Towards Practical Neural PDE Solvers

From RoPINN to ProPINN: Improved Optimization and Architecture

Haixu Wu

Computational Design and Fabrication Group, MIT CSAIL

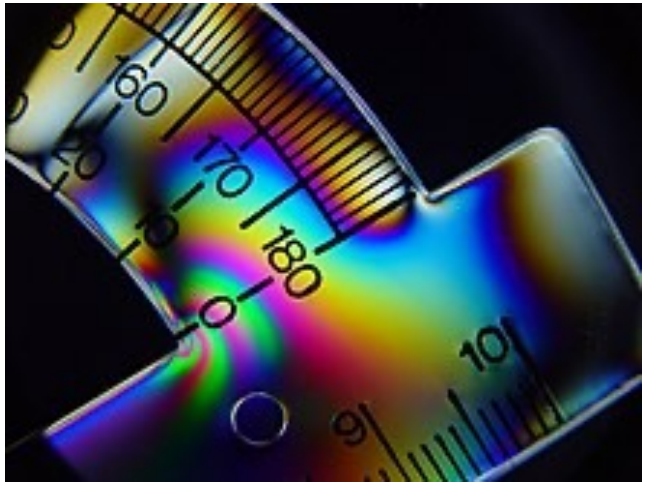
Dec 19, 2025

RoPINN ProPINN

Real-world Phenomena

Turbulence

Atmospheric circulation



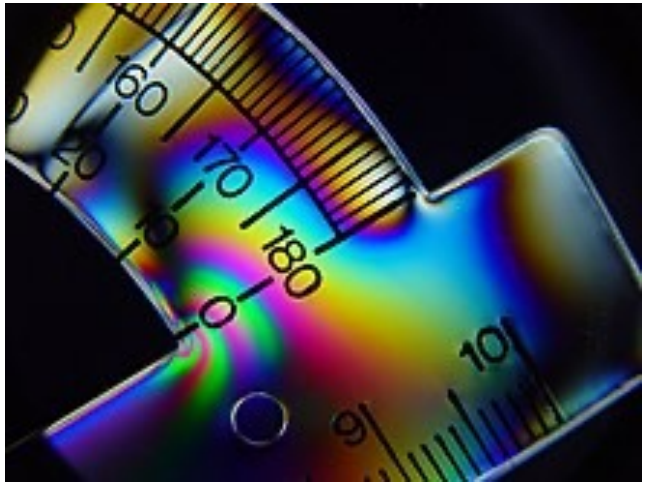
Stress

How to understand the world?

Real-world Phenomena

Turbulence

Atmospheric circulation



Stress

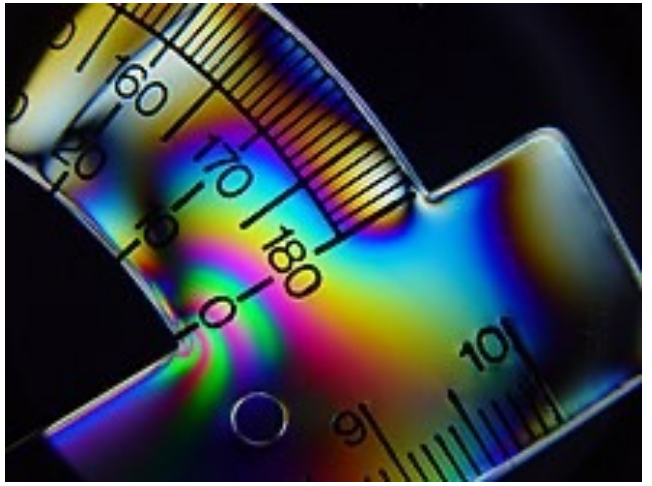
How to understand the world?

Images? Videos? World Model?

Real-world Phenomena

Turbulence

Atmospheric circulation

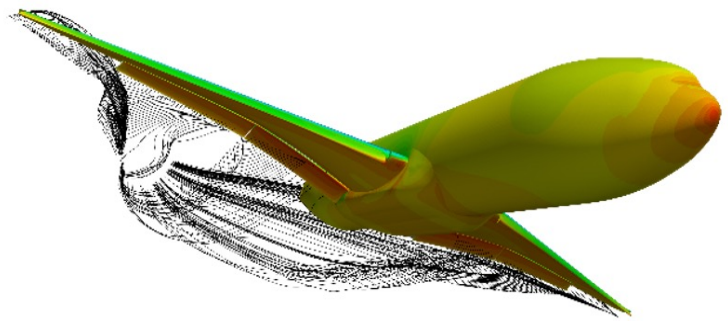


Stress

Beyond appearances, these phenomena are governed by **scientific rules**.

Partial Differential Equations

Extensive physics processes can be precisely described as PDEs.



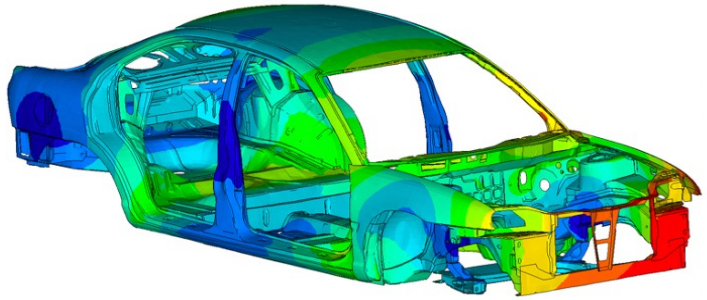
$$\frac{\partial v_x}{\partial x}+\frac{\partial v_y}{\partial y}+\frac{\partial v_z}{\partial z}=0$$

$$\rho\left(\frac{\partial v_x}{\partial t}+v_x\frac{\partial v_x}{\partial x}+v_y\frac{\partial v_x}{\partial y}+v_z\frac{\partial v_x}{\partial z}\right)=-\frac{\partial P}{\partial x}+\mu\left(\frac{\partial^2 v_x}{\partial x^2}+\frac{\partial^2 v_x}{\partial y^2}+\frac{\partial^2 v_x}{\partial z^2}\right)+\rho g_x$$

$$\rho\left(\frac{\partial v_y}{\partial t}+v_x\frac{\partial v_y}{\partial x}+v_y\frac{\partial v_y}{\partial y}+v_z\frac{\partial v_y}{\partial z}\right)=-\frac{\partial P}{\partial y}+\mu\left(\frac{\partial^2 v_y}{\partial x^2}+\frac{\partial^2 v_y}{\partial y^2}+\frac{\partial^2 v_y}{\partial z^2}\right)+\rho g_y$$

$$\rho\left(\frac{\partial v_z}{\partial t}+v_x\frac{\partial v_z}{\partial x}+v_y\frac{\partial v_z}{\partial y}+v_z\frac{\partial v_z}{\partial z}\right)=-\frac{\partial P}{\partial z}+\mu\left(\frac{\partial^2 v_z}{\partial x^2}+\frac{\partial^2 v_z}{\partial y^2}+\frac{\partial^2 v_z}{\partial z^2}\right)+\rho g_z$$

3-D Navier-Stokes equations



$$\varepsilon_{xx}=\frac{\partial u_x}{\partial x},\quad \varepsilon_{yy}=\frac{\partial u_y}{\partial y},\quad \varepsilon_{zz}=\frac{\partial u_z}{\partial z}$$

$$\varepsilon_{xy}=\frac{1}{2}\left(\frac{\partial u_x}{\partial y}+\frac{\partial u_y}{\partial x}\right),\quad \varepsilon_{xz}=\frac{1}{2}\left(\frac{\partial u_x}{\partial z}+\frac{\partial u_z}{\partial x}\right),\quad \varepsilon_{yz}=\frac{1}{2}\left(\frac{\partial u_y}{\partial z}+\frac{\partial u_z}{\partial y}\right)$$

3-D Stress-Strain relations

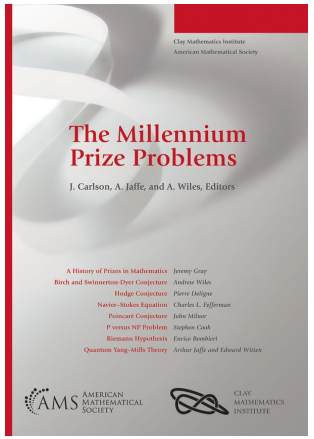
Difficulties in Solving PDEs

David Hilbert

John von Neumann

Peter Lax

Richard Courant



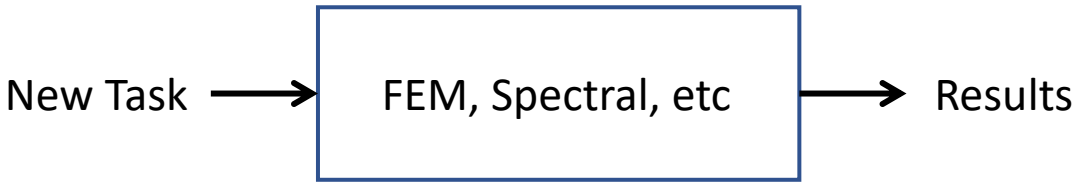
Millennium Prize Problems

- Birch and Swinnerton-Dyer conjecture
- Hodge conjecture
- **Navier–Stokes existence and smoothness**
- **P versus NP problem**
- Riemann hypothesis
- Yang–Mills existence and mass gap
- Poincaré conjecture (Solved)

It is hard (usually impossible) to obtain the analytic solution of PDEs

PDE Solvers

Classic Numerical Methods



- Recalculation for every new sample
- Each round will incur huge costs

Stable vs. Slow and Discretized

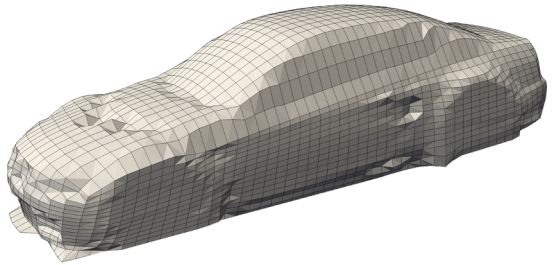
Ansys

DASSAULT SYSTEMES

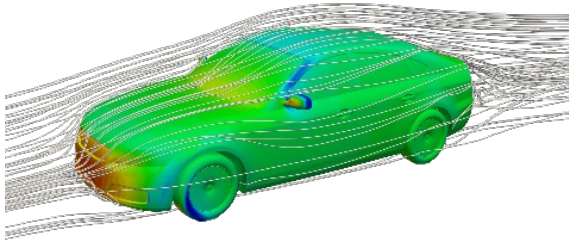
SIEMENS

HEXAGON

Discretized Mesh

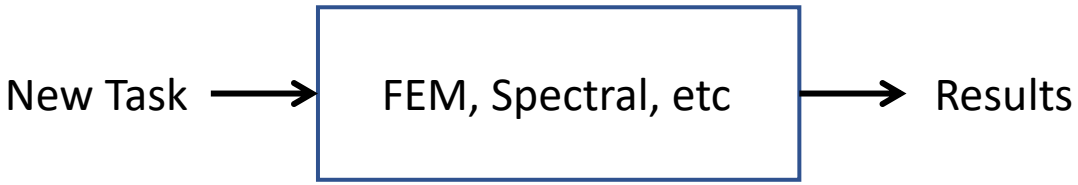


Days or even Months



PDE Solvers

Classic Numerical Methods

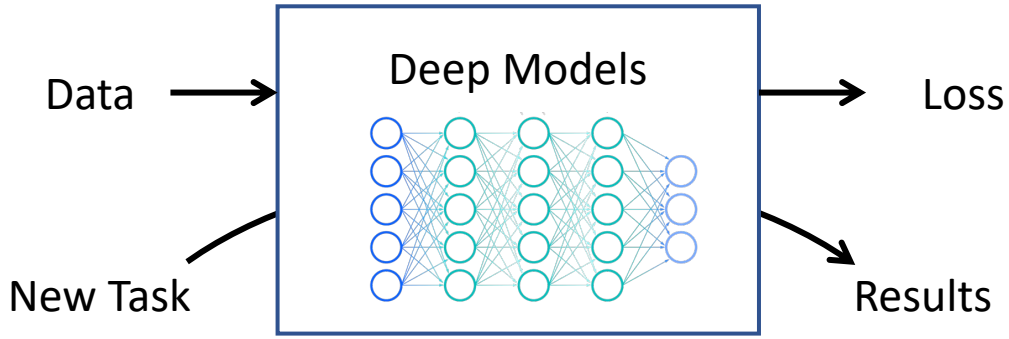


- Recalculation for every new sample
- Each round will incur huge costs

Stable vs. Slow and Discretized

Ansys DASSAULT SYSTEMES SIEMENS HEXAGON

Neural PDE Solvers

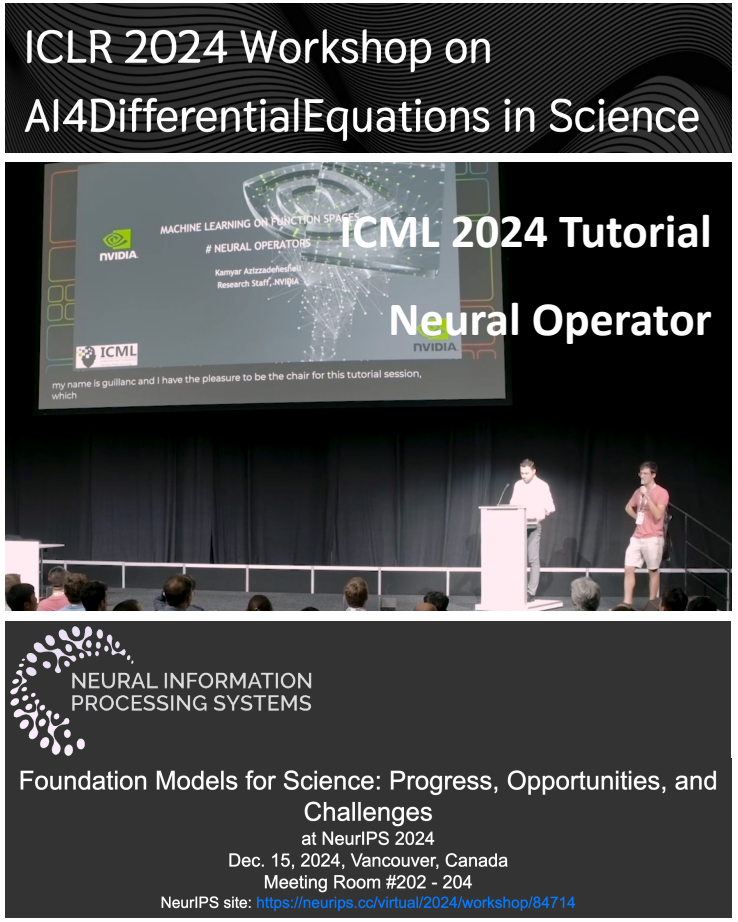


- Training once, inference a lot
- Each round needs several seconds

An efficient / precise surrogate tool

(Ideally)

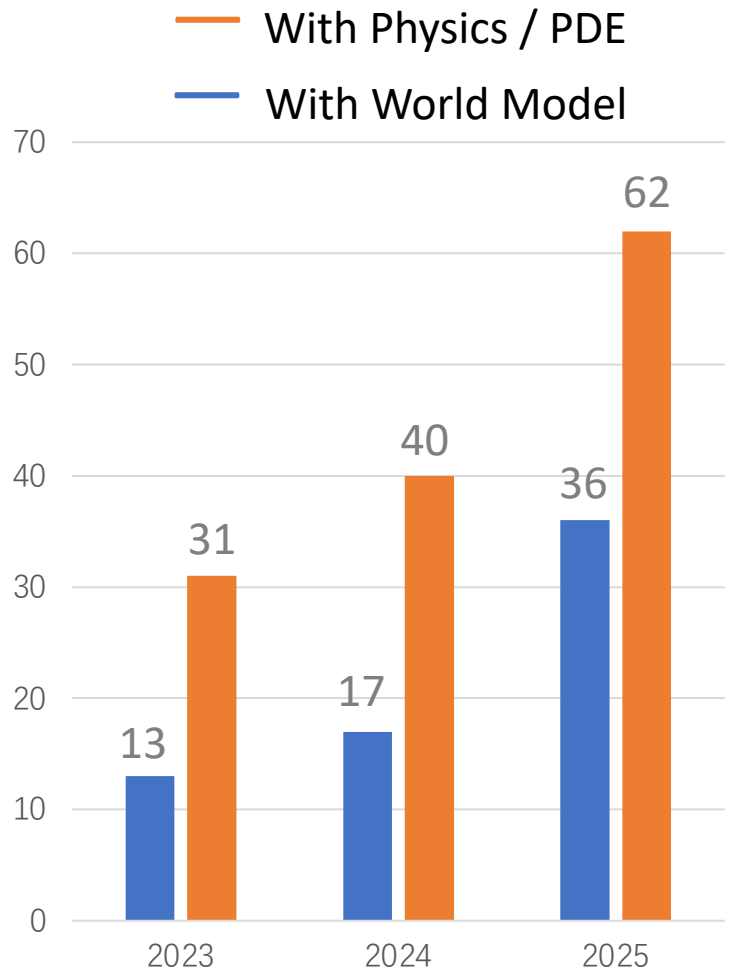
A Booming Direction



2024

ICLR 2025

2025

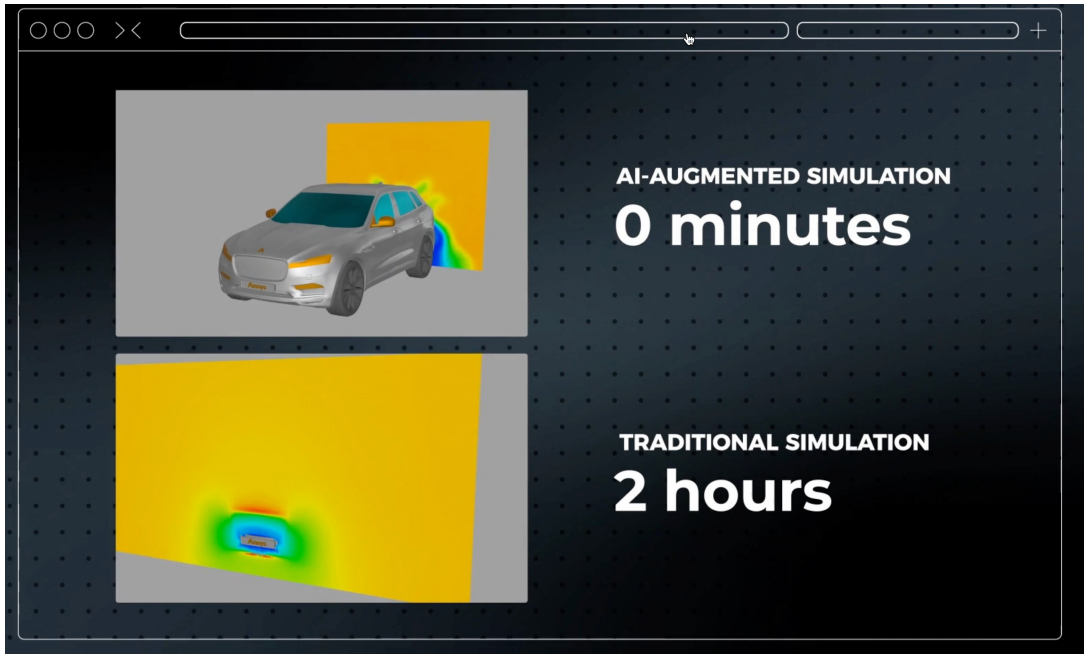


Accepted NeurIPS Papers

AI-empowered Simulation Software

Ansys SimAI

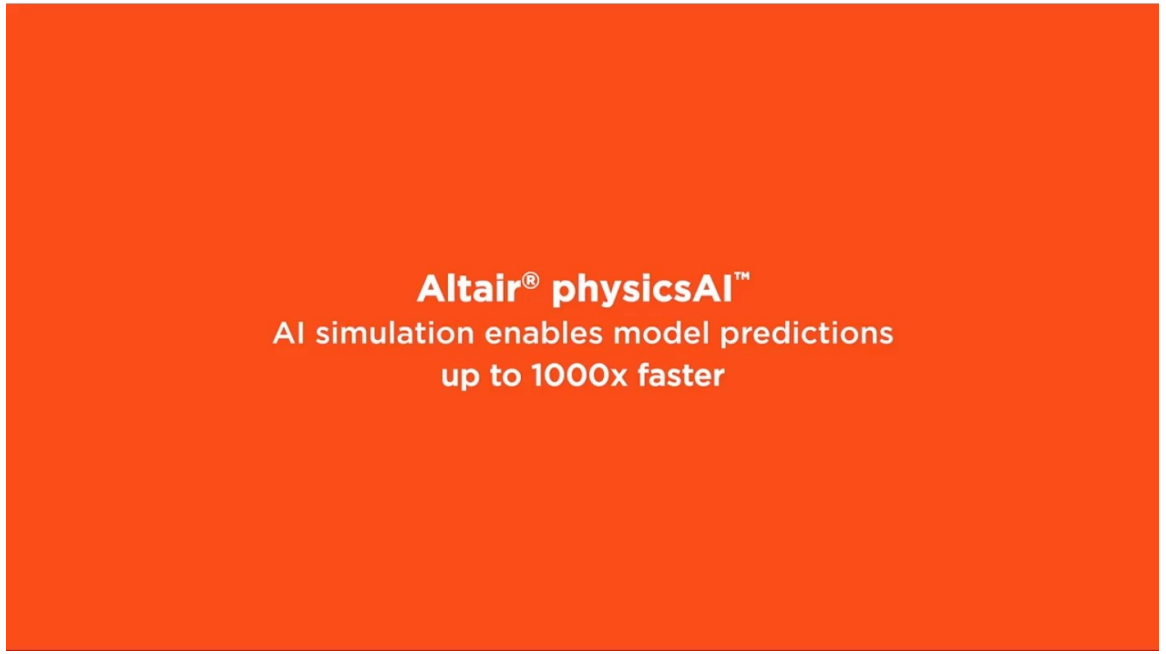
Predict at the Speed of AI



https://www.ansys.com/products/simai

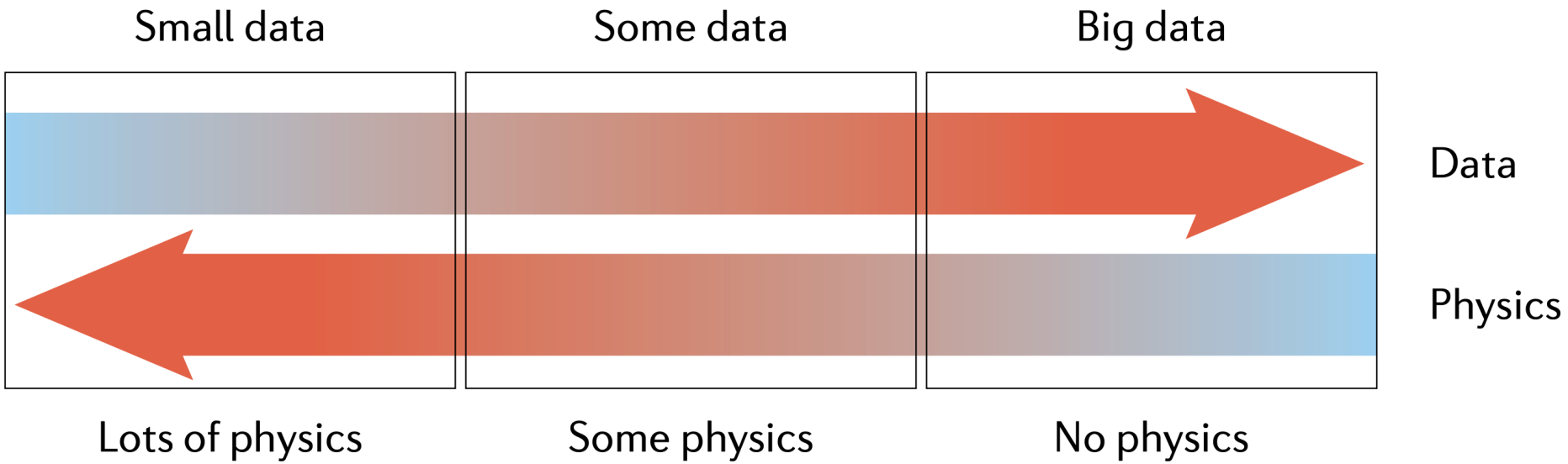
Altair® PhysicsAI™ Geometric Deep Learning

Better Design Insights Up to 1000x Faster than Solver Simulation



https://altair.com/physicsai

Overview of Neural PDE Solvers



Pure Physics

- *Numerical Solvers (FEMs)*
- Physics-Informed Neural Networks / Neural-FEMs

Hybrid Data-Physics

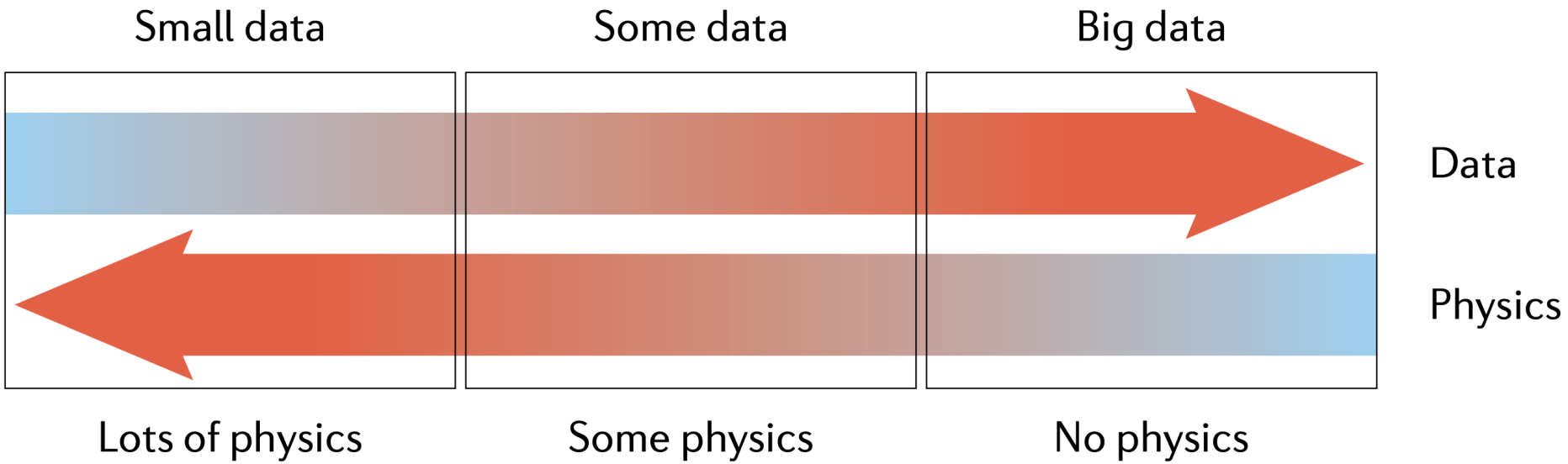
- Hybrid Simulators (NCLaw)
- Physics-Informed Neural Operator (PINO)

Pure Data

- Neural Operators / Neural Surrogates (DeepONet, Transolver)
- *General Deep Models*

Karniadakis, G. et al. Physics-informed machine learning, ***Nature Review Physics*** 3, 422–440 (2021)

Overview of Neural PDE Solvers



Pure Physics

- *Numerical Solvers (FEMs)*
- Physics-Informed Neural Networks / Neural-FEMs

Hybrid Data-Physics

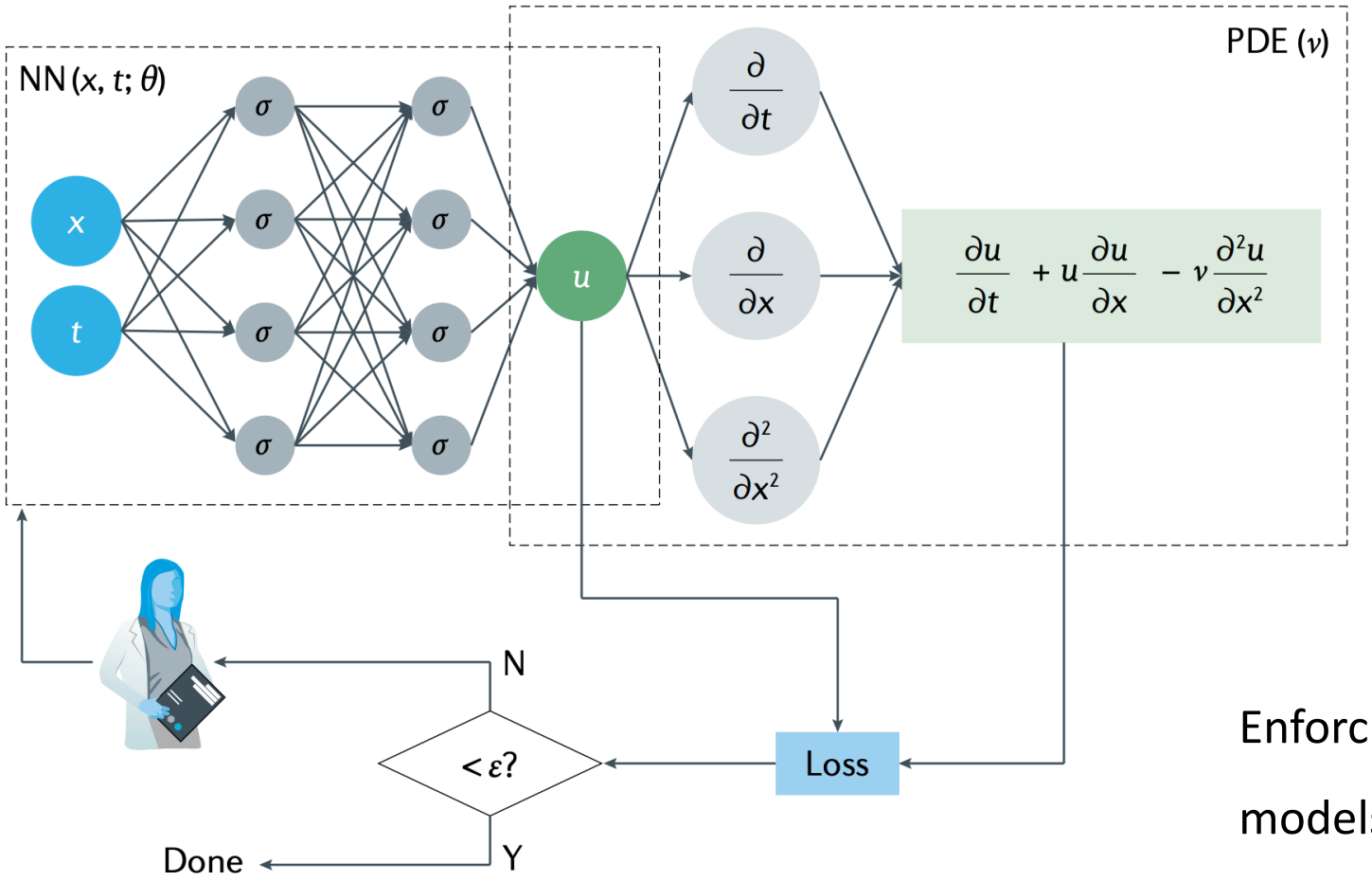
- Hybrid Simulators (NCLaw)
- Physics-Informed Neural Operator (PINO)

Pure Data

- Neural Operators / Neural Surrogates (DeepONet, Transolver)
- *General Deep Models*

Karniadakis, G. et al. Physics-informed machine learning, ***Nature Review Physics*** 3, 422–440 (2021)

Physics-Informed Neural Networks (PINNs)



Enforcing outputs and gradients of deep models to satisfy target equations

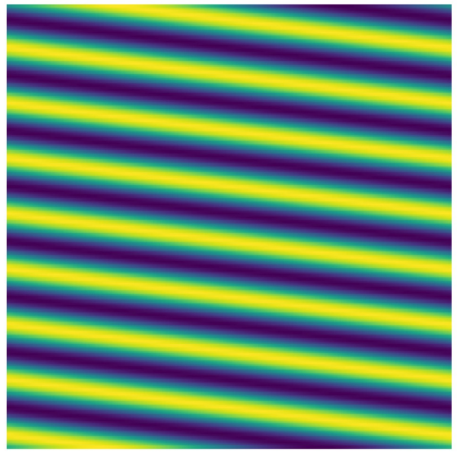
Karniadakis, G. et al. Physics-informed machine learning, ***Nature Review Physics*** 3, 422–440 (2021)

Physics-Informed Neural Networks (PINNs)

$$\mathcal{F}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega;\ \mathcal{I}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega_0;\ \mathcal{B}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \partial\Omega,$$

Convection

$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$



t

x

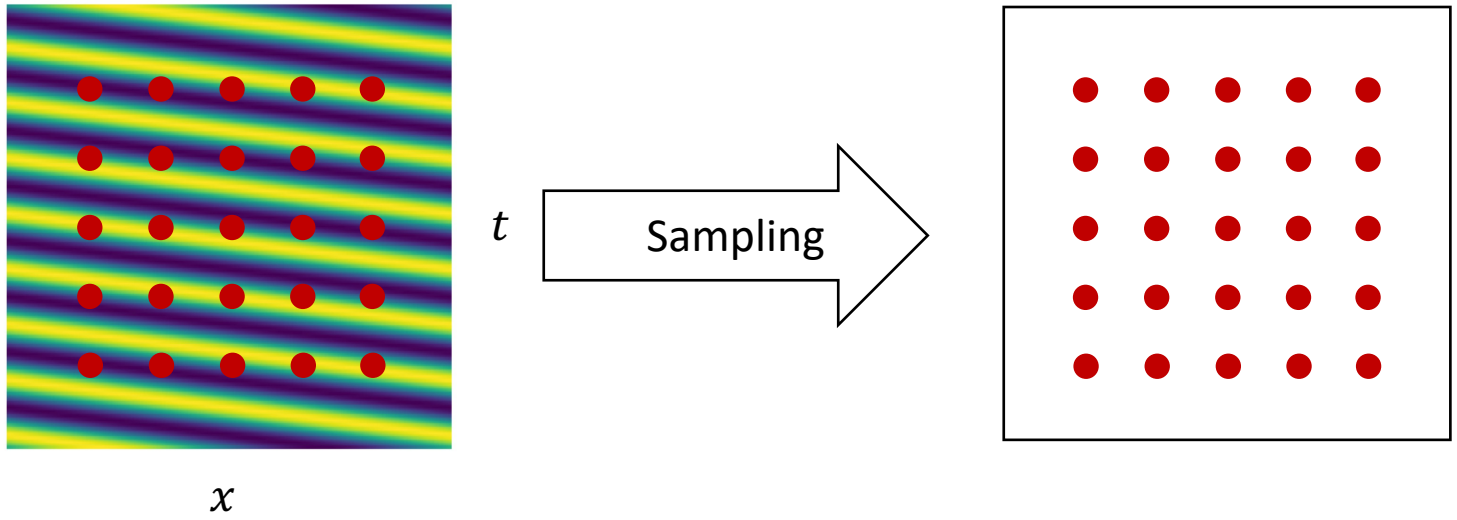
Raissi, M. et al. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, ***Journal of Computational Physics*** 378 (2019): 686-707.

Physics-Informed Neural Networks (PINNs)

$$\mathcal{F}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega;\ \mathcal{I}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega_0;\ \mathcal{B}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \partial\Omega,$$

Convection

$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$



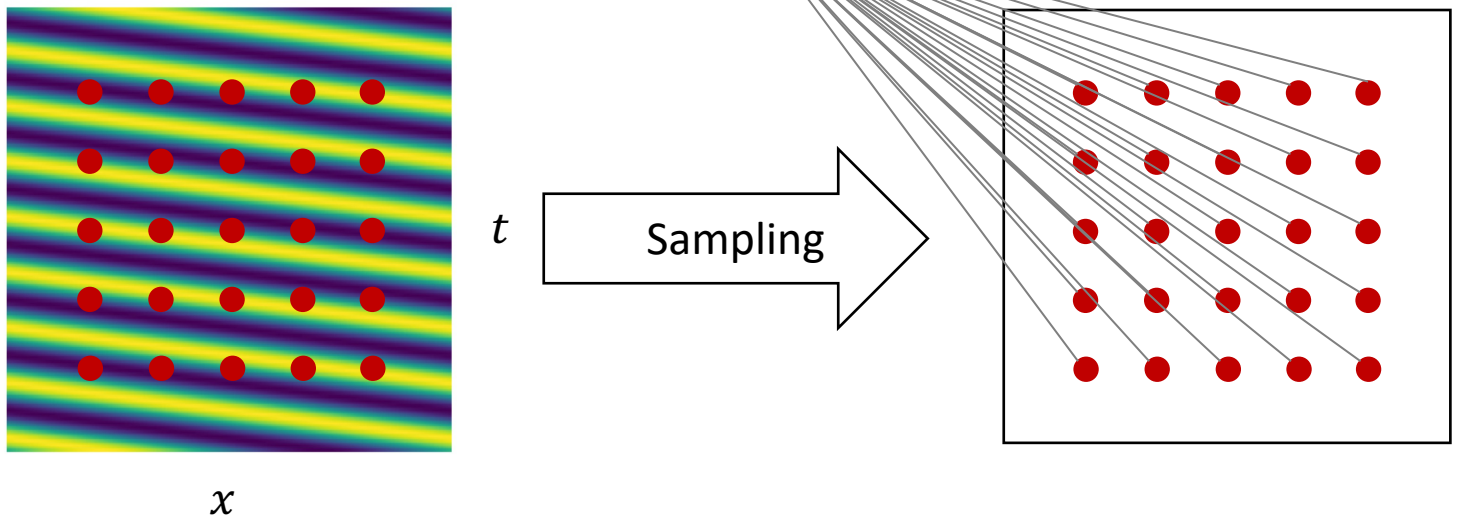
Raissi, M. et al. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, ***Journal of Computational Physics*** 378 (2019): 686-707.

Physics-Informed Neural Networks (PINNs)

$$\mathcal{L}(u_\theta) = \frac{\lambda_\Omega}{N_\Omega}\sum_{i=1}^{N_\Omega} \|\mathcal{F}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\Omega_0}}{N_{\Omega_0}}\sum_{i=1}^{N_{\Omega_0}} \|\mathcal{I}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}} \|\mathcal{B}(u_\theta)(\boldsymbol{x}_i)\|^2$$

Convection

$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$



(Informal define) $L = \left|\left|\frac{\partial u_\theta}{\partial t} + 50\frac{\partial u_\theta}{\partial x}\right|\right|^2 + \left|\left|u_\theta(x,0) - \sin(\mathrm{x})\right|\right|^2 + \left|\left|u_\theta(0,t) - u_\theta(1,t)\right|\right|^2$

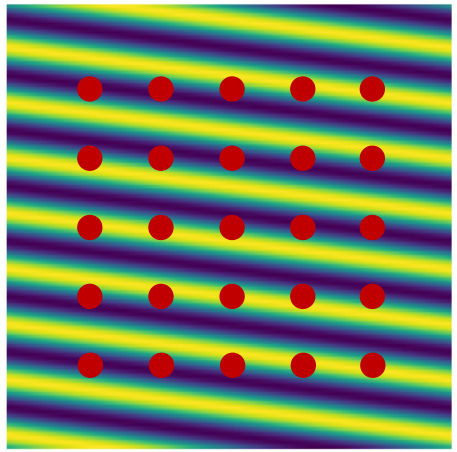
Raissi, M. et al. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, ***Journal of Computational Physics*** 378 (2019): 686-707.

Physics-Informed Neural Networks (PINNs)

$$\mathcal{L}(u_\theta) = \frac{\lambda_\Omega}{N_\Omega}\sum_{i=1}^{N_\Omega} \|\mathcal{F}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\Omega_0}}{N_{\Omega_0}}\sum_{i=1}^{N_{\Omega_0}} \|\mathcal{I}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}} \|\mathcal{B}(u_\theta)(\boldsymbol{x}_i)\|^2$$

Convection

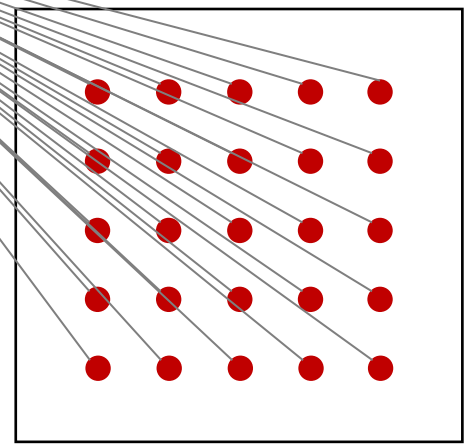
$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$



x

t

Sampling



PINN Loss on Finite

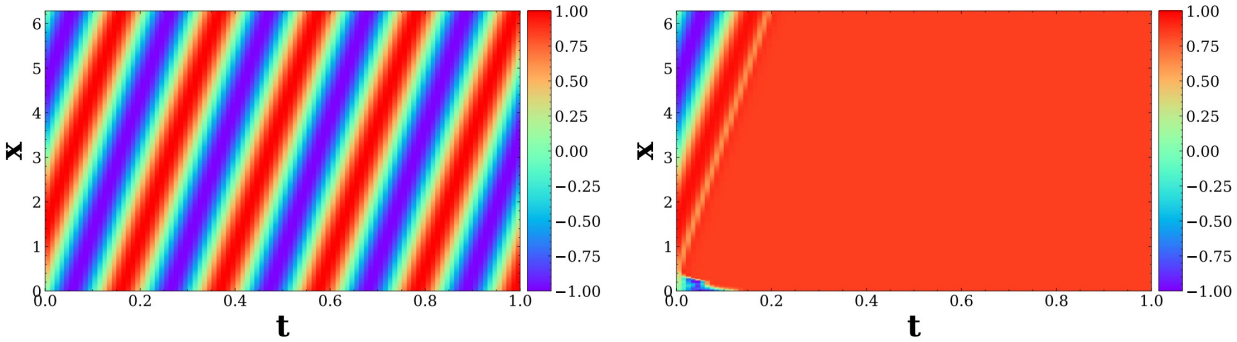
Collocation Points

Extremely elegant formalization (autodifferential gradient, explicit constraint)

but still has some underlying issues to be solved

PINN Failure Modes

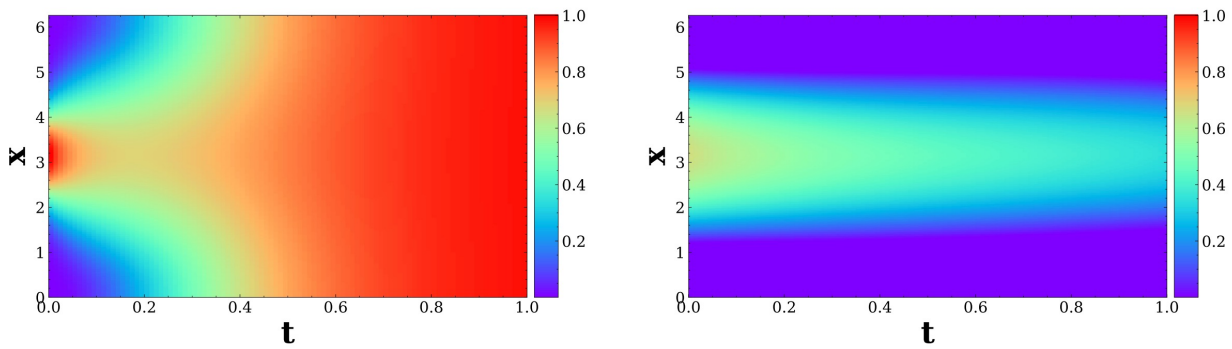
Convection



(b) *Exact solution for* $\beta = 30$ **(c)** *PINN solution for* $\beta = 30$

$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad x \in \Omega,\ t \in [0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega.$$

Reaction-Diffusion



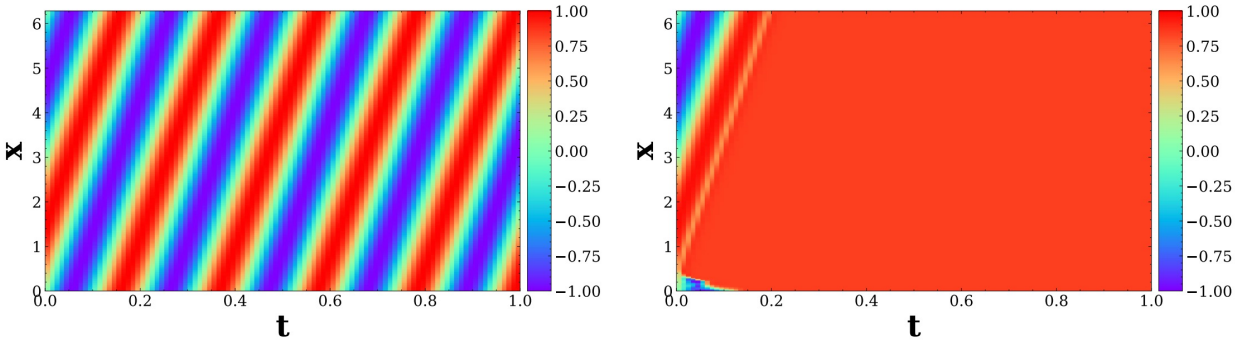
(a) *Exact solution for* $\rho = 5,\ \nu = 5$ **(b)** *PINN solution for* $\rho = 5,\ \nu = 5$

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1 - u) = 0, \quad x \in \Omega,\ t \in (0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega.$$

Krishnapriyan, Aditi S. et al. Characterizing possible failure modes in physics-informed neural networks. ***NeurIPS*** 2021.

PINN Failure Modes

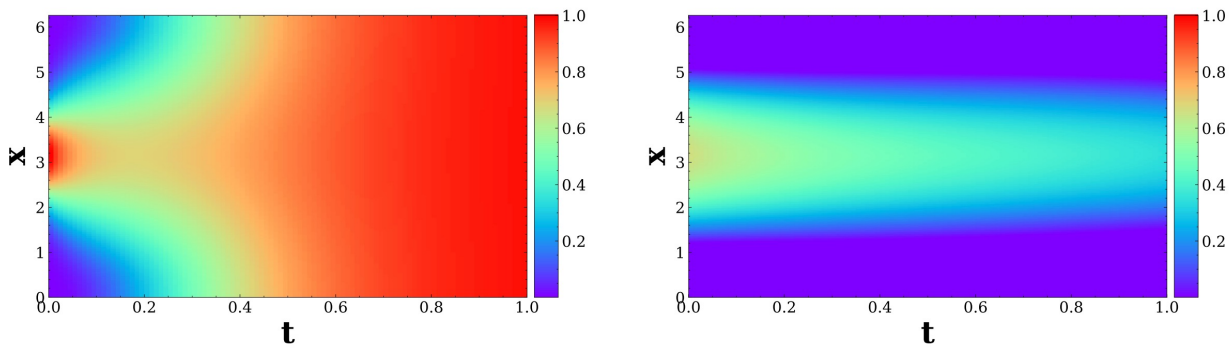
Convection



(b) *Exact solution for* $\beta = 30$ **(c)** *PINN solution for* $\beta = 30$

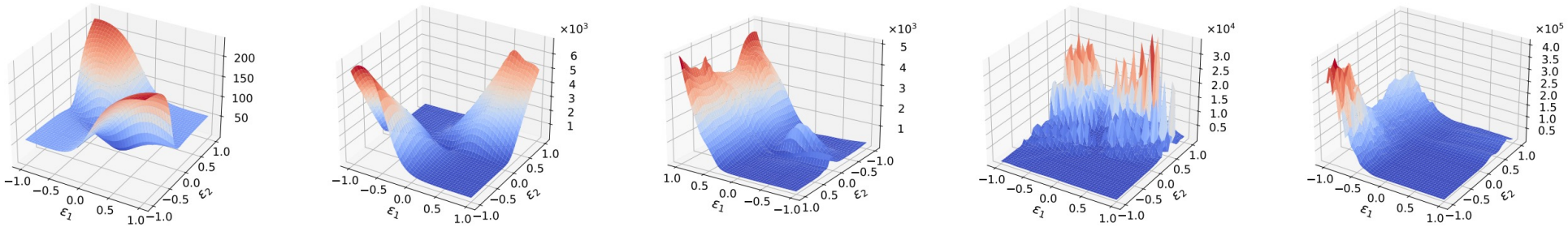
$$\frac{\partial u}{\partial t} + \beta \frac{\partial u}{\partial x} = 0, \quad x \in \Omega,\ t \in [0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega.$$

Reaction-Diffusion



(a) *Exact solution for* $\rho = 5,\ \nu = 5$ **(b)** *PINN solution for* $\rho = 5,\ \nu = 5$

$$\frac{\partial u}{\partial t} - \nu \frac{\partial^2 u}{\partial x^2} - \rho u(1 - u) = 0, \quad x \in \Omega,\ t \in (0, T],$$
$$u(x, 0) = h(x), \quad x \in \Omega.$$



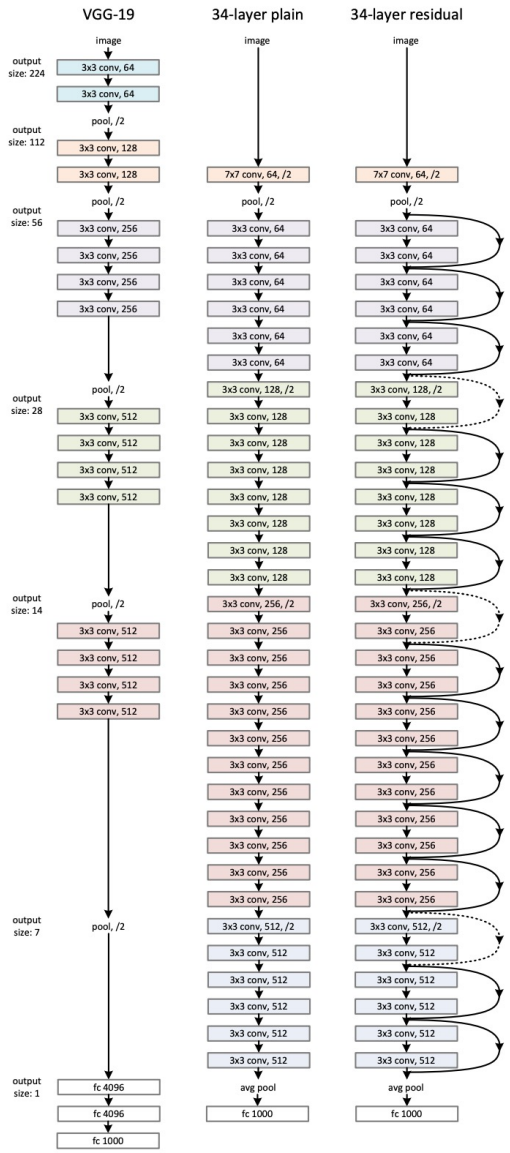
(a) $\beta = 1.0$ (b) $\beta = 10.0$ (c) $\beta = 20.0$ (d) $\beta = 30.0$ (e) $\beta = 40.0$

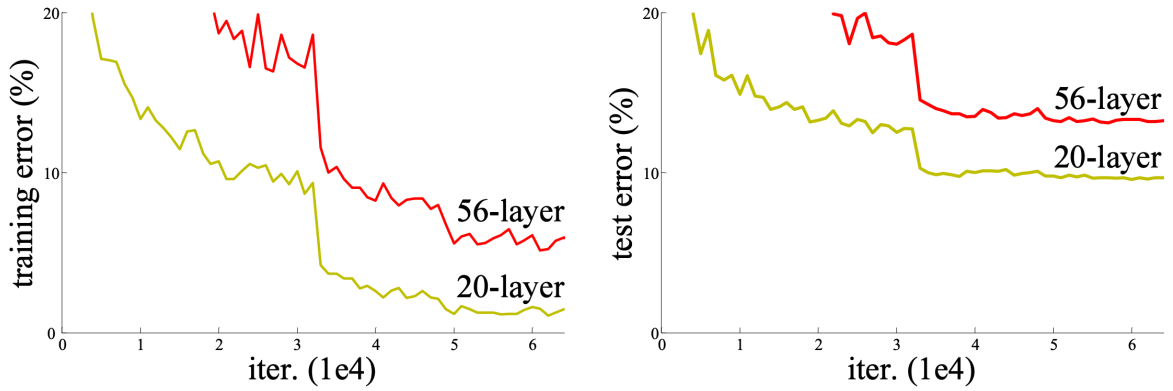
Hard-to-Optimize

Loss Landscape

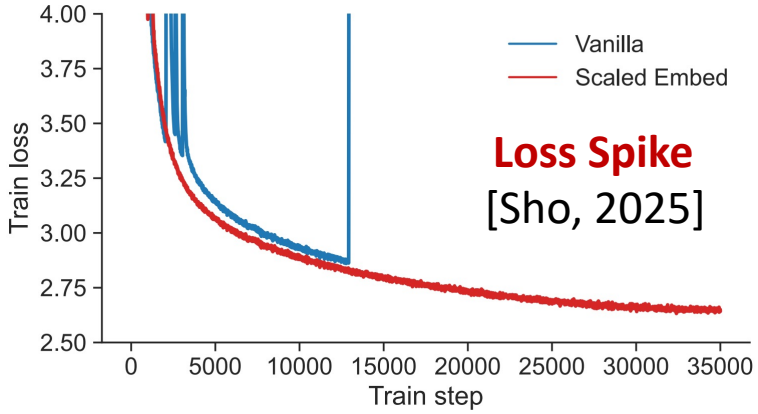
Krishnapriyan, Aditi S. et al. Characterizing possible failure modes in physics-informed neural networks. ***NeurIPS*** 2021.

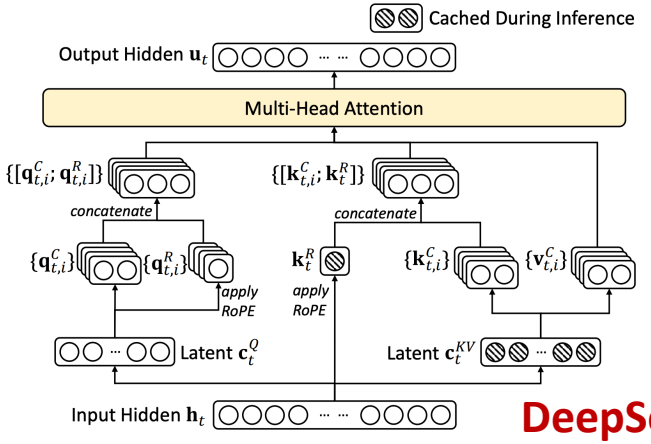
A Long-Standing Challenge in DL: Optimization





A plain neural network cannot benefit from adding layers.





DeepSeek-V3: New architecture ensures stable training.

He, et al. Deep Residual Learning for Image Recognition. ***CVPR*** 2026.
DeepSeek-V3 Technical Report. 2025

NEURAL INFORMATION PROCESSING SYSTEMS

RoPINN: Region Optimized Physics-Informed Neural Networks

Haixu Wu, Huakun Luo, Yuezhou Ma, Jianmin Wang, Mingsheng Long✉
School of Software, BNRist, Tsinghua University, China
{wuhx23,luohk19,mayz20}@mails.tsinghua.edu.cn, {jimwang,mingsheng}@tsinghua.edu.cn


Haixu Wu


Huakun Luo


Yuezhou Ma

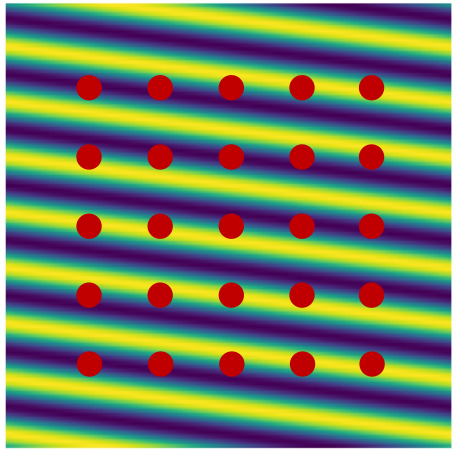

Jianmin Wang


Mingsheng Long

Paper Link: *https://openreview.net/pdf?id=wZigMVFURk*

Code Link: *https://github.com/thuml/RoPINN*

Rethinking the PINN Formalization



$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$

$$\mathcal{L}(u_\theta) = \frac{\lambda_\Omega}{N_\Omega}\sum_{i=1}^{N_\Omega} \|\mathcal{F}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\Omega_0}}{N_{\Omega_0}}\sum_{i=1}^{N_{\Omega_0}} \|\mathcal{I}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}} \|\mathcal{B}(u_\theta)(\boldsymbol{x}_i)\|^2$$

1. Point Optimization
2. Insufficient Enforcement of Physics Loss

- Train on **limited collocation points** but expect the model to **generalize to the whole domain**.
- Train with **"first-order" loss** but expect the model to **satisfy the infinite-order constraint**.

Direct Solution: High-order regularization

$$\mathcal{F}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega;\ \mathcal{I}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \Omega_0;\ \mathcal{B}(u)(\boldsymbol{x}) = 0, \boldsymbol{x} \in \partial\Omega,$$

⇩ *Differential function*

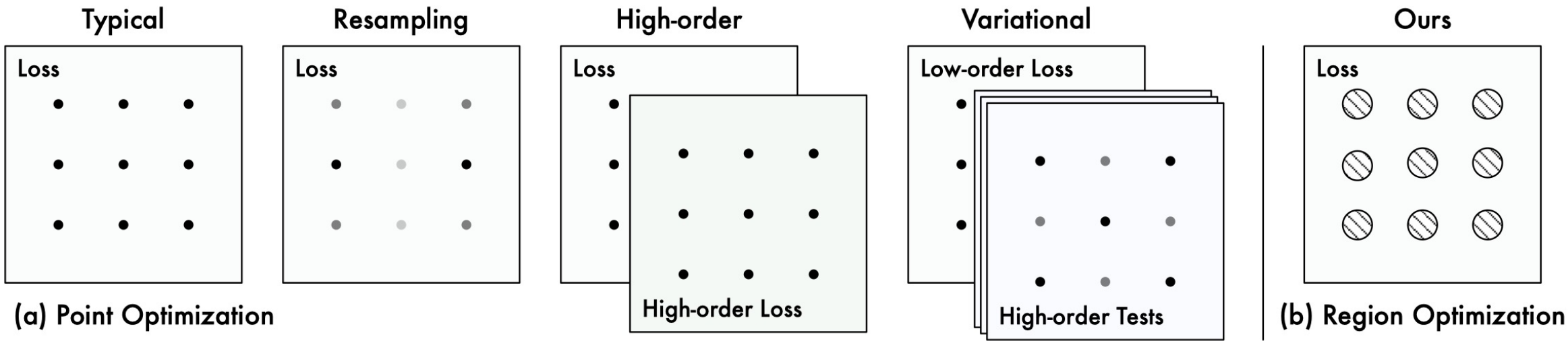
$$\frac{\partial^k}{\partial x_j^k}\mathcal{F}(u)(\boldsymbol{x}) = 0 \quad \Rightarrow \quad \mathcal{L}_{k,j}^{\mathrm{reg}}(u_\theta) = \frac{\lambda_{k,j}}{N_{k,j}} \sum_{i=1}^{N_{k,j}} \left\| \frac{\partial^k}{\partial x_j^k}\mathcal{F}(u_\theta)(\boldsymbol{x}_i) \right\|^2$$

✓ Add the high-order constraints of PDEs as regularization terms to the loss function

✗ Calculating high-order derivatives can be **extremely time-consuming and unstable**

Yu, Jeremy et al. Gradient-enhanced physics-informed neural networks for forward and inverse PDE problems, ***Computer Methods in Applied Mechanics and Engineering***, 2022

Region Optimization V.S. Point Optimization



Point Optimization:

$$\mathcal{L}(u_\theta) = \frac{\lambda_\Omega}{N_\Omega}\sum_{i=1}^{N_\Omega}\|\mathcal{F}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\Omega_0}}{N_{\Omega_0}}\sum_{i=1}^{N_{\Omega_0}}\|\mathcal{I}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}}\|\mathcal{B}(u_\theta)(\boldsymbol{x}_i)\|^2$$

Region Optimization:

$$\mathcal{L}_r^{\text{region}}(u_\theta, \mathcal{S}) = \frac{1}{|\mathcal{S}|}\sum_{\boldsymbol{x}\in\mathcal{S}}\mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x}) = \frac{1}{|\Omega_r|\times|\mathcal{S}|}\sum_{\boldsymbol{x}\in\mathcal{S}}\int_{\Omega_r}\mathcal{L}(u_\theta, \boldsymbol{x}+\boldsymbol{\xi})\mathrm{d}\boldsymbol{\xi}$$

Theoretical Analysis

- Generalization Error in Expectation

$$\mathcal{E}_{\text{gen}} = \left| \mathbb{E}_{\mathcal{S},\mathcal{A}} \left[\mathcal{L}\left(u_{\mathcal{A}(\mathcal{S})}, \Omega\right) - \mathcal{L}\left(u_{\mathcal{A}(\mathcal{S})}, \mathcal{S}\right) \right] \right|$$

Input Domain **Collocation Points**

- Basic Assumption

$$\|\mathcal{L}(u_{\theta_1}, \boldsymbol{x}) - \mathcal{L}(u_{\theta_2}, \boldsymbol{x})\| \leq L\|\theta_1 - \theta_2\|, \ \|\nabla_\theta \mathcal{L}(u_{\theta_1}, \boldsymbol{x}) - \nabla_\theta \mathcal{L}(u_{\theta_2}, \boldsymbol{x})\| \leq \beta\|\theta_1 - \theta_2\|.$$

Theorem 3.3 (Point optimization). *Suppose that the loss function $\mathcal{L}$ is L-Lipschitz-β-smooth for θ. If we run stochastic gradient method with step size α_t at the t-th step for T iterations, we have that:*

(1) If $\mathcal{L}$ is convex for θ and $\alpha_t \leq \frac{2}{\beta}$, then $\mathcal{E}_{\text{gen}} \leq \frac{2L^2}{|\mathcal{S}|} \sum_{t=1}^{T} \alpha_t$ (proved by [13, 52]).

(2) If $\mathcal{L}$ is bounded by a constant C for all $\theta, \boldsymbol{x}$ and is non-convex for θ with monotonically non-increasing step sizes $\alpha_t \leq \frac{1}{\beta t}$, then $\mathcal{E}_{\text{gen}} \leq \frac{C}{|\mathcal{S}|} + \frac{2L^2(T-1)}{\beta(|\mathcal{S}|-1)}$ (tighter bound than [13, 52]).

Theoretical Analysis: Generalization Bound

Theorem 3.5 (Region optimization). *Suppose that the point optimization loss function $\mathcal{L}$ is L-Lipschitz and β-smooth for θ. If we run stochastic gradient method with step size α_t for T iterations based on region optimization loss $\mathcal{L}_r^{\text{region}}$ in Eq. (5), the generalization error in expectation satisfies:*

(1) If $\mathcal{L}$ is convex for θ and $\alpha_t \leq \frac{2}{\beta}$, then $\mathcal{E}_{\text{gen}} \leq (1 - \frac{|\Omega_r|}{|\Omega|}) \frac{2L^2}{|\mathcal{S}|} \sum_{t=1}^{T} \alpha_t$.

Linearly related to region size

(2) If $\mathcal{L}$ is bounded by a constant C for all $\theta, \boldsymbol{x}$ and is non-convex for θ with monotonically non-increasing step sizes $\alpha_t \leq \frac{1}{\beta t}$, then $\mathcal{E}_{\text{gen}} \leq \frac{C}{|\mathcal{S}|} + \frac{2L^2(T-1)}{\beta(|\mathcal{S}|-1)} - JL(\frac{|\Omega_r|}{|\Omega|})^2$, where J is a finite number that depends on the training property at the several beginning iterations.

- Canonical Point Optimization: $\Omega_r = 0$

Cannot benefit from introducing "region"

- Globally sampling points: $\Omega_r = \Omega$

Equivalent to directly optimizing the loss defined on Ω, generalization error will be reduced to zero.

Cannot be satisfied in practice, which requires the precise calculation of the integral of Ω

Practical Algorithm

Algorithm 1 Region Optimized PINN (RoPINN)

Input: number of iterations T, number of past iterations T_0 retained to estimate the trust region, default region size r, initial PINN parameters θ_0 and trust region calibration value $\sigma_0 = 1$.
Output: optimized PINN parameters θ_T.
Initialize an empty buffer to record gradients as $\mathbf{g}$.
for $t = 0$ **to** T **do**
// Region Optimization with Monte Carlo Approximation ① Monte Carlo Approximation
Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$
Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$
Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$
// Trust Region Calibration ② Trust Region Calibration
Record the gradient of parameters g_t throughout optimization
Update gradient buffer $\mathbf{g}$ by adding g_t and keeping the latest T_0 elements
Trust region calibration with $\sigma_{t+1} = \|\sigma(\mathbf{g})\|$
end for

Part 1: Monte Carlo approximation

for $t = 0$ **to** T **do**

// Region Optimization with Monte Carlo Approximation

Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

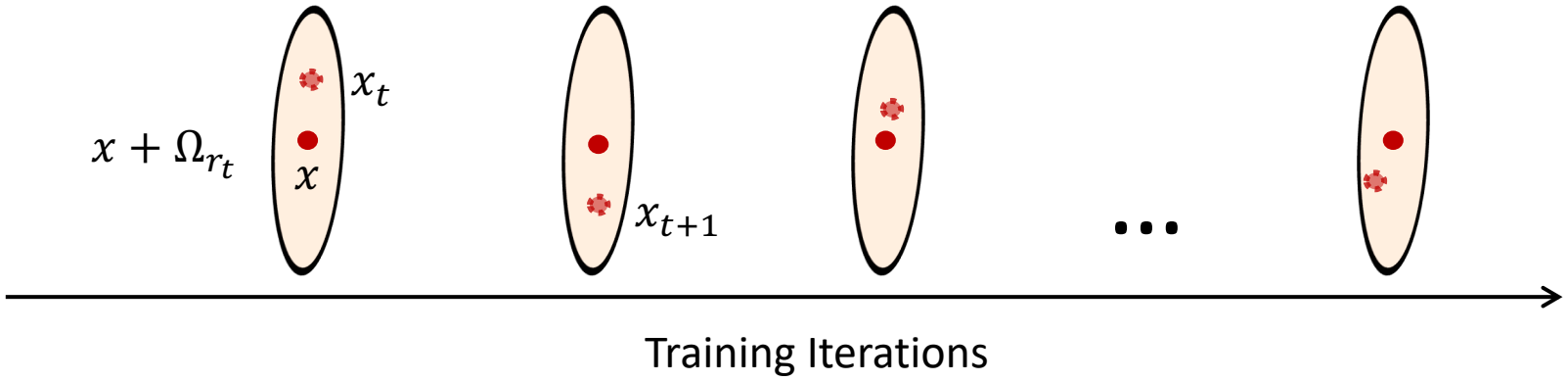
Sampling within a region

Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$

Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$

- Approximate the region optimization gradient by **Monte Carlo approximation**

$$\mathbb{E}_{\boldsymbol{\xi} \sim U(\Omega_r)}\left[\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi})\right] = \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})$$



Part 1: Monte Carlo approximation

for $t = 0$ **to** T **do**

// Region Optimization with Monte Carlo Approximation

Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

Sampling within a region

Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$

Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$

- Approximate the region optimization gradient by **Monte Carlo approximation**

$$\mathbb{E}_{\boldsymbol{\xi} \sim U(\Omega_r)}\left[\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi})\right] = \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})$$

- This sampling-based design is also equivalent to a **high-order loss function**

$$\mathbb{E}_{\boldsymbol{\xi} \sim U(\Omega_r)}\left(\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi})\right) = \mathbb{E}_{\boldsymbol{\xi} \sim U(\Omega_r)}\left(\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x}) + \nabla_\theta(\boldsymbol{\xi}^\top \mathcal{L}_1(u_\theta, \boldsymbol{x})) + \mathcal{O}(\|\boldsymbol{\xi}\|^2)\right)$$

Important Note: This design is tailored to PINN loss,

where we can precisely calculate the loss at any sampled point.

Part 1: Monte Carlo approximation

for $t = 0$ **to** T **do**

// Region Optimization with Monte Carlo Approximation

Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$

Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$

- Approximate the region optimization gradient by **Monte Carlo approximation**

$$\mathbb{E}_{\boldsymbol{\xi} \sim U(\Omega_r)}\left[\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi})\right] = \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})$$

Theorem 3.8 (Convergence rate). *Suppose that there exists a constant H, s.t. $\forall \boldsymbol{v}$ and $\forall \boldsymbol{x} \in \Omega$, $\left|\boldsymbol{v}^\mathsf{T} \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x}) \boldsymbol{v}\right| \leq H\|\boldsymbol{v}\|^2$. If the step size $\alpha_t = \frac{1}{\sqrt{t+1}}$ decreases over time for T iterations, the region optimization based on Monte Carlo approximation will converge at the speed of*

$$\mathbb{E}\left[\left\|\nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})\right\|^2\right] \leq \mathcal{O}\left(\frac{1}{\sqrt{T}}\right). \tag{10}$$

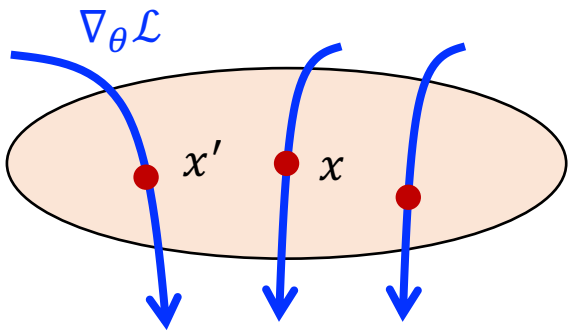
Part 2: Trust Region Calibration

Theorem 3.9 (Gradient estimation error). *The estimation error of gradient descent between Monte Carlo approximation and the original region optimization satisfies:*

$$\mathbb{E}_{\boldsymbol{\xi}\sim U(\Omega_r)}\left[\left\|\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x}+\boldsymbol{\xi}) - \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})\right\|^2\right]^{\frac{1}{2}} = \left\|\sigma_{\boldsymbol{\xi}\sim U(\Omega_r)}\left(\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x}+\boldsymbol{\xi})\right)\right\|, \quad (11)$$

where σ represents the standard deviation of gradients in region Ω_r.

Gradient variance within a region.



Region $x + \Omega_r$

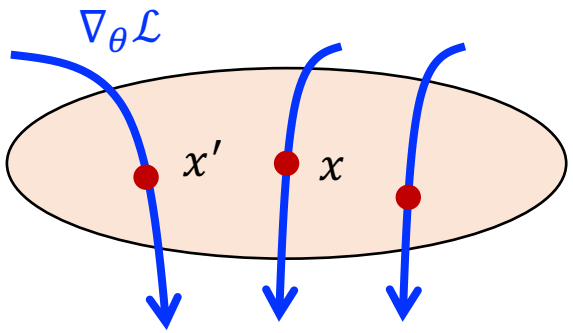
Part 2: Trust Region Calibration

Theorem 3.9 (Gradient estimation error). *The estimation error of gradient descent between Monte Carlo approximation and the original region optimization satisfies:*

$$\mathbb{E}_{\boldsymbol{\xi}\sim U(\Omega_r)}\left[\left\|\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x}+\boldsymbol{\xi}) - \nabla_\theta \mathcal{L}_r^{\text{region}}(u_\theta, \boldsymbol{x})\right\|^2\right]^{\frac{1}{2}} = \left\|\sigma_{\boldsymbol{\xi}\sim U(\Omega_r)}\left(\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x}+\boldsymbol{\xi})\right)\right\|, \quad (11)$$

where σ represents the standard deviation of gradients in region Ω_r.

Gradient variance within a region.



Region $x + \Omega_r$

Recall Generalization error: $\mathcal{E}_{\text{gen}} \leq (1 - \frac{|\Omega_r|}{|\Omega|})\frac{2L^2}{|\mathcal{S}|}\sum_{t=1}^{T}\alpha_t$

- A larger region size r: **better generalization** but will bring **larger gradient estimation error**.

Part 2: Trust Region Calibration

for $t = 0$ **to** T **do**

// Region Optimization with Monte Carlo Approximation

Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$

Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$

// Trust Region Calibration

Record the gradient of parameters g_t throughout optimization

Update gradient buffer $\mathbf{g}$ by adding g_t and keeping the latest T_0 elements

Trust region calibration with $\sigma_{t+1} = \|\sigma(\mathbf{g})\|$

end for

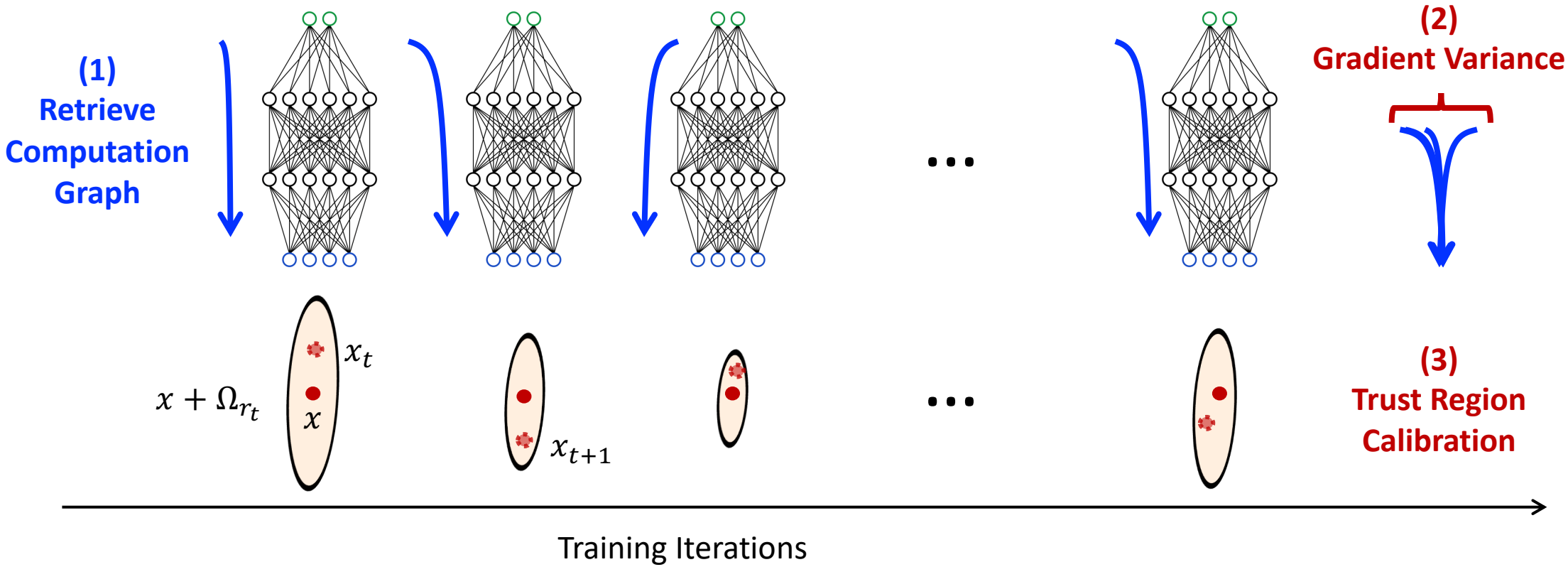
Adjust region size according to the gradient variance among successive iterations.

$$r \propto \frac{1}{\left\|\sigma_{\boldsymbol{\xi} \sim U(\Omega_r)} \left(\nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi})\right)\right\|}$$

Region size

✓ Similar ideas are widely used in deep learning optimizers, such as Adam and AdaGrad, which adopt multi-iteration statistics as the momentum of gradient descent.

Part 2: Trust Region Calibration



✓ The gradient of each iteration can be effectively obtained by retrieving the computation graph.

RoPINN has no extra gradient or backpropagation calculation w.r.t. point optimization.

Part 2: Trust Region Calibration

for $t = 0$ **to** T **do**

// Region Optimization with Monte Carlo Approximation

Sample points from neighborhood regions: $\mathcal{S}' = \{\boldsymbol{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|S|}, \boldsymbol{x}_i \in \mathcal{S}, \boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

Calculate loss function $\mathcal{L}_t = \mathcal{L}\left(u_{\theta_t}, \mathcal{S}'\right)$

Update θ_t to θ_{t+1} with optimizer (Adam [20], L-BFGS [25], etc) to minimize loss function $\mathcal{L}_t$

// Trust Region Calibration

Record the gradient of parameters g_t throughout optimization

Update gradient buffer $\mathbf{g}$ by adding g_t and keeping the latest T_0 elements

Trust region calibration with $\sigma_{t+1} = \|\sigma(\mathbf{g})\|$

end for

Theorem 3.11 (Trust region multi-iteration approximation). *Suppose that loss function $\mathcal{L}$ is L-Lipschitz and β-smooth for θ and the learning rate $\alpha_t \leq \frac{1}{\beta L}$ converges to zero over time t, then the estimation error can be approximated by the variance of optimization gradients in multiple successive iterations. Given hyperparameter T_0, our multi-iteration approximation is guaranteed by*

$$\lim_{t\to\infty} \sigma\left(\left\{\nabla_\theta \mathcal{L}(u_{\theta_{t-i+1}}, \boldsymbol{z}_i)\right\}_{i=1}^{T_0}\right) = \sigma\left(\left\{\nabla_\theta \mathcal{L}(u_{\theta_t}, \boldsymbol{z}_i)\right\}_{i=1}^{T_0}\right). \tag{14}$$

Theoretical Analysis

Theorem 3.12 (Region Optimization with gradient estimation error). *Based on the same assumption in Theorem 3.5 but optimize the model with the approximated region optimization loss* $\mathcal{L}_r^{\text{approx}}(u_\theta, \boldsymbol{x}) = \nabla_\theta \mathcal{L}(u_\theta, \boldsymbol{x} + \boldsymbol{\xi}), \boldsymbol{\xi} \sim U(\Omega_r)$ *for* T *iterations, we further denote the upper bound of gradient estimation error as* $\mathcal{E}_{r,\text{grad}} = \max_{t \le T} \|\nabla_\theta \mathcal{L}_r^{\text{approx}} - \nabla_\theta \mathcal{L}_r^{\text{region}}\|$, *then* $\mathcal{E}_{\text{gen}}$ *satisfies:*

(1) If $\mathcal{L}$ *is convex for* θ *and* $\alpha_t \le \frac{2}{\beta}$, $\mathcal{E}_{\text{gen}} \le \Big(\underbrace{(1 - |\Omega_r|/|\Omega|)L}_{\text{inversely proportional to } |\Omega_r|} + \underbrace{\mathcal{E}_{r,\text{grad}}}_{\text{generally} \propto |\Omega_r|} \Big) \frac{2L}{|\mathcal{S}|} \sum_{t=1}^{T} \alpha_t.$

(1) Benefit from region opt (2) Gradient approximation error

(2) If $\mathcal{L}$ *is bounded by a constant* C *and is non-convex for* θ *with monotonically non-increasing step sizes* $\alpha_t \le \frac{1}{\beta t}$, *then* $\mathcal{E}_{\text{gen}} \le \frac{C}{|\mathcal{S}|} + \frac{2L^2(T-1)}{\beta(|\mathcal{S}|-1)} \underbrace{-J' L (|\Omega_r|/|\Omega|)^2}_{\text{inversely proportional to } |\Omega_r|} + \underbrace{J' \mathcal{E}_{r,\text{grad}}(1 + |\Omega_r|/|\Omega|)}_{\text{generally} \propto |\Omega_r|},$

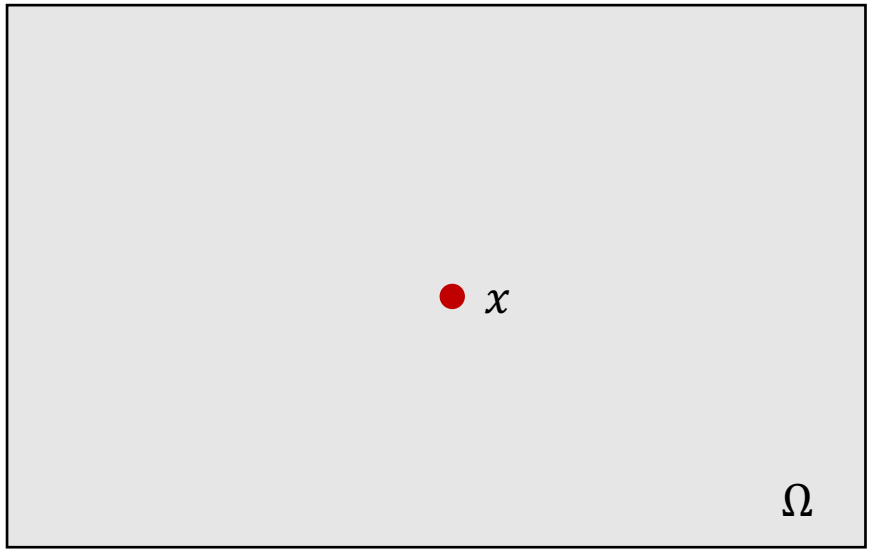
where J' *is a finite number that depends on the training property at the several beginning iterations.*

Generalization $\propto |\Omega_r|$ **Optimization** $\propto -|\Omega_r|$

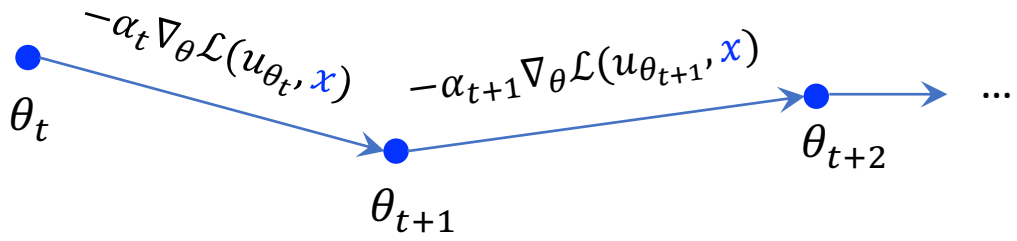
- Canonical point optimization ($\Omega_r = 0$) and globally sampling points ($\Omega_r = \Omega$) are fixed special cases.

RoPINN can adaptively balance optimization and generalization during training.

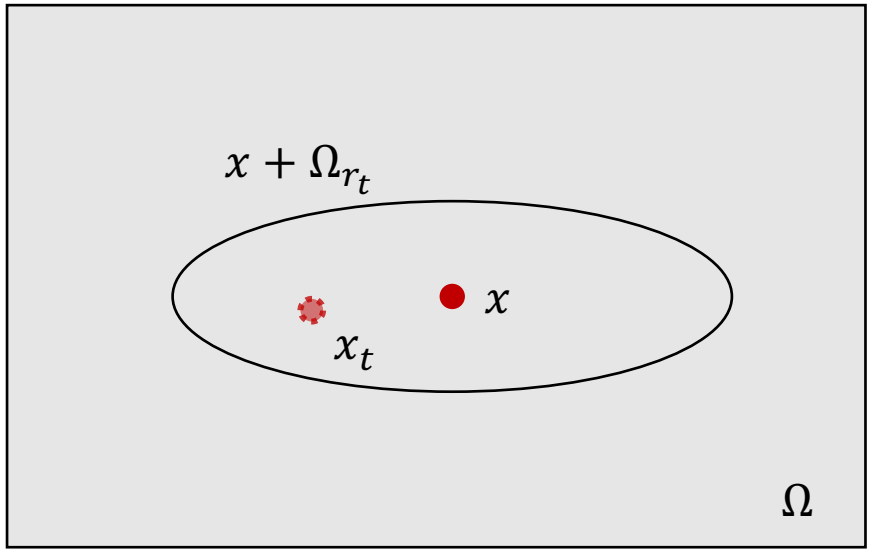
Intuitive Understanding



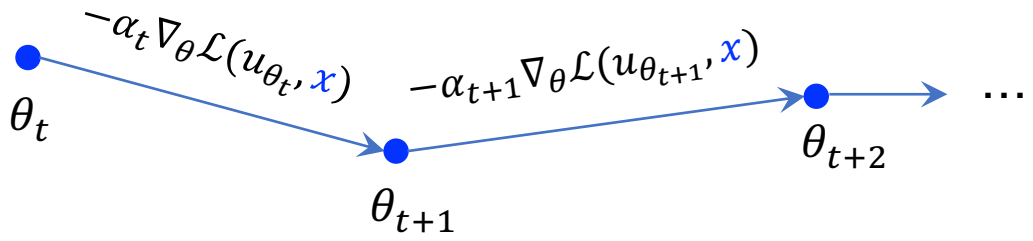
Point optimization: calculate gradient on the fixed collocation point in all iterations



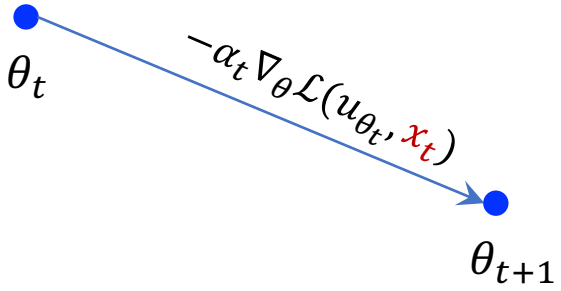
Intuitive Understanding



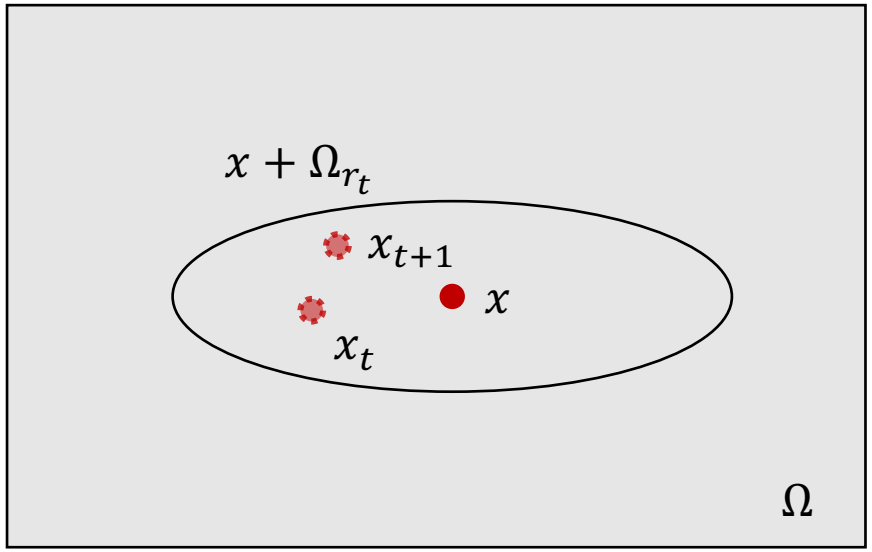
Point optimization: calculate gradient on the fixed collocation point in all iterations



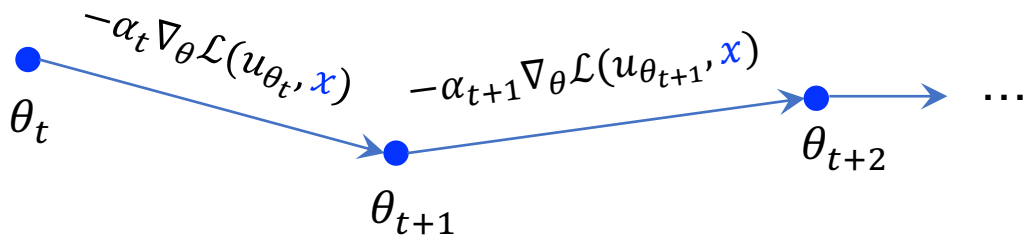
RoPINN: Approximate the region gradient by accumulating gradients from multiple iterations



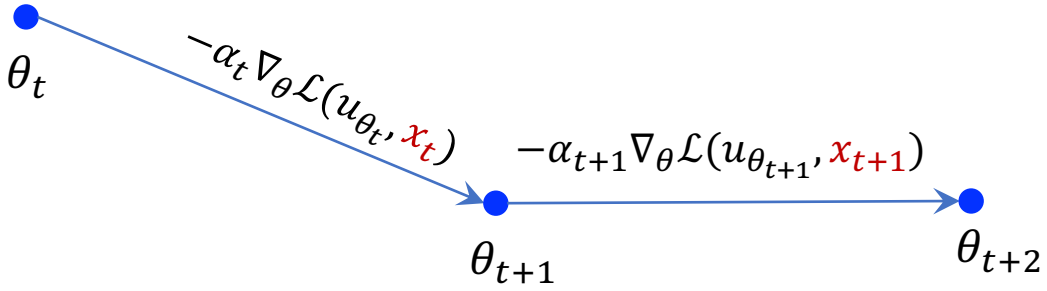
Intuitive Understanding



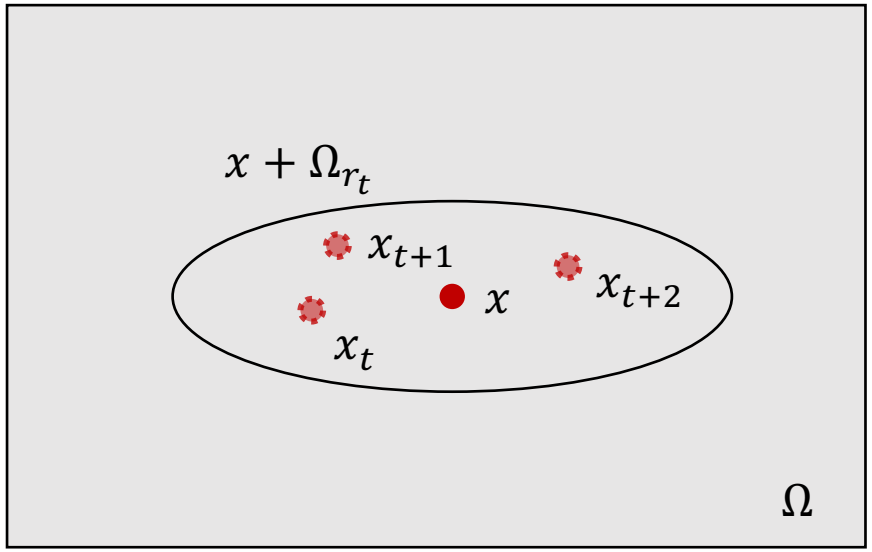
Point optimization: calculate gradient on the fixed collocation point in all iterations



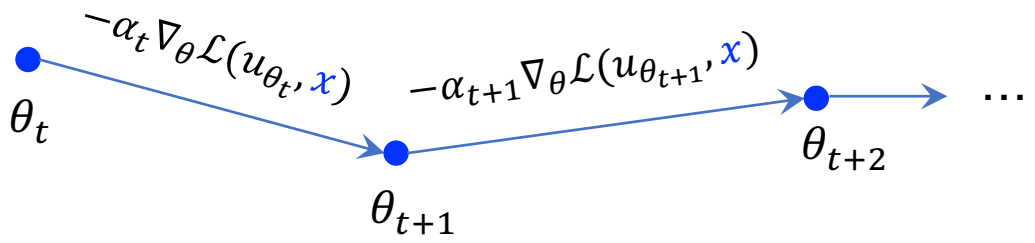
RoPINN: Approximate the region gradient by accumulating gradients from multiple iterations



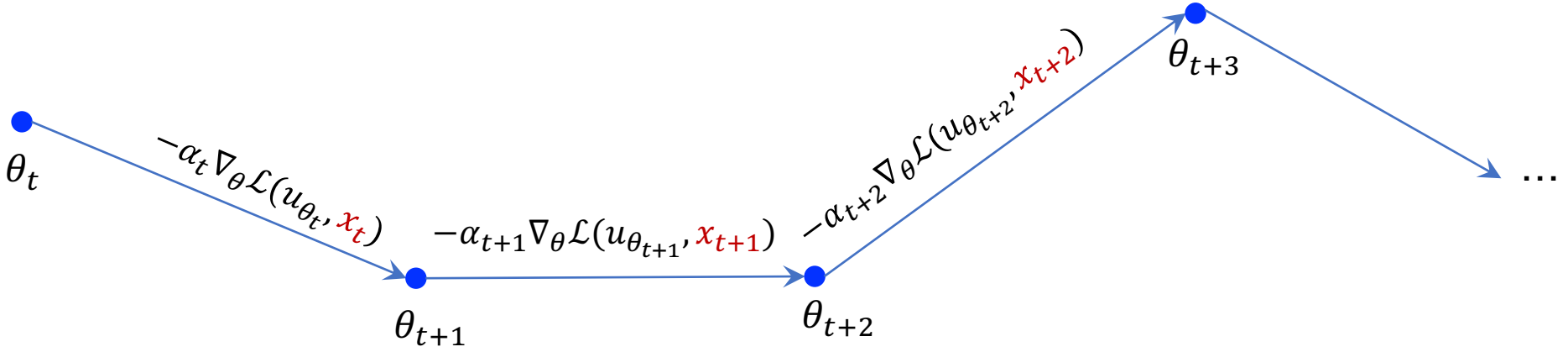
Intuitive Understanding



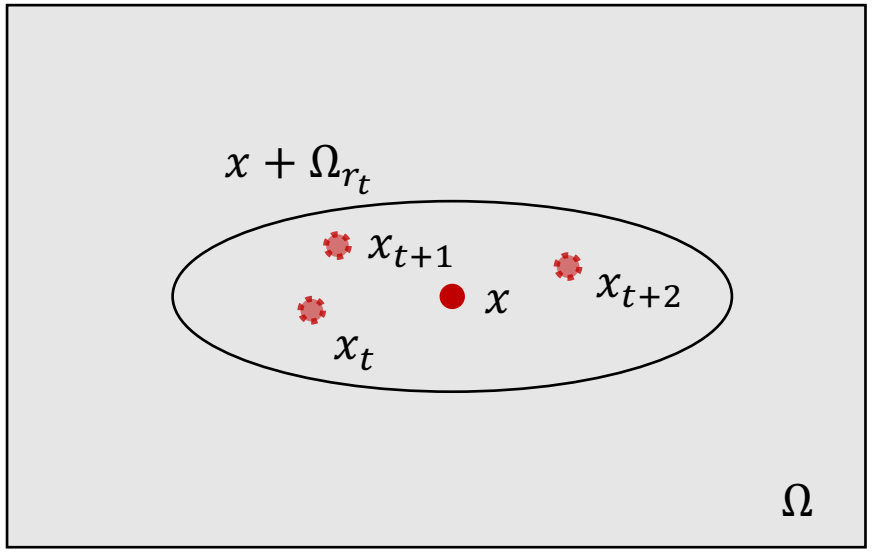
Point optimization: calculate gradient on the fixed collocation point in all iterations



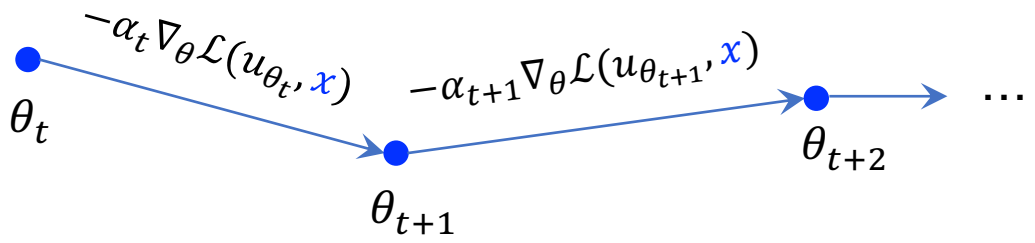
RoPINN: Approximate the region gradient by accumulating gradients from multiple iterations



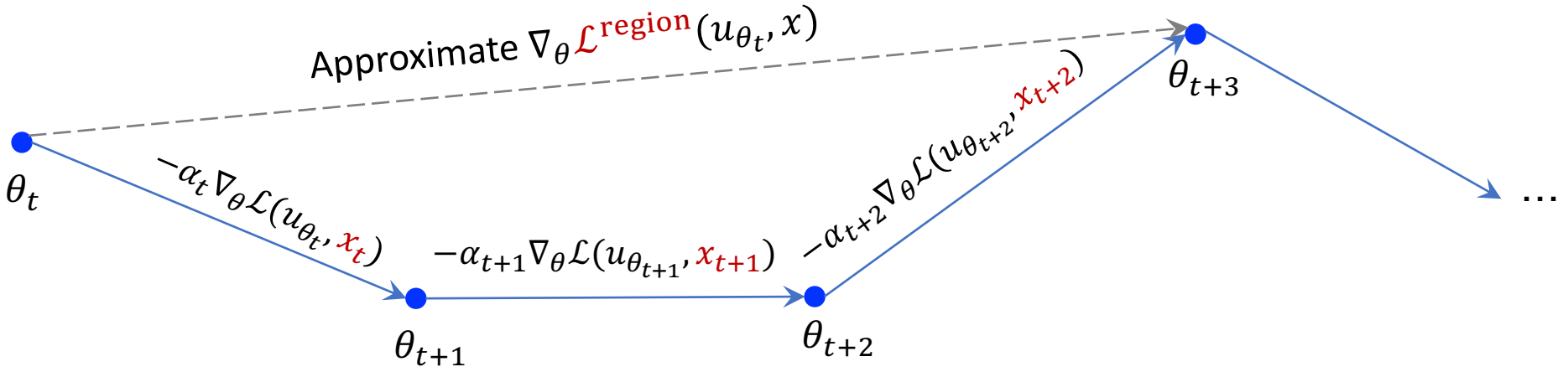
Intuitive Understanding



Point optimization: calculate gradient on the fixed collocation point in all iterations



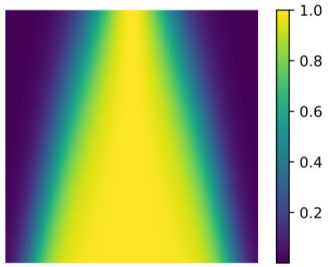
RoPINN: Approximate the region gradient by accumulating gradients from multiple iterations



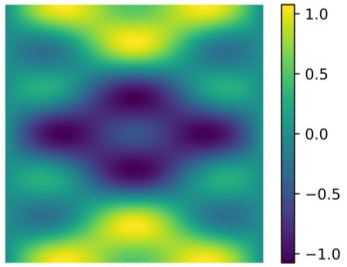
Experiments

Table 1: Summary of benchmarks. *Dimension* means the input space and *Derivative* is the highest derivative order.

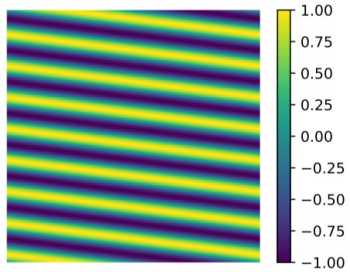
Benchmark	Dimension	Derivative	Property
1D-Reaction	1D+Time	1 (e.g. $\frac{\partial u}{\partial x}$)	Failure modes [24]
1D-Wave	1D+Time	2 (e.g. $\frac{\partial^2 u}{\partial x^2}$)	/
Convection	1D+Time	1 (e.g. $\frac{\partial u}{\partial x}$)	Failure modes [24]
PINNacle [12]	1D~5D+Time	1~2 (e.g. $\frac{\partial^2 u}{\partial x^2}$)	16 different tasks



(a) 1D-Reaction



(b) 1D-Wave



(c) Convection

Table 4: Details of datasets in PINNacle [12] (16 different PDEs included in our experiments), including the dimension of inputs, highest order of PDEs, number of train/test points and concrete equations. Here we only present the simplified PDE formalizations for intuitive understanding. More detailed descriptions of PDE type and coefficient meanings can be found in their paper [12].

PDE		Dimension	Order	N_{train}	N_{test}	Key Equations
Burges	1d-C	1D+Time	2	16384	12288	$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \nu \Delta \boldsymbol{u} = 0$
	2d-C	2D+Time	2	98308	82690	
Poisson	2d-C	2D	2	12288	10240	$-\Delta \boldsymbol{u} = 0$
	2d-CG	2D	2	12288	10240	$-\Delta \boldsymbol{u} + k^2 \boldsymbol{u} = f(x, y)$
	3d-CG	3D	2	49152	40960	$-\mu_i \Delta \boldsymbol{u} + k_i^2 \boldsymbol{u} = f(x, y, z), i = 1, 2$
	2d-MS	2D	2	12288	10329	$-\nabla(a(x) \nabla \boldsymbol{u}) = f(x, y)$
Heat	2d-VC	2D+Time	2	65536	49189	$\frac{\partial \boldsymbol{u}}{\partial t} - \nabla(a(x) \nabla \boldsymbol{u}) = f(x, t)$
	2d-MS	2D+Time	2	65536	49189	$\frac{\partial \boldsymbol{u}}{\partial t} - \frac{1}{(500\pi)^2} \boldsymbol{u}_{xx} - \frac{1}{\pi^2} \boldsymbol{u}_{yy} = 0$
	2d-CG	2D+Time	2	65536	49152	$\frac{\partial \boldsymbol{u}}{\partial t} - \Delta \boldsymbol{u} = 0$
NS	2d-C	2D	2	14337	12378	$\boldsymbol{u} \cdot \nabla \boldsymbol{u} + \nabla p - \frac{1}{Re} \Delta \boldsymbol{u} = 0, \nabla \cdot \boldsymbol{u} = 0$
	2d-CG	2D	2	14055	12007	
Wave	1d-C	1D+Time	2	12288	10329	$\boldsymbol{u}_{tt} - 4 \boldsymbol{u}_{xx} = 0$
	2d-CG	2D+Time	2	49170	42194	$\left[\nabla^2 - \frac{1}{c(x)} \frac{\partial^2}{\partial t^2}\right] u(x, t) = 0$
Chaotic	GS	2D+Time	2	65536	61780	$\boldsymbol{u}_t = \varepsilon_1 \Delta \boldsymbol{u} + b(1 - \boldsymbol{u}) - \boldsymbol{u} \boldsymbol{v}^2$ $\boldsymbol{v}_t = \varepsilon_2 \Delta \boldsymbol{v} - d \boldsymbol{v} + \boldsymbol{u} \boldsymbol{v}^2$
High-dim	PNd	5D	2	49152	67241	$-\Delta \boldsymbol{u} = \frac{\pi^2}{4} \sum_{i=1}^{n} \sin\left(\frac{\pi}{2} x_i\right)$
	HNd	5D+Time	2	65537	49152	$\frac{\partial \boldsymbol{u}}{\partial t} = k \Delta \boldsymbol{u} + f(x, t)$

- Five base models: PINN, FLS, QRes, PINNsFormer, KAN
- 19 different PDE solving tasks: 1D-Reaction, 1D-Wave, Convection and PINNacle

Main Results

Base Model	Objective	1D-Reaction			1D-Wave			Convection			PINNacle (16 tasks)	
		Loss	rMAE	rMSE	Loss	rMAE	rMSE	Loss	rMAE	rMSE	rMAE	rMSE
PINN [36]	Vanilla	2.0e-1	0.982	0.981	1.9e-2	0.326	0.335	1.6e-2	0.778	0.840	-	-
	gPINN	2.0e-1	0.978	0.978	2.8e-2	0.399	0.399	3.1e-2	0.890	0.935	18.8%	18.8%
	vPINN	2.3e-1	0.985	0.982	7.3e-3	0.162	0.173	1.1e-2	0.663	0.743	25.0%	25.0%
	RoPINN	**4.7e-5**	**0.056**	**0.095**	**1.5e-3**	**0.063**	**0.064**	**1.0e-2**	**0.635**	**0.720**	**93.8%**	**100.0%**
	Promotion	99%	94%	90%	92%	80%	80%	25%	18%	14%		
QRes [3]	Vanilla	2.0e-1	0.979	0.977	9.8e-2	0.523	0.515	4.2e-2	0.925	0.959	-	-
	gPINN	2.1e-2	0.984	0.984	1.3e-1	0.785	0.781	1.6e-1	1.111	1.222	12.5%	12.5%
	vPINN	2.2e-2	0.999	1.000	1.0e-1	0.709	0.721	5.5e-2	0.941	0.966	12.5%	12.5%
	RoPINN	**9.0e-6**	**0.007**	**0.013**	**1.7e-2**	**0.309**	**0.321**	**1.2e-2**	**0.819**	**0.870**	**81.3%**	**81.3%**
	Promotion	99%	99%	99%	83%	41%	38%	71%	11%	9%		
FLS [50]	Vanilla	2.0e-1	0.984	0.985	3.6e-3	0.102	0.119	1.2e-2	0.674	0.771	-	-
	gPINN	2.0e-1	0.978	0.979	9.2e-2	0.500	0.489	3.8e-1	0.913	0.949	12.5%	18.8%
	vPINN	2.1e-1	1.000	0.994	2.1e-3	0.069	0.069	1.1e-2	0.688	0.765	25.0%	18.8%
	RoPINN	**2.2e-5**	**0.022**	**0.039**	**1.5e-4**	**0.016**	**0.017**	**9.6e-4**	**0.173**	**0.197**	**81.3%**	**87.5%**
	Promotion	99%	98%	96%	96%	84%	86%	99%	74%	74%		
PINNs-Former [58]	Vanilla	3.0e-6	0.015	0.030	1.4e-2	0.270	0.283	3.7e-5	0.023	0.027	-	-
	gPINN	1.5e-6	0.009	0.018	OOM	OOM	OOM	3.7e-2	0.914	0.950	0.0%	0.0%
	vPINN	1.6e-4	0.065	0.124	4.5e-2	0.411	0.400	5.1e-5	0.016	0.022	0.0%	0.0%
	RoPINN	**1.0e-6**	**0.007**	**0.017**	**6.5e-3**	**0.165**	**0.172**	**1.2e-5**	**0.005**	**0.006**	**100.0%**	**100%**
	Promotion	66%	53%	43%	54%	39%	39%	68%	78%	78%		
KAN [28]	Vanilla	7.3e-5	0.031	0.061	9.2e-2	0.499	0.489	5.8e-2	0.922	0.954	-	-
	gPINN	2.9e-4	0.030	0.061	2.6e-1	1.131	1.110	1.2e-1	1.006	1.041	31.3%	31.3%
	vPINN	2.1e-1	0.998	0.996	9.0e-2	0.498	0.487	2.5e-2	0.853	0.853	43.8%	43.8%
	RoPINN	**4.9e-5**	**0.026**	**0.051**	**9.6e-3**	**0.177**	**0.191**	**2.2e-2**	**0.805**	**0.801**	**100%**	**93.8%**
	Promotion	33%	16%	16%	89%	65%	61%	62%	13%	16%		

- Two typical baselines:

  gPINN (high-order regularization)

  vPINN (variational formalization)

- ✓ **RoPINN consistently boost all five PINN base models in all 19 PDEs.**

- ✓ **RoPINN helps mitigate the "PINN failure modes"** (see results of 1D-Reaction and Convection).

Algorithm Analysis: Region Size

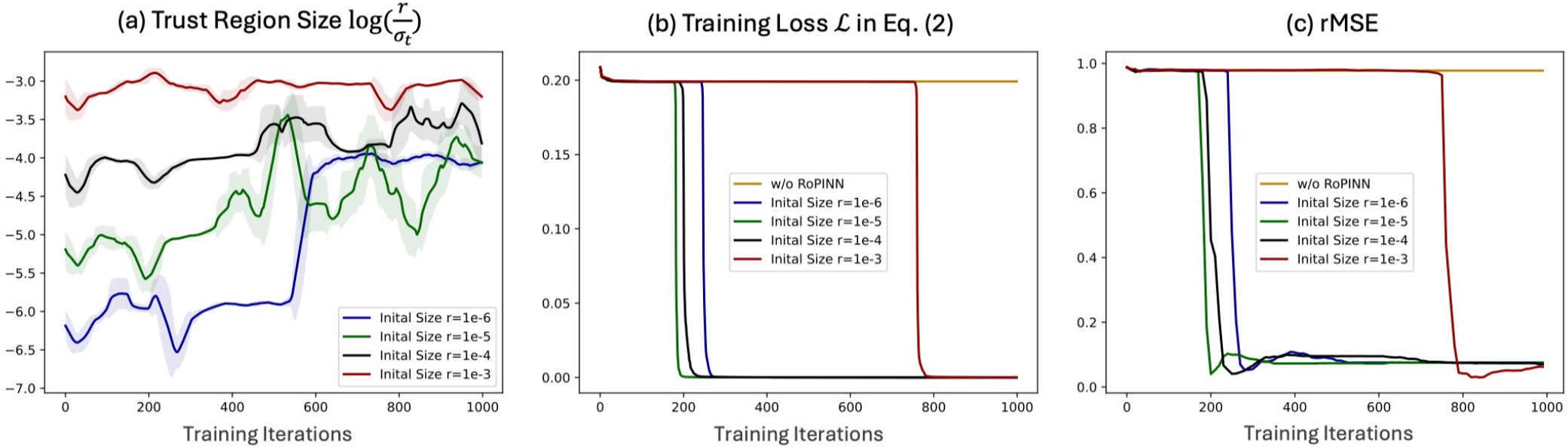


Figure 2: Optimization of canonical PINN [36] on the 1D-Reaction under different region sizes.

- ✓ **Adaptively find the "balance point":** Even though we initialize the region size as distinct values, RoPINN will progressively adjust the trust region size to similar values during training.
- ✓ **Affect convergence:** If r is initialized as a value closer to the balance point, the training will converge faster. Too large a region size will decrease the convergence speed due to the optimization noise.

Algorithm Analysis: Sampling Points

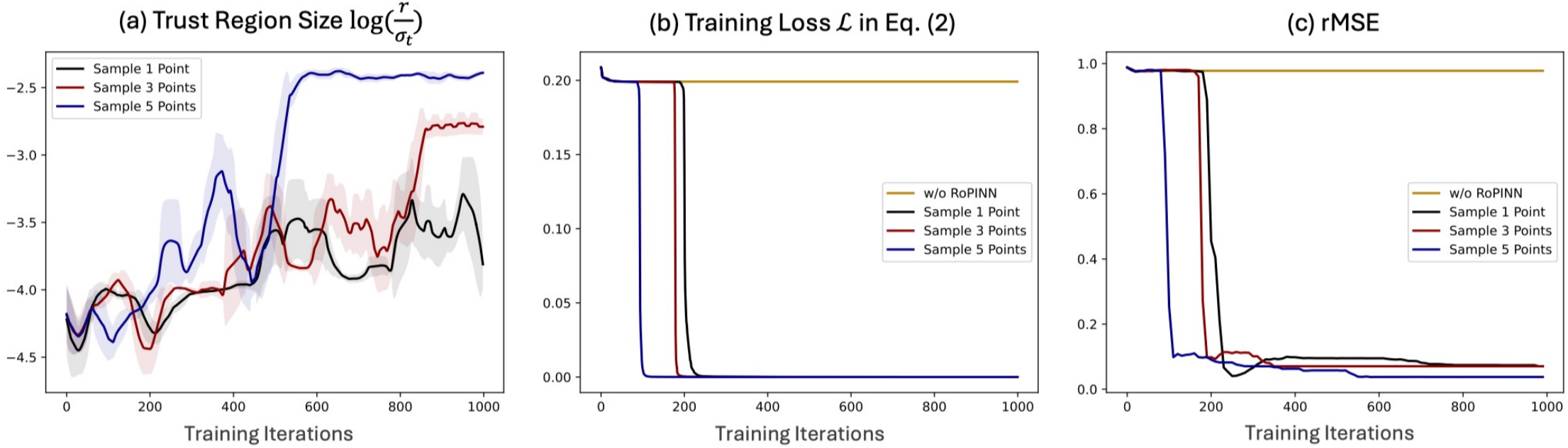


Figure 3: Optimization of canonical PINN [36] on the 1D-Reaction under different sample points.

Sampling more points in each region will bring a lower gradient estimation error,

which will lead to **larger region size, better convergence and final performance**.

Algorithm Analysis: Loss Landscape

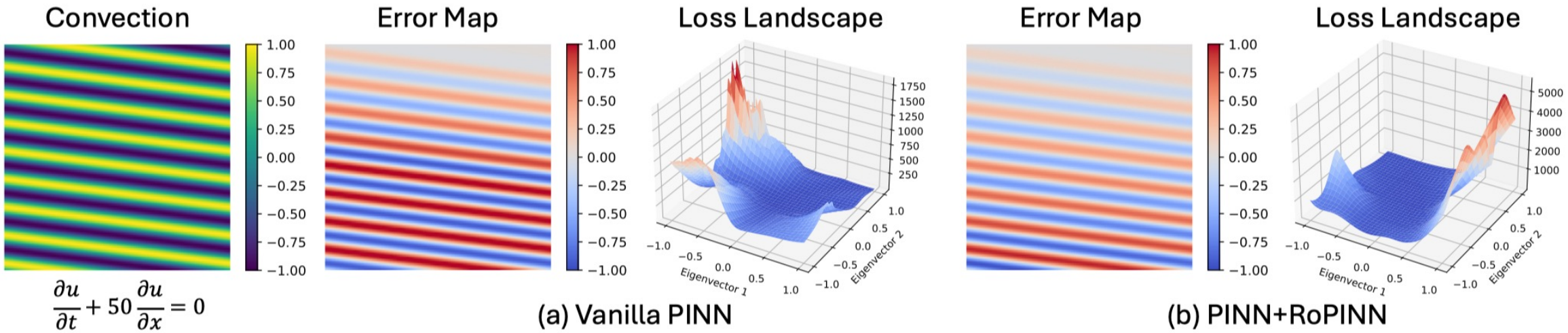


Figure 6: Loss landscape of RoPINN and vanilla PINNs on the Convection equation. *Error Map* refers to the distance between model prediction and the accurate solution, i.e. $(u_\theta - u)$.

"PINN failure modes" are not caused by limited model capacity but by **a hard-to-optimize loss landscape.**

Empowered by RoPINN, the loss landscape of PINN is significantly smoothed.

Krishnapriyan, Aditi S. et al. Characterizing possible failure modes in physics-informed neural networks. ***NeurIPS*** 2021.

More Showcases

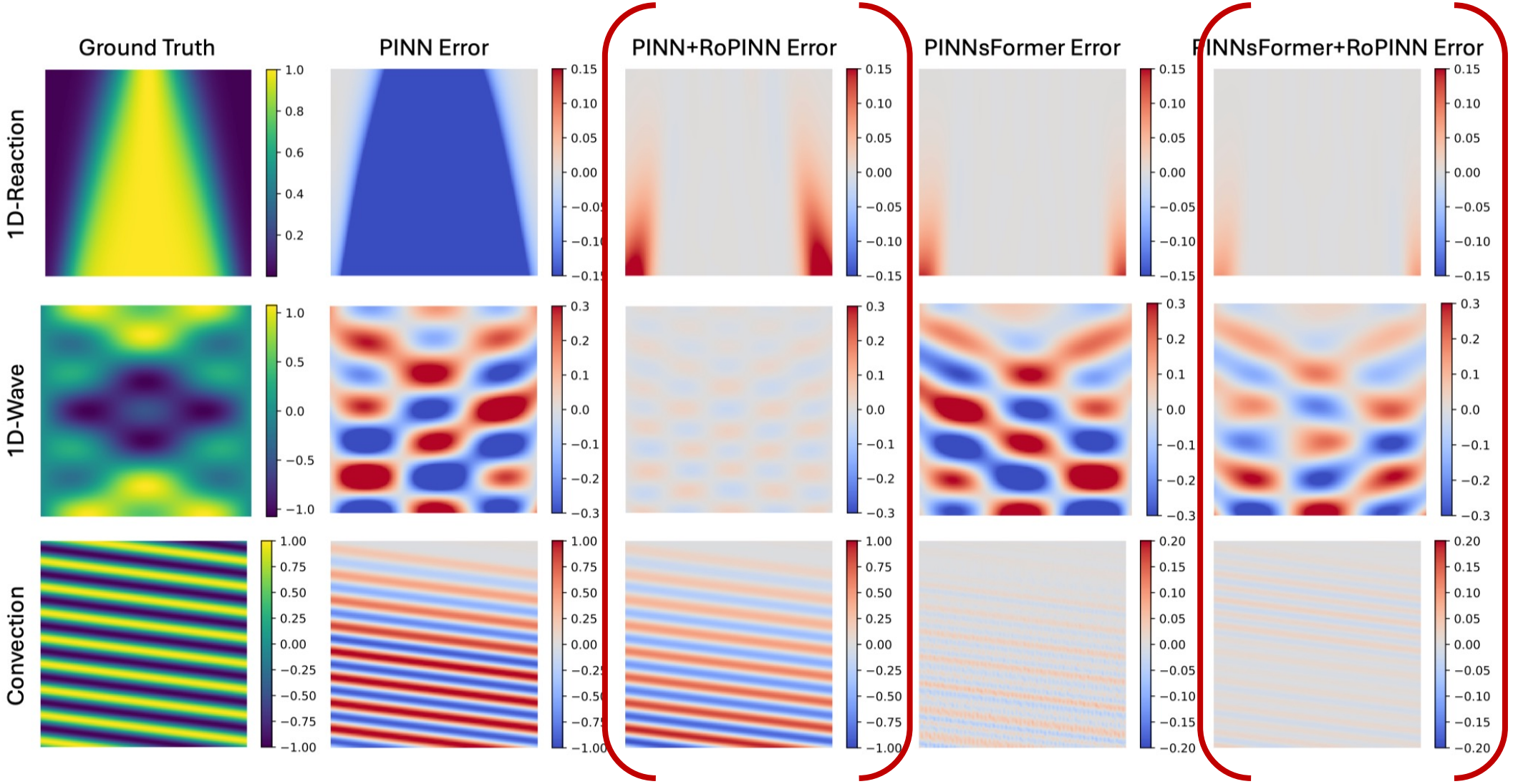
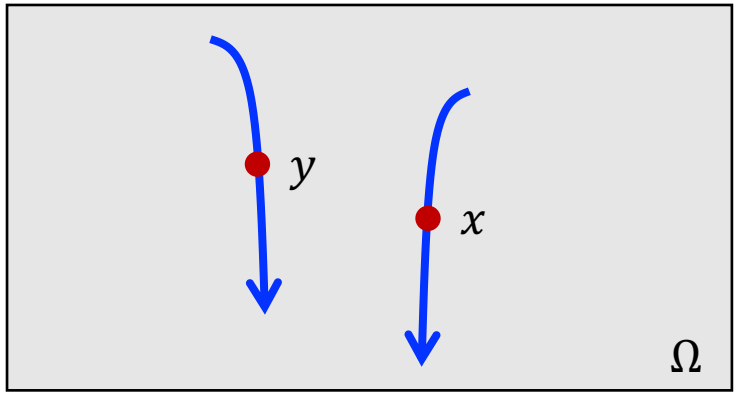
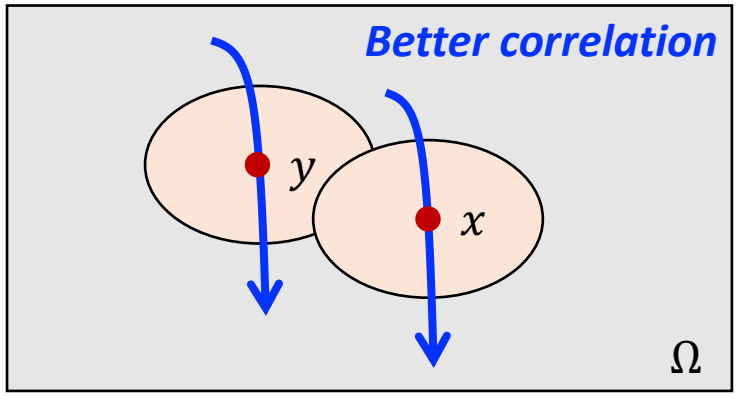


Figure 9: Showcases of RoPINN on the first three datasets based on PINN and PINNsFormer.

An Insight from the Proof Process





$$\mathcal{E}_{\text{gen}} \leq \left(1 - \frac{|\Omega_r|}{|\Omega|}\right) \frac{2L^2}{|\mathcal{S}|} \sum_{t=1}^{T} \alpha_t$$

*Proof Sketch (**The Importance of Gradient Correlation**)*

(1) $\mathcal{E}_{\text{gen}}$ can be bounded by a term relating to the expectation of distance between parameter θ optimized from different training sets.

(2) Region optimization paradigm brings a more "consistent" gradient direction than point optimization at each iteration.

What will happen if different collocation points have irrelevant gradients?

NEURAL INFORMATION PROCESSING SYSTEMS

ProPINN: Demystifying Propagation Failures in Physics-Informed Neural Networks

Haixu Wu, Yuezhou Ma, Hang Zhou, Huikun Weng, Jianmin Wang, Mingsheng Long✉
School of Software, BNRist, Tsinghua University, China
{wuhaixu98}@gmail.com, {mayz20,zhou-h23,wenghk22}@mails.tsinghua.edu.cn
{jimwang,mingsheng}@tsinghua.edu.cn


Haixu Wu


Yuezhou Ma


Hang Zhou


Huikun Weng

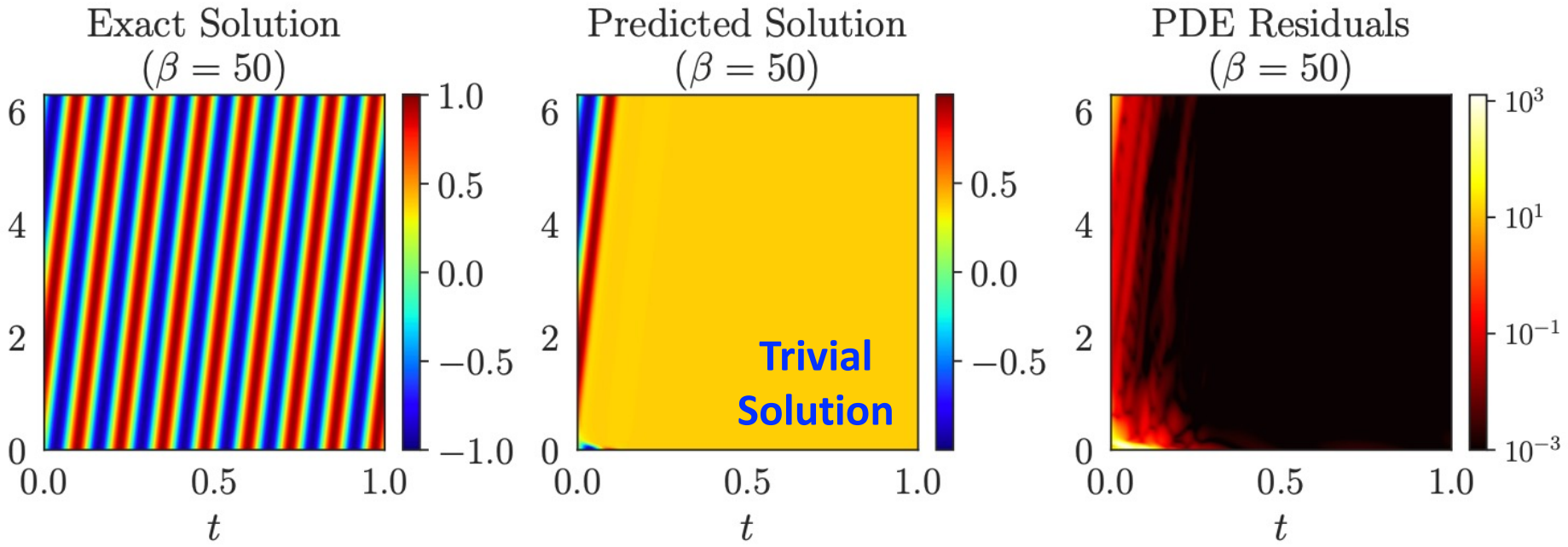

Jianmin Wang


Mingsheng Long

Paper Link: *https://arxiv.org/pdf/2502.00803*

Code Link: *https://github.com/thuml/ProPINN*

Peculiar Phenomena in PINN Optimization

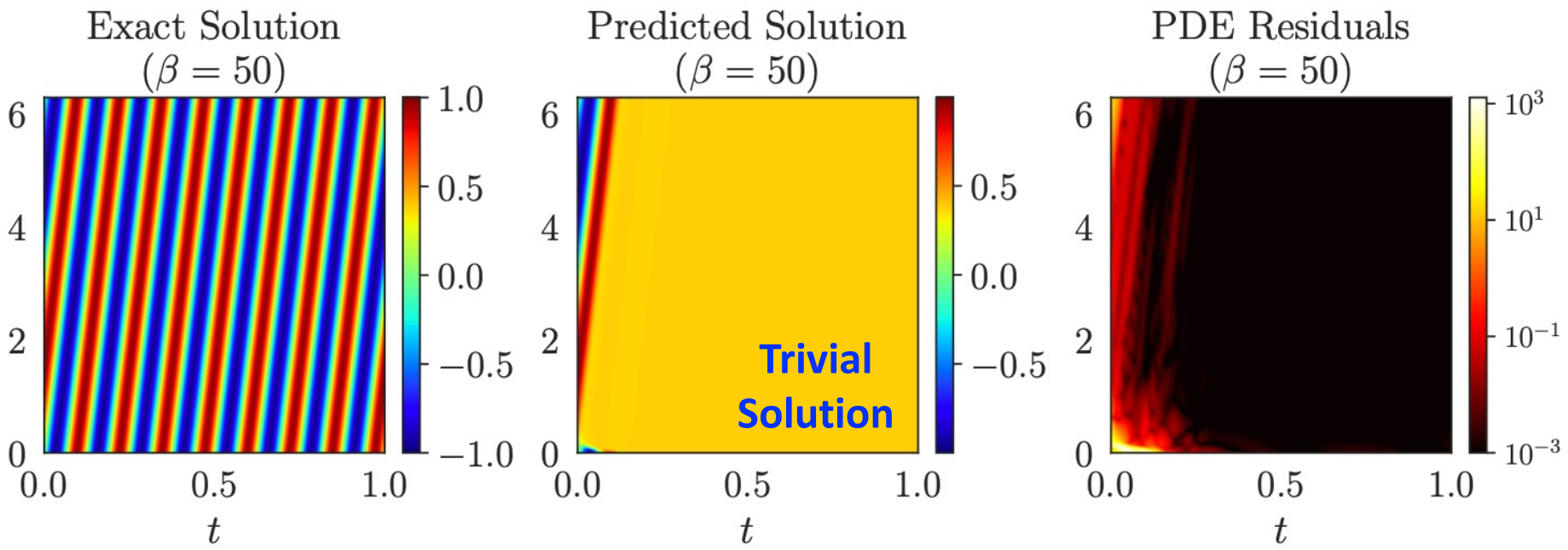


PINN loss values (PDE residuals) are very close to zero,

but the prediction error is still quite large.

Daw, A. et al. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling, ***ICML*** *2023*

Peculiar Phenomena in PINN Optimization

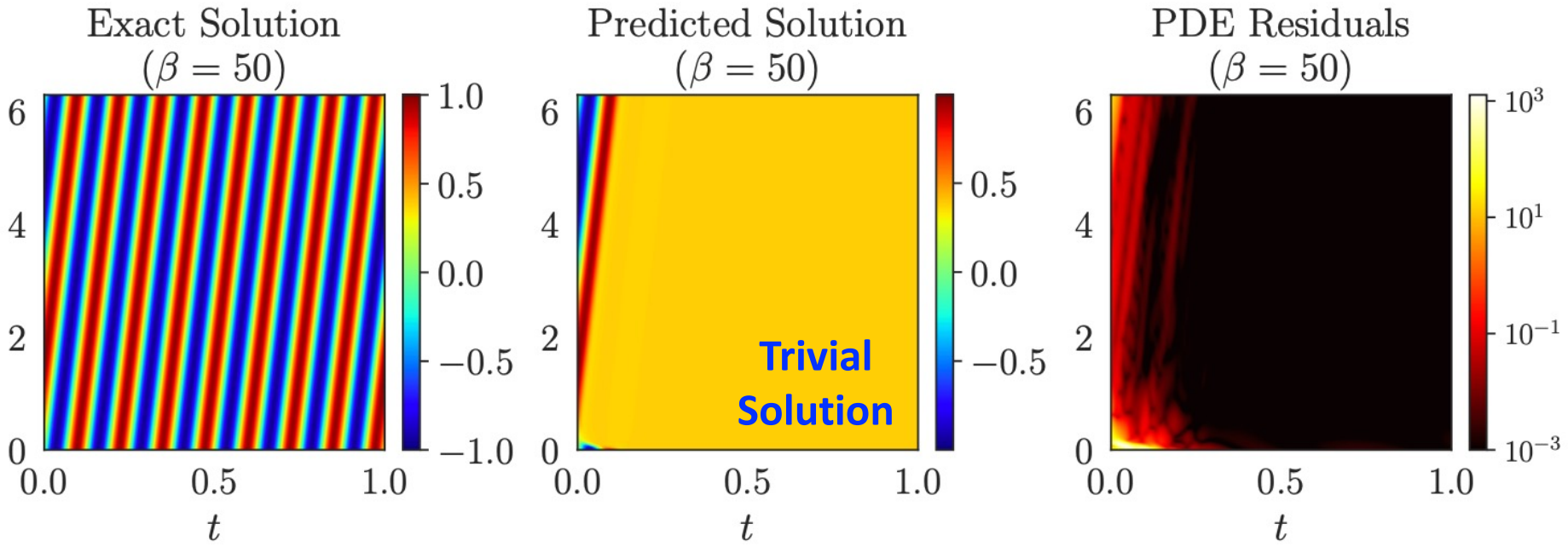


Only IC and BC have the correct supervision of solution values.

Propagation hypothesis: "In order for PINNs to avoid converging to trivial solutions at interior points, the correct solution must be **propagated from the initial/boundary points** to the interior points."

Daw, A. et al. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling, ***ICML** 2023*

Why Propagation Failure Happens?

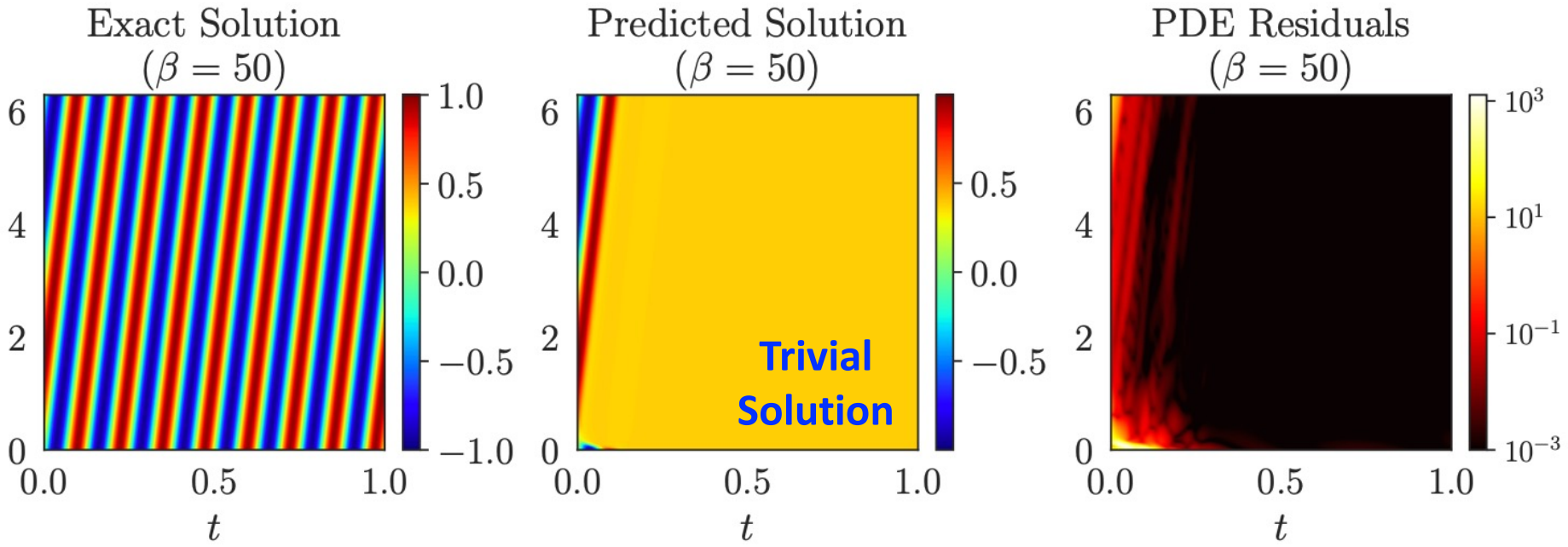


Explanation from Daw et al.:

Some collocation points start converging to trivial solutions before the correct solution from initial/boundary points is able to reach them.

Daw, A. et al. Mitigating Propagation Failures in Physics-informed Neural Networks using Retain-Resample-Release (R3) Sampling, ***ICML*** *2023*

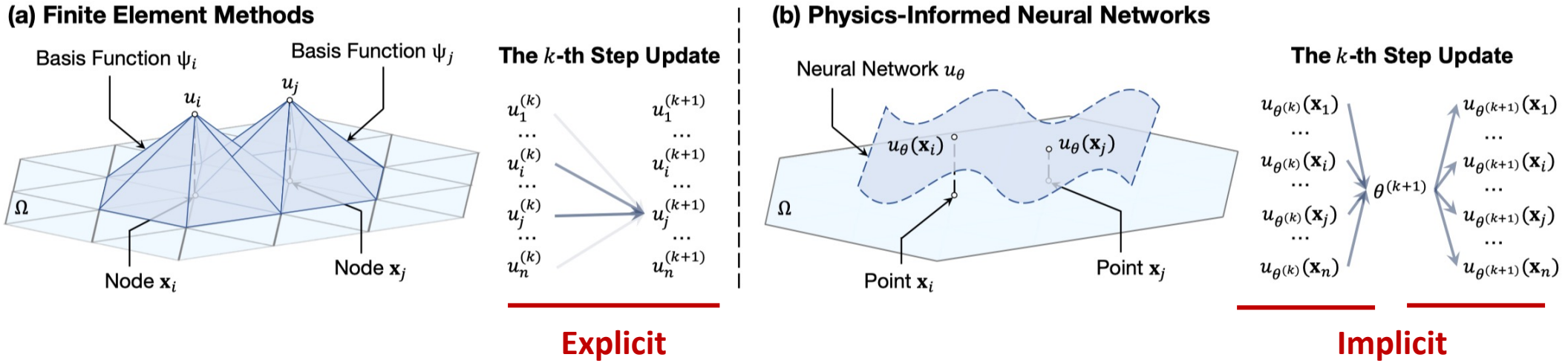
Why Propagation Failure Happens?



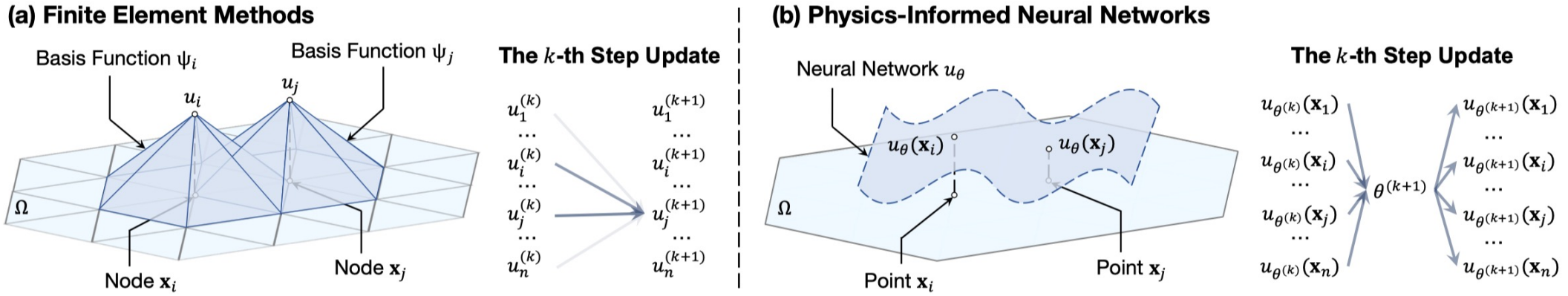
A comparison between PINNs and FEMs:

- Start from a trivial estimation and the interior areas hold incorrect output supervision in the beginning
- PINNs suffer from propagation failure, but FEMs do not.

PINNs v.s. FEMs



PINNs v.s. FEMs



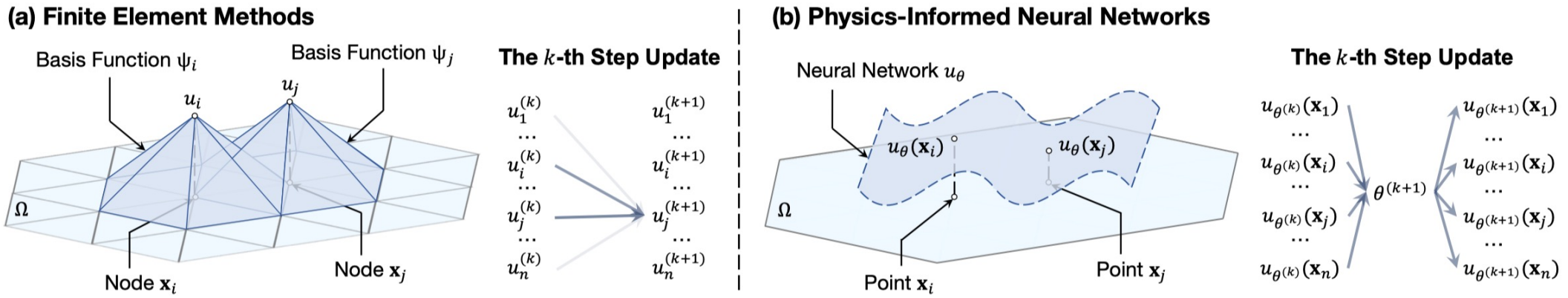
Theorem 3.1 (Propagation in FEMs). *[10] Suppose that FEMs discretize Ω into computation meshes with n nodes $\{\mathbf{x}_i\}_{i=1}^n$ and approximate the PDE solution by optimizing coefficients of basis functions $\{\Psi_i\}_{i=1}^n$, which are defined as region linear interpolation. Denote the coefficient of basis function Ψ_i as u_i, which is also the solution value of the i-th node. With the Jacobi iterative method for solution value update, the interaction among solution values $\{u_i\}_{i=1}^n$ at the k-th step is:*

$$u_j^{(k+1)} = \frac{1}{D(\Psi_j, \Psi_j)} \left(b_j - \sum_{i \neq j} D(\Psi_i, \Psi_j) u_i^{(k)} \right), \tag{3}$$

Active propagation

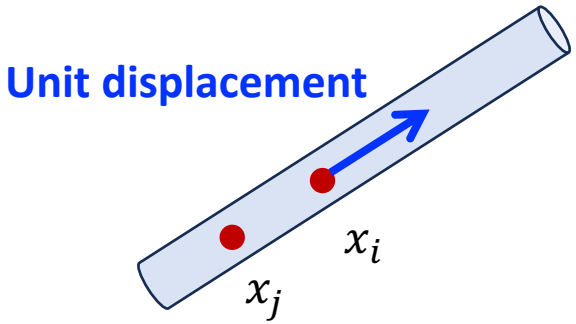
where $\{b_j\}_{j=1}^n$ are constants related to external force. $D(\cdot, \cdot)$ is a variational version of PDE equation $\mathcal{F}(\cdot)$, which presents non-zero values only for overlapped basis functions.

Understanding “Propagation”



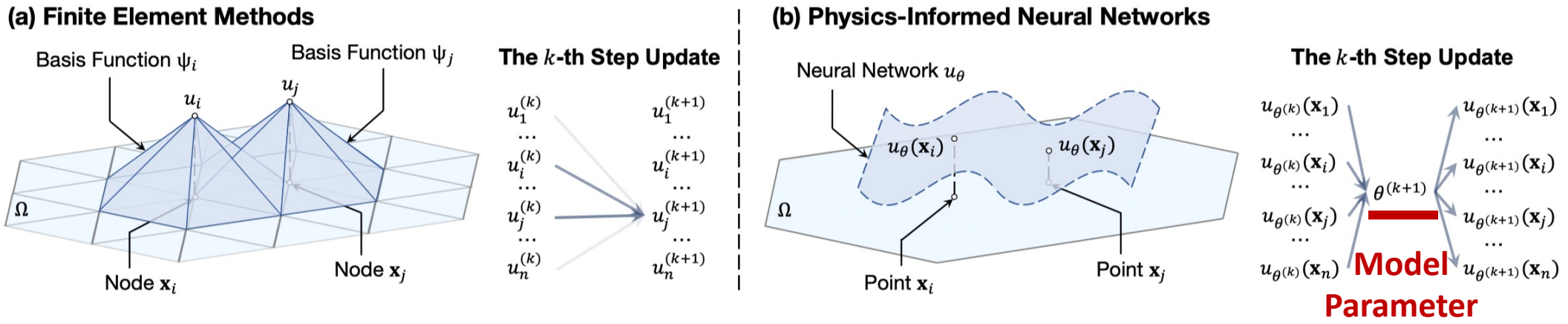
$$u_j^{(k+1)} = \frac{1}{D(\Psi_j, \Psi_j)} \left(b_j - \sum_{i \neq j} D(\Psi_i, \Psi_j) u_i^{(k)} \right), \tag{3}$$

Stiffness matrix: Force on the j-th node to make region balance when the i-th node has a unit displacement.



What does the “stiffness matrix” look like in PINNs?

Understanding "Propagation"



Definition 3.4 (**Propagation failure in PINN**). *In spirit of the physics meaning of Eq. (3), we define the "stiffness" coefficient between* $\mathbf{x}$ *and* $\mathbf{x}'$ *for PINN* u_θ *as the "slope" w.r.t. the parameter change:*

$$D_{PINN}(\mathbf{x}, \mathbf{x}') = \lim_{\lambda \to 0} \frac{\left\| u_\theta(\mathbf{x}') - u_{\theta - \lambda \frac{\partial u_\theta}{\partial \theta}\Big|_{\mathbf{x}}}(\mathbf{x}') \right\|}{\lambda}, \tag{4}$$

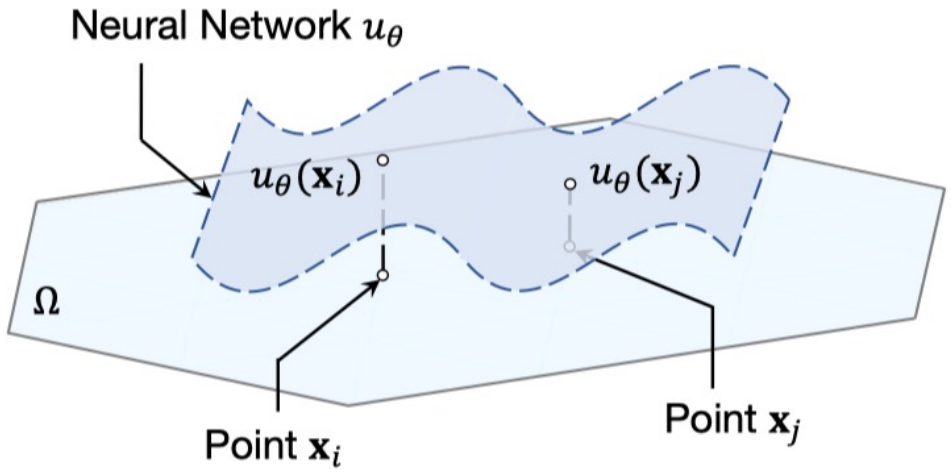
which measures the impact on model output at $\mathbf{x}'$ *after updating PINN with a unit step at* $\mathbf{x}$. *This formula is analogous to applying a unit force at* $\mathbf{x}$ *and observing a displacement at* $\mathbf{x}'$. *If* $\mathbf{x}$ *and* $\mathbf{x}'$ *are adjacent and* $D_{PINN}(\mathbf{x}, \mathbf{x}')$ *is less than an empirically defined threshold* ϵ, *we consider that propagation failure has occurred between* $\mathbf{x}$ *and* $\mathbf{x}'$.

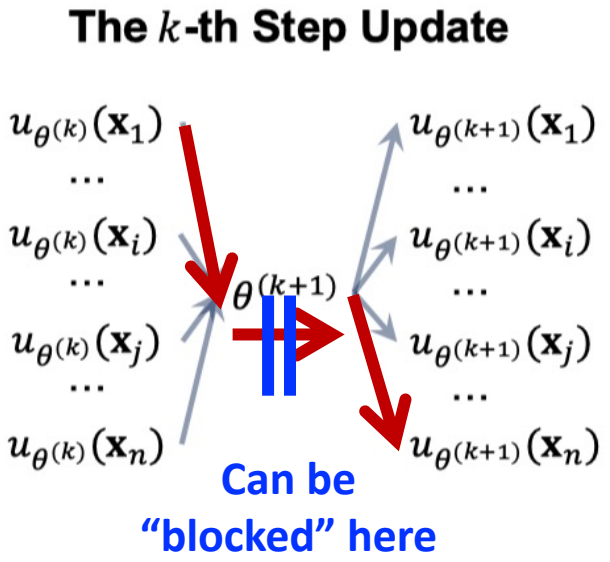
The effect on x' when we make an optimization step based on gradient at x.

Lower Gradient Correlation

$$D_{PINN}(\mathbf{x}, \mathbf{x}') = \lim_{\lambda \to 0} \frac{\left\| u_\theta(\mathbf{x}') - u_{\theta - \lambda \frac{\partial u_\theta}{\partial \theta}\Big|_{\mathbf{x}}}(\mathbf{x}') \right\|}{\lambda}, \tag{4}$$

Lower D-PINN, weak propagation.





Lower Gradient Correlation

$$D_{PINN}(\mathbf{x}, \mathbf{x}') = \lim_{\lambda \to 0} \frac{\left\| u_\theta(\mathbf{x}') - u_{\theta - \lambda \frac{\partial u_\theta}{\partial \theta}\Big|_{\mathbf{x}}}(\mathbf{x}') \right\|}{\lambda}, \tag{4}$$

Lower D-PINN, weak propagation.

Theorem 3.6 (Gradient correlation). *Given a PINN u_θ and adjacent points $\mathbf{x}, \mathbf{x}' \in \Omega$, the necessary and sufficient condition of propagation failure between $\mathbf{x}$ and $\mathbf{x}'$ is a small gradient correlation, which is formally defined as follows*

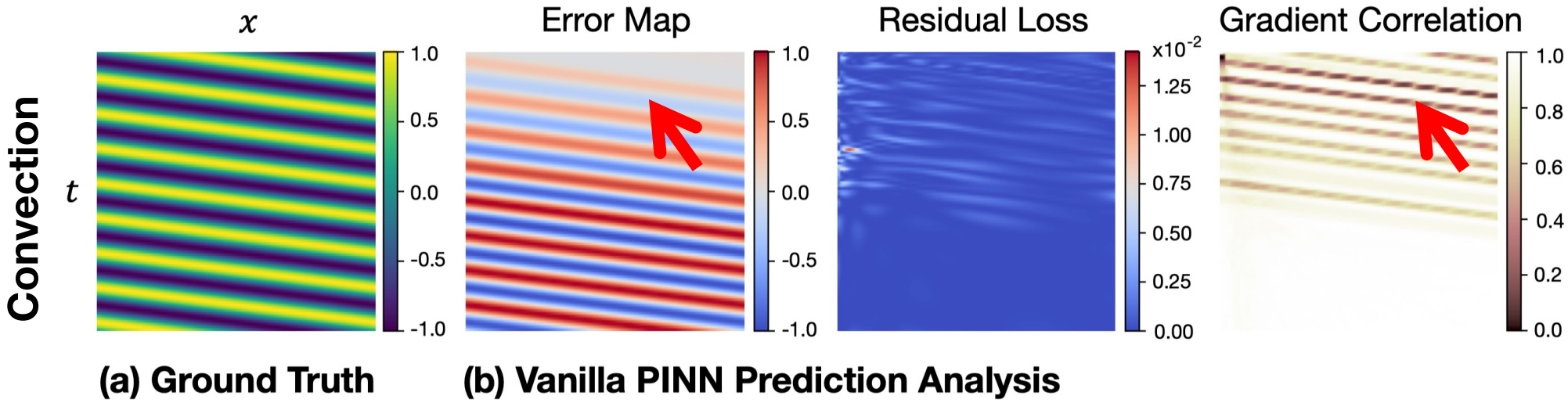
$$G_{u_\theta}(\mathbf{x}, \mathbf{x}') = \left\| \left\langle \frac{\partial u_\theta}{\partial \theta}\Big|_{\mathbf{x}}, \frac{\partial u_\theta}{\partial \theta}\Big|_{\mathbf{x}'} \right\rangle \right\|. \tag{5}$$

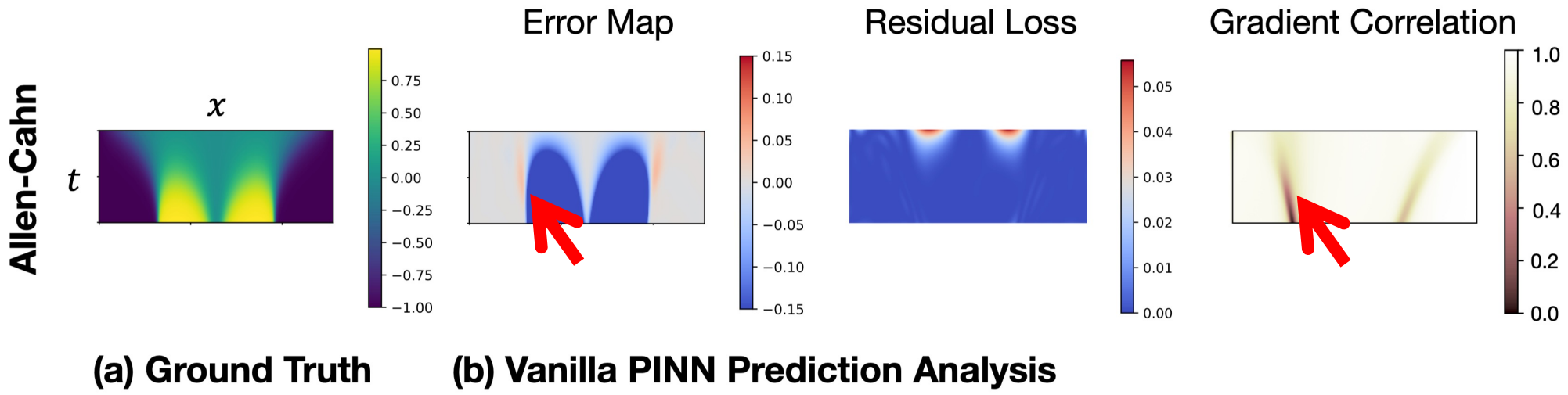
A lower gradient correlation will cause non-active propagation.

Also answers why PINNs cannot benefit from large models:

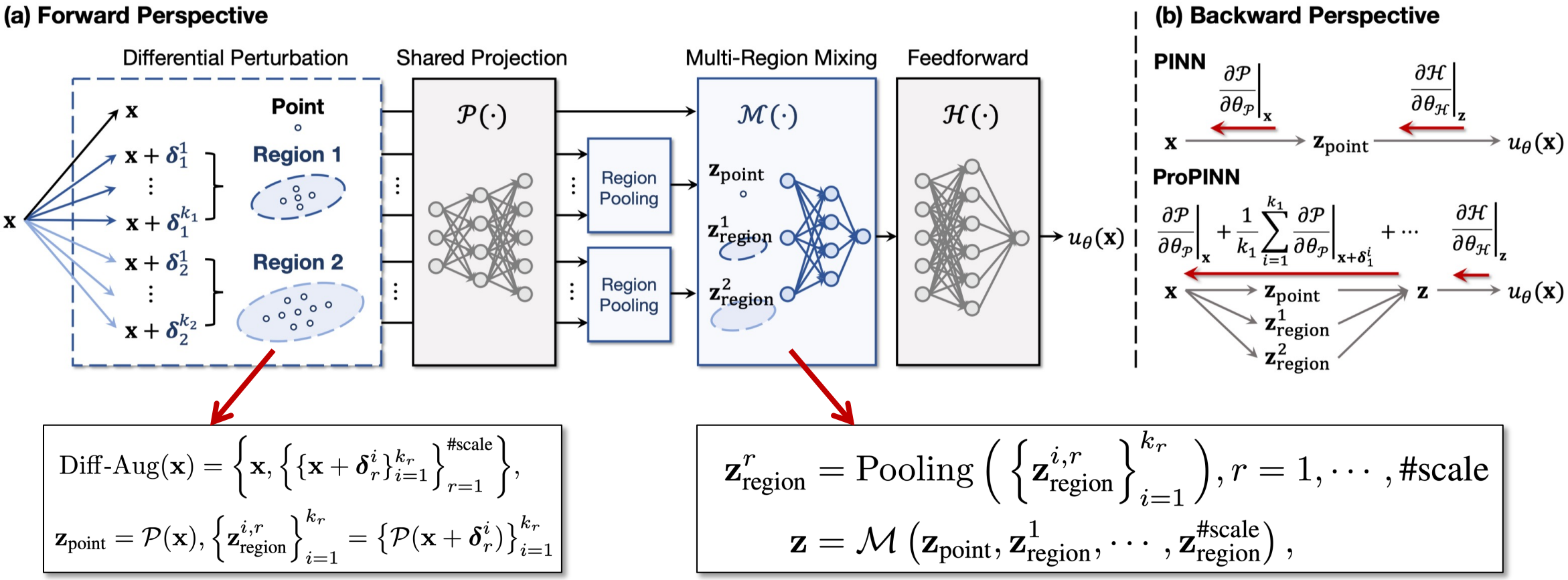
A larger parameter space is more likely to cause orthogonal gradients.

Lower Gradient Correlation





ProPINN with Active Propagation



$$\text{Diff-Aug}(\mathbf{x}) = \left\{\mathbf{x}, \left\{\{\mathbf{x}+\boldsymbol{\delta}_r^i\}_{i=1}^{k_r}\right\}_{r=1}^{\#\text{scale}}\right\},$$

$$\mathbf{z}_{\text{point}} = \mathcal{P}(\mathbf{x}), \left\{\mathbf{z}_{\text{region}}^{i,r}\right\}_{i=1}^{k_r} = \left\{\mathcal{P}(\mathbf{x}+\boldsymbol{\delta}_r^i)\right\}_{i=1}^{k_r}$$

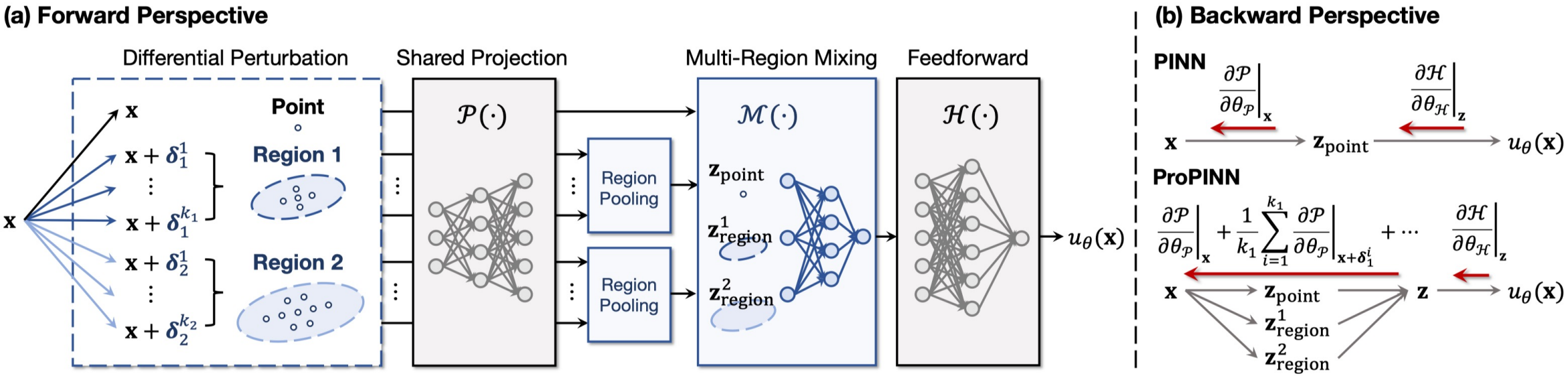
$$\mathbf{z}_{\text{region}}^r = \text{Pooling}\left(\left\{\mathbf{z}_{\text{region}}^{i,r}\right\}_{i=1}^{k_r}\right), r = 1, \cdots, \#\text{scale}$$

$$\mathbf{z} = \mathcal{M}\left(\mathbf{z}_{\text{point}}, \mathbf{z}_{\text{region}}^1, \cdots, \mathbf{z}_{\text{region}}^{\#\text{scale}}\right),$$

Ensure an explicit interaction among multiple collocation points within a region.

Transformer-based models, such as PINNsFormer or SetPINN, can also enable this, but are less efficient.

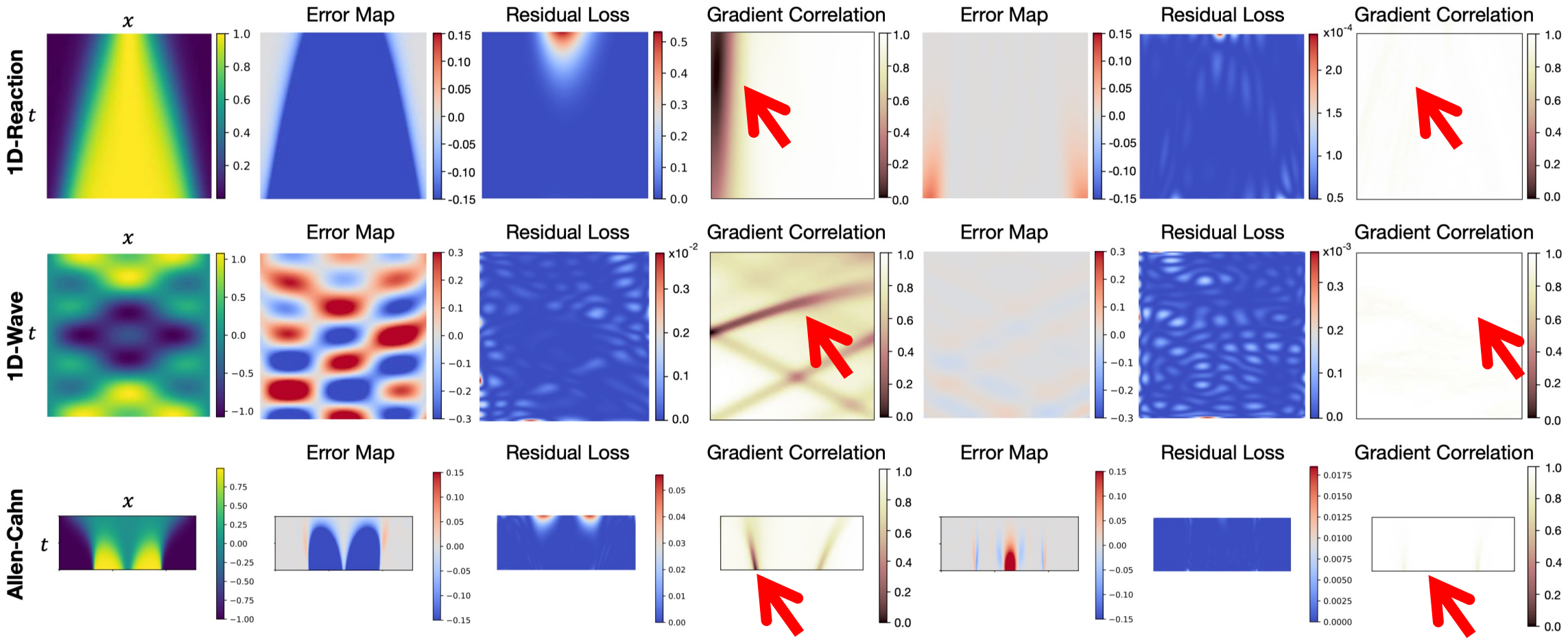
ProPINN with Active Propagation



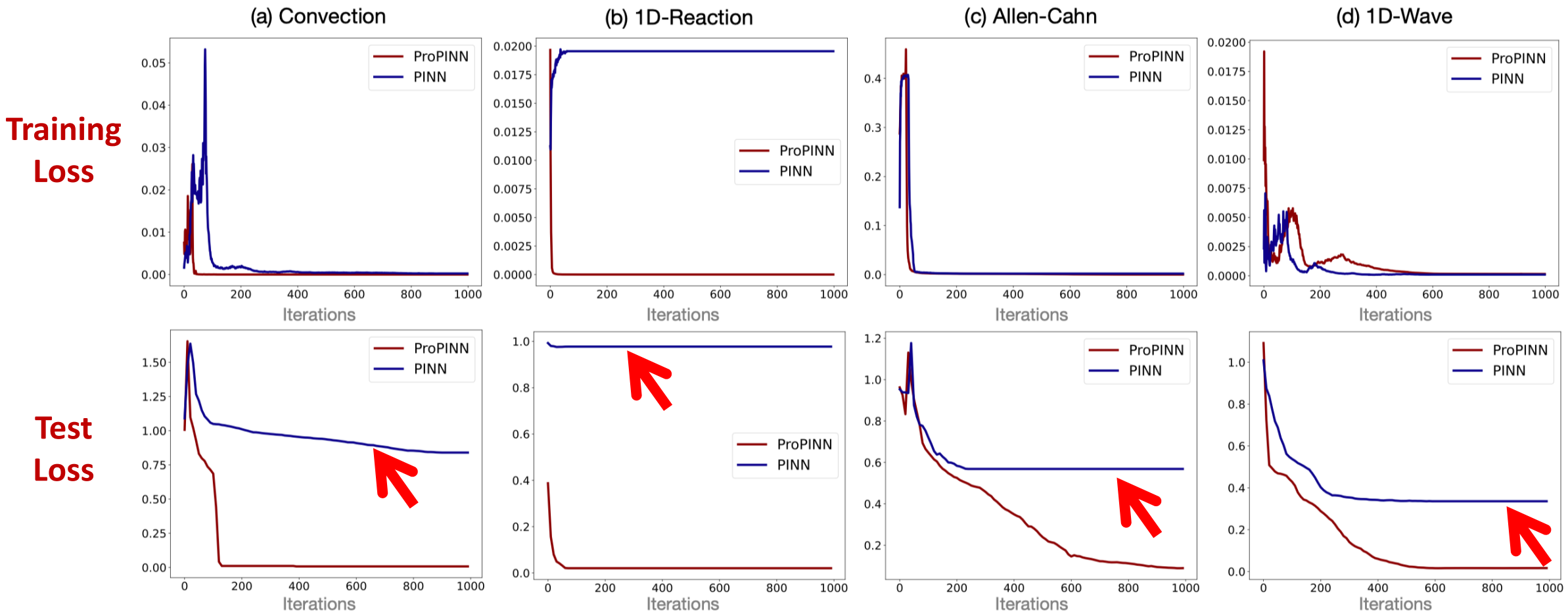
Theorem 3.8 (Gradient correlation improvement). *Under Assumption 3.7 with region size R, given k perturbations $\{\boldsymbol{\delta}_i\}_{i=1}^{k}$ with $\|\boldsymbol{\delta}_i\| \leq \frac{R}{3}$ and defining $u_\theta^{region}(\mathbf{x}) = u_\theta(\mathbf{x}) + \frac{\sum_{i=1}^{k} u_\theta(\mathbf{x}+\boldsymbol{\delta}_i)}{k}$, then $\forall \mathbf{x}, \mathbf{x}' \in \Omega$, if $\|\mathbf{x} - \mathbf{x}'\| \leq \frac{R}{3}$, we have $G_{u_\theta}(\mathbf{x}, \mathbf{x}') \leq G_{u_\theta^{region}}(\mathbf{x}, \mathbf{x}')$.*

Larger gradient correlation, better propagation.

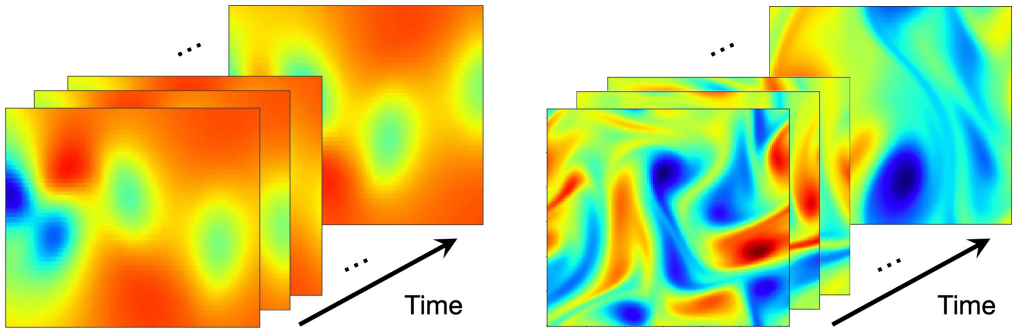
Visualization Comparison



Training Dynamics



Complex Dynamics



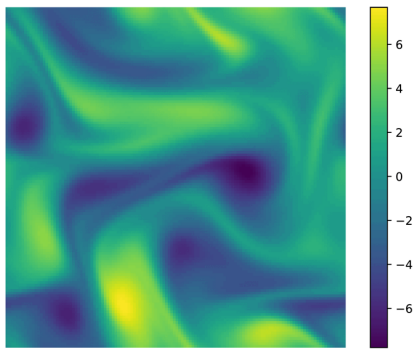
(a) Karman Vortex (b) Fluid Dynamics

Complex time-dependent

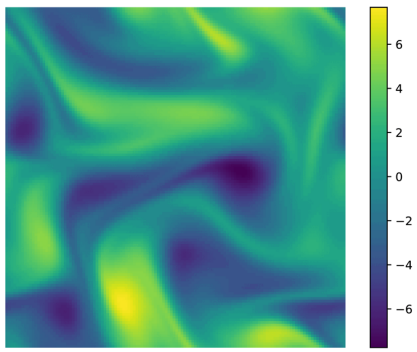
2D fluid dynamics

rRMSE = 0.2172 (21% error reduction)

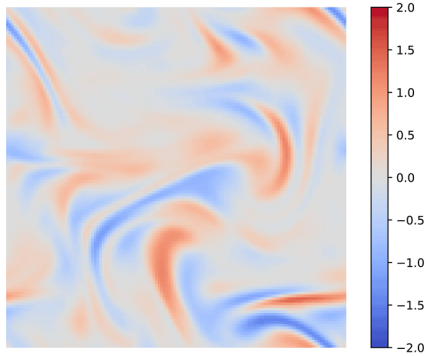
rRMSE = 0.2765 (Second Best)



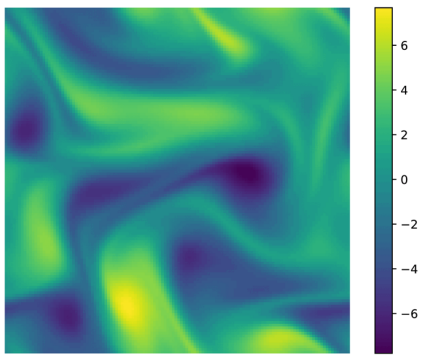
Ground Truth



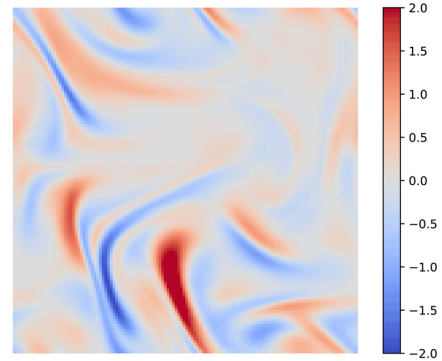
ProPINN Prediction



ProPINN Error



FLS Prediction

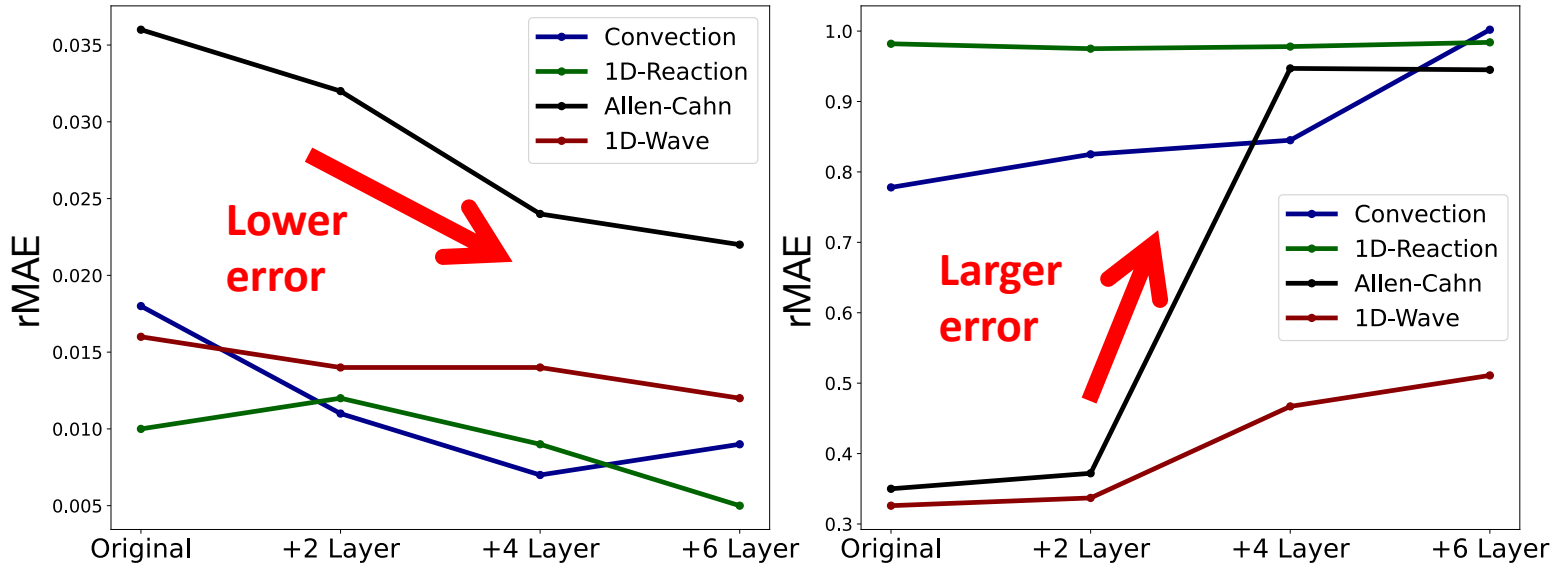


FLS Error

Scaling Performance

$$G_{u_\theta}(\mathbf{x}, \mathbf{x}') = \left\| \left\langle \left. \frac{\partial u_\theta}{\partial \theta} \right|_{\mathbf{x}}, \left. \frac{\partial u_\theta}{\partial \theta} \right|_{\mathbf{x}'} \right\rangle \right\|.$$

A larger parameter space is more likely to cause orthogonal gradients.



(a) ProPINN (b) Vanilla PINN

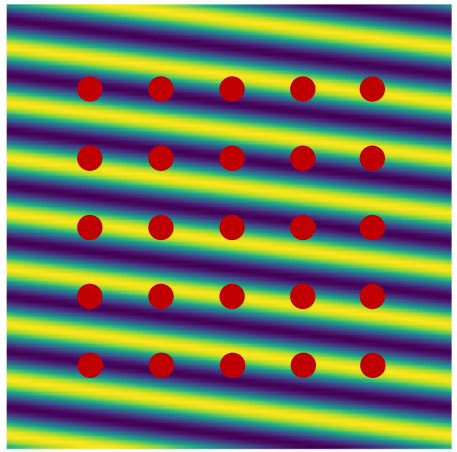
ProPINN releases the scaling capability of the neural network. Larger model, better performance.

Physics-Informed Neural Networks (PINNs)

$$\mathcal{L}(u_\theta) = \frac{\lambda_\Omega}{N_\Omega}\sum_{i=1}^{N_\Omega} \|\mathcal{F}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\Omega_0}}{N_{\Omega_0}}\sum_{i=1}^{N_{\Omega_0}} \|\mathcal{I}(u_\theta)(\boldsymbol{x}_i)\|^2 + \frac{\lambda_{\partial\Omega}}{N_{\partial\Omega}}\sum_{i=1}^{N_{\partial\Omega}} \|\mathcal{B}(u_\theta)(\boldsymbol{x}_i)\|^2$$

Convection

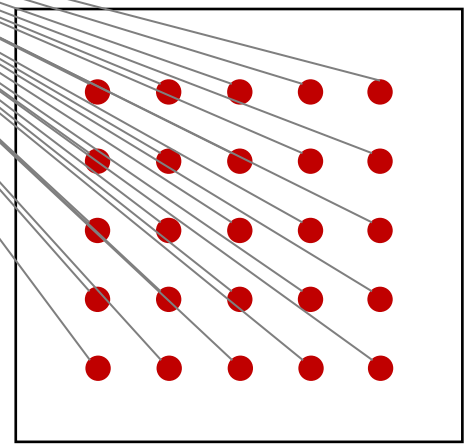
$$\frac{\partial u}{\partial t} + 50\frac{\partial u}{\partial x} = 0$$



x

t

Sampling



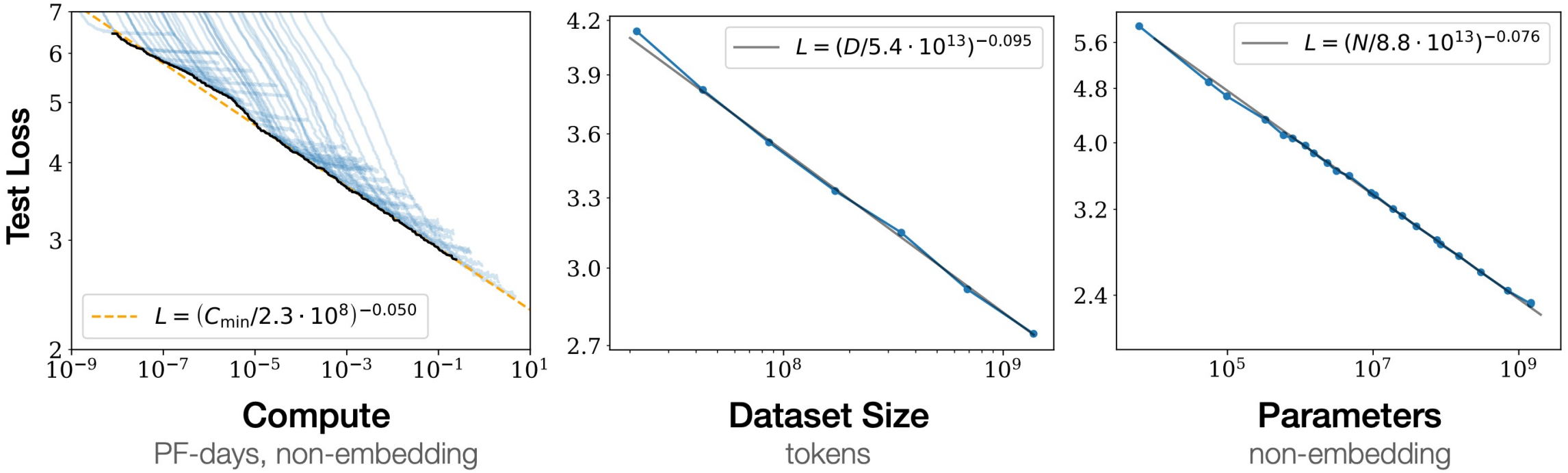
PINN Loss on Finite

Collocation Points

Extremely elegant formalization (autodifferential gradient, explicit constraint)

but still has some underlying issues to be solved

Open Question: Unlock the Deep Learning Capability

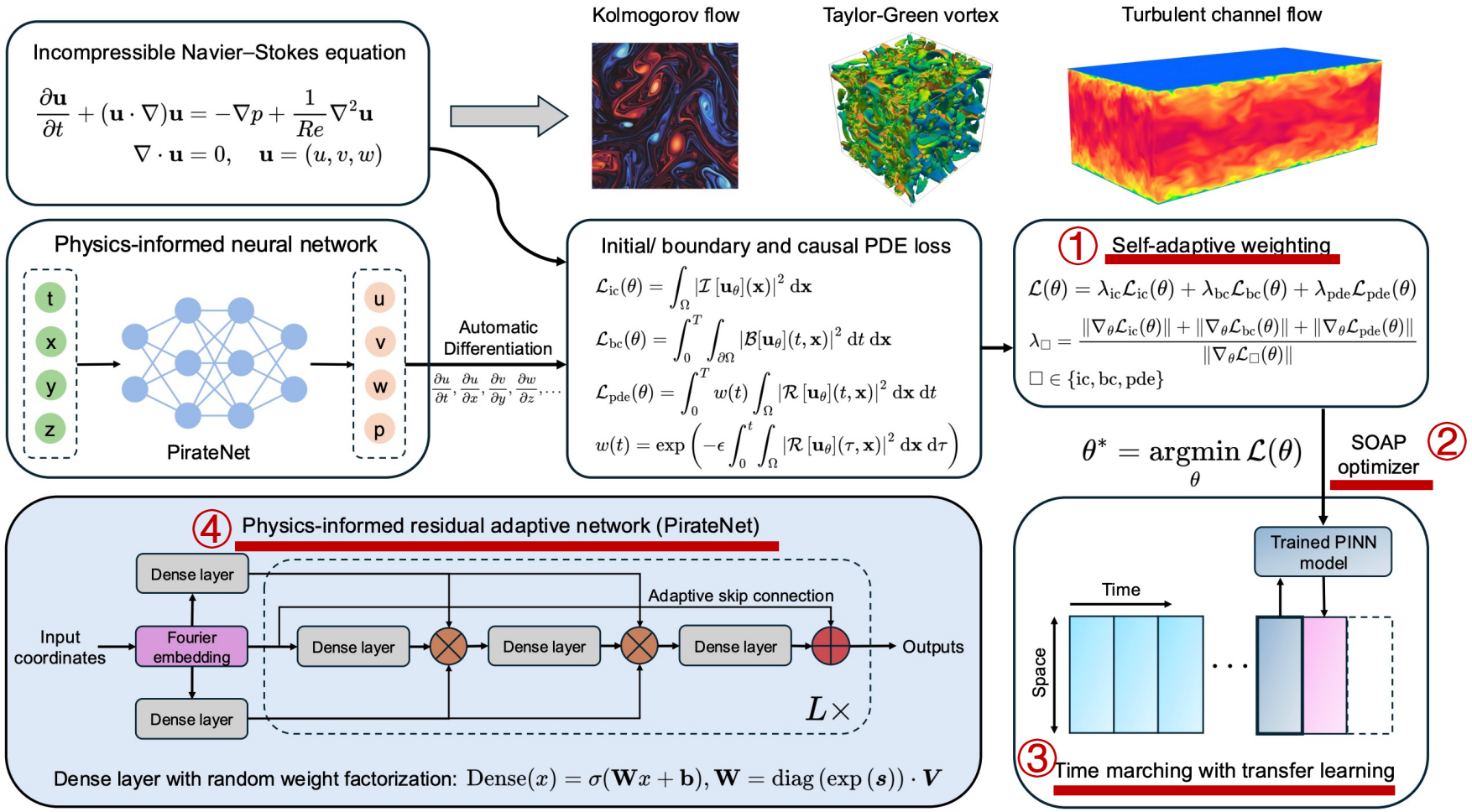


We need a general way to scale the capability of neural networks.

(such as a Transformer or a new optimization paradigm)

Kaplan et al. Scaling Laws for Neural Language Models, *ArXiv preprint 2001.08361 (2020)*

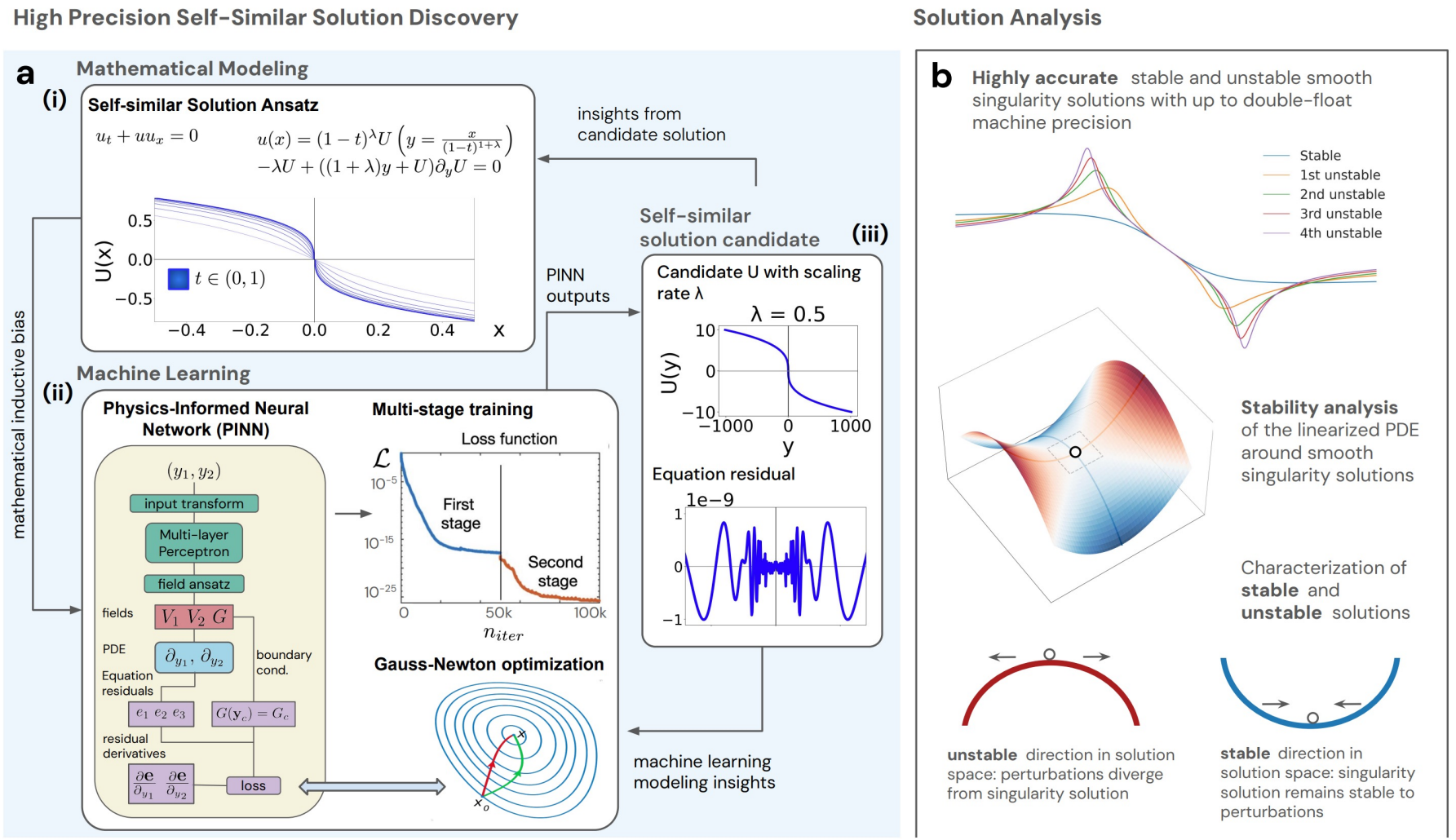
Recent Progress of PINNs



- Adopt a series of tricks to enable stable training
- First time to support 3D turbulence simulation

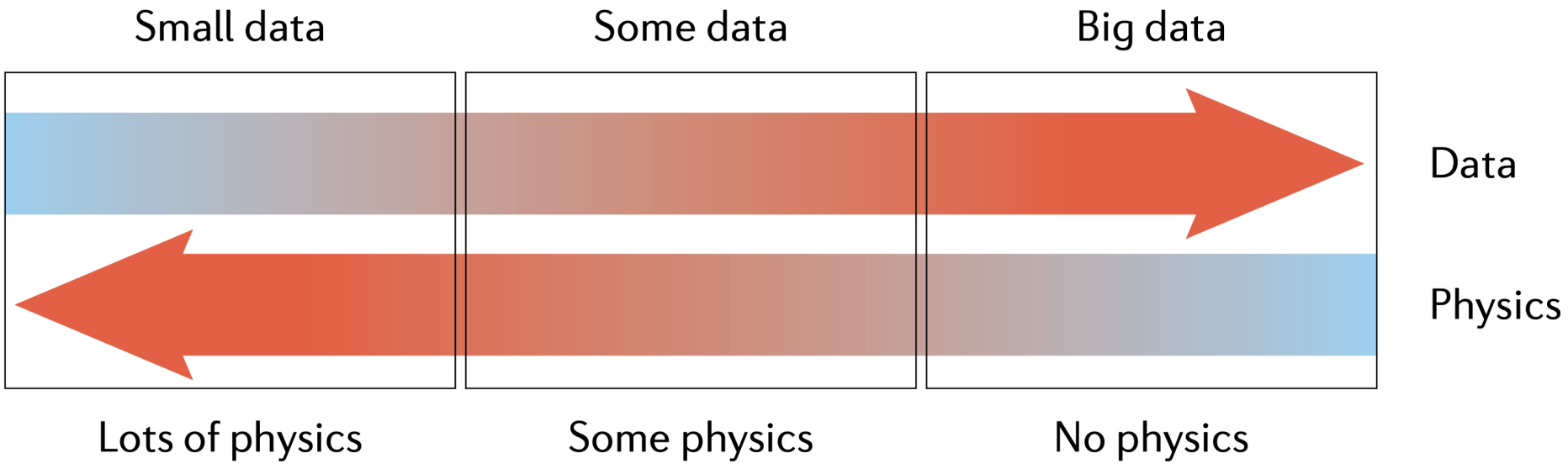
Wang et al. Simulating Three-dimensional Turbulence with Physics-informed Neural Networks, *ArXiv preprint 2507.08972 (2025)*

Recent Progress of PINNs



Wang et al. Discovery of Unstable Singularities, *ArXiv preprint 2509.14185 (2025)*

Overview of Neural PDE Solvers



Pure Physics

- *Numerical Solvers (FEMs)*
- Physics-Informed Neural Networks / Neural-FEMs

Hybrid Data-Physics

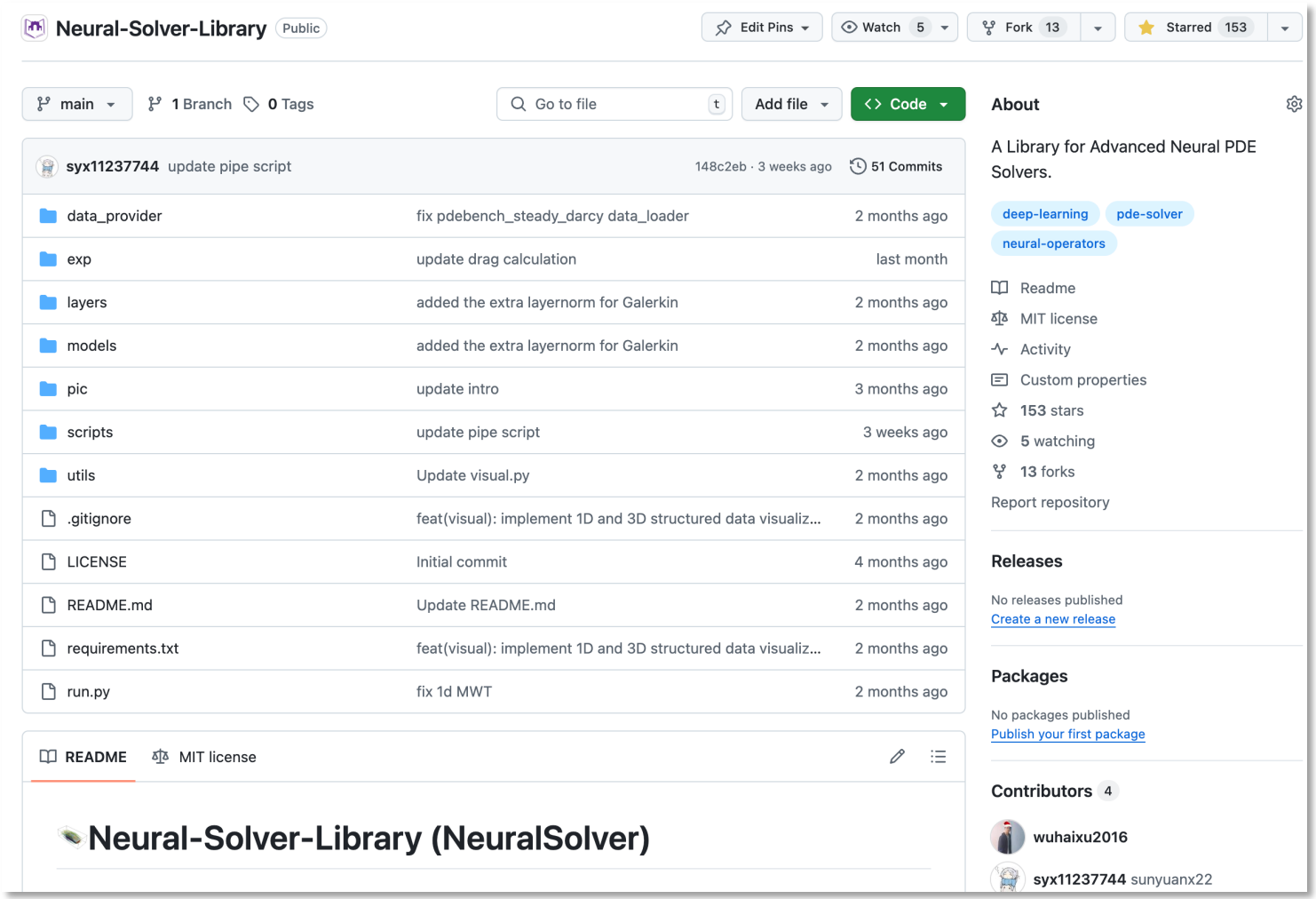
- Hybrid Simulators (NCLaw)
- Physics-Informed Neural Operator (PINO)

Pure Data

- Neural Operators / Neural Surrogates (DeepONet, Transolver)
- *General Deep Models*

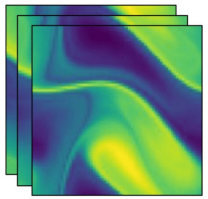
Karniadakis, G. et al. Physics-informed machine learning, ***Nature Review Physics*** 3, 422–440 (2021)

Neural-Solver-Library

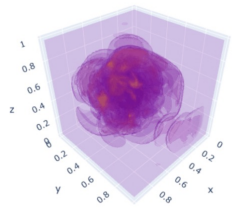


Code Link: *https://github.com/thuml/Neural-Solver-Library*

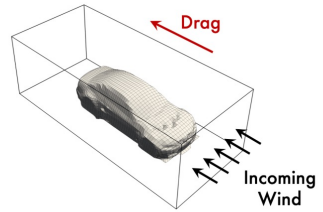
- ✓ 17 different PDE solvers
- ✓ 6 standard benchmarks, PDEBench and design tasks



Task 1: Standard



Task 2: PDEBench



Task 3: ShapeNet Car

Welcome to join us and add a new feature to this Library!

Acknowledgement

Jianmin Wang

Mingsheng Long

Wojciech Matusik

NVIDIA

physicsnemo

SIEMENS

Hang Zhou

Yuezhou Ma

Huakun Luo

Yuanxu Sun

Huikun Weng

Towards Practical Neural PDE Solvers

From RoPINN to ProPINN: Improved Optimization and Architecture

Haixu Wu

Computational Design and Fabrication Group, MIT CSAIL

Dec 19, 2025

RoPINN ProPINN