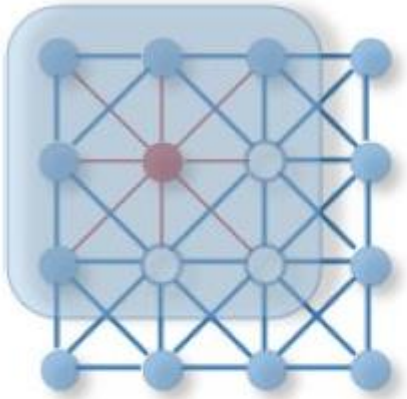
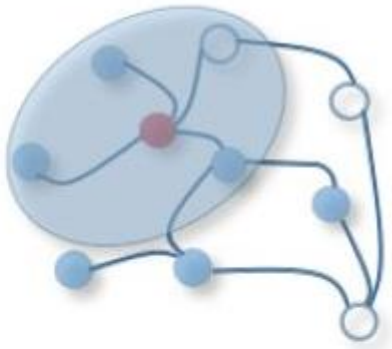
1. Neural operator Learning maps between function spaces
2. Neural operator Graph kernel network for partial differential equations
3. Fourier Neural Operator for Parametric Partial Differential Equations

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

Graph convolution neural network



(a) 2D Convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size.



(b) Graph Convolution. To get a hidden representation of the red node, one simple solution of graph convolution operation takes the average value of node features of the red node along with its neighbors. Different from image data, the neighbors of a node are unordered and variable in size.

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\}$$

⇩

	传统Fourier变换	Graph Fourier变换
*Fourier*变换基	$e^{-2\pi ixv}$	U^T
逆*Fourier*变换基	$e^{2\pi ixv}$	U
维度	∞	点的个数n

⇩

$$L = U\Lambda U^T$$

⇩

$$L = D - A$$

GCN → maps between function spaces?
Differential equations?

Problem settings

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

$$\min_{\theta \in \Theta} \mathbb{E}_{a \sim \mu}[C(\mathcal{F}(a, \theta), \mathcal{F}^{\dagger}(a))]$$

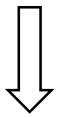
In differential equations:

$$\begin{aligned} (\mathcal{L}_a u)(x) &= f(x), && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned}$$

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

Green function

$$LG = \delta$$



$$L\varphi = Q \qquad \varphi = \varphi * \delta = \varphi * (LG) = (L\varphi) * G = Q * G$$

$$\begin{aligned} (\mathcal{L}_a u)(x) &= f(x), && x \in D \\ u(x) &= 0, && x \in \partial D \end{aligned} \longrightarrow u(x) = \int_D G_a(x,y) f(y)\, dy.$$

Final convolution layers: $$v_{t+1}(x) := \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\phi) v_t\big)(x)\Big), \qquad \forall x \in D$$

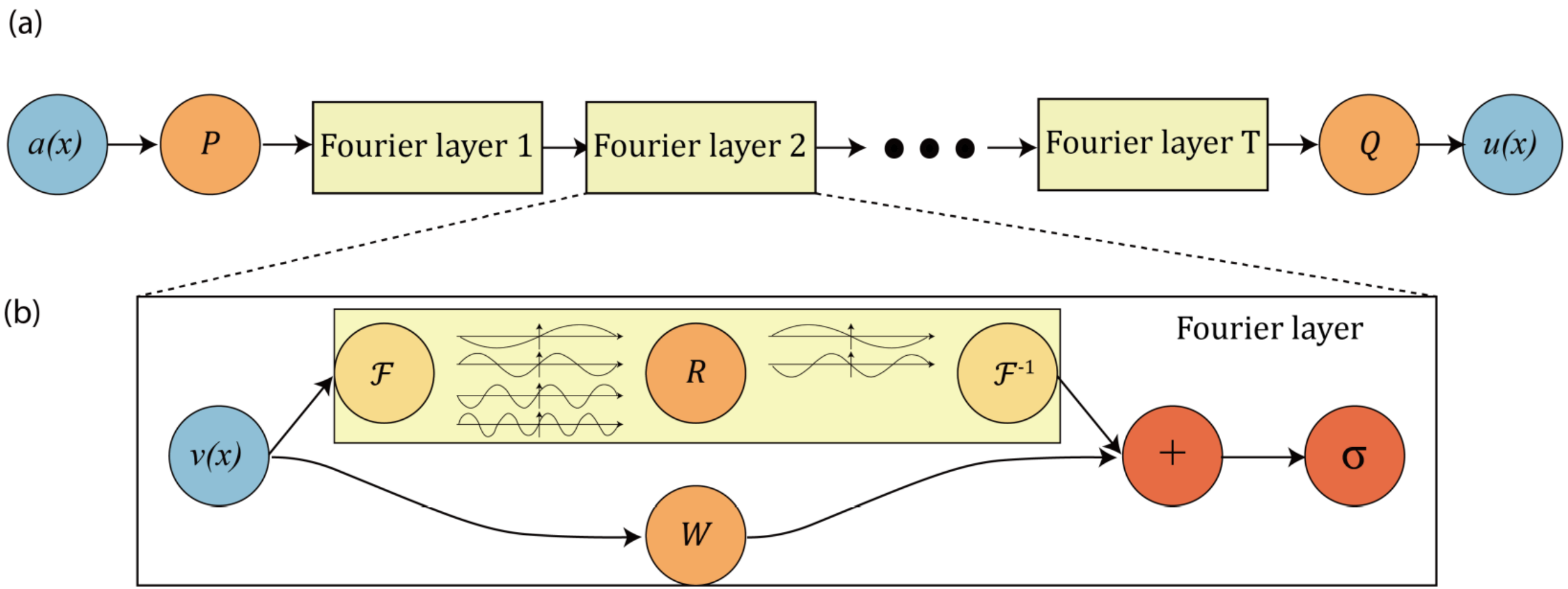
Fourier neural operator

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

$$f * g = \mathcal{F}^{-1}\{\mathcal{F}\{f\} \cdot \mathcal{F}\{g\}\} \longrightarrow \big(\mathcal{K}(a;\phi)v_t\big)(x) = \mathcal{F}^{-1}\big(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(v_t)\big)(x), \qquad \forall x \in D.$$

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

Connection between GCN? $\quad g_\theta(x) = U^\top \mathbf{diag}(\theta) U x$



(a) The full architecture of neural operator: start from input a. 1. Lift to a higher dimension channel space by a neural network P. 2. Apply four layers of integral operators and activation functions. 3. Project back to the target dimension by a neural network Q. Output u. **(b) Fourier layers**: Start from input v. On top: apply the Fourier transform $\mathcal{F}$; a linear transform R on the lower Fourier modes and filters out the higher modes; then apply the inverse Fourier transform $\mathcal{F}^{-1}$. On the bottom: apply a local linear transform W.

Figure 2: **top:** The architecture of the neural operators; **bottom:** Fourier layer.

Transformer

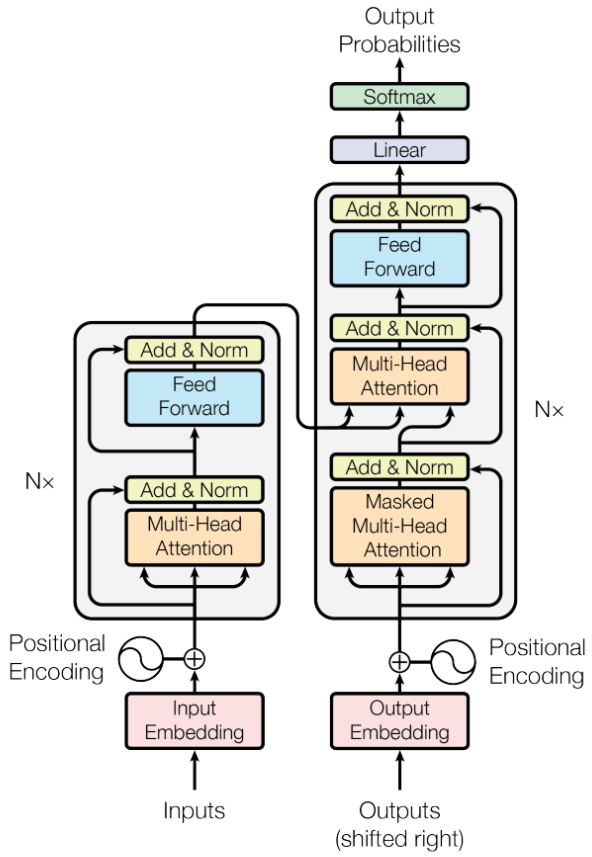


Figure 1: The Transformer - model architecture.

$$u(x) = \sigma\left(v(x) + \int_D \kappa_v(x, y, v(x), v(y))v(y)\,\mathrm{d}y\right) \qquad \forall x \in D$$

$$\kappa_v(v(x), v(y)) = g_v(v(x), v(y))R$$

What if:

$$g_v(v(x), v(y)) = \left(\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right)\mathrm{d}s\right)^{-1} \exp\left(\frac{\langle Av(x), Bv(y)\rangle}{\sqrt{m}}\right).$$

$$\Downarrow$$

$$u(x) = \sigma\left(v(x) + \int_D \frac{\exp\left(\frac{\langle Av(x), Bv(y)\rangle}{\sqrt{m}}\right)}{\int_D \exp\left(\frac{\langle Av(s), Bv(y)\rangle}{\sqrt{m}}\right)\mathrm{d}s} Rv(y)\,\mathrm{d}y\right) \qquad \forall x \in D.$$

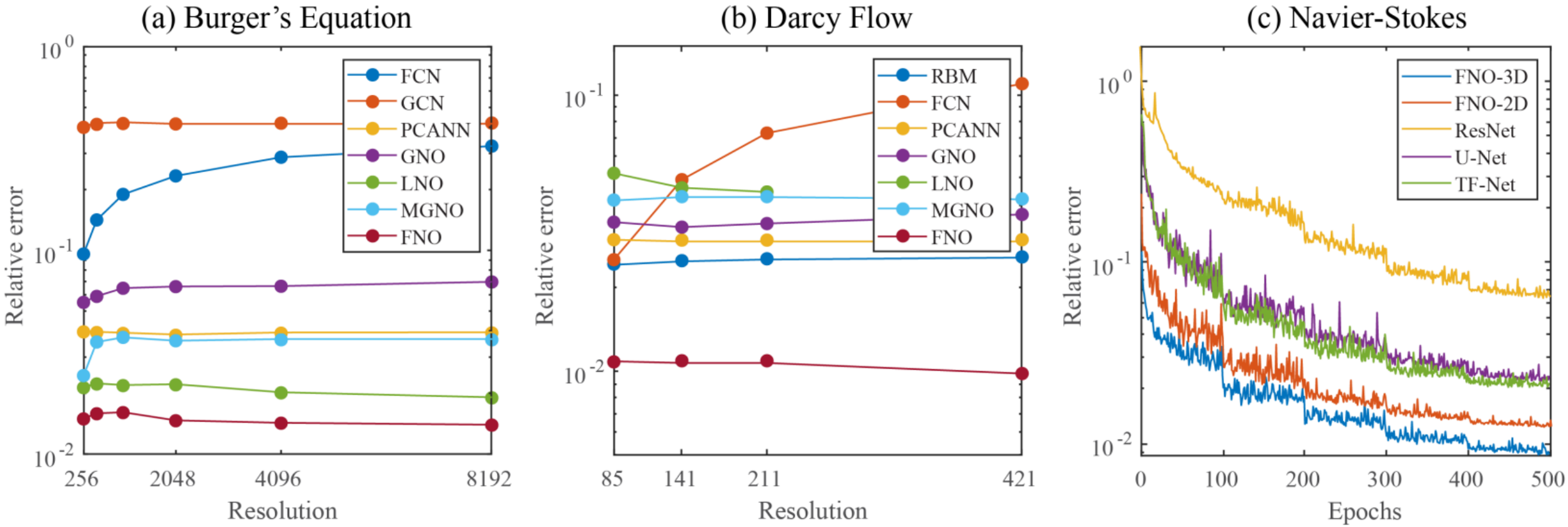
$$\Downarrow$$

$$u_j = \sigma\left(v_j + \sum_{q=1}^{k} \frac{\exp\left(\frac{\langle Av_j, Bv_q\rangle}{\sqrt{m}}\right)}{\sum_{l=1}^{k} \exp\left(\frac{\langle Av_l, Bv_q\rangle}{\sqrt{m}}\right)} Rv_q\right), \qquad j = 1, \ldots, k.$$

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

$$\Leftrightarrow$$

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Left: benchmarks on Burgers equation; **Mid:** benchmarks on Darcy Flow for different resolutions; **Right:** the learning curves on Navier-Stokes $\nu = 1\text{e}{-3}$ with different benchmarks. Train and test on the same resolution. For acronyms, see Section 5; details in Tables 1, 3, 4.

Table 1: Benchmarks on Navier Stokes (fixing resolution 64×64 for both training and testing)

Config	**Parameters**	**Time per epoch**	$\nu = 1\text{e}{-3}$ $T = 50$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 10000$	$\nu = 1\text{e}{-5}$ $T = 20$ $N = 1000$
FNO-3D	$6,558,537$	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	$414,517$	$127.80s$	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	$24,950,491$	$48.67s$	0.0245	0.2051	0.1190	0.1982
TF-Net	$7,451,724$	$47.21s$	0.0225	0.2253	0.1168	0.2268
ResNet	$266,641$	$78.47s$	0.0701	0.2871	0.2311	0.2753