JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

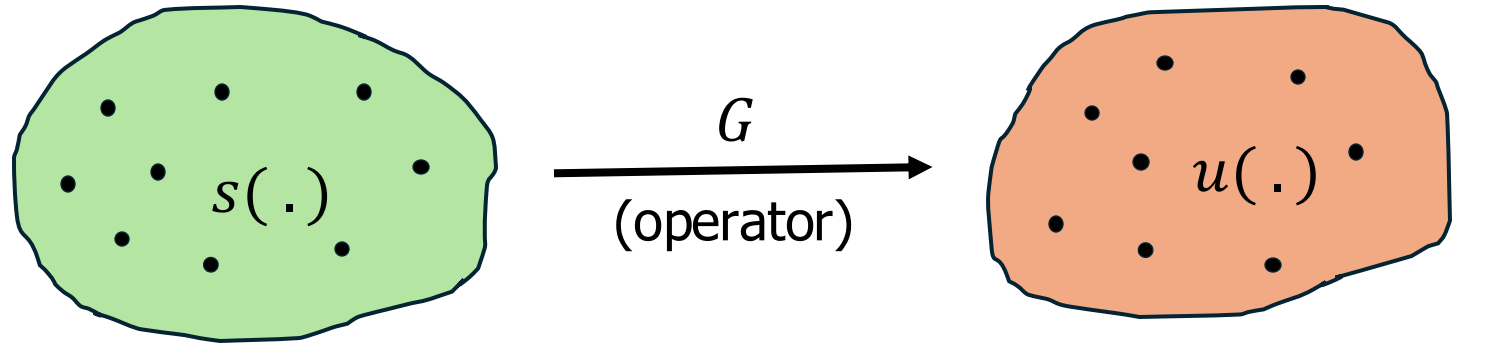
Our Proposed framework

Efficient Training of Deep Neural Operator Networks via Randomized Sampling

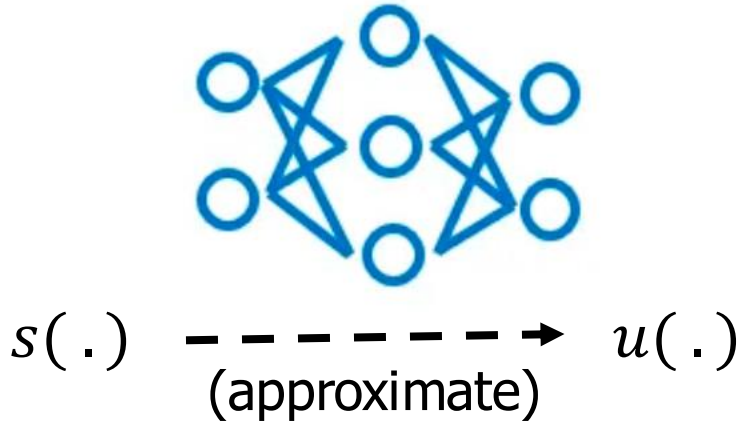


Karumuri Sharmila, Lori Graham-Brady, and Somdatta Goswami. "Efficient Training of Deep Neural Operator Networks via Randomized Sampling." *arXiv preprint arXiv:2409.13280* (2024).

Operator learning

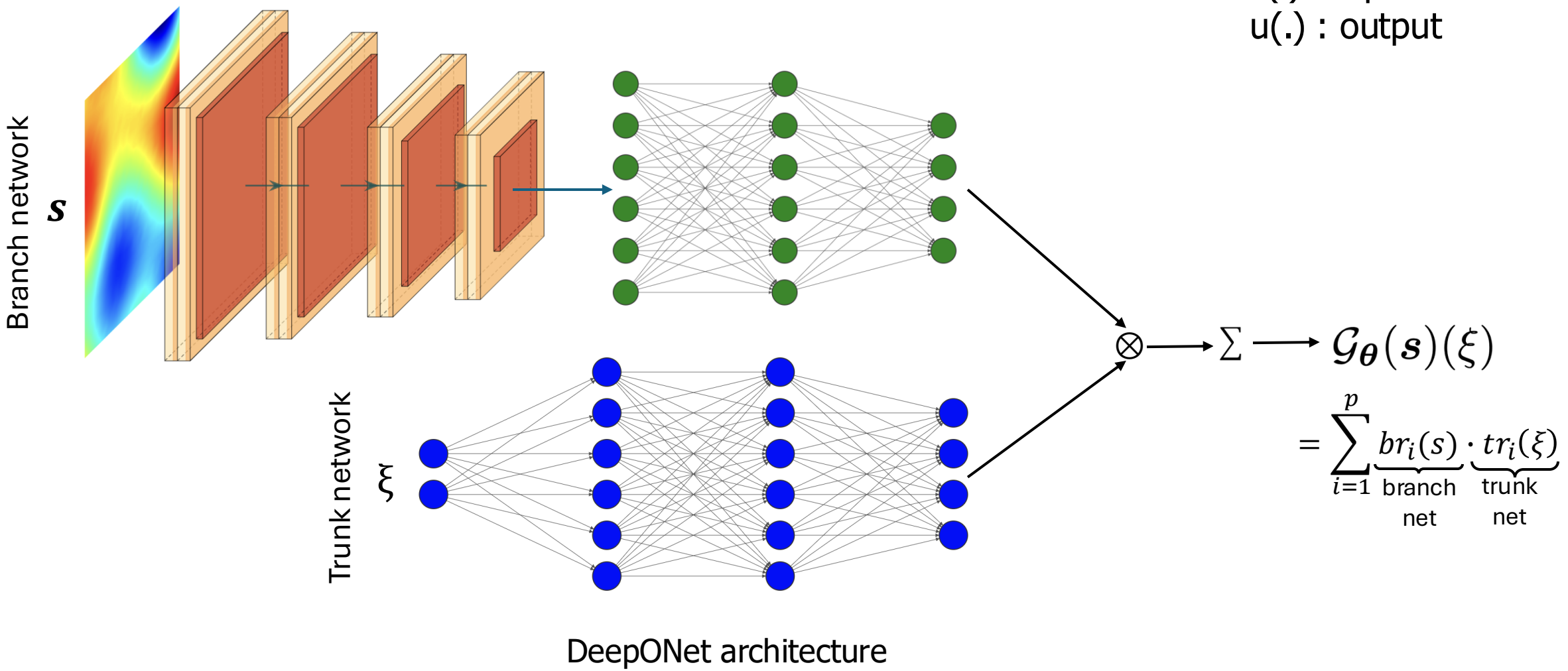


Neural Operator learning



DeepONet

s(.) : input
u(.) : output



DeepONet architecture

Traditional approach of training DeepONet

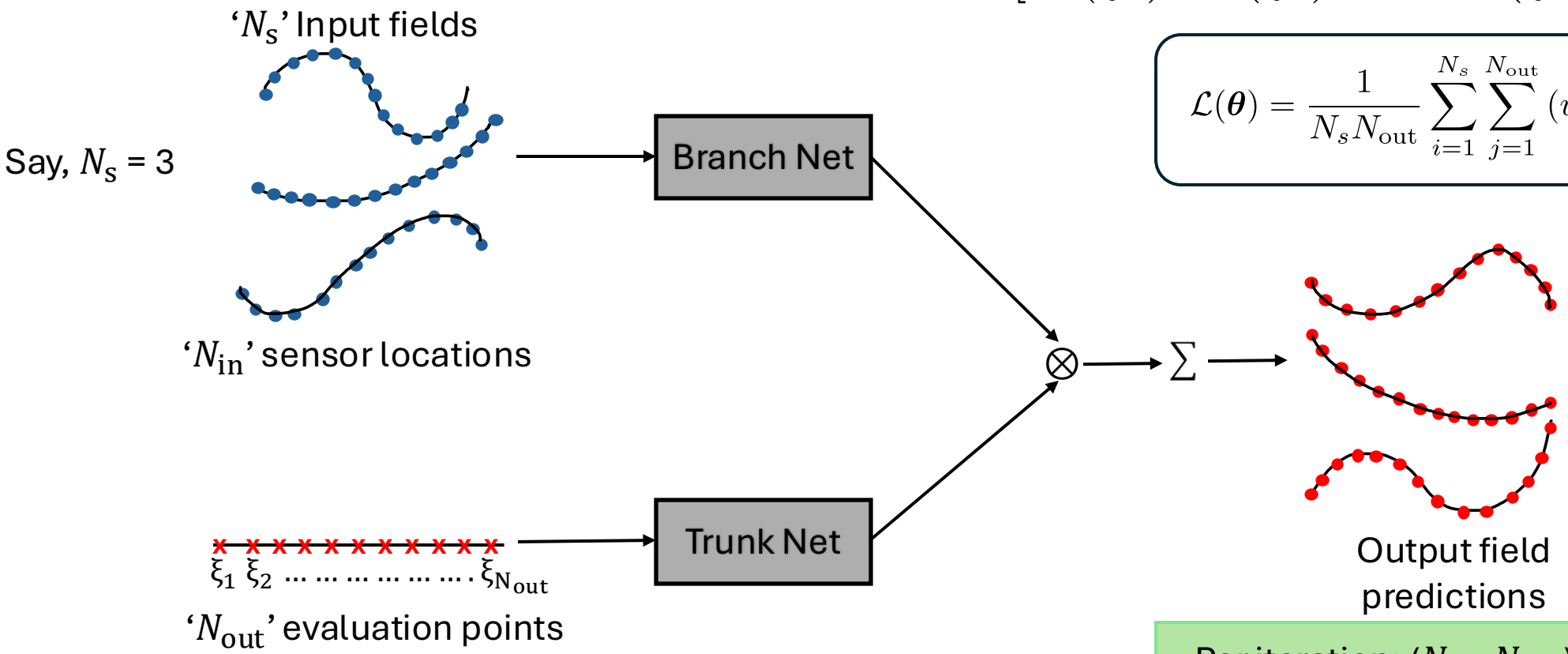
Training Dataset

$$\mathcal{D}_{\text{train}} = \{(\boldsymbol{s}_i, \boldsymbol{u}_i)\}_{i=1}^{N_{\text{train}}}$$

Input field data $\boldsymbol{s}_i = [s_i(\eta_1), s_i(\eta_2), \ldots, s_i(\eta_{N_{\text{in}}})]$

Output field data $\boldsymbol{u}_i = [u_i(\xi_1), u_i(\xi_2), \ldots, u_i(\xi_{N_{\text{out}}})]$

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_s N_{\text{out}}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_{\text{out}}} (u_i(\xi_j) - \mathcal{G}_{\boldsymbol{\theta}}(\boldsymbol{s}_i)(\xi_j))^2$$



Per iteration: (N_s x N_{out}) evaluations

In each iteration of training for a given input field, output field is evaluated at all 'N_{out}' sensor locations.

Shortcomings

The training cost increases drastically as the number of

- training samples in a batch (N_s) increases
- evaluation points (N_{out}) increases

Therefore, not an efficient training architecture for high-dimensional systems.

The advantages of employing a stochastic gradient descent optimizer is not leveraged.

Our approach of training DeepONet

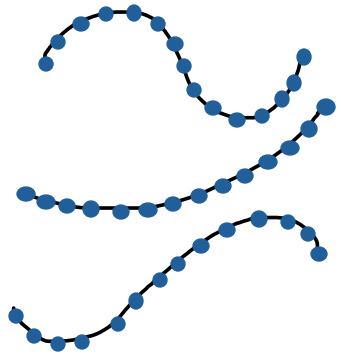
Training Dataset

Input field data $\boldsymbol{s}_i = [s_i(\eta_1), s_i(\eta_2), \ldots, s_i(\eta_{N_{\text{in}}})]$

Output field data $\boldsymbol{u}_i = [u_i(\xi_1), u_i(\xi_2), \ldots, u_i(\xi_{N_{\text{out}}})]$

'N_s' Input fields

Say, N_s = 3

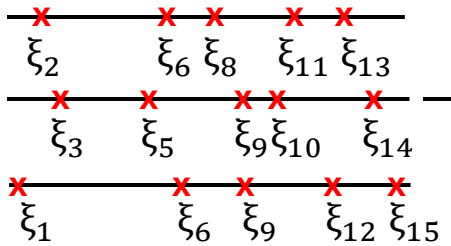


'N_{in}' sensor locations

Branch Net

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_s N_{\text{eval}}} \sum_{i=1}^{N_s} \sum_{j=1}^{N_{\text{eval}}} \left(u_i(\xi_{ij}^*) - \mathcal{G}_{\boldsymbol{\theta}}(\boldsymbol{s}_i)(\xi_{ij}^*)\right)^2$$

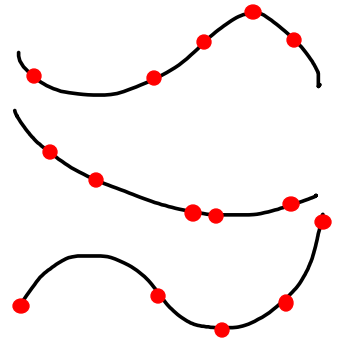
Idea: Randomized sampling of evaluation points during training



'N_{eval} = 5' evaluation points

Trunk Net

$\otimes$ → $\sum$ →



Output field predictions

Per iteration: (N_s x N_{eval}) evaluations

Hypothesis

- ✓ Training time would be faster.
- ✓ Loss converges to a given value faster.
- ✓ Same accuracy as traditional approach can be achieved faster.
- ✓ Traditional approach of training is like over training and memorizing.

Our approach of training DeepONet

Algorithm

Code in PyTorch

Algorithm 1 The proposed sampling technique to train DeepONet.

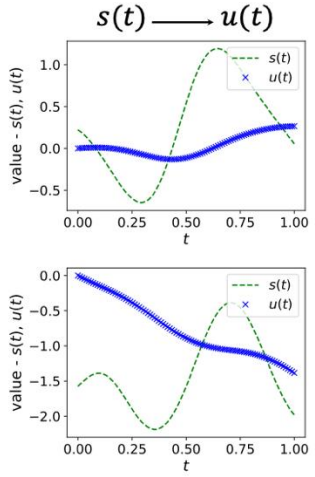
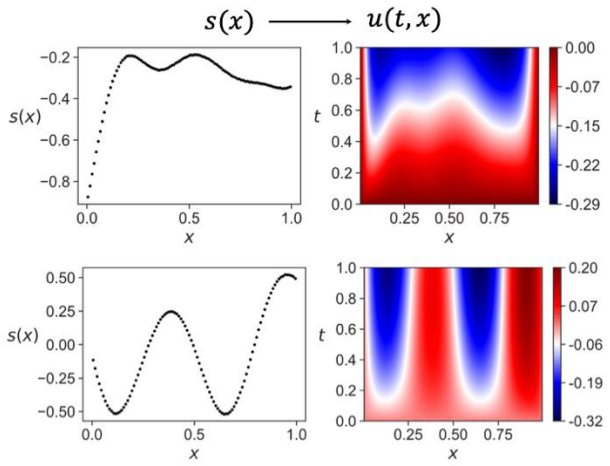
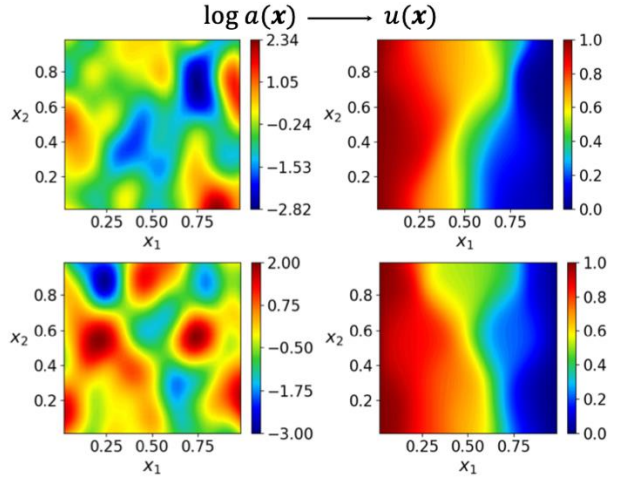
Require: Branch and trunk networks architecture, number of training samples N_{train}, training data $\mathcal{D}_{\text{train}} = \{(\boldsymbol{s}_i, \boldsymbol{u}_i)\}_{i=1}^{N_{\text{train}}}$, where $\boldsymbol{s}_i = [s_i(\eta_1), s_i(\eta_2), \ldots, s_i(\eta_{N_{\text{in}}})]$ and $\boldsymbol{u}_i = [u_i(\xi_1), u_i(\xi_2), \ldots, u_i(\xi_{N_{\text{out}}})]$, output sensor locations $\Xi = [\xi_1, \xi_2, \ldots, \xi_{N_{\text{out}}}]$, batch size bs, number of batches $N_{\text{batch}} = \lceil \frac{N_{\text{train}}}{\text{bs}} \rceil$, number of evaluation points N_{eval}, learning rate α, and number of epochs N_{epochs}.

Initialize parameters of branch and trunk networks $\boldsymbol{\theta}$.
for $k = 1$ **to** N_{epochs} **do**
  Shuffle training data: $\mathcal{D}_{\text{train}} \leftarrow \{(\boldsymbol{s}_{\sigma(i)}, \boldsymbol{u}_{\sigma(i)})\}_{i=1}^{N_{\text{train}}}$, where σ is a permutation.
  for $j = 1$ **to** N_{batch} **do**
    start $\leftarrow (j-1) \times \text{bs} + 1$, end $\leftarrow \min(j \times \text{bs}, N_{\text{train}})$
    Get mini-batch: $\mathcal{D}_{\text{train},j} \leftarrow \{(\boldsymbol{s}_a, \boldsymbol{u}_a)\}_{a=\text{start}}^{\text{end}}$
    for $a =$ start **to** end **do**
      Get $\Xi_a^* \subset \Xi$ such that $|\Xi_a^*| = N_{\text{eval}}$.
      Select N_{eval} distinct indices $\{i_{(a,1)}, i_{(a,2)}, \ldots, i_{(a,N_{\text{eval}})}\}$ uniformly at random from $\{1, 2, \ldots, N_{\text{out}}\}$ with $i_{(a,p)} \neq i_{(a,q)}$ for $p \neq q$.
      $\Xi_a^* = [\xi_{i_{(a,1)}}, \xi_{i_{(a,2)}}, \ldots, \xi_{i_{(a,N_{\text{eval}})}}]$
    end for
    Compute loss:

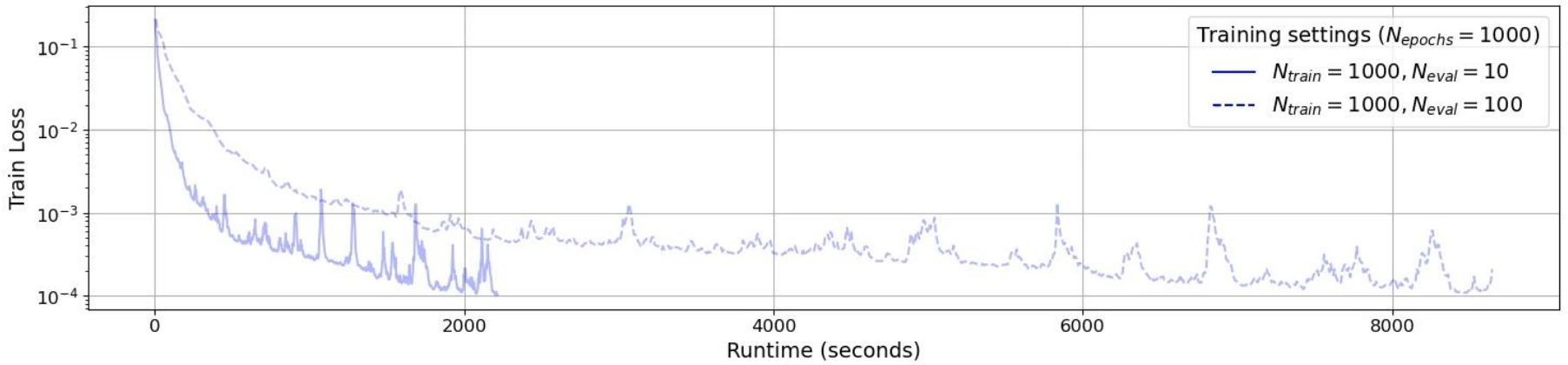
$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{(\text{end} - \text{start} + 1) \cdot N_{\text{eval}}} \sum_{a=\text{start}}^{\text{end}} \sum_{l=1}^{N_{\text{eval}}} \left(u_a(\xi_{i_{(a,l)}}) - \mathcal{G}_{\boldsymbol{\theta}}(\boldsymbol{s}_a)(\xi_{i_{(a,l)}}) \right)^2$$

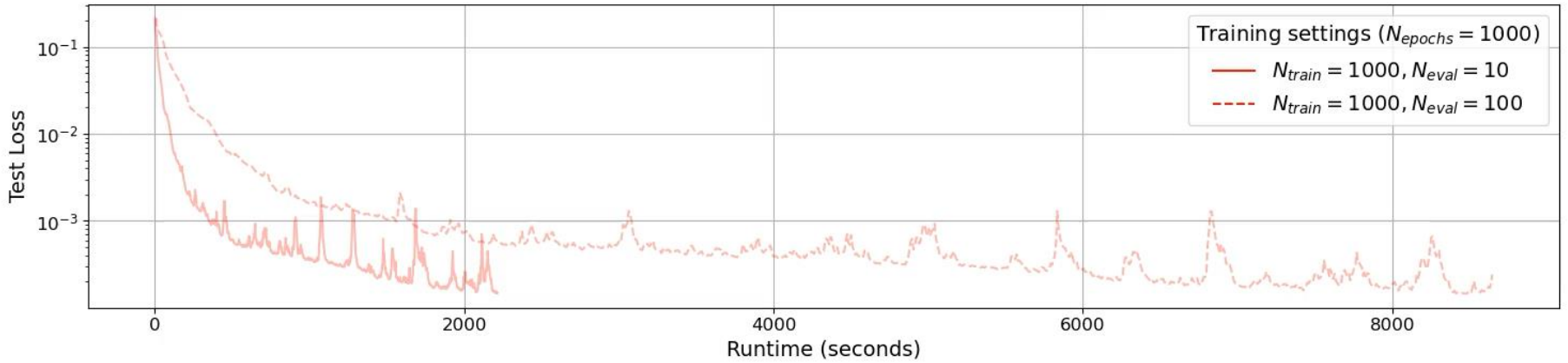
    Update parameters by computing $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$ (backpropagation)
  end for
end for
return $\boldsymbol{\theta}^*$ ▷ Return trained DeepONet parameters.

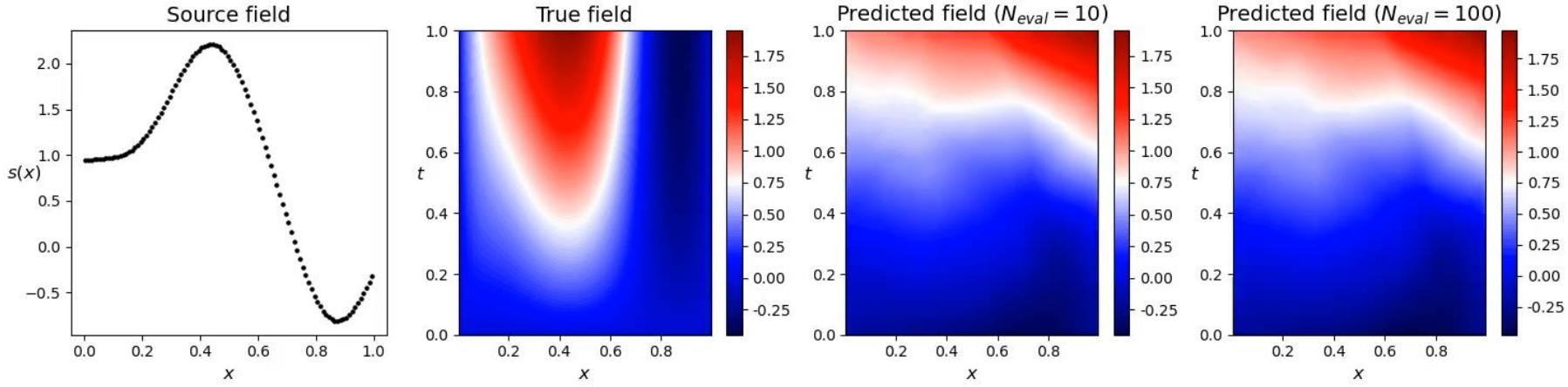
https://arxiv.org/pdf/2409.13280

Case	Dynamical System	Diffusion-reaction	Heat Equation
Model Output	$\frac{du}{dt} = s(t),$ $u(0) = 0$ and $t \in [0, 1]$ $\mathcal{G}_{\boldsymbol{\theta}} : s(t) \to u(t).$	$\frac{\partial u}{\partial t} = D\frac{\partial^2 u}{\partial x^2} + ku^2 + s(x),$ $D = 0.01,\ k = 0.01,$ $(t, x) \in (0, 1] \times (0, 1],$ $u(0, x) = 0,\ x \in (0, 1)$ $u(t, 0) = 0,\ t \in (0, 1)$ $u(t, 1) = 0,\ t \in (0, 1)$ $\mathcal{G}_{\boldsymbol{\theta}} : s(x) \to u(t, x).$	$-\nabla \cdot (a(\boldsymbol{x})\nabla u(\boldsymbol{x})) = 0,$ $\boldsymbol{x} = (x_1, x_2),$ $\boldsymbol{x} \in \Omega = [0, 1]^2,$ $u(0, x_2) = 1,\ u(1, x_2) = 0,$ $\frac{\partial u(x_1, 0)}{\partial n} = \frac{\partial u(x_1, 1)}{\partial n} = 0,$ $\mathcal{G}_{\boldsymbol{\theta}} : a(\boldsymbol{x}) \to u(\boldsymbol{x}).$
Input Function	$s(t) \sim \mathrm{GP}(0, k(t, t')),$ $\ell_t = 0.2,\ \sigma^2 = 1.0,\ t \in [0, 1]$ $k(t, t') = \sigma^2 \exp\left\{-\frac{\|t - t'\|^2}{2\ell_t^2}\right\}.$	$s(x) \sim \mathrm{GP}(0, k(x, x')),$ $\ell_x = 0.2,\ \sigma^2 = 1.0,$ $k(x, x') = \sigma^2 \exp\left\{-\frac{\|x - x'\|^2}{2\ell_x^2}\right\}.$	$\log(a(\boldsymbol{x})) \sim \mathrm{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$ $\mu(\boldsymbol{x}) = 0,\ \ell_{x_1} = 0.1,\ \ell_{x_2} = 0.15,\ \sigma^2 = 1.0$ $k(\boldsymbol{x}, \boldsymbol{x}') = \sigma^2 \exp\left\{-\sum_{i=1}^{2} \frac{\|x_i - x_i'\|^2}{2\ell_{x_i}^2}\right\}.$
Samples			

Diffusion reaction dynamics

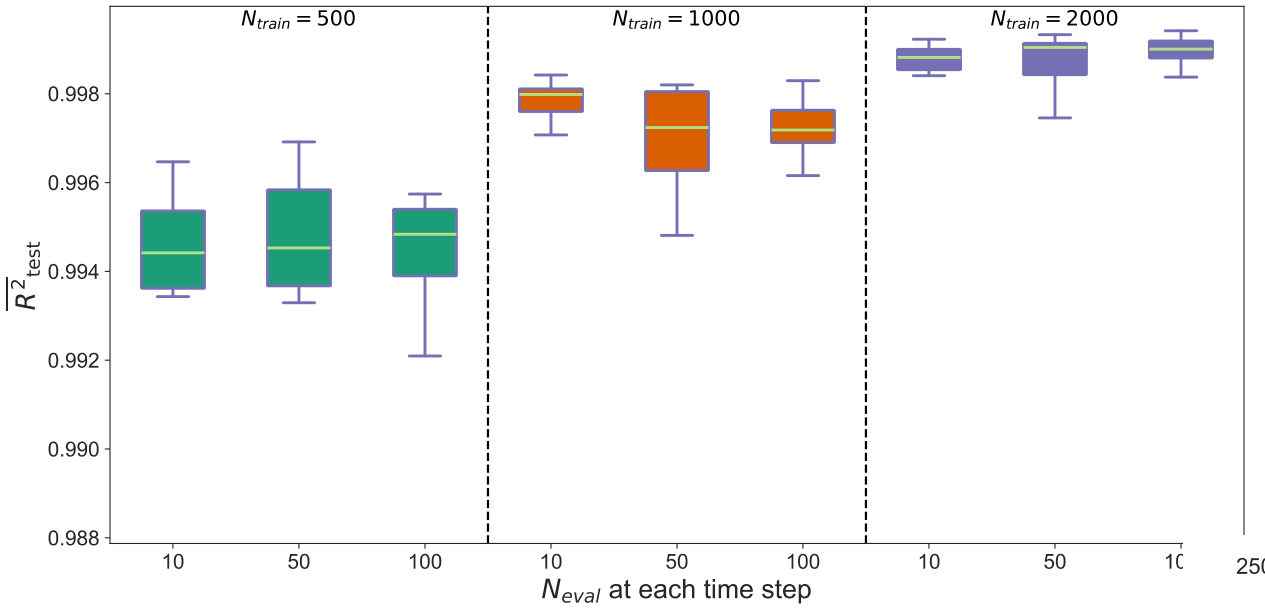




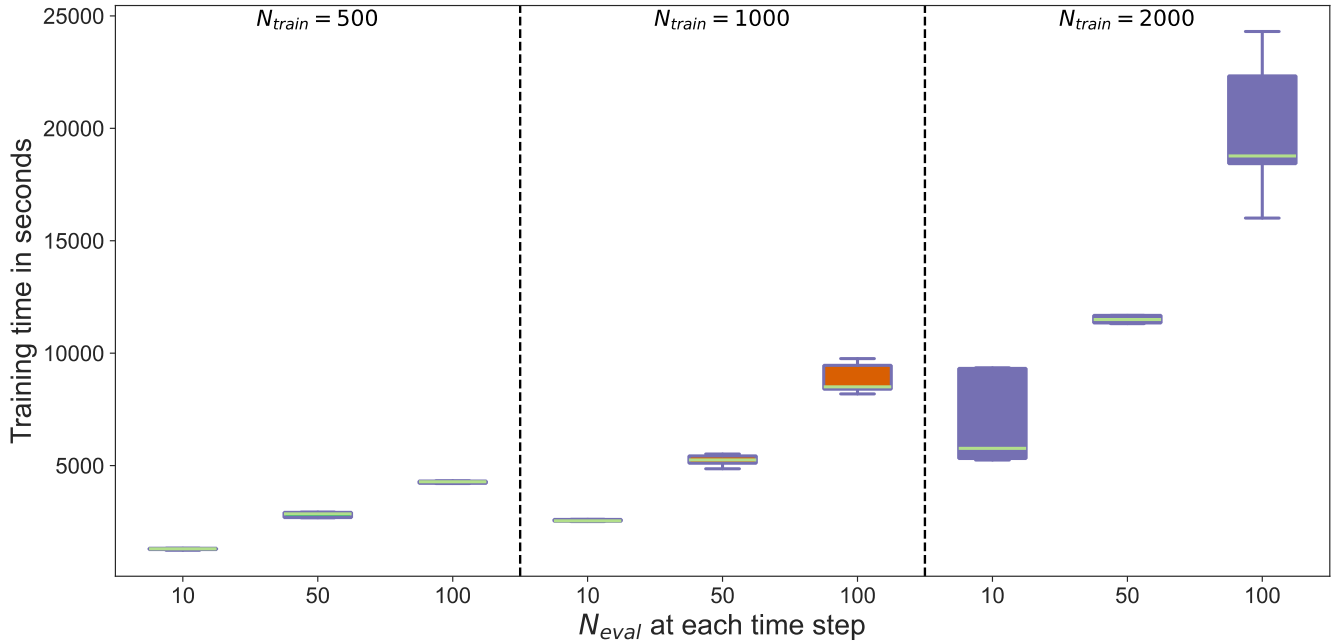


Diffusion reaction dynamics contd..

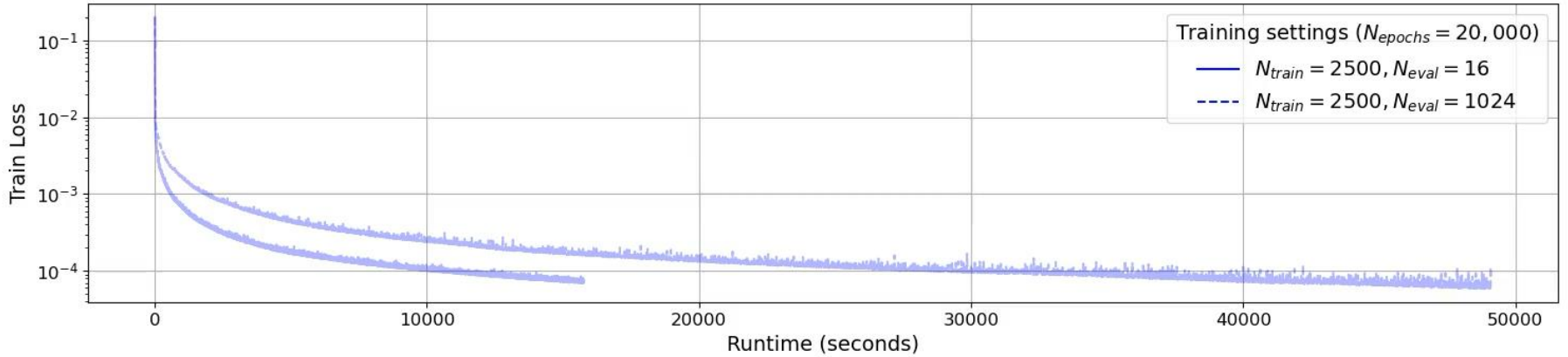
$\bar{R}^2{}_{\text{test}}$

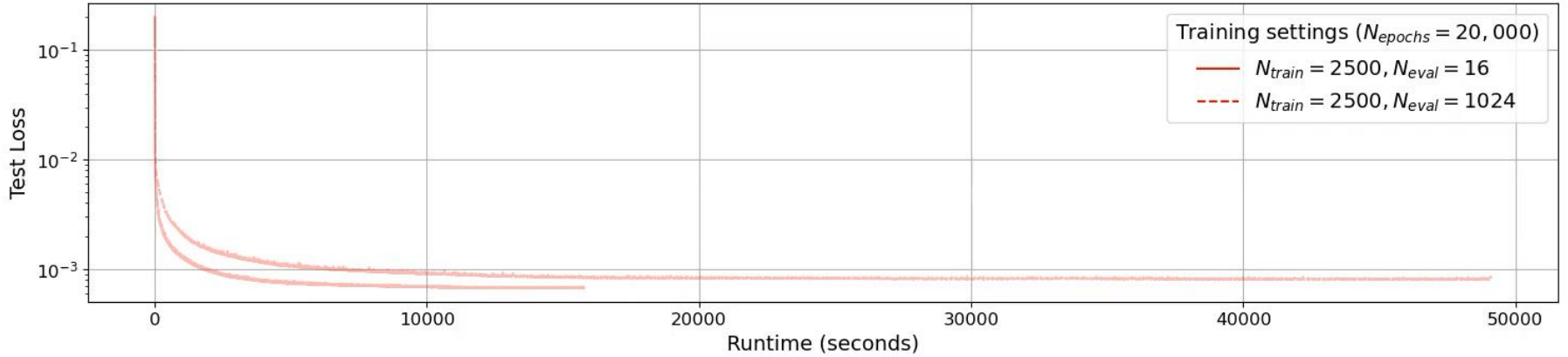


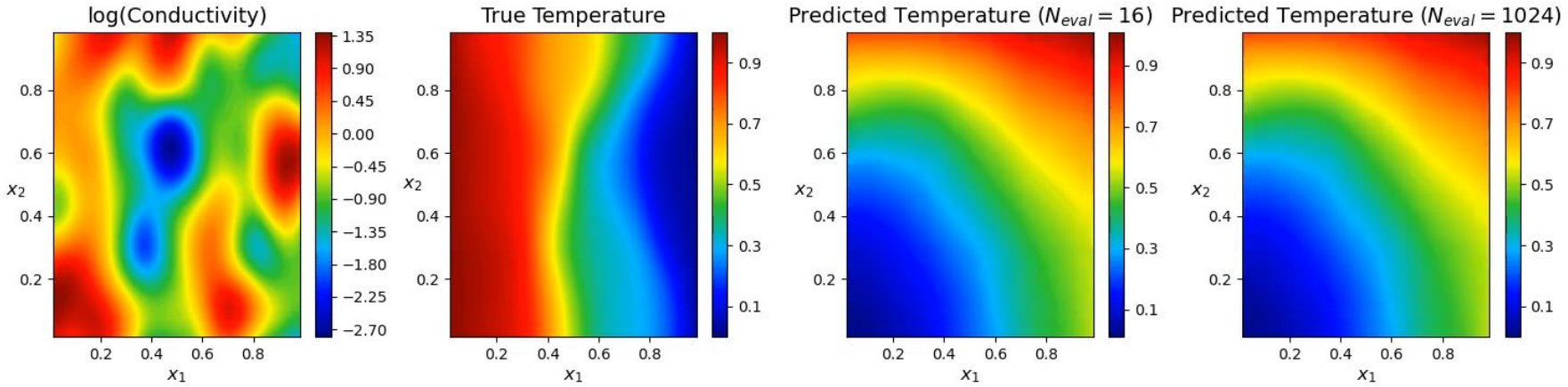
Training time



Heat equation

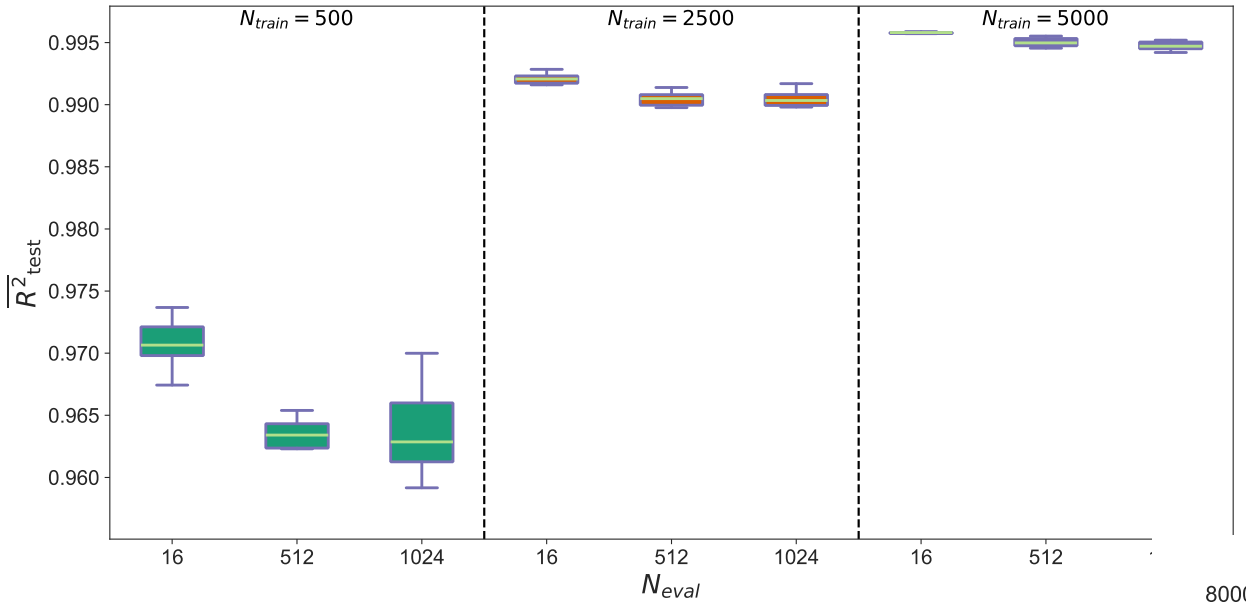




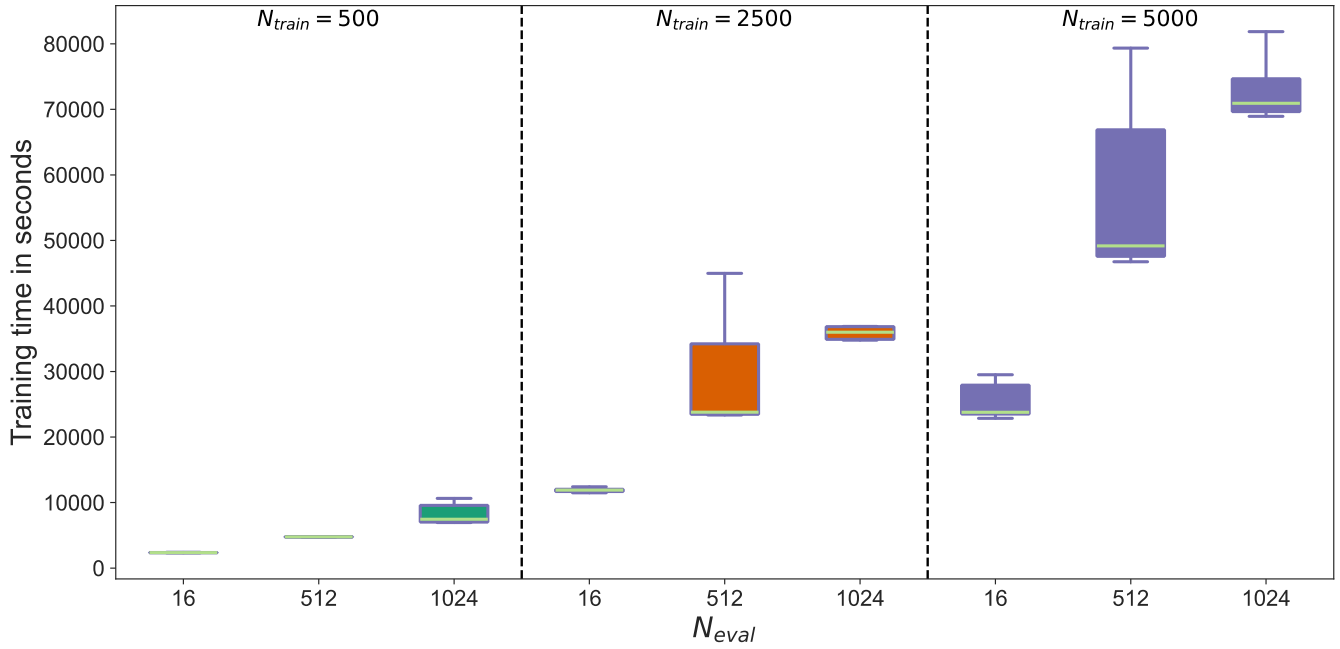


Heat equation contd..

$\bar{R}^2_{\text{test}}$

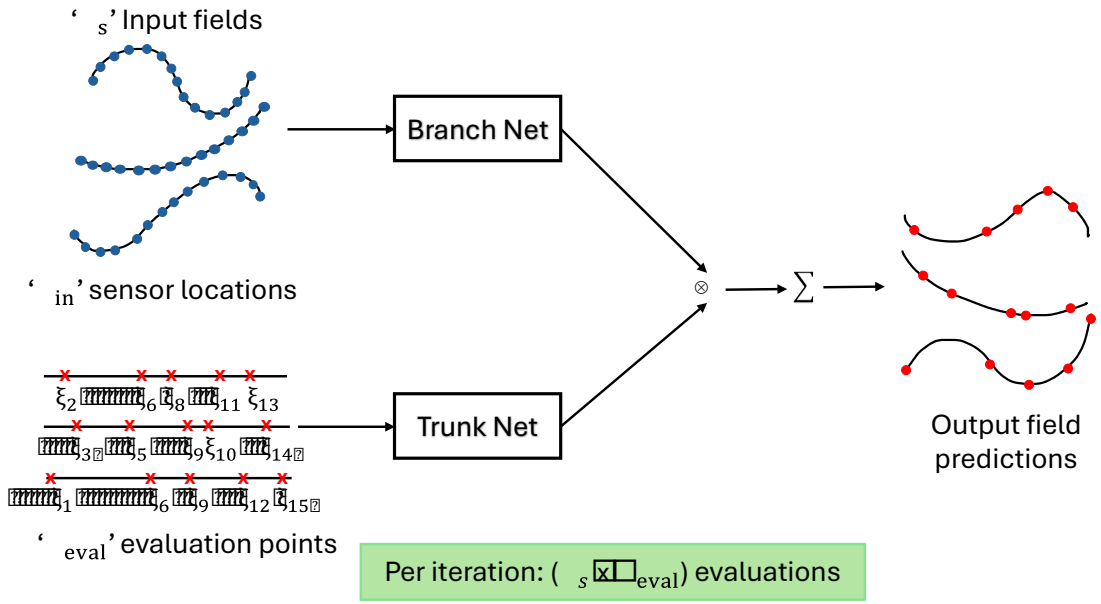


Training time



Take-away

The lower effective batch sizes with our randomized evaluation leads to flatter minima, improving generalization while reducing the training time.



Karumuri Sharmila, Lori Graham-Brady, and Somdatta Goswami. "Efficient Training of Deep Neural Operator Networks via Randomized Sampling." *arXiv preprint arXiv:2409.13280* (2024).

https://arxiv.org/pdf/2409.13280

https://github.com/Centrum-IntelliPhysics/DeepONet-Efficient-Training-with-Random-Sampling

References

1. Keskar, Nitish Shirish, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On large-batch training for deep learning: Generalization gap and sharp minima." *arXiv preprint arXiv:1609.04836* (2016).
2. Lu, Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. "Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators." *Nature machine intelligence* 3, no. 3 (2021): 218-229.
3. Lu, Lu, Xuhui Meng, Shengze Cai, Zhiping Mao, Somdatta Goswami, Zhongqiang Zhang, and George Em Karniadakis. "A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data." *Computer Methods in Applied Mechanics and Engineering* 393 (2022): 114778.

Thank you