

ML FOR SCIENCE · LECTURE 12

Operator Learning

Aparna Chandramowlishwaran · EECS 195/221 · Fall 2026

A CNN learns stencils for one grid

- **Recall**: SmallCNN trained at 64 x 64 and tested at 128 x 128 on the same frames: **the error grows**
- A 3 x 3 kernel is a **stencil for one cell size** h. On a grid with spacing $h/2$, the same kernel covers *half* the physical distance
- The finite-difference stencil for u_{xx} is $[1, -2, 1] / h^2$. The solver divides by h; a learned kernel has h **built into its weights**
- **The PDE has no grid.** With initial and boundary conditions it defines the **solution operator** $\mathcal{G}$, which maps *functions* to *functions*
- Every model so far learned $\mathcal{G}$ on one grid: an **array-to-array** map

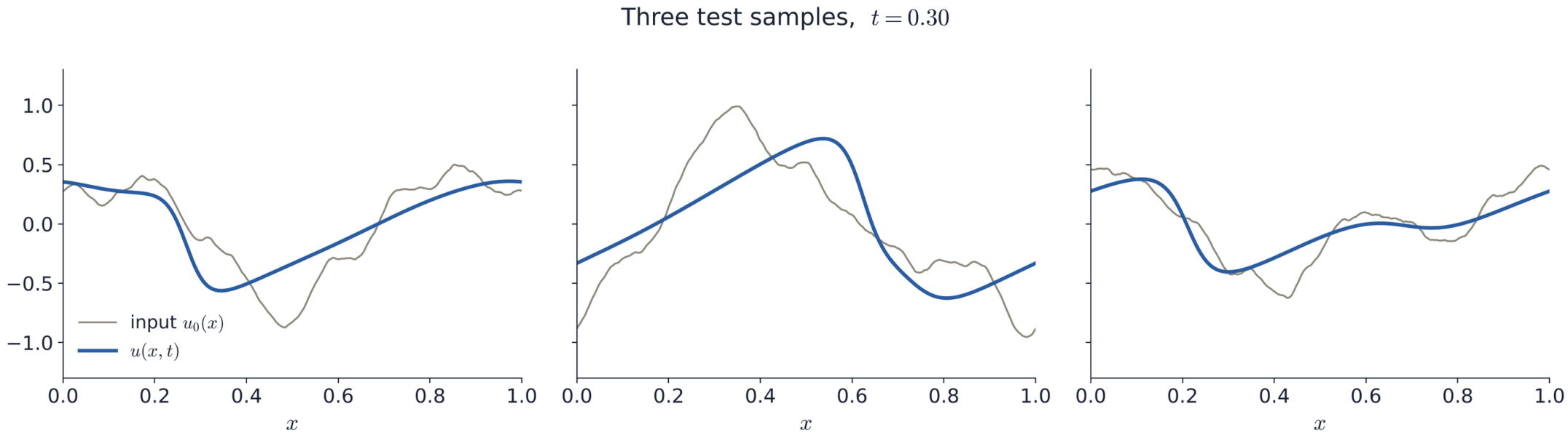
Today: learn the operator, so one set of weights works on any grid

Operators map functions to functions

- A *function* maps a point to a value: $u : \Omega \to \mathbb{R}$, $x \mapsto u(x)$. An **operator** maps a whole function to another function: $\mathcal{G} : a \mapsto u$
- Training data: **pairs** (a_i, u_i), $i = 1, \ldots, n$, each stored on *some* grid
- **Goal:** a model $\mathcal{G}_\theta$ that accepts a on **any grid** and returns u at **any point**

Problem	Input function a	Output function u
Darcy flow	permeability $a(x)$	pressure $u(x)$
Heat equation	initial temperature $u_0(x)$	temperature $u(x, t)$
Navier-Stokes	vorticity $\omega(x, t)$	vorticity $\omega(x, t + \Delta t)$
Pool boiling (BubbleML)	temperature, velocity, interface at time t	the same fields at later times

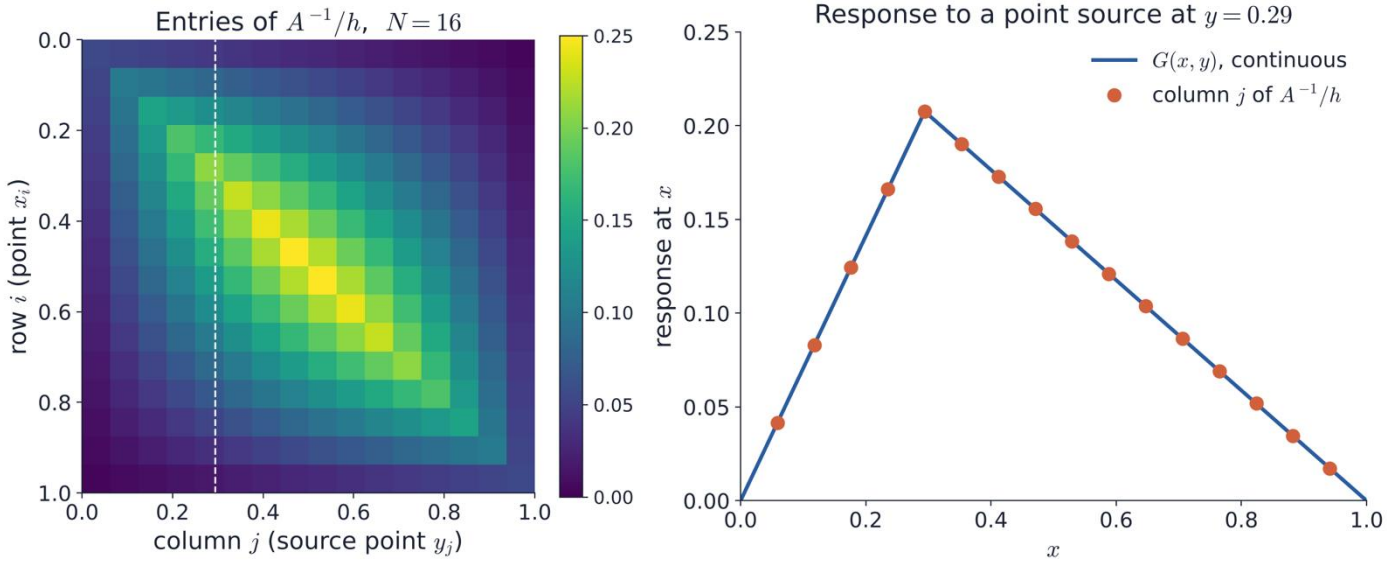
The Burgers solution operator

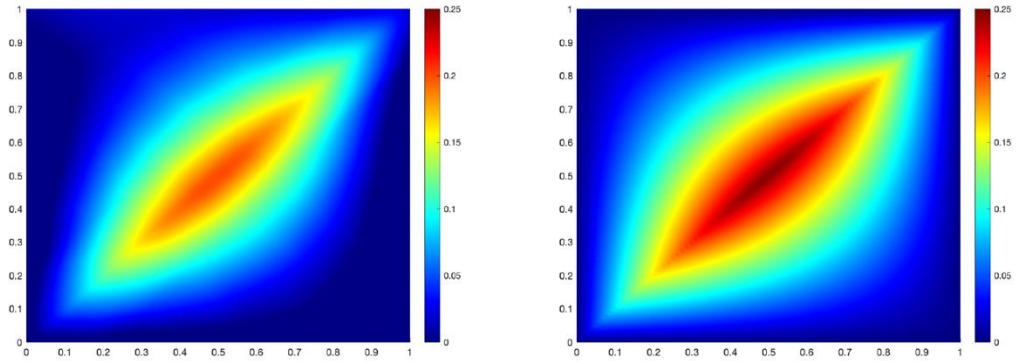


- Burgers equation: $u_t + u\,u_x = \nu\,u_{xx}$. The term $u\,u_x$ **steepens** the profile into fronts; $\nu\,u_{xx}$ **smooths** them
- **Input:** a random initial condition $u_0(x)$. **Output:** $u(x, t)$ at 50 times up to $t = 1$
- Operator $\mathcal{G} : u_0 \mapsto u(\cdot, t)$. **Fronts form** within $t \approx 0.2$, then move and decay

Linear operators are kernel integrals

- Discretizing $-u'' = f$ gives $A\,u = f$. A^{-1} is **dense**: u at each point depends on f everywhere
- As N grows, the entries of A^{-1}/h sample **one function** $G(x, y)$, the **Green's function**
- In the continuum: $u(x) = \int G(x, y)\, f(y)\, dy$. A^{-1} changes with N; ***G* does not**
- Bottom right: a neural operator trained on input–output pairs of this problem **learns a kernel close to *G*** (Kovachki et al.)

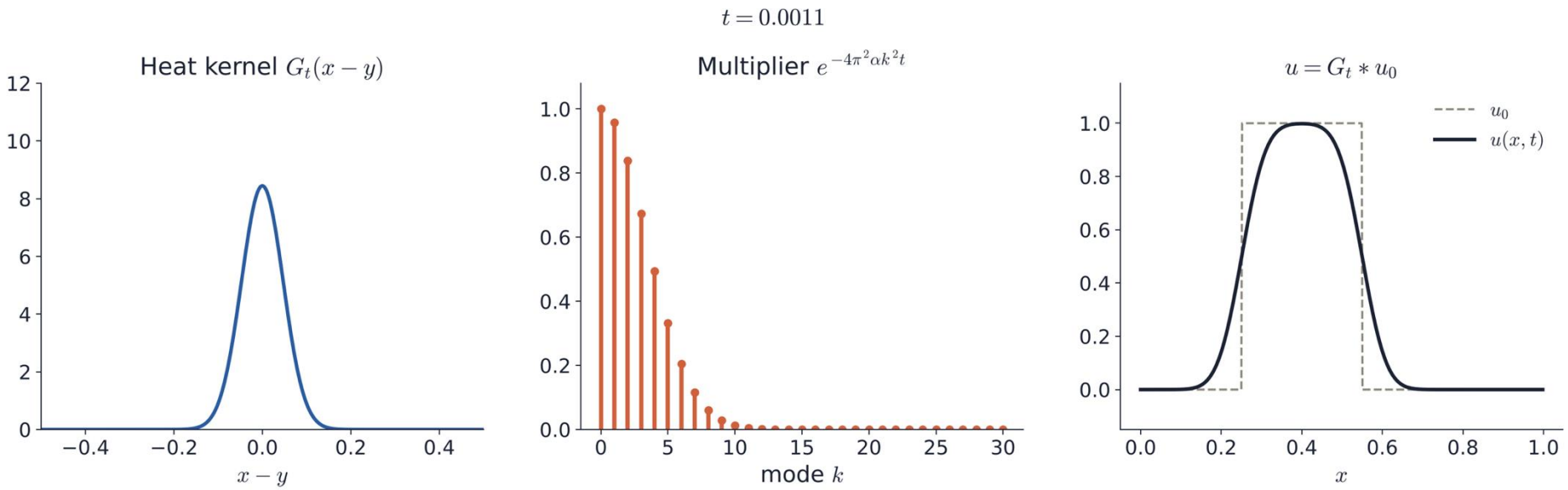




learned kernel (left), analytic G (right)

Bottom: Kovachki et al., JMLR 2023

The heat equation as a convolution



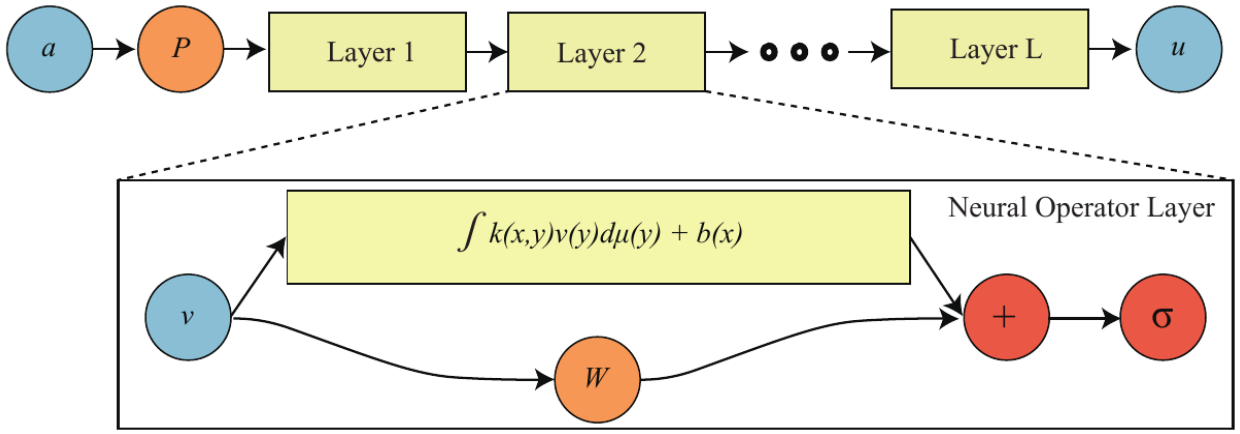
- Solution operator: $u(x,\, t) = \int G_t(x-y)\, u_0(y)\, dy$, a **convolution**: the kernel depends only on $x - y$
- In Fourier space a convolution is a **product**: mode k of u_0 is multiplied by $\exp(-4\pi^2\alpha k^2 t)$
- **High modes decay first** ($\exp(-4\pi^2\alpha k^2 t) \ll 1$ for large k), so a *few low modes* describe the solution

Convolution in x is multiplication in k

A neural operator layer

$$v_{l+1}(x) = \sigma\left(W\, v_l(x) + \int_\Omega \kappa_\theta(x, y)\, v_l(y)\, dy + b\right)$$

- $v_l(x)$: a vector of d_v channels at every point x, the **hidden state** (like the channels of a CNN)
- $W\, v_l(x)$: a **pointwise linear map**, the same at every point (a 1 x 1 convolution)
- The integral $\int_\Omega \kappa_\theta(x, y)\, v_l(y)\, dy$ mixes information across the **entire domain** Ω with a **learned kernel** κ_θ
- **Lift** P maps the input to d_v channels at each point; **project** Q maps back to the output
- **Every part of the layer acts on functions.** The grid enters only when we compute the integral



Equation and figure: Kovachki, Li, Liu, Azizzadenesheli, Bhattacharya, Stuart & Anandkumar, Neural Operator, JMLR 2023

Four ways to compute the kernel integral

- A direct sum over all point pairs costs **$O(N^2)$ per layer**. Each model family **restricts** κ to make it cheaper

Model	Restriction on $\kappa(x, y)$	Cost per layer
Graph neural operator	nonzero only for y near x; sum over neighbors	$O(N \times \text{neighbors})$
Low-rank (DeepONet-like)	$\kappa = \sum_r \varphi_r(x)\, \psi_r(y)$, rank r	$O(N\, r)$
Fourier neural operator	$\kappa(x - y)$, only $k \le k_{\max}$ modes	$O(N \log N)$
Attention	κ computed from $v(x)$ and $v(y)$	$O(N^2)$; $O(N)$ if linear

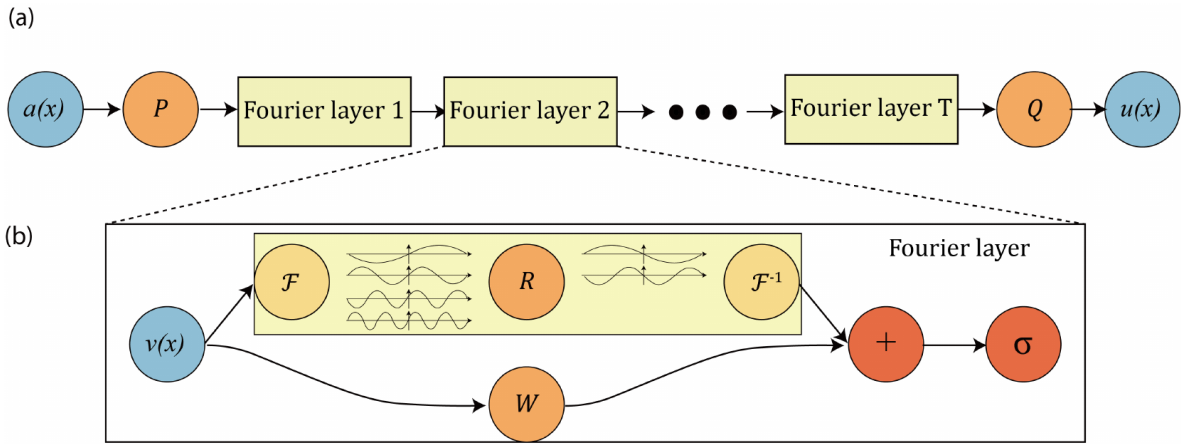
Lecture 9: attention is a learned, input-dependent kernel

Li et al., Neural operator: graph kernel network, 2020. Li et al., Fourier neural operator, ICLR 2021. Kovachki et al., JMLR 2023. Cao, Galerkin transformer, NeurIPS 2021

Fourier neural operator (FNO)

$$(\mathcal{K}v)(x) = \mathcal{F}^{-1}\big(R_k\,(\mathcal{F}v)_k\big)(x), \qquad |k| \le k_{\max}$$

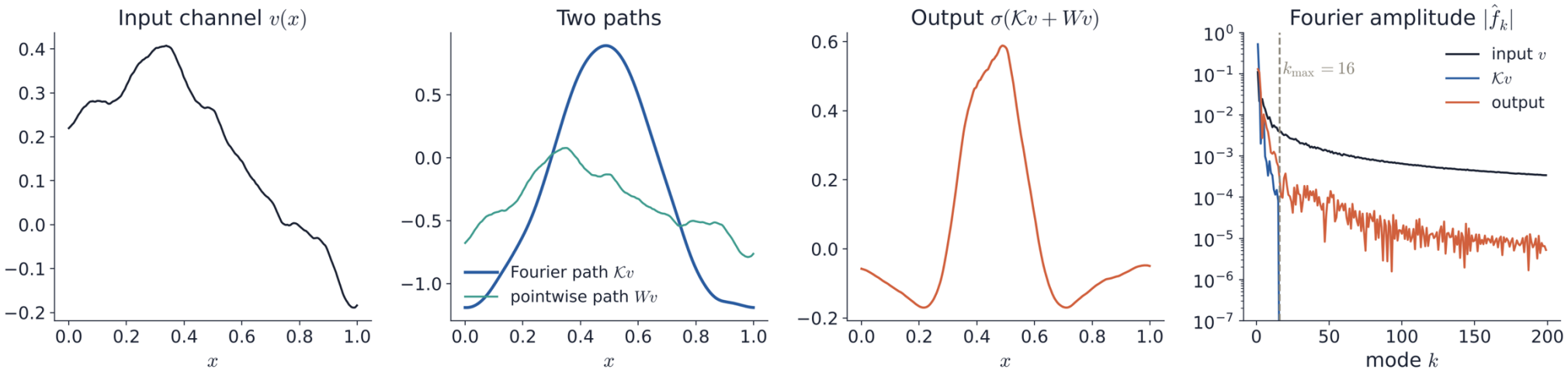
- With $\kappa(x, y) = \kappa(x - y)$ the integral is a **convolution**, $\int \kappa(x - y)\, v(y)\, dy$: **FFT** $\mathcal{F}$, **multiply**, **inverse FFT**
- R_k: a learned $d_v \times d_v$ **complex matrix** for each kept mode k. It mixes channels, *one mode at a time*
- Modes above $k_{\max}$ are **set to zero**. The number of weights, $d_v^2\, k_{\max}$ per layer, **does not depend on N**
- Add the pointwise path $W\, v(x)$, then apply σ at each point: $v_{l+1} = \sigma(\mathcal{K} v_l + W\, v_l)$



Weights are defined on Fourier modes, so they are independent of the grid

Figure: Li, Kovachki, Azizzadenesheli, Liu, Bhattacharya, Stuart & Anandkumar, Fourier Neural Operator for Parametric PDEs, ICLR 2021

One Fourier layer on one channel

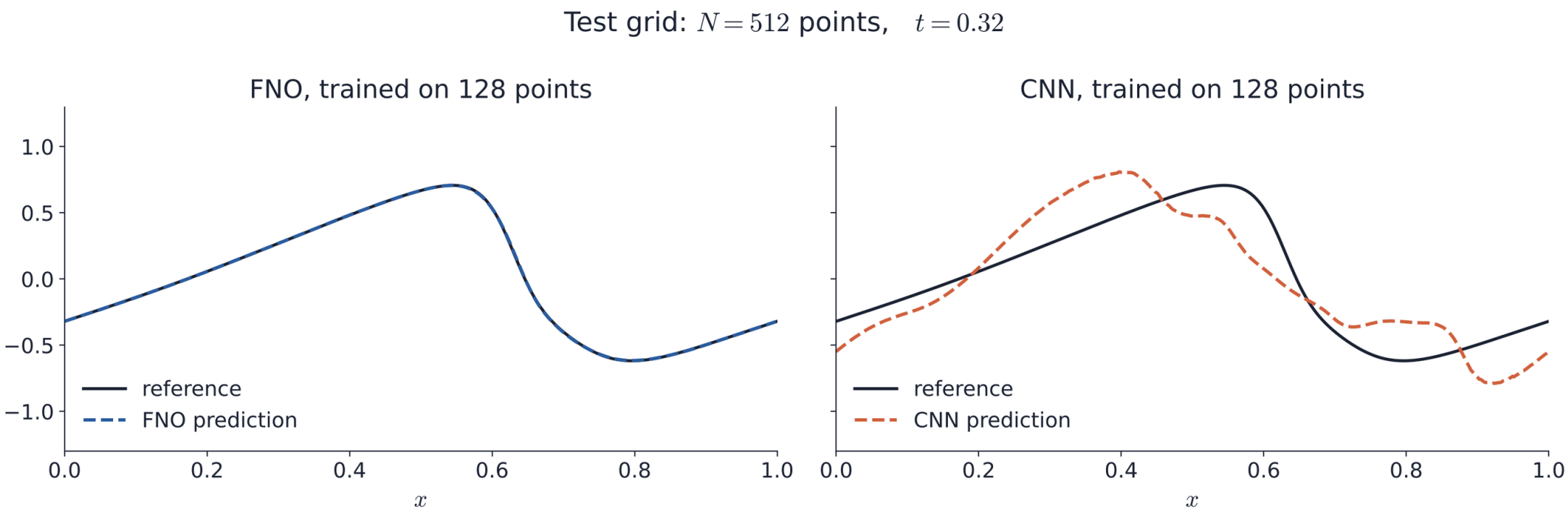


- The **Fourier path** returns a smooth function with modes $k \leq 16$. The **pointwise path** $W\,v$ keeps *every* mode of v
- σ acts at each point and **creates modes above $k_{\max}$**. Products of modes k and k' create $k + k'$
- Colagrande et al. (2026), a full FNO: hidden layers keep mostly **low modes**; the output's **high modes** come from the last nonlinear and projection steps

Fine-scale detail comes from W and σ, not from R_k

Our FNO from the notebook (width 64, k_{max} = 16, 4 layers, GELU), first layer, one test input at N = 1024
Colagrande, Caillon, Feillet & Allauzen, ICLR 2026 AI&PDE workshop

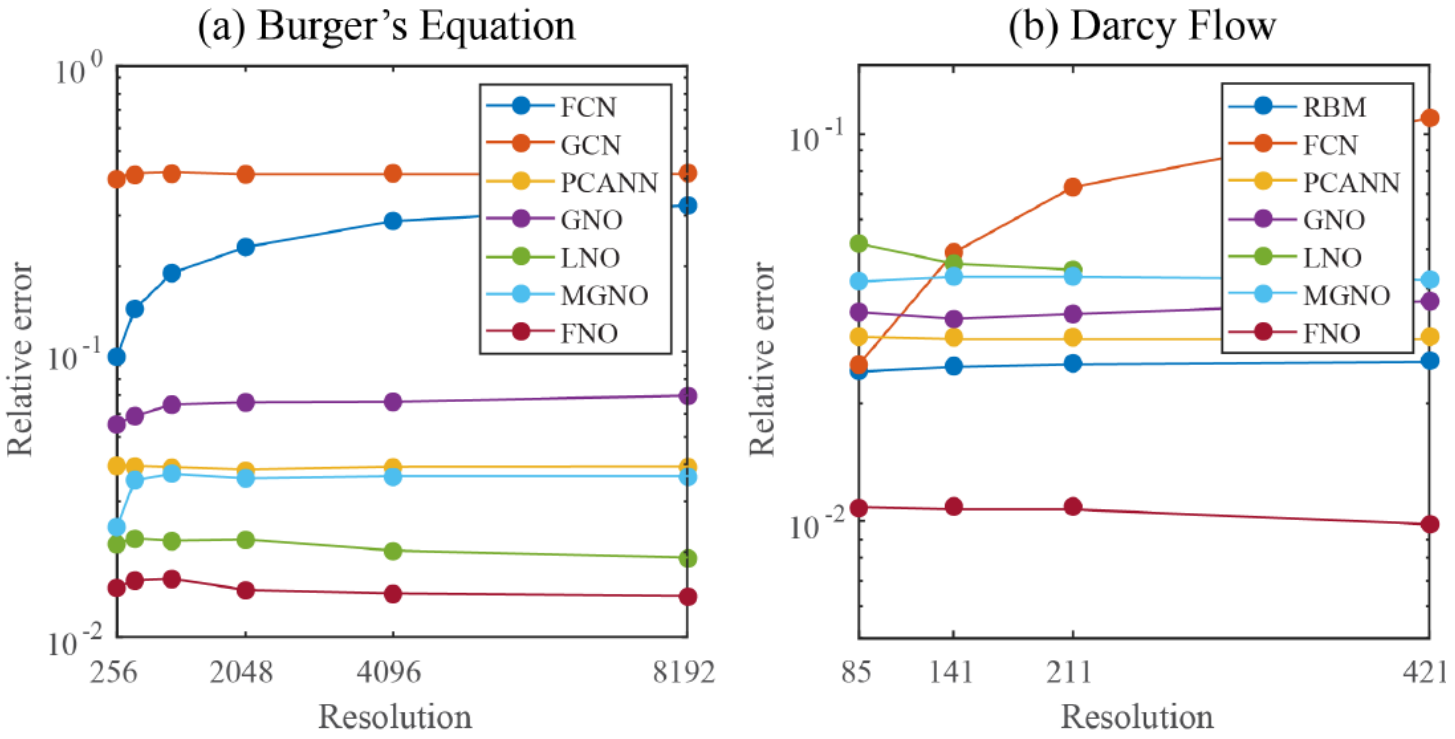
One set of weights on many grids



- **FNO**: the FFT of any grid gives the first $k_{\max}$ modes, so the **same R_k apply**. The coordinate $x \in [0, 1)$ is an input channel
- **CNN**: its kernels are stencils for $h = 1/128$. On other grids **the prediction breaks down**

Same test sample, prediction at t = 0.32. Both models trained on N = 128 for 300 epochs

Error at different resolutions

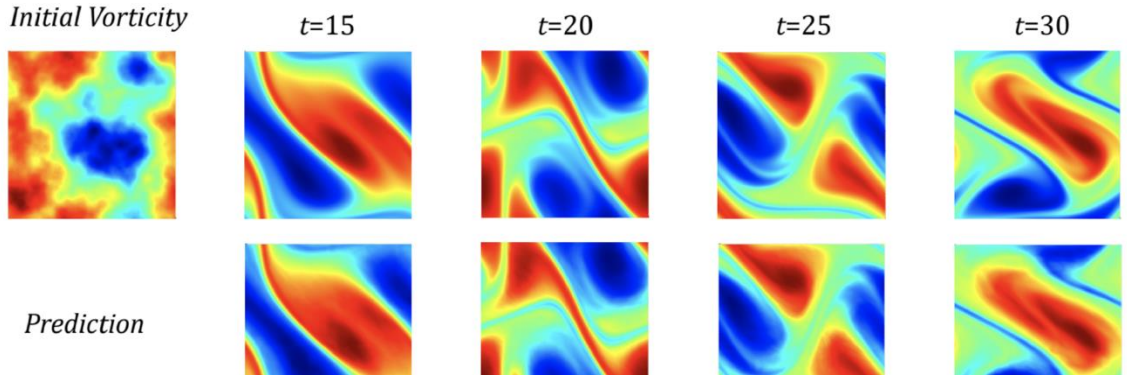


- Each model **trained and tested at the same resolution**: 256 to 8192 points for Burgers, 85² to 421² for Darcy
- **FNO**: about the **same error at every resolution**, and the lowest of all models
- **FCN**, a fully convolutional network: **error grows** as the grid gets finer. Its kernels cover a smaller physical region
- Graph and low-rank operators (GNO, LNO, MGNO) are also **flat**, at higher error

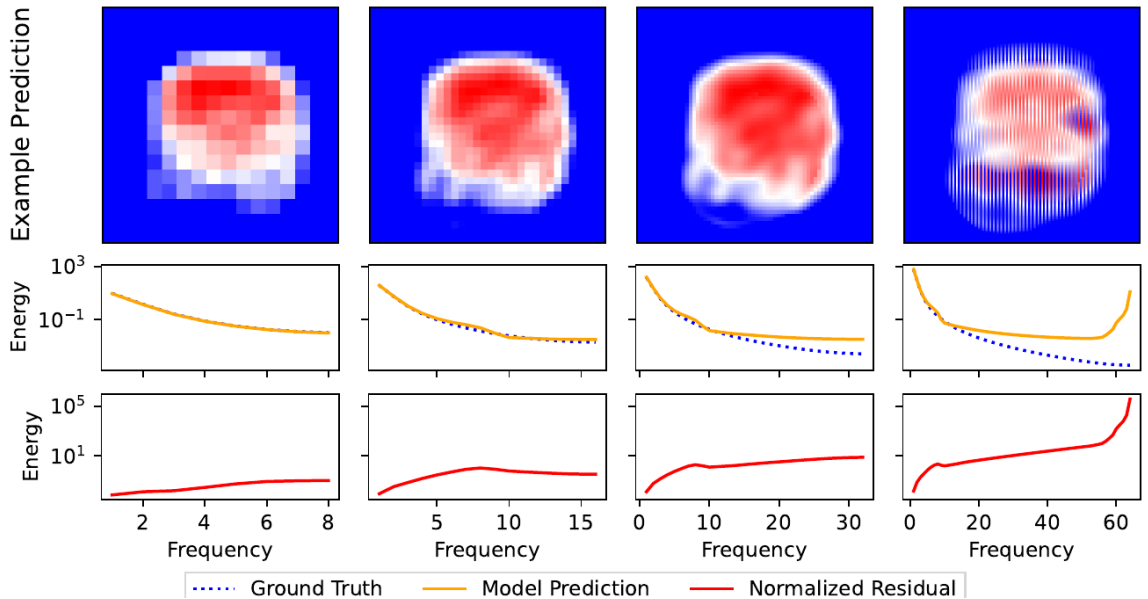
Operator error does not depend on the grid. CNN error does.

Li et al., ICLR 2021. PCANN: PCA + network; RBM: reduced basis method; GCN: graph convolutional network

Zero-shot super-resolution



- **Claim** (Li et al., top left): FNO trained on 64 x 64 Navier-Stokes, evaluated on 256 x 256. Top: truth; bottom: prediction
- **Test** (Sakarvadia et al., right): FNO trained on 16 x 16 Darcy, evaluated at 16, 32, 64, 128. **Stripes at 128**; the error grows at *high frequencies*
- A finer grid can contain **frequencies absent from training**. The model cannot extrapolate to them, and σ **aliases** them onto low modes



- It works when the finer data add **no new frequencies** (smooth outputs, our Burgers example)

Test a new resolution on solver data at that resolution

Top left: Li et al., ICLR 2021, Right: Sakarvadia et al, The false promise of zero-shot super-resolution in machine-learned operators, ICLR 2026

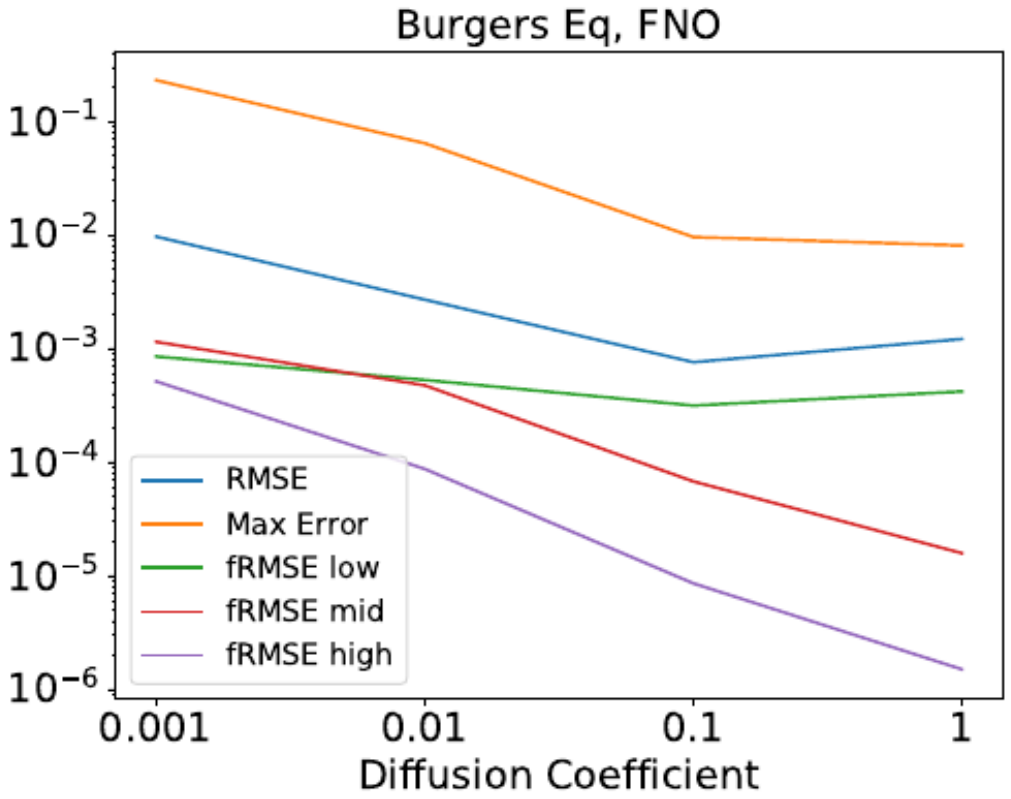
Making operators robust to resolution

- **Discretization convergence**: as the grid is refined, the outputs converge to one function (Kovachki et al. 2023). An FNO satisfies it *only approximately*
- **Aliasing**: σ at grid points creates modes above $N/2$, the highest the grid can represent. Mode $N/2 + j$ folds back onto $N/2 - j$, so the output depends on N
- Colagrande et al. (2026): running the FNO at its **training resolution** and **upsampling the output** (Fourier zero-padding) can beat running it on the fine grid
- **Remedies**
  - **Multi-resolution training**: mostly cheap low-resolution data plus a small share of high-resolution data (Sakarvadia et al. 2026)
  - **Alias-free layers** that act on band-limited functions: ReNO, CNO (Bartolucci et al. 2023; Raonić et al. 2023)

Train on the resolutions you will use, and compare against upsampling

Bartolucci et al., Representation equivalent neural operators, NeurIPS 2023. Raonić et al., Convolutional neural operators, NeurIPS 2023. Colagrande et al., ICLR 2026 AI&PDE workshop

Sharp fronts and mode cutoff



- **Recall**: a jump needs **many Fourier modes**; a fixed number of modes results in **Gibbs** phenomenon
- PDEBench (left): FNO error on Burgers grows as the viscosity ν **decreases** and fronts sharpen
- The **high-frequency error** (fRMSE high) grows the most; the low-frequency error barely changes
- **Bubble interfaces** in boiling are the same hard case
- **Remedies:** more modes, a U-Net path for local detail, or a local kernel

FNO errors concentrate at the sharpest features

Takamoto et al., PDEBench, NeurIPS 2022

DeepONet branch and trunk networks

$$\mathcal{G}_\theta(u_0)(y) = \sum_{k=1}^{p} b_k(u_0)\,\tau_k(y) + b_0$$

- **Branch** net: takes u_0 at m fixed sensors x_1, …, x_m; returns p coefficients $b_k(u_0)$
- **Trunk** net: takes a query point y; returns p functions $\tau_k(y)$
- **Output:** their dot product, $\sum_k b_k(u_0)\,\tau_k(y)$, at **any** y
- Inputs must use the **same m sensors** in training and testing
- Based on the **universal approximation theorem for operators** (Chen & Chen 1995)

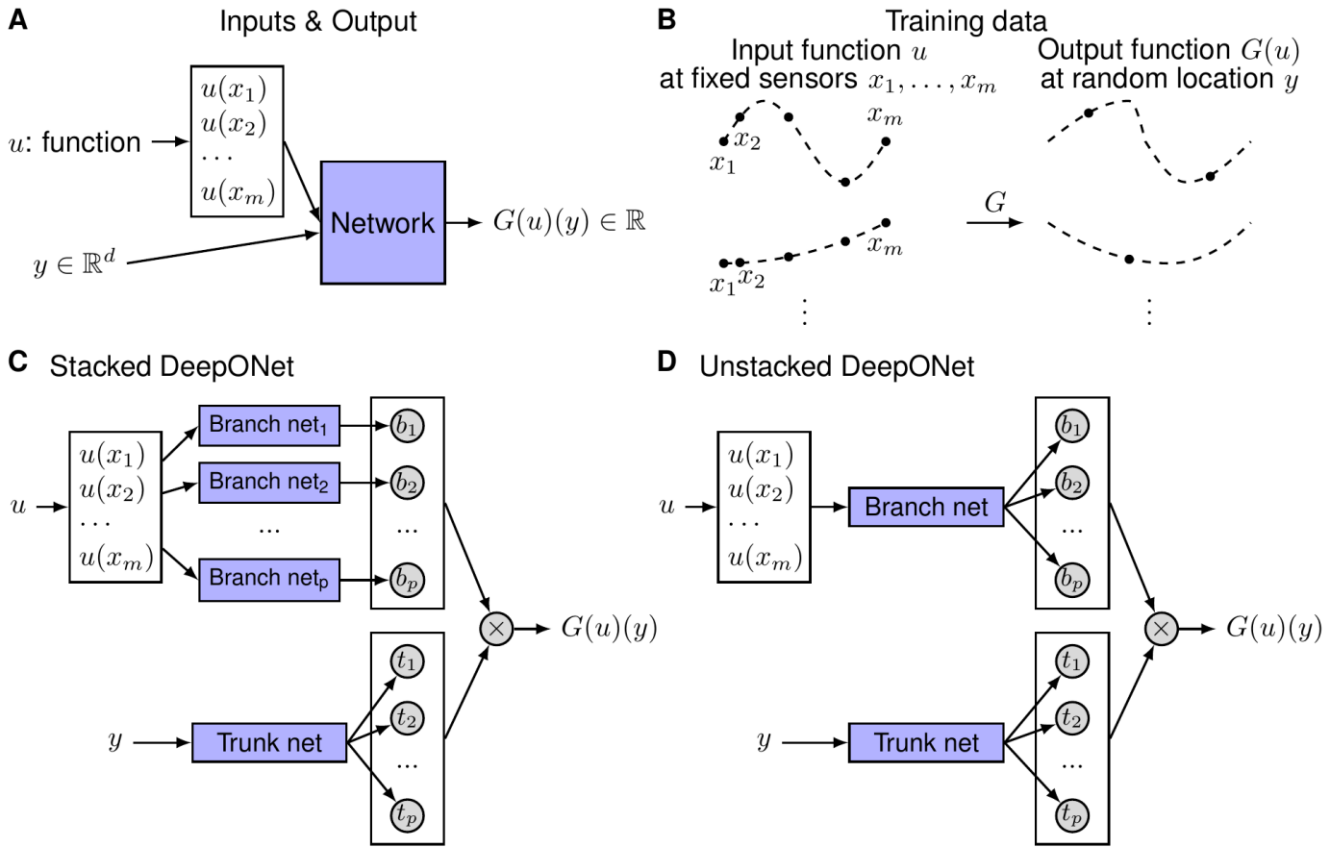
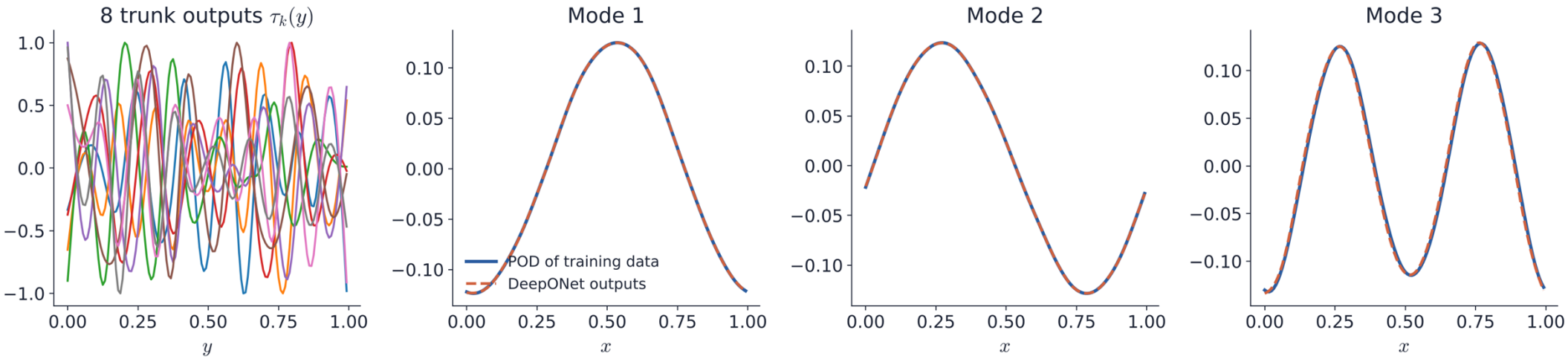


Figure: Lu, Jin, Pang, Zhang & Karniadakis, Learning nonlinear operators via DeepONet, Nature Machine Intelligence 2021

DeepONet learns a basis



- The trunk outputs $\tau_k(y)$ are p **learned basis functions**; the branch gives their **coefficients** for each input
- Recall: **POD** finds the basis that captures the most variance of the data. POD-DeepONet uses POD modes as the trunk
- The leading modes of the DeepONet outputs **match the POD modes** of the training data

A DeepONet output is a linear combination of p fixed functions

POD-DeepONet: Lu et al., CMAME 2022

Universal approximation for operators

- **Chen & Chen (1995)**: a branch-trunk network approximates any continuous operator on a compact set K of input functions to any accuracy: $\sup_{u \in K} \|\mathcal{G}(u) - \mathcal{G}_\theta(u)\| < \varepsilon$
- **Kovachki, Lanthaler & Mishra (2021)**: the same holds for FNO
- Like the theorem for networks in Lecture 2, these give **no bound** on the number of parameters or the amount of data, and **no guarantee** that training finds the weights
- Lanthaler & Stuart (2023): for general operators, the required size grows **exponentially** in $1/\varepsilon$, the *curse of parametric complexity*
- Operators from PDEs have **structure** (smoothness, locality, low-dimensional solutions) that avoids this in practice

Theory says the model can represent the operator; data and training decide whether it does

Chen & Chen, IEEE TNN 1995. Kovachki, Lanthaler & Mishra, JMLR 2021. Lanthaler & Stuart, The curse of parametric complexity in operator learning, arXiv 2023

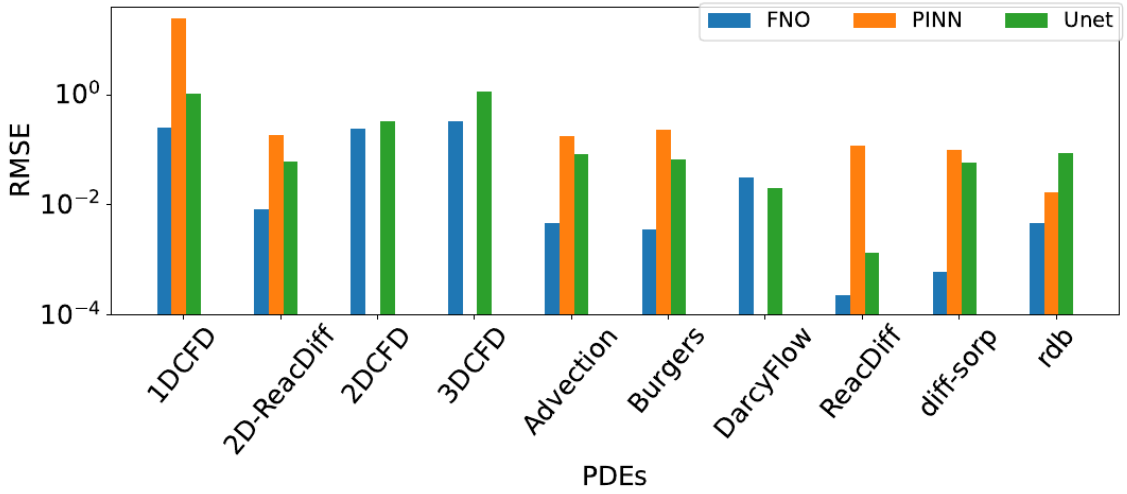
FNO and DeepONet compared

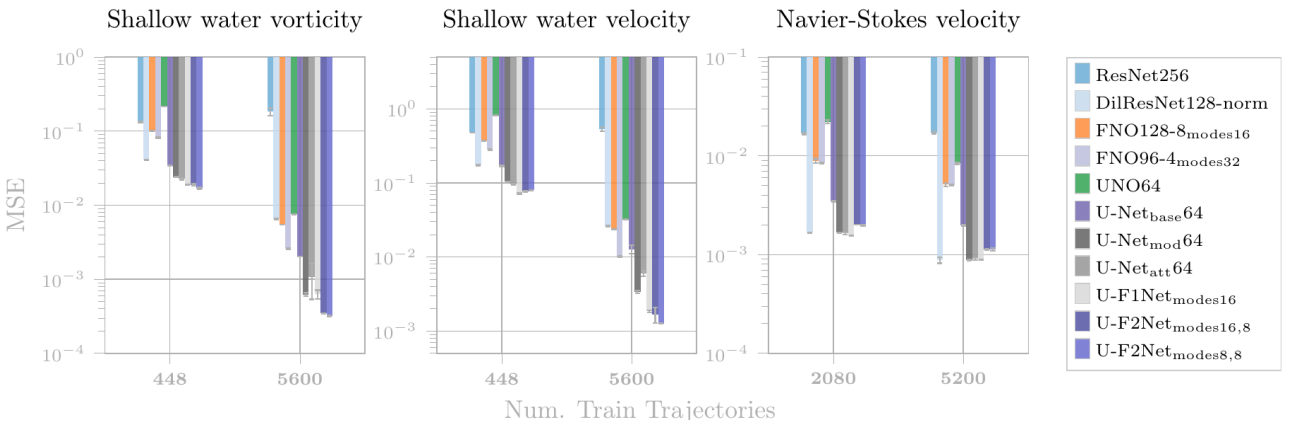
	FNO	DeepONet
Input	function on a uniform grid, any N	values at m fixed sensors
Output	values on the input grid	value at any query point y
Geometry	periodic box; others need padding or a map to a box	any; the trunk takes coordinates
Nonlinearity	in every layer, at every point	in the branch and trunk; output is linear in τ_k
Cost per layer	$O(N \log N)$	one trunk evaluation per query point
Weak case	sharp fronts, irregular domains	moving fronts (fixed basis)

- Lu et al. (2022) compared both on 16 benchmarks: **similar accuracy** once each was given the extensions it needs

Lu, Meng, Cai, Mao, Goswami, Zhang & Karniadakis, A comprehensive and fair comparison of two neural operators (with practical extensions) based on FAIR data, CMAME 2022

Neural operators and U-Nets on benchmarks





- PDEBench (left): FNO has the **lowest error on most** of 11 problems, with the authors' training setups
- PDEArena (right): one codebase for all models. **Modern U-Nets** reach **lower error than FNO** on shallow water and Navier-Stokes
- The ranking depends on the **problem, tuning and budget**. Many models combine both: **Fourier layers inside a U-Net** (U-NO, U-FNet)

Compare against a tuned U-Net at equal compute

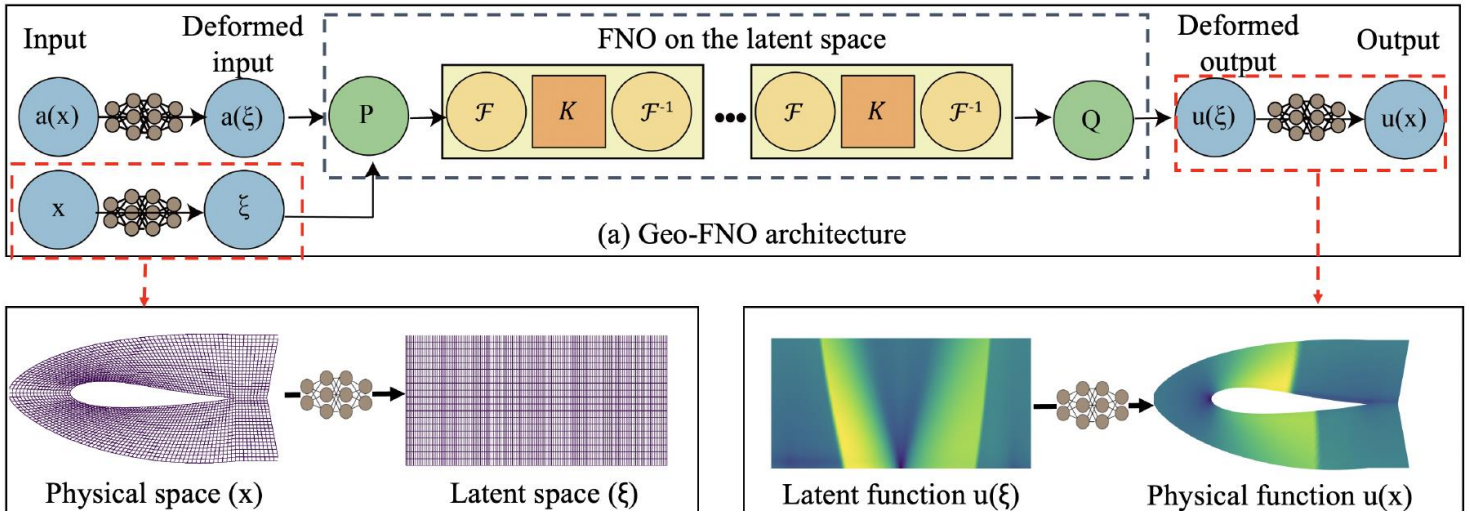
Left: Takamoto et al., PDEBench, NeurIPS 2022. Right: Gupta & Brandstetter, PDEArena, TMLR 2023. U-NO: Rahman et al., TMLR 2023

Operator models in practice

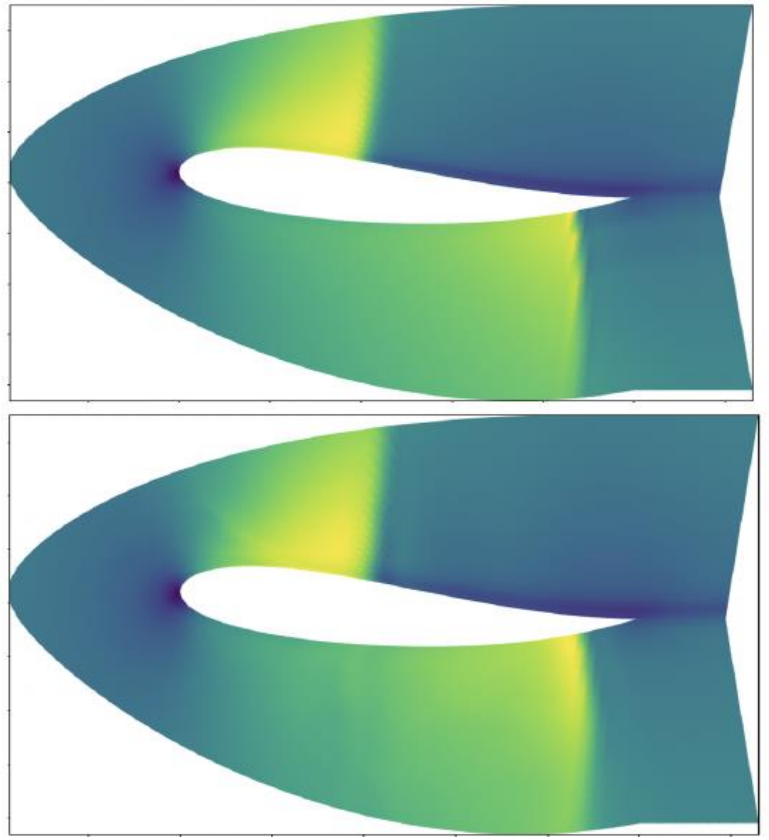
Model	Change from FNO	Used for
F-FNO (Tran et al. 2023)	Fourier layer along one dimension at a time; residual connections	deeper models, fewer weights
Geo-FNO (Li et al. 2023)	learns a map from an irregular domain to a uniform grid	airfoils, pipes, elasticity
GINO (Li et al. 2023)	graph operator from mesh to latent grid, FNO on the grid	3D car aerodynamics
AFNO in FourCastNet (Pathak et al. 2022)	Fourier mixing of patch tokens inside a vision transformer	global weather at 0.25°
Spherical FNO (Bonev et al. 2023)	spherical harmonics in place of Fourier modes; rotation equivariant	weather, shallow water on a sphere

Tran et al., ICLR 2023. Li, Huang, Liu & Anandkumar, JMLR 2023. Li et al., NeurIPS 2023. Pathak et al., arXiv 2022. Bonev et al., ICML 2023

Geo-FNO for irregular domains



- The FFT needs a **uniform grid on a box**. An airfoil mesh is neither
- Geo-FNO **maps the physical mesh** x to a uniform latent grid $\xi = \varphi(x)$, runs an FNO there, and maps the output back. The map φ is given or learned
- Right: the **shocks** above and below the airfoil are in the right place
- **GINO** learns this map with a graph operator on 3D meshes

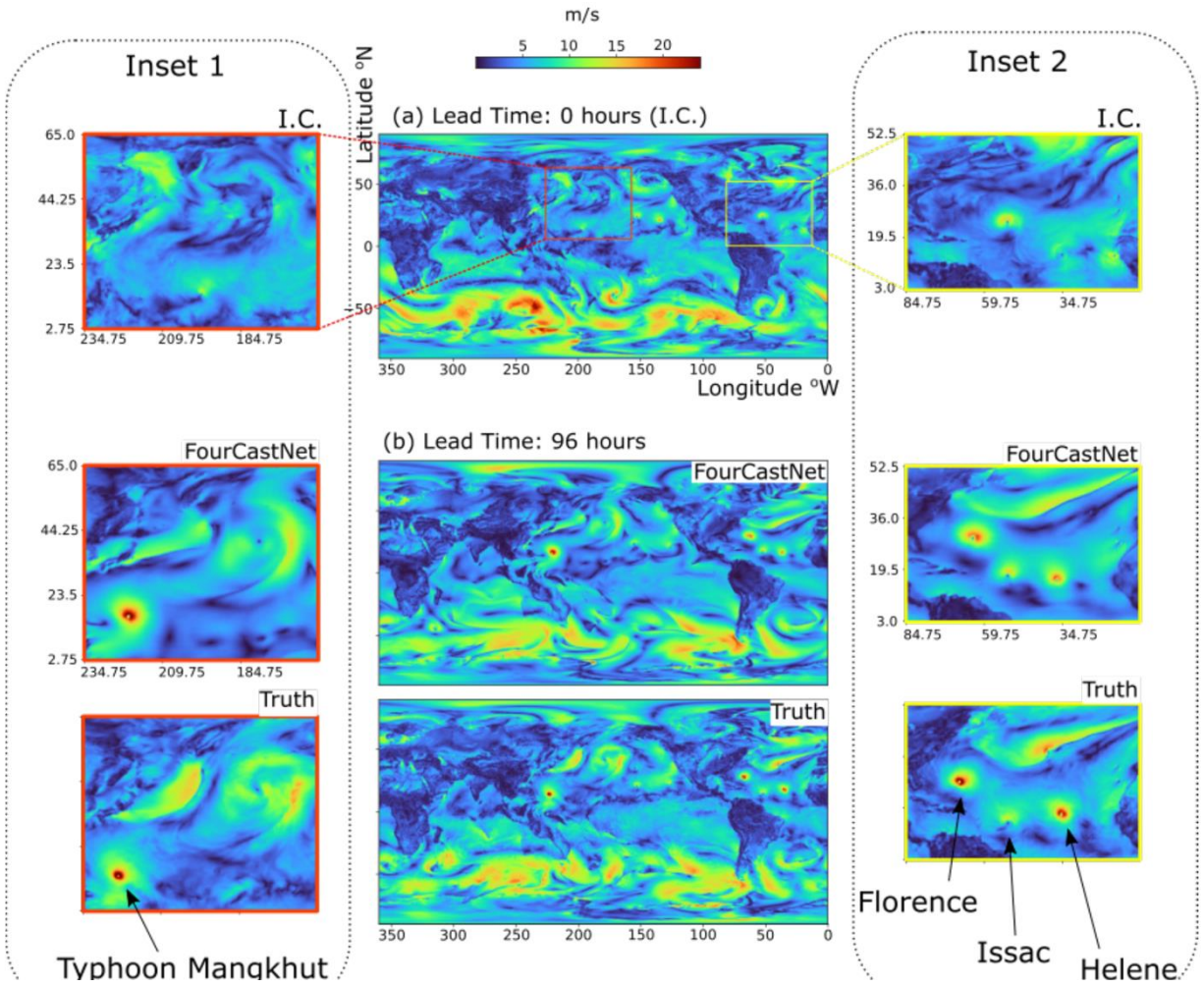


airfoil flow: truth (top), Geo-FNO (bottom)

Li, Huang, Liu & Anandkumar, Fourier neural operator with learned deformations for PDEs on general geometries, JMLR 2023. GINO: Li et al., NeurIPS 2023

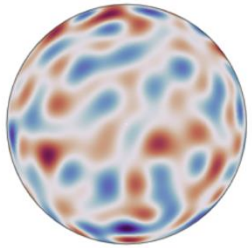
Fourier mixing in FourCastNet

- Global weather on a **0.25° grid**: 720 x 1440 points, about 30 km at the equator. 20 variables, 6-hour steps, trained on ERA5
- A vision transformer on 8 x 8 patches. The token-mixing step is an **adaptive FNO (AFNO)** in place of **attention**: $O(N \log N)$ instead of $O(N^2)$
  - FFT over the patch grid, a small **MLP on each mode**, inverse FFT
  - The MLP weights are **shared across modes**; FNO learns one R_k per mode
- Right: wind speed forecast at **96 hours**. Typhoon Mangkhut and three Atlantic hurricanes are tracked

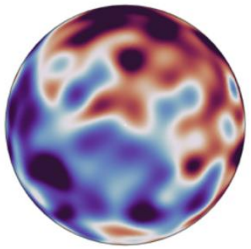


Pathak et al., FourCastNet: a global data-driven high-resolution weather model using adaptive Fourier neural operators, arXiv 2022, AFNO: Guibas et al., ICLR 2022

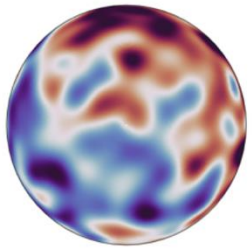
Spherical Fourier neural operator



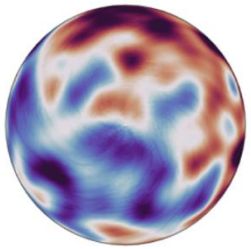
(a) initial condition, $t = 0h$



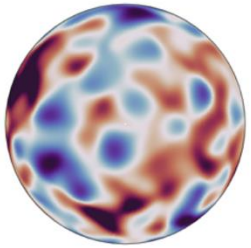
(b) ground truth, $t = 5h$



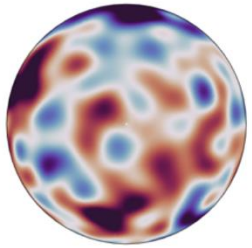
(d) SFNO, $t = 5h$



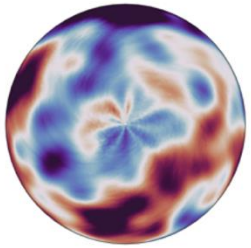
(f) FNO, $t = 5h$



(c) ground truth, $t = 10h$



(e) SFNO, $t = 10h$



(g) FNO, $t = 10h$

- FFT on a latitude-longitude grid treats the sphere as a **flat periodic rectangle**
- SFNO replaces the FFT with the **spherical harmonic transform**, the Fourier basis of the sphere
- The layer is **equivariant to rotations** of the sphere
- Left: shallow water on a rotating sphere, viewed from the south pole. FNO shows **artifacts at the pole** by 10 h. **SFNO stays close to the truth**
- **Stable** weather rollouts for a year (1460 six-hour steps)

Choose the transform that matches the geometry of the domain

Bonev, Kurth, Hundt, Pathak, Baust, Kashinath & Anandkumar, Spherical Fourier Neural Operators: learning stable dynamics on the sphere, ICML 2023

Operator learning in your project

- **State the operator**: input function, output function, and the grid and solver
- **Baselines**: an FNO from public code (neuraloperator), a tuned U-Net (PDEArena) as the standard non-operator baseline, and persistence for time stepping
- **Choose $k_{\max}$** from the spectrum of your outputs: keep the modes that carry most of the energy
- **Resolution tests**: train at one resolution, test on solver data at another; label them as **out-of-distribution**
- **Report error by frequency band** and at fronts or interfaces, as well as relative L^2 error

Mid-project check-in: show a trained baseline and its error on your test split

neuraloperator.github.io (Kossaifi et al. 2024). PDEArena: github.com/pdearena.

Summary

- An **operator** maps functions to functions. A CNN learns it on one grid; a neural operator learns it on functions
- Linear solution operators are **kernel integrals**: $u(x) = \int G(x, y)\, f(y)\, dy$
- Neural operator layer: $v_{l+1} = \sigma(W\, v_l + \int \kappa_\theta(\cdot, y)\, v_l(y)\, dy)$
- **FNO**: translation-invariant kernel on the first $k_{\max}$ Fourier modes, $\mathcal{K}v = \mathscr{F}^{-1}(R_k\, (\mathscr{F}v)_k)$; weights independent of N
- The same weights run on any grid. **Zero-shot transfer** holds only when finer data add no new frequencies; test it, or train on several resolutions
- FNO errors sit at **sharp fronts and interfaces**
- **DeepONet**: $\sum_k b_k(u_0)\, \tau_k(y)$; output at any point, input on fixed sensors
- Benchmarks disagree on FNO vs. U-Net; **compare at equal tuning and compute**

After class

- Kovachki et al., *Neural Operator: Learning Maps Between Function Spaces*, JMLR 2023: Sections 1-5
- Li et al., *Fourier Neural Operator for Parametric Partial Differential Equations*, ICLR 2021
- Lu et al., *Learning nonlinear operators via DeepONet*, Nature Machine Intelligence 2021
- Azizzadenesheli et al., *Neural operators for accelerating scientific simulations and design*, Nature Reviews Physics 2024 (review)
- Sakarvadia et al., *The false promise of zero-shot super-resolution in machine-learned operators*, ICLR 2026
- neuraloperator.github.io: library and tutorials

Next, Lecture 13 (Thu, Nov 5): GNNs and mesh-based learning

NEXT: HANDS-ON

From concepts to code

→ **Open the Lecture 12 notebook**

1. Pretrained FNO, CNN and DeepONet on 1D Burgers: evaluate each on grids from 32 to 1024 points
2. Inside one Fourier layer: spectra of the two paths and of the output
3. Change $k_{\max}$ and see where the error is largest
4. DeepONet trunk functions vs. POD modes