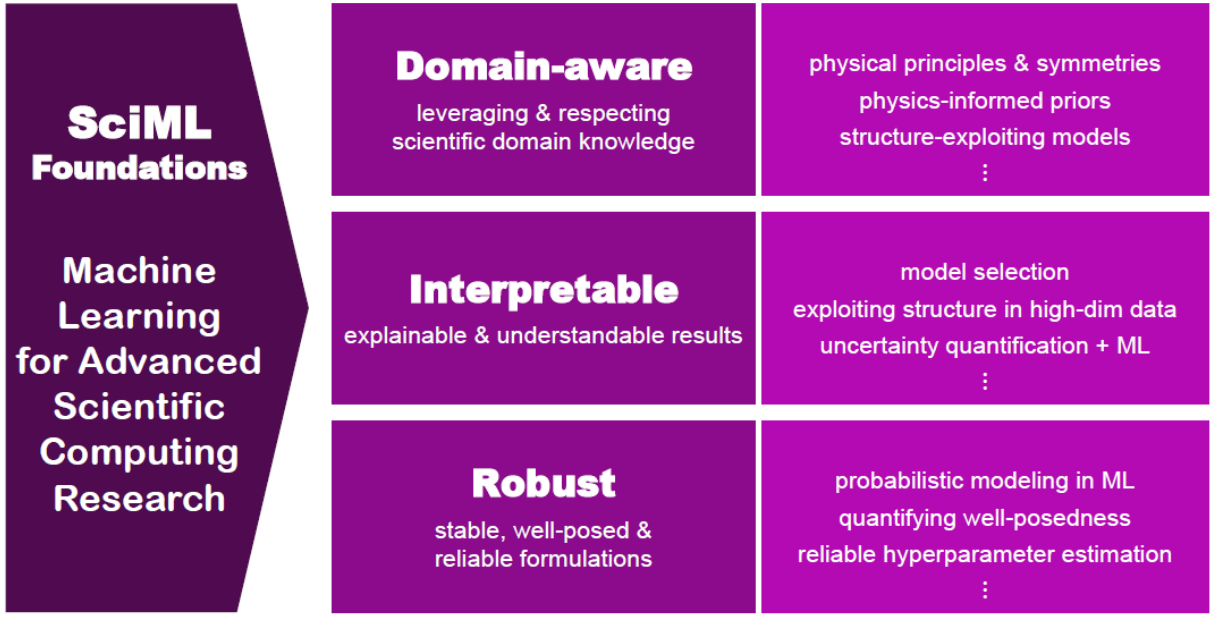
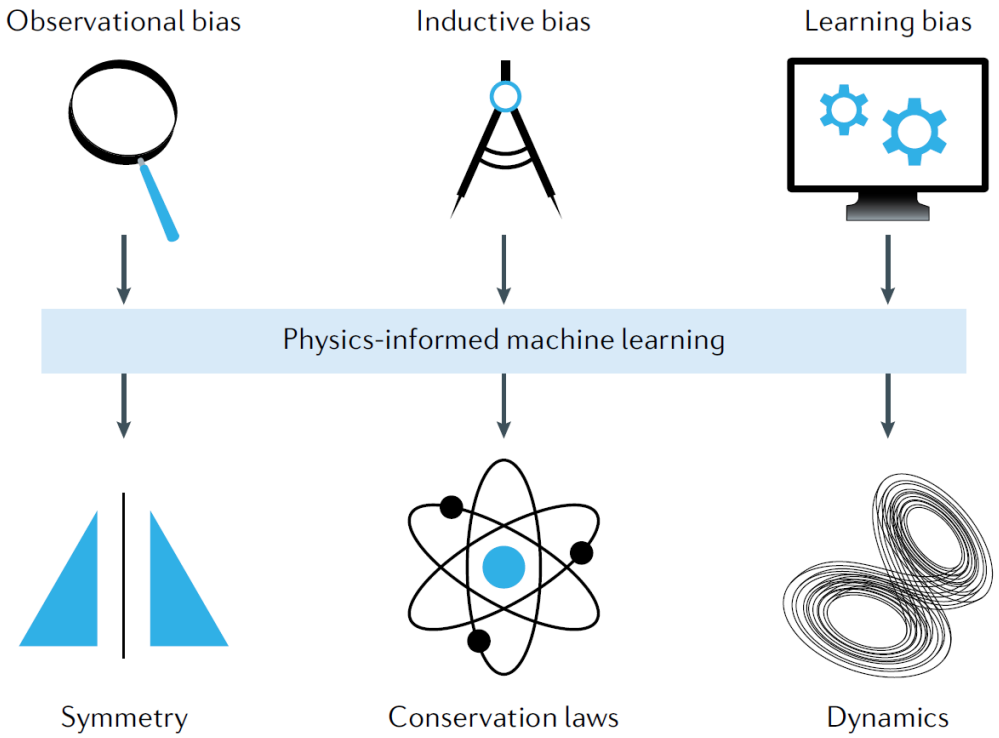
Introduction to Scientific ML

Nicolas Boulle

Scientific machine learning



[Baker et al., DOE Tech. Report, 2019]

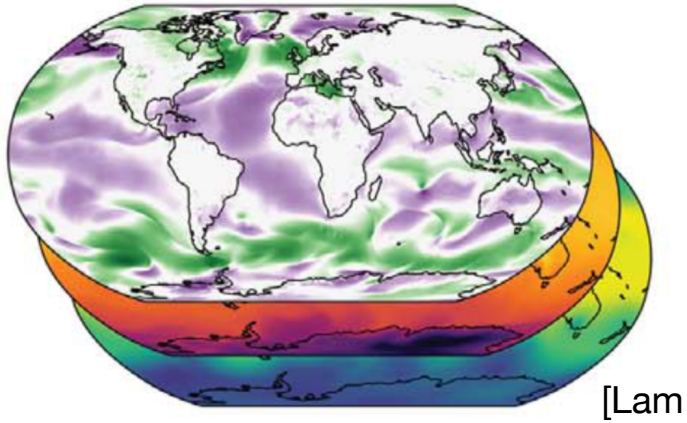


[Karniadakis et al., 2021]

Combining numerical analysis and machine learning for scientific discoveries.

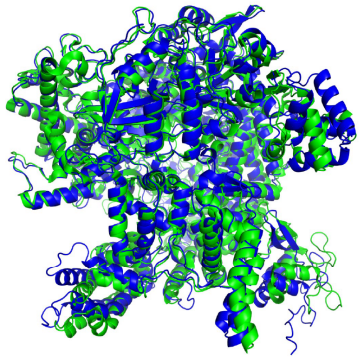
Recent applications of SciML

Weather forecasting



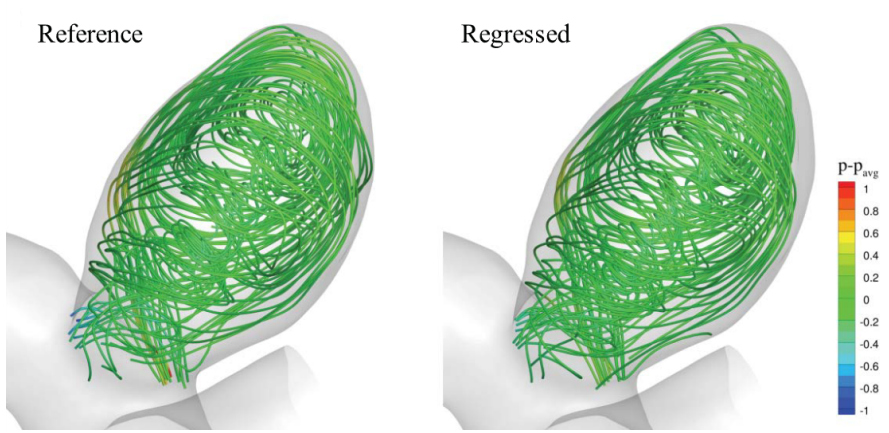
[Lam et al., Science, 2023]

Protein folding



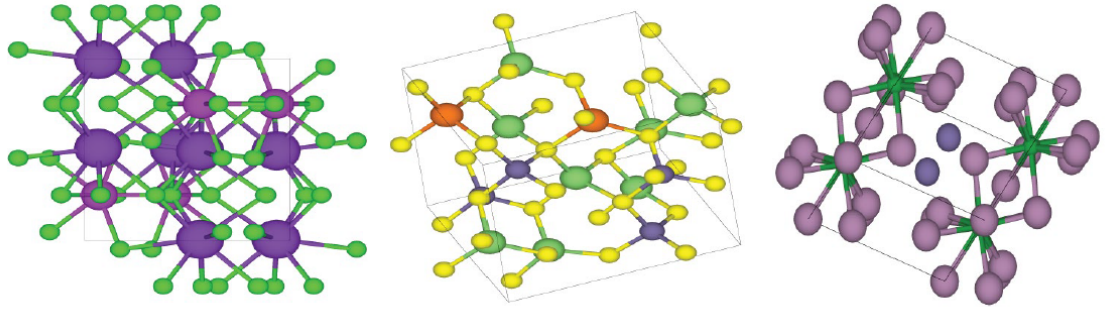
[Jumper et al., Nature, 2021]

Numerical simulations



[Raissi, Yazdani, Karniadakis, Science, 2020]

Materials discovery



K_2BiCl_5 $Li_4MgGe_2S_7$ Mo_5GeB_2

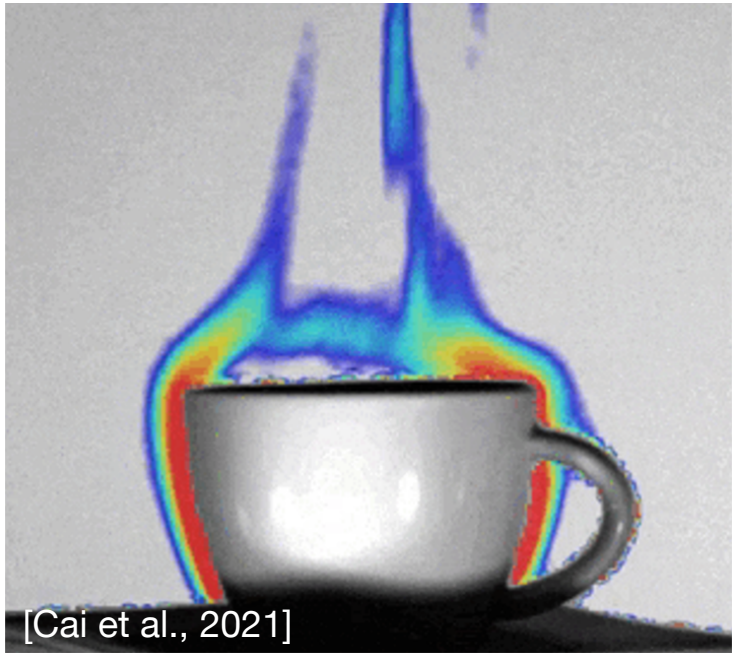
[Merchant et al., Nature, 2023]

Three areas to rule them all

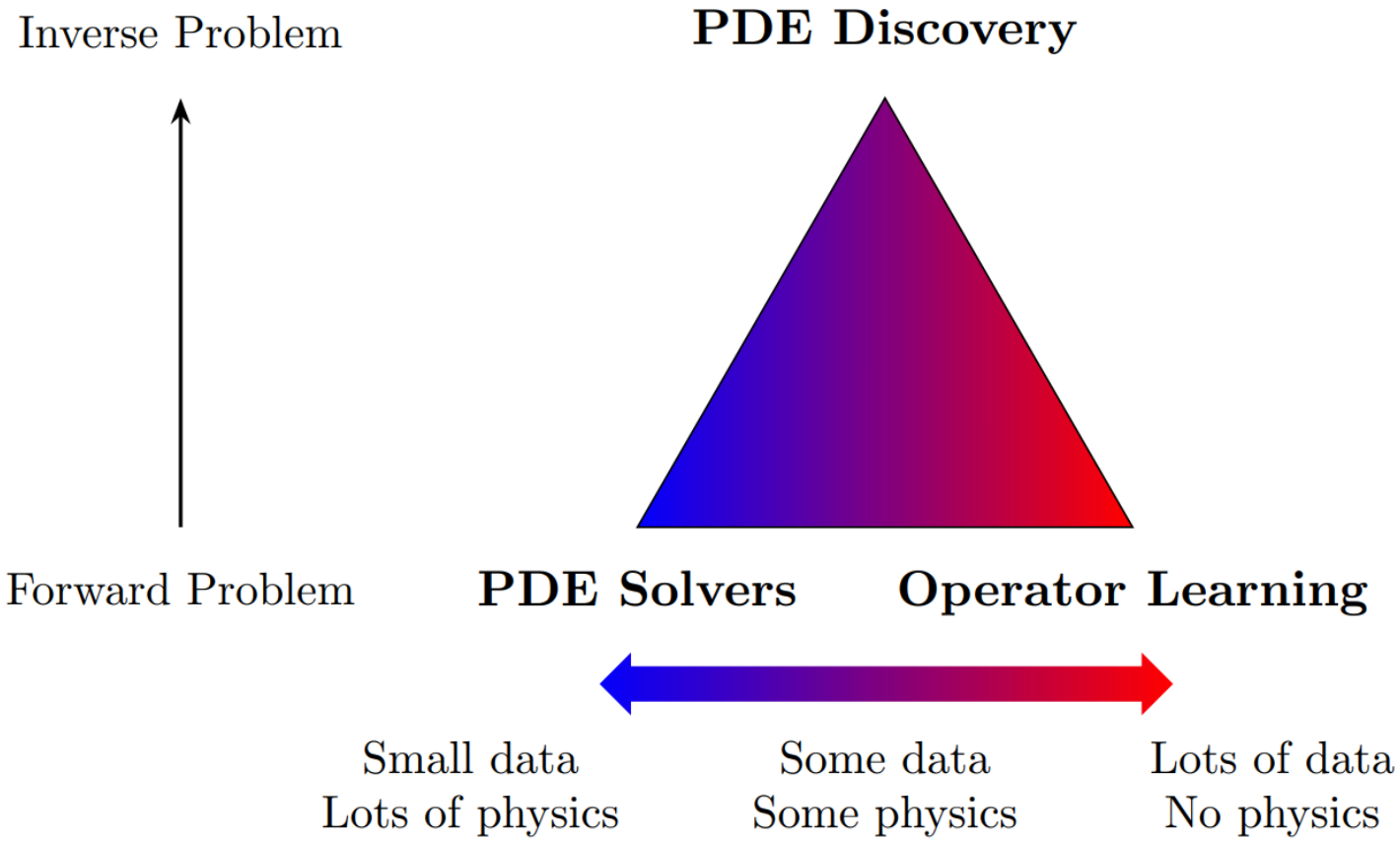
Physics-informed machine learning

George Em Karniadakis[1,2]✉, Ioannis G. Kevrekidis[3,4], Lu Lu[5], Paris Perdikaris[6], Sifan Wang[7] and Liu Yang[1]

Abstract | Despite great progress in simulating multiphysics problems using the numerical discretization of partial differential equations (PDEs), one still cannot seamlessly incorporate noisy data into existing algorithms, mesh generation remains complex, and high-dimensional problems governed by parameterized PDEs cannot be tackled. Moreover, solving inverse problems with hidden physics is often prohibitively expensive and requires different formulations and elaborate computer codes. Machine learning has emerged as a promising alternative, but training deep neural networks requires big data, not always available for scientific problems. Instead, such networks can be trained from additional information obtained by enforcing the physical laws (for example, at random points in the continuous space-time domain). Such physics-informed learning integrates (noisy) data and mathematical models, and implements them through neural networks or other kernel-based regression networks. Moreover, it may be possible to design specialized network architectures that automatically satisfy some of the physical invariants for better accuracy, faster training and improved generalization. Here, we review some of the prevailing trends in embedding physics into machine learning, present some of the current capabilities and limitations and discuss diverse applications of physics-informed learning both for forward and inverse problems, including discovering hidden physics and tackling high-dimensional problems.



[Cai et al., 2021]

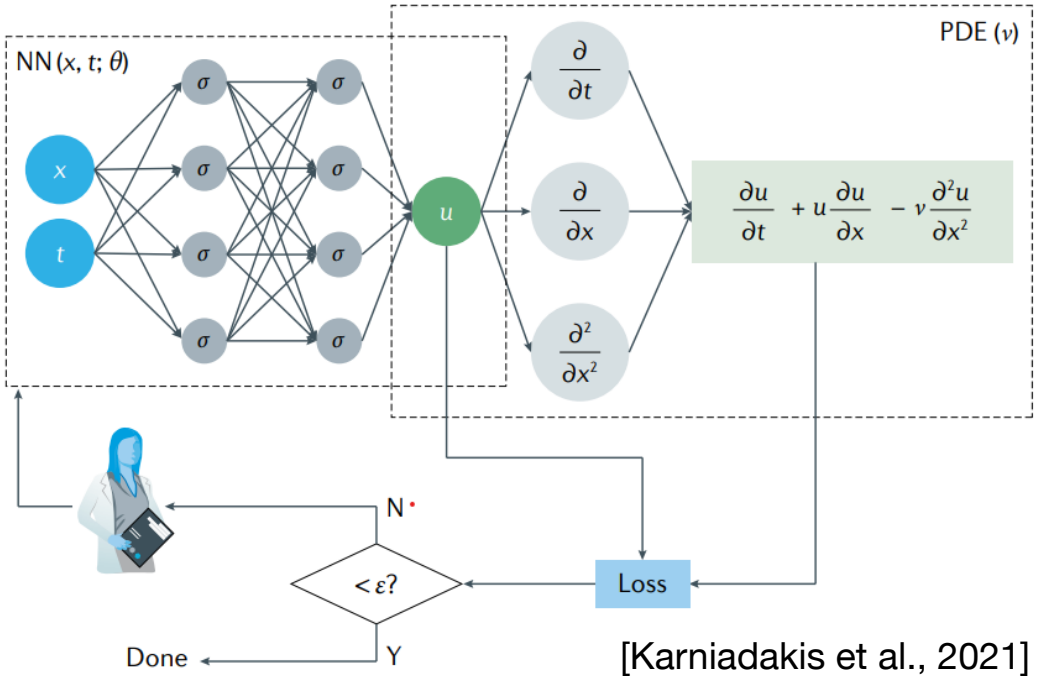


PDE solvers and SciML

Aim: Approximate the solution to a given PDE by a neural network

George Karniadakis

Physics-informed neural networks (PINNs)



[Karniadakis et al., 2021]

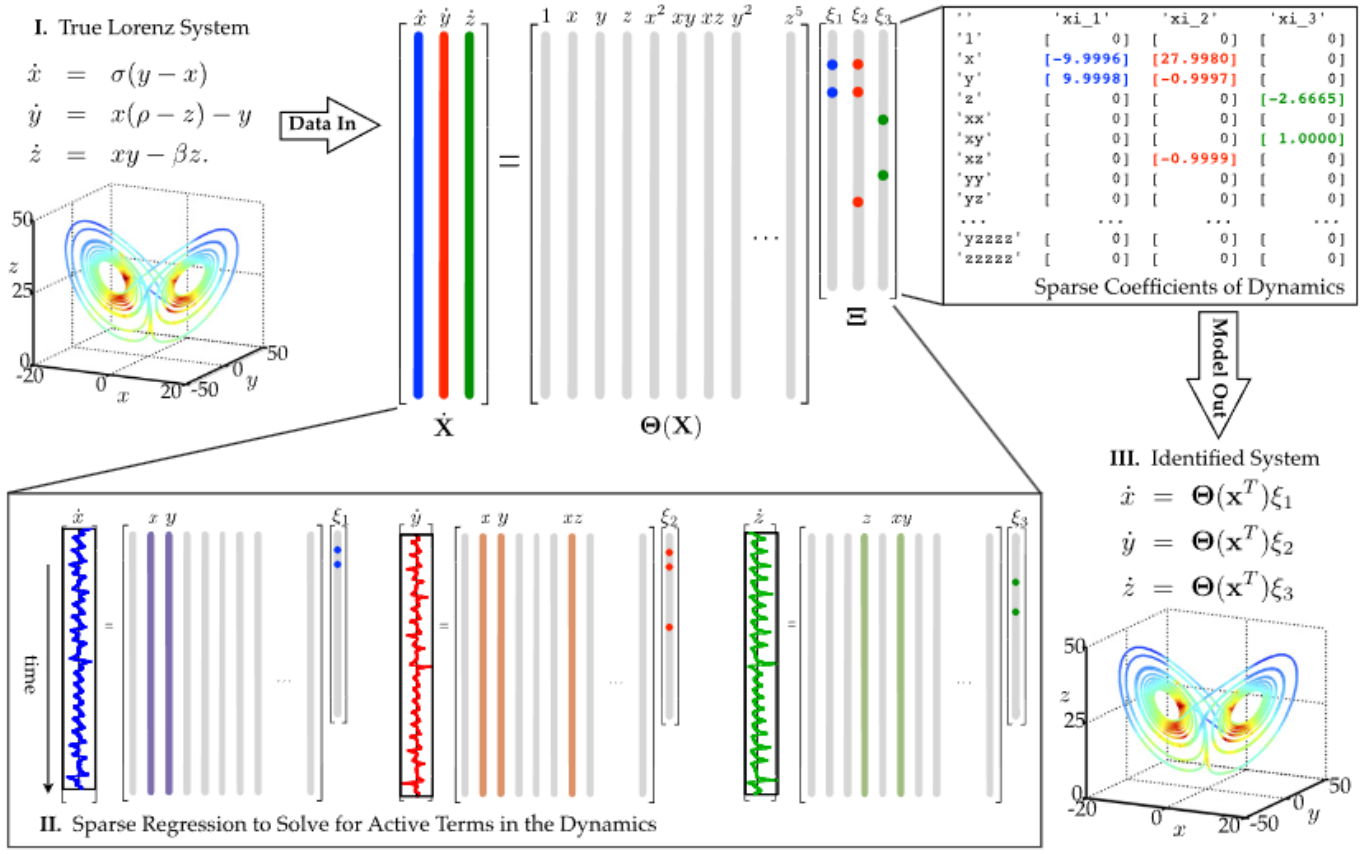
Advantages

- Easy to implement
- Can be embedded into downstream applications
- Works better for high-dimensional problems?

Related works: [Han et al, 2018], [Raissi et al., 2018], [Sirignano, Spiliopoulos, 2018], [Karniadakis et al., 2021], [Lu et al., 2021], [Cai et al., 2022]

PDE Discovery and SciML

Sparse Identification of Nonlinear Dynamics (SINDy)



[Brunton et al, 2016]

Steven Brunton

Nathan Kutz

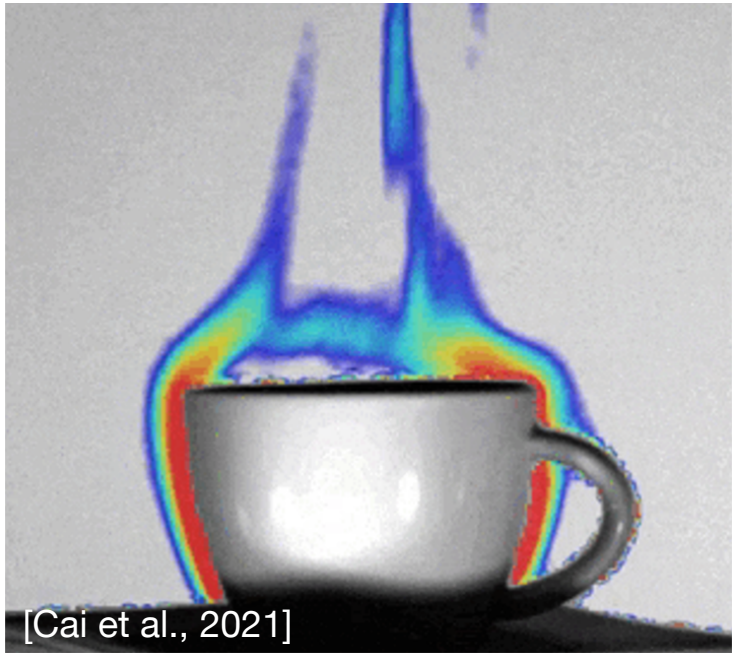
Related works: [Brunto et al., 2016], [Rudy et al., 2017], [Champion et al, 2019], [Udrescu, Tegmark, 2020], [Udrescu et al., 2020], [Liu, Tegmark, 2021]

Operator learning

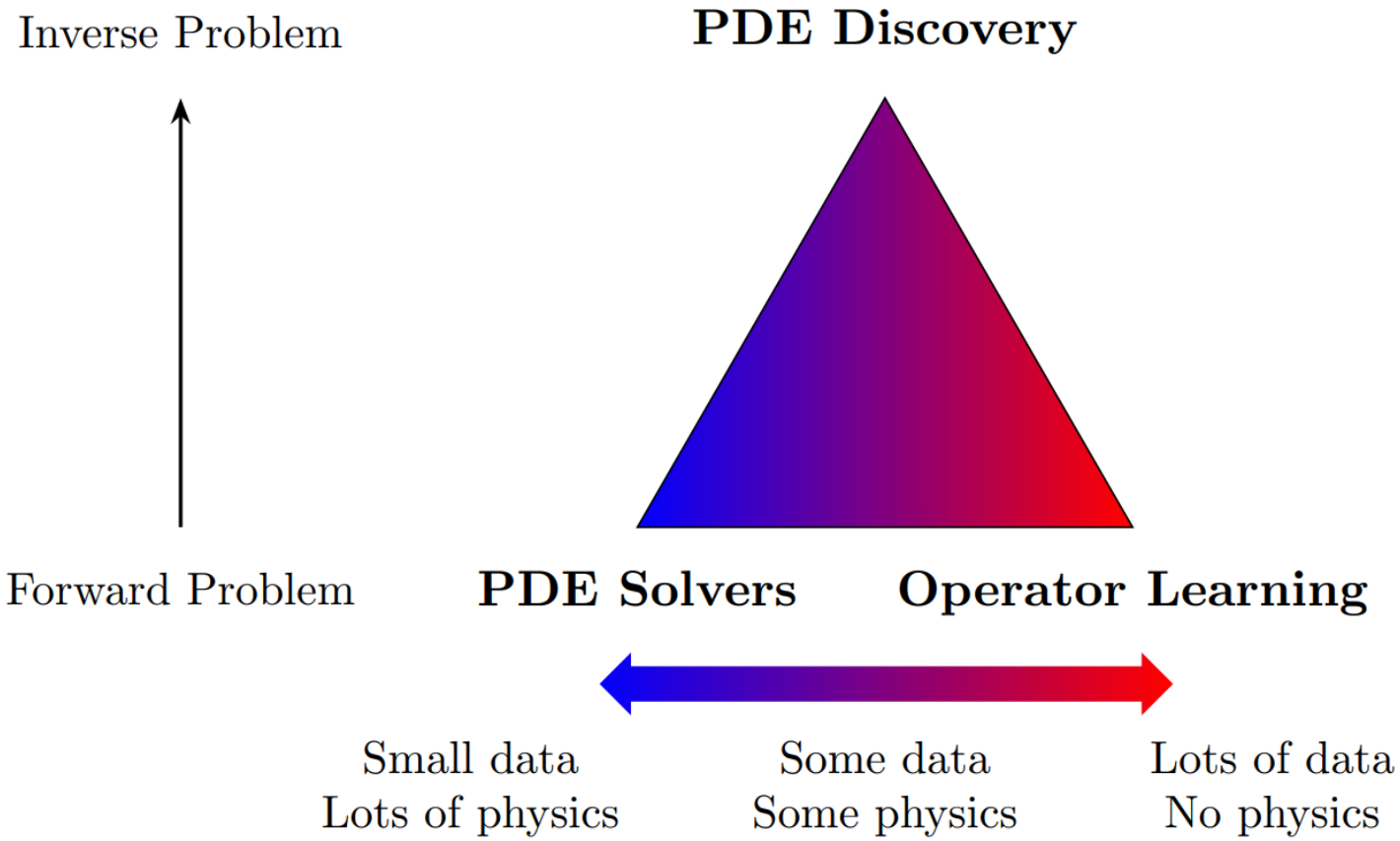
Physics-informed machine learning

George Em Karniadakis[1,2✉], *Ioannis G. Kevrekidis*[3,4], *Lu Lu*[5], *Paris Perdikaris*[6], *Sifan Wang*[7] *and Liu Yang*[1]

Abstract | Despite great progress in simulating multiphysics problems using the numerical discretization of partial differential equations (PDEs), one still cannot seamlessly incorporate noisy data into existing algorithms, mesh generation remains complex, and high-dimensional problems governed by parameterized PDEs cannot be tackled. Moreover, solving inverse problems with hidden physics is often prohibitively expensive and requires different formulations and elaborate computer codes. Machine learning has emerged as a promising alternative, but training deep neural networks requires big data, not always available for scientific problems. Instead, such networks can be trained from additional information obtained by enforcing the physical laws (for example, at random points in the continuous space-time domain). Such physics-informed learning integrates (noisy) data and mathematical models, and implements them through neural networks or other kernel-based regression networks. Moreover, it may be possible to design specialized network architectures that automatically satisfy some of the physical invariants for better accuracy, faster training and improved generalization. Here, we review some of the prevailing trends in embedding physics into machine learning, present some of the current capabilities and limitations and discuss diverse applications of physics-informed learning both for forward and inverse problems, including discovering hidden physics and tackling high-dimensional problems.



[Cai et al., 2021]

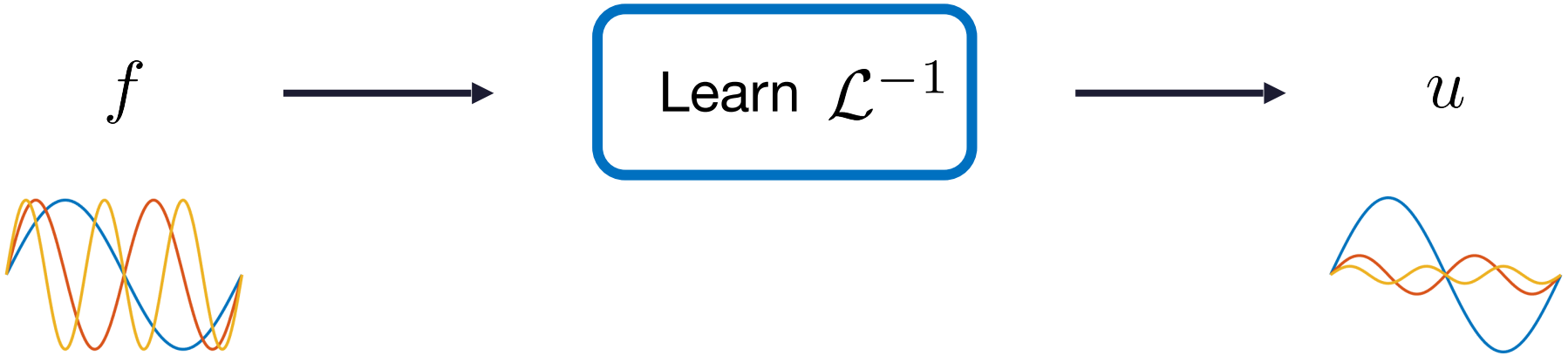


Recent surveys:

- B., Townsend, “A Mathematical Guide to Operator Learning”, 2023.
- Kovachki, Lanthaler, Stuart, “Operator Learning: Algorithms and Analysis”, 2024.

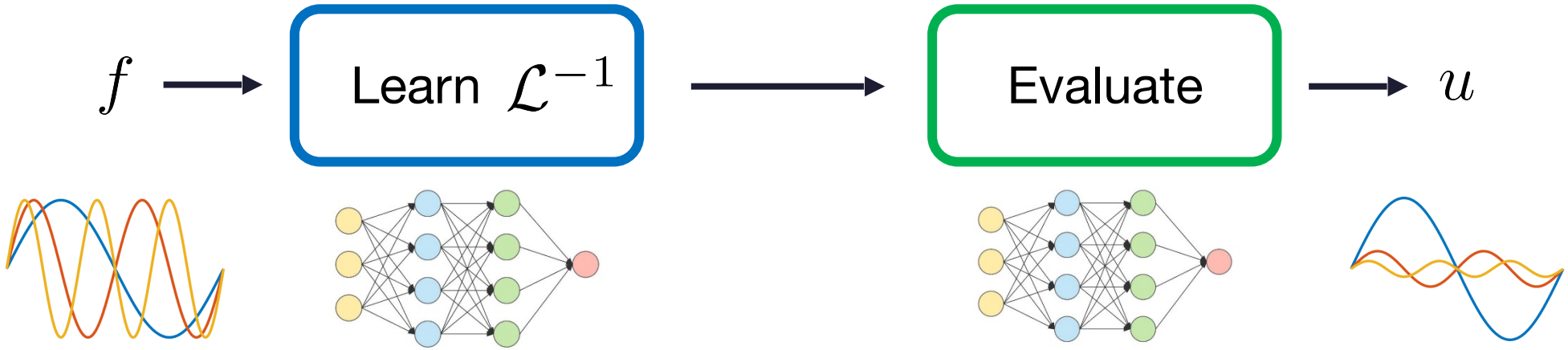
Introduction to operator learning

Aim: Approximate solution operators of unknown PDEs $\mathcal{L}(u) = f$



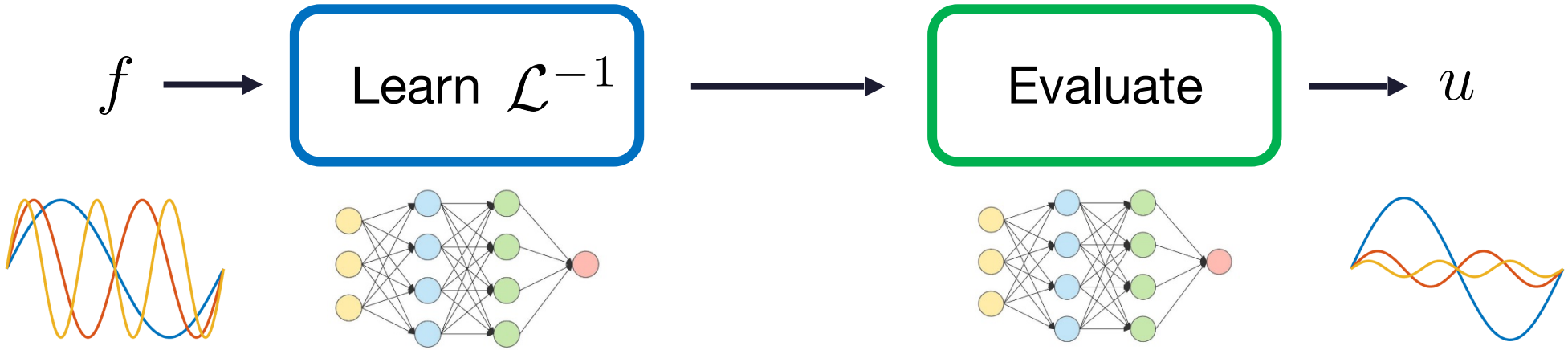
Introduction to operator learning

Aim: Approximate solution operators of unknown PDEs $\mathcal{L}(u) = f$

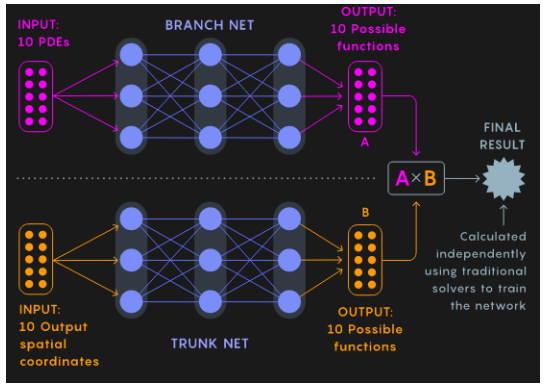


Introduction to operator learning

Aim: Approximate solution operators of unknown PDEs $\mathcal{L}(u) = f$

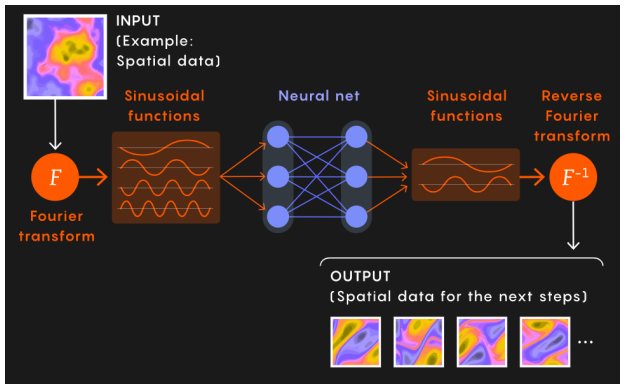


DeepONet



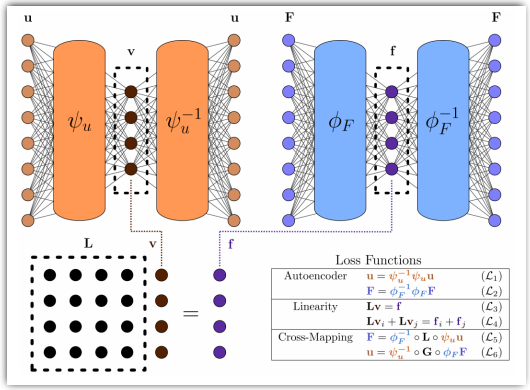
[Quanta Magazine; Lu et al, 2021]

Fourier Neural Operator



[Quanta Magazine; Li et al, 2020]

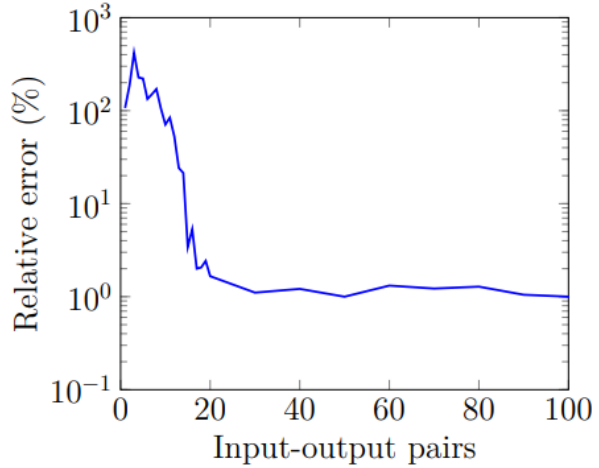
DeepGreen



[Gin et al., 2020]

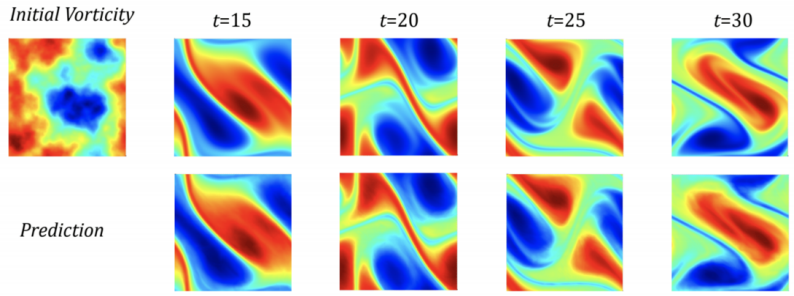
Main challenges in operator learning

1. Theoretical results



- Type and number of training data
- Performance guarantees
- Neural network architectures
- Noise robustness

2. Interpretability of the model



[Li et al, 2020]

- Dominant modes
- Symmetries
- Conservation laws
- Singularities

Operator learning and approximation theory

Density result for neural operators:

Theorem 1 (Universal Approximation Theorem for Operator).
Suppose that σ is a continuous non-polynomial function, X is a Banach space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p and m, constants c_i^k, ξ_{ij}^k, θ_i^k, $\zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$ and $j = 1, \ldots, m$, such that

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{\text{branch}} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{\text{trunk}} \right| < \epsilon$$

[Chen, Chen, 1995], [Lu et al., 2021]

Neural network results: [Cybenko, 1989], [Hornik et al., 1989], [Hornik, 1991], [Maioro, Pinkus, 1999], [Lu et al., 2017], [Kidger et al., 2020]

Operator learning and approximation theory

Entropy / asymptotic result for neural operators:

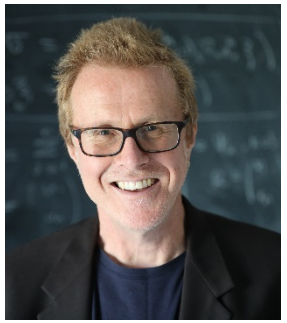
***Theorem* 2.11** (Curse of Parametric Complexity)**.** Let $K \subset \mathcal{X}$ be a compact set in an infinite-dimensional Banach space $\mathcal{X}$. Assume that K contains an infinite-dimensional hypercube Q_α for some $\alpha > 1$. Then for any $r \in \mathbb{N}$ and $\delta > 0$, there exists $\overline{\epsilon} > 0$ and an r-times Fréchet differentiable functional $\mathcal{S}^\dagger : K \subset \mathcal{X} \to \mathbb{R}$, such that approximation to accuracy $\epsilon \leq \overline{\epsilon}$ by a functional $\mathcal{S}_\epsilon$ of neural network-type,

$$\sup_{u \in K} |\mathcal{S}^\dagger(u) - \mathcal{S}_\epsilon(u)| \leq \epsilon, \tag{2.10}$$

implies complexity bound $\mathrm{cmplx}(\mathcal{S}_\epsilon) \geq \exp(c\epsilon^{-1/(\alpha+1+\delta)r})$; here c, $\overline{\epsilon} > 0$ are constants depending only on α, δ and r.

[Lanthaler, Stuart, 2024] ◊

Samuel Lanthaler



Andrew Stuart

Recent surveys:

- DeVore, Hanin, Petrova, “Neural network approximation”, 2021.
- Kovachki, Lanthaler, Stuart, “Operator Learning: Algorithms and Analysis”, 2024.

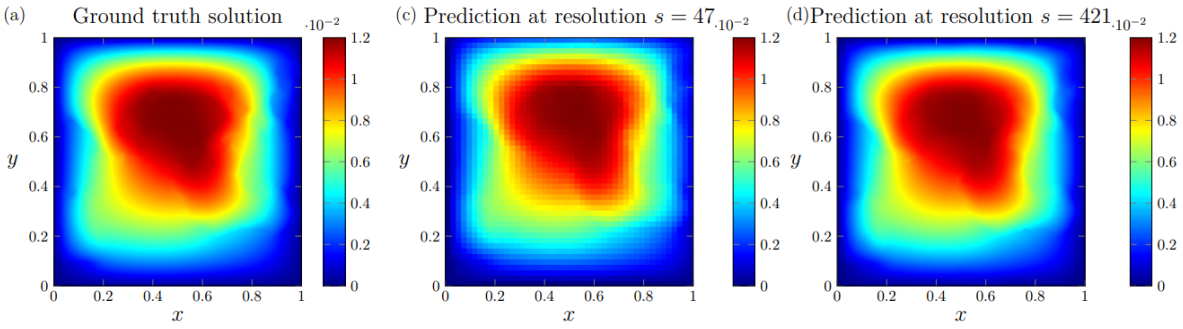
Related works: [Telgarsky, 2016], [Yarotsky, 2017], [Lanthaler et al., 2022], [Kovachki et al., 2023], [Lanthaler et al., 2023], [Lanthaler, Stuart, 2024]

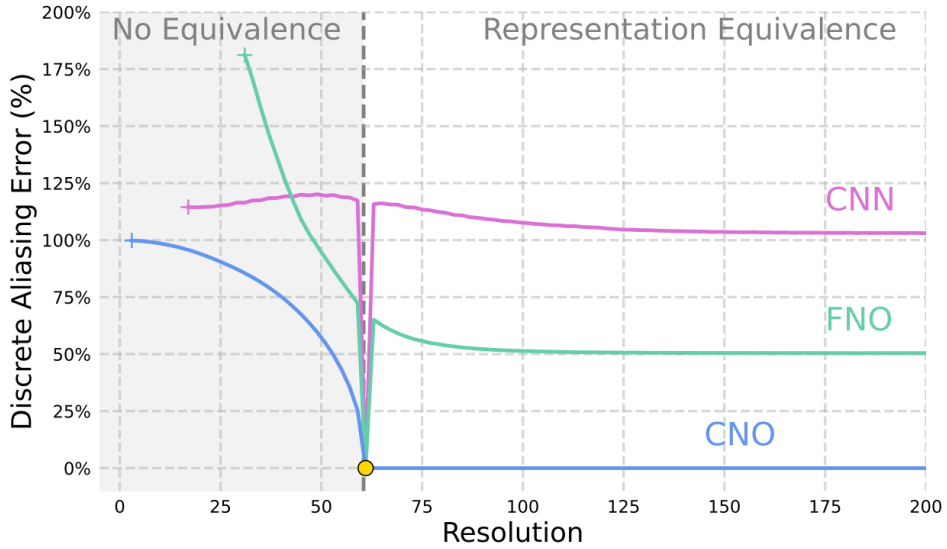
Operator learning and discretization errors

Sid Mishra

Resolution invariance / super-resolution:

Neural operators can be trained at low-resolution and evaluated at high-resolution



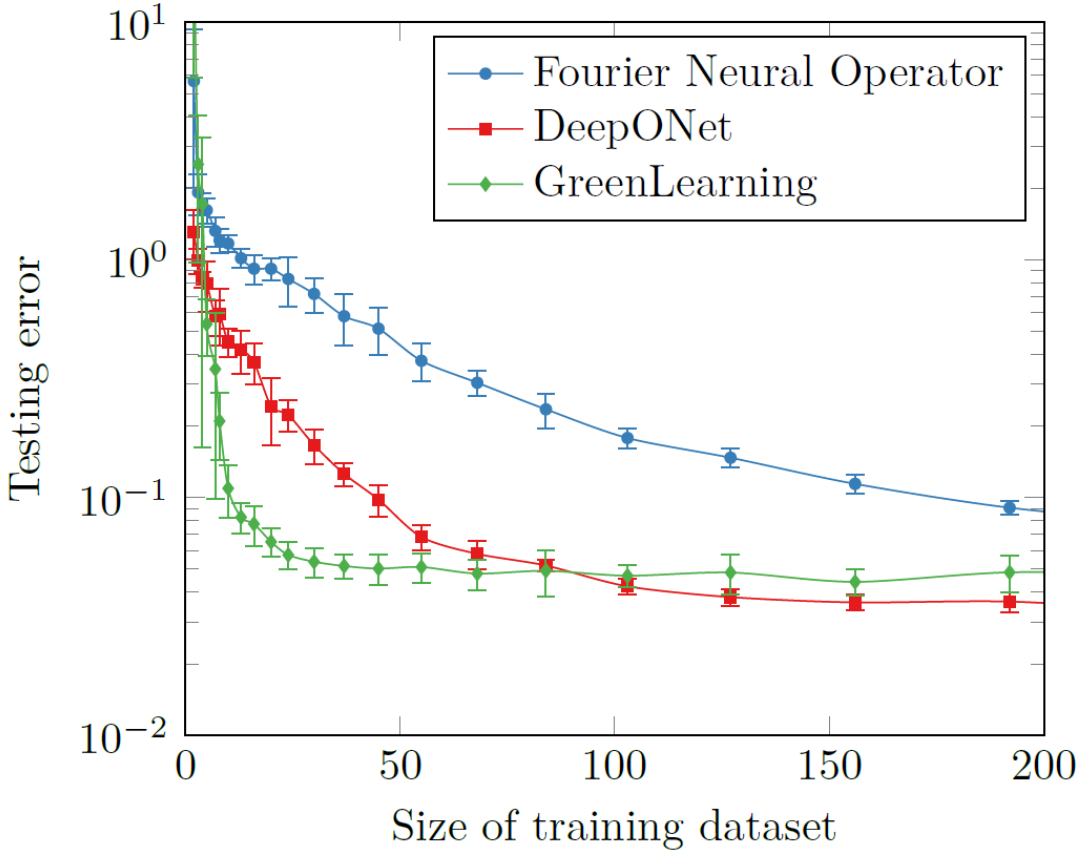


[Bartolucci et al., 2023]

Related works: [Li et al., 2020], [Bartolucci et al., 2023], [Kovachki et al., 2023], [Raonic et al., 2023]

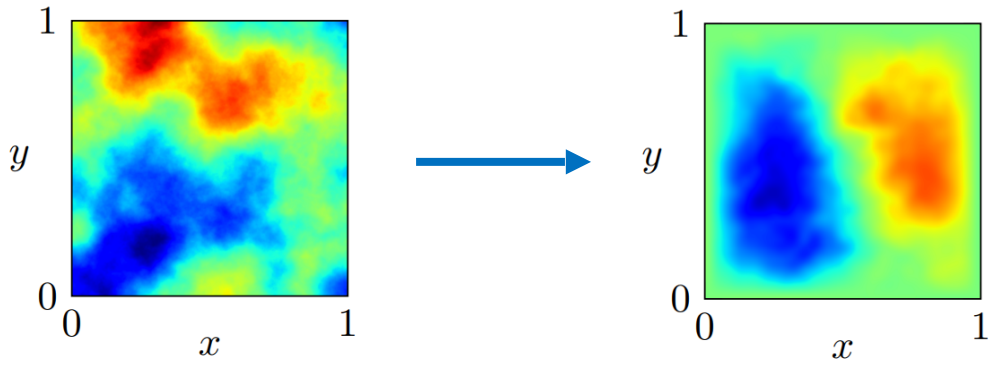
Operator learning and sample complexity

Learn the solution operator of Poisson equation



Random source

Solution



How much training data is needed to learn a PDE?

Related works: [B., Townsend, 2022], [B. et al., 2022], [Chen et al., 2023], [de Hoop et al., 2021], [Lu et al., 2021], [Schäfer, Owhadi, 2021], [Schäfer et al., 2017]

Operator learning in practice

We want to learn the solution operator associated with a PDE: $L(u) = f$

A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

Operator learning in practice

We want to learn the solution operator associated with a PDE: $L(u) = f$

A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

1. Generate random source terms f

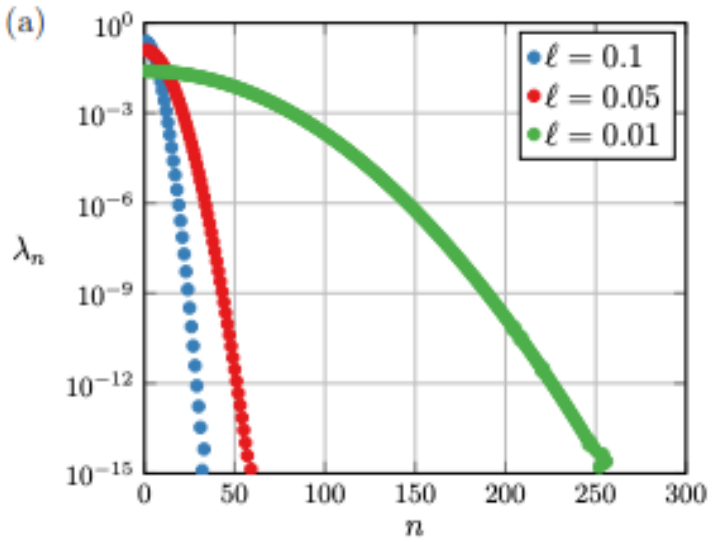
Select covariance kernel K

$$K(x,y) = \sum_{j=1}^{\infty} \lambda_j \psi_j(x) \psi_j(y)$$

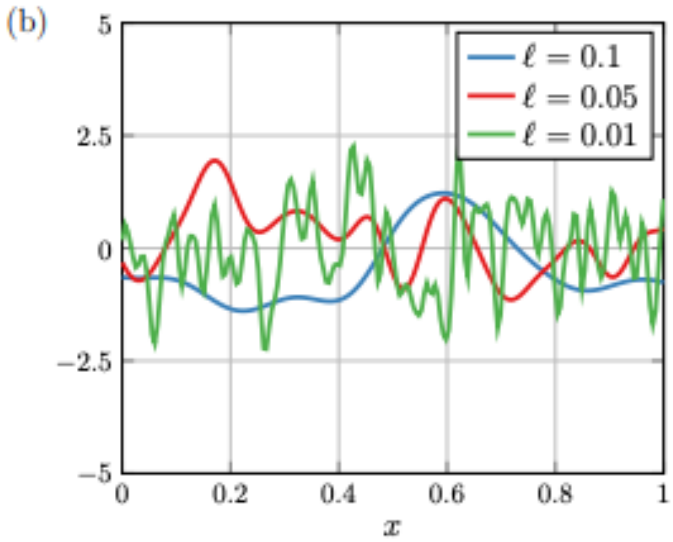
Sample f as

$$f(x) = \sum_{j=1}^{\infty} c_j \sqrt{(\lambda_j)} \psi_j(x), \quad c_j \sim \mathcal{N}(0,1)$$

λ_n



$f(x)$



Operator learning in practice

We want to learn the solution operator associated with a PDE: $L(u) = f$

A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

2. Compute solutions u

Either from a physical experiment or by numerically solving the PDE (e.g. with Firedrake)

Property	Finite differences	Finite elements	Spectral methods
Domain geometry	Simple	Complex	Simple
Approximation	Local	Local	Global
Linear system	Large sparse	Large sparse	Small dense
Convergence rate	Algebraic	Algebraic	Spectral

Operator learning in practice

We want to learn the solution operator associated with a PDE: $L(u) = f$

A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

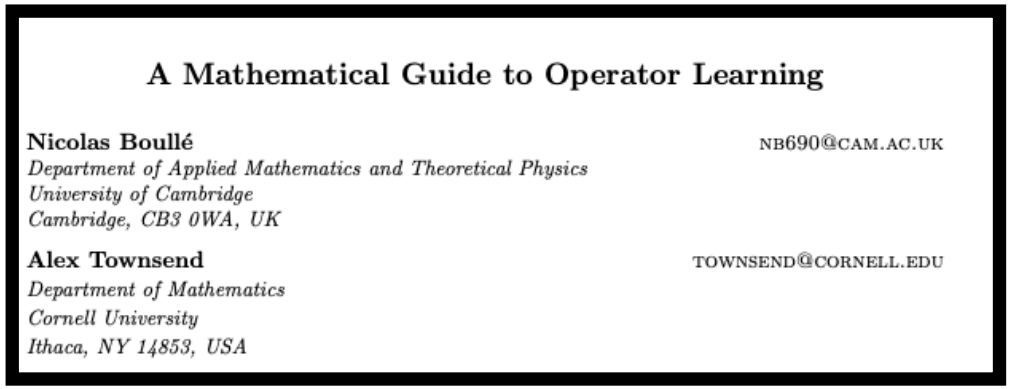
3. Define the neural network architecture

Depends on the data structure, problem, …

Neural operators	Property of the operator	Kernel parameterization
DeepONet	Low-rank	Branch and trunk networks
FNO	Translation-invariant	Fourier coefficients
GreenLearning	Linear	Rational neural network
DeepGreen	Semi-linear	Kernel matrix
Graph neural operator	Diagonally dominant	Message passing network
Multipole GNO	Off-diagonal low rank	Neural network

Operator learning in practice

We want to learn the solution operator associated with a PDE: $L(u) = f$



A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

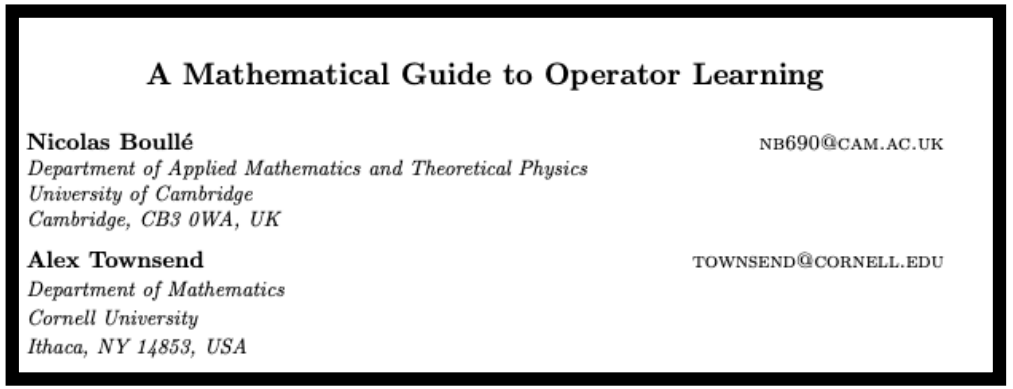
4. Define the optimisation problem

$$\min_{\theta \in \mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u) \in \text{data}} \|\mathcal{N}(f) - u\|_U$$

The norm is often computed by Monte-Carlo integration by sampling the solution at random points:

$$\min_{\theta \in \mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u) \in \text{data}} \frac{1}{n} \sum_{j=1}^{n} |\mathcal{N}(f)(y_j) - u(y_j)|^2$$

Operator learning in practice



A Mathematical Guide to Operator Learning

Nicolas Boullé NB690@CAM.AC.UK
Department of Applied Mathematics and Theoretical Physics
University of Cambridge
Cambridge, CB3 0WA, UK

Alex Townsend TOWNSEND@CORNELL.EDU
Department of Mathematics
Cornell University
Ithaca, NY 14853, USA

We want to learn the solution operator associated with a PDE: $L(u) = f$

4. Define the optimisation problem

$$\min_{\theta \in \mathbb{R}^N} \frac{1}{|\text{data}|} \sum_{(f,u) \in \text{data}} \|\mathcal{N}(f) - u\|_U$$

This is called a data-driven loss as it trains the network to satisfies the data-relation.

Physics-informed operator learning

We want to learn the solution operator associated with a PDE: $L(u) = f$

APPLIED PHYSICS

Learning the solution operator of parametric partial differential equations with physics-informed DeepONets

Sifan Wang[1], Hanwen Wang[1], Paris Perdikaris[2]*

Complement the data-driven loss with a physics-informed loss to weakly satisfy the PDE or some constraints

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{operator}}(\theta) + \mathcal{L}_{\text{physics}}(\theta) \tag{16}$$

where

$$\mathcal{L}_{\text{operator}}(\theta) = \frac{1}{NP}\sum_{i=1}^{N}\sum_{j=1}^{P} \left| G_\theta(\mathbf{u}^{(i)})(\mathbf{y}_{u,j}^{(i)}) - G(\mathbf{u}^{(i)})(\mathbf{y}_{u,j}^{(i)}) \right|^2 \tag{17}$$

Enforce data

$$\mathcal{L}_{\text{physics}}(\theta) = \frac{1}{NQm}\sum_{i=1}^{N}\sum_{j=1}^{Q}\sum_{k=1}^{m} \left| \mathcal{N}(u^{(i)}(\mathbf{x}_k), G_\theta(\mathbf{u}^{(i)})(\mathbf{y}_{r,j}^{(i)})) \right|^2 \tag{18}$$

Enforce PDE constraint

Coupling PyTorch and Firedrake

We try to minimise the loss function:

$$\mathscr{L}(\theta) = \underbrace{\|\mathscr{N}_\theta(f) - u\|}_{\text{data}} + \alpha \underbrace{F(\mathscr{N}_\theta(f), f)}_{\text{PDE residual}}$$

Differentiable programming across the PDE and Machine Learning barrier

Nacime Bouziani
I-X Centre for AI In Science,
Imperial College London, UK
n.bouziani18@imperial.ac.uk

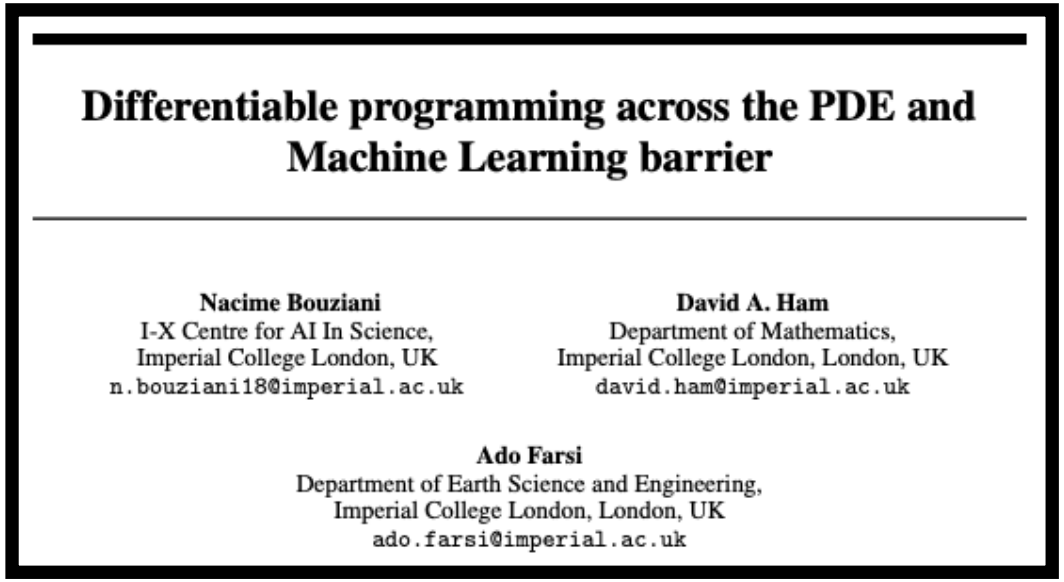
David A. Ham
Department of Mathematics,
Imperial College London, London, UK
david.ham@imperial.ac.uk

Ado Farsi
Department of Earth Science and Engineering,
Imperial College London, London, UK
ado.farsi@imperial.ac.uk

Coupling PyTorch and Firedrake

We try to minimise the loss function:

$$\mathscr{L}(\theta) = \underbrace{\|\mathscr{N}_\theta(f) - u\|}_{\text{data}} + \alpha \underbrace{F(\mathscr{N}_\theta(f), f)}_{\text{PDE residual}}$$



Differentiable programming across the PDE and Machine Learning barrier

Nacime Bouziani
I-X Centre for AI In Science,
Imperial College London, UK
n.bouziani18@imperial.ac.uk

David A. Ham
Department of Mathematics,
Imperial College London, London, UK
david.ham@imperial.ac.uk

Ado Farsi
Department of Earth Science and Engineering,
Imperial College London, London, UK
ado.farsi@imperial.ac.uk

We want to discrete the residual with finite element and compute it using Firedrake

Since we use gradient-based optimisation, we need to compute the gradient of the residual with respect to θ:

$$\nabla_\theta F(\mathscr{N}_\theta(f), f)$$

Coupling PyTorch and Firedrake

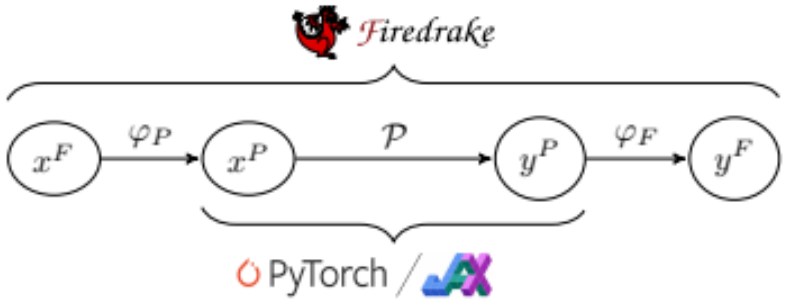


Figure 1: Subgraph of the Firedrake computational graph containing PyTorch/JAX operations of interest represented by $\mathcal{P}$, where P refers to PyTorch/JAX variables and F to Firedrake variables. φ_F and φ_P represent the casting of a PyTorch/JAX tensor to a Firedrake Function and vice versa.

Differentiable programming across the PDE and Machine Learning barrier

Nacime Bouziani
I-X Centre for AI In Science,
Imperial College London, UK
n.bouziani18@imperial.ac.uk

David A. Ham
Department of Mathematics,
Imperial College London, London, UK
david.ham@imperial.ac.uk

Ado Farsi
Department of Earth Science and Engineering,
Imperial College London, London, UK
ado.farsi@imperial.ac.uk

We do the optimisation of the networks in PyTorch and ask Firedrake to compute the gradient with pyadjoint

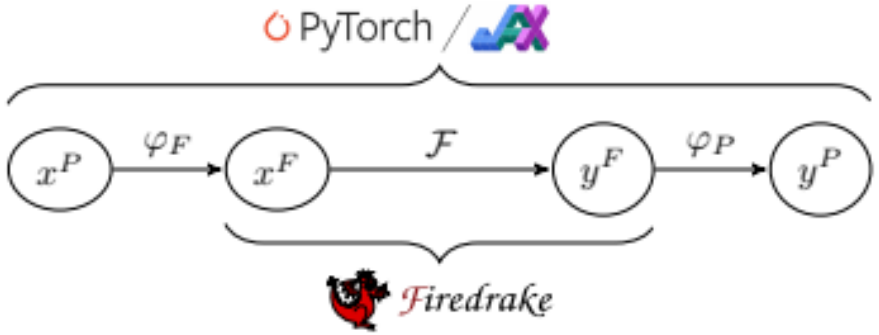
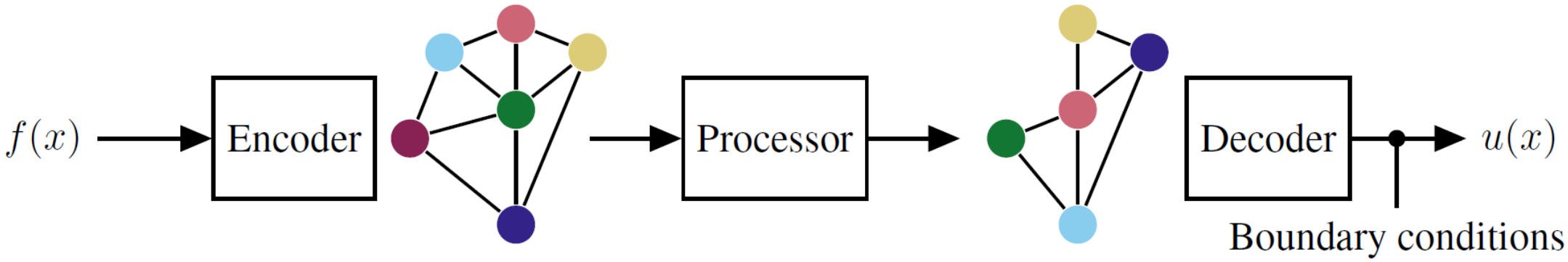


Figure 2: Subgraph of the PyTorch/JAX computational graph containing Firedrake operations of interest represented by $\mathcal{F}$, where P refers to PyTorch/JAX variables and F to Firedrake variables. φ_F and φ_P represent the casting of a PyTorch/JAX tensor to a Firedrake Function and vice versa.

Coupling PyTorch and Firedrake

```
1  import torch
2  import firedrake as fd
3  import firedrake.ml as fd_ml
4  import firedrake_adjoint as fda
5
6  ...
7
8  # Defined reduced functional ℱ with respect to control(s)
9  F = fda.ReducedFunctional(y_F, Control(x_F))
10
11 # Define the coupling operator: G := φ_P ∘ ℱ ∘ φ_F
12 G = fd_ml.torch_operator(F)
13
14 # Apply the coupling operator to a torch.Tensor x^P
15 y_P = G(x_P)
16
17 # Backpropagate through G: calculate J*_{y^P}(x^P, w^P)
18 w_P = ...
19 y_P.backward(w_P)
```

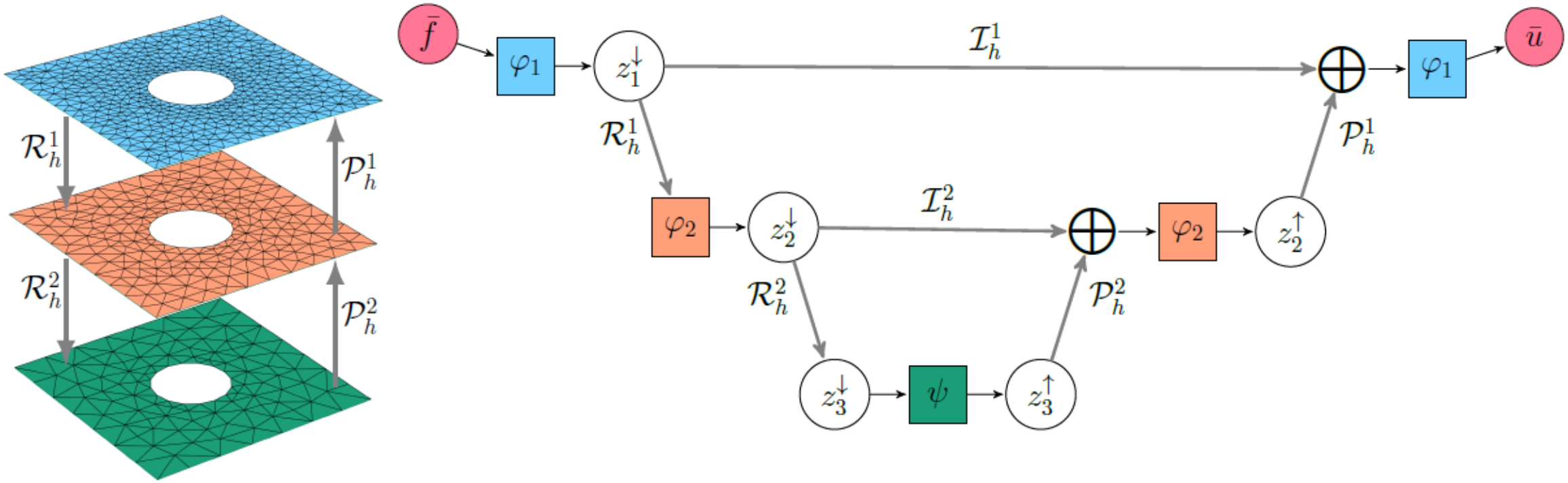
Structure-preserving operator learning



- Finite element encoder / decoder
- Structure-preserving discretization
- Finite element error estimation
- Flexible neural network architecture

See: https://arxiv.org/abs/2410.01065

Multigrid-inspired processor



- Finite element interpolation and projection operators
- Message-passing Graph Neural Networks

Tutorial

Solution operator of Stokes flow