

# Physics-Informed Neural Network (PINN): Algorithms, Applications, and Software

Lu Lu

Department of Chemical and Biomolecular Engineering  
Penn Institute for Computational Science  
University of Pennsylvania

Tianyuan Mathematical Center in Southeast China  
Dec 10, 2021



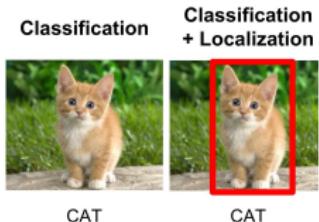
# Numerical discretization of partial differential equations

- Cannot seamlessly incorporate noisy data
- Mesh generation remains complex
- High-dimensional problems cannot be tackled
- Solving inverse problems is often prohibitively expensive

⇒ New opportunity: Machine learning



- Computer vision



- Machine translation



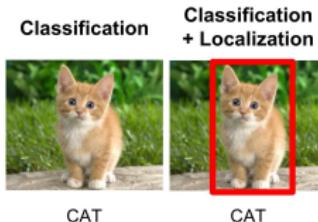
- Black-box & Trial-and-error
- Big data



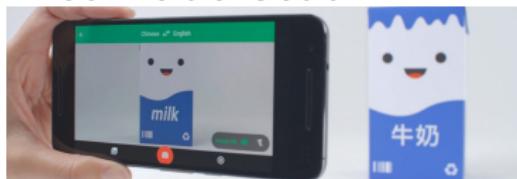
## Scientific Machine Learning: Learning from Small Data

- *Dinky, Dirty, Dynamic, Deceptive Data*

- Computer vision



- Machine translation



- Black-box & Trial-and-error
- Big data

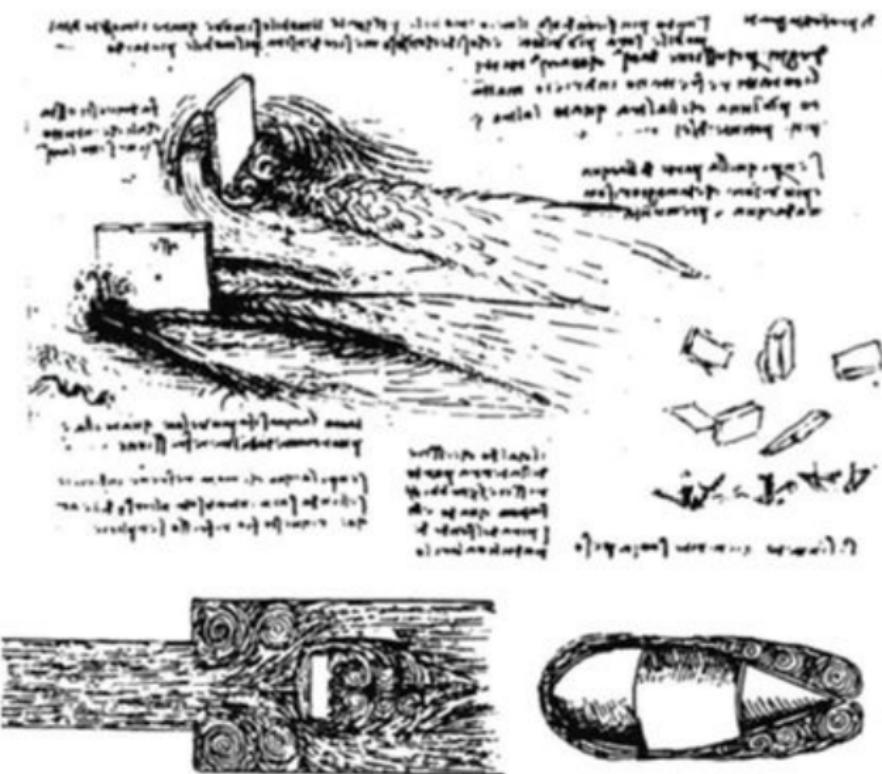


## Scientific Machine Learning: Learning from Small Data

- *Dinky, Dirty, Dynamic, Deceptive Data*
- Scientific domain knowledge  
e.g., physical principles, constraints, symmetries, computational simulations
- Accurate, Robust, Reliable, Interpretable, Explainable

# Leonardo da Vinci's drawing of turbulence

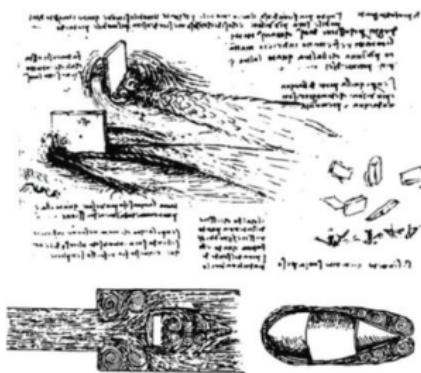
"Observe the motion of the surface of the water, Which resembles that of hair, which has two motions ..."



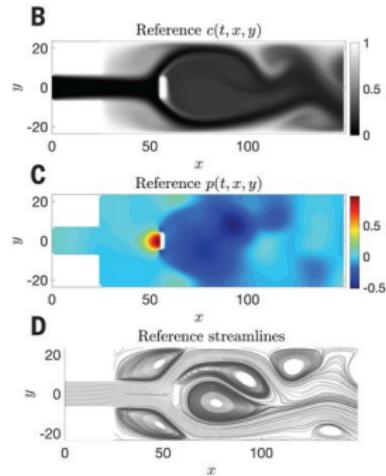
# Machine learning with physics

Infer pressure  $p$  and velocity fields  $(u, v)$  from concentration  $c$

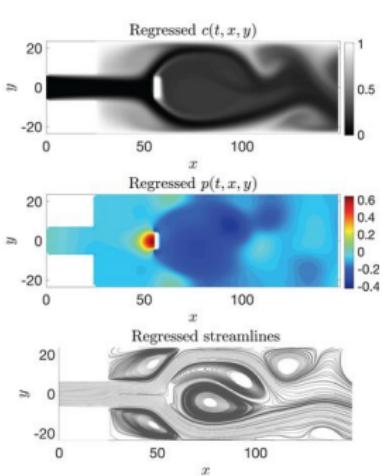
A



B



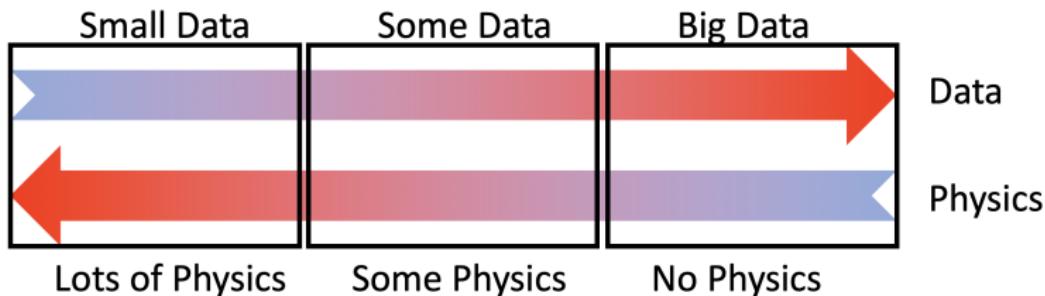
B



# Data & Physics

Karniadakis\*, Kevrekidis\*, **Lu\***, et al., *Nature Rev Phys*, 2021

Three scenarios:

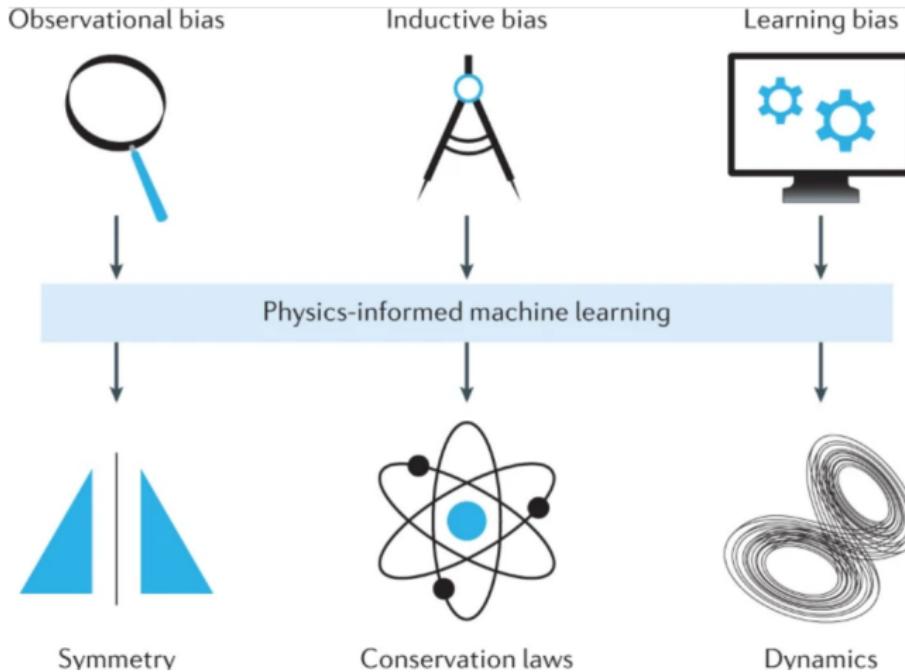


- ① Lots of physics—Forward problems
  - ▶ Finite difference/elements
- ② Some physics—Inverse problems
  - ▶ Multi-fidelity learning
  - ▶ Physics-informed neural network (PINN)
  - ▶ DeepM&Mnet
- ③ No physics—System identification/discovery
  - ▶ Operator learning (DeepONet)



# How to embed physics in ML?

Principles of physics-informed learning:



# Deep learning for partial differential equations (PDEs)

PDE-dependent approaches:

- image-like domain
  - ▶ e.g., (Long et al., *ICML*, 2018), (Zhu et al., *J Comput Phys*, 2019)
- parabolic PDEs, e.g., through Feynman-Kac formula
  - ▶ e.g., (Beck et al., *J Nonlinear Sci*, 2017), (Han et al., *PNAS*, 2018)
- variational form
  - ▶ e.g., (E & Yu, *Commun Math Stat*, 2018)

General approaches:

- Galerkin type projection
  - ▶ e.g., (Meade & Fernandez, *Math Comput Model*, 1994), (Kharazmi et al., *arXiv*, 2019)
- **strong form (Physics-informed neural networks)**
  - ▶ e.g., (Dissanayake & Phan-Thien, *Commun Numer Meth En*, 1994), (Lagaris et al., *IEEE Trans Neural Netw*, 1998), (Raissi et al., *J Comput Phys*, 2019), (**Lu** et al., *SIAM Rev*, 2021)



# Physics-informed neural network (PINN)

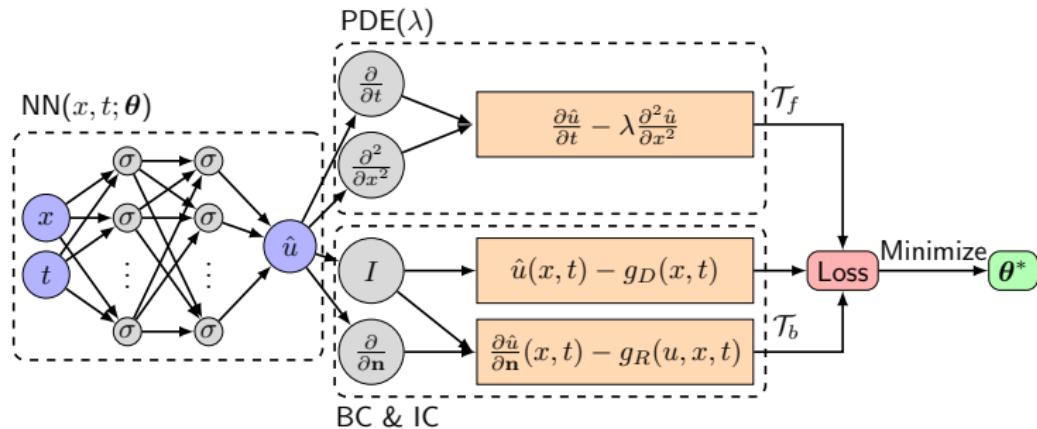
Diffusion equation

$$\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$$

BCs:  $u(x, t) = g_D(x, t)$  on  $\Gamma_D \subset \partial\Omega$

$\frac{\partial u}{\partial n}(x, t) = g_R(u, x, t)$  on  $\Gamma_R \subset \partial\Omega$

IC is treated as a Dirichlet BC on the spatio-temporal domain



# Physics-informed neural network (PINN)

- ① Construct a neural network  $\hat{u}(\mathbf{x}; \boldsymbol{\theta})$  with parameters  $\boldsymbol{\theta}$ .
- ② Specify the two training sets  $\mathcal{T}_f$  and  $\mathcal{T}_b$  for the equation and IC/BCs.
- ③ Specify a loss function of both the PDE equation and IC/BC residuals.

$$\mathcal{L}(\boldsymbol{\theta}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b)$$

where

$$\mathcal{L}_f(\boldsymbol{\theta}; \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f \left( \mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \dots; \boldsymbol{\lambda} \right) \right\|_2^2$$

$$\mathcal{L}_b(\boldsymbol{\theta}; \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2$$

- ④ Train the neural network to find the best parameters  $\boldsymbol{\theta}^*$  by minimizing the loss function  $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$ .

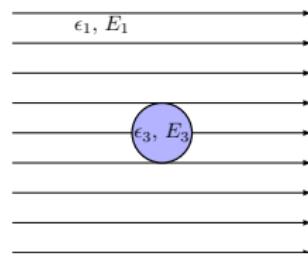
# Inverse problems

Challenge: *small* data + *incomplete* physics laws

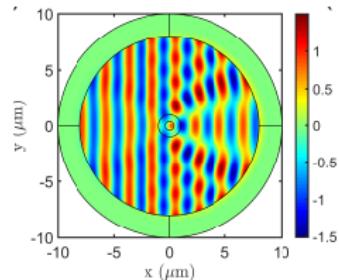
## Invisible cloaking

Permittivity  $\epsilon$ , permeability  $\mu$

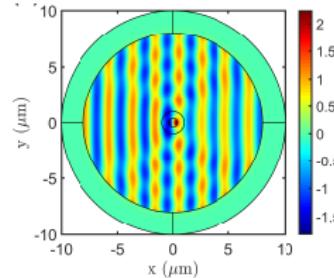
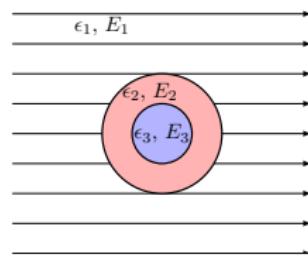
Electric field  $E$  without coating



joint work with Prof. Luca Dal Negro (Boston U)

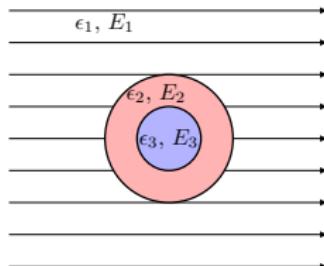


Electric field  $E$  with coating



# Invisible cloaking

**Goal:** Given  $\epsilon_1$  and  $\epsilon_3$ , find  $\epsilon_2(x, y)$  s.t.  $E_1 \approx E_{1,target}$



Helmholtz equation ( $k_0 = \frac{2\pi}{\lambda_0}$ )

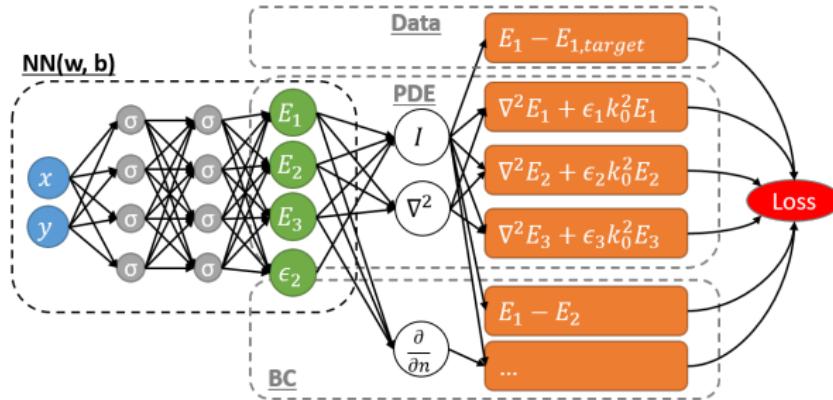
$$\nabla^2 E_i + \epsilon_i k_0^2 E_i = 0, \quad i = 1, 2, 3$$

Boundary conditions:

- Outer circle:  $E_1 = E_2, \frac{1}{\mu_1} \frac{\partial E_1}{\partial \mathbf{n}} = \frac{1}{\mu_2} \frac{\partial E_2}{\partial \mathbf{n}}$
- Inner circle:  $E_2 = E_3, \frac{1}{\mu_2} \frac{\partial E_2}{\partial \mathbf{n}} = \frac{1}{\mu_3} \frac{\partial E_3}{\partial \mathbf{n}}$

# Physics-informed neural networks (PINNs)

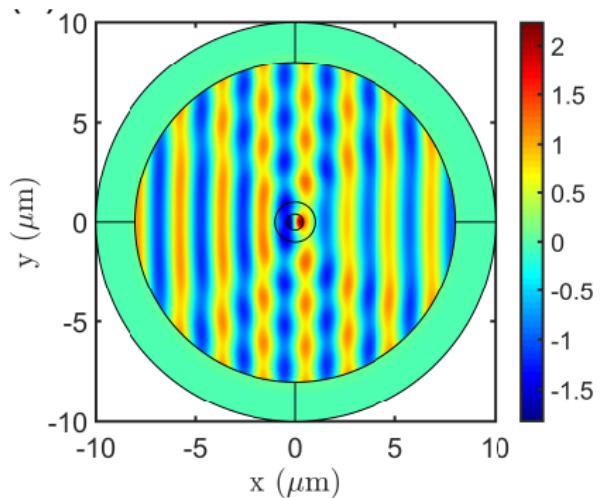
Idea: Embed a PDE into the loss via automatic differentiation (AD)



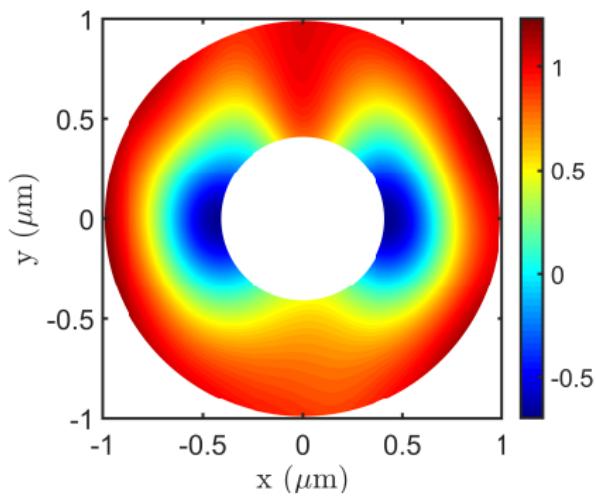
- mesh-free & particle-free
- inverse problems: seamlessly integrate data and physics
- black-box or noisy IC/BC/forcing terms (Pang\*, Lu\*, et al., *SIAM J Sci Comput*, 2019)
- a unified framework: PDE, integro-differential equations (Lu et al., *SIAM Rev*, 2021), fractional PDE (Pang\*, Lu\*, et al., *SIAM J Sci Comput*, 2019), stochastic PDE (Zhang, Lu, et al., *J Comput Phys*, 2019)

# Invisible cloaking

Electric field  $E_i$



Permittivity  $\epsilon_2$



# Physics-informed neural networks (PINNs)

$$f \left( \mathbf{x}; \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda} \right) = 0, \quad \mathbf{x} \in \Omega$$

- Initial/boundary conditions  $\mathcal{B}(u, \mathbf{x}) = 0$  on  $\partial\Omega$
- Extra information  $\mathcal{I}(u, \mathbf{x}) = 0$  for  $\mathbf{x} \in \mathcal{T}_i$

$$\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_b) + w_i \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i)$$

where

$$\mathcal{L}_f = \frac{1}{|\mathcal{T}_f|} \sum_{\mathbf{x} \in \mathcal{T}_f} \left\| f \left( \mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots; \boldsymbol{\lambda} \right) \right\|_2^2$$

$$\mathcal{L}_b = \frac{1}{|\mathcal{T}_b|} \sum_{\mathbf{x} \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2$$

$$\mathcal{L}_i = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \|\mathcal{I}(\hat{u}, \mathbf{x})\|_2^2$$



# Systems biology described by ODEs

Ultradian model for the glucose-insulin interaction (Sturis et al., 1991)

**Goal:** Infer 21 unknown parameters

$$\frac{dI_p}{dt} = f_1(G) - E\left(\frac{I_p}{V_p} - \frac{I_i}{V_i}\right) - \frac{I_p}{t_p}$$

$$\frac{dI_i}{dt} = E\left(\frac{I_p}{V_p} - \frac{I_i}{V_i}\right) - \frac{I_i}{t_i}$$

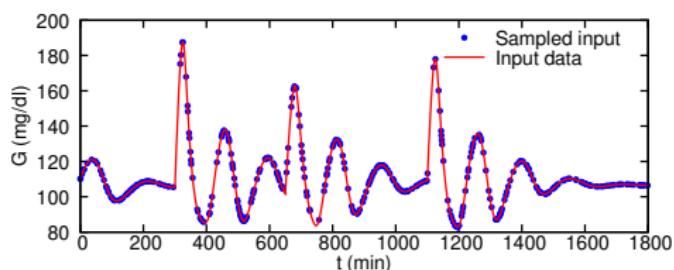
$$\frac{dG}{dt} = f_4(h_3) + I_G(t) - f_2(G) - f_3(I_i)G$$

$$\frac{dh_1}{dt} = \frac{1}{t_d}(I_p - h_1)$$

$$\frac{dh_2}{dt} = \frac{1}{t_d}(h_1 - h_2)$$

$$\frac{dh_3}{dt} = \frac{1}{t_d}(h_2 - h_3)$$

- $I_p$ : plasma insulin concentration
- $I_i$ : interstitial insulin concentration
- $G$ : glucose concentration (Data)
- $I_G$ : glucose from nutritional intake



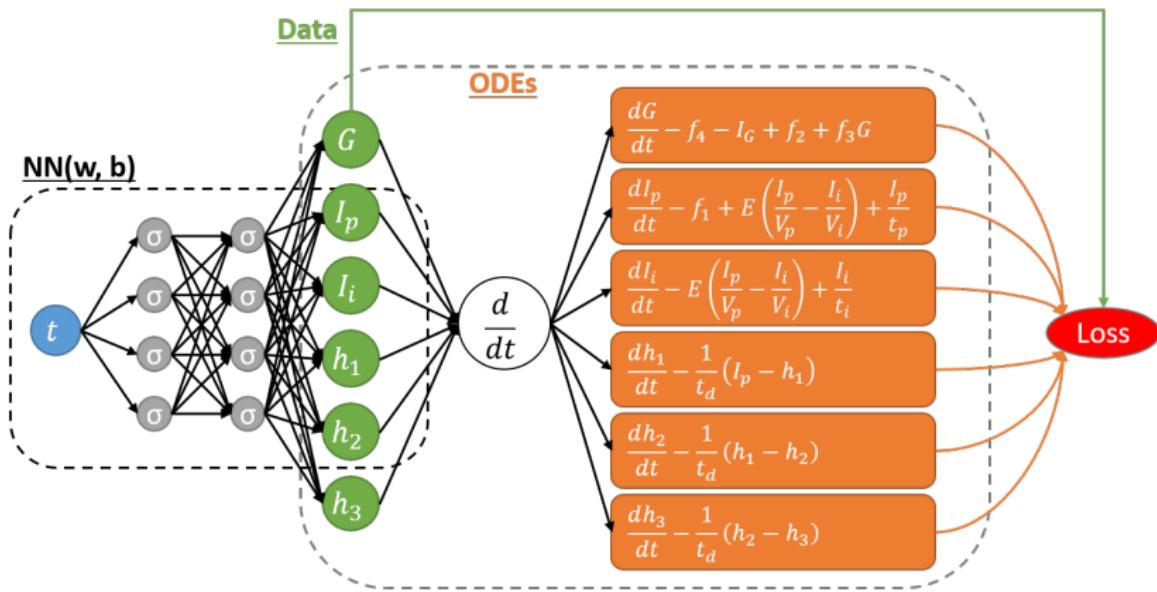
$$f_1(G) = \frac{R_m}{1 + \exp(\frac{-G}{V_g C_1} + a_1)}, f_2(G) = U_b \left(1 - \exp(\frac{-G}{C_2 V_g})\right), f_3(I_i) = \frac{1}{C_3 V_g} \left(U_0 + \frac{U_m}{1 + (\kappa I_i)^{-\beta}}\right)$$

$$f_4(h_3) = \frac{R_g}{1 + \exp(\alpha(\frac{h_3}{C_5 V_p} - 1))}, \kappa = \frac{1}{C_4} \left(\frac{1}{V_i} + \frac{1}{E t_i}\right), I_G(t) = \sum_{j=1}^N m_j k \exp(k(t_j - t))$$



# Physics-informed neural networks (PINNs)

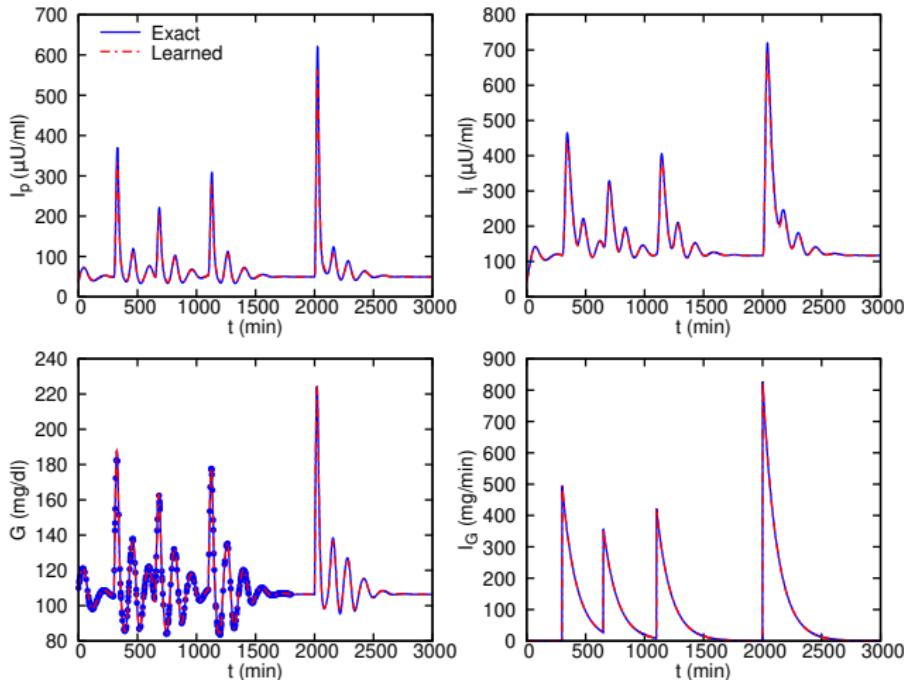
**Idea:** Embed ODEs with **unknown parameters** into the loss via automatic differentiation (backpropagation)



- Seamlessly integrate *sparse* data and *incomplete* physics



# Inferred dynamics and forecasting



Research Highlight in  
*Nature Computational Science*

SYSTEMS BIOLOGY  
**Parameter inference for systems biology models**

Ananya Rastogi

*Nature Computational Science* 1, 16(2021) | Cite this article

# Error analysis

Theorem (Universal approximation theorem; Cybenko, 1989)

Let  $\sigma$  be any continuous sigmoidal function. Then finite sums of the form  $G(x) = \sum_{j=1}^N \alpha_j \sigma(w_j \cdot x + b_j)$  are dense in  $C(I_d)$ .

Theorem (Pinkus, 1999)

Let  $\mathbf{m}^i \in \mathbb{Z}_+^d$ ,  $i = 1, \dots, s$ , and set  $m = \max_{i=1, \dots, s} (m_1^i + \dots + m_d^i)$ .

Assume  $\sigma \in C^m(\mathbb{R})$  and  $\sigma$  is not a polynomial. Then the space of single hidden layer neural nets

$$\mathcal{M}(\sigma) := \text{span}\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in

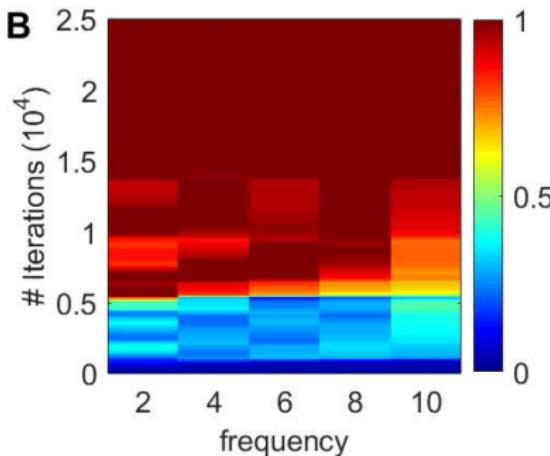
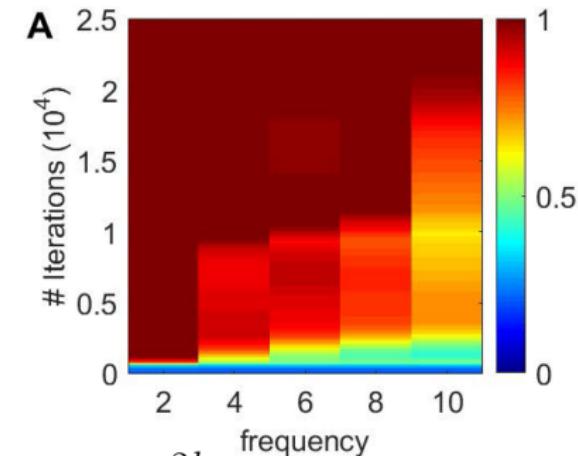
$$C^{\mathbf{m}^1, \dots, \mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^s C^{\mathbf{m}^i}(\mathbb{R}^d).$$

Optimization & Generalization:

Shin et al., 2020; Mishra & Molinaro, 2020; Luo & Yang, 2020

# Optimization

- A: approximate  $f(x) = \sum_{k=1}^5 \sin(2kx)/(2k)$ 
  - ▶ learn from low to high frequencies (Rahaman et al., *ICML*, 2019; Xu et al., arXiv, 2019)
- B: solve the Poisson equation  $-f_{xx} = \sum_{k=1}^5 2k \sin(2kx)$ 
  - ▶ all frequencies are learned almost simultaneously
  - ▶ faster learning



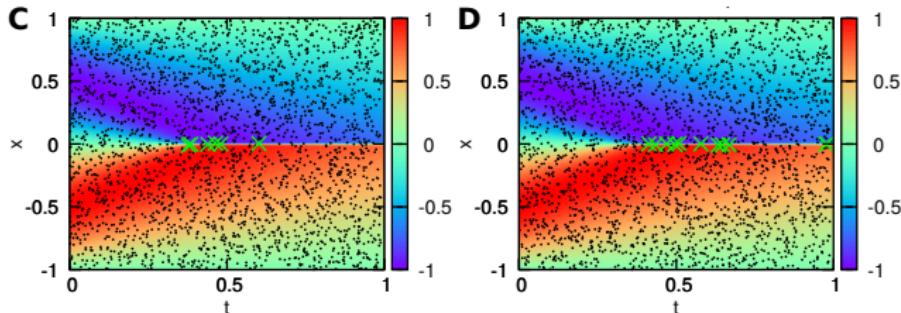
# Generalization: Residual-based adaptive refinement (RAR)

**Challenge:** Uniform residual points are not efficient for PDEs with steep solutions.

e.g., Burgers equation ( $x \in [-1, 1]$ ,  $t \in [0, 1]$ ):

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, \quad u(x, 0) = -\sin(\pi x), \quad u(-1, t) = u(1, t) = 0.$$

- **Idea:** adaptively add more points in locations with large PDE residual  $\left\| f \left( \mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda} \right) \right\|$



10,000 (Raissi et al., *J Comput Phys*, 2019) ↓ 2,540



# gPINN: Gradient-enhanced PINN

**Idea:** The derivatives of the PDE residual  $f$  are also zero.

$$\nabla f(\mathbf{x}) = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_d} \right) = \mathbf{0}, \quad \mathbf{x} \in \Omega$$

Embed the gradient of the PDE residual into the loss

$$\mathcal{L} = w_f \mathcal{L}_f + w_b \mathcal{L}_b + w_i \mathcal{L}_i + \sum_{i=1}^d w_{g_i} \mathcal{L}_{g_i} (\boldsymbol{\theta}; \mathcal{T}_{g_i})$$

$$\mathcal{L}_{g_i} (\boldsymbol{\theta}; \mathcal{T}_{g_i}) = \frac{1}{|\mathcal{T}_{g_i}|} \sum_{\mathbf{x} \in \mathcal{T}_{g_i}} \left| \frac{\partial f}{\partial x_i} \right|^2$$



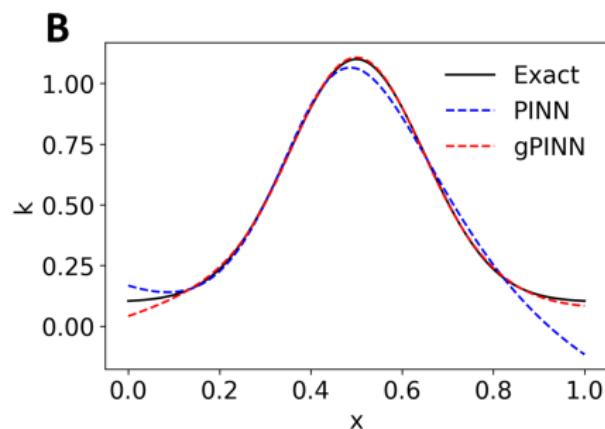
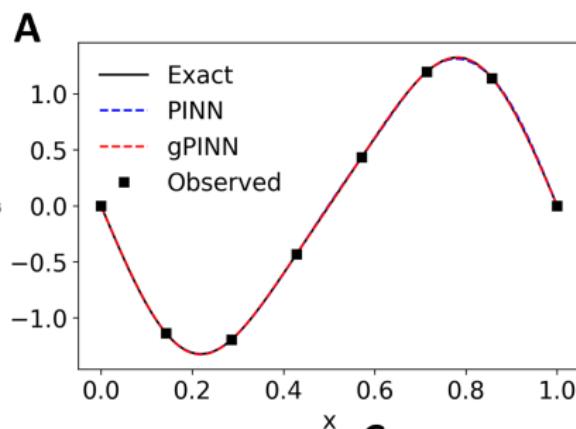
# Inferring the reaction rate in a diffusion-reaction system

A diffusion-reaction system with the diffusion coefficient  $\lambda = 0.01$ :

$$\lambda \frac{\partial^2 u}{\partial x^2} - k(x)u = \sin(2\pi x), \quad x \in [0, 1]$$

with zero boundary condition

- **Goal:** Infer space-dependent reaction rate  $k(x)$
- **Data:** 8 observations of  $u$  & 10 PDE residual points



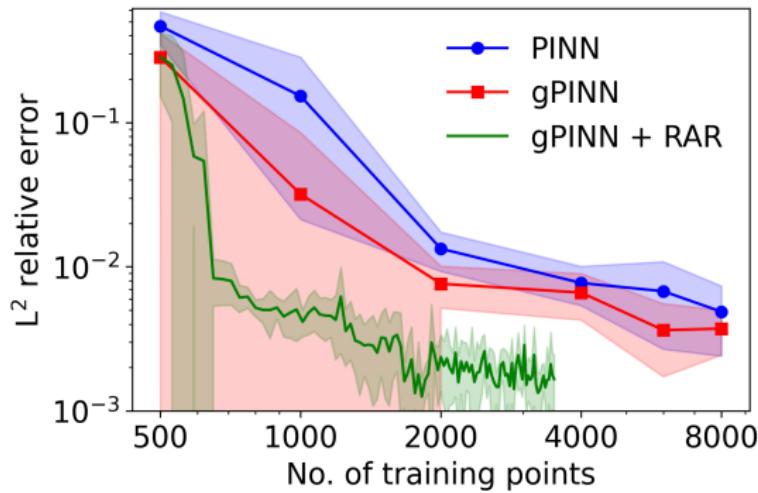
gPINN & RAR

### Allen-Cahn equation with $D = 0.001$

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} + 5(u - u^3), \quad x \in [-1, 1], \quad t \in [0, 1],$$

with the initial and boundary conditions:

$$u(x,0) = x^2 \cos(\pi x), \quad u(-1,t) = u(1,t) = -1$$



# Hard constraints in inverse design/topology optimization

Constrained optimization problem for  $\gamma(\mathbf{x})$

$$\min_{\mathbf{u}, \gamma} \mathcal{J}(\mathbf{u}; \gamma)$$

subject to

$$\begin{cases} \mathcal{F}[\mathbf{u}; \gamma] = \mathbf{0}, \\ \mathcal{B}[\mathbf{u}] = 0, \\ h(\mathbf{u}, \gamma) \leq 0, \end{cases}$$

Convert the constrained optimization to an unconstrained optimization via losses:

$$\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \frac{1}{MN} \sum_{j=1}^M \sum_{i=1}^N |\mathcal{F}_i[\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j)]|^2,$$

$$\mathcal{L}_h(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathbb{1}_{\{h(\hat{\mathbf{u}}, \hat{\gamma}) > 0\}} h^2(\hat{\mathbf{u}}, \hat{\gamma}),$$

# Hard constraints

- Soft constraints: large  $\mu_{\mathcal{F}}, \mu_h \Rightarrow$  ill-conditioned optimization

$$\mathcal{L}(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathcal{J} + \mu_{\mathcal{F}} \mathcal{L}_{\mathcal{F}} + \mu_h \mathcal{L}_h$$

- Penalty method (a sequence of soft constraints)

$$\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = \mathcal{J} + \mu_{\mathcal{F}}^k \mathcal{L}_{\mathcal{F}} + \mu_h^k \mathcal{L}_h$$

$$\mu_{\mathcal{F}}^{k+1} = \beta_{\mathcal{F}} \mu_{\mathcal{F}}^k, \quad \mu_h^{k+1} = \beta_h \mu_h^k$$

- Augmented Lagrangian method: new terms to mimic Lagrange multipliers

$$\begin{aligned}\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = & \mathcal{J} + \mu_{\mathcal{F}}^k \mathcal{L}_{\mathcal{F}} + \mu_h^k \mathbb{1}_{\{h>0 \vee \lambda_h^k > 0\}} h^2 \\ & + \frac{1}{MN} \sum_{j=1}^M \sum_{i=1}^N \lambda_{i,j}^k \mathcal{F}_i [\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j)] + \lambda_h^k h\end{aligned}$$

$$\lambda_{i,j}^k = \lambda_{i,j}^{k-1} + 2\mu_{\mathcal{F}}^{k-1} \mathcal{F}_i \left[ \hat{\mathbf{u}}(\mathbf{x}_j; \boldsymbol{\theta}_u^{k-1}); \hat{\gamma}(\mathbf{x}_j; \boldsymbol{\theta}_\gamma^{k-1}) \right]$$

$$\lambda_h^k = \max \left( \lambda_h^{k-1} + 2\mu_h^{k-1} h \left( \hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u^{k-1}), \hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma^{k-1}) \right), 0 \right)$$



# hPINN: PINN with hard constraints

## Augmented Lagrangian method

$$\begin{aligned}\mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma) = & \mathcal{J} + \mu_{\mathcal{F}}^k \mathcal{L}_{\mathcal{F}} + \mu_h^k \mathbb{1}_{\{h>0 \vee \lambda_h^k>0\}} h^2 \\ & + \frac{1}{MN} \sum_{j=1}^M \sum_{i=1}^N \lambda_{i,j}^k \mathcal{F}_i [\hat{\mathbf{u}}(\mathbf{x}_j); \hat{\gamma}(\mathbf{x}_j)] + \lambda_h^k h\end{aligned}$$

---

**Algorithm 2.2** hPINNs via the augmented Lagrangian method.

---

**Hyperparameters:** initial penalty coefficients  $\mu_{\mathcal{F}}^0$  and  $\mu_h^0$ , factors  $\beta_{\mathcal{F}}$  and  $\beta_h$

$k \leftarrow 0$

$\lambda_{i,j}^0 \leftarrow 0$  for  $1 \leq i \leq N, 1 \leq j \leq M$

$\lambda_h^0 \leftarrow 0$

$\boldsymbol{\theta}_u^0, \boldsymbol{\theta}_\gamma^0 \leftarrow \arg \min_{\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma} \mathcal{L}^0(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma)$ : Train the networks  $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u)$  and  $\hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma)$  from random initialization, until the training loss is converged

**repeat**

$k \leftarrow k + 1$

$\mu_{\mathcal{F}}^k \leftarrow \beta_{\mathcal{F}} \mu_{\mathcal{F}}^{k-1}$

$\mu_h^k \leftarrow \beta_h \mu_h^{k-1}$

$\lambda_{i,j}^k \leftarrow \lambda_{i,j}^{k-1} + 2\mu_{\mathcal{F}}^{k-1} \mathcal{F}_i [\hat{\mathbf{u}}(\mathbf{x}_j; \boldsymbol{\theta}_u^{k-1}); \hat{\gamma}(\mathbf{x}_j; \boldsymbol{\theta}_\gamma^{k-1})]$  for  $1 \leq i \leq N, 1 \leq j \leq M$

$\lambda_h^k \leftarrow \max (\lambda_h^{k-1} + 2\mu_h^{k-1} h (\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u^{k-1}), \hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma^{k-1})), 0)$

$\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k \leftarrow \arg \min_{\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma} \mathcal{L}^k(\boldsymbol{\theta}_u, \boldsymbol{\theta}_\gamma)$ : Train the networks  $\hat{\mathbf{u}}(\mathbf{x}; \boldsymbol{\theta}_u)$  and  $\hat{\gamma}(\mathbf{x}; \boldsymbol{\theta}_\gamma)$  from the initialization of  $\boldsymbol{\theta}_u^{k-1}$  and  $\boldsymbol{\theta}_\gamma^{k-1}$ , until the training loss is converged

**until**  $\mathcal{L}_{\mathcal{F}}(\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k)$  and  $\mathcal{L}_h(\boldsymbol{\theta}_u^k, \boldsymbol{\theta}_\gamma^k)$  are smaller than a tolerance

---



# Topology optimization of fluids in Stokes flow

- Solid:  $\rho = 0$ ; fluid:  $\rho = 1$
- Minimize a objective of dissipated power

$$\mathcal{J} = \int_{\Omega} \left( \frac{1}{2} \nabla \mathbf{u} : \nabla \mathbf{u} + \frac{1}{2} \alpha \mathbf{u}^2 \right) dx dy$$

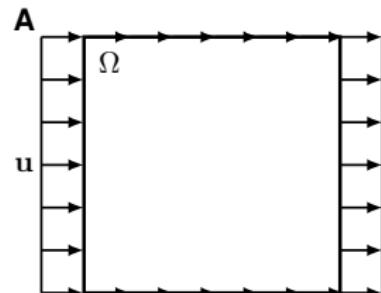
- Generalized Stokes equation with Darcy's law

$$-\nu \Delta \mathbf{u} + \nabla p = \mathbf{f}$$

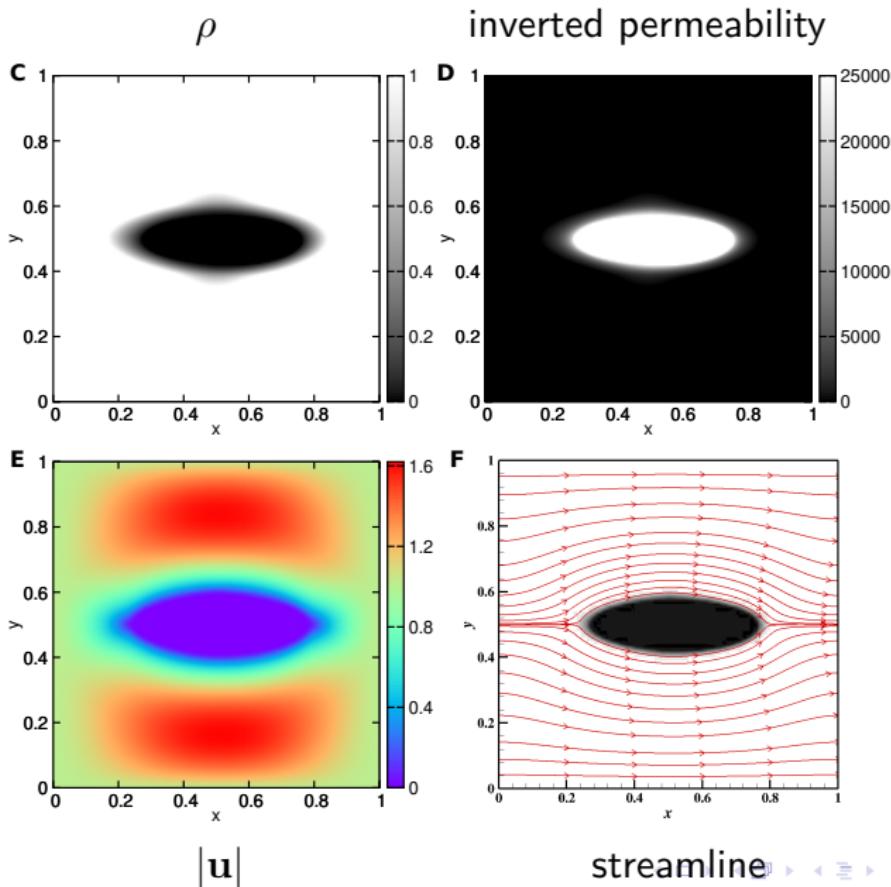
$$\nabla \cdot \mathbf{u} = 0$$

- $\mathbf{f} = \alpha \mathbf{u}$ : the Brinkman term
- $\alpha$ : inverted permeability  $\alpha(\rho) = \bar{\alpha} + (\underline{\alpha} - \bar{\alpha})\rho \frac{1+q}{\rho+q}$
- fluid volume constraint:

$$\int_{\Omega} \rho dx dy \leq \gamma = 0.9$$



# Topology optimization of fluids in Stokes flow



# Inductive bias

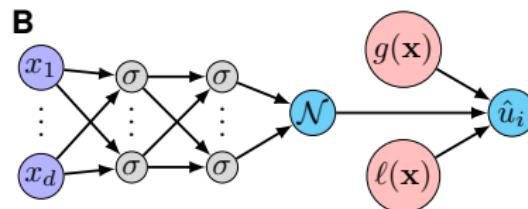
**ML model architecture:** Embed any prior knowledge

- CNN: translation invariance
- Partially fixed-value network (Dirichlet BC, IC):  
 $u(0) = 0, u(1) = 1: g(x) = x, \ell(x) = x(1 - x)$

$$\mathcal{N}(x) = g(x), \quad x \in \Gamma_D \subset \Omega$$

$$\hat{u}(x) = g(x) + \ell(x)\mathcal{N}(x)$$

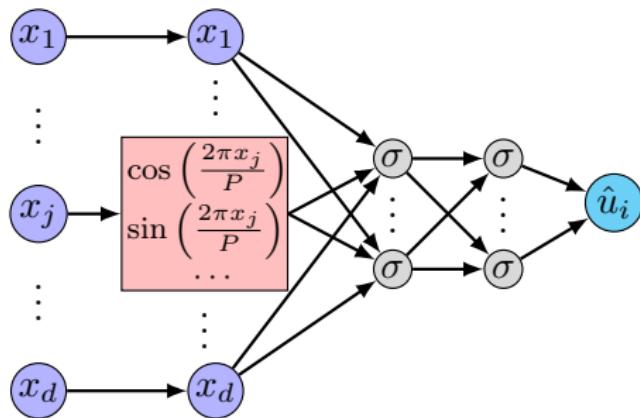
$$\begin{cases} \ell(\mathbf{x}) = 0, & \mathbf{x} \in \Gamma_D, \\ \ell(\mathbf{x}) > 0, & \mathbf{x} \in \Omega - \Gamma_D. \end{cases}$$



# Inductive bias

- Periodic network (Periodic BC) via Fourier series:

$$\{1, \cos\left(\frac{2\pi x_j}{P}\right), \sin\left(\frac{2\pi x_j}{P}\right), \cos\left(\frac{4\pi x_j}{P}\right), \sin\left(\frac{4\pi x_j}{P}\right), \dots\}$$



Can use as few as two terms  $\{\cos\left(\frac{2\pi x_j}{P}\right), \sin\left(\frac{2\pi x_j}{P}\right)\}$



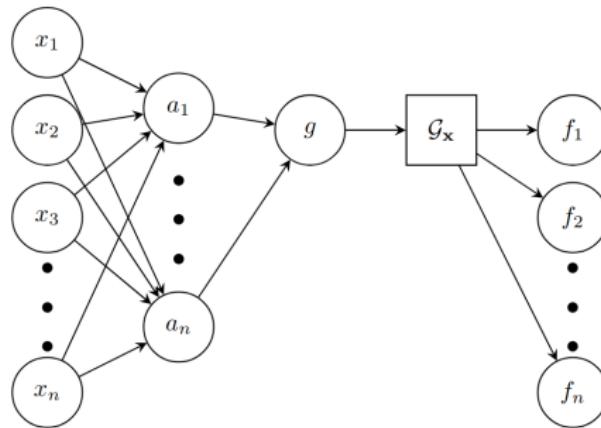
# Inductive bias

- Linearly constrained neural networks, e.g., divergence-free

$$\nabla \cdot \mathbf{f} = \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} = 0$$

Divergence of the curl is 0:  $\nabla \cdot (\nabla \times g) \equiv 0$ , so choose

$$\mathbf{f} = \nabla \times g$$



# Inductive bias

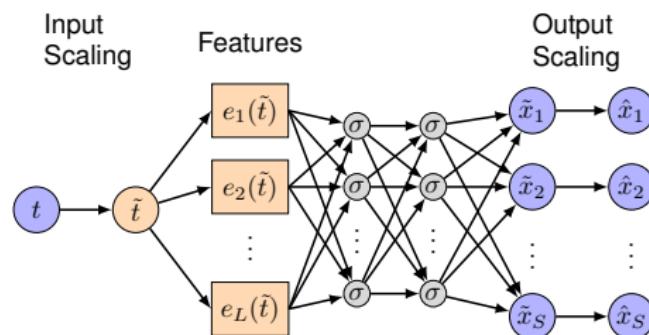
**Free lunch:** The network satisfies the physical constraints **automatically** and **exactly**.

- Do it if possible
- Designed case by case
- Currently limited to relatively simple and well-defined physics



# Tips on training PINN

- Normalize inputs and outputs to  $\mathcal{O}(1)$ 
  - Preprocessing via change of variable
  - Network normalization layer
- Feature expansion
  - Patterns of the system, e.g., periodicity  $\sin(kt)$ , fast decay  $e^{-kt}$ , multiscale features
  - Keep  $t$  as one feature



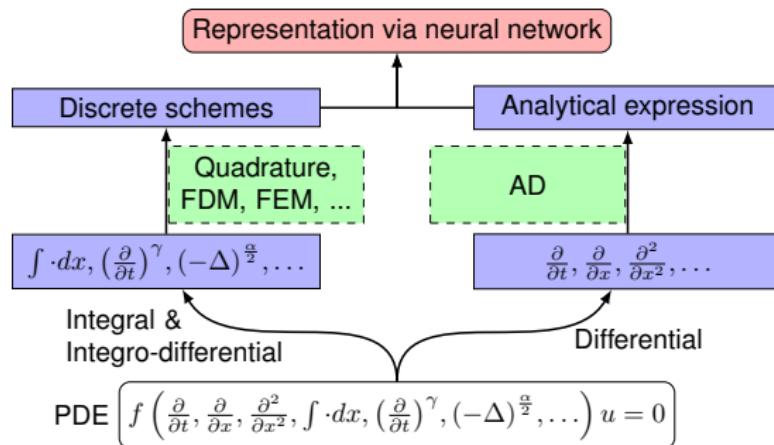
- Hard-constraint boundary/initial conditions
- Multiple outputs: One large network vs. multiple small networks

# Tips on training PINN

- Single precision vs. Double precision
- Adam + L-BFGS
- Loss balancing via inverse of losses
  - ▶ At the beginning of training, select weights s.t. the weighted losses are of the same order of magnitude  $\mathcal{O}(1)$
  - ▶ Optional: Update repeatedly during training
- Resample training points every certain epochs
- Inverse problem: Two-stage training
  - ① Train using the data loss only for some iterations, e.g., 1000
  - ② Train using all the losses
- Hyperparameter tuning
  - ▶ Grid search, Random search, Bayesian optimization



# PINNs for solving integro-differential equations



e.g., Riemann-Liouville directional fractional derivative of order  $\alpha \in (1, 2)$

$$\begin{aligned} D_{\theta}^{\alpha} u(x) &= \frac{1}{\Gamma(2-\alpha)} (\theta \cdot \nabla)^2 \int_0^{+\infty} \xi^{1-\alpha} u(x - \xi \theta) d\xi \quad x, \theta \in \mathbb{R}^D \\ &= \frac{1}{(\Delta x)^{\alpha}} \sum_{k=1}^{\lceil \lambda d(\mathbf{x}, \theta, \Omega) \rceil} (-1)^k \binom{\alpha}{k} u(x - (k-1)\Delta x \theta) + \mathcal{O}(\Delta x) \end{aligned}$$

---

Pang\*, Lu\*, et al., *SIAM J Sci Comput*, 2019

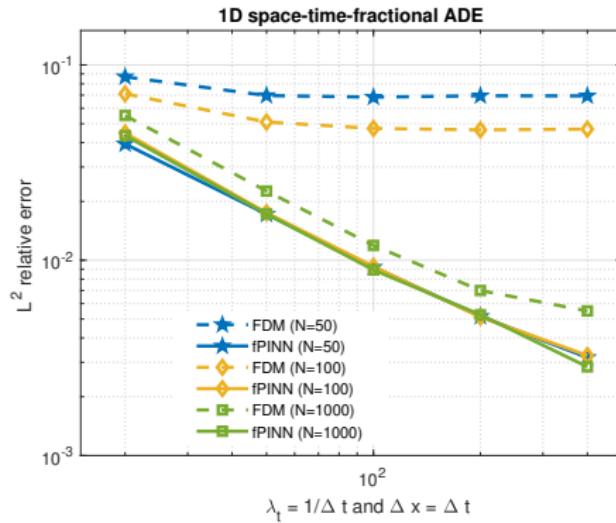
Lu et al., *SIAM Rev*, 2021

# Space-time-fractional PDEs with black-box (BB) terms

Fractional advection-diffusion equation with zero BC

$$\frac{\partial^\gamma u(\boldsymbol{x}, t)}{\partial t^\gamma} = -c(-\Delta)^{\alpha/2}u(\boldsymbol{x}, t) - \boldsymbol{v} \cdot \nabla u(\boldsymbol{x}, t) + f_{BB}(\boldsymbol{x}, t)$$

Forcing term  $f_{BB}$  only known at scattered spatio-temporal points



PINN vs. FDM: no interpolation error

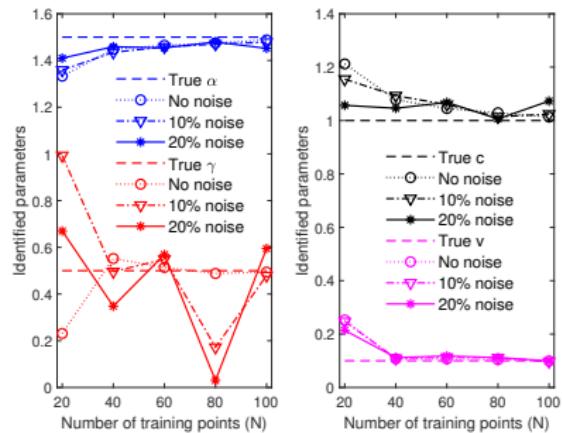
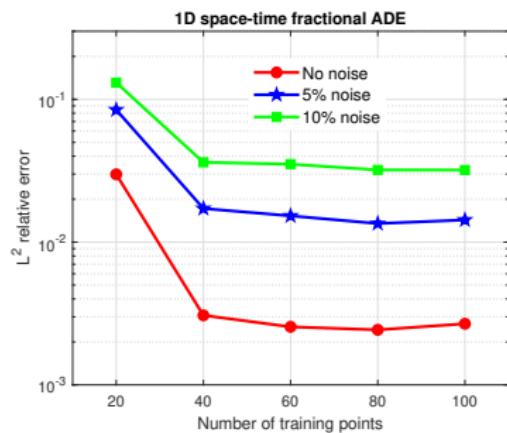
Pang\*, Lu\*, et al., SIAM J Sci Comput, 2019

# Space-time-fractional PDEs with noisy terms

$$\frac{\partial^\gamma u(\boldsymbol{x}, t)}{\partial t^\gamma} = -c(-\Delta)^{\alpha/2}u(\boldsymbol{x}, t) - \boldsymbol{v} \cdot \nabla u(\boldsymbol{x}, t) + f_{BB}(\boldsymbol{x}, t)$$

Gaussian noise added to the forcing term  $f_{BB}$

(left) Forward problem & (right) Inverse problem



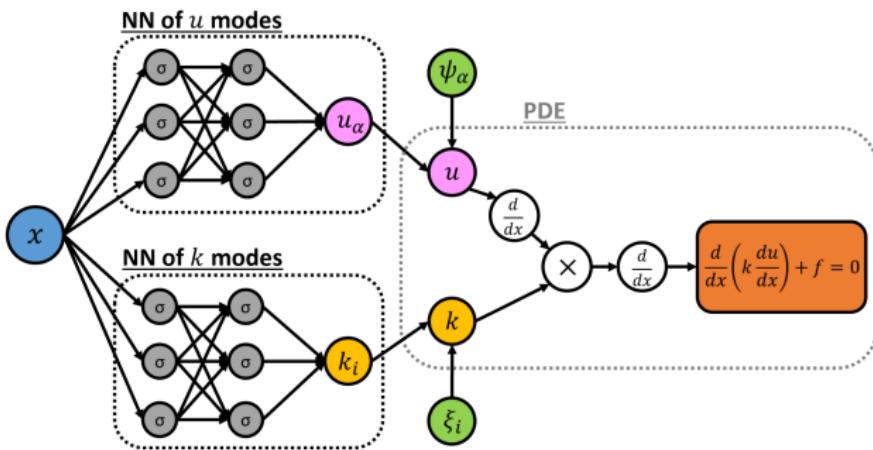
PINN: noise easily handled by regularization

Pang\*, Lu\*, et al., SIAM J Sci Comput, 2019

# PINNs for solving stochastic PDEs

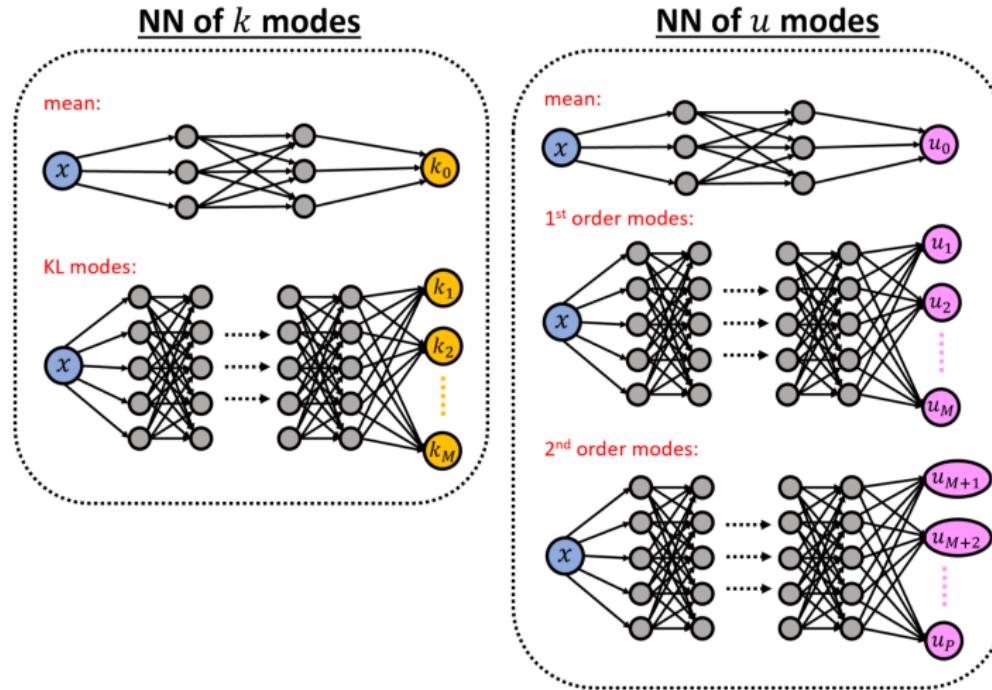
$$-\frac{d}{dx} \left( k(x; \omega) \frac{du}{dx} \right) = f(x), \quad x \in [-1, 1], \quad \omega \in \Omega$$
$$u(-1) = u(1) = 0$$

- Karhunen-Loëve expansion:  $k(x; \omega_s) = \widehat{k}_0(x) + \sum_{i=1}^M \sqrt{\lambda_i} \widehat{k}_i(x) \xi_{s,i}$
- arbitrary polynomial chaos:  $u(x; \omega_s) = \sum_{\alpha=0}^P \widehat{u}_{\alpha}(x) \psi_{\alpha}(\boldsymbol{\xi}_s)$



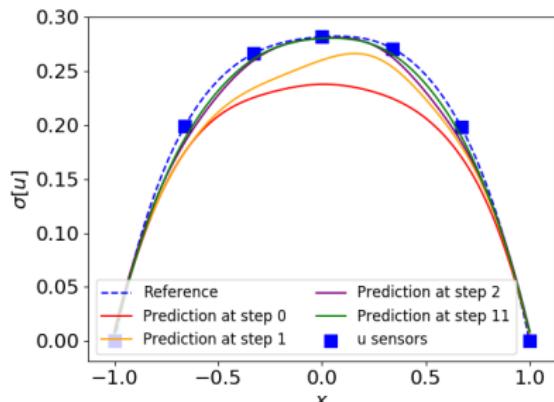
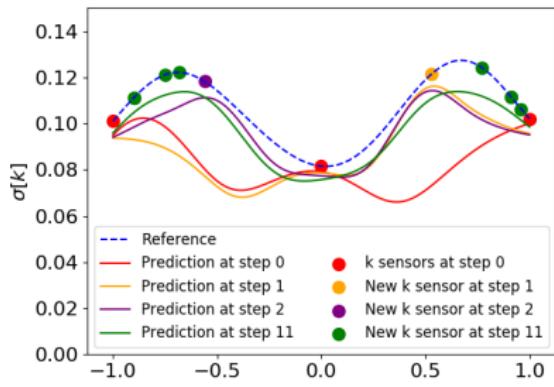
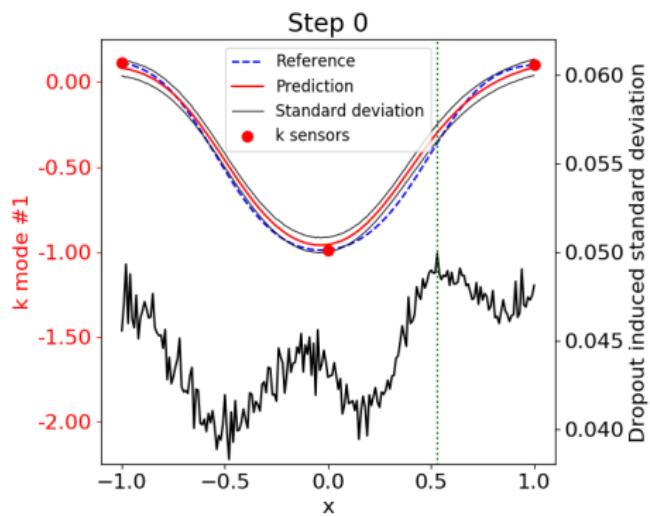
# PINNs for solving stochastic PDEs

Use different NNs to learn the mean and modes of different scales



# Active learning for stochastic inverse problem

**Dropout-induced uncertainty** serves as the guidance for *active learning*.



# Open-source software: DeepXDE

NEWS FEATURE · 20 JANUARY 2021

nature

## Ten computer codes that transformed science

From Fortran to arXiv.org, these advances in programming and platforms sent biology, climate science and physics into warp speed.



### Physics-informed deep learning



> 200,000 downloads, 700 GitHub Stars

Used in universities, national labs, and industry:



Los Alamos  
NATIONAL LABORATORY



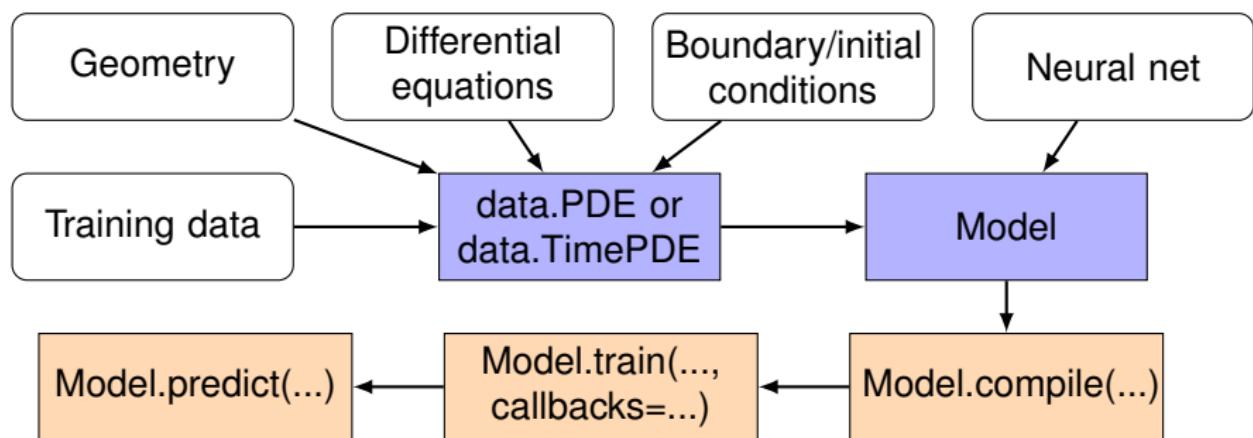
Sandia  
National  
Laboratories

Ansys SIEMENS rocketML ANAlytica



# Usage of DeepXDE

Solving differential equations in DeepXDE is no more than **specifying the problem using the build-in modules**.



# I. Time-independent problems: Poisson equation

2D Poisson equation over an L-shaped domain  $\Omega = [-1, 1]^2 \setminus [0, 1]^2$ :

$$-\Delta u(x, y) = 1, \quad (x, y) \in \Omega, \quad u(x, y) = 0, \quad (x, y) \in \partial\Omega$$

## ① geometry

```
1 geom = dde.geometry.Polygon(  
2     [[0, 0], [1, 0], [1, -1], [-1, -1], [-1, 1], [0, 1]])
```

## ② PDE via automatic differentiation

```
1 def pde(x, y):  
2     dy_xx = dde.grad.hessian(y, x, i=0, j=0)  
3     dy_yy = dde.grad.hessian(y, x, i=1, j=1)  
4     return -dy_xx - dy_yy - 1
```

## ③ BC: Dirichlet, Neumann, Robin, periodic, and a general BC

```
1 def boundary(x, on_boundary):  
2     return on_boundary # Default: entire geometry boundary  
3 def func(x):  
4     return 0 # Value  
5  
6 bc = dde.DirichletBC(geom, func, boundary)
```

# I. Time-independent problems: Poisson equation

- ④ “data”: geometry + PDE + BC + “training” points

```
1 data = dde.data.PDE(  
2     geom, pde, bc, num_domain=1000, num_boundary=100, ...)
```

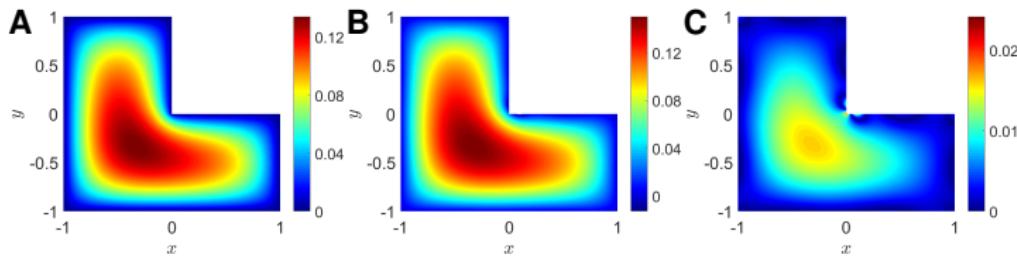
- ⑤ network, e.g., feed-forward network

```
1 net = dde.maps.FNN([2]+[50]*4+[1], "tanh", "Glorot uniform",  
...)
```

- ⑥ model: data + network, and train

```
1 model = dde.Model(data, net)  
2 model.compile("adam", lr=0.001, ...)  
3 model.train(epochs=50000, ...)
```

(A) spectral element, (B) PINN, (C) error



## II. Time-dependent problems: Diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} - e^{-t}(1 - \pi^2) \sin(\pi x), \quad x \in [-1, 1], t \in [0, 1]$$

with Dirichlet BC. (Exact solution  $u(x, t) = e^{-t} \sin(\pi x)$ )

- geometry

```
1 geom = dde.geometry.Interval(-1, 1)
2 timedomain = dde.geometry.TimeDomain(0, 1)
3 geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

- IC, similar to Dirichlet BC

```
1 def func(x):
2     return np.sin(np.pi * x[:, 0:1]) * np.exp(-x[:, 1:])
3
4 ic = dde.IC(geomtime, func, lambda _, on_initial: on_initial)
```

- “data”: geometry + PDE + BC/IC + “training” points

```
1 data = dde.data.TimePDE(
2     geomtime, ..., [bc, ic],
3     num_domain=40, num_boundary=20, num_initial=10, ...)
```

### III. ODE/PDE system: Lorenz system

$$\frac{dx}{dt} = \rho(y - x), \quad \frac{dy}{dt} = x(\sigma - z) - y, \quad \frac{dz}{dt} = xy - \beta z$$

- ODE system

```
1 def Lorenz_system(x, y):
2     y1, y2, y3 = y[:, 0:1], y[:, 1:2], y[:, 2:]
3     dy1_x = dde.grad.jacobian(y, x, i=0)
4     dy2_x = dde.grad.jacobian(y, x, i=1)
5     dy3_x = dde.grad.jacobian(y, x, i=2)
6     return [
7         dy1_x - C1 * (y2 - y1),
8         dy2_x - y1 * (C2 - y3) + y2,
9         dy3_x - y1 * y2 + C3 * y3
10    ]
```

- ICs

```
1 ic1 = dde.IC(geom, ..., ..., component=0)
2 ic2 = dde.IC(geom, ..., ..., component=1)
3 ic3 = dde.IC(geom, ..., ..., component=2)
```

## IV. Integro-differential equations: Volterra IDE

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x} y(t) dt, \quad y(0) = 1$$

- kernel

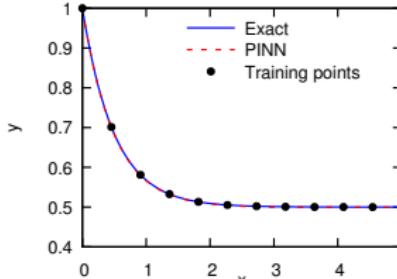
```
1 def kernel(x, s):  
2     return np.exp(s - x)
```

- IDE

```
1 def ide(x, y, int_mat):  
2     rhs = tf.matmul(int_mat, y) # \int_0^x exp(t-x) y(t) dt  
3     lhs1 = tf.gradients(y, x)[0] # dy/dx  
4     return (lhs1 + y)[:tf.size(rhs)] - rhs
```

- “data”: geometry + IDE + BC + “training” points

```
1 data = dde.data.IDE(..., ide, ..., quad_deg=20, kernel=kernel,  
...)
```



## V. Inverse problems

A diffusion-reaction system on  $x \in [0, 1]$ ,  $t \in [0, 10]$ :

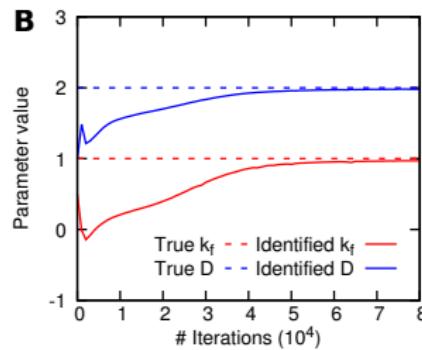
$$\frac{\partial C_A}{\partial t} = D \frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2, \quad \frac{\partial C_B}{\partial t} = D \frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2$$

- Define  $D$  and  $k_f$  as trainable variables

```
1 kf = dde.Variable(0.05)
2 D = dde.Variable(1.0)
```

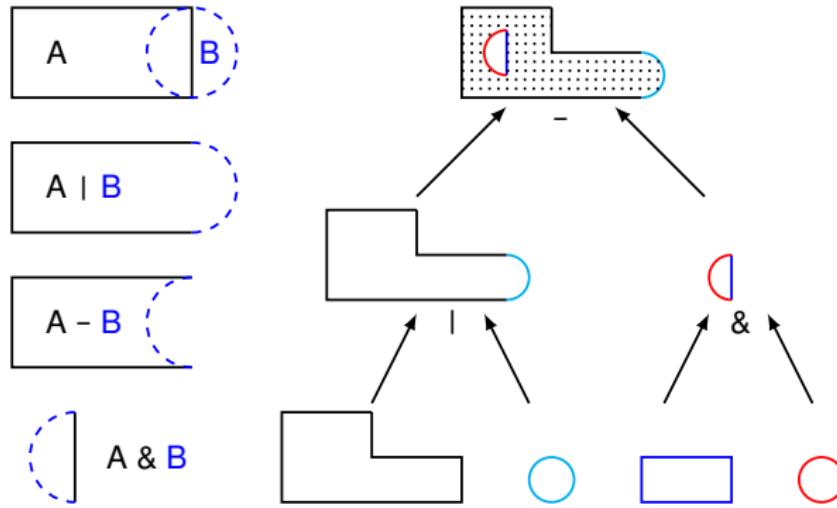
- Define  $C_A$  measurements as Dirichlet BC (the same for  $C_B$ )

```
1 observe_x = ... # All the locations of measurements
2 observe_Ca = ... # The corresponding measurements of Ca
3 observe = dde.PointSetBC.observe_x, observe_Ca, component=0)
```

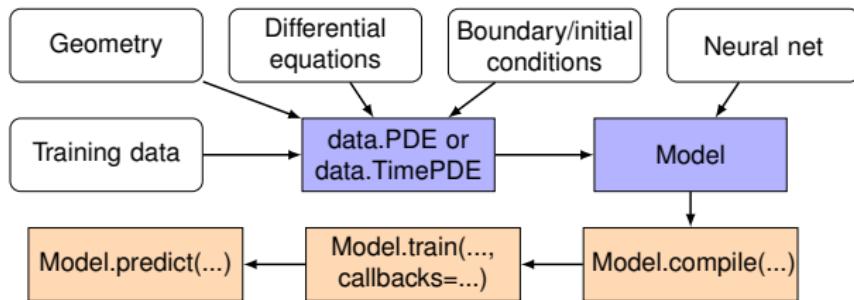


# Complex geometry: Constructive solid geometry

- Primitive geometries
  - ▶ interval, triangle, rectangle, polygon, disk, cuboid, sphere
- boolean operations:
  - ▶ Union  $A|B$
  - ▶ difference  $A - B$
  - ▶ intersection  $A\&B$



# DeepXDE



- Short and comprehensive code
- Three backends: TensorFlow 1.x, TensorFlow 2.x, and PyTorch
- No TensorFlow/PyTorch knowledge
- Lots of useful features
- Well-structured and highly configurable

## Other functions

- Multi-fidelity NN (**Lu** et al., *PNAS*, 2020)
- DeepONet: Operator learning (**Lu** et al., *Nature Mach Intell*, 2021)

# **L**u Group

University of Pennsylvania



- <https://lu.seas.upenn.edu>
- Email: lulu1@seas.upenn.edu

