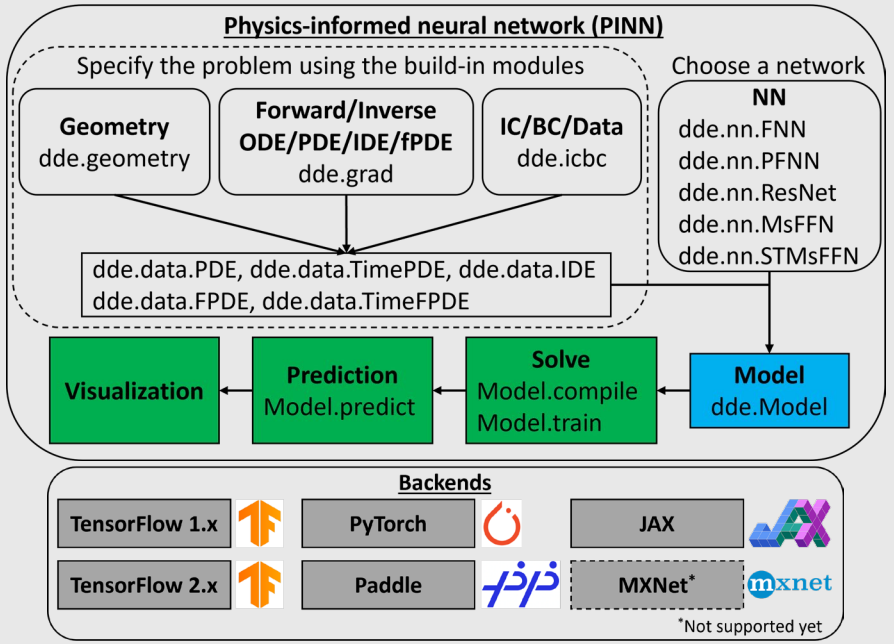
$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

$$u(t, 0, y) = u(t, x_{max}, y) = u(t, x, 0) = 0,$$
$$0 \leq y < y_{max}$$
$$u(t, x, y_{max}) = 1$$

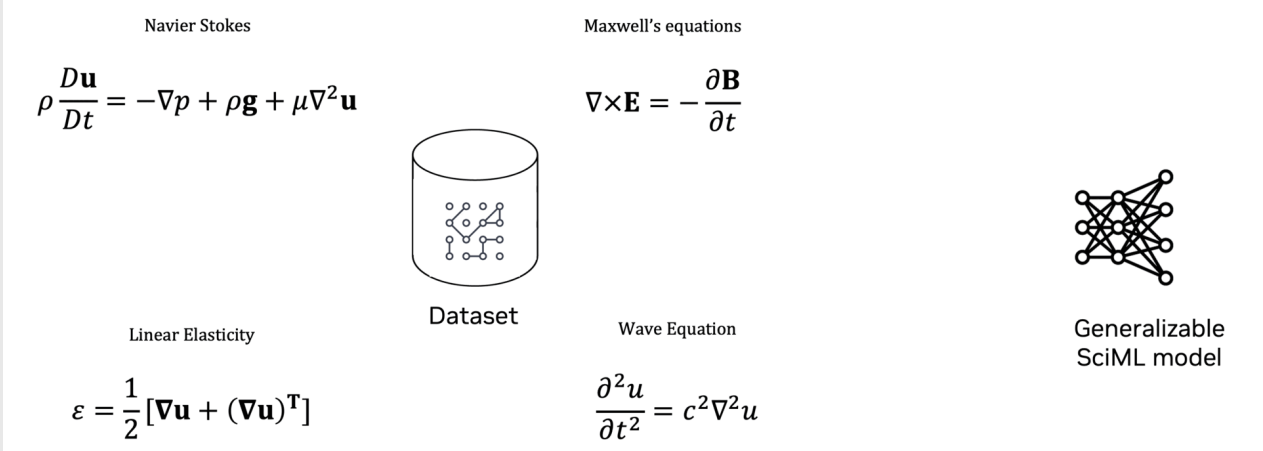
```python
def compute_pde_residual(self, x_colloc):
    x_colloc.requires_grad = True
    u = self.network(x_colloc)

    # First derivatives
    u_grad = torch.autograd.grad(u.sum(), x_colloc, create_graph=True)[0]
    u_t = u_grad[:, 2:3]
    u_x = u_grad[:, 0:1]
    u_y = u_grad[:, 1:2]

    # Second derivatives
    u_xx = torch.autograd.grad(u_x.sum(), x_colloc, create_graph=True)[0][:, 0:1]
    u_yy = torch.autograd.grad(u_y.sum(), x_colloc, create_graph=True)[0][:, 1:2]

    # PDE residual: ∂u/∂t = α(∂²u/∂x² + ∂²u/∂y²)
    residual = u_t - alpha * (u_xx + u_yy)
```

Software tools

- Basic libraries
  - Pytorch
  - Tensorflow
  - NumPy
  - JAX



- Specialized libraries (some)
  - DeepXDE
    - Great for prototyping
  - NVIDIA PhysicsNeMo (former Modulus)
    - Multiple architecture and SymPy support
  - PhiFlow
    - Extensive toolkit from TUM



Limitations and challenges - Overview

- **Computational Cost**: PINN training can be **expensive**, especially for complex PDEs or complex 3D domains.
  - Integrating PINN solvers with existing High-Performance Computing (HPC) codes remains a challenge.
- **Convergence Issues**: Handling stiff PDEs, complex boundary conditions, or chaotic systems.
- **Interpretability**: Ensuring that the network adheres to physical laws in high-dimensional scenarios.
- **Generalization vs. Overfitting**: Ensuring robust solutions that extrapolate beyond training data.

References:
Wang et al *When and why PINNs fail to train: A neural tangent kernel perspective*
Cuomo et al *Scientific Machine Learning through Physics-Informed Neural Networks: Where we are and What's next*
Li et al. *Survey of AI-Driven approaches for Solving Nonlinear Partial Differential Equations*
Sophiya et al. *A comprehensive analysis of PINNs: Variants, Applications, and Challenges*

Limitations and challenges - Technical

Training and Optimization Difficulties

- ❑ PINNs often face **highly non-convex loss landscapes**, which makes optimization challenging and can lead to unstable or slow training.
- ❑ Common issues include **spectral bias** (difficulty learning high-frequency solutions), **unbalanced loss terms**, and **causality violations** (where the network may prioritize fitting future data over accurately learning earlier dynamics).
- ❑ There are **numerous hyperparameters** that require manual tuning, and their optimal values are often unknown and difficult to determine, as there's no clear validation dataset for PDE-driven problems.

Accuracy and Generalization Limitations

- ❑ For forward problems, current PINNs **still lag behind traditional numerical methods** in terms of accuracy and computational efficiency.
- ❑ Errors can arise from the **approximation capabilities** of the neural network (discrepancy between the network's solution space and the true solution), and **integral errors** due to the chosen collocation points.
- ❑ The powerful fitting capacity of neural networks can sometimes lead to **non-unique numerical solutions**.
- ❑ PINNs are most effective when clear physical laws are available. They may struggle when **physics models are approximate, unclear, or don't perfectly match the data**, potentially leading to less accurate models than desired.

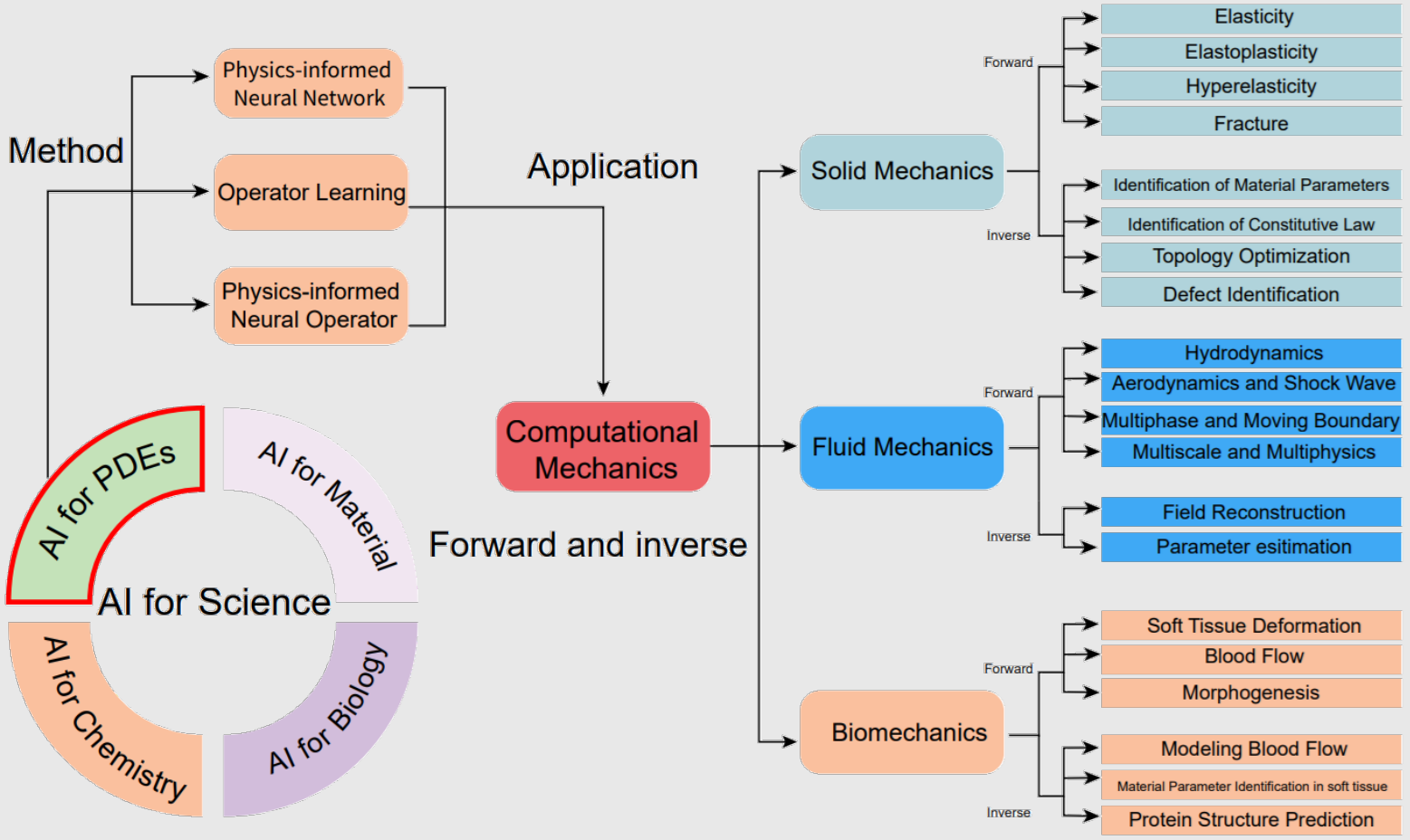
Data Requirements and Interpretability Challenges

- ❑ While PINNs reduce the reliance on large datasets by incorporating physical laws, they can still face significant challenges with **sparse or noisy experimental data**.

Conclusions

Future directions

Active fields of study and application

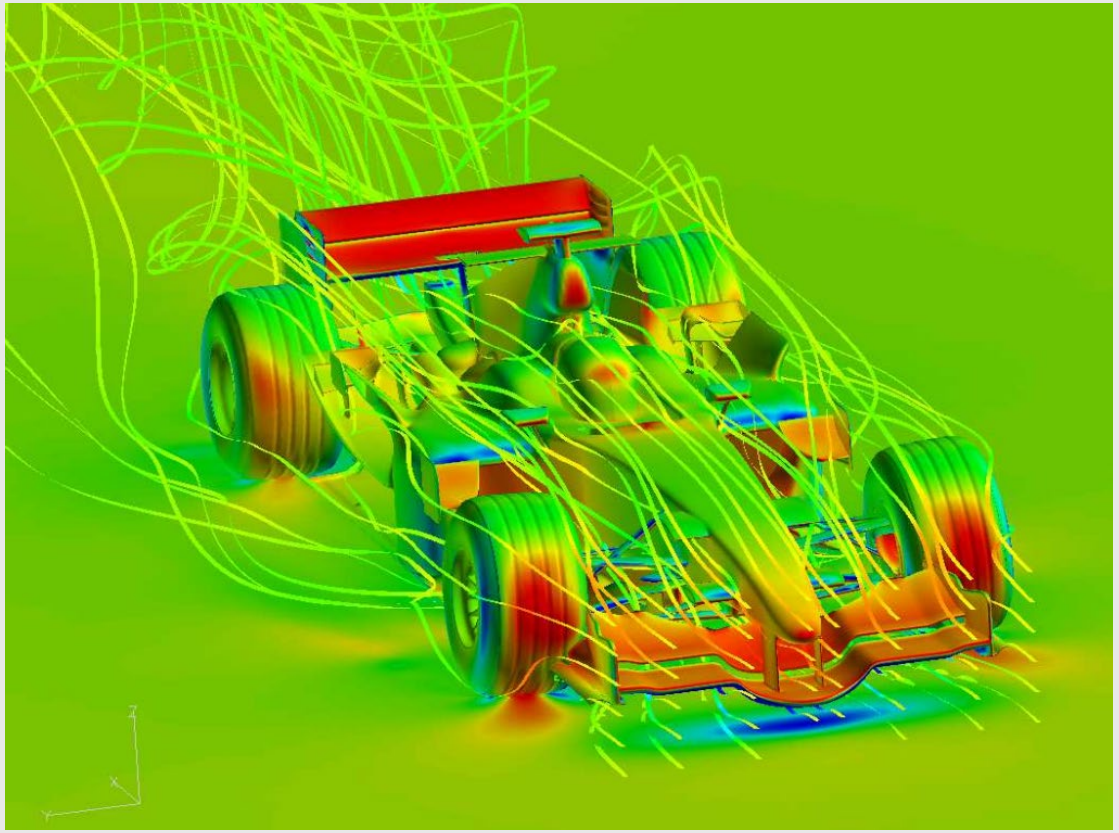


Future directions

- **Hybrid Models**
  - Coupling data-driven ML with classical solvers (e.g., differentiable physics engines).
- **Operator Learning**
  - Extending neural operators for more generalized PDE families
- **Physics-informed Reinforcement Learning**
  - Control systems and policy learning in physical environments
- **High-Performance Computing**
  - Leveraging GPUs, TPUs, HPC clusters for training large-scale PINNs
- **Robustness, Uncertainty, and Real-World Adoption**
  - Critical for practical deployment is the **effective incorporation** of Uncertainty Quantification (UQ)
- **“Foundation models" in computational mechanics**
  - expanding PINN applicability for **digital twins** and real-time industrial applications.

Takeaways

- PIML promise to merge **data** with **physical laws** for improved accuracy and generalization.
- When mature can have **wide applicability** in engineering, physics, and in general science.
- Current research focuses on scalability, reliability, uncertainty quantification, and more complex PDEs.



Physics Based Deep Learning book


BRAID