PDEBench: An Extensive Benchmark for Scientific Machine Learning

Makoto Takamoto[*]
NEC Labs Europe

Timothy Praditia[†]
University of Stuttgart

Raphael Leiteritz
University of Stuttgart

Dan MacKinlay
CSIRO's Data61

Francesco Alesiani
NEC Labs Europe

Dirk Pflüger
University of Stuttgart

Mathias Niepert
University of Stuttgart

Abstract

Machine learning-based modeling of physical systems has experienced increased interest in recent years. Despite some impressive progress, there is still a lack of benchmarks for Scientific ML that are easy to use but still challenging and representative of a wide range of problems. We introduce PDEBench, a benchmark suite of time-dependent simulation tasks based on Partial Differential Equations (PDEs). PDEBench comprises both code and data to benchmark the performance of novel machine learning models against both classical numerical simulations and machine learning baselines. Our proposed set of benchmark problems contribute the following unique features: (1) A much wider range of PDEs compared to existing benchmarks, ranging from relatively common examples to more realistic and difficult problems; (2) much larger ready-to-use datasets compared to prior work, comprising multiple simulation runs across a larger number of initial and boundary conditions and PDE parameters; (3) more extensible source codes with user-friendly APIs for data generation and baseline results with popular machine learning models (FNO, U-Net, PINN, Gradient-Based Inverse Method). PDEBench allows researchers to extend the benchmark freely for their own purposes using a standardized API and to compare the performance of new models to existing baseline methods. We also propose new evaluation metrics with the aim to provide a more holistic understanding of learning methods in the context of Scientific ML. With those metrics we identify tasks which are challenging for recent ML methods and propose these tasks as future challenges for the community. The code is available at `https://github.com/pdebench/PDEBench`.

1 Motivation

In the emergent area of *Scientific Machine Learning* (or *machine learning for physical sciences* or *data-driven science*), recent progress has broadened the scope of traditional machine learning (ML) methods to include the time-evolution of physical systems. Within this field, rapid progress has been made in the use of neural networks to make predictions using functional observations over continuous domains [8, 46] or with challenging constraints and with physically-motivated conservation laws [34, 61, 47]. These neural networks provide an approach to solving PDEs complementing traditional

[*]E-mail:`Makoto.Takamoto@neclab.eu` [†]E-mail:`timothy.praditia@iws.uni-stuttgart.de`

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks.

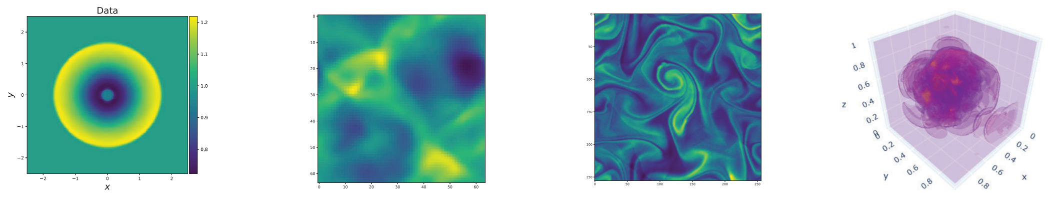


Figure 1: PDEBENCH provides multiple non-trivial challenges from the Sciences to benchmark current and future ML methods, including wave propagation and turbulent flow in 2D and 3D

numerical solvers. For instance, data-driven ML methods are useful when observations are noisy or the underlying physical model is not fully known or defined [11]. Moreover, neural models have the advantage of being continuously differentiable in their inputs, a useful property in several applications. In physical system design [1], for instance, the models are themselves physical objects and thus not analytically differentiable. Similarly, in many fields such as hydrology [14], benchmark physical simulation models exist but the forward simulation models non-differentiable black boxes. This complicates optimisation, control, sensitivity analysis, and solving inverse inference problems. While complex methods such as Bayesian optimisation [38, 50, 42] or reduced order modelling [16] are in part an attempt to circumvent this lack of differentiability, gradients for neural networks are readily available and efficient.

For classical ML applications such as image classification, time series prediction or text mining, various popular benchmarks exist, and evaluations using these benchmarks provides a standardised means of testing the effectiveness and efficiency of ML models. As yet, a widely accessible, practically simple, and statistically challenging benchmark with ready-to-use datasets to compare methods in Scientific ML is missing. While some progress towards reference benchmarks has been made in recent years (see section 2), we aim to provide a benchmark that is more comprehensive with respect to the PDEs covered and which enables more diverse methods for evaluating the efficiency and accuracy of the ML method. The problems span a range of governing equations as well as different assumptions and conditions; see Figure 1 for a visual teaser. Data may be generated by executing code through a common interface, or by downloading high-fidelity datasets of simulations. All code is released under a permissive open source license, facilitating re-use and extension. We also propose an API to ease the implementation and evaluation of new methods, provide recent competitive baseline methods such as FNOs and autoregressive models based on the U-Net, and a set of pre-computed performance metrics for these algorithms. We may thus compare their predictions against the "ground truth" provided by baseline simulators used to generate the data.

As in other machine learning application domains, benchmarks in Scientific ML may serve as a source of readily-available training data for algorithm development and testing without the overhead of generating data *de novo*. In these emulation tasks, the training/test data is notionally unlimited, since more data may always be generated by running a simulator. However, in practice, producing such datasets can have an extremely high burden in compute time, storage and in access to the specialised skills needed to produce them. PDEBENCH also addresses the need for quick, off-the-shelf training data, bypassing these barriers while providing an easy on-ramp to be extended.

In this work, we propose a versatile benchmark suite for Scientific ML (a) providing diverse data sets with distinct properties based on 11 well-known time-dependent and time-independent PDEs, (b) covering both "classical" forward learning problems and inverse settings, (c) all accessible via a uniform interface to read/store data across several applications, (d) extensible, (e) with results for popular state-of-the-art ML models (FNO, U-Net, PINN) for (f) a set of metrics that are better-suited for Scientific ML, (g) with both data to download and code to generate more data, and (h) pre-trained models to compare against. The inverse problem scenarios comprise initial and boundary conditions and PDE parameters (e.g. viscosity). Each data set has a sufficiently large number of samples for training and test, for a variety of parameter values, with a resolution high enough to capture local dynamics. As an additional note, our goal is not to provide a complete benchmark that includes all possible combinations of inference tasks on all known experiments, but rather to ease the task for subsequent researchers to benchmark their favoured methods. Part of our goal here is to invite other researchers to fill in the gaps for themselves by leveraging our ready-to-run models.

To evaluate ML methods for scientific problems, we consider several metrics that go beyond the standard RMSE and include properties of the underlying physics. The initial experimental results we obtained using PDEBENCH confirm the need for comprehensive Scientific ML benchmarks: There is no one-size-fits-all model and there is plenty of room for new ML developments. The results show that the standard error measure in ML, the RMSE on test data, is not a good proxy for evaluating ML models, in particular in turbulent and non-smooth regimes where it fails to capture small spatial scale changes. We furthermore cover an application where a parameter of the underlying PDE heavily influences the difficulty of the problem for ML baselines. We also observe unexpected experimental results for the generalization behavior of auto-regressive training, which seems to be more challenging in the scientific realm. In the remainder of the paper, we first address related work, introduce PDEBENCH and the underlying design choices, and discuss results for a few selected experiments.

2 Related Work

PDE benchmarking has particular challenges. Unlike many classic datasets, PDE datasets can be large on a gigabyte or terabyte scale and still contain only few data points. And unlike monolithic benchmark datasets such as ImageNet, the datasets for each PDE approximation task are specific to that task. Each set of governing equations or experiment design assumptions leads to a distinct dataset of PDE samples. Recent works in PDEs have attempted to produce standardised datasets covering well-known challenges [44, 19, 51]. [19] targets non-ML uses. [51] is specialised for particular classes of equations. Of these, the excellent work of [44] is most closely related, but with only four physical systems, it still lacks sufficient scale and diversity of data to challenge emerging ML algorithms. We expand the range of benchmarks in this domain by providing a larger, more diverse problem selection and scale than these previous attempts (11 PDEs with different parametrizations leading to 35 datasets). We additionally consider inverse problems for PDEs [53, 56], with the goal to identify unobserved latent parameters using ML. This has not been covered by benchmarks so far, despite its increasing importance in the community. Furthermore, most work in this scope considers classical statistical error measures such as the RMSE over the whole domain and at most PDE-motivated variants such as the RMSE of the gradient [44]. Measures based on properties of the underlying physical systems, as studied in this work, are lacking.

An overview and taxonomy of Scientific ML developments can be found in [28, 6]. For developing our baselines, we focus on using neural network models to approximate the outputs of some *ground truth* PDE solver, given data generated by that solver, which itself aims to directly implement the numerical solution of a given partial differential equation. A range of methods aim to solve problems fitting this description, reviewed in [22]. Methods include Physics-informed neural networks (PINNs) [47], Neural operators (NOs) [32, 26], treating ResNet as a PDE approximant [49], custom architectures for specific problems such as TFNet for turbulent fluid flows [61], and generic image-to-image regression models such as the U-Net [48]. These approaches each have different assumptions, domains of applicability, and data processing requirements. For a more comprehensive discussion of prior work in Scientific ML we refer the reader to Appendix A.

3 PDEBENCH: A Benchmark for Scientific Machine Learning

In the following we describe the general learning problem addressed with the benchmark, the currently covered PDEs, existing implemented baselines (all developed using PyTorch [45], and PINN specifically using DeepXDE [34]), and the ways in which the benchmark follows FAIR data principles [63].

3.1 General Problem Definition

A solution to a PDE is a vector-valued function $\mathbf{v} : \mathcal{T} \times \mathcal{S} \times \Theta \to \mathbb{R}^d$ on some spatial domain $\mathcal{S}$, with temporal index $\mathcal{T}$, and some possibly function-valued parameter-space Θ. For example in a heat diffusion equation, $\mathbf{v}$ might represent the local temperature $\tau \in \mathbb{R}^1$ of some substrate at some given point $\mathbf{s} \in \mathcal{S}$, at a given moment $t \in \mathcal{T}$, and conditional upon a spatially-varying scalar conductivity field representing an inhomogeneous substrate $\theta : \mathcal{S} \to \mathbb{R}^+$. The operator mapping from the state of

Table 1: Summary of PDEBENCH's datasets with their respective number of spatial dimensions N_d, time dependency, spatial resolution N_s, temporal resolution N_t, and number of samples generated.

PDE	N_d	Time	N_s	N_t	Number of samples
advection	1	yes	1 024	200	10 000
Burgers'	1	yes	1 024	200	10 000
diffusion-reaction	1	yes	1 024	200	10 000
diffusion-reaction	2	yes	128×128	100	1000
diffusion-sorption	1	yes	1 024	100	10 000
compressible Navier-Stokes	1	yes	1 024	100	10 000
compressible Navier-Stokes	2	yes	512×512	21	1000
compressible Navier-Stokes	3	yes	$128 \times 128 \times 128$	21	100
incompressible Navier-Stokes	2	yes	256×256	1000	1000
Darcy flow	2	no	128×128	–	10 000
shallow-water	2	yes	128×128	100	1000

the solution at one timestep to the solution one time step later, $\mathfrak{F}_\theta : \mathbf{v}_\theta(t, \cdot) \to \mathbf{v}_\theta(t+1, \cdot)$ is referred to as the *forward propagator*.

The objective of Scientific ML is to find some ML-based surrogate, sometimes referred to as an emulator, of this forward propagator by learning an approximation $\widehat{\mathfrak{F}}_\theta \simeq \mathfrak{F}_\theta$. The forward propagator of a PDE is dependent not only on the current state, but also upon both spatial and temporal derivatives of the state field. In practice, temporal derivatives of solutions are often not conveniently encoded by system states at one single time step. Hence, the forward propagator may also depend on multiple previous timesteps of the solution, enabling finite-difference approximations of the temporal derivatives. The discretised forward propagator $\mathring{\mathfrak{F}}_\theta$ then operates on $\ell \geq 1$ consecutive timesteps so that $\mathring{\mathfrak{F}}_\theta : \mathbf{v}_\theta(t-\ell, \cdot), \ldots, \mathbf{v}_\theta(t-1, \cdot) \mapsto \mathbf{v}_\theta(t, \cdot)$, which is abbreviated as $\mathbf{v}_\theta([t{-}\ell{:}t{-}1], \cdot) := \mathbf{v}_\theta(t-\ell, \cdot), \ldots, \mathbf{v}_\theta(t-1, \cdot)$.

We seek to approximate this discretised operator with an emulator $\widehat{\mathfrak{F}}_\theta \simeq \mathring{\mathfrak{F}}_\theta$ in the sense that predictions the emulator makes should be close to the ground truth simulation given the same inputs, with respect to some measure of cost. We fix a parametric class of models $\{\mathfrak{F}_{\theta,\phi}\}_\phi$. From this class we *learn* a surrogate $\widehat{\mathfrak{F}}_{\theta,\phi}$ from data. In learning, we take a dataset $\mathcal{D}$ comprising discretized PDE solutions conditional on selected parameter values (θ_k), $\mathcal{D} := \{\mathbf{v}^{(k)}_{\theta_k}([0{:}t_{\max}], \cdot) \mid k = 1, \ldots, K\}$. Fixing a loss functional L, we aim to find some ϕ achieving a minimal total loss on the training dataset

$$\widehat{\phi} = \operatorname{argmin}_\phi \sum_{t=1}^{t_{\max}} \sum_{k=1}^{K} L\left(\mathfrak{F}_{\theta_k,\phi}\{\mathbf{v}^{(k)}_{\theta_k}([t{-}\ell{:}t{-}1], \cdot)\}, \mathbf{v}^{(k)}_{\theta_k}(t, \cdot)\right). \tag{1}$$

Due to the use of iterative optimization algorithms such as stochastic gradient descent and the non-convex nature of the above optimization problem, we typically obtain local optima. $\mathcal{D}$ is generated by a ground-truth solver designed to simulate the desired dynamics with high precision. In this data we may vary initial conditions, that is, varying $\mathbf{v}_\theta(0, \cdot)$, varying θ, or both.

In addition to the forward problem, we also consider the use of learned surrogate models to approximately solve *inverse problems* [53, 56], where an unknown initial condition $\mathbf{v}_\theta(0, \cdot)$ or unknown parameter θ is chosen to be congruent with some observed outputs $\mathbf{v}_\theta([t{:}t+\ell], \cdot)$. We follow [32, 35] in using an approximate surrogate approach, taking the forward surrogates as mean predictors for the model. We assume $\mathbf{v}_\theta(t, \cdot) = \mathfrak{F}_{\theta,\phi}\{\mathbf{v}_\theta([t{-}\ell{:}t{-}1], \cdot)\} + \epsilon$ for some mean-zero observation noise ϵ, and assuming a prior distribution for the unknown of interest. Other inversion methods can be used in this domain, such as generative adversarial models, [9] or variational autoencoders [54].

3.2 Overview of Datasets and PDEs

The current version of the benchmark provides datasets generated for various PDEs ranging from 1 to 3 dimensional spatial domains. There are both time-dependent and time-independent PDEs. The current datasets are summarized in Table 1; see Figure 1 for a visual teaser. Each sample is generated with different parameters, initial conditions, and boundary conditions. Generalization to different parameters, varying initial conditions, and proper treatment of complex boundary conditions are still open challenges in Scientific ML [21, 4, 2, 29]. The parameters which are varied to provide several datasets include the advection speed in the advection equation, the forcing term in the Darcy flow, as well as the viscosity in the Burgers' and compressible Navier-Stokes equations all of which can lead to significantly different behaviors of the simulated systems. Additionally, besides the periodic boundary condition that is most commonly used in Scientific ML studies, we also provide datasets generated with the Neumann boundary condition in the 2D diffusion-reaction and shallow-water equations, the Cauchy boundary condition in the diffusion-sorption equation, and the Dirichlet condition in the incompressible Navier-Stokes equation.

We designed this benchmark to represent a diverse set of challenges for emulation algorithms. In particular, we focus on hydromechanical field equations. Following this philosophy, we selected 6 basic and 3 advanced real-world problems. The basic PDEs are stylized, simple models: 1D advection/Burgers/Diffusion-Reaction/Diffusion-Sorption equations, 2D Diffusion-Reaction equation, and 2D DarcyFlow; the advanced and real-world PDEs incorporate features of real-world modeling tasks: Compressible and incompressible Navier-Stokes equations, and shallow-water equations. The PDEs exhibit a variety of behaviors of real-world significance which are known to challenge emulators, such as sharp shock formation dynamics, sensitive dependence on initial conditions, diverse boundary conditions, and spatial heterogeneity. Finding a surrogate model which can approximate these challenging dynamics with high fidelity we argue is a necessary precondition to applying such models in the real world. While some of these have been used in prior work, a publicly available benchmark dataset is, to the best of our knowledge, not available.

In the following we provide a brief introduction and important features of the advanced PDEs. More detailed explanations for all the PDEs are provided in Appendix D.

Compressible Navier-Stokes equations The compressible fluid dynamics equations describe a fluid flow whose expression is given as:

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0, \quad \rho(\partial_t \mathbf{v} + \mathbf{v} \cdot \nabla \mathbf{v}) = -\nabla p + \eta \triangle \mathbf{v} + (\zeta + \eta/3) \nabla (\nabla \cdot \mathbf{v}), \tag{2a}$$

$$\partial_t (\epsilon + \rho v^2/2) + \nabla \cdot [(p + \epsilon + \rho v^2/2)\mathbf{v} - \mathbf{v} \cdot \sigma'] = \mathbf{0}, \tag{2b}$$

where ρ is the mass density, $\mathbf{v}$ is the fluid velocity, p is the gas pressure, ϵ is an internal energy described by the equation of state, σ' is the viscous stress tensor, and η and ζ are shear and bulk viscosity, respectively. This equation can describe more complex phenomena, such as shock wave formation and propagation. It is applied to many real-world problems, such as the aerodynamics around airplane wings and interstellar gas dynamics.

Incompressible Navier-Stokes equations The Navier-Stokes equation is the incompressible version of the compressible fluid dynamics equation, applicable to sub-sonic flows. This equation can model a variety of systems, from hydromechanical systems to weather forecasting or investigating turbulent dynamics.

Shallow-Water Equations The shallow-water equations, derived from the compressible Navier-Stokes equations, present a suitable framework for modeling free-surface flow problems. In 2D, these come in the form of the following system of hyperbolic PDEs,

$$\partial_t h + \nabla h \mathbf{u} = 0, \quad \partial_t h \mathbf{u} + \nabla \left(\mathbf{u}^2 h + \frac{1}{2} g_r h^2 \right) = -g_r h \nabla b \,, \tag{3a}$$

where $\mathbf{u} = u, v$ being the velocities in the horizontal and vertical direction, h describing the water depth, and b describing a spatially varying bathymetry. $h\mathbf{u}$ can be interpreted as the directional momentum components and g_r describes the gravitational acceleration. The mass and momentum conservation properties even hold across shocks in the solution and thus challenging datasets can be generated. Example applications include the simulation of tsunamis or general flooding events.

3.3 Overview of Metrics

The standard approach of computing the RMSE on test data falls short of capturing important optimization criteria in Scientific ML. A good fit to (often sparse) data is not sufficient if the physics of the underlying problem is severely violated. Physics-informed learning that aims to conserve

```python
from pyDaRUS import Dataset
p_id = "doi:10.18419/darus-2986"
dataset = Dataset.from_dataverse_doi(p_id, filedir="data/")
```

Listing 1: Including a benchmark dataset.

physical quantities must therefore be evaluated with appropriate metrics. A global, averaged metric for instance cannot capture small spatial scale changes critical in turbulent regimes. Moreover, a single evaluation metric is not sufficient to compare different methods with respect to their ability to extrapolate to unseen time steps and parameters which are important but underexplored evaluation criteria for ML surrogates. Hence, the proposed benchmark includes several novel metrics which we believe provide a deeper and more holistic understanding of the surrogate's behavior and which are designed to reflect both the data and physics perspective. The following table summarizes the metrics used; further details can be found in the Appendix B.

Scope	Acronym	Metric
Data view	RMSE	root-mean-squared-error
	nRMSE	normalized RMSE (ensuring scale independence)
	max error	maximum error (local worst case; also proxy for stability of time-stepping)
Physics view	cRMSE	RMSE of conserved value (deviation from conserved physical quantity)
	bRMSE	RMSE on boundary (whether boundary condition can be learned)
	fRMSE low	RMSE in Fourier space, low frequency regime (wavelength dependence)
	fRMSE mid	RMSE in Fourier space, medium frequency regime
	fRMSE high	RMSE in Fourier space, high frequency regime

3.4 Existing Baseline Surrogate Models

U-Net U-Net [48] is an auto-encoding neural network architecture used for processing images using multi-resolution convolutional networks with skip layers. U-Net is a black-box machine learning model that propagates information efficiently at different scales. Here, we extended the original implementation, which uses 2D-CNN, to the spatial dimension of the PDEs (i.e. 1D,3D).

Fourier neural operator (FNO) FNO [32] belongs to the family of Neural Operators (NOs), designed to approximate the forward propagator of PDEs. FNO learns a resolution-invariant NO by working in the Fourier space and has shown success in learning challenging PDEs.

Physics-Informed Neural Networks (PINNs) Physics-informed neural networks [47] are methods for solving differential equations using a neural network $u_\theta(t, x)$ to approximate the solution by turning it into a multi-objective optimization problem. The neural network is trained to minimize the PDE residual as well as the error with regard to the boundary and initial conditions. PINNs naturally integrate observational data [30], but require retraining for each new condition.

Gradient-Based Inverse Method Since the surrogate model is fully differentiable, we use its gradient to solve inverse inference by minimizing the prediction loss [7, 40], where a function surface defining the unknown initial conditions [35], is specified through bilinear interpolation.

3.5 Data Format, Benchmark Access, Maintenance, and Extensibility

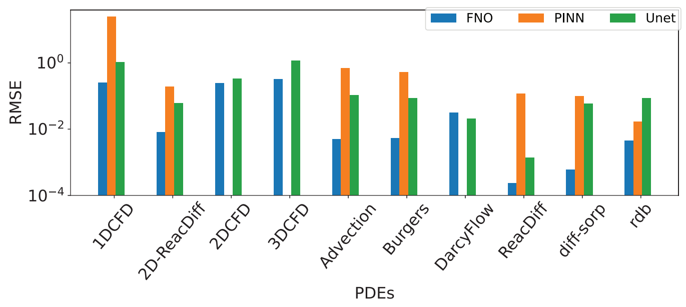
The benchmark consists of different data files, one for each equation, type of initial condition, and PDE parameter, using the HDF5 [15] binary data format. Each such file contains multiple arrays where each array has the dimensions N, T, X, Y, Z, V with N the number of samples, T the number of time steps, and X, Y, Z the spatial dimensions and V the dimension of the field. Additional information on the data format is provided in the Supplementary Material.

PDEBENCH's datasets are stored and maintained using DARUS, the University of Stuttgart's data repository based on the OpenSource Software DataVerse[3]. DARUS follows the Findable, Accessible, Interoperable and Reusable (FAIR) data principles [63]. All data uploaded to DaRUS gets a DOI as

[3] https://dataverse.org

```
from pdebench.fno.utils import FNODatasetSingle
filename = "data/2D_diff-react_NA_NA"
train_data = FNODatasetMult(filename)
train_loader = torch.utils.data.DataLoader(train_data)
```

Listing 2: Using the PyTorch data loader.



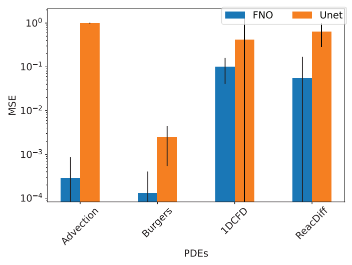


Figure 2: Comparisons of baseline models' performance for different problems for (a) the forward problem and (b) the inverse problem.

a persistent identifier, a license, and can be described with an extensive set of metadata, organized in metadata blocks. A dedicated team ensures that DaRUS is continuously maintained. Through DaRUS we provide a permanent DOI (doi:10.18419/darus-2986) [55] for the benchmark data. We also support a straightforward inclusion of the benchmark with a few lines of code. In Listing 1 we demonstrate the way in which the Dataverse [10] platform[4] supports the integration of pre-generated datasets using a few lines of code. Specifically, we utilize the easy-to-use pyDaRUS Python package to access the data. It provides a simple API for both downloading and uploading data as well as providing metadata to the Dataverse platform.

In Listing 2 we show an example leveraging pre-defined classes included in our benchmark code to load specific datasets as PyTorch [45] `Dataset` classes. Subsequently, these can be used to construct common `DataLoader` instances for training custom ML models. We utilize the Hydra [64] library simplifying the configuration of both surrogate model training as well as the generation of additional datasets. For the latter, we provide and expose various parameters of the underlying simulations for the end user to tweak. This provides a low barrier of entry for users to try out benchmarking with new experiments or baseline configurations.

4 A Selection of Experiments

In this section, we present a selection of experiments for the PDEBench datasets. An exhaustive discussion of all results is beyond the scope of this paper. An extensive set of additional results, tables, and plots can be found in the Appendix.

4.1 Baseline Setups

We trained and tested the baseline emulator models, namely U-Net, FNO, and PINN with the datasets generated with the PDEs described in subsection 3.2. The data was split into training and test data: 90% was for training and 10% for test data. For FNO, we followed the original implementation, hyperparameters, and training protocols. We trained U-Net similarly to FNO, but with the autoregressive methods with the pushforward trick with slight modification to the original implementation [4]. A more comprehensive comparison between different U-Net training methods is presented in subsection 4.3. The PINN baseline is implemented using the open-source DeepXDE [34] library. The training was performed on GeForce RTX 2080 GPUs for 1D/2D cases, and GeForce GTX 3090 for 3D cases. The detailed training protocol and hyper-parameters are provided in Appendix C.

[4] `https://darus.uni-stuttgart.de/dataverse/sciml_benchmark`

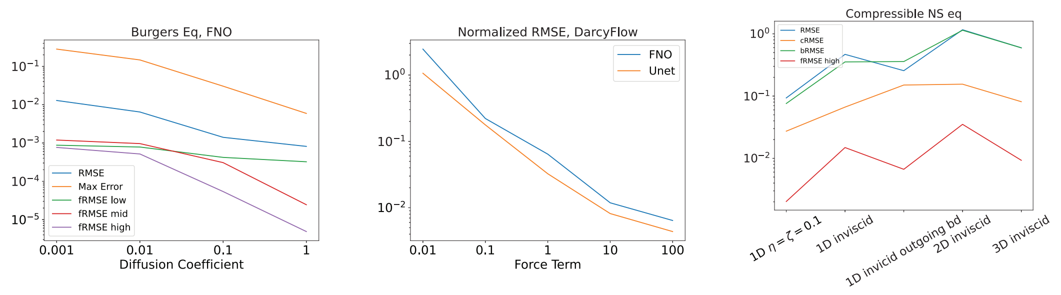


Figure 3: Detailed visualization of (a) Burgers', (b) DarcyFlow, and (c) Compressible NS eqs.

Training code and configurations are open and well documented, allowing researchers to easily reproduce or extend these methods.

4.2 Baseline Performance

Figure 2[5] visualizes the RMSE performance of the surrogate models, averaged for each trained model over different PDE parameters. A more detailed comparison is shown in Appendix E. [6] Among the baseline surrogate models, FNO provides the best prediction for most metrics. It learns the differential operators well, leading to low errors even for the conserved quantities and on the boundaries. Additionally, FNO has a consistent error of about 4×10^{-4} across the frequency spectrum for many problems highlighting its ability to learn in Fourier space. As an example, Figure 4 depicts the FNO and U-Net predictions for the 2D diffusion-reaction equation and the training data obtained from a numerical simulator. Our baseline results further indicate that the PINNs might deal better than expected with high-frequency features, despite prior observations [62]. As an example for an inverse problem setup, we identify the initial condition to minimize the prediction error of the ML surrogate over a 15 time steps horizon. In Figure 2b, we present the MSE of the prediction of estimated initial condition (with error bars) for 4 of the 11 datasets (1D). The results show that FNO outperforms U-Net also for the inverse problem. However, our benchmark also reveals several tasks which these methods cannot treat properly. First, Figure 3a shows that the FNO's error increases with decreasing diffusion coefficient where a strong discontinuity appears. This can be attributed to Gibb's phenomenon for FNO's limited maximum wave frequency in Fourier space, as shown by an increase of two orders of magnitude for high-frequency fRMSE. Second, Figure 3b shows that the normalized RMSE increases with decreasing force term, which is equivalent to decreasing the scale-value of the solution (in our case, force term 0.01 means $\mathrm{mean}(|u|) \approx 0.01$).[7] Third, Figure 3c shows several metrics for the compressible Navier-Stokes equations. It shows the overall RMSE is very bad in comparison to the basic PDEs, such as the Burgers equation. Interestingly, the 3D inviscid case shows lower error than 2D inviscid case. We posit this is due to lower resolution resulting in smooth train/validation samples which FNO can learn very efficiently. This also indicates that high-resolution training samples should be used to create a surrogate model for real-world problems with a Reynolds number of more than 10^6.

4.3 Temporal Error Analysis

Autoregressive Behaviour of U-Net We observed instabilities when training U-Net with fully autoregressive mode. When trained in an open loop, i.e. only 1 time step ahead without feedback of prediction, the error during testing quickly accumulates with more unrolled time steps. Therefore, we tried three different training methods as described in the previous section. Figure 5a shows the RMSE behaviour calculated at different unrolled time steps. It shows that for different U-Net training strategies, the RMSE behaviors are different. We observed that U-Net with the pushforward trick

[5] In Figure 2 CFD means compressible fluid dynamics or, equivalently, the compressible Navier-Stokes equations.
[6] Note that we omitted the PINN baseline score for the 2D/3D Compressible Navier-Stokes equations and the DarcyFlow. The reason for the former case is limited GPU memory, and the reason for the latter is that the DarcyFlow's problem setup is to learn the mapping from diffusion coefficients $a(x)$ to a steady state solution, which cannot be treated by PINNs. [7] Note that U-Net is consistently better than FNO in this case. We postulate that this is due to the strong similarity between this task and the diffusion-like regression task that the original U-Net targets.

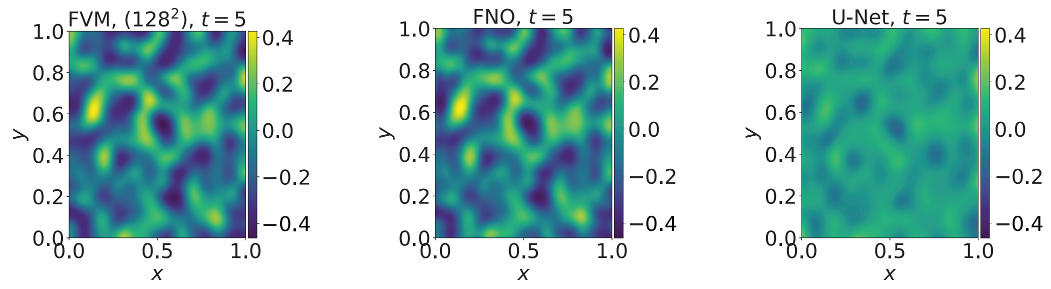


Figure 4: (a) Visualization of the 2D diffusion-reaction data generated with a standard finite volume (FVM) solver and a resolution of 128^2, (b) FNO prediction, and (c) U-Net prediction.

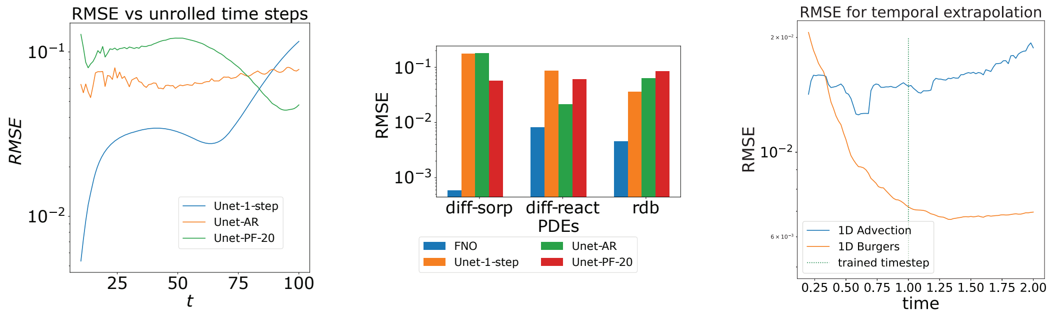


Figure 5: (a) Plots of the RMSE calculated at different unrolled time steps, (b) comparison of each autoregressive method, and (c) RMSE for temporal extrapolation.

provides better stability during training for all problems, and better accuracy for a longer prediction horizon. For this reason, we used the U-Net with pushforward trick as our baseline score for U-Net.

Temporal Extrapolation Figure 5c plots the RMSE temporal evolution of 1D Advection and Burgers equations predicted using the FNO model. Different from the other scenarios, we trained only until half of the time steps used in the other cases (the green dotted line in the figure). The main purpose of this experiment is to test how well the ML model (FNO) learns the temporal dependency of the PDEs with limited information. We observe monotonically increasing errors after the time steps used in training ($t > 1$). This indicates that the present ML models cannot properly capture the dynamic behavior of the PDEs, and it is a challenging task to provide reliable predictions beyond the time experienced during training.

4.4 Inference Time Comparison

In this subsection, we provide runtime comparisons between the numerical PDE solvers that we use to generate a single data sample and the FNO, the most accurate ML model according to our experiments. A fair runtime comparison between classical solvers and ML methods is challenging because these method classes are usually optimized for different hardware setups. In this paper, we provide the information on the hardware and system configuration in Appendix F[8]. In addition, we list the ML models' total training time. All the runtime numbers are given in seconds. The highest computational demand originates from the ML training time. However, once the ML models were trained, the ML models' predictions can be computed multiple orders of magnitudes more efficiently. A more complete overview of these experiments can be found in Appendix F. The ML model allows us to predict a solution even in the strong-viscous regime ($\eta = \zeta = 0.1$) efficiently since it eliminates the stability restriction defined by the Courant-Friedrichs-Lewy condition [31][9]. A more detailed analysis of the resolution sensitivity of inference time is provided in Appendix G.

[8] We used the same hardware resources and system configuration for the 2D/3D CFD simulations and the ML methods. Whether or not this consitutes a fair comparison is surely debatable. [9] Note that in the table the ML inference and training time does not monotonically increase with spatial-dimension. This is because the 2D/3D cases' time-step numbers and training sample numbers is much smaller than Diffusion-sorption case.

PDE	Resolution	Simulation time	ML inference time	ML training time
Diffusion-sorption	$1\,024^1$	59.83	0.32	48 760
2D CFD ($\eta = \zeta = 0.1$)	512^2	582.61	0.14	107 567
3D CFD (inviscid)	64^3	60.06	0.14	12 387

5 Conclusions and Limitations

With PDEBENCH we contribute an extensive benchmark suite for comparing and evaluating methods on the realm of Scientific ML. We provide both pre-computed datasets for easy access in a dataverse as well as code to generate new data from configurable simulation runs. The focus is on time-dependent PDE problems ranging from simple 1D equations to challenging 3D coupled systems of equations featuring challenging boundary conditions. Furthermore, we present a variety of different evaluation metrics in order to better understand strengths and weaknesses of the machine learning methods in a scientific computing context. We also provide an example application for utilizing Scientific ML methods for inverse modeling with our benchmark data. We believe this will be an important area in the future for machine learning models to produce competitive results both in accuracy as well as runtime when compared to numerical methods.

Limitations While PDEBENCH provides an easy-to-use, modular and extensible approach to devising and testing ML surrogates for PDE simulations, the scope of our benchmark is naturally limited. Our main focus is on time-dependent PDEs for different types of flow problems which pose a wide variety of challenges to Scientific ML. We currently do not cover other types of physics nor quantum mechanics as this goes beyond the scope of this paper. With respect to flow problems, our main limitations are that we do not yet cover multi-phase flows or non-rectangular domains. This is left for future work.

Acknowledgements

We thank MinMae (John) Kim, Ran Zhang, Tianqi Wang, Yizhou Yang, Gefei Shan and Simon Brown of the ANU Techlauncher for consulting on code design and implementation. Partially funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2075 – 390740016. We acknowledge support by the Stuttgart Center for Simulation Science (SimTech). We further thank the DaRUS-team, in particular Jan Range.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] All the contributions listed in the abstract are elaborated in section 3.
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph of section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We believe that our work does not have any potential negative societal impacts as it does not contain confidential or private data.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We do not include any confidential or private data, only numerical values related to general physical systems with no potential ethical issues or severe environmental damage.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] We do not include any theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] See the previous point.
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include the parameters used to reproduce the datasets in Appendix D and instructions to run the code in the README file in our code repository.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix C.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We ran several experiments using different parameters. The mean and standard deviation values for the errors are reported in the Figure 2.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] The types of GPUs used are provided in subsection 4.1.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We adopted the implementation of the baseline models with some modifications, with proper citation and credits to the authors, as well as existing software packages.
   (b) Did you mention the license of the assets? [Yes] Appropriate license notices are included in the affected source code files, and license of new assets is included in the supplementary materials.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] All the data generation scripts are included in the code repository.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] We generate all of our data, and the models we use are all openly accessible to the public, so we adopt and modify them, and include citations to the original authors.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] We do not include any personal information or offensive content in our datasets.

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We use neither crowdsourcing nor conduct research with human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] See the previous point.

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] See the previous point.

References

[1] K. R. Allen, T. Lopez-Guevara, K. Stachenfeld, A. Sanchez-Gonzalez, P. Battaglia, J. Hamrick, and T. Pfaff. Physical design using differentiable learned simulators. *arXiv preprint arXiv:2202.00728*, 2022.

[2] F. d. A. Belbute-Peres, Y.-f. Chen, and F. Sha. Hyperpinn: Learning parameterized differential equations with physics-informed hypernetworks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/pdf?id=LxUuRDUhRjM.

[3] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams. JAX-FLUIDS: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows, 2022. URL https://arxiv.org/abs/2203.13760.

[4] J. Brandstetter, D. Worrall, and M. Welling. Message passing neural pde solvers. In *The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/pdf?id=vSix3HPYKSU.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.

[6] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019. ISBN 978-1-108-42209-3. URL https://databookuw.com.

[7] K. Cao. *Inverse Problems for the Heat Equation Using Conjugate Gradient Methods*. PhD thesis, University of Leeds, 2018.

[8] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6572–6583. Curran Associates, Inc., 2018.

[9] M. Chu, N. Thuerey, H.-P. Seidel, C. Theobalt, and R. Zayer. Learning meaningful controls for fluids. *ACM Transactions on Graphics*, 40(4):1–13, Aug. 2021. doi: 10.1145/3476576.3476661.

[10] M. Crosas. The dataverse network: An open-source application for sharing, discovering and preserving data. *D-Lib Magazine*, Volume 17, 2011. URL http://www.dlib.org/dlib/january11/crosas/01crosas.html.

[11] H. Eivazi, M. Tahani, P. Schlatter, and R. Vinuesa. Physics-informed neural networks for solving reynolds-averaged navier-stokes equations. *arXiv:2107.10711 [physics]*, Jul 2021. URL http://arxiv.org/abs/2107.10711. arXiv: 2107.10711.

[12] R. P. Feynman. *Feynman lectures on physics - Volume 1*. 1963.

[13] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax–a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/3def184ad8f4755ff269862ea77393dd-Paper-round1.pdf.

[14] D. W. Gladish, D. E. Pagendam, L. J. M. Peeters, P. M. Kuhnert, and J. Vaze. Emulation Engines: Choice and Quantification of Uncertainty for Complex Hydrological Models. *Journal of Agricultural, Biological and Environmental Statistics*, 23(1):39–62, Mar. 2018. doi: 10.1007/s13253-017-0308-3.

[15] T. H. Group. An overview of the hdf5 technology suite and its applications, 2022. URL https://portal.hdfgroup.org/display/HDF5/HDF5.

[16] M. Guo and J. S. Hesthaven. Data-driven reduced order modeling for time-dependent problems. *Computer Methods in Applied Mechanics and Engineering*, 345:75–99, 2019. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2018.10.029. URL https://www.sciencedirect.com/science/article/pii/S0045782518305334.

[17] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, M. Rietmann, J. d. A. Ferrandis, W. Byeon, Z. Fang, and S. Choudhry. NVIDIA SimNet$\hat{\{}$TM}: An AI-accelerated multi-physics simulation framework. *arXiv:2012.07938 [physics]*, Dec. 2020.

[18] P. Holl and V. Koltun. Phiflow: A differentiable PDE solving framework for deep learning via physical simulations. page 5, 2020.

[19] Z. Huang, T. Schneider, M. Li, C. Jiang, D. Zorin, and D. Panozzo. A large-scale benchmark for the incompressible Navier-Stokes equations, 2021. URL https://arxiv.org/abs/2112.05309.

[20] T. Inoue, S.-i. Inutsuka, and H. Koyama. The Role of Ambipolar Diffusion in the Formation Process of Moderately Magnetized Diffuse Clouds. *The Astrophysical Journal*, 658(2):L99–L102, Apr. 2007. doi: 10.1086/514816.

[21] M. Karlbauer, T. Praditia, S. Otte, S. Oladyshkin, W. Nowak, and M. V. Butz. Composing partial differential equations with physics-aware neural networks. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research, Baltimore, USA, 16–23 Jul 2022.

[22] K. Kashinath, M. Mustafa, A. Albert, J.-L. Wu, C. Jiang, S. Esmaeilzadeh, K. Azizzadenesheli, R. Wang, A. Chattopadhyay, A. Singh, A. Manepalli, D. Chirila, R. Yu, R. Walters, B. White, H. Xiao, H. A. Tchelepi, P. Marcus, A. Anandkumar, P. Hassanzadeh, and n. Prabhat. Physics-informed machine learning: Case studies for weather and climate modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379 (2194):20200093, Apr. 2021. doi: 10.1098/rsta.2020.0093.

[23] D. I. Ketcheson, K. T. Mandli, A. J. Ahmadia, A. Alghamdi, M. Quezada de Luna, M. Parsani, M. G. Knepley, and M. Emmett. PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems. *SIAM Journal on Scientific Computing*, 34(4):C210–C231, Nov. 2012.

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[25] G. Klaasen and W. Troy. Stationary wave solutions of a system of reaction-diffusion equations derived from the fitzhugh–nagumo equations. *SIAM Journal on Applied Mathematics*, 44(1): 96–110, 1984. doi: 10.1137/0144008.

[26] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar. Neural operator: Learning Maps Between Function Spaces. *arXiv:2108.08481 [cs, math]*, Sept. 2021.

[27] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[28] A. Lavin, H. Zenil, B. Paige, D. Krakauer, J. Gottschlich, T. Mattson, A. Anandkumar, S. Choudry, K. Rocki, A. G. Baydin, C. Prunkl, B. Paige, O. Isayev, E. Peterson, P. L. McMahon, J. Macke, K. Cranmer, J. Zhang, H. Wainwright, A. Hanuka, M. Veloso, S. Assefa, S. Zheng, and A. Pfeffer. Simulation Intelligence: Towards a New Generation of Scientific Methods. *arXiv:2112.03235 [cs]*, Dec. 2021.

[29] R. Leiteritz, M. Hurler, and D. Pflüger. Learning free-surface flow with physics-informed neural networks. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1668–1673, 2021. doi: 10.1109/ICMLA52953.2021.00266.

[30] R. Leiteritz, P. Buchfink, B. Haasdonk, and D. Pflüger. Surrogate-data-enriched physics-aware neural networks. *Proceedings of the Northern Lights Deep Learning Workshop*, 2, 04 2022. doi: 10.7557/18.6268.

[31] H. Lewy, K. Friedrichs, and R. Courant. Über die partiellen differenzengleichungen der mathematischen physik. *Mathematische annalen*, 100:32–74, 1928.

[32] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *International Conference on Learning Representations (ICLR)*, 2021.

[33] G. Limousin, J.-P. Gaudet, L. Charlet, S. Szenknect, V. Barthès, and M. Krimissa. Sorption isotherms: A review on physical bases, modeling and measurement. *Applied Geochemistry*, 22 (2):249–275, 2007. ISSN 0883-2927. doi: https://doi.org/10.1016/j.apgeochem.2006.09.010. URL `https://www.sciencedirect.com/science/article/pii/S0883292706002629`.

[34] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021. doi: 10.1137/19M1274067.

[35] D. MacKinlay, D. Pagendam, P. M. Kuhnert, T. Cui, D. Robertson, and S. Janardhanan. Model Inversion for Spatio-temporal Processes using the Fourier Neural Operator. In *Neurips Workshop on Machine Learning for the Physical Sciences*, page 7, 2021.

[36] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 Competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, 36(1):54–74, Jan. 2020. doi: 10.1016/j.ijforecast.2019.04.014.

[37] S. K. Mitusch, S. W. Funke, and J. S. Dokken. Dolfin-Adjoint 2018.1: Automated adjoints for FEniCS and Firedrake. *Journal of Open Source Software*, 4(38):1292, June 2019. doi: 10.21105/joss.01292.

[38] J. Močkus. On Bayesian Methods for Seeking the Extremum. In G. I. Marchuk, editor, *Optimization Techniques IFIP Technical Conference: Novosibirsk, July 1–7, 1974*, Lecture Notes in Computer Science, pages 400–404, Berlin, Heidelberg, 1975. Springer. ISBN 978-3-662-38527-2. doi: 10.1007/978-3-662-38527-2_55.

[39] F. Moukalled, L. Mangani, and M. Darwish. *The Finite Volume Method in Computational Fluid Dynamics*. Springer, 1 edition, 2016. doi: 10.1007/978-3-319-16874-6.

[40] J. Nocedal and S. J. Wright. *Numerical optimization*. Springer, 1999.

[41] W. Nowak and A. Guthke. Entropy-based experimental design for optimal model discrimination in the geosciences. *Entropy*, 18(11), 2016. doi: 10.3390/e18110409.

[42] A. O'Hagan. Curve Fitting and Optimal Design for Prediction. *Journal of the Royal Statistical Society: Series B (Methodological)*, 40(1):1–24, 1978. doi: 10.1111/j.2517-6161.1978.tb01643.x.

[43] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore. PMLB: A large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec. 2017. doi: 10.1186/s13040-017-0154-4.

[44] K. Otness, A. Gjoka, J. Bruna, D. Panozzo, B. Peherstorfer, T. Schneider, and D. Zorin. An extensible benchmark suite for learning to simulate physical systems. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021. URL `https://openreview.net/forum?id=pY9MHwmrymR`.

[45] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[46] C. Rackauckas. The essential tools of scientific machine learning (Scientific ML). *The Winnower*, Aug. 2019. doi: 10.15200/winn.156631.13064.

[47] M. Raissi, P. Perdikaris, and G. 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, Feb. 2019. doi: 10.1016/j.jcp.2018.10.045.

[48] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

[49] L. Ruthotto and E. Haber. Deep Neural Networks motivated by Partial Differential Equations. *arXiv:1804.04272 [cs, math, stat]*, Apr. 2018.

[50] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959. Curran Associates, Inc., 2012.

[51] K. Stachenfeld, D. B. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, P. Battaglia, and A. Sanchez-Gonzalez. Learned coarse models for efficient turbulence simulation, 2022. URL `https://arxiv.org/abs/2112.15275`.

[52] J. M. Stone and M. L. Norman. ZEUS-2D: A Radiation Magnetohydrodynamics Code for Astrophysical Flows in Two Space Dimensions. I. The Hydrodynamic Algorithms and Tests. *The Astrophysical Journal Supplement*, 80:753, June 1992. doi: 10.1086/191680.

[53] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010. doi: 10.1017/S0962492910000061.

[54] D. J. Tait and T. Damoulas. Variational Autoencoding of PDE Inverse Problems. *arXiv:2006.15641 [cs, stat]*, June 2020.

[55] M. Takamoto, T. Pradita, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and M. Niepert. PDEBench: A diverse and comprehensive benchmark for scientific machine learning, 2022. URL `https://darus.uni-stuttgart.de/privateurl.xhtml?token=1be27526-348a-40ed-9fd0-c62f588efc01`.

[56] A. Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, Jan. 2005. ISBN 978-0-89871-792-1.

[57] E. F. Toro, M. Spruce, and W. Speares. Restoration of the contact surface in the HLL-Riemann solver. *Shock Waves*, 4(1):25–34, July 1994. doi: 10.1007/BF01414629.

[58] A. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society B*, 237:37–72, 1952.

[59] B. 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, July 1979. doi: 10.1016/0021-9991(79)90145-1.

[60] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero,

C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[61] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. Towards Physics-informed Deep Learning for Turbulent Flow Prediction. *arXiv:1911.08655 [physics, stat]*, June 2020.

[62] S. Wang, X. Yu, and P. Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2021.110768. URL `https://www.sciencedirect.com/science/article/pii/S002199912100663X`.

[63] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.

[64] O. Yadan. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL `https://github.com/facebookresearch/hydra`.