NUNO: A General Framework for Learning Parametric PDEs with Non-Uniform Data

Songming Liu [1] **Zhongkai Hao** [1] **Chengyang Ying** [1] **Hang Su** [1 2] **Ze Cheng** [3] **Jun Zhu** [1 2]

Abstract

The neural operator has emerged as a powerful tool in learning mappings between function spaces in PDEs. However, when faced with real-world physical data, which are often highly non-uniformly distributed, it is challenging to use mesh-based techniques such as the FFT. To address this, we introduce the Non-Uniform Neural Operator (NUNO), a comprehensive framework designed for efficient operator learning with non-uniform data. Leveraging a K-D tree-based domain decomposition, we transform non-uniform data into uniform grids while effectively controlling interpolation error, thereby paralleling the speed and accuracy of learning from non-uniform data. We conduct extensive experiments on 2D elasticity, (2+1)D channel flow, and a 3D multi-physics heatsink, which, to our knowledge, marks a novel exploration into 3D PDE problems with complex geometries. Our framework has reduced error rates by up to 60% and enhanced training speeds by $2\times$ to $30\times$. The code is now available at `https://github.com/thu-ml/NUNO`.

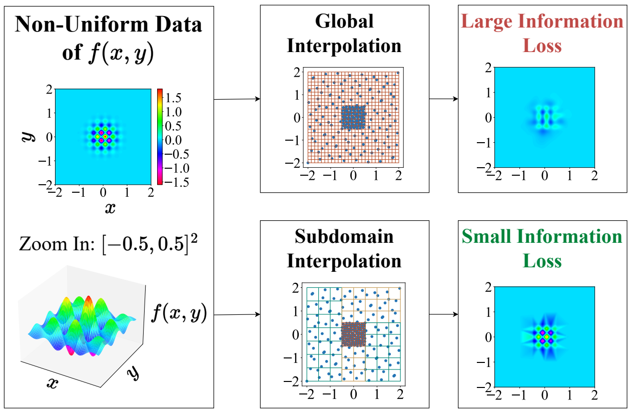


Figure 1: **Illustration of non-uniform data**. The physical quantity $f(x, y)$ varies greatly at the center but remains smooth in the far field, prompting a higher concentration of sensors (blue dots) in the center. Direct global interpolation may overlook these variations and yield a substantial error. Instead, our approach involves decomposing the domain into five subdomains and adaptively applying grids of varying sizes (denoted by different colored lines) to minimize error.

1. Introduction

In many fields such as fluid dynamics (Cebeci et al., 2005) and electromagnetism (de Castro et al., 2014), scientists and engineers often resort to numerical simulation rather than expensive experiments to study the behavior of physical systems under different parameters. However, traditional numerical methods can be too time-consuming for complex physical systems characterized by partial differential equations (PDEs) (Hao et al., 2022). Neural operator, a class of data-driven surrogate models, approximates the mapping from the parameter function to the solution of PDEs. When inference, one forward pass is several orders of magnitude faster than traditional numerical methods (Li et al., 2020a).

Among recently developed neural operators, mesh-based neural operators are typically fast and accurate, whose input and output are specified as uniform grids. For example, Fourier neural operator (FNO) (Li et al., 2020a) can efficiently extract highly nonlinear and nonlocal features and perform accurate predictions via the fast Fourier transform (FFT). Besides, a uniform grid is naturally compatible with a discretization of PDEs, which is the basis of many important works (Long et al., 2018; Gin et al., 2021; Liu et al., 2022). However, real-world physical data are usually highly non-uniformly distributed (see Figure 1 for an illustration), and mesh-based methods are difficult to apply due to large interpolation errors. To overcome this limitation, some attempts to develop mesh-less neural operators (Lu et al.,

[1]Dept. of Comp. Sci. and Tech., Institute for AI, THBI Lab, BNRist Center, Tsinghua-Bosch Joint ML Center, Tsinghua University [2]Pazhou Laboratory (Huangpu), Guangzhou, China [3]Bosch Center for Artificial Intelligence. Correspondence to: Jun Zhu <dcszj@tsinghua.edu.cn>.

Proceedings of the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

2021; Brandstetter et al., 2022) at the expense of degradation in speed and loss of astounding properties such as the FFT and the equivalence between convolution and differentiation (Long et al., 2018). Others have considered borrowing adaptive meshes from numerical methods to learn a bijection between point clouds (a general class of non-uniform data) and uniform grids (Li et al., 2022). But such an approach has not been theoretically justified yet and sometimes handcrafted designs are needed. In a word, the challenge of non-uniform problems is: how to achieve a good trade-off between speed (to efficiently process unstructured data) and accuracy (to control interpolation error).

In this paper, we propose a non-uniform neural operator (NUNO) to facilitate mesh-based neural operators over complex geometries and highly non-uniform data. Specifically, we design a K-D tree algorithm for domain decomposition, and then adaptively use different-size grids on different subdomains for interpolation. The interpolated subdomain grids are then used to train the underlying mesh-based model. In this way, we convert unstructured data into uniform grids under the premise of the controlled interpolation error, so that many fast mesh-based techniques can be used. In addition, different-size subdomain grids induce the neural network to learn multi-scale features and improve its abstraction ability.

In our experiments, we investigate three challenging problems of complex geometries, where non-uniform sensors are placed to capture complex physical phenomena: 2D elasticity from Li et al. (2022), (2+1)D channel flow adapted from Aparicio-Mauricio et al. (2020), and 3D heatsink from COMSOL application gallery (COMSOL AB, 2022), which is a fresh attempt to the 3D problems of complex geometries. The experimental results show that our method achieves state-of-the-art performance as well as fast speed, compared with comprehensive baselines including mesh-less and mesh-based neural operators (via global interpolation).

Contributions. In summary, the general contributions of this paper include

- We propose a general framework, a non-uniform neural operator (NUNO), for facilitating mesh-based operator learning with non-uniform data.
- We develop a K-D tree algorithm to iteratively divide the problem domain into several subdomains to reduce non-uniformity and subsequent interpolation error.
- Our method has achieved the best cost-accuracy trade-off. It lowers the error rates by 55% in 2D elasticity, by 61% in (2+1)D channel flow, and by 34% in 3D heatsink. At the same time, it achieves promising speedups: at most $30\times$ speedups compared with mesh-free baselines and $2\times$ to $4\times$ speedups compared with mesh-based baselines.

2. Related Work

In this section, we briefly discuss the related frontier work.

Traditional Numerical Methods. Nowadays, numerical methods are still the most prevalent ones to solve PDEs, involving the finite element method (FEM) (Reddy, 2019), finite volume method (FVM) (Moukalled et al., 2016), material point method (MPM), multi-grid methods (Hackbusch, 2013), etc. They suffer from the "curse of dimensionality" (Xue et al., 2020), despite their high accuracy and complete theoretical foundation. A moderately complex physical system can cost several minutes to simulate, rendering their infeasibility in scenarios like optimization where PDEs are solved frequently.

Mesh-based vs. Mesh-less Neural Operators. Mesh-based neural operators are a class of fast, flexible, and accurate neural operators in which both the input (parameter function) and output (solution to the PDEs) are discretized into uniform grids. Famous examples include convolutional neural operators (Gao et al., 2021; Khara et al., 2021), Fourier neural operator (FNO) (Li et al., 2020a), and Transformer neural operators (Cao, 2021; Hao et al., 2023). In contrast, mesh-less neural operators such as DeepONet (Lu et al., 2021) and graph neural operators (Sanchez-Gonzalez et al., 2020; Li et al., 2020b) allow for arbitrary inputs and queries, which can be used in non-uniform data. But they have lost the speed and efficient mesh-based techniques like the FFT. Besides, other lines of neural network-based PDE solvers involve physics-informed neural networks (PINNs) (Raissi et al., 2019) and neural fields (Sitzmann et al., 2020).

Deformable Meshes. Adaptive mesh methods (Babuška & Suri, 1990; Huang & Russell, 2010) come from the community of numerical methods, which can deform between irregular meshes and uniform meshes. Some have attempted to learn such a deformation to facilitate mesh-base neural operators (Li et al., 2022). However, in addition to the lack of theoretical guarantee, it is extremely difficult for a neural network to learn such a mapping, as there is no corresponding loss function to evaluate the progress and guide its learning towards a uniform grid.

3. Methodology

In this section, we will first give a formulation of the problem considered in this paper. We then present a K-D tree algorithm for domain decomposition, followed by an overall pipeline of our method, a non-uniform neural operator.

3.1. Problem Formulation

We consider a system of PDEs parameterized by some function $a(x) \in \mathcal{A}, x \in \Omega_a$. When the parameter is specified to be $a(x)$, the solution to the PDEs is denoted as $u(y) \in \mathcal{U}, y \in \Omega_u$. Here, $\mathcal{A}$ and $\mathcal{U}$ are two Banach function spaces with domains of definition $\Omega_a \subset \mathbb{R}^{d_a}$ and $\Omega_u \subset \mathbb{R}^{d_u}$, respectively, where d_a and d_u are two positive integers. Given a dataset $\{(a_j, u_j)\}_{j=1}^N$ where a follows some distribution μ and a, u are both represented as point clouds $a := \{a(x^{(i)})\}_{i=1}^M, u := \{u(y^{(i)})\}_{i=1}^M$[1], we hope to approximate the ground-truth operator $G^\dagger \colon \mathcal{A} \to \mathcal{U}$ or equivalently, $G^\dagger \colon a(x) \mapsto u(y)$ with a *mesh-based* neural operator $G_\theta, \theta \in \Theta$ for some parameter space Θ.

To capitalize on the speed and flexibility of mesh-based methods like the FFT, we interpolate the unstructured point clouds a and u into uniform grids, yielding vectors $\mathbf{a}$ and $\mathbf{u}$ respectively. Let ϕ denote a linear reconstruction or backward interpolation from the uniform grids to the point cloud. The prediction error can then be approximated using the following equation:

$$\begin{aligned}\|\phi(G_\theta(\mathbf{a})) - u\| \leq & \|\phi(G_\theta(\mathbf{a}) - \mathbf{u})\| + \|\phi(\mathbf{u}) - u\| \\ \leq & \|\phi\| \underbrace{\|G_\theta(\mathbf{a}) - \mathbf{u}\|}_{\text{prediction error}} + \underbrace{\|\phi(\mathbf{u}) - u\|}_{\text{interpolation error}},\end{aligned} \tag{1}$$

where $\|\cdot\|$ is the L^2 norm. The first term represents the prediction error of the neural operator, contingent upon the model's capability and the optimization's progression. Furthermore, the interpolation from a to $\mathbf{a}$, can also negatively influence this term. The information lost during the interpolation process can potentially degrade the model's performance. The second term signifies the error of the interpolation from $\mathbf{u}$ back to u, which can directly impact the overall accuracy. Upon this analysis, it is evident that interpolation errors hold significant sway over the total error.

Domain Decomposition. In cases of highly non-uniform data (refer to Figure 1 for an example), the interpolation error can at times overshadow the total error. In our endeavor to mitigate this particular error, we opt to decompose the domains Ω_a and Ω_u into several subdomains, expressed as $\Omega_a = \bigcup_{k=1}^{n_a} \Omega_a^{(k)}, \Omega_u = \bigcup_{k=1}^{n_u} \Omega_u^{(k)}$. Within these subdomains, we perform interpolation using varied grid sizes to procure uniform grids, denoted as $\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}, \{\mathbf{u}^{(k)}\}_{k=1}^{n_u}$. By judiciously decomposing the domain into subdomains, we can adaptively apply denser meshes in areas characterized by a high point density, and conversely, coarser meshes in areas where the points are more sparsely distributed.

Optimization Target. Formally speaking, with the domain decomposition strategy, our optimization target can be described as:

$$\min_{\theta \in \Theta} \mathbb{E}_{a \sim \mu} \left\| \phi \left(G_\theta(\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}) \right) - u \right\|, \tag{2}$$

[1]Here, we use the same notation for the function a, u and their point-cloud values, which might be seen as a notation abuse.

Algorithm 1 Domain Decomposition via a K-D Tree

```
1: Input: initial point cloud $D^{(0)}$ and the number of sub-point clouds $n$
2: Output: a sef of sub-point clouds $\mathcal{S}$
3: Initialize: $\mathcal{S} \leftarrow \{D^{(0)}\}$
4: repeat
5:    Choose $D^* = \arg\max_{D \in \mathcal{S}} (|D| \cdot \mathrm{KL}(P \parallel Q; D))$
6:    $\mathcal{S} \leftarrow \mathcal{S} - \{D^*\}$
7:    Select the dimension $k, 1 \leq k \leq d$, where the bounding box of $D^*$ has the largest scale
8:    Determine the hyperplane $x_k = b^*$, where $b^* = \arg\max_{b \in \mathcal{B}} \mathrm{Gain}(D^*, b)$ and $\mathcal{B}$ is a set of discrete candidates for $b$
9:    Partition $D^*$ with $x_k = b^*$
10:   $\mathcal{S} \leftarrow \mathcal{S} \cup \{D^*_{x_k > b^*}, D^*_{x_k \leq b^*}\}$
11: until $|\mathcal{S}| = n$
```

where $\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}$ is interpolated from a and G_θ takes $\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}$ as input and outputs a prediction for $\{\mathbf{u}^{(k)}\}_{k=1}^{n_u}$. Finally, the output of G_θ is interpolated back to the point cloud to obtain a prediction for u.

3.2. Domain Decomposition via a K-D Tree

Domain decomposition aims to approximate the distribution of points within a given point cloud by uniform distributions defined in a series of subdomains, which corresponds to varied uniform subdomain grids. Upon obtaining this approximation, we separately perform interpolation in each subdomain, using a grid size proportional to the number of points living in it to maintain a low interpolation error.

Let $D^{(0)} = \{x^{(i)} := (x_1^{(i)}, \dots, x_d^{(i)})\}_{i=1}^m$ be an initial point cloud, where d and m represents the dimensionality and the number of points in the point cloud, respectively. The optimization target of domain decomposition can be framed as minimizing the approximation error, which is quantified by the Kullback-Leibler (KL) divergence:

$$\min_{D^{(1)}, \dots, D^{(n)}} \sum_{i=1}^n \frac{|D^{(i)}|}{|D^{(0)}|} \mathrm{KL}(P \parallel Q; D^{(i)}), \tag{3}$$

where $\bigcup_{i=1}^n D^{(i)} = D^{(0)}$ constitutes a n-partition of $D^{(0)}$ with disjoint bounding boxes. A bounding box is defined as the smallest enclosure containing all the points. For instance, the bounding box for $D^{(0)}$ is given by $[x_{\min,1}, x_{\max,1}] \times \cdots \times [x_{\min,d}, x_{\max,d}]$, where $x_{\min,j} = \min\{x_j^{(i)}\}_{i=1}^m$ and $x_{\max,j} = \max\{x_j^{(i)}\}_{i=1}^m$. Additionally, $\mathrm{KL}(P \parallel Q; D)$ is the KL divergence between the distribution of points in a point cloud D, denoted by P, and the uniform distribution

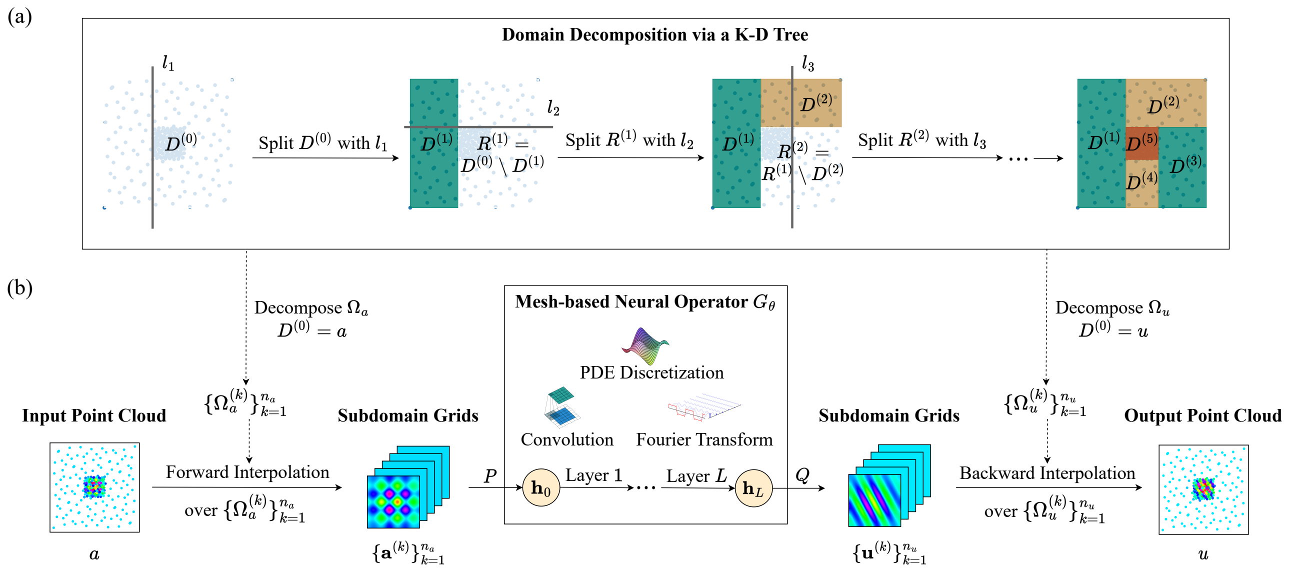

(a)
Domain Decomposition via a K-D Tree
l_1
$D^{(0)}$
Split $D^{(0)}$ with l_1
l_2
$D^{(1)}$
$R^{(1)} = D^{(0)} \setminus D^{(1)}$
Split $R^{(1)}$ with l_2
l_3
$D^{(2)}$
$R^{(2)} = R^{(1)} \setminus D^{(2)}$
Split $R^{(2)}$ with l_3
$D^{(5)}$
$D^{(4)}$
$D^{(3)}$
(b)
Decompose Ω_a $D^{(0)} = a$
Mesh-based Neural Operator G_θ
Decompose Ω_u $D^{(0)} = u$
PDE Discretization
Convolution
Fourier Transform
Input Point Cloud
$\{\Omega_a^{(k)}\}_{k=1}^{n_a}$
Subdomain Grids
Subdomain Grids
$\{\Omega_u^{(k)}\}_{k=1}^{n_u}$
Output Point Cloud
Forward Interpolation
over $\{\Omega_a^{(k)}\}_{k=1}^{n_a}$
P
$\mathbf{h}_0$
Layer 1
Layer L
$\mathbf{h}_L$
Q
Backward Interpolation
over $\{\Omega_u^{(k)}\}_{k=1}^{n_u}$
a
$\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}$
$\{\mathbf{u}^{(k)}\}_{k=1}^{n_u}$
u

Figure 2: **(a) Domain decomposition:** starting from $D^{(0)}$, each time we choose to split a sub-point cloud with a hyperplane l_i. After 4 iterations, we obtain 5 sub-point clouds distributed more uniformly within their bounding boxes. **(b) NUNO framework:** 1. interpolation to subdomain grids; 2. projection by P; 3. pass through the mesh-based neural operator with L layers ($\mathbf{h}_0, \ldots, \mathbf{h}_L$ indicate hidden embeddings of each layer); 4. projection by Q; 5. interpolation back to the point cloud.

within the bounding box of D, denoted by Q:

$$\mathrm{KL}(P \parallel Q; D) := \sum_j \frac{|D_j|}{|D|} \ln \left(\frac{|D_j|}{|D|} \Big/ \frac{1}{\prod_{i=1}^d N_i} \right). \tag{4}$$

In calculating P, we employ histogram density estimation. Specifically, we divide the bounding box of D uniformly into $N_1 \times \cdots \times N_d$ cells, and D_j is defined as $\{x \mid x \in D \wedge x \in \text{the } j\text{-th cell}\}$, for $1 \leq j \leq \prod_{i=1}^d N_i$.

While the optimization problem in Eq. (3) is reduced to an integer programming issue and is NP-hard, we have devised a heuristic method to approximate the solution. This method is inspired by the K-D tree algorithm, recognized for its quasilinear complexity and compatibility with arbitrary point clouds. In each iteration, we select a sub-point cloud D^* with the highest total KL divergence and split it using a hyperplane $x_k = b^*$. The parameter k is selected such that the bounding box of D^* has the largest scale. The chosen hyperplane maximizes the gain of the KL divergence, represented by $\mathrm{Gain}(D^*, b) :=$

$$\begin{aligned} \mathrm{KL}(P \parallel Q; D^*) &- \frac{|D^*_{x_k > b}|}{|D^*|} \mathrm{KL}(P \parallel Q; D^*_{x_k > b}) \\ &- \frac{|D^*_{x_k \leq b}|}{|D^*|} \mathrm{KL}(P \parallel Q; D^*_{x_k \leq b}), \end{aligned} \tag{5}$$

where $D^*_{x_k > b}, D^*_{x_k \leq b}$ are defined as $\{x \mid x \in D^* \wedge x_k > b\}$ and $\{x \mid x \in D^* \wedge x_k \leq b\}$, respectively. Through the recursive application of this process, we can partition $D^{(0)}$ into n sub-point clouds, decreasing the approximation error. We provide an outline of the algorithm in Algorithm 1 and showcase it in Figure 2a.

Time and Space Complexity. Under proper assumptions, the time and space complexity are, respectively, derived as $\mathcal{O}(nm \log m + nd)$ and $\mathcal{O}(md)$, where n is the number of sub-point clouds, m is the number of points in the initial point cloud, and d is the dimensionality. We leave a detailed analysis in Appendix A.1. In practice, the value of n acts as a cost-accuracy trade-off. As discussed in Section 4, we have empirically found that $n \leq 10^2$ are sufficient for general complex systems. In this way, the algorithm takes only a few seconds, which is completely negligible compared to the training time of neural operators.

3.3. Non-Uniform Neural Operator

In this subsection, we elucidate our framework for the non-uniform neural operator, designed to synergize rapid computation with high precision. As illustrated in Figure 2b, the framework can be methodically deconstructed into five primary steps:

1. Initially, the input point-cloud values, denoted as a, are interpolated into input subdomain grids which are subsequently harmonized to a uniform size. The resulting dimension from this step is denoted as $n_a \times s$, where

s represents the padded mesh size.

2. The subdomain grids are subsequently transformed into hidden embeddings of dimension $n_h \times s$ via a mapping function $P\colon \mathbb{R}^{n_a} \to \mathbb{R}^{n_h}$. Here, n_h symbolizes the hidden width. This mapping function is parameterized by a shallow fully connected layer, with parameters that are shared across all mesh locations.

3. These generated hidden embeddings are then propagated through the hidden layers of the underlying mesh-based neural operator.

4. Post this operation, the hidden embeddings are transformed into the output subdomain grids of size $n_u \times s$, facilitated by another mapping function $Q\colon \mathbb{R}^{n_h} \to \mathbb{R}^{n_u}$. This function is also parameterized by a shallow fully connected layer, with parameters shared across all mesh locations.

5. Lastly, these output subdomain grids are interpolated back into the output point cloud u, which constitutes the final prediction.

Outstanding Cost-Accuracy Balance. By transforming the point cloud into structured data through domain decomposition, we not only maintain a handle on the interpolation error but also expedite operations such as convolution and the FFT, bolstering our ability to extract non-local and non-linear features. Additionally, for the same total size, the subdomain grid is of smaller size compared to the global grid. As a result, the size of hidden embeddings (i.e., s) diminishes, yielding a more compact representation and thus a significantly lower space-time cost.

Versatility. Our framework exhibits compatibility with all mesh-based neural operators, necessitating no additional modifications for implementation. Moreover, domain decomposition facilitates flexibility in usage. For instance, we can execute different normalizations across varied subdomains, an approach particularly beneficial for multi-scale and discontinuous problems (Pang et al., 2020).

Alignment. We note that the subdomain grids $\{\mathbf{a}^{(k)}\}_{k=1}^{n_a}$ may differ in size, presenting challenges for batching and propagation. This issue is circumvented by aligning the grids to a uniform size. This alignment can be achieved by employing size-insensitive structures like the SPP-Net (He et al., 2015) or the Transformer (Vaswani et al., 2017) to generate embeddings of a consistent size. Alternatively, we could directly resize all the subdomain grids to match in size using up/down sampling or FFT-IFFT methods, a common technique in computer vision that we utilize in our implementation. Further details are elaborated in Appendix A.2.

4. Experiments

We consider a benchmark problem of 2D elasticity in Li et al. (2022) (Section 4.2), a (2+1)D problem of channel flow (Navier-Stokes equations, Section 4.3), and a challenging 3D multi-physics problem of heatsink (Section 4.4) to showcase the efficiency as well as the accuracy of our method against mesh-less and mesh-based baselines. We refer to Appendix A.2 for experimental details.

4.1. Experiment Setup

Evaluation. Consistent with previous work (Li et al., 2020a; 2022; Lu et al., 2022), we use L^2 relative error (abbreviated as L^2RE) to evaluate the performance of models, which can be described as:

$$L^2\mathrm{RE} := \frac{1}{N}\sum_{j=1}^{N}\sqrt{\frac{\sum_{i=1}^{M}\left(u_j(y^{(i)}) - \hat{u}_j(y^{(i)})\right)^2}{\sum_{i=1}^{M} u_j(y^{(i)})^2}}, \quad (6)$$

where $y^{(i)}$ is the coordinate and $u_j, \hat{u}_j$ are the ground truth and prediction, respectively. For each experiment, we report the mean and 95% confidence intervals of L^2 relative error after 5 independent runs.

Mesh-based Benchmarks: **U-Net:** a famous model for approximating image-to-image mappings with spatial convolutions and deconvolutions (Ronneberger et al., 2015). **FNO:** a powerful neural operator with FFT-based spectral convolutions (Li et al., 2020a). **MWNO:** a novel neural operator with multiwavelet-based spectral convolutions (Gupta et al., 2021). **NU-NO:** the newly proposed framework for facilitating mesh-based neural operators with non-uniform inputs. In the following, for example, "NU-FNO" indicates that our framework adopts an FNO as the underlying model.

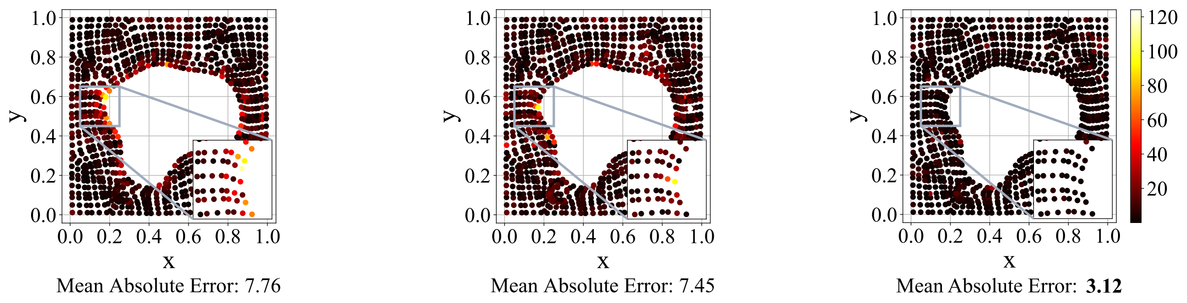
Mesh-less Benchmarks: **DeepONet:** a neural operator that takes the coordinate and the parameter function as input (Lu et al., 2021). **GraphNO:** a neural operator where both input and output are represented as graphs (Li et al., 2020b). **Geo-FNO:** an improved version of FNO that learns to deform the input irregular mesh into a uniform mesh (Li et al., 2022). **PointNet:** a well-known deep learning model for large 3D point clouds (Qi et al., 2017).

4.2. 2D Elasticity

In our first experiment, we revisit the benchmark of 2D hyper-elastic material in Li et al. (2022). The governing equation is given by:

$$\rho^s \frac{\partial^2 \boldsymbol{u}}{\partial t^2} + \nabla \cdot \boldsymbol{\sigma} = 0, \quad (7)$$

where $\boldsymbol{\sigma}$ denotes the stress tensor. The geometry Ω is a unit square with a void of arbitrary shape at the center as



(a) FNO (global interpolation) (b) Geo-FNO (learned) (c) NU-FNO (**ours**)

Figure 3: Visualization of 2D elasticity: the absolute prediction error of $\boldsymbol{\sigma}$ at point-cloud locations.

Table 1: Experimental results of 2D elasticity. R mesh and O mesh are adaptive meshes (Li et al., 2022).

Method	**Mesh Size**	**Training Time**		L^2 **Relative Error** ($\times 10^{-2}$)	
		per Epoch	**per Run**	**Training**	**Testing**
NU-FNO (**ours**)	1024[1]	2.3 s	19.6 min	$\mathbf{1.68 \pm 0.08}$	$\mathbf{1.93 \pm 0.11}$
FNO (global interpolation)	1681	**1.2** s	**9.7** min	3.41 ± 0.08	6.16 ± 0.16
U-Net (global interpolation)	1681	2.3 s	18.8 min	1.74 ± 0.02	6.82 ± 0.06
Geo-FNO (R mesh)	1681	1.7 s	14.2 min	3.53 ± 0.09	5.03 ± 0.09
Geo-FNO (O mesh)	1353	2.0 s	16.4 min	4.12 ± 0.16	4.28 ± 0.15
Geo-FNO (learned)	meshless	2.3 s	19.1 min	2.07 ± 0.99	4.31 ± 2.24
GraphNO	meshless	96.8 s	5.4 h	17.5 ± 1.26	16.9 ± 1.28
DeepONet	meshless	40.0 s	11.0 h	2.80 ± 0.13	12.0 ± 0.16

[1] In this paper, all mesh sizes mentioned for our method specifically refer to the *combined total number* of points across all subdomains, rather than the number of points within each individual subdomain.

shown in Figure 3. Our objective is to learn an operator capable of mapping the shape of the void to the values of $\boldsymbol{\sigma}$ within a point cloud of approximately 1000 points. For more comprehensive settings, we direct the reader to the work of Li et al. (2022).

We reproduce the baselines in Li et al. (2022) with 5 runs and employ the proposed framework in FNO (denoted as "NU-FNO") to benchmark against "Geo-FNO". Here, the geometry Ω varies within a limited range for different inputs. So, we stack all point clouds to generate a unified domain decomposition of 6 subdomains for all (1000 training and 200 testing) data, which takes 50 seconds.

Results. The results are reported in Table 1. Because of reduction in interpolation error, our non-uniform framework has paralleled the accuracy and efficiency. Specifically, as shown in Figure 3 (one of testing data, 1 run, random seed is 2023), FNO has a large prediction error at the edge of the void, because $\boldsymbol{\sigma}$ changes sharply here, and rough grids cannot capture such fine features. In contrast, our method uses grids of different densities in different subdomains, which can better capture features of various scales and achieve high accuracy even in places with a high variation of $\boldsymbol{\sigma}$.

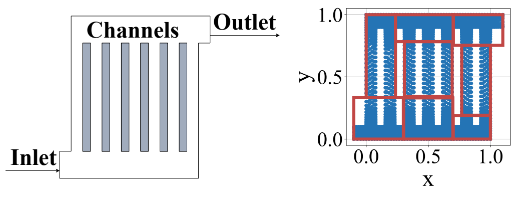
4.3. (2+1)D Channel Flow

We consider the 2D time-dependent incompressible flow in a row of channels adapted from Aparicio-Mauricio et al. (2020), which is governed by the notoriously hard-to-solve (incompressible) Navier-Stokes equations:

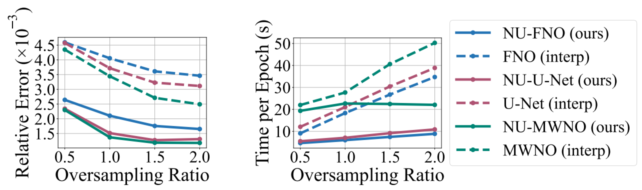
$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} &= -\nabla p + \nu \nabla^2 \boldsymbol{u}, \\ \nabla \cdot \boldsymbol{u} &= 0, \end{aligned} \tag{8}$$

where $x \in \Omega, t \in [0, 0.3)$ are space and time coordinates, respectively, $\boldsymbol{u} = (u(x,t), v(x,t))$ is the velocity field, $p = p(x,t)$ is the pressure, and ν is the viscosity. As displayed in Figure 4a, the geometry $\Omega \subset \mathbb{R}^2$ consists of an inlet, an outlet, and a row of channels. For this case, the target operator is defined to be a mapping: $(u,v,p)|_{x\in\Omega, t\in[0,0.15)} \mapsto (u,v,p)|_{x\in\Omega,t\in[0.15,0.3)}$.

We produce a dataset consisting of 1000 training instances and 200 test instances. The temporal interval $[0, 0.3)$ is discretized into 30 distinct steps, whereas the spatial domain is represented by a point cloud comprising 3809 points. Consequently, the total count of points within the point cloud amounts to $30 \times 3809 = 114,270$. Pertaining to mesh-free baselines, the inputs comprise the (u,v,p) values situated at the locations defined by the point cloud during



(a) The geometry, point cloud, and subdomains



(b) L^2 relative error (average) and training time per epoch

Figure 4: Visualization of (2+1)D channel flow. **(a)** Gray solid rectangles (left) indicate walls between channels while red hollow rectangles (right) indicate bounding boxes of point clouds on each subdomain. **(b)** Each value of the oversampling ratio is run only once, and the random seed is 2023. The "interp" is an abbreviation of "global interpolation".

Table 2: Experimental results of (2+1)D channel flow ($\text{oversampling ratio} = 1.5$).

Method	**Training Time per Epoch** (s)	**Testing L^2 Relative Error** ($\times 10^{-3}$)			
		u	v	p	**Average**
NU-FNO (**ours**)	**7.2**	3.04 ± 0.13	1.34 ± 0.04	0.83 ± 0.03	1.74 ± 0.05
FNO (global interpolation)	26.9	5.92 ± 0.25	3.52 ± 0.09	1.39 ± 0.24	3.61 ± 0.19
NU-U-Net (**ours**)	8.6	$\mathbf{2.18 \pm 0.04}$	1.11 ± 0.04	0.53 ± 0.03	1.27 ± 0.03
U-Net (global interpolation)	29.5	5.50 ± 0.25	3.41 ± 0.17	0.97 ± 0.03	3.29 ± 0.15
NU-MWNO (**ours**)	22.9	2.26 ± 0.08	$\mathbf{1.05 \pm 0.07}$	$\mathbf{0.35 \pm 0.05}$	$\mathbf{1.22 \pm 0.06}$
MWNO (global interpolation)	40.6	4.57 ± 0.99	2.76 ± 0.81	0.80 ± 0.14	2.71 ± 0.63
Geo-FNO (learned)	114.9	2.78 ± 1.31	2.75 ± 1.00	0.93 ± 0.41	2.15 ± 0.83
GraphNO	193.6	44.79 ± 0.00	44.17 ± 0.00	22.08 ± 0.01	37.02 ± 0.00
DeepONet	235.8	96.86 ± 13.45	234.75 ± 51.51	27.98 ± 6.82	119.86 ± 15.82

the time interval $t \in [0, 0.15)$. For mesh-based baselines, the input data is constituted by an interpolated mesh of size $\text{oversampling ratio} \times 3809$ during $t \in [0, 0.15)$. In contrast, the input for our proposed method consists of meshes spanning 8 subdomains (refer to Figure 4a), with a total size of $\text{oversampling ratio} \times 3809$ during the same time interval. Given that Ω remains invariant w.r.t. the input, we only necessitate a single domain decomposition, requiring a mere 0.5 seconds.

Results. The testing results are given in Table 2. Compared with mesh-based and mesh-free baselines, our method has achieved both state-of-the-art performance and the highest speed. This is because our method uses a grid adaptive to the point cloud distribution density in each subdomain, which reduces the interpolation error and encourages the neural operators to extract multi-scale features. Since the grid size of the subdomain is much smaller than that from global interpolation, the size of hidden embeddings is also reduced (the number of hidden-layer channels remains the same), allowing for a more compact representation and higher training speed.

Table 3: Results of an ablation study on the number of subdomains ($\text{oversampling ratio} = 1.5$). We run each choice once with the random seed 2023.

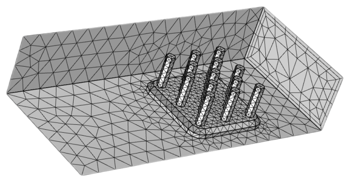
	Method	**Number of Subdomains**			
		4	**8**	**12**	**16**
L^2 **Rel. Err.**	NU-FNO	1.92	1.71	1.31	**0.85**
($\times 10^{-3}$)	NU-U-Net	1.33	1.27	1.21	**0.88**
Average	NU-MWNO	1.23	1.19	1.06	**0.72**
Training	NU-FNO	9.5	7.2	6.7	**5.7**
Time per	NU-U-Net	11.2	8.6	8.0	**6.9**
Epoch (s)	NU-MWNO	27.6	23.9	23.1	**22.7**

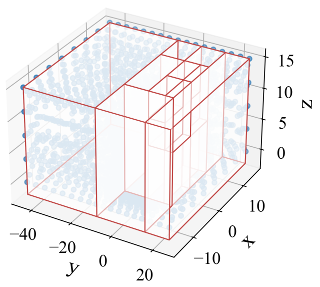
Mesh Size. We re-run mesh-based methods under different mesh sizes (or equivalently, oversampling ratios). As shown in Figure 4b, even with $\text{oversampling ratio} = 0.5$, our method significantly outperforms global interpolation with $\text{oversampling ratio} = 2.0$. In addition, we also find that the error decline of global interpolation slows down as the grid gets denser, showing a trend that it is difficult to outperform our method. This gap may be caused by the stronger ability of our method to extract multi-scale features. Besides, due to the compact representation, the time cost of our method grows far slower with mesh size than with global interpolation.

Ablation Study. We evaluated our methods (with various underlying models) under a range of choices for the number of subdomains. From the results shown in Table 3, we unexpectedly discover both an increase in accuracy and speed as

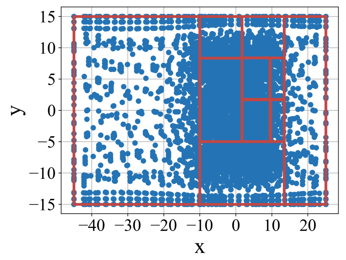
Table 4: Experimental results of 3D heatsink.

Method	**Parameters**	**Training Time per Epoch**	**Training Time per Run**	**Testing L^2 Relative Error** $(\times 10^{-2})$
NU-FNO (**ours**)	$3,285,856$	**4.7** s	**38.9** min	$\mathbf{5.09 \pm 0.48}$
FNO (global interpolation)	$3,283,741$	6.2 s	51.7 min	7.68 ± 0.25
NU-U-Net (**ours**)	$22,551,792$	5.2 s	43.4 min	6.31 ± 0.31
U-Net (global interpolation)	$22,549,857$	7.4 s	1.0 h	8.27 ± 0.14
NU-MWNO (**ours**)	$6,148,715$	18.3 s	2.5 h	5.36 ± 0.24
MWNO (global interpolation)	$6,145,565$	21.0 s	2.9 h	7.38 ± 0.12
PointNet	$1,677,221$	45.3 s	6.3 h	56.42 ± 27.60


(a) Complex geometry of 3D heat sink



(b) Point cloud and subdomains (3D)



(c) Point cloud and subdomains (2D)

Figure 5: Visualization of 3D heatsink. **(a)** The model of heatsink comes from real-world application. **(b)** Red lines indicate the borders of subdomains. **(c)** 2D projection (to the XY plane) of the 3D point cloud and subdomains.

the number of subdomains increases, which seems like the "blessing of subdomains". We attribute this to the following reasons: firstly, an increase in the number of subdomains results in lower interpolation error and yields finer features, thereby enhancing performance. Secondly, the most time-intensive step in our framework is the third step, which involves the hidden layers of the neural operator (as detailed in Section 3.3). As we have maintained a fixed total mesh size, an increase in the number of subdomains leads to a decrease in the mesh size for each individual subdomain (i.e., $s \downarrow$). Given a fixed hidden width n_h, this results in a smaller size for the hidden embeddings (i.e., $n_h \times s \downarrow$), thereby accelerating the forward pass. By expanding the number of subdomains, we reduce redundancy in the hidden embeddings, enabling a more compact representation.

4.4. 3D Heatsink

As a fresh attempt in the field of the neural operator, we consider a 3D heatsink from the COMSOL application gallery (COMSOL AB, 2022) in the last experiment. The governing equations consist of not only Navier-Stokes equations for fluid dynamics modeling, but also a series of equations for heat transfer modeling: conduction, convection, radiation, and the heat equation. For brevity, we refer to Haertel et al. (2015) for details of governing equations. The geometry $\Omega \subset \mathbb{R}^3$ is a rectangular enclosure with a heatsink inside as shown in Figure 5a. The physical quantities contain the velocity field $\boldsymbol{u} = (u(x), v(x), w(x)), x \in \Omega$, the pressure field $p = p(x)$, and the temperature field $T = T(x)$. Our interested operator maps the velocity field $\boldsymbol{u}$ to the resulting temperature T.

We generate 900 training and 100 testing data. The 3D point cloud contains $19,517$ points. Since previously considered mesh-free neural operators scale badly to 3D point clouds, we consider another mesh-free model, PointNet, which is designed for analyzing large 3D point clouds. For mesh-based baselines, the input is the interpolated mesh of the size aligned with the point cloud size and of the same aspect ratio as the geometry Ω. For our method, as shown in Figure 5b and 5c, the geometry is decomposed into 16 subdomains, taking 8 seconds. Subsequently, the point cloud is interpolated into uniform grids over subdomains, which are used for training our model.

Results. The challenge of this problem is that higher spatial degrees of freedom will significantly slow down the mesh-free neural operators, and point clouds that are non-uniformly distributed in 3D space are more difficult to be accurately interpolated to a uniform grid. Through domain decomposition, our method utilizes a set of grids with varied sizes to approximate the distribution of the point cloud, which is much more efficient than global interpolation. This is why our method is so accurate and high-speed in this 3D problem, as reported in Table 4. Moreover, we note that the parameter size of our method is only slightly larger than the original baseline. This is because when applying our framework, only additional projections P and Q are needed, whose parameter size is negligible.

5. Conclusion and Future Work

In this paper, we propose a framework named non-uniform neural operator (NUNO) to handle the challenge of non-uniform data in real-world applications. Through K-D tree-based domain decomposition and subsequent interpolation, we convert non-uniform data into uniform grids, allowing for fast mesh-based techniques such as the FFT while persevering a low interpolation error. In experiments, we benchmark on 2D elasticity, (2+1)D channel flow, and 3D heatsink. Results show that our framework has achieved both state-of-the-art accuracy and promising efficiency. Even at the coarsest resolution, our method still outperforms mesh-based baselines which are at the finest resolution.

3D Complex Geometry. Our work is a first work to investigate PDE problems of 3D complex geometries. However, there is still a lot of work to be done to identify and address potential challenges. For example, we may learn from computer graphics and design special architectures for large point clouds. **Uniform Grids of Varied Sizes.** In this work, we adopt the simplest technique, i.e., resizing, to deal with uniform grids of different sizes. It is interesting to explore other approaches such as SPP-Net (He et al., 2015). **Variable-Structure Problems.** One of the limitations of our work is that it is only applicable to the problems where the geometry does not change with the input a or does change within a limited range. In the other cases, we must generate different domain divisions for different inputs, to which the projections P and Q cannot generalize. As a future direction, we will try to combine our framework with GNNs and others, expediting applications in relevant fields such as structural optimization (Christensen & Klarbring, 2008).

Acknowledgements

This work was supported by the National Key Research and Development Program of China (2020AAA0106000, 2020AAA0104304), NSFC Projects (Nos. 62061136001, 62076145, 62076147, U19B2034, U1811461, U19A2081, 61972224), BNRist (BNR2022RC01006), Tsinghua Institute for Guo Qiang, and the High Performance Computing Center, Tsinghua University. J.Z is also supported by the New Cornerstone Science Foundation through the XPLORER PRIZE.

References

Aparicio-Mauricio, G., Rodríguez, F. A., Pijpers, J. J., Cruz-Díaz, M. R., and Rivero, E. P. Cfd modeling of residence time distribution and experimental validation in a redox flow battery using free and porous flow. *Journal of Energy Storage*, 29:101337, 2020.

Babuška, I. and Suri, M. The p-and hp versions of the finite element method, an overview. *Computer methods in applied mechanics and engineering*, 80(1-3):5–26, 1990.

Brandstetter, J., Worrall, D., and Welling, M. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

Cao, S. Choose a transformer: Fourier or galerkin. *Advances in Neural Information Processing Systems*, 34:24924–24940, 2021.

Cebeci, T., Shao, J. P., Kafyeke, F., and Laurendeau, E. *Computational fluid dynamics for engineers: from panel to Navier-Stokes methods with computer programs*. Springer, 2005.

Christensen, P. W. and Klarbring, A. *An introduction to structural optimization*, volume 153. Springer Science & Business Media, 2008.

COMSOL AB. Comsol multiphysics® v. 6.1, 2022. URL `https://www.comsol.com`.

de Castro, A. B., Gómez, D., and Salgado, P. *Mathematical models and numerical simulation in electromagnetism*, volume 74. Springer, 2014.

Gao, H., Sun, L., and Wang, J.-X. Phygeonet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428: 110079, 2021.

Gin, C. R., Shea, D. E., Brunton, S. L., and Kutz, J. N. Deepgreen: Deep learning of green's functions for nonlinear boundary value problems. *Scientific reports*, 11(1):1–14, 2021.

Gupta, G., Xiao, X., and Bogdan, P. Multiwavelet-based operator learning for differential equations. *Advances in Neural Information Processing Systems*, 34:24048–24062, 2021.

Hackbusch, W. *Multi-grid methods and applications*, volume 4. Springer Science & Business Media, 2013.

Haertel, J. H. K., Engelbrecht, K., Lazarov, B. S., and Sigmund, O. Topology optimization of thermal heat sinks. In *Proceedings of COMSOL conference*, volume 2015, 2015.

Hao, Z., Liu, S., Zhang, Y., Ying, C., Feng, Y., Su, H., and Zhu, J. Physics-informed machine learning: A survey on problems, methods and applications. *arXiv preprint arXiv:2211.08064*, 2022.

Hao, Z., Ying, C., Wang, Z., Su, H., Dong, Y., Liu, S., Cheng, Z., Zhu, J., and Song, J. Gnot: A general neural operator transformer for operator learning. *arXiv preprint arXiv:2302.14376*, 2023.

He, K., Zhang, X., Ren, S., and Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

Huang, W. and Russell, R. D. *Adaptive moving mesh methods*, volume 174. Springer Science & Business Media, 2010.

Khara, B., Balu, A., Joshi, A., Krishnamurthy, A., Sarkar, S., Hegde, C., and Ganapathysubramanian, B. Field solutions of parametric pdes. 2021.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020a.

Li, Z., Kovachki, N., Azizzadenesheli, K., Liu, B., Bhattacharya, K., Stuart, A., and Anandkumar, A. Neural operator: Graph kernel network for partial differential equations. *arXiv preprint arXiv:2003.03485*, 2020b.

Li, Z., Huang, D. Z., Liu, B., and Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. *arXiv preprint arXiv:2207.05209*, 2022.

Liu, X.-Y., Sun, H., and Wang, J.-X. Predicting parametric spatiotemporal dynamics by multi-resolution pde structure-preserved deep learning. *arXiv preprint arXiv:2205.03990*, 2022.

Long, Z., Lu, Y., Ma, X., and Dong, B. Pde-net: Learning pdes from data. In *International Conference on Machine Learning*, pp. 3208–3216. PMLR, 2018.

Lu, L., Jin, P., Pang, G., Zhang, Z., and Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Lu, L., Meng, X., Cai, S., Mao, Z., Goswami, S., Zhang, Z., and Karniadakis, G. E. A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data. *Computer Methods in Applied Mechanics and Engineering*, 393:114778, 2022.

Moukalled, F., Mangani, L., and Darwish, M. The finite volume method. In *The finite volume method in computational fluid dynamics*, pp. 103–135. Springer, 2016.

Pang, G., D'Elia, M., Parks, M., and Karniadakis, G. E. npinns: nonlocal physics-informed neural networks for a parametrized nonlocal universal laplacian operator. algorithms and applications. *Journal of Computational Physics*, 422:109760, 2020.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 652–660, 2017.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Reddy, J. N. *Introduction to the finite element method*. McGraw-Hill Education, 2019.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33:7462–7473, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Xue, T., Beatson, A., Adriaenssens, S., and Adams, R. Amortized finite element analysis for fast pde-constrained optimization. In *International Conference on Machine Learning*, pp. 10638–10647. PMLR, 2020.

Limitations and Broader Impacts

In this work, we mainly discuss the challenges of non-uniform data for neural operators. And we develop a general framework named non-uniform neural operator to tackle these challenges. Since this work is basic research, we think there are no relevant ethical issues.

A. Appendix

A.1. Complexity Analysis of K-D Tree Algorithm

In this subsection, we will analyze the time and space complexity of Algorithm 1. For ease of reference, we have replicated Algorithm 1 into Algorithm 2.

Assumptions. Let m be the number of points in the initial point cloud, i.e., $m := |D^{(0)}|$. Before the analysis, we make the following assumptions:

- To ensure that the problem is well defined, we assume that the number of sub-point clouds cannot exceed that of points in the initial point cloud, i.e., $n \leq m$.
- To ensure that the statistics in Eq. (4) are reasonable, we assume that $N_1 \times \cdots \times N_d \leq |D|$. Since $|D| \leq |D^{(0)}|$ (the number of points in the sub-point cloud will never exceed the initial point cloud), we have that $N_1 \times \cdots \times N_d \leq m$.
- In Line 8, we will not necessarily investigate all possible candidates of b (i.e., to enumerate all locations of the points in D) but consider $N_{\max}$ discrete equidistant candidates. We assume $N_{\max}$ to be a typical small constant, e.g., $N_{\max} = 5$, which is consistent with our implementation in experiments.

Time Complexity. Firstly, we discuss the cost of evaluating Eq. (4). During the execution of the algorithm, sub-point clouds that ever existed (including intermediates that were later decomposed) form a full binary tree with $2n - 1$ nodes. For each node, Eq. (4) is only evaluated once, whose time complexity is $\mathcal{O}(|D| + d)$, noticing that $N_1 \times \cdots \times N_d \leq |D|$. Since $|D| \leq m$, the total cost of Eq. (4) comes as $\mathcal{O}(nm + nd)$. Secondly, we analyze the cost of the loops. The number of sub-point clouds will increase by one each time the loop runs, so the number of loops is exactly $n - 1$. In each loop, Line 5, 6, 10 cost no more than $\mathcal{O}(n)$ (the cost of obtaining $\mathrm{KL}(P \parallel Q; D)$ by Eq. (4) has been calculated in the previous step); Line 7 cost $\mathcal{O}(d)$; Line 9 is nothing but a duplication, which is $\mathcal{O}(|D|)$. As for Line 8, we need to sort the point cloud along the k-th axis before evaluating Eq. (5) for $N_{\max}$ candidates, which cost no more than $\mathcal{O}(|D| \log |D| + d)$. To sum up, since $|D| \leq m$ and $n \leq m$, the time complexity of loops is $\mathcal{O}(nm \log m + nd)$ which is also the total time complexity.

Space Complexity In the lifetime of the algorithm, there is only a reorganization of point clouds, and no new point clouds are generated. So $|\bigcup_{D \in \mathcal{S}} D| = |D^{(0)}|$ holds all the time and the space complexity is simply $\mathcal{O}(md)$, where d comes from the length of the spatial coordinates of each point.

Algorithm 2 Domain Decomposition via a K-D Tree (replica)

```
1:  Input: initial point cloud $D^{(0)}$ and the number of sub-point clouds $n$
2:  Output: a sef of sub-point clouds $\mathcal{S}$
3:  Initialize: $\mathcal{S} \leftarrow \{D^{(0)}\}$
4:  repeat
5:      Choose $D^* = \arg\max_{D \in \mathcal{S}} (|D| \cdot \mathrm{KL}(P \parallel Q; D))$
6:      $\mathcal{S} \leftarrow \mathcal{S} - \{D^*\}$
7:      Select the dimension $k, 1 \leq k \leq d$, where the bounding box of $D^*$ has the largest scale
8:      Determine the hyperplane $x_k = b^*$, where $b^* = \arg\max_{b \in \mathcal{B}} \mathrm{Gain}(D^*, b)$ and $\mathcal{B}$ is a set of discrete candidates for $b$
9:      Partition $D^*$ with $x_k = b^*$
10:     $\mathcal{S} \leftarrow \mathcal{S} \cup \{D^*_{x_k > b^*}, D^*_{x_k \leq b^*}\}$
11: until $|\mathcal{S}| = n$
```

A.2. Experimental Details

In this part, We provide supplementary descriptions of the experimental environment, governing equations, hyperparameters, and implementation details.

Experimental environment. Our codes are written in Python and the deep learning part is based on PyTorch library. We refer to `README` in our code repository for the details of dependencies and instructions. All the codes are run on Ubuntu 18.04 LTS Intel(R) Xeon(R) with 32 Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz CPUs. All the models are trained on 1 NVIDIA TITAN Xp 12GB GPU. Besides, we use COMSOL Multiphysics as the software of numerical solver for data generation.

A.2.1. 2D Elasticity

Data Genearation. The dataset directly comes from Li et al. (2022). The settings and parameters of the PDEs are identical to the original experiment in Li et al. (2022).

Implementation of Our Method. The input of the operator is the void shape (see Figure 3 for an example) which can be naturally represented by the point cloud. So, we use the `KernelDensity` (of the bandwidth 0.05) from the scikit-learn to approximate the distribution of the point cloud. Then, we evaluate the likelihood at the 32×32 uniform grid, which serves as the network input. The network outputs the prediction for $\boldsymbol{\sigma}$ values at the subdomain grids, which are interpolated back to the point cloud for evaluation.

Hyperparameters. The hyperparameters of the baselines remain the same as in the original paper, except for three modifications:

- To accelerate the training process, the batch size for the DeepONet baseline is increased from 16 to 16384.
- In the original implementation, for the mesh-based neural operators, the error is measured by the deviation between the output of model and the mesh interpolated from the ground-truth labels at the point cloud. We think this may be inappropriate since our target is to predict the point cloud rather than the interpolated mesh. Therefore, we have corrected the calculation of the error as the deviation between the interpolation result of the model output and the ground-truth labels at the point cloud.
- We changed the original single experiment to parallel five experiments.

For our method, the optimizer is `Adam` with learning rate 0.001 and weight decay 10^{-4}. The training period consists of 501 training epochs with batch size 20. The learning rate scheduler is `StepLR` with step size 400 and gamma 0.1. In addition, NU-FNO uses 4 Fourier layers with modes 12 and width 32 for normal FNO processing. When it is finished, we use 2 linear layers and 2 Fourier layers (with modes 12 and width 32) to generate the prediction in subdomains and interpolate the result into the point cloud via NUDFT. The dropout and normalization are used to promote performance.

A.2.2. (2+1)D Channel Flow

Data Generation. The governing equations are given in Eq. (8), where $\nu = 0.02$. The geometry is shown in Figure 4a. The initial condition is $(u(x,0), v(x,0), p(x,0)) = (0,0,0), x \in \Omega$. We pose $\boldsymbol{u}(x,t) = (u_0(x), 0), t \in (0, 0.3)$ at the inlet, $p(x,t) = 1, t \in (0, 0.3)$ at the outlet, and no-slip boundary condition at other boundaries. We generate u_0 samples as $u_0 \sim \mu$, where $\mu = \mathcal{N}(0, 7^2(-\Delta + 49I)^{-2.5})$. Then, the numerical solver is used to solve the PDEs with corresponding u_0 parameters to generate final training and testing data. The link of the dataset is put into the `README` in our code repository.

Implementation of Our Method. The input of the operator is the (u, v, p) values at the point cloud from $t = 0$ to $t = 0.14$. We first interpolate them into uniform grids of 8 subdomains, which are long aligned to reduce padding. Then, the grids are resized to the same size (the largest one of their original sizes) via the FFT-IFFT and then passed to the network. The network outputs the prediction for (u, v, p) values at the subdomain grids from $t = 0.15$ to $t = 0.29$, which are interpolated back to the point cloud for evaluation.

Hyperparameters. The optimizer is `Adam` with learning rate 0.001 and weight decay 10^{-4}. The learning rate scheduler is `ReduceLROnPlateau` with patience $\lfloor \text{epochs}/20 \rfloor$ and factor 0.1. The hyperparameters of each baseline is listed below:

- **FNO (global interpolation).** It has 4 3D Fourier layers with modes 8 and width 20. It is trained for 501 epochs with batch size 20.
- **U-Net (global interpolation).** It has 4 3D convolution layers and 4 3D deconvolution layers. It is trained for 501 epochs with batch size 20.
- **MWNO (global interpolation).** It has 4 multiwavelet layers with $\alpha = 8, c = 3, k = 3, L = 0$ and Legendre basis. It is trained for 501 epochs with batch size 20.
- **NU-FNO, NU-U-Net, NU-MWNO.** The parameters of our method are the same as corresponding baselines, except for the input and output channels which are increased to allow for taking and producing subdomain uniform grids.
- **Geo-FNO (learned).** It has 2 2D Fourier layers with modes 8 and width 20 for interpolation of input and output via NUDFT and 3 3D Fourier layers with Fourier layers with modes 8 and width 20. It is trained for 101 epochs with batch size 10.
- **GraphNO.** It has 4 graph convolution layers with node width 32 and kernel width 128. The input graph is built from the point cloud with a Gaussian kernel with radius 0.05 and the resulting edge features have 4 channels. It is trained for 101 epochs with batch size 1.
- **DeepONet.** The input coordinate of DeepONet is organized as (x_i, y_i) while (u, v, p) values at different time steps are considered as multiple channels, rather than (x_i, y_i, t_i). Besides, it has 5 branch layers and 5 trunk layers, both with 64 neurons. When branch and trunk complete their calculations, their results are grouped, where the dot product is performed in each group separately to produce the final multi-channel results. It is trained for 101 epochs with batch size 2048. And the learning rate is 0.0001 which is different from other baselines.

A.2.3. 3D Heatsink

Data Genearation. We refer to the COMSOL application gallery (COMSOL AB, 2022) for the details of the governing equations [2]. The body force term in NS equations is denoted as $\boldsymbol{f} = (f, 0, 0)(x), x \in \Omega$, where $f(x) = f_0(x, y)$ only changes in the XY direction (here we abuse the notations for coordinates: the former x is a coordinate in the three-dimensional space while the latter x is the value on the X axis). We generate f_0 samples as $f_0 \sim \mu$, where $\mu = \mathcal{N}(0, 7^{1.5}(-\Delta + 49I)^{-2.5})$. Then, the numerical solver is used to solve the PDEs with corresponding f_0 parameters to generate final training and testing data. The link of the dataset is put into the `README` in our code repository.

Implementation of Our Method. The implementation details are similar to the details in Appendix A.2.1.

Hyperparameters. The optimizer is `Adam` with learning rate 0.001 and weight decay 10^{-4}. The learning rate scheduler is `ReduceLROnPlateau` with patience $\lfloor \text{epochs}/20 \rfloor$ and factor 0.1. All the models are trained for 501 epochs with batch size 20. The hyperparameters of each baseline is listed below:

- **FNO (global interpolation).** It has 4 3D Fourier layers with modes 8 and width 20.
- **U-Net (global interpolation).** It has 4 3D convolution layers and 4 3D deconvolution layers.
- **MWNO (global interpolation).** It has 4 multiwavelet layers with $\alpha = 8, c = 3, k = 3, L = 0$ and Legendre basis.
- **NU-FNO, NU-U-Net, NU-MWNO.** The parameters of our method are the same as corresponding baselines, except for the input and output channels which are increased to allow for taking and producing subdomain uniform grids.
- **PointNet.** It has 4 local transform layers and a PointNet feature layer.

In addition, for mesh-based baselines, the loss function consists of two terms: the first term is the deviation between the network output and the ground-truth subdomain grids while the last term is a regularization term, which is the interpolation error of between output and the ground-truth point cloud. The weight of the regularization term is 5×10^{-3}.

[2] `https://www.comsol.com/model/heat-sink-8574`

Remark. When calculating the shape of the uniform grid, we let the total size (i.e., the number of cells in the grid) be equal to the number of points in the point cloud, 19517, and let the aspect ratio be equal to that of the geometry. However, since the size of grid must be an integer, there are some rounding errors being ignored.