PDENNEval: A Comprehensive Evaluation of Neural Network Methods for Solving PDEs*

Ping Wei , Menghan Liu , Jianhuan Cen , Ziyang Zhou , Liao Chen and **Qingsong Zou**†

School of Computer Science and Engineering, Sun Yat-sen University, Guangzhou, China
Guangdong Province Key Laboratory of Computational Science, Guangzhou, China
{weip7, liumh59, cenjh3, zhouzy36, chenliao}@mail2.sysu.edu.cn, mcszqs@mail.sysu.edu.cn

Abstract

The rapid development of neural network (NN) methods for solving partial differential equations (PDEs) has created an urgent need for evaluation and comparison of these methods. In this study, we propose PDENNEval, a comprehensive and systematic evaluation of 12 NN methods for PDEs. These methods are classified into function learning type and operator learning type based on their different mathematical foundations. The evaluation is implemented using a diverse dataset comprising 19 distinct PDE problems selected from various scientific fields such as fluid, materials, finance, and electromagnetic. Several evaluation results are reported, aiming to provide guidance for further research in this field. Our code and data are publicly available at https://github.com/Sysuzqs/PDE NNEval.

1 Introduction

Owing to their exceptional approximation capabilities [Hornik, 1991; Barron, 1994] and multi-level feature extraction abilities [LeCun *et al.*, 2015], neural networks (NN) based methods have been widely applied and have achieved great success in various disciplines, including computer vision [Krizhevsky *et al.*, 2012; He *et al.*, 2016; Dosovitskiy *et al.*, 2020] and natural language processing ([Vaswani *et al.*, 2017; Devlin *et al.*, 2018; Brown *et al.*, 2020]). Recently, NN methods have excelled in solving partial differential equations (PDEs), yielding significant breakthroughs in this field, e.g. [E *et al.*, 2017; E and Yu, 2018; Raissi *et al.*, 2019; Li *et al.*, 2020; Lu *et al.*, 2021a]. Compared to traditional numerical PDEs methods like finite element methods, NN methods exhibit unique advantages. One notable advantage is that they are mesh-free, making them immune to the *curse of dimensionality*. In addition, NN methods can learn patterns and features from data which might be governed by multiple types of equations[Raissi and Karniadakis, 2018]. This capability enables a single solver to handle various types of equations, offering a more versatile approach for solving PDEs.

The widespread development of NN methods for solving PDEs generates a pressing need for the assessment and comparative analysis of these techniques. In 2021, Lu and his colleagues compare the Physics-Informed Neural Networks (PINN) to a standard finite element method across various problems, including Poisson equation, Burgers' equation, Volterra integral differential equation, Lorenz system and Diffusion-Reaction systems in [Lu *et al.*, 2021b]. In 2022, three benchmarks of NN methods for PDEs are published. First, Lu and his colleagues[Lu *et al.*, 2022] present the relative performance of Deep Operator Network (DeepONet) and Fourier Neural Operator (FNO) by using 16 different benchmarks covering Burgers' equation, Darcy problem, Advection equation, Compressible Euler equations, electroconvection problem and Navier-Stokes equation, etc. Secondly, a side-by-side analysis, called PDEArena, is presented in [Gupta and Brandstetter, 2022] to show the performance of NN methods FNO, ResNet, and U-Net on Shallow-Water equations and Navier-Stokes equations. Thirdly, a versatile benchmark suite[Takamoto *et al.*, 2022b], called PDEBench, is proposed to compare NN methods FNO, U-Net, PINN, using datasets derived from 11 hydromechanical field PDEs, including Advection, Navier-Stokes equations, etc. In 2023, a benchmark[Hao *et al.*, 2023], called PINNacle, is designed to assess the performance of 10 PINN variants, encompassed tasks derived from over 20 different PDEs spanning various domains, such as heat conduction, fluid dynamics, biology, and electromagnetic.

Previous research primarily benchmarks some popular methods such as DeepONet, FNO, U-Net, and PINN. However, a comprehensive evaluation of some other important NN methods is lacking. Additionally, different types of NN methods are often compared in the same setting without being classified and evaluated based on their distinct mathematical foundations. On the other hand, despite the selection of various types of PDE problems in previous work, some crucial equations, such as phase-field equations, have not been included in the evaluation tasks. Furthermore, challenging problems like PDEs with singular solutions have also been omitted. In summary, the development of the evaluation and comparison works for NN methods have not kept pace with the advancement of NN methods themselves.

*An extendede version of this paper(with the Appendix included) can be find in https://github.com/Sysuzqs/PDENNEval.

†Corresponding author.

In this work, we propose a comprehensive and systematic evaluation of various NN methods for solving diversified PDE problems, considering accuracy, efficiency and robustness.

Our evaluations are integrated into a benchmarks suite called PDENNEval, with some key features listed below.

- We provide a code reference that integrates 12 advanced NN methods, including six function learning-based NN methods: the Deep Ritz Method(DRM) [E and Yu, 2018], PINN [Raissi *et al.*, 2019], Weak Adversarial Network (WAN) [Zang *et al.*, 2020], Derivative-Free Loss Method (DFLM)[Han *et al.*, 2020], Random Feature Method (RFM) [Chen *et al.*, 2022], Deep Finite Volume Method (DFVM) [Cen and Zou, 2023], and six operator learning-based NN methods: U-Net[Ronneberger *et al.*, 2015], Message Passing Neural PDE Solvers (MPNN)[Brandstetter *et al.*, 2022b], FNO [Li *et al.*, 2020], DeepONet[Lu *et al.*, 2021a], Physics-Informed Neural Operator (PINO)[Li *et al.*, 2021], U-shaped Neural Operators (U-NO)[Rahman *et al.*, 2022]. We systematically evaluate the performance of these two categories in different setting.

- We carefully select 19 representative PDE problems from various fields such as fluid dynamics, materials science, finance, and electromagnetism to evaluate the performance of the aforementioned NN methods. These equations encompass some core mathematical challenges, including nonlinearity, singularity, high dimensionality and complex geometry. It's may worth mentioning that it is the first time NN methods have been evaluated by using a PDE with a singular solution. More detailed descriptions for all PDEs are provided in Appendix A.

- We design a diverse and high-fidelity dataset, generated using high-precision conventional scientific-computation methods. For instance, to generate the data for the phase-field equations, spatial discretization is accomplished using the spectral method, and time discretization is performed using the fourth-order Runge-Kutta method.

- We select a set of metrics that are better suited for each type of NN methods, including L_2 norm relative error and maximum error, to assess accuracy, Lipschitz constant to evaluate robustness, and efficiency metrics encompassing training time, inference time, and convergence time.

The rest of the paper is organized as below. In Section 2, we outline the 12 representative NN methods and present 19 distinct PDE tasks which will be evaluated in PDENNEval. In Section 3, we present an overview of the datasets and metrics used in PDENNEval. In Section 4, we illustrate a standardized testing procedure, and present the results of the pertinent tests. Some brief concluding remarks are given in Section 5.

2 An Overview of Selected PDEs and NN Methods

In this section, we briefly introduce the PDE problems and NN methods integrated into the PDENNEval.

2.1 Selected PDE Tasks

A well-posed PDE system often involves three fundamental components: the equation itself, boundary conditions, and initial values. Let $\Omega \subset \mathbb{R}^n$ be the spatial domain with boundary $\partial\Omega$, and let $[0, T]$ be the time interval. A general PDE system can be represented as:

$$\mathcal{F}(u; \mathbf{x}, t; \lambda) = \mathbf{f}(\mathbf{x}, t), \qquad (\mathbf{x}, t) \in \Omega \times [0, T], \tag{1}$$

where $\mathcal{F}$ represents a differential operator governing the behavior of the unknown function u, λ is an optional vector of parameters, appearing often as constant or variable coefficients of the differential operator, and $\mathbf{f}(\mathbf{x}, t)$ is a vector-valued function. This equation encapsulates the evolution of the system over the spatiotemporal domain. Boundary conditions are often expressed as:

$$\mathcal{B}(u; \mathbf{x}, t) = \mathbf{b}(\mathbf{x}, t), \qquad \mathbf{x} \in \partial\Omega, \tag{2}$$

where $\mathbf{b}$ is a given vector-valued function defined on $\partial\Omega$. The boundary conditions $\mathcal{B}$ define how the solution behaves or is constrained at the boundary of the domain. At $t = 0$, the system's state is often specified by initial values:

$$\mathcal{I}(u; (\mathbf{x}, t)|_{t=0}) = \mathbf{i}(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{3}$$

where $\mathbf{i}$ is a given vector-valued function defined only on Ω.

Usually, we call it as a *forward* problem when seeking a function u to satisfy (1), (2) and (3), given the information $\Omega, \lambda, \mathbf{f}, \mathbf{b}, \mathbf{i}$. Conversely, we call it as an *inverse* problem if we aim to seek parameters λ, or the shape of Ω, or initial conditions, or other essential information about the system based on some observed data or outcomes. In this bench, since an inverse problem might be solved by training a NN using the same loss function as its corresponding forward problem, we will focus only on solving forward problems and encourage subsequent researchers to benchmark their respective inverse problems.

In our study, we aim to benchmark 19 forward problems of PDEs, representing pivotal mathematical models in physics and engineering that span various fields, including fluid dynamics, materials science, finance, and electromagnetism. Within the realm of fluid dynamics, our benchmark incorporates 11 equations: 1D Advection, 1D and 2D Burgers', 1D Diffusion-Reaction, 1D Diffusion-Sorption, 2D Darcy Flow, 2D Shallow-Water, 1D, 2D, and 3D Navier-Stokes, and 3D Euler equations. These equations model gas and liquid flow phenomena, finding applications across aerospace, weather prediction, oceanography, chemistry, and hydraulic engineering fields. Notably, while nine fluid-related PDEs have been evaluated in PDEBench, they were restricted to only three methods, whereas our assessment extends to twelve methods in total. In materials science, our selection includes the 1D and 2D Allen-Cahn equations and the 1D Cahn-Hilliard equation, employed in understanding material phase transitions,

crystal growth, and surface evolution phenomena. Regarding finance, we encompass the 2D Black-Scholes-Barenblatt equation, crucial in the pricing of European options. In the domain of electromagnetism, our benchmark involves the Poisson equation and 3D Maxwell's equations, which model the behaviors of electromagnetic field and serve as fundamental elements in the realms of communication technology, electromagnetic device design, and optical systems. Specifically for the Poisson equation, we construct three distinct cases.

We select these PDEs to ensure that the benchmark introduces a wide range of mathematical challenges, including

- **Nonlinear behavior**: Numerous PDEs showcase nonlinear or even chaotic dynamics, wherein minor alterations in initial conditions result in significant divergence within the outcomes. Addressing these nonlinearities illuminates a method's capacity to capture intricate and sensitive dependencies within the system's behavior. In particular, our set of equations includes primarily nonlinear equations, such as the Navier-Stokes equations, phase field equations, and the Black-Scholes-Barenblatt equation.
- **High dimensionality**: Various fields, including materials, finance, and quantum mechanics, present high-dimensional PDEs. To evaluate the capability of NN methods in handling high-dimensional PDEs, we utilize high-dimensional ($d > 3$) Poisson equations to compare the performance of function learning-based NN methods.
- **Singularity**: Singularities in partial differential equations represent unconventional solution properties, often marked by unboundedness, discontinuities, or lack of differentiability at specific points. In our PDENNEval, we intentionally create a Poisson equation with a singular solution for assessment purposes.
- **Complex geometry**: PDEs of practical significance often extend beyond regular domains, underscoring the importance of evaluating NN methods' capability to handle irregular domain scenarios. In our study, focusing on the Poisson equation, we established a specific case involving the resolution of the equation over an L-shaped domain.

2.2 Selected NN Methods

An NN method heavily relies on its definition of the loss function $\mathcal{L}(\theta)$ which depends on θ, the trainable parameters of the NN. Generally, a loss $\mathcal{L}(\theta)$ for PDEs consolidates four essential components into a unified expression as follows:

$$\begin{aligned}\mathcal{L}(\theta) = \omega_{\text{PDE}}\, \mathcal{L}_{\text{PDE}}(\theta) + \omega_{\text{BC}}\, \mathcal{L}_{\text{BC}}(\theta) \\ + \omega_{\text{IC}}\, \mathcal{L}_{\text{IC}}(\theta) + \omega_{\text{Data}}\, \mathcal{L}_{\text{Data}}(\theta),\end{aligned} \tag{4}$$

where $\omega_{\text{PDE}}, \omega_{\text{BC}}, \omega_{\text{IC}}, \omega_{\text{Data}}$ are the weights of different loss terms, and a breakdown of each loss term is explained as below:

- **PDE constraint loss ($\mathcal{L}_{\text{PDE}}$)** enforces the PDE constraints within the domain Ω, defined as

$$\mathcal{L}_{\text{PDE}}(\theta) = \frac{1}{N_1} \sum_{(\mathbf{x},t)\in\mathcal{S}_{\text{PDE}}} \|\widehat{\mathcal{F}}(u_\theta, \mathbf{f}; \mathbf{x}, t; \lambda)\|^2, \tag{5}$$

  where $\widehat{\mathcal{F}}$ is a variant of the original equations (1), including its strong, weak, stochastic forms; $\mathcal{S}_{\text{PDE}}$ is the set of training points in the interior of $\Omega \times [0, T]$, and $N_1 = \#\mathcal{S}_{\text{PDE}}$.
- **Boundary condition loss ($\mathcal{L}_{\text{BC}}$)** ensures adherence to specified boundary conditions at $\partial\Omega$, defined as

$$\mathcal{L}_{\text{BC}}(\theta) = \frac{1}{N_2} \sum_{(\mathbf{x},t)\in\mathcal{S}_{\text{BC}}} \|\left(\mathcal{B}(u_\theta) - \mathbf{b}\right)(\mathbf{x}, t)\|^2, \tag{6}$$

  where $\mathcal{S}_{\text{BC}}$ is the set of training points sampled from the boundary $\partial\Omega \times [0, T]$, and $N_2 = \#\mathcal{S}_{\text{BC}}$.
- **Initial condition loss ($\mathcal{L}_{\text{IC}}$)** enforces accuracy at the initial time $t = 0$, defined as

$$\mathcal{L}_{\text{IC}}(\theta) = \frac{1}{N_3} \sum_{\mathbf{x}\in\mathcal{S}_{\text{IC}}} \|\mathcal{I}(u_\theta(\mathbf{x}, 0)) - \mathbf{i}(\mathbf{x})\|^2, \tag{7}$$

  where $\mathcal{S}_{IC}$ is the set of training points sampled from the Ω, and $N_3 = \#\mathcal{S}_{\text{IC}}$.
- **Data loss ($\mathcal{L}_{\text{Data}}$)** measures the discrepancy between predicted and observed data at designated points, defined as

$$\mathcal{L}_{\text{Data}}(\theta) = \frac{1}{N_4} \sum_{(\mathbf{x},t)\in\mathcal{S}_{\text{Data}}} \|u_\theta(\mathbf{x}, t) - u(\mathbf{x}, t)\|^2, \tag{8}$$

  where $\mathcal{S}_{\text{Data}}$ is the set of training points at which the value of the true solution u is known and $N_4 = \#\mathcal{S}_{\text{Data}}$.

In general, physics-driven methods typically encompass only the first three components of the loss function, i.e., $\omega_{\text{Data}} = 0$, while data-driven methods exclude the PDE constraint loss. However, in practice, certain physics-driven approaches encompass all loss components, blurring the distinction between data-driven and physics-driven paradigms. Hence, categorizing methods strictly into physics-driven and data-driven techniques becomes inadequate, given the strong coupling between these methodologies. The demarcation between these two approaches tends to be ambiguous.

In PDENNEval, we attempt to categorize NN methods for solving PDEs into two principal categories distinguished by their mathematical foundations. The first category employs elaborated NN to fit the solution function of the equation, rooted in the universal approximation theory[Hornik, 1991; Barron, 1994; He *et al.*, 2022]. The second category utilizes NN to fit solution operators for a spectrum of PDEs, based on the universal approximation of linear or nonlinear operators by NN[Lu *et al.*, 2021a]. Some details of these two methods are described as below.

Function Learning Based NN Methods

NN methodologies for function learning largely fall under the domain of physics-driven methods, resembling traditional approaches in their goal of approximating solutions to PDEs. While traditional numerical methods often utilize piece-wise polynomials, NN methods leverage networks for this purpose. Typically, these methods involve training the networks using the four aforementioned loss components. However, similar to a traditional numerical method which solves a PDE

only using the equation itself, the boundary and initial conditions, a function learning type NN method often does not need to use exceptional data to train the NN. That is, ω_{Data} is often set to 0. Certainly, data might contribute to the boundary and initial value losses, but they are typically treated separately from the data loss component.

Function learning based methods differ mainly in their treatment of the PDE constraint loss item. Strong form-based methods like PINN and RFM integrate the PDE constraint loss utilizing the *residual* $\widehat{\mathcal{F}} = \mathcal{F} - f$. PINN and RFM both employ strong form-based loss functions, yet their optimization processes differ. PINN utilizes deep learning optimizers, while RFM, employing linear algebraic equations, fixes the parameters of the preceding layers to compute the final layer's parameters. Weak form-based methods encompass DRM, WAN, DFLM, and DFVM. DRM is rooted in the variational format, emphasizing the principle of minimal potential energy. WAN adopts the virtual-work-principle form, focusing on finding weak solutions to the equations. DFLM tailors its loss to match the equivalent stochastic-differential form of the original PDE. DFVM aligns its loss with the local-conservation form of the PDE.

Operator Learning Based NN Methods

Operator learning-based methods tackle solving a set of PDEs by learning some operators, categorized into autoregressive and non-autoregressive approaches. Autoregressive methods focus on learning mappings from solutions at preceding time steps to subsequent time steps, including FNO, MPNN, U-Net, and U-NO. Non-autoregressive methods focus on learning mappings from auxiliary functions to solution functions, including DeepONet and PINO.

Autoregressive methods learn mappings from preceding time-step solutions to subsequent time-step solutions. These methods fundamentally simulate an image-to-image mapping, forecasting the future state based on the current state, which can be described as below :

$$\mathcal{G} : u(\mathbf{x}, t)|_{t=t_0} \mapsto u(\mathbf{x}, t)|_{t=t_1}. \tag{9}$$

These methods involve formulating the loss function to minimize the difference between the predicted solution at the next time step and the actual solution. This enables the network to learn the temporal evolution of the system. For instance, FNO focuses on parameterizing the integral kernel directly in Fourier space, enabling an expressive and efficient architecture for learning solution operators. MPNN operates on graph data by iteratively generating messages, aggregating neighbor information, and updating node representations. U-Net adapts an autoregressive approach for solving time-dependent PDEs. Its architecture predicts subsequent time step solutions based on preceding sequences of solutions, making it adept at capturing both local and global patterns for progressive solutions. U-NO employs a U-shaped architecture, mitigating memory constraints while efficiently mapping functions across different domains. By handling encoding and decoding with skip connections, it offers an effective solution for complex neural operator modeling.

Non-autoregressive methods involve learning an operator which maps the known information $(\lambda, \mathbf{f}, \mathbf{b}, \mathbf{i})$ to solution functions $u \in \mathcal{U}$, described as

$$\mathcal{G} : (\lambda, \mathbf{f}, \mathbf{b}, \mathbf{i}) \mapsto u. \tag{10}$$

In other words, these methods aim to learn mappings from right-hand side functions, boundary conditions, or other auxiliary information to the solution function. The loss function formulation for such mappings varies based on specific model designs and target functions, and may or may not involve the explicit PDE constraint loss term. For instance, DeepONet utilizes separate feed-forward neural networks (FNNs) to encode input functions and query locations, effectively mapping auxiliary functions directly to solutions. PINO blends data and physics information to learn resolution-invariant operators in Fourier space. It efficiently handles temporal PDEs by combining data-driven and physics-driven methodologies for accurate solutions across various resolutions and dimensions.

PDE	Dataset	PDE	Dataset
Advection	Ad1	Diffusion-Reaction	DR1
Burgers 1D	Bu1	Diffusion-Sorption	DS1
Allen-Cahn 1D	AC1	Cahn-Hilliard	CH1
Comp Navier-Stokes 1D	NS1	Comp Navier-Stokes 2D	NS2
Allen-Cahn 2D	AC2	Burgers 2D	Bu2
Darcy Flow	DF2	Shallow-Water	SW2
Black-Scholes-Barenblatt	BS2	Comp Navier-Stokes 3D	NS3
Maxwell's	Ma3	Euler	Eu3

Table 1: The name of datasets corresponding to PDEs

3 An Overview of Datasets and Metrics

In this section, we present two fundamental components of PDENNEval: the datasets and the metrics.

3.1 Datasets

The PDENNEval datasets consist of 16 data files, each representing a distinct PDE, excluding three instances of the Poisson equation which are solved using only the first two loss components in (4) (i.e. $\omega_{\text{IC}} = \omega_{\text{Data}} = 0$). These 16 data files are named using the following abbreviations of PDEs.

In each data file, we store multiple samples, each of which is a solution of the related PDE. All solution samples are obtained by using traditional numerical methods to solve the PDE with different initial functions, sampled in a certain function space according to a certain distribution. Each sample contains the spatiotemporal coordinates of the underlying mesh and the values of the solution at those locations. Listed in Table 2 is a summary of all 16 data files. In this table, N_d is the number of spatial dimensions of the equation, and N_S is the size of the data file. Moreover, we use N_{sp}, N_t, N_{sa} to denote the spatial resolution, the temporal resolution of the underlying mesh, and the number of samples, respectively. Note that, since the Darcy flow equation is time-independent, we have not provided the number N_t for the dataset DF2.

Some details about the traditional numerical methods for generating datasets are reported below. First, the following four datasets: AC1, AC2, CH1, and Bu2 are generated using the Matlab R2022a Software. More precisely, the samples in AC1, AC2 and CH1 are obtained by discretizing the spatial

Dataset	N_d	N_S	N_{sp}	N_t	N_{sa}
Ad1	1	7.7 G	1024	201	10000
DR1	1	3.9 G	1024	101	10000
Bu1	1	7.7 G	1024	201	10000
DS1	1	4.0 G	1024	101	10000
AC1	1	3.9 G	1024	101	10000
CH1	1	3.9 G	1024	101	10000
NS1	1	12 G	1024	101	10000
Bu2	2	13 G	128×128	101	1000
NS2	2	52 G	128×128	21	10000
DF2	2	1.3 G	128×128	-	10000
SW2	2	6.2 G	128×128	101	1000
AC2	2	6.2 G	128×128	101	1000
BS2	2	6.2 G	128×128	101	1000
Ma3	3	5.9 G	$32 \times 32 \times 32$	8	1000
Eu3	3	83 G	$128 \times 128 \times 128$	21	100
NS3	3	83 G	$128 \times 128 \times 128$	21	100

Table 2: Summary of datasets

variable with a spectral method and discretizing the temporal variable with a fourth-order Runge-Kutta scheme. The samples in the dataset Bu2 are obtained by discretizing the spatial first-order derivatives with a first-order upwind scheme and discretizing the spatial second-order derivatives with a central difference scheme. Secondly, the datasets Ma3, BS2, Eu3 are generated using Python. To be more specific, the samples in Ma3 are obtained by discretizing the 3D Maxwell equation using the finite-difference time-domain method (FDTD, [Kunz and Luebbers, 1993]), and the resulting discrete algebraic system is solved with the leapfrog method. The BS2 dataset is sampled from some exact solutions of the Black-Scholes-Barenblatt equation given in [Raissi, 2024]. The samples in Eu3 are obtained by solving the Euler equation utilizing the second-order HLLC scheme [Toro *et al.*, 1994] for the inviscid part, the MUSCL schemes [Van Leer, 1979] for its inviscid counterpart, and a central difference scheme for the viscous part. The remaining nine datasets are related to PDEs in fluid dynamics, their samples are generated using the numerical methods which been used to generate the dataset in the PDEBench [Takamoto *et al.*, 2022a].

All datasets are stored in HDF5 format, which is convenient to use and share and widely supported in a host of programs including Python, R and Matlab. The data and usage example can be found in https://github.com/Sysuzqs/PDENNEval. More details of datasets can be found in Appendix C.

3.2 Metrics

In this section, we briefly introduce the metrics used in PDENNEval. To evaluate the accuracy, efficiency, and robustness of our selected NN methods, we collect 7 metrics which can be categorized into two groups: performance metrics and functional metrics. They are summarized in Table 3. Furthermore, we provide the definition of one metric from each of the aforementioned groups. The L_2 norm relative error is defined as L2RE $= \|u_\theta - u\|_2 / \|u\|_2$. The Lipschitz constant **Lips** is defined as the smallest L satisfying

$$\|u_\theta(\mathbf{x}) - u_\theta(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|_2, \quad \forall \mathbf{x}, \mathbf{y}.$$

Group	Metrics	Symbolic meanings
Performance	L2RE	L_2 norm relative error
	mERR	L_∞ norm error
	t_{train}	Model training time
	t_{infer}	Model inference time for one sample
	t_{conv}	The time of model convergence
Functional	Cove	Model's coverage ability on multiple tasks
	Lips	The Lipschitz constant

Table 3: Metrics of PDENNEval

Here $\|\cdot\|_2$ is the L_2 norm, $\mathbf{u}_\theta$ is the predictive solution, and $\mathbf{u}$ is the reference solution. The definition of other metrics and their detailed description are given in Appendix D.

4 A Selection of Evaluation Results

In this section, we separately present partial experimental results for function learning-based NN methods and operator learning-based NN methods. Additional results can be found in Appendix F. For a fair comparison, all our experiments are conducted using the following system configuration: CPU : 2× Intel Xeon Gold 6230R @ 2.10GHz; GPU : 1 × NVIDIA TITAN RTX; Software : PyTorch@1.13.1, CUDA@11.6.

Recall that we have introduced 12 NN methods and 19 PDE tasks in Section 2. In our experiments, to test the performance of function learning NN methods, we introduce three additional PDE tasks P_S, P_L, P_H which will be explained in details in Section 4.1. Apparently, in PDENNEval, not every method has been applied to all 19 PDE tasks. Listed in Table 4 is a summary of tasks coverage by our selected 12 NN methods.

From this table, we observe that PINN, DFVM, and RFM have been used to solve all 19 PDE problems; Meanwhile, the function learning methods WAN, DRM, and DFLM have only been applied to a part of these PDE problems. Following the vanilla methods, we use DRM [E and Yu, 2018] and DFLM [Han *et al.*, 2020] only to solve symmetric and time-independent problems P_S, P_L, P_H and DF2. In the same way, we use WAN [Zang *et al.*, 2020] only to solve scalar problems. In contrast, all operator learning type methods have not been used to solve the three Poisson tasks. Moreover, DeepONet, PINO and MPNN do not solve 3D problems, while the MPNN does not solve the time-independent problem DF2 either. It is worth mentioning that we do not mean that a specific NN method can not be applied to solve some specific PDE problems if it has not been done in our PDENNEval, it simply suggests that implementing this NN method for individual equation problems may pose challenges.

4.1 Results on Function Learning Methods

As already pointed out in Table 4, function learning-based NN methods can solve almost all kinds of PDE tasks. Here, we primarily present the experimental results of three tasks P_H, P_S and P_L, which tackle three special mathematical challenges high-dimensionality, singularity, and complex geometry. The governing PDE of these three tasks is the Poisson

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
P_S	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×
P_L	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×
P_H	✓	✓	✓	✓	✓	✓	×	×	×	×	×	×
Ad1	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
DR1	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
Bu1	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
DS1	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
AC1	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
CH1	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓
NS1	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓
Bu2	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓
NS2	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓
DF2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
SW2	✓	✓	✓	×	×	×	✓	✓	✓	✓	✓	✓
AC2	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
BS2	✓	✓	✓	✓	×	×	✓	✓	✓	✓	✓	✓
Ma3	✓	✓	✓	×	×	×	×	×	✓	✓	✓	×
Eu3	✓	✓	✓	×	×	×	×	×	✓	✓	✓	×
NS3	✓	✓	✓	×	×	×	×	×	✓	✓	✓	×

Table 4: Tasks coverage of NN methods. Here M1-M12 denote the NN methods PINN, DFVM, RFM, WAN, DRM, DFLM, DeepONet, PINO, U-NO, FNO, U-Net, MPNN, respectively.

equation

$$-\Delta u = f, \text{in } \Omega, \quad u = g, \text{on } \partial\Omega. \tag{11}$$

where Ω, f, g will be specified later.

1) High dimensional case (P_H). In the first case, we evaluate the performance of function learning-based NN methods on high dimensional Poisson equations. Let $\Omega = [-1,1]^d$, here $d = 3, 5, 10, 20, 40, 80, 120$ is the spatial dimension. The functions f and g are chosen so that $u(\mathbf{x}) = \left(\frac{1}{d}\sum_{i=1}^d x_i\right)^2 + \sin\left(\frac{1}{d}\sum_{i=1}^d x_i\right), \forall \mathbf{x} \in \mathbf{R}^d$, which is sufficiently smooth.

	d	PINN	DFVM	RFM	WAN	DRM	DFLM
	3	0.0019	0.0019	0.0074	0.0672	0.0176	0.0311
	5	0.0017	0.0017	0.0479	0.0431	0.0035	0.0034
	10	0.0026	0.0025	0.3079	0.0573	0.0047	0.0038
L2RE	20	0.0035	0.0035	0.5133	0.2503	0.0062	0.0043
	40	0.0037	0.0036	0.5108	0.1316	0.0090	0.0057
	80	0.0067	0.0062	0.5678	0.0159	0.0378	0.0296
	120	0.0129	0.0112	0.6654	0.0230	0.1066	0.1053
	3	395.5	145.3	-	115.7	147.1	302.8
	5	549.7	156.0	-	135.1	150.1	303.1
	10	1,289	221.6	-	171.4	299.6	320.6
t_{train}(s)	20	2,512	290.3	-	291.8	314.2	324.6
	40	4,526	412.7	-	444.4	314.3	325.7
	80	7,472	776.0	-	818.7	328.2	329.7
	120	10,701	1,151	-	1,272	344.3	370.1

Table 5: L2RE and t_{train} of function learning methods for problem P_H.

Listed in Table 5 are L_2 norm relative error and training times of function learning methods for P_H. From Table 5, we observe that all methods are immune to the *curse of dimensionality*. That is, the computation time of no method increases exponentially in terms of the dimension. Moreover, PINN exhibits superior performance in terms of the L2RE metric. The L2RE of the DFVM is almost as good as the PINN. In addition, on evaluation on the training time metric t_{train}, the WAN excels for the $3, 5, 10, 20$ dimensional cases, while DRM excels for the 40, 80 and 120 dimension cases. It is worth mentioning that since the RFM solution is obtained by solving a linear system of algebraic equations instead of training, no training time has been reported for this method.

2) Singular case (P_S). In the second case, we consider the equation 11 with a singular solution. Precisely, let $\Omega = [0,1]^2$ and f, g be selected to ensure $u(x_1, x_2) = x_1^2$ if $0 \le x_1 \le \frac{1}{2}$ and $u(x_1, x_2) = (1 - x_1)^2$ otherwise. Note that the exact solution u is continuous but not smooth at the location $x_1 = \frac{1}{2}$.

	PINN	DFVM	RFM	WAN	DRM	DFLM
L2RE	1.0793	0.0209	1.0298	0.0543	1.0050	0.0207
t_{train}(s)	235.7	134.0	-	175.4	141.2	228.9

Table 6: L2RE and t_{train} of function learning methods for problem P_S.

Listed in Table 6 are L2RE and t_{train} of function learning methods for this Poisson singular problem. We observe that in this case, the error of the DFLM is the smallest, followed by the DFVM. Moreover, the training time of the DFVM is the shortest. The error of the PINN and the RFM seems to be too big, which might be caused by the fact that they use the strong form PDE to construct their loss.

3) L-shape domain case (P_L). In the third case, we choose $\Omega = [-1,1]^2 \backslash (0,1]^2$ and $f = 1, g = 0$. There is no analytical solution for this problem. We utilize the software COMSOL to generate a numerical solution whose configuration is shown in Fig1.

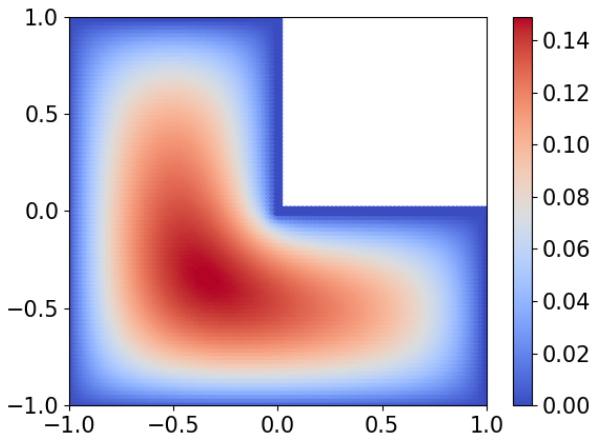

Figure 1: The solution of the problem P_L.

Listed in Table 7 are the L_2 norm relative errors and training times for the problem P_L. We observe that the error of the PINN is the smallest, followed by the DFVM. The training time of the DRM is the shortest.

	PINN	DFVM	RFM	WAN	DRM	DFLM
L2RE	0.0042	0.0043	0.0189	0.0539	0.0216	0.5576
t_{train}(s)	233.9	289.9	-	182.1	94.80	173.5

Table 7: L2RE and t_{train} of function learning methods for problem P_L.

Note that here, only results concerning L2RE and t_{train} are reported, those concerning other metrics such as mERR, t_{infer} and t_{conv} will be presented in the Appendix F.

4.2 Results on Operator Learning Methods

The six operator learning methods to be measured in PDENNEval are DeepONet, PINO, U-NO, FNO, U-Net, and MPNN, which are data-driven. Therefore, they are applied to solve 16 PDE problems except the three Poisson problems in the previous section, as shown in Table 4. Due to the space limitation, here we only present the evaluation results of the following three equations: the 1D Advection equation (Ad1), the 2D Allen-Cahn equation (AC2), and the 2D Darcy Flow equation (DF2). In our experiments, 90% of the data in each dataset is used for training, and 10% is used for testing. Moreover, here we only report the evaluation results in terms of the accuracy metric L2RE and three time metrics: the training time t_{train}, the inference time t_{infer}, and the convergence time t_{conv}.

Listed in Table 8 are the L_2 norm relative error of the six operator learning methods. We observe that the U-NO achieves the best computational accuracy for problems Ad1 and AC2, while PINO achieves the best accuracy for the problem DF2. Note that we did not employ MPNN to solve the DF2.

	DeepONet	PINO	U-NO	FNO	U-Net	MPNN
Ad1	0.0792	0.0191	0.0069	0.0128	0.0699	0.0451
AC2	0.9999	0.0191	0.0054	0.0084	0.1117	0.0079
DF2	0.3971	0.0709	0.0702	0.1328	0.0828	-

Table 8: L2RE of operator learning methods for Ad1,AC2 and DF2.

Listed in Table 9 are the training, inference and convergence time. We observe that for all three PDE problems, DeepONet demonstrates noteworthy efficiency.

		DeepONet	PINO	U-NO	FNO	U-Net	MPNN
t_{train}	Ad1	6.6e02	9.0e03	2.9e04	1.1e04	2.4e04	2.6e04
	AC2	5.3e03	8.5e05	2.0e05	9.0e04	6.7e04	3.1e04
	DF2	1.1e03	1.4e05	2.2e04	8.4e03	1.3e04	-
t_{infer}	Ad1	8.0e-04	2.1e-03	2.9e-01	2.5e-01	7.7e-02	1.8e-02
	AC2	6.1e-02	1.7e-02	4.3e-01	3.6e-01	3.7e-01	3.8e-01
	DF2	1.3e-03	2.1e-03	2.3e-01	2.1e-01	3.6e-03	-
t_{conv}	Ad1	5.7e02	7.2e03	2.7e04	8.7e03	2.3e04	2.6e04
	AC2	3.2e01	8.5e04	4.2e04	1.9e04	6.7e04	3.1e04
	DF2	8.8e02	8.2e03	1.8e04	5.1e03	1.2e04	-

Table 9: Training, inference and convergence time of operator learning methods.

5 Concluding Remarks

In this study, we evaluate 6 function learning and 6 operation learning NN methods for solving 19 PDEs. Our experiments with function learning methods demonstrate that, for the PDEs with smooth solution, the PINN exhibits highest accuracy, and the DRM and WAN exhibits excellent training speed. In the case of PDE with non-smooth solution, the DFVM and WAN perform very well. Our experiments with operator learning methods reveal that, for autoregressive tasks, both U-NO and FNO perform admirably, while for non-autoregressive tasks, DeepONet exhibits the best performance.

Certainly, many other important NN methods, including variants of PINN, and many other fundamental PDE tasks such as multiscale problems in various fields have not been covered yet in PDENNEval. Our evaluation system is scalable and easily accessible, everyone is welcome to contribute data and/or codes to the benchmark.

Developing PDE solvers with high accuracy, efficiency, and strong generalization is the pursuit of many computational scientists. The combination of traditional numerical methods and NN methods provides technical possibilities for achieving this pursuit. It can be anticipated that in the near future, there will likely be the emergence of many large NN models capable of efficiently solving lots of complex PDEs simultaneously (see e.g. [Yang *et al.*, 2023a; Yang *et al.*, 2023b]). Therefore, in the future, we will pay much attention to the evaluation of large NN models for solving PDE problems.

Acknowledgments

This work was supported by the National Science and Technology Major Project (2022ZD0117805), by the National Natural Science Foundation of China under grants 92370113 and 12071496, and by the Natural Science Foundation of the Guangdong Province grant 2023A1515012097.

We thank graduate students Haolong Fan, Hongji Li and Changye He, all from Sun Yat-sen University, for their helps with numerical experiments.

References

[Barron, 1994] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14:115–133, 1994.

[Brandstetter *et al.*, 2022a] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for PDE modeling. *arXiv preprint arXiv:2209.04934*, 2022.

[Brandstetter *et al.*, 2022b] Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers. *arXiv preprint arXiv:2202.03376*, 2022.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[Cen and Zou, 2023] Jianhuan Cen and Qingsong Zou. Deep finite volume method for high-dimensional partial differential equations. *arXiv preprint arXiv:2305.06863*, 2023.

[Chen *et al.*, 2022] Jingrun Chen, Xurong Chi, Zhouwang Yang, et al. Bridging traditional and machine learning-based algorithms for solving PDEs: The random feature method. *arXiv preprint arXiv:2207.13380*, 2022.

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dosovitskiy *et al.*, 2020] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[E and Yu, 2018] Weinan E and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[E *et al.*, 2017] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

[Gupta and Brandstetter, 2022] Jayesh K Gupta and Johannes Brandstetter. Towards multi-spatiotemporal-scale generalized pde modeling. *arXiv preprint arXiv:2209.15616*, 2022.

[Han *et al.*, 2020] Jihun Han, Mihai Nica, and Adam R Stinchcombe. A derivative-free method for solving elliptic partial differential equations with deep neural networks. *Journal of Computational Physics*, 419:109672, 2020.

[Hao *et al.*, 2023] Zhongkai Hao, Jiachen Yao, Chang Su, Hang Su, Ziao Wang, Fanzhi Lu, Zeyu Xia, Yichi Zhang, Songming Liu, Lu Lu, et al. PINNacle: A comprehensive benchmark of physics-informed neural networks for solving PDEs. *arXiv preprint arXiv:2306.08827*, 2023.

[He *et al.*, 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[He *et al.*, 2022] Juncai He, Lin Li, and Jinchao Xu. Approximation properties of deep relu cnns. *Research in the mathematical sciences*, 9(3):38, 2022.

[Hornik, 1991] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[Krizhevsky *et al.*, 2012] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[Kunz and Luebbers, 1993] Karl S Kunz and Raymond J Luebbers. *The finite difference time domain method for electromagnetics*. CRC press, 1993.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[Li *et al.*, 2020] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[Li *et al.*, 2021] Zongyi Li, Hongkai Zheng, Nikola B. Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. Physics-informed neural operator for learning partial differential equations. *CoRR*, abs/2111.03794, 2021.

[Limousin *et al.*, 2007] Gaudet Limousin, Jean-Paul Gaudet, Laurent Charlet, Stéphanie Szenknect, Véronique Barthès, and Mohamed Krimissa. Sorption isotherms: A review on physical bases, modeling and measurement. *Applied geochemistry*, 22(2):249–275, 2007.

[Lu *et al.*, 2021a] 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(3):218–229, 2021.

[Lu *et al.*, 2021b] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM review*, 63(1):208–228, 2021.

[Lu *et al.*, 2022] 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*, 393:114778, 2022.

[Rahman *et al.*, 2022] Md Ashiqur Rahman, Zachary E Ross, and Kamyar Azizzadenesheli. U-NO: U-shaped neural operators. *arXiv preprint arXiv:2204.11127*, 2022.

[Raissi and Karniadakis, 2018] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[Raissi *et al.*, 2019] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. 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.

[Raissi, 2024] Maziar Raissi. Forward-backward stochastic neural networks: deep learning of high-dimensional partial differential equations. In *Peter Carr Gedenkschrift: Research Advances in Mathematical Finance*, pages 637–655. World Scientific, 2024.

[Ronneberger *et al.*, 2015] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[Takamoto *et al.*, 2022a] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Dan MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench Datasets, 2022.

[Takamoto *et al.*, 2022b] Makoto Takamoto, Timothy Praditia, Raphael Leiteritz, Daniel MacKinlay, Francesco Alesiani, Dirk Pflüger, and Mathias Niepert. PDEBench: An extensive benchmark for scientific machine learning. *Advances in Neural Information Processing Systems*, 35:1596–1611, 2022.

[Takamoto *et al.*, 2023] Makoto Takamoto, Francesco Alesiani, and Mathias Niepert. Learning neural pde solvers with parameter-guided channel attention. *arXiv preprint arXiv:2304.14118*, 2023.

[Toro *et al.*, 1994] Eleuterio F Toro, Michael Spruce, and William Speares. Restoration of the contact surface in the hll-riemann solver. *Shock waves*, 4:25–34, 1994.

[Van Leer, 1979] Bram Van Leer. Towards the ultimate conservative difference scheme. v. a second-order sequel to godunov's method. *Journal of computational Physics*, 32(1):101–136, 1979.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Virmaux and Scaman, 2018] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.

[Yang *et al.*, 2023a] Liu Yang, Siting Liu, Tingwei Meng, and Stanley J. Osher. In-context operator learning with data prompts for differential equation problems. *Proceedings of the National Academy of Sciences*, 120(39):e2310142120, 2023.

[Yang *et al.*, 2023b] Liu Yang, Siting Liu, and Stanley J. Osher. Fine-tune language models as multi-modal differential equation solvers. *arXiv preprint arXiv:2308.05061*, 2023.

[Zang *et al.*, 2020] Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020.

Supplementary Materials

6 Detailed PDEs and Datasets

We present the specific representation of 16 PDEs along with their respective datasets. Within each data file, we store multiple samples, each representing an approximate solution to the associated PDE. These solution samples are obtained through traditional numerical methods, as outlined in Section 3. Each sample, characterized by unique initial values, is drawn from a specific distribution within a designated function space, further explained in the subsequent text.

1. **1D Advection equation (Ad1)**
   The PDE and its boundary condition are:

   $$\partial_t u + 0.1\partial_x u = 0, \quad (x,t) \in (0,1) \times (0,2],$$
   $$u(0,t) = u(1,t).$$

   The initial condition is chosen as

   $$u(x,0) = \sum_{i=1}^{2} A_i \sin(k_i x + \phi_i). \tag{12}$$

   Here $k_i = 2\pi n_i, (n_i \in \{1,2,\cdots,8\}, i = 1,2)$ are the wave numbers, $A_i \in [0,1]$ the float numbers, $\phi_i \in (0, 2\pi)$ the phase, all according to the uniform distribution.

   To generate dataset Ad1, we employ the equation given in (12) to generate 10,000 distinct initial conditions. Subsequently, we numerically solve these 10,000 initial value problems to obtain 10,000 solution samples. The solution values at discrete spatiotemporal points in each sample are stored, with the space domain $(0,1)$ discretized into 1024 equidistant points and the time domain $(0,2]$ discretized into 201 equidistant points.

2. **1D Diffusion-Reaction equation(DR1)**
   The PDE and its boundary condition are :

   $$\partial_t u - 0.5\partial_{xx} u - u(1-u) = 0, (x,t) \in (0,1) \times (0,1],$$
   $$u(0,t) = u(1,t).$$

   The initial condition takes the form as (12). The dataset in DR1 is generated using the same method employed for generating the Ad1 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with a spatial resolution of 1024 and a temporal resolution of 101.

3. **1D Burgers equation(Bu1)**
   The PDE and its boundary condition are:

   $$\partial_t u + u\partial_x u - 0.001\partial_{xx} u = 0, (x,t) \in (-1,1) \times (0,2],$$
   $$u(-1,t) = u(1,t) = 0,$$

   The initial condition takes the form (12). The dataset in Bu1 is generated using the same method employed for generating the Ad1 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with the space domain $(-1,1)$ discretized into 1024 equidistant points and the time domain $(0,2]$ discretized into 201 equidistant points.

4. **1D Diffusion-Sorption equation(DS1)**
   The PDE and its boundary condition are:

   $$\partial_t u - D/R(u)\partial_{xx} u = 0, \quad (x,t) \in (0,1) \times (0,500],$$
   $$u(0,t) = 1, \quad u(1,t) = D\partial_x u(1,t).$$

   Here $D = 5 \times 10^{-4}$ and the retardation factor $R(u)$ based on the Freundlich sorption isotherm [Limousin *et al.*, 2007]:

   $$R(u) = 1 + \frac{1-\phi}{\phi}\rho_s k n_f u^{n_f - 1},$$

   where $\phi = 0.29$ is the porosity of the porous medium, $\rho_s = 2880$ the bulk density, $k = 3.5 \times 10^{-4}$ the Freundlich's parameter, $n_f = 0.874$ the Freundlich's exponent. The initial condition is generated with a uniform distribution

   $$u(x,0) \sim \mathcal{U}(0, 0.2), \quad x \in \Omega, \tag{13}$$

   where x is a sampled coordinate.

   The space domain $(0,1)$ is discretized into 1024 equidistant points and the time domain $(0,500]$ is discretized into 201 equidistant points. The initial conditions are sampled at discrete points to create initial data points (13). We utilize 10,000 distinct initial values, solving them numerically to generate diverse samples. Each solution sample stores the solution values at discrete spatiotemporal points.

5. **1D Allen-Cahn equation(AC1)**
   The PDE and its boundary condition are:

   $$\partial_t u - \epsilon\partial_{xx} u + 5u^3 - 5u = 0, (x,t) \in (-1,1) \times (0,1],$$
   $$u(-1,t) = u(1,t), \quad \partial_x u(-1,t) = \partial_x u(1,t),$$

   where the parameter $\epsilon = 0.0001$. The initial condition takes the form as (12).

   The dataset in AC1 is generated using the same method employed for generating the Ad1 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with the spatial resolution of 1024, and the temporal resolution of 101.

6. **1D Cahn-Hilliard equation(CH1)**
   The PDE and its boundary condition are:

   $$\partial_t u - \partial_{xx}(\gamma_1(u^3 - u) - \gamma_2\partial_{xx} u) = 0, {\scriptstyle (x,t) \in (-1,1) \times (0,1]}$$
   $$u(-1,t) = u(1,t), \quad \partial_x u(-1,t) = \partial_x u(1,t).$$

   Here $\gamma_1 = 0.01, \gamma_2 = 10^{-6}$ are the parameters. The initial condition takes the form as (12).

   The dataset in CH1 is generated using the same method employed for generating the Ad1 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with the spatial resolution of 1024, and the temporal resolution of 101.

7. **1D Compressible Navier Stokes equation (N S1)**
   The PDE in the space domain $\Omega = (0,1)$ and the time domain $[0,1]$ is:

$$\partial_t \rho + \partial_x(\rho v) = 0,$$
$$\rho\left(\partial_t v + v \cdot \partial_x v\right) + \partial_x p - (\zeta + 4\eta/3)\partial_{xx} v = 0,$$
$$\partial_t \left(\epsilon + \frac{\rho v^2}{2}\right) + \partial_x \left[\left(\epsilon + p + \frac{\rho v^2}{2}\right) v\right] = 0,$$

where $\epsilon = 3p/2$ is the internal energy, $\eta = 0.1, \zeta = 0.1$ are the shear and bulk viscosity. The initial value function takes the form as (12).

The dataset in NS1 is generated using the same method employed for generating the Ad1 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with Ω discrete into 1024 equidistant points, and the time domain $[0, 1]$ discrete into 101 equidistant points.

8. **2D compressible Navier Stokes equation (NS2)**

The PDE in the space domain $\Omega = (0, 1)^2$ and the time domain $[0, 1]$ is:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \tag{14}$$
$$\rho\left(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}\right) + \nabla p - \eta \triangle \mathbf{v} - (\zeta + \eta/3)\nabla(\nabla \cdot \mathbf{v}) = 0, \tag{15}$$
$$\partial_t \left(\epsilon + \frac{\rho v^2}{2}\right) + \nabla \cdot \left[\left(\epsilon + p + \frac{\rho v^2}{2}\right) \mathbf{v}\right] = 0, \tag{16}$$

where $\mathbf{v}(\mathbf{x}, t) = (v_x, v_y), v = \sqrt{v_x^2 + v_y^2}$, $\eta = 0.1, \zeta = 0.1$. The initial value function is given by

$$\mathbf{v}(\mathbf{x}, 0) = \sum_{i=1}^{4} A_i sin(k_i \mathbf{x} + \phi_i), \tag{17}$$

where $A_i = c_s M/|k_i|$, $c_s$is the sound velocity and $M = 0.1$ is Mach number, other parameter in (17) can be found in (12).

To generate dataset NS2, we employ the equation given in (17) to generate 1000 distinct initial conditions. Subsequently, we numerically solve these 1000 problems to obtain 1000 solution samples. The solution values at discrete spatiotemporal points in each sample are stored, with Ω discretized into 128×128 equidistant points and the time domain $[0, 1]$ discretized into 101 equidistant points.

9. **3D compressible Navier Stokes equation(NS3)**

Consider 3D equation (14)-(16) in the space domain $\Omega = (0, 1)^3$, where $\mathbf{v}(\mathbf{x}, t) = (v_x, v_y, v_z)$,$\eta = 10^{-8}, \zeta = 10^{-8}$. The initial condition is also given by (17), where $A_i = c_s M/|k_i|^2$ and $M = 1$.

The dataset in NS3 is generated using the same method employed for generating the NS2 dataset. with the distinction being that the number of samples is only 100. Each solution sample store the solution values at discrete spatiotemporal points, with Ω discretized into $128 \times 128 \times 128$ equidistant points and the time domain $[0, 1]$ discretized into 21 equidistant points.

10. **2D Burgers equation(Bu2)**

The PDE and the boundary conditions are:

$$\boldsymbol{u}_t + \boldsymbol{u} \cdot \nabla \boldsymbol{u} - \frac{\epsilon}{\pi} \Delta \boldsymbol{u} = 0, \quad (\mathbf{x}, t) \in \Omega \times (0, 1],$$
$$\boldsymbol{u}(\mathbf{x}, t) = 0, \quad (\mathbf{x}, t) \in \partial\Omega \times (0, 1].$$

Here $\boldsymbol{u}(\mathbf{x}, t) = (u, v)^T, \epsilon = 0.01, \Omega = (0, 2)^2$. The initial conditions

$$\boldsymbol{u}(\mathbf{x}, 0) = (randn(1), randn(1))^T, \tag{18}$$

where $randn(1)$ is a random number between $[0, 1]$.

To generate dataset Bu2, we employ the equation (18) to generate 1000 distinct initial conditions. Subsequently, we numerically solve these 1000 initial value problems to obtain 1000 solution samples. The solution values at discrete spatiotemporal points in each sample are stored, with Ω discretized into 128×128 equidistant points and the time domain $(0, 1]$ discretized into 101 equidistant points.

11. **2D Darcy Flow equation(DF2)**

The PDE and its boundary condition are:

$$-\nabla(a \nabla u) = 1, \quad \mathbf{x} \in \Omega,$$
$$u(\mathbf{x}) = 0, \quad \mathbf{x} \in \partial\Omega,$$

where $\Omega = (0, 1)^2$, and a is a binary function taking the values 0.1 or 1.

For the DF2 dataset, we discretize Ω into 128×128 points. The value of a at each coordinate point is either 0.1 or 1. Form 10,000 distinct values of a to create 10000 unique boundary value problems, and solve them numerically to obtain the corresponding set of 10,000 samples.

12. **2D Shallow Water equation(SW2)** The PDEs are:

$$\partial_t h + \partial_x hu + \partial_y hv = 0,$$
$$\partial_t hu + \partial_x \left(u^2 h + \frac{1}{2} g_r h^2\right) + g_r h \partial_x b = 0,$$
$$\partial_t hv + \partial_y \left(v^2 h + \frac{1}{2} g_r h^2\right) + g_r h \partial_y b = 0,$$

where $\Omega = [-2.5, 2.5]^2$ is the space domain, [0,1] the time domain, u, v the velocities in horizontal and vertical direction, h the water depth, b a spatially varying bathymetry, hu, hv the directional momentum components, and g_r the gravitational acceleration. We initialize the water height as a circular bump in the center of the domain

$$h(x, y, 0) = \begin{cases} 2.0, & \text{for } r < \sqrt{x^2 + y^2} \\ 1.0, & \text{for } r \geq \sqrt{x^2 + y^2} \end{cases} \tag{19}$$

where $r \sim \mathcal{U}(0.3, 0.7)$.

To generate dataset SW2, we randomly sample the radius r in the equation (19) to generate 1000 distinct initial conditions. Subsequently, we numerically solve these 1000 problems to obtain 1000 solution samples. The solution values at discrete spatiotemporal

points in each sample are stored, with the spatial domain $[-2.5, 2.5]^2$ discretized into 128×128 equidistant points and the temporal domain $[0, 1]$ discretized into 101 equidistant points.

13. **2D Allen-Cahn equation(AC2)**
The PDE and its boundary condition are:

$$\begin{aligned}
&\partial_t u - \epsilon^2 \triangle u + u^3 - u = 0, && (x, y, t) \in (0, 1)^2 \times (0, 1], \\
&u(1, y, t) = u(0, y, t), && (y, t) \in [0, 1] \times (0, 1], \\
&\partial_x u(1, y, t) = \partial_x u(0, y, t) && (y, t) \in [0, 1] \times (0, 1], \\
&u(x, 1, t) = u(x, 0, t), && (x, t) \in [0, 1] \times (0, 1] \\
&\partial_y u(x, 1, t) = \partial_y u(x, 0, t) && (x, t) \in [0, 1] \times (0, 1]
\end{aligned}$$

Here $u(x, y, t)$ is the phase field, and $\epsilon = 0.01$ is a parameter. The initial condition is represented by a hyperbolic sine curve(17) where $k_i = 2\pi \{n_i/2\}$.

The dataset in AC2 is generated using the same method employed for generating the NS2 dataset. Each solution sample stores the solution values at discrete spatiotemporal points, with the spatial domain $[0, 1]^2$ discretized into 128×128 equidistant points and the temporal domain $[0, 1]$ discretized into 101 equidistant points.

14. **2D Black-Scholes-Barenblatt equation(BS2)**
The PDE and its boundary condition are:

$$\begin{aligned}
\partial_t u = &-\frac{1}{2} \operatorname{Tr}(0.16 \operatorname{diag}(\mathbf{x}^2) \operatorname{Hess}_{\mathbf{x}} u) \qquad (20) \\
&+ 0.05(u - \nabla u \cdot \mathbf{x}), (\mathbf{x}, t) \in [0, 1]^2 \times [0, 1), \\
u(\mathbf{x}, 1) = &\, c\|\mathbf{x}\|^2, \qquad \mathbf{x} \in [0, 1]^2. \qquad (21)
\end{aligned}$$

The equations (20)-(21) discussed in [Raissi, 2024] have an explicit solution:

$$u(\mathbf{x}, t) = \exp\left(\left(0.05 + 0.4^2\right)(1 - t)\right) c\|\mathbf{x}\|^2,$$

where $c \sim \mathcal{U}(0, 1)$.

To generate dataset BS2, we employ the different values of c given in (21) to generate 1000 distinct Black-Scholes-Barenblatt equations. Subsequently, we numerically solve these 1000 problems to obtain 1000 solution samples. The solution values at discrete spatiotemporal points in each sample are stored, with the space domain $[0, 1]^2$ discretized into 128×128 equidistant points and the time domain $[0, 1]$ discretized into 101 equidistant points.

15. **3D Maxwell's equation(Ma3)**
This system of PDE includes equations for both the electric field $\mathbf{E}$ and the magnetic field $\mathbf{H}$.

$$\begin{aligned}
&\nabla \cdot \mathbf{E} = 0, \qquad \nabla \cdot \mathbf{H} = 0, \\
&\nabla \times \mathbf{E} = -\frac{\mu}{c} \frac{\partial \mathbf{H}}{\partial t}, \quad \nabla \times \mathbf{H} = \frac{\epsilon}{c} \frac{\partial \mathbf{E}}{\partial t}.
\end{aligned}$$

Here, c is the velocity of light, $\varepsilon = 10$ the permittivity of free space, and $\mu = 1$ the permeability of free space. Following the [Brandstetter *et al.*, 2022b], we randomly place 18 different light sources outside the cube, which emit light with different amplitudes and phase shifts, with 6 on each plane. As a result, the generated $\mathbf{E}$ and $\mathbf{H}$ fields interfere with each other. The wavelength of the emitted light is 10^{-5} m.

To generate dataset Ma3, We utilize 1000 distinct setting and numerically solve them to obtain 1000 solution samples. The solution values at discrete spatiotemporal points in each sample are stored, with the space domain $[0, 1.6\text{E-}05]^3$ discretized into $32 \times 32 \times 32$ equidistant points and the time domain $[0, 1.91\text{E-}13]$ discretized into 8 equidistant points.

We utilize 1000 distinct setting and numerically solve to generate varied samples, consequently producing the Ma3 dataset.

16. **3D Euler equation (Eu3)**
The PDE in the space domain $\Omega = (0, 1)^3$ and the time domain $[0, 1]$:

$$\begin{aligned}
&\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \\
&\rho\left(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}\right) + \nabla p = 0, \\
&\partial_t \left[\epsilon + \frac{\rho v^2}{2}\right] + \nabla \cdot \left[\left(\epsilon + p + \frac{\rho v^2}{2}\right) \mathbf{v}\right] = 0,
\end{aligned}$$

where $\epsilon = 3p/2$ is the internal energy. The initial condition can be given by (17), where $A_i = c_s/|k_i|^2$ and c_s is the sound velocity.

The dataset in Eu3 is generated using the same method employed for generating the NS2 dataset, with the distinction that the number of solution samples is only 100. Each solution sample stores the solution values at discrete spatiotemporal points, with the space domain $[0, 1]^3$ discretized into $128 \times 128 \times 128$ equidistant points and the time domain $[0, 1]$ discretized into 21 equidistant points.

7 Detailed Description of NN Methods

The code implementation of the methods will be released publicly.

7.1 Function Learning Methods

A function learning-based method directly approximates the solution of a PDE using a neural network u_θ. The main difference in these methods lies in the construction of the PDE constraint loss $\mathcal{L}_{PDE}$, while the boundary and initial value losses are constructed in almost the same way. Moreover, with this type of method, u_θ is usually trained without data.

1. **DRM** is an early contribution to NN methods for PDEs, whose PDE loss is based on the variational form of the original equation established by the principle of minimal potential energy. For instance, the PDE loss for the Poisson equation is constructed as

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_1} \sum_{\mathbf{x} \in \mathcal{S}_{\text{PDE}}} \left(\frac{1}{2} \left|\nabla_{\mathbf{x}} u_\theta(\mathbf{x})\right|^2 - f(\mathbf{x}) u_\theta(\mathbf{x})\right).$$

2. **PINN** is a popular NN method for solving PDEs. Its PDE loss is designed using the PDE residuals:

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_1} \sum_{(\mathbf{x}, t) \in \mathcal{S}_{\text{PDE}}} \left[(\mathcal{F} u_\theta - f)(\mathbf{x}, t)\right]^2.$$

3. **WAN** is a NN method which trains the solution using the weak form of the PDE. In WAN, we use two neural networks, denoted by u_θ and φ_η, to simulate the approximate solution and the test function, respectively. Their respective parameters, θ and η, are learned through adversarial training. The PDE loss term is designed as

$$\mathcal{L}_{\text{PDE}} = \log|(\mathcal{F}u_\theta - f, \varphi_\eta)|^2 - \log\|\varphi_\eta\|_2^2.$$

4. **DFLM** is a NN method specifically for elliptic PDEs. Precisely, an elliptic equation

$$\frac{1}{2}\text{Tr}(\sigma\sigma^T \text{Hess}_x u) + F \cdot \nabla u - G = 0$$

is transformed to the Bellman equation

$$u(\mathbf{x}) = \mathbb{E}\left[u(X_t) - \int_0^t G(X_s, u(X_s))ds \middle| X_0 = \mathbf{x}\right] \quad (22)$$

by introducing a stochastic process $\{X_t\}$ satisfying the law

$$dX_t = F(X_t, u(X_t))dt + \sigma(X_t)dB_t.$$

The PDE loss of the DFLM is then defined by

$$\mathcal{L}_{\text{PDE}}(\theta) := \frac{1}{N_1}\sum_{\mathbf{x}_i \in \mathcal{S}_{\text{PDE}}} \Big(y(\mathbf{x}_i) - u_\theta(\mathbf{x}_i)\Big)^2, \quad (23)$$

where $y(\mathbf{x}_i)$ is the average of the values predicted with (22) at a certain number of neighborhood points sampled according to the stochastic process X_t. Obviously, the PDE loss defined by (23) does not involve any derivatives of the neural network function u_θ.

5. **RFM** is a NN method combining techniques from both traditional numerical methods and machine learning methods. The core of RFM is to represent the numerical solution $u_\theta(\boldsymbol{x})$ as a linear combination of M shallow network basis functions $\{\phi_m\}$ defined over the domain Ω. That is,

$$u_\theta(\boldsymbol{x}) = \sum_{m=1}^{M} u_m \phi_m(\boldsymbol{x}).$$

Here, ϕ_m are network basis functions whose internal weights are randomly initialized and will be fixed in subsequent steps. An instance of ϕ_m is a shallow neural function $\phi_m(\boldsymbol{x}) = \sigma(\boldsymbol{k}_m \cdot \boldsymbol{x} + b_m)$. The PDE loss $\mathcal{L}_{\text{PDE}}$ of RFM is chosen the same as that of the PINN. The coefficients u_m might be obtained by solving an algebraic system directly, instead of using a machine learning optimizer such as Adam.

6. **DFVM** is a NN method based on the local conservation forms of the original PDE. Its PDE loss takes the form

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_1}\sum_{(\mathbf{x},t)\in\mathcal{S}_{\text{PDE}}} \left[\int_{V_\mathbf{x}} (\mathcal{F}u_\theta - f)\,\mathrm{d}\mathbf{x}\right]^2,$$

where $V_\mathbf{x}$ is a deliberately chosen *control volume* associated with the point $\mathbf{x}$. Since the integral $\int_{V_\mathbf{x}} \mathcal{F}u_\theta$ can often be transformed to an integral on the boundary $\partial V_\mathbf{x}$ by the divergence theorem, the calculation of a DFVM loss only involves the first-order derivatives of u_θ which can be efficiently implemented by the automatic differentiation mechanism (AD).

7.2 Operator Learning Methods

1. **FNO** is an autoregressive NN method that learns the mappings from the solution of the kth time step to the solution of the $(k+1)$th time step. The core part of FNO is the Fourier layer: input high-dimensional representation v. Apply Fourier transform F, linearly transform low Fourier modes R, and filter out high Fourier modes. Then apply inverse Fourier transform F^{-1}. Finally, add the output with the output obtained by applying local linear transformation W.

Then the full architecture of FNO is: start from an input function $\mathbf{a} = u(\cdot, t_k)$. Lift to a higher dimension channel space by a neural network P. Apply several Fourier layers and activation functions. Project back to the target dimension by a neural network Q. Output $u(\cdot, t_{k+1})$.

2. **MPNN** is an autoregressive NN method that uses temporal bundling techniques to predict the next K time steps' solutions based on the solutions of the previous K steps all at once. As a general framework designed for handling graph data, MPNN first discretizes the spatial domain using a grid and models the grid as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Nodes represent grid cells, and edges represent the spatial adjacency relationship between cells.

The MPNN model consists of three main components: encoder, processor, and decoder. The encoder computes node embeddings by mapping the last K solution values, node position, current time, and equation embedding to a node embedding vector. The processor updates the graph through M steps by generating messages, aggregating neighbor messages, and updating node embeddings. The decoder projects node embeddings onto the solution values for the next K time steps. The processor output is fed into a CNN to compute the vector $\boldsymbol{d}_i = (d_i^1, ..., d_i^K)$ corresponding to a different time step. The solution is computed as:

$$\hat{\boldsymbol{u}}_i^{k+l} = \hat{\boldsymbol{u}}_i^k + (t_{k+l} - t_k)\boldsymbol{d}_i^l, \quad 1 \le l \le K.$$

3. **U-Net** is an autoregressive method that uses a convolutional neural network with an encoder-decoder structure to solve time-dependent PDEs. Within this framework, the model is trained to learn the mapping from a sequence of solutions at preceding K time steps $\hat{\boldsymbol{u}}^{k-K+1:k} = \{\hat{\boldsymbol{u}}^{k-l+1}, ...\hat{\boldsymbol{u}}^k\}$ to the subsequent time step solution $\hat{\boldsymbol{u}}^{k+1}$, formalized as:

$$\hat{\boldsymbol{u}}^{k+1} = \text{U-Net}(\hat{\boldsymbol{u}}^{k-K+1:k}; \boldsymbol{\theta}).$$

When using U-Net to solve specific PDEs, the initial sequence of solutions for the first K time steps $\hat{\boldsymbol{u}}^{0:K-1} = \boldsymbol{u}^{0:K-1}$ is computed by numerical method, and the model generates solutions $\hat{\boldsymbol{u}}^k = \{u_\theta(\mathbf{x}_1, t_k), \cdots, u_\theta(\mathbf{x}_m, t_k)\}$ for the ensuing time steps auto-regressively.

4. **U-NO** is an autoregressive method that learns a mapping from a sequence of solutions from the previous K time steps to solutions from subsequent time steps, formalized as:

$$\hat{\boldsymbol{u}}^{k+1} = \text{UNO}(\hat{\boldsymbol{u}}^{k-K+1}, ..., \hat{\boldsymbol{u}}^k; \theta).$$

U-NO adopts a U-shaped architecture, a design that gradually maps the input function to functions defined on smaller domains (encoding), and then reverses this operation to generate an appropriate output function (decoding). The encoder and decoder parts are connected by skip connections. For the implementation of the inner integral operator, U-NO leverages the Fourier transform-based integration method developed in FNO.

5. **DeepONet** is a non-autoregressive method that learns the mappings from the input function $\mathbf{a} = u(\cdot, 0)$ to the solution $u(\cdot, t)$ for $t \in (0, T]$. DeepONet constructs a neural operator architecture that encodes the input function $\mathbf{a}$ and the query location $\mathbf{y} \in \Omega$ by two separated FNNs respectively. These two FNNs are referred to as the branch net (u_B) and the trunk net (u_T). The architecture computes the output by performing the dot-product of the encoded vectors obtained from the branch and trunk nets, along with an additional scalar bias $\hat{b}$:

$$\mathcal{G}_\theta(\mathbf{a})(\mathbf{y}) = u_B(\mathbf{a}; \theta_B) \cdot u_T(\mathbf{y}; \theta_T) + \hat{b},$$

where the input function $\mathbf{a}(\cdot)$ is discretized into the flattened tensor $\{\mathbf{a}(\mathbf{x}_1), \mathbf{a}(\mathbf{x}_2), \cdots, \mathbf{a}(\mathbf{x}_n)\}$ obtained from the grid positions of the spatial space.

6. **PINO** is a non-autoregressive method that represents a physics-driven extension of the FNO, introducing a simultaneous utilization of data and physics information to learn a resolution-invariant neural operator in Fourier space. This method deals with temporal PDEs in a non-autoregressive manner, mapping the initial condition function $u(\cdot, 0)$ to $u(\cdot, t), t \in [0, T]$, akin to the continuous-time scheme of PINN. However, in the loss of PINO, spatial or temporal derivatives of the NN function are calculated using traditional difference schemes on grid positions instead of using AD directly.

8 Some Details of Datasets

As already mentioned in the main context, the PDENNEval datasets consist of 16 data files generated by using traditional numerical methods. The data in the files Ad1, DR1, Bu1, DS1, NS1, NS2, DF2, SW2, and NS3 are downloaded directly from the PDEBench library and those in files AC1, CH1, AC2, Bu2, BS2, Ma3, Eu3 are generated by ourselves. All datasets are stored in HDF5 format. To specify the details of the PDE, we also give a full name for each dataset, following the file naming convention in PDEBench as ${spatial dimension}_${PDE name}_${PDE parameters}_${other configs}.hdf5. For example, the full name of the data file Bu2 is given as *2D_Burger_Epsilon0.01.hdf5*. Significantly, while each current file houses just one variable, we intend to broaden this database in the future. This expansion might encompass incorporating various coefficients or modifying boundary conditions, aiming to enhance the overall scope and logical structure of our database.

Each dataset file consists of spatial coordinates, temporal coordinates and several solution samples with the sample size N_{sa} ranging from 100 to 10000. We emphasize here that from one specific initial value function, we generate one corresponding sample.

There are two types of organizational structures for our dataset files: single-group (Ad1, DR1, Bu1, AC1, CH1, NS1, NS2, AC2, DF2, BS2, NS3, Eu3) and multi-group (DS1, Bu2, SW2, Ma3). Using the dataset file of 1D compressible Navier Stokes equation as an illustration of single-group organizational structure, the equation involves variables ρ, p, v, with the sample size N_{sa} of 10000. The temporal resolution N_t is 101, and the spatial resolution N_x is 1024. In the context of the single-group, there are three arrays with a shape of $(N_{\text{sa}}, N_t, N_x)$ that store the solution data corresponding to the three variables of the equation. The spatial coordinates and temporal coordinates of the equation are stored in arrays with shapes of $(N_{x_i},)$ and $(N_t,)$ named with 'coordinate' as a suffix. Specially, since the 2D Darcy Flow equation is time-independent, its dataset file does not include temporal coordinates and the array shape for storing solutions is $(N_{\text{sa}}, 1, N_x, N_y)$. Additionally, the values of the function a are stored in an array named 'nu', with the shape of $(N_{\text{sa}}, 1, N_x, N_y)$. Taking the dataset file of 2D Burgers equation as an illustration of a multi-group organizational structure, the equation variables are u, v, with a sample number N_{sa} of 1000, the temporal resolution N_t is 101, and the spatial resolution $N_x \times N_y$ is 128×128. For the multi-group organizational structure, there are N_{sa} groups in file, each containing an array named 'data' and a group named 'grid'. The 'data' array with shape $(N_t, N_x, N_y, 2)$ holds the solution of one sample. The 'grid' group contains three arrays named 'x', 'y' and 't' with shapes of $(N_x,)$, $(N_y,)$ and $(N_t,)$, which store the spatial and temporal coordinates respectively.

9 Details of Metrics

In this section, we provide a detailed presentation on how to define and calculate the metrics mentioned in the main text.

We begin with the presentation of three time metrics.

1. **Training time**. We define

$$t_{\text{train}} = N_{\text{epoch}} \times t_{\text{epoch}},$$

where t_{epoch} is the training time for one single epoch and N_{epoch} is the number of epochs for neural network training. Note that the definition of t_{epoch} are different for function learning and operator learning methods:

- For function learning methods, t_{epoch} denotes the time needed for one iteration of all NN parameters.
- For operator learning methods, t_{epoch} represents the duration to complete one traverse through the entire training dataset.

By default, N is set to be 25000 for function learning methods and set to be 500 for operator learning methods.

2. **Inference time**. We define the inference time t_{infer} as the time required for a trained NN model to infer the solution of an equation. Precisely,

- For function learning methods, t_{infer} is the time of calculating the values of a trained NN solution at certain coordinates in the dataset.
- For operator learning methods, t_{infer} is the time of calculating a solution (actually the values of the solution at certain coordinates) by applying a trained NN operator

to a certain initial value function (actually the values of the initial value function at certain coordinates).

Note that to ensure a fair comparison between these two types of NN models, we let the coordinates (the query location) of the same PDE problem be the same during inference.

3. **Convergence time**. We define

$$t_{\text{conv}} = N_{\text{conv}} \times t_{\text{epoch}},$$

where N_{conv} is the number of the epochs that the NN model converges. In other words, at the N_{conv}-th epoch, the loss of the NN method will almost not decay in the sense that the relative loss error between n consecutive epochs will be less than 0.001. Note that different models may employ distinct n.

Next, we present the definitions and calculation approaches for the other three metrics.

4. **The definition of L_∞ norm error**. We define

$$\text{mERR} = \max_{(\mathbf{x},t)\in\Omega\times[0,T]} |u_\theta(\mathbf{x},t) - u(\mathbf{x},t)|.$$

5. **The calculation of L2RE and mERR**, Both L2RE and mERR are metrics to evaluate the accuracy of an NN solution. Since the $L2$ and L_∞ norms of a function need to be calculated by using a certain discrete scheme, next, we specify how to choose the discrete points in the calculation of these two metrics.

- For the function learning methods, the discrete points are all spatiotemporal coordinates in one sample of the data file. And for three Poisson problems which have no sampling points a prior, the discrete points are 10000 points randomly sampled from the domain of a PDE.
- For the operation learning methods, the discrete points are all spatiotemporal coordinates in the test sample of the data file.

Note that if a PDE problem involves multiple unknown functions to be solved, we perform separate calculations of L2RE and mERR for each unknown function.

6. **The calculation of Lipschitz constant**. The exact computation of the Lipschitz constant is NP-hard even for the simplest 2-layer MLP with ReLU as activation function, so most existing works focus on finding a compact upper bound. We use the SeqLip algorithm proposed in [Virmaux and Scaman, 2018] to estimate the Lipschitz upper bound for some of our trained models. The Lipschitz constant is important for characterizing many properties of a neural network, including robustness[Huang *et al.*, 2021; Szegedy *et al.*, 2013; Tsuzuku *et al.*, 2018], generalization[Bartlett *et al.*, 2017] and explanation[Fel *et al.*, 2022].

We compute the Lipschitz upper bound for all function learning methods, whose neural networks are MLP. A K-layer MLP u_θ: $\mathbb{R}^n \to \mathbb{R}^m$ is the function defined by:

$$u_\theta(x) = T_K \circ \rho_{K-1} \circ \cdots \circ \rho_1 \circ T_1(x), \tag{24}$$

where $T : x \to M_k x + b_k$ is an affine transformation and $\rho_k : x \to g_k(x)$ is a non-linear activation function. The Lipschitz constant of MLPs has an explicit formula:

$$\text{Lips} = \sup_{x\in\mathbb{R}^n} \left\| M_K \operatorname{diag}\left(g'_{K-1}(x_{K-1})\right) \ldots \operatorname{diag}\left(g'_1(x_1)\right) M_1 \right\|_2, \tag{25}$$

where $x_k = T_k \circ \rho_{k-1} \circ \cdots \circ \rho_1 \circ T_1(x)$ is the output of the k-th layer. In practice, the algorithm splits the norm operator in $K-1$ parts with the SVD decomposition of each matrix $M_i = U_i \Sigma_i V_i^\top$ and calculates the Lipschitz upper bound of MLP by the scheme

$$\text{Lips} = \prod_{k=1}^{K-1} \max_{\sigma_k\in[0,1]^{n_k}} \left\| \widetilde{\Sigma}_{k+1} V_{k+1}^\top \operatorname{diag}(\sigma_k)\, U_k \widetilde{\Sigma}_k \right\|_2, \tag{26}$$

where n_k is the number of neurons in the k-th layer, σ_k corresponds to all possible derivative values of the activation function, $\widetilde{\Sigma}_k = \Sigma_k$ if $k \in \{1, \cdots, K\}$ and $\widetilde{\Sigma}_k = \Sigma_k^{1/2}$ otherwise.

Additionally, we estimate the Lipschitz upper bound for U-Net and DeepONet methods. Recall that U-Net has an encoder-decoder structure including skip connections where vector concatenation happens. Concretely, the input of the model decoder is obtained by concatenating the output of the previous decoder with the output of the corresponding encoder. Because SeqLip can only handle sequential neural networks, we modified the original algorithm to cope with this special case. The architecture DeepONet used consists of two sub-networks, named *branch* and *trunk*, both of which are MLPs. Each sub-network takes different data as inputs. During estimation, we keep the input of the *branch* fixed and compute the Lipschitz constant for the *trunk*. The input to *branch* comes from the test samples in the dataset. We use the maximum estimate among 100 samples as the Lipschitz upper bound for DeepONet.

10 The Setting of Hyperparemeters

In this section, we present the setting of hyperparameters used in our evaluation experiments. Following the structure in Appendix B, we will present them by dividing the NN methods into function learning and operator learning types.

10.1 Settings for Function Learning Methods

For all function learning methods, we use the following common hyperparameters setting in our experiments. The underlying neural network is chosen as a fully connected network with a depth of 6, comprising 40 neurons in each layer. The activation function is chosen to be tanh, along with the Adam optimizer and a learning rate of 1E-03. The training process involves 15,000 epochs for the three Poisson problems and 25,000 epochs for the other 16 PDE problems.

Other special settings for some specific methods are listed below.

- **Hyperparameters for DFVM**: It involves parameters like a cube-shaped control volume with a radius of 10^{-5}, 1 numerical integration point within the control volume, and $2 \times d$ numerical integration points at the boundary of the control volume. d is the spatial dimension.
- **Hyperparameters for DFLM**: It introduces specific attributes, such as 10 diffusion trajectories and a discrete time step size of 0.0005 for Brownian motion.
- **Hyperparameters for WAN**: We set the boundary loss weights for different problems, where λ_{P_S}=4E+06, λ_{P_L}=1E+05, and λ_{P_H}=1000.

- **Hyperparameters for RFM**: We choose 1,000 global basis functions and utilize the tanh function as the activation function. The weights and biases of the linear layers are initialized with a uniform distribution between $(-1, 1)$.

	SR	TR	batch size	learn rate	weight decay	initial step	unroll step	N_{epoch}
Ad1	1024/4	201/5	64	1E-3	1E-4	10	20	500
DR1	1024/4	101/1	64	1E-3	1E-4	10	20	500
Bu1	1024/4	201/5	64	1E-3	1E-4	10	20	500
DS1	1024/4	101/1	64	1E-3	1E-4	10	20	500
AC1	1024/4	101/1	64	1E-3	1E-4	10	20	500
CH1	1024/4	101/1	64	1E-3	1E-4	10	20	500
NS1	1024/4	101/1	64	1E-3	1E-4	10	20	500
Bu2	128/1	101/1	8	1E-3	1E-4	10	20	500
NS2	128/2	21/1	32	1E-3	1E-4	10	20	500
DF2	128/1	-	64	1E-3	1E-4	1	1	500
SW2	128/1	101/1	8	1E-3	1E-4	10	20	500
AC2	128/1	101/1	8	1E-3	1E-4	10	20	500
BS2	128/1	101/1	8	1E-3	1E-4	10	20	500
Ma3	32/1	8/1	2	1E-3	1E-4	2	-	500
Eu3	128/2	21/1	2	1E-3	1E-4	10	20	500
NS3	128/2	21/1	2	1E-3	1E-4	10	20	500

Table 10: Hyperparameters for the operator learning methods. Here SR denotes spatial resolution and TR denotes temporal resolution.

10.2 Settings for Operator Learning Methods

For all operator learning NN methods, the datasets are split into 90% training and 10% validation and testing. Table 10 presents the default hyperparameter settings for solving 16 different equations.

Other special settings for operator learning methods are listed below.

1. **Hyperparameters for FNO**. We use the settings as in the PDEBench. The underlying neural network has 4 Fourier layers with 12 modes in Fourier space. The width of each Fourier layer is set to 20 and the GELU is chosen as the activation function. The width of fully connected layer is 128.
2. **Hyperparameters for DeepONet**. We implement DeepONet with reference to the codes in DeepXDE [Lu *et al.*, 2021b]. The model consists of two sub-networks: branch net and trunk net, both are MLPs with 4 and 3 hidden layers of width 128 respectively, using tanh as the activation function. The output dimensions of two sub-networks are set to the equal value of $N_d \times 128$, where N_d is the number of variables. Therefore, we can handle PDEs with multiple variables. In particular, we choose the approach mentioned in [Lu *et al.*, 2022], splitting the output of both the branch and the trunk into N_d groups, and the k-th group outputs the solution of the k-th variable. Based on DeepONet's structure, we only use this method to solve 1D and 2D problems. The batch size is configured as 50, and for 2D problems, the number of epochs is set to 200.
3. **Hyperparameters for PINO**. We implement PINO methods using FNO with four Fourier layers as backbone following [Li *et al.*, 2021]. For 1D problems, the width of the Fourier layer is set to 32 and the number of Fourier modes is set to 12. For 2D problems, the width is doubled to 64 and the number of modes remains 12. The activation function is GELU. The width of the fully connected layer is 128. Based on PINO's structure, we only use this method to solve 1D and 2D problems. The PINO is a hybrid approach that combines coarse-resolution training data with PDE constraints imposed at a higher resolution. We align the resolution of the PDE constraints with default settings and set the spatial resolution of the training data to one-quarter of it. The batch size is set to 50 for 1D problems and 2 for 2D problems. And the number of epochs is set to 200 for 2D problems.
4. **Hyperparameters for MPNN**. The parameter selection of this model is the same as the parameters in [Brandstetter *et al.*, 2022b], specifically, the size of node embedding is 128 and the number of GNN layers is 6. To ensure that MPNN incorporates previous time-step solutions of comparable length to other autoregressive operator learning methods (e.g., FNO, U-NO) in its input, we set "time-window" [Takamoto *et al.*, 2023] to 10.

   MPNN data through graphs constructed using the $k-$NN criterion. As the number of spatial dimensions in the problem increases, the graph size grows rapidly. Consequently, for 1D problems, we downsample spatial resolution 4 times as in the default settings and set k to 3. For 2D problems, we downsample spatial resolution 2 times and set k to 1. We exclusively employ this method for solving 1D and 2D problems. The learning rate for MPNN is set to 1E-04, the weight decay is set to 1E-8 and the batch size for NS2 is set to 8. We followed the training strategy in the original work but adjusted the number of model iterations in one epoch.
5. **Hyperparameters for U-Net**. We use the implementation in PDEBench [Takamoto *et al.*, 2022b], which extended the original implementation (2D-CNN) to the spatial dimension of the PDEs, i.e. 1D-CNN for 1D PDEs, 2D-CNN for 2D PDEs. The number of initial features is set to 32 for all problems. We train U-Net with the pushforward trick which provides better training stability and prediction accuracy.
6. **Hyperparameters for U-NO**. We implement U-NO referring to [Rahman *et al.*, 2022], where the inner integral operator is Fourier transform-based integration method developed in FNO. In our implementation, the network consists of 2 encoder blocks, 1 bottleneck blocks and 2 decoder blocks sequentially. Each block composed of a Fourier neural operator and a point-wise operator, using instance normalization and GELU activation function. For 1D problems, the modes of each block are set to $(11, 5, 5, 5, 11)$ and the width is set to 20. For 2D and 3D problems, the modes of each block are set to $(17, 7, 7, 7, 17)$ and the width is set to 30. Specially, when solving the 3D Maxwell's equation, the modes are set to$(10, 5, 5, 5, 10)$ and the width remains 30.

11 Detailed Experimental Results

In this section, we present a set of results for our evaluation experiments.

11.1 Performance of Function learning Methods

In this section, we present the results in two parts: one pertaining to the results of time-independent problems including the three Poisson problems and the Darcy Flow problem, and the other focusing on the outcomes of solving time-independent problems (e.g.,Ad1, Bu2).

Evaluation on time-independent problems

First, we present the results for three Poisson problems P_S, P_L, P_H, obtained by using function learning methods. We showcase our results which have not provided in the main text by the following two tables. Table 11 presents the evaluation results for the metrics t_{infer} and t_{conv}. Note that her, no convergence time has been reported for the RFM, since the RFM solution is obtained by solving a linear system of algebraic equations which is much faster than the training procedures of the other methods.

Metric	d	PDE	PINN	DFVM	RFM	WAN	DRM	DFLM
t_{infer}(s)	2	P_S	2.60e-04	2.46e-04	9.10e-05	5.70e-04	2.48e-04	2.46e-04
		P_L	3.25e-04	3.29e-04	9.50e-05	5.69e-04	3.18e-04	3.21e-04
	3	P_H	3.12e-04	3.20e-04	9.10e-05	5.44e-04	3.15e-04	3.04e-04
	5		3.16e-04	3.14e-04	1.02e-04	5.61e-04	3.10e-04	3.27e-04
	10		3.88e-04	4.19e-04	1.06e-04	4.40e-04	3.90e-04	3.76e-04
	20		3.74e-04	3.74e-04	1.07e-04	5.56e-04	3.71e-04	4.03e-04
	40		3.82e-04	3.95e-04	1.14e-04	4.52e-04	3.58e-04	3.84e-04
	80		3.98e-04	4.12e-04	1.25e-04	4.01e-04	4.16e-04	4.18e-04
	120		4.27e-04	4.33e-04	2.23e-04	5.46e-04	4.09e-04	4.19e-04
t_{conv}(s)	2	P_S	2.36e02	1.33e02	-	3.59e01	1.28e02	2.28e02
		P_L	2.32e02	2.86e02	-	1.72e02	9.47e01	1.27e02
	3	P_H	3.95e02	1.44e02	-	4.96e01	4.89e01	3.00e02
	5		5.48e02	1.55e02	-	1.08e02	7.02e01	3.03e02
	10		1.29e03	2.21e02	-	1.43e02	1.83e02	3.20e02
	20		2.51e03	2.90e02	-	2.84e02	2.08e02	3.24e02
	40		4.51e03	4.09e02	-	4.34e02	2.25e02	3.24e02
	80		7.47e03	7.68e02	-	8.09e02	3.22e02	3.30e02
	120		1.07e04	1.15e03	-	1.15e03	3.44e02	3.68e02

Table 11: Evaluating the Poisson equations in terms of the metrics t_{infer}(s) and t_{conv}(s)

We begin by reporting the evaluation results in terms of two accuracy metrics L2RE and mERR. Listed in Table 16 are results for 1D and 2D problems, while in Table 17 are results for 3D problems. It's worth noting that the data set chosen by the Maxwell equations has a relatively small spatiotemporal scale, and the magnitude of the true solution is on the order of 10^{-3}. L2RE is a relative value, while mERR is an absolute value, so the numerical results of L2RE and mERR differ greatly.

Table 12 encapsulates the assessment outcomes for the metrics mERR and Lips. From this table, it can be seen that for the Poisson equations, PINN and DFVM demonstrate great performance in terms of the mERR metric. Note that for problems P_S and P_L, the RFM model structure employs local basis functions, rendering it impossible to directly estimate the Lipschitz constant using the SeqLip algorithm. Therefore, we refrained from calculating their Lipschitz constants. In Table 12, according to the calculation formula for Lipschitz constant, we know that there are significant differences in Lipschitz for different network structures and weight matrix M_k. Nevertheless, for identical network structures, the computed Lipschitz constant should exhibit a relatively consistent value. This can be derived from the PINN and DFVM models, both employing the same set of network structures. The maximum fluctuation range of Lipschitz constant in RFM method.

Metric	d	PDE	PINN	DFVM	RFM	WAN	DRM	DFLM
mERR	2	P_S	0.3211	0.0097	0.2853	0.0283	0.2973	0.0088
		P_L	0.0067	0.0066	0.01877	0.0555	0.0051	0.0845
	3	P_H	0.0032	0.0031	0.0270	0.3053	0.0273	0.0412
	5		0.0030	0.0027	0.0962	0.2928	0.0048	0.0047
	10		0.0046	0.0046	0.5767	0.2998	0.0087	0.0070
	20		0.0058	0.0057	0.5144	0.3375	0.0092	0.0074
	40		0.0050	0.0049	0.3169	0.1683	0.0109	0.0075
	80		0.0048	0.0043	0.1862	0.0232	0.0173	0.0137
	120		0.0055	0.0049	0.1681	0.0359	0.0290	0.0293
Lips	2	P_S	1.39e02	4.57e01	-	2.02e00	6.95e01	4.67e01
		P_L	2.05e02	1.98e02	-	1.00e01	7.27e01	4.97e00
	3	P_H	2.50e01	2.50e01	3.95e03	7.33e00	2.19e01	2.10e01
	5		1.12e01	1.12e01	1.69e01	1.52e01	1.07e01	1.05e01
	10		8.79e00	8.78e00	1.05e00	3.06e01	7.88e00	7.67e00
	20		3.54e00	3.53e00	4.23e-01	2.41e01	3.78e00	3.59e00
	40		3.69e00	3.67e00	2.84e-01	1.89e01	4.24e00	3.90e00
	80		3.57e00	3.36e00	2.19e-01	2.02e00	4.69e00	4.34e00
	120		3.41e00	3.37e00	1.65e-01	1.68e00	4.15e00	4.14e00

Table 12: Evaluating the Poisson equations in terms of metrics mERR and Lips.

Secondly, we present the results of the Poisson equation with a sector domain (P_{Se}). In this case, we set $f = 0$, $g = h$, and $\Omega = \{(r, \theta) : 0 \leq r \leq 1, 0 \leq \theta \leq \frac{\pi}{6}\}$ in (11). The exact solution is given by $u(r, \theta) = r^{\frac{2}{3}} \sin(\frac{2}{3}\theta)$.

PDE	Metrics	PINN	DFVM	RFM	WAN	DRM	DFLM
P_{Se}	L2RE	0.0060	0.0060	0.0223	0.5786	0.0798	0.0049
	mERR	0.0298	0.0298	0.0068	0.2720	0.1581	0.0223
	t_{train}	2.43e03	1.21e02	1.15e01	8.19e02	1.20e02	1.14e02
	t_{infer}	2.42e-04	2.50e-04	5.75e-05	6.56e-04	2.40e-04	2.44e-04
	t_{conv}	2.42e02	1.21e02	-	4.67e02	1.20e02	1.14e02
	Lips	2.75e01	2.76e01	-	1.46e01	4.65e01	2.37e01

Table 13: Evaluating results for solving the 2D Darcy Flow equation.

In Table 13, we observe that DFLM exhibits superior performance in terms of the L2RE metric, and RFM exhibits superior performance in terms of the mERR metric. In addition, DRM excels in the evaluation of time metrics.

In Table 12-13, we provide the Lipschitz constant for solving PDEs with different models. However, for each distinct PDE, there exists a ground truth (GT) solution's Lipschitz constant (GT Lips) for the truth solution. Here, we present the Lipschitz constant of the true solution to facilitate comparison with the Lipschitz constant computed by the models. For PDEs where the true solution does not exist or is challenging to compute, we omit the GT solution's Lipschitz constant.

PDE	P_S	3d-P_H	5d-P_H	10d-P_H	20d-P_H
GT Lips	1.0000	1.4666	1.1361	0.8033	0.5680
PDE	40d-P_H	80d-P_H	120d-P_H	P_{Se}	
GT Lips	0.4017	0.2840	0.2319	1.0587	

Table 14: GT solution's Lipschitz constant for the poisson equations.

Thirdly, we present the results of the Dracy Flow equation in Table 15. we observe that PINN exhibits superior performance in terms of the L2RE metric, and RFM exhibits superior performance in terms of the mERR metric. In addition, DRM excels in the evaluation of time metrics.

PDE	Metrics	PINN	DFVM	RFM	WAN	DRM	DFLM
DF2	L2RE	0.0006	0.3850	1.5107	0.6131	0.5331	0.7024
	mERR	0.0033	0.3258	0.0029	0.0553	0.5743	0.9672
	t_{train}	2.22e03	2.25e02	2.63e03	4.78e02	1.12e02	5.32e02
	t_{infer}	3.79e-04	3.60e-04	9.75e-05	4.49e-04	4.14e-04	4.12e-04
	t_{conv}	2.22e03	2.24e02	6.58e02	3.76e02	1.12e02	1.42e02
	Lips	4.29e00	4.00e01	1.05e01	5.37e01	9.93e00	1.81e01

Table 15: Evaluating results for solving the 2D Darcy Flow equation.

Evaluation on time-dependent problems

		L2RE				mERR			
PDE		PINN	DFVM	RFM	WAN	PINN	DFVM	RFM	WAN
Ad1		0.9971	0.1042	0.9767	0.0711	0.7207	0.2116	0.6075	0.1163
DR1		0.0470	0.2434	0.5769	0.0713	0.0253	0.8152	0.9280	0.5037
Bu1		0.1736	0.0347	1.1146	0.5358	0.1062	0.0347	0.2861	0.5745
DS1		0.1920	0.5967	0.4296	0.6899	0.1062	0.4372	0.4243	0.8168
AC1		0.9999	1.4700	247.86	0.9620	1.0078	2.0041	1.0054	1.3048
CH1		0.1520	1.1887	10.579	-	0.3292	1.3366	0.8402	-
NS1	ρ	0.3084	0.2161	0.3995	-	1.7183	19.748	1.8436	-
	p	1.0015	0.0365	315.38	-	81.820	51.539	83.161	-
	v	0.9822	1.6727	6.5070	-	0.3113	4.6685	0.4049	-
Bu2	u	1.5809	0.0117	39.859	-	0.5690	0.2761	0.9274	-
	v	2.0041	0.0130	8.1979	-	0.6075	0.3321	0.8083	-
NS2	ρ	1.0761	0.0093	1.2898	-	12.898	4.5105	8.6476	-
	p	3.3239	0.0025	5593.9	-	39.903	1.9470	39.663	-
	v_x	13.646	1.4554	1.2977	-	0.0504	0.1202	0.0405	-
	v_y	1.0000	1.8458	1.0458	-	0.0623	0.1889	0.0808	-
SW2		0.0120	0.0114	0.0601	-	0.1551	0.7253	0.5985	-
AC2		0.4125	0.1363	588.58	1.0008	0.8379	2.1906	0.7089	0.8702
BS2		0.2365	0.0357	0.5423	0.6231	0.5807	0.2784	0.5913	0.1250

Table 16: Evaluating results in terms of L2RE and mERR for the 1D and 2D time-dependent problems.

PDE		L2RE			mERR		
		PINN	DFVM	RFM	PINN	DFVM	RFM
Ma3	E_x	0.9999	13.118	8.1405	0.1368	0.0490	0.0496
	E_y	0.9999	48.238	58.495	0.1992	0.0448	0.0448
	E_z	0.9999	11.846	51.741	0.1631	0.0453	0.0443
	H_x	0.9999	23.401	23.668	0.6530	0.1587	0.1587
	H_y	0.9999	30.076	30.938	0.6530	0.1703	0.1680
	H_z	0.9999	93.778	67.090	0.6206	0.1617	0.1626
Eu3	ρ	6.8376	0.1502	1.6598	9.6599	24.215	2.2353
	p	96.585	0.2131	7383.9	151.31	398.61	43.402
	v_x	0.3730	2.9805	5.5206	1.2784	7.6144	1.7630
	v_y	0.3683	3.1094	7.8708	1.2785	7.2665	1.7929
	v_z	0.2724	7.2088	13.748	1.3371	7.6575	1.9670
NS3	ρ	1.7696	0.1379	3.0895	2.7454	11.577	8.7557
	p	15.784	0.3583	8885.8	28.528	313.92	64.581
	v_x	0.3179	50.338	4.9910	1.8512	7.9745	1.4922
	v_y	0.3125	29.158	6.8152	1.7601	8.9036	1.6663
	v_z	0.2788	5.7752	16.803	1.5863	6.8745	1.6962

Table 17: Evaluating results in terms of the metrics of L2RE and mERR for 3D time-dependent equations.

In this subsection, we present the evaluation results of experiments that use function learning methods to solve time-dependent problems. Precisely, our experiments involve solving all 15 time-dependent PDE problems using PINN, DFVM, RFM, and solving 7 one-dimensional or two-dimensional PDE problems using WAN. Note that neither DRM nor DFLM has been applied to solve time-dependent problems.

Next, we present the results in terms of the three time metrics (t_{train}, t_{infer}, t_{conv}). Depicted in Table 18, 19, 20 are the results corresponding to the 1D, 2D and 3D problems, respectively. From these tables, we find that in terms of time metrics, DFVM excels for all 1D and 3D PDE problems. While for the 2D PDE problems, DFVM excels in terms of t_{train} and t_{infer} metrics, and PINN excels in terms of the t_{conv} metric.

Metrics	PDE	PINN	DFVM	RFM	WAN
t_{train}	Ad1	2.20e02	8.76e01	1.62e02	8.91e02
	DR1	4.18e02	8.91e01	2.94e02	9.10e02
	Bu1	4.03e02	5.22e02	2.99e02	1.06e03
	DS1	6.12e02	8.95e01	4.63e03	1.40e03
	AC1	4.13e02	8.83e01	2.97e02	1.12e03
	CH1	1.38e03	8.45e01	1.18e03	-
	NS1	2.48e03	8.46e01	3.01e03	-
t_{infer}	Ad1	2.14e-04	3.49e-04	1.22e-04	5.02e-04
	DR1	5.06e-04	3.48e-04	1.22e-04	6.37e-04
	Bu1	2.27e-04	3.96e-04	1.17e-04	5.07e-04
	DS1	4.77e-04	4.31e-04	1.85e-04	4.81e-04
	AC1	5.15e-04	2.75e-04	1.17e-04	5.94e-04
	CH1	5.19e-04	3.00e-04	1.15e-04	-
	NS1	5.29e-04	3.40e-04	4.85e-05	-
t_{conv}	Ad1	2.20e02	4.86e01	1.62e02	2.89e02
	DR1	4.18e02	8.90e01	2.94e02	9.09e02
	Bu1	4.03e02	5.29e01	2.99e02	9.66e02
	DS1	6.10e02	3.11e00	4.63e03	1.39e03
	AC1	1.36e00	8.28e01	2.97e02	1.12e03
	CH1	6.51e00	8.35e01	1.18e03	-
	NS1	8.82e00	2.19e01	3.01e03	-

Table 18: Results in terms of time metrics for 1D time-dependent equations.

Metrics	PDE	PINN	DFVM	RFM	WAN
t_{train}	Bu2	4.08e03	3.68e02	7.13e03	-
	NS2	1.37e04	3.18e02	1.27e04	-
	SW2	4.43e03	1.78e02	1.16e04	-
	AC2	2.22e03	2.94e02	2.72e03	1.25e03
	BS2	1.24e04	3.00e03	1.27e04	8.24e02
t_{infer}	Bu2	2.97e-02	7.92e-04	4.63e-05	-
	NS2	9.99e-04	7.43e-04	3.67e-05	-
	SW2	3.02e-02	7.61e-04	4.66e-05	-
	AC2	2.99e-02	7.29e-04	1.02e-04	5.46e-04
	BS2	2.84e-02	9.01e-04	1.00e-04	6.51e-04
t_{conv}	Bu2	3.42e00	4.67e01	2.94e01	-
	NS2	7.41e01	3.11e02	4.29e03	-
	SW2	4.43e03	6.63e00	9.27e03	-
	AC2	1.88e01	2.37e02	9.14e00	1.25e03
	BS2	9.89e01	2.38e02	3.77e01	5.90e02

Table 19: Results in terms of time metrics for 2D time-dependent equations.

Metrics	PDE	PINN	DFVM	RFM
t_{train}	NS3	1.36e04	7.48e02	1.16e03
	Ma3	2.89e01	7.81e02	1.19e04
	Eu3	1.24e05	9.16e02	1.20e04
t_{infer}	NS3	3.61e-02	1.94e-03	4.19e-05
	Ma3	3.88e-03	8.24e-04	4.19e-05
	Eu3	3.63e-02	1.93e-03	4.23e-05
t_{conv}	NS3	7.62e03	6.00e02	4.19e00
	Ma3	2.89e01	3.96e00	1.13e04
	Eu3	7.62e02	2.30e02	1.20e04

Table 20: Results in terms of time metrics for 3D time-dependent equations.

Finally, we present the results in terms of the metric Lips. Listed in Table 21 are the results for all 15 time-dependent equations. Note that since the network architectures for six function learning methods are all MLP, we can directly apply the formula (25) to calculate the Lipschitz constant. From Table 21, we find that the WAN excels for the scalar problems, and the RFM excels for the multivariable problems.

For Ad1, the GT solution's Lipschitz constant is 6.3145, while for BS2, it is 1.7638. GT solution's Lipschitz constants are not provided for the remaining equations where obtaining true solutions is infeasible.

	PINN	DFVM	RFM	WAN
Ad1	1.38e02	5.21e00	2.79e01	7.00e00
DR1	6.42e02	4.15e02	6.46e01	6.50e00
Bu1	2.23e02	6.81e-01	2.59e01	1.93e01
DS1	4.95e01	1.18e04	2.58e01	2.63e01
AC1	1.96e01	2.13e02	1.10e01	2.09e01
CH1	2.15e03	6.38e01	2.39e01	-
NS1	3.29e01	3.84e03	1.15e01	-
Bu2	1.21e03	1.22e04	1.65e01	-
NS2	5.82e00	1.17e03	7.23e00	-
SW2	4.77e02	2.12e00	2.51e01	-
AC2	4.52e00	2.79e02	1.06e01	4.96e00
BS2	4.25e00	4.00e01	1.07e01	4.84e00
Ma3	4.16e00	5.83e00	4.71e00	-
Eu3	7.58e00	1.27e04	4.71e00	-
NS3	8.07e00	1.54e03	5.06e00	-

Table 21: Results in terms of the metric Lips for all time-dependent PDEs.

11.2 Performance of Operator Learning Methods

In this subsection, we present the evaluation results of operator learning methods for solving 16 PDEs which accompany with datasets.

PDE		DeepONet	PINO	U-NO	FNO	U-Net	MPNN
Ad1		0.0792	0.0191	0.0069	0.0128	0.0699	0.0451
DR1		0.0289	0.0163	0.0036	0.0098	0.0351	0.0039
Bu1		0.3595	0.1206	0.0601	0.0603	0.3841	0.1143
DS1		0.0413	0.0078	0.0012	0.0024	0.1250	0.0029
AC1		0.2943	0.0062	0.0019	0.0041	0.0718	0.0033
CH1		0.6697	0.0259	0.0041	0.0053	0.1759	0.0554
NS1	ρ	0.3113	0.0728	0.1860	0.0981	0.6762	0.3300
	p	0.0792	0.0101	0.0271	0.0111	0.1888	0.7398
	v	1.0143	0.2806	0.9675	0.8769	1.9768	4.6513
Bu2	u	0.6575	0.0298	0.0089	0.2201	0.2938	0.0115
	v	0.6886	0.0328	0.0093	0.2425	0.2878	0.0116
NS2	ρ	1.7336	0.0593	0.0075	0.0054	0.0866	0.0008
	p	3.7537	0.0205	0.0023	0.0019	0.1075	0.0002
	v_x	1.0007	14.0426	1.5322	1.8911	39.3686	0.1407
	v_y	1.0006	1.4571	1.5677	1.1679	18.3559	0.1484
DF2		0.3971	0.0702	0.0742	0.1328	0.0828	-
SW2		0.0953	0.0166	0.0025	0.0036	0.1014	0.0013
AC2		0.9999	0.0191	0.0054	0.0084	0.1117	0.0075
BS2		0.2654	0.0447	0.0021	0.0072	0.2819	0.0015

Table 22: L2RE for the 1D and 2D problems using operator learning methods.

We commence by presenting the evaluation results in terms of two accuracy metrics: L2RE and mERR. The L2RE results for 1D and 2D problems are enumerated in Table 22, while Table 23 displays the mERR outcomes for 1D and 2D problems. Additionally, Table 24 features the results for 3D problems. When solving the 3D Maxwell equations with FNO and UNO methods, we input normalized data due to small values in the actual solution and coordinates, omit grid coordinates.

PDE		DeepONet	PINO	U-NO	FNO	U-Net	MPNN
Ad1		1.9227	0.5008	0.2493	0.3892	1.8000	1.2668
DR1		0.8150	0.1452	0.1316	0.0618	0.2678	0.0192
Bu1		2.0055	2.0109	1.1365	1.2186	1.6663	1.4130
DS1		0.4485	0.1679	0.0271	0.0313	0.2193	0.0160
AC1		1.4740	0.2853	0.1951	0.2095	1.6084	0.3832
CH1		1.1861	0.3048	0.0658	0.0469	0.9016	0.6298
NS1	ρ	20.291	15.542	20.504	13.488	25.646	18.953
	p	89.890	29.723	39.381	32.909	103.72	1348.4
	v	3.4619	3.5683	4.2427	4.7510	3.7099	67.019
Bu2	u	0.7204	0.3887	0.2077	0.2201	0.7046	0.2751
	v	0.8127	0.3911	0.2181	0.2425	0.6563	0.1430
NS2	ρ	19.488	8.9382	3.0041	2.6072	10.667	0.3743
	p	122.42	73.587	1.1069	0.2296	17.640	0.0660
	v_x	1.2539	2.0798	0.1634	0.0718	1.1109	0.0054
	v_y	1.3363	1.3288	0.1224	0.0469	0.4472	0.0063
DF2		0.4453	0.2732	0.2472	0.2920	0.2748	-
SW2		0.8478	0.4520	0.1097	0.1770	1.3220	0.0447
AC2		0.9588	0.1517	0.0616	0.0611	0.4367	0.0708
BS2		0.4330	0.0697	0.0170	0.0272	0.7897	0.0060

Table 23: mERR for the 1D and 2D problems using operator learning methods.

From Table 22-24, we observe that FNO demonstrates exceptional performance in the L2RE metric for 1D problems, while U-NO excels in the L2RE metric for both 2D and 3D problems. Furthermore, U-NO exhibits commendable performance in the mERR metric for 1D and 3D problems, whereas MPNN leads in the mERR metric for 2D problems.

PDE		L2RE			mERR		
		U-NO	FNO	U-Net	U-NO	FNO	U-Net
Ma3	E_x	0.7038	0.9999	0.5927	0.4508	0.5906	0.0605
	E_y	0.7604	1.0000	0.5934	0.6280	0.6680	0.1301
	E_z	0.7129	1.0000	0.5932	0.9988	1.0225	0.1580
	H_x	0.7549	1.0000	0.5472	0.7480	0.7825	0.4689
	H_y	0.7016	1.0000	0.5550	0.5406	0.6139	0.2744
	H_z	0.7592	1.0000	0.5519	0.3130	0.4411	0.1778
Eu3	ρ	0.1998	0.2767	3.3382	6.6386	8.2500	10.858
	p	0.1639	0.1486	4.5508	130.91	121.53	62.039
	v_x	1.0211	1.0459	2.2938	2.8235	3.0606	2.8924
	v_y	1.2094	1.0327	2.6815	2.8358	2.7447	2.7037
	v_z	1.0176	1.0887	2.3393	2.6471	3.2404	2.9951
NS3	ρ	0.2292	0.2141	2.2368	8.4076	8.2553	14.147
	p	0.1567	0.1500	7.3059	136.49	126.58	72.281
	v_x	1.0665	1.0436	2.7942	2.7959	2.7876	2.5613
	v_y	1.0932	1.0217	2.8227	2.8880	2.6573	2.6263
	v_z	1.2956	1.0204	2.7901	3.0015	2.8553	2.6200

Table 24: L2RE and mERR for the 3D problems.

Next, we employ Table 25,26,27 to present time metrics for 1D, 2D, 3D problems, respectively.

Metrics	PDE	DeepONet	PINO	U-NO	FNO	U-Net	MPNN
t_{train}	Ad1	6.65e02	8.97e03	2.87e04	1.13e04	2.41e04	2.57e04
	DR1	7.95e02	2.51e04	6.50e04	3.05e04	4.16e04	1.82e04
	Bu1	6.80e02	2.56e03	2.53e04	2.40e04	2.36e04	2.59e04
	DS1	2.34e03	8.98e03	8.30e04	3.28e04	4.25e04	2.48e04
	AC1	8.05e02	2.52e04	7.50e04	3.88e04	4.21e04	1.90e04
	CH1	8.25e02	2.52e04	7.55e04	3.90e04	4.24e04	1.86e04
	NS1	1.43e03	4.83e04	7.65e04	3.95e04	4.23e04	4.52e04
t_{infer}	Ad1	8.01e-04	2.14e-03	2.90e-01	2.48e-01	7.69e-02	1.83e-02
	DR1	1.34e-03	2.30e-03	4.79e-01	3.37e-01	2.22e-01	5.42e-02
	Bu1	8.33e-04	2.07e-03	2.88e-01	2.44e-01	7.55e-02	1.86e-02
	DS1	2.08e-03	4.40e-03	4.76e-01	3.44e-01	2.16e-01	5.54e-02
	AC1	6.78e-04	2.06e-03	3.96e-01	2.84e-01	2.18e-01	5.41e-02
	CH1	6.76e-04	2.03e-03	4.70e-01	4.05e-01	2.20e-01	7.14e-02
	NS1	1.56e-03	2.73e-03	4.76e-01	3.72e-01	2.19e-01	4.08e-01
t_{conv}	Ad1	5.75e02	7.18e03	2.71e04	8.71e03	2.28e04	2.56e04
	DR1	3.50e01	2.49e04	6.41e04	2.45e04	3.92e04	1.57e04
	Bu1	1.36e02	5.41e03	2.46e04	1.97e04	1.43e04	2.24e04
	DS1	1.87e03	2.57e04	6.67e04	2.64e04	2.12e04	1.99e04
	AC1	1.61e02	2.01e04	6.03e04	3.88e04	2.43e04	8.46e03
	CH1	3.30e02	2.01e04	6.07e04	3.74e03	1.19e04	1.82e04
	NS1	1.43e03	2.91e04	4.65e04	3.92e04	6.78e03	4.41e04

Table 25: Time metrics for 1D equations.

Metrics	PDE	DeepONet	PINO	U-NO	FNO	U-Net	MPNN
t_{train}	Bu2	4.15e03	1.15e05	2.27e05	8.65e04	6.87e04	5.88e04
	NS2	8.76e03	2.35e05	1.42e05	2.30e04	4.85e04	2.44e05
	DF2	2.99e03	1.10e03	1.37e04	8.40e03	1.33e04	-
	SW2	2.55e03	1.15e05	2.08e05	8.10e04	6.71e04	3.06e04
	AC2	2.13e03	1.13e05	2.04e05	9.00e04	6.73e04	3.09e04
	BS2	2.12e03	1.14e05	2.35e05	1.08e05	6.72e04	3.14e04
t_{infer}	Bu2	7.61e-02	1.83e-02	5.60e-01	4.38e-01	3.26e-01	7.41e-01
	NS2	2.08e-02	7.96e-03	2.69e-01	2.33e-01	4.19e-02	1.70e-02
	DF2	1.30e-03	2.13e-03	2.30e-01	2.13e-01	3.60e-03	-
	SW2	6.26e-02	1.71e-02	5.85e-01	4.45e-01	3.41e-01	3.77e-01
	AC2	6.13e-02	1.70e-02	4.26e-01	3.62e-01	3.40e-01	3.78e-01
	BS2	6.36e-02	1.70e-02	4.31e-01	6.04e-01	3.61e-01	3.79e-01
t_{conv}	Bu2	4.08e03	1.12e05	5.68e04	8.62e04	6.83e04	5.13e04
	NS2	-	1.80e05	1.38e05	2.30e04	3.24e04	2.42e05
	DF2	8.80e02	8.24e03	1.80e04	5.09e03	1.22e04	-
	SW2	1.65e02	4.71e04	1.26e05	7.47e04	3.81e04	3.02e04
	AC2	3.19e01	8.49e04	4.20e04	1.85e04	2.92e04	3.02e04
	BS2	2.20e03	1.05e05	6.43e04	9.48e04	3.98e04	2.84e04

Table 26: Time metrics for 2D equations

Metrics	PDE	U-NO	FNO	U-Net
t_{train}	NS3	4.77e04	1.96e04	4.57e04
	Ma3	4.64e04	2.23e04	5.22e04
	Eu3	2.89e04	4.56e04	9.17e04
t_{infer}	NS3	4.46e-01	2.43e-01	1.26e-01
	Ma3	2.95e-02	2.25e-02	4.32e-02
	Eu3	1.80e-01	2.63e-01	1.26e-01
t_{conv}	NS3	4.74e04	1.33e04	3.99e04
	Ma3	1.01e04	8.91e01	1.01e05
	Eu3	2.40e04	2.55e04	6.98e04

Table 27: Time metrics for 3D equations.

From Table 25-27, we observe that DeepONet excels in the evaluation for 1D and 2D PDE problems. Furthermore, we observe that FNO excels in the evaluation metrics t_{train} and t_{conv} for 3D problems. Additionally, U-Net outperforms in the t_{infer} metric for 3D problems.

Finally, we present the Lips in Table 28 for the U-Net and DeepONet methods. It seems that the MPNN encompasses a broader array of network structures, contributing to a significantly larger Lipschitz constant compared to that of DeepONet.

PDE	DeepONet	U-Net	PDE	DeepONet	U-Net
Ad1	8.71e02	9.13e11	NS2	1.28e01	1.17e15
DR1	2.16e-02	2.70e08	DF2	1.18e01	4.05e04
Bu1	5.27e01	6.13e12	SW2	3.69e00	3.28e08
DS1	8.49e00	1.31e05	AC2	1.41e00	3.37e05
AC1	1.12e03	4.39e03	BS2	3.32e00	2.97e07
CH1	4.51e02	2.14e09	Ma3	-	1.22e04
NS1	1.68e02	5.59e14	Eu3	-	4.22e10
Bu2	8.75e00	4.87e04	NS3	-	3.52e13

Table 28: Lipschitz constants for the U-Net and DeepONet.

References

[Bartlett *et al.*, 2017] Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. *Advances in neural information processing systems*, 30, 2017.

[Brandstetter *et al.*, 2022b] Johannes Brandstetter, Rianne van den Berg, Max Welling, and Jayesh K Gupta. Clifford neural layers for PDE modeling. *arXiv preprint arXiv:2209.04934*, 2022b.

[Fel *et al.*, 2022] Thomas Fel, David Vigouroux, Rémi Cadène, and Thomas Serre. How good is your explanation? algorithmic stability measures to assess the quality of explanations for deep neural networks. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 720–730, 2022.

[Huang *et al.*, 2021] Yujia Huang, Huan Zhang, Yuanyuan Shi, J Zico Kolter, and Anima Anandkumar. Training certifiably robust neural networks with efficient local lipschitz bounds. *Advances in Neural Information Processing Systems*, 34:22745–22757, 2021.

[Limousin *et al.*, 2007] Gaudet Limousin, Jean-Paul Gaudet, Laurent Charlet, Stéphanie Szenknect, Véronique Barthès, and Mohamed Krimissa. Sorption isotherms: A review on physical bases, modeling and measurement. *Applied geochemistry*, 22(2):249–275, 2007.

[Szegedy *et al.*, 2013] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[Takamoto *et al.*, 2023] Makoto Takamoto, Francesco Alesiani, and Mathias Niepert. Learning neural pde solvers with parameter-guided channel attention. *arXiv preprint arXiv:2304.14118*, 2023.

[Tsuzuku *et al.*, 2018] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in neural information processing systems*, 31, 2018.

[Virmaux and Scaman, 2018] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. *Advances in Neural Information Processing Systems*, 31, 2018.