

---

# 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



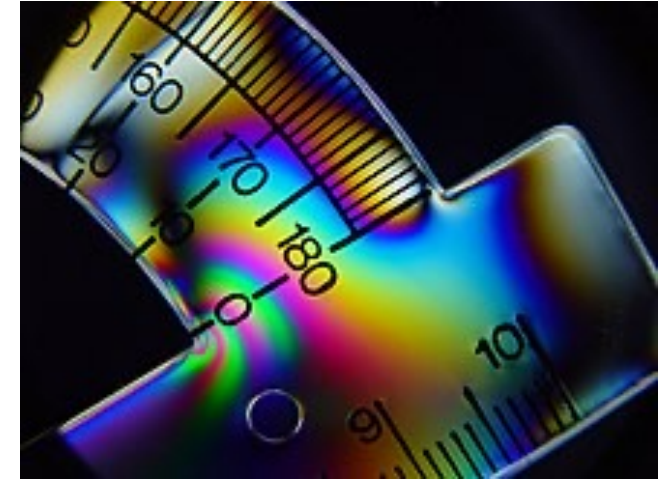
# Partial Differential Equations



Turbulence



Atmospheric circulation



Stress

## Navier-Stokes Equation for Fluid Dynamics

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{U}) = 0$$

$$\frac{\partial \mathbf{U}}{\partial t} + \mathbf{U} \cdot \nabla \mathbf{U} = \mathbf{f} + \frac{1}{\rho} \nabla \cdot (\mathbf{T}_{ij} \mathbf{e}_i \mathbf{e}_j)$$

$$\frac{\partial (e + \frac{1}{2} \mathbf{U}^2)}{\partial t} + \mathbf{U} \cdot \nabla (e + \frac{1}{2} \mathbf{U}^2) = \mathbf{f} \cdot \mathbf{U} + \frac{1}{\rho} \nabla \cdot (\mathbf{U} \cdot \mathbf{T}_{ij} \mathbf{e}_i \mathbf{e}_j) + \frac{\lambda}{\rho} \Delta T.$$

## Inner Stress of Solid Materials

$$\rho^s \frac{\partial^2 \mathbf{u}}{\partial t^2} + \nabla \cdot \boldsymbol{\sigma} = 0$$



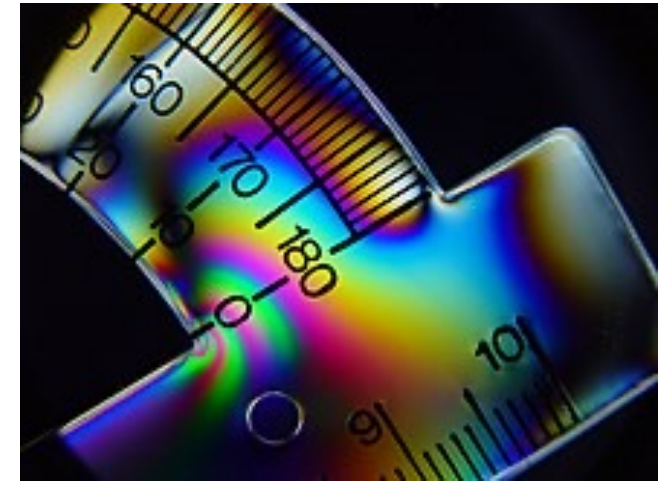
# Partial Differential Equations



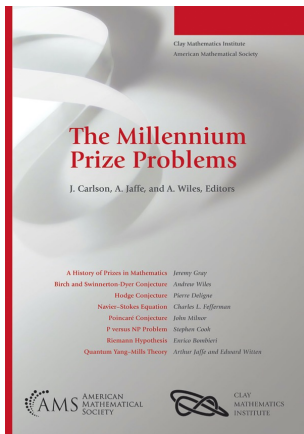
Turbulence



Atmospheric circulation



Stress

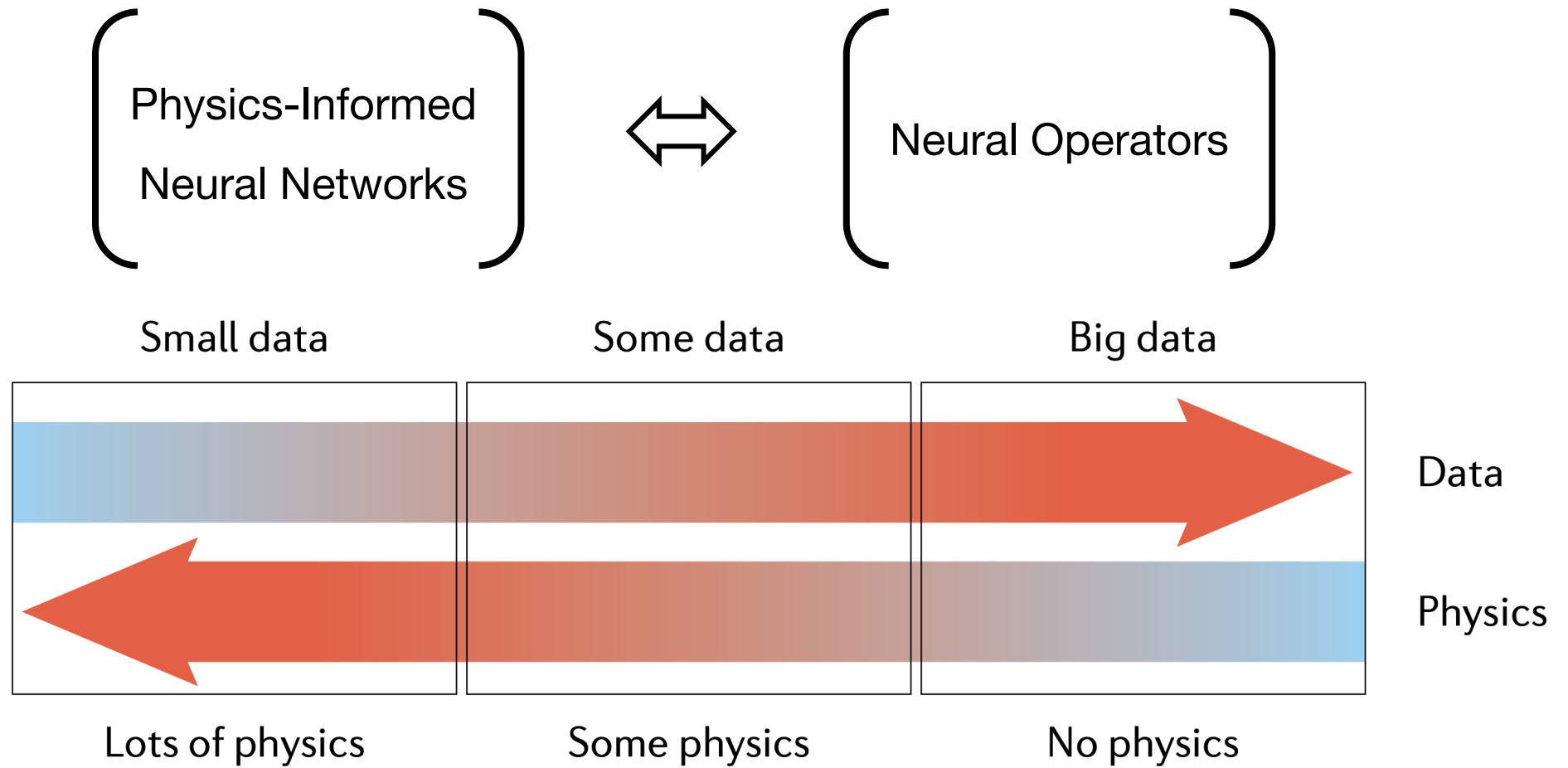


## Millennium Prize Problems

- 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)

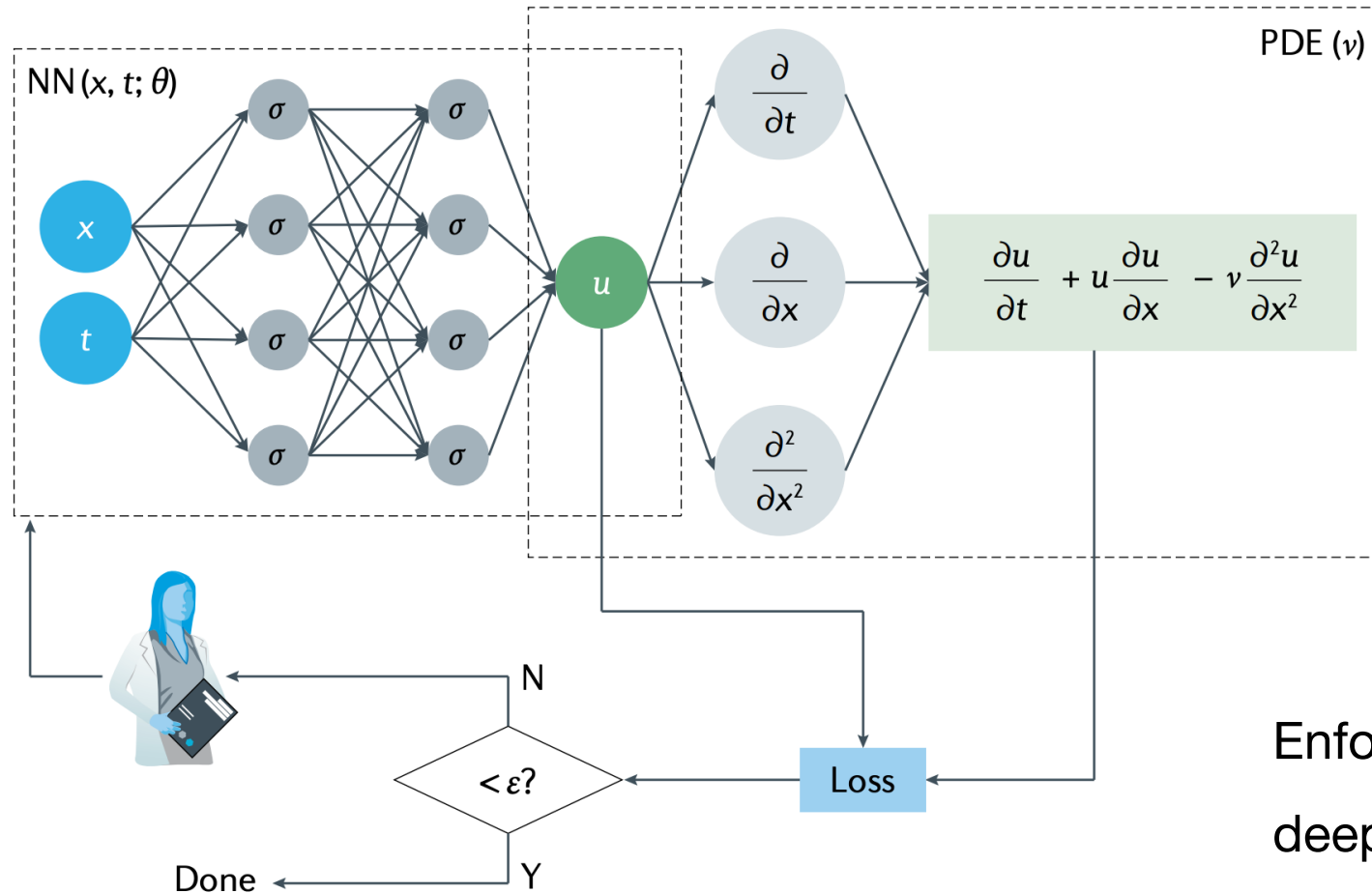
*It is really hard (usually impossible) to obtain the analytic solution of PDEs*

# Neural PDE Solvers





# Physics-Informed Neural Networks



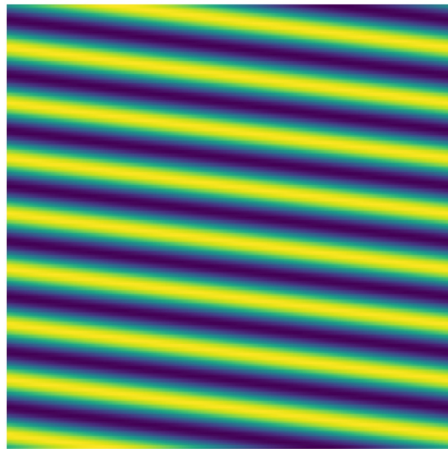
Enforcing outputs and gradients of deep models to satisfy target equations

# Physics-Informed Neural Networks

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

Convection

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

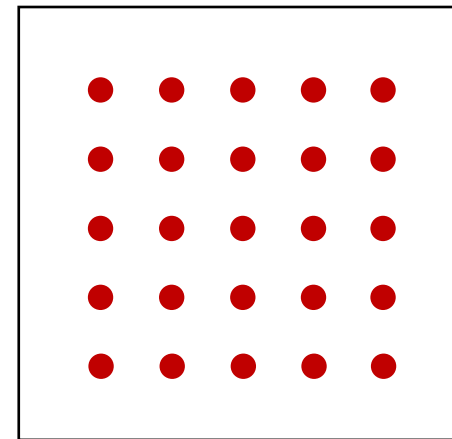
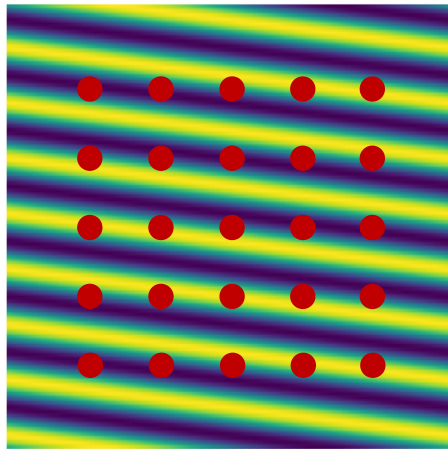


# Physics-Informed Neural Networks

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

Convection

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



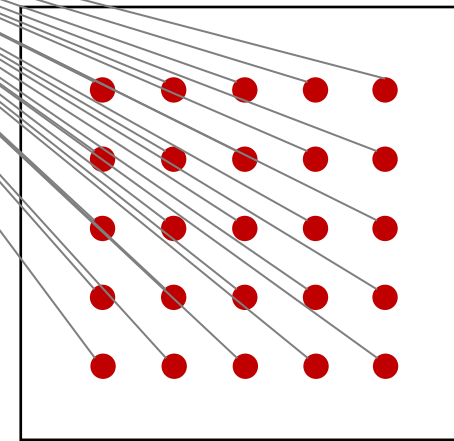
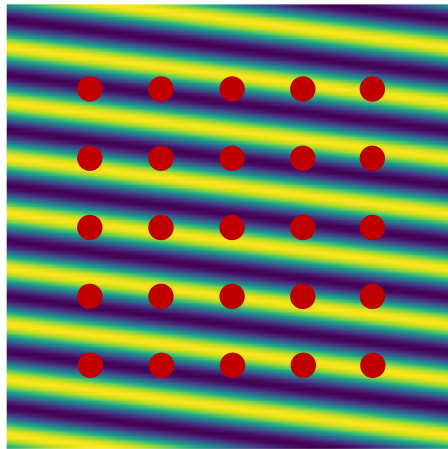


# Physics-Informed Neural Networks

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

Convection

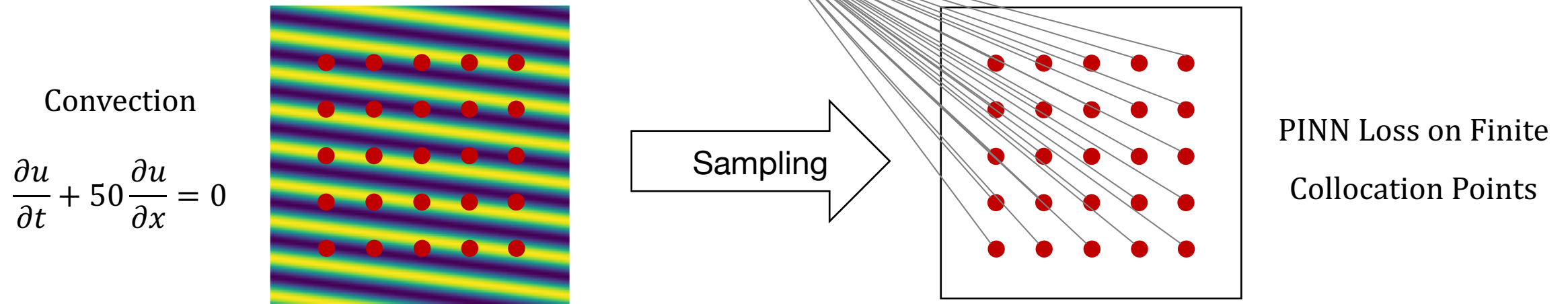
$$\frac{\partial u}{\partial t} + 50 \frac{\partial u}{\partial x} = 0$$



PINN Loss on Finite  
Collocation Points

# Physics-Informed Neural Networks

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



***Point Optimization (optimizing models on scattered points)***

**Insufficient to obtain an accurate solution for the whole domain**

# Related Works: High-order regularization

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



*Differential function*

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

- ✓ Add the high-order constraints of PDEs as regularization terms to loss function
- X Calculating high-order derivatives can be extremely time-consuming and unstable



# Related Works: Variational formulation

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



*Select test functions*

$$\langle \mathcal{F}(u), \underline{v} \rangle_{\Omega} = 0$$

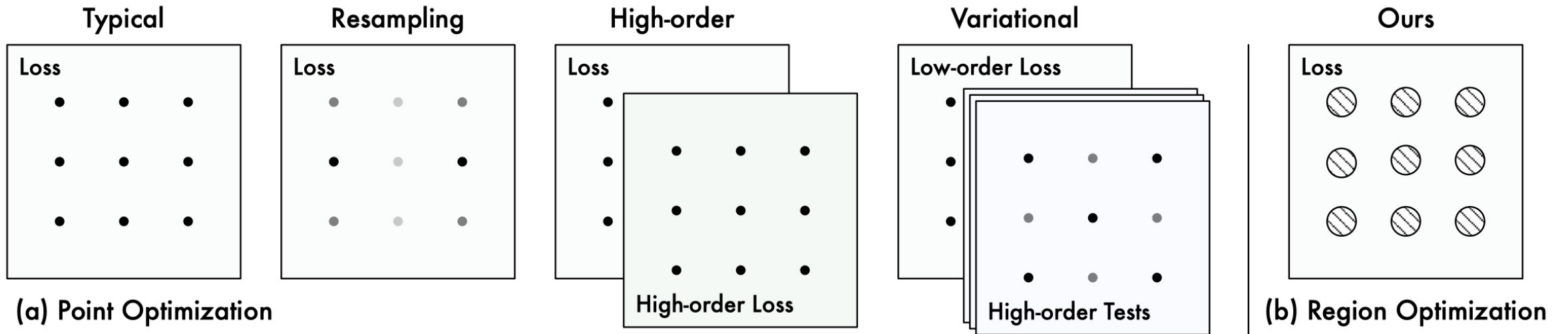


*Integrals by parts*

$$\mathcal{L}^{\text{equ}}(u_{\theta}) = \frac{1}{M} \sum_{k=1}^M \left\| \left\langle \mathcal{F}^{(x_j)}(u_{\theta})(\mathbf{x}), v_k(\mathbf{x}) \right\rangle \Big|_{\partial(x_j)\Omega} - \int_{\Omega} \left\langle \mathcal{F}^{(x_j)}(u_{\theta})(\mathbf{x}), \frac{\partial}{\partial x_j} v_k(\mathbf{x}) \right\rangle d\mathbf{x} \right\|^2$$

- ✓ High-order derivative operation in loss function is transferred to test functions
- X Integral on  $\Omega$  is still hard to compute, requires massive quadrature points
- X test function selection requires extra manual effort and M times computation costs

# Region Optimization V.S. Point Optimization



**Point Optimization:**

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

**Region Optimization:**

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

# Theoretical Analysis

➤ Generalization Error in Expectation

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

➤ Basic Assumption

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

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

(1) *If  $\mathcal{L}$  is convex for  $\theta$  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  $\theta, \mathbf{x}$  and is non-convex for  $\theta$  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  $\beta$ -smooth for  $\theta$ . If we run stochastic gradient method with step size  $\alpha_t$  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  $\theta$  and  $\alpha_t \leq \frac{2}{\beta}$ , then  $\mathcal{E}_{\text{gen}} \leq \left(1 - \frac{|\Omega_r|}{|\Omega|}\right) \frac{2L^2}{|\mathcal{S}|} \sum_{t=1}^T \alpha_t$ .*

(2) *If  $\mathcal{L}$  is bounded by a constant  $C$  for all  $\theta, \mathbf{x}$  and is non-convex for  $\theta$  with monotonically non-increasing step sizes  $\alpha_t \leq \frac{1}{\beta t}$ , then  $\mathcal{E}_{\text{gen}} \leq \frac{C}{|\mathcal{S}|} + \frac{2L^2(T-1)}{\beta(|\mathcal{S}|-1)} - JL\left(\frac{|\Omega_r|}{|\Omega|}\right)^2$ , 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  $\Omega$ , generalization error will be reduced to zero.

**Cannot be satisfied in practice, which requires the precise calculation of the integral of  $\Omega$**

# Theoretical Analysis: High-order PDE Constraints

**Corollary 3.6 (Region optimization for first-order constraints).** *Suppose that  $\mathcal{L}$  is bounded by  $C$  for all  $\theta, \mathbf{x}$  and is  $L$ -Lipschitz and  $\beta$ -smooth for  $\theta$ . If we run stochastic gradient method based on first-order  $j$ -th dimension loss function  $\frac{\partial}{\partial x_j} \mathcal{L}_r^{\text{region}}$  for  $T$  iterations, the generalization error in Theorem 3.5(2) still holds when we adopt the monotonically non-increasing step size  $\alpha_t \leq \frac{1}{2\beta t}$ .*

Introducing “region” can implicitly help training PINNs  
with high-order constraints.

**Example 3.7 (Point optimization fails in optimizing with first-order constraints).** *Under the same assumption with Corollary 3.6 we cannot obtain the Lipschitz and smoothness property of  $\frac{\partial}{\partial x_j} \mathcal{L}(u_\theta, \mathbf{x})$ . For example, suppose that  $\mathcal{L}(u_\theta, \mathbf{x}) = |\theta^\top \sqrt{\mathbf{x}}|$ ,  $\mathbf{x} \in [0, 1]^{(d+1)}$ , which is 1-Lipschitz-1-smooth. However,  $\nabla_\theta \frac{\partial}{\partial x_j} \mathcal{L}(u_\theta, \mathbf{x})$  is unbounded when  $\mathbf{x} \rightarrow \mathbf{0}$ , thereby not Lipschitz constant.*

# 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  $\theta_0$  and trust region calibration value  $\sigma_0 = 1$ .

**Output:** optimized PINN parameters  $\theta_T$ .

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

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

Update  $\theta_t$  to  $\theta_{t+1}$  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:  $\mathcal{S}' = \{\mathbf{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|\mathcal{S}|}$ ,  $\mathbf{x}_i \in \mathcal{S}$ ,  $\boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

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

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

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

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

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

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

**Important Note: This design is tailored to PINN loss,  
where we can precisely calculate 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:  $\mathcal{S}' = \{\mathbf{x}_i + \boldsymbol{\xi}_i\}_{i=1}^{|\mathcal{S}|}$ ,  $\mathbf{x}_i \in \mathcal{S}$ ,  $\boldsymbol{\xi}_i \sim U[0, \frac{r}{\sigma_t}]^{(d+1)}$

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

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

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

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

**Theorem 3.8 (Convergence rate).** *Suppose that there exists a constant  $H$ , s.t.  $\forall \mathbf{v}$  and  $\forall \mathbf{x} \in \Omega$ ,  $|\mathbf{v}^{\top} \nabla_{\theta} \mathcal{L}_r^{\text{region}}(u_{\theta}, \mathbf{x}) \mathbf{v}| \leq H \|\mathbf{v}\|^2$ . 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*

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



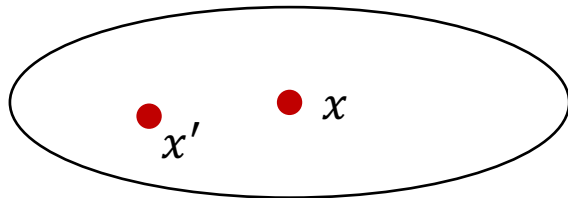
## Part 2: Trust Region Calibration

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

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

where  $\sigma$  represents the standard deviation of gradients in region  $\Omega_r$ .

Region  $x + \Omega_r$



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

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

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

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

*// Trust Region Calibration*

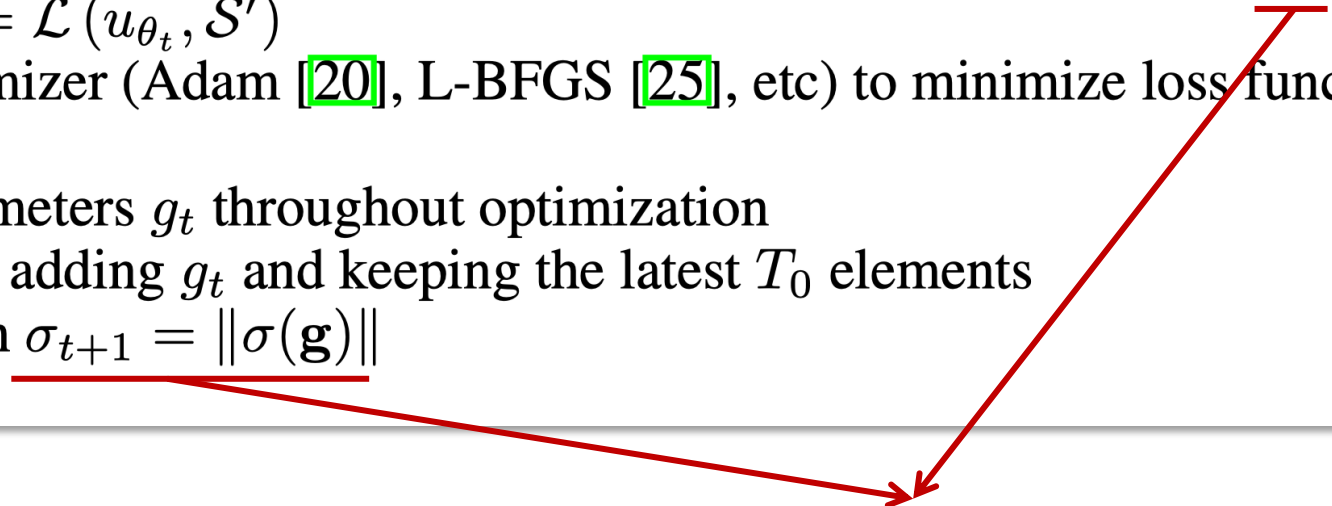
Record the gradient of parameters  $g_t$  throughout optimization

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.**


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

- ✓ 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

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

*// Region Optimization with Monte Carlo Approximation*

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

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

Update  $\theta_t$  to  $\theta_{t+1}$  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.**

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

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

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

Update  $\theta_t$  to  $\theta_{t+1}$  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  $\beta$ -smooth for  $\theta$  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*

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

# 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  $\mathcal{L}_r^{\text{approx}}(u_\theta, \mathbf{x}) = \nabla_\theta \mathcal{L}(u_\theta, \mathbf{x} + \boldsymbol{\xi})$ ,  $\boldsymbol{\xi} \sim U(\Omega_r)$  for  $T$  iterations, we further denote the upper bound of gradient estimation error as  $\mathcal{E}_{r,\text{grad}} = \max_{t \leq T} \|\nabla_\theta \mathcal{L}_r^{\text{approx}} - \nabla_\theta \mathcal{L}_r^{\text{region}}\|$ , then  $\mathcal{E}_{\text{gen}}$  satisfies:*

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

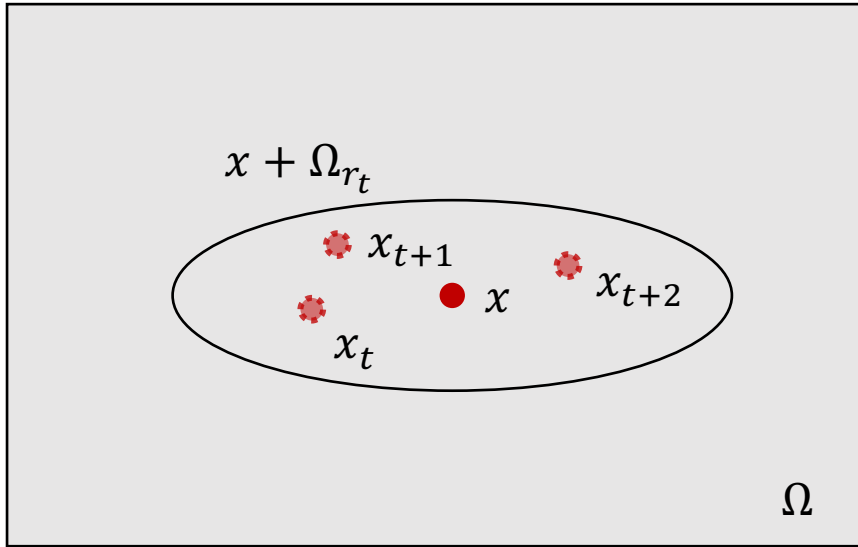
(2) *If  $\mathcal{L}$  is bounded by a constant  $C$  and is non-convex for  $\theta$  with monotonically non-increasing step sizes  $\alpha_t \leq \frac{1}{\beta t}$ , then  $\mathcal{E}_{\text{gen}} \leq \frac{C}{|\mathcal{S}|} + \frac{2L^2(T-1)}{\beta(|\mathcal{S}|-1)} \underbrace{\frac{-J' L (|\Omega_r|/|\Omega|)^2}{|\Omega_r|}}_{\text{inversely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

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

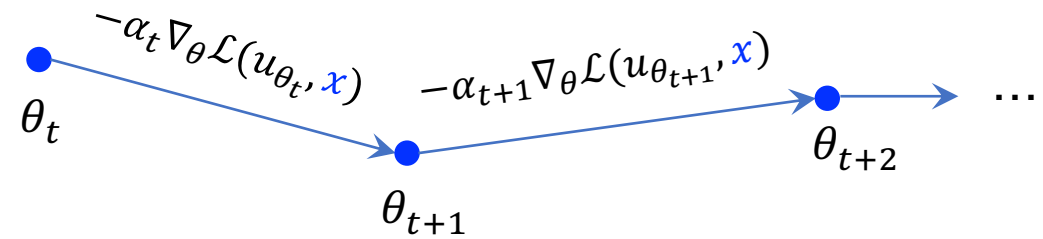
➤ Canonical point optimization ( $\Omega_r = 0$ ) and globally sampling points ( $\Omega_r = \Omega$ ) are fixed special cases.

**RoPINN can adaptively balance optimization and generalization during training.**

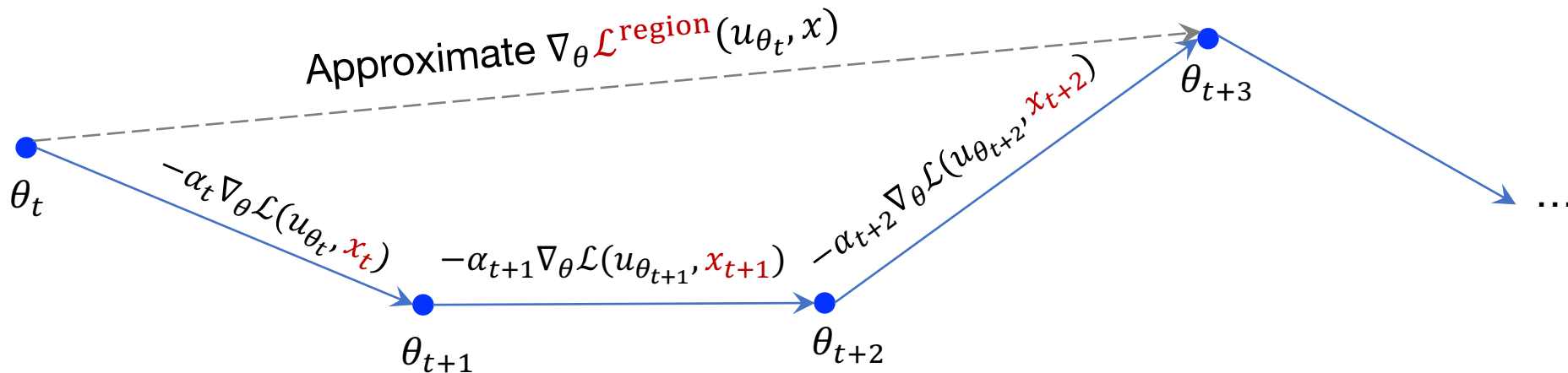
# Intuitive Understanding



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



**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

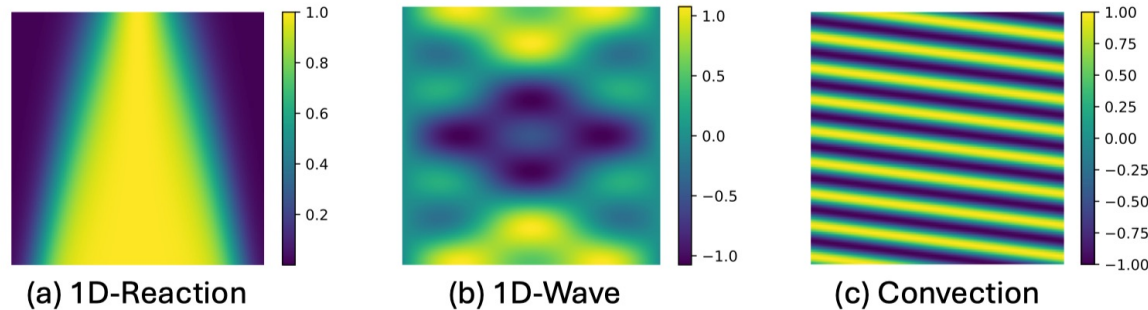


Table 4: Details of datasets in PINNacle [12] (16 different PDEs included in our experiments), including the dimension of inputs, highest order of PDEs, number of train/test points and concrete equations. Here we only present the simplified PDE formalizations for intuitive understanding. More detailed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

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

➤ 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 Promotion	<b>4.7e-5</b>	<b>0.056</b>	<b>0.095</b>	<b>1.5e-3</b>	<b>0.063</b>	<b>0.064</b>	<b>1.0e-2</b>	<b>0.635</b>	<b>0.720</b>	<b>93.8%</b>	<b>100.0%</b>
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 Promotion	<b>9.0e-6</b>	<b>0.007</b>	<b>0.013</b>	<b>1.7e-2</b>	<b>0.309</b>	<b>0.321</b>	<b>1.2e-2</b>	<b>0.819</b>	<b>0.870</b>	<b>81.3%</b>	<b>81.3%</b>
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 Promotion	<b>2.2e-5</b>	<b>0.022</b>	<b>0.039</b>	<b>1.5e-4</b>	<b>0.016</b>	<b>0.017</b>	<b>9.6e-4</b>	<b>0.173</b>	<b>0.197</b>	<b>81.3%</b>	<b>87.5%</b>
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 Promotion	<b>1.0e-6</b>	<b>0.007</b>	<b>0.017</b>	<b>6.5e-3</b>	<b>0.165</b>	<b>0.172</b>	<b>1.2e-5</b>	<b>0.005</b>	<b>0.006</b>	<b>100.0%</b>	<b>100%</b>
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 Promotion	<b>4.9e-5</b>	<b>0.026</b>	<b>0.051</b>	<b>9.6e-3</b>	<b>0.177</b>	<b>0.191</b>	<b>2.2e-2</b>	<b>0.805</b>	<b>0.801</b>	<b>100%</b>	<b>93.8%</b>

➤ 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).

# Main Results

Table 3: Adding RoPINN to other strategies based on PINN. *Time* is for every  $10^2$  training iterations on 1D-Reaction.

Method rMSE	1D-Reaction	1D-Wave	Convection	Time (s)
PINN [36]	0.981	0.335	0.840	18.47
+gPINN [55]	0.978	0.399	0.935	37.91
+vPINN [18]	0.982	0.173	0.743	38.78
+RoPINN	<b>0.095</b>	<b>0.064</b>	<b>0.720</b>	20.04
+NTK [47]	0.098	0.149	0.798	27.99
+NTK+RoPINN	<b>0.052</b>	<b>0.023</b>	<b>0.693</b>	29.96
+RAR [51]	0.981	0.126	0.771	19.71
+RAR+RoPINN	<b>0.080</b>	<b>0.030</b>	<b>0.695</b>	20.89

Other two PINN training strategies:

- NTK (loss-reweighting)
- RAR (data-sampling)

**RoPINN can be integrated seamlessly with other strategies without extra gradient calculation, which verifies its orthogonal contribution and favorable efficiency.**

# Algorithm Analysis: Region Size

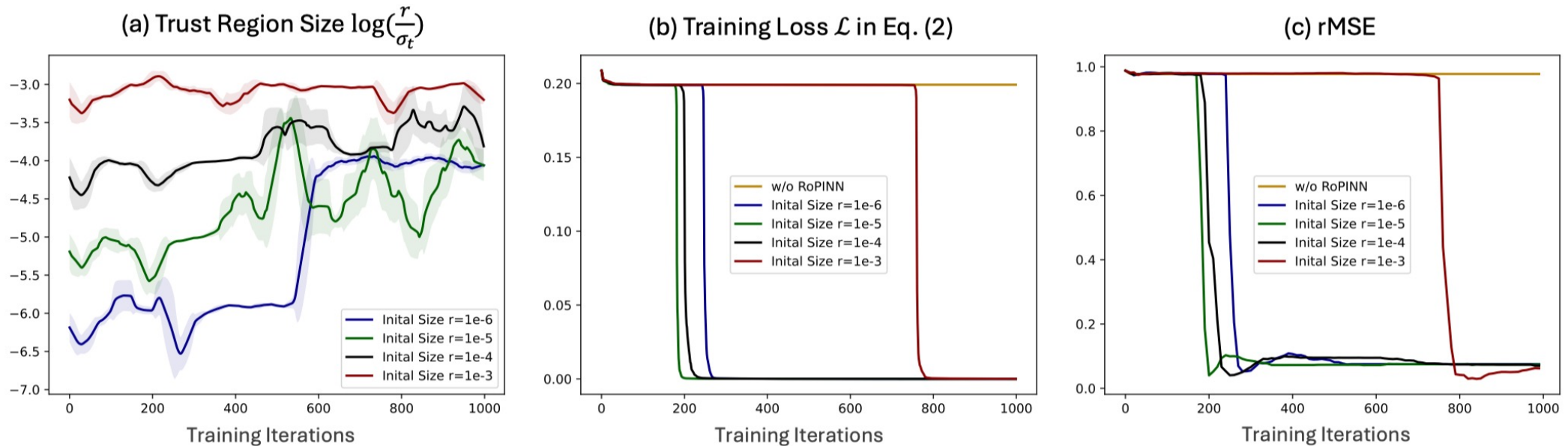


Figure 2: Optimization of canonical PINN [36] on the 1D-Reaction under different region sizes.

- ✓ **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

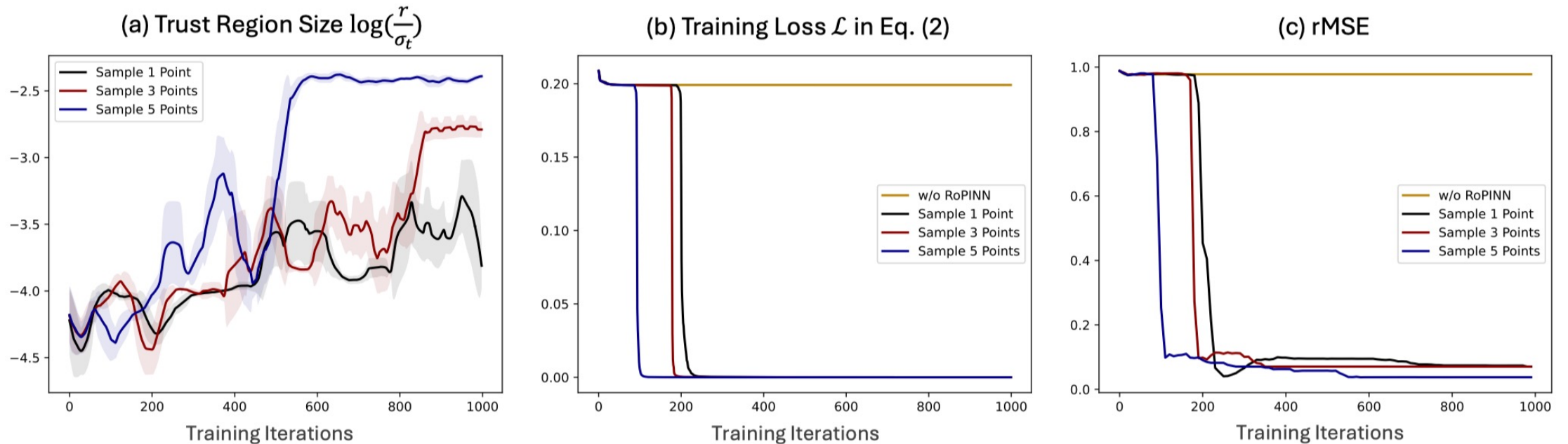


Figure 3: Optimization of canonical PINN [36] on the 1D-Reaction under different sample points.

**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: Efficiency

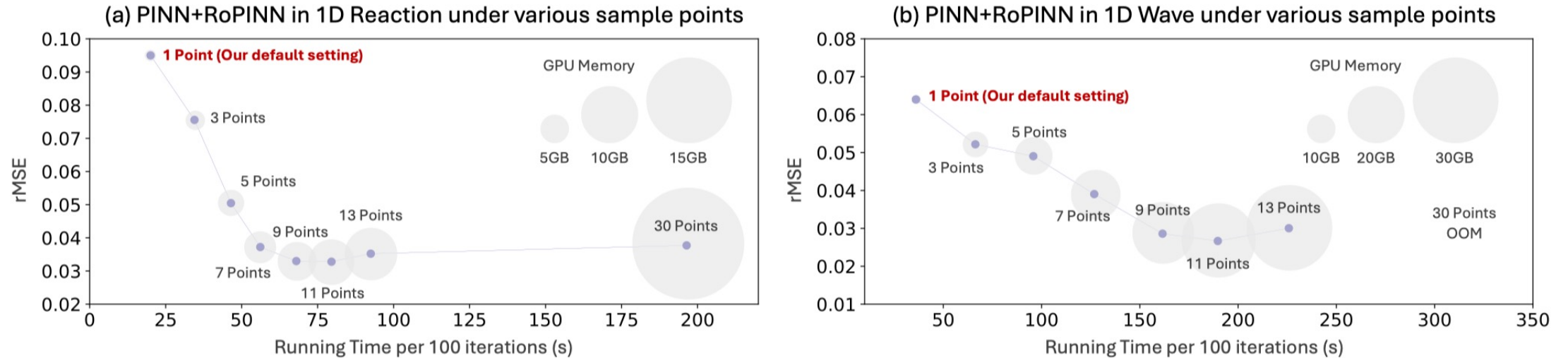


Figure 4: Efficiency and model performance w.r.t. number of samples. Note that the default setting of RoPINN is just sampling one point, which will not bring extra gradient calculation costs.

The benefits brought by more sampling points will saturate around 10 points.

**Our default setting is just sampling 1 point**, which can keep the similar efficiency as point optimization.

# Algorithm Analysis: Ablations

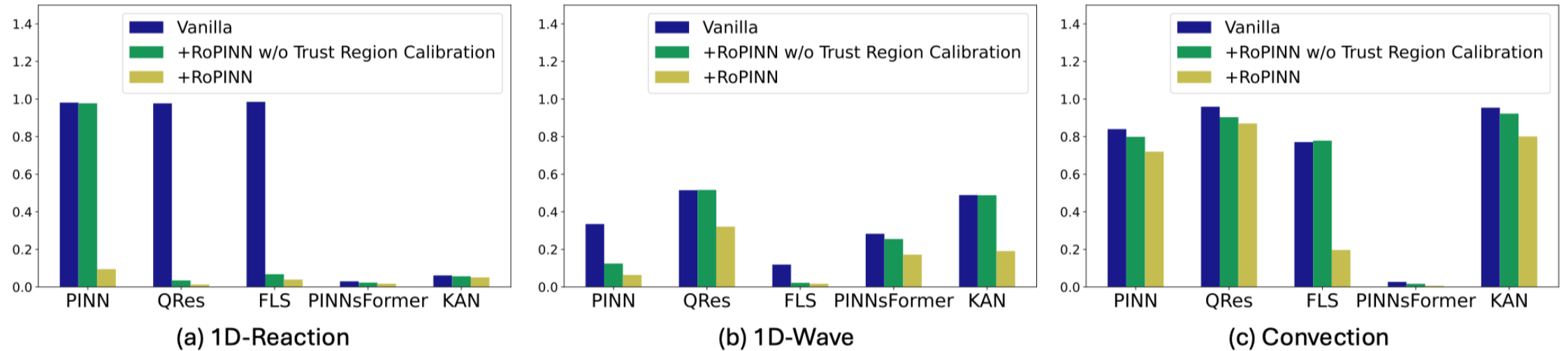


Figure 5: Ablation study of RoPINN on different PDEs and diverse base models. rMSE is recorded.

**Without trust region calibration, RoPINN (only region sampling) can already boost the performance.**

**Trust region calibration can make the performance better and more stable.**



# Algorithm Analysis: Loss Landscape

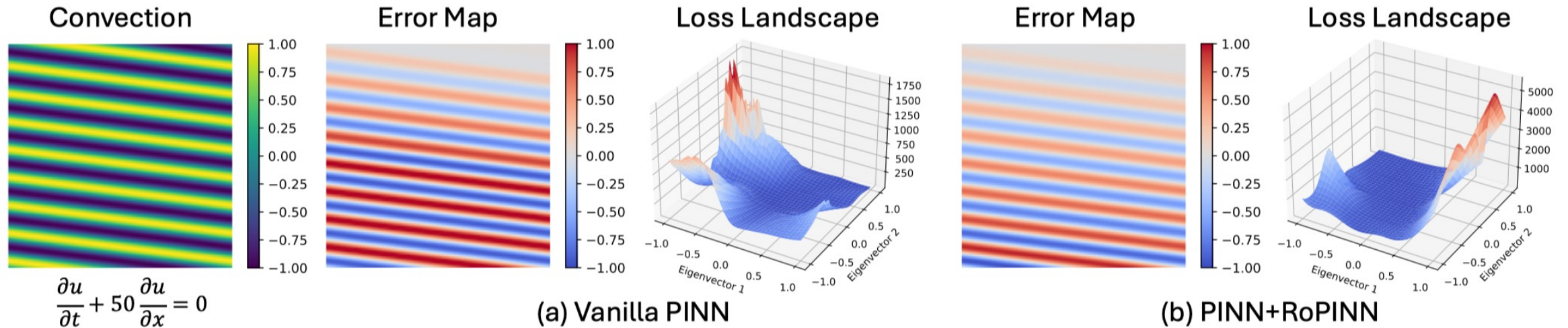


Figure 6: Loss landscape of RoPINN and vanilla PINNs on the Convection equation. *Error Map* refers to the distance between model prediction and the accurate solution, i.e.  $(u_\theta - u)$ .

“PINN failure modes” is not caused by limited model capacity but by **hard-to-optimize loss landscape**.

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

# More Showcases

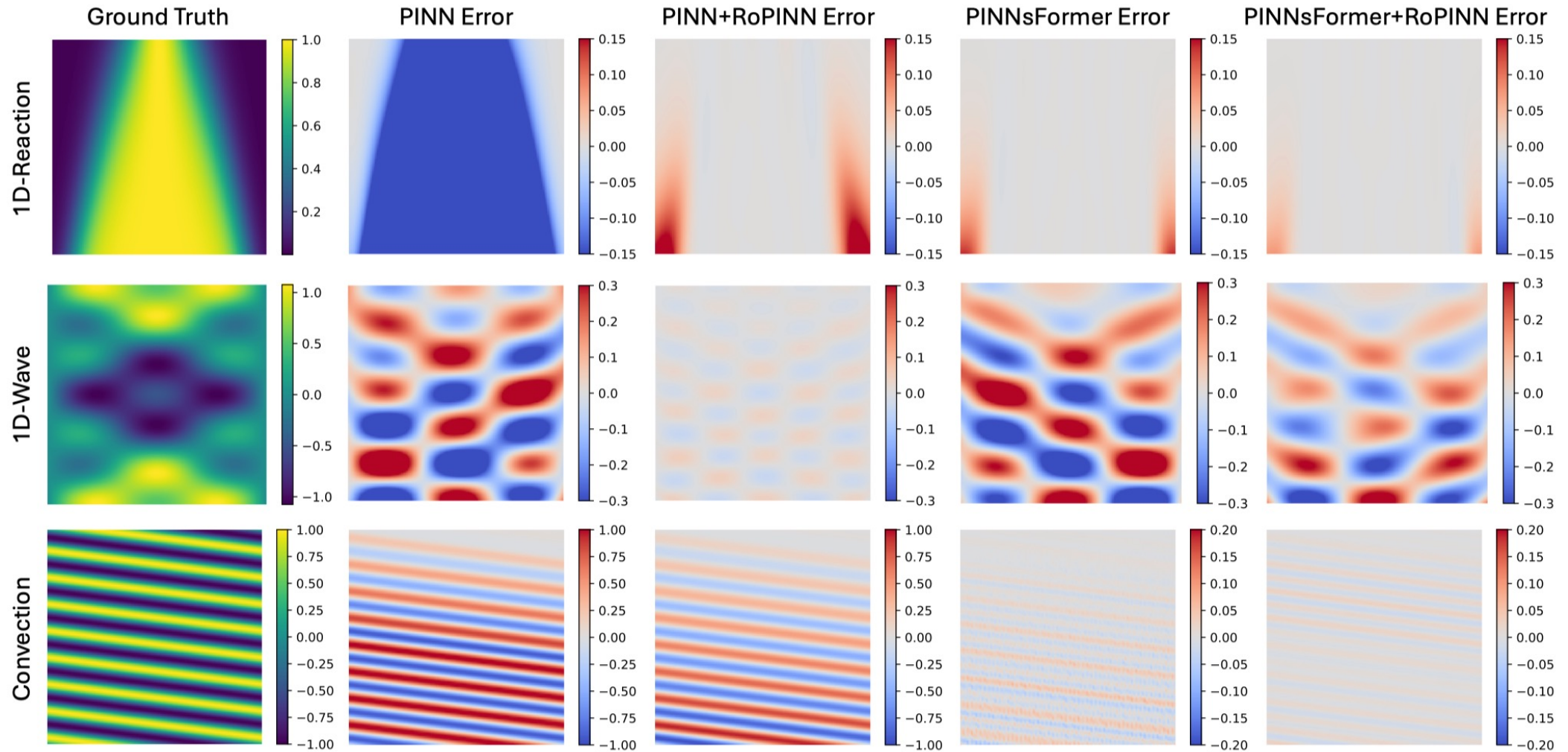


Figure 9: Showcases of RoPINN on the first three datasets based on PINN and PINNsFormer.

# Open Source

The screenshot shows the GitHub interface for the repository 'thuml / RoPINN'. The repository is public and has 7 stars, 3 watchers, and 0 forks. The main branch is 'main'. The repository contains 16 files and folders, including 'models', 'pic', 'scripts', '.gitignore', and several Python files for optimization. The repository is licensed under MIT. The right sidebar shows repository statistics and options to view the README, MIT license, activity, and custom properties. There are no releases or packages published yet.

File/Folder	Commit Message	Commit Date
models	add pinnsformer	2 days ago
pic	init	5 days ago
scripts	add pinnsformer	2 days ago
.gitignore	Initial commit	2 weeks ago
1d_reaction_point_optimization.py	add pinnsformer	2 days ago
1d_reaction_region_optimization.py	add pinnsformer	2 days ago
1d_wave_point_optimization.py	add pinnsformer	2 days ago
1d_wave_region_optimization.py	add pinnsformer	2 days ago
LICENSE	Initial commit	2 weeks ago
README.md	Update README.md	3 days ago
convection_point_optimization.py	add pinnsformer	2 days ago
convection_region_optimization.py	add pinnsformer	2 days ago
model_dict.py	add pinnsformer	2 days ago
requirements.txt	init	5 days ago
util.py	init	5 days ago

Code is available at <https://github.com/thuml/RoPINN>



Thank You!

wuhx23@mails.tsinghua.edu.cn



长按关注，获取最新资讯