AFNOs and FourCastNet

Pathak et al., 2022

arXiv preprint arXiv:2202.11214

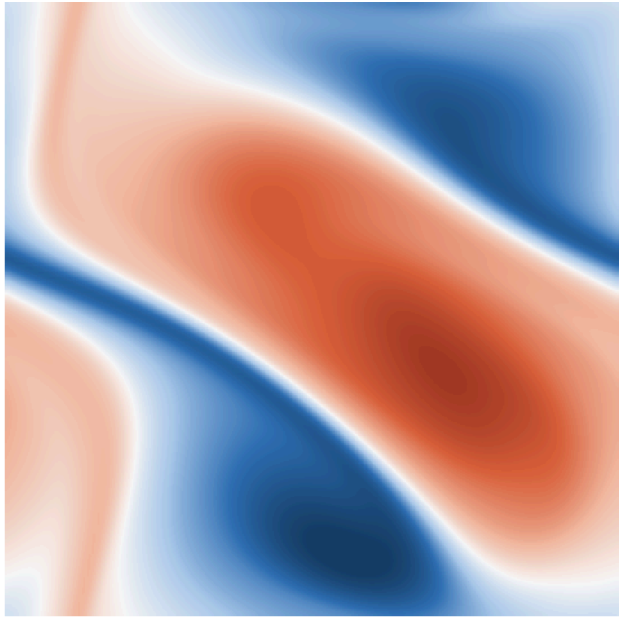
Dongyeob Kim · Ram Narayanan

Roadmap

I PDEs in physics
II Numerical solvers
III Learning the solution operator
IV Neural operators
V Fourier neural operators
VI Adaptive FNO
VII FourCastNet
VIII Results

Part I

PDEs in physics



PDEs in physics

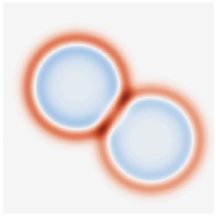
Linear



Heat

$\partial_t u = \alpha\, \Delta u$

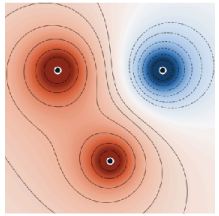
$u(x, 0) \mapsto u(x, t)$



Waves

$\partial_t^2 u = c^2 \Delta u$

$u(x, 0) \mapsto u(x, t)$

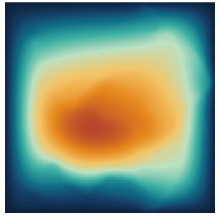


Electrostatics

$-\Delta V = \rho/\varepsilon_0$

$\rho(x) \mapsto V(x)$

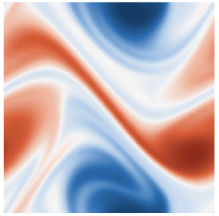
Nonlinear



Porous media

$-\nabla \cdot (a \nabla u) = f$

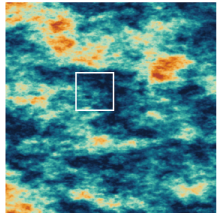
$a(x) \mapsto u(x)$



Fluids

$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \Delta \omega$

$\omega(x, 0) \mapsto \omega(x, t)$



Weather

fluids on a rotating sphere

$X(x, 0) \mapsto X(x, t)$

- Each maps a function of x to another function of x: a **time rollout** $u(x, 0) \mapsto u(x, t)$, or a **steady state** $a(x) \mapsto u(x)$.
- A **local** change in the input changes the solution **everywhere**.

Heat equation



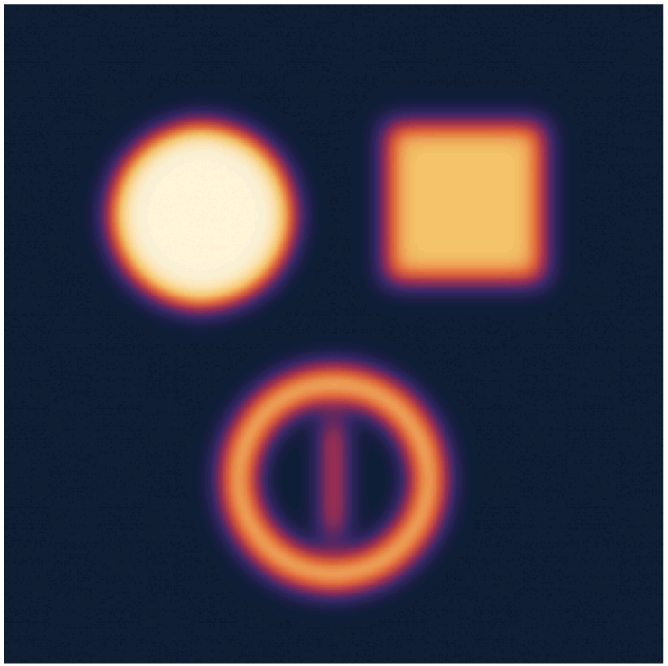
$t = 0$

$$\partial_t u = \Delta u$$

Time rollout: $u(x, 0) \mapsto u(x, t)$

- **Linear:** $u(x, t)$ is a weighted average of $u(x, 0)$ over all x.

Heat equation



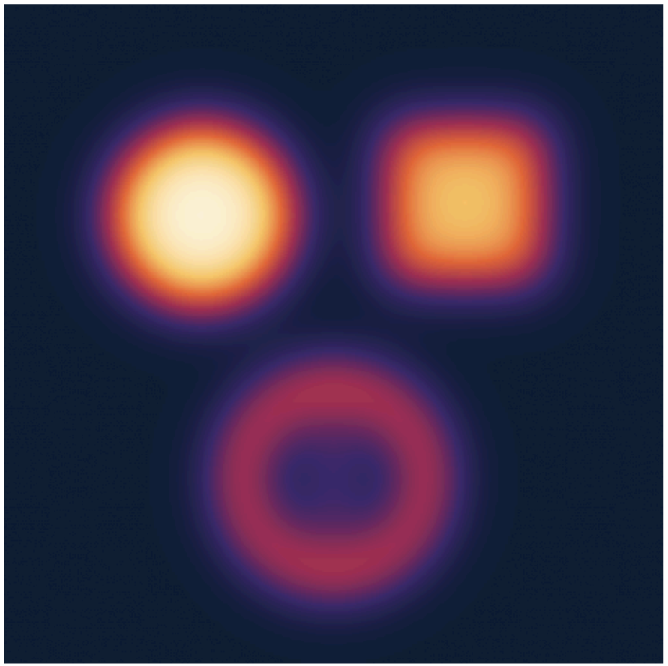
$t = 2 \cdot 10^{-4}$

$$\partial_t u = \Delta u$$

Time rollout: $u(x, 0) \mapsto u(x, t)$

- **Linear:** $u(x, t)$ is a weighted average of $u(x, 0)$ over all x.

Heat equation



$t = 10^{-3}$

$$\partial_t u = \Delta u$$

Time rollout: $u(x, 0) \mapsto u(x, t)$

- **Linear:** $u(x, t)$ is a weighted average of $u(x, 0)$ over all x.

Heat equation



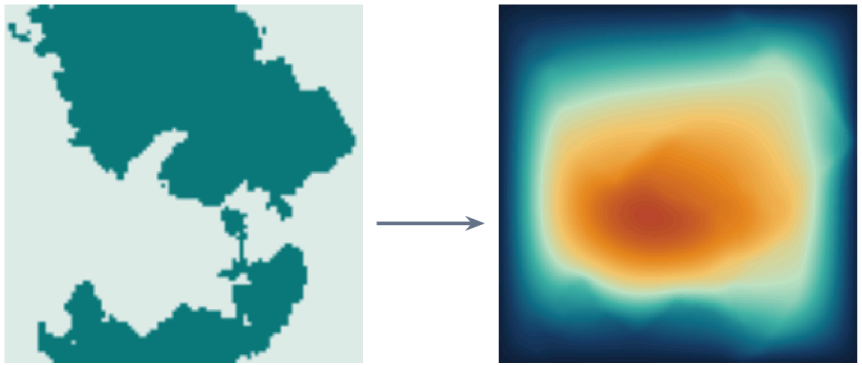
$t = 4 \cdot 10^{-3}$

$$\partial_t u = \Delta u$$

Time rollout: $u(x, 0) \mapsto u(x, t)$

- **Linear:** $u(x, t)$ is a weighted average of $u(x, 0)$ over all x.

Darcy flow



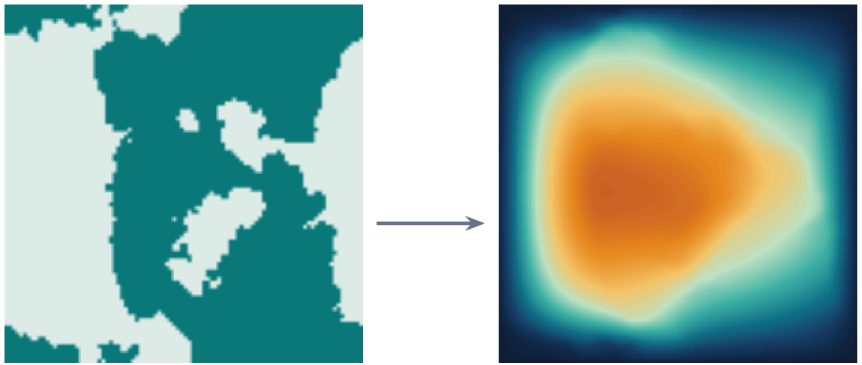
$a(x)$ $u(x)$

$$-\nabla \cdot \big(a(x)\,\nabla u(x)\big) = f(x)$$

Steady state: $a(x) \mapsto u(x)$

- The map $a \mapsto u$ is **nonlinear**.
- Every new a is a new solve.

Darcy flow



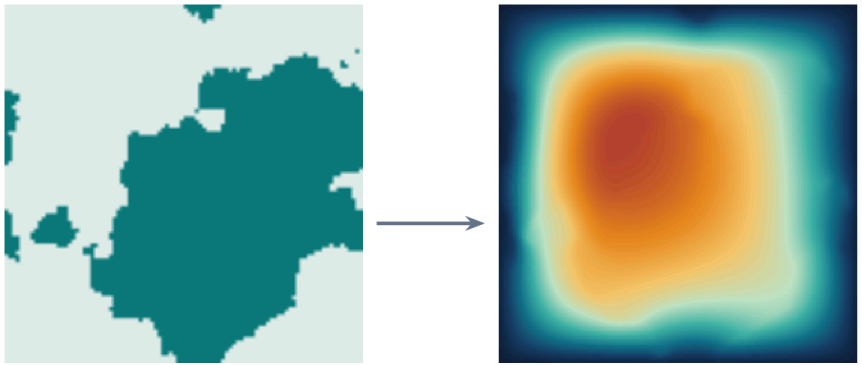
$a(x)$ $u(x)$

$$-\nabla \cdot \big(a(x)\,\nabla u(x)\big) = f(x)$$

Steady state: $a(x) \mapsto u(x)$

- The map $a \mapsto u$ is **nonlinear**.
- Every new a is a new solve.

Darcy flow



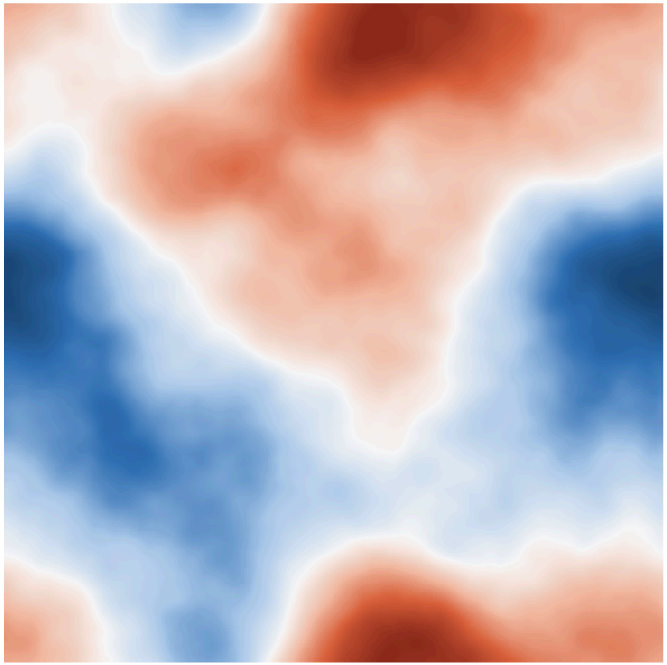
$a(x)$ $u(x)$

$$-\nabla \cdot \big(a(x)\,\nabla u(x)\big) = f(x)$$

Steady state: $a(x) \mapsto u(x)$

- The map $a \mapsto u$ is **nonlinear**.
- Every new a is a new solve.

2-D incompressible flow



$t = 0$

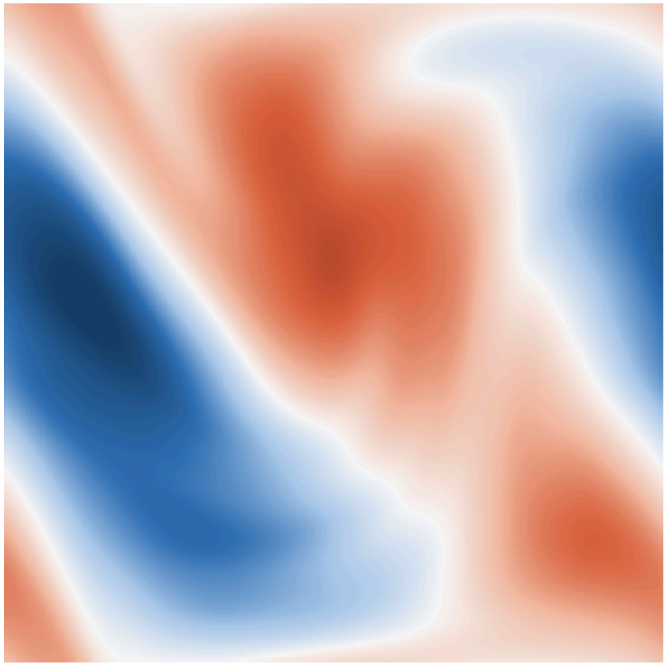
$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \, \Delta \omega + f$$

$$\mathbf{u} = (\partial_y \psi, \, -\partial_x \psi), \qquad -\Delta \psi = \omega$$

Time rollout: $\omega(x, 0) \mapsto \omega(x, t)$

- **Nonlinear**: the velocity $\mathbf{u}$ is recovered from ω through the stream function ψ.
- No closed form, so it is simulated step by step.

2-D incompressible flow



$t = 4$

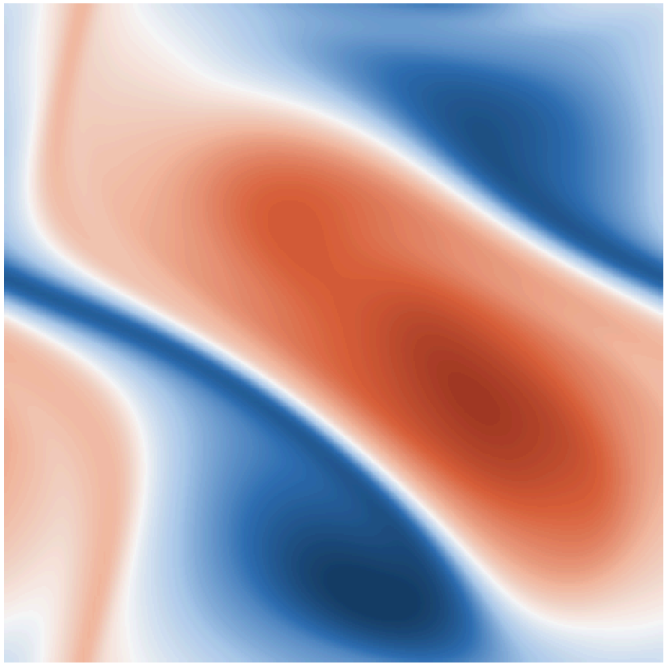
$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \, \Delta \omega + f$$

$$\mathbf{u} = (\partial_y \psi, \, -\partial_x \psi), \qquad -\Delta \psi = \omega$$

Time rollout: $\omega(x, 0) \mapsto \omega(x, t)$

- **Nonlinear**: the velocity $\mathbf{u}$ is recovered from ω through the stream function ψ.
- No closed form, so it is simulated step by step.

2-D incompressible flow



$t = 12$

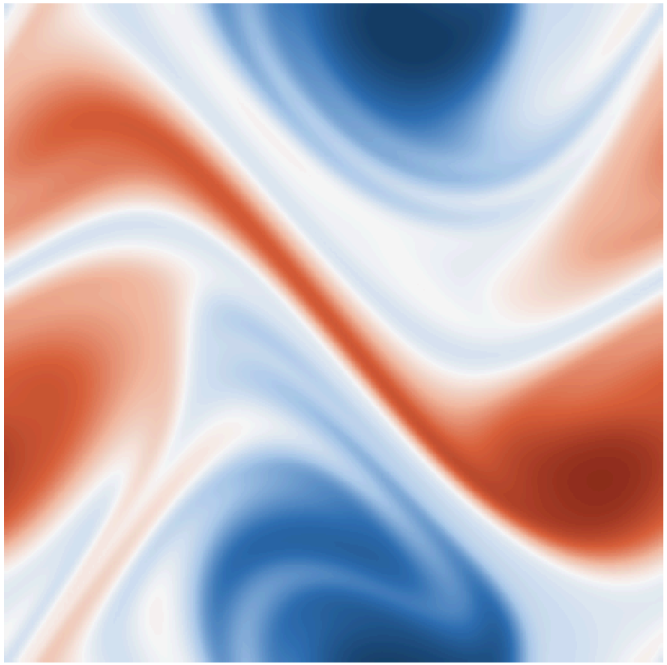
$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \, \Delta \omega + f$$

$$\mathbf{u} = (\partial_y \psi, \, -\partial_x \psi), \qquad -\Delta \psi = \omega$$

Time rollout: $\omega(x, 0) \mapsto \omega(x, t)$

- **Nonlinear**: the velocity $\mathbf{u}$ is recovered from ω through the stream function ψ.
- No closed form, so it is simulated step by step.

2-D incompressible flow



$t = 30$

$$\partial_t \omega + \mathbf{u} \cdot \nabla \omega = \nu \, \Delta \omega + f$$

$$\mathbf{u} = (\partial_y \psi, \, -\partial_x \psi), \qquad -\Delta \psi = \omega$$

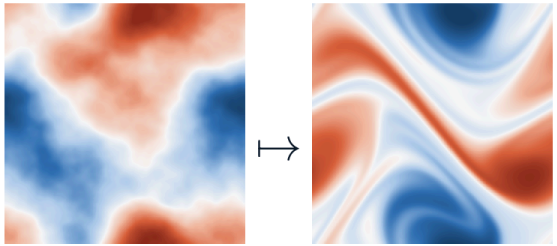
Time rollout: $\omega(x, 0) \mapsto \omega(x, t)$

- **Nonlinear**: the velocity $\mathbf{u}$ is recovered from ω through the stream function ψ.
- No closed form, so it is simulated step by step.

Solution operators

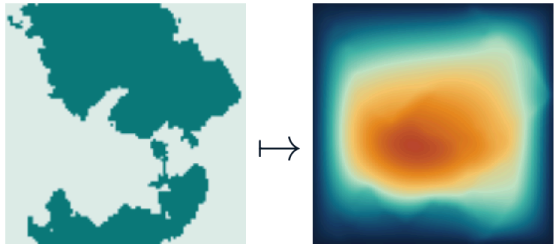
$$\mathcal{G}^\dagger : \ a(x) \longmapsto u(x), \qquad x \in D$$

Time rollout: $u(x,0) \mapsto u(x,t)$



heat, waves, fluids, weather

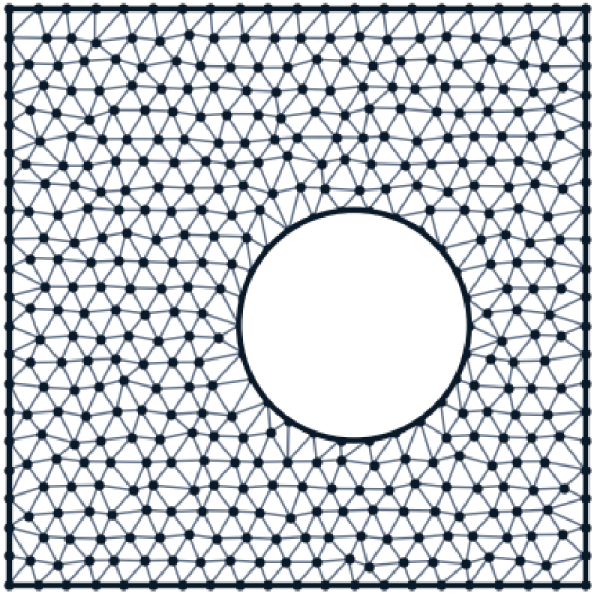
Steady state: $a(x) \mapsto u(x)$



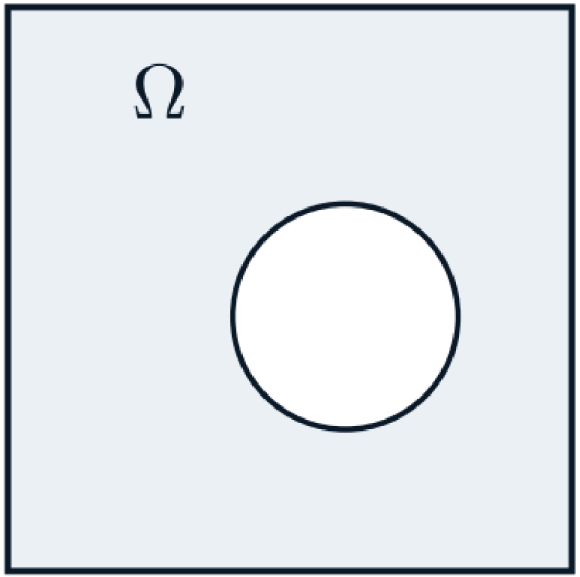
electrostatics, Darcy flow

Part II

Numerical solvers



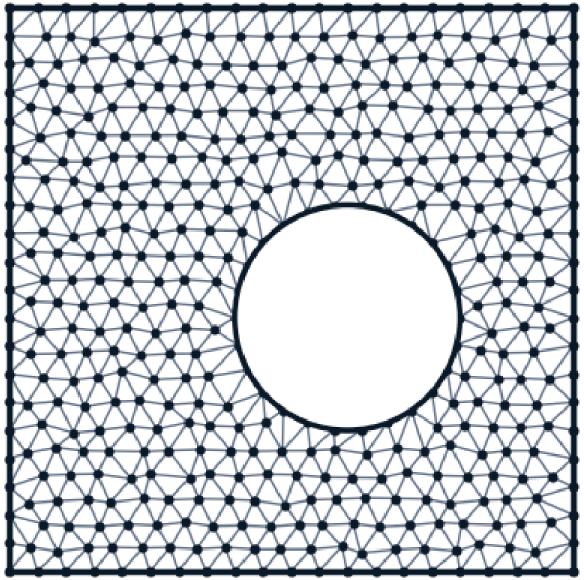
Discretize, then solve



the domain

1. **Mesh** the domain into N unknowns.

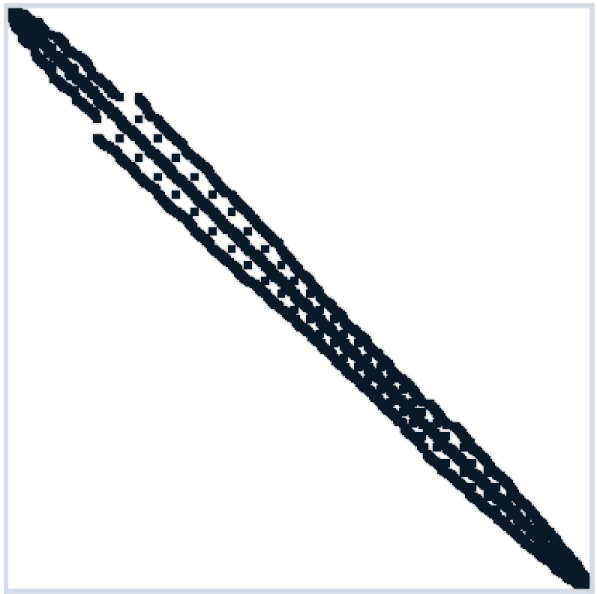
Discretize, then solve



a mesh: unknowns at the nodes

1. **Mesh** the domain into N unknowns.

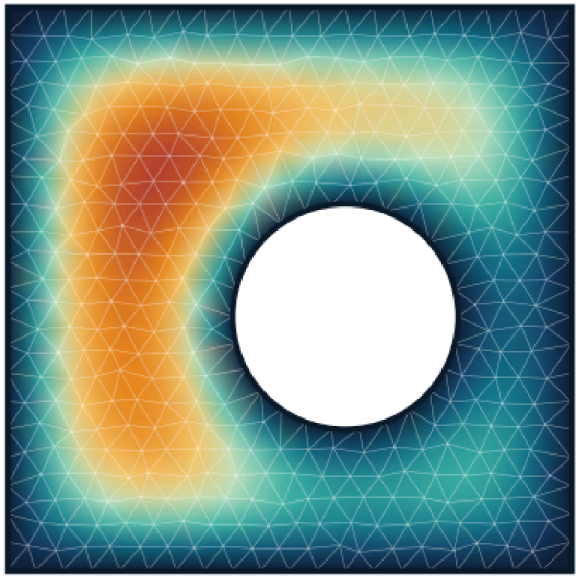
Discretize, then solve



the sparse matrix K

1. **Mesh** the domain into N unknowns.
2. Neighbouring nodes couple into a **sparse** system $KU = F$.

Discretize, then solve



the solution

1. **Mesh** the domain into N unknowns.
2. Neighbouring nodes couple into a **sparse** system $KU = F$.
3. **Solve** iteratively, at every time step.

Systems can have 10^6–10^9 unknowns.

Solver cost

for each query a

Solver cost

for each query a

for each time step

Solver cost

for each query a

for each time step

for each nonlinear iteration: linearize, then solve $KU = F$

Solver cost

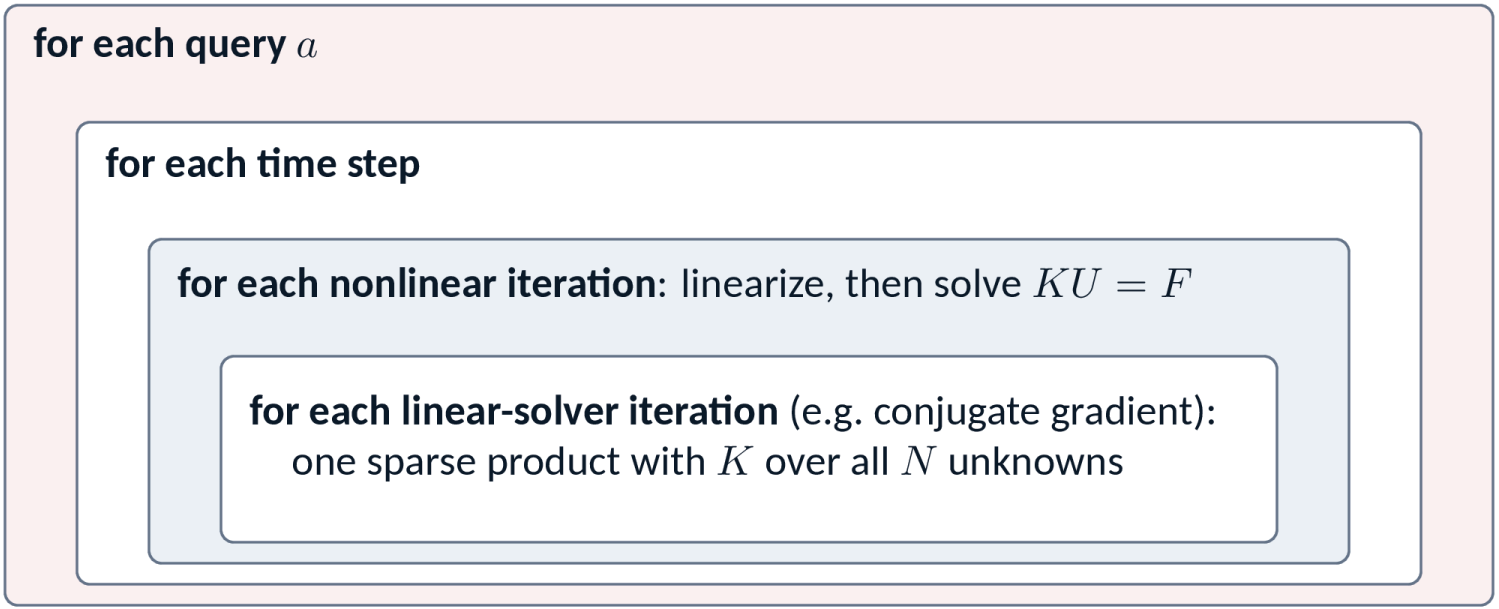
for each query a

for each time step

for each nonlinear iteration: linearize, then solve $KU = F$

for each linear-solver iteration (e.g. conjugate gradient):
one sparse product with K over all N unknowns

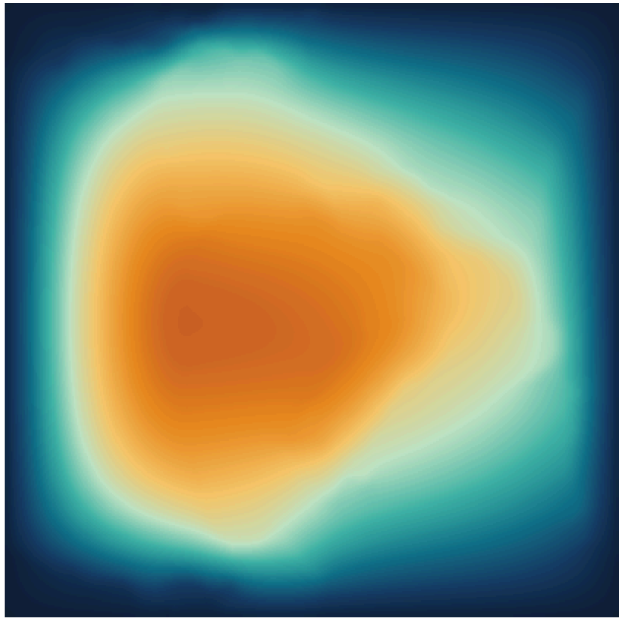
Solver cost



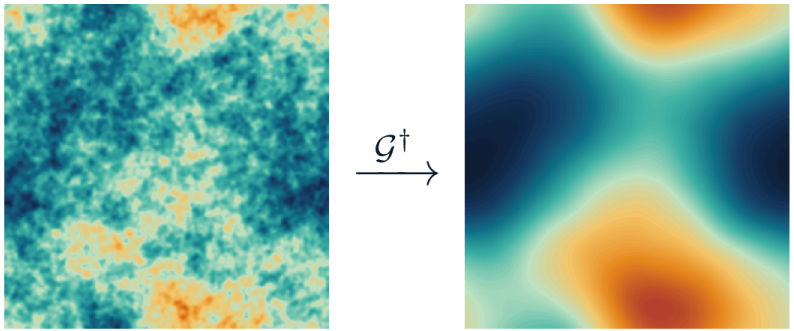
Nothing is reused between queries.

Part III

Learning the solution operator



Learning problem



input $a(x)$ output $u(x)$

$\mathcal{G}^\dagger : \mathcal{A} \to \mathcal{U}$ between **function spaces**

From pairs $(a_j,\ u_j = \mathcal{G}^\dagger(a_j))$, fit

$$\min_\theta \ \mathbb{E}_a \, \ell\big(\mathcal{G}_\theta(a),\ \mathcal{G}^\dagger(a)\big)$$

- $\mathcal{G}_\theta$ should work on any grid.

CNNs

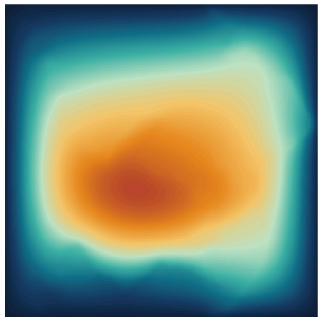


a on an $n \times n$ grid

CNN / U-Net

$\mathbb{R}^{n \times n} \to \mathbb{R}^{n \times n}$

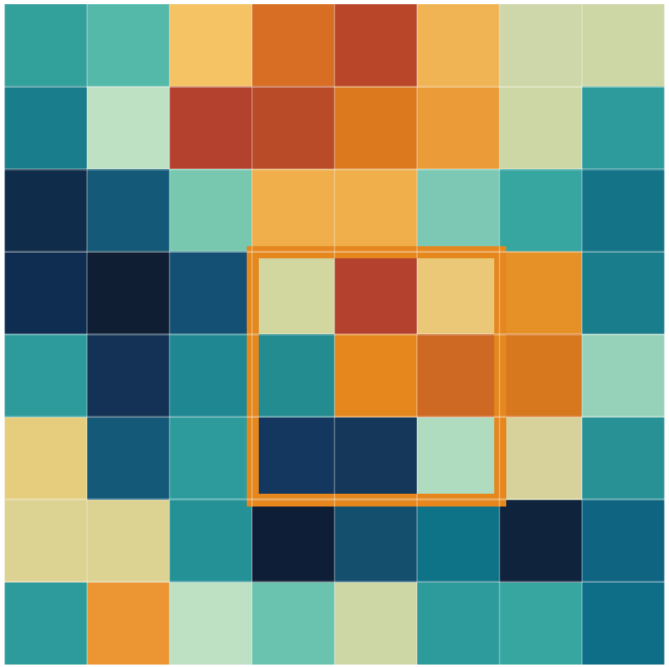
stacks of 3×3 convolutions



u on the same grid

- Works well at a fixed resolution.

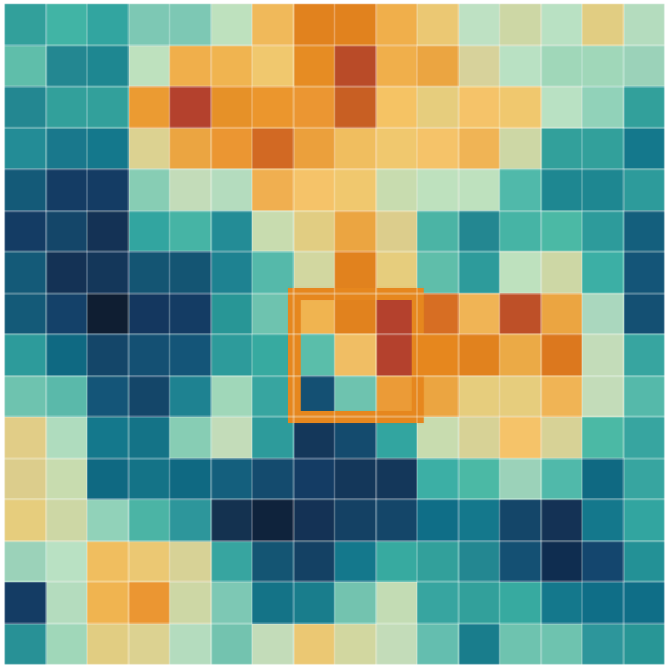
Problem 1: resolution



8×8 grid: the 3×3 stencil spans $3/8$ of the domain

- A weight's physical length depends on the grid spacing.
- The same weights give a **different operator** at each resolution.

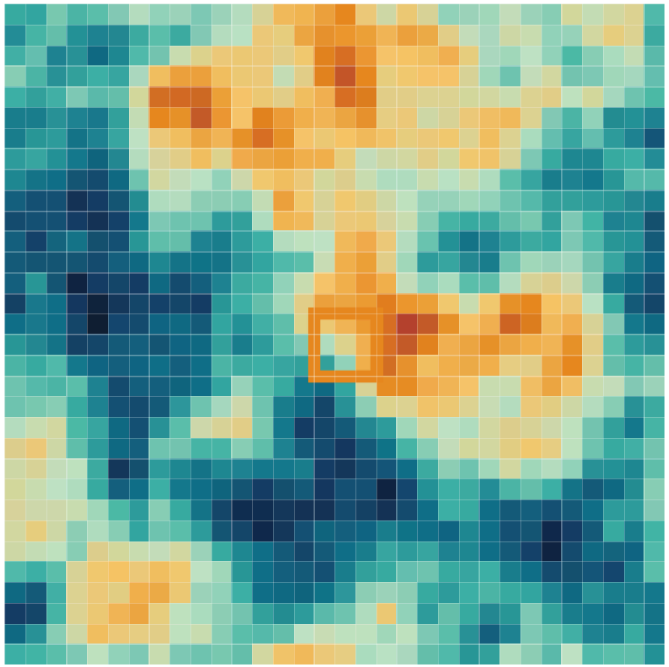
Problem 1: resolution



16×16 grid: now $3/16$

- A weight's physical length depends on the grid spacing.
- The same weights give a **different operator** at each resolution.

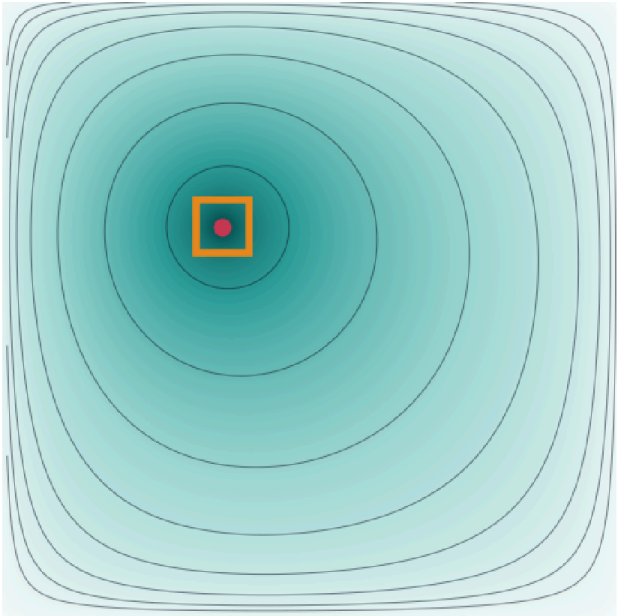
Problem 1: resolution



32×32 grid: now $3/32$

- A weight's physical length depends on the grid spacing.
- The same weights give a **different operator** at each resolution.

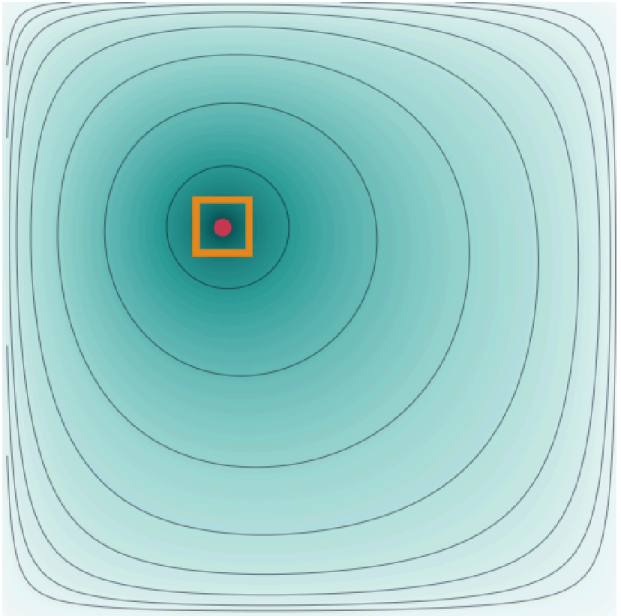
Problem 2: kernel size



response of $-\Delta u = f$ to a point source

- A source at one point (red) changes u **everywhere**.
- A CNN with L layers of 3×3 filters sees only $(2L+1)^2$ pixels (amber box: $L = 5$).
- Covering the domain needs depth that grows with resolution, or down-sampling (U-Nets).

Problem 2: kernel size



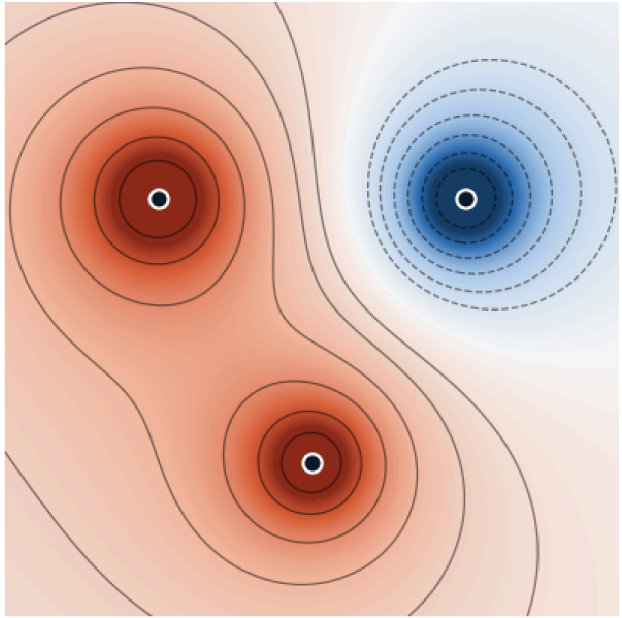
response of $-\Delta u = f$ to a point source

- A source at one point (red) changes u **everywhere**.
- A CNN with L layers of 3×3 filters sees only $(2L+1)^2$ pixels (amber box: $L = 5$).
- Covering the domain needs depth that grows with resolution, or down-sampling (U-Nets).

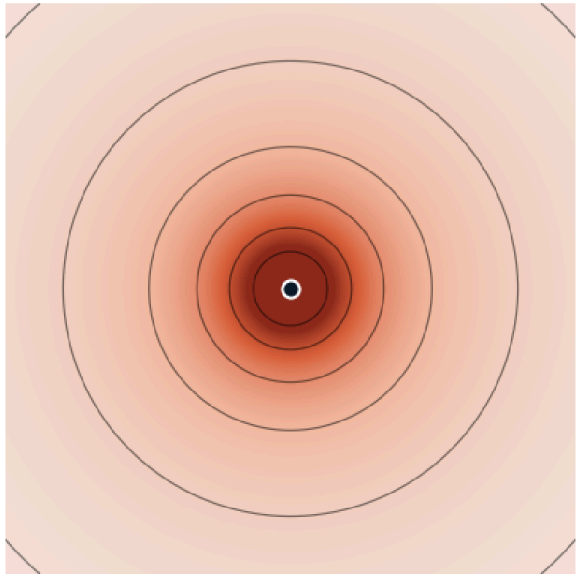
What we want: global coupling in every layer, on any grid.

Part IV

Neural operators



Electrostatics



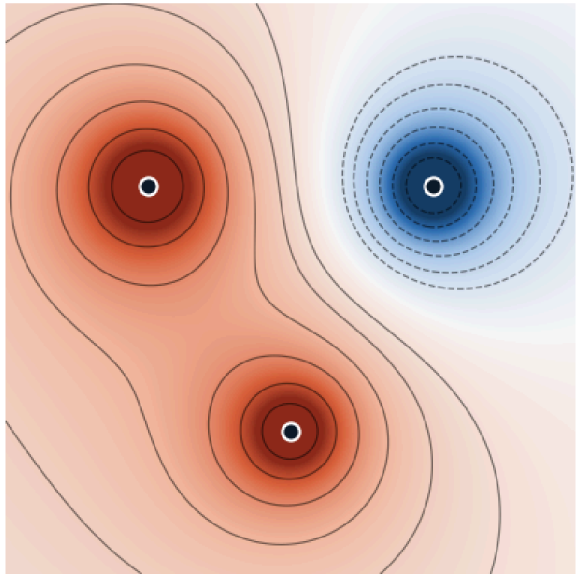
one point charge: the kernel K

$$-\nabla^2 V = \rho/\varepsilon_0 \quad \Longrightarrow \quad V(\mathbf{r}) = \int K(\mathbf{r}-\mathbf{r}')\,\rho(\mathbf{r}')\,d\mathbf{r}'$$

$$K(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0\,|\mathbf{r}|}$$

- K is the potential of a unit point charge.

Electrostatics



several charges: superposition

$$-\nabla^2 V = \rho/\varepsilon_0 \quad \Longrightarrow \quad V(\mathbf{r}) = \int K(\mathbf{r}-\mathbf{r}')\,\rho(\mathbf{r}')\,d\mathbf{r}'$$

$$K(\mathbf{r}) = \frac{1}{4\pi\varepsilon_0\,|\mathbf{r}|}$$

- K is the potential of a unit point charge.
- Any ρ is a sum of point charges, so V is a sum of shifted copies of K.
- So $\rho \mapsto V$ is one **integral**, with no iterations.

Operator layer

Standard layer on vectors $v \in \mathbb{R}^d$:

$$v_{t+1} = \sigma\big(W\, v_t + b\big)$$

Operator layer

Standard layer on vectors $v \in \mathbb{R}^d$:

$$v_{t+1} = \sigma\big(W\, v_t + b\big)$$

Neural operator layer on functions $v_t : D \to \mathbb{R}^d$:

$$v_{t+1}(x) = \sigma\Big(W\, v_t(x) \;+\; \underbrace{\int_D \kappa_\phi(x,y)\, v_t(y)\, dy}_{(\mathcal{K} v_t)(x)}\Big)$$

Operator layer

Standard layer on vectors $v \in \mathbb{R}^d$:

$$v_{t+1} = \sigma\big(W\, v_t + b\big)$$

Neural operator layer on functions $v_t : D \to \mathbb{R}^d$:

$$v_{t+1}(x) = \sigma\Big(W\, v_t(x) \;+\; \underbrace{\int_D \kappa_\phi(x,y)\, v_t(y)\, dy}_{(\mathcal{K} v_t)(x)}\Big)$$

- W: pointwise linear map (local).
- $\mathcal{K}$: integral with a **learned** kernel κ_ϕ, like K in $V = \int K\rho$ (global).
- σ: pointwise nonlinearity.

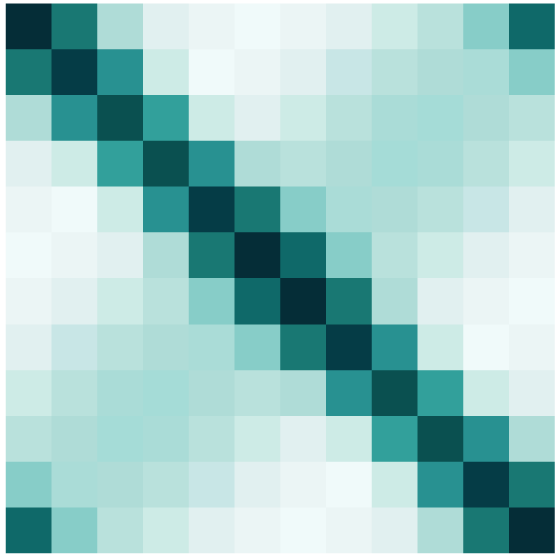
Discretization

$$(\mathcal{K}v)(x) = \int_D \kappa(x, y)\, v(y)\, dy$$

On any n-point grid $\{x_j\}$:

$$(\mathcal{K}v)(x_i) \approx \frac{|D|}{n} \sum_{j=1}^{n} \kappa(x_i, x_j)\, v(x_j)$$

- Works on any discretization by **sampling** one κ.



κ sampled on a 12-point grid

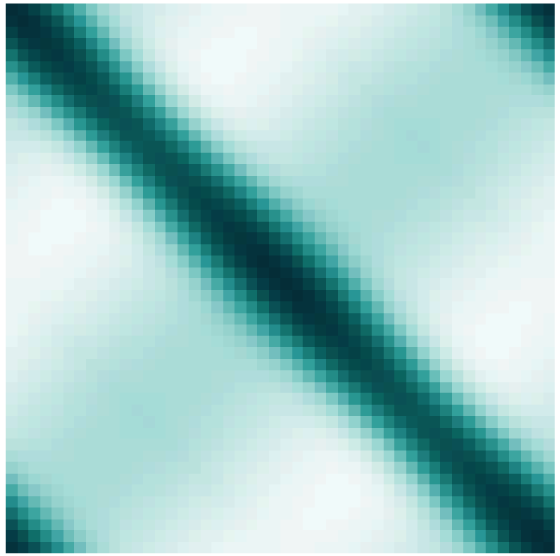
Discretization

$$(\mathcal{K}v)(x) = \int_D \kappa(x, y)\, v(y)\, dy$$

On any n-point grid $\{x_j\}$:

$$(\mathcal{K}v)(x_i) \approx \frac{|D|}{n} \sum_{j=1}^{n} \kappa(x_i, x_j)\, v(x_j)$$

- Works on any discretization by **sampling** one κ.



the same κ on a 48-point grid

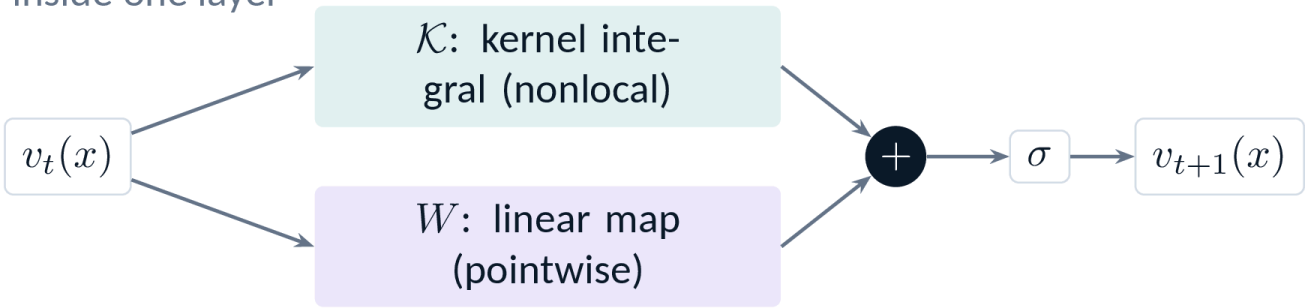
Neural operator



Neural operator

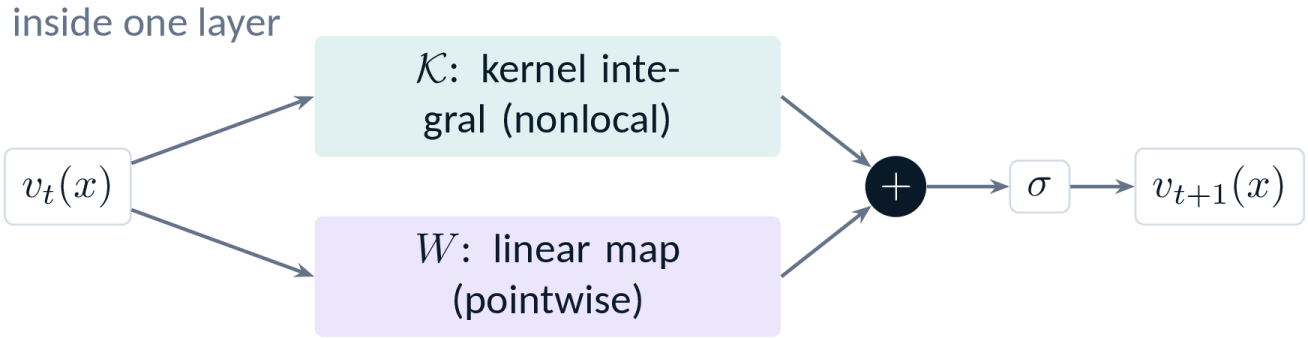


inside one layer



Neural operator



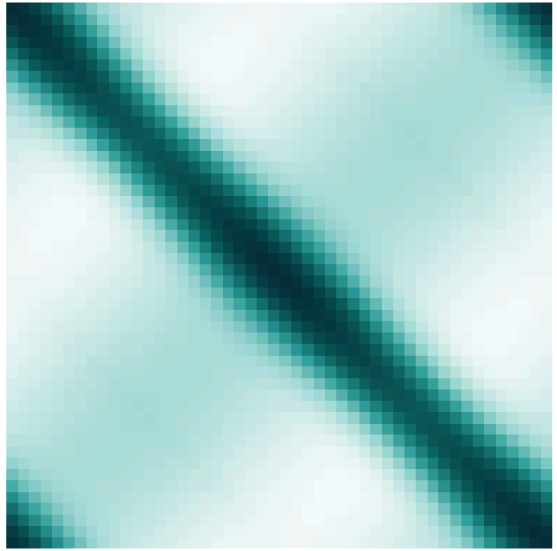


$$v_0(x) = P\big(a(x)\big), \qquad v_{t+1}(x) = \sigma\big(W v_t(x) + (\mathcal{K} v_t)(x)\big), \qquad u(x) = Q\big(v_T(x)\big)$$

Dense kernel cost

$$(\mathcal{K}v)(x_i) \approx \frac{|D|}{n} \sum_{j=1}^{n} \kappa(x_i, x_j)\, v(x_j)$$

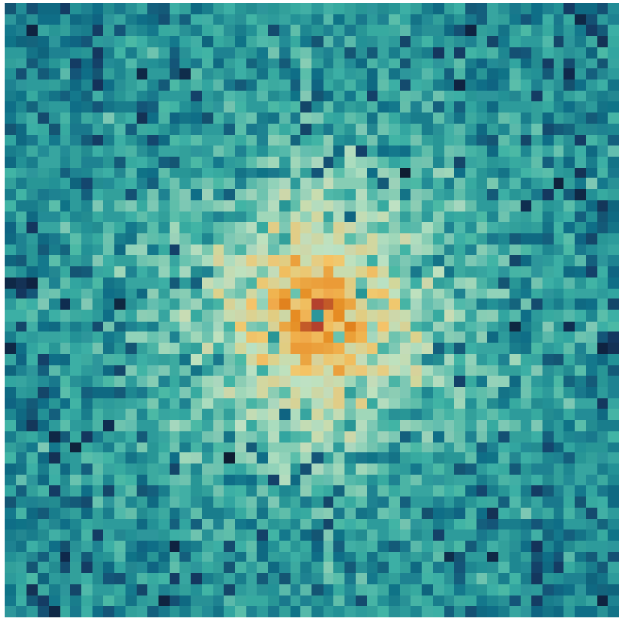
- Every point talks to every other point: n^2 terms per layer.
- A 256×256 grid has $n \approx 65{,}000$, so $n^2 \approx 4 \times 10^9$.



$\kappa(x_i, x_j)$ is an $n \times n$ matrix

Part V

Fourier neural operators



Convolution theorem

Like Coulomb's kernel, let κ depend only on $x - y$:

$$\kappa(x, y) = \kappa(x - y) \quad \Rightarrow \quad (\mathcal{K}v)(x) = \int_D \kappa(x - y)\, v(y)\, dy = (\kappa * v)(x)$$

Convolution theorem

Like Coulomb's kernel, let κ depend only on $x - y$:

$$\kappa(x, y) = \kappa(x - y) \quad \Rightarrow \quad (\mathcal{K}v)(x) = \int_D \kappa(x - y)\, v(y)\, dy = (\kappa * v)(x)$$

In Fourier space a convolution is a pointwise product:

$$\mathcal{F}(\kappa * v)(k) = \hat{\kappa}(k) \cdot \hat{v}(k) \qquad O(n \log n) \text{ with the FFT}$$

Convolution theorem

Like Coulomb's kernel, let κ depend only on $x - y$:

$$\kappa(x, y) = \kappa(x - y) \quad \Rightarrow \quad (\mathcal{K}v)(x) = \int_D \kappa(x - y)\, v(y)\, dy = (\kappa * v)(x)$$

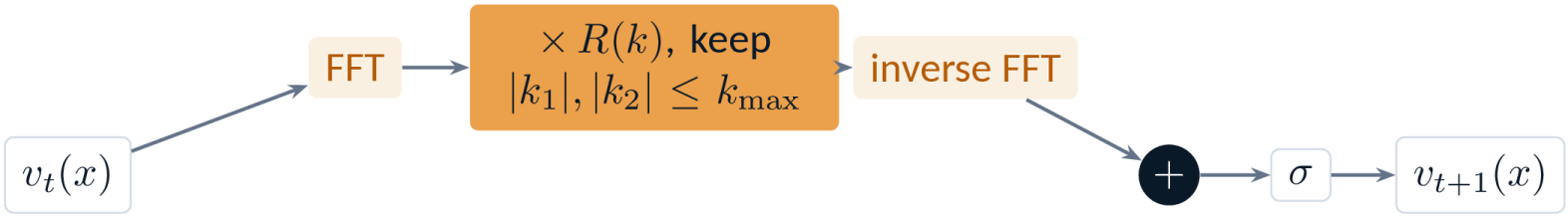
In Fourier space a convolution is a pointwise product:

$$\mathcal{F}(\kappa * v)(k) = \hat{\kappa}(k) \cdot \hat{v}(k) \qquad O(n \log n) \text{ with the FFT}$$

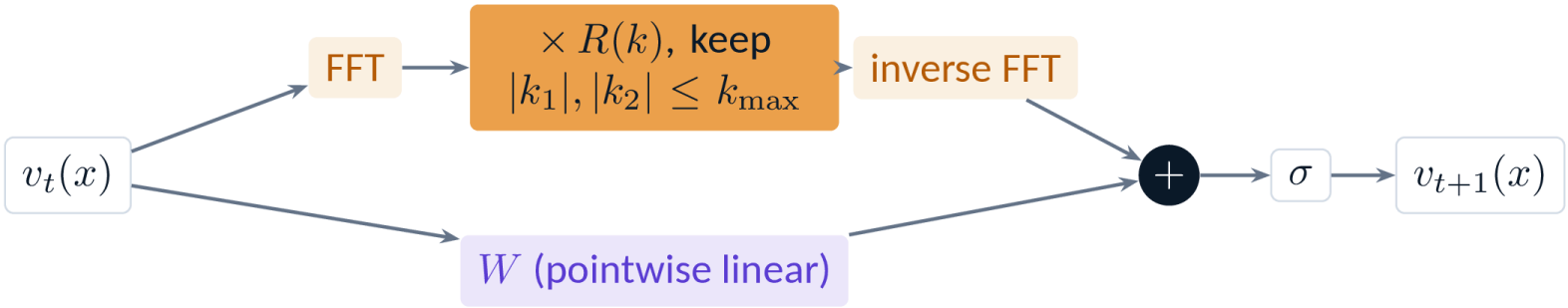
Electrostatics in Fourier space:

$$-\nabla^2 V = \rho/\varepsilon_0 \quad \xrightarrow{\mathcal{F}} \quad \hat{V}(k) = \frac{1}{\varepsilon_0 |k|^2}\, \hat{\rho}(k)$$

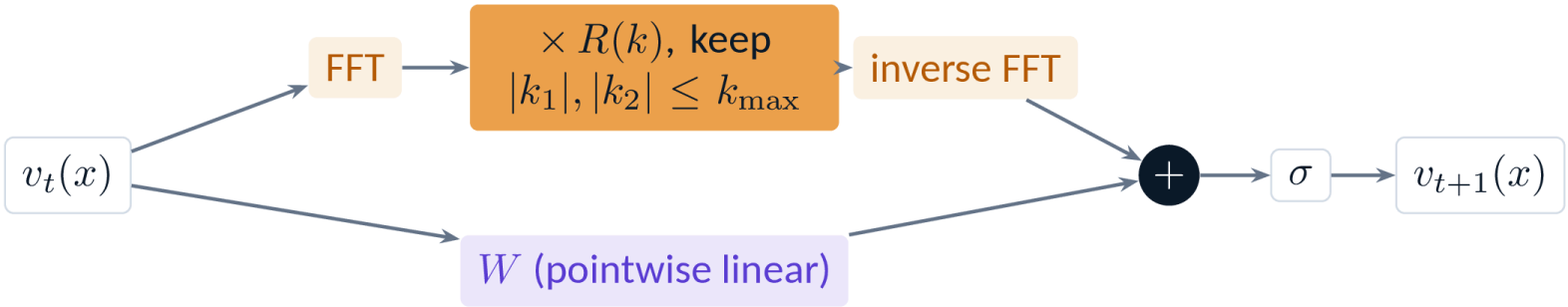
Fourier layer



Fourier layer



Fourier layer

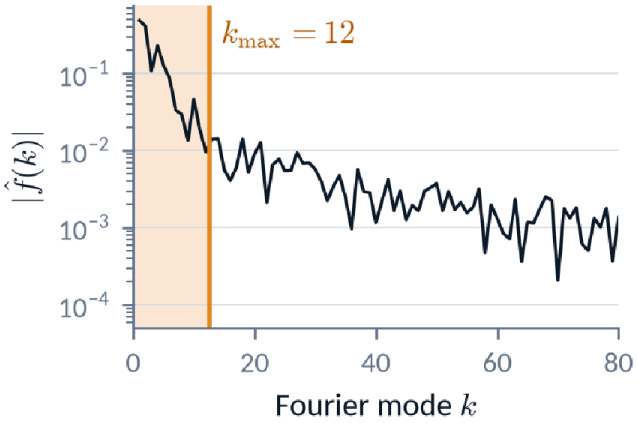


$$(\mathcal{K}v)(x) = \mathcal{F}^{-1}\big(R_\phi \cdot \mathcal{F}v\big)(x)$$

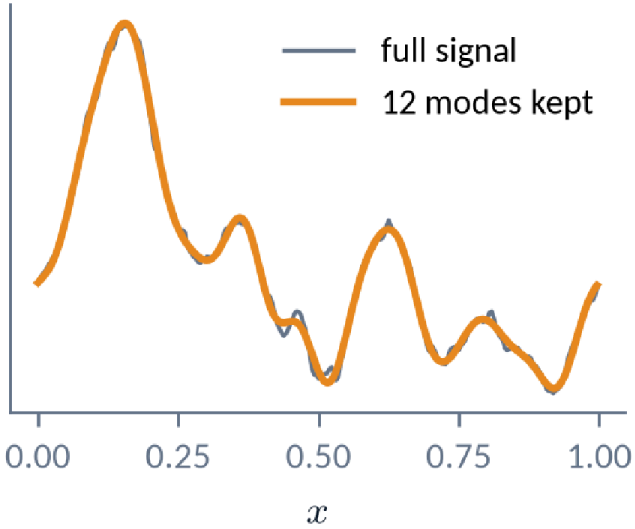
$$R \in \mathbb{C}^{2k_{\max} \times k_{\max} \times d_v \times d_v}$$

- One learned $d_v \times d_v$ complex matrix per kept mode.
- Train at 64^2, evaluate at 256^2.
- W and σ restore the dropped high frequencies.

Mode truncation



Kept modes (amber) hold nearly all energy.

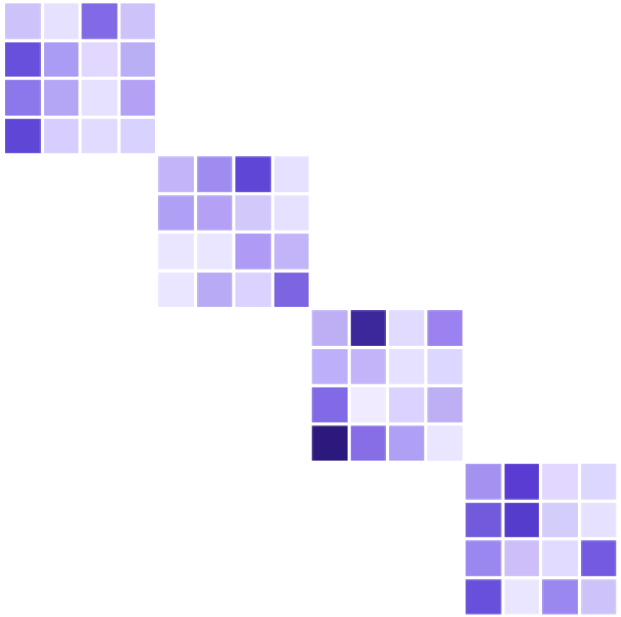


Reconstruction from the 12 lowest modes.

- Smooth fields have most of their energy at low wavenumbers.
- Cost per layer: $O(n \log n) + O(k_{\max}^2 d_v^2)$.

Part VI

Adaptive FNO



Transformer block

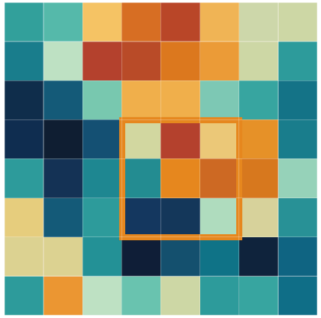
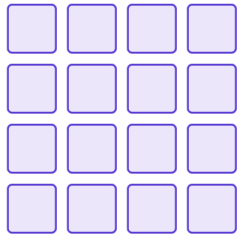


image $\to h \times w$ patches



tokens X: $h \times w$ positions, d channels each

Self-attention compares every pair of tokens: $O(N^2 d)$.



AFNO replaces the token mixer.

Attention as a kernel

1. Attention is a weighted sum over tokens:

$$\mathrm{Att}(X)[s] = \sum_{t=1}^{N} K[s,t]\, X[t]\, W_v, \qquad K[s,t] = \mathrm{softmax}_t\left(\frac{(X[s]W_q)(X[t]W_k)^\top}{\sqrt{d}}\right)$$

Attention as a kernel

1. Attention is a weighted sum over tokens:

$$\mathrm{Att}(X)[s] = \sum_{t=1}^{N} K[s,t]\, X[t]\, W_v, \qquad K[s,t] = \mathrm{softmax}_t\left(\frac{(X[s]W_q)(X[t]W_k)^\top}{\sqrt{d}}\right)$$

2. The weights form a kernel computed from the tokens themselves:

$$\kappa[s,t] = K[s,t]\, W_v \quad \longrightarrow \quad \mathcal{K}(X)(s) = \int_D \kappa\big(X(s), X(t)\big)\, X(t)\, dt$$

Attention as a kernel

1. Attention is a weighted sum over tokens:

$$\mathrm{Att}(X)[s] = \sum_{t=1}^{N} K[s,t]\, X[t]\, W_v, \qquad K[s,t] = \mathrm{softmax}_t\left(\frac{(X[s]W_q)(X[t]W_k)^\top}{\sqrt{d}}\right)$$

2. The weights form a kernel computed from the tokens themselves:

$$\kappa[s,t] = K[s,t]\, W_v \quad \longrightarrow \quad \mathcal{K}(X)(s) = \int_D \kappa\big(X(s), X(t)\big)\, X(t)\, dt$$

Attention:	$\kappa\big(X(s), X(t)\big)$, recomputed for every input	$O(N^2 d)$
Fourier mixer:	$\kappa(s-t)$, learned once	$O(Nd \log N)$

FNO as a token mixer

(1) token mixing: $z_{m,n} = [\mathrm{DFT}(X)]_{m,n}$

(2) channel mixing: $\tilde{z}_{m,n} = W_{m,n}\, z_{m,n}$

(3) token demixing: $y_{m,n} = [\mathrm{IDFT}(\tilde{Z})]_{m,n} + X_{m,n}$

$W_{m,n} \in \mathbb{C}^{d \times d}$: one matrix per frequency (m, n).

FNO as a token mixer

(1) token mixing: $z_{m,n} = [\mathrm{DFT}(X)]_{m,n}$

(2) channel mixing: $\tilde{z}_{m,n} = W_{m,n}\, z_{m,n}$

(3) token demixing: $y_{m,n} = [\mathrm{IDFT}(\tilde{Z})]_{m,n} + X_{m,n}$

$W_{m,n} \in \mathbb{C}^{d\times d}$: one matrix per frequency (m, n).

1 **Too many parameters.** $O(Nd^2)$, about 10^{10} per layer for FourCastNet.

FNO as a token mixer

(1) token mixing: $z_{m,n} = [\mathrm{DFT}(X)]_{m,n}$

(2) channel mixing: $\tilde{z}_{m,n} = W_{m,n}\, z_{m,n}$

(3) token demixing: $y_{m,n} = [\mathrm{IDFT}(\tilde{Z})]_{m,n} + X_{m,n}$

$W_{m,n} \in \mathbb{C}^{d \times d}$: one matrix per frequency (m, n).

1. **Too many parameters.** $O(Nd^2)$, about 10^{10} per layer for FourCastNet.
2. **Static filters.** $W_{m,n}$ is the same for every input.

FNO as a token mixer

(1) token mixing: $z_{m,n} = [\mathrm{DFT}(X)]_{m,n}$

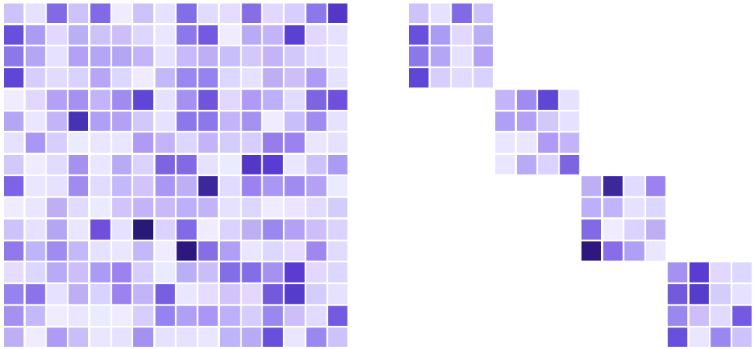
(2) channel mixing: $\tilde{z}_{m,n} = W_{m,n}\, z_{m,n}$

(3) token demixing: $y_{m,n} = [\mathrm{IDFT}(\tilde{Z})]_{m,n} + X_{m,n}$

$W_{m,n} \in \mathbb{C}^{d \times d}$: one matrix per frequency (m, n).

1. **Too many parameters.** $O(Nd^2)$, about 10^{10} per layer for FourCastNet.
2. **Static filters.** $W_{m,n}$ is the same for every input.
3. **Images are not smooth.** Edges spread over all frequencies, so a fixed $k_{\max}$ loses detail.

Fix 1: block-diagonal weights



dense: d^2 weights k blocks: d^2/k weights

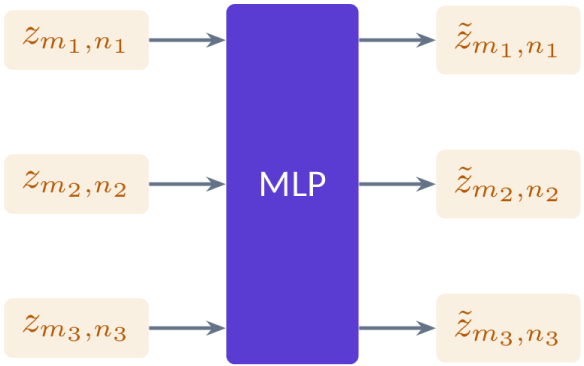
$$W = \mathrm{diag}\big(W^{(1)}, \ldots, W^{(k)}\big), \qquad W^{(\ell)} \in \mathbb{C}^{\frac{d}{k} \times \frac{d}{k}}$$

- Each block mixes its own d/k channels, like attention heads.
- k times fewer weights and FLOPs.
- $k = 1$ is dense FNO. FourCastNet uses $k = 8$.

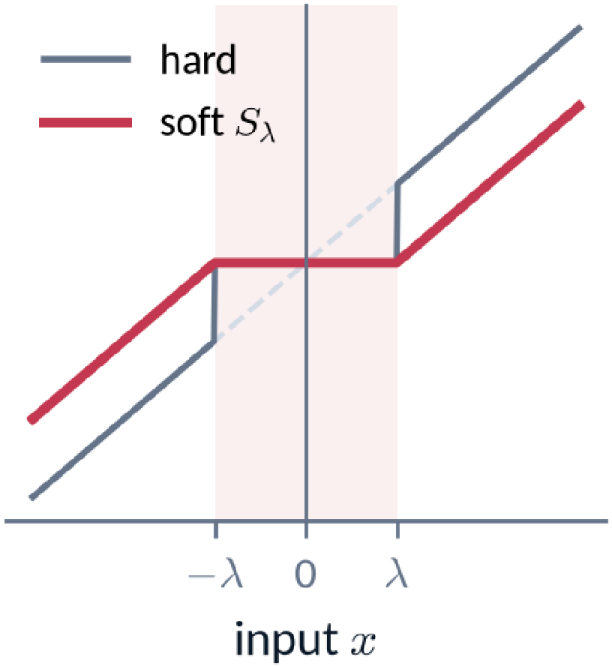
Fix 2: shared MLP

$$\tilde{z}_{m,n} = W_2\,\sigma\big(W_1 z_{m,n} + b_1\big) + b_2$$

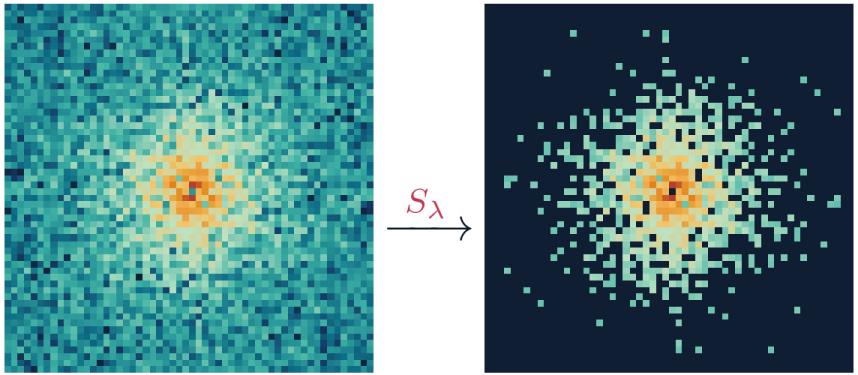
- **Shared**: the same weights at every frequency, so parameters do not grow with N.
- **Adaptive**: the nonlinearity makes the filter depend on the input, as in attention.
- Still block-diagonal: W_1, W_2 have shape $[k, d/k, d/k]$.



Fix 3: soft-thresholding



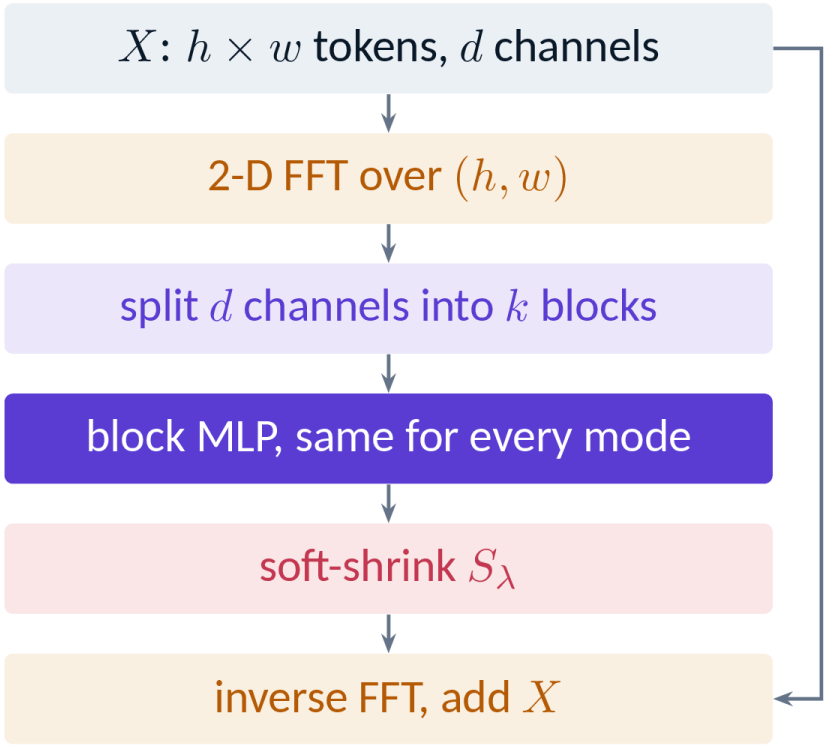
$$S_\lambda(x) = \operatorname{sign}(x)\max(|x| - \lambda, 0)$$



$|\mathrm{DFT}|$ of the tokens (log) after shrinkage

- Zeroes small coefficients **per input**, replacing the fixed $k_{\max}$.
- The ℓ_1 (LASSO) shrinkage step, which promotes sparse spectra.
- FourCastNet uses $\lambda = 0.01$.

AFNO mixer



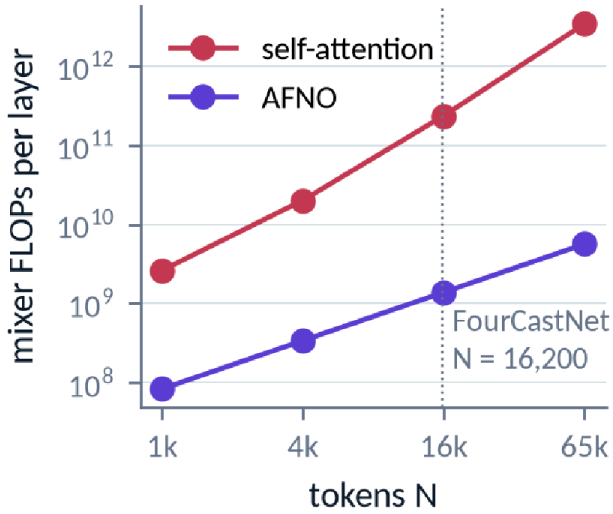
$$y = \mathrm{IDFT}\Big(S_\lambda\big(\mathrm{MLP}(\mathrm{DFT}(X))\big)\Big) + X$$

Cost

Mixer	FLOPs	Parameters
Self-attention	$N^2d + 3Nd^2$	$3d^2$
FNO	$Nd^2 + Nd\log N$	Nd^2
AFNO	$Nd^2/k + Nd\log N$	$4d^2/k + 4d$

$N = hw$ tokens, d channels, k blocks.

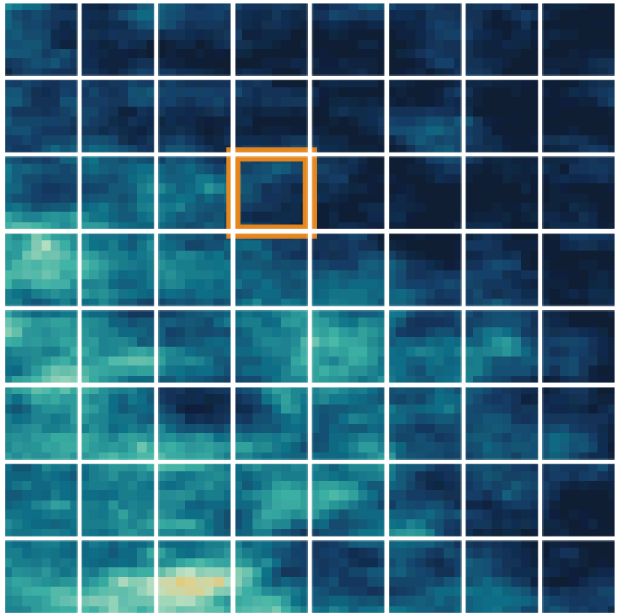
- Linear in N up to the FFT's $\log N$.
- Parameter count independent of N.



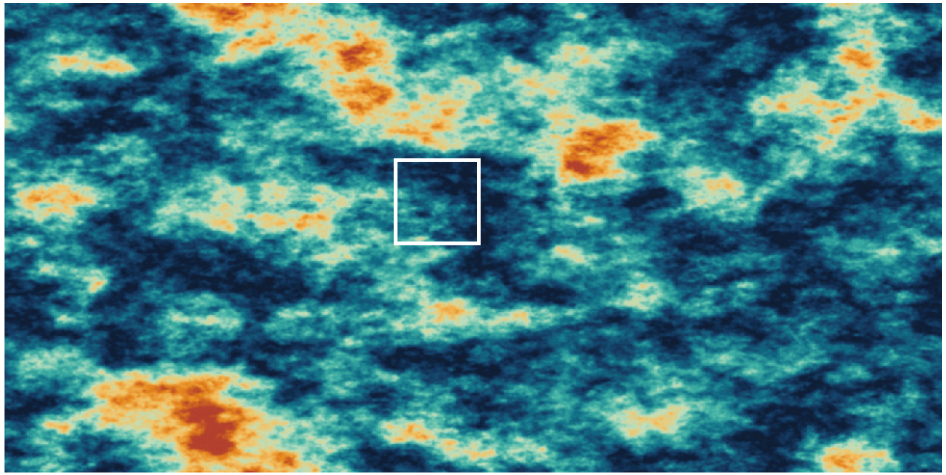
FLOPs at $d = 768$, $k = 8$.

Part VII

FourCastNet



Task



$0.25°$ grid: 720×1440 points.

$$X(k{+}1) \approx \mathcal{N}_\theta\big(X(k)\big), \qquad \Delta t = 6\ \mathrm{h}$$

20 ERA5 variables = 20 channels

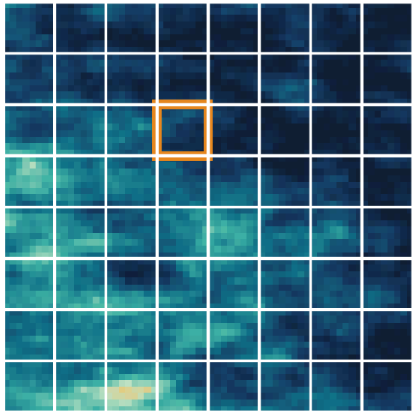
50 hPa	Z
500 hPa	$T,\ U,\ V,\ Z,\ RH$
850 hPa	$T,\ U,\ V,\ Z,\ RH$
1000 hPa	$U,\ V,\ Z$
Surface	$U_{10},\ V_{10},\ T_{2m},\ sp,\ mslp$
Column	$TCWV$

Trained on ERA5 reanalysis, 1979–2015.

Architecture

$X(k)$ $720 \times 1440 \times 20$ → **Patch embedding** 8×8 patches → $90 \times 180 = 16{,}200$ tokens, $d = 768$ → **12 × AFNO block** LayerNorm → AFNO → LayerNorm → MLP → **Linear decoder** token → $8 \times 8 \times 20$ patch → $X(k+1)$ $720 \times 1440 \times 20$

Patch size	8×8
Embedding dim d	768
Depth	12
Blocks k	8
Sparsity λ	0.01
MLP ratio, activation	4, GELU

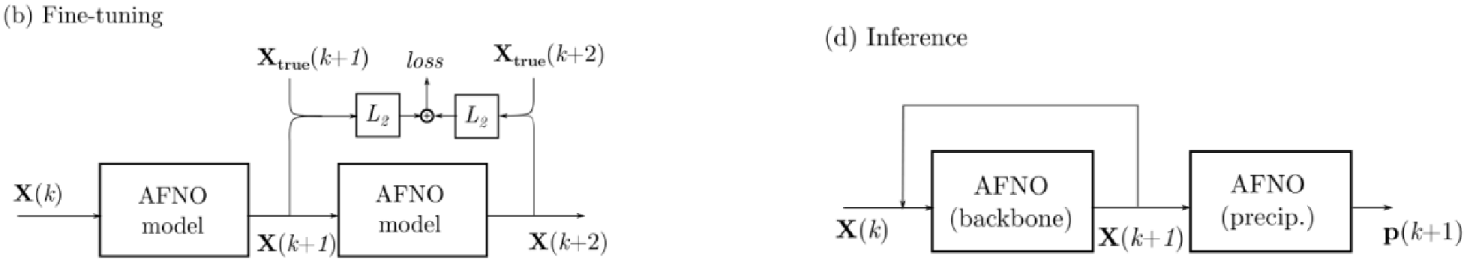


one 8×8 patch (amber) → one token

- Learned position embedding added to the tokens.
- Every AFNO layer mixes the whole globe.

Training

- Train to predict atmospheric state 6 hours ahead; $X(k) \rightarrow X(k+1)$
- Pre-training: supervised one step loss
- Fine-tuning: Feed the first prediction back to the model and sum two losses
- Inference: recursively feed predictions back into the trained model



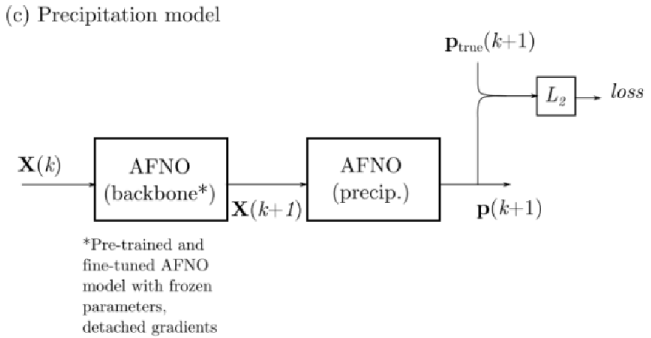
- Cosine learning rate

- Dataset: Train (1979-2015) / Validation (2016-2017) / Test (2018 -)

Precipitation model

Why separate model?

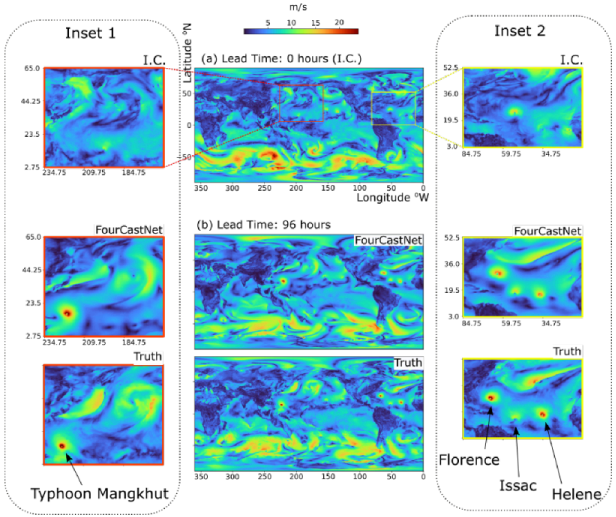
- Total precipitation (TP) is sparse and heavy-tailed
  : strongly peaked at zero, with a long positive tail
- TP has little influence on variables guiding dynamical evolution
- Train separate model to predict 6 hour accumulated precipitation from frozen backbone model



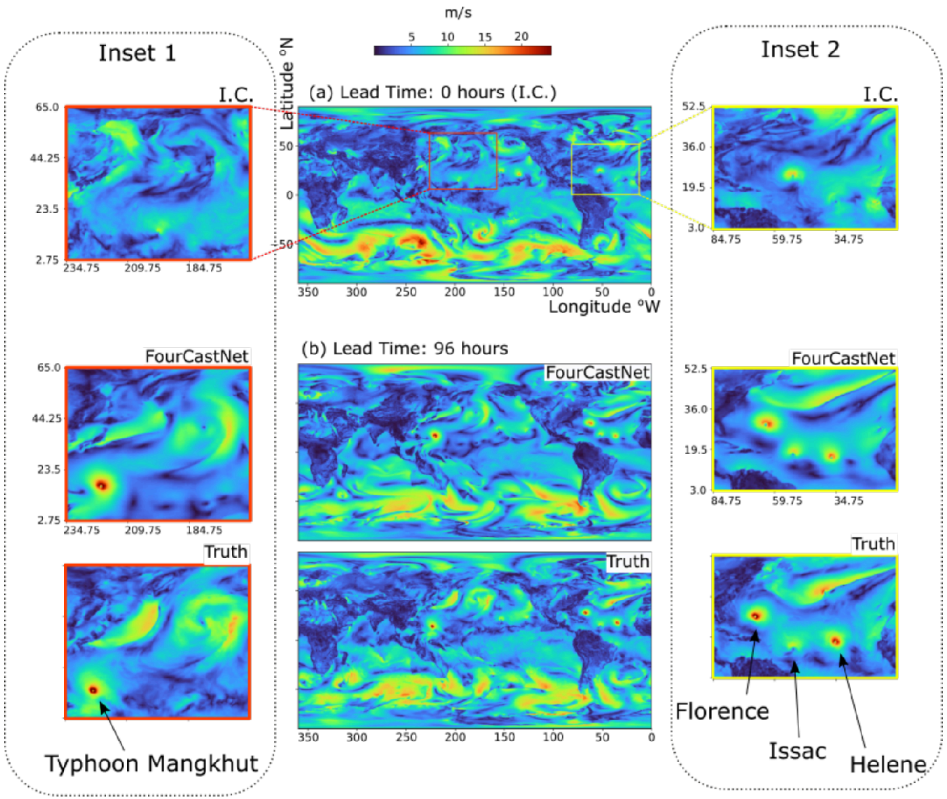
- Design: AFNO + 2D CNN + ReLU → non-negative value
- Log transform total precipitation: $\widetilde{TP} = \log(1 + TP/\varepsilon)$, $\varepsilon = 10^{-5}$
  : Discourages all-zero predictions, reduces the skewness of the target distribution

Part VIII

Results

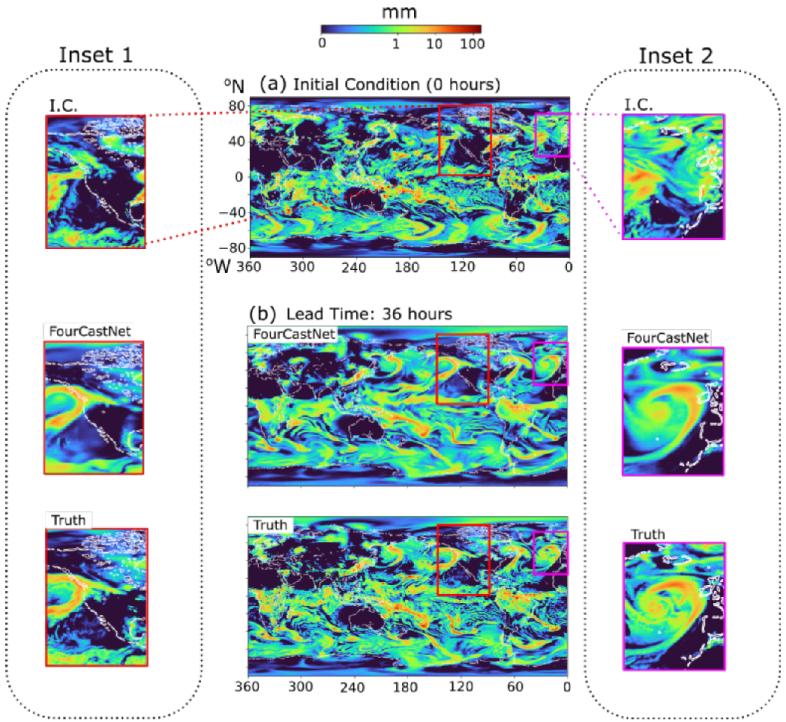


Result: 96-hour global near-surface wind forecast

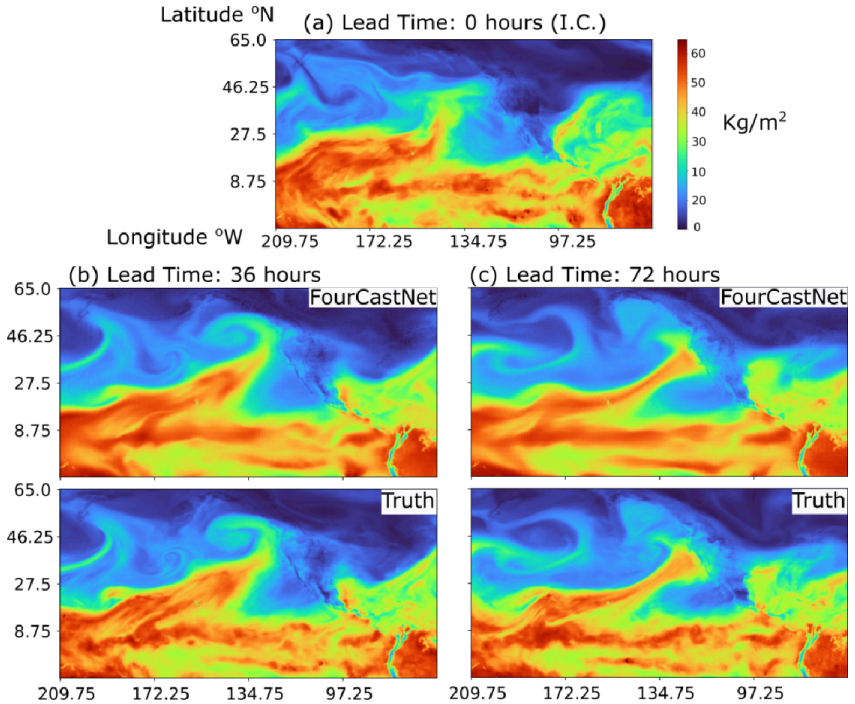


- Captures the formation/development of Typhoon and Hurricanes
- The forecast is visibly smoother/blurred than the truth (ERA5)

Result: precipitation and an atmospheric river

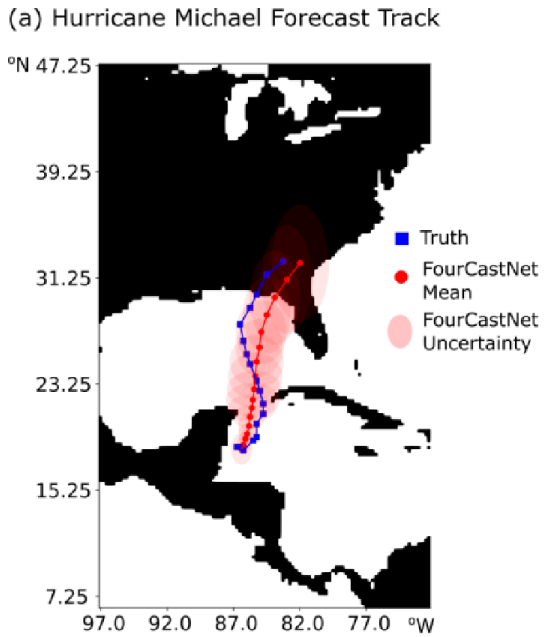


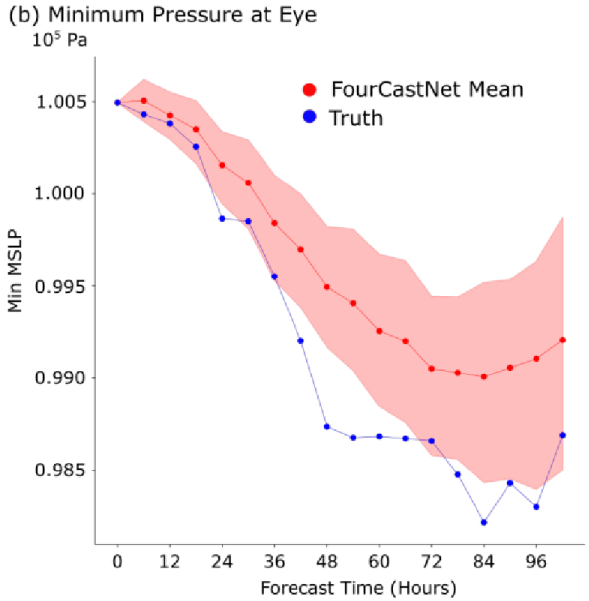
36h precipitation forecast



36/72h TCWV forecasts

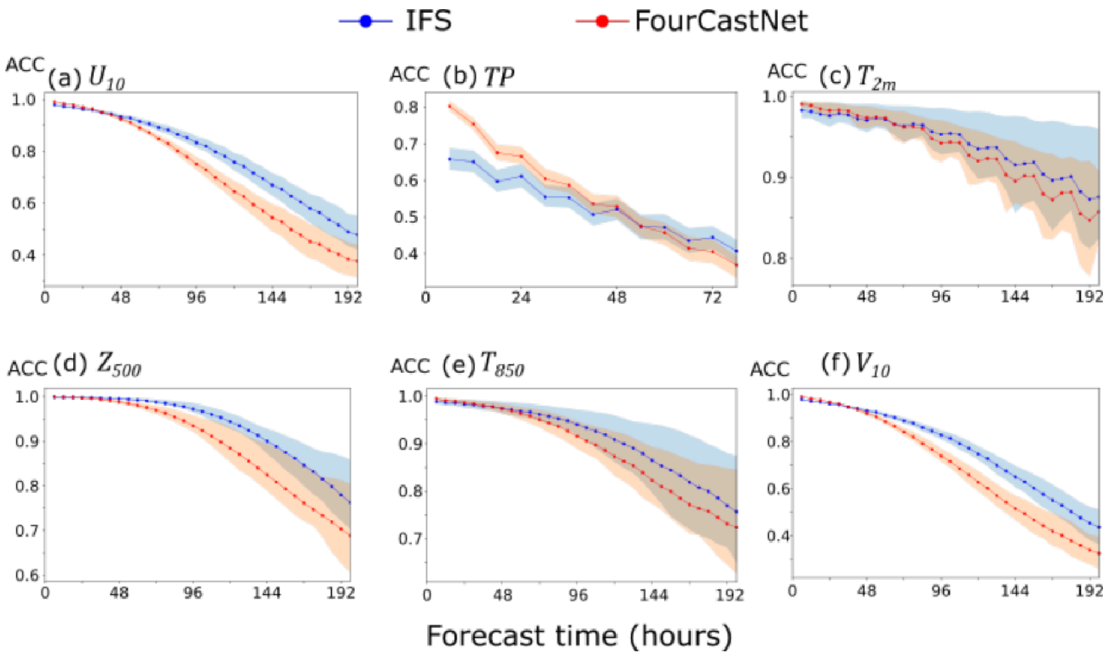
Case study: Hurricane forecast with a 100-member ensemble





- Can predict the track and intensification of Hurricane
- Misses the rapid intensification during 36-48 h

Quantitative skill vs ECMWF IFS (test year 2018)



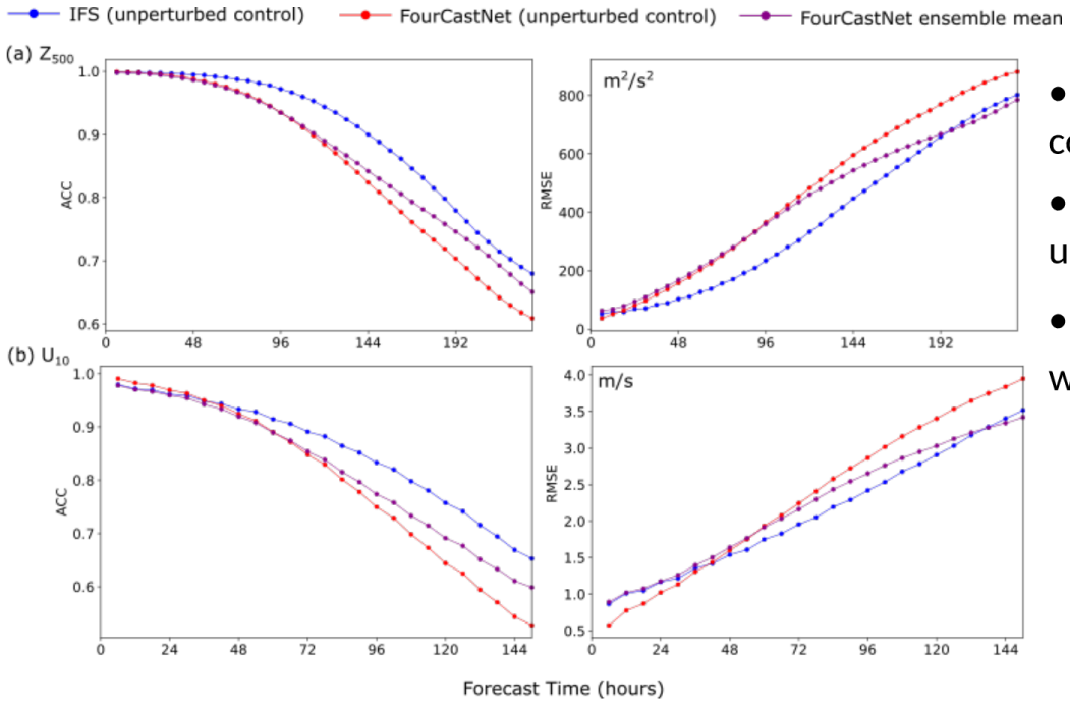
- Latitude weighted ACC

$$\mathrm{ACC} = \frac{\sum_i \omega_i \left(X^i_{\mathrm{pred}} - \overline{X}^i\right)\left(X^i_{\mathrm{true}} - \overline{X}^i\right)}{\sqrt{\sum_i \omega_i \left(X^i_{\mathrm{pred}} - \overline{X}^i\right)^2}\sqrt{\sum \omega_i \left(X^i_{\mathrm{true}} - \overline{X}^i\right)^2}}$$

$\bar{X}$: climatological mean,
ω_i: Latitude weight = cos(latitude)

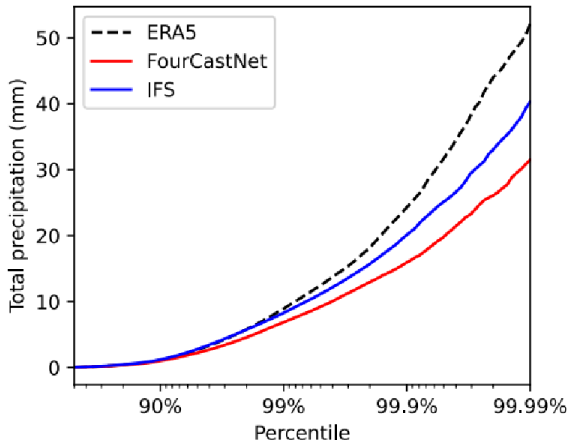
- Competitive with IFS at short time
  - higher precipitation ACC up to 48 h
- Falls behind IFS in longer time

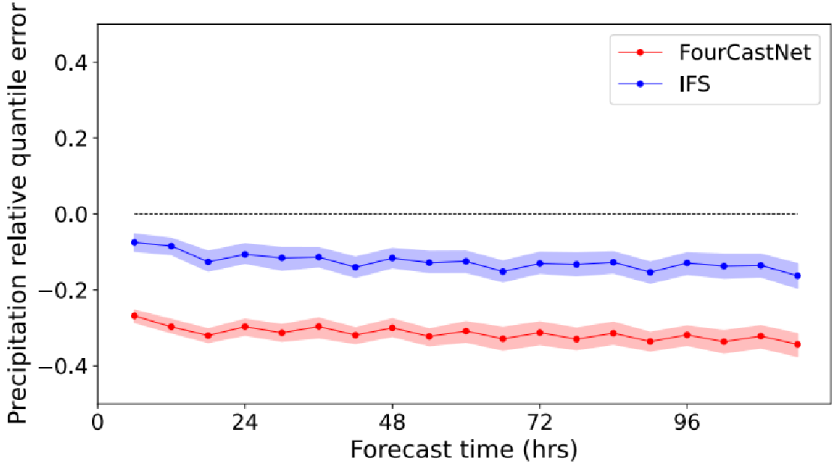
Improved long term forecast through ensemble averaging



- Gaussian noise is added to the initial condition
- Ensemble average outperforms the unperturbed FCN in long term
- Slightly worse at short time: averaging washes out fine scales

Underestimation of extreme precipitation





- Relative quantile error over 50 quantiles from 90% to 99.99%, log-spaced.

$$\mathrm{RQE}(l) = \sum_{q \in Q} \frac{X^q_{\mathrm{pred}}(l) - X^q_{\mathrm{true}}(l)}{X_{\mathrm{true}}(l)}, \quad Q = 1 - \{10^{-1}, \cdots, 10^{-4}\}$$

- Underestimates the upper tail of precipitation

  - fails to fully resolve extreme values concentrated in small regions

Computational cost: orders of magnitude cheaper at inference

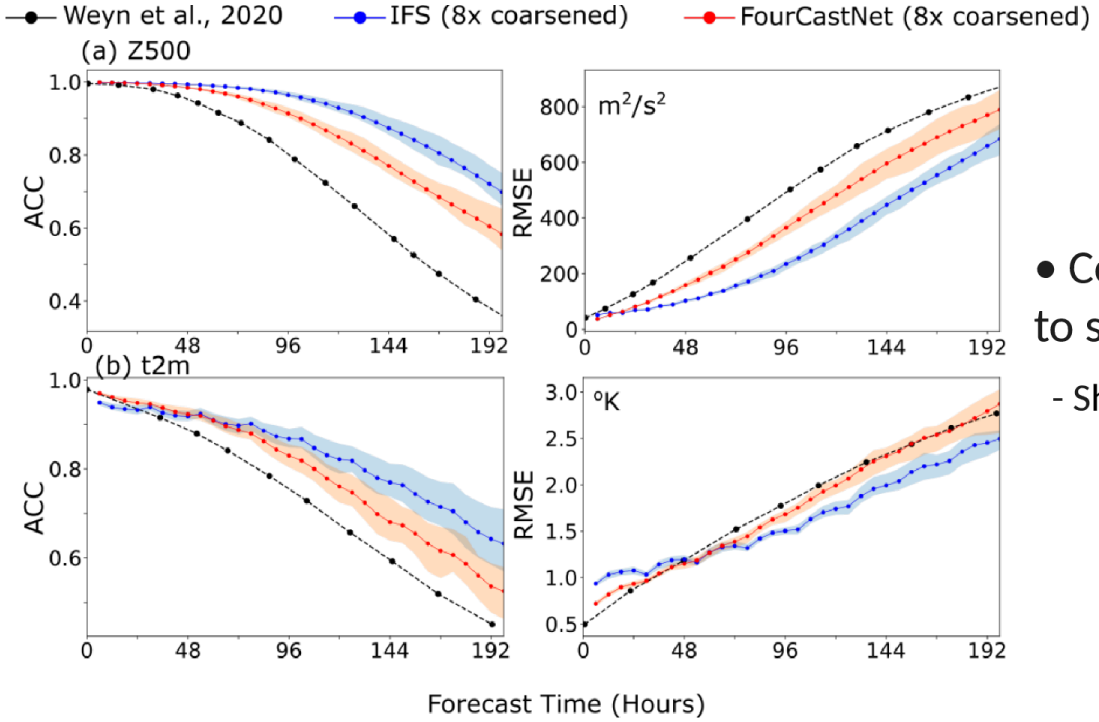
- Caveat: These are rough cost estimates; the cost of FCN-18km is extrapolated estimate

IFS runs on CPU. FCN runs on GPU

Latency and Energy consumption for a 24-hour 100-member ensemble forecast				
	IFS	FCN - 30km (actual)	FCN - 18km (extrapolated)	IFS / FCN(18km) Ratio
Nodes required	3060	1	2	**1530**
Latency (Node-seconds)	984000	7	22	**44727**
Energy Consumed (kJ)	271000	7	22	**12318**

- FourCastNet can compute a 100-member ensemble, 24-hour forecast in 7s, using one node of 4 A100 GPUs
- Enables large ensembles, compared to 51 ensemble members of IFS

Comparison with the prior DL state of the art (DLWP)



- Compare with DLWP after coarsening to same resolution $\sim 2°$
  - Shows clear improvement

Implications and Limitations

- Predictions are 4–5 orders of magnitude faster than traditional NWP models
  - Large ensembles can be generated in few seconds
  - Suitable for rapidly testing hypothesis

- Spherical geometry is neglected: Fourier transform on Euclidean 2d grid
- Physical knowledge is not incorporated.

Weather models after FCN

- Geometric information
  - Pangu-Weather (2022): 3D Earth-specific Transformer
  - GraphCast (2022): Multi-scale spherical mesh GNN
  - SFNO / FourCastNet v2 (2023): spherical harmonic transform
- Combine with physics
  - NeuralGCM (2023): ODE solved by physical solver (large scale) + neural network (small scale)
- Probabilistic forecasting
  - GenCast (2023): Diffusion based ensemble model
  - FourCastNet v3 (2025): local/global spherical convolution + probabilistic latent state

Summary

- FCN trains to predict states after 6 hours
- AFNO reduces the computational cost for spatial correlation, compared to Transformer
- Its performance matches with state-of-art NWP in short time scale ~48h, with way less computational cost
- It enables forecasting with large number of ensembles / rapid test of hypothesis