Computational Mathematics and AI

Lecture 7: Scientific Machine Learning for PDEs

Lars Ruthotto
Departments of Mathematics and Computer Science
lruthotto@emory.edu
larsruthotto

slido.com #CBMS25

Reading List

Historical Context: First works on neural approximations of PDEs and operators in the 90s. Popularized in the mid 2010s, benchmarks reveal accuracy gap to traditional methods.

Key Readings:

1. Raissi et al. (2019) – Physics-Informed Neural Networks. *J. Comp. Physics*
   Foundational PINN framework for forward/inverse problems.
2. Lu et al. (2021a) – DeepONet: Learning Nonlinear Operators. *Nature Mach. Intell.*
   Universal approximation for operators.
3. Li et al. (2021a) – Fourier Neural Operator for Parametric PDEs. *ICLR*
   Spectral methods for fast operator learning.
4. Takamoto et al. (2022a) – PDEBench. *NeurIPS Datasets*
   Standardized benchmarks revealing accuracy gaps.
5. Krishnapriyan et al. (2021a) – PINN Failure Modes. *NeurIPS*
   Spectral bias and optimization challenges.

Lecture Outline: Classical Methods $\to$ PINNs $\to$ Neural Operators $\to$ Hybrid

Running Example: 2D Heterogeneous Darcy Flow

The PDE

$$-\nabla \cdot (\kappa(x,y)\nabla u) = f \quad \text{in } \Omega = [0,1]^2$$

with $u = 0$ on $\partial\Omega$

Physical meaning: Porous media flow

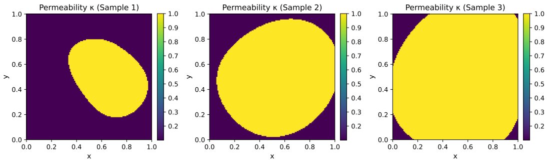
- $\kappa(x,y)$: permeability field (input)
- $u(x,y)$: pressure/potential (output)
- $f = 1$: constant forcing

Challenge

- κ varies 2–3 orders of magnitude
- High-frequency features in κ
- 128×128 grid = 16,384 unknowns

Dataset: PDEBench

- 10,000 samples (train/val/test)
- κ from Gaussian random fields
- Reference solutions via FEM



running example: same problem, all methods, fair comparison

Classical Baseline: Finite Differences + CG

Discretization

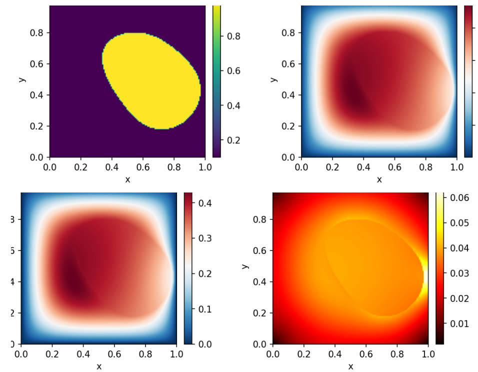
- 5-point stencil (FD $\equiv$ P1 FEM)
- Harmonic averaging of κ at faces
- Sparse linear system $A\mathbf{u} = \mathbf{b}$

Solver

- Conjugate Gradient (CG)
- IC(0) preconditioner
- Tolerance: 10^{-8} relative

Performance (5 samples)

- Solve time: 0.28s
- Iterations: 5–8 (with IC)
- Rel. L^2 vs PDEBench: **8.7%**

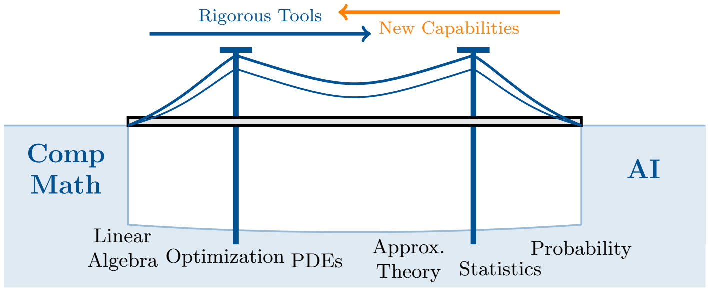


Why $\sim$9% Error?

- FD uses harmonic avg of κ
- PDEBench: spectral solver
- *Different discretizations!*

baseline: 0.28s, $\sim$9% vs PDEBench—grounds later neural comparisons

Roadmap: Scientific ML for PDEs



Goal: Use AI to accelerate or improve classical PDE solvers for

- Outer-loop problems: Inverse problems, optimal design
- Multi-scale closures (turbulence)
- High dimensions ($d > 6$)

Lecture Outline: Theory $\rightarrow$ PINNs $\rightarrow$ Neural Operators $\rightarrow$ Hybrid Methods

Theoretical Foundations

Why Neural Networks for PDEs?

Classical Universal Approximation Theorem (Cybenko 1989)
Single hidden layer net can approximate any $f \in C(\mathbb{R}^n, \mathbb{R})$ to arbitrary accuracy
Common argument: PDE solutions $u(x, t)$ are functions $\Rightarrow$ NNs can represent them

Operator Approximation Theorem (Chen & Chen 1995)
Neural nets can approximate nonlinear operators $G : V \to W$ between function spaces
Common argument: PDEs define operators mapping inputs (ICs, BCs, params) to solutions $\Rightarrow$ NNs can represent solution operators

Critical Caveats

- Approximation *exists* $\neq$ *efficiently learnable*
- May require infeasible width / data
- Finding good weights is a non-convex optimization challenge

In theory there is no gap between theory and practice, in practice there may be

Automatic Differentiation: Enabling PINNs

Modern ML frameworks (PyTorch, JAX) compute **exact derivatives** through computational graphs

How It Enables PINNs

Neural network: $u_\theta(x,t)$

$\downarrow$ autodiff

Derivatives: $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial t}, \frac{\partial^2 u}{\partial x^2}$ (exact via chain rule)

$\downarrow$

PDE residual: $u_t - \alpha u_{xx} = ?$

$\downarrow$

Loss: minimize residual w.r.t. θ

Why This is Helpful

- **Exact derivatives** (not finite difference approximations)
- **Dimension-agnostic** (same code 1D $\to$ 10D)
- **Complex PDEs** (nonlinear, coupled, high-order)

Two Paradigms: PINNs vs Neural Operators

Fundamental Distinction

Aspect	PINNs	Neural Operators
Learn what?	One solution $u(x,t)$	Operator G: inputs $\to u(x,t)$
Theory	Function approximation	Operator approximation
Training data	PDE residual + BCs	Many solved instances
Optimization	Physics-informed	Supervised learning
Data cost	Low (physics-only)	High (need 1000s PDEs)
Training cost	High (optimization)	Medium (supervised)
Inference cost	High (solve each)	Very low (forward pass)
Use case	One-off, inverse	Parametric, real-time

Example

- **PINN**: Given heat equation with specific $u_0(x)$, learn that $u(x,t)$
- **Neural Op**: Given 1000s of heat equations with varying u_0, learn map $u_0 \to u(x,t)$

function vs operator learning—fundamentally different

Physics-Informed Neural Networks

Physics-Informed Neural Networks (PINNs)

Idea: Train neural net to satisfy PDEs, boundary conditions, and data simultaneously

The PINN Method

Given PDE: $\mathcal{N}[u(\mathbf{x})] = 0$ (e.g., Burgers: $\mathcal{N}[u(\mathbf{x})] = u_t + uu_x - \nu u_{xx}$)

Three Steps:

1. **Represent solution**: Neural network $u_\theta(x, t)$
2. **Define composite loss**:

$$L = \lambda_r L_{\text{PDE}} + \lambda_b L_{\text{BC}} + \lambda_d L_{\text{data}}$$

   where $L_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} |\mathcal{N}[u_\theta](x_i)|^2$

3. **Train**: Gradient descent to minimize L

Theoretical Appeal

- Mesh-free, dimension-agnostic, seamless data fusion
- Joint solution-parameter learning for inverse problems

next: reality check from rigorous benchmarking

PINN for Darcy Flow: Heterogeneous κ

Given κ, find u_θ by minimizing

$$L_{\text{PINN}}(\theta) = L_{\text{PDE}}(\theta) + \lambda L_{BC}(\theta)$$

$$L_{\text{PDE}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} |\nabla\cdot(\kappa(x_i, y_i)\nabla u_\theta(x_i, y_i)) - f|^2$$

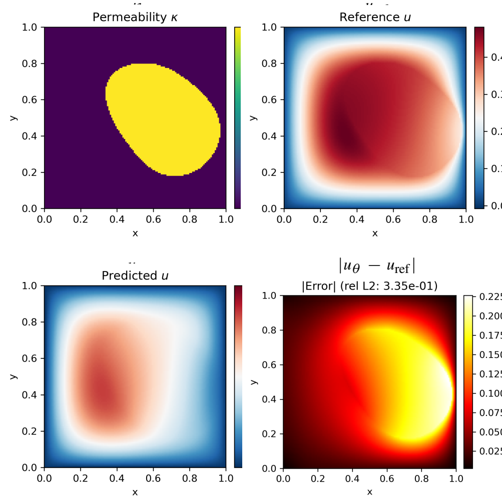
with cell-centered grid points (x_i, y_i).

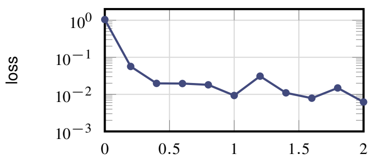
Architecture (HPO-tuned)

- 4 layers $\times$ 32 neurons, GELU
- 500 interior + 800 boundary points
- $\lambda_{\text{BC}} = 85$ (strong BC weighting)

Results

- Best loss: 1.3×10^{-4}
- Training: $\sim$300s (20k iterations)





Documented Difficulties in PINNs

1. **Spectral Bias** Krishnapriyan et al. 2021b
   - Networks learn low frequencies first, struggle with high frequencies
   - Cannot capture shocks, sharp gradients, thin boundary layers
   - Partial fix: Fourier features help but don't eliminate problem
2. **Gradient Pathologies** Wang et al. 2021
   - PDE, BC, data losses operate at vastly different scales
   - Gradient imbalance: some terms dominate, others ignored
   - Requires problem-specific tuning (no general rule for λ ratios)
3. **Optimization Difficulties** Krishnapriyan et al. 2021b; Takamoto et al. 2022b
   - Non-convex landscape with many poor local minima
   - Extreme sensitivity to initialization, learning rate, architecture
   - Reproducibility issues: early papers missed hyperparameter details

making PINNs work is more difficult and problem-specific than initially thought

Neural Operator Learning

Learn Once, Solve Many Times

The Concept

Train once on many examples $\rightarrow$ instant solve for new instances

Comparison

Aspect	PINNs	Neural Operators
Training paradigm	Solve one instance	Learn from many
Training cost	Low (physics)	High (need dataset)
Inference cost	High (optimization)	Very low (forward pass)
Use case	One-off	Parametric, real-time

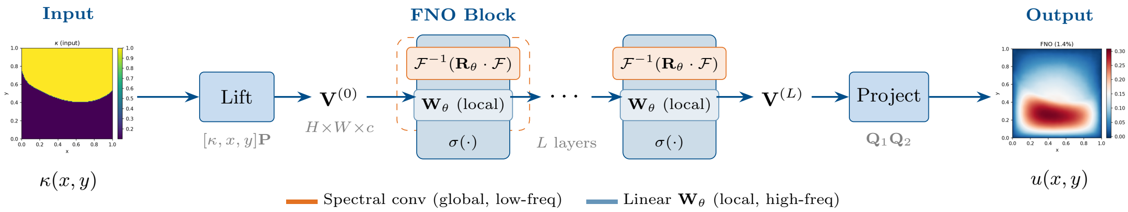
Two Architectures

- **DeepONet**: Branch (encode input) + Trunk (encode location) $\rightarrow$ $G(u)(x) \approx \sum_k b_k(u) \cdot t_k(x)$
- **Fourier Neural Operator (FNO)**: Learn in frequency domain, O($N \log N$) via FFT

Goal: amortize expensive offline training in massive outer-loop problems

Fourier Neural Operator (FNO) Architecture

Key Idea Li et al. 2021b: Learn operators in *frequency domain*



Spectral Convolution Layer

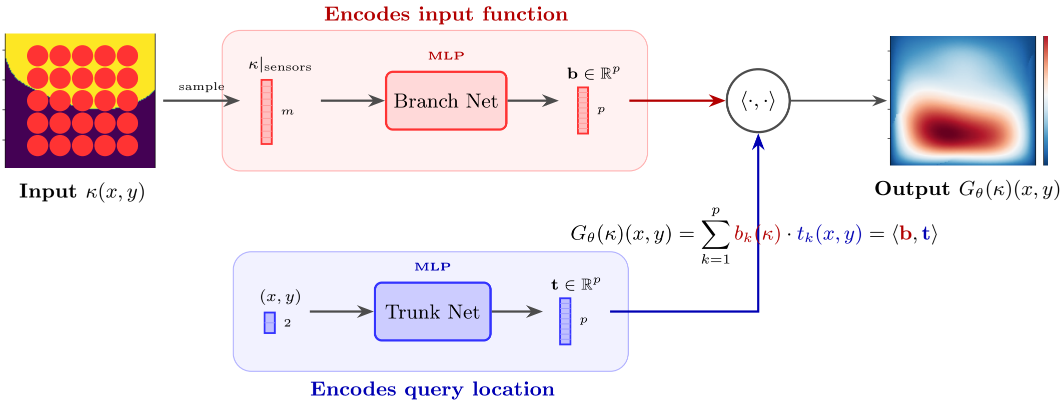
$$\mathcal{K}(\mathbf{V}) = \mathcal{F}^{-1}\left(\mathbf{R}_\theta \cdot \mathcal{F}(\mathbf{V})\right)$$

- $\mathbf{R}_\theta \in \mathbb{C}^{c \times c \times k \times k}$: learnable weights
- Truncate to k lowest modes per dimension

Key Properties

- FFT: $O(N \log N)$ per layer
- Resolution-invariant (discretization-free)
- Global receptive field (vs. local CNNs)

DeepONet Architecture



Key Idea Lu et al. 2021b: Separate encoding of *input function* and *query location*

Theoretical Foundation

Universal approximation theorem for operators (Chen & Chen, 1995)

Key Properties

- Mesh-free evaluation
- Flexible sensor placement
- Scales with latent dim p, not grid size

When Does Upfront Training Pay Off?

The Economics

Training cost: Dataset generation + GPU training time

Darcy Flow example:

- Training: 9000 samples, ~20-40 min GPU
- Inference: 5-8 ms per solve
- Classical: 0.56s per solve

Break-even Analysis

Neural operators beat classical when:

$$\underbrace{T_{\text{train}} + N \cdot T_{\text{infer}}}_{\text{Neural Op}} < \underbrace{N \cdot T_{\text{classical}}}_{\text{Classical}}$$

Rule of thumb: 10,000+ solves to amortize training

The Trade-off

	Accuracy	**Time**
Classical	~9%	0.3s
FNO	~7%	5ms
DeepONet	~8%	8ms

When It Makes Sense

- Parametric optimization (50k+ evals)
- Real-time control ($<$10ms required)
- Uncertainty quantification (100k samples)

trade-off: ~8% accuracy for 100× speed—economical for 10k+ solves

Darcy Flow Results

DeepONet for Darcy Flow: Branch-Trunk Architecture

Learn operator $G : \kappa \mapsto u$ from data

$$G(\kappa)(x,y) \approx \sum_{k=1}^{p} b_k(\kappa) \cdot t_k(x,y)$$

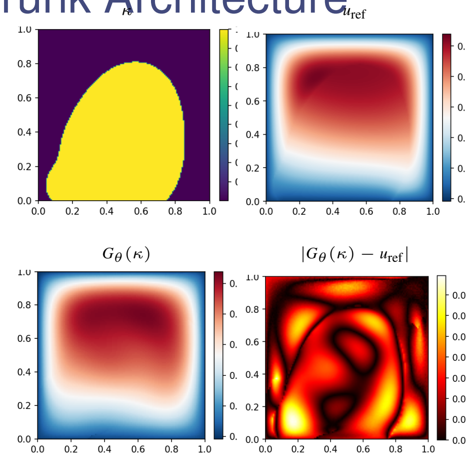
- **Branch**: encodes κ at sensor points
- **Trunk**: encodes query location (x, y)

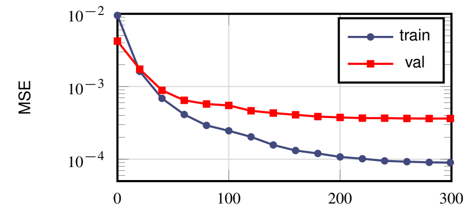
Architecture

- Latent dim: 256, Hidden: 3×512
- Parameters: 3.4M

Results

- Test MSE: 3.6×10^{-4}
- Test Rel. L^2: 8.3%
- Training: $\sim$12 min (300 epochs)





FNO for Darcy Flow: Fourier Neural Operator

Learn in frequency domain: $O(N \log N)$ via FFT

$$f_\theta(\mathbf{V}) = \underbrace{\mathcal{F}^{-1}\left(\mathbf{R}_\theta \cdot \mathcal{F}(\mathbf{V})\right)}_{\text{global}} + \underbrace{\mathbf{V}\,\mathbf{W}}_{\text{local}}$$

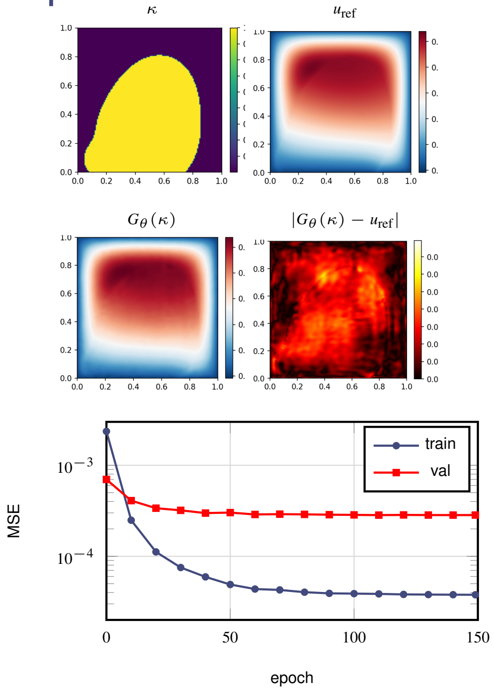
- **Fourier**: mixes k low-freq. modes
- **Linear**: channel mixing (high freq.)

Architecture (HPO-tuned)

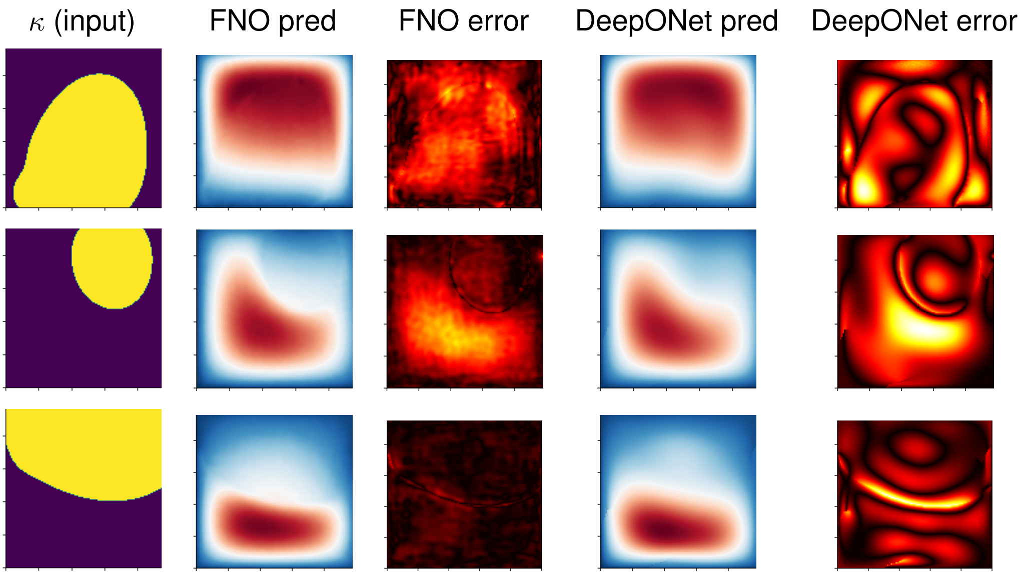
- Modes: 12, Width: 20, 4 layers
- Parameters: 926K

Results

- Test MSE: 2.8×10^{-4}
- Test Rel. L^2: 7.3%
- Training: $\sim$13 min (150 epochs)



FNO vs DeepONet: Test Samples



FNO: 7.3% rel. error — DeepONet: 8.3% rel. error

Summary: Performance on 2D Darcy Flow, PDEBench)

Method	Accuracy	Time	Training	Params	Data
Classical (CG+ILU)	10^{-8}	0.56s	—	—	—
PINN	$\sim 10^{-3*}$	80s	80s	31K	0
DeepONet	8.3%	8ms	20 min	3.4M	9K
FNO	7.3%	5ms	40 min	926K	9K

When to Use What

- **Single solve, high accuracy** $\rightarrow$ Classical
- **Inverse problem, sparse data** $\rightarrow$ PINN
- **1000+ parametric solves** $\rightarrow$ Neural operators
- **Real-time ($<$10ms)** $\rightarrow$ Neural operators

Hybrid Approaches

GNN-Enhanced Preconditioners (Trifonov et al. (2024))

Key Idea: Learn correction to incomplete Cholesky:

$$L(\theta) = L_{\text{IC}} + \alpha \cdot \text{GNN}(\theta, L_{\text{IC}}, b)$$

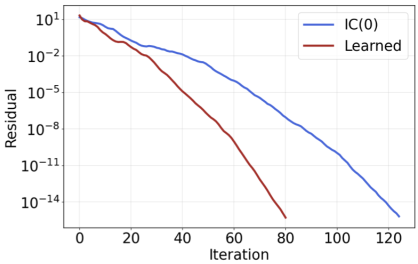
- Start from IC(0) or ICt(1) factor
- GNN: 5 message-passing rounds
- Preserves SPD structure

Training Loss

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \|L(\theta)L(\theta)^\top x_i - b_i\|_2^2$$

$b_i \sim \mathcal{N}(0, I)$, $x_i = A^{-1} b_i \Rightarrow$ emphasizes low frequencies

Results: 2D Diffusion



Method	Iters
IC(0)	95
PreCorrector	52

$\kappa : 270 \to 55$ (79% $\downarrow$)

ML augments classical preconditioner to improve performance

Summary

Benchmarking Best Practices

Need for Tough Baselines

- Always compare against state-of-the-art classical methods (FEniCS, PETSc)
- Same problem, same metrics, fair compute budgets

Reproducibility Checklist

- ☐ Full hyperparameters documented?
- ☐ Multiple runs with confidence intervals?
- ☐ Open-source code provided?
- ☐ Failure modes documented?

Honest Assessment

- PDEBench revealed 10^{-3} vs 10^{-6} accuracy gap
- Document limitations, don't cherry-pick successes
- Rigorous benchmarking prevents wasted effort

extraordinary claims require extraordinary evidence

Σ: Scientific ML for PDEs

What We Learned

- **Theory**: UAT + autodiff enable neural PDE methods
- **PINNs**: optimization overhead, perhaps promising for inverse problems
- **Neural Operators**: ∼8% accuracy, 100× faster after training
- **Hybrid**: Augment classical at bottlenecks (20-30% speedup)

When to Use What

- **High accuracy needed?** → Classical (only option)
- **10,000+ parametric solves?** → Neural operators
- **Real-time (<10ms)?** → Neural operators
- **High-dim (d>6)?** → PINNs (Lecture 8)

Running Example Results (Darcy Flow)

Method	Accuracy	Time
Classical CG	10^{-8}	0.56s
PINN	$\sim 10^{-3}$	300s
FNO	7.3%	5ms
DeepONet	8.3%	8ms

References I

- Krishnapriyan, A. S. et al. (2021a). “Characterizing Possible Failure Modes in Physics-Informed Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34.
- Krishnapriyan, Aditi et al. (2021b). “Characterizing Possible Failure Modes in Physics-Informed Neural Networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 34. Documented when/why PINNs fail—highly valuable negative results.
- Li, Z. et al. (2021a). “Fourier Neural Operator for Parametric Partial Differential Equations”. In: *International Conference on Learning Representations (ICLR)*.
- Li, Zongyi et al. (2021b). “Fourier Neural Operator for Parametric Partial Differential Equations”. In: *International Conference on Learning Representations (ICLR)*. Foundational FNO paper for fast operator learning.
- Lu, L. et al. (2021a). “Learning Nonlinear Operators via DeepONet Based on the Universal Approximation Theorem of Operators”. In: *Nature Machine Intelligence* 3.3, pp. 218–229.

References II

- Lu, Lu et al. (2021b). “Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators”. In: *Nature Machine Intelligence* 3.3. Original DeepONet paper for operator learning, pp. 218–229.
- Raissi, M. et al. (2019). “Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations”. In: *Journal of Computational Physics* 378, pp. 686–707.
- Takamoto, M. et al. (2022a). “PDEBench: An Extensive Benchmark for Scientific Machine Learning”. In: *NeurIPS Datasets and Benchmarks*.
- Takamoto, Makoto et al. (2022b). “PDEBench: An Extensive Benchmark for Scientific Machine Learning”. In: *NeurIPS Datasets and Benchmarks*. Benchmark revolution: standardized evaluation revealing accuracy gaps.
- Trifonov, Vladislav et al. (2024). “Learning from Linear Algebra: A Graph Neural Network Approach to Preconditioner Design for Conjugate Gradient Solvers”. In: *arXiv preprint arXiv:2405.15557*. GNN-based learned corrections to incomplete Cholesky.

References III

Wang, Sifan et al. (2021). "Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks". In: *SIAM Journal on Scientific Computing* 43.5. Critical analysis of PINN optimization difficulties, A3055–A3081.