Journal of Scientific Computing (2022) 92:88
https://doi.org/10.1007/s10915-022-01939-z

Scientific Machine Learning Through Physics–Informed Neural Networks: Where we are and What's Next

Salvatore Cuomo[1] · **Vincenzo Schiano Di Cola**[2] · **Fabio Giampaolo**[1] · **Gianluigi Rozza**[3] · **Maziar Raissi**[4] · **Francesco Piccialli**[1]

Received: 14 January 2022 / Revised: 6 July 2022 / Accepted: 6 July 2022 /
Published online: 26 July 2022

Abstract
Physics-Informed Neural Networks (PINN) are neural networks (NNs) that encode model equations, like Partial Differential Equations (PDE), as a component of the neural network itself. PINNs are nowadays used to solve PDEs, fractional equations, integral-differential equations, and stochastic PDEs. This novel methodology has arisen as a multi-task learning framework in which a NN must fit observed data while reducing a PDE residual. This article provides a comprehensive review of the literature on PINNs: while the primary goal of the study was to characterize these networks and their related advantages and disadvantages. The review also attempts to incorporate publications on a broader range of collocation-based physics informed neural networks, which stars form the vanilla PINN, as well as many other variants, such as physics-constrained neural networks (PCNN), variational hp-VPINN, and conservative PINN (CPINN). The study indicates that most research has focused on customizing the PINN through different activation functions, gradient optimization techniques, neural network structures, and loss function structures. Despite the wide range of applications for which PINNs have been used, by demonstrating their ability to be more feasible in some contexts than classical numerical techniques like Finite Element Method (FEM), advancements are still possible, most notably theoretical issues that remain unresolved.

Keywords Physics–Informed Neural Networks · Scientific Machine Learning · Deep Neural Networks · Nonlinear equations · Numerical methods · Partial Differential Equations · Uncertainty

S. Cuomo, V. Schiano, F. Giampaolo, G. Rozza and M. Raissi are contributed equally to this study

✉ Francesco Piccialli
francesco.piccialli@unina.it

Extended author information available on the last page of the article

1 Introduction

Deep neural networks have succeeded in tasks such as computer vision, natural language processing, and game theory. Deep Learning (DL) has transformed how categorization, pattern recognition, and regression tasks are performed across various application domains.

Deep neural networks are increasingly being used to tackle classical applied mathematics problems such as partial differential equations (PDEs) utilizing machine learning and artificial intelligence approaches. Due to, for example, significant nonlinearities, convection dominance, or shocks, some PDEs are notoriously difficult to solve using standard numerical approaches. Deep learning has recently emerged as a new paradigm of scientific computing thanks to neural networks' universal approximation and great expressivity. Recent studies have shown deep learning to be a promising method for building meta-models for fast predictions of dynamic systems. In particular, NNs have proven to represent the underlying nonlinear input-output relationship in complex systems. Unfortunately, dealing with such high dimensional-complex systems are not exempt from the curse of dimensionality, which Bellman first described in the context of optimal control problems [15]. However, machine learning-based algorithms are promising for solving PDEs [19].

Indeed, Blechschmidt and Ernst [19] consider machine learning-based PDE solution approaches will continue to be an important study subject in the next years as deep learning develops in methodological, theoretical, and algorithmic developments. Simple neural network models, such as MLPs with few hidden layers, were used in early work for solving differential equations [89]. Modern methods, based on NN techniques, take advantage of optimization frameworks and auto-differentiation, like Berg and Nyström [16] that suggested a unified deep neural network technique for estimating PDE solutions. Furthermore, it is envisioned that DNN will be able to create an interpretable hybrid Earth system model based on neural networks for Earth and climate sciences [68].

Nowadays, the literature does not have a clear nomenclature for integrating previous knowledge of a physical phenomenon with deep learning. 'Physics-informed,' 'physics-based,' 'physics-guided,' and 'theory-guided' are often some used terms. Kim et al [80] developed the overall taxonomy of what they called informed deep learning, followed by a literature review in the field of dynamical systems. Their taxonomy is organized into three conceptual stages: (i) what kind of deep neural network is used, (ii) how physical knowledge is represented, and (iii) how physical information is integrated. Inspired by their work, we will investigate PINNs, a 2017 framework, and demonstrate how neural network features are used, how physical information is supplied, and what physical problems have been solved in the literature.

1.1 What the PINNs are

Physics–Informed Neural Networks (PINNs) are a scientific machine learning technique used to solve problems involving Partial Differential Equations (PDEs). PINNs approximate PDE solutions by training a neural network to minimize a loss function; it includes terms reflecting the initial and boundary conditions along the space-time domain's boundary and the PDE residual at selected points in the domain (called collocation point). PINNs are deep-learning networks that, given an input point in the integration domain, produce an estimated solution in that point of a differential equation after training. Incorporating a residual network that encodes the governing physics equations is a significant novelty with PINNs. The basic concept behind PINN training is that it can be thought of as an unsupervised strategy that does not require labelled data, such as results from prior simulations or experiments. The PINN algorithm is essentially a mesh-free technique that finds PDE solutions by converting the problem of directly solving the governing equations into a loss function optimization problem. It works by integrating the mathematical model into the network and reinforcing the loss function with a residual term from the governing equation, which acts as a penalizing term to restrict the space of acceptable solutions.

PINNs take into account the underlying PDE, i.e. the physics of the problem, rather than attempting to deduce the solution based solely on data, i.e. by fitting a neural network to a set of state-value pairs. The idea of creating physics-informed learning machines that employ systematically structured prior knowledge about the solution can be traced back to earlier research by Owhadi [125], which revealed the promising technique of leveraging such prior knowledge. Raissi et al [141, 142] used Gaussian process regression to construct representations of linear operator functionals, accurately inferring the solution and providing uncertainty estimates for a variety of physical problems; this was then extended in [140, 145]. PINNs were introduced in 2017 as a new class of data-driven solvers in a two-part article [143, 144] published in a merged version afterwards in 2019 [146]. Raissi et al [146] introduce and illustrate the PINN approach for solving nonlinear PDEs, like Schrödinger, Burgers, and Allen–Cahn equations. They created physics-informed neural networks (PINNs) which can handle both forward problems of estimating the solutions of governing mathematical models and inverse problems, where the model parameters are learnt from observable data.

The concept of incorporating prior knowledge into a machine learning algorithm is not entirely novel. In fact Dissanayake and Phan-Thien [39] can be considered one of the first PINNs. This paper followed the results of the universal approximation achievements of the late 1980s, [65]; then in the early 90s several methodologies were proposed to use neural networks to approximate PDEs, like the work on constrained neural networks [89, 135] or [93]. In particular Dissanayake and Phan-Thien [39] employed a simple neural networks to approximate a PDE, where the neural network's output was a single value that was designed to approximate the solution value at the specified input position. The network had two hidden layers, with 3, 5 or 10 nodes for each layer. The network loss function approximated the L^2 error of the approximation on the interior and boundary of the domain using point-collocation. While, the loss is evaluated using a quasi-Newtonian approach and the gradients are evaluated using finite difference.
In Lagaris et al [89], the solution of a differential equation is expressed as a constant term and an adjustable term with unknown parameters, the best parameter values are determined via a neural network. However, their method only works for problems with regular borders. Lagaris et al [90] extends the method to problems with irregular borders.

As computing power increased during the 2000s, increasingly sophisticated models with more parameters and numerous layers became the standard Özbay et al [127]. Different deep model using MLP, were introduced, also using Radial Basis Function Kumar and Yadav [87].

Research into the use of NNs to solve PDEs continued to gain traction in the late 2010s, thanks to advancements in the hardware used to run NN training, the discovery of better practices for training NNs, and the availability of open-source packages, such as Tensorflow [55], and the availability of Automatic differentiation in such packages [129].

Finally, more recent advancements by Kondor and Trivedi [83], and Mallat [102], brought to Raissi et al [146] solution that extended previous notions while also introducing fundamentally new approaches, such as a discrete time-stepping scheme that efficiently leverages the predictive ability of neural networks [82]. The fact that the framework could be utilized directly by plugging it into any differential problem simplified the learning curve for beginning to use such, and it was the first step for many researchers who wanted to solve their problems with a Neural network approach [105]. The success of the PINNs can be seen from the rate at which Raissi et al [146] is cited, and the exponentially growing number of citations in the recent years (Fig. 1).

However, PINN is not the only NN framework utilized to solve PDEs. Various frameworks have been proposed in recent years, and, while not exhaustive, we have attempted to highlight the most relevant ones in this paragraph.

The Deep Ritz method (DRM) [42], where the loss is defined as the energy of the problem's solution.

Then there approaches based on the Galerkin method, or Petrov–Galerkin method, where the loss is given by multiplying the residual by a test function, and when is the volumetric residual we have a Deep Galerkin Method (DGM) [160]. Whereas, when a Galerkin approach is used on collocation points the framework is a variant of PINNs, i.e. a hp-VPINN Kharazmi et al [78].

Within the a collocation based approach, i.e. PINN methodology [109, 146, 191], many other variants were proposed, as the variational hp-VPINN, as well as conservative PINN (CPINN)[71]. Another approach is physics-constrained neural networks (PCNNs) [97, 168, 200]. while PINNs incorporate both the PDE and its initial/boundary conditions (soft BC) in the training loss function, PCNNs, are "data-free" NNs, i.e. they enforce the initial/boundary conditions (hard BC) via a custom NN architecture while embedding the PDE in the training loss. This soft form technique is described in Raissi et al [146], where the term "physics-informed neural networks" was coined (PINNs).

Because there are more articles on PINN than any other specific variant, such as PCNN, hp-VPINN, CPINN, and so on, this review will primarily focus on PINN, with some discussion of the various variants that have emerged, that is, NN architectures that solve differential equations based on collocation points.

Finally, the acronym PINN will be written in its singular form rather than its plural form in this review, as it is considered representative of a family of neural networks of the same type.

Various review papers involving PINN have been published. About the potentials, limitations and applications for forward and inverse problems [74] for three-dimensional flows [20], or a comparison with other ML techniques [19]. An introductory course on PINNs that covers the fundamentals of Machine Learning and Neural Networks can be found from Kollmannsberger et al [82]. PINN is also compared against other methods that can be applied to solve PDEs, like the one based on the Feynman–Kac theorem [19]. Finally, PINNs have also been extended to solve integrodifferential equations (IDEs) [128, 193] or stochastic differential equations (SDEs) [189, 195].

Being able to learn PDEs, PINNs have several advantages over conventional methods. PINNs, in particular, are mesh-free methods that enable on-demand solution computation after a training stage, and they allow solutions to be made differentiable using analytical gradients. Finally, they provide an easy way to solve forward jointly and inverse problems using the same optimization problem. In addition to solving differential equations (the forward problem), PINNs may be used to solve inverse problems such as characterizing fluid flows from sensor data. In fact, the same code used to solve forward problems can be used to solve inverse problems with minimal modification. Moreover, in the context of inverse design, PDEs can also be enforced as hard constraints (hPINN) [101]. Indeed, PINNs can address PDEs in domains with very complicated geometries or in very high dimensions that are all difficult to numerically simulate as well as inverse problems and constrained optimization problems.

1.2 What is this Review About

In this survey, we focus on how PINNs are used to address different scientific computing problems, the building blocks of PINNs, the aspects related to learning theory, what toolsets are already available, future directions and recent trends, and issues regarding accuracy and convergence. According to different problems, we show how PINNs solvers have been customized in literature by configuring the depth, layer size, activation functions and using transfer learning.

This article can be considered as an extensive literature review on PINNs. It was mostly conducted by searching Scopus for the terms:

```
((physic* OR physical) W/2 (informed OR constrained) W/2
"neural network")
```

The primary research question was to determine what PINNs are and their associated benefits and drawbacks. The research also focused on the outputs from the CRUNCH research group in the Division of Applied Mathematics at Brown University and then on the (Physics–Informed Learning Machines for Multiscale and Multiphysics Problems) PhILMs Center, which is a collaboration with the Pacific Northwest National Laboratory. In such a frame, the primary authors who have been involved in this literature research are Karniadakis G.E., Perdikaris P., and Raissi M. Additionally, the review considered studies that addressed a broader topic than PINNs, namely physics-guided neural networks, as well as physics-informed machine learning and deep learning.

Figure 1 summarizes what the influence of PINN is in today's literature and applications.

2 The Building Blocks of a PINN

Physically-informed neural networks can address problems that are described by few data, or noisy experiment observations. Because they can use known data while adhering to any given physical law specified by general nonlinear partial differential equations, PINNs can also be considered neural networks that deal with supervised learning problems [52]. PINNs can solve differential equations expressed, in the most general form, like:

$$\begin{aligned} \mathcal{F}(u(z);\gamma) &= f(z) \qquad z \text{ in } \Omega, \\ \mathcal{B}(u(z)) &= g(z) \qquad z \text{ in } \partial\Omega \end{aligned} \tag{1}$$

defined on the domain $\Omega \subset \mathbb{R}^d$ with the boundary $\partial\Omega$. Where $z := [x_1, \ldots, x_{d-1}; t]$ indicates the space-time coordinate vector, u represents the unknown solution, γ are the parameters related to the physics, f is the function identifying the data of the problem and $\mathcal{F}$ is the non linear differential operator. Finally, since the initial condition can actually be considered as a type of Dirichlet boundary condition on the spatio-temporal domain, it is possible to denote $\mathcal{B}$ as the operator indicating arbitrary initial or boundary conditions related to the problem and g the boundary function. Indeed, the boundary conditions can be Dirichlet, Neumann, Robin, or periodic boundary conditions.

Equation (1) can describe numerous physical systems including both forward and inverse problems. The goal of forward problems is to find the function u for every z, where γ are specified parameters. In the case of the inverse problem, γ must also be determined from the data. The reader can find an operator based mathematical formulation of Eq. (1) in the work of Mishra and Molinaro [111].

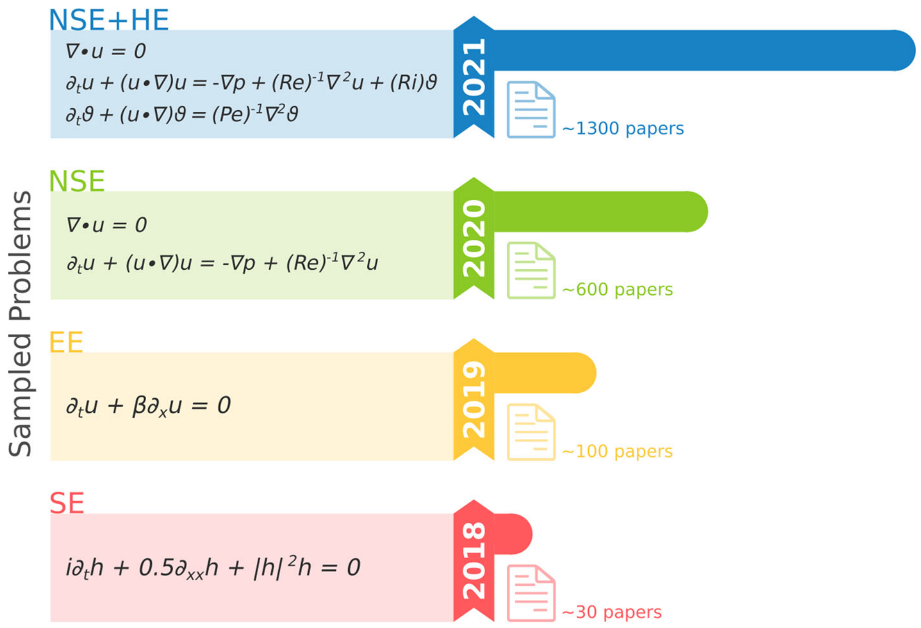

Fig. 1 A number of papers related to PINNs (on the right) addressed problems on which PINNs are applied (on the left) by year. PINN is having a significant impact in the literature and in scientific applications. The number of papers referencing Raissi or including PINN in their abstract title or keywords is increasing exponentially. The number of papers about PINN more than quintupled between 2019 and 2020, and there were twice as many new papers published in 2021 as there were in 2020. Since Raissi's first papers on arXiv in 2019 [146], a boost in citations can be seen in late 2018 and 2019. On the left, we display a sample problem solved in that year by one of the articles, specifically a type of time-dependent equation. Some of the addressed problems to be solved in the first vanilla PINN, for example, were the Allen–Cahn equation, the Korteweg–de Vries equation, or the 1D nonlinear Shrödinger problem (SE). By the end of 2019 Mao et al [103] solved with the same PINN Euler equations (EE) that model high-speed aerodynamic flows. By the end of 2020 Jin et al [73] solved the incompressible Navier–Stokes equations (NSE). Finally, in 2021 Cai et al [22] coupled the Navier–Stokes equations with the corresponding temperature equation for analyzing heat flow convection (NSE+HE)

In the PINN methodology, $u(z)$ is computationally predicted by a NN, parametrized by a set of parameters θ, giving rise to an approximation

$$\hat{u}_\theta(z) \approx u(z);$$

where $\hat{(\cdot)}_\theta$ denotes a NN approximation realized with θ.

In this context, where forward and inverse problems are analyzed in the same framework, and given that PINN can adequately solve both problems, we will use θ to represent both the vector of all unknown parameters in the neural network that represents the surrogate model and unknown parameters γ in the case of an inverse problem.

In such a context, the NN must learn to approximate the differential equations through finding θ that define the NN by minimizing a loss function that depends on the differential equation $\mathcal{L}_\mathcal{F}$, the boundary conditions $\mathcal{L}_\mathcal{B}$, and eventually some known data $\mathcal{L}_{data}$, each of them adequately weighted:

$$\theta^* = \arg\min_\theta \left(\omega_\mathcal{F}\mathcal{L}_\mathcal{F}(\theta) + \omega_\mathcal{B}\mathcal{L}_\mathcal{B}(\theta) + \omega_d\mathcal{L}_{data}(\theta)\right). \tag{2}$$

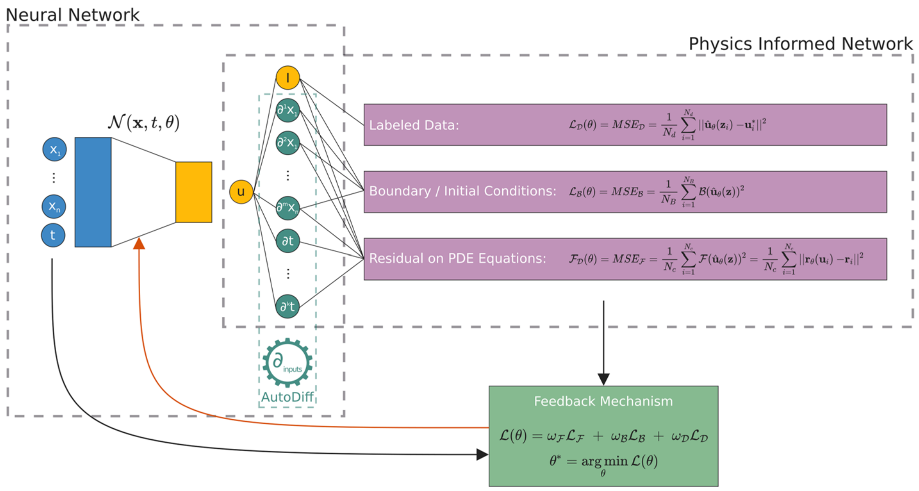
Neural Network
Physics Informed Network
$\mathcal{N}(\mathbf{x}, t, \theta)$
x_1
x_n
t
u
I
$\partial^1 x_1$
$\partial^2 x_1$
$\partial^m x_n$
∂t
$\partial^l t$
AutoDiff
inputs
Labeled Data: $\mathcal{L}_\mathcal{D}(\theta) = MSE_\mathcal{D} = \frac{1}{N_d}\sum_{i=1}^{N_d} ||\hat{\mathbf{u}}_\theta(\mathbf{z}_i) - \mathbf{u}_i^*||^2$
Boundary / Initial Conditions: $\mathcal{L}_\mathcal{B}(\theta) = MSE_\mathcal{B} = \frac{1}{N_B}\sum_{i=1}^{N_B} \mathcal{B}(\hat{\mathbf{u}}_\theta(\mathbf{z}))^2$
Residual on PDE Equations: $\mathcal{F}_\mathcal{D}(\theta) = MSE_\mathcal{F} = \frac{1}{N_c}\sum_{i=1}^{N_c} \mathcal{F}(\hat{\mathbf{u}}_\theta(\mathbf{z}))^2 = \frac{1}{N_c}\sum_{i=1}^{N_c} ||\mathbf{r}_\theta(\mathbf{u}_i) - \mathbf{r}_i||^2$
Feedback Mechanism
$\mathcal{L}(\theta) = \omega_\mathcal{F}\mathcal{L}_\mathcal{F} + \omega_\mathcal{B}\mathcal{L}_\mathcal{B} + \omega_\mathcal{D}\mathcal{L}_\mathcal{D}$
$\theta^* = \arg\min_\theta \mathcal{L}(\theta)$

Fig. 2 Physics-informed neural networks building blocks. PINNs are made up of differential equation residual (loss) terms, as well as initial and boundary conditions. The network's inputs (variables) are transformed into network outputs (the field u). The network is defined by θ. The second area is the physics informed network, which takes the output field u and computes the derivative using the given equations. The boundary/initial condition is also evaluated (if it has not already been hard-encoded in the neural network), and also the labeled data observations are calculated (in case there is any data available). The final step with all of these residuals is the feedback mechanism, that minimizes the loss, using an optimizer, according to some learning rate in order to obtain the NN parameters θ

To summarize, PINN can be viewed as an unsupervised learning approach when they are trained solely using physical equations and boundary conditions for forward problems; however, for inverse problems or when some physical properties are derived from data that may be noisy, PINN can be considered supervised learning methodologies.

In the following paragraph, we will discuss the types of NN used to approximate $u(z)$, how the information derived by $\mathcal{F}$ is incorporated in the model, and how the NN learns from the equations and additional given data.

Figure 2 summarizes all the PINN's building blocks discussed in this section. PINN are composed of three components: a neural network, a physics-informed network, and a feedback mechanism. The first block is a NN, $\hat{u}_\theta$, that accepts vector variables z from the Eq. (1) and outputs the filed value u. The second block can be thought of PINN's functional component, as it computes the derivative to determine the losses of equation terms, as well as the terms of the initial and boundary conditions of Eq. (2). Generally, the first two blocks are linked using algorithmic differentiation, which is used to inject physical equations into the NN during the training phase. Thus, the feedback mechanism minimizes the loss according to some learning rate, in order to fix the NN parameters vector θ of the NN $\hat{u}_\theta$. In the following, it will be clear from the context to what network we are referring to, whether the NN or the functional network that derives the physical information.

2.1 Neural Network Architecture

The representational ability of neural networks is well established. According to the universal approximation theorem, any continuous function can be arbitrarily closely approximated by a multi-layer perceptron with only one hidden layer and a finite number of neurons [17, 34, 65, 192]. While neural networks can express very complex functions compactly, determining the precise parameters (weights and biases) required to solve a specific PDE can be difficult [175]. Furthermore, identifying the appropriate artificial neural network (ANN) architecture can be challenging. There are two approaches: shallow learning networks, which have a single hidden layer and can still produce any non-linear continuous function, and deep neural networks (DNN), a type of ANN that uses more than two layers of neural networks to model complicated relationships [2]. Finally, the taxonomy of various Deep Learning architectures is still a matter of research [2, 117, 155]

The main architectures covered in the Deep Learning literature include fully connected feed-forward networks (called FFNN, or also FNN, FCNN, or FF–DNN), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) [29, 92]. Other more recent architectures encompass Auto–Encoder (AE), Deep Belief Network (DBN), Generative Adversarial Network (GAN), and Bayesian Deep Learning (BDL) [5, 17].

Various PINN extensions have been investigated, based on some of these networks. An example is estimating the PINN solution's uncertainty using Bayesian neural networks. Alternatively, when training CNNs and RNNs, finite-difference filters can be used to construct the differential equation residual in a PINN-like loss function. This section will illustrate all of these examples and more; first, we will define the mathematical framework for characterizing the various networks.

According to Caterini and Chang [25] a generic deep neural network with L layers, can be represented as the composition of L functions $f_i(x_i, \theta_i)$ where x_i are known as state variables, and θ_i denote the set of parameters for the i-th layer. So a function $u(x)$ is approximated as

$$u_\theta(x) = f_L \circ f_{L-1} \ldots \circ f_1(x). \tag{3}$$

where each f_i is defined on two inner product spaces E_i and H_i, being $f_i \in E_i \times H_i$ and the layer composition, represented by $\circ$, is to be read as $f_2 \circ f_1(x) = f_2(f_1(x))$.

Since Raissi et al [146] original vanilla PINN, the majority of solutions have used feed-forward neural networks. However, some researchers have experimented with different types of neural networks to see how they affect overall PINN performance, in particular with CNN, RNN, and GAN, and there seems there has been not yet a full investigation of other networks within the PINN framework. In this subsection, we attempt to synthesize the mainstream solutions, utilizing feed-forward networks, and all of the other versions. Table 1 contains a synthesis reporting the kind of neural network and the paper that implemented it.

2.1.1 Feed-forward Neural Network

A feed-forward neural network, also known as a multi-layer perceptron (MLP), is a collection of neurons organized in layers that perform calculations sequentially through the layers. Feed-forward networks, also known as multilayer perceptrons (MLP), are DNNs with several hidden layers that only move forward (no loopback). In a fully connected neural network, neurons in adjacent layers are connected, whereas neurons inside a single layer are not linked. Each neuron is a mathematical operation that applies an activation function to the weighted sum of it's own inputs plus a bias factor [175]. Given an input $x \in \Omega$ an MLP transforms it to an output, through a layer of units (neurons) which compose of affine-linear maps between units (in successive layers) and scalar non-linear activation functions within units, resulting in a composition of functions. So for MLP, it is possible to specify (3) as

$$f_i(x_i; W_i, b_i) = \alpha_i(W_i \cdot x_i + b_i)$$

equipping each E_i and H_i with the standard Euclidean inner product, i.e. $E = H = \mathbb{R}$ [26], and α_i is a scalar (non-linear) activation function. The machine learning literature has studied several different activation functions, which we shall discuss later in this section. Equation (3) can also be rewritten in conformity with the notation used in Mishra and Molinaro [111]:

$$u_\theta(x) = C_K \circ \alpha \circ C_{K-1} \ldots \circ \alpha \circ C_1(x), \tag{4}$$

where for any $1 \leq k \leq K$, it it is defined

$$C_k(x_k) = W_k x_k + b_k. \tag{5}$$

Thus a neural network consists of an input layer, an output layer, and $(K-2)$ hidden layers.

FFNN Architectures

While the ideal DNN architecture is still an ongoing research; papers implementing PINN have attempted to empirically optimise the architecture's characteristics, such as the number of layers and neurons in each layer. Smaller DNNs may be unable to effectively approximate unknown functions, whereas too large DNNs may be difficult to train, particularly with small datasets. Raissi et al [146] used different typologies of DNN, for each problem, like a 5-layer deep neural network with 100 neurons per layer, an DNN with 4 hidden layers and 200 neurons per layer or a 9 layers with 20 neuron each layer. Tartakovsky et al [170] empirically determine the feedforward network size, in particular they use three hidden layers and 50 units per layer, all with an hyperbolic tangent activation function. Another example of how the differential problem affects network architecture can be found in Kharazmi et al [78] for their hp-VPINN. The architecture is implemented with four layers and twenty neurons per layer, but for an advection equation with a double discontinuity of the exact solution, they use an eight-layered deep network. For a constrained approach, by utilizing a specific portion of the NN to satisfy the required boundary conditions, Zhu et al [199] use five hidden layers and 250 neurons per layer to constitute the fully connected neural network. Bringing the number of layers higher, in PINNeik [175], a DNN with ten hidden layers containing twenty neurons each is utilized, with a locally adaptive inverse tangent function as the activation function for all hidden layers except the final layer, which has a linear activation function. He et al [60] examines the effect of neural network size on state estimation accuracy. They begin by experimenting with various hidden layer sizes ranging from three to five, while maintaining a value of 32 neurons per layer. Then they set the number of hidden layers to three, the activation function to hyperbolic tangent, while varying the number of neurons in each hidden layer. Other publications have attempted to understand how the number of layers, neurons, and activation functions effect the NN's approximation quality with respect to the problem to be solved, like [19].

Multiple FFNN

Although many publications employ a single fully connected network, a rising number of research papers have been addressing PINN with multiple fully connected network blended together, e.g. to approximate specific equation of a larger mathematical model. An architecture composed of five the feed-forward neural network is proposed by Haghighat et al [57]. In a two-phase Stefan problem, discussed later in this review and in Cai et al [22], a DNN is used to model the unknown interface between two different material phases, while another DNN describes the two temperature distributions of the phases. Instead of a single NN across the entire domain, Moseley et al [116] suggests using multiple neural networks, one for each subdomain. Finally, Lu et al [99] employed a pair of DNN, one for encoding the input space (branch net) and the other for encoding the domain of the output functions (trunk net). This architecture, known as DeepONet, is particularly generic because no requirements are made to the topology of the branch or trunk network, despite the fact that the two sub-networks have been implemented as FFNNs as in Lin et al [96].

Shallow Networks

To overcome some difficulties, various researchers have also tried to investigate shallower network solutions: these can be sparse neural networks, instead of fully connected architectures, or more likely single hidden layers as ELM (Extreme Learning Machine) [66]. When compared to the shallow architecture, more hidden layers aid in the modeling of complicated nonlinear relationships [155], however, using PINNs for real problems can result in deep networks with many layers associated with high training costs and efficiency issues. For this reason, not only deep neural networks have been employed for PINNs but also shallow ANN are reported in the literature. X–TFC, developed by Schiassi et al [153], employs a single-layer NN trained using the ELM algorithm. While PIELM [41] is proposed as a faster alternative, using a hybrid neural network-based method that combines two ideas from PINN and ELM. ELM only updates the weights of the outer layer, leaving the weights of the inner layer unchanged.
Finally, in Ramabathiran and Ramachandran [148] a Sparse, Physics-based, and partially Interpretable Neural Networks (SPINN) is proposed. The authors suggest a sparse architecture, using kernel networks, that yields interpretable results while requiring fewer parameters than fully connected solutions. They consider various kernels such as Radial Basis Functions (RBF), softplus hat, or Gaussian kernels, and apply their proof of concept architecture to a variety of mathematical problems.

Activation Function

The activation function has a significant impact on DNN training performance. ReLU, Sigmoid, Tanh are commonly used activations [168]. Recent research has recommended training an adjustable activation function like Swish, which is defined as $x \cdot \text{Sigmoid}(\beta x)$ and β is a trainable parameter, and where Sigmoid is supposed to be a general sigmoid curve, an S-shaped function, or in some cases a logistic function. Among the most used activation functions there are logistic sigmoid, hyperbolic tangent, ReLu, and leaky ReLu. Most authors tend to use the infinitely differentiable hyperbolic tangent activation function $\alpha(x) = \tanh(x)$ [60], whereas Cheng and Zhang [31] use a Resnet block to improve the stability of the fully-connected neural network (FC–NN). They also prove that Swish activation function outperforms the others in terms of enhancing the neural network's convergence rate and accuracy. Nonetheless, because of the second order derivative evaluation, it is pivotal to choose the activation function in a PINN framework with caution. For example, while rescaling the PDE to dimensionless form, it is preferable to choose a range of [0, 1] rather than a wider domain, because most activation functions (such as Sigmoid, Tanh, Swish) are nonlinear near 0. Moreover the regularity of PINNs can be ensured by using smooth activation functions like as the sigmoid and hyperbolic tangent, allowing estimations of PINN generalization error to hold true [111].

2.1.2 Convolutional Neural Networks

Convolutional neural networks (ConvNet or CNN) are intended to process data in the form of several arrays, for example a color image made of three 2D arrays. CNNs usually have a number of convolution and pooling layers. The convolution layer is made up of a collection of filters (or kernels) that convolve across the full input rather than general matrix multiplication. The pooling layer instead performs subsampling, reducing the dimensionality.

For CNNs, according to Caterini and Chang [26], the layerwise function f can be written as

$$f_i(x_i; W_i) = \Phi_i(\alpha_i(\mathcal{C}_i(W_i, x_i)))$$

where α is an elementwise nonlinearity, Φ is the max-pooling map, and $\mathcal{C}$ the convolution operator.

It is worth noting that the convolution operation preserves translations and pooling is unaffected by minor data translations. This is applied to input image properties, such as corners, edges, and so on, that are translationally invariant, and will still be represented in the convolution layer's output.

As a result, CNNs perform well with multidimensional data such as images and speech signals, in fact is in the domain of images that these networks have been used in a physic informed network framework.

For more on these topic the reader can look at LeCun et al [92], Chen et al [29], Muhammad et al [117], Aldweesh et al [2], Berman et al [17], Calin [23]

CNN Architectures

Because CNNs were originally created for image recognition, they are better suited to handling image-like data and may not be directly applicable to scientific computing problems, as most geometries are irregular with non-uniform grids; for example, Euclidean-distance-based convolution filters lose their invariance on non-uniform meshes.

A physics-informed geometry-adaptive convolutional neural network (PhyGeoNet) was introduced in Gao et al [48]. PhyGeoNet is a physics-informed CNN that uses coordinate transformation to convert solution fields from an irregular physical domain to a rectangular reference domain. Additionally, boundary conditions are strictly enforced making it a physics-constrained neural network.

Fang [44] observes that a Laplacian operator has been discretized in a convolution operation employing a finite-difference stencil kernel. Indeed, a Laplacian operator can be discretized using the finite volume approach, and the discretization procedure is equivalent to convolution. As a result, he enhances PINN by using a finite volume numerical approach with a CNN structure. He devised and implemented a CNN-inspired technique in which, rather than using a Cartesian grid, he computes convolution on a mesh or graph. Furthermore, rather than zero padding, the padding data serve as the boundary condition. Finally, Fang [44] does not use pooling because the data does not require compression.

Convolutional Encoder–Decoder Network

Autoencoders (AE) (or encoder–decoders) are commonly used to reduce dimensionality in a nonlinear way. It consists of two NN components: an encoder that translates the data from the input layer to a finite number of hidden units, and a decoder that has an output layer with the same number of nodes as the input layer [29].

For modeling stochastic fluid flows, Zhu et al [200] developed a physics-constrained convolutional encoder–decoder network and a generative model. Zhu et al [200] propose a CNN-based technique for solving stochastic PDEs with high-dimensional spatially changing coefficients, demonstrating that it outperforms FC–NN methods in terms of processing efficiency.
AE architecture of the convolutional neural network (CNN) is also used in Wang et al [177]. The authors propose a framework called a Theory-guided Auto–Encoder (TgAE) capable of incorporating physical constraints into the convolutional neural network.

Geneva and Zabaras [51] propose a deep auto-regressive dense encoder–decoder and the physics-constrained training algorithm for predicting transient PDEs. They extend this model to a Bayesian framework to quantify both epistemic and aleatoric uncertainty. Finally, Grubišić et al [54] also used an encoder–decoder fully convolutional neural network.

2.1.3 Recurrent Neural Networks

Recurrent neural network (RNN) is a further ANN type, where unlike feed-forward NNs, neurons in the same hidden layer are connected to form a directed cycle. RNNs may accept sequential data as input, making them ideal for time-dependent tasks [29]. The RNN processes inputs one at a time, using the hidden unit output as extra input for the next element [17]. An RNN's hidden units can keep a state vector that holds a memory of previous occurrences in the sequence.

It is feasible to think of RNN in two ways: first, as a state system with the property that each state, except the first, yields an outcome; secondly, as a sequence of vanilla feedforward neural networks, each of which feeds information from one hidden layer to the next. For RNNs, according to Caterini and Chang [26], the layerwise function f can be written as

$$f_i(h_{i-1}) = \alpha(W \cdot h_{i-1} + U \cdot x_i + b).$$

where α is an elementwise nonlinearity (a typical choice for RNN is the tanh function), and where the hidden vector state h evolves according to a hidden-to-hidden weight matrix W, which starts from an input-to-hidden weight matrix U and a bias vector b.

RNNs have also been enhanced with several memory unit types, such as long short time memory (LSTM) and gated recurrent unit (GRU) [2]. Long short-term memory (LSTM) units have been created to allow RNNs to handle challenges requiring long-term memories, since LSTM units have a structure called a memory cell that stores information.
Each LSTM layer has a set of interacting units, or cells, similar to those found in a neural network. An LSTM is made up of four interacting units: an internal cell, an input gate, a forget gate, and an output gate. The cell state, controlled by the gates, can selectively propagate relevant information throughout the temporal sequence to capture the long short-term time dependency in a dynamical system [197].

The gated recurrent unit (GRU) is another RNN unit developed for extended memory; GRUs are comparable to LSTM, however they contain fewer parameters and are hence easier to train [17].

RNN Architectures

Viana et al [174] introduce, in the form of a neural network, a model discrepancy term into a given ordinary differential equation. Recurrent neural networks are seen as ideal for dynamical systems because they expand classic feedforward networks to incorporate time-dependent responses. Because a recurrent neural network applies transformations to given states in a sequence on a periodic basis, it is possible to design a recurrent neural network cell that does Euler integration; in fact, physics-informed recurrent neural networks can be used to perform numerical integration.

Viana et al [174] build recurrent neural network cells in such a way that specific numerical integration methods (e.g., Euler, Riemann, Runge–Kutta, etc.) are employed. The recurrent neural network is then represented as a directed graph, with nodes representing individual kernels of the physics-informed model. The graph created for the physics-informed model can be used to add data-driven nodes (such as multilayer perceptrons) to adjust the outputs of certain nodes in the graph, minimizing model discrepancy.

LSTM Architectures

Physicists have typically employed distinct LSTM networks to depict the sequence-to-sequence input-output relationship; however, these networks are not homogeneous and cannot be directly connected to one another. In Zhang et al [197] this relationship is expressed using a single network and a central finite difference filter-based numerical differentiator. Zhang et al [197] show two architectures for representation learning of sequence-to-sequence features from limited data that is augmented by physics models. The proposed networks is made up of two ($PhyLSTM^2$) or three ($PhyLSTM^3$) deep LSTM networks that describe state space variables, nonlinear restoring force, and hysteretic parameter. Finally, a tensor differentiator, which determines the derivative of state space variables, connects the LSTM networks.

Another approach is Yucesan and Viana [194] for temporal integration, that implement an LSTM using a previously introduced Euler integration cell.

2.1.4 Other Architectures for PINN

Apart from fully connected feed forward neural networks, convolutional neural networks, and recurrent neural networks, this section discusses other approaches that have been investigated. While there have been numerous other networks proposed in the literature, we discovered that only Bayesian neural networks (BNNs) and generative adversarial networks (GANs) have been applied to PINNs. Finally, an interesting application is to combine multiple PINNs, each with its own neural network.

Bayesian Neural Network

In the Bayesian framework, Yang et al [190] propose to use Bayesian neural networks (BNNs), in their B-PINNs, that consists of a Bayesian neural network subject to the PDE constraint that acts as a prior. BNN are neural networks with weights that are distributions rather than deterministic values, and these distributions are learned using Bayesian inference. For estimating the posterior distributions, the B-PINN authors use the Hamiltonian Monte Carlo (HMC) method and the variational inference (VI). Yang et al [190] find that for the posterior estimate of B-PINNs, HMC is more appropriate than VI with mean field Gaussian approximation.

They analyse also the possibility to use the Karhunen–Loève expansion as a stochastic process representation, instead of BNN. Although the KL is as accurate as BNN and considerably quicker, it cannot be easily applied to high-dimensional situations. Finally, they observe that to estimate the posterior of a Bayesian framework, KL–HMC or deep normalizing flow (DNF) models can be employed. While DNF is more computationally expensive than HMC, it is more capable of extracting independent samples from the target distribution after training. This might be useful for data-driven PDE solutions, however it is only applicable to low-dimensional problems.

GAN Architectures

In generative adversarial networks (GANs), two neural networks compete in a zero-sum game to deceive each other. One network generates and the other discriminates. The generator accepts input data and outputs data with realistic characteristics. The discriminator compares the real input data to the output of the generator. After training, the generator can generate new data that is indistinguishable from real data [17].

Yang et al [189] propose a new class of generative adversarial networks (PI–GANs) to address forward, inverse, and mixed stochastic problems in a unified manner. Unlike typical GANs, which rely purely on data for training, PI–GANs use automatic differentiation to embed the governing physical laws in the form of stochastic differential equations (SDEs) into the architecture of PINNs. The discriminator in PI–GAN is represented by a basic FFNN, while the generators are a combination of FFNNs and a NN induced by the SDE.

Multiple PINNs

A final possibility is to combine several PINNs, each of which could be implemented using a different neural network. Jagtap et al [71] propose a conservative physics-informed neural network (cPINN) on discrete domains. In this framework, the complete solution is recreated by patching together all of the solutions in each sub-domain using the appropriate interface conditions. This type of domain segmentation also allows for easy network parallelization, which is critical for obtaining computing efficiency. This method may be expanded to a more general situation, called by the authors as Mortar PINN, for connecting non-overlapping deconstructed domains. Moreover, the suggested technique may use totally distinct neural networks, in each subdomain, with various architectures to solve the same underlying PDE. Stiller et al [167] proposes the GatedPINN architecture by incorporating conditional computation into PINN. This architecture design is composed of a gating network and set of PINNs, hereinafter referred to as “experts”; each expert solves the problem for each space-time point, and their results are integrated via a gating network. The gating network determines which expert should be used and how to combine them. We will use one of the expert networks as an example in the following section of this review.

2.2 Injection of Physical Laws

To solve a PDE with PINNs, derivatives of the network’s output with respect to the inputs are needed. Since the function u is approximated by a NN with smooth activation function, $\hat{u}_\theta$, it can be differentiated. There are four methods for calculating derivatives: hand-coded, symbolic, numerical, and automatic. Manually calculating derivatives may be correct, but it is not automated and thus impractical [175].

Symbolic and numerical methods like finite differentiation perform very badly when applied to complex functions; automatic differentiation (AD), on the other hand, overcomes numerous restrictions as floating-point precision errors, for numerical differentiation, or

Table 1 The main neural network utilized in PINN implementations is synthesized in this table

NN family	NN type	Papers
FF–NN	1 layer / EML	Dwivedi and Srinivasan [41], Schiassi et al [153]
	2–4 layers	32 neurons per layer He et al [60] 50 neurons per layer Tartakovsky et al [170]
	5–8 layers	250 neurons per layer Zhu et al [199]
	9+ layers	Cheng and Zhang [31] Waheed et al [175]
	Sparse	Ramabathiran and Ramachandran [148]
	multi FC-DNN	Amini Niaki et al [6] Islam et al [69]
CNN	plain CNN	Gao et al [48] Fang [44]
	AE CNN	Zhu et al [200], Geneva and Zabaras [51] Wang et al [177]
RNN	RNN	Viana et al [174]
	LSTM	Zhang et al [197] Yucesan and Viana [194]
Other	BNN	Yang et al [190]
	GAN	Yang et al [189]

We summarize Sect. 2 by showcasing some of the papers that represent each of the many Neural Network implementations of PINN. Feedforward neural networks (FFNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNN) are the three major families of Neural Networks reported here. A publication is reported for each type that either used this type of network first or best describes its implementation. In the literature, PINNs have mostly been implemented with FFNNs with 5–10 layers. CNN appears to have been applied in a PCNN manner for the first time, by incorporating the boundary condition into the neural network structure rather than the loss memory intensive symbolic approach. AD use exact expressions with floating-point values rather than symbolic strings, there is no approximation error [175].

Automatic differentiation (AD) is also known as autodiff, or algorithmic differentiation, although it would be better to call it algorithmic differentiation since AD does not totally automate differentiation: instead of symbolically evaluating derivatives operations, AD performs an analytical evaluation of them.

Considering a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ of which we want to calculate the Jacobian J, after calculating the graph of all the operations composing the mathematical expression, AD can then work in either forward or reverse mode for calculating the numerical derivative.

AD results being the main technique used in literature and used by all PINN implementations, in particular only Fang [44] use local fitting method approximation of the differential operator to solve the PDEs instead of automatic differentiation (AD). Moreover, by using local fitting method rather than employing automated differentiation, Fang is able to verify that a his PINN implementation has a convergence.

Essentially, AD incorporates a PDE into the neural network's loss Eq. (2), where the differential equation residual is

$$r_{\mathcal{F}}[\hat{u}_\theta](z) = r_\theta(z) := \mathcal{F}(\hat{u}_\theta(z); \gamma) - f,$$

and similarly the residual NN corresponding to boundary conditions (BC) or initial conditions (IC) is obtained by substituting $\hat{u}_\theta$ in the second equation of (1), i.e.

$$r_{\mathcal{B}}[\hat{u}_\theta](z) := \mathcal{B}(\hat{u}_\theta(z)) - g(z).$$

Using these residuals, it is possible to assess how well an approximation u_θ satisfies (1). It is worth noting that for the exact solution, u, the residuals are $r_{\mathcal{F}}[u] = r_{\mathcal{B}}[u] = 0$ [38].

In Raissi and Karniadakis [140], Raissi et al [146], the original formulation of the aforementioned differential equation residual, leads to the form of

$$r_{\mathcal{F}}[\hat{u}_\theta](z) = r_\theta(x,t) = \frac{\partial}{\partial t}\hat{u}_\theta(x,t) + \mathcal{F}_x \hat{u}_\theta(x,t).$$

In the deep learning framework, the principle of imposing physical constraints is represented by differentiating neural networks with respect to input spatiotemporal coordinates using the chain rule. In Mathews et al [106] the model loss functions are embedded and then further normalized into dimensionless form. The repeated differentiation, with AD, and composition of networks used to create each individual term in the partial differential equations results in a much larger resultant computational graph. As a result, the cumulative computation graph is effectively an approximation of the PDE equations [106].

The chain rule is used in automatic differentiation for several layers to compute derivatives hierarchically from the output layer to the input layer. Nonetheless, there are some situations in which the basic chain rule does not apply. Pang et al [128] substitute fractional differential operators with their discrete versions, which are subsequently incorporated into the PINNs' loss function.

2.3 Model Estimation by Learning Approaches

The PINN methodology determines the parameters θ of the NN, $\hat{u}_\theta$, by minimizing a loss function, i.e.

$$\theta = \arg\min_\theta \mathcal{L}(\theta)$$

where

$$\mathcal{L}(\theta) = \omega_{\mathcal{F}} \mathcal{L}_{\mathcal{F}}(\theta) + \omega_{\mathcal{B}} \mathcal{L}_{\mathcal{B}}(\theta) + \omega_d \mathcal{L}_{data}(\theta). \tag{6}$$

The three terms of $\mathcal{L}$ refer to the errors in describing the initial $\mathcal{L}_i$ or boundary condition $\mathcal{L}_b$, both indicated as $\mathcal{L}_{\mathcal{B}}$, the loss respect the partial differential equation $\mathcal{L}_{\mathcal{F}}$, and the validation of known data points $\mathcal{L}_{data}$. Losses are usually defined in the literature as a sum, similar to the previous equations, however they can be considered as integrals

$$\mathcal{L}_{\mathcal{F}}(\theta) = \int_{\bar{\Omega}} \left(\mathcal{F}(\hat{u}_\theta(z)) - f(z_i)\right)^2 \, dz$$

This formulation is not only useful for a theoretical study, as we will see in 2.4, but it is also implemented in a PINN package, NVIDIA Modulus [123], allowing for more effective integration strategies, such as sampling with higher frequency in specific areas of the domain to more efficiently approximate the integral losses.

Note that, if the PINN framework is employed as a supervised methodology, the neural network parameters are chosen by minimizing the difference between the observed outputs and the model's predictions; otherwise, just the PDE residuals are taken into account.

As in Eq. (6) the first term, $\mathcal{L}_{\mathcal{F}}$, represents the loss produced by a mismatch with the governing differential equations $\mathcal{F}$ [60, 167]. It enforces the differential equation $\mathcal{F}$ at the *collocation points*, which can be chosen uniformly or unevenly over the domain Ω of Eq. (1).

The remaining two losses attempt to fit the known data over the NN. The loss caused by a mismatch with the data (i.e., the measurements of u) is denoted by $\mathcal{L}_{data}(\theta)$. The second term typically forces $\hat{u}_\theta$ to mach the measurements of u over provided points (z, u^*), which can be given as synthetic data or actual measurements, and the weight ω_d can account for the quality of such measurements.

The other term is the loss due to mismatch with the boundary or initial conditions, $\mathcal{B}(\hat{u}_\theta) = g$ from Eq. (1).

Essentially, the training approach recovers the shared network parameters θ from:

- few scattered observations of $u(z)$, specifically $\{z_i, u_i^*\}$, $i = 1, \ldots, N_d$
- as well as a greater number of collocation points $\{z_i, r_i = 0\}$, $i = 1, \ldots, N_r$ for the residual,

The resulting optimization problem can be handled using normal stochastic gradient descent without the need for constrained optimization approaches by minimizing the combined loss function. A typical implementation of the loss uses a mean square error formulation [82], where:

$$\mathcal{L}_{\mathcal{F}}(\theta) = MSE_{\mathcal{F}} = \frac{1}{N_c} \sum_{i=1}^{N_c} \|\mathcal{F}(\hat{u}_\theta(z_i)) - f(z_i))\|^2 = \frac{1}{N_c} \sum_{i=1}^{N_c} \|r_\theta(u_i) - r_i\|^2$$

enforces the PDE on a wide set of randomly selected collocation locations inside the domain, i.e. penalizes the difference between the estimated left-hand side of a PDE and the known right-hand side of a PDE [82]; other approaches may employ an integral definition of the loss [61]. As for the boundary and initial conditions, instead

$$\mathcal{L}_{\mathcal{B}}(\theta) = MSE_{\mathcal{B}} = \frac{1}{N_B} \sum_{i=1}^{N_B} \|\mathcal{B}(\hat{u}_\theta(z)) - g(z_i))\|^2$$

whereas for the data points,

$$\mathcal{L}_{data}(\theta) = MSE_{data} = \frac{1}{N_d} \sum_{i=1}^{N_d} \|\hat{u}_\theta(z_i) - u_i^*\|^2.$$

computes the error of the approximation $u(t, x)$ at known data points. In the case of a forward problem, the data loss might also indicate the boundary and initial conditions, while in an inverse problem it refers to the solution at various places inside the domain [82].

In Raissi and Karniadakis [140], Raissi et al [146], original approach the overall loss (6) was formulated as

$$\mathcal{L}(\theta) = \frac{1}{N_c} \sum_{i=1}^{N_c} \| \frac{\partial}{\partial t} \hat{u}_\theta(x, t) + \mathcal{F}_x \hat{u}_\theta(x, t) - r_i \|^2 + \frac{1}{N_d} \sum_{i=1}^{N_d} \|\hat{u}_\theta(x_i, t_i) - u_i^*\|^2.$$

The gradients in $\mathcal{F}$ are derived using automated differentiation. The resulting predictions are thus driven to inherit any physical attributes imposed by the PDE constraint [191]. The physics constraints are included in the loss function to enforce model training, which can accurately reflect latent system nonlinearity even when training data points are scarce [197].

Observations About the Loss

The loss $\mathcal{L}_\mathcal{F}(\theta)$ is calculated by utilizing automated differentiation (AD) to compute the derivatives of $\hat{u}_\theta(z)$ [60]. Most ML libraries, including TensorFlow and Pytorch, provide AD, which is mostly used to compute derivatives with respect to DNN weights (i.e. θ). AD permits the PINN approach to implement any PDE and boundary condition requirements without numerically discretizing and solving the PDE [60].

Additionally, by applying PDE constraints via the penalty term $\mathcal{L}_\mathcal{F}(\theta)$, it is possible to use the related weight $\omega_\mathcal{F}$ to account for the PDE model's fidelity. To a low-fidelity PDE model, for example, can be given a lower weight. In general, the number of unknown parameters in θ is substantially greater than the number of measurements, therefore regularization is required for DNN training [60].

By removing loss for equations from the optimization process (i.e., setting $\omega_\mathcal{F} = 0$), neural networks could be trained without any knowledge of the underlying governing equations. Alternatively, supplying initial and boundary conditions for all dynamical variables would correspond to solving the equations directly with neural networks on a regular basis [106].

While it is preferable to enforce the physics model across the entire domain, the computational cost of estimating and reducing the loss function (6), while training, grows with the number of residual points [60]. Apart the number of residual points, also the position (distribution) of residual points are crucial parameters in PINNs because they can change the design of the loss function [103].

A deep neural network can reduce approximation error by increasing network expressivity, but it can also produce a large generalization error. Other hyperparameters, such as learning rate, number of iterations, and so on, can be adjusted to further control and improve this issue.

The addition of extra parameters layer by layer in a NN modifies the slope of the activation function in each hidden-layer, improving the training speed. Through the slope recovery term, these activation slopes can also contribute to the loss function [71].

Soft and Hard Constraint

BC constraints can be regarded as penalty terms (soft BC enforcement) [200], or they can be encoded into the network design (hard BC enforcement) [168]. Many existing PINN frameworks use a *soft* approach to constrain the BCs by creating extra loss components defined on the collocation points of borders. The disadvantages of this technique are twofold:

1. satisfying the BCs accurately is not guaranteed;
2. the assigned weight of BC loss might effect learning efficiency, and no theory exists to guide determining the weights at this time.

Zhu et al [199] address the Dirichlet BC in a *hard* approach by employing a specific component of the neural network to purely meet the specified Dirichlet BC. Therefore, the initial boundary conditions are regarded as part of the labeled data constraint.

When compared to the residual-based loss functions typically found in the literature, the variational energy-based loss function is simpler to minimize and so performs better [52]. Loss function can be constructed using collocation points, weighted residuals derived by the Galerkin–Method [76], or energy based. Alternative loss functions approaches are compared in Li et al [94], by using either only data-driven (with no physics model), a PDE-based loss, and an energy-based loss. They observe that there are advantages and disadvantages for both PDE-based and energy-based approaches. PDE-based loss function has more hyperparameters than the energy-based loss function. The energy-based strategy is more sensitive to the size and resolution of the training samples than the PDE-based strategy, but it is more computationally efficient.

Optimization Methods

The minimization process of the loss function is called *training*; in most of the PINN literature, loss functions are optimized using minibatch sampling using Adam and the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm, a quasi-Newton optimization algorithm. When monitoring noisy data, Mathews et al [106] found that increasing the sample size and training only with L-BFGS achieved the optimum for learning.

For a moderately sized NN, such as one with four hidden layers (depth of the NN is 5) and twenty neurons in each layer (width of the NN is 20), we have over 1000 parameters to optimize. There are several local minima for the loss function, and the gradient-based optimizer will almost certainly become caught in one of them; finding global minima is an NP-hard problem [128].

The Adam approach, which combines adaptive learning rate and momentum methods, is employed in Zhu et al [199] to increase convergence speed, because stochastic gradient descent (SGD) hardly manages random collocation points, especially in 3D setup.

Yang et al [189] use Wasserstein GANs with gradient penalty (WGAN–GP) and prove that they are more stable than vanilla GANs, in particular for approximating stochastic processes with deterministic boundary conditions.

cPINN [71] allow to flexibly select network hyper-parameters such as optimization technique, activation function, network depth, or network width based on intuitive knowledge of solution regularity in each sub-domain. E.g. for smooth zones, a shallow network may be used, and a deep neural network can be used in an area where a complex nature is assumed.

He et al [60] propose a two-step training approach in which the loss function is minimized first by the Adam algorithm with a predefined stop condition, then by the L-BFGS-B optimizer. According to the aforementioned paper, for cases with a little amount of training data and/or residual points, L-BFGS-B, performs better with a faster rate of convergence and reduced computing cost.

Finally, let's look at a practical examples for optimizing the training process; dimensionless and normalized data are used in DeepONet training to improve stability [96]. Moreover the governing equations in dimensionless form, including the Stokes equations, electric potential, and ion transport equations, are presented in DeepM&Mnets [21].

In terms of training procedure initialization, slow and fast convergence behaviors are produced by bad and good initialization, respectively, but Pang et al [128] reports a technique for selecting the most suitable one. By using a limited number of iterations, one can first solve the inverse problem. This preliminary solution has low accuracy due to discretization, sampling, and optimization errors. The optimized parameters from the low-fidelity problem can then be used as a suitable initialization.

2.4 Learning Theory of PINN

This final subsection provides most recent theoretical studies on PINN to better understand how they work and their potential limits. These investigations are still in their early stages, and much work remains to be done.

Let us start by looking at how PINN can approximate the true solution of a differential equation, similar to how error analysis is done a computational framework. In traditional numerical analysis, we approximate the true solution $u(z)$ of a problem with an approximation scheme that computes $\hat{u}_\theta(z)$. The main theoretical issue is to estimate the global error

$$\mathcal{E} = \hat{u}_\theta(z) - u(z).$$

Ideally, we want to find a set of parameters, θ, such that $\mathcal{E} = 0$.

When solving differential equations using a numerical discretization technique, we are interested in the numerical method's stability, consistency, and convergence [8, 150, 171]. In such setting, discretization's error can be bound in terms of consistency and stability, a basic result in numerical analysis. The Lax–Richtmyer Equivalence Theorem is often referred to as a fundamental result of numerical analysis where roughly the convergence is ensured when there is consistency and stability.

When studying PINN to mimic this paradigm, the convergence and stability are related to how well the NN learns from physical laws and data. In this conceptual framework, we use a NN, which is a parameterized approximation of problem solutions modeled by physical laws. In this context, we will (i) introduce the concept of convergence for PINNs, (ii) revisit the main error analysis definitions in a statistical learning framework, and (iii) finally report results for the generalization error.

2.4.1 Convergence Aspects

The goal of a mathematical foundation for the PINN theory is to investigate the convergence of the computed $\hat{u}_\theta(z)$ to the solution of problem (1), $u(z)$.

Consider a NN configuration with coefficients compounded in the vector θ and a cardinality equal to the number of coefficients of the NN, $\#\theta$. In such setting, we can consider the hypothesis class

$$\mathcal{H}_n = \{u_\theta : \#\theta = n\}$$

composed of all the predictors representing a NN whose number of coefficients of the architecture is n. The capacity of a PINN to be able to learn, is related to how big is n, i.e. the expressivity of $\mathcal{H}_n$.

In such setting, a theoretical issue, is to investigate, how dose the sequence of compute predictors, $\hat{u}_\theta$, converges to the solution of the physical problem (1)

$$\hat{u}_{\theta(n)} \to u, \qquad n \to \infty.$$

A recent result in this direction was obtained by De Ryck et al [37] in which they proved that the difference $\hat{u}_{\theta(n)} - u$ converges to zero as the width of a predefined NN, with activation function tanh, goes to infinity.

Practically, the PINN requires choosing a network class $\mathcal{H}_n$ and a loss function given a collection of N-training data [157]. Since the quantity and quality of training data affect $\mathcal{H}_n$, the goal is to minimize the loss, by finding a $u_{\theta^*} \in \mathcal{H}_n$, by training the N using an optimization process. Even if $\mathcal{H}_n$ includes the exact solution u to PDEs and a global minimizer is established, there is no guarantee that the minimizer and the solution u will coincide. A first work related on PINN [157], the authors show that the sequence of minimizers $\hat{u}_{\theta^*}$ strongly converges to the solution of a linear second-order elliptic and parabolic type PDE.

2.4.2 Statistical Learning Error Analysis

The entire learning process of a PINN can be considered as a statistical learning problem, and it involves mathematical foundations aspects for the error analysis [88]. For a mathematical treatment of errors in PINN, it is important to take into account: optimization, generalization errors, and approximation error. It is worth noting that the last one is dependent on the architectural design.

Let be N collocation points on $\bar{\Omega} = \Omega \cup \partial\Omega$, a NN approximation realized with θ, denoted by $\hat{u}_\theta$, evaluated at points z_i, whose exact value is h_i. Following the notation of Kutyniok [88], the *empirical risk* is defined as

$$\widehat{\mathcal{R}}[u_\theta] := \frac{1}{N} \sum_{i=1}^{N} \|\hat{u}_\theta(z_i) - h_i, \|^2 \tag{7}$$

and represents how well the NN is able to predict the exact value of the problem. The empirical risk actually corresponds to the loss defined in 2.3, where $\hat{u}_\theta = \mathcal{F}(\hat{u}_\theta(z_i))$ and $h_i = f(z_i)$ and similarly for the boundaries.

A continuum perspective is the *risk* of using an approximator $\hat{u}_\theta$, calculated as follows:

$$\mathcal{R}[\hat{u}_\theta] := \int_{\bar{\Omega}} (\hat{u}_\theta(z) - u(z))^2 \, dz, \tag{8}$$

where the distance between the approximation $\hat{u}_\theta$ and the solution u is obtained with the L^2-norm. The final approximation computed by the PINN, after a training process of DNN, is $\hat{u}_\theta^*$. The main aim in error analysis, is to find suitable estimate for the risk of predicting u i.e. $\mathcal{R}[\hat{u}_\theta^*]$.

The training process, uses gradient-based optimization techniques to minimize a generally non convex cost function. In practice, the algorithmic optimization scheme will not always find a global minimum. So the error analysis takes into account the *optimization error* defined as follows:

$$\mathcal{E}_O := \widehat{\mathcal{R}}[\hat{u}_\theta^*] - \inf_{\theta \in \Theta} \widehat{\mathcal{R}}[u_\theta]$$

Because the objective function is nonconvex, the optimization error is unknown. Optimization frequently involves a variety of engineering methods and time-consuming fine-tuning, using, gradient-based optimization methods. Several stochastic gradient descent methods have been proposed,and many PINN use Adam and L-BFGS. Empirical evidence suggests that gradient-based optimization techniques perform well in different challenging tasks; however, gradient-based optimization might not find a global minimum for many ML tasks, such as for PINN, and this is still an open problem [157].

Moreover a measure of the prediction accuracy on unseen data in machine learning is the *generalization error*:

$$\mathcal{E}_G := \sup_{\theta \in \Theta} |\mathcal{R}[u_\theta] - \widehat{\mathcal{R}}[u_\theta]|$$

The generalization error measures how well the loss integral is approximated in relation to a specific trained neural network. One of the first paper focused with convergence of generalization error is Shin et al [157].

About the ability of the NN to approximate the exact solution, the *approximation error* is defined as

$$\mathcal{E}_A := \inf_{\theta \in \Theta} \mathcal{R}[u_\theta]$$

The approximation error is well studied in general, in fact we know that one layer neural network with a high width can evenly estimate a function and its partial derivative as shown by Pinkus [133].

Finally, as stated in Kutyniok [88], the global error between the trained deep neural network $\hat{u}_\theta^*$ and the correct solution function u of problem (1), can so be bounded by the previously defined error in the following way

$$\mathcal{R}[\hat{u}_\theta^*] \leq \mathcal{E}_O + 2\mathcal{E}_G + \mathcal{E}_A \tag{9}$$

These considerations lead to the major research threads addressed in recent studies, which are currently being investigated for PINN and DNNs in general Kutyniok [88].

2.4.3 Error Analysis Results for PINN

About the approximating error, since it depends on the NN architecture, mathematical foundations results are generally discussed in papers deeply focused on this topic Calin [24], Elbrächter et al [43].

However, a first argumentation strictly related to PINN is reported in Shin et al [158]. One of the main theoretical results on $\mathcal{E}_A$, can be found in De Ryck et al [37]. They demonstrate that for a neural network with a tanh activation function and only two hidden layers, $\hat{u}_\theta$, may approximate a function u with a bound in a Sobolev space:

$$\|\hat{u}_{\theta_N} - u\|_{W^{k,\infty}} \leq C \frac{\ln(cN)^k}{N^{s-k}}$$

where N is the number of training points, $c, C > 0$ are constants independent of N and explicitly known, $u \in W^{s,\infty}([0,1]^d)$. We remark that the NN has width N^d, and $\#\theta$ depends on both the number of training points N and the dimension of the problem d.

Formal findings for generalization errors in PINN are provided specifically for a certain class of PDE. In Shin et al [157] they provide convergence estimate for linear second-order elliptic and parabolic type PDEs, while in Shin et al [158] they extend the results to all linear problems, including hyperbolic equations. Mishra and Molinaro [113] gives an abstract framework for PINN on forward problem for PDEs, they estimate the generalization error by means of training error (empirical risk), and number of training points, such abstract framework is also addressed for inverse problems [111]. In De Ryck et al [38] the authors specifically address Navier–Stokes equations and show that small training error imply a small generalization error, by proving that

$$\mathcal{R}[\hat{u}_\theta] = \|u - \hat{u}_\theta\|_{L^2} \le \left(C\widehat{\mathcal{R}}[u_\theta] + \mathcal{O}\left(N^{-\frac{1}{d}}\right)\right)^{\frac{1}{2}}.$$

This estimate suffer from the curse of dimensionality (CoD), that is to say, in order to reduce the error by a certain factor, the number of training points needed and the size of the neural network, scales up exponentially.

Finally, a recent work [18] has proposed explicit error estimates and stability analyses for incompressible Navier–Stokes equations.

De Ryck and Mishra [36] prove that for a Kolmogorov type PDE (i.e. heat equation or Black–Scholes equation), the following inequality holds, almost always,

$$\mathcal{R}[\hat{u}_\theta] \le \left(C\widehat{\mathcal{R}}[u_\theta] + \mathcal{O}\left(N^{-\frac{1}{2}}\right)\right)^{\frac{1}{2}},$$

and is not dependant on the dimension of the problem d.

Finally, Mishra and Molinaro [112] investigates the radiative transfer equation, which is noteworthy for its high-dimensionality, with the radiative intensity being a function of 7 variables (instead of 3, as common in many physical problems). The authors prove also here that the generalization error is bounded by the training error and the number of training points, and the dimensional dependence is on a logarithmic factor:

$$\mathcal{R}[\hat{u}_\theta] \le \left(C\widehat{\mathcal{R}}[u_\theta]^2 + c\left(\frac{(\ln N)^{2d}}{N}\right)\right)^{\frac{1}{2}}.$$

The authors are able to show that PINN does not suffer from the dimensionality curse for this problem, observing that the training error does not depend on the dimension but only on the number of training points.

3 Differential Problems Dealt with PINNs

The first vanilla PINN [146] was built to solve complex nonlinear PDE equations of the form $u_t + \mathcal{F}_x u = 0$, where x is a vector of space coordinates, t is a vector time coordinate, and $\mathcal{F}_x$ is a nonlinear differential operator with respect to spatial coordinates. First and mainly, the PINN architecture was shown to be capable of handling both forward and inverse problems. Eventually, in the years ahead, PINNs have been employed to solve a wide variety of ordinary differential equations (ODEs), partial differential equations (PDEs), Fractional PDEs, and integro-differential equations (IDEs), as well as stochastic differential equations (SDEs). This section is dedicated to illustrate where research has progressed in addressing various sorts of equations, by grouping equations according to their form and addressing the primary work in literature that employed PINN to solve such equation. All PINN papers dealing with ODEs will be presented first. Then, works on steady-state PDEs such as Elliptic type equations, steady-state diffusion, and the Eikonal equation are reported. The Navier–Stokes equations are followed by more dynamical problems such as heat transport, advection-diffusion-reaction system, hyperbolic equations, and Euler equations or quantum harmonic oscillator. Finally, while all of the previous PDEs can be addressed in their respective Bayesian problems, the final section provides insight into how uncertainly is addressed, as in stochastic equations.

3.1 Ordinary Differential Equations

ODEs can be used to simulate complex nonlinear systems which are difficult to model using simply physics-based models [91]. A typical ODE system is written as

$$\frac{du(x,t)}{dt} = f(u(x,t),t)$$

where initial conditions can be specified as $\mathcal{B}(u(t)) = g(t)$, resulting in an initial value problem or boundary value problem with $\mathcal{B}(u(x)) = g(x)$. A PINN approach is used by Lai et al [91] for structural identification, using Neural Ordinary Differential Equations (Neural ODEs). Neural ODEs can be considered as a continuous representation of ResNets (Residual Networks), by using a neural network to parameterize a dynamical system in the form of ODE for an initial value problem (IVP):

$$\frac{du(t)}{dt} = f_\theta(u(t),t);\, u(t_0) = u_0$$

where f is the neural network parameterized by the vector θ.

The idea is to use Neural ODEs as learners of the governing dynamics of the systems, and so to structure of Neural ODEs into two parts: a physics-informed term and an unknown discrepancy term. The framework is tested using a spring-mass model as a 4-degree-of-freedom dynamical system with cubic nonlinearity, with also noisy measured data. Furthermore, they use experimental data to learn the governing dynamics of a structure equipped with a negative stiffness device [91].

Zhang et al [197] employ deep long short-term memory (LSTM) networks in the PINN approach to solve nonlinear structural system subjected to seismic excitation, like steel moment resistant frame and the single degree-of-freedom Bouc–Wen model, a nonlinear system with rate-dependent hysteresis [197]. In general they tried to address the problems of nonlinear equation of motion :

$$\ddot{\mathbf{u}} + \mathbf{g} = -\boldsymbol{\Gamma} a_g$$

where $\mathbf{g}(t) = \mathbf{M}^{-1}\mathbf{h}(t)$ denotes the mass-normalized restoring force, being $\mathbf{M}$ the mass matrices; $\mathbf{h}$ the total nonlinear restoring force, and $\boldsymbol{\Gamma}$ force distribution vector.

Directed graph models can be used to directly implement ODE as deep neural networks [174], while using an Euler RNN for numerical integration.

In Nascimento et al [121] is presented a tutorial on how to use Python to implement the integration of ODEs using recurrent neural networks.

ODE-net idea is used in Tong et al [172] for creating Symplectic Taylor neural networks. These NNs consist of two sub-networks, that use symplectic integrators instead of Runge-Kutta, as done originally in ODE-net, which are based on residual blocks calculated with the Euler method. Hamiltonian systems as Lotka–Volterra, Kepler, and Hénon–Heiles systems are also tested in the aforementioned paper.

3.2 Partial Differential Equations

Partial Differential Equations are the building bricks of a large part of models that are used to mathematically describe physics phenomenologies. Such models have been deeply investigated and often solved with the help of different numerical strategies. Stability and convergence of these strategies have been deeply investigated in literature, providing a solid theoretical framework to approximately solve differential problems. In this Section, the application of the novel methodology of PINNs on different typologies of Partial Differential models is explored.

3.2.1 Steady State PDEs

In Kharazmi et al [76, 78], a general steady state problem is addressed as:

$$\mathcal{F}_s(u(x); q) = f(x) \quad x \in \Omega,$$
$$\mathcal{B}(u(x)) = 0 \quad x \in \partial\Omega$$

over the domain $\Omega \subset \mathbb{R}^d$ with dimensions d and bounds $\partial\Omega$. $\mathcal{F}_s$ typically contains differential and/or integro-differential operators with parameters q and $f(x)$ indicates some forcing term.

In particular an Elliptic equation can generally be written by setting

$$\mathcal{F}_s(u(x); \sigma, \mu) = -div(\mu\nabla u)) + \sigma u$$

Tartakovsky et al [170] consider a linear

$$\mathcal{F}_s(u(x); \sigma) = \nabla \cdot (K(\mathbf{x})\nabla u(\mathbf{x})) = 0$$

and non linear

$$\mathcal{F}_s(u(x); \sigma) = \nabla \cdot [K(u)\nabla u(\mathbf{x})] = 0$$

diffusion equation with unknown diffusion coefficient K. The equation essentially describes an unsaturated flow in a homogeneous porous medium, where u is the water pressure and $K(u)$ is the porous medium's conductivity. It is difficult to measure $K(u)$ directly, so Tartakovsky et al [170] assume that only a finite number of measurements of u are available.

Tartakovsky et al [170] demonstrate that the PINN method outperforms the state-of-the-art maximum a posteriori probability method. Moreover, they show that utilizing only capillary pressure data for unsaturated flow, PINNs can estimate the pressure-conductivity for unsaturated flow. One of the first novel approach, PINN based, was the variational physics-informed neural network (VPINN) introduced in Kharazmi et al [76], which has the advantage of decreasing the order of the differential operator through integration-by-parts. The authors tested VPINN with the steady Burgers equation, and on the two dimensional Poisson's equation. VPINN Kharazmi et al [76] is also used to solve Schrödinger Hamiltonians, i.e. an elliptic reaction-diffusion operator [54].

In Haghighat et al [56] a nonlocal approach with the PINN framework is used to solve two-dimensional quasi-static mechanics for linear-elastic and elastoplastic deformation. They define a loss function for elastoplasticity, and the input variables to the feed-forward neural network are the displacements, while the output variables are the components of the strain tensor and stress tensor. The localized deformation and strong gradients in the solution make the boundary value problem difficult solve. The Peridynamic Differential Operator (PDDO) is used in a nonlocal approach with the PINN paradigm in Haghighat et al [56].

They demonstrated that the PDDO framework can capture stress and strain concentrations using global functions.

In Dwivedi and Srinivasan [41] the authors address different 1D-2D linear advection and/or diffusion steady-state problems from Berg and Nyström [16], by using their PIELM, a PINN combined with ELM (Extreme Learning Machine). A critical point is that the proposed PIELM only takes into account linear differential operators.

In Ramabathiran and Ramachandran [148] they consider linear elliptic PDEs, such as the solution of the Poisson equation in both regular and irregular domains, by addressing non-smoothness in solutions.

The authors in Ramabathiran and Ramachandran [148] propose a class of partially interpretable sparse neural network architectures (SPINN), and this architecture is achieved by reinterpreting meshless representation of PDE solutions.

Laplace–Beltrami Equation is solved on 3D surfaces, like complex geometries, and high dimensional surfaces, by discussing the relationship between sample size, the structure of the PINN, and accuracy [46].

The PINN paradigm has also been applied to Eikonal equations, i.e. are hyperbolic problems written as

$$\|\nabla u(x)\|^2 = \frac{1}{v^2(x)}, \qquad \forall x \in \Omega,$$

where v is a velocity and u an unknown activation time. The Eikonal equation describes wave propagation, like the travel time of seismic wave [162, 175] or cardiac activation electrical waves [53, 151].

By implementing EikoNet, for solving a 3D Eikonal equation, Smith et al [162] find the travel-time field in heterogeneous 3D structures; however, the proposed PINN model is only valid for a single fixed velocity model, hence changing the velocity, even slightly, requires retraining the neural network. EikoNet essentially predicts the time required to go from a source location to a receiver location, and it has a wide range of applications, like earthquake detection.

PINN is also proved to outperform the first-order fast sweeping solution in accuracy tests [175], especially in the anisotropic model.

Another approach involves synthetic and patient data for learning heart tissue fiber orientations from electroanatomical maps, modeled with anisotropic Eikonal equation [53]. In their implementation the authors add to the loss function also a Total Variation regularization for the conductivity vector.

By neglecting anisotropy, cardiac activation mapping is also addressed by Sahli Costabal et al [151] where PINNs are used with randomized prior functions to quantify data uncertainty and create an adaptive sampling strategy for acquiring and creating activation maps.

Helmholtz equation for weakly inhomogeneous two-dimensional (2D) media under transverse magnetic polarization excitation is addressed in Chen et al [30] as:

$$\nabla^2 E_z(x, y) + \varepsilon_r(x, y) k_0^2 E_z = 0,$$

whereas high frequency Helmholtz equation (frequency domain Maxwell's equation) is solved in Fang and Zhan [45].

3.2.2 Unsteady PDEs

Unsteady PDEs usually describe the evolution in time of a physics phenomena. Also in this case, PINNs have proven their reliability in solving such type of problems resulting in a flexible methodology.

3.2.2.1 Advection–Diffusion–Reaction Problems Originally Raissi et al [146] addressed unsteady state problem as:

$$\begin{aligned} u_t &= \mathcal{F}_x(u(x)) \quad x \in \Omega, \\ \mathcal{B}(u(x)) &= 0 \quad x \in \partial\Omega \end{aligned}$$

where $\mathcal{F}_x$ typically contains differential operators of the variable x. In particular a general advection-diffusion reaction equation can be written by setting

$$\mathcal{F}_x(u(x); b, \mu, \sigma) = -div(\mu \nabla u)) + b\nabla u + \sigma u,$$

where, given the parameters b, μ, σ, $-div(\mu\nabla u))$ is the *diffusion* term, while the advection term is $b\nabla u$ which is also known as *transport* term, and finally σu is the *reaction* term.

Diffusion Problems

For a composite material, Amini Niaki et al [6] study a system of equations, that models heat transfer with the known heat equation,

$$\frac{\partial T}{\partial t} = a\frac{\partial^2 T}{\partial x^2} + b\frac{d\alpha}{dt}$$

where a, b are parameters, and a second equation for internal heat generation expressed as a derivative of time of the degree of cure $\alpha \in (0, 1)$ is present.

Amini Niaki et al [6] propose a PINN composed of two disconnected subnetworks and the use of a sequential training algorithm that automatically adapts the weights in the loss, hence increasing the model's prediction accuracy.

Based on physics observations, an activation function with a positive output parameter and a non-zero derivative is selected for the temperature describing network's last layer, i.e. a Softplus activation function, that is a smooth approximation to the ReLU activation function. The Sigmoid function is instead chosen for the last layer of the network that represents the degree of cure. Finally, because of its smoothness and non-zero derivative, the hyperbolic-tangent function is employed as the activation function for all hidden layers.

Since accurate exotherm forecasts are critical in the processing of composite materials inside autoclaves, Amini Niaki et al [6] show that PINN correctly predict the maximum part temperature, i.e. exotherm, that occurs in the center of the composite material due to internal heat.

A more complex problem was addressed in Cai et al [22], where the authors study a kind of free boundary problem, known as the Stefan problem.

The Stefan problems can be divided into two types: the direct Stefan problem, in its traditional form, entails determining the temperature distribution in a domain during a phase transition. The latter, inverse problem, is distinguished by a set of free boundary conditions known as Stefan conditions Wang and Perdikaris [178].

The authors characterize temperature distributions using a PINN model, that consists of a DNN to represent the unknown interface and another FCNN with two outputs, one for each phase. This leads to three residuals, each of which is generated using three neural networks, namely the two phases $u_\theta^{(1)}, u_\theta^{(2)}$, as well as the interface s_β that takes the boundary conditions into consideration. The two sets of parameters θ and β are minimized through the mean squared errors losses:

$$\mathcal{L}_{\mathcal{F}}(\theta) = \mathcal{L}_r^{(1)}(\theta) + \mathcal{L}_r^{(2)}(\theta)$$

enforces the two PDEs of the heat equation, one for each phase state:

$$\mathcal{L}_r^{(k)}(\theta) = \frac{1}{N_c}\sum_{i=1}^{N_c}\left\| \frac{\partial u_\theta^{(k)}}{\partial t}(x^i, t^i) - \omega_k \frac{\partial^2 u_\theta^{(k)}}{\partial x^2}(x^i, t^i)\right\|^2, \qquad k = 1, 2.$$

on a set of randomly selected collocation locations $\{(x^i, t^i)\}_{i=1}^{N_c}$, and ω_1, ω_2 are two additional training parameters. While, as for the boundary and initial conditions:

$$\mathcal{L}_{\mathcal{B}}(\theta) = \mathcal{L}_{s_{bc}}^{(1)}(\theta, \beta) + \mathcal{L}_{s_{bc}}^{(2)}(\theta, \beta) + \mathcal{L}_{s_{N_c}}(\theta, \beta) + \mathcal{L}_{s_0}(\beta)$$

where $\mathcal{L}_{s_{bc}}^{(k)}$ are the boundary condition of $u^{(k)}$ on the moving boundary $s(t)$, $\mathcal{L}_{s_{N_c}}$ is the free boundary Stefan problem equation, and $\mathcal{L}_{s_0}$ is the initial condition on the free boundary function. Finally, as for the data,

$$\mathcal{L}_{data}(\theta) = \frac{1}{N_d}\sum_{i=1}^{N_d} \|u_\theta(x_{data}^i, t_{data}^i) - u_i^*\|^2,$$

computes the error of the approximation $u(x, t)$ at known data points.

With the previous setup the authors in Cai et al [22] find an accurate solution, however, the basic PINN model fails to appropriately identify the unknown thermal diffusive values, for the inverse problem, due to a local minimum in the training procedure.

So they employ a dynamic weights technique [179], which mitigates problems that arise during the training of PINNs due to stiffness in the gradient flow dynamics. The method significantly minimizes the relative prediction error, showing that the weights in the loss function are crucial and that choosing ideal weight coefficients can increase the performance of PINNs [22].

In Wang and Perdikaris [178], the authors conclude that the PINNs prove to be versatile in approximating complicated functions like the Stefan problem, despite the absence of adequate theoretical analysis like approximation error or numerical stability.

Advection Problems

In He and Tartakovsky [59], multiple advection-dispersion equations are addressed, like

$$u_t + \nabla \cdot (-\kappa \nabla u + \mathbf{v}u) = s$$

where κ is the dispersion coefficient.

The authors find that the PINN method is accurate and produces results that are superior to those obtained using typical discretization-based methods. Moreover both Dwivedi and Srinivasan [41] and He and Tartakovsky [59] solve the same 2D advection-dispersion equation,

$$u_t + \nabla \cdot (-\kappa \nabla u + \mathbf{a}u) = 0$$

In this comparison, the PINN technique [59] performs better that the ELM method [41], given the errors that emerge along borders, probably due to larger wights assigned to boundary and initial conditions in He and Tartakovsky [59].

Moreover, in Dwivedi and Srinivasan [41], an interesting case of PINN and PIELM failure in solving the linear advection equation is shown, involving PDE with sharp gradient solutions.

He et al [60] solve Darcy and advection-dispersion equations proposing a Multiphysics-informed neural network (MPINN) for subsurface transport problems, and also explore the influence of the neural network size on the accuracy of parameter and state estimates.

In Schiassi et al [153], a comparison of two methods is shown, Deep–TFC and X–TFC, on how the former performs better in terms of accuracy when the problem becomes sufficiently stiff. The examples are mainly based on 1D time-dependent Burgers' equation and the Navier–Stokes (NS) equations.

In the example of the two-dimensional Burgers equation, Jagtap et al [71] demonstrate that by having an approximate a priori knowledge of the position of shock, one can appropriately partition the domain to capture the steep descents in solution. This is accomplished through the cPINN domain decomposition flexibility.

While Arnold and King [9] addressed the Burgers equations in the context of model predictive control (MPC).

In Meng et al [109] the authors study a two-dimensional diffusion-reaction equation that involves long-time integration and they use a parareal PINN (PPINN) to divide the time interval into equal-length sub-domains. PPINN is composed of a fast coarse-grained (CG) solver and a finer solver given by PINN.

3.2.2.2 Flow Problems

Particular cases for unsteady differential problems are the ones connected to the motion of fluids. Navier–Stokes equations are widely present in literature, and connected to a large number of problems and disciplines. This outlines the importance that reliable strategies for solving them has for the scientific community. Many numerical strategies have been developed to solve this kind of problems. However, computational issues connected to specific methods, as well as difficulties that arise in the choice of the discrete spatio-temporal domain may affect the quality of numerical solution. PINNs, providing mesh-free solvers, may allow to overcome some issues of standard numerical methods, offering a novel perspective in this field.

Navier–Stokes Equations

Generally Navier–Stokes equations are written as

$$\mathcal{F}_x(u(x); \nu, p) = -div[\nu(\nabla u + \nabla u^T)] + (u + \nabla)u + \nabla p - f,$$

where, u is the speed of the fluid, p the pressure and ν the viscosity [136]. The dynamic equation is coupled with

$$div(u) = 0$$

for expressing mass conservation.

The Burgers equation, a special case of the Navier–Stokes equations, was covered in the previous section.

Using quick parameter sweeps, Arthurs and King [10] demonstrate how PINNs may be utilized to determine the degree of narrowing in a tube. PINNs are trained using finite element data to estimate Navier–Stokes pressure and velocity fields throughout a parametric domain. The authors present an active learning algorithm (ALA) for training PINNs to predict PDE solutions over vast areas of parameter space by combining ALA, a domain and mesh generator, and a traditional PDE solver with PINN.

PINNs are also applied on the drift-reduced Braginskii model by learning turbulent fields using limited electron pressure data [106]. The authors simulated synthetic plasma using the global drift-ballooning (GDB) finite-difference algorithm by solving a fluid model, ie. two-fluid drift-reduced Braginskii equations. They also observe the possibility to infer 3D turbulent fields from only 2D observations and representations of the evolution equations. This can be used for fluctuations that are difficult to monitor or when plasma diagnostics are unavailable.

Xiao et al [186] review available turbulent flow databases and propose benchmark datasets by systematically altering flow conditions.

Zhu et al [199] predict the temperature and melt pool fluid dynamics in 3D metal additive manufacturing AM processes.

The thermal-fluid model is characterized by Navier–Stokes equations (momentum and mass conservation), and energy conservation equations.

They approach the Dirichlet BC in a "hard" manner, employing a specific piece of the neural network to solely meet the prescribed Dirichlet BC; while Neumann BCs, that account for surface tension, are treated conventionally by adding the term to the loss function. They choose the loss weights based on the ratios of the distinct components of the loss function [199].

Cheng and Zhang [31] solve fluid flows dynamics with Res–PINN, PINN paired with a Resnet blocks, that is used to improve the stability of the neural network. They validate the model with Burgers' equation and Navier–Stokes (N–S) equation, in particular, they deal with the cavity flow and flow past cylinder problems. A curious phenomena observed by Cheng and Zhang [31] is a difference in magnitude between the predicted and actual pressure despite the fact that the distribution of the pressure filed is essentially the same.

To estimate the solutions of parametric Navier–Stokes equations, Sun et al [168] created a physics-constrained, data-free, FC–NN for incompressible flows. The DNN is trained purely by reducing the residuals of the governing N–S conservation equations, without employing CFD simulated data. The boundary conditions are also hard-coded into the DNN architecture, since the aforementioned authors claim that in data-free settings, "hard" boundary enforcement is preferable than "soft" boundary approach.

Three flow examples relevant to cardiovascular applications were used to evaluate the suggested approaches.

In particular, the Navier–Stokes equations are given [168] as:

$$\mathcal{F}(\mathbf{u}, p) = 0 := \begin{cases} \nabla \cdot \mathbf{u} = 0, & \mathbf{x}, t \in \Omega, \boldsymbol{\gamma} \in \mathbb{R}^d, \\ \dfrac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} + \dfrac{1}{\rho}\nabla p - \nu \nabla^2 \mathbf{u} + \mathbf{b}_f = 0, & \mathbf{x}, t \in \Omega, \boldsymbol{\gamma} \in \mathbb{R}^d \end{cases}$$

where $\boldsymbol{\gamma}$ is a parameter vector, and with

$$\begin{aligned} \mathcal{I}(\mathbf{x}, p, \mathbf{u}, \boldsymbol{\gamma}) &= 0, & \mathbf{x} \in \Omega, t = 0, \boldsymbol{\gamma} \in \mathbb{R}^d, \\ \mathcal{B}(t, \mathbf{x}, p, \mathbf{u}, \boldsymbol{\gamma}) &= 0, & \mathbf{x}, t \in \partial\Omega \times [0, T], \boldsymbol{\gamma} \in \mathbb{R}^d, \end{aligned}$$

where $\mathcal{I}$ and $\mathcal{B}$ are generic differential operators that determine the initial and boundary conditions.

The boundary conditions (IC/BC) are addressed individually in Sun et al [168]. The Neumann BC are formulated into the equation loss, i.e., in a soft manner, whereas the IC and Dirichlet BC are encoded in the DNN, i.e., in a hard manner.

As a last example, NSFnets [73] has been developed considering two alternative mathematical representations of the Navier–Stokes equations: the velocity-pressure (VP) formulation and the vorticity-velocity (VV) formulation.

Hyperbolic Equations

Hyperbolic conservation law is used to simplify the Navier–Stokes equations in hemodynamics [81].

Hyperbolic partial differential equations are also addressed by Abreu and Florindo [1]: in particular, they study the inviscid nonlinear Burgers' equation and 1D Buckley–Leverett two-phase problem. They actually try to address problems of the following type:

$$\frac{\partial u}{\partial t} + \frac{\partial H(u)}{\partial x} = 0, \quad x \in \mathbb{R}, \quad t > 0, \qquad u(x, 0) = u_0(x),$$

whose results were compared with those obtained by the Lagrangian–Eulerian and Lax–Friedrichs schemes. While Patel et al [130] proposes a PINN for discovering thermodynamically consistent equations that ensure hyperbolicity for inverse problems in shock hydrodynamics.
Euler equations are hyperbolic conservation laws that might permit discontinuous solutions such as shock and contact waves, and in particular a one dimensional Euler system is written as [71]

$$\frac{\partial U}{\partial t} + \nabla \cdot f(U) = 0, \ x \in \Omega \subset \mathbb{R}^2,$$

where

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho E \end{bmatrix} \qquad f = \begin{bmatrix} \rho u, \\ p + \rho u^2 \\ pu + \rho u E \end{bmatrix}$$

given ρ as the density, p as the pressure, u the velocity, and E the total energy. These equations regulate a variety of high-speed fluid flows, including transonic, supersonic, and hypersonic flows. Mao et al [103] can precisely capture such discontinuous flows solutions for one-dimensional Euler equations, which is a challenging task for existing numerical techniques. According to Mao et al [103], appropriate clustering of training data points around a high gradient area can improve solution accuracy in that area and reduces error propagation to the entire domain. This enhancement suggests the use of a separate localized powerful network in the region with high gradient solution, resulting in the development of a collection of individual local PINNs with varied features that comply with the known prior knowledge of the solution in each sub-domain. As done in Jagtap et al [71], cPINN splits the domain into a number of small subdomains in which multiple neural networks with different architectures (known as sub-PINN networks) can be used to solve the same underlying PDE.

Still in reference to Mao et al [103], the authors solve the one-dimensional Euler equations and a two-dimensional oblique shock wave problem. The authors can capture the solutions with only a few scattered points distributed randomly around the discontinuities. The above-mentioned paper employ density gradient and pressure $p(x, t)$ data, as well as conservation laws, to infer all states of interest (density, velocity, and pressure fields) for the inverse problem without employing any IC/BCs. They were inspired by the experimental photography technique of Schlieren.

They also illustrate that the position of the training points is essential for the training process. The results produced by combining the data with the Euler equations in characteristic form outperform the results obtained using conservative forms.

3.2.2.3 Quantum Problems A 1D nonlinear Schrödinger equation is addressed in Raissi [139], and Raissi et al [146] as:

$$i\frac{\partial\psi}{\partial t}+\frac{1}{2}\frac{\partial^2\psi}{\partial x^2}+|\psi|^2\psi=0 \tag{10}$$

where $\psi(x,t)$ is the complex-valued solution. This problem was chosen to demonstrate the PINN's capacity to handle complex-valued answers and we will develop this example in Sect. 3.4. Also a quantum harmonic oscillator (QHO)

$$i\frac{\partial\psi(x,t)}{\partial t}+\frac{1}{2}\Delta\psi(x,t)-V(x,t)=0$$

is addressed in Stiller et al [167], with V a scalar potential. They propose a gating network that determines which MLP to use, while each MLP consists of linear layers and tanh activation functions; so the solution becomes a weighted sum of MLP predictions. The quality of the approximated solution of PINNs was comparable to that of state-of-the-art spectral solvers that exploit domain knowledge by solving the equation in the Fourier domain or by employing Hermite polynomials.

Vector solitons, which are solitary waves with multiple components in the the coupled nonlinear Schrödinger equation (CNLSE) is addressed by Mo et al [115], who extended PINNs with a pre-fixed multi-stage training algorithm. These findings can be extendable to similar type of equations, such as equations with a rogue wave [176] solution or the Sasa–Satsuma equation and Camassa–Holm equation.

3.3 Other Problems

Physics Informed Neural Networks have been also applied to a wide variety of problems that go beyond classical differential problems. As examples, in the following, the application of such a strategy as been discussed regarding Fractional PDEs and Uncertainty estimation.

3.3.1 Differential Equations of Fractional Order

Fractional PDEs can be used to model a wide variety of phenomenological properties found in nature with parameters that must be calculated from experiment data, however in the spatiotemporal domain, field or experimental measurements are typically scarce and may be affected by noise [128]. Because automatic differentiation is not applicable to fractional operators, the construction of PINNs for fractional models is more complex. One possible solution is to calculate the fractional derivative using the L1 scheme [108].

In the first example from Mehta et al [108] they solve a turbulent flow with one dimensional mean flow.

In Pang et al [128], the authors focus on identifying the parameters of fractional PDEs with known overall form but unknown coefficients and unknown operators, by giving rise to fPINN. They construct the loss function using a hybrid technique that includes both automatic differentiation for integer-order operators and numerical discretization for fractional operators. They also analyse the convergence of fractional advection-diffusion equations (fractional ADEs)

The solution proposed in Pang et al [128], is then extended in Kharazmi et al [77] where they address also time-dependent fractional orders. The formulation in Kharazmi et al [77] uses separate neural network to represent each fractional order and use a large neural network to represent states.

3.3.2 Uncertainty Estimation

In data-driven PDE solvers, there are several causes of uncertainty. The quality of the training data has a significant impact on the solution's accuracy.

To address forward and inverse nonlinear problems represented by partial differential equations (PDEs) with noisy data, Yang et al [190] propose a Bayesian physics-informed neural network (B-PINN). The Bayesian neural network acts as the prior in this Bayesian framework, while an Hamiltonian Monte Carlo (HMC) method or variational inference (VI) method can be used to estimate the posterior.

B-PINNs [190] leverage both physical principles and scattered noisy observations to give predictions and quantify the aleatoric uncertainty coming from the noisy data.

Yang et al [190] test their network on some forward problems (1D Poisson equation, Flow in one dimension across porous material with a boundary layer, Nonlinear Poisson equation in one dimension and the 2D Allen–Cahn equation), while for inverse problems the 1D diffusion-reaction system with nonlinear source term and 2D nonlinear diffusion-reaction system are addressed.

Yang et al [190] use also the B-PINNs for a high-dimensional diffusion-reaction system, where the locations of three contaminating sources are inferred from a set of noisy data.

Yang et al [189] considers the solution of elliptic stochastic differential equations (SDEs) that required approximations of three stochastic processes: the solution $u(x;\gamma)$, the forcing term $f(x;\gamma)$, and the diffusion coefficient $k(x;\gamma)$.

In particular, Yang et al [189] investigates the following, time independent, SDE

$$\begin{aligned}\mathcal{F}_{\boldsymbol{x}}[u(\boldsymbol{x};\gamma);k(\boldsymbol{x};\gamma)] &= f(\boldsymbol{x};\gamma),\\ B_{\boldsymbol{x}}[u(\boldsymbol{x};\gamma)] &= b(\boldsymbol{x};\gamma),\end{aligned}$$

where $k(x;\gamma)$ and $f(x;\gamma)$ are independent stochastic processes, with k strictly positive.

Furthermore, they investigate what happens when there are a limited number of measurements from scattered sensors for the stochastic processes. They show how the problem gradually transform from forward to mixed, and finally to inverse problem. This is accomplished by assuming that there are a sufficient number of measurements from some sensors for $f(x;\gamma)$, and then as the number of sensors measurements for $k(x;\gamma)$ is decreased, the number of sensors measurements on $u(x;\gamma)$ is increased, and thus a forward problem is obtained when there are only sensors for k and not for u, while an inverse problem has to be solved when there are only sensors for u and not for k.

A similar result was previously obtained by [195], where they used stochastic data from sparse sensors, and a PINN to learn the modal functions of the arbitrary polynomial chaos (aPC) expansion of its solution.

Furthermore, [196] propose two PINNs for solving time-dependent stochastic partial differential equation (SPDE), based on spectral dynamically orthogonal (DO) and borthogonal (BO) stochastic process representation approaches. They tested their PINN on a linear stochastic advection problem, a stochastic Burgers' equation, and a nonlinear reaction-diffusion equation.

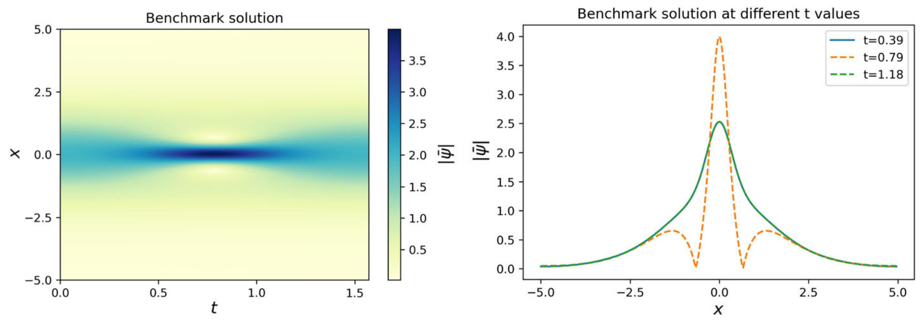

Fig. 3 1D nonlinear Shrödinger (NLS) solution module $|\bar{\psi}|$. The solution is generated with Chebfun open-source software [40] and used as a reference solution. We also show the solution at three different time frames, $t = \frac{\pi}{8}, \frac{\pi}{4}, \frac{3\pi}{8}$. We expect the solution to be symmetric with respect to $\frac{\pi}{4}$

In order to characterizing shape changes (morphodynamics) for cell-drug interactions, Cavanagh et al [27] use kernel density estimation (KDE) for translating morphspace embeddings into probability density functions (PDFs). Then they use a top-down Fokker–Planck model of diffusive development over Waddington-type landscapes, with a PINN learning such landscapes by fitting the PDFs to the Fokker–Planck equation. The architecture includes a neural network for each condition to learn: the PDF, diffusivity, and landscape. All parameters are fitted using approximate Bayesian computing with sequential Monte Carlo (aBc-SMC) methods: in this case, aBc selects parameters from a prior distribution and runs simulations; if the simulations match the data within a certain level of similarity, the parameters are saved. So the posterior distribution is formed by the density over the stored parameters [27].

3.4 Solving a Differential Problem with PINN

Finally, this subsection discusses a realistic example of a 1D nonlinear Shrödinger (NLS) problem, as seen in Fig. 3. The nonlinear problem is the same presented in Raissi [139], Raissi et al [143], used to demonstrate the PINN's ability to deal with periodic boundary conditions and complex-valued solutions.

Starting from an initial state $\psi(x,0) = 2\,\text{sech}(x)$ and assuming periodic boundary conditions Eq. (10) with all boundary conditions results in the initial boundary value problem, given a domain $\Omega = [-5,5] \times (0,T]$ written as:

$$\begin{cases} i\psi_t + 0.5\psi_{xx} + |\psi|^2\psi = 0 & (x,t) \in \Omega \\ \psi(0,x) = 2\,\text{sech}(x) & x \in [-5,5] \\ \psi(t,-5) = \psi(t,5) & t \in (0,T] \\ \psi_x(t,-5) = \psi_x(t,5) & t \in (0,T] \end{cases} \tag{11}$$

where $T = \pi/2$.

To assess the PINN's accuracy, Raissi et al [143] created a high-resolution data set by simulating the Schrödinger equation using conventional spectral methods. The authors integrated the Schrödinger equation up to a final time $T = \pi/2$ using the MATLAB based Chebfun open-source software [40].

The PINN solutions are trained on a subset of measurements, which includes initial data, boundary data, and collocation points inside the domain. The initial time data, $t = 0$, are

$\{x_0^i, \psi_0^i\}_{i=1}^{N_0}$, the boundary collocation points are $\{t_b^i\}_{i=1}^{N_b}$, and the collocation points on $\mathcal{F}(t, x)$ are $\{t_c^i, x_c^i\}_{i=1}^{N_c}$. In Raissi et al [143] a total of $N_0 = 50$ initial data points on $\psi(x, 0)$ are randomly sampled from the whole high-resolution data-set to be included in the training set, as well as $N_b = 50$ randomly sampled boundary points to enforce the periodic boundaries. Finally for the solution domain, it is assumed $N_c = 20\,000$ randomly sampled collocation points.

The neural network architecture has two inputs, one for the time t and the other one the location x, while the output has also length 2 rather than 1, as it would normally be assumed, because the output of this NN is expected to find the real and imaginary parts of the solution.

The network is trained in order to minimize the losses due to the initial and boundary conditions, $\mathcal{L}_{\mathcal{B}}$, as well as to satisfy the Schrodinger equation on the collocation points, i.e. $\mathcal{L}_{\mathcal{F}}$. Because we are interested in a model that is a surrogate for the PDE, no extra data, $\mathcal{L}_{data} = 0$, is used. In fact we only train the PINN with the known data point from the initial time step $t = 0$. So the losses of (6) are:

$$\begin{aligned}\mathcal{L}_{\mathcal{B}} = &\frac{1}{N_0}\sum_{i=1}^{N_0}|\psi(0, x_0^i) - \psi_0^i|^2 \\ &+\frac{1}{N_b}\sum_{i=1}^{N_b}\left(|\psi^i(t_b^i, -5) - \psi^i(t_b^i, 5)|^2 + |\psi_x^i(t_b^i, -5) - \psi_x^i(t_b^i, 5)|^2\right)\end{aligned}$$

and

$$\mathcal{L}_{\mathcal{F}} = \frac{1}{N_c}\sum_{i=1}^{N_c}|\mathcal{F}(t_c^i, x_c^i)|^2.$$

The Latin Hypercube Sampling technique [164] is used to create all randomly sampled points among the benchmark data prior to training the NN.

In our training we use Adam, with a learning rate of 10^{-3}, followed by a final fine-tuning with LBFGS. We then explore different settings and architectures as in Table 2, by analysing the Mean Absolute Error (MAE) and Mean Squared Error (MSE). We used the PyTorch implementation from Stiller et al [167] which is accessible on GitHub. While the benchmark solutions are from the GitHub of Raissi et al [143].

Raissi et al [143] first used a DNN with 5 layers each with 100 neurons per layer and a hyperbolic tangent activation function in order to represent the unknown function ψ for both real and imaginary parts. In our test, we report the original configuration but we also analyze other network architectures and training point amounts.

A comparison of the predicted and exact solutions at three different temporal snapshots is shown in Figs. 3, and 4. All the different configurations reported in Table 2 show similar patterns of Fig. 4, the only difference is in the order of magnitude of the error.

In particular in such Figure, we show the best configuration obtained in our test, with an average value of $5, 17 \cdot 10^{-04}$ according to the MSE. Figure 4 first shows the modulus of the predicted spatio-temporal solution $|\psi(x, t)|$, with respect to the benchmark solution, by plotting the error at each point. This PINN has some difficulty predicting the central height around $(x, t) = (0, \pi/4)$, as well as in mapping value on in $t \in (\pi/4, \pi/2)$ that are symmetric to the time interval $t \in (0, \pi/4)$.

Finally, in Table 2, we present the results of a few runs in which we analyze what happens to the training loss, relative L_2, MAE, and MSE when we vary the boundary data, and initial

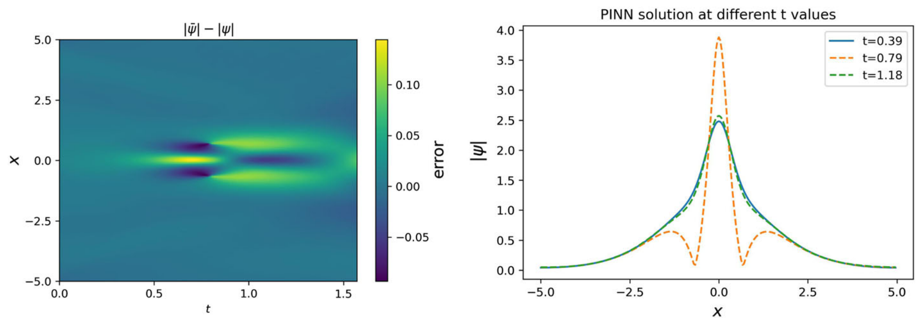

Fig. 4 PINN solutions of the 1D nonlinear Shrödinger (NLS) problem. As illustrated in the first image, the error is represented as the difference among the benchmark solution and PINN result, whereas the second image depicts the PINN's solution at three independent time steps: $t = \pi/8, \pi/4, 3/8\pi$. In this case, we would have expected the solution to overlap at both $\pi/8$ and $3/8\pi$, but there isn't a perfect adherence; in fact, the MSE is $5, 17 \cdot 10^{-04}$ in this case, with some local peak error of 10^{-01} in the second half of the domain

value data. In addition, we can examine how the error grows as the solution progresses through time.

4 PINNs: Data, Applications and Software

The preceding sections cover the neural network component of a PINN framework and which equations have been addressed in the literature. This section first discusses the physical information aspect, which is how the data and model are managed to be effective in a PINN framework. Following that, we'll look at some of the real-world applications of PINNs and how various software packages such as DeepXDE, NeuralPDE, NeuroDiffEq, and others were launched in 2019 to assist with PINNs' design.

4.1 Data

The PINN technique is based not only on the mathematical description of the problems, embedded in the loss or the NN, but also on the information used to train the model, which takes the form of training points, and impacts the quality of the predictions. Working with PINNs requires an awareness of the challenges to be addressed, i.e. knowledge of the key constitutive equations, and also experience in Neural Network building.

Moreover, the learning process of a PINN can be influenced by the relative magnitudes of the different physical parameters. For example to address this issue, Kissas et al [81] used a non-dimensionalization and normalization technique.

However, taking into account the geometry of the problem can be done without effort. PINNs do not require fixed meshes or grids, providing greater flexibility in solving high-dimensional problems on complex-geometry domains.

The training points for PINNs can be arbitrarily distributed in the spatio-temporal domain [128]. However, the distribution of training points influences the flexibility of PINNs. Increasing the number of training points obviously improves the approximation, although in some applications, the location of training places is crucial. Among the various methods for select-

Table 2 Two subtables exploring the effect of the amount of data points on convergence for PINN and the inherent problems of vanilla pINN to adhere to the solution for longer time intervals, in solving the 1D nonlinear Shrödinger (NLS) Eq. (11)

N_0	N_b	N_c	Training loss	relative L_2	MAE	MSE
(a) Case where the NN is fixed with 100 neurons per layer and four hidden layers. Only the number the number of training points on boundary conditions are doubled						
40	50	20000	9×10^{-4}	0.025 ± 0.003	0.065 ± 0.004	0.019 ± 0.002
40	100	20000	1×10^{-3}	0.024 ± 0.002	0.065 ± 0.003	0.019 ± 0.002
80	50	20000	6×10^{-4}	0.007 ± 0.001	0.035 ± 0.004	0.005 ± 0.001
80	100	20000	6×10^{-4}	0.006 ± 0.002	0.033 ± 0.005	0.005 ± 0.002

time t	relative L_2	MAE	MSE
(b) Error behavior at various time steps for the best occurrence in Table, when Table 2a, where $N_0 = 80$, $N_b = 100$, $N_c = 20000$			
0	$(5 \pm 1) \times 10^{-4}$	0.012 ± 0.002	$(4 \pm 1) \times 10^{-4}$
0.39	$(15 \pm 5) \times 10^{-4}$	0.015 ± 0.002	$(12 \pm 4) \times 10^{-3}$
0.79	0.009 ± 0.003	0.038 ± 0.006	0.007 ± 0.003
1.18	0.01 ± 0.004	0.051 ± 0.009	0.009 ± 0.003
1.56	0.005 ± 0.001	0.044 ± 0.005	0.004 ± 0.001

In Table 2a, the NN consists of four layers, each of which contains 100 neurons, and we can observe how increasing the number of training points over initial or boundary conditions will decrease error rates. Furthermore, doubling the initial condition points has a much more meaningful influence impact than doubling the points on the spatial daily domain in this problem setup. In Table 2b, diven the best NN, we can observe that a vanilla PINN has difficulties in maintaining a strong performance overall the whole space-time domain, especially for longer times, this issue is a matter of research discussed in 5.3. All errors, MAE, MSE and L2-norm Rel are average over 10 runs. For all setups, the same optimization parameters are used, including training with 9000 epochs using Adam with a learning rate of 0.001 and a final L-BFGS fine-tuning step ing training points, Pang et al [128] addressed lattice-like sampling, i.e. equispaced, and quasi-random sequences, such as the Sobol sequences or the Latin hypercube sampling.

Another key property of PINN is its ability to describe latent nonlinear state variables that are not observable. For instance, Zhang et al [197] observed that when a variable's measurement was missing, the PINN implementation was capable of accurately predicting that variable.

4.2 Applications

In this section, we will explore the real-world applications of PINNs, focusing on the positive leapfrog applications and innovation that PINNs may bring to our daily lives, as well as the implementation of such applications, such as the ability to collect data in easily accessible locations and simulate dynamics in other parts of the system, or applications to hemodynamics flows, elastic models, or geoscience.

Hemodynamics

Three flow examples in hemodynamics applications are presented in Sun et al [168], for addressing either stenotic flow and aneurysmal flow, with standardized vessel geometries and varying viscosity. In the paper, they not only validate the results against CFD benchmarks, but they also estimate the solutions of the parametric Navier–Stokes equation without labeled data by designing a DNN that enforces the initial and boundary conditions and training the DNN by only minimizing the loss on conservation equations of mass and momentum.

In order to extract velocity and pressure fields directly from images, Raissi et al [147] proposed the Hidden fluid mechanics (HFM) framework. The general structure could be applied for electric or magnetic fields in engineering and biology. They specifically apply it to hemodynamics in a three-dimensional intracranial aneurysm.

Patient-specific systemic artery network topologies can make precise predictions about flow patterns, wall shear stresses, and pulse wave propagation. Such typologies of systemic artery networks are used to estimate Windkessel model parameters [81]. PINN methodology is applied on a simplified form of the Navier–Stokes equations, where a hyperbolic conservation law defines the evolution of blood velocity and cross-sectional area instead of mass and momentum conservation. Kissas et al [81] devise a new method for creating 3D simulations of blood flow: they estimate pressure and retrieve flow information using data from medical imaging. Pre-trained neural network models can quickly be changed to a new patient condition. This allows Windkessel parameters to be calculated as a simple post-processing step, resulting in a straightforward approach for calibrating more complex models. By processing noisy measurements of blood velocity and wall displacement, Kissas et al [81] present a physically valid prediction of flow and pressure wave propagation derived directly from non-invasive MRI flow. They train neural networks to provide output that is consistent with clinical data.

Flows Problems

Mathews et al [106] observe the possibility to infer 3D turbulent fields from only 2D data. To infer unobserved field dynamics from partial observations of synthetic plasma, they simulate the drift-reduced Braginskii model using physics-informed neural networks (PINNs) trained to solve supervised learning tasks while preserving nonlinear partial differential equations.

This paradigm is applicable to the study of quasineutral plasmas in magnetized collisional situations and provides paths for the construction of plasma diagnostics using artificial intelligence. This methodology has the potential to improve the direct testing of reduced turbulence models in both experiment and simulation in ways previously unattainable with standard analytic methods. As a result, this deep learning technique for diagnosing turbulent fields is simply transferrable, allowing for systematic application across magnetic confinement fusion experiments. The methodology proposed in Mathews et al [106] can be adapted to many contexts in the interdisciplinary research (both computational and experimental) of magnetized collisional plasmas in propulsion engines and astrophysical environments.

Xiao et al [186] examine existing turbulent flow databases and proposes benchmark datasets by methodically changing flow conditions.

In the context of high-speed aerodynamic flows, Mao et al [103], investigates Euler equations solutions approximated by PINN. The authors study both forward and inverse, 1D and 2D, problems. As for inverse problems, they analyze two sorts of problems that cannot be addressed using conventional approaches. In the first problem they determine the density, velocity, and pressure using data from the density gradient; in the second problem, they determine the value of the parameter in a two-dimensional oblique wave equation of state by providing density, velocity, and pressure data.

Projecting solutions in time beyond the temporal area used in training is hard to address with the vanilla version of PINN, and such problem is discussed and tested in Kim et al [79]. The authors show that vanilla PINN performs poorly on extrapolation tasks in a variety of Burges' equations benchmark problems, and provide a novel NN with a different training approach.

PINN methodology is also used also to address the 1D Buckley–Leverett two-phase problem used in petroleum engineering that has a non-convex flow function with one inflection point, making the problem quite complex [1]. Results are compared with those obtained by the Lagrangian–Eulerian and Lax–Friedrichs schemes.

Finally, Buckley–Leverett's problem is also addressed in Almajid and Abu-Al-Saud [4], where PINN is compared to an ANN without physical loss: when only early-time saturation profiles are provided as data, ANN cannot predict the solution.

Optics and Electromagnetic Applications

The hybrid PINN from Fang [44], based on CNN and local fitting method, addresses applications such as the 3-D Helmholtz equation, quasi-linear PDE operators, and inverse problems. Additionally, the author tested his hybrid PINN on an icosahedron-based mesh produced by Meshzoo for the purpose of solving surface PDEs.

A PINN was also developed to address power system applications [114] by solving the swing equation, which has been simplified to an ODE. The authors went on to expand on their research results in a subsequent study [165].

In [85] a whole class of microelectromagnetic problems is addressed with a single PINN model, which learns the solutions of classes of eigenvalue problems, related to the nucleation field associated with defects in magnetic materials.

In Chen et al [30], the authors solve inverse scattering problems in photonic metamaterials and nano-optics. They use PINNs to retrieve the effective permittivity characteristics of a number of finite-size scattering systems involving multiple interacting nanostructures and multi-component nanoparticles. Approaches for building electromagnetic metamaterials are explored in Fang and Zhan [45]. E.g. cloaking problem is addressed in both Fang and

Zhan [45], Chen et al [30]. A survey of DL methodologies, including PINNs, applied to nanophotonics may be found in Wiecha et al [183].

As a benchmark problem for DeepM&Mnet, Cai et al [21] choose electroconvection, which depicts electrolyte flow driven by an electric voltage and involves multiphysics, including mass, momentum, cation/anion transport, and electrostatic fields.

Molecular Dynamics and Materials Related Applications

Long-range molecular dynamics simulation is addressed with multi-fidelity PINN (MPINN) by estimating nanofluid viscosity over a wide range of sample space using a very small number of molecular dynamics simulations Islam et al [69]. The authors were able to estimate system energy per atom, system pressure, and diffusion coefficients, in particular with the viscosity of argon-copper nanofluid. PINNs can recreate fragile and non-reproducible particles, as demonstrated by Stielow and Scheel [166], whose network reconstructs the shape and orientation of silver nano-clusters from single-shot scattering images. An encoder–decoder architecture is used to turn 2D images into 3D object space. Following that, the loss score is computed in the scatter space rather than the object space. The scatter loss is calculated by computing the mean-squared difference error between the network prediction and the target's dispersion pattern. Finally, a binary loss is applied to the prediction in order to reinforce the physical concept of the problem's binary nature. They also discover novel geometric shapes that accurately mimic the experimental scattering data.

A multiscale application is done in Lin et al [95, 96], where the authors describe tiny bubbles at both the continuum and atomic levels, the former level using the Rayleigh–Plesset equation and the latter using the dissipative particle dynamics technique. In this paper, the DeepONet architecture [99] is demonstrated to be capable of forecasting bubble growth on the fly across spatial and temporal scales that differ by four orders of magnitude.

Geoscience and Elastostatic Problems

Based on Föppl–von Kármán (FvK) equations, Li et al [94] test their model to four loading cases: in-plane tension with non-uniformly distributed stretching forces, central-hole in-plane tension, deflection out-of-plane, and compression buckling. Moreover stress distribution and displacement field in solid mechanics problems was addressed by Haghighat and Juanes [55].

For earthquake hypocenter inversion, Smith et al [163] use Stein variational inference with a PINN trained to solve the Eikonal equation as a forward model, and then test the method against a database of Southern California earthquakes.

Industrial Application

A broad range of applications, particularly in industrial processes, extends the concept of PINN by adapting to circumstances when the whole underlying physical model of the process is unknown.

The process of lubricant deterioration is still unclear, and models in this field have significant inaccuracies; this is why Yucesan and Viana [194] introduced a hybrid PINN for main bearing fatigue damage accumulation calibrated solely through visual inspections. They use a combination of PINN and ordinal classifier called discrete ordinal classification (DOrC) approach. The authors looked at a case study in which 120 wind turbines were inspected once

a month for six months and found the model to be accurate and error-free. The grease damage accumulation model was also trained using noisy visual grease inspection data. Under fairly conservative ranking, researchers can utilize the derived model to optimize regreasing intervals for a particular useful life goal.
As for, corrosion-fatigue crack growth and bearing fatigue are examples studied in Viana et al [174].

For simulating heat transmission inside a channel, Modulus from NVIDIA [61] trains on a parametrized geometry with many design variables. They specifically change the fin dimensions of the heat sink (thickness, length, and height) to create a design space for various heat sinks [123]. When compared to traditional solvers, which are limited to single geometry simulations, the PINN framework can accelerate design optimization by parameterizing the geometry. Their findings make it possible to perform more efficient design space search tasks for complex systems [22].

4.3 Software

Several software packages, including DeepXDE [100], NVIDIA Modulus (previously SimNet) [61], PyDEns [84], and NeuroDiffEq [28] were released in 2019 to make training PINNs easier and faster. The libraries all used feed-forward NNs and the automatic differentiation mechanism to compute analytical derivatives necessary to determine the loss function. The way packages deal with boundary conditions, whether as a hard constraint or soft constraint, makes a significant difference. When boundary conditions are not embedded in the NN but are included in the loss, various losses must be appropriately evaluated. Multiple loss evaluations and their weighted sum complicate hyper-parameter tuning, justifying the need for such libraries to aid in the design of PINNs.

More libraries have been built in recent years, and others are being updated on a continuous basis, making this a dynamic field of research and development. In this subsection, we will examine each library while also presenting a comprehensive synthesis in Table 3.

DeepXDE

DeepXDE [100] was one of the initial libraries built by one of the vanilla PINN authors. This library emphasizes its problem-solving capability, allowing it to combine diverse boundary conditions and solve problems on domains with complex geometries. They also present residual-based adaptive refinement (RAR), a strategy for optimizing the distribution of residual points during the training stage that is comparable to FEM refinement approaches. RAR works by adding more points in locations where the PDE residual is larger and continuing to add points until the mean residual is less than a threshold limit. DeepXDE also supports complex geometry domains based on the constructive solid geometry (CSG) technique. This package showcases five applications in its first version in 2019, all solved on scattered points: Poisson Equation across an L-shaped Domain, 2D Burgers Equation, first-order Volterra integrodifferential equation, Inverse Problem for the Lorenz System, and Diffusion–Reaction Systems.

Since the package's initial release, a significant number of other articles have been published that make use of it. DeepXDE is used by the authors for: for inverse scattering [30], or deriving mechanical characteristics from materials with MFNNs [98].

While other PINNs are implemented using DeepXDE, like hPINN [78].

Furthermore, more sophisticated tools are built on DeepXDE, such as DeepONets [99], and its later extension DeepM&Mnet [21, 104]. DeepONets and their derivatives are considered by the authors to have the significant potential in approximating operators and addressing multi-physics and multi-scale problems, like inferring bubble dynamics [95, 96].

Finally, DeepXDE is utilized for medical ultrasonography applications to simulate a linear wave equation with a single time-dependent sinusoidal source function [3], and the open-source library is also employed for the Buckley–Leverett problem [4]. A list of research publications that made use of DeepXDE is available online [1]

NeuroDiffEq

NeuroDiffEq [28] is a PyTorch-based library for solving differential equations with neural networks, which is being used at Harvard IACS. NeuroDiffEq solves traditional PDEs (such as the heat equation and the Poisson equation) in 2D by imposing strict constraints, i.e. by fulfilling initial/boundary conditions via NN construction, which makes it a PCNN. They employ a strategy that is similar to the trial function approach [89], but with a different form of the trial function. However, because NeuroDiffEq enforces explicit boundary constraints rather than adding the corresponding losses, they appear to be inadequate for arbitrary bounds that the library does not support [12].

Modulus

Modulus [123], previously known as NVIDIA SimNet [61], from Nvidia, is a toolset for academics and engineers that aims to be both an extendable research platform and a problem solver for real-world and industrial challenges.

It is a PINN toolbox with support to Multiplicative Filter Networks and a gradient aggregation method for larger batch sizes. Modulus also offers Constructive Solid Geometry (CSG) and Tessellated Geometry (TG) capabilities, allowing it to parameterize a wide range of geometries.

In terms of more technical aspects of package implementations, Modulus uses an integral formulation of losses rather than a summation as is typically done. Furthermore, global learning rate annealing is used to fine-tune the weights parameters ω in the loss Eq. 6. Unlike many other packages, Modulus appears to be capable of dealing with a wide range of PDE in either strong or weak form. Additionally, the toolbox supports a wide range of NN, such as Fourier Networks and the DGM architecture, which is similar to the LSTM architecture.

Nvidia showcased the PINN-based code to address multiphysics problems like heat transfer in sophisticated parameterized heat sink geometry [32], 3D blood flow in Intracranial Aneurysm or address data assimilation and inverse problems on a flow passing a 2D cylinder [123]. Moreover, Modulus solves the heat transport problem more quickly than previous solvers.

SciANN

SciANN [55] is an implementation of PINN as a high-level Keras wrapper. Furthermore, the SciANN repository collects a wide range of examples so that others can replicate the results and use those examples to build other solutions, such as elasticity, structural mechanics, and

[1] https://deepxde.readthedocs.io/en/latest/user/research.html.

vibration applications. SciANN is used by the same authors also for creating a nonlocal PINN method in Haghighat et al [56], or for a PINN multi-network model applied on solid mechanics [57]. Although tests for simulating 2D flood, on Shallow Water Equations, are conducted using SciANN [72], the authors wrote the feedforward step into a separate function to avoid the overhead associated with using additional libraries. A general comparison of many types of mesh-free surrogate models based on machine learning (ML) and deep learning (DL) methods is presented in Hoffer et al [64], where SciANN is used among other toolsets. Finally, the PINN framework for solving the Eikonal equation by Waheed et al [175] was implemented using SciAnn.

PyDENs

PyDENs [84] is an open-source neural network PDE solver that allows to define and configure the solution of heat and wave equations. It impose initial/boundary conditions in the NN, making it a PCNN. After the first release in 2019, the development appears to have stopped in 2020.

NeuralPDE.jl

NeuralPDE.jl is part of SciML, a collection of tools for scientific machine learning and differential equation modeling. In particular SciML (Scientific Machine Learning) [137] is a program written in Julia that combines physical laws and scientific models with machine learning techniques.

ADCME

ADCME [187] can be used to develop numerical techniques and connect them to neural networks: in particular, ADCME was developed by extending and enhancing the functionality of TensorFlow. In Xu and Darve [187], ADCME is used to solve different examples, like nonlinear elasticity, Stokes problems, and Burgers' equations. Furthermore, ADCME is used by Xu and Darve [188] for solving inverse problems in stochastic models by using a neural network to approximate the unknown distribution.

Nangs

Nangs [131] is a Python library that uses the PDE's independent variables as NN (inputs), it then computes the derivatives of the dependent variables (outputs), with the derivatives they calculate the PDEs loss function used during the unsupervised training procedure. It has been applied and tested on a 1D and 2D advection-diffusion problem. After a release in 2020, the development appears to have stopped. Although, NeuroDiffEq and Nangs libraries were found to outperform PyDEns in solving higher-dimensional PDEs [134].

TensorDiffEq

TensorDiffEq [107] is a Scientific Machine Learning PINN based toolkit on Tensorflow for Multi–Worker Distributed Computing. Its primary goal is to solve PINNs (inference) and inverse problems (discovery) efficiently through scalability. It implements a Self–Adaptive PINN to increase the weights when the corresponding loss is greater; this task is accomplished by training the network to simultaneously minimize losses and maximize weights.

IDRLnet

IDRLnet [132] is a Python's PINN toolbox inspired by Nvidia SimNet [61]. It provides a way to mix geometric objects, data sources, artificial neural networks, loss metrics, and optimizers. It can also solve noisy inverse problems, variational minimization problem, and integral differential equations.

Elvet

Elvet [7] is a Python library for solving differential equations and variational problems. It can solve systems of coupled ODEs or PDEs (like the quantum harmonic oscillator) and variational problems involving minimization of a given functional (like the catenary or geodesics solutions-based problems).

Other Packages

Packages that are specifically created for PINN can not only solve problems using PINN, but they can also be used to provide the groundwork for future research on PINN developments. However, there are other packages that can take advantage of future research developments, such as techniques based on kernel methods [74]. Indeed, rather than switching approaches on optimizers, losses, and so on, an alternative approach with respect to PINN framework is to vary the way the function is represented. Throughout this aspect, rather than using Neural Networks, a kernel method based on Gaussian process could be used. The two most noteworthy Gaussian processes toolkit are the Neural Tangents [122] kernel (NTK), based on JAX, and GPyTorch [49], written using PyTorch. Neural Tangents handles infinite-width neural networks, allowing for the specification of intricate hierarchical neural network topologies. While GPyTorch models Gaussian processes based on Blackbox Matrix–Matrix multiplication using a specific preconditioner to accelerate convergence.

5 PINN Future Challenges and Directions

What comes next in the PINN theoretical or applied setup is unknown. What we can assess here are the paper's incomplete results, which papers assess the most controversial aspects of PINN, where we see unexplored areas, and where we identify intersections with other disciplines.

Although several articles have been published that have enhanced PINNs capabilities, there are still numerous unresolved issues, such as various applications to real-world situations and equations. These span from more theoretical considerations (such as convergence and stability) to implementation issues (boundary conditions management, neural networks design, general PINN architecture design, and optimization aspects). PINNs and other DL methods using physics prior have the potential to be an effective way of solving high-dimensional PDEs, which are significant in physics, engineering, and finance. PINNs, on the other hand, struggle to accurately approximate the solution of PDEs when compared to other numerical methods designed for a specific PDE, in particular, they can fail to learn complex physical phenomena, like solutions that exhibit multi-scale, chaotic, or turbulent behavior.

Table 3 Major software libraries specifically designed for physics-informed machine learning

Software Name	Backend	Available for	Usage	License	First release	Code Link
DeepXDE Lu et al [100]	TensorFlow, PyTorch, JAX, PaddlePaddle	Python	Solver	Apache-2.0 License	v0.1.0 - Jun 13, 2019	*deepxde*
PyDEns Koryagin et al [84]	Tensorflow	Python	Solver	Apache-2.0 License	v alpha - Jul 14, 2019	*pydens*
NVIDIA Modulus Hennigh et al [61]	PyTorch	Python based API	Solver	Proprietary	v21.06 on Nov 9, 2021	*modulus*
NeuroDiffEq Chen et al [28]	PyTorch	Python	Solver	MIT License	v alpha - Mar 24, 2019	*neurodiffeq*
SciANN Haghighat and Juanes [55]	TensorFlow	Python 2.7–3.6	Wrapper	MIT License	v alpha - Jul 21, 2019	*sciann*
NeuralPDE Zubov et al [201]	Julia	Julia	Solver	MIT License	v0.0.1 - Jun 22, 2019	*NeuralPDE.jl*
ADCME Xu and Darve [187]	Julia TensorFlow	Julia	Wrapper	MIT License	v alpha - Aug 27, 2019	*ADCME.jl*
Nangs Pedro et al [131]	Pytorch	Python	Solver	Apache-2.0 License	v0.0.1 - Jan 9, 2020	*nangs*
TensorDiffEq McClenny et al [107]	Tensorflow 2.x	Python	Solver	MIT License	v0.1.0 - Feb 03, 2021	*TensorDiffEq*
IDRLnet Peng et al [132]	PyTorch, Sympy	Python	Solver	Apache-2.0 License	v0.0.1-alpha Jul 05, 2021	*idrlnet*
Elvet Araz et al [7]	Tensorflow	Python	Solver	MIT License	v0.1.0 Mar 26, 2021	*elvet*

5.1 Overcoming Theoretical Difficulties in PINN

A PINN can be thought of as a three-part modular structure, with an approximation (the neural network), a module to define how we want to correct the approximation (the physics informed network, i.e. the loss), and the module that manages the minimization of the losses. The NN architecture defines how well the NN can approximate a function, and the error we make in approximating is called approximation error, as seen in Sect. 2.4. Then, how we decide to iteratively improve the approximator will be determined by how we define the loss and how many data points we are integrating or calculating the sum, with the quality of such deviation measured as the generalization error. Finally, the quality of iterations in minimizing the losses is dependent on the optimization process, which gives the optimization error.

All of these factors raise numerous questions for future PINN research, the most important of which is whether or not PINN converges to the correct solution of a differential equation. The approximation errors must tend to zero to achieve stability, which is influenced by the network topology. The outcomes of this research are extremely limited. For example, the relative error for several neural architectures was calculated by altering the number of hidden layers and the number of neurons per layer in Mo et al [115]. In another example, Blechschmidt and Ernst [19] shows the number of successes (i.e. when training loss that is less than a threshold) after ten different runs and for different network topologies (number of layers, neurons, and activation function). Instead, Mishra and Molinaro [111] obtain error estimates and identify possible methods by which PINNs can approximate PDEs. It seems that initial hidden layers may be in charge of encoding low-frequency components (fewer points are required to represent low-frequency signals) and the subsequent hidden layers may be in charge of representing higher-frequency components [105]. This could be an extension of the Frequency-principle, F-principle [198], according to which DNNs fit target functions from low to high frequencies during training, implying a low-frequency bias in DNNs and explaining why DNNs do not generalize well on randomized datasets. For PINN, large-scale features should arise first while small-scale features will require multiple training epochs.

The effects of initialization and loss function on DNN learning, specifically on generalization error, should be investigated. Many theoretical results treat loos estimation based on quadrature rules, on points selected randomly and identically distributed. There are some PINN approaches that propose to select collocations points in specific areas of the space-time domain [118]; this should be investigated as well. Finally, dynamic loss weighting for PINN appears to be a promising research direction [120].

Optimization tasks are required to improve the performances of NNs, which also holds for PINN. However, given the physical integration, this new PINN methodology will require additional theoretical foundations on optimization and numerical analysis, and dynamical systems theory. According to [179, 181], a key issue is to understand the relationship between PDE stiffness and the impact of algorithms as the gradient descent on the PINNs.

Another intriguing research topic is why PINN does not suffer from dimensionality. According to the papers published on PINN, they can scale up easily independently of the size of the problems [112]. The computational cost of PINN does not increase exponentially as the problem dimensionality increases; this property is common to neural network architecture, and there is no formal explanation for such patterns [36]. Bauer and Kohler [13] recently demonstrated that least-squares estimates based on FNN can avoid the curse of dimensionality in nonparametric regression. While Zubov et al [202] demonstrates the capability of the PINN technique with quadrature methods in solving high dimensional problems.

In PINN the process of learning gives rise to a predictor, u_θ, which minimizes the empirical risk (loss). In machine learning theory, the prediction error can be divided into two components: bias error and variance error. The bias-variance trade-off appears to contradict the empirical evidence of recent machine-learning systems when neural networks trained to interpolate training data produce near-optimal test results. It is known that a model should balance underfitting and overfitting, as in the typical U curve, according to the bias-variance trade-off. However, extremely rich models like neural networks are trained to exactly fit (i.e., interpolate) the data. Belkin et al [14], demonstrate the existence of a double-descent risk curve across a wide range of models and datasets, and they offer a mechanism for its genesis. The behavior of PINN in such a framework of Learning Theory for Deep Learning remains to be investigated and could lead to further research questions. In particular, the function class $\mathcal{H}$ of the hypothesis space in which PINN is optimized might be further examined by specifying such space based on the type of differential equations that it is solving and thus taking into account the physics informed portion of the network.

In general, PINN can fail to approximate a solution, not due to the lack of expressivity in the NN architecture but due to soft PDE constraint optimization problems Krishnapriyan et al [86].

5.2 Improving Implementation Aspects in PINN

When developing a PINN, the PINN designer must be aware that there may be additional configurations that need to be thoroughly investigated in the literature, as well as various tweaks to consider and good practices that can aid in the development of the PINN, by systematically addressing each of the PINN's three modules. From neural network architecture options to activation function type. In terms of loss development, the approaching calculation of the loss integral, as well as the implementation of the physical problem from adimensionalization or the solution space constrains. Finally, the best training procedure should be designed. How to implement these aspects at the most fundamental level, for each physical problem appears to be a matter of ongoing research, giving rise to a diverse range of papers, which we addressed in this survey, and the missing aspects are summarized in this subsection.

Regarding neural networks architectures, there is still a lack of research for non FFNN types, like CNN and RNN, and what involves their theoretical impact on PINNs [181]; moreover, as for the FFNNs many questions remain, like implementations ones regarding the selection of network size [56]. By looking at Table 1, only a small subset of Neural Networks has been instigated as the network architecture for a PINN, and on a quite restricted set of problems. Many alternative architectures are proposed in the DL literature [35], and PINN development can benefit from this wide range of combinations in such research fields. A possible idea could be to apply Fourier neural operator (FNO) [182], in order to learn a generalized functional space [138]. N–BEATS [124], a deep stack of fully connected layers connected with forward and backward residual links, could be used for time series based phenomena. Transformer architecture instead could handle long-range dependencies by modeling global relationships in complex physical problems [75]. Finally, sinusoidal representation networks (SIRENs) Sitzmann et al [161] that are highly suited for expressing complicated natural signals and their derivatives, could be used in PINN. Some preliminary findings are already available [67, 185]. A research line is to study if it is better to increase the breadth or depth of the FFNN to improve PINN outcomes. Some general studies on DNN make different assertions about whether expanding width has the same benefits as increasing depth. This may prompt the question of whether there is a minimum depth/width below which a network cannot understand the physics [173]. The interoperability of PINN will also play an important role in future research [149]. A greater understanding of PINN's activation function is needed. Jagtap et al [70] show that scalable activation function may be tuned to maximize network performance, in terms of convergence rate and solution correctness. Further research can look into alternative or hybrid methods of differentiating the differential equations. To speed up PINN training, the loss function in Chiu et al [33] is defined using numerical differentiation and automatic differentiation. The proposed can-PINN, i.e. coupled-automatic-numerical differentiation PINN, shows to be more sample efficient and more accurate than traditional PINN; because PINN with automatic differentiation can only achieve high accuracy with many collocation points. While the PINNs training points can be distributed spatially and temporally, making them highly versatile, on the other hand, the position of training locations affects the quality of the results. One downside of PINNs is that the boundary conditions must be established during the training stage, which means that if the boundary conditions change, a new network must be created [183].

As for the loss, it is important to note that a NN will always focus on minimizing the largest loss terms in the weighted equation, therefore all loss terms must be of the same order of magnitude; increasing emphasis on one component of the loss may affect other parts of it. There does not appear to be an objective method for determining the weight variables in the loss equation, nor does there appear to be a mechanism to assure that a certain equation can be solved to a predetermined tolerance before training begins; these are topics that still need to be researched [119].

It seems there has been a lack of research on optimization tasks for PINNs. Not many solutions appear to be implemented, apart from standard solutions such as Adam and BFGS algorithms [184]. The Adam algorithm generates a workflow that can be studied using dynamical systems theory, giving a gradient descent dynamic. More in detail, to reduce stiffness in gradient flow dynamics, studying the limiting neural tangent kernel is needed. Given that there has been great work to solve optimization problems or improve these approaches in machine learning, there is still room for improvement in PINNs optimization techniques [169]. The L-BFGS-B is the most common BFGS used in PINNs, and it is now the most critical PINN technology Markidis [105].
Moreover, the impact of learning rate on PINN training behavior has not been fully investigated. Finally, gradient normalization is another key research topic [120]. It is an approach that dynamically assigns weights to different constraints to remove the dominance of any component of the global loss function.

It is necessary to investigate an error estimation for PINN. One of the few examples comes from Hillebrecht and Unger [62], where using an ODE, they construct an upper bound on the PINN prediction error. They suggest adding an additional weighting parameter to the physics-inspired part of the loss function, which allows for balancing the error contribution of the initial condition and the ODE residual. Unfortunately, only a toy example is offered in this article, and a detailed analysis of the possibility of providing lower bounds on the error estimator needs still to be addressed, as well as an extension to PDEs.

5.3 PINN in the SciML Framework

PINNs, and SciML in general, hold a lot of potential for applying machine learning to critical scientific and technical challenges. However, many questions remain unsolved, particularly if neural networks are to be used as a replacement for traditional numerical methods such as finite difference or finite volume. In Krishnapriyan et al [86] the authors analyze two basic PDE problems of diffusion and convection and show that PINN can fail to learn the ph problem physics when convection or viscosity coefficients are high. They found that the

PINN loss landscape becomes increasingly complex for large coefficients. This is partly due to an optimization problem, because of the PINN soft constraint. However, lower errors are obtained when posing the problem as a sequence-to-sequence learning task instead of solving for the entire space-time at once. These kinds of challenges must be solved if PINN has to be used beyond basic copy-paste by creating in-depth relations between the scientific problem and the machine learning approaches.

Moreover, unexpected uses of PINN can result from applying this framework to different domains. PINN has been employed as linear solvers for the Poisson equation Markidis [105], by bringing attention to the prospect of using PINN as linear solvers that are as quick and accurate as other high-performance solvers such as PETSc solvers. PINNs appear to have some intriguing advantages over more traditional numerical techniques, such as the Finite Element Method (FEM), as explained in Lu et al [100]. Given that PINNs approximate functions and their derivatives nonlinearly, whereas FEM approximates functions linearly, PINNs appear to be well suited for broad use in a wide variety of engineering applications. However, one of the key disadvantages is the requirement to train the NN, which may take significantly longer time than more extensively used numerical methods.

On the other hand, PINNs appear to be useful in a paradigm distinct from that of standard numerical approaches. PINNs can be deployed in an online-offline fashion, with a single PINN being utilized for rapid evaluations of dynamics in real-time, improving predictions. Moving from 2D to 3D poses new obstacles for PINN. As training complexity grows in general, there is a requirement for better representation capacity of neural networks, a demand for a larger batch size that can be limited by GPU memory, and an increased training time to convergence [119]. Another task is to incorporate PINN into more traditional scientific programs and libraries written in Fortran and C/C++, as well as to integrate PINN solvers into legacy HPC applications [105]. PINN could also be implemented on Modern HPC Clusters, by using Horovod [156]. Additionally, when developing the mathematical model that a PINN will solve, the user should be aware of pre-normalizing the problem. At the same time, packages can assist users in dealing with such problems by writing the PDEs in a symbolic form, for example, using SymPy.

PINNs have trouble propagating information from the initial condition or boundary condition to unseen areas of the interior or to future times as an iterative solver [41, 73]. This aspect has recently been addressed by Wang et al [180] that provided a re-formulation of PINNs loss functions that may explicitly account for physical causation during model training. They assess that PINN training algorithms should be designed to respect how information propagates in accordance with the underlying rules that control the evolution of a given system. With the new implementation they observe considerable accuracy gains, as well as the possibility to assess the convergence of a PINNs model, and so PINN, can run for the chaotic Lorenz system, the Kuramoto–Sivashinsky equation in the chaotic domain, and the Navier–Stokes equations in the turbulent regime. However there is still research to be done for hybrid/inverse problems, where observational data should be considered as point sources of information, and PDE residuals should be minimized at those points before propagating information outwards. Another approach is to use ensemble agreement as to the criterion for incorporating new points in the loss calculated from PDEs [58]. The idea is that in the neighborhood of observed/initial data, all ensemble members converge to the same solution, whereas they may be driven towards different incorrect solutions further away from the observations, especially or large time intervals.

PINN can also have a significant impact on our daily lives, as for the example, from Yucesan and Viana [194], where PINNs are used to anticipate grease maintenance; in the industry 4.0 paradigm, they can assist engineers in simulating materials and constructions or analyzing in real-time buildings structures by embedding elastostatic trained PINNs [57, 110]. PINNs also fail to solve PDEs with high-frequency or multi-scale structure [47, 179, 181]. The region of attraction of a specific equilibria of a given autonomous dynamical system could also be investigated with PINN [152].

However, to employ PINN in a safety-critical scenario it will still be important to analyze stability and focus on the method's more theoretical component. Many application areas still require significant work, such as the cultural heritage sector, the healthcare sector, fluid dynamics, particle physics, and the modeling of general relativity with PINN.

It will be important to develop a PINN methodology for stiff problems, as well as use PINN in digital twin applications such as real-time control, cybersecurity, and machine health monitoring [119]. Finally, there is currently a lack of PINNs applications in multi-scale applications, particularly in climate modeling [68], although the PINN methodology has proven capable of addressing its capabilities in numerous applications such as bubble dynamics on multiple scales [95, 96].

5.4 PINN in the AI Framework

PINN could be viewed as a building block in a larger AI framework, or other AI technologies could help to improve the PINN framework.

For more practical applications, PINNs can be used as a tool for engaging deep reinforcement learning (DRL) that combines reinforcement Learning (RL) and deep learning. RL enables agents to conduct experiments to comprehend their environment better, allowing them to acquire high-level causal links and reasoning about causes and effects [11]. The main principle of reinforcement learning is to have an agent learn from its surroundings through exploration and by defining a reward [159]. In the DRL framework, the PINNs can be used as agents. In this scenario, information from the environment could be directly embedded in the agent using knowledge from actuators, sensors, and the prior-physical law, like in a transfer learning paradigm.

PINNs can also be viewed as an example of merging deep learning with symbolic artificial intelligence. The symbolic paradigm is based on the fact that intelligence arises by manipulating abstract models of representations and interactions. This approach has the advantage to discover features of a problem using logical inference, but it lacks the easiness of adhering to real-world data, as in DL. A fulfilling combination of symbolic intelligence and DL would provide the best of both worlds. The model representations could be built up directly from a small amount of data with some priors [50]. In the PINN framework, the physical injection of physical laws could be treated as symbolic intelligence by adding reasoning procedures.

Causal Models are intermediate descriptions that abstract physical models while answering statistical model questions [154]. Differential equations model allows to forecast a physical system's future behavior, assess the impact of interventions, and predict statistical dependencies between variables. On the other hand, a statistical model often doesn't refer to dynamic processes, but rather how some variables allow the prediction of others when the experimental conditions remain constant. In this context, a causal representation learning, merging Machine learning and graphical causality, is a novel research topic, and given the need to model physical phenomena given known data, it may be interesting to investigate whether

PINN, can help to determine causality when used to solve hybrid (mixed forward and inverse) problems.

6 Conclusion

This review can be considered an in-depth study of an innovation process over the last four years rather than a simple research survey in the field of PINNs. Raissi's first research [143, 144], which developed the PINN framework, focused on implementing a PINN to solve known physical models. These innovative papers helped PINN methodology gain traction and justify its original concept even more. Most of the analyzed studies have attempted to personalize the PINNs by modifying the activation functions, gradient optimization procedures, neural networks, or loss function structures. A border extension of PINNs original idea brings to use in the physical loss function bare minimum information of the model, without using a typical PDE equation, and on the other side to embed directly in the NN structure the validity of initial or boundary conditions. Only a few have looked into alternatives to automatic differentiation [44] or at convergence problems [179, 181]. Finally, a core subset of publications has attempted to take this process to a new meta-level by proposing all-inclusive frameworks for many sub-types of physical problems or multi-physics systems [21]. The brilliance of the first PINN articles [143, 144] lies in resurrecting the concept of optimizing a problem with a physical constraint by approximating the unknown function with a neural network [39] and then extending this concept to a hybrid data-equation driven approach within modern research. Countless studies in previous years have approximated the unknown function using ways different than neural networks, such as the kernel approaches [126], or other approaches that have used PDE functions as constraints in an optimization problem [63].

However, PINNs are intrinsically driven by physical information, either from the data point values or the physical equation. The former can be provided at any point in the domain but is usually only as initial or boundary data. More importantly, the latter are the collocation points where the NN is forced to obey the physical model equation.

We examined the literature on PINNs in this paper, beginning with the first papers from Raissi et al [143, 144] and continuing with the research on including physical priors on Neural Networks. This survey looks at PINNs, as a collocation-based method for solving differential questions with Neural networks. Apart from the vanilla PINN solution we look at most of its variants, like variational PINN (VPINN) as well as in its soft form, with loss including the initial and boundary conditions, and its hard form version with boundary conditions encoded in the Neural network structure.

This survey explains the PINN pipeline, analyzing each building block; first, the neural networks, then the loss construction based on the physical model and feedback mechanism. Then, an overall analysis of examples of equations in which the PINN methodology has been used, and finally, an insight into PINNs on where they can be concretely applied and the packages available.

Finally, we can conclude that numerous improvements are still possible; most notably, in unsolved theoretical issues. There is still potential for development in training PINNs optimally and extending PINNs to solve multiple equations.

Funding Open access funding provided by Universitá degli Studi di Napoli Federico II within the CRUI-CARE Agreement. The authors have not disclosed any funding.

Data Availability Enquiries about data availability should be directed to the authors.

Declarations

Competing interests The authors have not disclosed any competing interests.

References

1. Abreu, E., Florindo, J.B.: A Study on a Feedforward Neural Network to Solve Partial Differential Equations in Hyperbolic-Transport Problems. In: Paszynski M, Kranzlmüller D, Krzhizhanovskaya VV, et al (eds) Computational Science – ICCS 2021. Springer International Publishing, Cham, Lecture Notes in Comput. Sci. pp. 398–411, (2021) https://doi.org/10.1007/978-3-030-77964-1_31
2. Aldweesh, A., Derhab, A., Emam, A.Z.: Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. Knowledge-Based Systems **189**, 105,124 (2020). https://doi.org/10.1016/j.knosys.2019.105124, https://www.sciencedirect.com/science/article/pii/S0950705119304897
3. Alkhadhr, S., Liu, X., Almekkawy, M: Modeling of the Forward Wave Propagation Using Physics-Informed Neural Networks. In: 2021 IEEE International Ultrasonics Symposium (IUS), pp. 1–4, (2021) https://doi.org/10.1109/IUS52206.2021.9593574, iSSN: 1948-5727
4. Almajid, M.M., Abu-Al-Saud, M.O.: Prediction of porous media fluid flow using physics informed neural networks. J. Pet. Sci, Eng. **208**, 109,205 (2022). https://doi.org/10.1016/j.petrol.2021.109205, https://www.sciencedirect.com/science/article/pii/S0920410521008597
5. Alom, M.Z., Taha, T.M., Yakopcic, C., et al: A state-of-the-art survey on deep learning theory and architectures. Electron. **8**(3) (2019). https://doi.org/10.3390/electronics8030292, https://www.mdpi.com/2079-9292/8/3/292
6. Amini Niaki, S., Haghighat, E., Campbell, T., et al.: Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. Computer Methods in Applied Mechanics and Engineering **384**, 113,959 (2021). https://doi.org/10.1016/j.cma.2021.113959, https://www.sciencedirect.com/science/article/pii/S0045782521002966
7. Araz, J.Y., Criado, J.C., Spannowsky, M: Elvet – a neural network-based differential equation and variational problem solver (2021). arXiv:2103.14575 [hep-lat, physics:hep-ph, physics:hep-th, stat] , arXiv: 2103.14575
8. Arnold, D.N.: Stability, Consistency, and Convergence of Numerical Discretizations, pp. 1358–1364. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-540-70529-1_407
9. Arnold, F., King, R: State–space modeling for control based on physics-informed neural networks. Eng. Appl. Artif. Intell. **101**, 104,195 . https://doi.org/10.1016/j.engappai.2021.104195, https://www.sciencedirect.com/science/article/pii/S0952197621000427
10. Arthurs, C.J., King, A.P.: Active training of physics-informed neural networks to aggregate and interpolate parametric solutions to the Navier-Stokes equations. J. Comput. Phys. 438:110,364 (2021). https://doi.org/10.1016/j.jcp.2021.110364, https://www.sciencedirect.com/science/article/pii/S002199912100259X
11. Arulkumaran, K., Deisenroth, M.P., Brundage, M., et al.: Deep Reinforcement Learning: A Brief Survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017). https://doi.org/10.1109/MSP.2017.2743240
12. Balu, A., Botelho, S., Khara, B., et al.: Distributed multigrid neural solvers on megavoxel domains. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Association for Computing Machinery, New York, NY, USA, SC '21, (2021) https://doi.org/10.1145/3458817.3476218
13. Bauer, B.: Kohler, M: On deep learning as a remedy for the curse of dimensionality in nonparametric regression. Ann. Statist. **47**(4), 2261–2285 (2019). https://doi.org/10.1214/18-

AOS1747, http://projecteuclid.org/journals/annals-of-statistics/volume-47/issue-4/On-deep-learning-as-a-remedy-for-the-curse-of/10.1214/18-AOS1747.full
14. Belkin, M., Hsu, D., Ma, S., et al.: Reconciling modern machine-learning practice and the classical bias-variance trade-off. Proc. Nat. Acad. Sci. India Sect. **116**(32), 15849–15854 (2019). https://doi.org/10.1073/pnas.1903070116, www.pnas.org/doi/10.1073/pnas.1903070116
15. Bellman, R.: Dynamic programming. Sci. **153**(3731), 34–37 (1966)
16. Berg, J., Nyström, K.: A unified deep artificial neural network approach to partial differential equations in complex geometries. Neurocomputing **317**, 28–41 (2018). https://doi.org/10.1016/j.neucom.2018.06.056, www.sciencedirect.com/science/article/pii/S092523121830794X
17. Berman, D.S., Buczak, A.L., Chavis, J.S., et al.: A survey of deep learning methods for cyber security. Inform. **10**(4) (2019). https://doi.org/10.3390/info10040122, https://www.mdpi.com/2078-2489/10/4/122
18. Biswas, A., Tian, J., Ulusoy, S.: Error estimates for deep learning methods in fluid dynamics. Numer. Math. **151**(3), 753–777 (2022). https://doi.org/10.1007/s00211-022-01294-z
19. Blechschmidt, J., Ernst, O.G.: Three ways to solve partial differential equations with neural networks – A review. GAMM-Mitteilungen **44**(2), e202100,006 (2021). https://doi.org/10.1002/gamm.202100006, https://onlinelibrary.wiley.com/doi/abs/10.1002/gamm.202100006
20. Cai, S., Mao, Z., Wang, Z., et al.: Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta. Mech. Sin. **37**(12), 1727–1738 (2021). https://doi.org/10.1007/s10409-021-01148-1
21. Cai, S., Wang, Z., Lu, L., et al.: DeepM&Mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. J. Comput. Phys. **436**, 110,296 (2021b). https://doi.org/10.1016/j.jcp.2021.110296, https://www.sciencedirect.com/science/article/pii/S0021999121001911
22. Cai, S., Wang, Z., Wang, S., et al.: Physics-Informed Neural Networks for Heat Transfer Problems. J. Heat Transf. **143**(6) (2021c). https://doi.org/10.1115/1.4050542
23. Calin, O.: Convolutional Networks, pp. 517–542. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-36721-3_16
24. Calin, O.: Universal Approximators, pp. 251–284. Springer Series in the Data Sciences, Springer International Publishing, Cham (2020b). https://doi.org/10.1007/978-3-030-36721-3_9
25. Caterini, A.L., Chang, D.E.: In: Caterini, A.L., Chang, D.E. (eds.) Deep Neural Networks in a Mathematical Framework. Generic Representation of Neural Networks, pp. 23–34. SpringerBriefs in Computer Science, Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-75304-1_3
26. Caterini, A.L., Chang, D.E.: Specific Network Descriptions. In: Caterini, A.L., Chang, D.E. (eds.) Deep Neural Networks in a Mathematical Framework, pp. 35–58. Springer International Publishing, Cham, SpringerBriefs in Computer Science (2018). https://doi.org/10.1007/978-3-319-75304-1_4
27. Cavanagh, H., Mosbach, A., Scalliet, G., et al.: Physics-informed deep learning characterizes morphodynamics of asian soybean rust disease. Nat. Commun. **12**(1), 6424 (2021). https://doi.org/10.1038/s41467-021-26577-1
28. Chen, F., Sondak, D., Protopapas, P., et al.: Neurodiffeq: A python package for solving differential equations with neural networks. J. Open Source Softw. **5**(46), 1931 (2020)
29. Chen, H., Engkvist, O., Wang, Y., et al.: The rise of deep learning in drug discovery. Drug Discov. Today **23**(6), 1241–1250 (2018). https://doi.org/10.1016/j.drudis.2018.01.039, www.sciencedirect.com/science/article/pii/S1359644617303598
30. Chen, Y., Lu, L., Karniadakis, G.E., et al.: Physics-informed neural networks for inverse problems in nano-optics and metamaterials. Opt. Express **28**(8), 11618–11633 (2020). https://doi.org/10.1364/OE.384875, www.osapublishing.org/oe/abstract.cfm?uri=oe-28-8-11618
31. Cheng, C., Zhang, G.T.: Deep Learning Method Based on Physics Informed Neural Network with Resnet Block for Solving Fluid Flow Problems. Water **13**(4), 423 (2021). https://doi.org/10.3390/w13040423, www.mdpi.com/2073-4441/13/4/423
32. Cheung, K.C., See, S.: Recent advance in machine learning for partial differential equation. CCF Trans. High Performance Comput. **3**(3), 298–310 (2021). https://doi.org/10.1007/s42514-021-00076-7
33. Chiu, P.H., Wong, J.C., Ooi, C., et al.: CAN-PINN: A fast physics-informed neural network based on coupled-automatic–numerical differentiation method. Comput. Methods Appl. Mech. Engrg. **395**, 114,909 (2022). https://doi.org/10.1016/j.cma.2022.114909, https://www.sciencedirect.com/science/article/pii/S0045782522001906
34. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Systems **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274
35. Dargan, S., Kumar, M., Ayyagari, M.R., et al.: A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning. Arch. Comput. Methods Engrg. **27**(4), 1071–1092 (2020). https://doi.org/10.1007/s11831-019-09344-w

36. De Ryck, T., Mishra, S.: Error analysis for physics informed neural networks (PINNs) approximating Kolmogorov PDEs. (2021) arXiv:2106.14473 [cs, math]
37. De Ryck, T., Lanthaler, S., Mishra, S.: On the approximation of functions by tanh neural networks. Neural Netw. **143**, 732–750 (2021). https://doi.org/10.1016/j.neunet.2021.08.015, www.sciencedirect.com/science/article/pii/S0893608021003208
38. De Ryck, T., Jagtap, A.D., Mishra, S.: Error estimates for physics informed neural networks approximating the Navier-Stokes equations. (2022) arXiv:2203.09346 [cs, math]
39. Dissanayake, M.W.M.G., Phan-Thien, N.: Neural-network-based approximations for solving partial differential equations. Commun. Numer. Methods Eng. **10**(3), 195–201 (1994). https://doi.org/10.1002/cnm.1640100303, https://onlinelibrary.wiley.com/doi/abs/10.1002/cnm.1640100303
40. Driscoll, T.A., Hale, N., Trefethen, L.N.: Chebfun Guide. Pafnuty Publications, http://www.chebfun.org/docs/guide/ (2014)
41. Dwivedi, V., Srinivasan, B.: Physics Informed Extreme Learning Machine (PIELM)-A rapid method for the numerical solution of partial differential equations. Neurocomputing **391**, 96–118 (2020). https://doi.org/10.1016/j.neucom.2019.12.099, www.sciencedirect.com/science/article/pii/S0925231219318144
42. EW, Yu. B.: The Deep Ritz Method: A Deep Learning-Based Numerical Algorithm for Solving Variational Problems. Commun. Math. Stat. **6**(1), 1–12 (2018). https://doi.org/10.1007/s40304-018-0127-z
43. Elbrächter, D., Perekrestenko, D., Grohs, P., et al.: Deep Neural Network Approximation Theory. IEEE Trans. Inf. Theory **67**(5), 2581–2623 (2021). https://doi.org/10.1109/TIT.2021.3062161
44. Fang, Z.: A High-Efficient Hybrid Physics-Informed Neural Networks Based on Convolutional Neural Network. IEEE Transactions on Neural Networks and Learning Systems pp. 1–13. (2021) https://doi.org/10.1109/TNNLS.2021.3070878
45. Fang, Z., Zhan, J.: Deep Physical Informed Neural Networks for Metamaterial Design. IEEE Access **8**, 24506–24513 (2020). https://doi.org/10.1109/ACCESS.2019.2963375
46. Fang, Z., Zhan, J.: A Physics-Informed Neural Network Framework for PDEs on 3D Surfaces: Time Independent Problems. IEEE Access **8**, 26328–26335 (2020). https://doi.org/10.1109/ACCESS.2019.2963390
47. Fuks, O., Tchelepi, H.A.: LIMITATIONS OF PHYSICS INFORMED MACHINE LEARNING FOR NONLINEAR TWO-PHASE TRANSPORT IN POROUS MEDIA. Journal of Machine Learning for Modeling and Computing **1**(1) (2020). https://doi.org/10.1615/.2020033905, https://www.dl.begellhouse.com/journals/558048804a15188a,583c4e56625ba94e,415f83b5707fde65.html
48. Gao, H., Sun, L., Wang, J.X.: PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. J. Comput. Phys. **428**, 110,079 (2021). https://doi.org/10.1016/j.jcp.2020.110079, https://www.sciencedirect.com/science/article/pii/S0021999120308536
49. Gardner, J.R., Pleiss, G., Bindel, D., et al.: Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. In: Advances in Neural Information Processing Systems (2018)
50. Garnelo, M., Shanahan, M.: Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. Curr. Opinion in Behav. Sci. **29**, 17–23 (2019). https://doi.org/10.1016/j.cobeha.2018.12.010, www.sciencedirect.com/science/article/pii/S2352154618301943
51. Geneva, N., Zabaras, N.: Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. J. Comput. Phys. **403**, 109,056 (2020). https://doi.org/10.1016/j.jcp.2019.109056, https://www.sciencedirect.com/science/article/pii/S0021999119307612
52. Goswami, S., Anitescu, C., Chakraborty, S., et al.: Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. Theoret. Appl. Fracture Mech. **106**, 102,447 (2020). https://doi.org/10.1016/j.tafmec.2019.102447, https://www.sciencedirect.com/science/article/pii/S016784421930357X
53. Grandits, T., Pezzuto, S., Costabal, F.S., et al.: Learning Atrial Fiber Orientations and Conductivity Tensors from Intracardiac Maps Using Physics-Informed Neural Networks. In: Ennis, D.B., Perotti, L.E., Wang, V.Y. (eds) Functional Imaging and Modeling of the Heart. Springer International Publishing, Cham, Lecture Notes in Comput. Sci., pp. 650–658 (2021), https://doi.org/10.1007/978-3-030-78710-3_62
54. Grubišić, L., Hajba, M., Lacmanović, D.: Deep Neural Network Model for Approximating Eigenmodes Localized by a Confining Potential. Entropy **23**(1), 95 (2021). https://doi.org/10.3390/e23010095, www.mdpi.com/1099-4300/23/1/95
55. Haghighat, E., Juanes, R.: SciANN: A Keras/Tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks. Comput. Methods Appl. Mech. Engrg. **373**, 113,552 (2021). https://doi.org/10.1016/j.cma.2020.113552, arXiv: 2005.08803

56. Haghighat, E., Bekar, A.C., Madenci, E., et al.: A nonlocal physics-informed deep learning framework using the peridynamic differential operator. Comput. Methods Appl. Mech. Engrg. **385**, 114,012 (2021a). https://doi.org/10.1016/j.cma.2021.114012, https://www.sciencedirect.com/science/article/pii/S0045782521003431
57. Haghighat, E., Raissi, M., Moure, A., et al.: A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput. Methods Appl. Mech. Engrg. **379**, 113,741 (2021b). https://doi.org/10.1016/j.cma.2021.113741, https://www.sciencedirect.com/science/article/pii/S0045782521000773
58. Haitsiukevich, K., Ilin, A.: Improved Training of Physics-Informed Neural Networks with Model Ensembles. (2022) arXiv:2204.05108 [cs, stat]
59. He, Q., Tartakovsky, A.M.: Physics-informed neural network method for forward and backward advection-dispersion equations. Water Resources Research **57**(7), e2020WR029,479 (2021). https://doi.org/10.1029/2020WR029479, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2020WR029479, e2020WR029479 2020WR029479
60. He, Q., Barajas-Solano, D., Tartakovsky, G., et al.: Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. Adv. Water Resources **141**, 103,610 (2020). https://doi.org/10.1016/j.advwatres.2020.103610, https://www.sciencedirect.com/science/article/pii/S0309170819311649
61. Hennigh, O., Narasimhan, S., Nabian, M.A., et al.: NVIDIA SimNet: An AI-Accelerated Multi-Physics Simulation Framework. In: Paszynski M, Kranzlmüller D, Krzhizhanovskaya VV, et al (eds) Computational Science – ICCS 2021. Springer International Publishing, Cham, Lecture Notes in Comput. Sci., pp. 447–461 (2021), https://doi.org/10.1007/978-3-030-77977-1_36
62. Hillebrecht, B., Unger, B.: Certified machine learning: A posteriori error estimation for physics-informed neural networks. Tech. rep., (2022) https://doi.org/10.48550/arXiv.2203.17055, arXiv:2203.17055 [cs, math] type: article
63. Hinze, M., Pinnau, R., Ulbrich, M., et al.: Optimization with PDE constraints, vol. 23. Springer Science & Business Media, Berlin (2008)
64. Hoffer, J.G., Geiger, B.C., Ofner, P., et al.: Mesh-Free Surrogate Models for Structural Mechanic FEM Simulation: A Comparative Study of Approaches. Appl. Sci. **11**(20), 9411 (2021). https://doi.org/10.3390/app11209411, www.mdpi.com/2076-3417/11/20/9411
65. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5), 359–366 (1989). https://doi.org/10.1016/0893-6080(89)90020-8, www.sciencedirect.com/science/article/pii/0893608089900208
66. Huang, G.B., Wang, D.H., Lan, Y.: Extreme learning machines: a survey. Int. J. Mach. Learn. Cybern. **2**(2), 107–122 (2011). https://doi.org/10.1007/s13042-011-0019-y
67. Huang, X., Liu, H., Shi, B., et al.: Solving Partial Differential Equations with Point Source Based on Physics-Informed Neural Networks. (2021) arXiv:2111.01394 [physics]
68. Irrgang, C., Boers, N., Sonnewald, M., et al.: Towards neural Earth system modelling by integrating artificial intelligence in Earth system science. Nat. Mach. Intelligence **3**(8), 667–674 (2021). https://doi.org/10.1038/s42256-021-00374-3, www.nature.com/articles/s42256-021-00374-3
69. Islam, M., Thakur, M.S.H., Mojumder, S., et al.: Extraction of material properties through multi-fidelity deep learning from molecular dynamics simulation. Comput. Mater. Sci. **188**, 110,187 (2021). https://doi.org/10.1016/j.commatsci.2020.110187, https://www.sciencedirect.com/science/article/pii/S0927025620306789
70. Jagtap, A.D., Kawaguchi, K., Karniadakis, G.E.: Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. J. Comput. Phys. **404**, 109,136 (2020a). https://doi.org/10.1016/j.jcp.2019.109136, https://www.sciencedirect.com/science/article/pii/S0021999119308411
71. Jagtap, A.D., Kharazmi, E., Karniadakis, G.E.: Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Comput. Methods Appl. Mech. Engrg. **365**, 113,028 (2020b). https://doi.org/10.1016/j.cma.2020.113028, https://www.sciencedirect.com/science/article/pii/S0045782520302127
72. Jamali, B., Haghighat, E., Ignjatovic, A., et al.: Machine learning for accelerating 2D flood models: Potential and challenges. Hydrological Processes **35**(4), e14,064 (2021). https://doi.org/10.1002/hyp.14064, https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.14064
73. Jin, X., Cai, S., Li, H., et al.: NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. J. Comput. Phys. **426**, 109,951 (2021). https://doi.org/10.1016/j.jcp.2020.109951, https://www.sciencedirect.com/science/article/pii/S0021999120307257
74. Karniadakis, G.E., Kevrekidis, I.G., Lu, L., et al.: Physics-informed machine learning. Nature Reviews Phys. **3**(6), 422–440 (2021). https://doi.org/10.1038/s42254-021-00314-5, www.nature.com/articles/s42254-021-00314-5

75. Kashinath, K., Mustafa, M., Albert, A., et al.: Physics-informed machine learning: case studies for weather and climate modelling. Philosophical Transactions of the Royal Society A: Mathematical, Phys. Eng. Sci. **379**(2194), 20200,093 (2021). https://doi.org/10.1098/rsta.2020.0093, https://royalsocietypublishing.org/doi/full/10.1098/rsta.2020.0093
76. Kharazmi, E., Zhang, Z., Karniadakis, G.E.: Variational Physics-Informed Neural Networks For Solving Partial Differential Equations. (2019) arXiv:1912.00873 [physics, stat]
77. Kharazmi, E., Cai, M., Zheng, X., et al.: Identifiability and predictability of integer- and fractional-order epidemiological models using physics-informed neural networks. Nature Comput. Sci. **1**(11), 744–753 (2021). https://doi.org/10.1038/s43588-021-00158-0
78. Kharazmi, E., Zhang, Z., Karniadakis, G.E.M.: hp-VPINNs: Variational physics-informed neural networks with domain decomposition. Comput. Methods Appl. Mech. Engrg. **374**, 113,547 (2021b). https://doi.org/10.1016/j.cma.2020.113547, https://www.sciencedirect.com/science/article/pii/S0045782520307325
79. Kim, J., Lee, K., Lee, D., et al.: DPM: A Novel Training Method for Physics-Informed Neural Networks in Extrapolation. Proc. AAAI Conf. Artif. Intell. **35**(9), 8146–8154 (2021a). https://ojs.aaai.org/index.php/AAAI/article/view/16992
80. Kim, S.W., Kim, I., Lee, J., et al.: Knowledge Integration into deep learning in dynamical systems: an overview and taxonomy. J. Mech. Sci. Technol. **35**(4), 1331–1342 (2021). https://doi.org/10.1007/s12206-021-0342-5
81. Kissas, G., Yang, Y., Hwuang, E., et al.: Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks. Comput. Methods Appl. Mech. Engrg. **358**, 112,623 (2020). https://doi.org/10.1016/j.cma.2019.112623, https://www.sciencedirect.com/science/article/pii/S0045782519305055
82. Kollmannsberger, S., D'Angella, D., Jokeit, M., et al.: Physics-Informed Neural Networks. In: Kollmannsberger, S., D'Angella, D., Jokeit, M., et al. (eds.) Deep Learning in Computational Mechanics, pp. 55–84. Studies in Computational Intelligence, Springer International Publishing, Cham (2021). https://doi.org/10.1007/978-3-030-76587-3_5
83. Kondor, R., Trivedi, S.: On the Generalization of Equivariance and Convolution in Neural Networks to the Action of Compact Groups. In: Dy J, Krause A (eds) Proceedings of the 35th International Conference on Machine Learning, Proc. Mach. Learn. Res., vol **80**. PMLR, pp. 2747–2755 (2018), https://proceedings.mlr.press/v80/kondor18a.html
84. Koryagin, A., Khudorozkov, R., Tsimfer, S.: PyDEns: a Python Framework for Solving Differential Equations with Neural Networks. (2019) arXiv:1909.11544 [cs, stat]
85. Kovacs, A., Exl, L., Kornell, A., et al.: Conditional physics informed neural networks. Commun. Nonlinear Sci. Numer. Simulation **104**, 106,041 (2022). https://doi.org/10.1016/j.cnsns.2021.106041, https://www.sciencedirect.com/science/article/pii/S1007570421003531
86. Krishnapriyan, A., Gholami, A., Zhe, S., et al.: Characterizing possible failure modes in physics-informed neural networks. In: Ranzato, M., Beygelzimer, A., Dauphin, Y., et al (eds) Advances in Neural Information Processing Systems, vol **34**. Curran Associates, Inc., pp. 26,548–26,560 (2021), https://proceedings.neurips.cc/paper/2021/file/df438e5206f31600e6ae4af72f2725f1-Paper.pdf
87. Kumar, M., Yadav, N.: Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. Computers & Mathematics with Applications **62**(10), 3796–3811 (2011). https://doi.org/10.1016/j.camwa.2011.09.028, www.sciencedirect.com/science/article/pii/S0898122111007966
88. Kutyniok, G.: The Mathematics of Artificial Intelligence (2022). arXiv:2203.08890 [cs, math, stat]
89. Lagaris, I., Likas, A., Fotiadis, D.: Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans. Neural Networks **9**(5), 987–1000 (1998). https://doi.org/10.1109/72.712178
90. Lagaris, I., Likas, A., Papageorgiou, D.: Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans. Neural Networks **11**(5), 1041–1049 (2000). https://doi.org/10.1109/72.870037
91. Lai, Z., Mylonas, C., Nagarajaiah, S., et al.: Structural identification with physics-informed neural ordinary differential equations. J. Sound and Vibration **508**, 116,196 (2021). https://doi.org/10.1016/j.jsv.2021.116196, https://www.sciencedirect.com/science/article/pii/S0022460X21002686
92. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539
93. Lee, H., Kang, I.S.: Neural algorithm for solving differential equations. J. Comput. Phys. **91**(1), 110–131 (1990). https://doi.org/10.1016/0021-9991(90)90007-N, www.sciencedirect.com/science/article/pii/002199919090007N
94. Li, W., Bazant, M.Z., Zhu, J.: A physics-guided neural network framework for elastic plates: Comparison of governing equations-based and energy-based approaches. Comput. Methods Appl. Mech.

Engrg. **383**, 113,933 (2021). https://doi.org/10.1016/j.cma.2021.113933, https://www.sciencedirect.com/science/article/pii/S004578252100270X

95. Lin, C., Li, Z., Lu, L., et al.: Operator learning for predicting multiscale bubble growth dynamics. J. Chem. Phys. **154**(10), 104,118 (2021a). https://doi.org/10.1063/5.0041203, https://aip.scitation.org/doi/10.1063/5.0041203
96. Lin, C., Maxey, M., Li, Z., et al.: A seamless multiscale operator neural network for inferring bubble dynamics. J. Fluid Mech. 929 (2021b). https://doi.org/10.1017/jfm.2021.866, https://www.cambridge.org/core/journals/journal-of-fluid-mechanics/article/seamless-multiscale-operator-neural-network-for-inferring-bubble-dynamics/D516AB0EF954D0FF56AD864DB2618E94
97. Liu, D., Wang, Y.: A Dual-Dimer method for training physics-constrained neural networks with minimax architecture. Neural Netw. **136**, 112–125 (2021). https://doi.org/10.1016/j.neunet.2020.12.028, www.sciencedirect.com/science/article/pii/S0893608020304536
98. Lu, L., Dao, M., Kumar, P., et al.: Extraction of mechanical properties of materials through deep learning from instrumented indentation. Proc. Nat. Acad. Sci. India Sect. **117**(13), 7052–7062 (2020). https://doi.org/10.1073/pnas.1922210117, www.pnas.org/content/117/13/7052
99. Lu, L., Jin, P., Pang, G., et al.: Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. Nat. Mac. Intell. **3**(3), 218–229 (2021). https://doi.org/10.1038/s42256-021-00302-5, www.nature.com/articles/s42256-021-00302-5
100. Lu, L., Meng, X., Mao, Z., et al.: DeepXDE: A deep learning library for solving differential equations. SIAM Rev. **63**(1), 208–228 (2021). https://doi.org/10.1137/19M1274067
101. Lu, L., Pestourie, R., Yao, W., et al.: Physics-informed neural networks with hard constraints for inverse design. SIAM J. Sci. Comput. **43**(6), B1105–B1132 (2021). https://doi.org/10.1137/21M1397908
102. Mallat, S.: Understanding deep convolutional networks. Philosophical Transactions of the Royal Society A: Mathematical, Phys. Eng. Sci. **374**(2065), 20150,203 (2016). https://doi.org/10.1098/rsta.2015.0203, https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0203
103. Mao, Z., Jagtap, A.D., Karniadakis, G.E.: Physics-informed neural networks for high-speed flows. Comput. Methods Appl. Mech. Engrg. **360**, 112,789 (2020). https://doi.org/10.1016/j.cma.2019.112789, https://www.sciencedirect.com/science/article/pii/S0045782519306814
104. Mao, Z., Lu, L., Marxen, O., et al.: DeepM&Mnet for hypersonics: Predicting the coupled flow and finite-rate chemistry behind a normal shock using neural-network approximation of operators. J. Comput. Phys. **447**, 110,698 (2021). https://doi.org/10.1016/j.jcp.2021.110698, https://www.sciencedirect.com/science/article/pii/S0021999121005933
105. Markidis, S.: The Old and the New: Can Physics-Informed Deep-Learning Replace Traditional Linear Solvers? Frontiers in Big Data 4 (2021). https://www.frontiersin.org/article/10.3389/fdata.2021.669097
106. Mathews, A., Francisquez, M., Hughes, J.W., et al.: Uncovering turbulent plasma dynamics via deep learning from partial observations. Phys. Review E **104**(2) (2021). https://doi.org/10.1103/physreve.104.025205, https://www.osti.gov/pages/biblio/1813020
107. McClenny, L.D., Haile, M.A., Braga-Neto, U.M.: Tensordiffeq: Scalable multi-gpu forward and inverse solvers for physics informed neural networks. (2021) arXiv preprint arXiv:2103.16034
108. Mehta, P.P., Pang, G., Song, F., et al.: Discovering a universal variable-order fractional model for turbulent couette flow using a physics-informed neural network. Fract. Calc. Appl. Anal. **22**(6), 1675–1688 (2019). https://doi.org/10.1515/fca-2019-0086
109. Meng, X., Li, Z., Zhang, D., et al.: Ppinn: Parareal physics-informed neural network for time-dependent pdes. Comput. Methods Appl. Mech. Engrg. **370**, 113,250 (2020). https://doi.org/10.1016/j.cma.2020.113250, https://www.sciencedirect.com/science/article/pii/S0045782520304357
110. Minh Nguyen-Thanh, V., Trong Khiem Nguyen, L., Rabczuk, T., et al.: A surrogate model for computational homogenization of elastostatics at finite strain using high-dimensional model representation-based neural network. Int. J. Numer. Methods Eng. **121**(21), 4811–4842 (2020). https://doi.org/10.1002/nme.6493, https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6493
111. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating a class of inverse problems for PDEs. IMA J. Numer. Anal. (2021). https://doi.org/10.1093/imanum/drab032
112. Mishra, S., Molinaro, R.: Physics informed neural networks for simulating radiative transfer. J. Quant. Spectroscopy and Radiative Transf. **270**, 107,705 (2021b). https://doi.org/10.1016/j.jqsrt.2021.107705, https://www.sciencedirect.com/science/article/pii/S0022407321001989
113. Mishra, S., Molinaro, R.: Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA J. Numer. Anal. p drab093 (2022). https://doi.org/10.1093/imanum/drab093
114. Misyris, G.S., Venzke, A., Chatzivasileiadis, S.: Physics-informed neural networks for power systems. 2020 IEEE Power & Energy Society General Meeting (PESGM) pp. 1–5 (2020)

115. Mo, Y., Ling, L., Zeng, D.: Data-driven vector soliton solutions of coupled nonlinear Schrödinger equation using a deep learning algorithm. Phys. Lett. A **421**, 127,739 (2022). https://doi.org/10.1016/j.physleta.2021.127739, https://www.sciencedirect.com/science/article/pii/S0375960121006034
116. Moseley, B., Markham, A., Nissen-Meyer, T.: Finite Basis Physics-Informed Neural Networks (FBPINNs): a scalable domain decomposition approach for solving differential equations. (2021) arXiv:2107.07871 [physics]
117. Muhammad, A.N., Aseere, A.M., Chiroma, H., et al.: Deep learning application in smart cities: recent development, taxonomy, challenges and research prospects. Neural Comput. Appl. **33**(7), 2973–3009 (2021). https://doi.org/10.1007/s00521-020-05151-8
118. Nabian, M.A., Gladstone, R.J., Meidani, H.: Efficient training of physics-informed neural networks via importance sampling. Comput. Aided Civil Infrastruct. Eng. **36**(8), 962–977 (2021). https://doi.org/10.1111/mice.12685, https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12685
119. Nandi, T., Hennigh, O., Nabian, M., et al.: Progress Towards Solving High Reynolds Number Reacting Flows in SimNet. Tech. rep., (2021) https://www.osti.gov/biblio/1846970-progress-towards-solving-high-reynolds-number-reacting-flows-simnet
120. Nandi, T., Hennigh, O., Nabian, M., et al.: Developing Digital Twins for Energy Applications Using Modulus. Tech. rep., (2022) https://www.osti.gov/biblio/1866819
121. Nascimento, R.G., Fricke, K., Viana, F.A.: A tutorial on solving ordinary differential equations using python and hybrid physics-informed neural network. Eng. Appl. Artif. Intell. **96**, 103,996. (2020) https://doi.org/10.1016/j.engappai.2020.103996, https://www.sciencedirect.com/science/article/pii/S095219762030292X
122. Novak, R., Xiao, L., Hron, J., et al.: Neural tangents: Fast and easy infinite neural networks in python. In: International Conference on Learning Representations, (2020) https://github.com/google/neural-tangents
123. NVIDIA Corporation (2021) Modulus User Guide. https://developer.nvidia.com/modulus-user-guide-v2106, release v21.06 – November 9, (2021)
124. Oreshkin, B.N., Carpov, D., Chapados, N., et al.: N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. (2020) arXiv:1905.10437 [cs, stat]
125. Owhadi, H.: Bayesian numerical homogenization. Multiscale Model. Simul **13**(3), 812–828 (2015). https://doi.org/10.1137/140974596
126. Owhadi, H., Yoo, G.R.: Kernel Flows: From learning kernels from data into the abyss. J. Comput. Phys. **389**, 22–47 (2019). https://doi.org/10.1016/j.jcp.2019.03.040, www.sciencedirect.com/science/article/pii/S0021999119302232
127. Özbay, A.G., Hamzehloo, A., Laizet, S., et al.: Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. Data-Centric Engineering 2. (2021) https://doi.org/10.1017/dce.2021.7, https://www.cambridge.org/core/journals/data-centric-engineering/article/poisson-cnn-convolutional-neural-networks-for-the-solution-of-the-poisson-equation-on-a-cartesian-mesh/8CDFD5C9D5172E51B924E9AA1BA253A1
128. Pang, G., Lu, L., Karniadakis, G.E.: fPINNs: Fractional Physics-Informed Neural Networks. SIAM J. Sci. Comput. **41**(4), A2603–A2626 (2019). https://doi.org/10.1137/18M1229845, https://epubs.siam.org/doi/abs/10.1137/18M1229845
129. Paszke, A., Gross, S., Chintala, S., et al.: Automatic differentiation in PyTorch. Tech. rep., (2017) https://openreview.net/forum?id=BJJsrmfCZ
130. Patel, R.G., Manickam, I., Trask, N.A., et al.: Thermodynamically consistent physics-informed neural networks for hyperbolic systems. J. Comput. Phys. **449**, 110,754 (2022). https://doi.org/10.1016/j.jcp.2021.110754, https://www.sciencedirect.com/science/article/pii/S0021999121006495
131. Pedro, J.B., Maroñas, J., Paredes, R.: Solving Partial Differential Equations with Neural Networks. (2019) arXiv:1912.04737 [physics]
132. Peng, W., Zhang, J., Zhou, W., et al.: IDRLnet: A Physics-Informed Neural Network Library. (2021) arXiv:2107.04320 [cs, math]
133. Pinkus, A.: Approximation theory of the MLP model in neural networks. Acta Numer. **8**, 143–195 (1999). https://doi.org/10.1017/S0962492900002919, www.cambridge.org/core/journals/acta-numerica/article/abs/approximation-theory-of-the-mlp-model-in-neural-networks/18072C558C8410C4F92A82BCC8FC8CF9
134. Pratama, D.A., Bakar, M.A., Man, M., et al.: ANNs-Based Method for Solving Partial Differential Equations : A Survey. (2021) Preprint https://doi.org/10.20944/preprints202102.0160.v1, https://www.preprints.org/manuscript/202102.0160/v1
135. Psichogios, D.C., Ungar, L.H.: A hybrid neural network-first principles approach to process modeling. AIChE J. **38**(10), 1499–1511 (1992). https://doi.org/10.1002/aic.690381003, https://onlinelibrary.wiley.com/doi/abs/10.1002/aic.690381003

136. Quarteroni, A.: Numerical Models for Differential Problems, 2nd edn. Springer Publishing Company, Incorporated (2013)
137. Rackauckas, C., Ma, Y., Martensen, J., et al.: Universal Differential Equations for Scientific Machine Learning. (2021) arXiv:2001.04385 [cs, math, q-bio, stat]
138. Rafiq, M., Rafiq, G., Choi, G.S.: DSFA-PINN: Deep Spectral Feature Aggregation Physics Informed Neural Network. IEEE Access **10**, 1 (2022). https://doi.org/10.1109/ACCESS.2022.3153056
139. Raissi, M.: Deep hidden physics models: Deep learning of nonlinear partial differential equations. J. Mach. Learn. Res. **19**(25), 1–24 (2018). http://jmlr.org/papers/v19/18-046.html
140. Raissi, M., Karniadakis, G.E.: Hidden physics models: Machine learning of nonlinear partial differential equations. J. Comput. Phys. **357**, 125–141 (2018). https://doi.org/10.1016/j.jcp.2017.11.039, www.sciencedirect.com/science/article/pii/S0021999117309014
141. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Inferring solutions of differential equations using noisy multi-fidelity data. J. Comput. Phys. **335**, 736–746 (2017). https://doi.org/10.1016/j.jcp.2017.01.060, www.sciencedirect.com/science/article/pii/S0021999117300761
142. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Machine learning of linear differential equations using Gaussian processes. J. Comput. Phys. **348**, 683–693 (2017). https://doi.org/10.1016/j.jcp.2017.07.050, www.sciencedirect.com/science/article/pii/S0021999117305582
143. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. (2017c) arXiv:1711.10561 [cs, math, stat]
144. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. (2017d) arXiv:1711.10566 [cs, math, stat]
145. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Numerical gaussian processes for time-dependent and nonlinear partial differential equations. SIAM J. Sci. Comput. **40**(1), A172–A198 (2018). https://doi.org/10.1137/17M1120762
146. Raissi, M., Perdikaris, P., Karniadakis, G.E.: Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. **378**, 686–707 (2019). https://doi.org/10.1016/j.jcp.2018.10.045, www.sciencedirect.com/science/article/pii/S0021999118307125
147. Raissi, M., Yazdani, A., Karniadakis, G.E.: Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. Sci. **367**(6481), 1026–1030 (2020). https://doi.org/10.1126/science.aaw4741, www.science.org/doi/10.1126/science.aaw4741
148. Ramabathiran, A.A., Ramachandran, P.: SPINN: Sparse, Physics-based, and partially Interpretable Neural Networks for PDEs. J. Comput. Phys. **445**, 110,600 (2021). https://doi.org/10.1016/j.jcp.2021.110600, https://www.sciencedirect.com/science/article/pii/S0021999121004952
149. Rudin, C., Chen, C., Chen, Z., et al.: Interpretable machine learning: Fundamental principles and 10 grand challenges. Stat. Surveys **16**(none), 1–85 (2022). https://doi.org/10.1214/21-SS133, https://projecteuclid.org/journals/statistics-surveys/volume-16/issue-none/Interpretable-machine-learning-Fundamental-principles-and-10-grand-challenges/10.1214/21-SS133.full
150. Ryaben'kii, V.S., Tsynkov, S.V.: A Theoretical Introduction to Numerical Analysis. CRC Press, Boca Raton, FL (2006)
151. Sahli Costabal, F., Yang, Y., Perdikaris, P., et al.: Physics-Informed Neural Networks for Cardiac Activation Mapping. Front. Phys. **8**, 42 (2020). https://doi.org/10.3389/fphy.2020.00042, www.frontiersin.org/article/10.3389/fphy.2020.00042
152. Scharzenberger, C., Hays, J.: Learning To Estimate Regions Of Attraction Of Autonomous Dynamical Systems Using Physics-Informed Neural Networks. Tech. rep., (2021) https://doi.org/10.48550/arXiv.2111.09930, arXiv:2111.09930 [cs] type: article
153. Schiassi, E., Furfaro, R., Leake, C., et al.: Extreme theory of functional connections: A fast physics-informed neural network method for solving ordinary and partial differential equations. Neurocomputing **457**, 334–356 (2021). https://doi.org/10.1016/j.neucom.2021.06.015, www.sciencedirect.com/science/article/pii/S0925231221009140
154. Schölkopf, B., Locatello, F., Bauer, S., et al.: Toward Causal Representation Learning. Proc. IEEE **109**(5), 612–634 (2021). https://doi.org/10.1109/JPROC.2021.3058954
155. Sengupta, S., Basak, S., Saikia, P., et al.: A review of deep learning with special emphasis on architectures, applications and recent trends. Knowledge-Based Systems **194**, 105,596 (2020). https://doi.org/10.1016/j.knosys.2020.105596, https://www.sciencedirect.com/science/article/pii/S095070512030071X
156. Sergeev, A., Del Balso, M.: Horovod: fast and easy distributed deep learning in TensorFlow. Tech. rep., (2018) https://doi.org/10.48550/arXiv.1802.05799, arXiv:1802.05799 [cs, stat] type: article
157. Shin, Y., Darbon, J., Karniadakis, G.E.: On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type PDEs. Commun. Comput. Phys. **28**(5), 2042–2074 (2020a). https://doi.org/10.4208/cicp.OA-2020-0193, arXiv: 2004.01806

158. Shin, Y., Zhang, Z., Karniadakis, G.E.: Error estimates of residual minimization using neural networks for linear PDEs. (2020b) arXiv:2010.08019 [cs, math]
159. Shrestha, A., Mahmood, A.: Review of Deep Learning Algorithms and Architectures. IEEE Access **7**, 53040–53065 (2019). https://doi.org/10.1109/ACCESS.2019.2912200
160. Sirignano, J., Spiliopoulos, K.: DGM: A deep learning algorithm for solving partial differential equations. J. Comput. Phys. **375**, 1339–1364 (2018). https://doi.org/10.1016/j.jcp.2018.08.029, www.sciencedirect.com/science/article/pii/S0021999118305527
161. Sitzmann, V., Martel, J.N.P., Bergman, A.W., et al.: Implicit Neural Representations with Periodic Activation Functions (2020). arXiv:2006.09661 [cs, eess]
162. Smith, J.D., Azizzadenesheli, K., Ross, Z.E.: EikoNet: Solving the Eikonal Equation With Deep Neural Networks. IEEE Trans. Geosci. Remote Sens. **59**(12), 10685–10696 (2021). https://doi.org/10.1109/TGRS.2020.3039165
163. Smith, J.D., Ross, Z.E., Azizzadenesheli, K., et al.: HypoSVI: Hypocentre inversion with Stein variational inference and physics informed neural networks. Geophys. J. Int. **228**(1), 698–710 (2021). https://doi.org/10.1093/gji/ggab309
164. Stein, M.L.: Large sample properties of simulations using latin hypercube sampling. Technometrics **29**, 143–151 (1987)
165. Stiasny J, Misyris GS, Chatzivasileiadis S (2021) Physics-Informed Neural Networks for Non-linear System Identification for Power System Dynamics. In: 2021 IEEE Madrid PowerTech, pp 1–6, https://doi.org/10.1109/PowerTech46648.2021.9495063
166. Stielow, T., Scheel, S.: Reconstruction of nanoscale particles from single-shot wide-angle free-electron-laser diffraction patterns with physics-informed neural networks. Phys. Review E **103**(5), 053,312 (2021). https://doi.org/10.1103/PhysRevE.103.053312, https://link.aps.org/doi/10.1103/PhysRevE.103.053312
167. Stiller, P., Bethke, F., Böhme, M., et al.: Large-Scale Neural Solvers for Partial Differential Equations. In: Nichols J, Verastegui B, Maccabe AB, et al (eds) Driving Scientific and Engineering Discoveries Through the Convergence of HPC, Big Data and AI. Springer International Publishing, Cham, Communications in Computer and Information Science, pp. 20–34, (2020) https://doi.org/10.1007/978-3-030-63393-6_2
168. Sun, L., Gao, H., Pan, S., et al.: Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Comput. Methods Appl. Mech. Engrg. **361**, 112,732 (2020a). https://doi.org/10.1016/j.cma.2019.112732, https://www.sciencedirect.com/science/article/pii/S004578251930622X
169. Sun, S., Cao, Z., Zhu, H., et al.: A Survey of Optimization Methods From a Machine Learning Perspective. IEEE Trans. Cybernet. **50**(8), 3668–3681 (2020). https://doi.org/10.1109/TCYB.2019.2950779
170. Tartakovsky, A.M., Marrero, C.O., Perdikaris, P., et al.: Physics-Informed Deep Neural Networks for Learning Parameters and Constitutive Relationships in Subsurface Flow Problems. Water Resources Research **56**(5), e2019WR026,731 (2020). https://doi.org/10.1029/2019WR026731, https://onlinelibrary.wiley.com/doi/abs/10.1029/2019WR026731
171. Thompson, D.B.: Numerical Methods 101 – Convergence of Numerical Models. ASCE, pp. 398–403 (1992), https://cedb.asce.org/CEDBsearch/record.jsp?dockey=0078142
172. Tong, Y., Xiong, S., He, X., et al.: Symplectic neural networks in taylor series form for hamiltonian systems. J. Comput. Phys. **437**, 110,325 (2021). https://doi.org/10.1016/j.jcp.2021.110325, https://www.sciencedirect.com/science/article/pii/S0021999121002205
173. Torabi Rad, M., Viardin, A., Schmitz, G.J., et al.: Theory-training deep neural networks for an alloy solidification benchmark problem. Comput. Mater. Sci. **180**, 109,687 (2020). https://doi.org/10.1016/j.commatsci.2020.109687, https://www.sciencedirect.com/science/article/pii/S0927025620301786
174. Viana, F.A.C., Nascimento, R.G., Dourado, A., et al.: Estimating model inadequacy in ordinary differential equations with physics-informed neural networks. Comput. Structures **245**, 106,458 (2021). https://doi.org/10.1016/j.compstruc.2020.106458, https://www.sciencedirect.com/science/article/pii/S0045794920302613
175. Waheed, U.b., Haghighat, E., Alkhalifah, T., et al.: PINNeik: Eikonal solution using physics-informed neural networks. Comput. Geosci. **155**, 104,833 (2021). https://doi.org/10.1016/j.cageo.2021.104833, https://www.sciencedirect.com/science/article/pii/S009830042100131X
176. Wang, L., Yan, Z.: Data-driven rogue waves and parameter discovery in the defocusing nonlinear Schrödinger equation with a potential using the PINN deep learning. Phys. Lett. A **404**, 127,408 (2021). https://doi.org/10.1016/j.physleta.2021.127408, https://www.sciencedirect.com/science/article/pii/S0375960121002723
177. Wang, N., Chang, H., Zhang, D.: Theory-guided auto-encoder for surrogate construction and inverse modeling. Comput. Methods Appl. Mech. Engrg. **385**, 114,037 (2021a). https://doi.org/10.1016/j.cma.2021.114037, https://www.sciencedirect.com/science/article/pii/S0045782521003686

178. Wang, S., Perdikaris, P.: Deep learning of free boundary and Stefan problems. J. Comput. Phys. **428**, 109,914 (2021). https://doi.org/10.1016/j.jcp.2020.109914, https://www.sciencedirect.com/science/article/pii/S0021999120306884
179. Wang, S., Teng, Y., Perdikaris, P.: Understanding and Mitigating Gradient Flow Pathologies in Physics-Informed Neural Networks. SIAM J. Sci. Comput. **43**(5), A3055–A3081 (2021). https://doi.org/10.1137/20M1318043, https://epubs.siam.org/doi/abs/10.1137/20M1318043
180. Wang, S., Sankaran, S., Perdikaris, P.: Respecting causality is all you need for training physics-informed neural networks. (2022a) arXiv:2203.07404 [nlin, physics:physics, stat]
181. Wang, S., Yu, X., Perdikaris, P.: When and why PINNs fail to train: A neural tangent kernel perspective. J. Comput. Phys. **449**, 110,768 (2022b). https://doi.org/10.1016/j.jcp.2021.110768, https://www.sciencedirect.com/science/article/pii/S002199912100663X
182. Wen, G., Li, Z., Azizzadenesheli, K., et al.: U-FNO–An enhanced Fourier neural operator-based deep-learning model for multiphase flow. Adv. Water Resources **163**, 104,180 (2022). https://doi.org/10.1016/j.advwatres.2022.104180, https://www.sciencedirect.com/science/article/pii/S0309170822000562
183. Wiecha, P.R., Arbouet, A., Arbouet, A., et al.: Deep learning in nano-photonics: inverse design and beyond. Photonics Research **9**(5), B182–B200 (2021). https://doi.org/10.1364/PRJ.415960, www.osapublishing.org/prj/abstract.cfm?uri=prj-9-5-B182
184. Wong, J.C., Gupta, A., Ong, Y.S.: Can Transfer Neuroevolution Tractably Solve Your Differential Equations? IEEE Comput. Intell. Mag. **16**(2), 14–30 (2021). https://doi.org/10.1109/MCI.2021.3061854
185. Wong, J.C., Ooi, C., Gupta, A., et al.: Learning in Sinusoidal Spaces with Physics-Informed Neural Networks. (2022) arXiv:2109.09338 [physics]
186. Xiao, H., Wu, J.L., Laizet, S., et al.: Flows over periodic hills of parameterized geometries: A dataset for data-driven turbulence modeling from direct simulations. Comput. Fluids **200**, 104,431 (2020). https://doi.org/10.1016/j.compfluid.2020.104431, https://www.sciencedirect.com/science/article/pii/S0045793020300074
187. Xu, K., Darve, E.: ADCME: Learning Spatially-varying Physical Fields using Deep Neural Networks. (2020) arXiv:2011.11955 [cs, math]
188. Xu, K., Darve, E.: Solving inverse problems in stochastic models using deep neural networks and adversarial training. Comput. Methods Appl. Mech. Engrg. **384**, 113,976 (2021). https://doi.org/10.1016/j.cma.2021.113976, https://www.sciencedirect.com/science/article/pii/S0045782521003078
189. Yang, L., Zhang, D., Karniadakis, G.E.: Physics-informed generative adversarial networks for stochastic differential equations. SIAM J. Sci. Comput. **42**(1), A292–A317 (2020). https://doi.org/10.1137/18M1225409
190. Yang, L., Meng, X., Karniadakis, G.E.: B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. J. Comput. Phys. **425**, 109,913 (2021). https://doi.org/10.1016/j.jcp.2020.109913, https://www.sciencedirect.com/science/article/pii/S0021999120306872
191. Yang, Y., Perdikaris, P.: Adversarial uncertainty quantification in physics-informed neural networks. J. Comput. Phys. **394**, 136–152 (2019). https://doi.org/10.1016/j.jcp.2019.05.027, www.sciencedirect.com/science/article/pii/S0021999119303584
192. Yarotsky, D.: Error bounds for approximations with deep relu networks. Neural Netw. **94**, 103–114 (2017). https://doi.org/10.1016/j.neunet.2017.07.002, www.sciencedirect.com/science/article/pii/S0893608017301545
193. Yuan, L., Ni, Y.Q., Deng, X.Y., et al.: A-PINN: Auxiliary physics informed neural networks for forward and inverse problems of nonlinear integro-differential equations. J. Comput. Phys. **462**, 111,260 (2022). https://doi.org/10.1016/j.jcp.2022.111260,https://www.sciencedirect.com/science/article/pii/S0021999122003229
194. Yucesan, Y.A., Viana, F.A.C.: Hybrid physics-informed neural networks for main bearing fatigue prognosis with visual grease inspection. Comput. Ind. **125**:103,386 (2021). https://doi.org/10.1016/j.compind.2020.103386, https://www.sciencedirect.com/science/article/pii/S0166361520306205
195. Zhang, D., Lu, L., Guo, L., et al.: Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. J. Comput. Phys. **397**, 108,850 (2019). https://doi.org/10.1016/j.jcp.2019.07.048, https://www.sciencedirect.com/science/article/pii/S0021999119305340
196. Zhang, D., Guo, L., Karniadakis, G.E.: Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks. SIAM J. Sci. Comput. **42**(2), A639–A665 (2020). https://doi.org/10.1137/19M1260141
197. Zhang, R., Liu, Y., Sun, H.: Physics-informed multi-LSTM networks for metamodeling of nonlinear structures. Comput. Methods Appl. Mech. Engrg. **369**, 113,226 (2020b). https://doi.org/10.1016/j.cma.2020.113226, https://www.sciencedirect.com/science/article/pii/S0045782520304114

198. Zhi-Qin, Xu, J., et al.: Frequency principle: Fourier analysis sheds light on deep neural networks. Commun. Comput. Phys. **28**(5), 1746–1767 (2020). https://doi.org/10.4208/cicp.OA-2020-0085, http://global-sci.org/intro/article_detail/cicp/18395.html
199. Zhu, Q., Liu, Z., Yan, J.: Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks. Comput. Mech. **67**(2), 619–635 (2021). https://doi.org/10.1007/s00466-020-01952-9
200. Zhu, Y., Zabaras, N., Koutsourelakis, P.S., et al.: Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J. Comput. Phys. **394**, 56–81 (2019). https://doi.org/10.1016/j.jcp.2019.05.024, www.sciencedirect.com/science/article/pii/S0021999119303559
201. Zubov, K., McCarthy, Z., Ma, Y., et al.: NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations. (2021a) arXiv:2107.09443 [cs]
202. Zubov, K., McCarthy, Z., Ma, Y., et al.: NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations. (2021b) arXiv:2107.09443 [cs]

Authors and Affiliations

Salvatore Cuomo[1] · Vincenzo Schiano Di Cola[2] · Fabio Giampaolo[1] · Gianluigi Rozza[3] · Maziar Raissi[4] · Francesco Piccialli[1]

Salvatore Cuomo
salvatore.cuomo@unina.it

Vincenzo Schiano Di Cola
vincenzo.schianodicola@unina.it

Fabio Giampaolo
fabio.giampaolo@unina.it

Gianluigi Rozza
grozza@sissa.it

Maziar Raissi
mara4513@colorado.edu

1 Department of Mathematics and Applications "Renato Caccioppoli", University of Naples Federico II, Napoli 80126, Italy

2 Department of Electrical Engineering and Information Technology, University of Naples Federico II, Via Claudio, Napoli 80125, Italy

3 SISSA, International School for Advanced Studies, Mathematics Area, mathLab, via Bonomea 265, Trieste 34136, Italy

4 Department of Applied Mathematics, University of Colorado Boulder, Boulder, CO 80309, USA