Wang & Lai, J. Comput. Phys. (2024)
Wang, Lai, Gómez-Serrano, Buckmaster, Physical Review Letters (2023)
Wang et al. arXiv:2509.14185

Stanford Doerr School of Sustainability

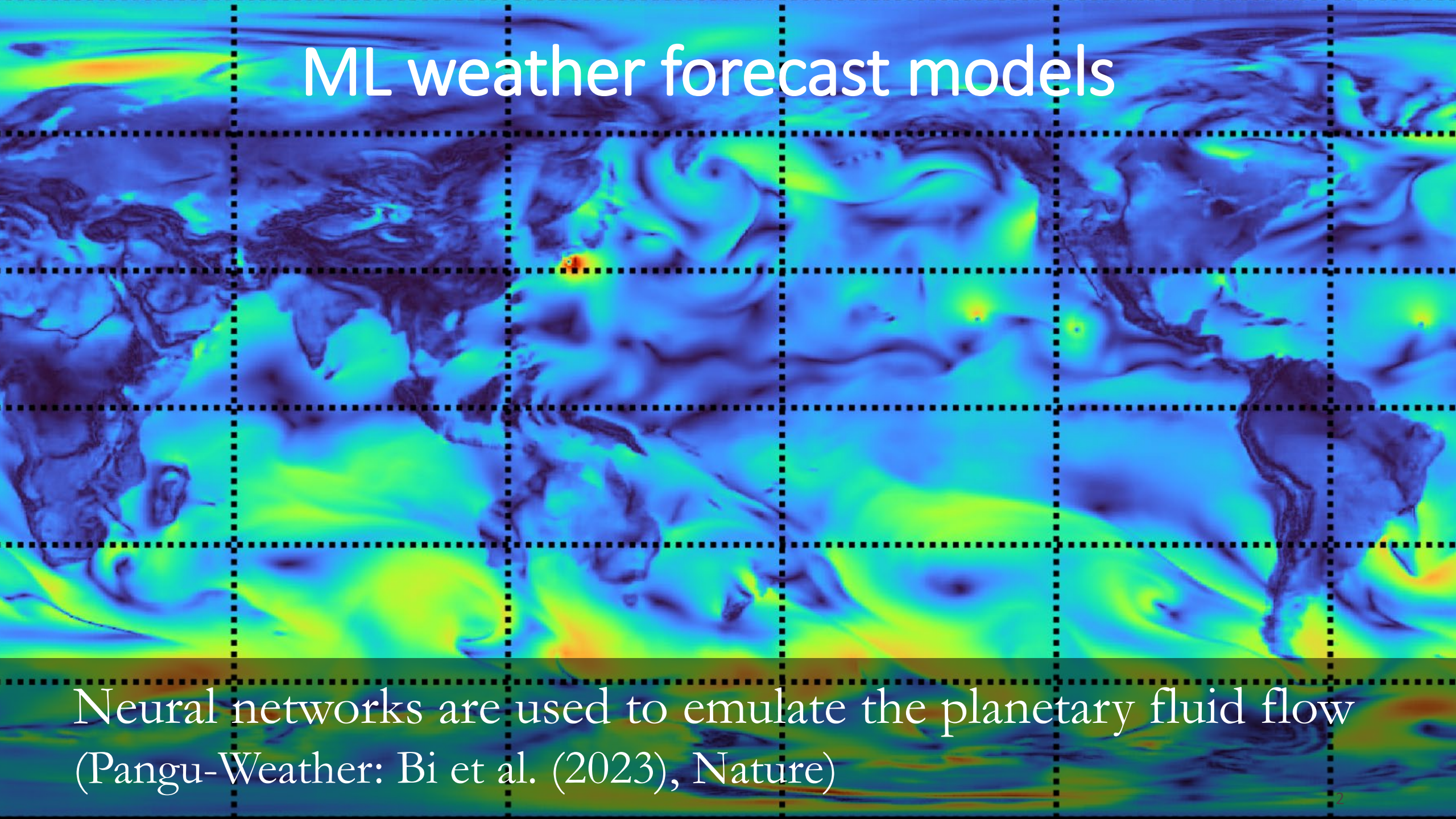
Solution discovery in fluids with high precision using neural networks

Ching-Yao Lai

Stanford University

Image generated by Generative AI

ML weather forecast models

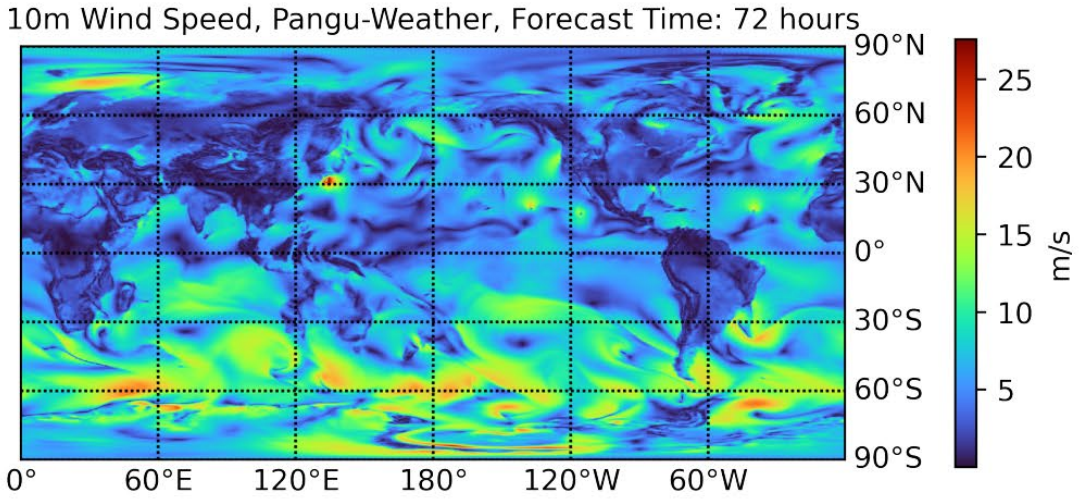


Neural networks are used to emulate the planetary fluid flow
(Pangu-Weather: Bi et al. (2023), Nature)

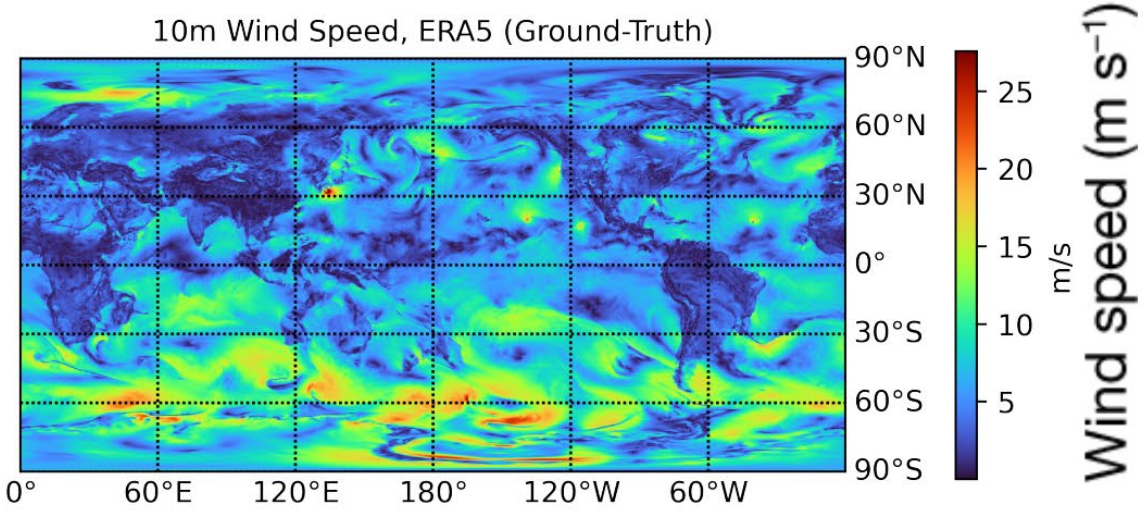
Some challenges and progress

AI weather forecast

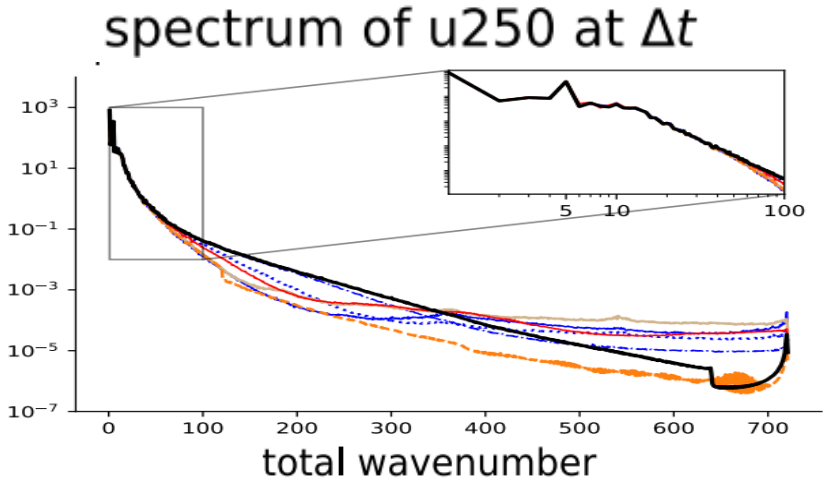
Bi et al (2023) Nature

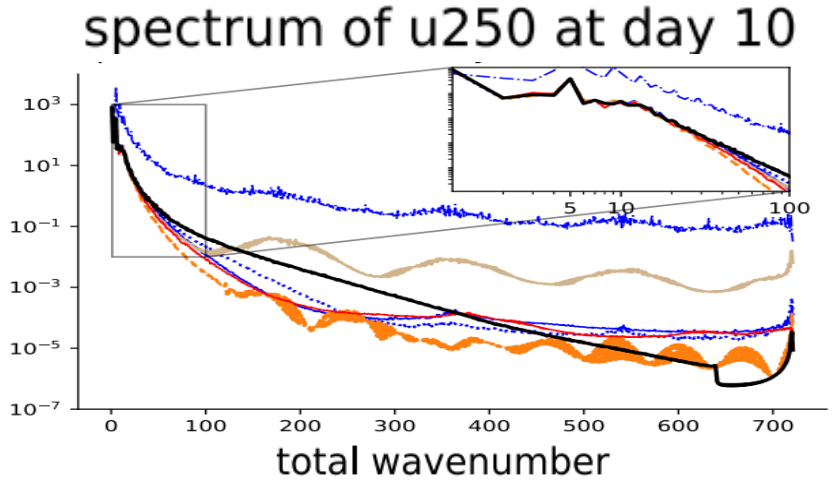


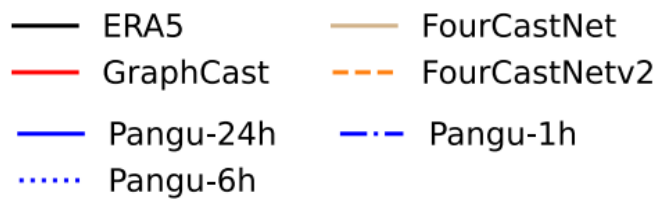
Training data (Re-analysis)



- Spectral bias of NN:



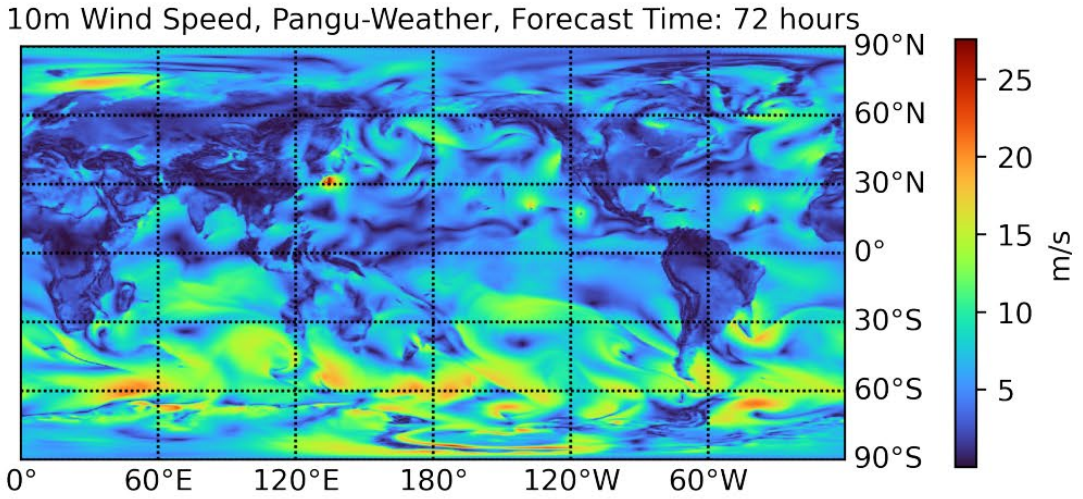




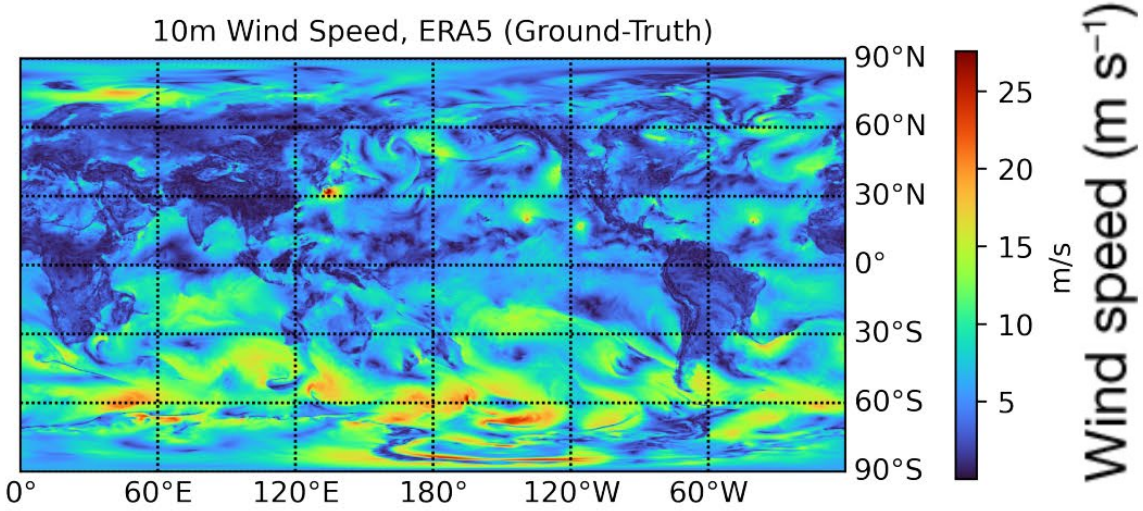
Lai et al, Annu. Rev. Condens. Matter Phys. (2024)

Spectral bias is bad for multiscale problems

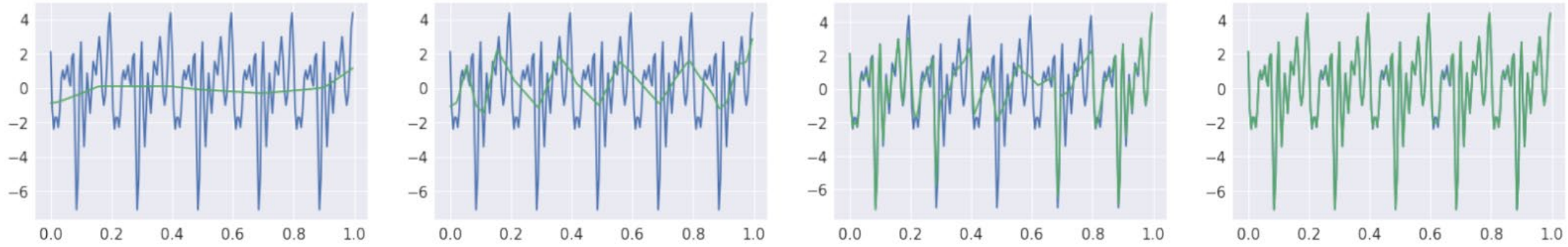
AI weather forecast Bi et al (2023) Nature



Training data (Re-analysis)



- Spectral bias of NN: Rahaman et al (2019), Frequency Principle: Xu et al (2020)



(a) Iteration 100 (b) Iteration 1000 (c) Iteration 10000 (d) Iteration 80000

Outline

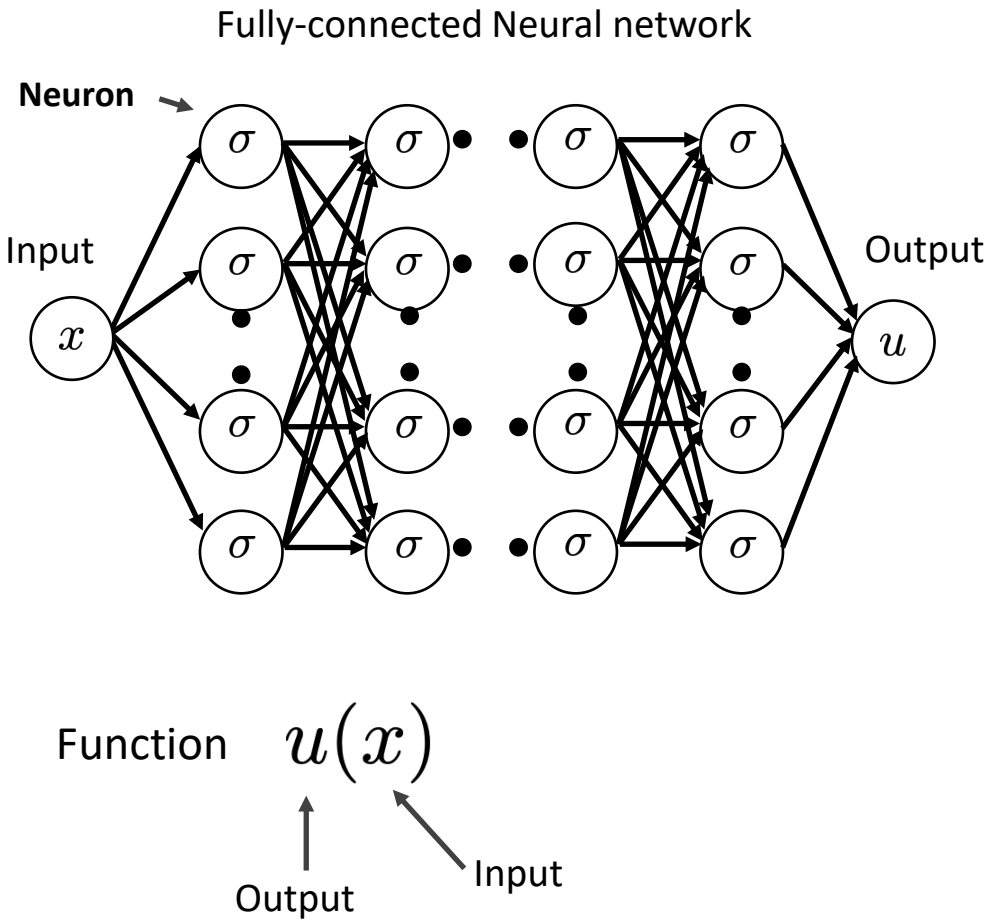
- High-precision NN algorithm for tackling the spectral bias
- Applications to finding PDE solutions numerically
- Applications of finding singularities in fluids

Outline

- High-precision NN algorithm for tackling the spectral bias
- Applications to finding PDE solutions numerically
- Applications of finding singularities in fluids

Neural network

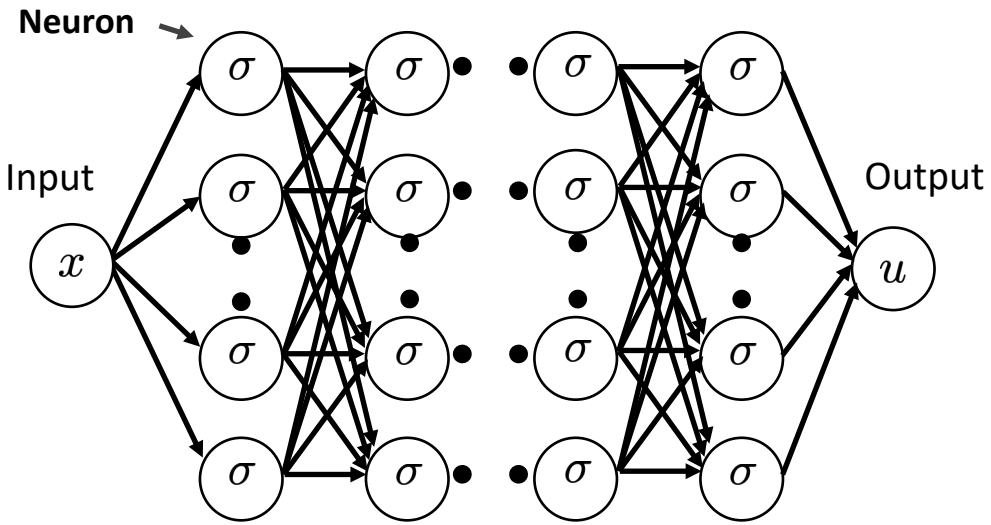
Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*



Neural network

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*

Fully-connected Neural network



Function $u(x) = \sum_{j=1} w_{lk}^{(n)} \sigma \left(\sum_{i=1} w_{kj}^{(n-1)} \sigma \left(\dots \sigma \left(\sum_{i=1} w_{ji}^{(1)} \sigma \left(w_i^{(0)} x + b_i^{(0)} \right) + b_j^{(1)} \right) \dots \right) + b_k^{(n-1)} \right) + b_l^{(n)}$

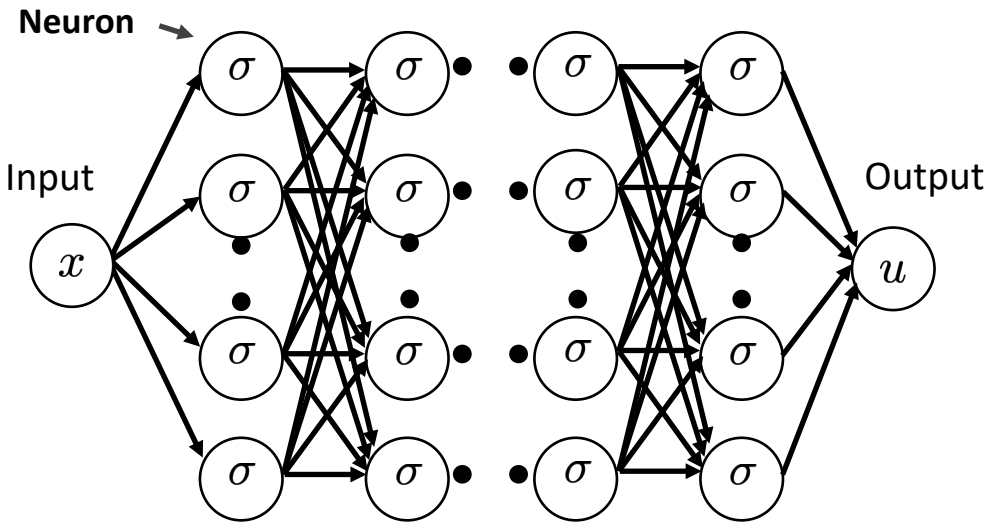
$\mathbf{w}$: weights $\quad \mathbf{b}$: biases $\quad \sigma(x)$: activation function

(free parameters to be trained) *(fixed and selected by users)*

Neural network

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*

Fully-connected Neural network



$$u(x) = \sum_{j=1} w_{lk}^{(n)} \sigma \left(\dots \sigma \left(\sum_{i=1} w_{ji}^{(1)} \, \sigma \left(w_i^{(0)} x + b_i^{(0)} \right) + b_j^{(1)} \right) \dots \right) + b_l^{(n)}$$

$\mathbf{w}$: weights $\quad$ $\mathbf{b}$: biases $\quad$ $\sigma(x)$: activation function

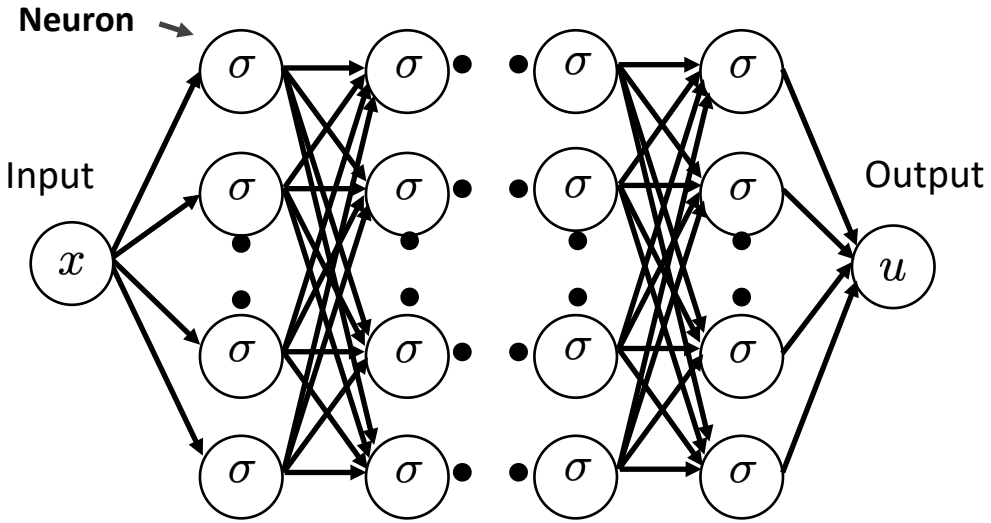
Universal function approximator

Hornik et. al. (1989), *Neural Netw*. **2**

Neural network and Fourier series

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*

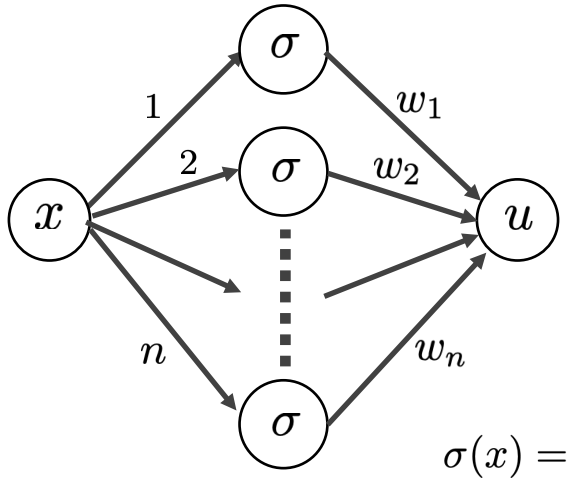
Fully-connected Neural network



$$u(x) = \sum_{j=1} w_{lk}^{(n)} \sigma \left(\dots \sigma \left(\sum_{i=1} w_{ji}^{(1)} \, \sigma \left(w_i^{(0)} x + b_i^{(0)} \right) + b_j^{(1)} \right) \dots \right) + b_l^{(n)}$$

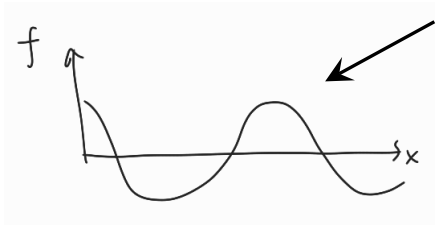
$\mathbf{w}$: weights $\quad$ $\mathbf{b}$: biases $\quad$ $\sigma(x)$: activation function

Fourier series: $u(x, w_n, b_n) = \sum_{n=0}^{N} w_n \ \sin(nx)$

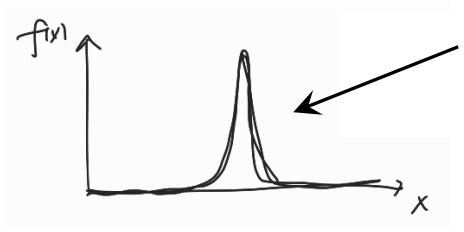


Universal function approximator

Hornik et. al. (1989), *Neural Netw.* **2**



Use sin() activation



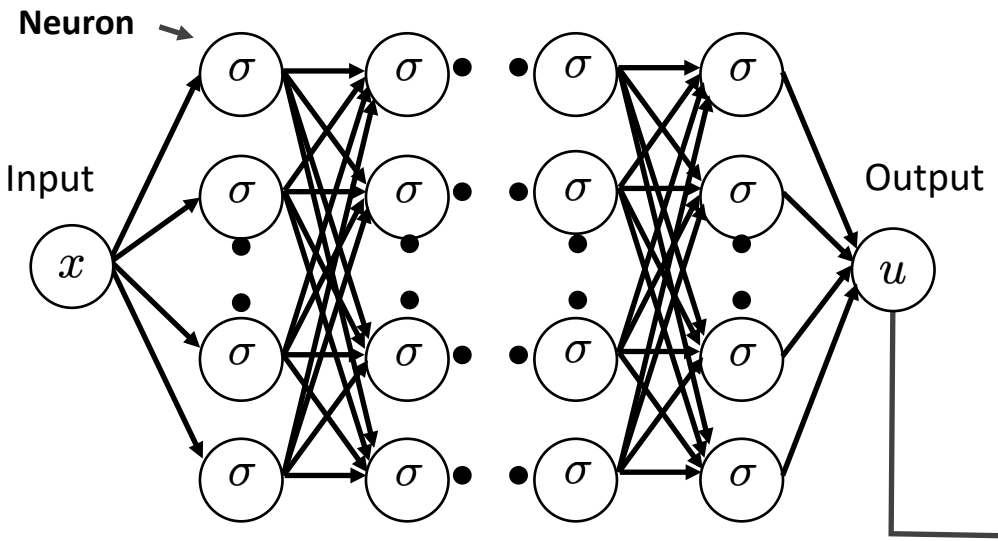
If using sin() activation, need lots of neurons (frequencies)

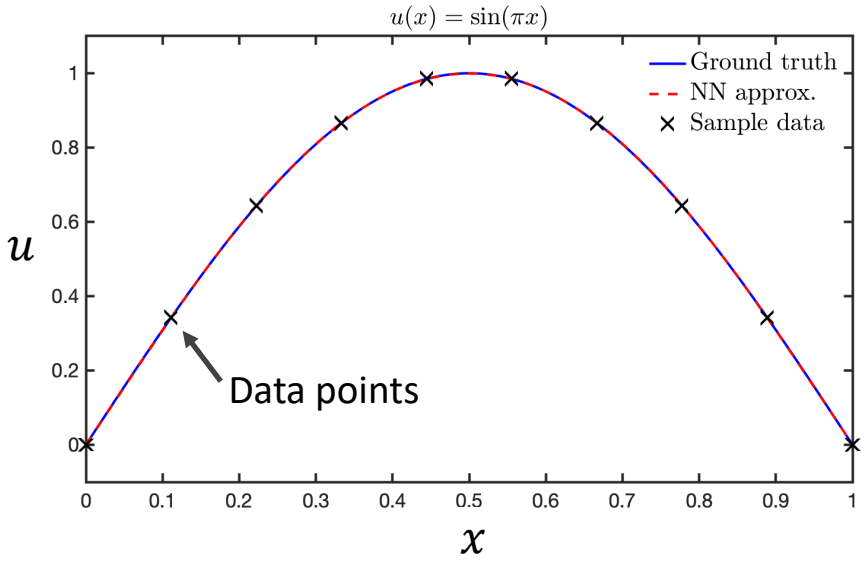
Better to use tanh() activation

Neural network for curve fitting

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*

Fully-connected Neural network





$$u(x) = \sum_{j=1} w_{lk}^{(n)} \sigma \left(\dots \sigma \left(\sum_{i=1} w_{ji}^{(1)} \, \sigma \left(w_i^{(0)} x + b_i^{(0)} \right) + b_j^{(1)} \right) \dots \right) + b_l^{(n)}$$

Updating variables: $\mathbf{w}$: weights $\mathbf{b}$: biases

Optimization ← Loss

Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \, \nabla_{\mathbf{w}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$$\mathbf{b}^{(i+1)} = \mathbf{b}^{(i)} - \eta \, \nabla_{\mathbf{b}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$\mathbf{w}^{(i)}, \mathbf{b}^{(i)}$: value at the i-th iteration η : learning rate

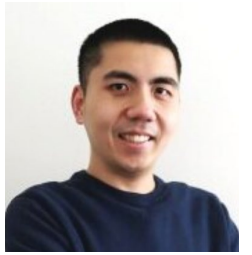
Cost function: *mean squared error*

data of u at $x = x_i$

$$J(x, \mathbf{w}, \mathbf{b}) = loss_d = \frac{1}{N_d} \sum_{i=1}^{N_d} [u(x_i, \mathbf{w}, \mathbf{b}) - u_i^{(d)}]^2$$

Expressiveness of neural networks

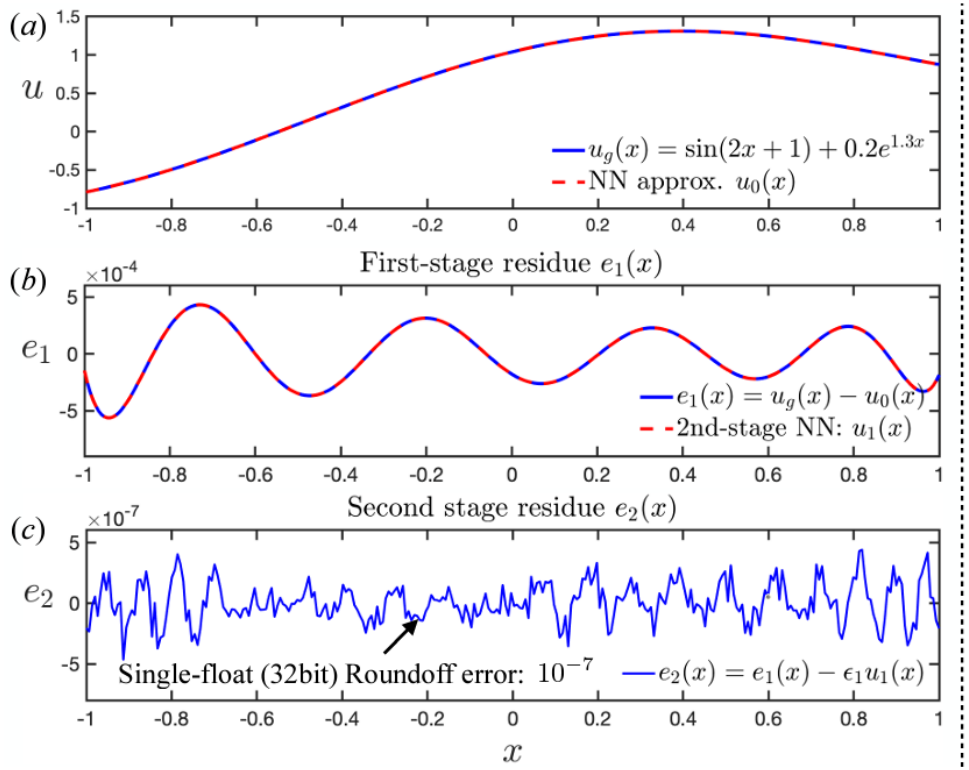
Yongji Wang
Postdoc

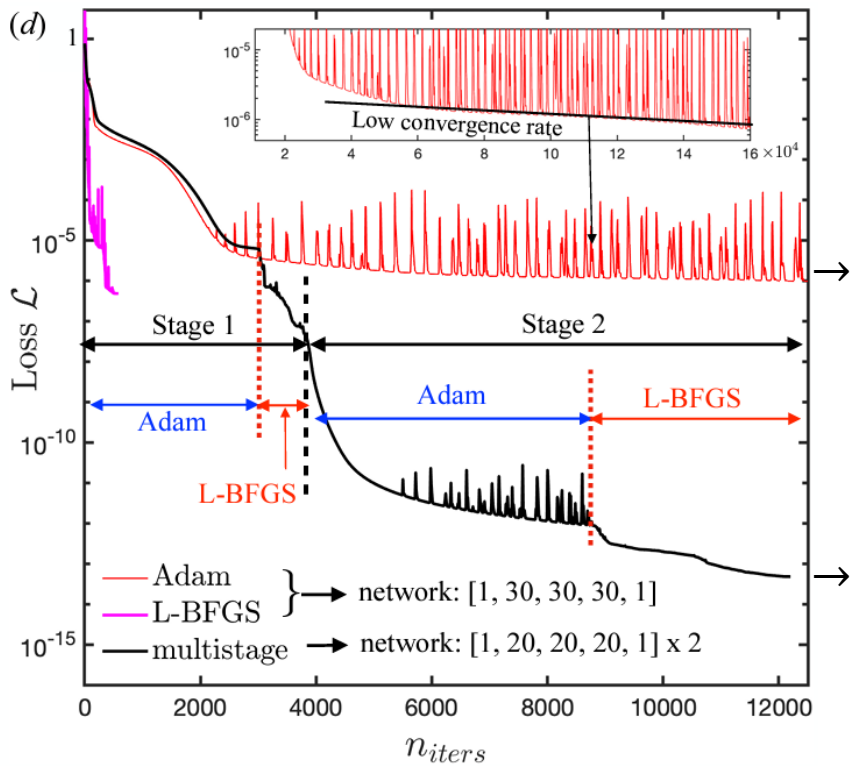


- NN's function approximation errors → Multistage neural network

NN as universal function approximators: Hornik et al (1990)

Wang & Lai, J. Comput. Phys. (2024)





→ Regular NN

→ Multistage NN

Spectral bias: A Fourier analysis of NN

Xu et al. (2022)

For a single-hidden-layer NN with input $x \in \mathbb{R}$ and output $u \in \mathbb{R}$

$$u(x) = \sum_{j=1}^{m} w_j^{(1)} \sigma(w_j^{(0)} x + b_j), \quad \text{where} \quad \theta_j = \{w_j^{(0)}, w_j^{(1)}, b_j\}$$

The mean square loss measures the distance between the NN output $u(x)$ and ground truth $u_g(x)$

$J \equiv \int_{-\infty}^{\infty} \frac{1}{2}(u(x) - u_g(x))^2 dx = \int_{-\infty}^{\infty} \frac{1}{2}|\hat{u}(k) - \hat{u}_g(k)|^2 dk$, where $\hat{u}(k)$ denotes the Fourier transform of $u(x)$

The loss at frequency k is $J(k) = \frac{1}{2}|\hat{u} - \hat{u}_g|^2$

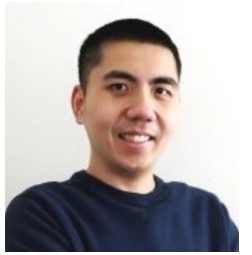
For tanh activation function, the gradient of loss at $J(k)$ with respect to NN parameters θ_j is

$$\left|\frac{\partial J(k)}{\partial \theta_j}\right| \approx A(k) e^{-\left|\frac{\pi k}{2w_j^{(0)}}\right|}$$ where $A(k) \in [0, +\infty)$ is the amplitude of $\hat{u} - \hat{u}_g$

This gradient decay rapidly with frequency k!! This is a simple explanation of the spectral bias.

Capturing high-frequency functions

Yongji Wang
Postdoc

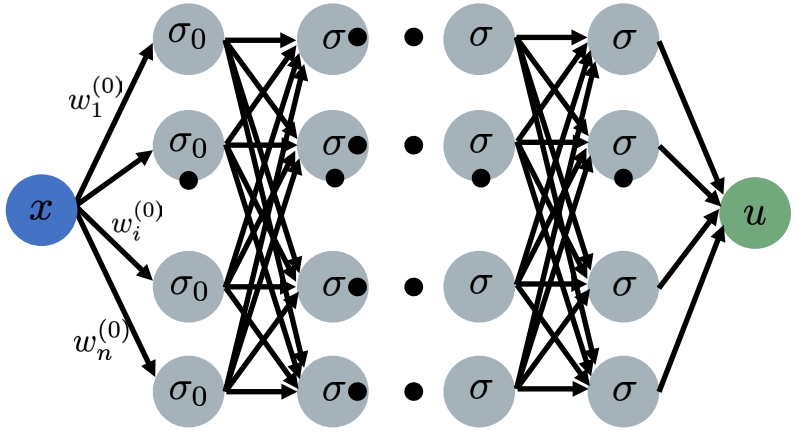


- For multiscale problems, it is important to rescale the NN weights so that it actives the learning of high-frequency (large k) functions

$$\left|\frac{\partial J(k)}{\partial \theta_j}\right| \approx A(k)e^{-\left|\frac{\pi k}{2w_j^{(0)}}\right|}$$

We make $w_j^{(0)} \propto k$

Our implementation: Wang & Lai, J. Comput. Phys. (2024)



First layer: $\sigma_0(x) = \sin\left(\kappa w_i^{(0)} x + b_i^{(0)}\right)$

κ: scaling factor

κ is user-defined hyperparameter

Main idea

Wang & Lai, J. Comput. Phys. (2024)

- When NN training error is small, the learning becomes really slow.
  MSNN includes a superposition of multi-stage NNs, with each stage using a new NN to fit the residue rescaled to O(1) from the previous NN.

$$u_c^{(n)}(x) = u_0(x) + \epsilon_1 u_1(x) + \epsilon_2 u_2(x) + \ldots + \epsilon_n u_n(x) = \sum_{j=0}^{n} \epsilon_j u_j(x)$$

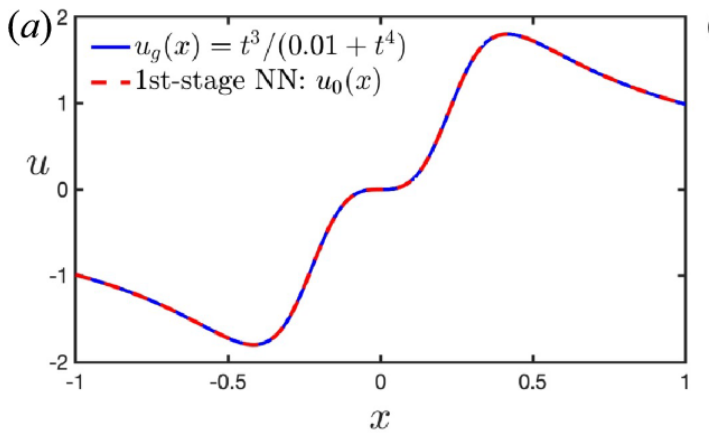
where $1 \gg \epsilon_1 \gg \epsilon_2 \gg \ldots \gg \epsilon_n$

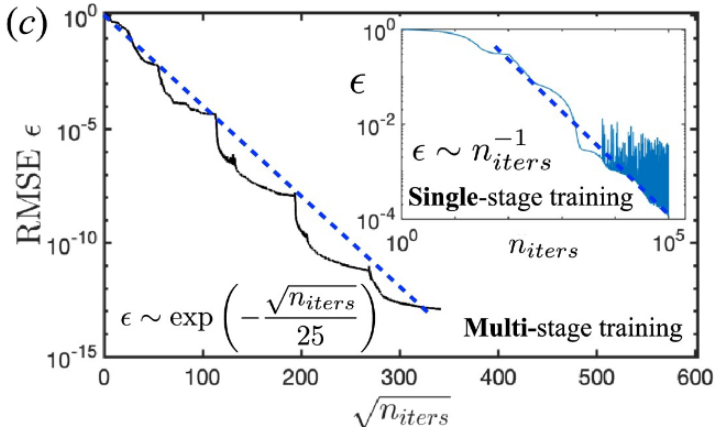
$u_0(x)$, $u_1(x)$ and $u_n(x)$ are all of order $\mathcal{O}(1)$ NNs of different stages

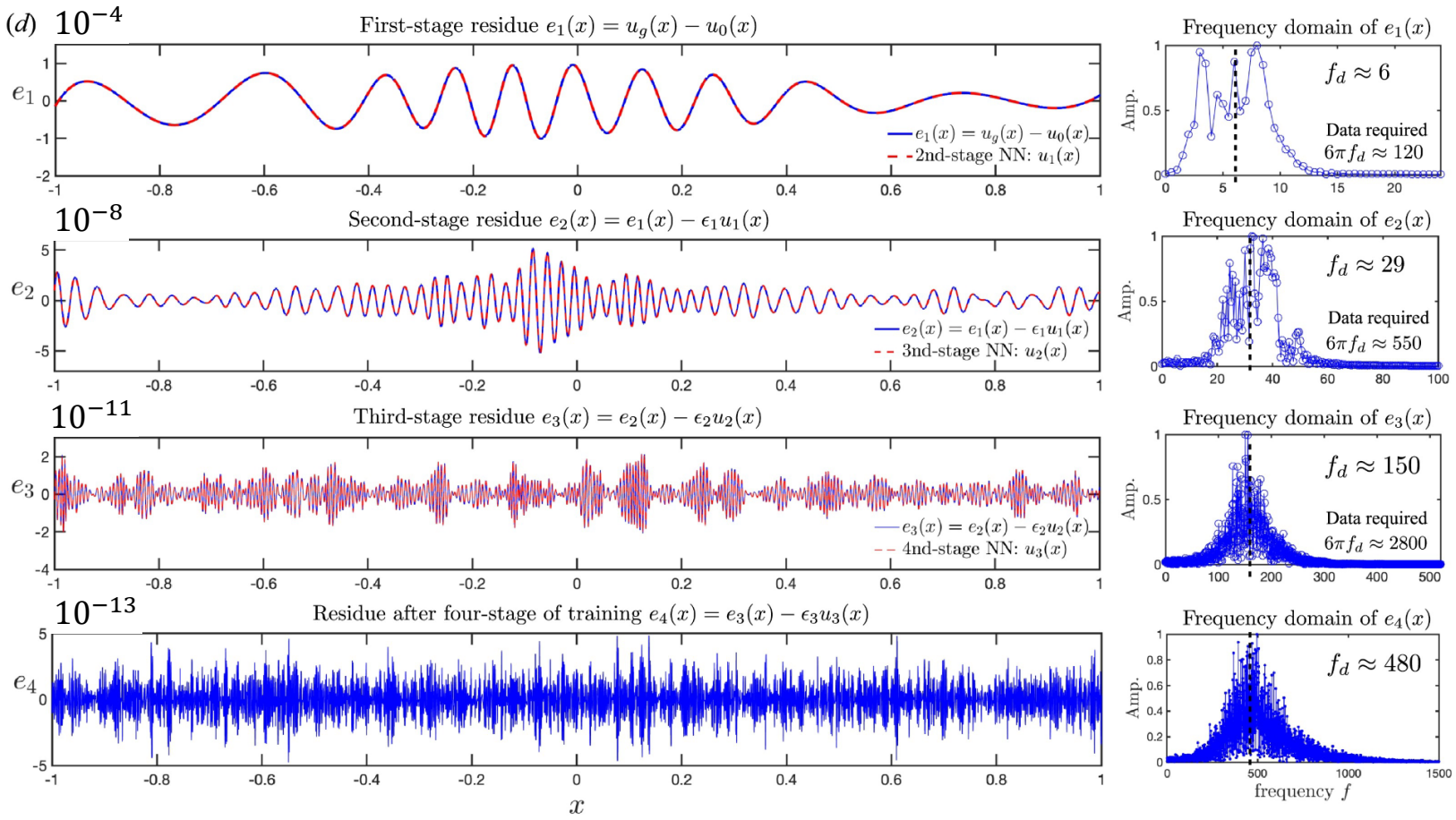
Related work with similar flavors: Sequential residual learning: Krishnanunni & Bui-Thanh arXiv:2211.06860, Multi-level neural networks: Aldirany et al. (2024), Stacked networks: Howard et al. arXiv:2311.06483, Precision Machine Learning: Michaud et al. arXiv:2210.13447…etc

Multistage-NN (MSNN)

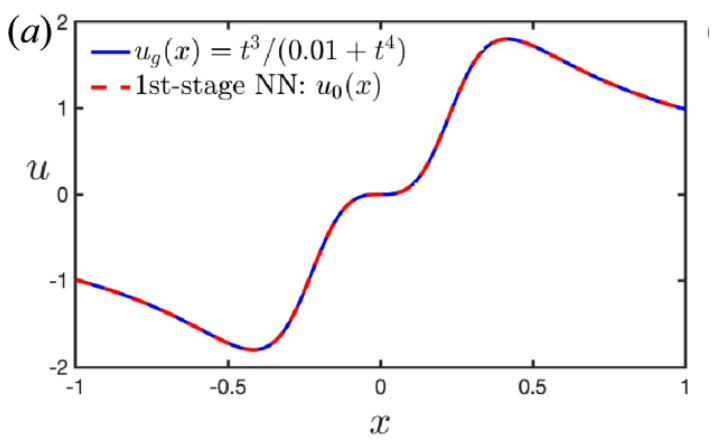
Wang & Lai, J. Comput. Phys. (2024)

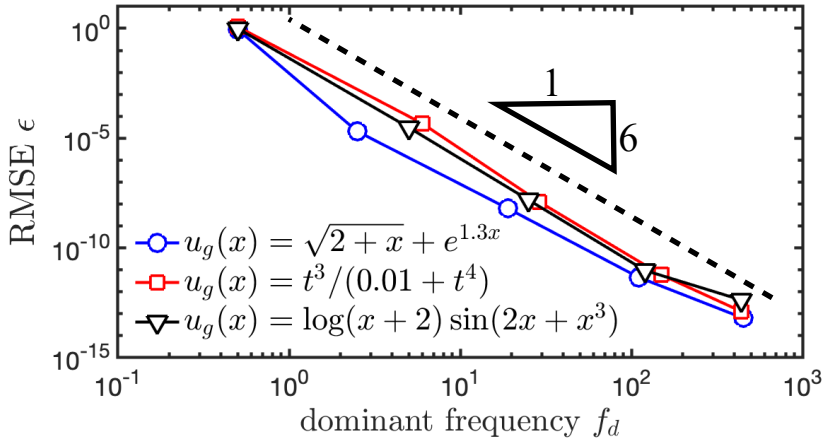


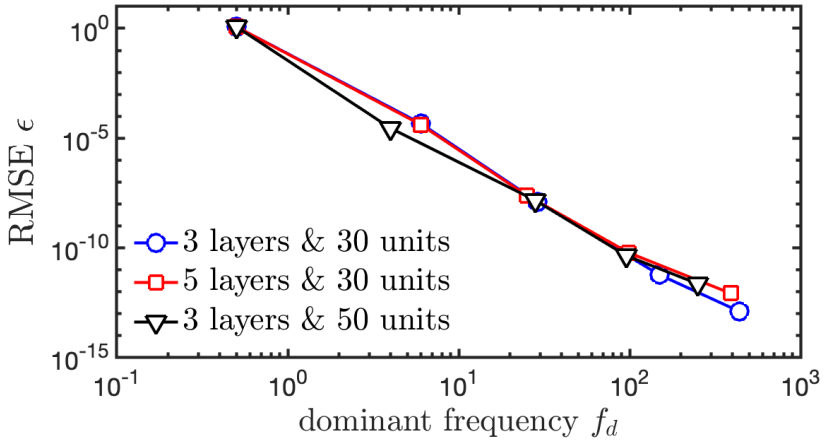




Scaling laws: Smaller errors, higher frequencies







Observed universal power-law:

$$f_d \propto \epsilon^{-1/\alpha} \text{ where } \alpha = 6$$

Error of $u_g(x)$ $\longrightarrow$ ϵ

Error of $\frac{d}{dx}u_g(x)$ $\longrightarrow$ $\epsilon(2\pi f_d)$

Error of $\frac{d^n}{dx}u_g(x)$ $\longrightarrow$ $\epsilon(2\pi f_d)^n \sim \epsilon^{1-n/\alpha}$

Can multistage-NN be used to tackle the spectral bias in fluids?

Jakin Ng
Student

Example:

How precisely can a NN approximate a 2D fluid flow?

2D incompressible Navier-Stokes eqn:

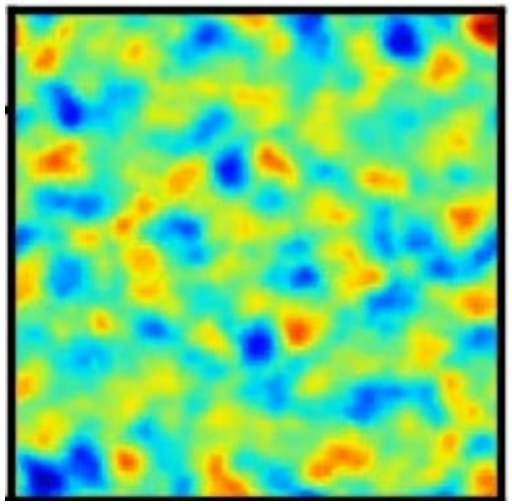
ω: vorticity
ψ: stream function

$$\frac{\partial \omega}{\partial t} + \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} = \frac{1}{\mathrm{Re}}\nabla^2\omega$$

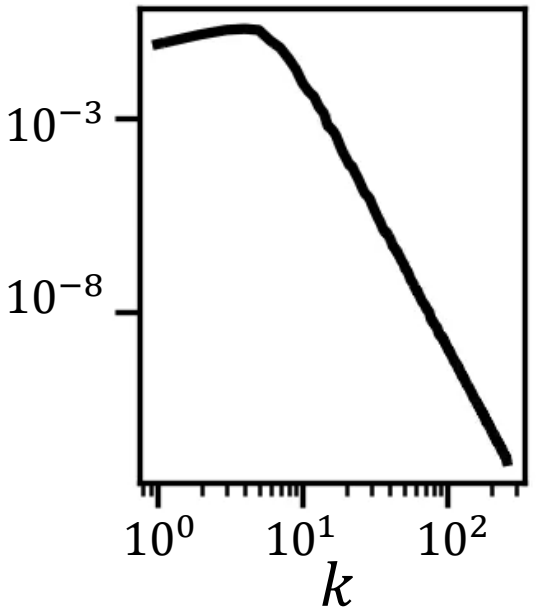
$$\nabla^2\psi = -\omega$$

Training data:
Numerical solution at $\mathrm{Re} = 2000$

$\omega(x, y, t)$



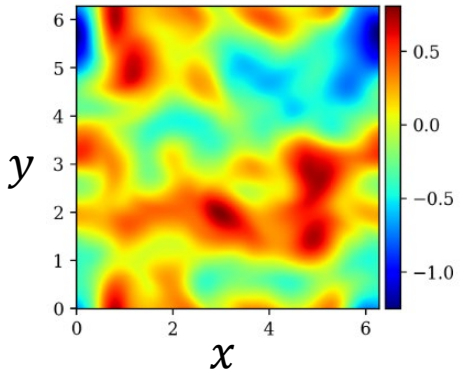
$E(k)$



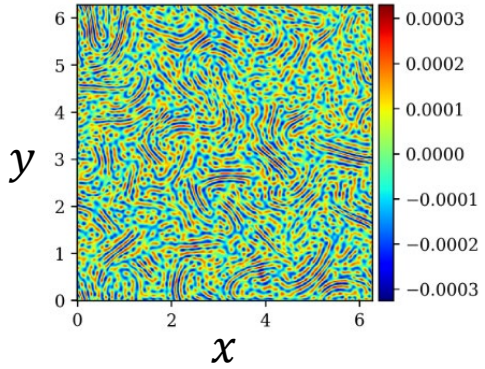
Energy spectrum: $E(k,t) = \sum_{k-\Delta k \le |\boldsymbol{k}| \le k+\Delta k} \frac{1}{2}k^2 |\tilde{\psi}(\boldsymbol{k},t)|^2$

E.g. 2D incompressible Navier-Stokes

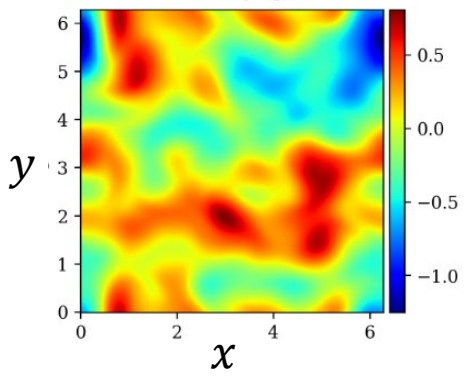
Target $\psi(x, y)$



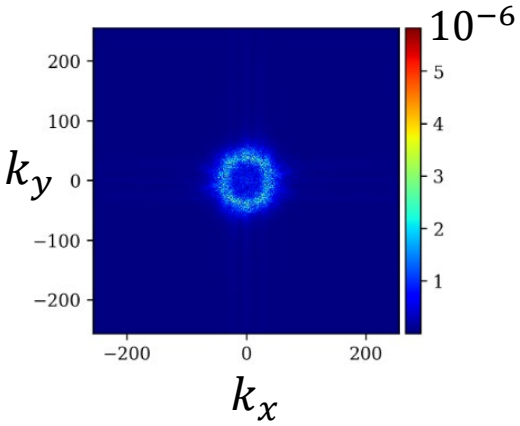
Error



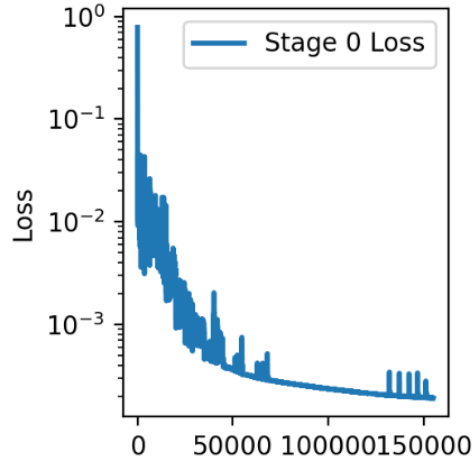
NN $\psi(x, y)$



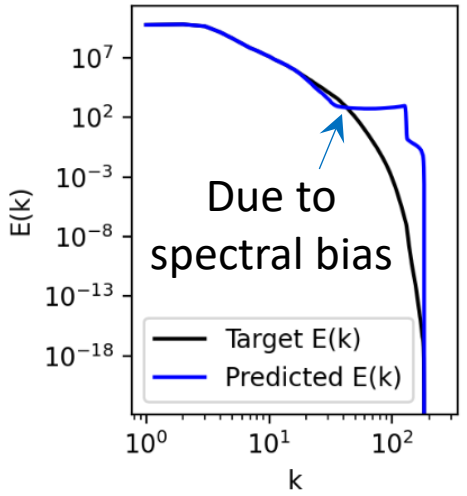
Error in Fourier domain



Loss convergence



Energy spectrum

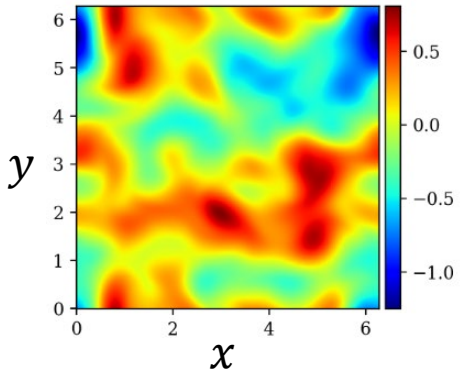


E.g. 2D incompressible Navier-Stokes

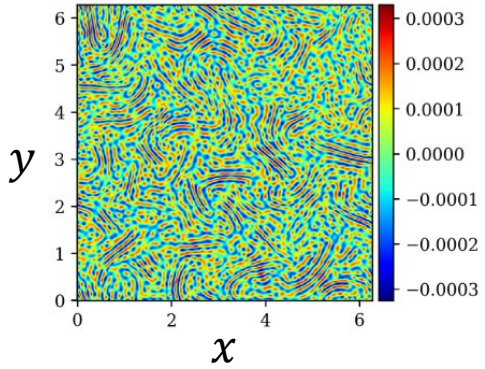
Stage 0

Stage 1

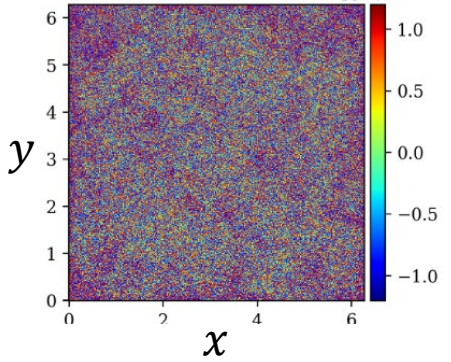
Target $\psi(x, y)$



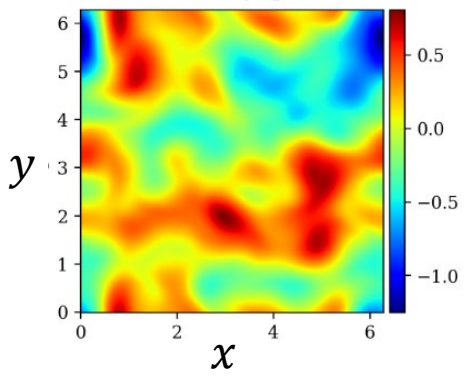
Error



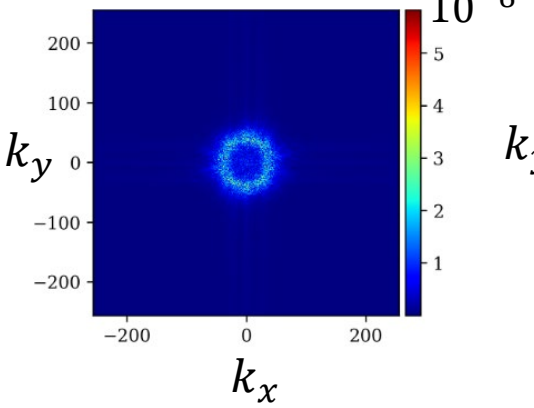
Error



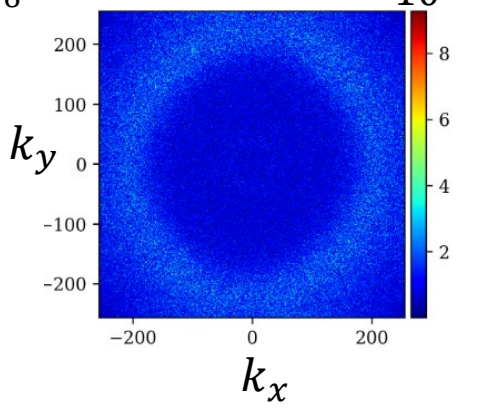
NN $\psi(x, y)$



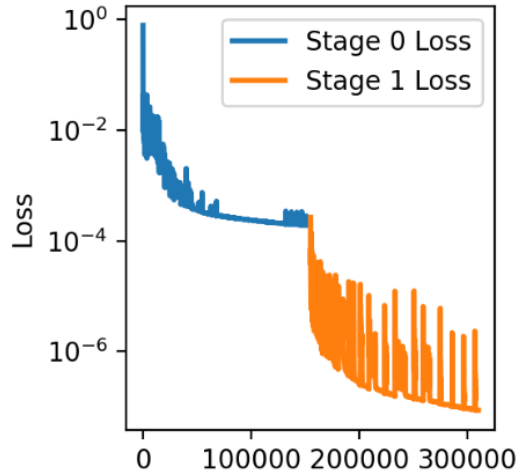
Error in Fourier domain



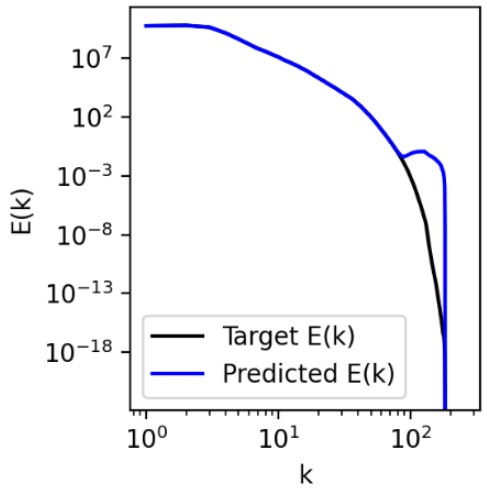
Error in Fourier domain



Loss convergence

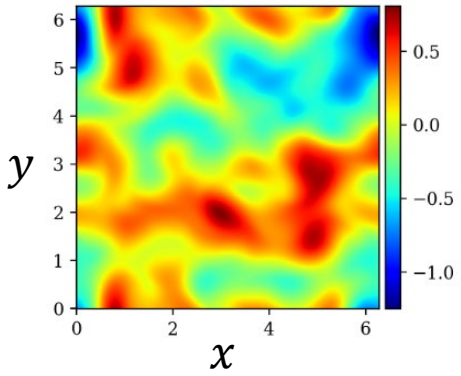


Energy spectrum



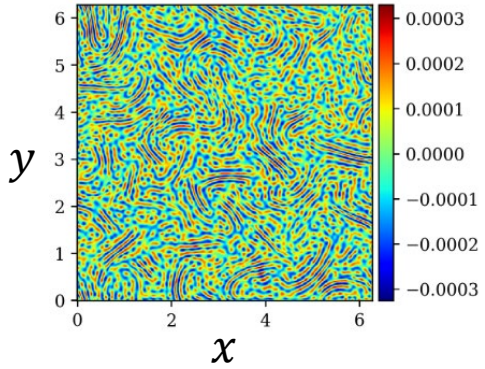
E.g. 2D incompressible Navier-Stokes

Target $\psi(x, y)$



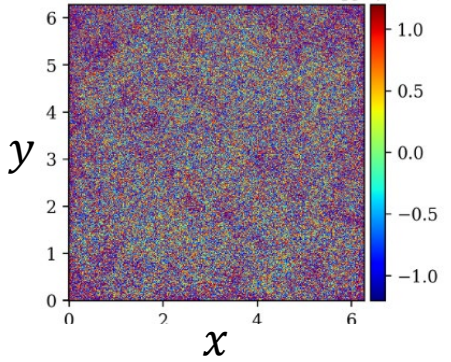
Stage 0

Error



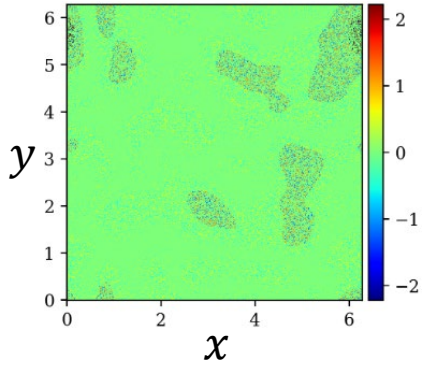
Stage 1

Error 10^{-7}

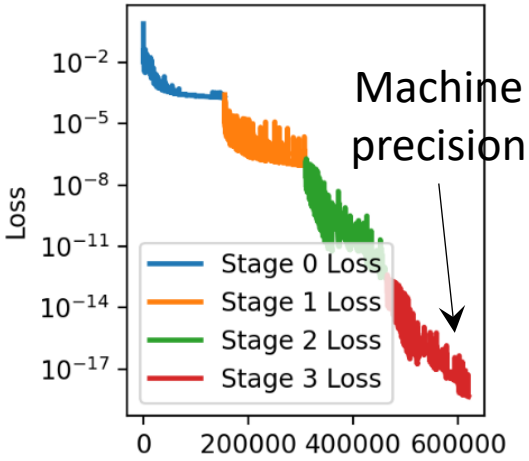


.... Stage 3

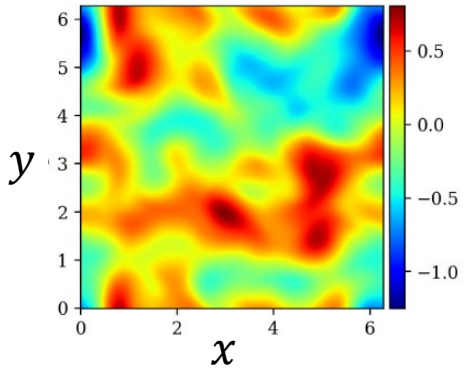
Error 10^{-16}



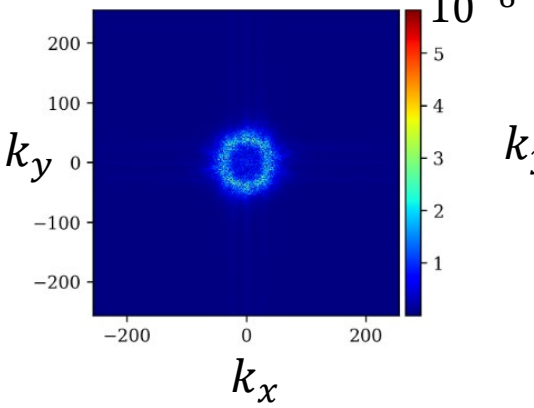
Loss convergence



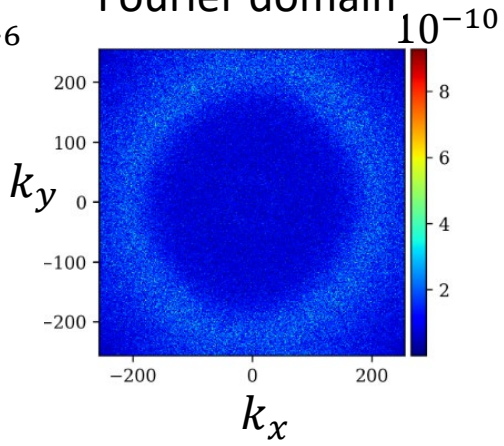
NN $\psi(x, y)$



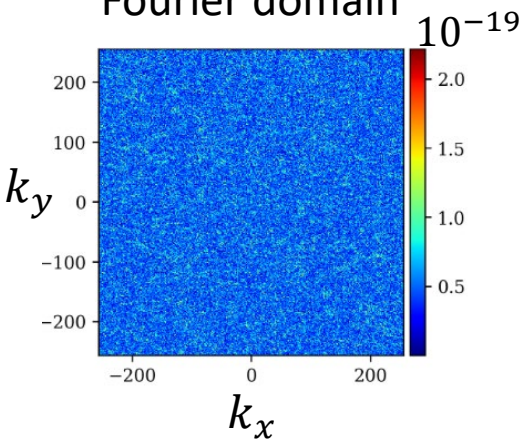
Error in Fourier domain



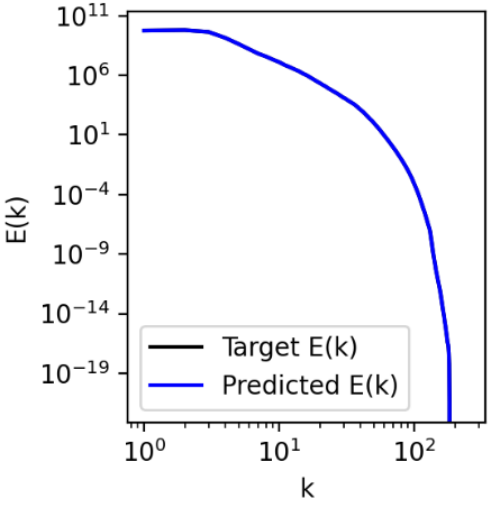
Error in Fourier domain



Error in Fourier domain



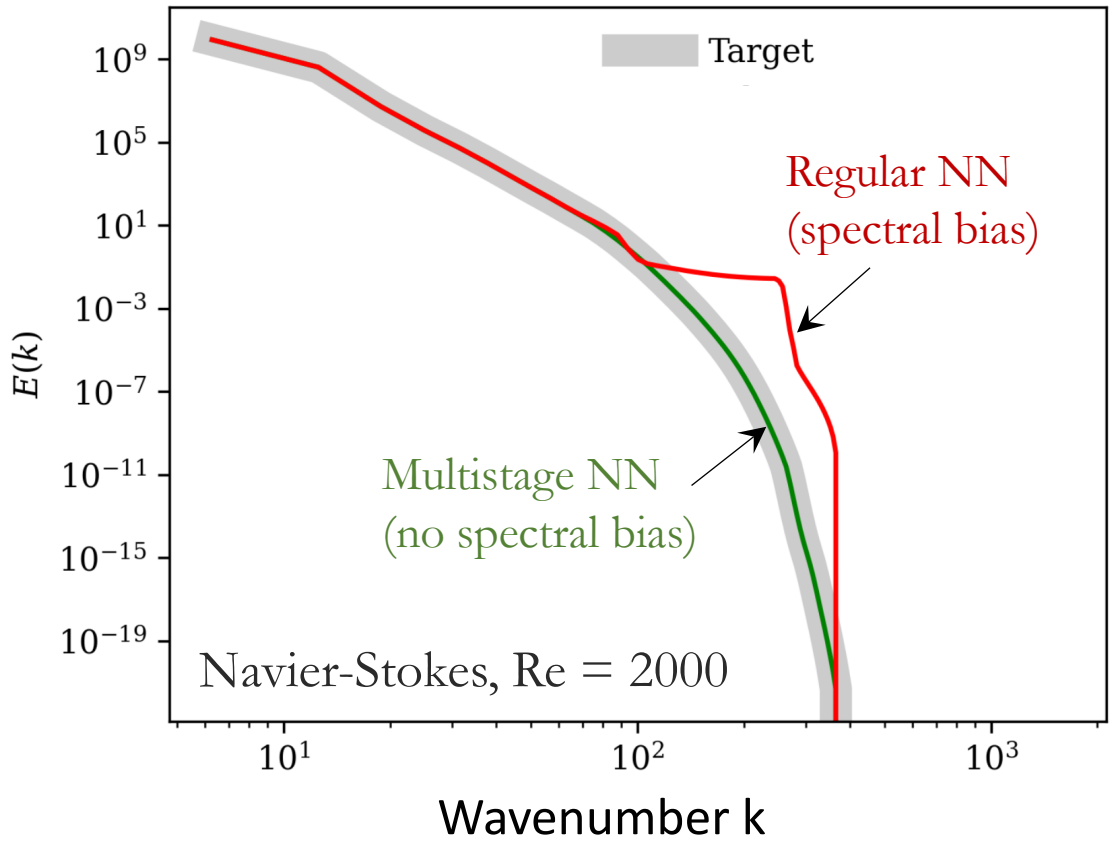
Energy spectrum



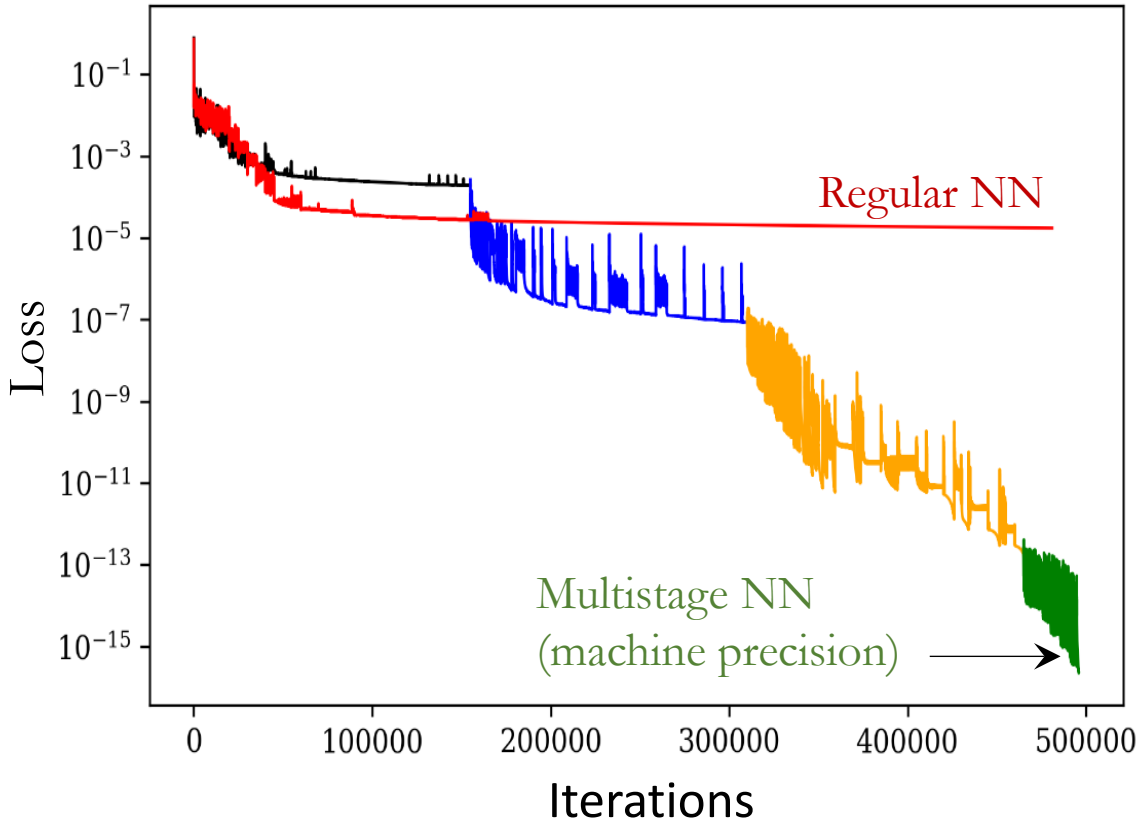
Multistage NN can tackle the spectral bias & accelerate loss convergence

Wang & Lai, J. Comput. Phys. (2024)
Ng, Wang & Lai, ICML workshop (2024), arXiv:2407.17213

Energy spectrum



Loss convergence



Multi-stage NN for regression problems

Regression

	Target function	Network prediction	Error	Amplitude	Frequency
First-stage training	$u_g(x)$	$u_0(x)$	$e_1(x) = u_g - u_0$	ϵ_1	f_{d_1}
Second-stage training	$e_1(x)/\epsilon_1$	$u_1(x)$ (with $\kappa = 2\pi f_{d_1}$)	$e_2(x) = e_1(x) - \epsilon u_1(x)$	ϵ_2	f_{d_2}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
nth-stage training	$e_{n-1}(x)/\epsilon_{n-1}$	$u_{n-1}(x)$ (with $\kappa = 2\pi f_{d_{n-1}}$)	$e_n(x) = e_1(x) - \sum_i^{n-1} \epsilon_i u_i(x)$	ϵ_n	f_{d_n}

Final expression $$u_c^{(n)}(x) = u_0(x) + \epsilon_1 u_1(x) + \epsilon_2 u_2(x) + \ldots + \epsilon_n u_n(x) = \sum_{j=0}^{n} \epsilon_j u_j(x)$$

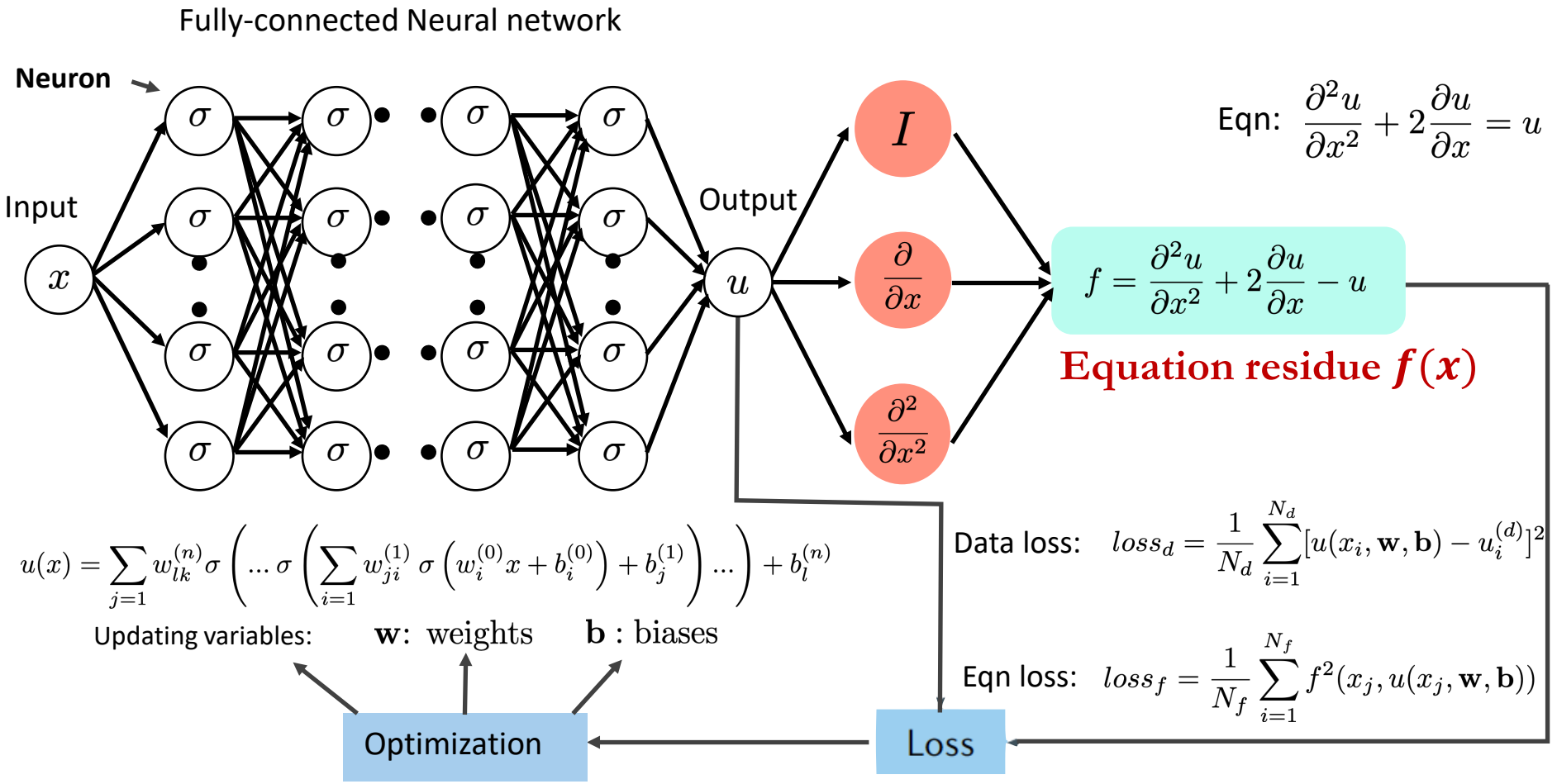
where $$\epsilon_0 = 1 \gg \epsilon_1 \gg \epsilon_2 \gg \ldots \gg \epsilon_n$$

We have shown that MSNN are good function approximators, can it be used to find PDE solutions numerically?

Wang & Lai, J. Comput. Phys. (2024)

Physics-informed neural networks

Karniadakis *et. al.* (2021), *Nat. Rev. Phys.*



Gradient descent

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla_{\mathbf{w}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$$\mathbf{b}^{(i+1)} = \mathbf{b}^{(i)} - \eta \nabla_{\mathbf{b}} J(x, \mathbf{w}^{(i)}, \mathbf{b}^{(i)})$$

$\mathbf{w}^{(i)}, \mathbf{b}^{(i)}$: value at the i-th iteration η : learning rate

Cost function: *data + equation loss*

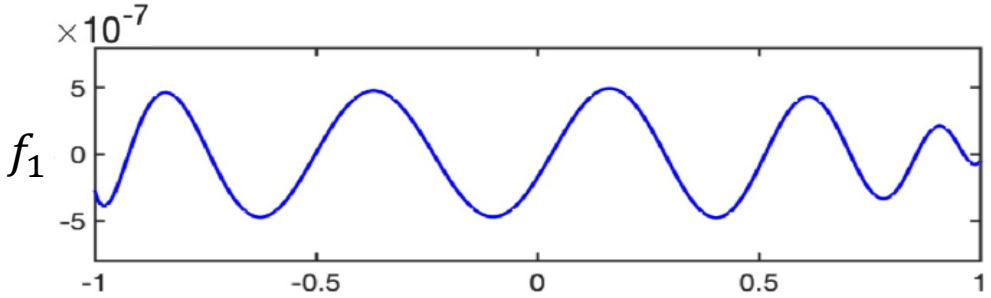
$$J(x, \mathbf{w}, \mathbf{b}) = loss_d + loss_f$$

Multistage-PINN for simple ODE

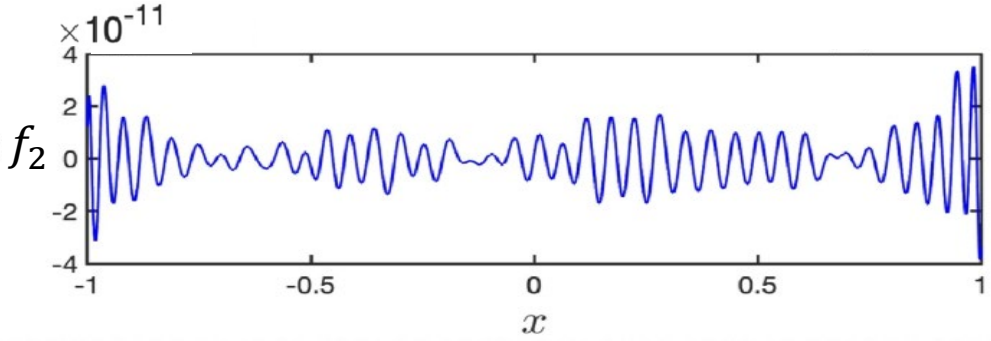
Equation residue: $f(u, x)$

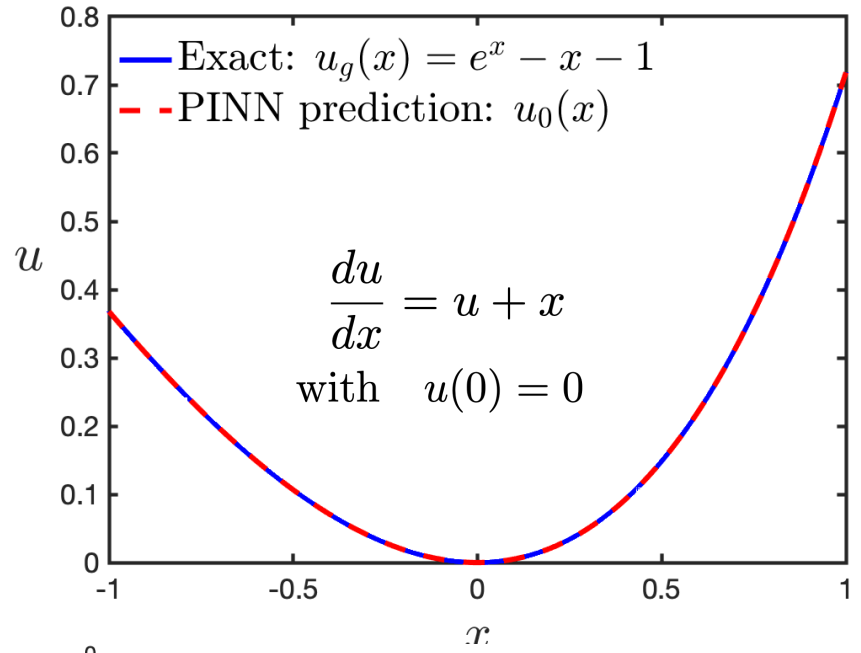
Ansatz: $u(x) = u_0(x) + \epsilon_1 u_1(x)$

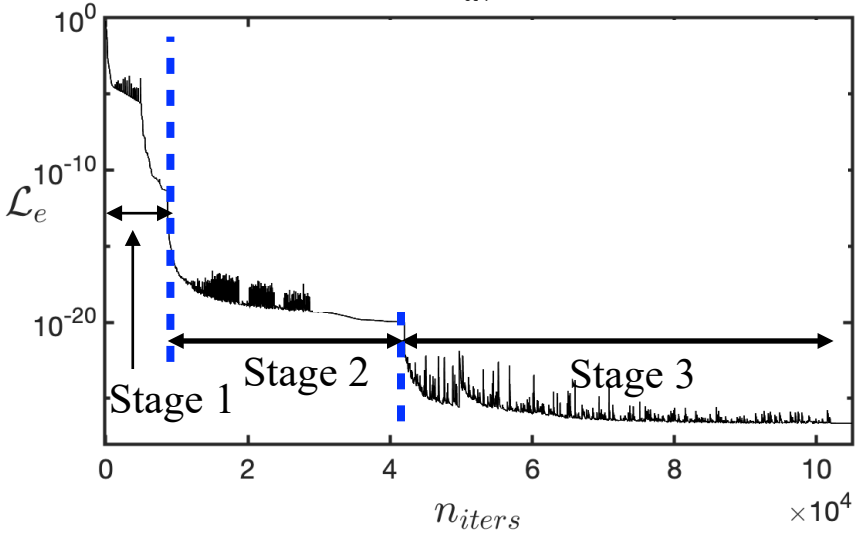
Equation residue of 1st stage NN u_0: $f_1 \equiv f(u_0, x)$



Equation residue of 2nd stage NN u_1: $f_2 \equiv f(u_0 + \epsilon_1 u_1, x)$







Outline

- High-precision NN algorithm for tackling the spectral bias
- Applications to finding PDE solutions numerically
- Applications of finding singularities in fluids

With Tristan Buckmaster, Gonzalo Cao-Labora, Javier Gomez-Serran, Yongji Wang, and Google DeepMind

A Millennium Prize problem:

Does the Navier-Stokes equation always admit a unique and smooth (infinitely differentiable) solution at any times?

One pathway: If one can find a solution to the Navier Stokes that "blows up" at finite time, that would be a counter-example.

Can highly-precise NN help find the blow-up solution?

For the Euler equations:

Wang, Lai, Gómez-Serrano, Buckmaster, *Physical Review Letters* (2023), Wang et al. arXiv:2509.14185

Benchmark example: Burgers equation

Burgers' equation

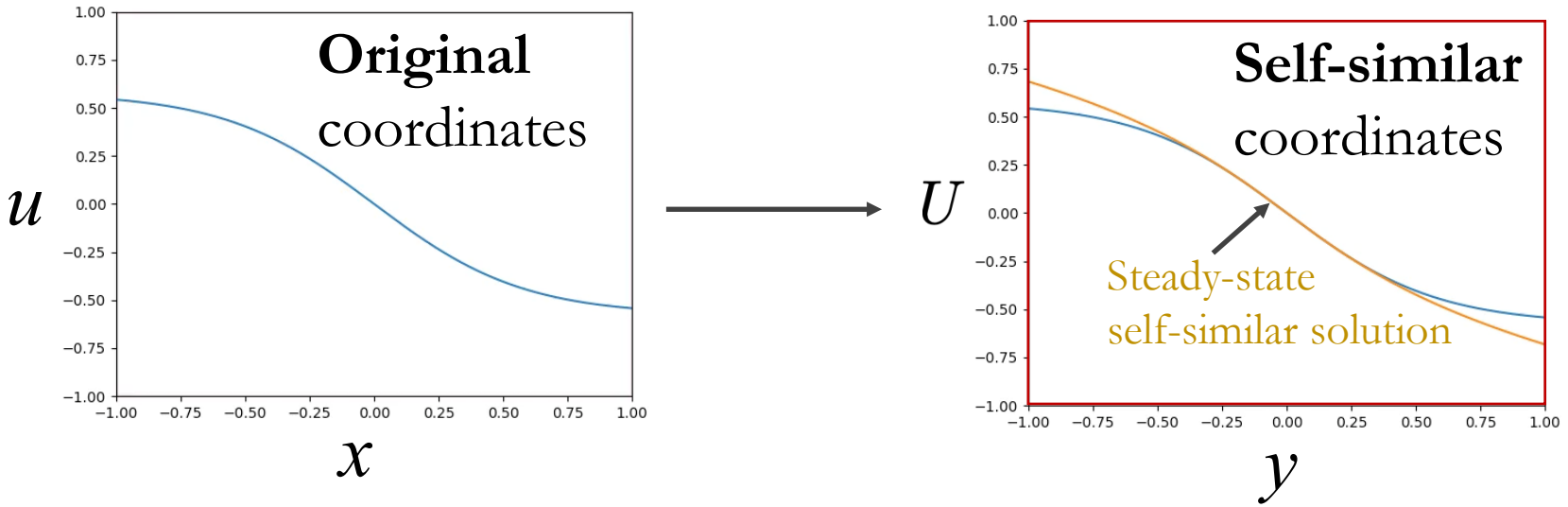
$$u_t + uu_x = 0$$

Shock wave

Assuming the self-similar ansatz

$$u = (1-t)^{\lambda} U\left(\frac{x}{(1-t)^{1+\lambda}}\right)$$ is the solution that blows up at $t = 1$.

$$y = \frac{x}{(1-t)^{1+\lambda}}$$ is the self-similar variable.



Benchmark example: Burgers equation

Burgers' equation

$$u_t + uu_x = 0$$

Shock wave

Assuming the self-similar ansatz

$$u = (1-t)^{\lambda} U\left(\frac{x}{(1-t)^{1+\lambda}}\right)$$ is the solution that blows up at $t = 1$.

$$y = \frac{x}{(1-t)^{1+\lambda}}$$ is the self-similar variable.

we obtain the self-similar Burgers' equation

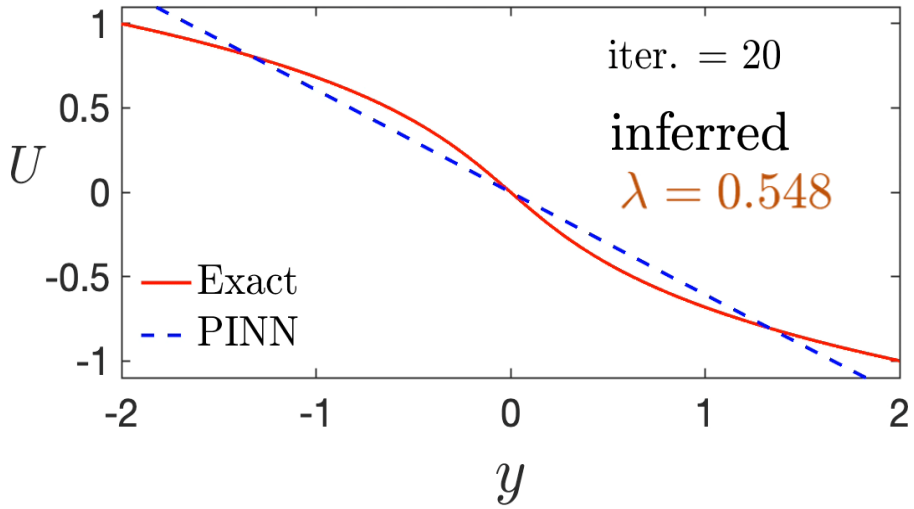
$$-\lambda U + ((1+\lambda)y + U)\partial_y U = 0$$

The goal for NN:
1. Find the solution $U(y)$
2. Find the self-similar exponent λ

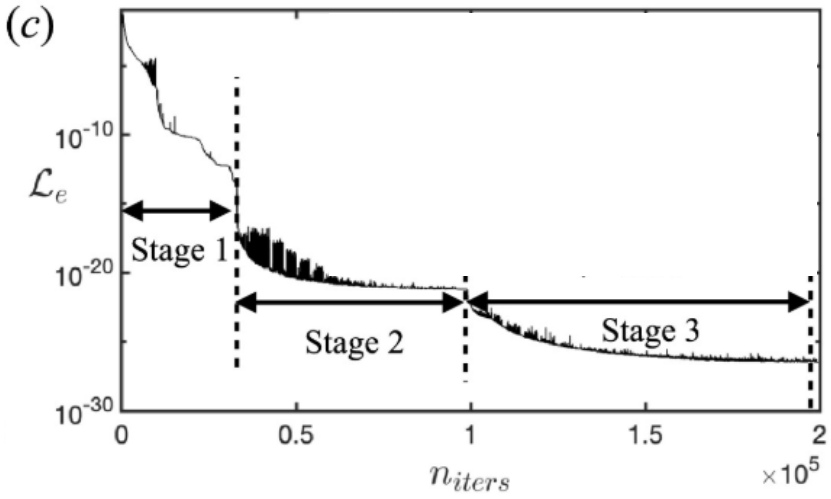
This is an inverse problem!

Multistage NN for Burgers

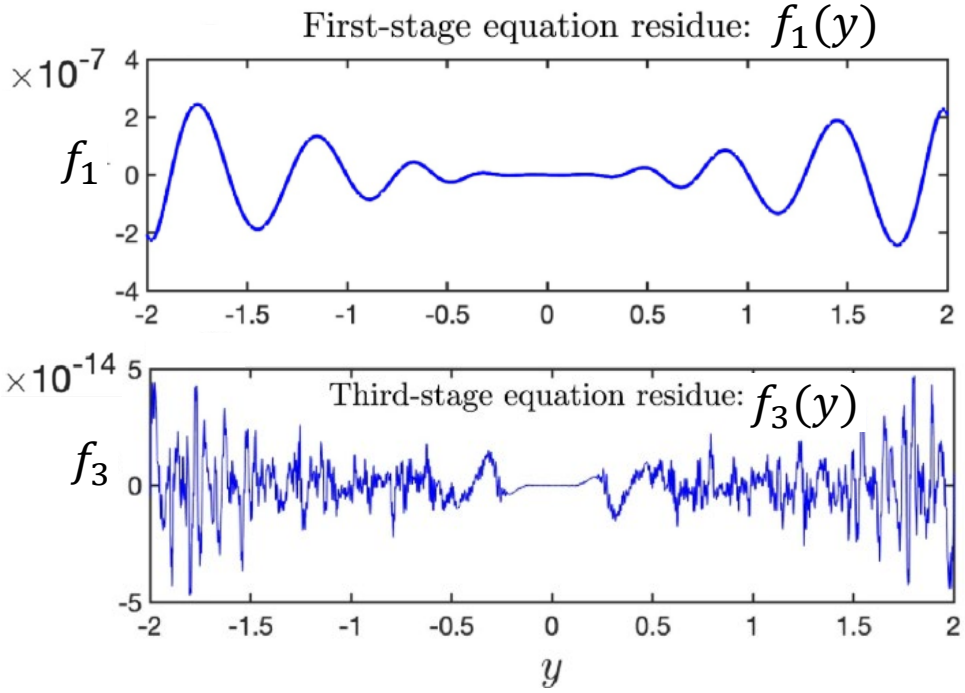
Solution profile:



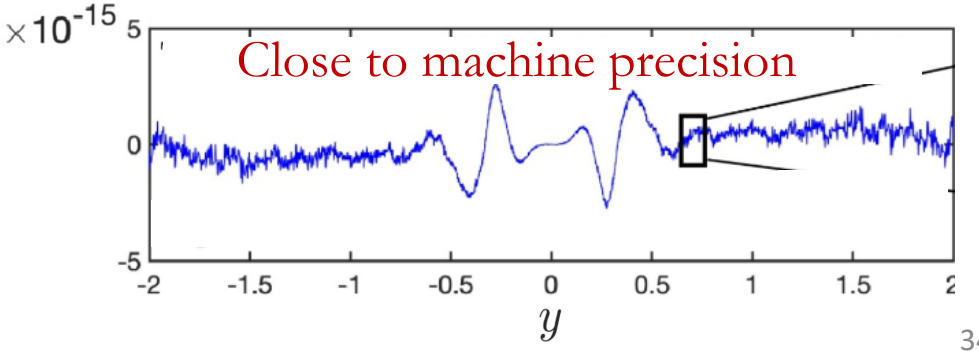
Loss function:



Equation residues:



Solution error:



Incompressible Euler equations

The pair $(\mathbf{u}, p)$ solves the incompressible 3-D Euler equations if

$$\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \cancel{\mu\Delta\mathbf{u}}^{\,0}, \quad \operatorname{div}(\mathbf{u}) = 0, \quad \text{and} \quad \mathbf{u}(\cdot, t) = \mathbf{u_0}$$

for velocity $\mathbf{u}$, pressure p and initial velocity $\mathbf{u_0}$.

Major question:

Does there exist smooth, finite energy initial condition $\mathbf{u_0}$ leading to a solution **blowing up** in finite time?

DNS: Luo and Hou, PNAS (2014), Luo and Hou, MMS (2014)
Proof: Chen and Hou, arXiv:2210.07191 (2022), MMS (2025)

Can we leverage neural networks' expressiveness to find highly accurate numerical blow-up solution?

Incompressible Euler equations

The pair $(\mathbf{u}, p)$ solves the incompressible 3-D Euler equations if

$$\partial_t \mathbf{u} + (\mathbf{u}\cdot\nabla)\mathbf{u} + \nabla p = \underbrace{\mu\Delta}_{0}\mathbf{u}, \quad \operatorname{div}(\mathbf{u}) = 0, \quad \text{and} \quad \mathbf{u}(\cdot, t) = \mathbf{u_0}$$

for velocity $\mathbf{u}$, pressure p and initial velocity $\mathbf{u_0}$.

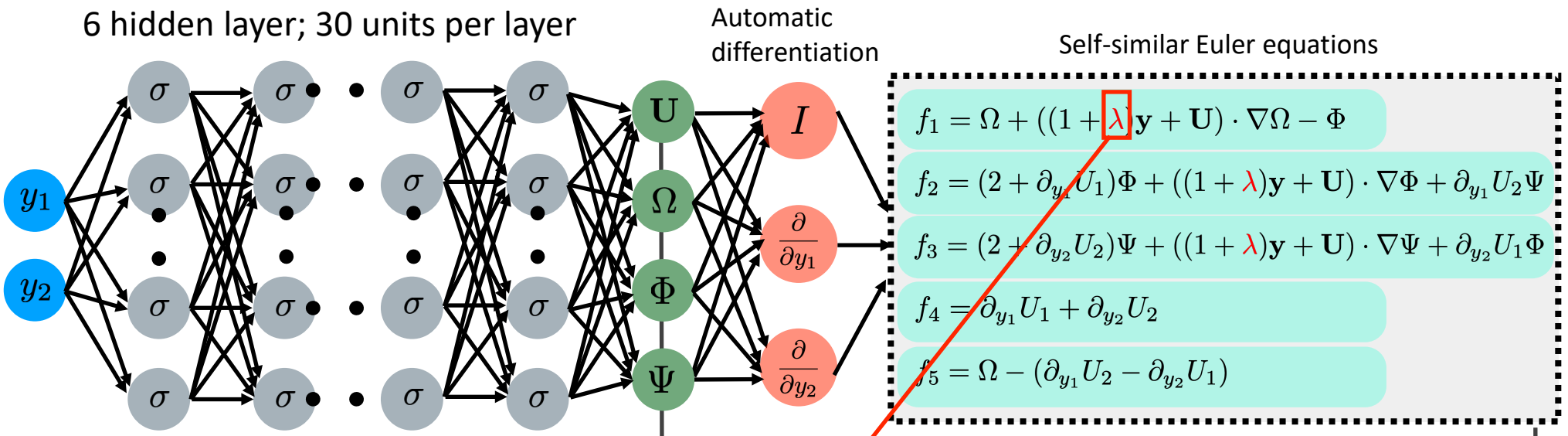
Steady self-similar equations for axisymmetric Euler with boundary

$$\begin{aligned}
\Omega + ((1+\lambda)\mathbf{y} + \mathbf{U})\cdot\nabla\Omega &= \Phi \\
(2 + \partial_{y_1}U_1)\Phi + ((1+\lambda)\mathbf{y} + \mathbf{U})\cdot\nabla\Phi &= -\partial_{y_1}U_2\Psi \\
(2 + \partial_{y_2}U_2)\Psi + ((1+\lambda)\mathbf{y} + \mathbf{U})\cdot\nabla\Psi &= -\partial_{y_2}U_1\Phi \\
\Omega = \partial_{y_1}U_2 - \partial_{y_2}U_1 \qquad & \operatorname{div}\mathbf{U} = 0
\end{aligned}$$

The goal for NN:
1. Find the solutions
$\boldsymbol{U}(y_1, y_2), \Omega(y_1, y_2), \Psi(y_1, y_2), \Phi(y_1, y_2)$
2. Find the self-similar exponent λ

PINN (self-similar coordinate)

PINN: Raissi *et. al.* (2019), *J Comp.Phys*
Wang, Lai, Gómez-Serrano, Buckmaster,
Physical Review Letters (2023)



$$q(\mathbf{y}) = \sum_{j=1} w_{lk}^{(n)} \sigma\left(\sum w_{kj}^{(n-1)}\left(\ldots\left(\sum_{i=1} w_{ji}^{(1)} y_i + b_j^{(1)}\right)\ldots\right) + b_k^{(n-1)}\right) + b_l^{(n)}$$

Updating variables: $\mathbf{w}$: weights, $\mathbf{b}$: biases + λ: unknown in eqns

BC loss:

$$loss_c^{(j)} = \frac{1}{N_c^{(j)}} \sum_{i=1}^{N_c^{(j)}} [q^{(j)}(\mathbf{y}_i, \mathbf{w}, \mathbf{b}) - q_i^{(j)}]^2 \quad (j = 1, 2, \ldots, 5)$$

Eqn loss: $loss_f^{(k)} = \frac{1}{N_f^{(k)}} \sum_{i=1}^{N_f^{(k)}} f_k^2(\mathbf{y}_i, q(\mathbf{y}_i, \mathbf{w}, \mathbf{b}))$ $(k = 1, 2, \ldots, 5)$

Optimization

Loss

$q^{(j)}$: j-th output variable

$q_i^{(j)}$: data of $q^{(j)}$ at $\mathbf{y} = \mathbf{y}_i$

Adding structures to the NN-PDE solver

1. Symmetry constraints

U_1, Φ, Ω are odd in y_1

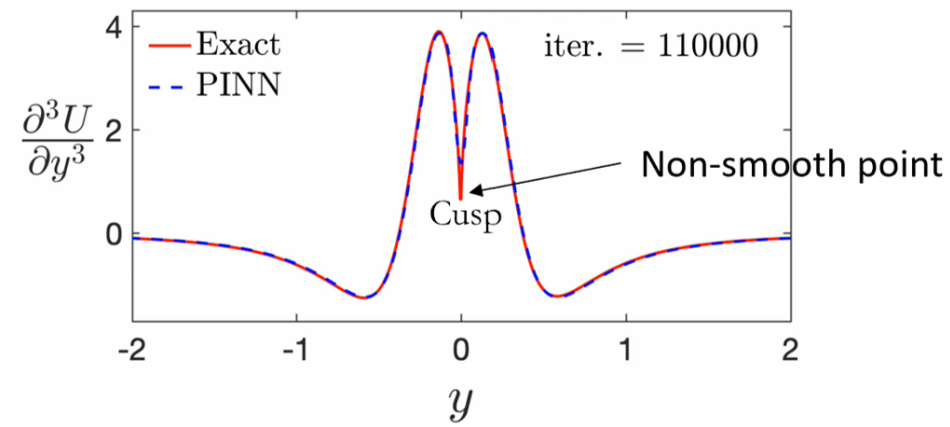
U_2, Ψ are even in y_1

e.g., to impose odd symmetry

$$U = [\mathrm{NN}_u(y) - \mathrm{NN}_u(-y)]/2$$

2. Smoothness constraint

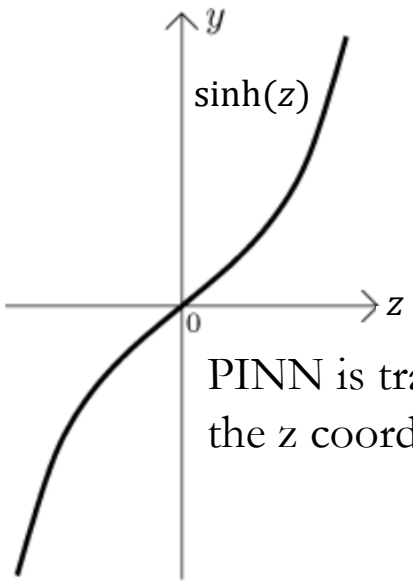
e.g., $loss_s = [\partial_{yyy} f(y)]^2 \to 0$



3. Map infinite to finite domain size

Change of coordinate:

e.g. Domain size $\sinh(30) \approx 5 \times 10^{12}$

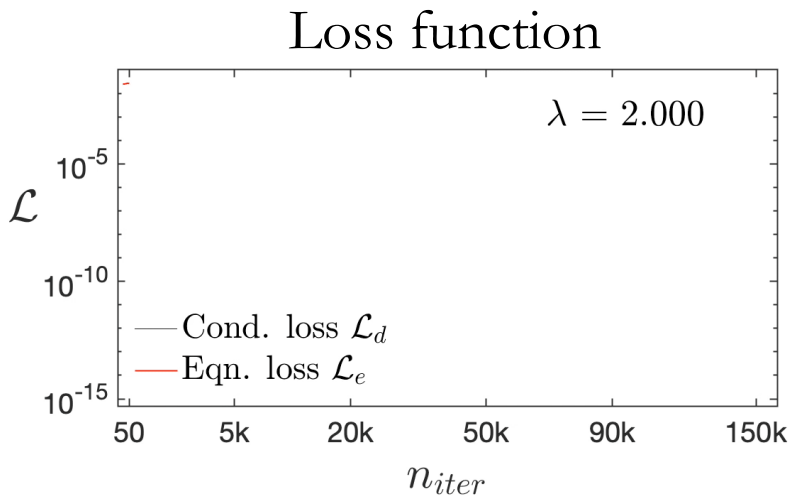


PINN is trained in the z coordinate

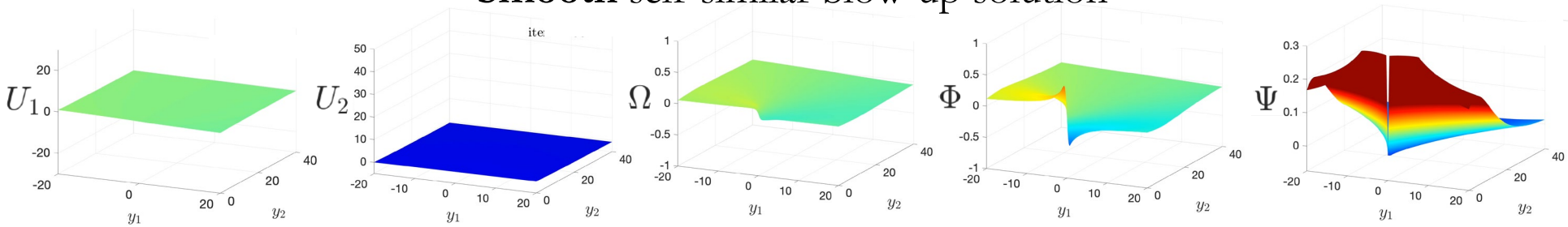
Find self-similar singularities to the Euler equation

Inferred $\lambda = 1.917$

Wang, Lai, Gómez-Serrano, Buckmaster, *PRL* (2023)

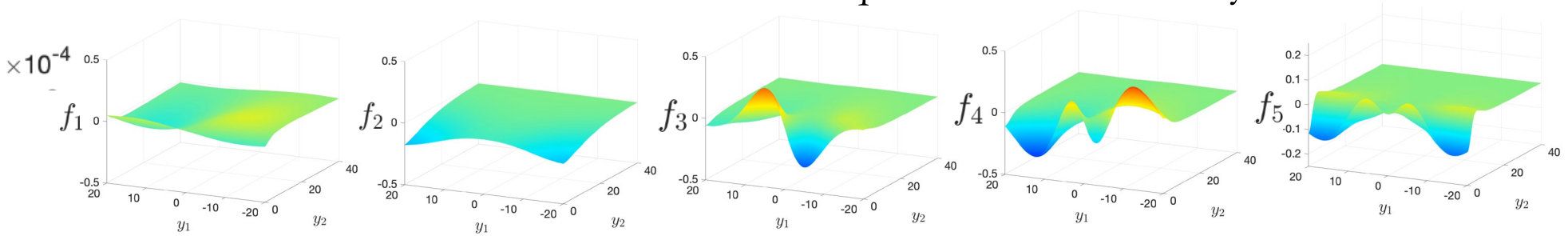


Smooth self-similar blow-up solution



iter. $= 50$

Uniform and small equation residues everywhere



Multistage reduces the PDE residuals to $O(10^{-13})$

Inferred $\lambda = 1.920560013482733$

With multistage NN, the error is further reduced.

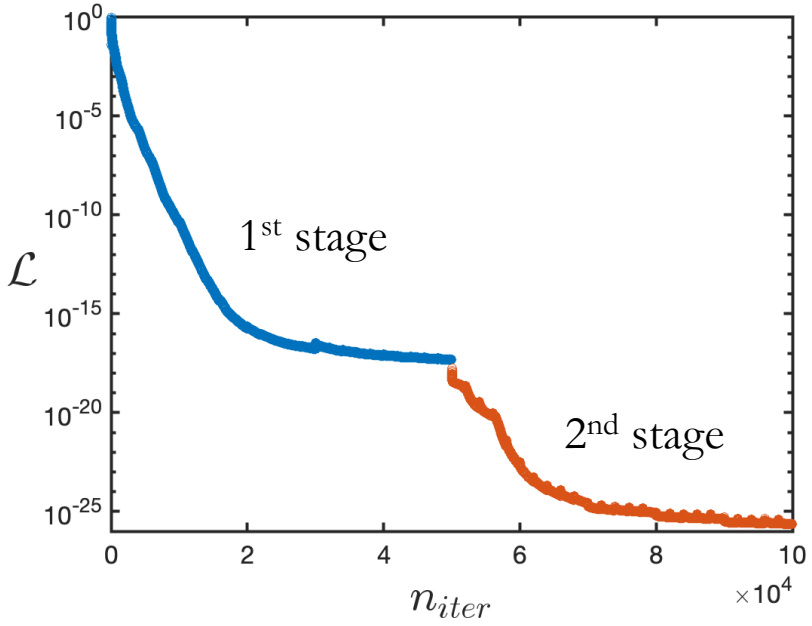
Wang et al., arXiv:2509.14185

Chen and Hou: $\lambda = 1.9205600$

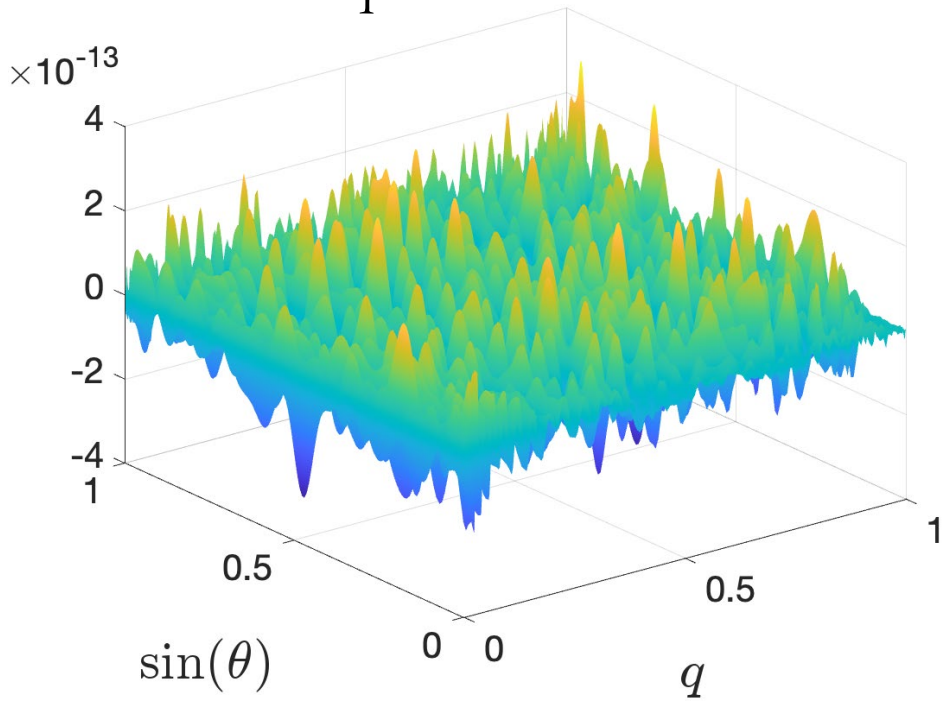
PDE residue of $O(10^{-8})$ is required for a computer-assisted proof:

Chen and Hou, MMS (2025), arXiv:2210.07191

Loss function

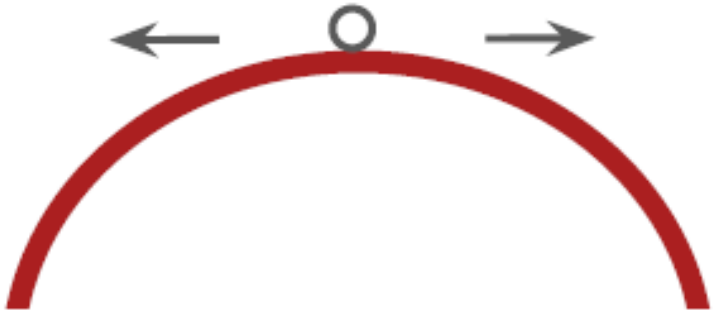


Equation residue

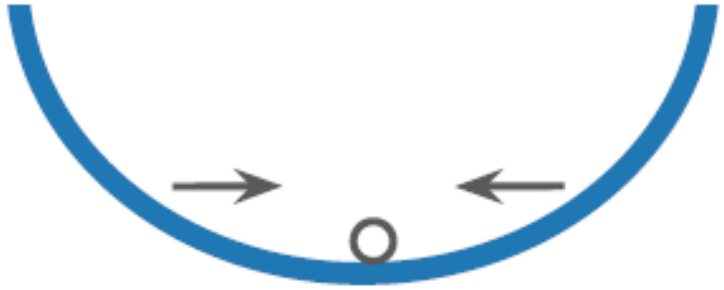


Stable v.s. unstable singularities

Stability analysis of the linearized PDE around smooth singularity solutions



unstable direction in solution space: perturbations diverge from singularity solution



stable direction in solution space: singularity solution remains stable to perturbations

To the best of our knowledge there hasn't been any unstable smooth blowup solutions to incompressible fluid equations reported in the literature

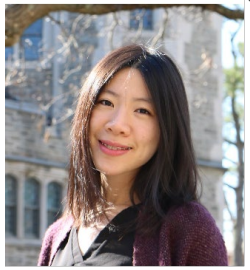
Yongji Wang
Postdoc

Tristan Buckmaster
New York University

Javier Gomez-Serran
Brown University

Gonzalo Cao-Labora
New York University

Ching-Yao Lai
Stanford University



Bogdan Georgiev
Google DeepMind

Ray Jiang
Google DeepMind

+ 17 Collaborators from
Google DeepMind

DeepMind

Mehdi Bennani James Martens Sebastien Racaniere Sam Blackwe Alex Matthews

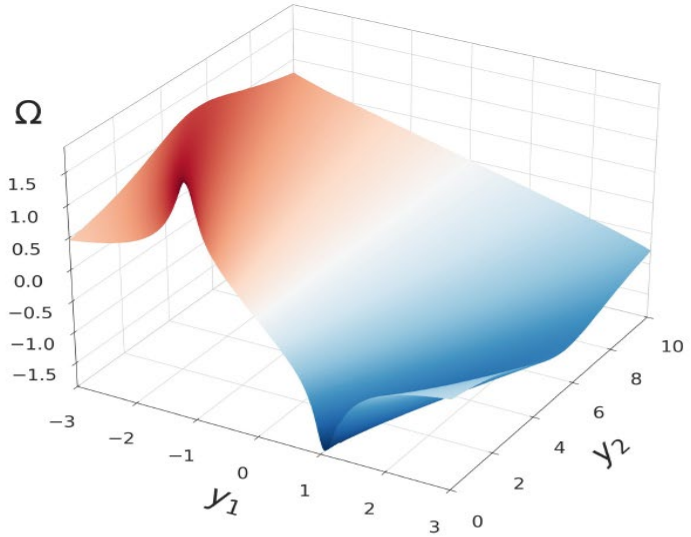
Stanislav Nikolov Daniel S. Park Martin Arjovsky Daniel Worrall Chongli Qin

Ferran Alet Borislav Kozlovskii Nenad Tomasev Alex Davies Pushmeet Kohli

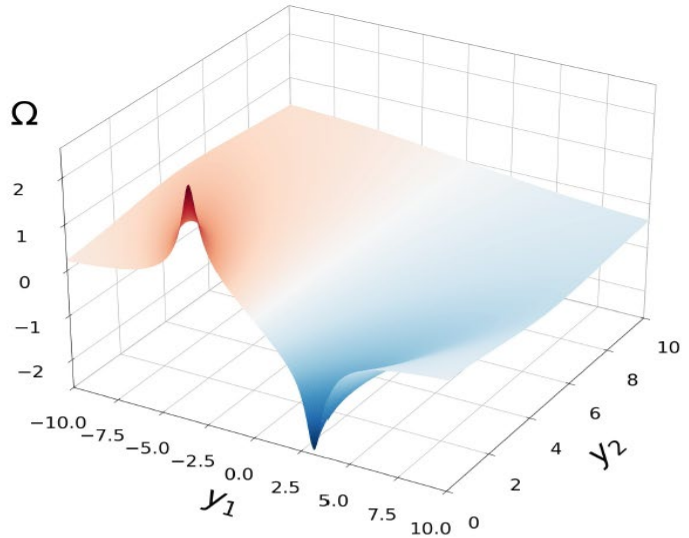
Discovery of a zoom of new unstable singularities!

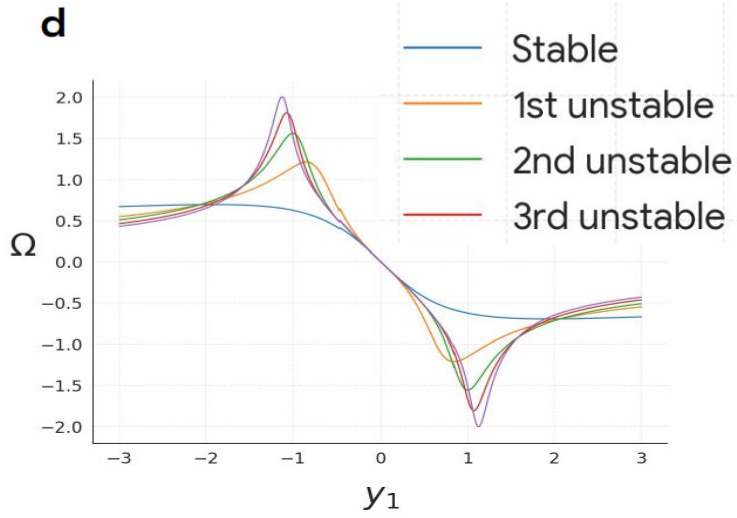
Wang et al., arXiv:2509.14185

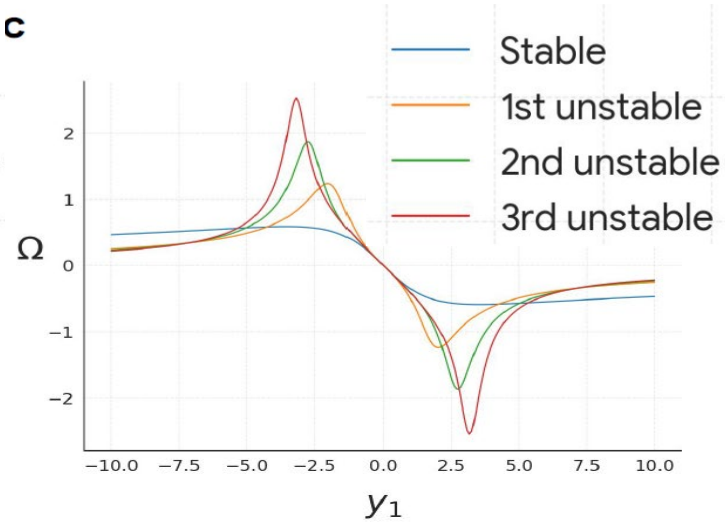
Euler: 3rd unstable solution



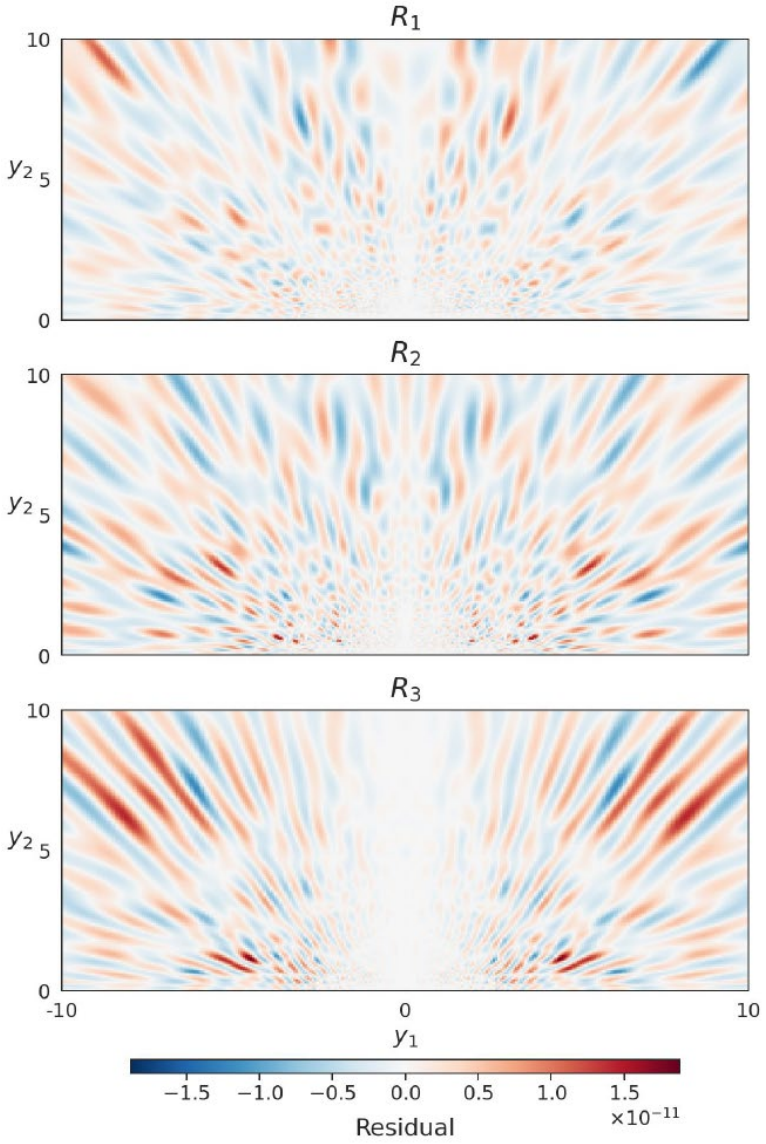
IPM: 3rd unstable solution



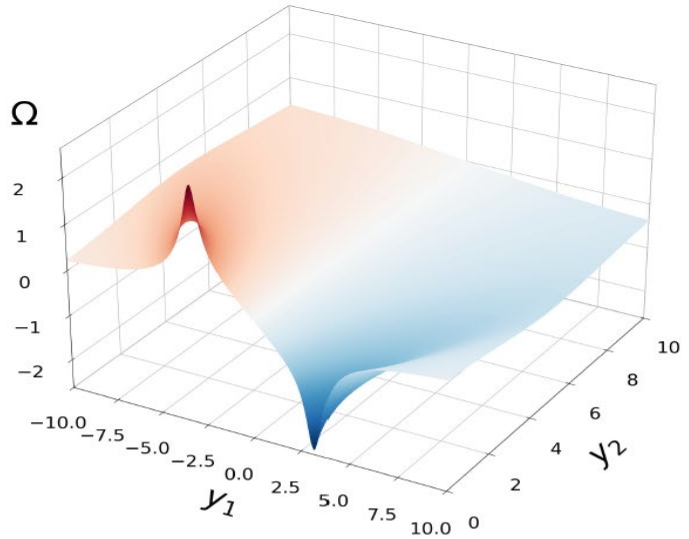


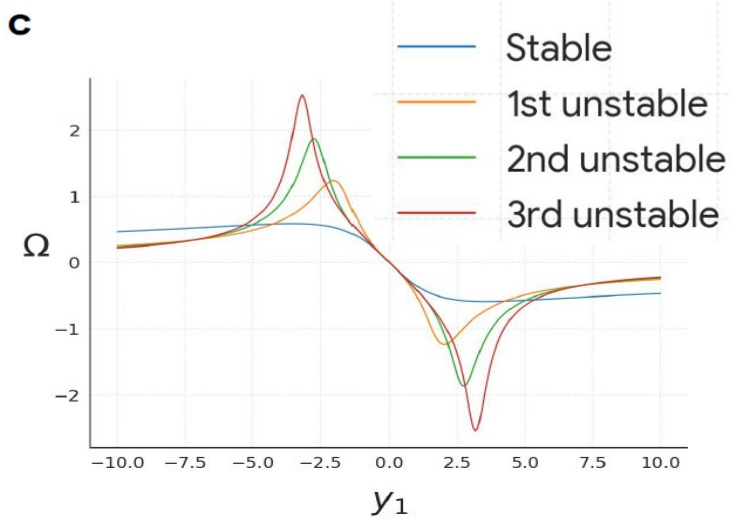


Discovery of a zoom of new unstable singularities!



IPM: 3rd unstable solution

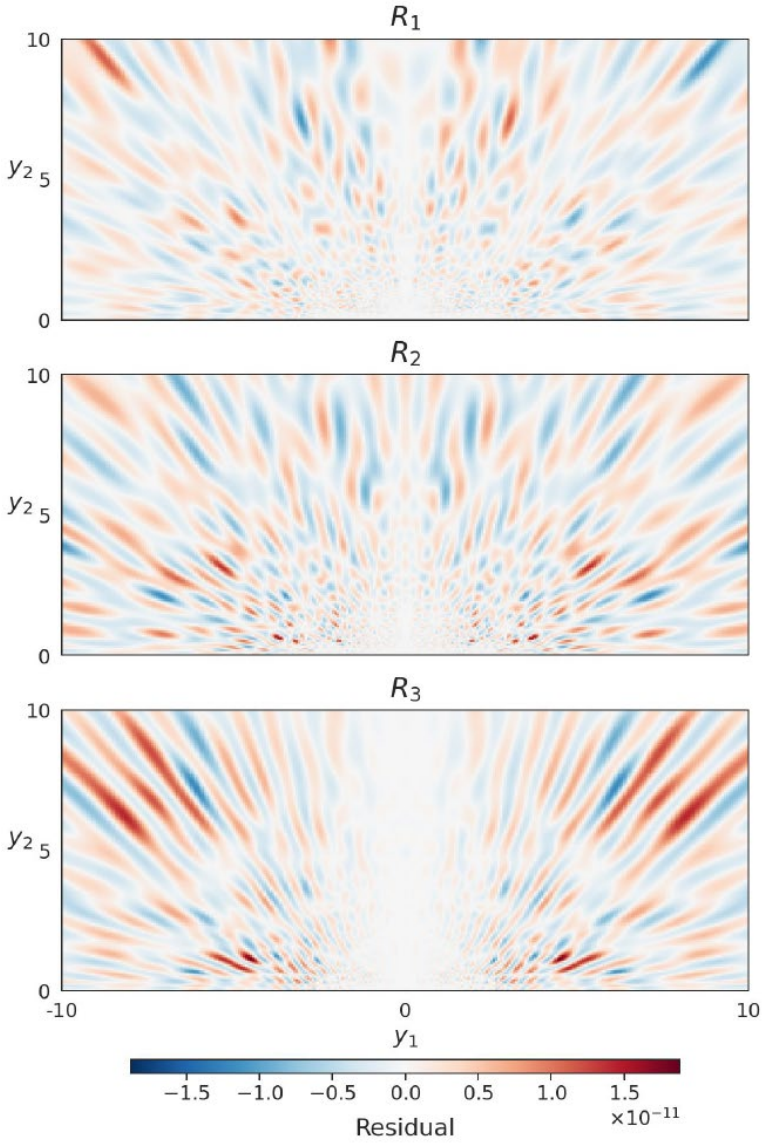




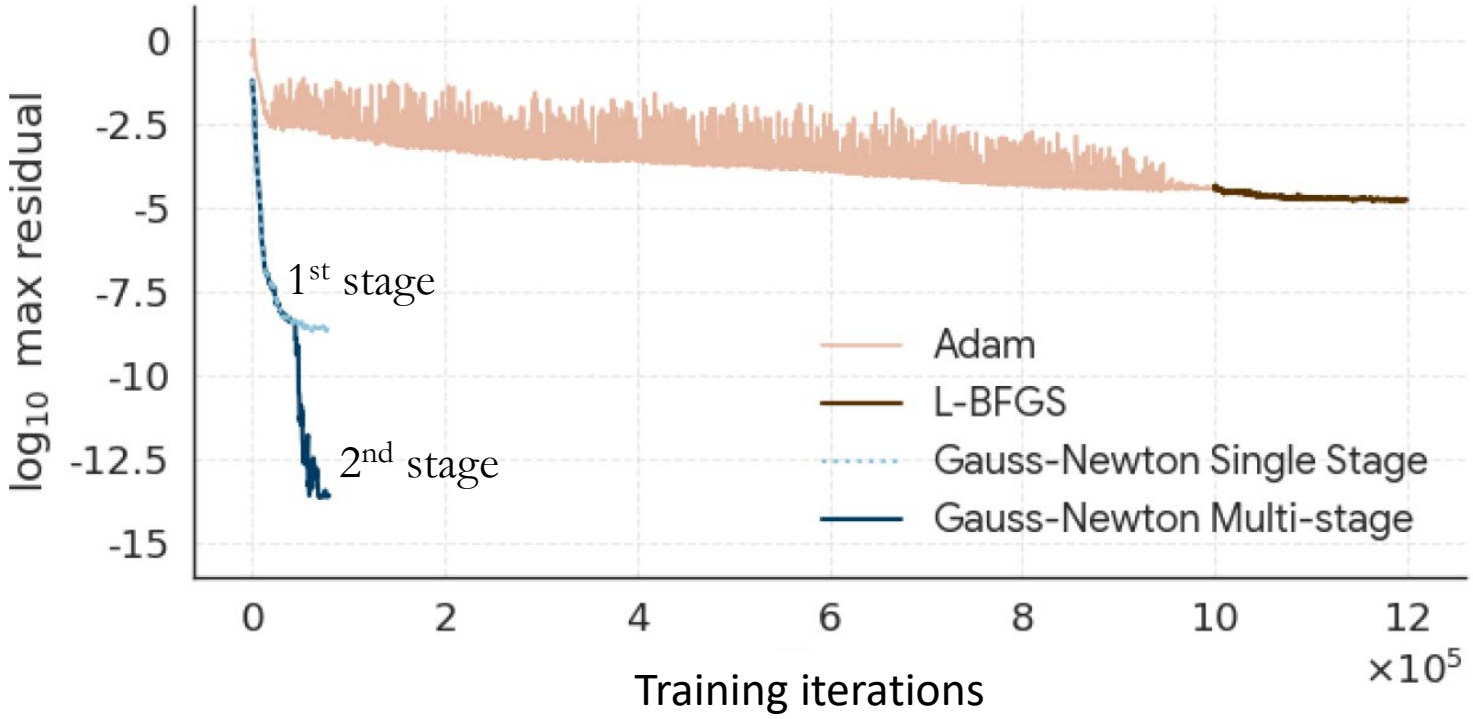
Wang et al., arXiv:2509.14185

Equation	Solution	$\log_{10}$ Maximum Residual
IPM with boundary	Stable	-11.183
	1st unstable	-10.510
	2nd unstable	-8.101
	3rd unstable	-7.526
Euler	Stable	-8.178
	1st unstable	-8.038
	2nd unstable	-7.772
	3rd unstable	-7.558
	4th unstable	-7.020

Discovery of a zoom of new unstable singularities!



Wang et al., arXiv:2509.14185



Discovery of a zoom of new unstable singularities!

Wang et al., arXiv:2509.14185

[Submitted on 17 Sep 2025]

Discovery of Unstable Singularities

Yongji Wang, Mehdi Bennani, James Martens, Sébastien Racanière, Sam Blackwell, Alex Matthews, Stanislav Nikolov, Gonzalo Cao-Labora, Daniel S. Park, Martin Arjovsky, Daniel Worrall, Chongli Qin, Ferran Alet, Borislav Kozlovskii, Nenad Tomašev, Alex Davies, Pushmeet Kohli, Tristan Buckmaster, Bogdan Georgiev, Javier Gómez-Serrano, Ray Jiang, Ching-Yao Lai

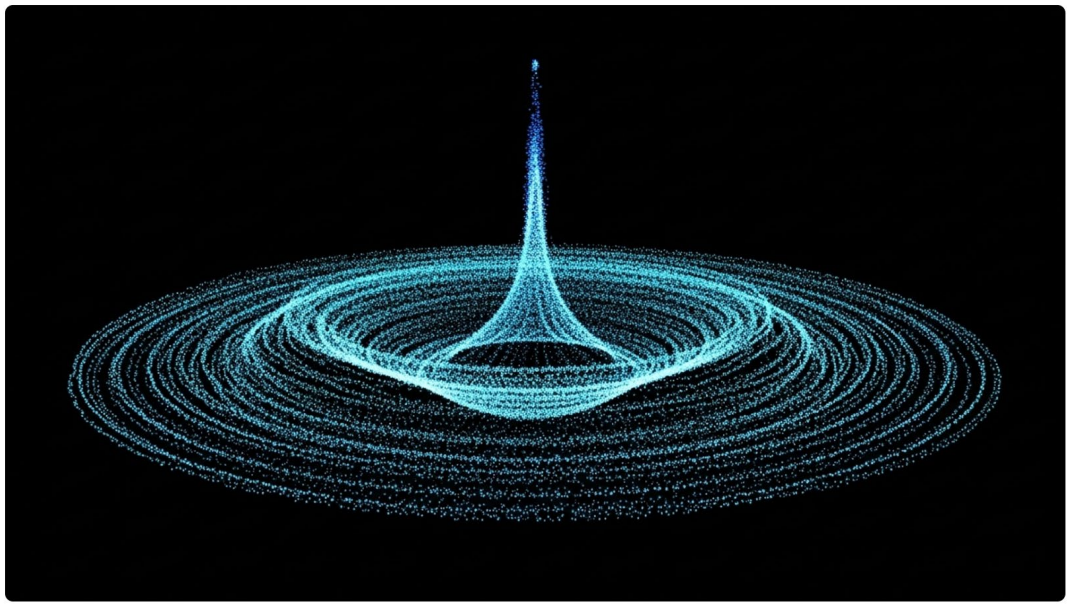
SCIENCE

Discovering new solutions to century-old problems in fluid dynamics

18 SEPTEMBER 2025

Yongji Wang, Sam Blackwell

Share



MATHEMATICAL PHYSICS

Deep Learning Poised to 'Blow Up' Famed Fluid Equations

5

For centuries, mathematicians have tried to prove that Euler's fluid equations can produce nonsensical answers. A new approach to machine learning has researchers betting that "blowup" is near.

Fefferman, "[NN solution] is quantitative and precise and has a much better chance of being made rigorous,"

EXISTENCE AND SMOOTHNESS OF THE NAVIER–STOKES EQUATION

CHARLES L. FEFFERMAN

The Euler and Navier–Stokes equations describe the motion of a fluid in $\mathbb{R}^n$ ($n = 2$ or 3). These equations are to be solved for an unknown velocity vector $u(x,t) = (u_i(x,t))_{1\le i\le n} \in \mathbb{R}^n$ and pressure $p(x,t) \in \mathbb{R}$, defined for position $x \in \mathbb{R}^n$ and time $t \ge 0$. We restrict attention here to incompressible fluids filling all of $\mathbb{R}^n$. The *Navier–Stokes* equations are then given by

What have we learned?

- Multistage neural networks can boost approximation accuracy, tackle the spectral bias and approach machine precision.
- This is useful for representing multiscale functions accurately.
- This is useful for numerically finding highly accurate PDE solutions.
- It can be used for searching singularities in fluids, or more broadly for finding solutions via imposing mathematical structures.
- It found (numerically) unstable blow-up solutions that are otherwise difficult to find.
- Require A TONS of mathematical knowledge to guide the NN find the right thing

Acknowledgement

Code available at:
https://github.com/YaoGroup/MultistageNN

References:

Wang, Lai, Gómez-Serrano, Buckmaster, *Physical Review Letters* (2023)

Wang & Lai, "Multi-stage neural networks: Function approximator of machine precision", *J. Comput. Phys.* (2024)

Ng, Wang & Lai, *ICML workshop* (2024), arXiv:2407.17213

Wang et al. arXiv:2509.14185

Yongji Wang

Postdoc, now at Google DeepMind