Resolution-Agnostic Learning: Neural Operators That Scale

Junbin Gao

The University of Sydney Business School

Sydney AI Meetup, 24 November 2025

Outline

Motivation

Why neural operators?

- Expensive simulators (PDEs, multi-physics) $\Rightarrow$ need fast surrogates.
- Standard nets learn point-to-point maps $\mathbb{R}^d \to \mathbb{R}^{out}$; **neural operators** learn maps *between functions*.

$$\text{input field } a(\mathbf{x}) \longrightarrow \boxed{\mathcal{G}_\theta} \longrightarrow \text{solution } u(\mathbf{x})$$

From input field $a(\mathbf{x})$ to solution field $u(\mathbf{x})$

- **Resolution-agnostic:** train on coarse meshes; infer on fine without retraining.
- **Generalize** across parameters, geometries, forcing.
- In general, functions a and u may have different domains. Because of this we can use notation $u(\mathbf{y})$

Neural Operators (NOs)

- Neural operators are a powerful class of models designed to learn mappings between infinite-dimensional function spaces, making them well-suited for functional regression problems and solving parametric partial differential equations (PDEs) in a resolution-agnostic way.
- Typically, given two Banach spaces $\mathcal{A}, \mathcal{U}$, the "input-output" relationship is an operator $F : \mathcal{A} \to \mathcal{U}$ acting between this two function spaces.
- A neural operator F_θ, with learnable parameters θ, serve as a continuous approximation of F. It is independent of any particular input discretization, allowing a single model to generalize consistently across different meshes.
- Specifically, the architecture F_θ has two key properties: **1) universality** and **2) discretization-invariant**.

Operator intuition: functional analysis

- Functional view of vectors: A d-dimensional vector $\mathbf{x} = (x_1, x_2, ..., x_d) \in \mathbb{R}^d$ can be regarded as a function

$$x : \mathcal{D} = \{1, 2, ..., d\} \to \mathbb{R}, \quad x(i) = x_i.$$

- Vector view of functions: A function a defined on domain, e.g., $\mathcal{D} = [0, 1]$, is a vector.

$$a : \mathcal{D} = [0, 1] \to \mathbb{R}, \quad \forall x \in \mathcal{D}, y = a(x) \in \mathbb{R}$$

  by thinking of $\mathcal{D} = \{x_1, x_2, x_3, ...\}$, the function can be a (long) vector $a = (a(x_1), a(x_2), a(x_3), ...)$. a.k.a function value table.
- Education: A function is a with its domain $\mathcal{D}$, not $a(x)$ where $a(x)$ is the calculation rule for the function or mapping rule.
- An **operator** maps one function to another: $\mathcal{G} : a \mapsto u$, which means mapping a (long) vector to another (long) vector. Exactly what a neural network does!

Samples of functions

- In application, a function is normally presented as a finite number of samples in terms of $\{(\mathbf{x}_i, u_i)\}_{i=1}^{N}$ where $\mathbf{x}_i \in \mathcal{D}$ and $u_i \in \mathbb{R}$. Or when $\mathbf{x}_i$ is clear (e.g. uniformed discrete points on $\mathcal{D}$), then the function is presented by N-dimension vector $(u_1, u_2, \ldots, u_N) \in \mathbb{R}^N$.
- **Neural operators** learn this map *without fixing N or M*: same parameters work across meshes/resolutions.

$$\begin{array}{c} a(x) \\ \downarrow \\ \big(a(x_1), \ldots, a(x_N)\big) \end{array} \longrightarrow \boxed{\mathcal{G}_\theta} \longrightarrow \begin{array}{c} u(x) \\ \downarrow \\ \big(u(x'_1), \ldots, u(x'_M)\big) \end{array}$$

Same θ works for different N, M (resolutions).

- How to design such “operators”?

From Neural Networks to Neural Operators

- **Start: standard NN is vector→vector**

$$f_\theta : \mathbb{R}^N \to \mathbb{R}^M, \quad \mathbf{y} = f_\theta(\mathbf{x})$$

  Make "functions" into vectors by sampling on grids:

$$a(\mathbf{x}) \;\Rightarrow\; \mathbf{a} \in \mathbb{R}^N, \qquad u(\mathbf{x}) \;\Rightarrow\; \mathbf{u} \in \mathbb{R}^M.$$

- **Goal:** learn a *single* f_θ that works across resolutions (different N, M, different $\{x_1, ..., x_N\}$ and $\{x'_1, ..., x'_M\}$) $\Rightarrow$ neural operator.
- **Key ideas**
  - *Parameter sharing across grids*: same θ for coarse/fine meshes.
  - *Coordinate-aware* features (positional encodings, trunk nets).
  - *Physics-friendly priors* (e.g., Fourier layers).

Neural Operator Construction

Examples of NO Architectures

- Deep Operator Network (DeepONet), learning nonlinear operators by combining a "branch" network that encodes sampled values of the input function with a "trunk" network that encodes evaluation locations to produce the operator's output.
- Graph Neural Operator (GNO), generalizing graph neural networks to learn "linear" mappings between functions, aka *integral operators*, by iteratively applying message-passing layers whose weights are shared across nodes (location of input or output function values) and conditioned on positional or edge features, thus approximating nonlinear operators
- Fourier Neural Operator (FNO), learning nonlinear operators on continuous domains by repeatedly lifting input fields into spectral space, applying learned complex-valued multipliers to selected Fourier modes, and transforming back to physical space, enabling mesh-independent prediction of function-to-function mappings.

Two Practical Constructions

DeepONet (basis expansion)

$$u(\mathbf{x}) \approx \sum_{r=1}^{R} \underbrace{b_r(\mathbf{a})}_{\text{branch}} \underbrace{t_r(\mathbf{x})}_{\text{trunk}}$$

Works on arbitrary point sets; resolution-agnostic via **x**-query.

FNO (spectral convolution)

$$\hat{u}'(\mathbf{k}) = W(\mathbf{k})\,\hat{u}(\mathbf{k}) \text{ on low modes,} \quad u' = \mathcal{F}^{-1}[\hat{u}'] + \mathrm{skip}$$

Grid-agnostic via FFT basis; strong bias for PDE dynamics.

Outcome: a learned map $\mathcal{G}_\theta : a \mapsto u$ that generalizes across meshes, domains, and parameters.

DeepONet Branch Design (Resolution-Invariant Encoders)

Goal: map sampled input function a on *any* grid $\{(x_j, a_j)\}_{j=1}^N$ to branch features $b(a) \in \mathbb{R}^R$ that are *independent of N and spacing*.

1. **DeepSets / Sample–then–Pool**

$$b(a) = \rho\left(\sum_{j=1}^{N} \phi(a_j,\ x_j)\ w_j\right)$$

   Permutation-invariant; w_j can be quadrature weights. ϕ, ρ are small MLPs.

2. **Attention Pooling (Set Transformer)**

$$\alpha_j = \mathrm{softmax}_j\big(g(a_j, x_j)\big), \quad b(a) = \sum_{j=1}^{N} \alpha_j\, h(a_j, x_j)$$

   Learns where to "look"; handles nonuniform sampling.

DeepONet Branch Design (Resolution-Invariant Encoders)

1. **Projection onto (Learned or Fixed) Bases**

$$b_r(a) \;=\; \sum_{j=1}^{N} a_j \, \psi_r(x_j) \, w_j, \quad \psi_r \in \{\text{Fourier/SIREN/poly or learned}\}$$

   Acts like resolution-agnostic features via quadrature.

2. **Graph Encoder (Message Passing)** Build k-NN graph on $\{x_j\}$; run L message-passing layers, then global pool:

$$z = \mathrm{Pool}\big(\mathrm{GNN}_\theta(\{(x_j, a_j)\})\big), \quad b(a) = Wz$$

   Robust to irregular meshes and varying density.

3. **Spectral Truncation Encoder (FNO-style)** FFT on the input grid, keep low modes $|k| \leq K$ (fixed K), iFFT or linear map:

$$b(a) = \mathrm{vec}\big(\hat{a}(k)\big)_{|k|\leq K}$$

   Fixed number of retained modes $\Rightarrow$ fixed branch size across resolutions.

DeepONet Branch Design (Resolution-Invariant Encoders)

1. **CNN with Adaptive Pooling (for gridded inputs)** Apply small CNN/ResNet, then *adaptive global average pooling* to a fixed size:

   $$b(a) = \mathrm{AGAP}\big(\mathrm{CNN}(a)\big)$$

   Practical when inputs live on images/structured grids.

2. **Trunk (coordinate net):** $t(x)$ can be a standard MLP/SIREN with positional encodings. **DeepONet output:** $u(x) \approx \sum_{r=1}^{R} b_r(a)\, t_r(x)$.

From Linear Layers to Linear Operators (Kernel View)

Finite-dimensional (NN) view

$$\mathbf{u} = W\mathbf{x} \quad \text{(ignore bias)}$$

where $W \in \mathbb{R}^{M\times N}$ is a *linear layer*.

Function-space (operator) view

$$u(x) := (\mathcal{K}a)(x) \;=\; \int_{\mathcal{D}} k(x,\xi)\, a(\xi)\, d\xi$$

A *linear operator* applied to a function a; $k(x,\xi)$ is the **kernel** (simulating W).

Discretize functions as long vectors

$$a(\xi_\ell) \;\Rightarrow\; \mathbf{a} \in \mathbb{R}^N, \quad u(x_j) \;\Rightarrow\; \mathbf{u} \in \mathbb{R}^M.$$

Integral $\Rightarrow$ matrix multiply

$$u(x_j) \;\approx\; \sum_{\ell=1}^{N} k(x_j,\xi_\ell)\, a(\xi_\ell)\, w_\ell$$

Let $K_{j\ell} = k(x_j,\xi_\ell)$ and $W_{\ell\ell} = w_\ell$ (quadrature weights):

$$\underbrace{\mathbf{u}}_{\mathbb{R}^M} \;\approx\; \underbrace{K\,W}_{\mathbb{R}^{M\times N}}\,\underbrace{\mathbf{a}}_{\mathbb{R}^N} \quad\Longleftrightarrow\quad \mathbf{u} = \widetilde{W}\mathbf{a}.$$

Takeaway: a linear operator on functions becomes that a *kernel matrix* times a *function vector* after sampling.
One way to implement this kernel is to use an NN to compute $k(x,\xi)$ for a pair of x and ξ.

Special case: Convolution

- Choose the kernel as $k(x,\xi) = h(x-\xi)$, then

$$u(x) = (\mathcal{K}a)(x) = \int h(x-\xi)\, a(\xi)\, d\xi = (h * a)(x) \;\Rightarrow\; \text{Resulting Toeplitz/Circulant } \widetilde{W}.$$

$$T = \begin{pmatrix} 3 & 2 & 5 & 7 & 4 \\ 6 & 3 & 2 & 5 & 7 \\ 8 & 6 & 3 & 2 & 5 \\ 9 & 8 & 6 & 3 & 2 \\ 1 & 9 & 8 & 6 & 3 \end{pmatrix} \quad \text{and} \quad C = \text{circ}(1,2,3,4,5) = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 5 & 1 & 2 & 3 & 4 \\ 4 & 5 & 1 & 2 & 3 \\ 3 & 4 & 5 & 1 & 2 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}$$

- As $u(x) = (h * a)(x) \Leftrightarrow \hat{u}(k) = \hat{h}(k) \cdot \hat{a}(k)$, This choice is equivalent to design Fourier Operator layers.

From Kernel Operators to FNO (Fourier View)

Linear operator as integral:

$$u(x) = (\mathcal{K}a)(x) = \int_{\mathcal{D}} k(x,\xi)\, a(\xi)\, d\xi$$

If translation-invariant $k(x,\xi) = \kappa(x-\xi)$ (convolution):

$$\widehat{(\mathcal{K}a)}(k) = \widehat{\kappa}(k)\, \hat{a}(k)$$

Discretize & learn in frequency:

$$\hat{u}'(k) = W_\theta(k)\, \hat{u}(k) \quad \text{(per low-frequency mode)}$$

Keep only $|k| \leq K$ (spectral truncation) for efficiency/inductive bias.

One FNO layer (practical form):

$$u^{(\ell+1)} = \sigma\Big(W_0 u^\ell + \mathcal{F}^{-1}\big[\, R_\theta(k)\, (\mathcal{F}u^\ell)(k)\, \big]_{|k|\leq K}\Big)$$

- $R_\theta(k)$: learnable complex weights per mode (the kernel in Fourier space).
- W_0: pointwise mixing (skip/linear).
- σ: nonlinearity; residual stacking builds *nonlinear* operator.

Why this yields a neural operator

- Parameters live on modes $\Rightarrow$ *resolution-agnostic*.
- Convolution theorem encodes PDE-friendly inductive bias.

From Kernel Integrals to Graph Neural Operators (GNO)

- **Function $\to$ Graphs: bipartite first, then primal**

$$u(x) = \int_{\Omega} k(x, \xi)\, a(\xi)\, d\xi$$

- **Input discretization**

$$\{x_i\}_{i=1}^{M} \text{ for } u, \qquad \{\xi_j\}_{j=1}^{N} \text{ for } a$$

  - *Bipartite graph* $\mathcal{G}_{a\to u}$ between u-nodes x_i and a-nodes ξ_j.
  - Edges: $j \in \mathcal{N}_a(i)$ by k-NN/radius in $\|x_i - \xi_j\|$.
  - Edge feats e_{ij}: $(x_i - \xi_j, \|x_i - \xi_j\|, \mathrm{BC/geom})$, weights w_{ij} (quadrature/normalization).
- **After the first layer**
  - Build a *primal u-graph* $\mathcal{G}_{u\leftrightarrow u}$ on $\{x_i\}$ (kNN/radius).
  - Optionally keep (or refresh) a cross-graph $\mathcal{G}_{a\to u}$ for re-conditioning.
- **Resolution-agnostic:** same kernels on new meshes; only neighborhoods change.

From Kernel Integrals to Graph Neural Operators (GNO)

- **Layer 1: bipartite (encode $a \to u$)** (passing $\{a(x_i)\}$ to nodes $\{\xi_j\}$)

$$u_i^{(1)} = \sigma\Big(b_u + \sum_{j \in \mathcal{N}_a(i)} k_\theta^{(0)}(x_i, \xi_j, e_{ij})\, a_j\, w_{ij}\Big)$$

- **Layers $\ell \geq 1$: primal u-graph propagation** (all nodes $\{x_i, \xi_j\}$)

$$u_i^{(\ell+1)} = \sigma\Big(W^{(\ell)} u_i^{(\ell)} + \sum_{p \in \mathcal{N}_u(i)} g_\theta^{(\ell)}(x_i, x_p, e_{ip})\, u_p^{(\ell)}\Big)$$

- **Design choices**
  - *Hybrid (default):* 1 bipartite layer, then L primal layers.
  - *Alternating:* $(a \to u)$ then $(u \leftrightarrow u)$, repeat.
  - *Shared grid:* if $x_i \equiv \xi_i$, the first layer reduces to a standard GNN conv.
  - *Attention form:* normalize $k_\theta^{(0)}$ or $g_\theta^{(\ell)}$ with softmax over neighbors.

From Linear GNO to Nonlinear Kernel Neural Operators

- **Operator viewpoint**

$$(\mathcal{K}[a, u^{(\ell)}]\, v)(x) = \int_\Omega k_\theta\big(x, \xi;\ a(\xi), u^{(\ell)}(x), u^{(\ell)}(\xi)\big)\, v(\xi)\, d\xi$$

  - If k_θ does *not* depend on $a, u^\ell \Rightarrow$ linear kernel.
  - If k_θ depends on $(a, u^\ell) \Rightarrow$ **nonlinear, input-/state-dependent** operator.

- **Discretization (bipartite then primal)**

$$u_i^{(1)} = \sigma\Big(\sum_{j \in \mathcal{N}_a(i)} k_\theta^{(0)}(x_i, \xi_j;\ a_j)\, a_j\, w_{ij} \Big)$$

$$u_i^{(\ell+1)} = \sigma\Big(W^{(\ell)} u_i^{(\ell)} + \sum_{p \in \mathcal{N}_u(i)} k_\theta^{(\ell)}(x_i, x_p;\ a_p, u_i^{(\ell)}, u_p^{(\ell)})\, u_p^{(\ell)}\, w_{ip} \Big)$$

Recovery: remove dependence on (a, u^ℓ) to get the linear GNO layer.

From Linear GNO to Nonlinear Kernel Neural Operators

- **Three useful nonlinear kernel designs**
  1. *Attention (nonlocal) kernel*

$$k_\theta = \mathrm{softmax}_{p\in\mathcal{N}_u(i)}\big(q_\theta(x_i, u_i^{(\ell)})^\top k_\theta(x_p, u_p^{(\ell)})\big)$$
$$\Rightarrow \; u_i^{(\ell+1)} = \sum_p k_\theta(\cdot)\, v_\theta(u_p^{(\ell)})$$

     (cross-attn): replace u^ℓ at source with a.

  2. *Dynamic (space-variant) convolution*

$$k_\theta(x_i, x_p; \cdot) = \sum_{m=1}^{M} \beta_m^{(\ell)}(x_i, u_i^{(\ell)})\, h_m(x_i - x_p)$$

     Apply M fixed (FFT-able) kernels h_m then mix by β_m.

  3. *Neural RKHS kernel (feature map)*

$$k_\theta(x_i, x_p; \cdot) = \phi_\theta(x_i, u_i^{(\ell)}, a_i)^\top\, \phi_\theta(x_p, u_p^{(\ell)}, a_p)$$

     Positive definite by construction; control via $\|\phi_\theta\|$.

Operators: Overall Archictectures

Neural Operators: Overall Architecture

- **Goal.** Learn an operator $\mathcal{G}_\theta : a \mapsto u$ that maps input function a on a domain $\mathcal{D}$ to output function u, *independent of discretization*, pipeplined in three stages:

$$\boxed{\text{Lift}} \Rightarrow \boxed{\text{Operator Backbone (L blocks)}} \Rightarrow \boxed{\text{Readout}}$$

- **Lift (local embedding, resolution-agnostic)**

$$u^{(0)}(x) \;=\; P\big(a(x),\, x,\, \mathrm{PE}(x),\, \mathrm{BC/geom}\big) \in \mathbb{R}^C$$

  where P is a pointwise map (e.g. 1×1 conv/MLP). Include coordinates or positional encodings (PE) and boundary/geometry tags. Works on grids or point clouds.

- **Operator Backbone (nonlocal field mixing)**

$$u^{(\ell+1)} \;=\; \sigma\Big(\mathcal{A}_\theta^{(\ell)}[u^{(\ell)}] \;+\; B^{(\ell)} u^{(\ell)}\Big), \quad \ell = 0, \ldots, L-1$$

  with residual $B^{(\ell)}$ (pointwise).

Neural Operators: Overall Architecture

- The operator block $\mathcal{A}_\theta^{(\ell)}[\cdot]$ is *one of*:
  - **FNO (spectral):** FFT $\Rightarrow$ low-mode multipliers $W^{(\ell)}(k)$ $\Rightarrow$ IFFT.
  - **GNO (graph/point cloud):** message passing on k-NN / radius graphs:

$$(\mathcal{A}_\theta^{(\ell)} u^{(\ell)})_i = \sum_{j \in \mathcal{N}(i)} k_\theta(x_i, x_j, e_{ij})\, u_j^{(\ell)}\, w_{ij}.$$

  - **Attention (nonlocal kernel):** q, k, v maps with softmax over neighbors/modes (self- or cross-attn with a).
  - **Hybrid/mixtures:** dynamic mixtures of fixed kernels, multigrid spectral+graph stacks, or alternating cross-attn refreshes from a.
- **Readout (projection to target channels)**

$$u(x) \;=\; Q\big(u^{(L)}(x), x\big), \quad Q : \mathbb{R}^C \to \mathbb{R}^{C_{\text{out}}} \text{ (pointwise)}$$

Objectives and Stability

Training objectives

- **Supervised**: $\|u_\theta - u_{\text{ref}}\|$ (e.g., L^1/L^2), spectral losses, gradient/energy terms.
- **Physics-informed**: PDE residuals $\|\mathcal{R}(u_\theta, a)\|$, BC penalties, integral constraints.

Stability & regularization

- *Residual/skip* paths, normalization (layer/instance), degree/softmax normalization on graphs.
- *Spectral control*: limit Fourier bandwidth (FNO), spectral norm/weight decay on $W^{(\ell)}$, anti-aliasing.
- *Lipschitz control*: clamp attention logits, bound dynamic mixing coefficients, Jacobian regularizers.

I/O variants

- *Bipartite input→output*: first layer reads a at $\{\xi_j\} \to u$ at $\{x_i\}$, then propagate on u-graph.
- *Shared grid*: if $x \equiv \xi$, use a single graph/grid with concatenated channels for a.

FNO Parameter Complexity (channels C, layers L)

- **Setup.** Keep K_d low Fourier modes per axis $d = 1, \dots, D$. Number of learned spectral coefficients:

$$N_{\text{modes}} \approx \prod_{d=1}^{D} K_d \quad \text{(up to a small symmetry factor for real FFTs).}$$

- **Per Fourier layer** (width C) + Pointwise skip (1×1 conv):

$$\underbrace{W(k) \in \mathbb{C}^{C\times C}}_{\text{modewise multiplier}} + \text{Skip Connection} \quad \Rightarrow \quad \#\text{params} \approx 2\,N_{\text{modes}}\,C^2 + C^2$$

- **Lift/Readout (once each):**

$$\#P = (C_{\text{in}} + \text{coord})\,C \qquad \#Q = C\,C_{\text{out}} \quad \text{(typically negligible vs. layers).}$$

Total (L layers):

$$\boxed{\#\theta_{\text{FNO}} \approx L\,(2\,N_{\text{modes}} + 1)\,C^2 + \#P + \#Q}$$

Independent of grid size n (pixels/points).

Concrete Examples & Comparisons

- **Example A (2D):** $K_x = K_y = 12 \Rightarrow N_{\text{modes}} = 144,\ C = 64,\ L = 4,\ C_{\text{in}} = 2,\ C_{\text{out}} = 1.$

$$\#\text{per layer} \approx (2 \cdot 144 + 1) \cdot 64^2 = 289 \cdot 4096 \approx 1.18\text{M}$$

$$\#\text{total} \approx 4 \times 1.18\text{M} \approx 4.7\text{M} \quad (+\ \#P+\#Q \approx \text{few} \times 10^2)$$

- **Example B (3D):** $K_x{=}K_y{=}K_z{=}12 \Rightarrow N_{\text{modes}} = 1728,\ C = 64,\ L = 4.$

$$\#\text{per layer} \approx (2 \cdot 1728 + 1) \cdot 4096 \approx 14.15\text{M} \quad \Rightarrow \quad \#\text{total} \approx 56.6\text{M}.$$

- **Compute vs. params:**
  - *Params:* depend on C, K_d, L (not grid).
  - *FLOPs / pass:* FFT/IFFT $O(n \log n \cdot C)$ + low-mode multiplies $O(N_{\text{modes}}\, C^2)$.
- **Practical knobs**
  - Reduce K_d (bandlimit) or C (width) to trade accuracy for size.
  - Tie/share $W(k)$ across symmetric modes to cut constants $\sim 2\times$.
  - Use bottleneck pointwise maps $(C \to C_b \to C)$ to reduce C^2.

Functional Tensor Decomposition

Siren Fourier Neural Operators (SirenFNOs): Re-design

- Consider the following Fourier layer

$$u^{(\ell+1)}(x) = \sigma\Big(W^{(\ell)}u^{(\ell)}(x) + \mathcal{F}^{-1}\big(\mathcal{R}_\phi^{(\ell)}(k)\cdot\mathcal{F}(u^{(\ell)})(k)\big)(x)\Big), \forall x \in \mathcal{D}$$

- In practical design, $\mathcal{R}_\phi^{(\ell)}$ is a complex tensor of size $K_1 \times K_2 \times \cdots \times K_D \times C \times C$, where C means input and output channels
- SIREN employs sinusoidal activations to learn continuous implicit representations.
- A key advantage of SirenFNO is that the number of learnable parameters is fixed by the SIREN architecture and does not depend on the grid resolution.
- Thereby, our proposed SirenFNO efficiently represents both lower- and higher-frequency details across varying discretizations.

Siren Fourier Neural Operators (SirenFNOs): Re-design

- In our design, we take $\mathcal{R}_\phi^{(\ell)}$ as a parametric function $\Phi_\theta : [-1,1]^D \to \mathbb{R}^{C\times C}$ with only number $|\theta|$ of parameters.
- Specifically, for each frequency mode $\mathbf{k}$,

$$\Phi_\theta(\xi_k) = (\mathcal{O} \circ \phi^{(L)} \circ \cdots \circ \phi^{(1)} \circ \mathcal{E}_B)(\xi_k),$$

  where $\mathcal{E}_B$ is a learnable random Fourier feature (RFF) embedding of the input frequency mode coordinates,

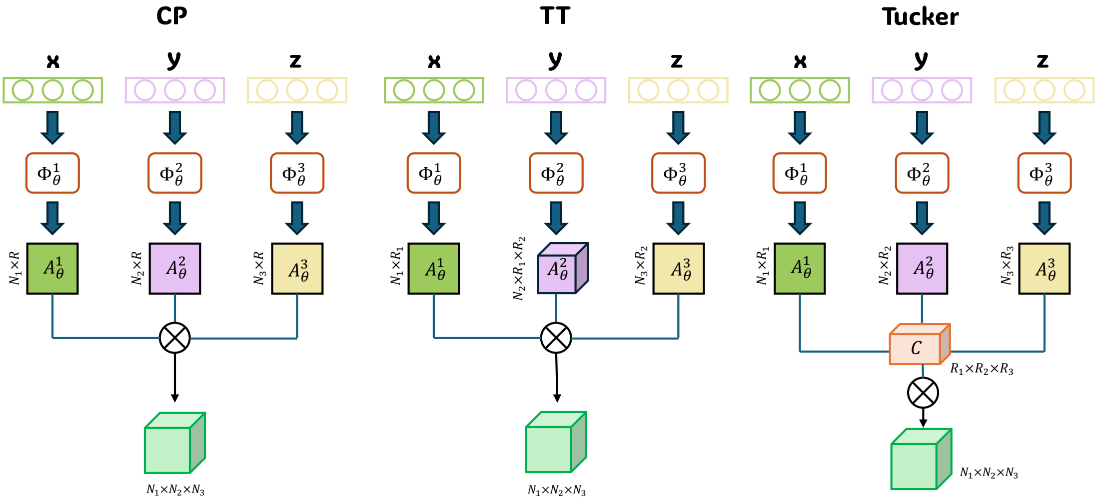
$$\mathcal{E}_B(\xi_k) = \left[\cos(\pi B^\top \xi_k),\ \sin(\pi B^\top \xi_k)\right] \in \mathbb{R}^{2m},$$

- $\phi^{(1)} : \mathbb{R}^{2m} \to \mathbb{R}^h$, and $\phi^{(\ell)} : \mathbb{R}^h \to \mathbb{R}^h$ for $\ell = 2, \ldots L$ each is a SIREN layer

$$\xi^{(\ell)} \leftarrow \phi^{(\ell)}(\xi_k^{(\ell-1)}) = \sin(w \cdot W^{(\ell)} \xi_k^{(\ell-1)} + b^{(\ell)}),$$

- And $\mathcal{O} : \mathbb{R}^h \to \mathbb{C}^{C\times C}$ is an MLP layer.

Functional Tensor Decomposition for SirenFNO



Benchmarks and Baselines

- 2D Darcy flow problem with Dirichlet boundary conditions

$$-\nabla \cdot (a(\mathbf{x})\nabla u(\mathbf{x})) = f(\mathbf{x}), \mathbf{x} \in \mathcal{D}; \quad u(\mathbf{x}) = 0, \mathbf{x} \in \partial\mathcal{D}.$$

- Navier Stokes Equation in Velocity–pressure form (conceptual): Unknowns velocity $u(\mathbf{x}, t)$; given pressure $p(\mathbf{x}, t)$ and viscosity $\nu > 0$ and forcing $f(\mathbf{x}, t)$.

$$\partial_t u(\mathbf{x}, t) + (u \cdot \nabla) u(\mathbf{x}, t) - \nu \Delta u(\mathbf{x}, t) + \nabla p(\mathbf{x}, t) = f(\mathbf{x}, t), \mathbf{x}\mathcal{D}, t > 0$$
$$\nabla u(\mathbf{x}, t) = 0, \mathbf{x} \in \mathcal{D}, t > 0; \quad u(\mathbf{x}, 0) = u_0(\mathbf{x})$$

- 1-D Burgers' equation: **PDE** on $\mathcal{D} = (0, 1)$, $t \in [0, T]$ (periodic BCs):

$$\partial_t u(x, t) + u(x, t)\, \partial_x u(x, t) \; = \; \nu\, \partial_{xx} u(x, t), \qquad u(x, 0) = u_0(x).$$

Name	Dataset	Resolution
Darcy	2-D Darcy flow	32×32, 128×128
NS	2-D Navier Stokes	128×128
Burgers	1-D Burgers' equation	1024, $T = 20$

Main Results: ℓ_2 Relative Test Errors

Dataset	Darcy		NS	Burgers	1DCFD	2DCFD	ReacDiff
Resolution	32×32	128×128	128×128	1024	1024	128×128	1024
FNO	7.30*e*-2	6.26*e*-2	5.61*e*-2	1.12*e*-2	8.27*e*-3	1.64*e*-2	1.18*e*-4
UFNO	7.07*e*-2	6.00*e*-2	5.37*e*-2	1.70*e*-2	8.29*e*-3	1.21*e*-2	1.98*e*-4
U-NO	6.57*e*-2	7.68*e*-2	4.96*e*-2	7.53*e*-3	7.62*e*-3	1.62*e*-2	7.79*e*-5
AM-FNO (MLP)	3.72*e*-2	5.10*e*-2	4.02*e*-2	8.92*e*-3	7.89*e*-3	1.21*e*-2	1.78*e*-3
SirenFNO†	6.32*e*-2	5.82*e*-2	5.52*e*-2	8.21*e*-3	7.41*e*-3	1.15*e*-2	7.40*e*-5
CP-SirenFNO†	4.99*e*-2	3.96*e*-2	5.04*e*-2	8.12*e*-3	5.91*e*-3	7.28*e*-3	8.98*e*-5
TT-SirenFNO†	5.42*e*-2	5.36*e*-2	5.87*e*-2	7.94*e*-3	5.58*e*-3	8.75*e*-3	8.51*e*-5
Tucker-SirenFNO†	4.45*e*-2	4.51*e*-2	4.19*e*-2	7.18*e*-3	7.31*e*-3	7.30*e*-3	7.81*e*-5
SirenFNO	**3.51e-2**	**2.36e-2**	3.51*e*-2	**5.52e-3**	6.77*e*-3	**6.83e-3**	**6.31e-5**
CP-SirenFNO	4.04*e*-2	3.03*e*-2	3.78*e*-2	8.48*e*-3	**4.75e-3**	7.40*e*-3	7.58*e*-5
TT-SirenFNO	4.18*e*-2	3.33*e*-2	**3.24e-2**	7.52*e*-3	6.32*e*-3	7.41*e*-3	8.92*e*-5
Tucker-SirenFNO	3.98*e*-2	3.18*e*-2	4.67*e*-2	8.11*e*-3	5.29*e*-3	6.90*e*-3	8.71*e*-5

ℓ_2 relative test errors. All models are trained and tested on the same corresponding resolution. Models highlighted with † are using the identical architecture with FNO for ablation study. SirenFNO and its variants in boldface but without † are our proposed methods.

Main Results: Number of Parameters

Dataset	Darcy		NS	Burgers	1DCFD	2DCFD	ReacDiff
Resolution	32×32	128×128	128×128	1024	1024	128×128	1024
FNO	1192.8	1192.8	4469.6	4216.2	4216.2	4469.9	4216.2
UFNO	5876.2	5876.2	990.8	4258.5	1892.2	991.0	1892.2
U-NO	1792.7	4250.3	8726.8	1726.1	1726.1	7602.3	5658.2
AM-FNO (MLP)	385.5	385.5	4443.1	207.6	823.1	385.6	823.1
SirenFNO	308.9	308.9	579.5	308.9	304.8	304.8	304.8
CP-SirenFNO	**63.9**	**63.9**	**138.9**	**70.1**	**57.7**	**64.0**	**57.7**
TT-SirenFNO	92.5	109.2	211.6	84.5	72.0	92.7	72.0
Tucker-SirenFNO	162.6	162.6	596.4	74.2	61.8	96.8	61.8

Time-aware Neural Operators

Example PDE: 1D Viscous Burgers

- **PDE** on $\mathcal{D} = (0,1)$, $t \in [0,T]$ (periodic BCs):

$$\partial_t u(x,t) + u(x,t)\,\partial_x u(x,t) \;=\; \nu\,\partial_{xx} u(x,t), \qquad u(x,0) = u_0(x).$$

- **Task (operator learning).** Learn $\mathcal{G}_\theta : \;(u_0,\nu) \;\mapsto\; u(\cdot,t)$

$$\text{either}\quad \mathcal{G}_\theta^{\text{traj}} : \;(u_0,\nu) \;\mapsto\; u(\cdot,\cdot) \quad\text{or}\quad \mathcal{G}_\theta^{\Delta t} : \;(u(\cdot,t),\nu) \;\mapsto\; u(\cdot,t{+}\Delta t).$$

- **Data (typical).** Draw u_0 from a random band-limited field; simulate with a spectral solver (forward Euler/ETDRK4) at high resolution; subsample in space/time for training.
- **Loss.** Supervised spacetime error L_2 (optionally + physics residuals/BC penalties).

Time-Aware Neural Operators: Three Blueprints

- **Spacetime FNO (t as another axis)**

$$\boxed{\text{Lift:}}\ u^{(0)}(x,t) = P[u_0(x),\ x,\ t,\ \nu]$$

$$\boxed{\text{Fourier layer:}}\ \widehat{v}^{(\ell)}(k_x,k_t) = W_\ell(k_x,k_t)\,\widehat{u}^{(l)}(k_x,k_t)\ \ \text{for low } (k_x,k_t)$$

$$u^{(\ell+1)} = \sigma\big(\mathcal{F}^{-1}[\widehat{v}^{(\ell)}] + Bu^{(\ell)}\big), \qquad u = Q\,u^{(L)}.$$

Pros: global coupling in space *and* time; single forward pass for $u(x,t)$. *Notes:* keep few k_t modes; window/pad for non-periodic t.

- **Autoregressive (one-step) Operator**

$$\phi_{\Delta t}:\ \ u(\cdot,t)\ \mapsto\ u(\cdot,t{+}\Delta t)\quad \text{(FNO/GNO/Attn backbone)}$$

$$u_{n+1} = \phi_{\Delta t}(u_n;\nu), \qquad n = 0,\dots,N{-}1.$$

Pros: flexible horizons; plug-in with variable Δt. *Training:* teacher forcing & scheduled sampling; rollout loss across multiple steps.

Time-Aware Neural Operators: Three Blueprints

- **Graph/Attention Neural Operator with Time** *Pros:* irregular meshes, nonuniform time grids, complex BC/geometry.
  - **Spacetime discretization.** Let $X = \{x_i\}_{i=1}^M$, $T = \{t_n\}_{n=0}^N$. Create nodes $\mathcal{V} = \{(i,n) : x_i \in X,\ t_n \in T\}$ and edges $\mathcal{E}$ by, *past light-cone and enforcing causality with* $t_m \le t_n$,

$$\mathcal{N}(i,n) = \{(j,m) : \|x_i - x_j\| \le r_x,\ 0 \le t_n - t_m \le r_t\}$$

  - **Node features.** $h_{i,0}^{(0)} = P\big(u_0(x_i), x_i, t_0, \nu\big)$; for $n > 0$, initialize $h_{i,n}^{(0)} = P\big(x_i, t_n, \nu\big)$.
  - **Separable space–time operator (efficient).**

$$h_{i,n}^{(\ell+1)} = \sigma\Big(W^{(\ell)} h_{i,n}^{(\ell)} + \sum_{j\in\mathcal{N}_x(i)} \kappa_s^{(\ell)}(\phi_{ij};\ h_{i,n}^{(\ell)}, h_{j,n}^{(\ell)})\, U_s^{(\ell)} h_{j,n}^{(\ell)}\, w_{ij}^x + \sum_{m\in\mathcal{N}_t(n)} \kappa_t^{(\ell)}(\tau_{nm};\ h_{i,n}^{(\ell)}, h_{i,m}^{(\ell)})\, U_t^{(\ell)} h_{i,m}^{(\ell)}\, w_{nm}^t\Big)$$

with $\mathcal{N}_t(n) \subseteq \{m : 0 \le t_n - t_m \le r_t\}$ and (optional) cross term:

$$+\sum_{j,m} \beta^{(\ell)}(\phi_{ij}, \tau_{nm})\, U_c^{(\ell)} h_{j,m}^{(\ell)}.$$

Burgers with FNO: Two Concrete Setups

1. **Spacetime FNO (2D FFT over** (x, t)**)** (showing one layer)

$$\widehat{v}(k_x, k_t) = W(k_x, k_t)\,\widehat{u}(k_x, k_t) \quad \text{for } |k_x| \le K_x,\ \ |k_t| \le K_t.$$
$$u' = \sigma\big(\mathcal{F}^{-1}[\widehat{v}] + Bu\big),$$

BCs in t: pad/window or cosine transform along t if non-periodic.

2. **Autoregressive Spatial-FNO (1D FFT over** x**)**

$$\boxed{\text{Lift}}:\ \ u^{(0)}(x) = P\big(u_n(x),\ x,\ \nu,\ \Delta t\big) \in \mathbb{R}^C$$

For $\ell = 0, \dots, L-1$:

$$\widehat{v}^{(\ell)}(k_x) = W_\ell(k_x)\,\widehat{u^{(\ell)}}(k_x) \quad \text{for } |k_x| \le K_x, \qquad u^{(\ell+1)}(x) = \sigma\big(\mathcal{F}_x^{-1}[\widehat{v}^{(\ell)}] + B_\ell u^{(\ell)}(x)\big)$$

$$\boxed{\text{Readout}}:\ \ \tilde{u}_{n+1}(x) = Q\,u^{(L)}(x), \qquad \boxed{\text{Stabilize}}:\ \ u_{n+1} = u_n + \gamma\,\tilde{u}_{n+1} \ \ (\text{residual gain } \gamma \in (0, 1]).$$

- *Stable rollouts*: spectral padding, limit low modes, small residual gain.
- *Curriculum*: train 1-step, then 4-step, then full horizon.

Training, Physics, and Practical Tips (Burgers)

- Predict full trajectory $u(\cdot, t)$ or endpoints $u(\cdot, T)$.

$$\mathcal{L} = \underbrace{\|u_\theta - u_{\text{ref}}\|_{L^2}}_{\text{data}} + \lambda_{\text{PDE}} \underbrace{\|\partial_t u_\theta + u_\theta \partial_x u_\theta - \nu \partial_{xx} u_\theta\|_{L^2}}_{\text{physics}} + \lambda_{\text{BC}} \|\text{BC}(u_\theta)\|.$$

- or rollout

$$\mathcal{L} = \underbrace{\sum_{n=0}^{N-1} \alpha_n \|u_{n+1} - \phi_{\Delta t}(u_n)\|_{L^2}^2}_{\text{data rollout}} + \lambda_{\text{PDE}} \underbrace{\|\partial_t u_\theta + u_\theta \partial_x u_\theta - \nu \partial_{xx} u_\theta\|_{L^2}}_{\text{physics}} + \lambda_{\text{BC}} \|\text{BC}(u_\theta)\|.$$

Training, Physics, and Practical Tips

Stability & generalization

- Limit temporal bandwidth (K_t) or use AR with small Δt.
- Anti-aliasing / spectral filtering; weight sharing across time blocks.
- Randomize grids (space/time) at train time for resolution-agnosticity.

When to pick which

- *Smooth global dynamics, fixed T:* spacetime FNO (fast, single pass).
- *Variable horizons or control-in-time:* autoregressive FNO/GNO.
- *Irregular meshes or moving boundaries:* graph/attention operator with temporal links.

Neural Operators for SDE/SPDEs

From ODE/PDEs to SDE/SPDEs: What Changes?

- So far: **deterministic PDEs**, e.g.,

$$\partial_t u = \mathcal{F}(u, a), \quad u(\cdot, 0) = u_0.$$

Neural operator learns

$$\mathcal{G}_\theta : \ (u_0, a) \ \mapsto \ u(\cdot, t).$$

- For **stochastic differential equations (SDEs)**, dynamics are driven by noise:

$$dX_t = f(X_t, t)\, dt + g(X_t, t)\, dW_t,$$

where W_t is Brownian motion (or another stochastic process).

- **Stochastic PDE (SPDE):** add noise in time and space, e.g.

$$\partial_t u(x, t) = \mathcal{F}(u, a)(x, t) + \mathcal{G}(u, a)(x, t)\, \dot{W}(x, t),$$

where $\dot{W}(x, t)$ denotes space–time noise (formally the derivative of a space–time Wiener process, often idealised as space–time white noise).

From ODE/PDEs to SDE/SPDEs: What Changes?

- The solution is a **random path** $t \mapsto X_t(\omega)$. We care about:
  - individual sample paths (pathwise behavior),
  - or statistics: $\mathbb{E}[X_t]$, $\mathrm{Var}[X_t]$, full law p_{X_t}.
- Neural operators can learn the **solution map** (Itô map):

$$\mathcal{G}: \;\; (X_0, \text{noise}) \mapsto X(\cdot),$$

  using the same operator backbones (FNO, DeepONet, GNO) we use for PDEs.
- Solution is now a **random field** $(x, t) \mapsto u(x, t, \omega)$. We can learn:
  - *pathwise map*: a specific noise realization $\Rightarrow$ a field,
  - or *statistical map*: $(x, t) \mapsto$ quantities like $\mathbb{E}[u(x, t)]$, $\mathrm{Var}[u(x, t)]$, or even the full distribution of $u(x, t)$
- Neural operators still target an operator

$$\mathcal{G}: \;\; (u_0, \text{forcing/noise}) \mapsto u(\cdot, \cdot),$$

  just now in *space–time*.

SDE Solution as an Operator (Pathwise View)

- Consider an Itô SDE on $[0, T]$:

$$dX_t = f(X_t, t; \theta)\, dt + g(X_t, t; \theta)\, dW_t, \quad X_0 \sim p_0.$$

- After discretization (e.g. Euler–Maruyama with step Δt):

$$X_{n+1} = X_n + f(X_n, t_n)\, \Delta t + g(X_n, t_n)\, \Delta W_n,$$

  where $\{\Delta W_n\}$ are Gaussian increments.

- **Pathwise solution map (Itô map)**:

$$\mathcal{G} : \ (X_0, \{\Delta W_n\}_{n=0}^{N-1}) \longmapsto \{X_n\}_{n=0}^{N}.$$

  Deterministic map once the noise path is fixed.

- **Neural operator perspective:**
  - Input function(s): noise path, control signals, parameters θ.
  - Output function: state trajectory $X(\cdot)$ (possibly multi-dimensional).
  - Use a time-aware operator backbone (spacetime FNO, AR-FNO, GNO-in-time) to approximate $\mathcal{G}$.

Neural Operator Blueprints for SDEs

1. **Pathwise SDE Neural Operator (trajectory generator)**

$$\mathcal{G}_\theta : \ (X_0, \{\Delta W_n\}) \ \mapsto \ \{X_n\}_{n=0}^N$$

   - Treat $\{\Delta W_n\}$ as an *input function in time*.
   - Use spacetime FNO or autoregressive FNO/GNO as in the Burgers setup.
   - Sample new trajectories by sampling new noise paths.

2. **Statistic / moment operator**

$$\mathcal{H}_\theta : \ (X_0, \theta) \ \mapsto \ \big(m(t), s^2(t)\big)_{t\in[0,T]}$$

   - Learn map to mean/variance curves or other summary functionals.
   - Useful when full path samples are not required.

3. **Example: OU process with FNO**

$$dX_t = \alpha(\mu - X_t)\, dt + \sigma\, dW_t.$$

SPDE Solution as a Space–Time Operator

- Consider an SPDE on $\mathcal{D} \times [0, T]$:

$$\partial_t u(x,t) = \mathcal{L}u(x,t) + \mathcal{N}(u,a)(x,t) + \mathcal{G}(u,a)(x,t)\,\dot{W}(x,t),$$

  with $u(x,0) = u_0(x)$ and boundary conditions on $\partial\mathcal{D}$.

- After discretization (space + time), e.g. finite difference + Euler–Maruyama:

$$u_{n+1} = \Phi\big(u_n,\ a,\ \xi_n\big),$$

  where $\xi_n(x)$ are random noise fields on the spatial grid at time t_n.

- **Pathwise solution map** (Itô map for SPDE):

$$\mathcal{G} :\ \big(u_0,\ \{\xi_n\}_{n=0}^{N-1}\big) \longmapsto \{u_n\}_{n=0}^{N}.$$

  Once the noise fields $\{\xi_n\}$ are fixed, this is deterministic.

- **Neural operator perspective:**
  - Inputs are functions on space–time: $u_0(x)$, $a(x)$, noise field(s).
  - Output is a function on space–time: $u(x,t)$.
  - Use the same backbones: FNO, DeepONet, GNO, attention operators.

Neural Operator Blueprints for SPDEs (FNO / GNO)

1. **Spacetime FNO with stochastic forcing**
   - Treat (x, t) as a 2D domain and stack channels:
   $$\text{channels: } [\, u(x,t),\ a(x),\ \text{noise}(x,t),\ \text{coords}\,].$$
   - Apply 2D FNO layers over (x, t):
   $$u^{(\ell+1)} = \sigma\Big(W_0 u^{(\ell)} + \mathcal{F}^{-1}\big[W_\ell(k_x, k_t)\, \widehat{u^{(\ell)}}\big]\Big).$$
   - Learn the map $(u_0, a, \text{noise}) \mapsto u(\cdot,\cdot)$ in one shot.
2. **Autoregressive spatial operator with noisy increments**
   - Time is stepped; space is handled by FNO / GNO:
   $$u_{n+1}(x) = \phi_\theta\big(u_n(\cdot), a(\cdot), \xi_n(\cdot)\big)(x),$$
   where $\xi_n(\cdot)$ is the noise field at step n.
   - Roll out $u_0 \to u_1 \to \cdots \to u_N$; train with multi-step losses.

SDE vs SPDE Neural Operators (What's Really Different?)

- **SDE:** state is finite-dimensional, $X_t \in \mathbb{R}^d$.

$$dX_t = f(X_t, t)\, dt + g(X_t, t)\, dW_t.$$
$$\mathcal{G}_\theta : \ \big(X_0, \{\Delta W_n\}\big) \ \mapsto \ \{X_n\}_{n=0}^N.$$

  - Operator on *time paths* $[0, T] \to \mathbb{R}^d$.
  - Use 1D (in t) operator backbones (FNO-in-time, AR operators).
- **SPDE:** state is a field, $u : \mathcal{D} \times [0, T] \to \mathbb{R}^m$.

$$\partial_t u(x, t) = \mathcal{F}(u, a)(x, t) + \mathcal{G}(u, a)(x, t)\, \dot{W}(x, t).$$
$$\mathcal{G}_\theta : \ \big(u_0(\cdot), a(\cdot), \{\xi_n(\cdot)\}\big) \ \mapsto \ \{u_n(\cdot)\}_{n=0}^N.$$

  - Operator on *space–time fields* $\mathcal{D} \times [0, T] \to \mathbb{R}^m$.
  - Use spacetime FNO, or spatial FNO/GNO + temporal stepping.
- **Same philosophy:** learn the Itô map (noise $\Rightarrow$ solution), but on different function spaces (time vs space–time).

Q & A

All questions are welcome. Thank you!