Caltech

# Partial Differential Equations and Neural Operators

Zongyi Li
May 2022

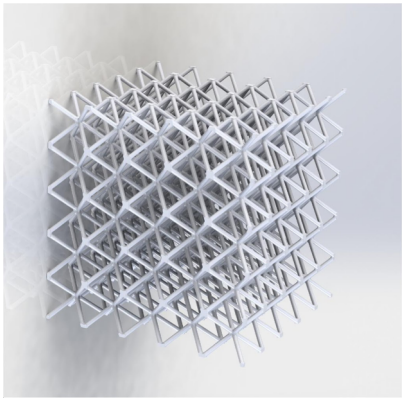
CS159: Representation Learning for Science
Prof. Yisong Yue

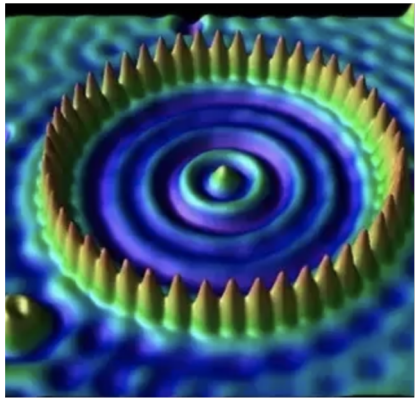
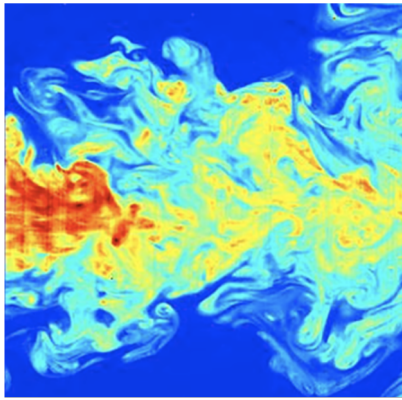
# Overview

- Partial differential equations (20 min)
  - Examples
  - Numerical solvers
  - PINN
- Operator learning (30 min)
  - Neural operator
  - DeepONet
  - Fourier neural operator
- Examples and extensions (30 min)
  - Weather forecast
  - And more

# 1. Introduction

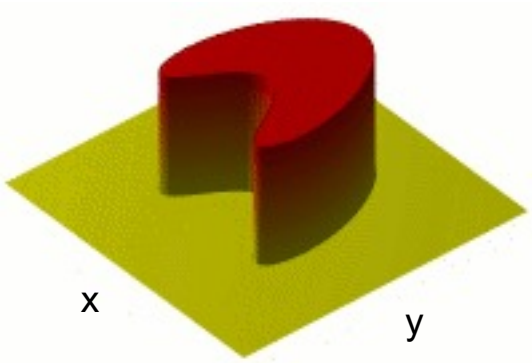
Problems in science and engineering reduce to PDEs.



# Example: heat equation

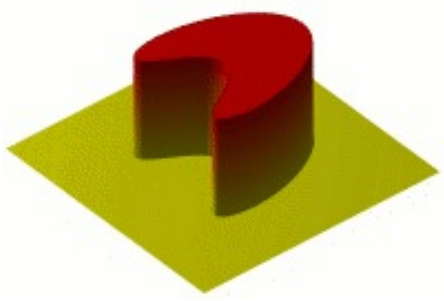
2D parabolic PDE




How long will my pan cool down?

Figure: Oleg Alexandrov

# Example: heat equation



Spatial domain: $\mathrm{D} = [0,1]\times[0,1]$

Time domain: $\mathrm{T} = [0,1]$

Points: $x = (x, y) = (x_1, x_2) \in D, \quad \mathrm{t} \in T$

Solution function (Temperature) $u: T\times D \to \mathbb{R}$

Heat equation:

$$u_t = \Delta u$$

Where $\Delta u = u_{xx} + u_{yy}$

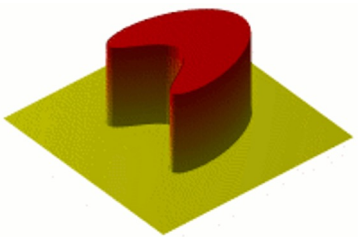
“The change in time is equal to 2nd-order difference in space.”

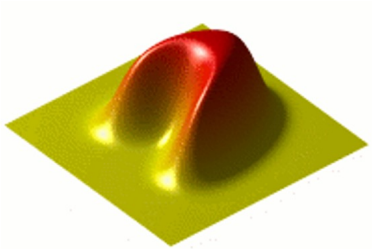
# Initial value problems

Heat equation:

$$u_t = \Delta u$$

Initial value problem: given the temperature at time t=0,
What is the temperature a time t=1?
Given u(0,x), what is u(1,x)?





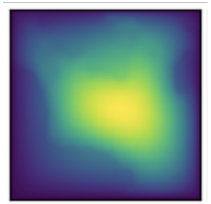
# Problems with coefficients

2D elliptic PDE:

$$-\nabla \cdot (a(x)\nabla u(x)) = f(x), \quad x \in D$$

$\nabla u = (u_x, u_y)$
$\Delta u = \nabla \cdot \nabla u$

Input: a(x)

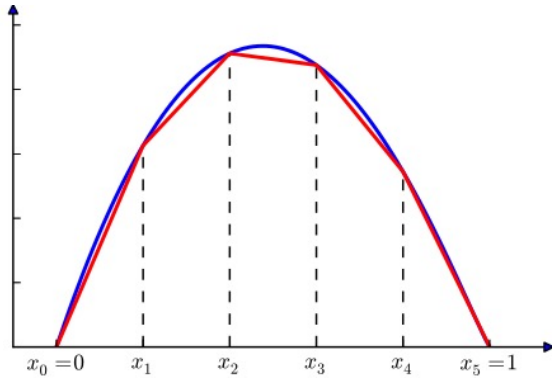


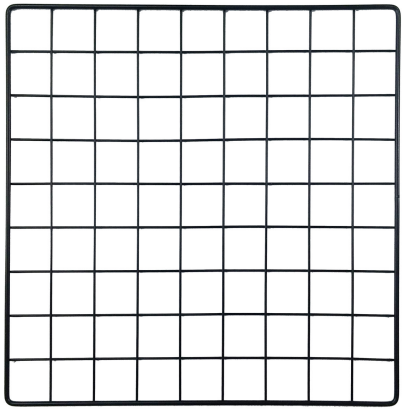
Output: u(x)

My domain has two types of media, described by a(x)
Given a(x), what is u(x)?

# Numerical solver

Idea: discretize the problem onto some grid
Solve a multi-variable equation of $\{u(i,j)\}$, $i = 0, \dots, 10, j = 0, \dots, 10$
Solve a linear system Au = f



# Numerical solver

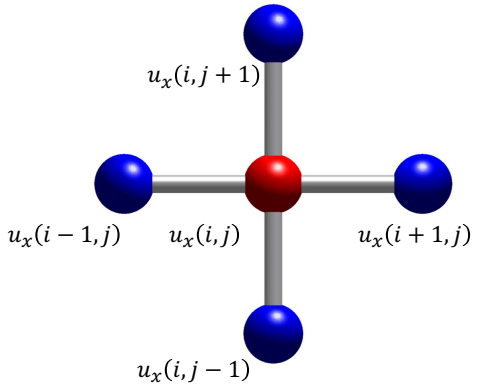
Idea: discretize the problem onto some grid
Solve a multi-variable equation of $\{u(i,j)\}$, $i = 0, \dots, 10, j = 0, \dots, 10$

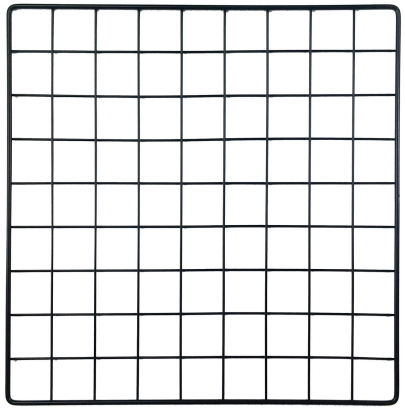
How to compute the derivative?
Finite difference method (FDM): approximate by its neighbor points

$$u_x(x) = \lim_{dx \to 0} \frac{[u(x+dx) - u(x)]}{dx}$$

$$u_x(x) \sim \frac{[u(x+dx) - u(x)]}{dx}$$

$$u_x(i,j) \sim \frac{[u(i+1,j) - u(i,j)]}{dx}$$



# Numerical solver

Pipeline:

1. Discretize the space
2. Write out a linear system
3. Solve the linear system

Common numerical methods:

- Finite difference methods
- Finite element methods
- Spectral methods
- Iterative methods

Trade-off: finer grids are more accurate, but also more expensive.

# Physics-informed neural networks

Heat equation:

$$u_t = \Delta u$$

Physics-informed neural networks (PINNs)

- Idea: represent function $u$ as a neural network
- Compute the derivatives $(u_t, \Delta u)$ using the chain rule
- Solve the system using gradient descent methods

PINN: M Raissi, P Perdikaris, GE Karniadakis

# Recap: PINNs

Heat equation:

$$u_t = \Delta u$$

Pipeline:

- Initialize NN u(x)
- For N iterations
- Sample collocation points $\{(x, t)\}$
- Compute the derivatives $u_t(x, t),\ \Delta u(x, t)$
- Minimize $\| u_t - \Delta u \|$

PINN: M Raissi, P Perdikaris, GE Karniadakis

# Recap: PINNs

Pros:

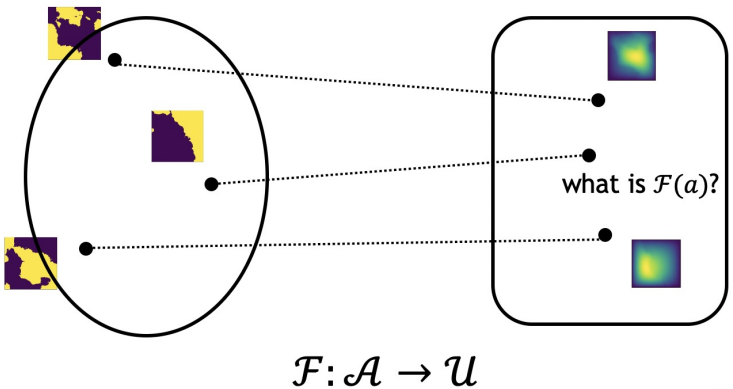
- Simple and flexible
- No need to worry about stability conditions, etc
- Complex geometry, high-dim, inverse problem, etc

Cons:

- Optimization is tricky
- Less accurate than existing solvers (FDM/FEM)
- No guarantee of (optimization)

PINN: M Raissi, P Perdikaris, GE Karniadakis

# 2. Operator learning

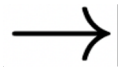


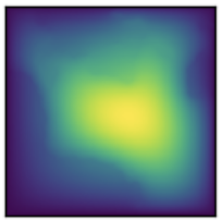
# Operator learning

Operators are map between function space.
Given a dataset of input-output pairs, find the map (operator)

F: a->u

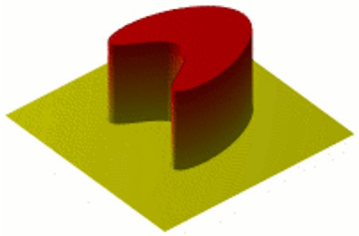
Input: coefficient



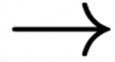


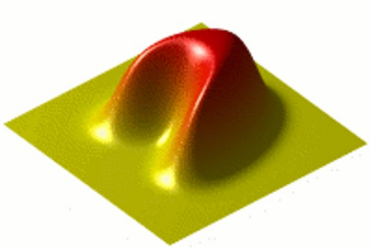
Output: solution

F: u0->u1



Input: initial



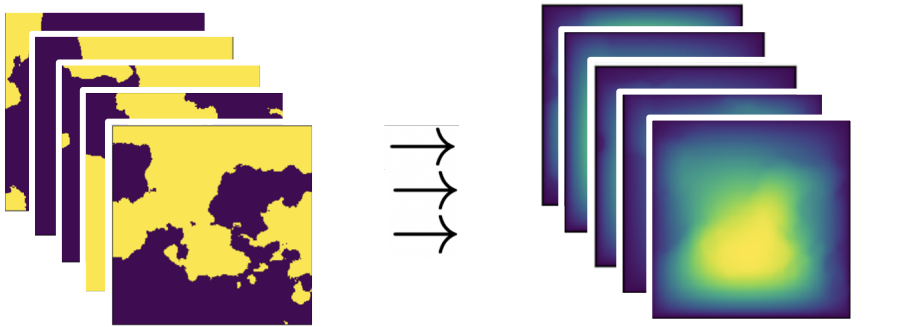


Output: solution

# Operator learning

Operators are map between function space.
Learn the Operators from data (coefficients & solutions pairs).

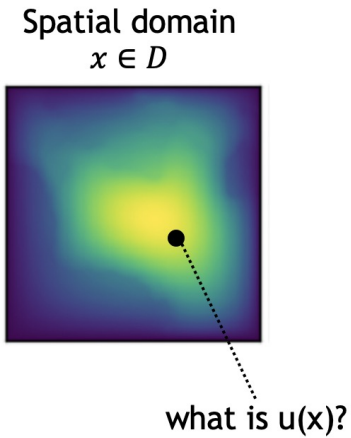
- Fix an equation
- Multiple training instances
- Learn the mapping



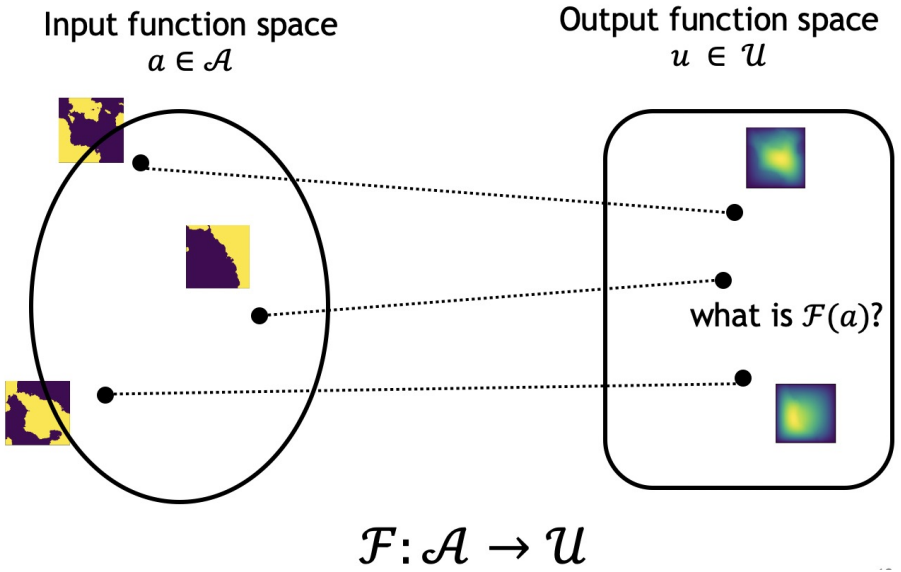

$$\mathcal{F} : \mathcal{A} \times \Theta \to \mathcal{U}$$

# Solve vs learn

Solving for a PDE instance $u$
approximate $u(x)$ in the spatial space.




learn the solution operator $\mathcal{F}$
interpolate $u$ in the function space.



# Pipeline of operator learning

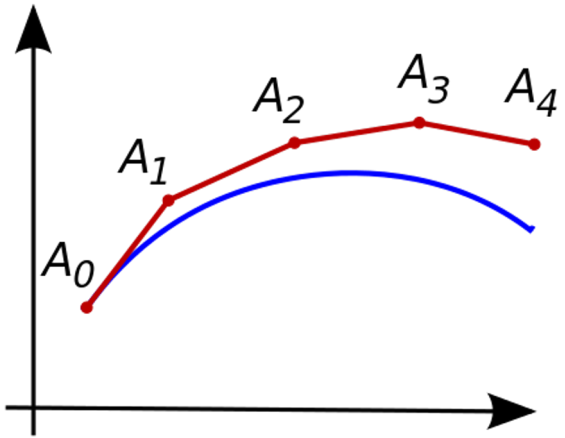
Supervised learning

Get a dataset {(a , u)} (existing solvers or experiments)

- Initialize NN, F: a->u
- For N epoch, For batches of data pairs (a, u)
- Compute the prediction F(a)
- Minimize ||F(a)-u||

# Solve vs learn

Conventional methods:
Solve the equation
By approximation on a mesh




Data-driven methods:
Learn the trajectory
From a distribution

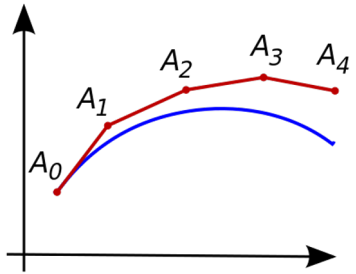
Input: coefficients

Output: solutions

# Solve vs learn

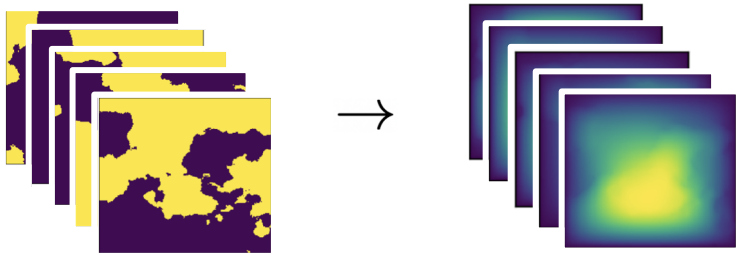
## Conventional methods:

- Solve for any parameters
- Worst case guarantees
- Consistency




## Data-driven methods:

- Parameters from a distribution
- Less guaranteed
- Not “consistent”

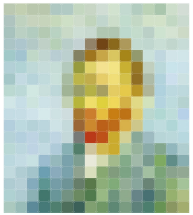


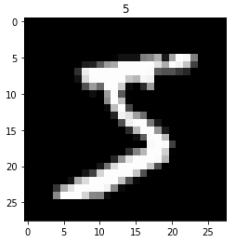
Input: coefficients

Output: solutions

# Architecture design

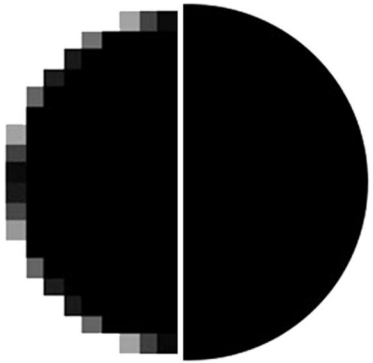
- Not vector-to-vector mapping.
- But function-to-function mapping.

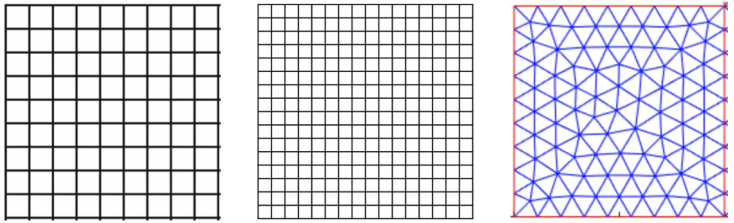


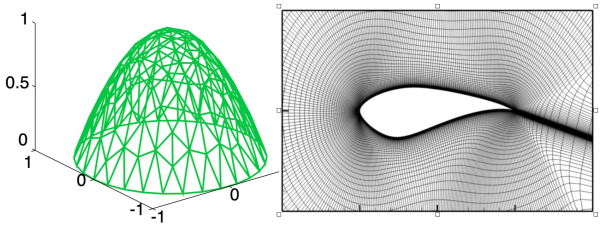



Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| man | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| woman | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| boy | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| girl | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| prince | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| princess | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| queen | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| king | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| monarch | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |







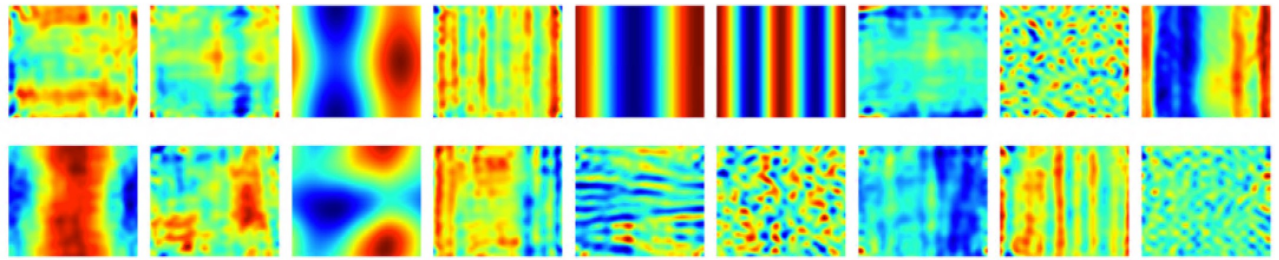

Discretized vector

Continuous function

# CNN vs Neural operators

Key idea: represent function & operator in mesh-invariant way

Filters in CNN



Fourier Filters

# 3.1 Neural operator

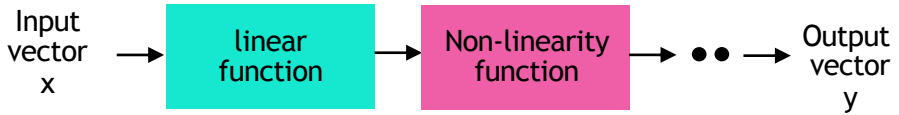
$$u = (K_l \circ \sigma_l \circ \cdots \circ \sigma_1 \circ K_0)\, v$$

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu,
Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar

# Neural networks

f: $x \to y$

$R^n \to R^m$

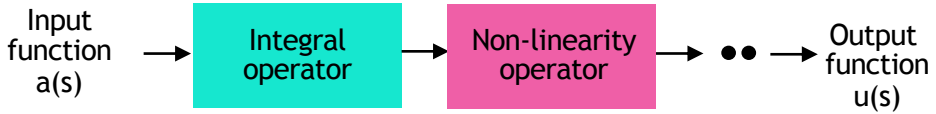



$$
\begin{aligned}
v_0 &= x \\
v_{l+1} &= \sigma\left(A_l v_l + b_l\right), \quad l = 0, \ldots, L-1 \\
y &= A_L v_L + b_L \\
\sigma : \mathbb{R} \to \mathbb{R}, \quad A_l \in \mathbb{R}^{d_{l+1} \times d_l}, \quad b_l &\in \mathbb{R}^{d_{l+1}}
\end{aligned}
$$

# Neural operators

F: a(s) → u(s)

$(R^n \to R^m) \to (R^n \to R^m)$




$$v_0(s) = P(x(s), s)$$

$$v_{l+1}(s) = \sigma\left(W_l v_l(s) + \int_D \kappa_l(s,z) v_l(z)\, dz + b_l(s)\right), \quad l = 0, \ldots, L-1$$

$$y(s) = Q(v_L(s))$$

$$P : \mathbb{R}^{m+n} \to \mathbb{R}^{d_0}, \quad W_l \in \mathbb{R}^{d_{l+1} \times d_l}, \quad \kappa_l : \mathbb{R}^{2n} \to \mathbb{R}^{d_{l+1} \times d_l}, \quad b_l : \mathbb{R}^n \to \mathbb{R}^{d_{l+1}}, \quad Q : \mathbb{R}^L \to \mathbb{R}^r$$

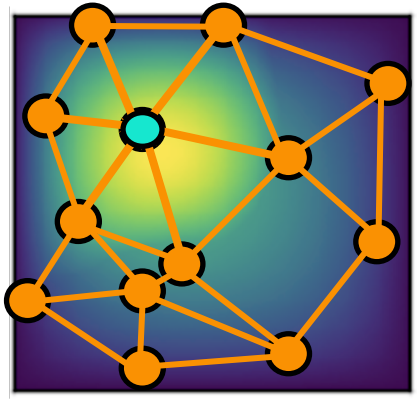
# Approximation bound

For any continuous operator, there exists a two-layers neural operators can approximate it.

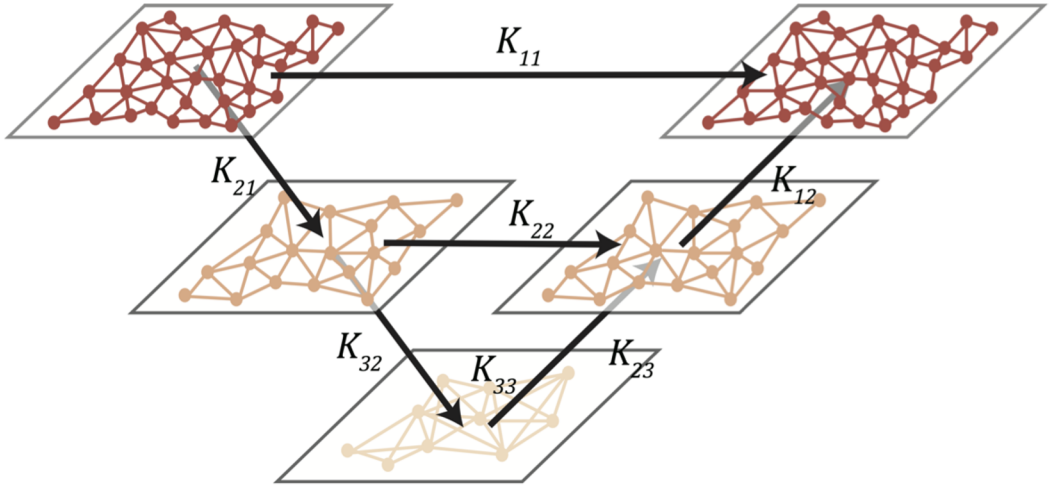
For the solution operator of Navier-Stokes equation, the number of parameters depends sub-linearly on the error

Neural Operator: Learning Maps Between Function Spaces. Kovachki et. al.
On universal approximation and error bounds for Fourier Neural Operators. Kovachki et. al.

# Graph-based neural operators



GKN




MGKN

https://arxiv.org/abs/2003.03485

https://arxiv.org/abs/2006.09535 (Neurips2020)

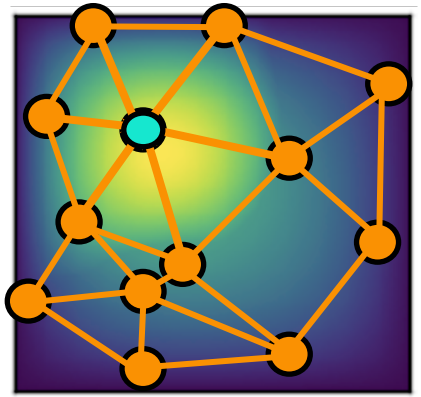
# Kernel convolution as message passing on graph

$$v_{t+1}(x) = \sigma\left(W v_t(x) + \int_D \kappa_\phi(x, y, a(x), a(y)) v_t(y)\, \nu_x(dy)\right)$$

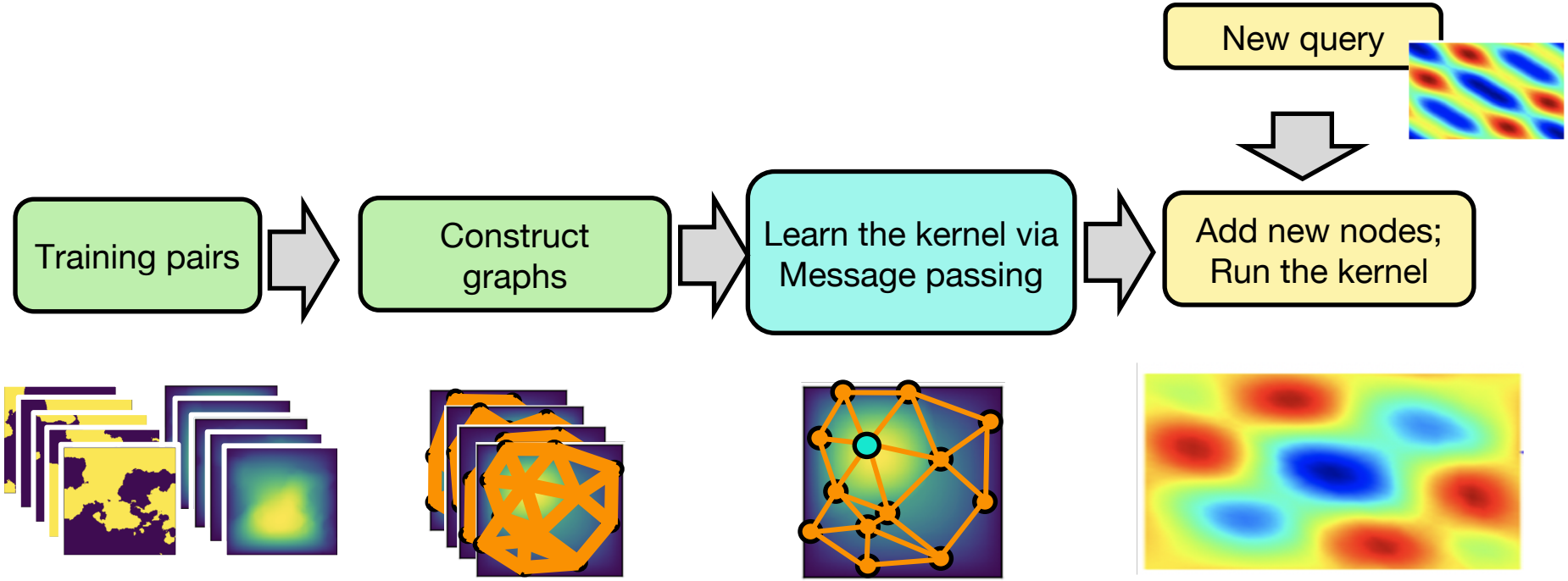
$$v_{t+1}(x) = W v_t(x) + \sum_{y \in N(x)} \kappa_\phi\big(e(x,y)\big) v_t(y)$$

Graph neural network

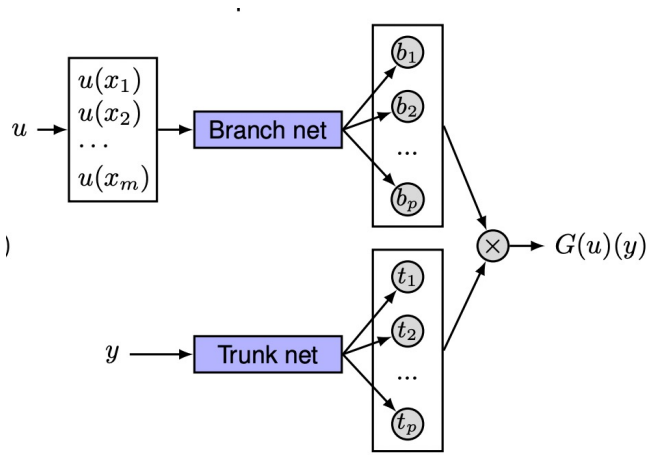
- Adjacency matrix = kernel matrix.
- Kernel integration = message passing



# Graph Kernel Operator



# 3.2 Deep Operator Network




DeepONet: Lu Lu, Pengzhan Jin, George Em Karniadakis

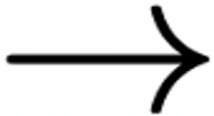
# DeepONet

F: [a(x), y] → u(y)
$[(R^n \to R^m), R^n] \to R^m$

Input functions + query point → the solution at the query point

Input: a(x)

(y)



u(y)

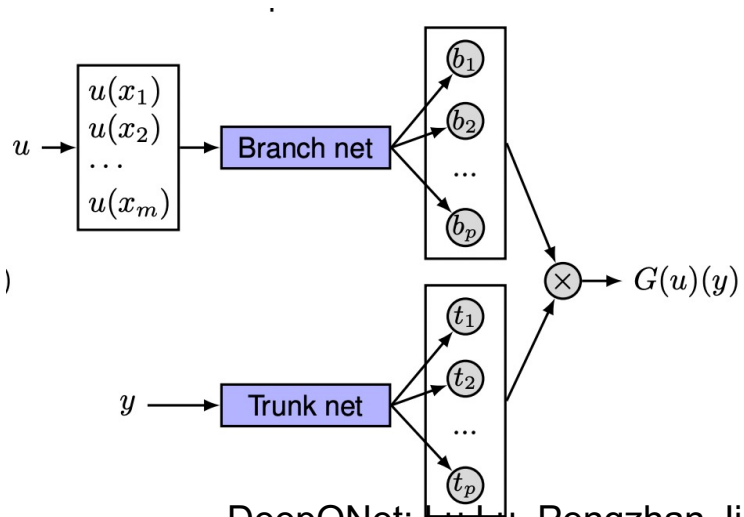
DeepONet: Lu Lu, Pengzhan Jin, George Em Karniadakis

# DeepONet

Two networks:

- Branch net takes the input function
- Trunk net takes the query point

Input: a(x)




u(y)

(y)

DeepONet: Lu Lu, Pengzhan Jin, George Em Karniadakis

# DeepONet and Neural operator

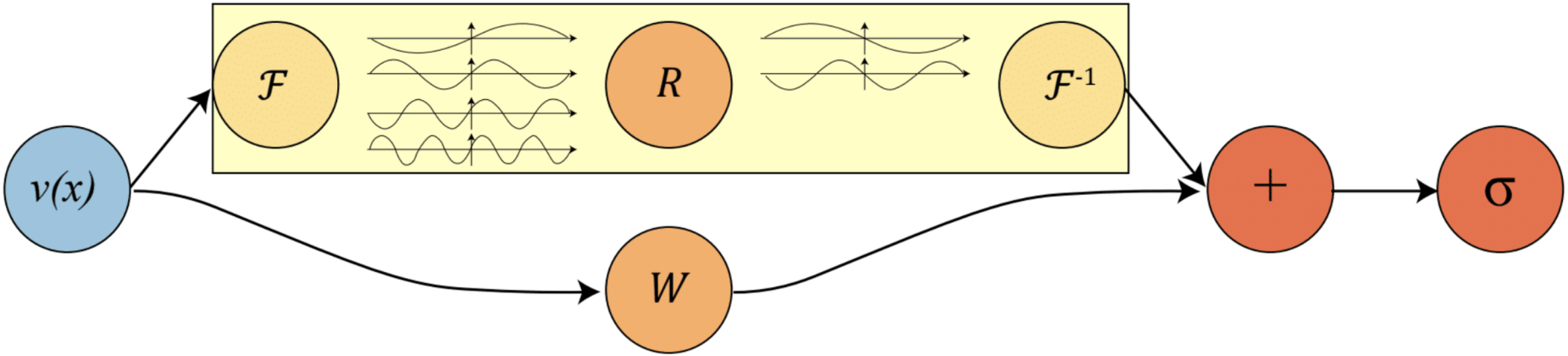
Two slightly different formulations.
Potentially one can be converted into the other.

Implementation difference:

- Neural operator is easier if input and output are fully observed functions.
- DeepONet is easier when given sparse observation from sensors a(x) and query u(y).

DeepONet: Lu Lu, Pengzhan Jin, George Em Karniadakis

# 3.3 Fourier neural operator



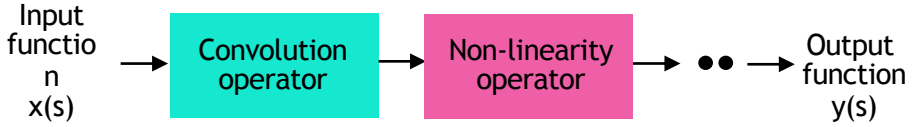

https://arxiv.org/abs/2006.09535 (ICLR2021)

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar

# Fourier neural operators

F: $x(s) \to y(s)$

$(R^n \to R^m) \to (R^n \to R^m)$




## Architecture

$$v_{l+1}(s) = \sigma\left(W_l v_l(s) + \int_D \kappa_l(s,z) v_l(z)\, dz + b_l(s)\right)$$

## Fourier space

- Assume $\kappa_l(s,z) = \kappa_l(s-z)$ and parametrize its Fourier components $\theta_l$.
  - Convolution theorem: $\int_D \kappa_l(s-z) v_l(z)\, dz = \mathcal{F}^{-1}(\theta_l \cdot \mathcal{F}(v_l))(s) \qquad \mathcal{O}(n \log n)$

# Fourier layer

Use convolution as the integral operator
and implement with Fourier transform

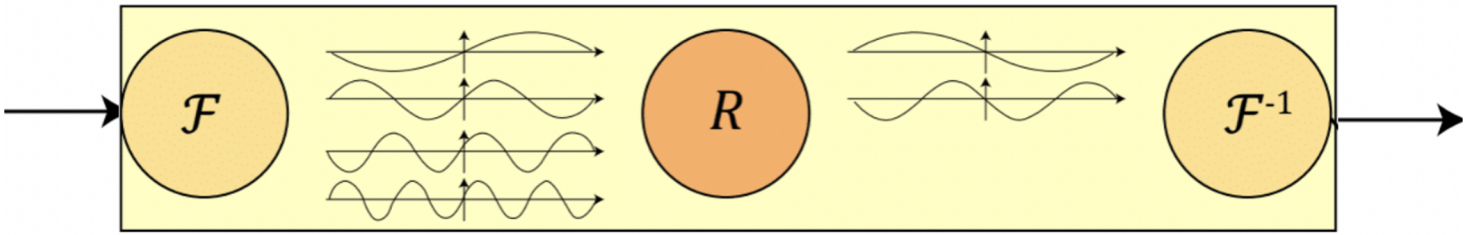
$$\big(\mathcal{K}(a;\phi)v_t\big)(x) := \int_D \kappa\big(x, y, a(x), a(y); \phi\big) v_t(y) \mathrm{d}y,$$

$$\big(\mathcal{K}(\phi)v_t\big)(x) = \mathcal{F}^{-1}\Big(R_\phi \cdot (\mathcal{F}v_t)\Big)(x)$$

# Fourier layer

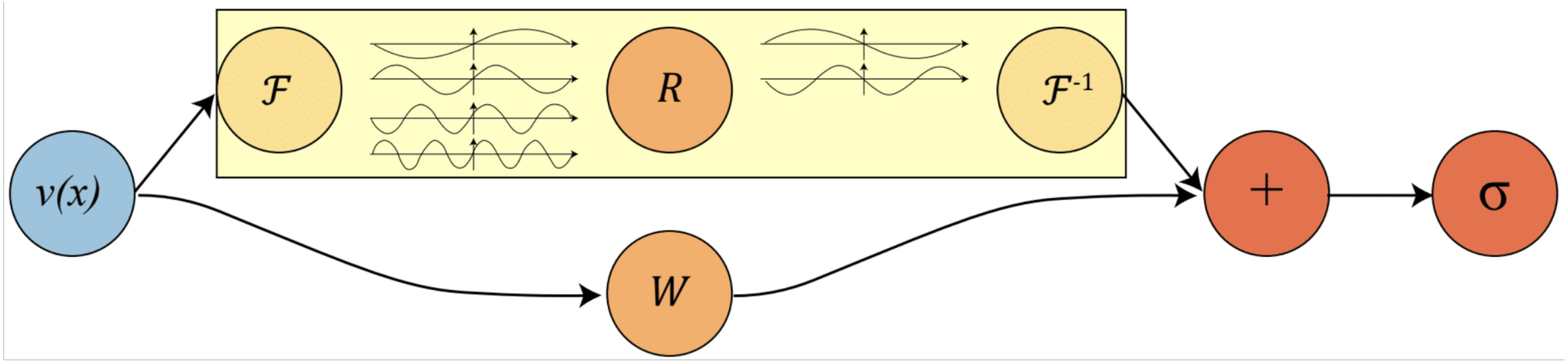
1. Fourier transform
2. Linear transform
3. Inverse Fourier transform

$$\big(\mathcal{K}(\phi)v_t\big)(x) = \mathcal{F}^{-1}\Big(R_\phi \cdot (\mathcal{F}v_t)\Big)(x)$$



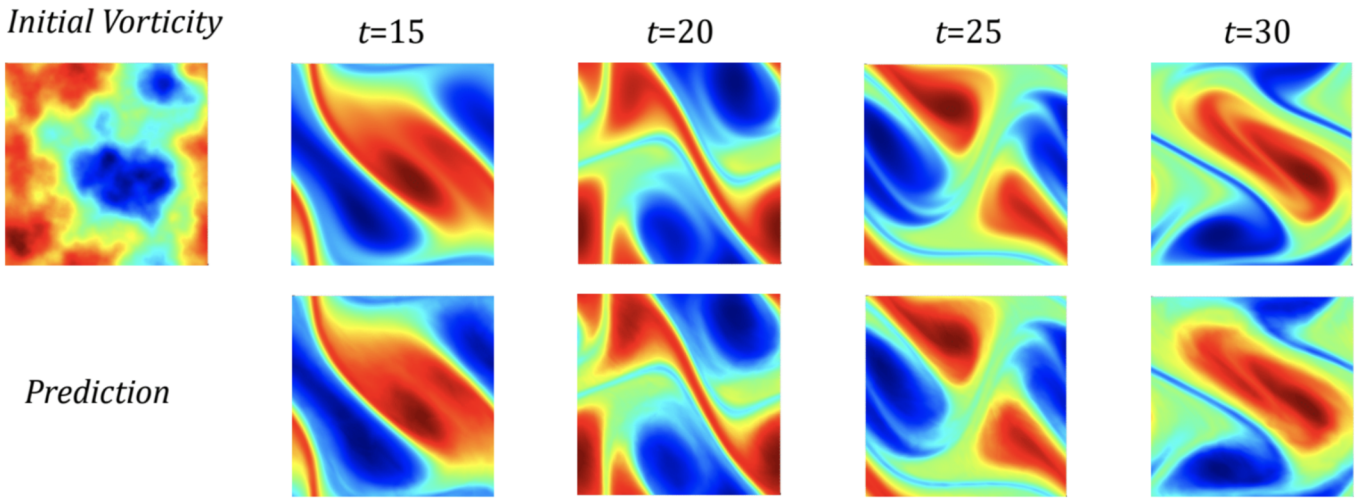
# Fourier layer

The linear transform *W* outside keep the track of the location information (x) and non-periodic boundary




$$v_{t+1}(x) = \sigma\left(Wv_t(x) + \int_D \kappa_\phi(x, y, a(x), a(y))v_t(y)\,\nu_x(dy)\right)$$

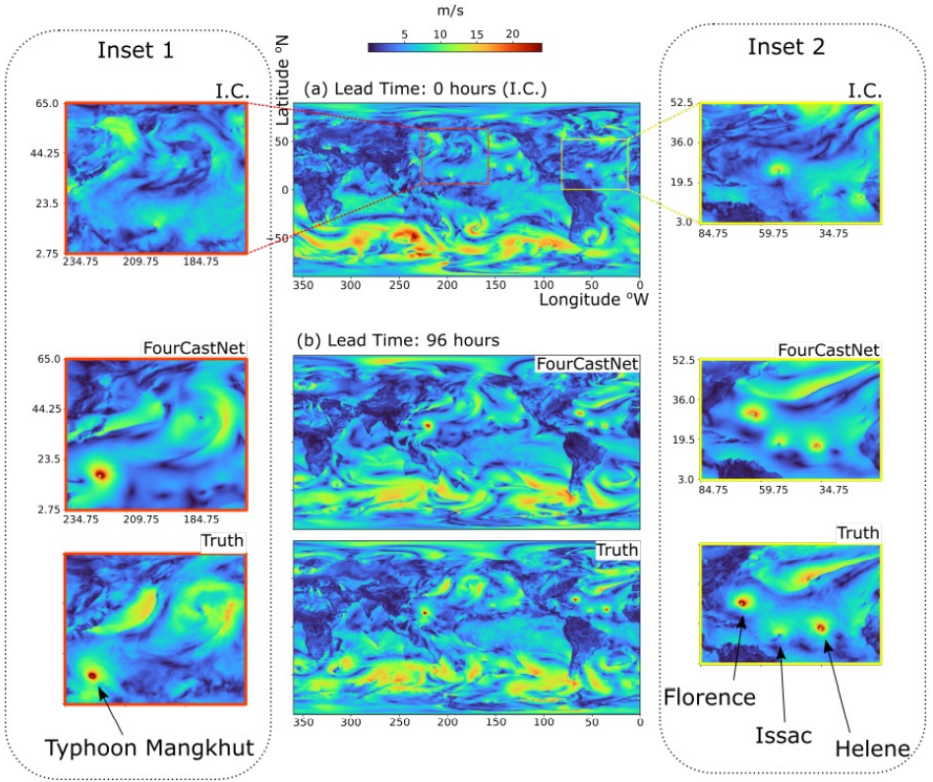
# 4. Experiments



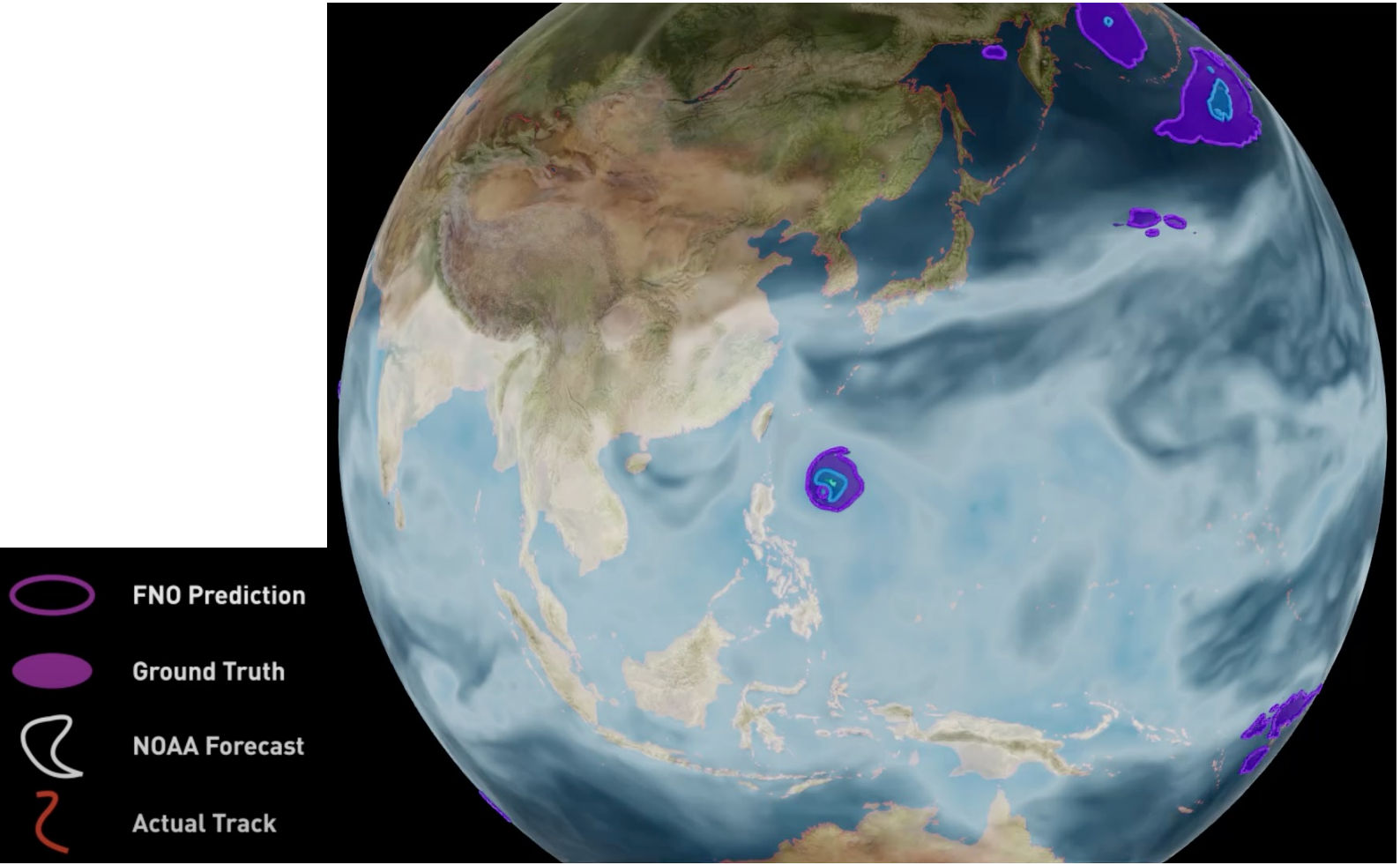
# Weather Forecast

Weather forecast: the AFNO model on the ERA5 dataset

Input the current weather variables, output the weather at 6 hours later.

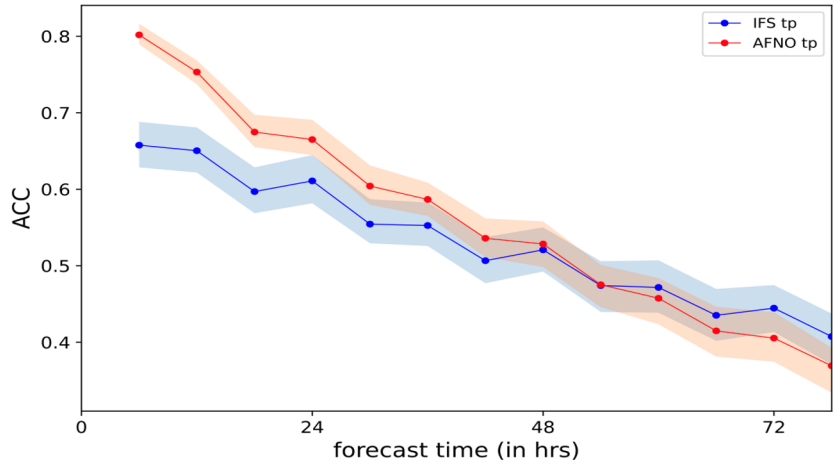



Pathak et al. (2022), FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, arXiv: https://arxiv.org/abs/2202.11214


FNO Prediction
Ground Truth
NOAA Forecast
Actual Track

# Weather Forecast

- Unprecedented skill on precipitation forecasts
- 6X higher resolution than other DL models
- 1000-member ensemble in a fraction of a second
- 4 to 5 orders of magnitude speedup over NWP
- 25000X smaller energy footprint

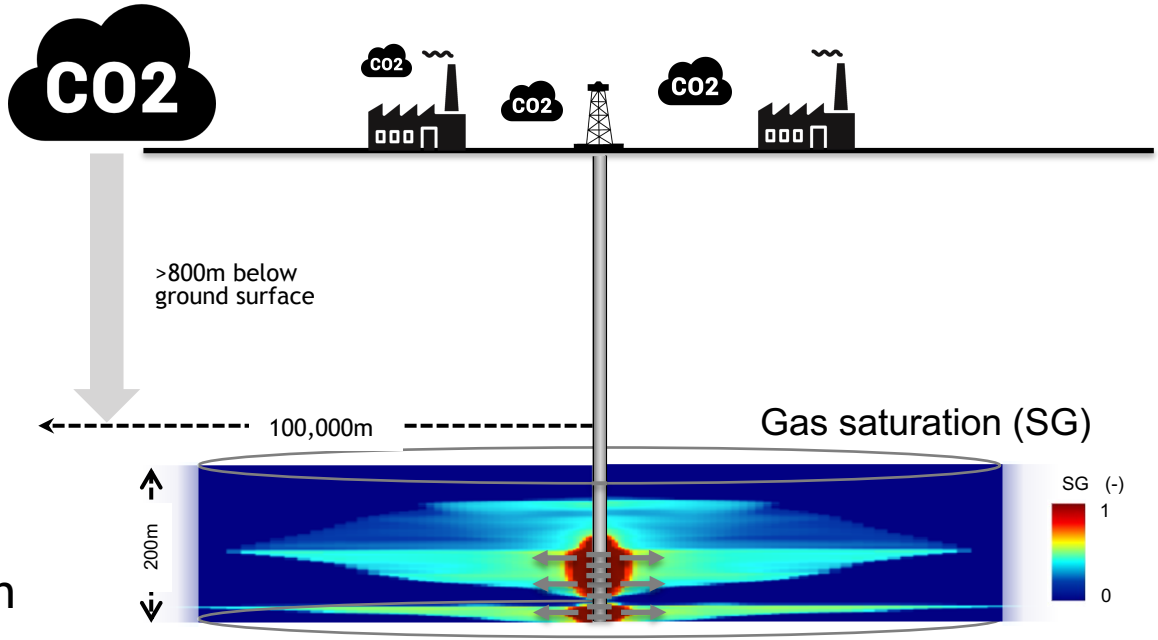



Pathak et al. (2022), FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators, arXiv: https://arxiv.org/abs/2202.11214

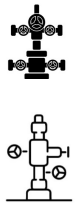
# $CO_2$ injection

Inject $CO_2$ into the earth

Measure $CO_2$ migration

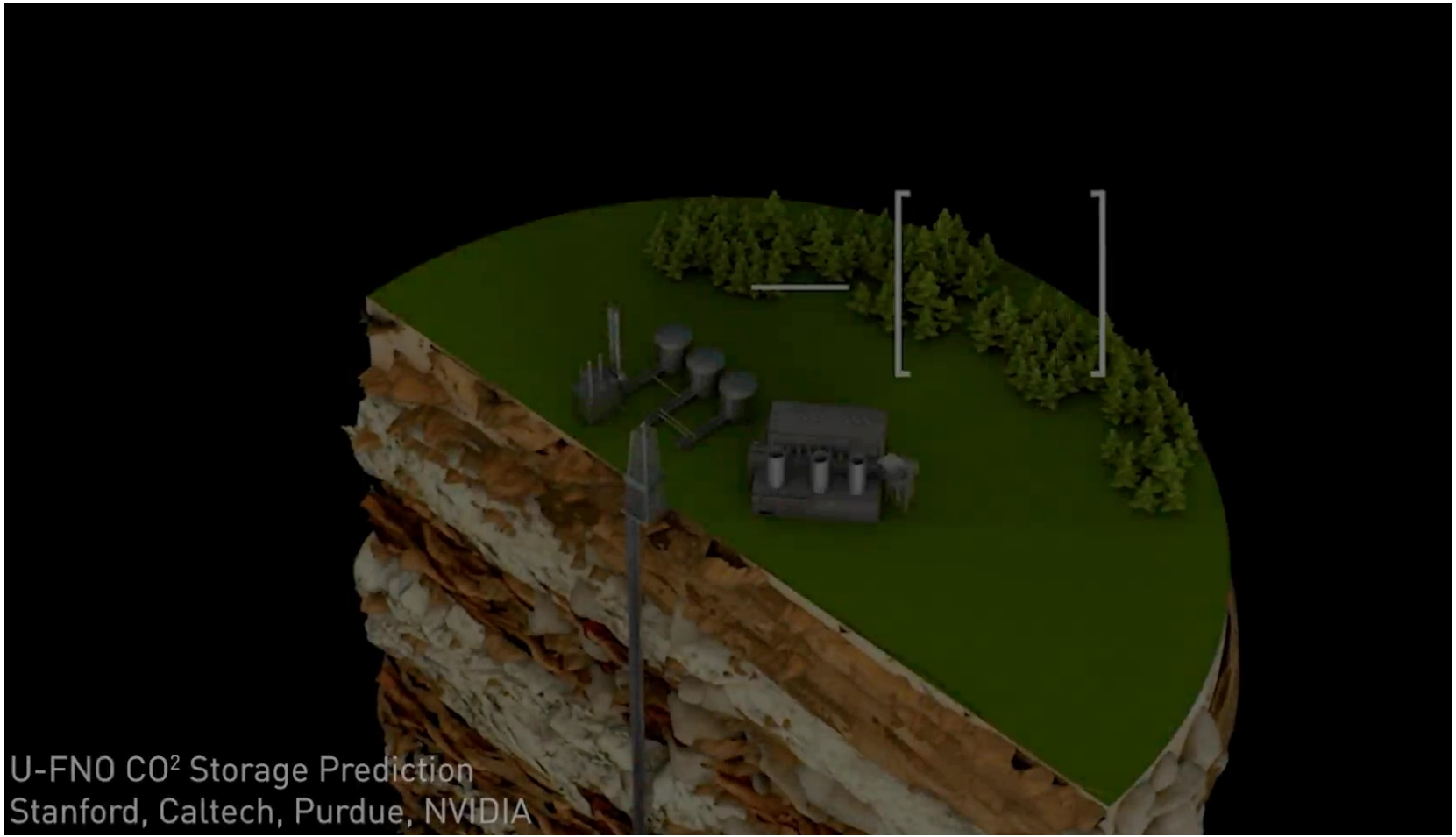



Alternative injection well icons



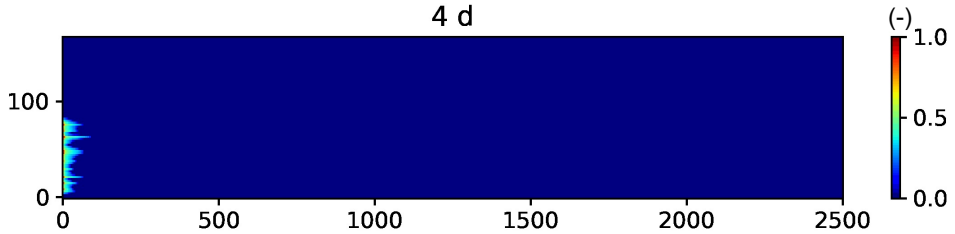
FNO provides 10^5x speedup

compared to a conventional Finite Difference Method solver

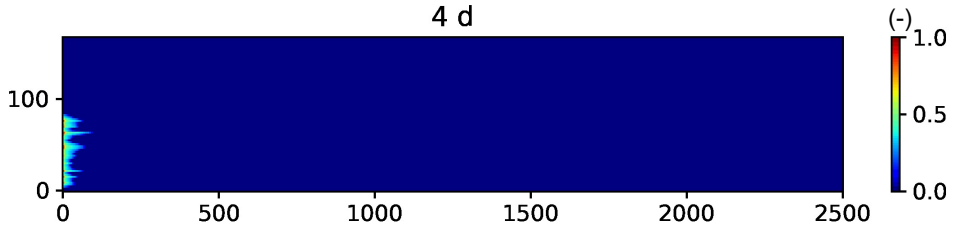

U-FNO CO² Storage Prediction
Stanford, Caltech, Purdue, NVIDIA

# Animation of Gas Saturation values over 30 years

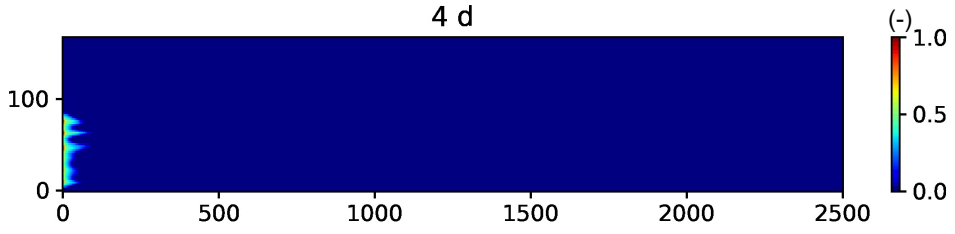
Ground truth




U-FNO Prediction

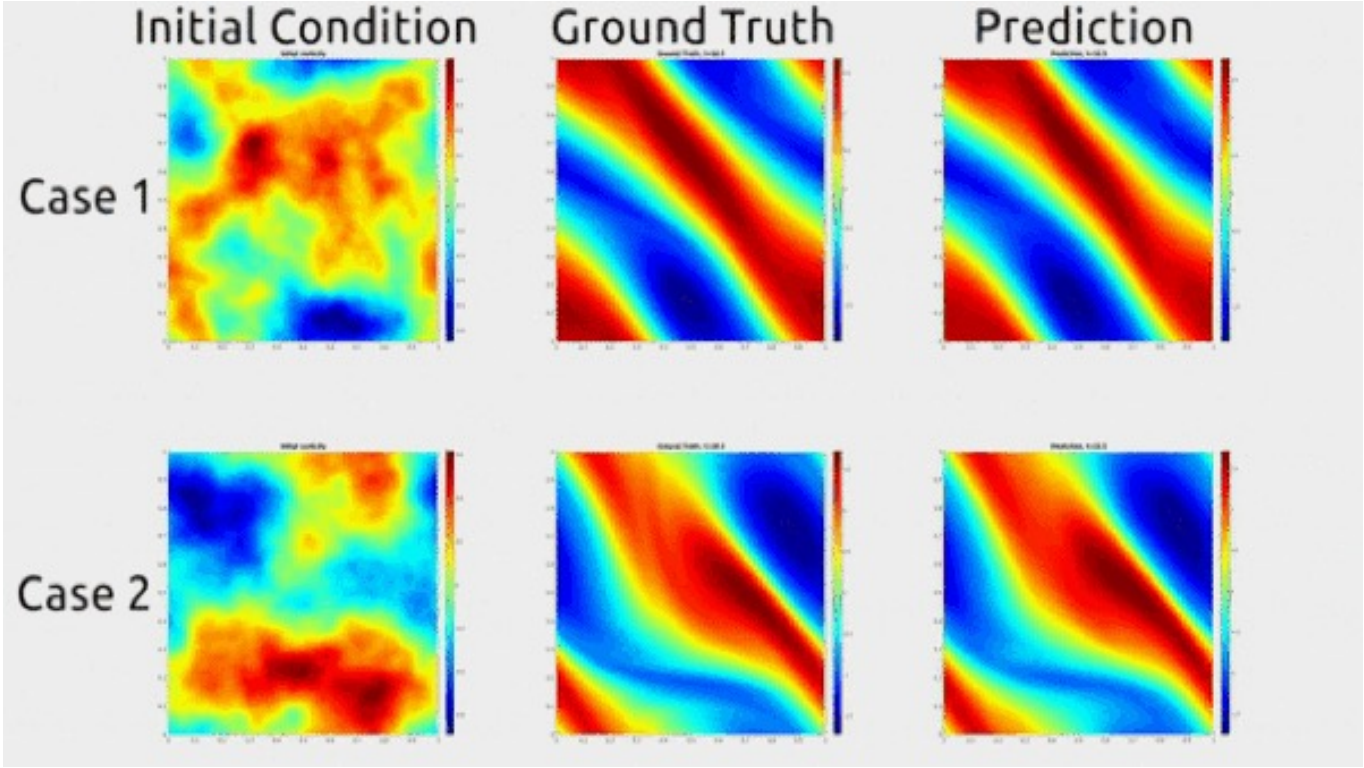



CNN Prediction

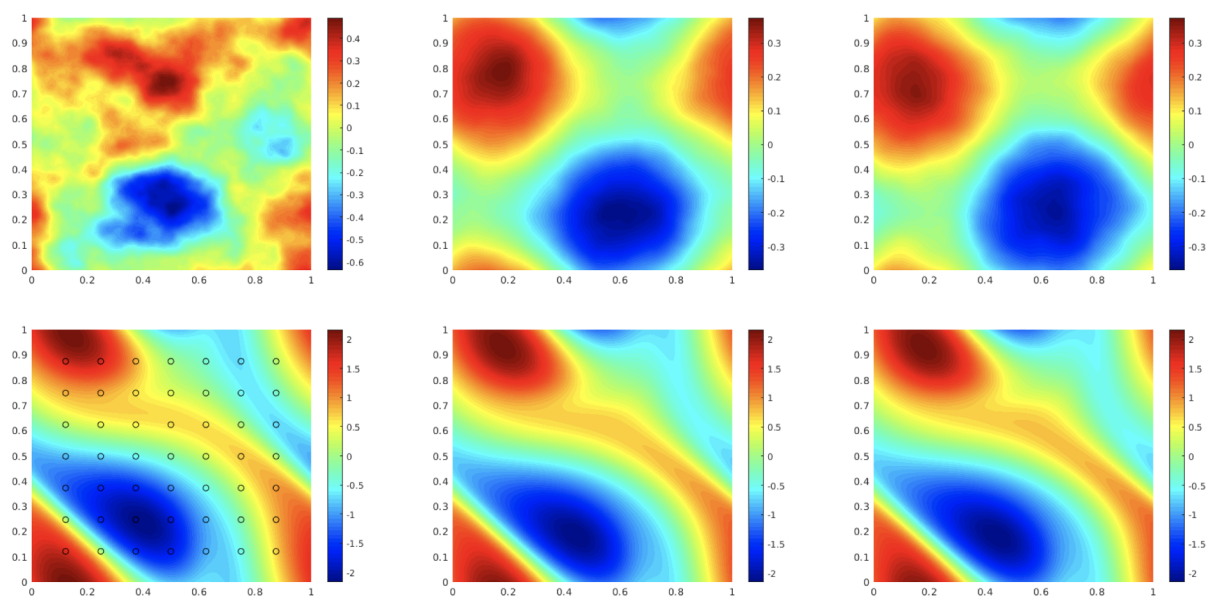



FNO is 10^5x faster than the FDM solver, less blurry compared to CNN

# V=1e-4, zero-shot super-resolution

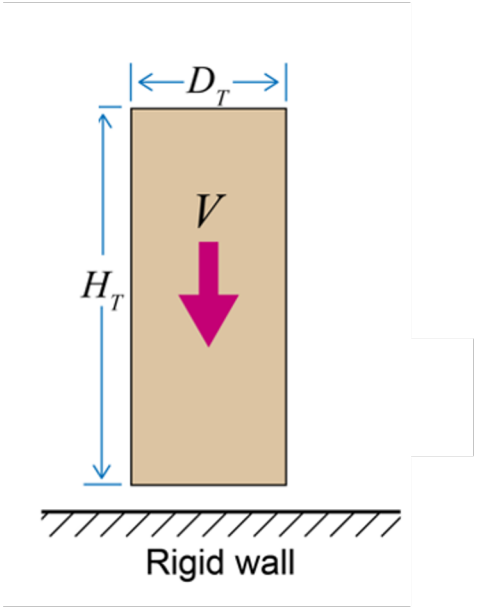


# Bayesian inverse problem:



We use a MCMC method, sampling initial conditions and evaluating them with the traditional solver and Fourier operator. The Fourier operator takes **0.005s** to evaluate each initial condition, while the traditional solver takes **2.2s**.

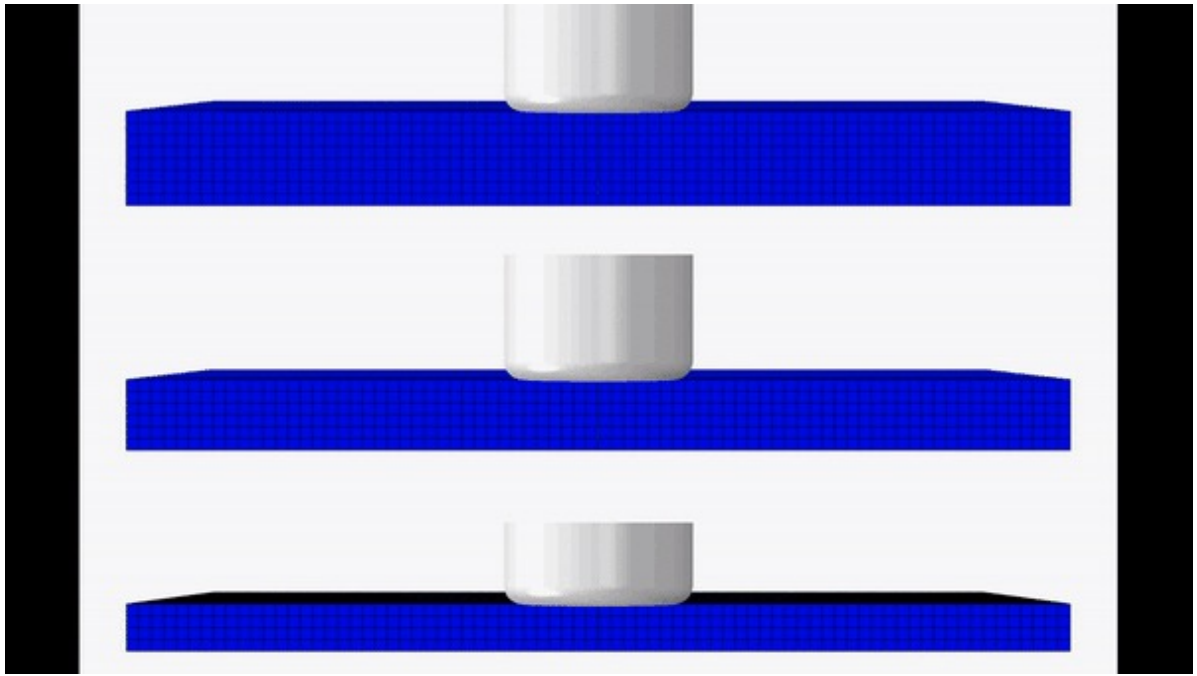
# Plasticity




$$\nabla \cdot S^\varepsilon = \rho^\varepsilon u_{tt}^\varepsilon$$
$$K^\varepsilon = 0$$
$$u^\varepsilon(x,0) = u_0(x), \quad u_t^\varepsilon(x,0) = v_0(x), \quad \xi^\varepsilon(x,0) = \xi_0$$
$$u^\varepsilon(x,t) = u^*(x,t)$$
$$S^\varepsilon(\nabla u^\varepsilon, \xi^\varepsilon, x) n(x) = s^*(x,t)$$

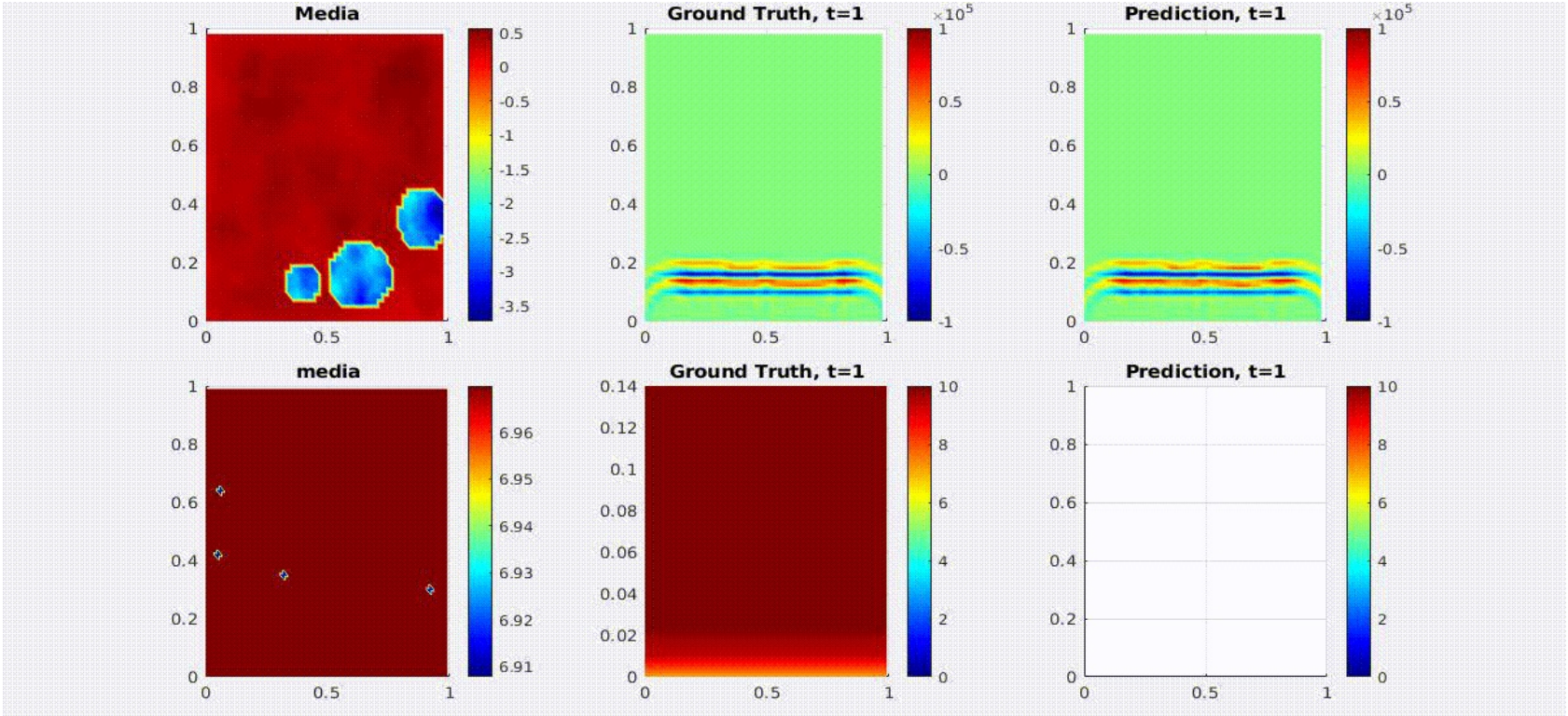
Multi-scale method: use neural operator to map from strain to stress on the unit cell; update macroscale with Abaqus solver.

# Plasticity

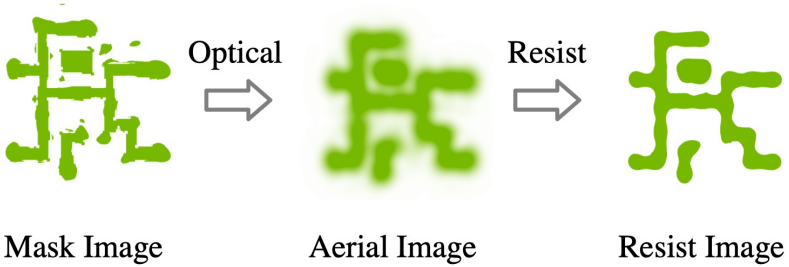


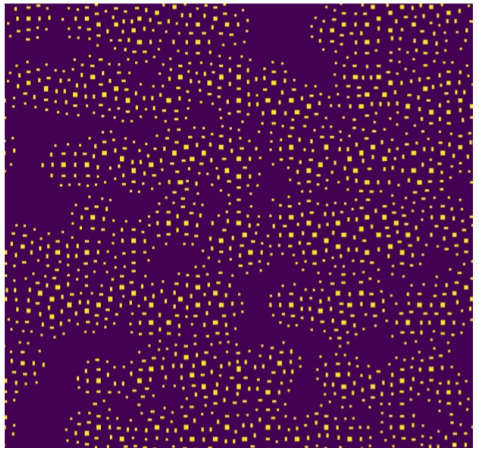
PCA-operator solves multi-scale plasticity problem (Burigede et. al.)

# Ultrasound


Media
Ground Truth, t=1
Prediction, t=1
media
Ground Truth, t=1
Prediction, t=1

# lithography

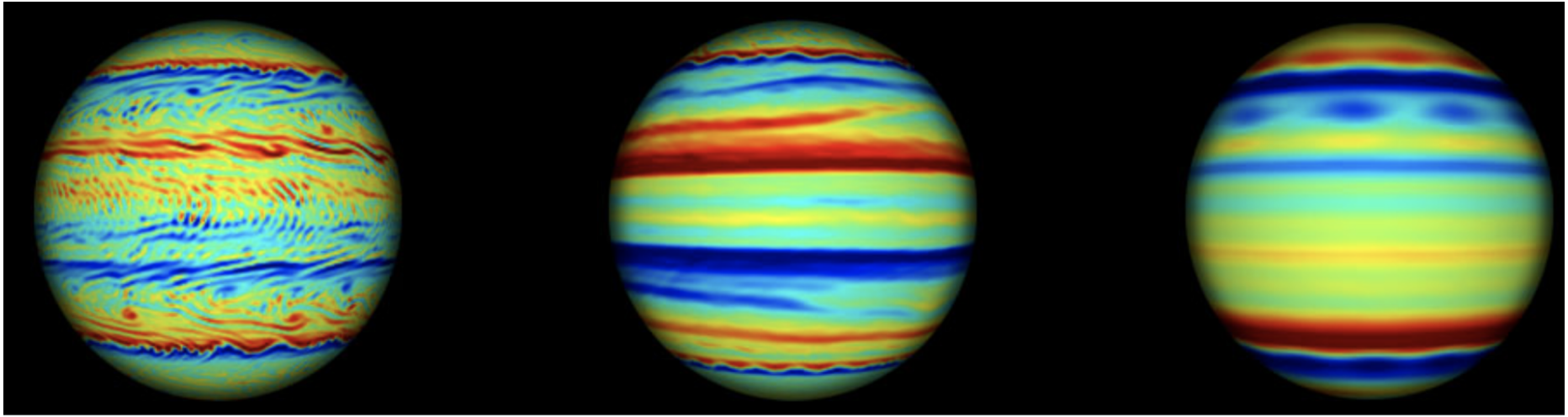


| Benchmark | UNet@10epoch [25] | | DAMO-DLS@10epoch [10] | | DAMO-DLS@100epoch [10] | | Ours@10epoch | |
|---|---|---|---|---|---|---|---|---|
| | mPA (%) | mIOU (%) | mPA (%) | mIOU (%) | mPA (%) | mIOU (%) | mPA (%) | mIOU (%) |
| ISPD-2019 (L) | 99.40 | 98.03 | 98.40 | 97.50 | 99.25 | 98.11 | **99.43** | **98.27** |
| ISPD-2019 (H) | 99.08 | 97.97 | - | - | - | - | **99.21** | **98.45** |
| ICCAD-2013 (L) | 97.30 | 95.38 | 96.24 | 95.15 | 98.94 | 96.97 | **98.98** | **97.79** |
| ICCAD-2013 (H) | 95.16 | 93.04 | - | - | - | - | **99.12** | **97.77** |

Haoyu Yang et. al.

# 5. Extensions



# PINO: physics-informed neural operator

Zongyi Li*, Hongkai Zheng*, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, Anima Anandkumar

# PINO: Physics-informed neural operator

Data loss: compare prediction and ground-truth solution

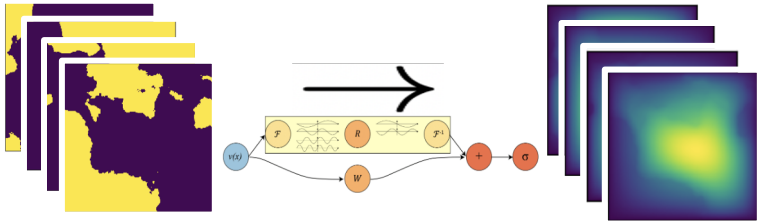
Equation loss: plug prediction in PDE and compute residual

$$
\begin{aligned}
-\nabla \cdot (a(x)\nabla u(x)) &= f(x), \quad && x \in D \\
u(x) &= 0, && x \in \partial D
\end{aligned}
$$

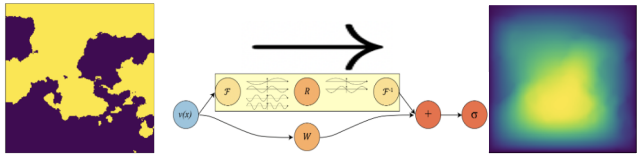
# PINO: PHYSICS-INFORMED NEURAL OPERATOR

Neural operators use only the data. If the explicit form of the PDE is known, we can combine neural operators with the PINNs setting.

1. Pre-train
- operator-learning setting
- a family of PDE
- use the data loss (and equation loss)



2. test-time optimize
- solver setting (PINN)
- a specific instance
- use the equation loss

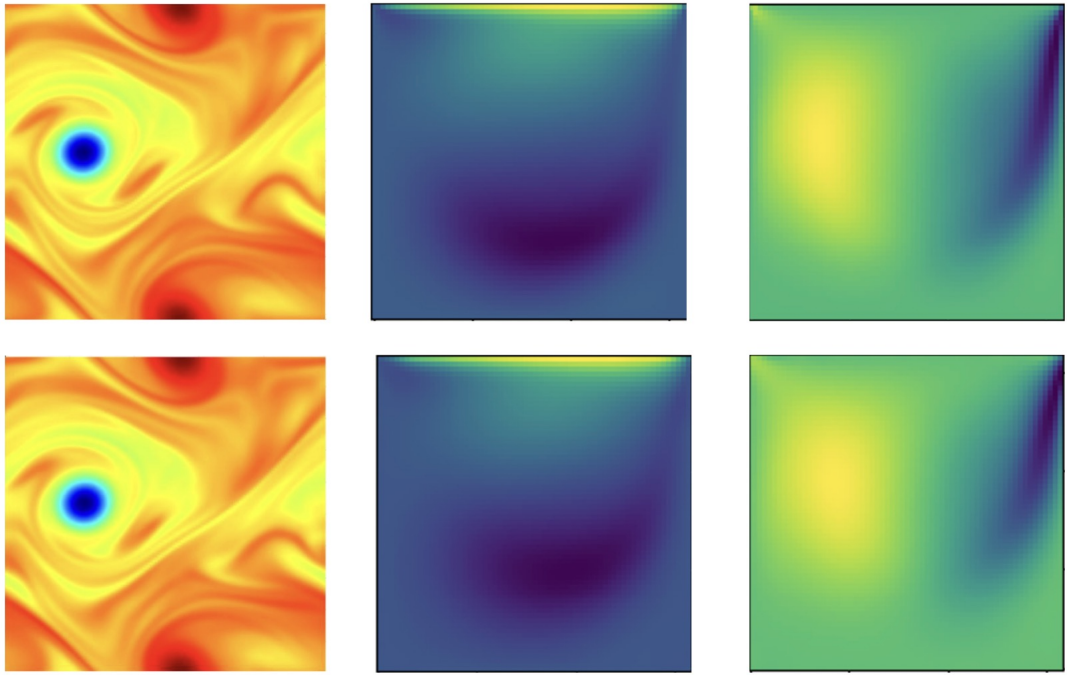


# Physics-informed neural operator

Standard cases (Burgers, Darcy, KF, Cavity): no need to use any data

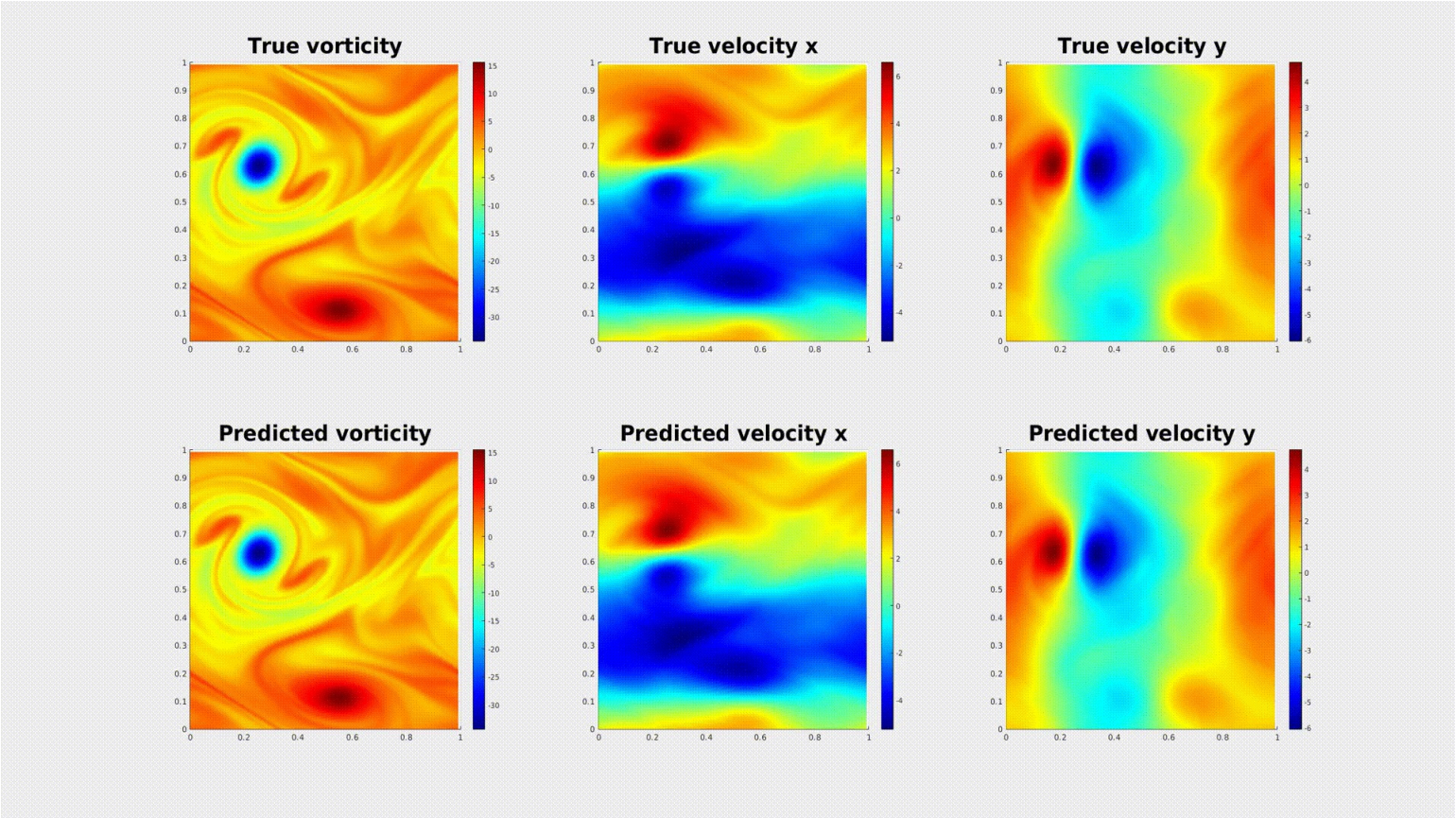
- Pretrain improves both the convergence speed and final error.
- ground truth

prediction



(a) KF: vorticity (b) Cavity: velocity in x, y

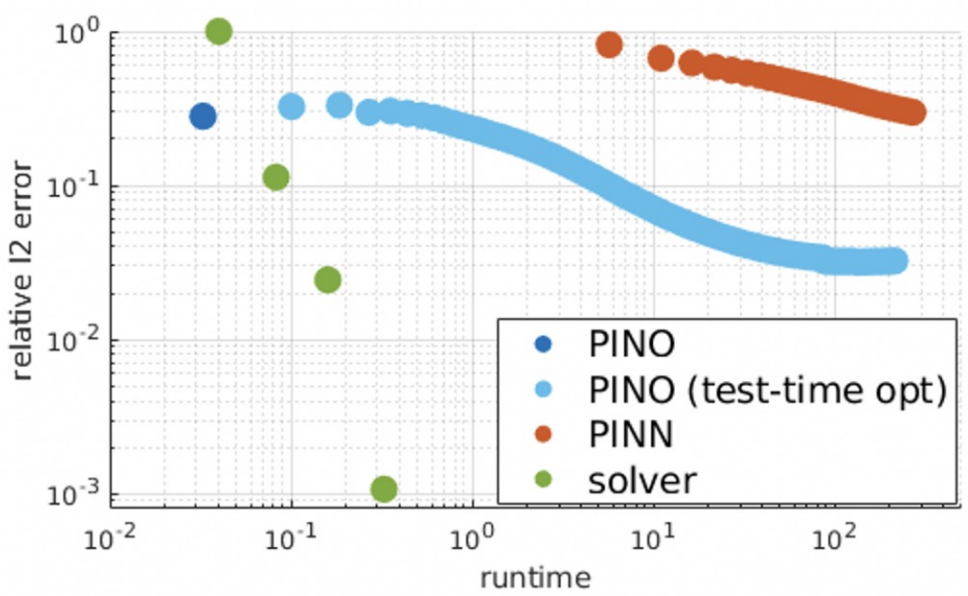
- PINO gets 2% error on Re500 [2π x 2π x 1s]
- Easily generalize from one Re to another




Relative error: PINO: 0.9%, PINN:18.7%

# Physics-informed neural operator

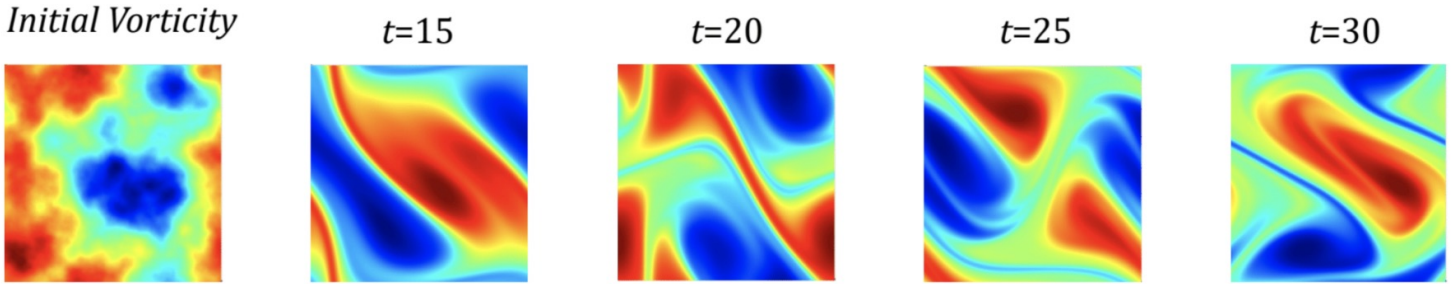
- Pretrain improves both the convergence speed and final error.
- More robust to hyperparameters




PINN: Raissi, Maziar; Perdikaris, Paris; Karniadakis, George Em

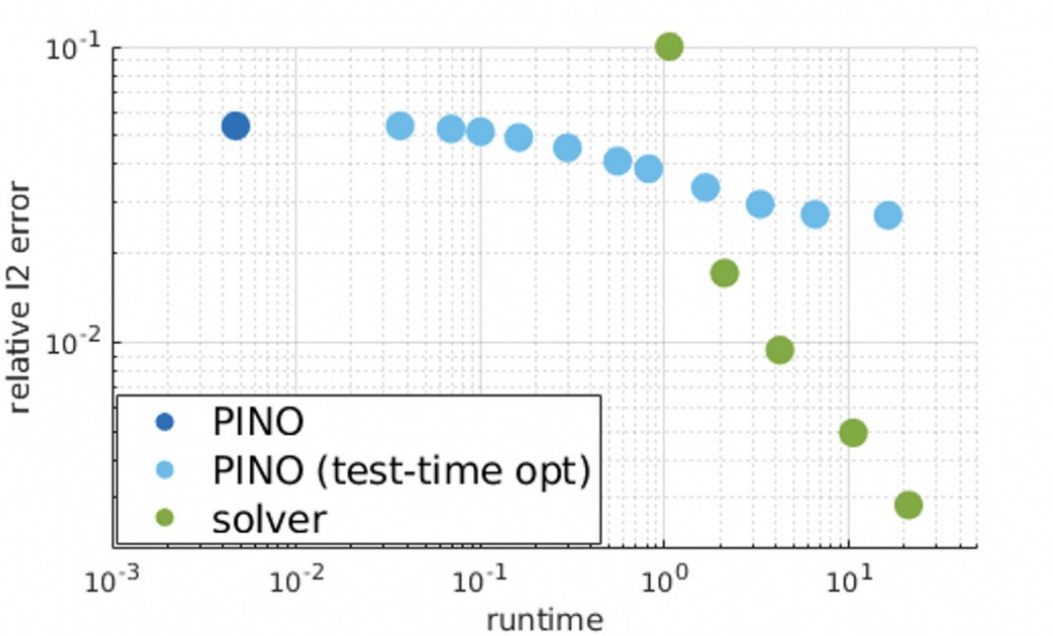
# Physics-informed neural operator

If data is available: PINO can solve very challenging scenarios such as long-temporal transient flow T=50, which is intractable to PINNs.

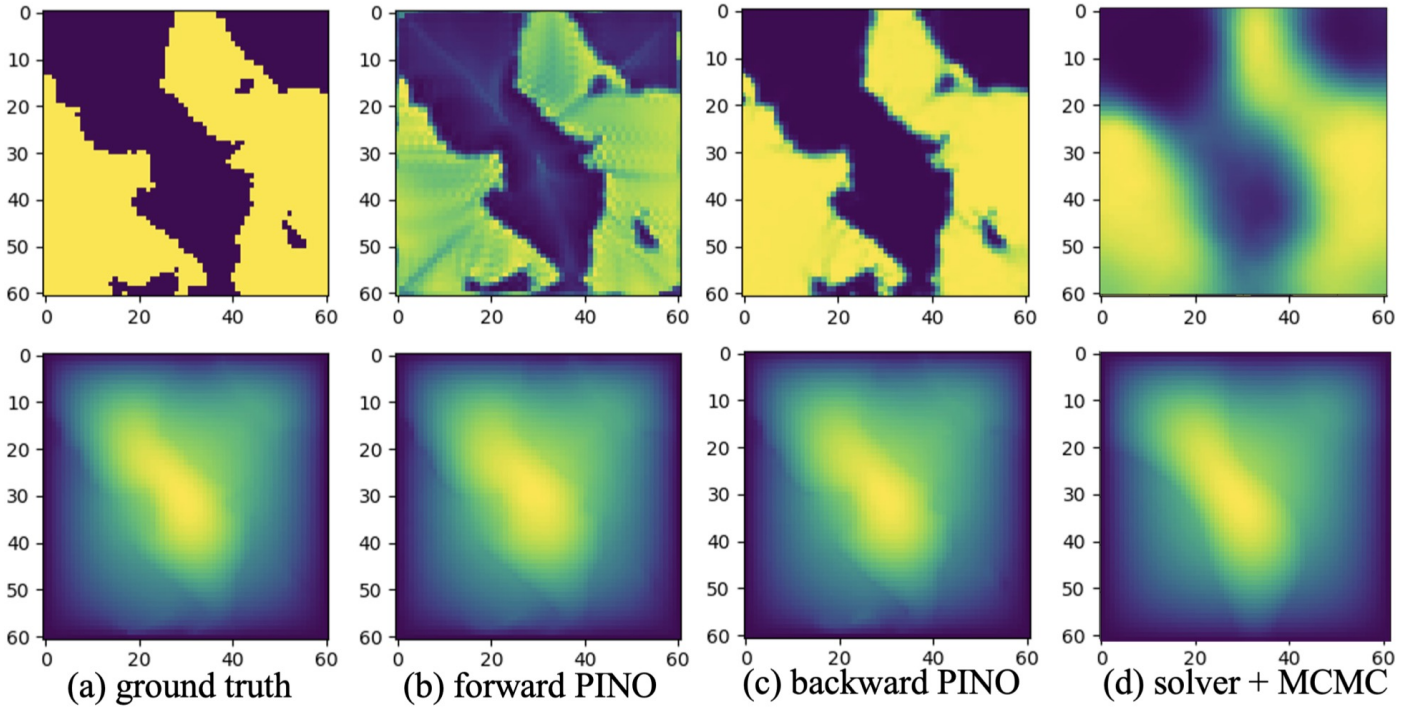


# Physics-informed neural operator

Long-temporal transient flow: preserve the 400x speedup compared to the pseudo-spectral solver.

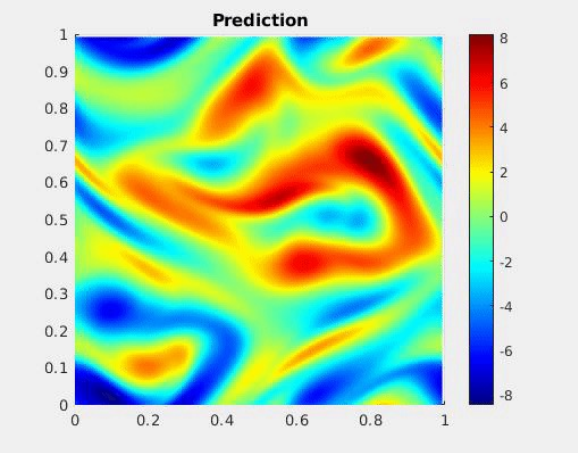


# Solving inverse problems

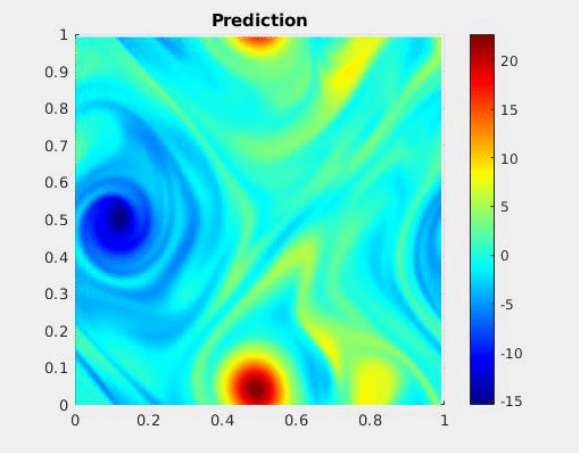



Backward PINO accurately solves inverse problem. 3000x speedup

# Transfer Learning




Re100




Re500

Pre-train on Re100, fine-tune on Re500. Converge 3x faster.

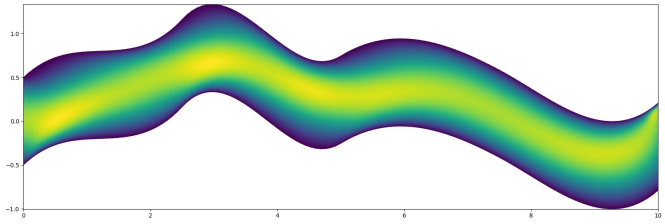
# Geometric-FNO

Zongyi Li, Zhengyu Huang, Burigede Liu, Anima Anandkumar

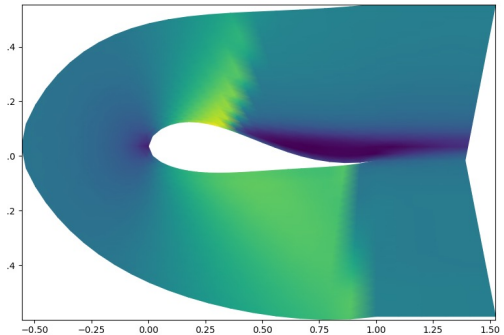
# Geometric FNO

Previous FNO implementation relies on FFT. It can be only applied to rectangular domains with a uniform grid. We want to extend FNO to different geometries.

Further, we want to learn solution operators mapping from the shapes to the solution, and solve the inverse design problems.



Pipe



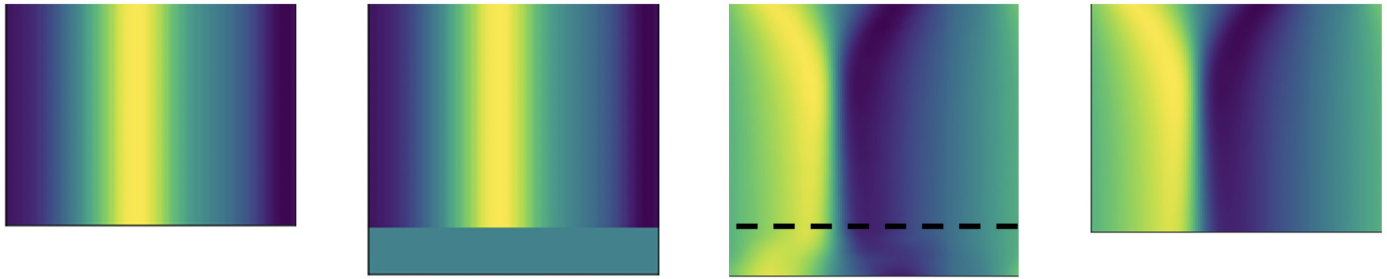
Airfoils

# Geometric FNO

- Any boundary -> embed (Fourier continuation)
- Any mesh -> learned interpolation /DFT
- Any shape -> deform (adaptive/moving mesh)
- Any topology -> decompose (pants/handler decomposition)

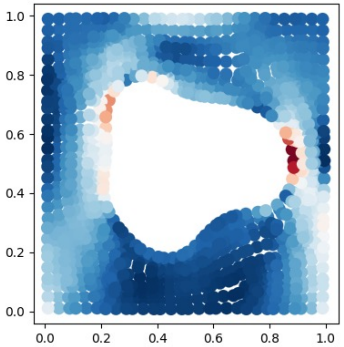
# boundary

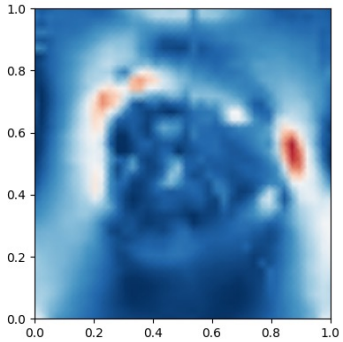
For non-periodic boundary, we can embed the domain into a larger domain with periodic condition



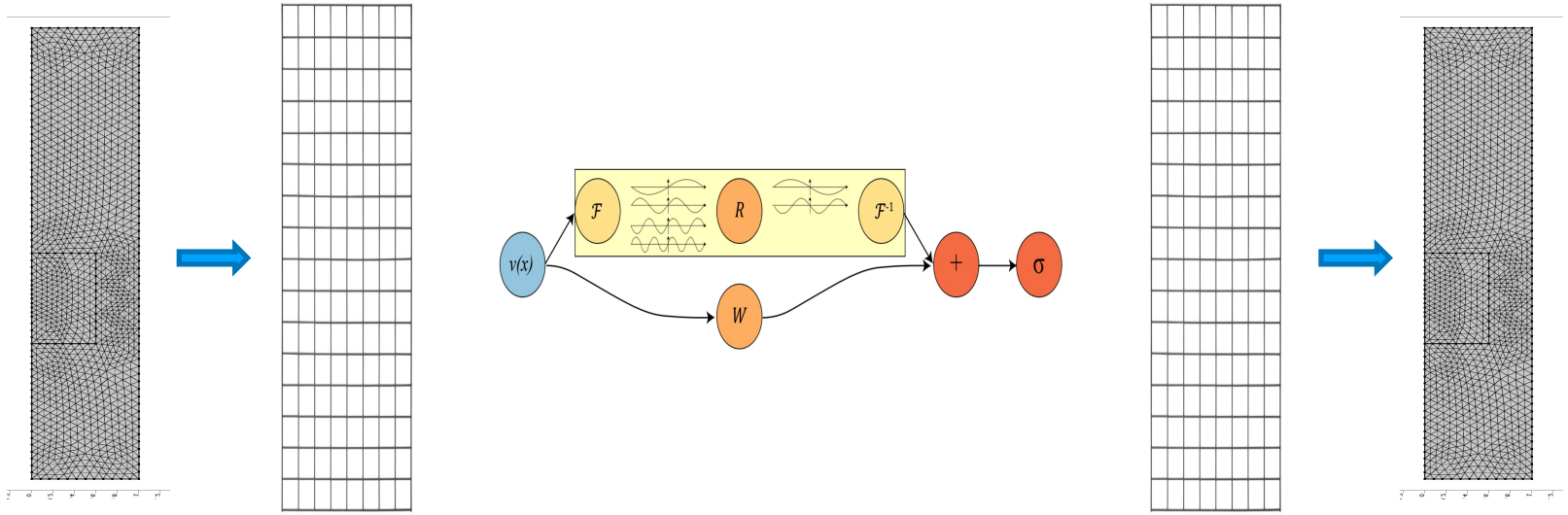
(a) Input (b) padded input (c) padded output (d) target output

Similar, for any obstacles and holes. we can fill in zeros.



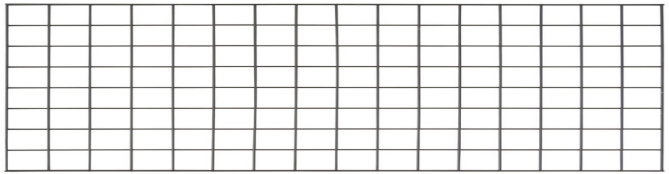



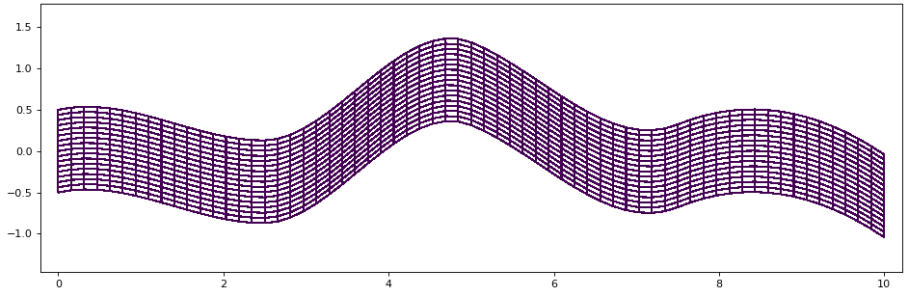
# Mesh




Given any input mesh, we first transform it to a uniform mesh.
We use the learned interpolation /discrete Fourier transform to reduce the interpolation error.

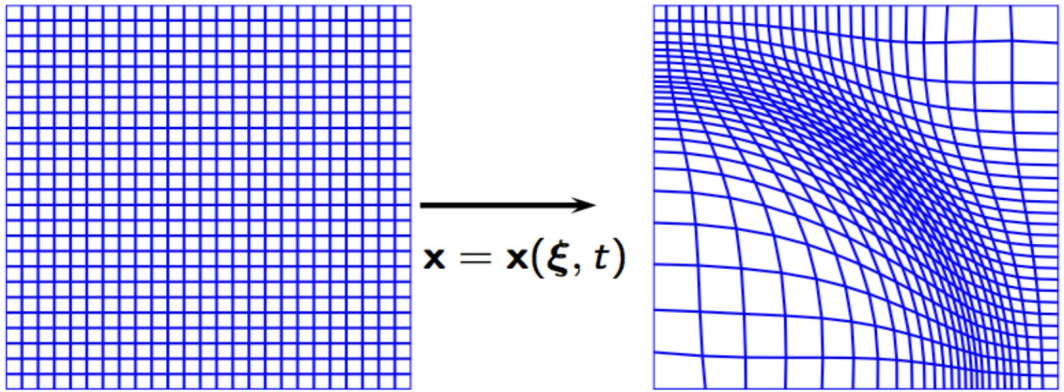
# Shape






Given any input domain, we want to find a deformation (diffeomorphism)
To convert the domain into a regular one.

# Adaptive meshes



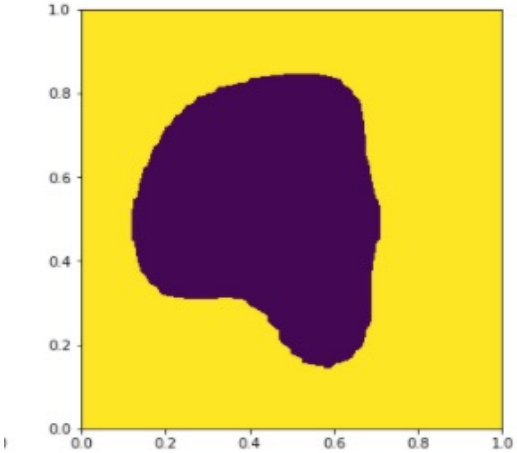

FFT computation mesh

Adaptive mesh

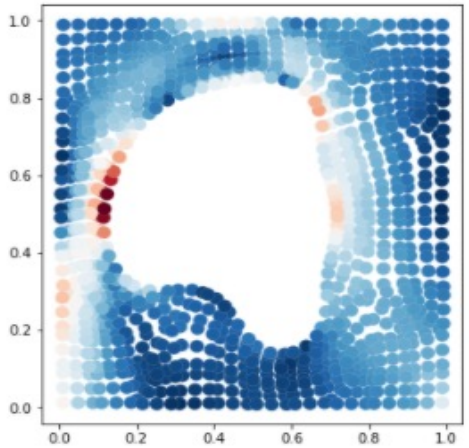
Such deformation can be also used to construct adaptive meshes for multiscale structures

# Examples: elastic equation

A constant forcing is applied from the left. Given the shape, we want to predict the stress.
Unstructured mesh + irregular shape.



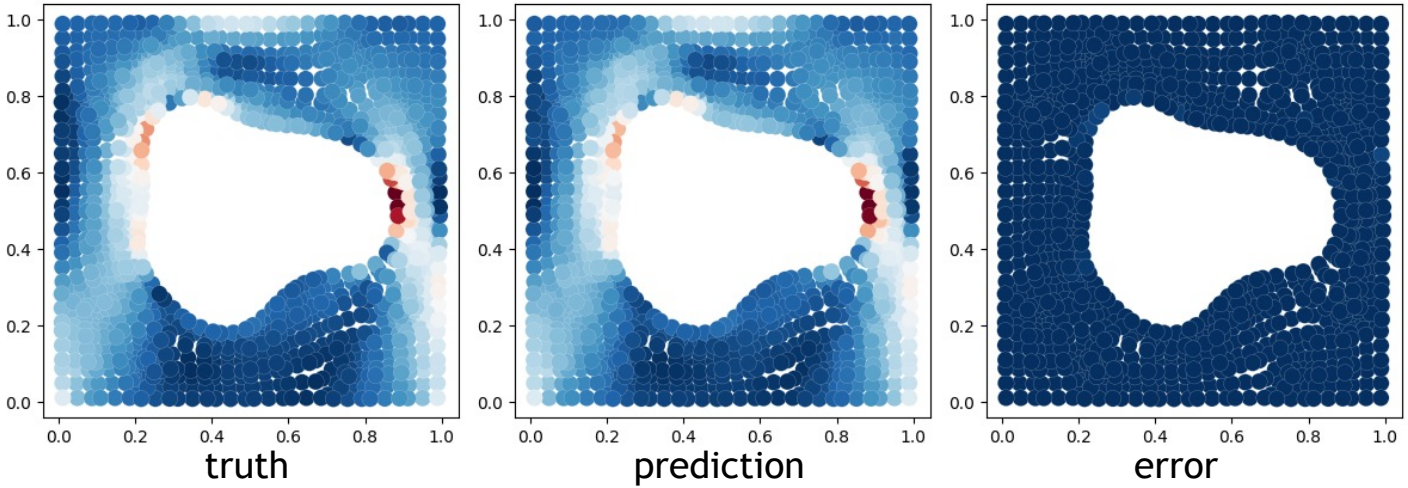

Input: shape




Output: stress

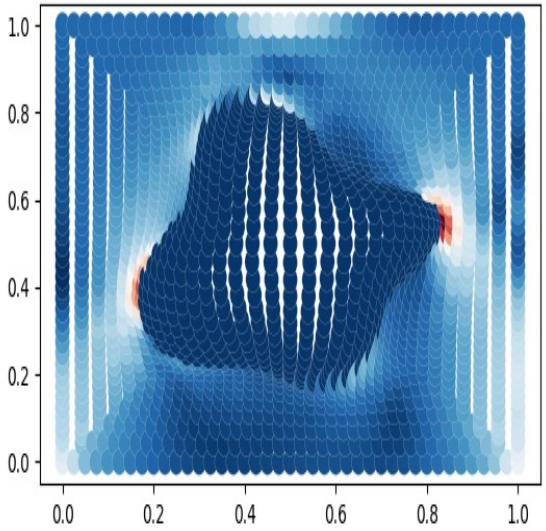
# Examples: elastic equation

Using dis

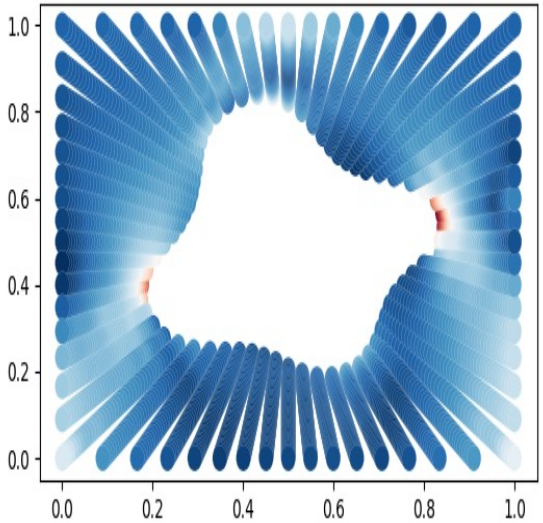


# Examples: elastic equation

Using adaptive meshes further improves the performance.




Adaptive mesh: (x,y)

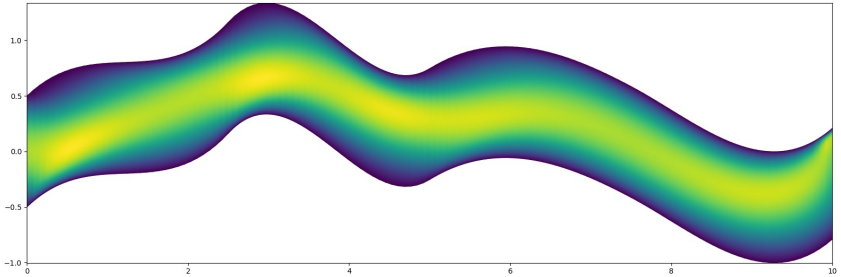



Adaptive mesh: (θ,r)

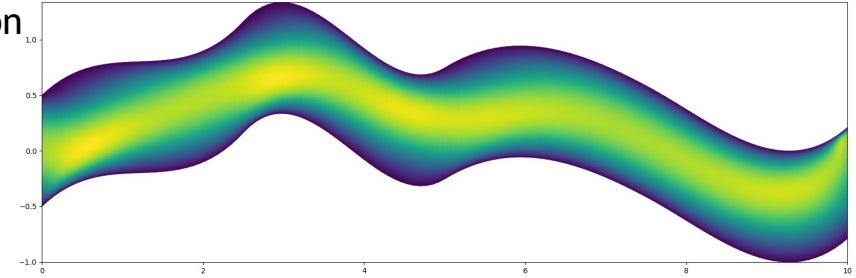
| | Train Raw | Test Raw | Test DFT |
|---|---|---|---|
| Uniform | 7% | 10% | 11% |
| X-Y deform | 9% | 10% | 9% |
| Theta-R | 6% | 6% | ~6% |

# Examples: Channel flow (pipe)

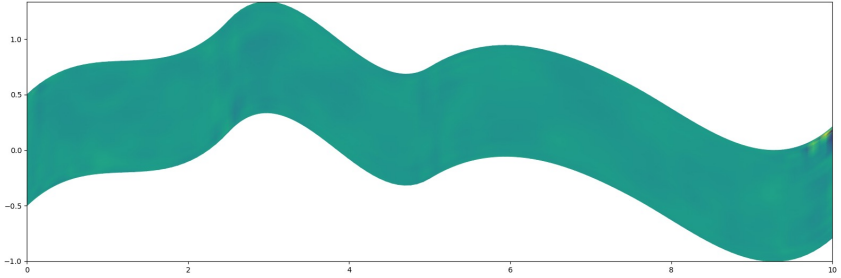
truth




prediction



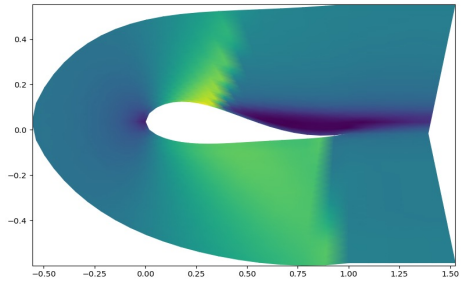

error



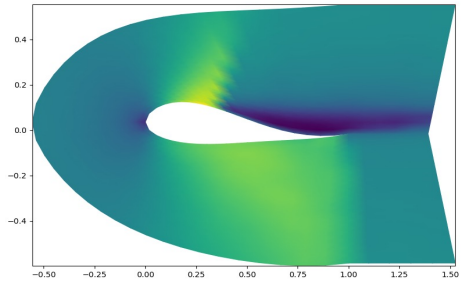

Error = 0.5%

# Examples: Airfoils

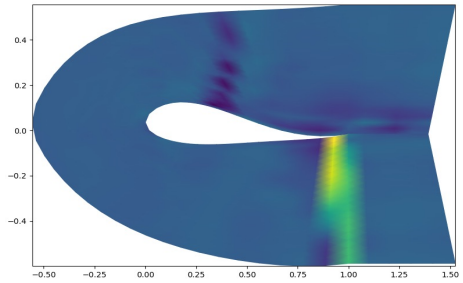
truth




prediction




error

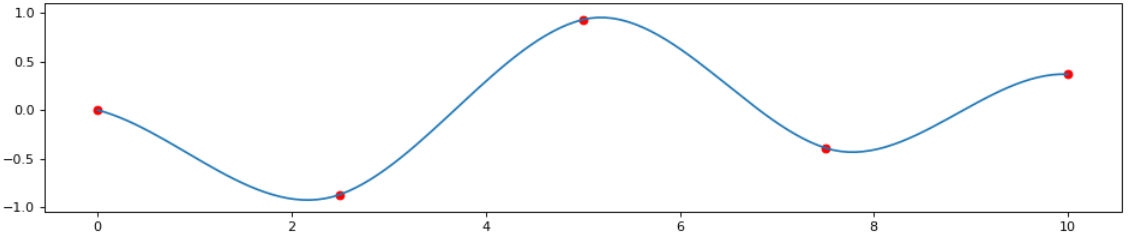


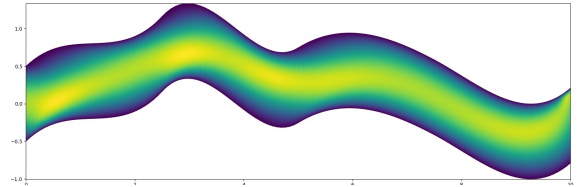

Error = 2%

# Examples: inverse design

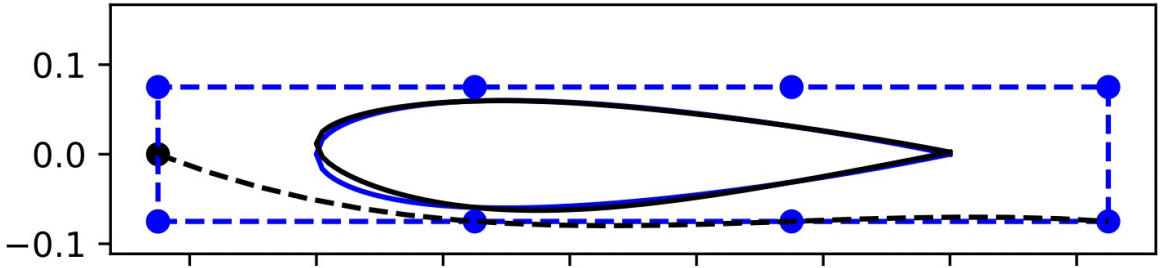
Optimize the spline nodes to achieve the design objectives.

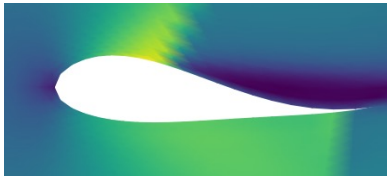
Pipe: optimize the upper (and lower) boundary



Airfoil: optimize the bounding box

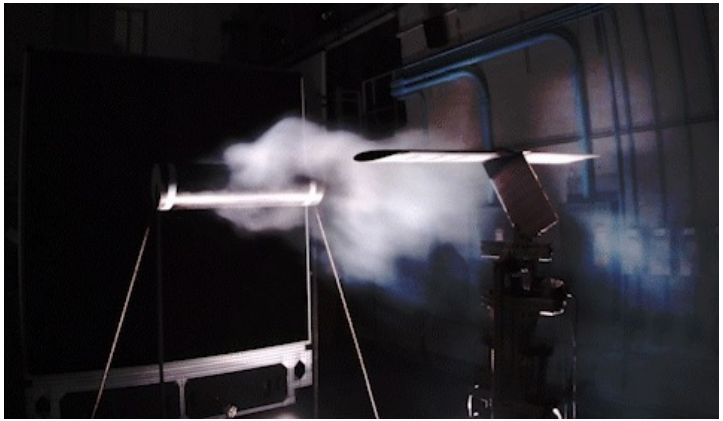


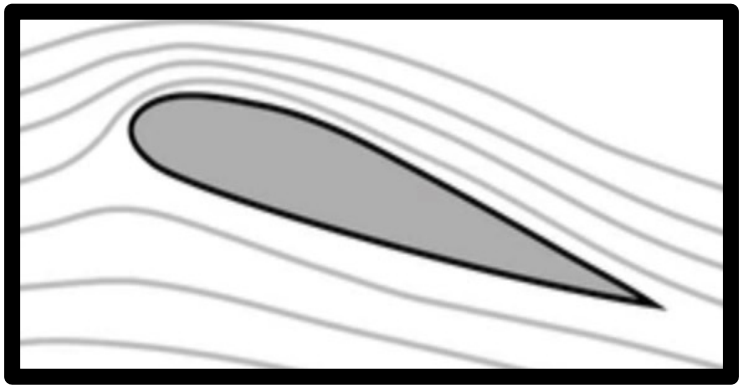



# PDE-observer

Yuanyuan Shi, Zongyi Li, Huan Yu, Drew Steeves,
Anima Anandkumar, and Miroslav Krstic

# MOTIVATING PROBLEM IN ROBOTICS



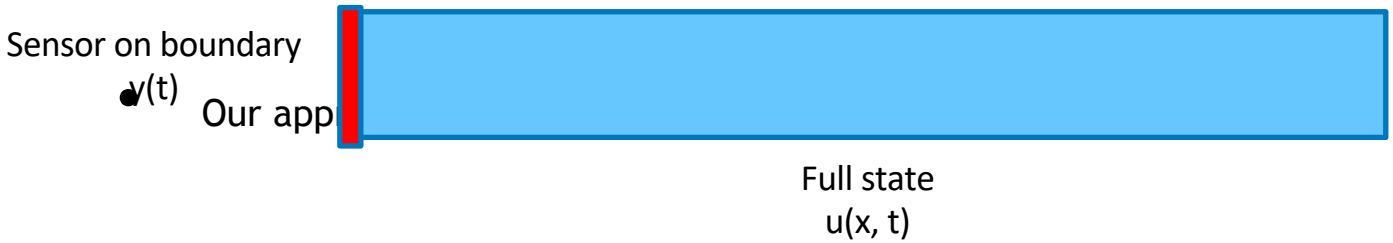


Given sensors placed on the body of airfoil/drone (boundary),
Can we estimate the velocity field (interior) for more accurate control?

Yuanyuan Shi, Zongyi Li, Huan Yu, Drew Steeves, Anima Anandkumar, and Miroslav Krstic

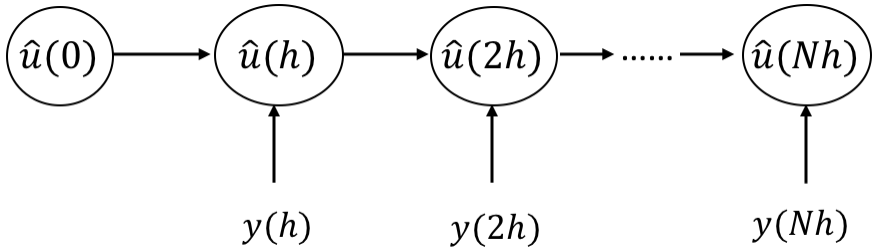
# STATE ESTIMATION THROUGH PDE OBSERVER

- Given the partial observation, estimate the state.
- PDE for state estimation: Back-stepping PDE observer.
- The estimate converges to the truth (exponentially)
- Slow and not real-time.

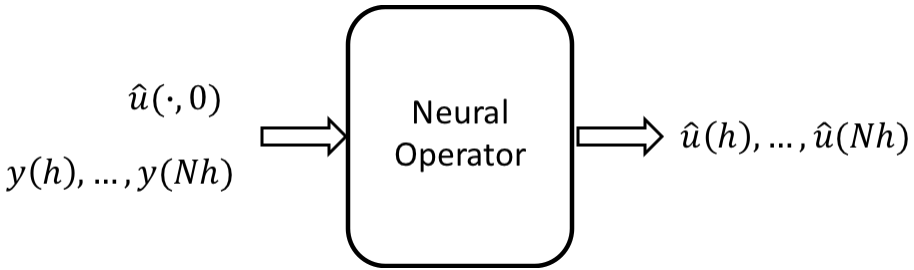


# Neural observer with FNO

Recurrent observer: make the estimation at each step recursively

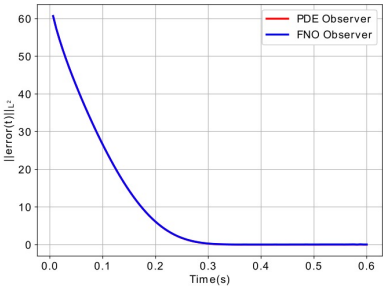



Feedforward observer: make the estimate over a time interval each step.



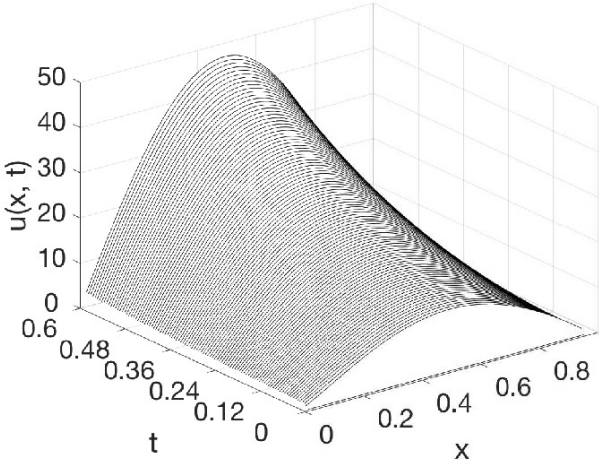

Yuanyuan Shi, Zongyi Li, Huan Yu, Drew Steeves, Anima Anandkumar, and Miroslav Krstic

# REACTION-DIFFUSION PDE
# PRESCRIBED-TIME OBSERVER

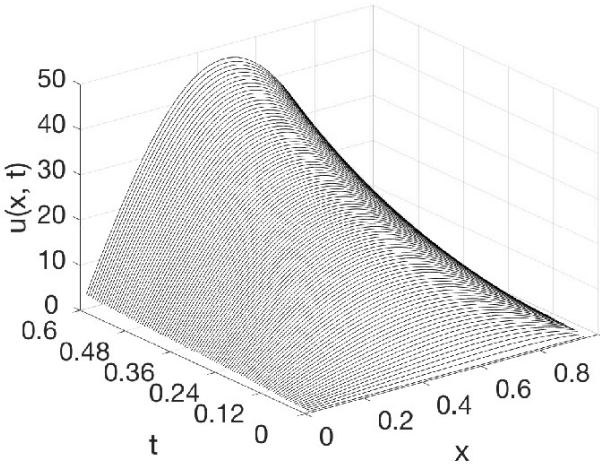



Same convergence rate

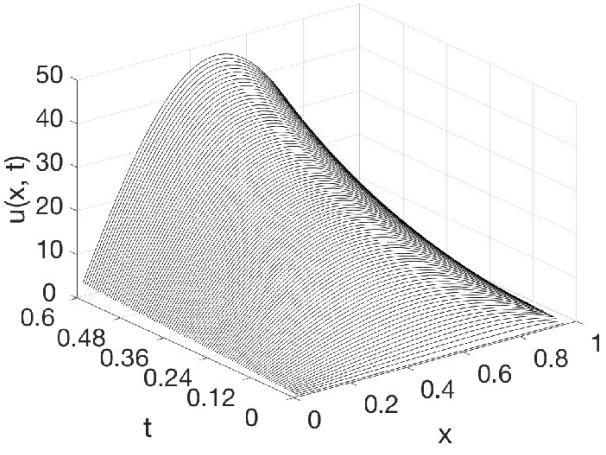
Chemical tubular reactor system:
Learn a time-dependent, fast-converging observer.




True system




Conventional PT Observer




FNO Neural Observer

**2000x** Speed up

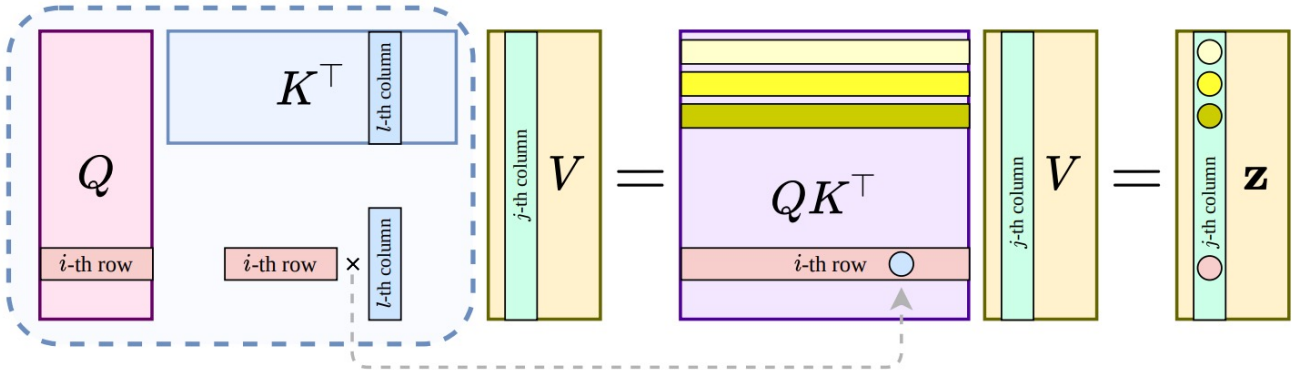
# FNO-transformer

John Guibas , Morteza Mardani , Zongyi Li ,
Andrew Tao, Anima Aanandkumar, Bryan Catanzaro

# Fourier-based Transformer

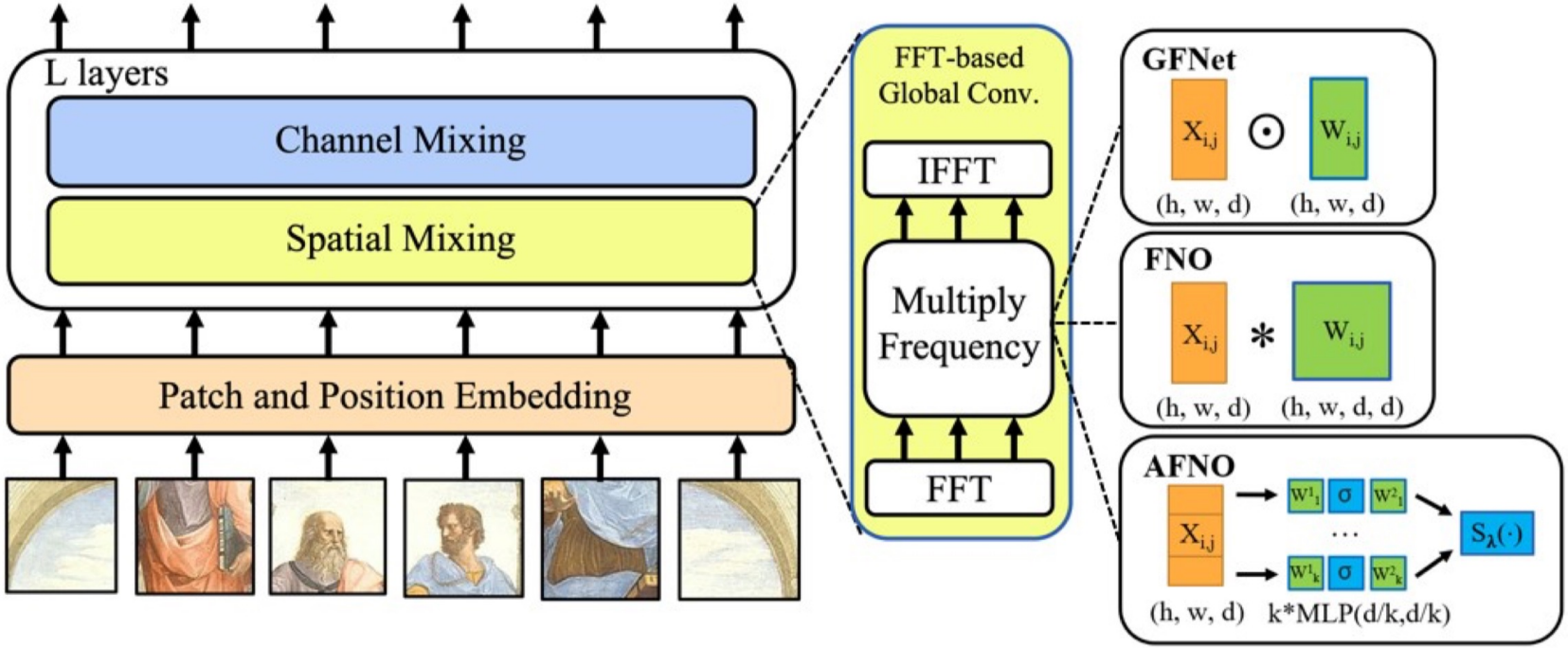
Neural operator can be viewed as a continuous generalization of Transformers

- Attention mechanism is a kernel-integration.
- Query-key can be viewed a a low-rank decomposition of the kernel.
- Replace the kernel by Fourier transform.



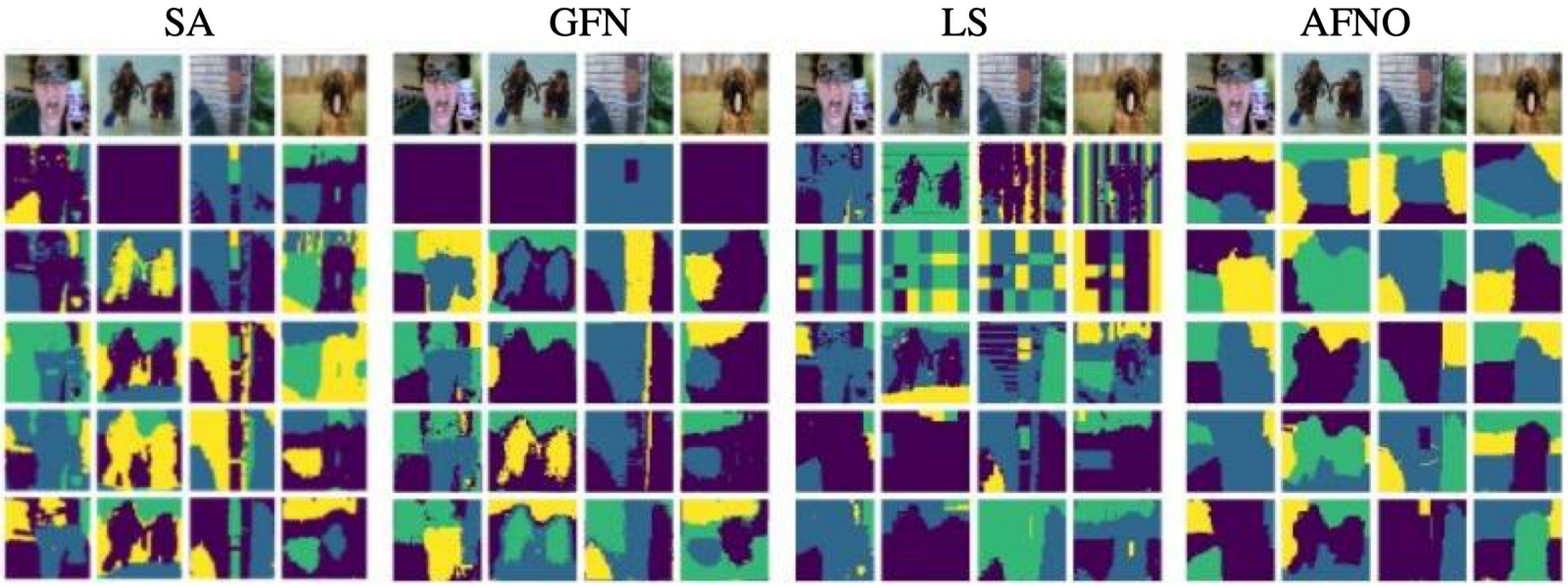

Shuhao Cao. Choose a Transformer: Fourier or Galerkin

# Fourier-based Transformers




Lee-Thorp et. al. FNet: Mixing Tokens with Fourier Transforms
Rao et. al. GFNet: Global Filter Networks for Image Classification

| Backbone | Mixer | Params | GFLOPs | Latency(sec) | SSIM | PSNR(dB) |
|---|---|---|---|---|---|---|
| ViT-B/4 | Self-Attention | 87M | 357.2 | 1.2 | **0.931** | **27.06** |
| ViT-B/4 | LS | 87M | 274.2 | 1.4 | 0.920 | 26.18 |
| ViT-B/4 | GFN | 87M | 177.8 | 0.7 | 0.928 | 26.76 |
| ViT-B/4 | AFNO (ours) | 87M | 257.2 | 0.8 | **0.931** | 27.05 |



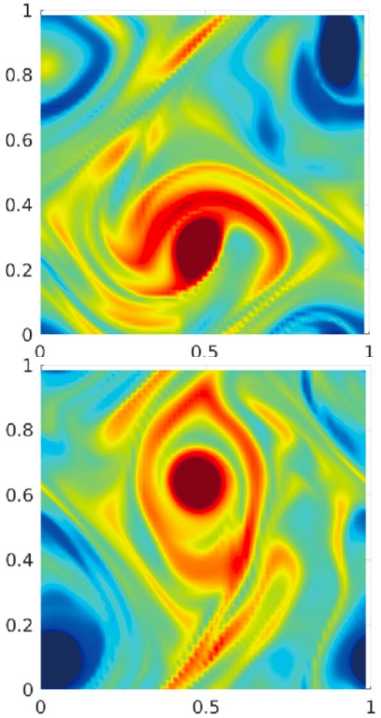
# Chaotic system

Miguel Liu-Schiaffini, Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, Anima Anandkumar

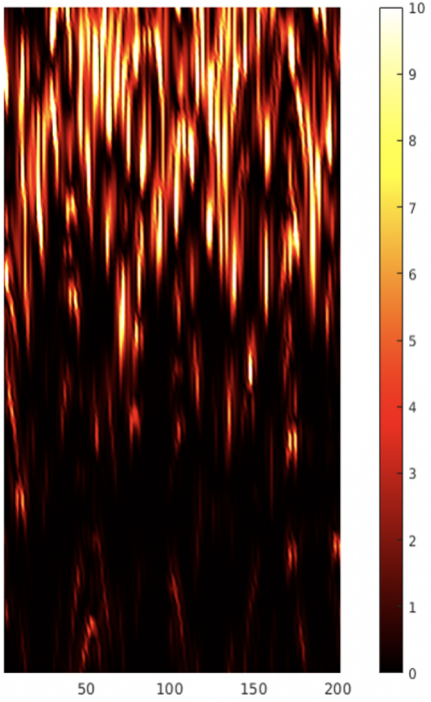
# Chaotic system

Chaotic systems are intrinsically unstable. Smaller errors will accumulate and make the simulation diverge from the truth.

Can we predict long-time trajectories that, while eventually diverging from the truth, still preserve the same orbit (**attractor**) of the system and its **statistical properties**?




Navier-Stokes




Kuramoto-Sivashinsky

# Semigroup and Markov neural operator

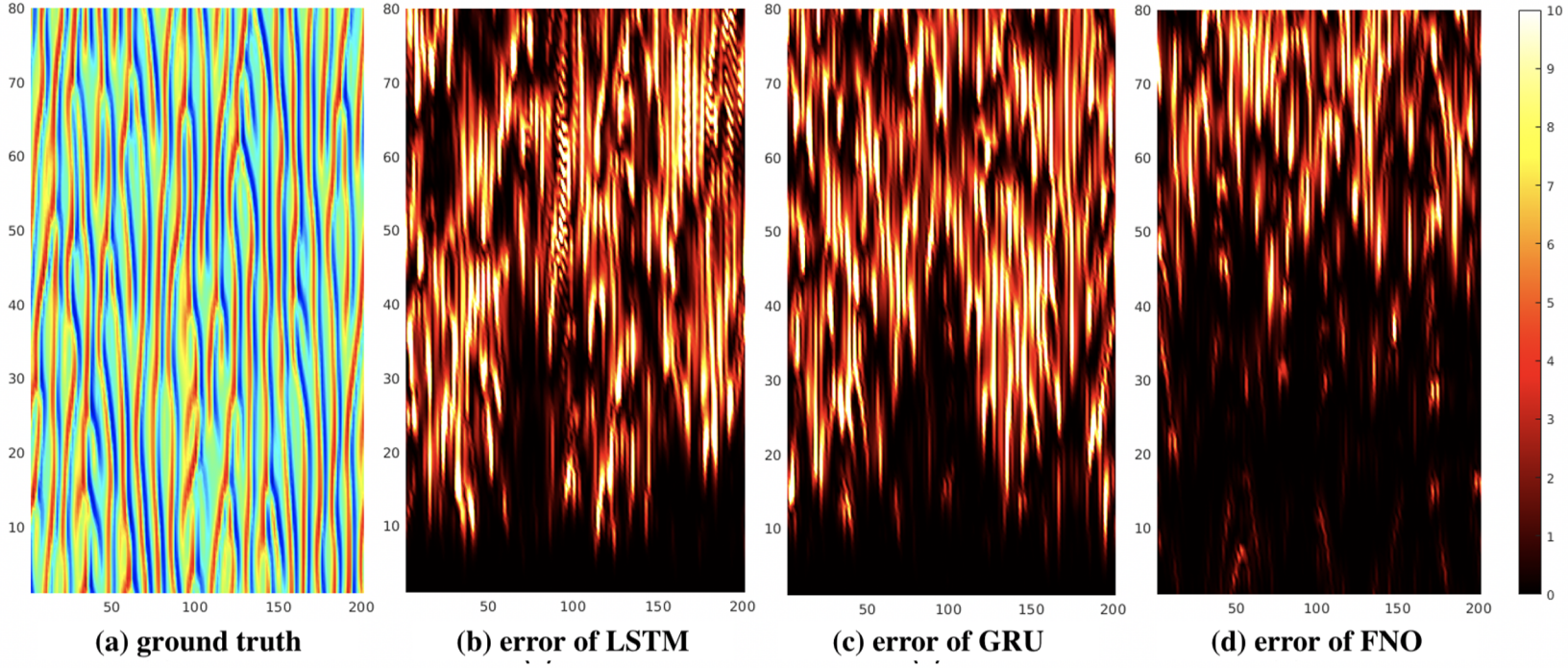
Use Markov neural operator to model the local evolution.

$$u(t) = S_t u(0)$$

Compose the operator as a semigroup

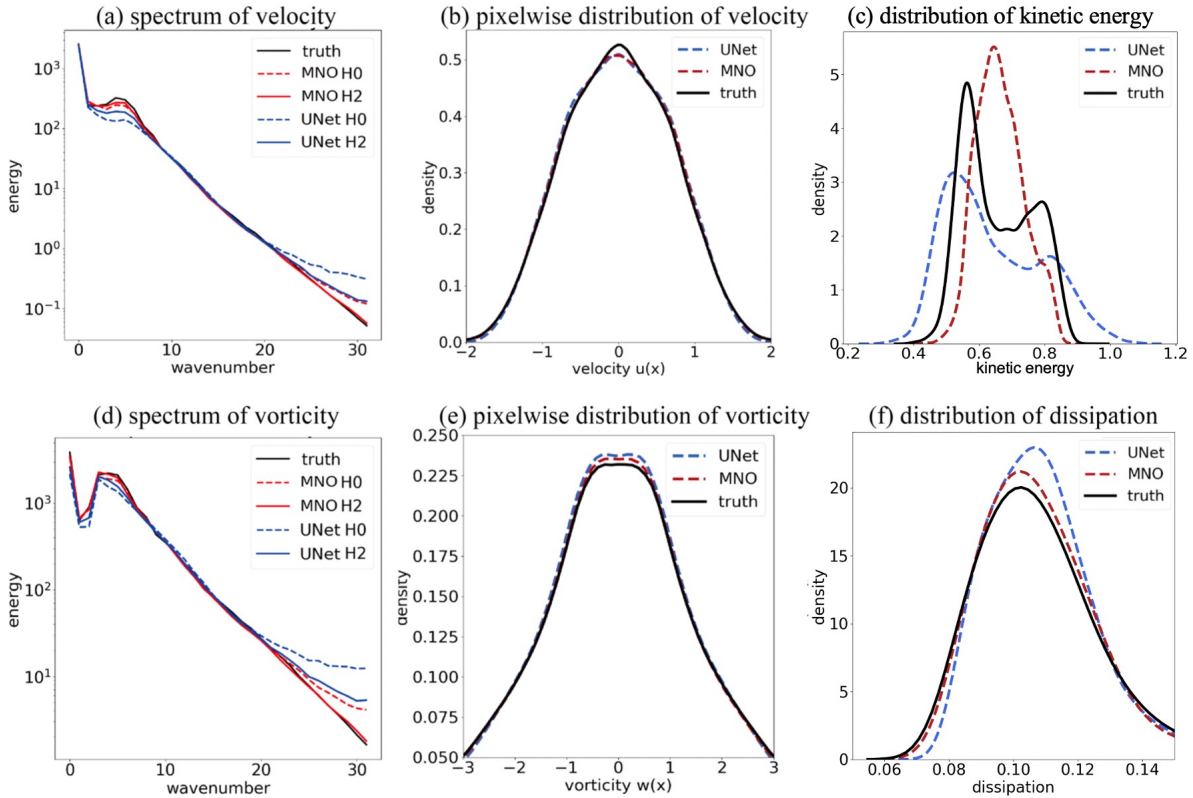
$$u(n \cdot h) \approx \hat{S}_h^n(u_0) := \underbrace{(\hat{S}_h \circ \cdots \circ \hat{S}_h)}_{n \text{ times}}(u_0)$$

# KS equation

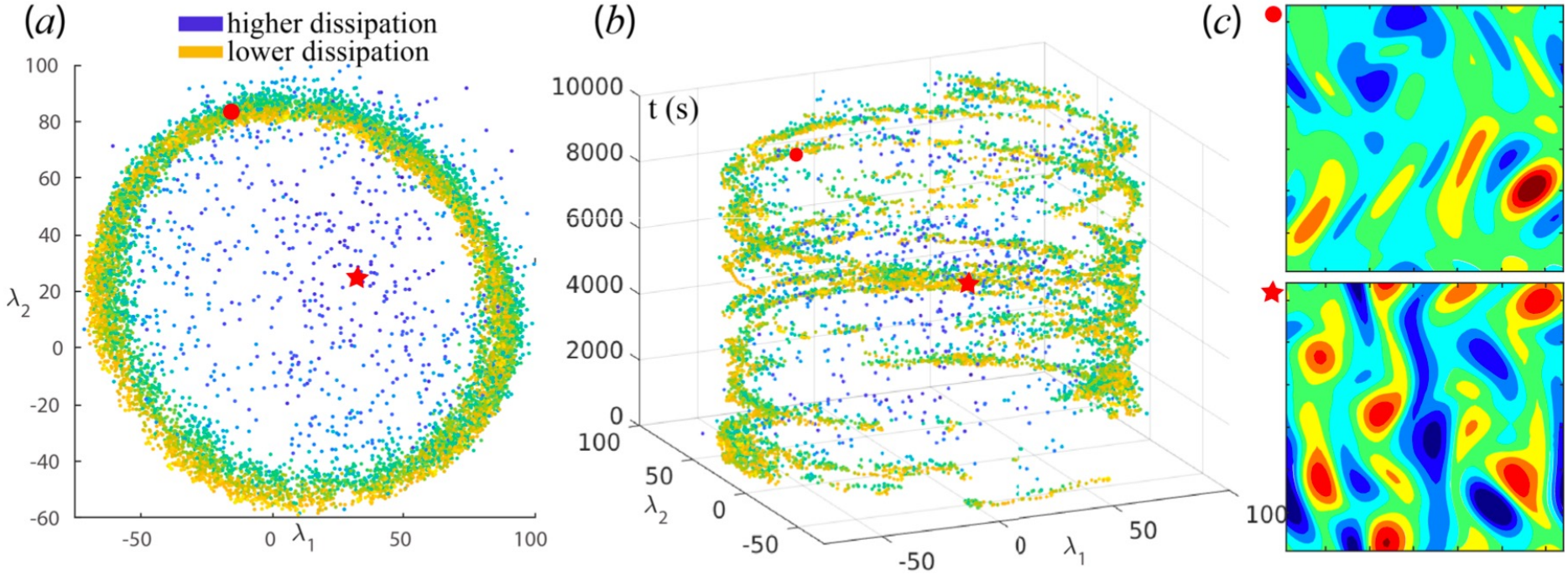



FNO captures the trajectory of chaotic systems for a longer period compared to RNNs.

# Markov neural operator captures the invariant statistics



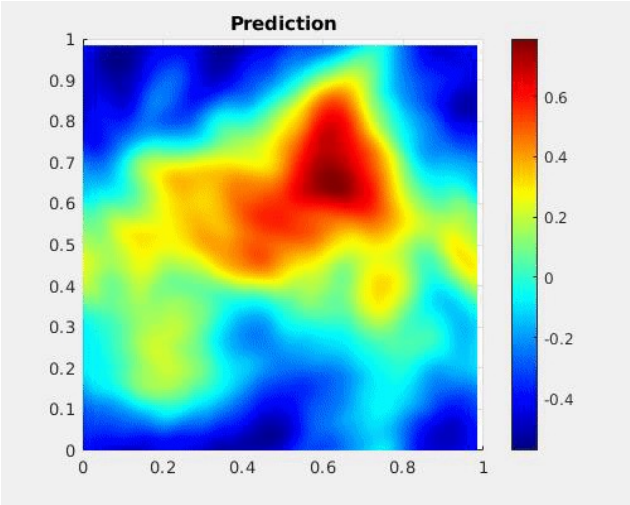
# Markov neural operator simulates the global attractor



# Enforcing dissipativity: Motivations

Many chaotic systems have the **dissipative** property, which pushes the dynamics back to the attractor.

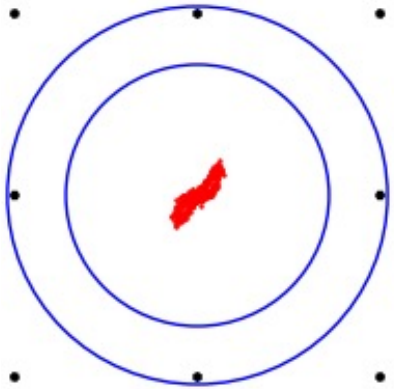
Without such dissipativity, the model is prone to blow up.




Accelerating flow

# Methods for Enforcing dissipativity

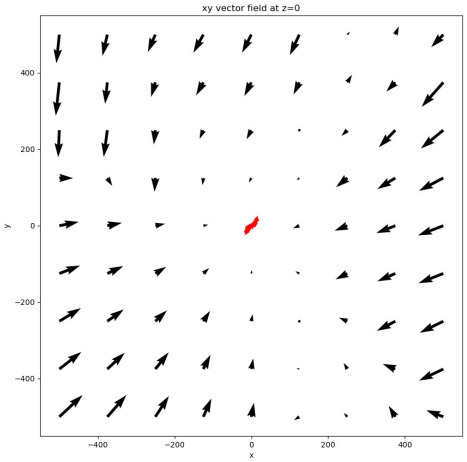
Enforce dissipativity by augmenting virtual data points around the raw data (equivalent to adding a loss term)



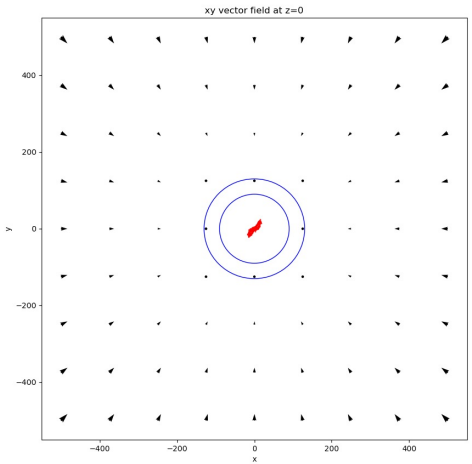
$$\mathcal{L}_{\text{data}} + \frac{\alpha}{2} \int \left| \Phi(x, \theta) - \lambda x \right|^2 \nu(dx)$$

Red: the raw data (attractor)
Blue shell: the region to add data points.

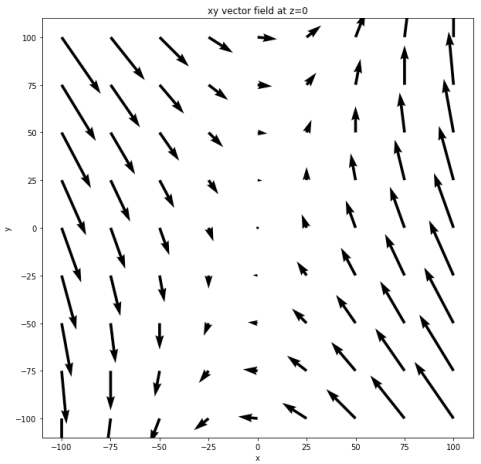
# Lorenz 63 learned flow maps




Before




After

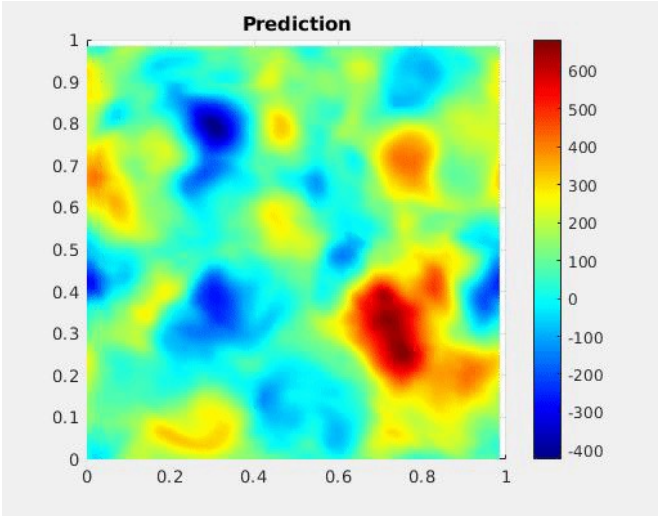



True system

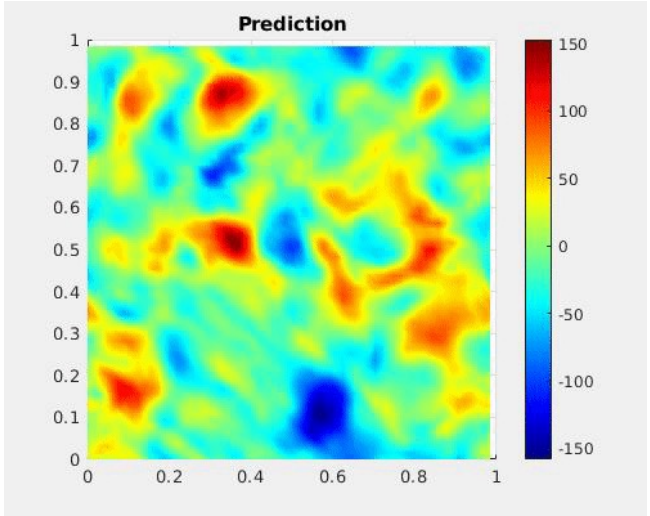
Red points are training points on attractor, blue circles represents shell where dissipativity is enforced.

# Kolmogorov flow with enforced dissipativity

Start with an initial condition outside the attractor



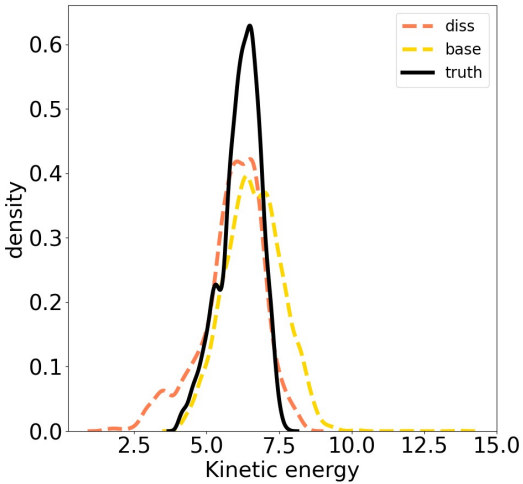

Before: quickly blowing-up (10^16)



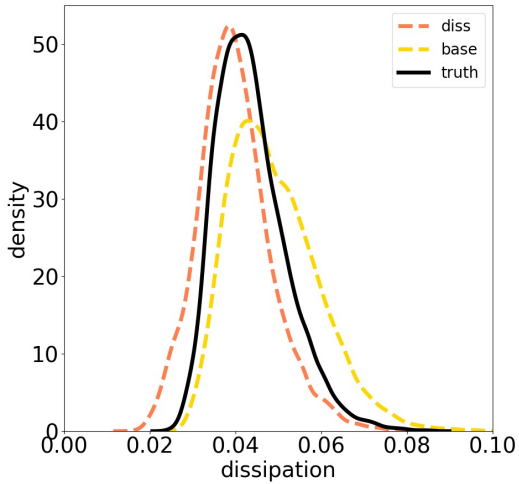

After: converge back to the attractor

# Kolmogorov flow with enforced dissipativity

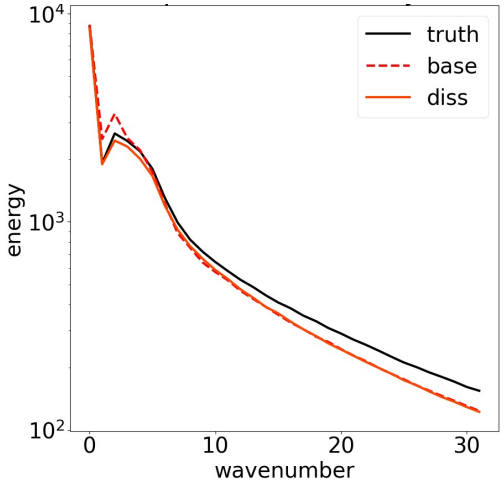
Distribution of kinetic energy




Distribution of dissipation




Spectrum of vorticity




The distribution of dissipation is improved

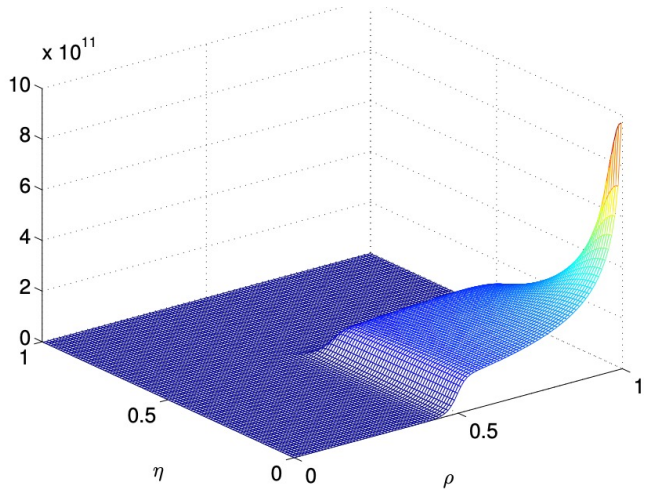
# Self-similar flow

Haydn, Zongyi Li, Yixuan Wang, Thomas Hou, Anima Anandkumar

# Blowup

The existence of the solution to the Navier-Stokes equation is one of the most interesting problems in PDE.

Can we construct a counterexample (blowup)?




G. Luo and T. Y. Hou (2014)

# Blowup

Idea: find a blowup scenario with a self-similar structure, such as **u**->inf as t->T.

$$u(t,x) = c_0(T-t)^{-c_1}U(x\cdot(T-t)^{c_2}).$$

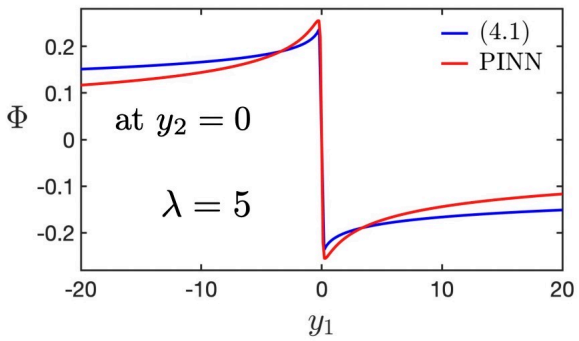
The original solution **u** is singular, but the profile function **U** is smooth.
Need to find the profile parameters c1 and c2 (inverse problem).

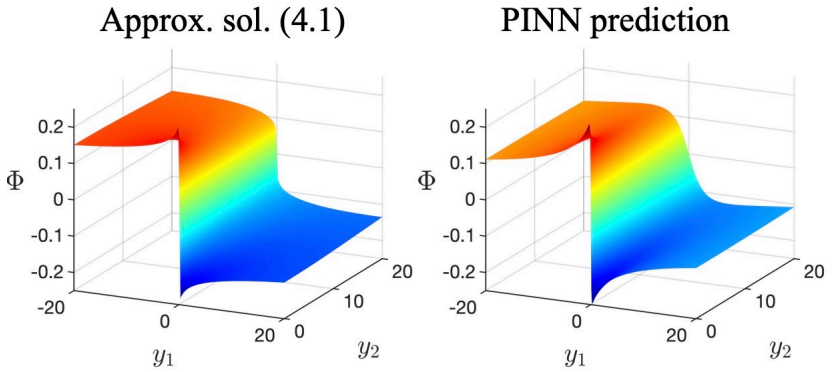
Y. Wang, C.-Y. Lai, J. G´omez-Serrano, and T. Buckmaster. Self-similar blow-up profile for the boussinesq equations via a physics-informed neural network.

# Blowup

We can use the physics-informed methods (PINN and PINO) to discover such profiles.

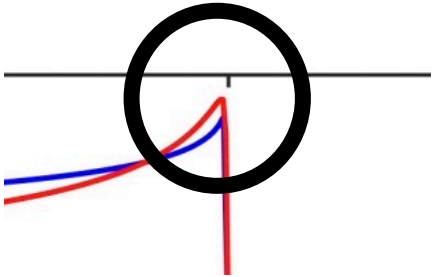
Pros: can find the profile parameters (inverse problem)
Cons: less accurate (10^-5) compared to the conventional solver (10^-10, adaptive FEM).
Since the Euler equation is highly sensitive, extreme accuracy is needed.



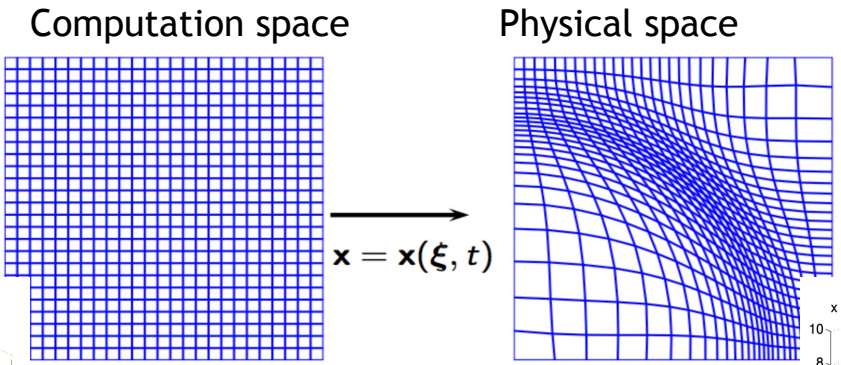

Y. Wang, C.-Y. Lai, J. G´omez-Serrano, and T. Buckmaster. Self-similar blow-up profile for the boussinesq equations via a physics-informed neural network.

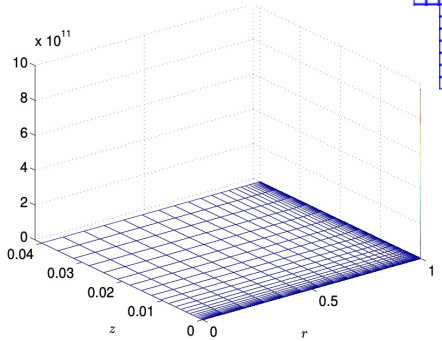
# Blowup

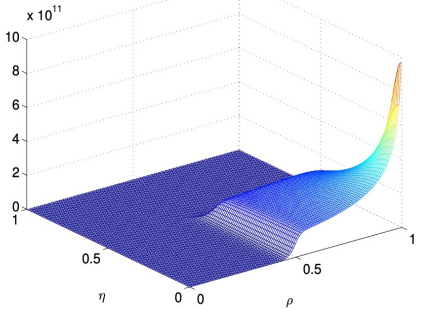


Use PINO to improve the optimization landscape:

- High frequency (singular) structures are challenging for neural networks
- Use the adaptive PINO (like the adaptive FEM) to better capture the singularity



# Reference

Arxiv:
https://arxiv.org/abs/2003.03485
https://arxiv.org/abs/2006.09535
https://arxiv.org/abs/2010.08895

Code:
https://github.com/zongyi-li/graph-pde
https://github.com/zongyi-li/fourier_neural_operator

Blog posts:
https://zongyi-li.github.io/blog/2020/graph-pde/
https://zongyi-li.github.io/blog/2020/fourier-pde/