Fourier Neural Operator for Parametric Partial Differential Equations

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

Eren Homburg

Luca Apolloni

PDEs TODAY

Challenges in Science and Engineering

Solving complicated partial differential equations (PDE) systems is extremely important

Examples

Solving PDEs

PDEs TODAY

Challenges in Science and Engineering

Solving complicated partial differential equations (PDE) systems is extremely important

Examples

Solving PDEs

PDEs TODAY

Challenges in Science and Engineering

Examples

Molecular dynamics, micro-mechanics, turbulent flows, etc.

Solving PDEs

PDEs TODAY

Challenges in Science and Engineering

Examples

Molecular dynamics, micro-mechanics, turbulent flows, etc.

Solving PDEs

PDEs TODAY

Challenges in Science and Engineering

Examples

Solving PDEs

Conventional solvers vs data-driven methods

Conventional solvers VS Data-Driven methods

Conventional solvers

VS

Data-Driven methods

Conventional solvers

Examples:

Finite Element Methods (FEM)

Finite Difference Methods (FDM)

Trade-off:

Coarse Grids: Faster but less accurate

Fine Grids: Accurate but slow

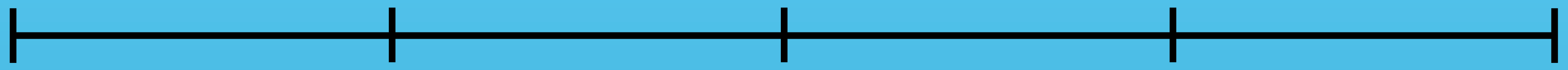
Challenge:

Complicated PDEs require fine discretization to capture the phenomenon

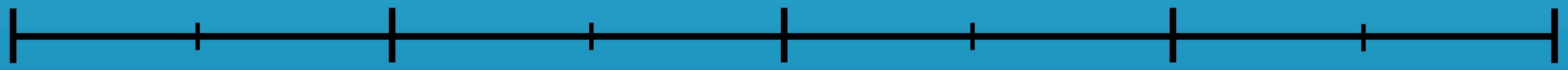
VS

Data-Driven methods

Coarse grid



Fine grid



Conventional solvers

Examples:

Finite Element Methods (FEM)

Finite Difference Methods (FDM)

Trade-off:

Coarse Grids: Faster but less accurate

Fine Grids: Accurate but slow

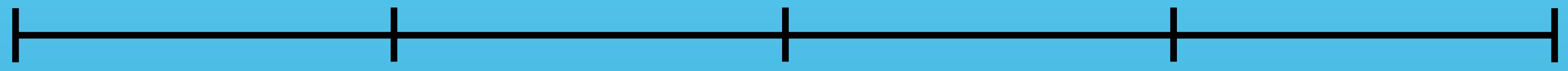
Challenge:

Complicated PDEs require fine discretization to capture the phenomenon

VS

Data-Driven methods

Coarse grid



Fine grid

Conventional solvers

Examples:

- Finite Element Methods (FEM)
- Finite Difference Methods (FDM)

Trade-off:

- **Coarse Grids:** Faster but less accurate
- **Fine Grids:** Accurate but slow

Challenge:

- Complicated PDEs require fine discretization to capture the phenomenon

VS

Data-Driven methods

Direct Learning:

- Learn trajectories directly from data
- Better runtime

Challenge:

- Classical Neural Networks (NNs) still limited by discretization

Solution:

- Develop discretization-invariant Neural Operators (NOs)

PAPER’S WORK

- New neural operator
- Parametrize the integral kernel in the Fourier space

Experiments on Burgers’ equation, Darcy flow, and Navier-Stokes equation

- Three orders of magnitude faster
- Superior accuracy

- New neural operator
- Parametrize the integral kernel in the Fourier space

New neural operator

NEURAL OPERATORS

Key features

- Infinite-Dimensional Operator
- Unified Parameters
  → Discretization-Invariance
    → Solution Transfer
- Data-Driven

New neural operator

- New neural operator
- Parametrize the integral kernel in the Fourier space

Fourier space

FOURIER SPACE

Differentiation → Multiplication

FOURIER SPACE

Differentiation → Multiplication

Potentially a lot faster!

- New neural operator
- Parametrize the integral kernel in the Fourier space

PAPER'S WORK

- New neural operator
- Parametrize the integral kernel in the Fourier space

Experiments on Burgers' equation, Darcy flow, and Navier-Stokes equation

- Three orders of magnitude faster
- Superior accuracy

PAPER’S WORK

- New neural operator
- Parametrize the integral kernel in the Fourier space

Experiments on Burgers’ equation, Darcy flow, and Navier-Stokes equation

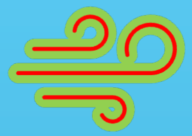
- Three orders of magnitude faster
- Superior accuracy

Experiments on Burgers' equation, Darcy flow, and Navier-Stokes equation

Navier-Stokes equation

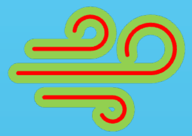
NAVIER-STOKES EQUATION

- Models fluid dynamics



NAVIER-STOKES EQUATION

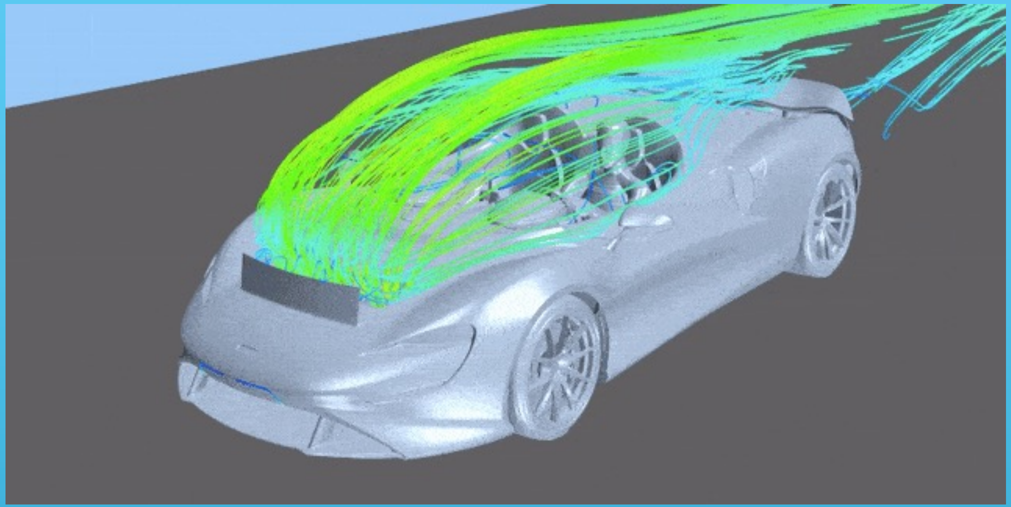
- Models fluid dynamics



- Extremely useful
  - Airflow? → Planes, weather, etc.
  - Blood flow? → Medical diagnostics tools
  - Etc.

NAVIER-STOKES EQUATION

- Models fluid dynamics
- Extremely useful
  - ❖ Airflow? → Planes, weather, etc.
  - ❖ Blood flow? → Medical diagnostics tools
  - ❖ Etc.



NAVIER-STOKES EQUATION

- Models fluid dynamics

- Extremely useful
  - ❖ Airflow? → Planes, weather, etc.
  - ❖ Blood flow? → Medical diagnostics tools
  - ❖ Etc.

- Can be tweaked to model non-Newtonian fluids and turbolence (Reynolds number)

NAVIER-STOKES EQUATION

- Models fluid dynamics
- Extremely useful
  - Airflow? → Planes, weather, etc.
  - Blood flow? → Medical diagnostics tools
  - Etc.
- Can be tweaked to model non-Newtonian fluids and turbolence (Reynolds number)

NAVIER-STOKES EQUATION

$$\begin{cases} \partial_t w(x,t) + u(x,t) \cdot \nabla w(x,t) = \nu \Delta w(x,t) + f(x) \\ \nabla \cdot u(x,t) = 0 \\ w(x,0) = w_0(x) \end{cases}$$

NAVIER-STOKES EQUATION

$$\begin{cases} \partial_t \boldsymbol{w}(x,t) + \boldsymbol{u}(x,t) \cdot \nabla \boldsymbol{w}(x,t) = \boldsymbol{\nu} \Delta \boldsymbol{w}(x,t) + \boldsymbol{f}(x) \\ \nabla \cdot \boldsymbol{u}(x,t) = 0 \\ \boldsymbol{w}(x,0) = \boldsymbol{w}_{\boldsymbol{0}}(x) \end{cases}$$

$\boldsymbol{u}$: velocity field

$\boldsymbol{w}$: vorticity

∇: gradient

Δ: Laplacian

$\boldsymbol{\nu}$: viscosity coefficient

$\boldsymbol{f}$: forcing function

NAVIER-STOKES EQUATION

$$\begin{cases} \partial_t \boldsymbol{w}(x,t) + \boldsymbol{u}(x,t) \cdot \nabla \boldsymbol{w}(x,t) = \boldsymbol{\nu} \Delta \boldsymbol{w}(x,t) + \boldsymbol{f}(x) \\ \nabla \cdot \boldsymbol{u}(x,t) = 0 \\ \boldsymbol{w}(x,0) = \boldsymbol{w}_{\mathbf{0}}(x) \end{cases}$$

$\boldsymbol{u}$: velocity field

$\boxed{\boldsymbol{w} : \text{vorticity}}$

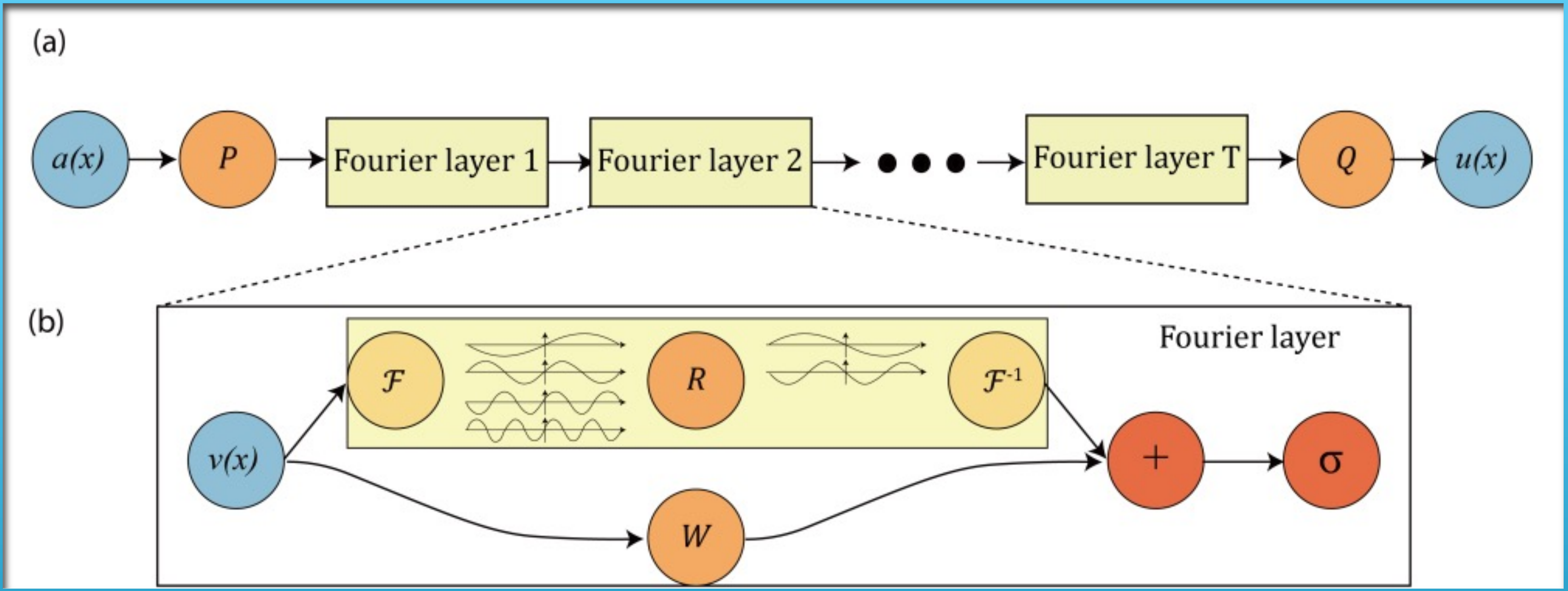
∇ : gradient

Δ: Laplacian

$\boldsymbol{\nu}$: viscosity coefficient

$\boldsymbol{f}$: forcing function

OPERATOR LEARNING



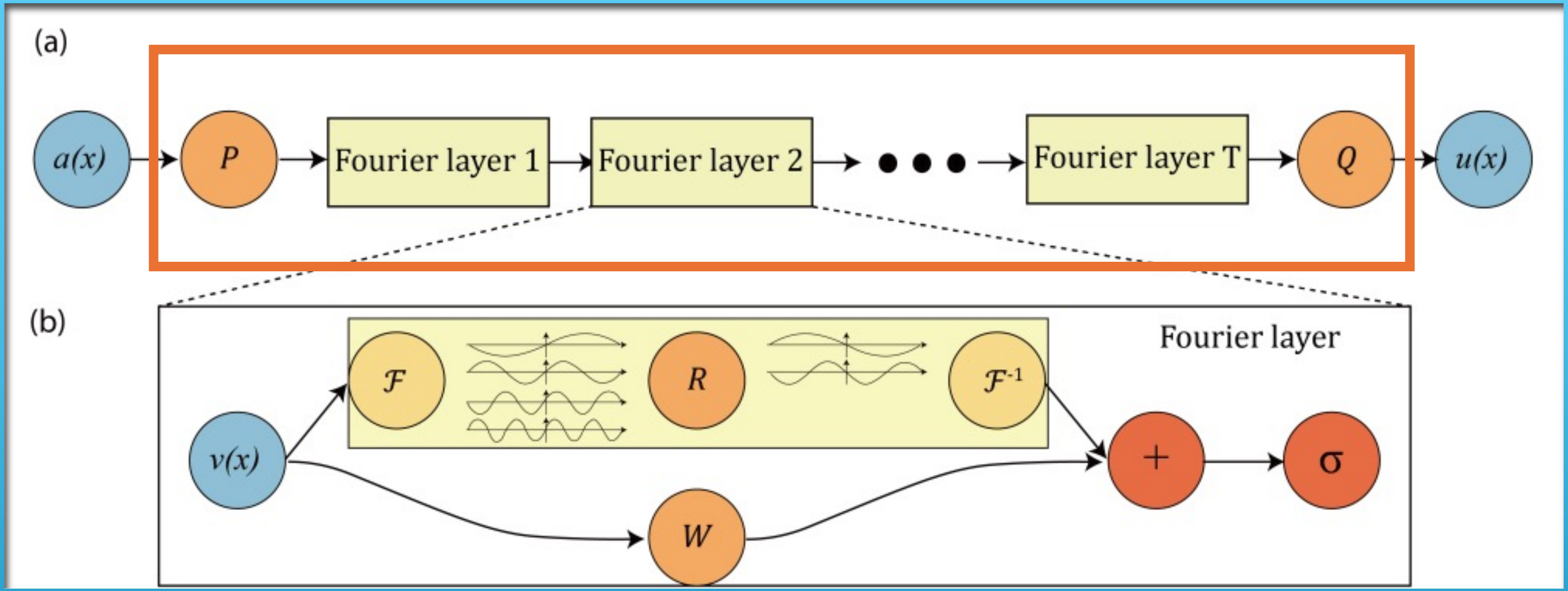
GOAL

Learn a mapping $\boldsymbol{G}^{\dagger}$

Inputs $\mathbf{a}(\mathbf{x}) \in \mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$

Outputs $\mathbf{u}(x) \in \mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$

OPERATOR LEARNING



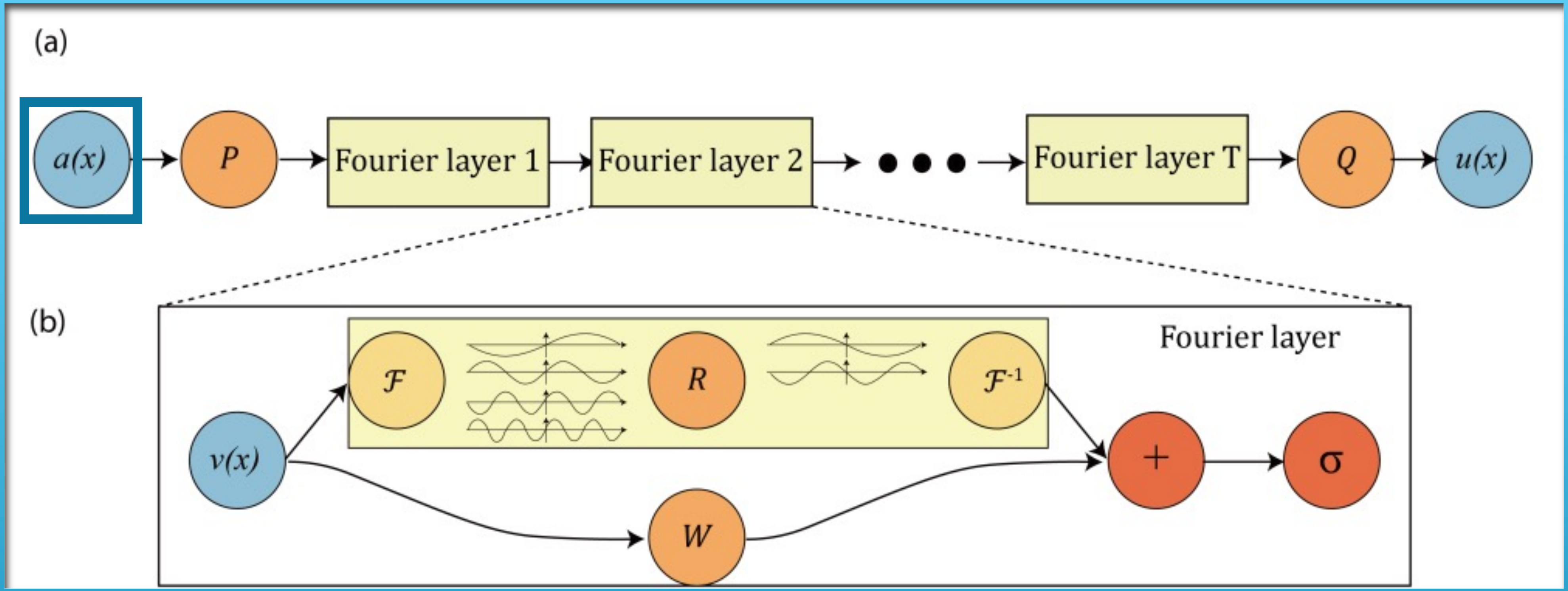
GOAL

Learn a mapping $G^{\dagger}$

Inputs $\mathbf{a}(\mathbf{x}) \in \mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$

Outputs $\mathbf{u}(x) \in \mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$

OPERATOR LEARNING



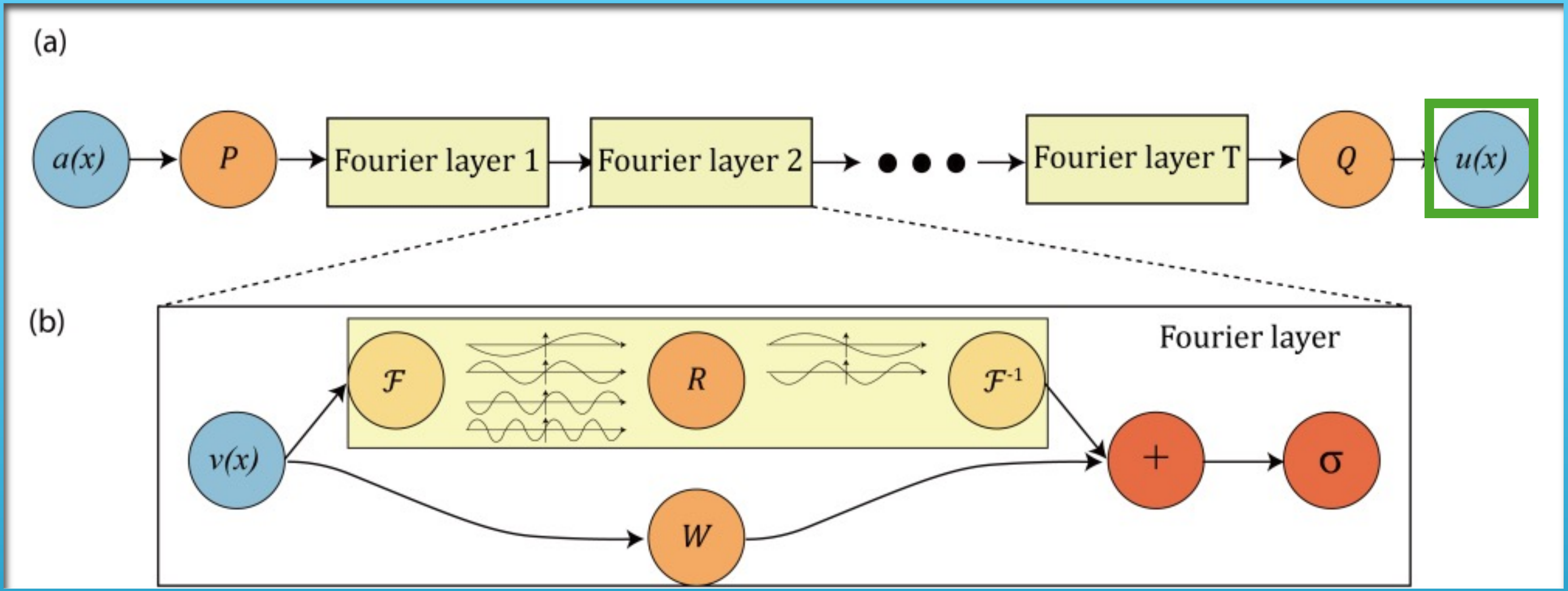
GOAL

Learn a mapping $G^{\dagger}$

Inputs $\mathbf{a}(\mathbf{x}) \in \mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$

Outputs $\mathbf{u}(x) \in \mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$

OPERATOR LEARNING



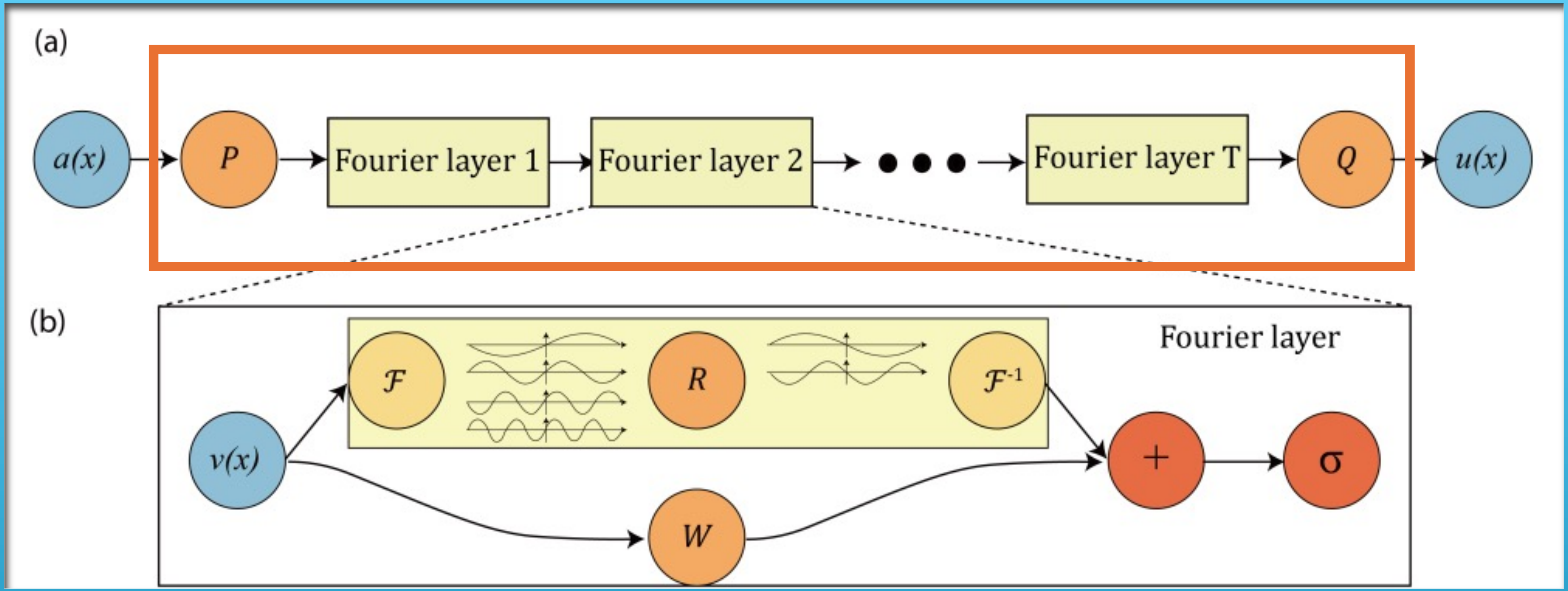
GOAL

Learn a mapping $G^{\dagger}$

Inputs $\mathbf{a}(\mathbf{x}) \in \mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$

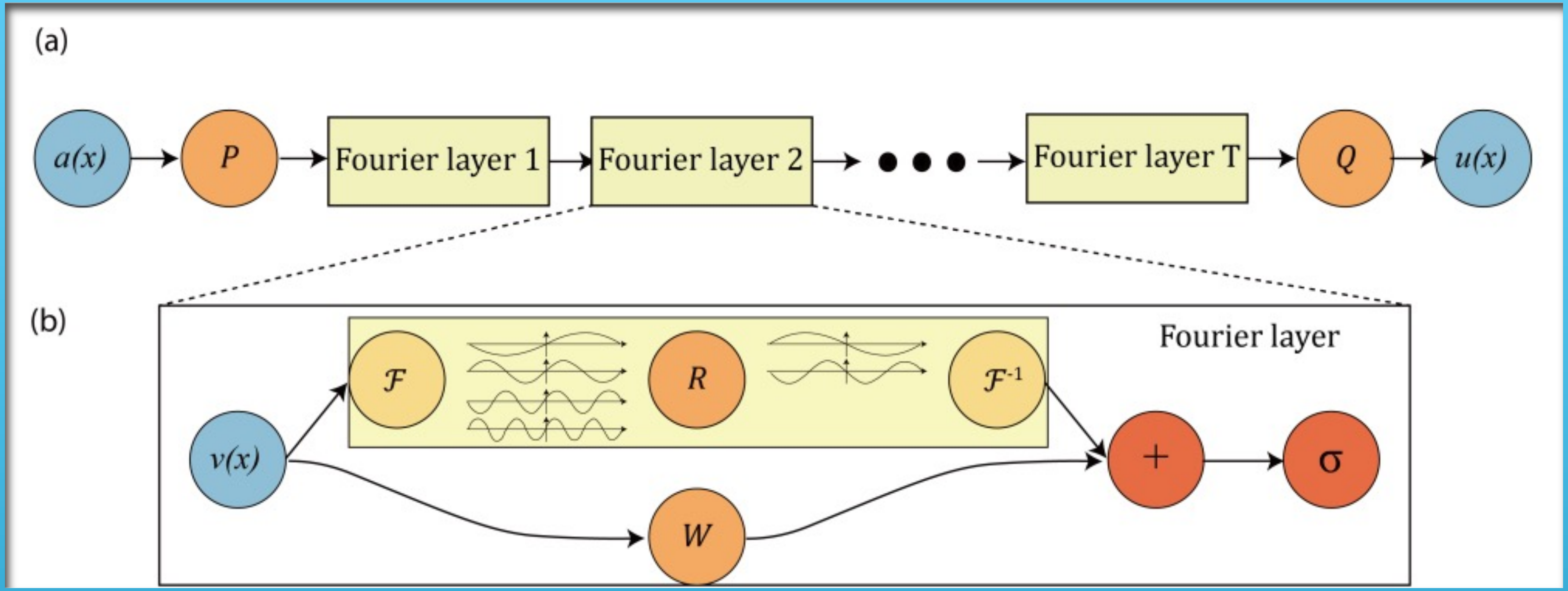
Outputs $\mathbf{u}(x) \in \mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$

OPERATOR LEARNING

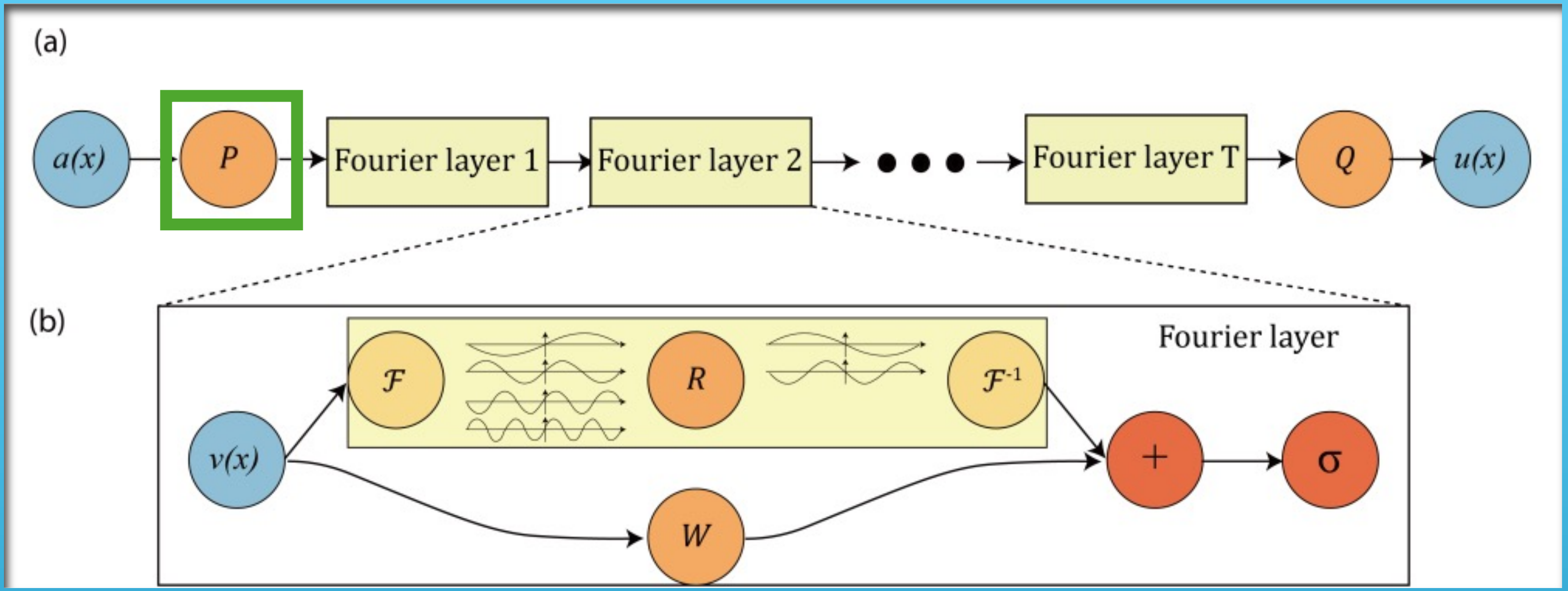


HOW Construct parametric map such that $G_{\theta^\dagger}$ is as close as possible to $G^\dagger$.

THE NEURAL OPERATOR

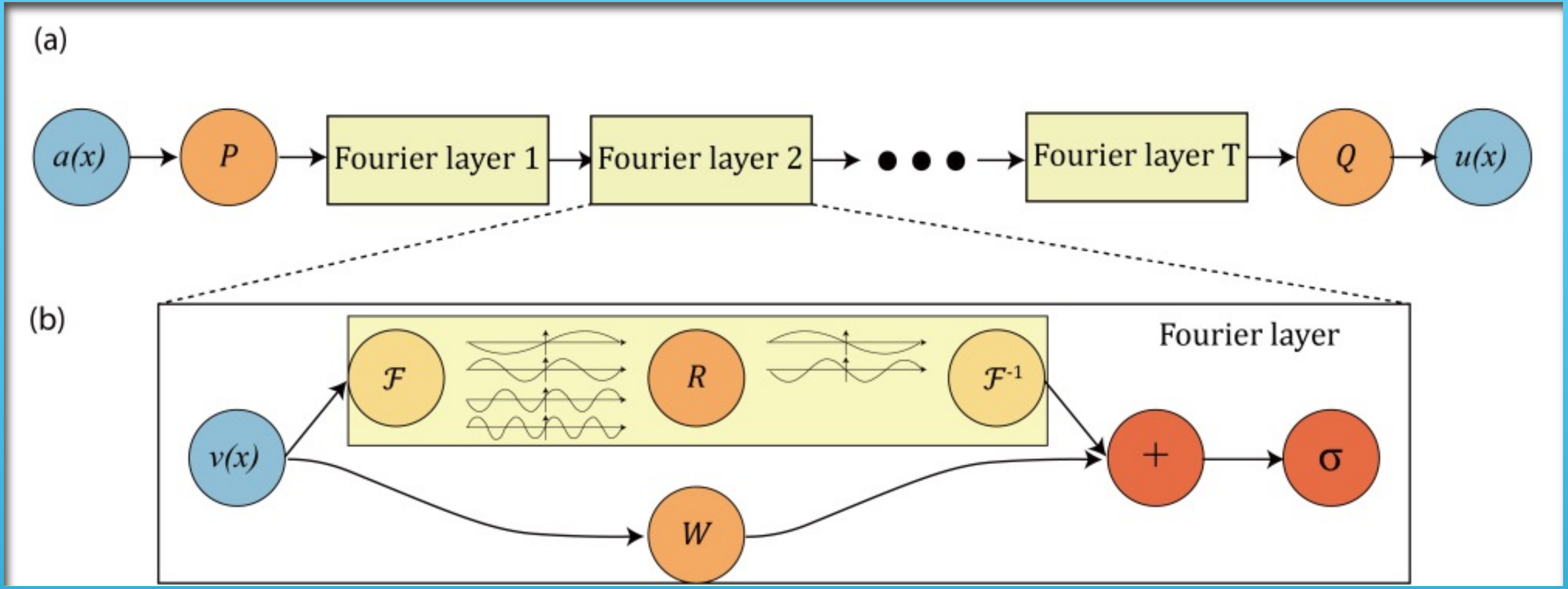


THE NEURAL OPERATOR

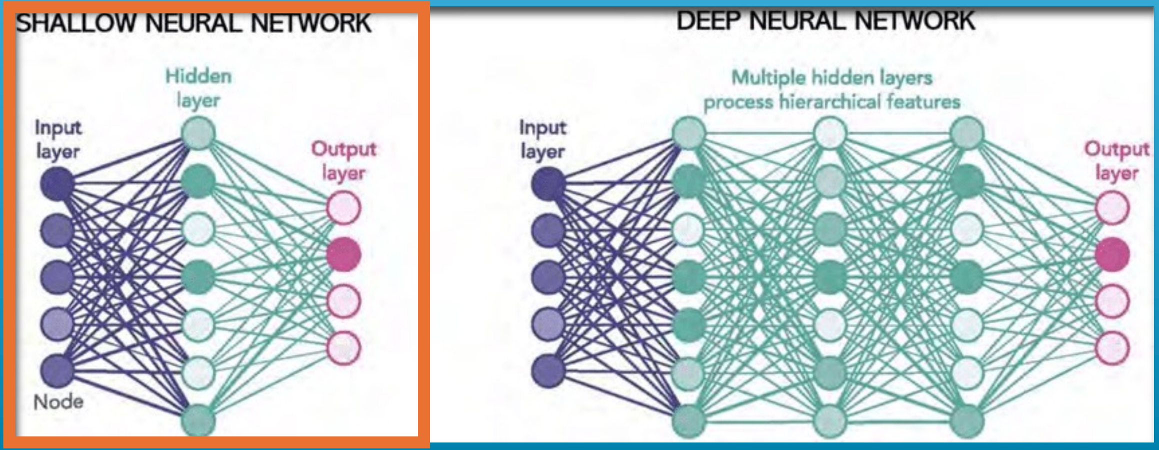


Lift input to higher dimension:
$v_0(x) = P(a(x))$

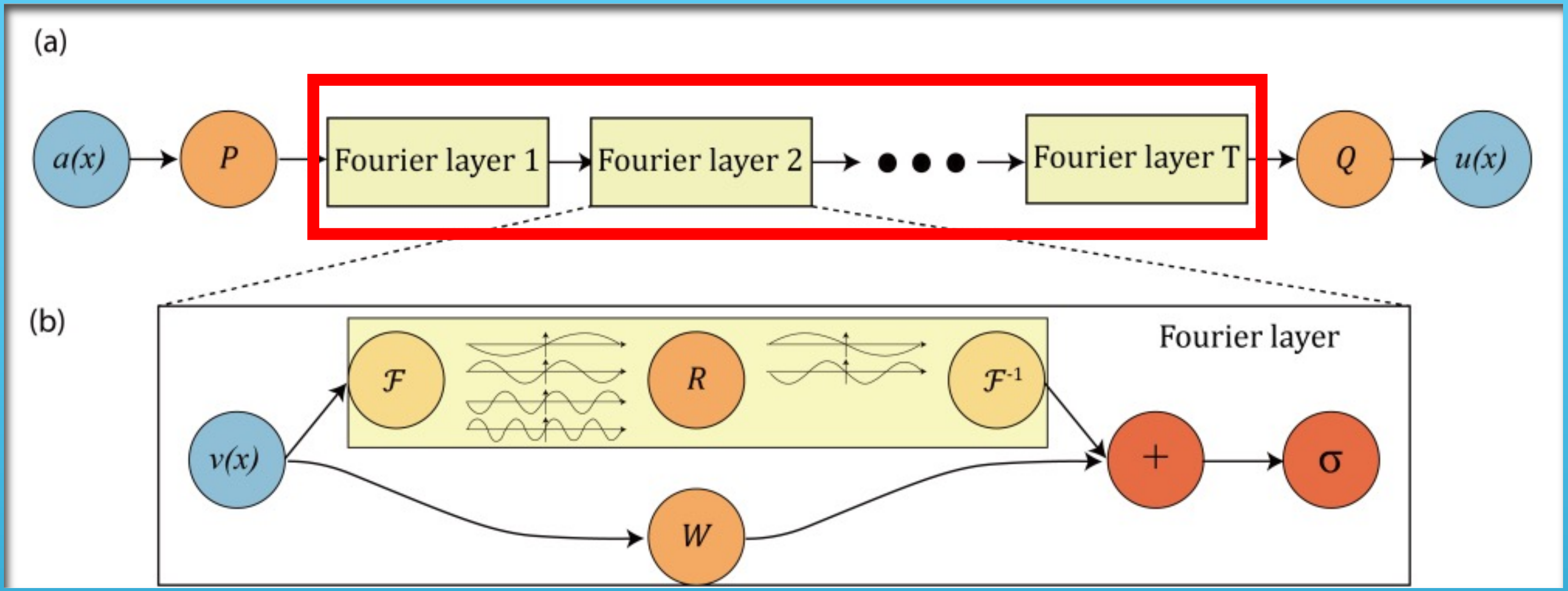
THE NEURAL OPERATOR



$P: \mathbb{R}^{d_u} \mapsto \mathbb{R}^{d_v}$, parametrized by a **shallow fully-connected NN**



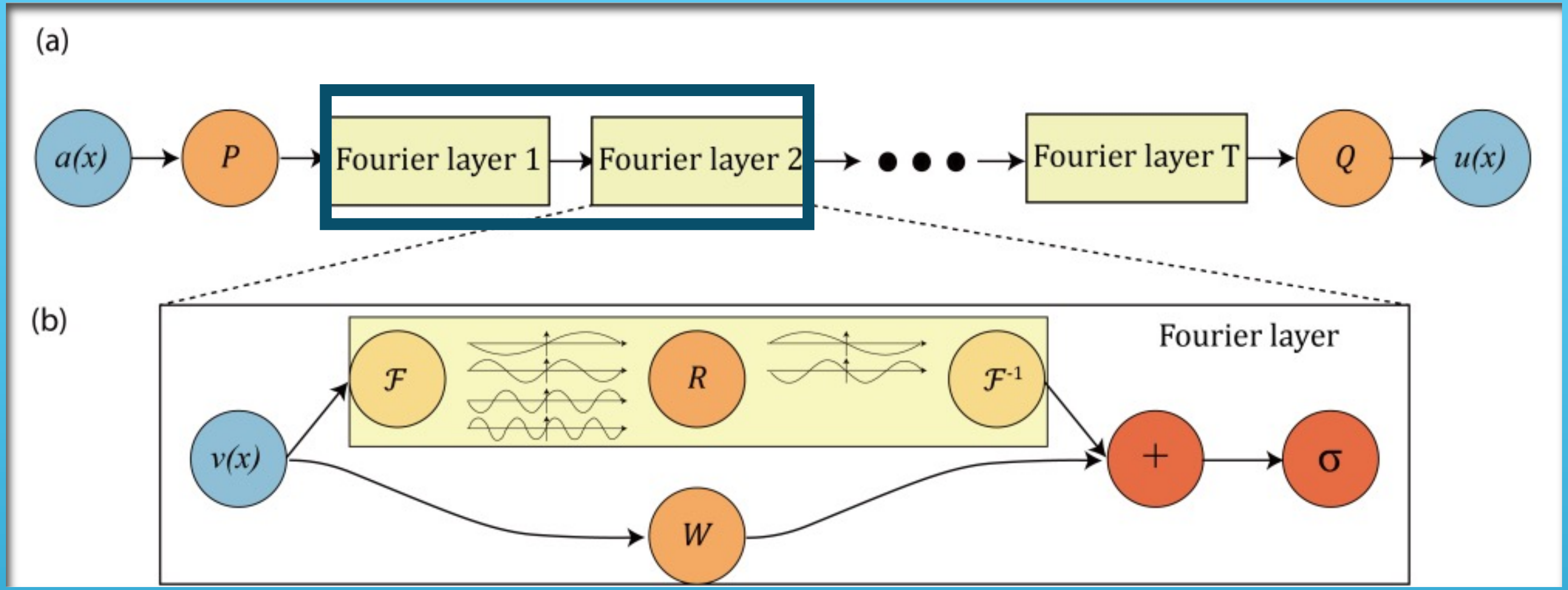
THE NEURAL OPERATOR



Iterative architecture:

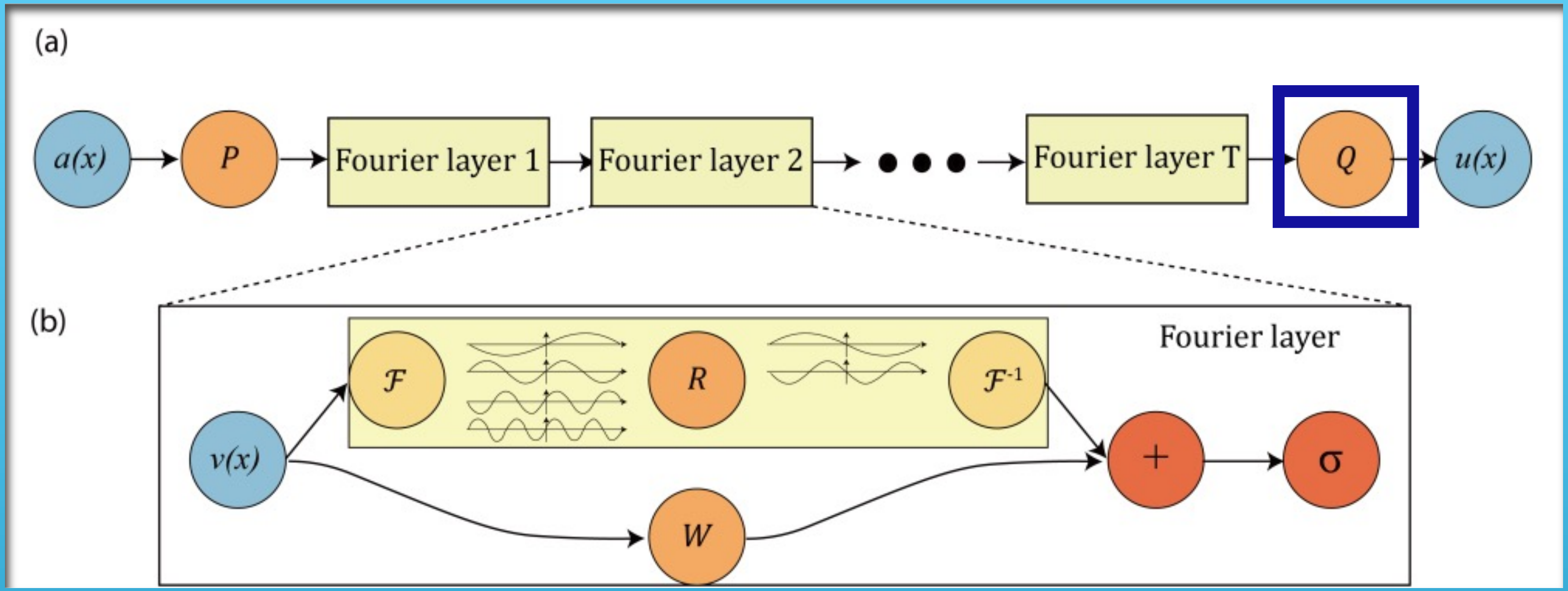
$v_0(x) \longmapsto \ \dots \ \mapsto \ v_T(x), x \in \ \mathbb{R}^{d_v}$

THE NEURAL OPERATOR



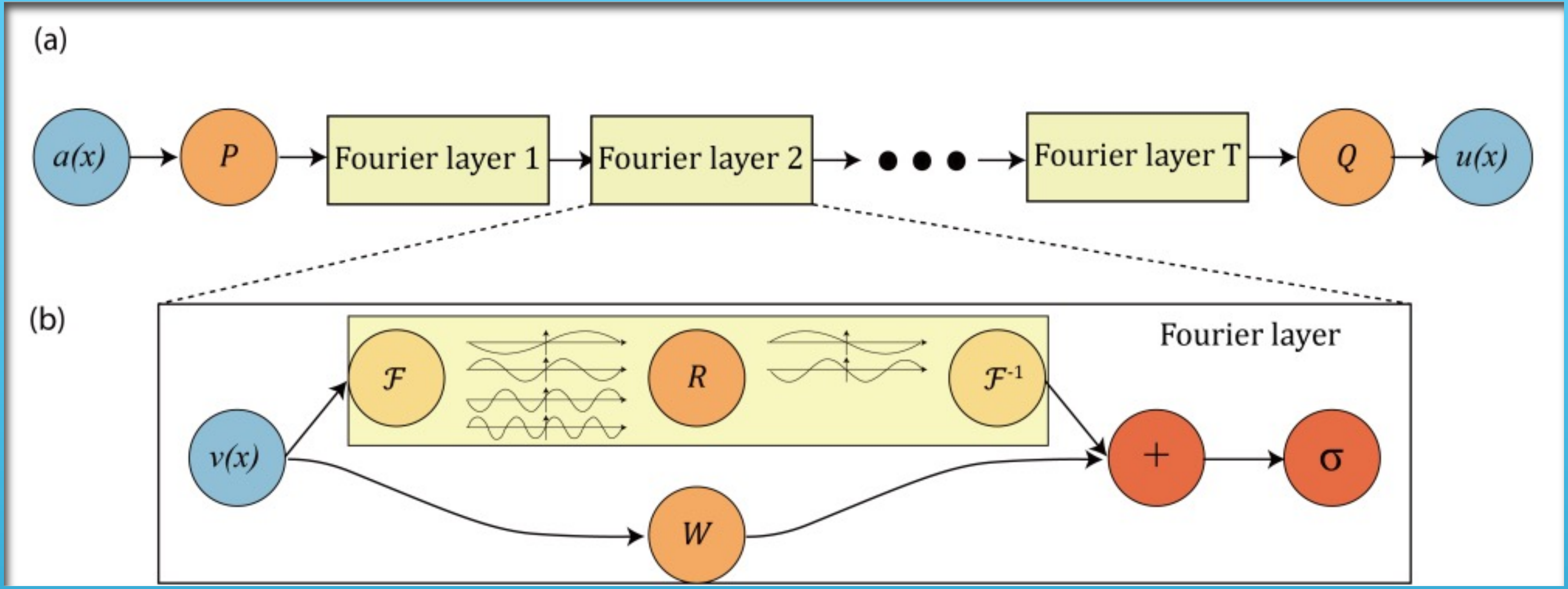
Iteration: $v_t \longmapsto v_{t+1}$

THE NEURAL OPERATOR



The $\boxed{\textbf{output } \boldsymbol{u(x) = Q(v_T(x))}}$,
$Q: \mathbb{R}^{d_v} \mapsto \mathbb{R}^{d_u}$

THE NEURAL OPERATOR



Lift input to higher dimension:
$v_0(x) = P(a(x))$

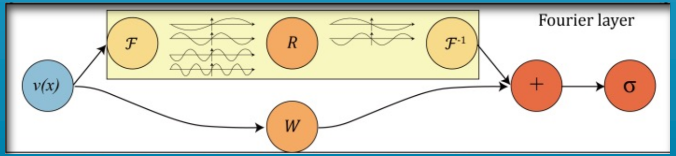
$P: \mathbb{R}^{d_u} \mapsto \mathbb{R}^{d_v}$, parametrized by a **shallow fully-connected NN**

Iterative architecture:
$v_0(x) \longmapsto \dots \mapsto v_T(x), x \in \mathbb{R}^{d_v}$

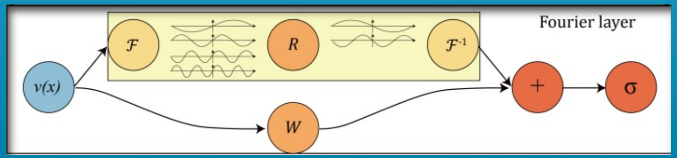
Iteration: $v_t \longmapsto v_{t+1}$

The **output** $u(x) = Q(v_T(x))$,
$Q: \mathbb{R}^{d_v} \mapsto \mathbb{R}^{d_u}$

$$v_{t+1}(x) \coloneqq \sigma\Big(Wv_t(x) + \big(\mathcal{K}(a;\, \phi)v_t\big)(x)\Big), \qquad \forall x \in D$$



$$v_{t+1}(x) \coloneqq \sigma\big(Wv_t(x) + (\mathcal{K}(a;\ \phi)v_t)(x)\big), \qquad \forall x \in D$$

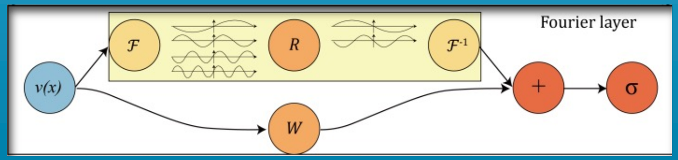


$$v_{t+1}(x) := \boldsymbol{\sigma}\Big(W v_t(x) + \big(\mathcal{K}(a;\ \phi)v_t\big)(x)\Big), \qquad \forall x \in D$$

$\sigma : \mathbb{R} \to \mathbb{R}$

Non-linear activation function

Component-wise operations

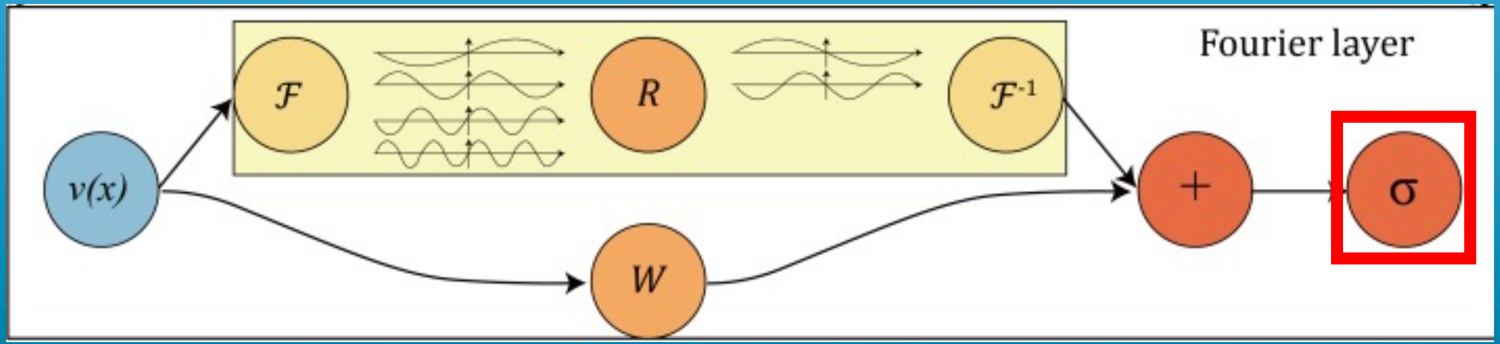


$$v_{t+1}(x) \coloneqq \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\,\phi)v_t\big)(x)\Big), \qquad \forall x \in D$$

$\sigma : \mathbb{R} \to \mathbb{R}$

Non-linear activation function

Component-wise operations



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

$\sigma : \mathbb{R} \to \mathbb{R}$

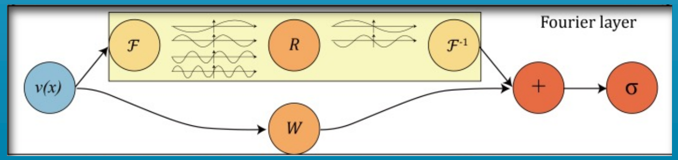
Non-linear activation function

Component-wise operations

$W: \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$

Linear transformation

Spatial domain



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

$\sigma : \mathbb{R} \to \mathbb{R}$

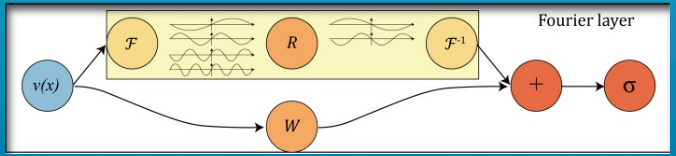
Non-linear activation function

Component-wise operations

$W: \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$

Linear transformation

Spatial domain



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

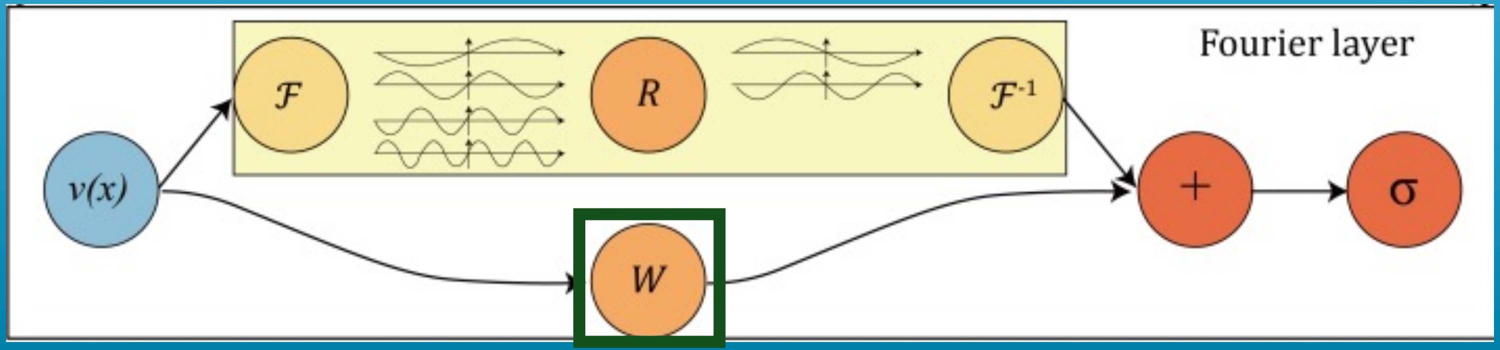
$\sigma : \mathbb{R} \to \mathbb{R}$

Non-linear activation function

Component-wise operations

$W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$

Linear transformation
Spatial domain



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

$\sigma : \mathbb{R} \to \mathbb{R}$

Non-linear activation function

Component-wise operations

$W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$

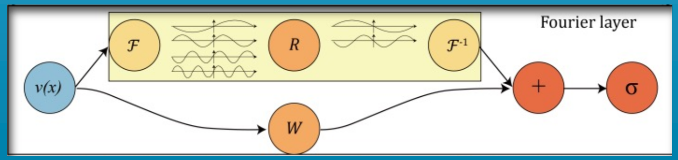
Linear transformation

Spatial domain

$$\mathcal{K}: \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}\big(\mathcal{U}(D;\ \mathbb{R}^{d_v}), \mathcal{U}(D;\ \mathbb{R}^{d_v})\big)$$

Maps to operators on $\mathcal{U}(D;\ \mathbb{R}^{d_v})$

parameterized by $\phi \in \Theta_{\mathcal{K}}$

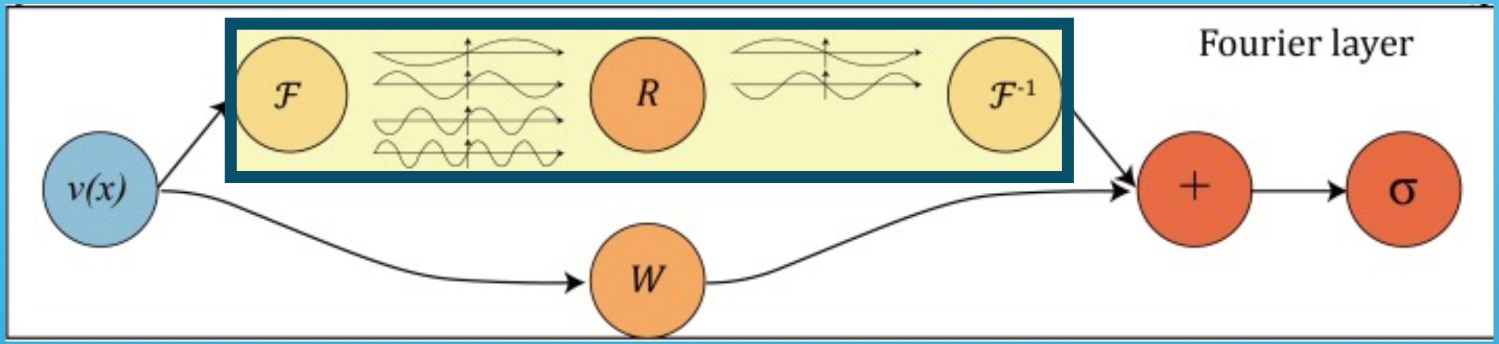


$$v_{t+1}(x) := \sigma\big(Wv_t(x) + (\mathcal{K}(a;\ \phi)v_t)(x)\big), \qquad \forall x \in D$$

$\sigma : \mathbb{R} \rightarrow \mathbb{R}$

Non-linear activation function

Component-wise operations



$W : \mathbb{R}^{d_v} \rightarrow \mathbb{R}^{d_v}$

Linear transformation

Spatial domain

$\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \rightarrow \mathcal{L}\big(\mathcal{U}(D;\ \mathbb{R}^{d_v}), \mathcal{U}(D;\ \mathbb{R}^{d_v})\big)$

Maps to operators on $\mathcal{U}(D;\ \mathbb{R}^{d_v})$

parameterized by $\phi \in \Theta_{\mathcal{K}}$

$$v_{t+1}(x) \coloneqq \sigma\Big(Wv_t(x) + \big(\mathcal{K}(a;\ \phi)v_t\big)(x)\Big), \qquad \forall x \in D$$

$\sigma\colon\ \mathbb{R} \ \to\ \mathbb{R}$

Non-linear activation function

Component-wise operations

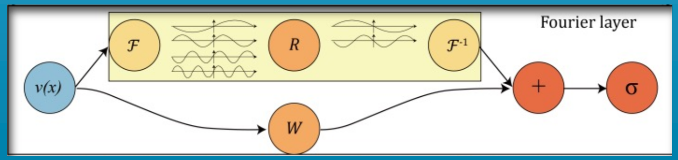
$W:\ \mathbb{R}^{d_v}\ \to\ \mathbb{R}^{d_v}$

Linear transformation

Spatial domain

$\mathcal{K}: \mathcal{A} \times \Theta_{\mathcal{K}}\ \to$
$\mathcal{L}\left(\mathcal{U}(D;\ \mathbb{R}^{d_v}), \mathcal{U}(D;\ \mathbb{R}^{d_v})\right)$

Maps to operators on $\mathcal{U}(D;\ \mathbb{R}^{d_v})$ parameterized by $\phi\ \in\ \Theta_{\mathcal{K}}$



KERNEL INTEGRAL OPERATOR

$$(\mathcal{K}(a;\ \phi)v_t)(x) \ \coloneqq \int_D \kappa(x, y, a(x), a(y);\ \phi)v_t(y)\, dy, \qquad \forall x \ \in D$$

where $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a NN parameterized by $\phi \in \Theta_{\mathcal{K}}$ ➔ kernel function

$$v_{t+1}(x) \coloneqq \sigma(Wv_t(x) + (\mathcal{K}(a;\ \phi)v_t)(x)), \qquad \forall x \in D$$

NEURAL OPERATOR

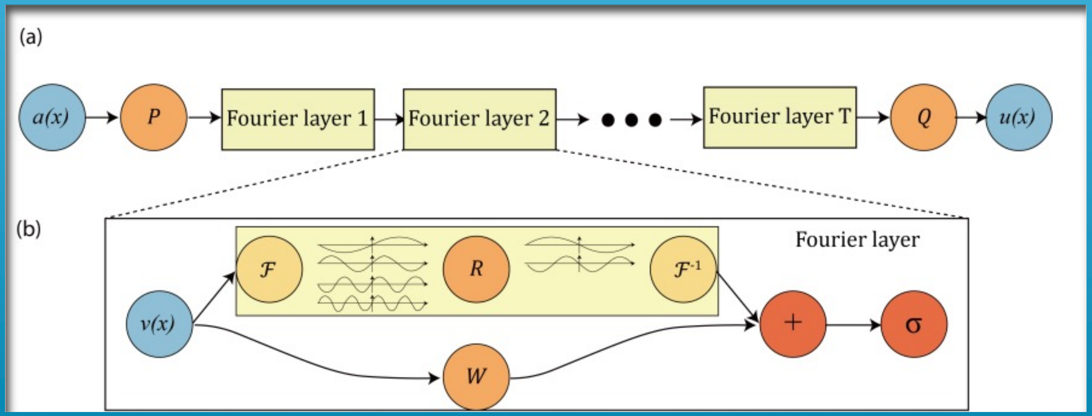
$$\boxed{v_{t+1}(x) \;:=\; \sigma\Big(W v_t(x) + \big(\mathcal{K}(a;\,\phi) v_t\big)(x)\Big)}$$

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

NEURAL OPERATOR

$$\boxed{v_{t+1}(x) \;\coloneqq\; \sigma\big(W v_t(x) + (\mathcal{K}(a;\,\phi) v_t)(x)\big)}$$

$$(\mathcal{K}(a;\,\phi) v_t)(x) \coloneqq \int_D \kappa(x, y, a(x), a(y);\,\phi) v_t(y)\,dy$$



Linear integral operator

Neural Operator learns non-linear operators:

- Linear integral operators

+

- Non-linear activation functions

NEURAL OPERATOR

$$\boxed{\boldsymbol{v_{t+1}(x) := \sigma\big(W v_t(x) + (\mathcal{K}(a;\ \phi) v_t)(x)\big)}}$$

$$\boxed{\boldsymbol{(\mathcal{K}(a;\ \phi)v_t)(x) := \int_D \kappa(x, y, a(x), a(y);\ \phi) v_t(y)\, dy}}$$

Removing dependence of a

+

$$\kappa_\phi\big(x, y, a(x), a(y)\big) = \kappa_\phi\big(\mathrm{x} - \mathrm{y}, a(x), a(y)\big)$$

➔ Convolution operator

Removing dependence of a

+

$$\mathscr{k}_\phi\big(x, y, a(x), a(y)\big) = \mathscr{k}_\phi\big(\mathrm{x} - \mathrm{y}, a(x), a(y)\big)$$

CONVOLUTION THEOREM

Removing dependence of a

$+$

$$\mathscr{k}_\phi(x,y) = \mathscr{k}_\phi(\mathrm{x}-\mathrm{y})$$

Theorem

Fourier transform convolution ➔ pointwise product Fourier transforms

CONVOLUTION THEOREM

Removing dependence of a

$+$

$k_{\phi}(x, y) = k_{\phi}(\mathrm{x} - \mathrm{y})$

Theorem

$$r(x) = \{u * v\}(x) = \int u(x - \tau) v(\tau)\, d\tau$$

CONVOLUTION THEOREM

Removing dependence of a

$+$

$$k_\phi(x, y) = k_\phi(\mathrm{x} - \mathrm{y})$$

Theorem

$$r(x) = \{u * v\}(x) = \int u(x - \tau)v(\tau)\, d\tau$$

Theorem

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$$

$$Where\, U(f) \triangleq \mathcal{F}\{u\}(f);\ V(f) \triangleq \mathcal{F}\{v\}(f)$$

CONVOLUTION THEOREM

Removing dependence of a

$$+$$

$$k_{\phi}(x, y) = k_{\phi}(\mathrm{x} - \mathrm{y})$$

Theorem

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$$

$$\text{Where } U(f) \triangleq \mathcal{F}\{u\}(f);\ V(f) \triangleq \mathcal{F}\{v\}(f)$$

CONVOLUTION THEOREM

Removing dependence of a

$+$

$k_\phi(x, y) = k_\phi(\mathrm{x} - \mathrm{y})$

Theorem

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$$

$$Where\, U(f) \triangleq \mathcal{F}\{u\}(f);\ V(f) \triangleq \mathcal{F}\{v\}(f)$$

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle}\, dx$$

$$(\mathcal{F}^{-1}f)_j(x) = \int_D f_j(k) e^{2i\pi\langle x,k\rangle}\, dk$$

APPLICATION

Removing dependence of a

$+$

$\mathscr{k}_{\phi}(x, y) = \mathscr{k}_{\phi}(x - y)$

Theorem

$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$

Where $U(f) \triangleq \mathcal{F}\{u\}(f);\ V(f) \triangleq \mathcal{F}\{v\}(f)$

$$(\mathcal{K}(a;\ \phi)v_t)(x) = \int_D \mathscr{k}(x - y;\ \phi)v_t(y)\,dy \quad , \quad \forall x \in D$$

APPLICATION

Removing dependence of a

$+$

$\Bbbk_\phi(x, y) = \Bbbk_\phi(\mathrm{x} - \mathrm{y})$

$+$

Theorem

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$$

$$\textit{Where}\; U(f) \triangleq \mathcal{F}\{u\}(f);\; V(f) \triangleq \mathcal{F}\{v\}(f)$$

$$(\mathcal{K}(a;\, \phi)v_t)(x) = \int_D \Bbbk(x - y;\, \phi)v_t(y)\, dy = \int u(x - \tau)v(\tau)\, d\tau\,,$$

$$\forall x \in D$$

APPLICATION

Removing dependence of a
$+$
$\mathcal{k}_\phi(x, y) = \mathcal{k}_\phi(\mathrm{x} - \mathrm{y})$

$+$

Theorem

$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}$$
$$Where\, U(f) \triangleq \mathcal{F}\{u\}(f);\, V(f) \triangleq \mathcal{F}\{v\}(f)$$

$$(\mathcal{K}(a;\ \phi)v_t)(x) = \int_D \mathcal{k}(x - y;\ \phi)v_t(y)\, dy = \mathcal{F}^{-1}(\mathcal{F}(\mathcal{k}_\phi) \cdot \mathcal{F}(v_t))(x),$$
$$\forall x \in D$$

FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(k_\phi)\cdot\mathcal{F}(v_t))(x)$$

$\rightarrow$ *directly parameterize* k_ϕ *in Fourier space*

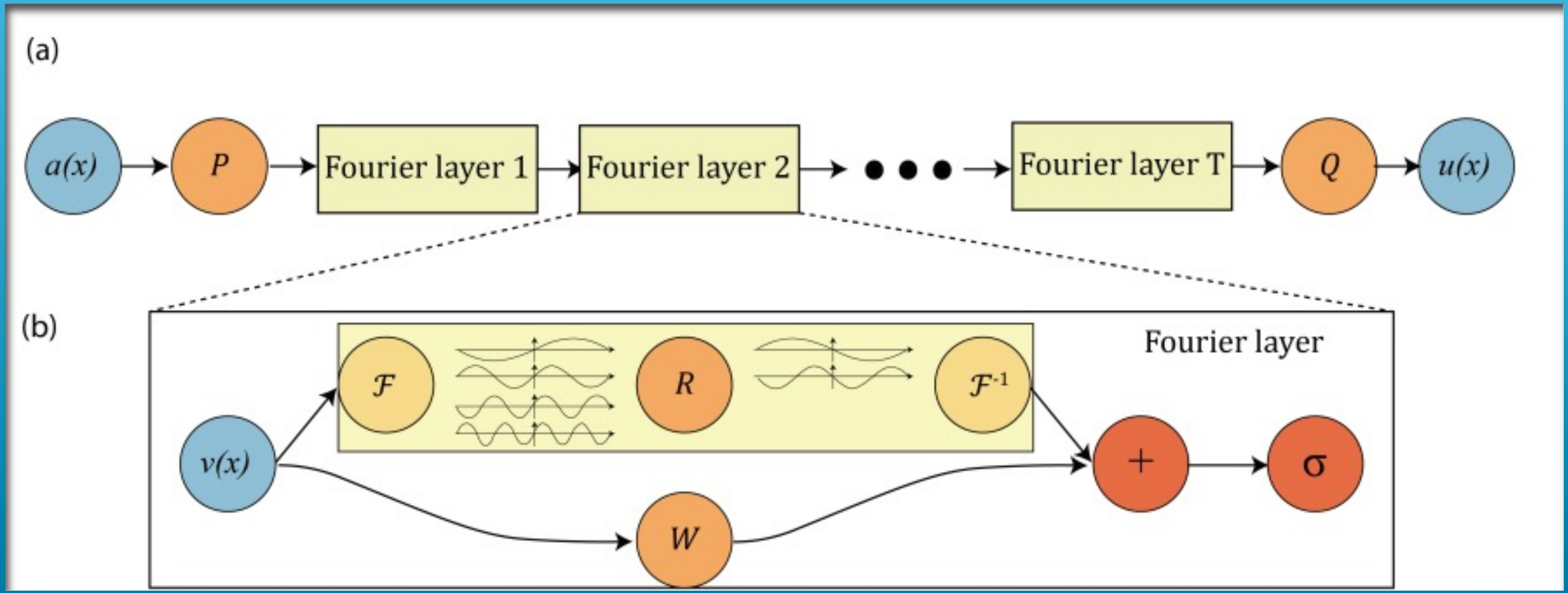
FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}\left(\boldsymbol{R}_\phi \cdot \mathcal{F}(v_t)\right)(x)$$

FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}\left(\boldsymbol{R_\phi} \cdot \mathcal{F}(v_t)\right)(x)$$

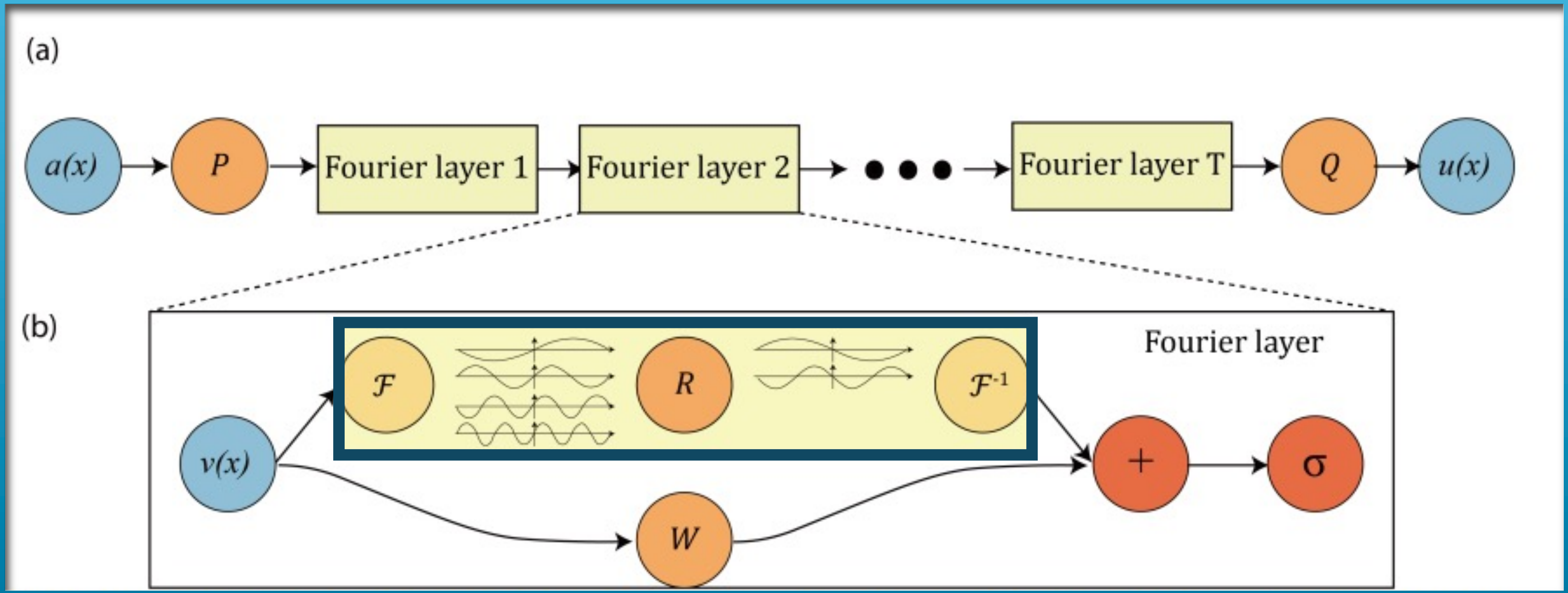
R_ϕ Fourier transform of periodic function κ



FOURIER INTEGRAL OPERATOR

$$\boxed{(\mathcal{K}(a;\phi)v_t)(x)} = \mathcal{F}^{-1}\left(R_\phi \cdot \mathcal{F}(v_t)\right)(x)$$

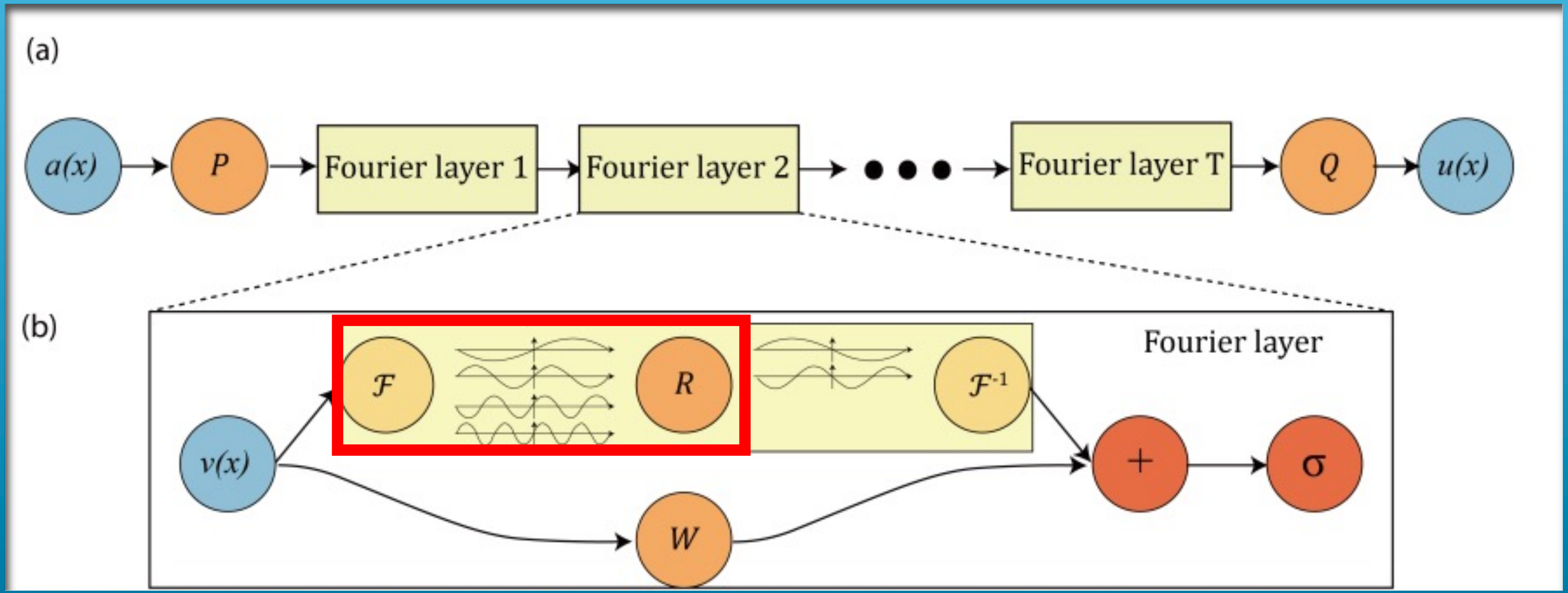
R_ϕ Fourier transform of periodic function κ



FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}\boxed{\left(R_\phi \cdot \mathcal{F}(v_t)\right)}(x)$$

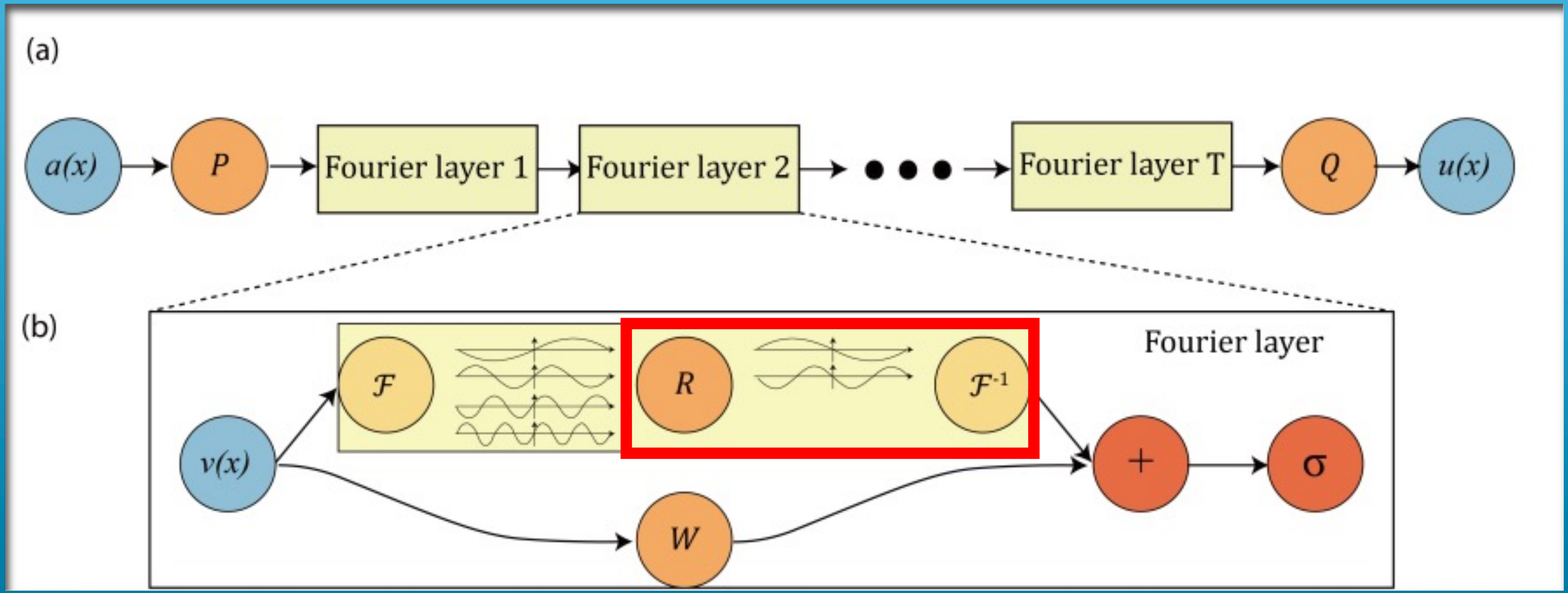
R_ϕ Fourier transform of periodic function κ



FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \boxed{\mathcal{F}^{-1}(\quad R_\phi} \cdot \mathcal{F}(v_t))(x)$$

R_ϕ Fourier transform of periodic function κ



FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\mathcal{k}_\phi)\cdot\mathcal{F}(v_t))(x)$$

$$\downarrow$$

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

FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\textcolor{red}{\mathscr{k}_\phi}) \cdot \mathcal{F}(v_t))(x)$$
$$\downarrow$$
$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v_t))(x)$$

Assumed $\mathscr{k}_\phi$ periodic

FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\mathscr{k}_\phi)\cdot\mathcal{F}(v_t))(x)$$

$$\downarrow$$

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

$$(\mathcal{F}f)_j(k) = \int_D f_j(x)e^{-2i\pi\langle x,k\rangle}\,dx$$

$$\downarrow$$

Fourier series expansion

$$\downarrow$$

Work with the discrete modes $\mathscr{k}_\phi \in \mathbb{Z}^d$

FOURIER INTEGRAL OPERATOR

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

FOURIER INTEGRAL OPERATOR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\quad R_\phi \quad \cdot \mathcal{F}(v_t))(x)$$

R_ϕ Fourier transform of periodic function κ

Finite-dimensional parameterization ➔ Truncating at k_{max}

SO FAR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\mathscr{k}_\phi)\cdot\mathcal{F}(v_t))(x)$$

SO FAR

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\mathscr{k}_\phi)\cdot\mathcal{F}(v_t))(x)$$

$$\downarrow$$

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

$$\downarrow$$

Finite-dimensional parameterization ➔ Truncating at k_{max}

DISCRETE FOURIER TRANSFORM

$$(\mathcal{K}(a;\phi)v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\mathscr{k}_\phi) \cdot \mathcal{F}(v_t))(x)$$

$\downarrow$

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

$\downarrow$

Finite-dimensional parameterization ➔ Truncating at k_{max}

$$R_\phi \cdot \mathcal{F}(v_t)$$

$\downarrow$

Point-wise multiplication

$+$

Lying in different dimensions

$\downarrow$

Truncate higher modes of $\mathcal{F}(v_t)$

$\downarrow$

$$(R \cdot (\mathcal{F}_{v_t}))_{k,l} = \sum_{j=1}^{d_v} R_{k,l,j} (\mathcal{F}_{v_t})_{k,l}$$

DISCRETE FOURIER TRANSFORM

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle}\, dx$$

$$\downarrow$$

$$(\mathcal{F}f)_j[k] = \sum_{x=0}^{n-1} f_j[x] e^{-2i\pi\frac{\langle x,k\rangle}{n}}$$

$$where: k \in \left[-{}^{k_{max}}/_{2}, \dots, {}^{k_{max}}/_{2}\right]$$

FAST FOURIER TRANSFORM

$$(\mathcal{F}f)_j(k) = \int_D f_j(x) e^{-2i\pi\langle x,k\rangle}\, dx$$

$$\downarrow$$

$$(\mathcal{F}f)_j[k] = \sum_{x=0}^{n-1} f_j[x] e^{-2i\pi\frac{\langle x,k\rangle}{n}}$$

$$where: k \in \left[-{}^{k_{max}}/_{2}, \ldots, {}^{k_{max}}/_{2}\right]$$

$$(\hat{\mathcal{F}}f)_l(k) = \sum_{x_1}^{s_1} \ldots \sum_{x_d}^{s_d-1} f_l(x_1, \ldots, x_d) e^{-2i\pi \sum_{j=1}^{d} x_j k_j}$$

$$where: l = 1, \ldots, d_v$$

INVARIANCE TO DISCRETIZATION

- The Fourier layers are discretization-invariant

INVARIANCE TO DISCRETIZATION

- The Fourier layers are discretization-invariant

Parameters learned in Fourier space

$\downarrow$

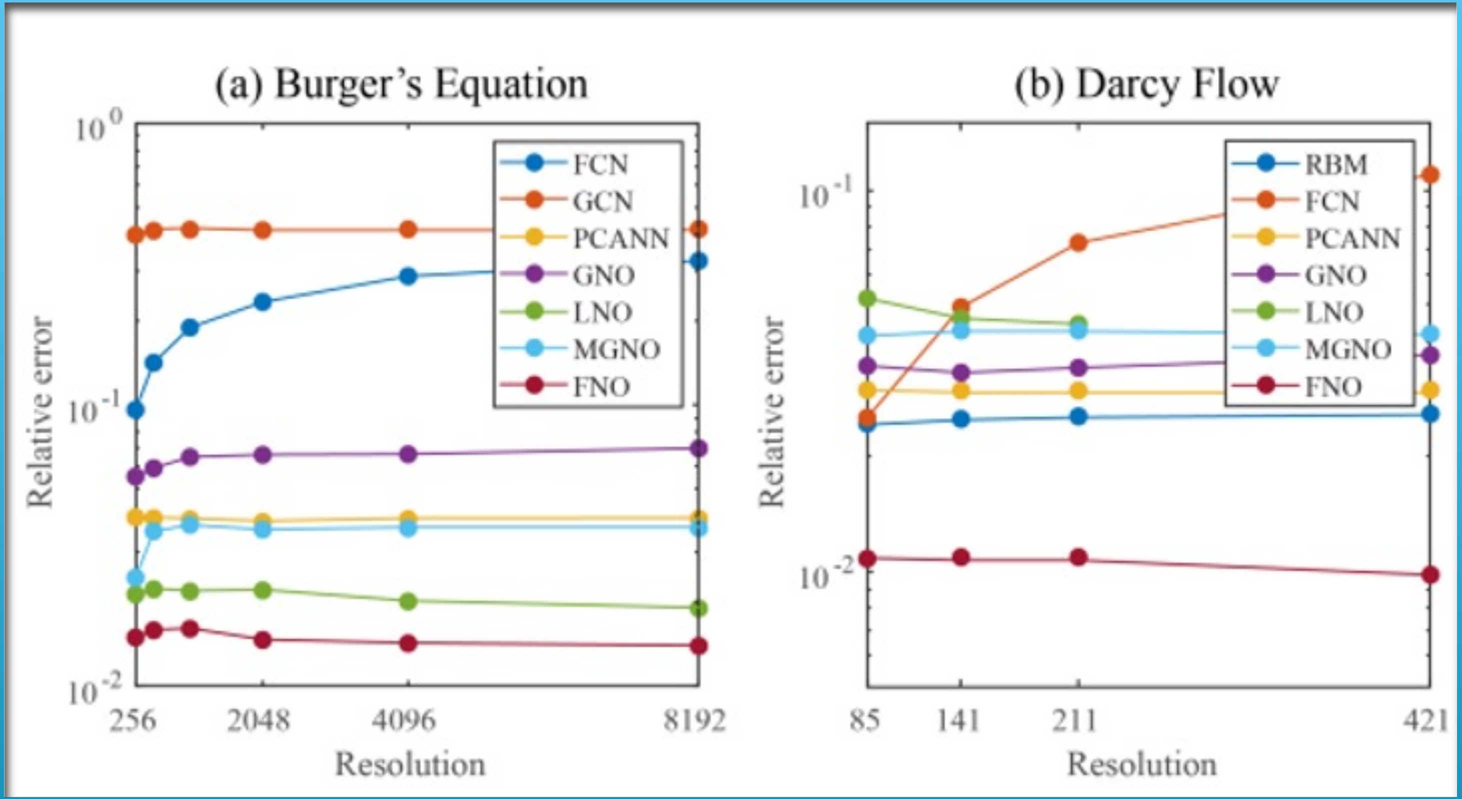
Projecting on the basis $e^{2\pi i\langle x,k\rangle}$

$\rightarrow$ Physical space

$\downarrow$

Well-defined everywhere on $\mathbb{R}^d$ + Consistent error at any resolution

INVARIANCE TO DISCRETIZATION



RUNTIME

- Inner multiplication has complexity $O(k_{max})$
- Fourier transforms have complexity $O(nk_{max})$

RUNTIME

- Inner multiplication has complexity O(k_{max})
- Fourier transforms have complexity O($\mathrm{n}k_{max}$)

FFT has complexity O($nlog(n)$)

↓

Uniform discretization is required

NUMERICAL EXPERIMENTS

THE MODEL

- 4 Fourier layers

+ ReLU

+ Batch normalization

- 500 epochs
- Learning rate: 0.001, halved every 100 epochs

TESTS

- 1-d Burgers' equation
- 2-d Darcy Flow problem
- 2-d Navier-Stokes equation
- Bayesian Inverse Problem

SIMULATION

- Compared against other popular solvers

SIMULATION

- Compared against other popular solvers
- Two different Fourier Neural Operators:
  - ❖ FNO-2D learns spatial relation between time steps to predict next step

SIMULATION

- Compared against other popular solvers
- Two different Fourier Neural Operators:
  - ❖ FNO-2D learns spatial relation between time steps to predict next step

2DConv

SIMULATION

- Compared against other popular solvers
- Two different Fourier Neural Operators:
  - ❖ FNO-2D learns spatial relation between time steps to predict next step
  - ❖ FNO-3D learns full space-time relation of whole interval

2DConv

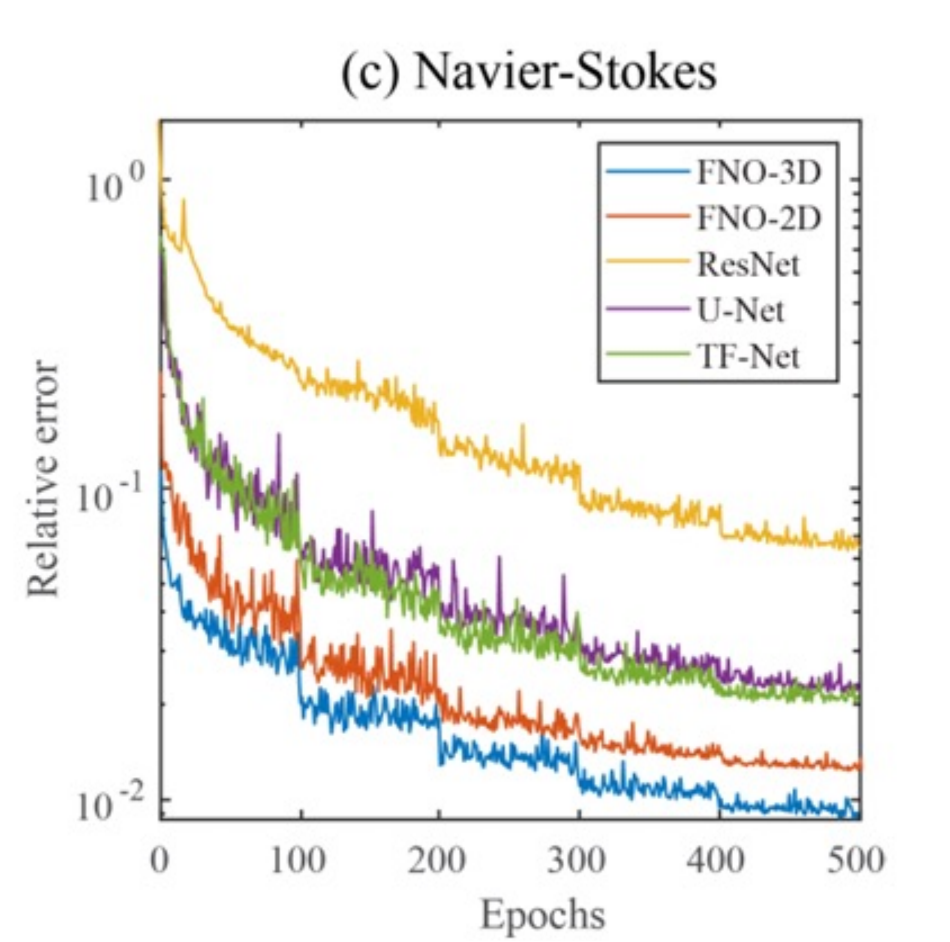
SIMULATION

- Compared against other popular solvers
- Two different Fourier Neural Operators:
  - ❖ FNO-2D learns spatial relation between time steps to predict next step
  - ❖ FNO-3D learns full space-time relation of whole interval

2DConv

3DConv

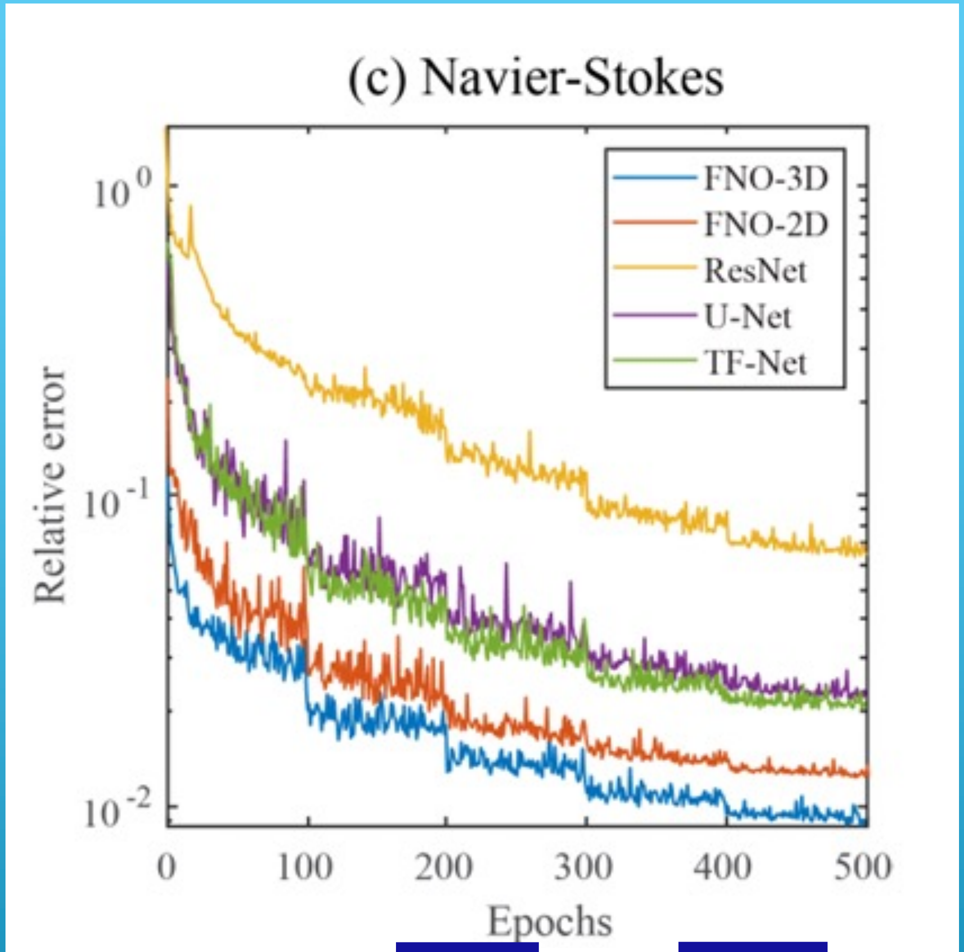
SIMULATION



Config	Parameters	Time per epoch	$\nu = 1e{-3}$ $T = 50$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 10000$	$\nu = 1e{-5}$ $T = 20$ $N = 1000$
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

- With sufficient data:

SIMULATION

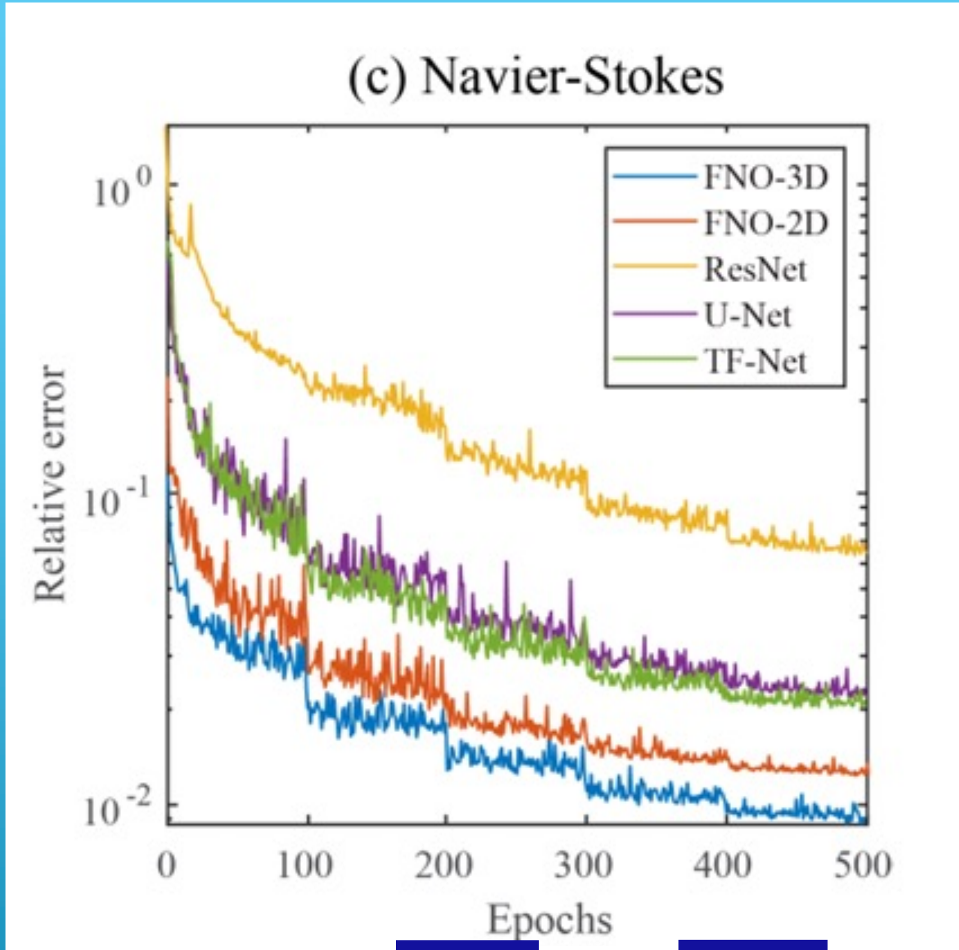


Config	Parameters	Time per epoch	$\nu = 1e{-}3$ $T = 50$ $N = 1000$	$\nu = 1e{-}4$ $T = 30$ $N = 1000$	$\nu = 1e{-}4$ $T = 30$ $N = 10000$	$\nu = 1e{-}5$ $T = 20$ $N = 1000$
FNO-3D	6, 558, 537	38.99s	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	414, 517	127.80s	0.0128	**0.1559**	0.0834	**0.1556**
U-Net	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

- With sufficient data:

FNO-3D

SIMULATION

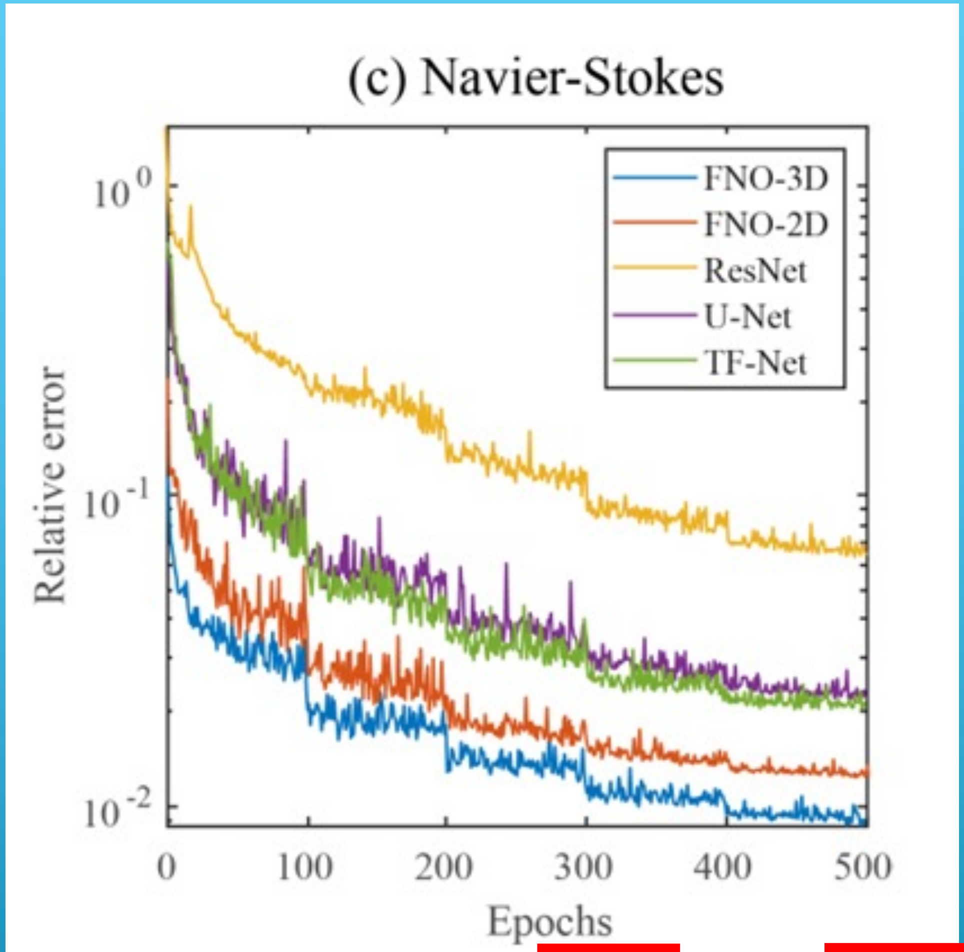


Config	Parameters	Time per epoch	$\nu = 1\text{e}{-3}$ $T = 50$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 1000$	$\nu = 1\text{e}{-4}$ $T = 30$ $N = 10000$	$\nu = 1\text{e}{-5}$ $T = 20$ $N = 1000$
FNO-3D	6, 558, 537	$38.99s$	**0.0086**	0.1918	**0.0820**	0.1893
FNO-2D	414, 517	$127.80s$	0.0128	**0.1559**	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

- With sufficient data:

FNO-3D

SIMULATION



Config	Parameters	Time per epoch	$\nu = 1e{-3}$ $T = 50$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 10000$	$\nu = 1e{-5}$ $T = 20$ $N = 1000$
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

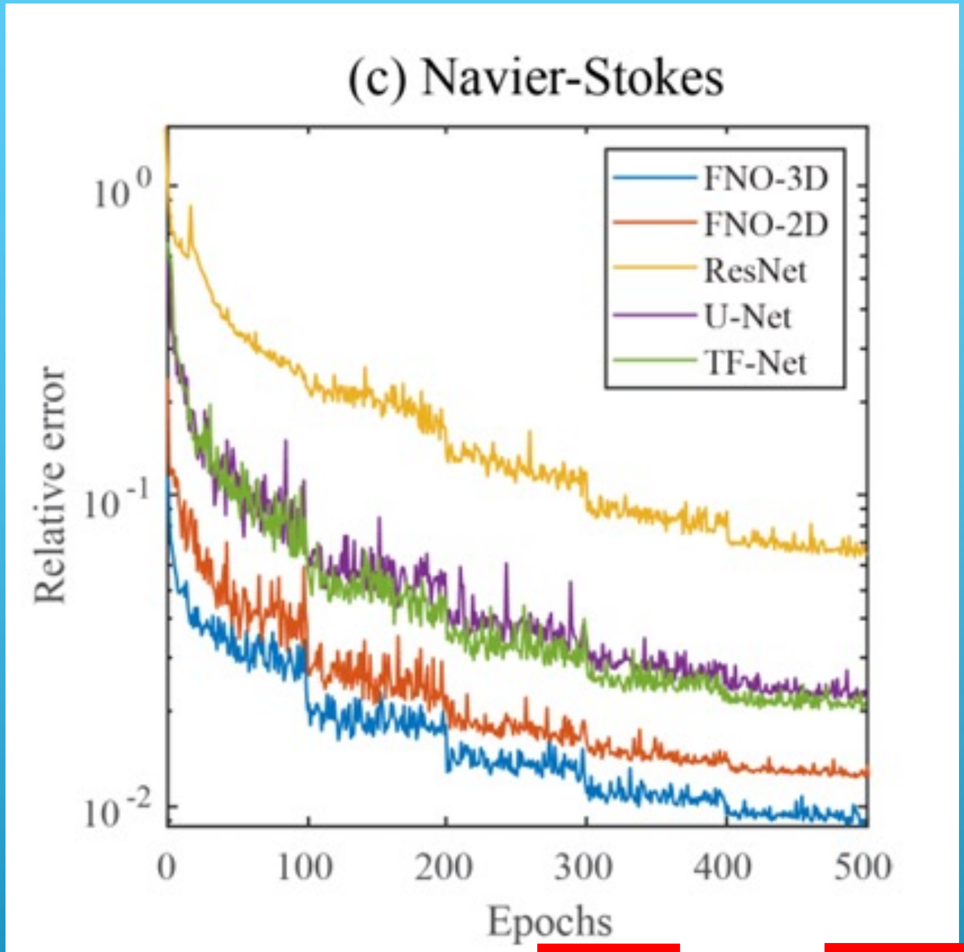
- With sufficient data:

FNO-3D

- With insufficient data:

FNO-2D

SIMULATION



Config	Parameters	Time per epoch	$\nu = 1e{-}3$, $T = 50$, $N = 1000$	$\nu = 1e{-}4$, $T = 30$, $N = 1000$	$\nu = 1e{-}4$, $T = 30$, $N = 10000$	$\nu = 1e{-}5$, $T = 20$, $N = 1000$
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

- With sufficient data:

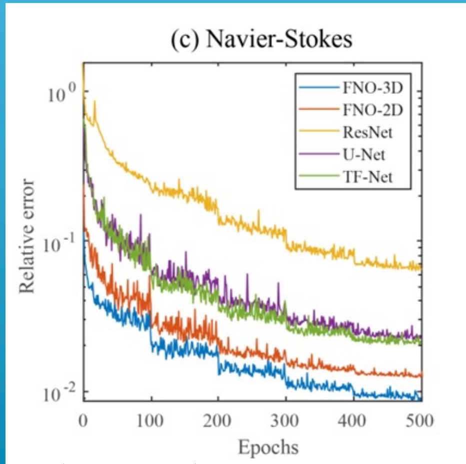
FNO-3D

- With insufficient data:

FNO-2D 

SIMULATION

FOURIER NEURAL OPERATOR WINS!



Config	Parameters	Time per epoch	$\nu = 1e{-3}$ $T = 50$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 1000$	$\nu = 1e{-4}$ $T = 30$ $N = 10000$	$\nu = 1e{-5}$ $T = 20$ $N = 1000$
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

KEY FEATURES

Speed and Accuracy:

- Speedy & no accuracy degradation
- Effective in downstream applications

Approximates:

- Highly non-linear operators with high frequency modes and slow energy decay

KEY FEATURES

Resolution-Invariant Solution Operator:

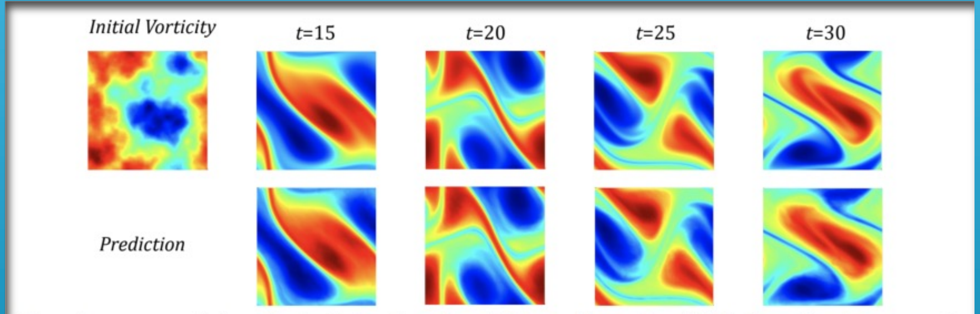
- First for Navier-Stokes equations in the turbulent regime

Shared Parameters:

- Maintains same learned network parameters

Capable of zero-shot super-resolution:

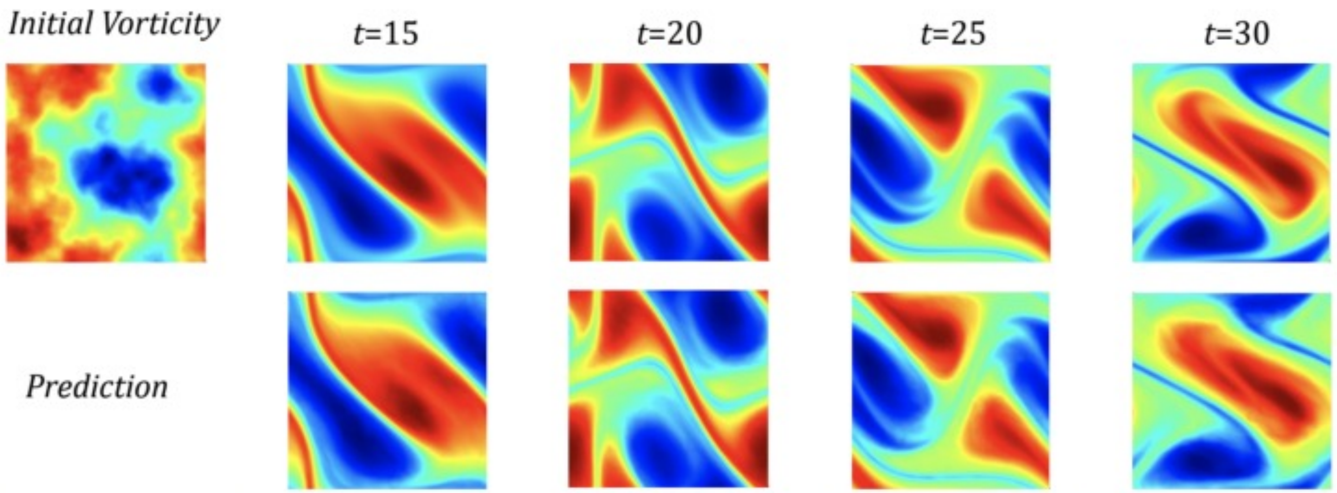
- Training lower resolution ➔ Evaluated higher resolution



Zero-shot super-resolution: Navier-Stokes Equation with Reynolds number 10000; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$ (see Section 5.4).

Figure 1: **top:** The architecture of the Fourier layer; **bottom:** Example flow from Navier-Stokes.

KEY FEATURES



Zero-shot super-resolution: Navier-Stokes Equation with Reynolds number 10000; Ground truth on top and prediction on bottom; trained on $64 \times 64 \times 20$ dataset; evaluated on $256 \times 256 \times 80$ (see Section 5.4).

Figure 1: **top:** The architecture of the Fourier layer; **bottom:** Example flow from Navier-Stokes.

Capable of zero-shot super-resolution:

- Training lower resolution ➔ Evaluated higher resolution