

# Accepted Manuscript

Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations

M. Raissi, P. Perdikaris, G.E. Karniadakis

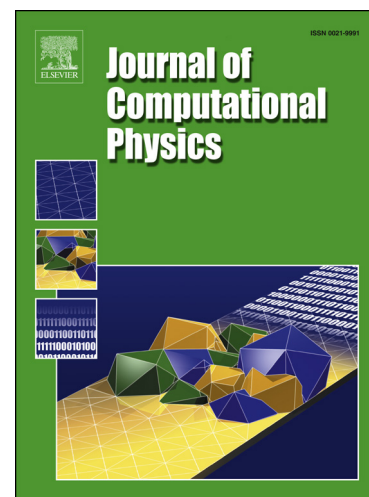
PII: S0021-9991(18)30712-5  
DOI: <https://doi.org/10.1016/j.jcp.2018.10.045>  
Reference: YJCPH 8347

To appear in: *Journal of Computational Physics*

Received date: 13 June 2018  
Revised date: 26 October 2018  
Accepted date: 28 October 2018

Please cite this article in press as: M. Raissi et al., Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations, *J. Comput. Phys.* (2018), <https://doi.org/10.1016/j.jcp.2018.10.045>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.



## Highlights

- We put forth a deep learning framework that creates a new interface between machine learning and scientific computing by enabling the synergistic combination of models and data.
- We introduce an effective mechanism for incorporating first physical principles into machine learning and regularize the training of deep neural networks in small data regimes.
- The resulting data-driven algorithms open a new path to discovering governing equations, and predicting complex dynamics from incomplete models and incomplete data.

# Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations

M. Raissi<sup>1</sup>, P. Perdikaris<sup>2</sup> and G.E. Karniadakis<sup>1</sup>

<sup>1</sup>*Division of Applied Mathematics, Brown University,  
Providence, RI, 02912, USA*

<sup>2</sup>*Department of Mechanical Engineering and Applied Mechanics,  
University of Pennsylvania,  
Philadelphia, PA, 19104, USA*

---

## Abstract

We introduce *physics-informed neural networks* – neural networks that are trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations. In this work, we present our developments in the context of solving two main classes of problems: data-driven solution and data-driven discovery of partial differential equations. Depending on the nature and arrangement of the available data, we devise two distinct types of algorithms, namely continuous time and discrete time models. The first type of models forms a new family of *data-efficient* spatio-temporal function approximators, while the latter type allows the use of arbitrarily accurate implicit Runge-Kutta time stepping schemes with unlimited number of stages. The effectiveness of the proposed framework is demonstrated through a collection of classical problems in fluids, quantum mechanics, reaction-diffusion systems, and the propagation of nonlinear shallow-water waves.

**Keywords:** Data-driven scientific computing, Machine learning, Predictive modeling, Runge-Kutta methods, Nonlinear dynamics

---

## 1. Introduction

With the explosive growth of available data and computing resources, recent advances in machine learning and data analytics have yielded trans-

formative results across diverse scientific disciplines, including image recognition [1], cognitive science [2], and genomics [3]. However, more often than not, in the course of analyzing complex physical, biological or engineering systems, the cost of data acquisition is prohibitive, and we are inevitably faced with the challenge of drawing conclusions and making decisions under partial information. In this *small data* regime, the vast majority of state-of-the-art machine learning techniques (e.g., deep/convolutional/recurrent neural networks) are lacking robustness and fail to provide any guarantees of convergence.

At first sight, the task of training a deep learning algorithm to accurately identify a nonlinear map from a few – potentially very high-dimensional – input and output data pairs seems at best naive. Coming to our rescue, for many cases pertaining to the modeling of physical and biological systems, there exists a vast amount of prior knowledge that is currently not being utilized in modern machine learning practice. Let it be the principled physical laws that govern the time-dependent dynamics of a system, or some empirically validated rules or other domain expertise, this prior information can act as a regularization agent that constrains the space of admissible solutions to a manageable size (e.g., in incompressible fluid dynamics problems by discarding any non realistic flow solutions that violate the conservation of mass principle). In return, encoding such structured information into a learning algorithm results in amplifying the information content of the data that the algorithm sees, enabling it to quickly steer itself towards the right solution and generalize well even when only a few training examples are available.

The first glimpses of promise for exploiting structured prior information to construct data-efficient and physics-informed learning machines have already been showcased in the recent studies of [4–6]. There, the authors employed Gaussian process regression [7] to devise functional representations that are tailored to a given linear operator, and were able to accurately infer solutions and provide uncertainty estimates for several prototype problems in mathematical physics. Extensions to nonlinear problems were proposed in subsequent studies by Raissi *et. al.* [8, 9] in the context of both inference and systems identification. Despite the flexibility and mathematical elegance of Gaussian processes in encoding prior information, the treatment of nonlinear problems introduces two important limitations. First, in [8, 9] the authors had to locally linearize any nonlinear terms in time, thus limiting the applica-

bility of the proposed methods to discrete-time domains and compromising the accuracy of their predictions in strongly nonlinear regimes. Secondly, the Bayesian nature of Gaussian process regression requires certain prior assumptions that may limit the representation capacity of the model and give rise to robustness/brittleness issues, especially for nonlinear problems [10].

## 2. Problem setup

In this work we take a different approach by employing deep neural networks and leverage their well known capability as universal function approximators [11]. In this setting, we can directly tackle nonlinear problems without the need for committing to any prior assumptions, linearization, or local time-stepping. We exploit recent developments in automatic differentiation [12] – one of the most useful but perhaps under-utilized techniques in scientific computing – to differentiate neural networks with respect to their input coordinates and model parameters to obtain *physics-informed neural networks*. Such neural networks are constrained to respect any symmetries, invariances, or conservation principles originating from the physical laws that govern the observed data, as modeled by general time-dependent and nonlinear partial differential equations. This simple yet powerful construction allows us to tackle a wide range of problems in computational science and introduces a potentially transformative technology leading to the development of new data-efficient and physics-informed learning machines, new classes of numerical solvers for partial differential equations, as well as new data-driven approaches for model inversion and systems identification.

The general aim of this work is to set the foundations for a new paradigm in modeling and computation that enriches deep learning with the longstanding developments in mathematical physics. To this end, our manuscript is divided into two parts that aim to present our developments in the context of two major classes of problems: data-driven solution and data-driven discovery of partial differential equations. All code and data-sets accompanying this manuscript are available on GitHub at <https://github.com/maziarraissi/PINNs>. Throughout this work we have been using relatively simple deep feed-forward neural networks architectures with hyperbolic tangent activation functions and no additional regularization (e.g., L1/L2 penalties, dropout, etc.). Each numerical example in the manuscript is accompanied with a detailed discussion about the neural network architecture we

employed as well as details about its training process (e.g. optimizer, learning rates, etc.). Finally, a comprehensive series of systematic studies that aims to demonstrate the performance of the proposed methods is provided in Appendix A and Appendix B.

In this work, we consider parametrized and nonlinear partial differential equations of the general form

$$u_t + \mathcal{N}[u; \lambda] = 0, \quad x \in \Omega, \quad t \in [0, T], \quad (1)$$

where  $u(t, x)$  denotes the latent (hidden) solution,  $\mathcal{N}[\cdot; \lambda]$  is a nonlinear operator parametrized by  $\lambda$ , and  $\Omega$  is a subset of  $\mathbb{R}^D$ . This setup encapsulates a wide range of problems in mathematical physics including conservation laws, diffusion processes, advection-diffusion-reaction systems, and kinetic equations. As a motivating example, the one dimensional Burgers equation [13] corresponds to the case where  $\mathcal{N}[u; \lambda] = \lambda_1 u u_x - \lambda_2 u_{xx}$  and  $\lambda = (\lambda_1, \lambda_2)$ . Here, the subscripts denote partial differentiation in either time or space. Given noisy measurements of the system, we are interested in the solution of two distinct problems. The first problem is that of inference, filtering and smoothing, or data-driven solutions of partial differential equations [4, 8] which states: *given fixed model parameters  $\lambda$  what can be said about the unknown hidden state  $u(t, x)$  of the system?* The second problem is that of learning, system identification, or data-driven discovery of partial differential equations [5, 9, 14] stating: *what are the parameters  $\lambda$  that best describe the observed data?*

### 3. Data-driven solutions of partial differential equations

Let us start by concentrating on the problem of computing data-driven solutions to partial differential equations (i.e., the first problem outlined above) of the general form

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega, \quad t \in [0, T], \quad (2)$$

where  $u(t, x)$  denotes the latent (hidden) solution,  $\mathcal{N}[\cdot]$  is a nonlinear differential operator, and  $\Omega$  is a subset of  $\mathbb{R}^D$ . In sections 3.1 and 3.2, we put forth two distinct types of algorithms, namely continuous and discrete time models, and highlight their properties and performance through the lens of different benchmark problems. In the second part of our study (see section 4), we shift our attention to the problem of data-driven discovery of partial differential equations [5, 9, 14].

### 111 3.1. Continuous Time Models

112 We define  $f(t, x)$  to be given by the left-hand-side of equation (2); i.e.,

$$f := u_t + \mathcal{N}[u], \quad (3)$$

113 and proceed by approximating  $u(t, x)$  by a deep neural network. This as-  
 114 sumption along with equation (3) result in a *physics-informed neural net-*  
 115 *work*  $f(t, x)$ . This network can be derived by applying the chain rule for  
 116 differentiating compositions of functions using automatic differentiation [12],  
 117 and has the same parameters as the network representing  $u(t, x)$ , albeit with  
 118 different activation functions due to the action of the differential operator  
 119  $\mathcal{N}$ . The shared parameters between the neural networks  $u(t, x)$  and  $f(t, x)$   
 120 can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f, \quad (4)$$

121 where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

122 and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

123 Here,  $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$  denote the initial and boundary training data on  $u(t, x)$   
 124 and  $\{t_f^i, x_f^i\}_{i=1}^{N_f}$  specify the collocations points for  $f(t, x)$ . The loss  $MSE_u$   
 125 corresponds to the initial and boundary data while  $MSE_f$  enforces the struc-  
 126 ture imposed by equation (2) at a finite set of collocation points. Although  
 127 similar ideas for constraining neural networks using physical laws have been  
 128 explored in previous studies [15, 16], here we revisit them using modern  
 129 computational tools, and apply them to more challenging dynamic problems  
 130 described by time-dependent nonlinear partial differential equations.

131  
 132 Here, we should underline an important distinction between this line of  
 133 work and existing approaches in the literature that elaborate on the use of  
 134 machine learning in computational physics. The term *physics-informed ma-*  
 135 *chine learning* has been also recently used by Wang *et. al.* [17] in the context  
 136 of turbulence modeling. Other examples of machine learning approaches for  
 137 predictive modeling of physical systems include [18–29]. All these approaches

138 employ machine learning algorithms like support vector machines, random  
 139 forests, Gaussian processes, and feed-forward/convolutional/recurrent neural  
 140 networks merely as *black-box* tools. As described above, the proposed work  
 141 aims to go one step further by revisiting the construction of “custom” activa-  
 142 tion and loss functions that are tailored to the underlying differential opera-  
 143 tor. This allows us to open the black-box by understanding and appreciating  
 144 the key role played by automatic differentiation within the deep learning field.  
 145 Automatic differentiation in general, and the back-propagation algorithm in  
 146 particular, is currently the dominant approach for training deep models by  
 147 taking their derivatives with respect to the parameters (e.g., weights and  
 148 biases) of the models. Here, we use the exact same automatic differentiation  
 149 techniques, employed by the deep learning community, to physics-inform  
 150 neural networks by taking their derivatives with respect to their input co-  
 151 ordinates (i.e., space and time) where the physics is described by partial  
 152 differential equations. We have empirically observed that this structured ap-  
 153 proach introduces a regularization mechanism that allows us to use relatively  
 154 simple feed-forward neural network architectures and train them with small  
 155 amounts of data. The effectiveness of this simple idea may be related to  
 156 the remarks put forth by Lin, Tegmark and Rolnick [30] and raises many  
 157 interesting questions to be quantitatively addressed in future research. To  
 158 this end, the proposed work draws inspiration from the early contributions of  
 159 Psychogios and Ungar [16], Lagaris *et. al.* [15], as well as the contemporary  
 160 works of Kondor [31, 32], Hirn [33], and Mallat [34].

161  
 162 In all cases pertaining to data-driven solution of partial differential equa-  
 163 tions, the total number of training data  $N_u$  is relatively small (a few hundred  
 164 up to a few thousand points), and we chose to optimize all loss functions using  
 165 L-BFGS, a quasi-Newton, full-batch gradient-based optimization algorithm  
 166 [35]. For larger data-sets, such as the data-driven model discovery examples  
 167 discussed in section 4, a more computationally efficient mini-batch setting can  
 168 be readily employed using stochastic gradient descent and its modern vari-  
 169 ants [36, 37]. Despite the fact that there is no theoretical guarantee that this  
 170 procedure converges to a global minimum, our empirical evidence indicates  
 171 that, if the given partial differential equation is well-posed and its solution is  
 172 unique, our method is capable of achieving good prediction accuracy given  
 173 a sufficiently expressive neural network architecture and a sufficient num-  
 174 ber of collocation points  $N_f$ . This general observation deeply relates to the  
 175 resulting optimization landscape induced by the mean square error loss of



equation 4, and defines an open question for research that is in sync with recent theoretical developments in deep learning [38, 39]. To this end, we will test the robustness of the proposed methodology using a series of systematic sensitivity studies that are provided in Appendix A and Appendix B.

### 3.1.1. Example (Schrödinger Equation)

This example aims to highlight the ability of our method to handle periodic boundary conditions, complex-valued solutions, as well as different types of nonlinearities in the governing partial differential equations. The one-dimensional nonlinear Schrödinger equation is a classical field equation that is used to study quantum mechanical systems, including nonlinear wave propagation in optical fibers and/or waveguides, Bose-Einstein condensates, and plasma waves. In optics, the nonlinear term arises from the intensity dependent index of refraction of a given material. Similarly, the nonlinear term for Bose-Einstein condensates is a result of the mean-field interactions of an interacting, N-body system. The nonlinear Schrödinger equation along with periodic boundary conditions is given by

$$\begin{aligned} ih_t + 0.5h_{xx} + |h|^2h &= 0, \quad x \in [-5, 5], \quad t \in [0, \pi/2], \\ h(0, x) &= 2 \operatorname{sech}(x), \\ h(t, -5) &= h(t, 5), \\ h_x(t, -5) &= h_x(t, 5), \end{aligned} \quad (5)$$

where  $h(t, x)$  is the complex-valued solution. Let us define  $f(t, x)$  to be given by

$$f := ih_t + 0.5h_{xx} + |h|^2h,$$

and proceed by placing a complex-valued neural network prior on  $h(t, x)$ . In fact, if  $u$  denotes the real part of  $h$  and  $v$  is the imaginary part, we are placing a multi-out neural network prior on  $h(t, x) = [u(t, x) \ v(t, x)]$ . This will result in the complex-valued (multi-output) *physic-informed neural network*  $f(t, x)$ . The shared parameters of the neural networks  $h(t, x)$  and  $f(t, x)$  can be learned by minimizing the mean squared error loss

$$MSE = MSE_0 + MSE_b + MSE_f, \quad (6)$$

where

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(0, x_0^i) - h_0^i|^2,$$

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} (|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2),$$

and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

Here,  $\{x_0^i, h_0^i\}_{i=1}^{N_0}$  denotes the initial data,  $\{t_b^i\}_{i=1}^{N_b}$  corresponds to the collocation points on the boundary, and  $\{t_f^i, x_f^i\}_{i=1}^{N_f}$  represents the collocation points on  $f(t, x)$ . Consequently,  $MSE_0$  corresponds to the loss on the initial data,  $MSE_b$  enforces the periodic boundary conditions, and  $MSE_f$  penalizes the Schrödinger equation not being satisfied on the collocation points.

In order to assess the accuracy of our method, we have simulated equation (5) using conventional spectral methods to create a high-resolution data set. Specifically, starting from an initial state  $h(0, x) = 2 \operatorname{sech}(x)$  and assuming periodic boundary conditions  $h(t, -5) = h(t, 5)$  and  $h_x(t, -5) = h_x(t, 5)$ , we have integrated equation (5) up to a final time  $t = \pi/2$  using the Chebfun package [40] with a spectral Fourier discretization with 256 modes and a fourth-order explicit Runge-Kutta temporal integrator with time-step  $\Delta t = \pi/2 \cdot 10^{-6}$ . Under our data-driven setting, all we observe are measurements  $\{x_0^i, h_0^i\}_{i=1}^{N_0}$  of the latent function  $h(t, x)$  at time  $t = 0$ . In particular, the training set consists of a total of  $N_0 = 50$  data points on  $h(0, x)$  randomly parsed from the full high-resolution data-set, as well as  $N_b = 50$  randomly sampled collocation points  $\{t_b^i\}_{i=1}^{N_b}$  for enforcing the periodic boundaries. Moreover, we have assumed  $N_f = 20,000$  randomly sampled collocation points used to enforce equation (5) inside the solution domain. All randomly sampled point locations were generated using a space filling Latin Hypercube Sampling strategy [41].

Here our goal is to infer the entire spatio-temporal solution  $h(t, x)$  of the Schrödinger equation (5). We chose to jointly represent the latent function  $h(t, x) = [u(t, x) \ v(t, x)]$  using a 5-layer deep neural network with 100 neurons per layer and a hyperbolic tangent activation function. In general, the neural network should be given sufficient approximation capacity in order to accommodate the anticipated complexity of  $u(t, x)$ . Although more systematic procedures such as Bayesian optimization [42] can be employed in order

234 to fine-tune the design of the neural network, in the absence of theoretical  
 235 error/convergence estimates, the interplay between the neural architecture/-  
 236 training procedure and the complexity of the underlying differential equation  
 237 is still poorly understood. One viable path towards assessing the accuracy  
 238 of the predicted solution could come by adopting a Bayesian approach and  
 239 monitoring the variance of the predictive posterior distribution, but this goes  
 240 beyond the scope of the present work and will be investigated in future stud-  
 241 ies.

242 In this example, our setup aims to highlight the robustness of the pro-  
 243 posed method with respect to the well known issue of over-fitting. Specifi-  
 244 cally, the term in  $MSE_f$  in equation (6) acts as a regularization mechanism  
 245 that penalizes solutions that do not satisfy equation (5). Therefore, a key  
 246 property of *physics-informed neural networks* is that they can be effectively  
 247 trained using small data sets; a setting often encountered in the study of  
 248 physical systems for which the cost of data acquisition may be prohibitive.  
 249 Figure 1 summarizes the results of our experiment. Specifically, the top  
 250 panel of figure 1 shows the magnitude of the predicted spatio-temporal solu-  
 251 tion  $|h(t, x)| = \sqrt{u^2(t, x) + v^2(t, x)}$ , along with the locations of the initial and  
 252 boundary training data. The resulting prediction error is validated against  
 253 the test data for this problem, and is measured at  $1.97 \cdot 10^{-3}$  in the rela-  
 254 tive  $\mathbb{L}_2$ -norm. A more detailed assessment of the predicted solution is pre-  
 255 sented in the bottom panel of Figure 1. In particular, we present a compar-  
 256 ison between the exact and the predicted solutions at different time instants  
 257  $t = 0.59, 0.79, 0.98$ . Using only a handful of initial data, the *physics-informed*  
 258 *neural network* can accurately capture the intricate nonlinear behavior of the  
 259 Schrödinger equation.

260  
 261 One potential limitation of the continuous time neural network models  
 262 considered so far stems from the need to use a large number of collocation  
 263 points  $N_f$  in order to enforce physics-informed constraints in the en-  
 264 tire spatio-temporal domain. Although this poses no significant issues for  
 265 problems in one or two spatial dimensions, it may introduce a severe bot-  
 266 tleneck in higher dimensional problems, as the total number of collocation  
 267 points needed to globally enforce a physics-informed constrain (i.e., in our  
 268 case a partial differential equation) will increase exponentially. Although  
 269 this limitation could be addressed to some extend using sparse grid or quasi  
 270 Monte-Carlo sampling schemes [43, 44], in the next section, we put forth a  
 271 different approach that circumvents the need for collocation points by in-

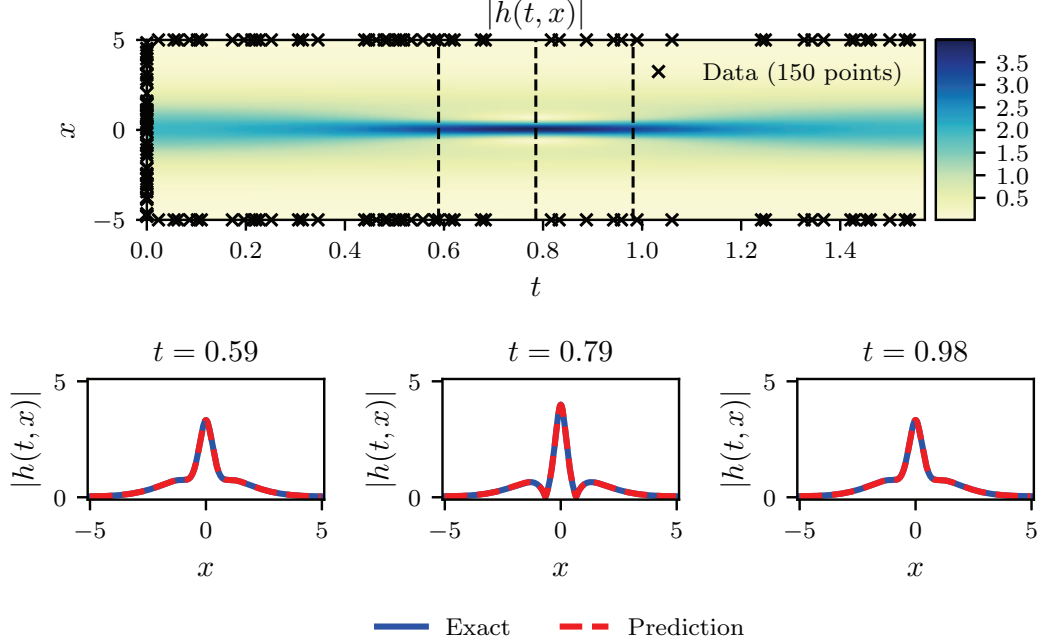


Figure 1: *Schrödinger equation*: Top: Predicted solution  $|h(t, x)|$  along with the initial and boundary training data. In addition we are using 20,000 collocation points generated using a Latin Hypercube Sampling strategy. Bottom: Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel. The relative  $\mathbb{L}_2$  error for this case is  $1.97 \cdot 10^{-3}$ .

272 producing a more structured neural network representation leveraging the  
273 classical Runge-Kutta time-stepping schemes [45].

### 274 3.2. Discrete Time Models

275 Let us apply the general form of Runge-Kutta methods with  $q$  stages [45]  
276 to equation (2) and obtain

$$\begin{aligned} u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \dots, q, \\ u^{n+1} &= u^n - \Delta t \sum_{j=1}^q b_j \mathcal{N}[u^{n+c_j}]. \end{aligned} \quad (7)$$

277 Here,  $u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)$  for  $j = 1, \dots, q$ . This general form en-  
278 capsulates both implicit and explicit time-stepping schemes, depending on  
279 the choice of the parameters  $\{a_{ij}, b_j, c_j\}$ . Equations (7) can be equivalently

expressed as

$$\begin{aligned} u^n &= u_i^n, \quad i = 1, \dots, q, \\ u^n &= u_{q+1}^n, \end{aligned} \quad (8)$$

where

$$\begin{aligned} u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \dots, q, \\ u_{q+1}^n &:= u^{n+1} + \Delta t \sum_{j=1}^q b_j \mathcal{N}[u^{n+c_j}]. \end{aligned} \quad (9)$$

We proceed by placing a multi-output neural network prior on

$$[u^{n+c_1}(x), \dots, u^{n+c_q}(x), u^{n+1}(x)]. \quad (10)$$

This prior assumption along with equations (9) result in a *physics-informed neural network* that takes  $x$  as an input and outputs

$$[u_1^n(x), \dots, u_q^n(x), u_{q+1}^n(x)]. \quad (11)$$

### 3.2.1. Example (Allen-Cahn Equation)

This example aims to highlight the ability of the proposed discrete time models to handle different types of nonlinearity in the governing partial differential equation. To this end, let us consider the Allen-Cahn equation along with periodic boundary conditions

$$\begin{aligned} u_t - 0.0001u_{xx} + 5u^3 - 5u &= 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(0, x) &= x^2 \cos(\pi x), \\ u(t, -1) &= u(t, 1), \\ u_x(t, -1) &= u_x(t, 1). \end{aligned} \quad (12)$$

The Allen-Cahn equation is a well-known equation from the area of reaction-diffusion systems. It describes the process of phase separation in multi-component alloy systems, including order-disorder transitions. For the Allen-Cahn equation, the nonlinear operator in equation (9) is given by

$$\mathcal{N}[u^{n+c_j}] = -0.0001u_{xx}^{n+c_j} + 5(u^{n+c_j})^3 - 5u^{n+c_j},$$

and the shared parameters of the neural networks (10) and (11) can be learned by minimizing the sum of squared errors

$$SSE = SSE_n + SSE_b, \quad (13)$$

296 where

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

297 and

$$\begin{aligned} SSE_b &= \sum_{i=1}^q |u^{n+c_i}(-1) - u^{n+c_i}(1)|^2 + |u^{n+1}(-1) - u^{n+1}(1)|^2 \\ &+ \sum_{i=1}^q |u_x^{n+c_i}(-1) - u_x^{n+c_i}(1)|^2 + |u_x^{n+1}(-1) - u_x^{n+1}(1)|^2. \end{aligned}$$

298 Here,  $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$  corresponds to the data at time-step  $t^n$ . In classical nu-  
 299 merical analysis, these time-steps are usually confined to be small due to sta-  
 300 bility constraints for explicit schemes or computational complexity constraints  
 301 for implicit formulations [45]. These constraints become more severe as the  
 302 total number of Runge-Kutta stages  $q$  is increased, and, for most problems  
 303 of practical interest, one needs to take thousands to millions of such steps  
 304 until the solution is resolved up to a desired final time. In sharp contrast to  
 305 classical methods, here we can employ implicit Runge-Kutta schemes with  
 306 an arbitrarily large number of stages at effectively very little extra cost.<sup>1</sup>  
 307 This enables us to take very large time steps while retaining stability and  
 308 high predictive accuracy, therefore allowing us to resolve the entire spatio-  
 309 temporal solution in a single step.

310  
 311 In this example, we have generated a training and test data-set set by  
 312 simulating the Allen-Cahn equation (12) using conventional spectral meth-  
 313 ods. Specifically, starting from an initial condition  $u(0, x) = x^2 \cos(\pi x)$  and  
 314 assuming periodic boundary conditions  $u(t, -1) = u(t, 1)$  and  $u_x(t, -1) =$   
 315  $u_x(t, 1)$ , we have integrated equation (12) up to a final time  $t = 1.0$  using the  
 316 Chebfun package [40] with a spectral Fourier discretization with 512 modes  
 317 and a fourth-order explicit Runge-Kutta temporal integrator with time-step  
 318  $\Delta t = 10^{-5}$ .

319  
 320 Our training data-set consists of  $N_n = 200$  initial data points that are  
 321 randomly sub-sampled from the exact solution at time  $t = 0.1$ , and our goal

---

<sup>1</sup>To be precise, it is only the number of parameters in the last layer of the neural network that increases linearly with the total number of stages.

is to predict the solution at time  $t = 0.9$  using a single time-step with size  $\Delta t = 0.8$ . To this end, we employ a discrete time *physics-informed neural network* with 4 hidden layers and 200 neurons per layer, while the output layer predicts 101 quantities of interest corresponding to the  $q = 100$  Runge-Kutta stages  $u^{n+c_i}(x)$ ,  $i = 1, \dots, q$ , and the solution at final time  $u^{n+1}(x)$ . The theoretical error estimates for this scheme predict a temporal error accumulation of  $\mathcal{O}(\Delta t^{2q})$  [45], which in our case translates into an error way below machine precision, i.e.,  $\Delta t^{2q} = 0.8^{200} \approx 10^{-20}$ . To our knowledge, this is the first time that an implicit Runge-Kutta scheme of that high-order has ever been used. Remarkably, starting from smooth initial data at  $t = 0.1$  we can predict the nearly discontinuous solution at  $t = 0.9$  in a single time-step with a relative  $\mathbb{L}_2$  error of  $6.99 \cdot 10^{-3}$ , as illustrated in Figure 2. This error is entirely attributed to the neural network's capacity to approximate  $u(t, x)$ , as well as to the degree that the sum of squared errors loss allows interpolation of the training data.

The key parameters controlling the performance of our discrete time algorithm are the total number of Runge-Kutta stages  $q$  and the time-step size  $\Delta t$ . As we demonstrate in the systematic studies provided in Appendix A and Appendix B, low-order methods, such as the case  $q = 1$  corresponding to the classical trapezoidal rule, and the case  $q = 2$  corresponding to the 4<sup>th</sup>-order Gauss-Legendre method, cannot retain their predictive accuracy for large time-steps, thus mandating a solution strategy with multiple time-steps of small size. On the other hand, the ability to push the number of Runge-Kutta stages to 32 and even higher allows us to take very large time steps, and effectively resolve the solution in a single step without sacrificing the accuracy of our predictions. Moreover, numerical stability is not sacrificed either as implicit Gauss-Legendre is the only family of time-stepping schemes that remain A-stable regardless of their order, thus making them ideal for stiff problems [45]. These properties are unprecedented for an algorithm of such implementation simplicity, and illustrate one of the key highlights of our discrete time approach.

#### 4. Data-driven discovery of partial differential equations

In the current part of our study, we shift our attention to the problem of data-driven discovery of partial differential equations [5, 9, 14]. In sections 4.1 and 4.2, we put forth two distinct types of algorithms, namely continuous

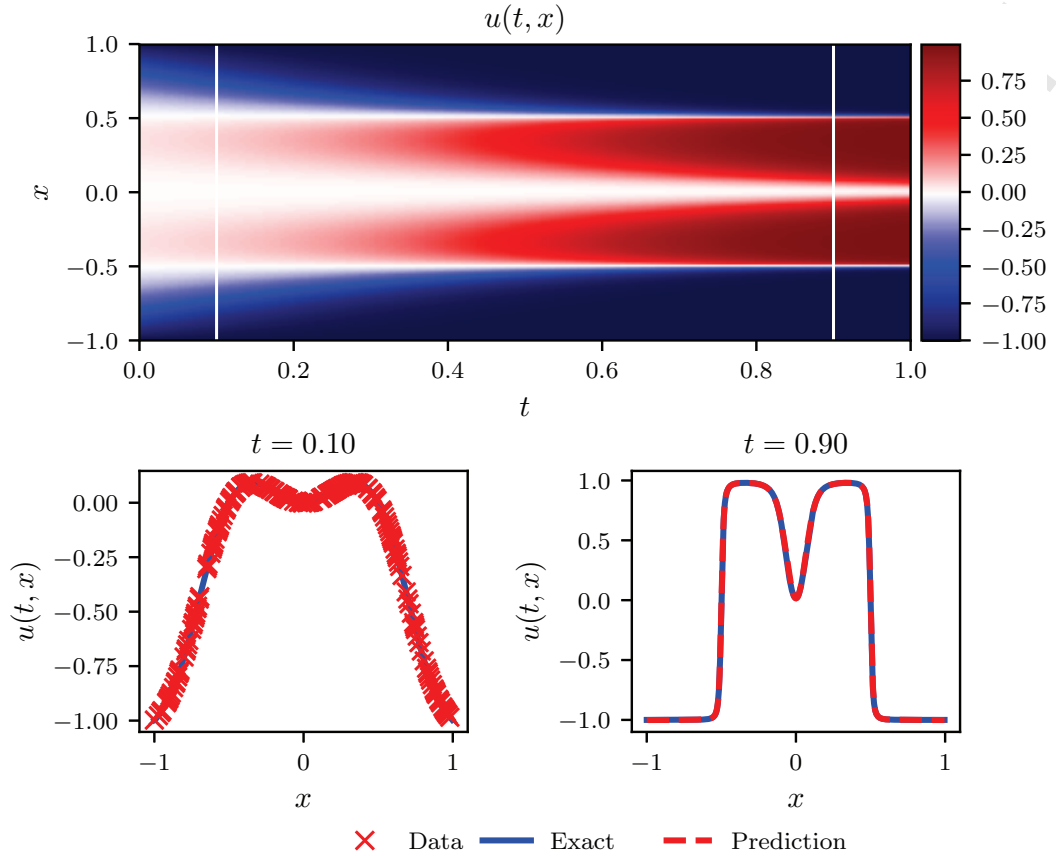


Figure 2: *Allen-Cahn equation*: Top: Solution  $u(t, x)$  along with the location of the initial training snapshot at  $t = 0.1$  and the final prediction snapshot at  $t = 0.9$ . Bottom: Initial training data and final prediction at the snapshots depicted by the white vertical lines in the top panel. The relative  $L_2$  error for this case is  $6.99 \cdot 10^{-3}$ .

and discrete time models, and highlight their properties and performance through the lens of various canonical problems.

#### 4.1. Continuous Time Models

Let us recall equation (1) and similar to section 3.1 define  $f(t, x)$  to be given by the left-hand-side of equation (1); i.e.,

$$f := u_t + \mathcal{N}[u; \lambda]. \quad (14)$$

We proceed by approximating  $u(t, x)$  by a deep neural network. This assumption along with equation (14) result in a *physics-informed neural net-*



work  $f(t, x)$ . This network can be derived by applying the chain rule for differentiating compositions of functions using automatic differentiation [12]. It is worth highlighting that the parameters of the differential operator  $\lambda$  turn into parameters of the *physics-informed neural network*  $f(t, x)$ .

#### 4.1.1. Example (Navier-Stokes Equation)

Our next example involves a realistic scenario of incompressible fluid flow described by the ubiquitous Navier-Stokes equations. Navier-Stokes equations describe the physics of many phenomena of scientific and engineering interest. They may be used to model the weather, ocean currents, water flow in a pipe and air flow around a wing. The Navier-Stokes equations in their full and simplified forms help with the design of aircrafts and cars, the study of blood flow, the design of power stations, the analysis of the dispersion of pollutants, and many other applications. Let us consider the Navier-Stokes equations in two dimensions<sup>2</sup> (2D) given explicitly by

$$\begin{aligned} u_t + \lambda_1(uu_x + vu_y) &= -p_x + \lambda_2(u_{xx} + u_{yy}), \\ v_t + \lambda_1(uv_x + vv_y) &= -p_y + \lambda_2(v_{xx} + v_{yy}), \end{aligned} \quad (15)$$

where  $u(t, x, y)$  denotes the  $x$ -component of the velocity field,  $v(t, x, y)$  the  $y$ -component, and  $p(t, x, y)$  the pressure. Here,  $\lambda = (\lambda_1, \lambda_2)$  are the unknown parameters. Solutions to the Navier-Stokes equations are searched in the set of divergence-free functions; i.e.,

$$u_x + v_y = 0. \quad (16)$$

This extra equation is the continuity equation for incompressible fluids that describes the conservation of mass of the fluid. We make the assumption that

$$u = \psi_y, \quad v = -\psi_x, \quad (17)$$

for some latent function  $\psi(t, x, y)$ .<sup>3</sup> Under this assumption, the continuity equation (16) will be automatically satisfied. Given noisy measurements

$$\{t^i, x^i, y^i, u^i, v^i\}_{i=1}^N$$

<sup>2</sup>It is straightforward to generalize the proposed framework to the Navier-Stokes equations in three dimensions (3D).

<sup>3</sup>This construction can be generalized to three dimensional problems by employing the notion of vector potentials.

of the velocity field, we are interested in learning the parameters  $\lambda$  as well as the pressure  $p(t, x, y)$ . We define  $f(t, x, y)$  and  $g(t, x, y)$  to be given by

$$\begin{aligned} f &:= u_t + \lambda_1(uu_x + vv_y) + p_x - \lambda_2(u_{xx} + u_{yy}), \\ g &:= v_t + \lambda_1(uv_x + vv_y) + p_y - \lambda_2(v_{xx} + v_{yy}), \end{aligned} \quad (18)$$

and proceed by jointly approximating  $[\psi(t, x, y) \ p(t, x, y)]$  using a single neural network with two outputs. This prior assumption along with equations (17) and (18) results into a *physics-informed neural network*  $[f(t, x, y) \ g(t, x, y)]$ . The parameters  $\lambda$  of the Navier-Stokes operator as well as the parameters of the neural networks  $[\psi(t, x, y) \ p(t, x, y)]$  and  $[f(t, x, y) \ g(t, x, y)]$  can be trained by minimizing the mean squared error loss

$$\begin{aligned} MSE &:= \frac{1}{N} \sum_{i=1}^N (|u(t^i, x^i, y^i) - u^i|^2 + |v(t^i, x^i, y^i) - v^i|^2) \\ &+ \frac{1}{N} \sum_{i=1}^N (|f(t^i, x^i, y^i)|^2 + |g(t^i, x^i, y^i)|^2). \end{aligned} \quad (19)$$

Here we consider the prototype problem of incompressible flow past a circular cylinder; a problem known to exhibit rich dynamic behavior and transitions for different regimes of the Reynolds number  $Re = u_\infty D / \nu$ . Assuming a non-dimensional free stream velocity  $u_\infty = 1$ , cylinder diameter  $D = 1$ , and kinematic viscosity  $\nu = 0.01$ , the system exhibits a periodic steady state behavior characterized by a asymmetrical vortex shedding pattern in the cylinder wake, known as the Kármán vortex street [46].

To generate a high-resolution data set for this problem we have employed the spectral/*hp*-element solver NekTar [47]. Specifically, the solution domain is discretized in space by a tessellation consisting of 412 triangular elements, and within each element the solution is approximated as a linear combination of a tenth-order hierarchical, semi-orthogonal Jacobi polynomial expansion [47]. We have assumed a uniform free stream velocity profile imposed at the left boundary, a zero pressure outflow condition imposed at the right boundary located 25 diameters downstream of the cylinder, and periodicity for the top and bottom boundaries of the  $[-15, 25] \times [-8, 8]$  domain. We integrate equation (15) using a third-order stiffly stable scheme [47] until the system reaches a periodic steady state, as depicted in figure 3(a). In what follows, a small portion of the resulting data-set corresponding to this steady state

416 solution will be used for model training, while the remaining data will be  
417 used to validate our predictions. For simplicity, we have chosen to confine  
418 our sampling in a rectangular region downstream of cylinder as shown in  
419 figure 3(a).

420  
421 Given scattered and potentially noisy data on the stream-wise  $u(t, x, y)$   
422 and transverse  $v(t, x, y)$  velocity components, our goal is to identify the un-  
423 known parameters  $\lambda_1$  and  $\lambda_2$ , as well as to obtain a qualitatively accurate  
424 reconstruction of the entire pressure field  $p(t, x, y)$  in the cylinder wake, which  
425 by definition can only be identified up to a constant. To this end, we have  
426 created a training data-set by randomly sub-sampling the full high-resolution  
427 data-set. To highlight the ability of our method to learn from scattered and  
428 scarce training data, we have chosen  $N = 5,000$ , corresponding to a mere  
429 1% of the total available data as illustrated in figure 3(b). Also plotted are  
430 representative snapshots of the predicted velocity components  $u(t, x, y)$  and  
431  $v(t, x, y)$  after the model was trained. The neural network architecture used  
432 here consists of 9 layers with 20 neurons in each layer.

433  
434 A summary of our results for this example is presented in figure 4. We  
435 observe that the *physics-informed neural network* is able to correctly identify  
436 the unknown parameters  $\lambda_1$  and  $\lambda_2$  with very high accuracy even when the  
437 training data was corrupted with noise. Specifically, for the case of noise-  
438 free training data, the error in estimating  $\lambda_1$  and  $\lambda_2$  is 0.078%, and 4.67%,  
439 respectively. The predictions remain robust even when the training data are  
440 corrupted with 1% uncorrelated Gaussian noise, returning an error of 0.17%,  
441 and 5.70%, for  $\lambda_1$  and  $\lambda_2$ , respectively.

442  
443 A more intriguing result stems from the network’s ability to provide a  
444 qualitatively accurate prediction of the entire pressure field  $p(t, x, y)$  in the  
445 absence of any training data on the pressure itself. A visual comparison  
446 against the exact pressure solution is presented in figure 4 for a represen-  
447 tative pressure snapshot. Notice that the difference in magnitude between  
448 the exact and the predicted pressure is justified by the very nature of the  
449 *incompressible* Navier-Stokes system, as the pressure field is only identifiable  
450 up to a constant. This result of inferring a continuous quantity of interest  
451 from auxiliary measurements by leveraging the underlying physics is a great  
452 example of the enhanced capabilities that *physics-informed neural networks*  
453 have to offer, and highlights their potential in solving high-dimensional in-

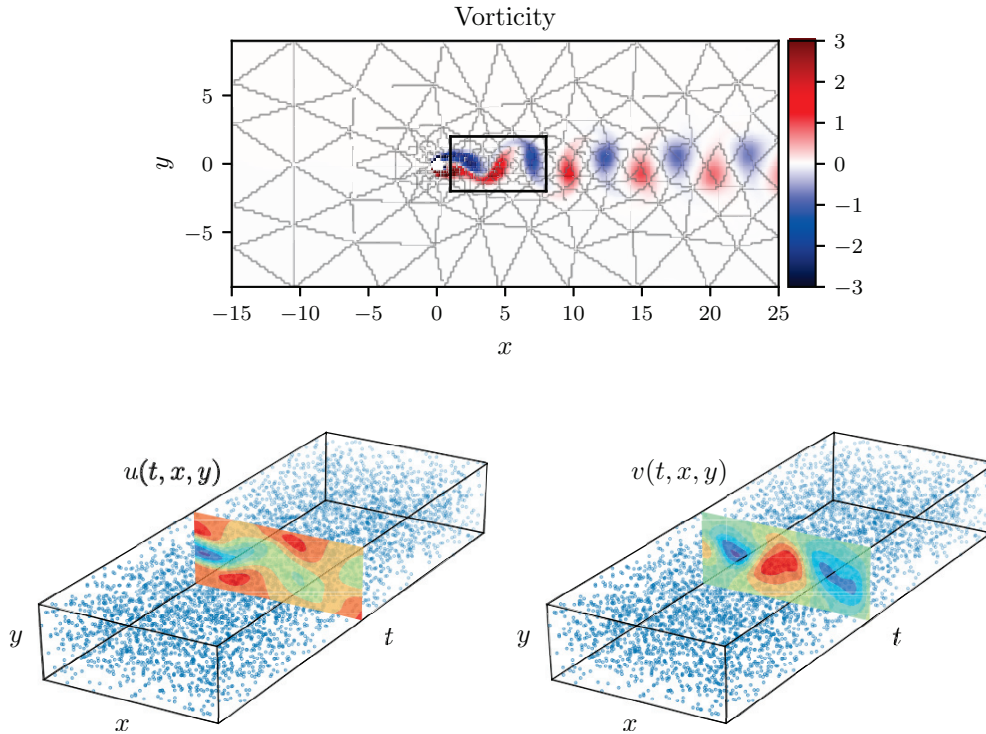


Figure 3: *Navier-Stokes equation*: *Top*: Incompressible flow and dynamic vortex shedding past a circular cylinder at  $Re = 100$ . The spatio-temporal training data correspond to the depicted rectangular region in the cylinder wake. *Bottom*: Locations of training data-points for the the stream-wise and transverse velocity components,  $u(t, x, y)$  and  $v(t, x, y)$ , respectively.

verse problems.

Our approach so far assumes availability of scattered data throughout the entire spatio-temporal domain. However, in many cases of practical interest, one may only be able to observe the system at distinct time instants. In the next section, we introduce a different approach that tackles the data-driven discovery problem using only two data snapshots. We will see how, by leveraging the classical Runge-Kutta time-stepping schemes, one can construct discrete time *physics-informed neural networks* that can retain high predictive accuracy even when the temporal gap between the data snapshots is very large.

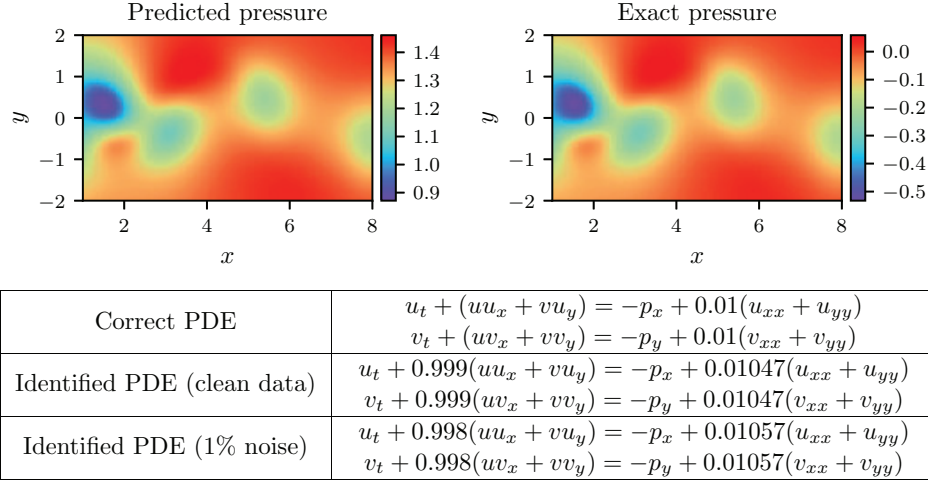


Figure 4: *Navier-Stokes equation*: Top: Predicted versus exact instantaneous pressure field  $p(t, x, y)$  at a representative time instant. By definition, the pressure can be recovered up to a constant, hence justifying the different magnitude between the two plots. This remarkable qualitative agreement highlights the ability of *physics-informed neural networks* to identify the entire pressure field, despite the fact that no data on the pressure are used during model training. Bottom: Correct partial differential equation along with the identified one obtained by learning  $\lambda_1, \lambda_2$  and  $p(t, x, y)$ .

#### 4.2. Discrete Time Models

We begin by applying the general form of Runge-Kutta methods [45] with  $q$  stages to equation (1) and obtain

$$\begin{aligned} u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \dots, q, \\ u^{n+1} &= u^n - \Delta t \sum_{j=1}^q b_j \mathcal{N}[u^{n+c_j}; \lambda]. \end{aligned} \quad (20)$$

Here,  $u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)$  is the hidden state of the system at time  $t^n + c_j \Delta t$  for  $j = 1, \dots, q$ . This general form encapsulates both implicit and explicit time-stepping schemes, depending on the choice of the parameters  $\{a_{ij}, b_j, c_j\}$ . Equations (20) can be equivalently expressed as

$$\begin{aligned} u^n &= u_i^n, \quad i = 1, \dots, q, \\ u^{n+1} &= u_i^{n+1}, \quad i = 1, \dots, q. \end{aligned} \quad (21)$$

472 where

$$\begin{aligned} u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \dots, q, \\ u_i^{n+1} &:= u^{n+c_i} + \Delta t \sum_{j=1}^q (a_{ij} - b_j) \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \dots, q. \end{aligned} \quad (22)$$

473 We proceed by placing a multi-output neural network prior on

$$[u^{n+c_1}(x), \dots, u^{n+c_q}(x)]. \quad (23)$$

474 This prior assumption along with equations (22) result in two *physics-informed*  
475 *neural networks*

$$[u_1^n(x), \dots, u_q^n(x), u_{q+1}^n(x)], \quad (24)$$

476 and

$$[u_1^{n+1}(x), \dots, u_q^{n+1}(x), u_{q+1}^{n+1}(x)]. \quad (25)$$

477 Given noisy measurements at two distinct temporal snapshots  $\{\mathbf{x}^n, \mathbf{u}^n\}$  and  
478  $\{\mathbf{x}^{n+1}, \mathbf{u}^{n+1}\}$  of the system at times  $t^n$  and  $t^{n+1}$ , respectively, the shared  
479 parameters of the neural networks (23), (24), and (25) along with the pa-  
480 rameters  $\lambda$  of the differential operator can be trained by minimizing the sum  
481 of squared errors

$$SSE = SSE_n + SSE_{n+1}, \quad (26)$$

482 where

$$SSE_n := \sum_{j=1}^q \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

483 and

$$SSE_{n+1} := \sum_{j=1}^q \sum_{i=1}^{N_{n+1}} |u_j^{n+1}(x^{n+1,i}) - u^{n+1,i}|^2.$$

484 Here,  $\mathbf{x}^n = \{x^{n,i}\}_{i=1}^{N_n}$ ,  $\mathbf{u}^n = \{u^{n,i}\}_{i=1}^{N_n}$ ,  $\mathbf{x}^{n+1} = \{x^{n+1,i}\}_{i=1}^{N_{n+1}}$ , and  $\mathbf{u}^{n+1} =$   
485  $\{u^{n+1,i}\}_{i=1}^{N_{n+1}}$ .

#### 486 4.2.1. Example (Korteweg-de Vries Equation)

487 Our final example aims to highlight the ability of the proposed frame-  
488 work to handle governing partial differential equations involving higher or-  
489 der derivatives. Here, we consider a mathematical model of waves on shallow  
490 water surfaces; the Korteweg-de Vries (KdV) equation. This equation can  
491 also be viewed as Burgers equation with an added dispersive term. The KdV

equation has several connections to physical problems. It describes the evolution of long one-dimensional waves in many physical settings. Such physical settings include shallow-water waves with weakly non-linear restoring forces, long internal waves in a density-stratified ocean, ion acoustic waves in a plasma, and acoustic waves on a crystal lattice. Moreover, the KdV equation is the governing equation of the string in the Fermi-Pasta-Ulam problem [48] in the continuum limit. The KdV equation reads as

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xxx} = 0, \quad (27)$$

with  $(\lambda_1, \lambda_2)$  being the unknown parameters. For the KdV equation, the nonlinear operator in equations (22) is given by

$$\mathcal{N}[u^{n+c_j}] = \lambda_1 u^{n+c_j} u_x^{n+c_j} - \lambda_2 u_{xxx}^{n+c_j}$$

and the shared parameters of the neural networks (23), (24), and (25) along with the parameters  $\lambda = (\lambda_1, \lambda_2)$  of the KdV equation can be learned by minimizing the sum of squared errors (26).

To obtain a set of training and test data we simulated (27) using conventional spectral methods. Specifically, starting from an initial condition  $u(0, x) = \cos(\pi x)$  and assuming periodic boundary conditions, we have integrated equation (27) up to a final time  $t = 1.0$  using the Chebfun package [40] with a spectral Fourier discretization with 512 modes and a fourth-order explicit Runge-Kutta temporal integrator with time-step  $\Delta t = 10^{-6}$ . Using this data-set, we then extract two solution snapshots at time  $t^n = 0.2$  and  $t^{n+1} = 0.8$ , and randomly sub-sample them using  $N_n = 199$  and  $N_{n+1} = 201$  to generate a training data-set. We then use these data to train a discrete time *physics-informed neural network* by minimizing the sum of squared error loss of equation (26) using L-BFGS [35]. The network architecture used here comprises of 4 hidden layers, 50 neurons per layer, and an output layer predicting the solution at the  $q$  Runge-Kutta stages, i.e.,  $u^{n+c_j}(x)$ ,  $j = 1, \dots, q$ , where  $q$  is empirically chosen to yield a temporal error accumulation of the order of machine precision  $\epsilon$  by setting<sup>4</sup>

$$q = 0.5 \log \epsilon / \log(\Delta t), \quad (28)$$

---

<sup>4</sup>This is motivated by the theoretical error estimates for implicit Runge-Kutta schemes suggesting a truncation error of  $\mathcal{O}(\Delta t^{2q})$  [45].



520 where the time-step for this example is  $\Delta t = 0.6$ .

521

522 The results of this experiment are summarized in figure 5. In the top  
523 panel, we present the exact solution  $u(t, x)$ , along with the locations of the  
524 two data snapshots used for training. A more detailed overview of the exact  
525 solution and the training data is given in the middle panel. It is worth notic-  
526 ing how the complex nonlinear dynamics of equation (27) causes dramatic  
527 differences in the form of the solution between the two reported snapshots.  
528 Despite these differences, and the large temporal gap between the two train-  
529 ing snapshots, our method is able to correctly identify the unknown param-  
530 eters regardless of whether the training data is corrupted with noise or not.  
531 Specifically, for the case of noise-free training data, the error in estimating  
532  $\lambda_1$  and  $\lambda_2$  is 0.023%, and 0.006%, respectively, while the case with 1% noise  
533 in the training data returns errors of 0.057%, and 0.017%, respectively.

## 534 5. Conclusions

535 We have introduced *physics-informed neural networks*, a new class of  
536 universal function approximators that is capable of encoding any underlying  
537 physical laws that govern a given data-set, and can be described by par-  
538 tial differential equations. In this work, we design data-driven algorithms for  
539 inferring solutions to general nonlinear partial differential equations, and con-  
540 structing computationally efficient physics-informed surrogate models. The  
541 resulting methods showcase a series of promising results for a diverse collec-  
542 tion of problems in computational science, and open the path for endowing  
543 deep learning with the powerful capacity of mathematical physics to model  
544 the world around us. As deep learning technology is continuing to grow  
545 rapidly both in terms of methodological and algorithmic developments, we  
546 believe that this is a timely contribution that can benefit practitioners across  
547 a wide range of scientific domains. Specific applications that can readily en-  
548 joy these benefits include, but are not limited to, data-driven forecasting of  
549 physical processes, model predictive control, multi-physics/multi-scale mod-  
550 eling and simulation.

551

552 We must note however that the proposed methods should not be viewed  
553 as replacements of classical numerical methods for solving partial differential  
554 equations (e.g., finite elements, spectral methods, etc.). Such methods have  
555 matured over the last 50 years and, in many cases, meet the robustness and



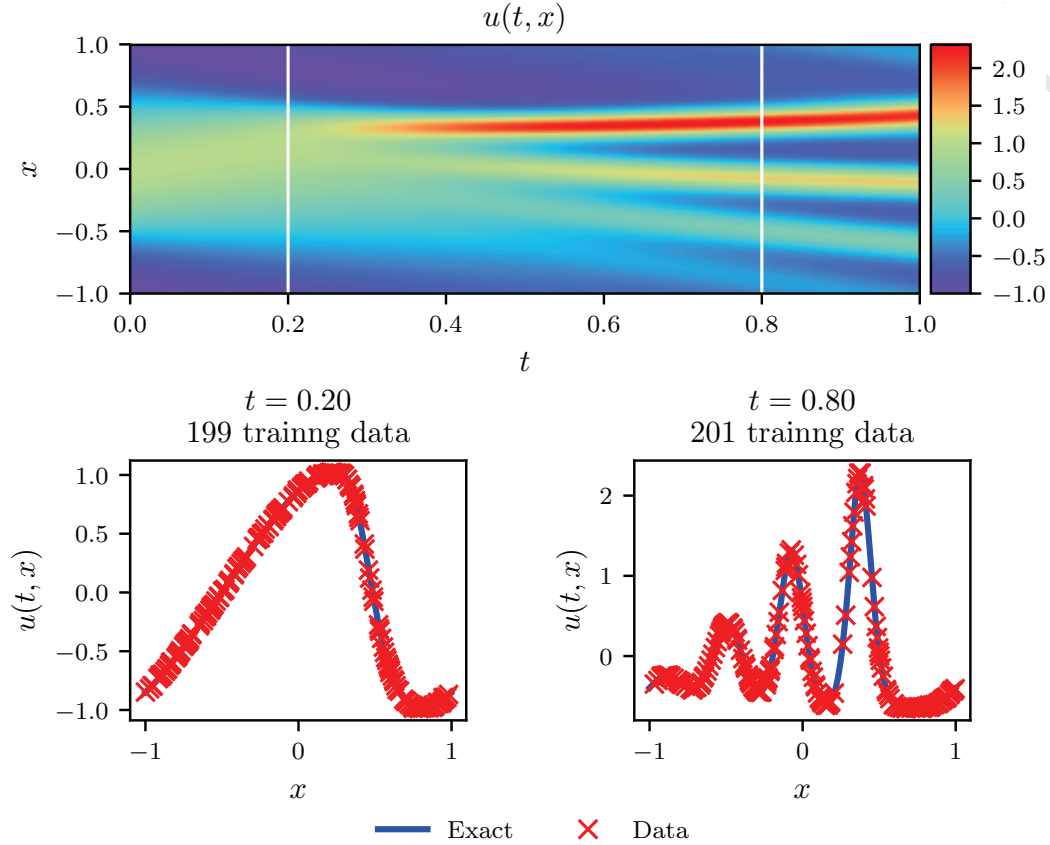


Figure 5: *KdV equation*: *Top*: Solution  $u(t, x)$  along with the temporal locations of the two training snapshots. *Middle*: Training data and exact solution corresponding to the two temporal snapshots depicted by the dashed vertical lines in the top panel. *Bottom*: Correct partial differential equation along with the identified one obtained by learning  $\lambda_1, \lambda_2$ .

556 computational efficiency standards required in practice. Our message here,  
 557 as advocated in Section 3.2, is that classical methods such as the Runge-  
 558 Kutta time-stepping schemes can coexist in harmony with deep neural net-  
 559 works, and offer invaluable intuition in constructing structured predictive

560 algorithms. Moreover, the implementation simplicity of the latter greatly  
561 favors rapid development and testing of new ideas, potentially opening the  
562 path for a new era in data-driven scientific computing.

563  
564 Although a series of promising results was presented, the reader may per-  
565 haps agree this work creates more questions than it answers. How deep/wide  
566 should the neural network be? How much data is really needed? Why does  
567 the algorithm converge to unique values for the parameters of the differen-  
568 tial operators, i.e., why is the algorithm not suffering from local optima for  
569 the parameters of the differential operator? Does the network suffer from  
570 vanishing gradients for deeper architectures and higher order differential op-  
571 erators? Could this be mitigated by using different activation functions?  
572 Can we improve on initializing the network weights or normalizing the data?  
573 Are the mean square error and the sum of squared errors the appropriate  
574 loss functions? Why are these methods seemingly so robust to noise in the  
575 data? How can we quantify the uncertainty associated with our predictions?  
576 Throughout this work, we have attempted to answer some of these questions,  
577 but we have observed that specific settings that yielded impressive results for  
578 one equation could fail for another. Admittedly, more work is needed collec-  
579 tively to set the foundations in this field.

580  
581 In a broader context, and along the way of seeking answers to those  
582 questions, we believe that this work advocates a fruitful synergy between  
583 machine learning and classical computational physics that has the potential  
584 to enrich both fields and lead to high-impact developments.

## 585 Acknowledgements

586 This work received support by the DARPA EQUiPS grant N66001-15-2-  
587 4055 and the AFOSR grant FA9550-17-1-0013.

## 588 Appendix A. Data-driven solution of partial differential equations

589 This Appendix accompanies the main manuscript, and contains a series  
590 of systematic studies that aim to demonstrate the performance of the pro-  
591 posed algorithms for problems pertaining to *data-driven solution of partial*  
592 *differential equations*. Throughout this document, we will use the Burgers'  
593 equation as a canonical example.

594 *Appendix A.1. Continuous Time Models*

595 In one space dimension, the Burger's equation along with Dirichlet bound-  
596 ary conditions reads as

$$\begin{aligned} u_t + uu_x - (0.01/\pi)u_{xx} &= 0, \quad x \in [-1, 1], \quad t \in [0, 1], \\ u(0, x) &= -\sin(\pi x), \\ u(t, -1) &= u(t, 1) = 0. \end{aligned} \quad (\text{A.1})$$

597 Let us define  $f(t, x)$  to be given by

$$f := u_t + uu_x - (0.01/\pi)u_{xx},$$

598 and proceed by approximating  $u(t, x)$  by a deep neural network. To highlight  
599 the simplicity in implementing this idea we have included a Python code  
600 snippet using Tensorflow [49]; currently one of the most popular and well  
601 documented open source libraries for machine learning computations. To  
602 this end,  $u(t, x)$  can be simply defined as

```
603 def u(t, x):
604     u = neural_net(tf.concat([t, x], 1), weights, biases)
605     return u
```

606 Correspondingly, the *physics-informed neural network*  $f(t, x)$  takes the form

```
607 def f(t, x):
608     u = u(t, x)
609     u_t = tf.gradients(u, t)[0]
610     u_x = tf.gradients(u, x)[0]
611     u_xx = tf.gradients(u_x, x)[0]
612     f = u_t + u*u_x - (0.01/tf.pi)*u_xx
613     return f
```

614 The shared parameters between the neural networks  $u(t, x)$  and  $f(t, x)$  can  
615 be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f, \quad (\text{A.2})$$

616 where

$$MSE_u = \frac{1}{N_u} \sum_{i=1}^{N_u} |u(t_u^i, x_u^i) - u^i|^2,$$

617 and

$$MSE_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2.$$

618 Here,  $\{t_u^i, x_u^i, u^i\}_{i=1}^{N_u}$  denote the initial and boundary training data on  $u(t, x)$   
 619 and  $\{t_f^i, x_f^i\}_{i=1}^{N_f}$  specify the collocations points for  $f(t, x)$ . The loss  $MSE_u$   
 620 corresponds to the initial and boundary data while  $MSE_f$  enforces the struc-  
 621 ture imposed by equation (A.1) at a finite set of collocation points. Although  
 622 similar ideas for constraining neural networks using physical laws have been  
 623 explored in previous studies [15, 16], here we revisit them using modern  
 624 computational tools, and apply them to more challenging dynamic problems  
 625 described by time-dependent nonlinear partial differential equations.

626  
 627 The Burgers equation is often considered as a prototype example of a  
 628 hyperbolic conservation law (as  $\nu \rightarrow 0$ ). Notice that if we want to “fabri-  
 629 cate” an “exact” solution to this equation we would select a solution  $u(t, x)$   
 630 (e.g.,  $e^{-t} \sin(\pi x)$ ) and obtain the corresponding right hand side  $f(t, x)$  by  
 631 differentiation. The resulting  $u(t, x)$  and  $f(t, x)$  are “guaranteed” to satisfy  
 632 the Burgers equation and conserve all associated invariances by construction.  
 633 In our work, we replace  $u(t, x)$  by a neural network  $u(t, x; W, b)$  and obtain  
 634 a physics-informed neural network  $f(t, x; W, b)$  by automatic differentiation.  
 635 Consequently, the resulting pair  $u(t, x; W, b)$  and  $f(t, x; W, b)$  must satisfy  
 636 the Burgers equation regardless of the choice of the weights  $W$  and bias  $b$   
 637 parameters. Hence, at this “prior” level, i.e. before we train the networks  
 638 on a given set of data, our model should exactly preserves the continuity  
 639 and momentum equations by construction. During training, given a data-set  
 640  $t_i, x_i, u_i$  and  $t_j, x_j, f_j$ , we then try to find the “correct” parameters  $W^*$  and  $b^*$   
 641 such that we get as good a fit as possible to both the observed data and the  
 642 differential equation residual. During this process the residual, albeit small,  
 643 will not be exactly zero, and therefore our approximation will conserve mass  
 644 and momentum within the accuracy of the residual loss. Similar behavior is  
 645 observed in classical Galerkin finite element methods, while the only numer-  
 646 ical methods that are known to have exact conservation properties in this  
 647 setting are discontinuous Galerkin and finite volumes.

648  
 649 In all benchmarks considered in this work, the total number of training  
 650 data  $N_u$  is relatively small (a few hundred up to a few thousand points), and  
 651 we chose to optimize all loss functions using L-BFGS a quasi-Newton, full-

batch gradient-based optimization algorithm [35]. For larger data-sets a more computationally efficient mini-batch setting can be readily employed using stochastic gradient descent and its modern variants [36, 37]. Despite the fact that there is no theoretical guarantee that this procedure converges to a global minimum, our empirical evidence indicates that, if the given partial differential equation is well-posed and its solution is unique, our method is capable of achieving good prediction accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points  $N_f$ . This general observation deeply relates to the resulting optimization landscape induced by the mean square error loss of equation 4, and defines an open question for research that is in sync with recent theoretical developments in deep learning [38, 39]. Here, we will test the robustness of the proposed methodology using a series of systematic sensitivity studies that accompany the numerical results presented in the following.

Figure A.6 summarizes our results for the data-driven solution of the Burgers equation. Specifically, given a set of  $N_u = 100$  randomly distributed initial and boundary data, we learn the latent solution  $u(t, x)$  by training all 3021 parameters of a 9-layer deep neural network using the mean squared error loss of (A.2). Each hidden layer contained 20 neurons and a hyperbolic tangent activation function. The top panel of Figure A.6 shows the predicted spatio-temporal solution  $u(t, x)$ , along with the locations of the initial and boundary training data. We must underline that, unlike any classical numerical method for solving partial differential equations, this prediction is obtained without any sort of discretization of the spatio-temporal domain. The exact solution for this problem is analytically available [13], and the resulting prediction error is measured at  $6.7 \cdot 10^{-4}$  in the relative  $\mathbb{L}_2$ -norm. Note that this error is about two orders of magnitude lower than the one reported in our previous work on data-driven solution of partial differential equation using Gaussian processes [8]. A more detailed assessment of the predicted solution is presented in the bottom panel of figure A.6. In particular, we present a comparison between the exact and the predicted solutions at different time instants  $t = 0.25, 0.50, 0.75$ . Using only a handful of initial and boundary data, the *physics-informed neural network* can accurately capture the intricate nonlinear behavior of the Burgers' equation that leads to the development of a sharp internal layer around  $t = 0.4$ . The latter is notoriously hard to accurately resolve with classical numerical methods and requires a laborious spatio-temporal discretization of equation (A.1).

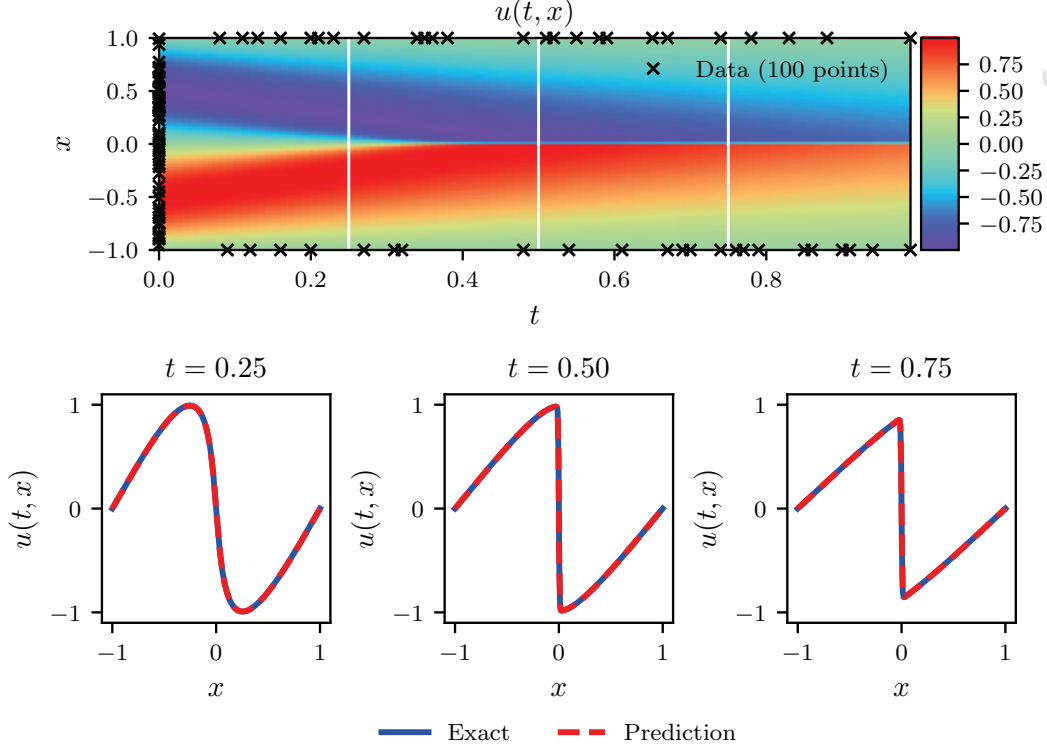


Figure A.6: *Burgers' equation*: *Top*: Predicted solution  $u(t, x)$  along with the initial and boundary training data. In addition we are using 10,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom*: Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the white vertical lines in the top panel. The relative  $\mathbb{L}_2$  error for this case is  $6.7 \cdot 10^{-4}$ . Model training took approximately 60 seconds on a single NVIDIA Titan X GPU card.

690

691 To further analyze the performance of our method, we have performed  
 692 the following systematic studies to quantify its predictive accuracy for differ-  
 693 ent number of training and collocation points, as well as for different neural  
 694 network architectures. In table A.1 we report the resulting relative  $\mathbb{L}_2$  error  
 695 for different number of initial and boundary training data  $N_u$  and different  
 696 number of collocation points  $N_f$ , while keeping the 9-layer network archi-  
 697 tecture fixed. The general trend shows increased prediction accuracy as the  
 698 total number of training data  $N_u$  is increased, given a sufficient number of  
 699 collocation points  $N_f$ . This observation highlights a key strength of *physics-*  
 700 *informed neural networks*: by encoding the structure of the underlying phys-

$N_u \backslash N_f$	2000	4000	6000	7000	8000	10000
20	2.9e-01	4.4e-01	8.9e-01	1.2e+00	9.9e-02	4.2e-02
40	6.5e-02	1.1e-02	5.0e-01	9.6e-03	4.6e-01	7.5e-02
60	3.6e-01	1.2e-02	1.7e-01	5.9e-03	1.9e-03	8.2e-03
80	5.5e-03	1.0e-03	3.2e-03	7.8e-03	4.9e-02	4.5e-03
100	6.6e-02	2.7e-01	7.2e-03	6.8e-04	2.2e-03	6.7e-04
200	1.5e-01	2.3e-03	8.2e-04	8.9e-04	6.1e-04	4.9e-04

Table A.1: *Burgers' equation*: Relative  $\mathbb{L}_2$  error between the predicted and the exact solution  $u(t, x)$  for different number of initial and boundary training data  $N_u$ , and different number of collocation points  $N_f$ . Here, the network architecture is fixed to 9 layers with 20 neurons per hidden layer.

$\text{Layers} \backslash \text{Neurons}$	10	20	40
2	7.4e-02	5.3e-02	1.0e-01
4	3.0e-03	9.4e-04	6.4e-04
6	9.6e-03	1.3e-03	6.1e-04
8	2.5e-03	9.6e-04	5.6e-04

Table A.2: *Burgers' equation*: Relative  $\mathbb{L}_2$  error between the predicted and the exact solution  $u(t, x)$  for different number of hidden layers and different number of neurons per layer. Here, the total number of training and collocation points is fixed to  $N_u = 100$  and  $N_f = 10,000$ , respectively.

701 ical law through the collocation points  $N_f$ , one can obtain a more accurate  
702 and data-efficient learning algorithm.<sup>5</sup> Finally, table A.2 shows the resulting  
703 relative  $\mathbb{L}_2$  for different number of hidden layers, and different number of  
704 neurons per layer, while the total number of training and collocation points  
705 is kept fixed to  $N_u = 100$  and  $N_f = 10,000$ , respectively. As expected, we  
706 observe that as the number of layers and neurons is increased (hence the  
707 capacity of the neural network to approximate more complex functions), the  
708 predictive accuracy is increased.

---

<sup>5</sup>Note that the case  $N_f = 0$  corresponds to a standard neural network model, i.e., a neural network that does not take into account the underlying governing equation.

709 *Appendix A.2. Discrete Time Models*

710 Let us apply the general form of Runge-Kutta methods with  $q$  stages [45]  
711 to a general equation of the form

$$u_t + \mathcal{N}[u] = 0, \quad x \in \Omega, \quad t \in [0, T], \quad (\text{A.3})$$

712 and obtain

$$\begin{aligned} u^{n+c_i} &= u^n - \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \dots, q, \\ u^{n+1} &= u^n - \Delta t \sum_{j=1}^q b_j \mathcal{N}[u^{n+c_j}]. \end{aligned} \quad (\text{A.4})$$

713 Here,  $u^{n+c_j}(x) = u(t^n + c_j \Delta t, x)$  for  $j = 1, \dots, q$ . This general form en-  
714 capsulates both implicit and explicit time-stepping schemes, depending on  
715 the choice of the parameters  $\{a_{ij}, b_j, c_j\}$ . Equations (7) can be equivalently  
716 expressed as

$$\begin{aligned} u^n &= u_i^n, \quad i = 1, \dots, q, \\ u^n &= u_{q+1}^n, \end{aligned} \quad (\text{A.5})$$

717 where

$$\begin{aligned} u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}], \quad i = 1, \dots, q, \\ u_{q+1}^n &:= u^{n+1} + \Delta t \sum_{j=1}^q b_j \mathcal{N}[u^{n+c_j}]. \end{aligned} \quad (\text{A.6})$$

718 We proceed by placing a multi-output neural network prior on

$$[u^{n+c_1}(x), \dots, u^{n+c_q}(x), u^{n+1}(x)]. \quad (\text{A.7})$$

719 This prior assumption along with equations (A.6) result in a *physics-informed*  
720 *neural network* that takes  $x$  as an input and outputs

$$[u_1^n(x), \dots, u_q^n(x), u_{q+1}^n(x)]. \quad (\text{A.8})$$

721 To highlight the key features of the discrete time representation we revisit  
722 the problem of data-driven solution of the Burgers' equation. For this case,  
723 the nonlinear operator in equation (A.6) is given by

$$\mathcal{N}[u^{n+c_j}] = u^{n+c_j} u_x^{n+c_j} - (0.01/\pi) u_{xx}^{n+c_j},$$

724 and the shared parameters of the neural networks (A.7) and (A.8) can be  
725 learned by minimizing the sum of squared errors

$$SSE = SSE_n + SSE_b, \quad (\text{A.9})$$



726 where

$$SSE_n = \sum_{j=1}^{q+1} \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

727 and

$$SSE_b = \sum_{i=1}^q (|u^{n+c_i}(-1)|^2 + |u^{n+c_i}(1)|^2) + |u^{n+1}(-1)|^2 + |u^{n+1}(1)|^2.$$

728 Here,  $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$  corresponds to the data at time  $t^n$ . The Runge-Kutta  
 729 scheme now allows us to infer the latent solution  $u(t, x)$  in a sequential fash-  
 730 ion. Starting from initial data  $\{x^{n,i}, u^{n,i}\}_{i=1}^{N_n}$  at time  $t^n$  and data at the  
 731 domain boundaries  $x = -1$  and  $x = 1$ , we can use the aforementioned loss  
 732 function (A.9) to train the networks of (A.7), (A.8), and predict the solu-  
 733 tion at time  $t^{n+1}$ . A Runge-Kutta time-stepping scheme would then use this  
 734 prediction as initial data for the next step and proceed to train again and  
 735 predict  $u(t^{n+2}, x)$ ,  $u(t^{n+3}, x)$ , etc., one step at a time.

736

737 The result of applying this process to the Burgers' equation is presented  
 738 in figure A.7. For illustration purposes, we start with a set of  $N_n = 250$  initial  
 739 data at  $t = 0.1$ , and employ a *physics-informed neural network* induced by an  
 740 implicit Runge-Kutta scheme with 500 stages to predict the solution at time  
 741  $t = 0.9$  in a single step. The theoretical error estimates for this scheme predict  
 742 a temporal error accumulation of  $\mathcal{O}(\Delta t^{2q})$  [45], which in our case translates  
 743 into an error way below machine precision, i.e.,  $\Delta t^{2q} = 0.8^{1000} \approx 10^{-97}$ . To  
 744 our knowledge, this is the first time that an implicit Runge-Kutta scheme  
 745 of that high-order has ever been used. Remarkably, starting from smooth  
 746 initial data at  $t = 0.1$  we can predict the nearly discontinuous solution at  
 747  $t = 0.9$  in a single time-step with a relative  $\mathbb{L}_2$  error of  $8.2 \cdot 10^{-4}$ . This error is  
 748 two orders of magnitude lower than the one reported in [8], and it is entirely  
 749 attributed to the neural network's capacity to approximate  $u(t, x)$ , as well as  
 750 to the degree that the sum of squared errors loss allows interpolation of the  
 751 training data. The network architecture used here consists of 4 layers with  
 752 50 neurons in each hidden layer.

753

754 A detailed systematic study to quantify the effect of different network  
 755 architectures is presented in table A.3. By keeping the number of Runge-  
 756 Kutta stages fixed to  $q = 500$  and the time-step size to  $\Delta t = 0.8$ , we have

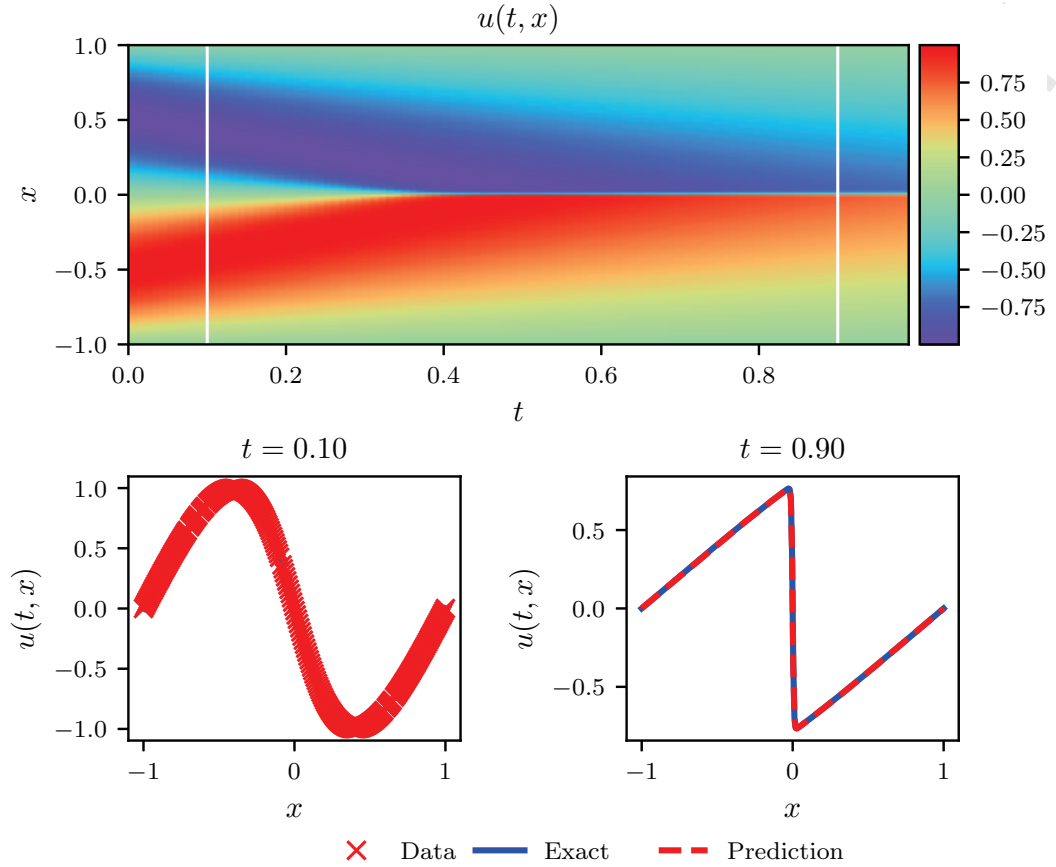


Figure A.7: *Burgers equation*: Top: Solution  $u(t, x)$  along with the location of the initial training snapshot at  $t = 0.1$  and the final prediction snapshot at  $t = 0.9$ . Bottom: Initial training data and final prediction at the snapshots depicted by the white vertical lines in the top panel. The relative  $\mathbb{L}_2$  error for this case is  $8.2 \cdot 10^{-4}$ .

757 varied the number of hidden layers and the number of neurons per layer, and  
 758 monitored the resulting relative  $\mathbb{L}_2$  error for the predicted solution at time  
 759  $t = 0.9$ . Evidently, as the neural network capacity is increased the predictive  
 760 accuracy is enhanced.

761

762 The key parameters controlling the performance of our discrete time al-  
 763 gorithm are the total number of Runge-Kutta stages  $q$  and the time-step size  
 764  $\Delta t$ . In table A.4 we summarize the results of an extensive systematic study  
 765 where we fix the network architecture to 4 hidden layers with 50 neurons

Layers \ Neurons	Neurons		
	10	25	50
1	4.1e-02	4.1e-02	1.5e-01
2	2.7e-03	5.0e-03	2.4e-03
3	3.6e-03	1.9e-03	9.5e-04

Table A.3: *Burgers' equation*: Relative final prediction error measure in the  $\mathbb{L}_2$  norm for different number of hidden layers and neurons in each layer. Here, the number of Runge-Kutta stages is fixed to 500 and the time-step size to  $\Delta t = 0.8$ .

per layer, and vary the number of Runge-Kutta stages  $q$  and the time-step size  $\Delta t$ . Specifically, we see how cases with low numbers of stages fail to yield accurate results when the time-step size is large. For instance, the case  $q = 1$  corresponding to the classical trapezoidal rule, and the case  $q = 2$  corresponding to the 4<sup>th</sup>-order Gauss-Legendre method, cannot retain their predictive accuracy for time-steps larger than 0.2, thus mandating a solution strategy with multiple time-steps of small size. On the other hand, the ability to push the number of Runge-Kutta stages to 32 and even higher allows us to take very large time steps, and effectively resolve the solution in a single step without sacrificing the accuracy of our predictions. Moreover, numerical stability is not sacrificed either as implicit Gauss-Legendre is the only family of time-stepping schemes that remain A-stable regardless of their order, thus making them ideal for stiff problems [45]. These properties are unprecedented for an algorithm of such implementation simplicity, and illustrate one of the key highlights of our discrete time approach.

Finally, in table A.5 we provide a systematic study to quantify the accuracy of the predicted solution as we vary the spatial resolution of the input data. As expected, increasing the total number of training data results in enhanced prediction accuracy.

## Appendix B. Data-driven discovery of partial differential equations

This Appendix accompanies the main manuscript, and contains a series of systematic studies that aim to demonstrate the performance of the proposed algorithms for problems pertaining to *data-driven discovery of partial differential equations*. Throughout this document, we will use the Burgers' equation as a canonical example.

$\Delta t \backslash q$	0.2	0.4	0.6	0.8
1	3.5e-02	1.1e-01	2.3e-01	3.8e-01
2	5.4e-03	5.1e-02	9.3e-02	2.2e-01
4	1.2e-03	1.5e-02	3.6e-02	5.4e-02
8	6.7e-04	1.8e-03	8.7e-03	5.8e-02
16	5.1e-04	7.6e-02	8.4e-04	1.1e-03
32	7.4e-04	5.2e-04	4.2e-04	7.0e-04
64	4.5e-04	4.8e-04	1.2e-03	7.8e-04
100	5.1e-04	5.7e-04	1.8e-02	1.2e-03
500	4.1e-04	3.8e-04	4.2e-04	8.2e-04

Table A.4: *Burgers' equation*: Relative final prediction error measured in the  $\mathbb{L}_2$  norm for different number of Runge-Kutta stages  $q$  and time-step sizes  $\Delta t$ . Here, the network architecture is fixed to 4 hidden layers with 50 neurons in each layer.

$N$	250	200	150	100	50	10
Error	4.02e-4	2.93e-3	9.39e-3	5.54e-2	1.77e-2	7.58e-1

Table A.5: *Burgers equation*: Relative  $\mathcal{L}_2$  error between the predicted and the exact solution  $u(t, x)$  for different number of training data  $N_n$ . Here, we have fixed  $q = 500$ , and used a neural network architecture with 3 hidden layers and 50 neurons per hidden layer.

### 791 Appendix B.1. Continuous Time Models

792 As a first example, let us consider the Burgers' equation. This equation  
793 arises in various areas of applied mathematics, including fluid mechanics,  
794 nonlinear acoustics, gas dynamics, and traffic flow [13]. It is a fundamen-  
795 tal partial differential equation and can be derived from the Navier-Stokes  
796 equations for the velocity field by dropping the pressure gradient term. For  
797 small values of the viscosity parameters, Burgers' equation can lead to shock  
798 formation that is notoriously hard to resolve by classical numerical methods.  
799 In one space dimension the equation reads as

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0. \quad (\text{B.1})$$

800 Let us define  $f(t, x)$  to be given by

$$f := u_t + \lambda_1 u u_x - \lambda_2 u_{xx}, \quad (\text{B.2})$$

801 and proceed by approximating  $u(t, x)$  by a deep neural network. This will  
802 result in the *physics-informed neural network*  $f(t, x)$ . The shared parameters

of the neural networks  $u(t, x)$  and  $f(t, x)$  along with the parameters  $\lambda = (\lambda_1, \lambda_2)$  of the differential operator can be learned by minimizing the mean squared error loss

$$MSE = MSE_u + MSE_f, \quad (\text{B.3})$$

where

$$MSE_u = \frac{1}{N} \sum_{i=1}^N |u(t_u^i, x_u^i) - u^i|^2,$$

and

$$MSE_f = \frac{1}{N} \sum_{i=1}^N |f(t_u^i, x_u^i)|^2.$$

Here,  $\{t_u^i, x_u^i, u^i\}_{i=1}^N$  denote the training data on  $u(t, x)$ . The loss  $MSE_u$  corresponds to the training data on  $u(t, x)$  while  $MSE_f$  enforces the structure imposed by equation (B.1) at a finite set of collocation points, whose number and location is taken to be the same as the training data.

To illustrate the effectiveness of our approach, we have created a training data-set by randomly generating  $N = 2,000$  points across the entire spatio-temporal domain from the exact solution corresponding to  $\lambda_1 = 1.0$  and  $\lambda_2 = 0.01/\pi$ . The locations of the training points are illustrated in the top panel of figure B.8. This data-set is then used to train a 9-layer deep neural network with 20 neurons per hidden layer by minimizing the mean square error loss of (B.3) using the L-BFGS optimizer [35]. Upon training, the network is calibrated to predict the entire solution  $u(t, x)$ , as well as the unknown parameters  $\lambda = (\lambda_1, \lambda_2)$  that define the underlying dynamics. A visual assessment of the predictive accuracy of the *physics-informed neural network* is given in the middle and bottom panels of figure B.8. The network is able to identify the underlying partial differential equation with remarkable accuracy, even in the case where the scattered training data is corrupted with 1% uncorrelated noise.

To further scrutinize the performance of our algorithm, we have performed a systematic study with respect to the total number of training data, the noise corruption levels, and the neural network architecture. The results are summarized in tables B.6 and B.7. The key observation here is that the proposed methodology appears to be very robust with respect to noise levels in the data, and yields a reasonable identification accuracy even for noise

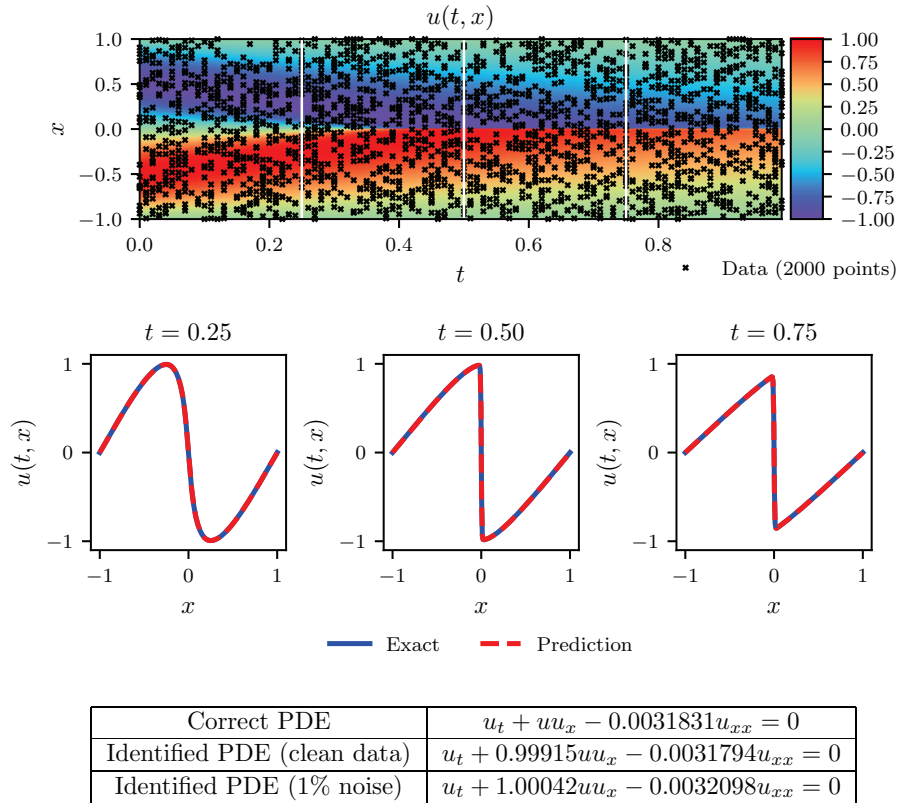


Figure B.8: *Burgers equation*: *Top*: Predicted solution  $u(t, x)$  along with the training data. *Middle*: Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel. *Bottom*: Correct partial differential equation along with the identified one obtained by learning  $\lambda_1$  and  $\lambda_2$ .

corruption up to 10%. This enhanced robustness seems to greatly outperform competing approaches using Gaussian process regression as previously reported in [9] as well as approaches relying on sparse regression that require relatively clean data for accurately computing numerical gradients [50]. We also observe some variability and non monotonic trends in tables B.6 and B.7 as the network architecture and total number of training points are changed. This variability could be potentially attributed to different factors pertaining to the equation itself as well as the particular neural network setup, and gives rise to a series of questions that require further investigation, as discussed in the concluding remarks section of this paper.

noise $N_u$	% error in $\lambda_1$				% error in $\lambda_2$			
	0%	1%	5%	10%	0%	1%	5%	10%
500	0.131	0.518	0.118	1.319	13.885	0.483	1.708	4.058
1000	0.186	0.533	0.157	1.869	3.719	8.262	3.481	14.544
1500	0.432	0.033	0.706	0.725	3.093	1.423	0.502	3.156
2000	0.096	0.039	0.190	0.101	0.469	0.008	6.216	6.391

Table B.6: *Burgers' equation*: Percentage error in the identified parameters  $\lambda_1$  and  $\lambda_2$  for different number of training data  $N$  corrupted by different noise levels. Here, the neural network architecture is kept fixed to 9 layers and 20 neurons per layer.

Neurons Layers	% error in $\lambda_1$			% error in $\lambda_2$		
	10	20	40	10	20	40
2	11.696	2.837	1.679	103.919	67.055	49.186
4	0.332	0.109	0.428	4.721	1.234	6.170
6	0.668	0.629	0.118	3.144	3.123	1.158
8	0.414	0.141	0.266	8.459	1.902	1.552

Table B.7: *Burgers' equation*: Percentage error in the identified parameters  $\lambda_1$  and  $\lambda_2$  for different number of hidden layers and neurons per layer. Here, the training data is considered to be noise-free and fixed to  $N = 2,000$ .

## 844 Appendix B.2. Discrete Time Models

845 Our starting point here is similar to as described in section 3.2. Now  
846 equations (7) can be equivalently expressed as

$$\begin{aligned} u^n &= u_i^n, \quad i = 1, \dots, q, \\ u^{n+1} &= u_i^{n+1}, \quad i = 1, \dots, q. \end{aligned} \tag{B.4}$$

847 where

$$\begin{aligned} u_i^n &:= u^{n+c_i} + \Delta t \sum_{j=1}^q a_{ij} \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \dots, q, \\ u_i^{n+1} &:= u^{n+c_i} + \Delta t \sum_{j=1}^q (a_{ij} - b_j) \mathcal{N}[u^{n+c_j}; \lambda], \quad i = 1, \dots, q. \end{aligned} \tag{B.5}$$

848 We proceed by placing a multi-output neural network prior on

$$[u^{n+c_1}(x), \dots, u^{n+c_q}(x)]. \tag{B.6}$$

849 This prior assumption along with equations (22) result in two *physics-informed*  
850 *neural networks*

$$[u_1^n(x), \dots, u_q^n(x), u_{q+1}^n(x)], \quad (\text{B.7})$$

851 and

$$[u_1^{n+1}(x), \dots, u_q^{n+1}(x), u_{q+1}^{n+1}(x)]. \quad (\text{B.8})$$

852 Given noisy measurements at two distinct temporal snapshots  $\{\mathbf{x}^n, \mathbf{u}^n\}$  and  
853  $\{\mathbf{x}^{n+1}, \mathbf{u}^{n+1}\}$  of the system at times  $t^n$  and  $t^{n+1}$ , respectively, the shared  
854 parameters of the neural networks (B.6), (B.7), and (B.8) along with the  
855 parameters  $\lambda$  of the differential operator can be trained by minimizing the  
856 sum of squared errors

$$SSE = SSE_n + SSE_{n+1}, \quad (\text{B.9})$$

857 where

$$SSE_n := \sum_{j=1}^q \sum_{i=1}^{N_n} |u_j^n(x^{n,i}) - u^{n,i}|^2,$$

858 and

$$SSE_{n+1} := \sum_{j=1}^q \sum_{i=1}^{N_{n+1}} |u_j^{n+1}(x^{n+1,i}) - u^{n+1,i}|^2.$$

859 Here,  $\mathbf{x}^n = \{x^{n,i}\}_{i=1}^{N_n}$ ,  $\mathbf{u}^n = \{u^{n,i}\}_{i=1}^{N_n}$ ,  $\mathbf{x}^{n+1} = \{x^{n+1,i}\}_{i=1}^{N_{n+1}}$ , and  $\mathbf{u}^{n+1} =$   
860  $\{u^{n+1,i}\}_{i=1}^{N_{n+1}}$ .

### 861 *Appendix B.3. Example (Burgers' Equation)*

862 Let us illustrate the key features of this method through the lens of the  
863 Burgers' equation. Recall the equation's form

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0, \quad (\text{B.10})$$

864 and notice that the nonlinear spatial operator in equation (B.5) is given by

$$\mathcal{N}[u^{n+c_j}] = \lambda_1 u^{n+c_j} u_x^{n+c_j} - \lambda_2 u_{xx}^{n+c_j}.$$

865 Given merely two training data snapshots, the shared parameters of the neu-  
866 ral networks (B.6), (B.7), and (B.8) along with the parameters  $\lambda = (\lambda_1, \lambda_2)$   
867 of the Burgers' equation can be learned by minimizing the sum of squared  
868 errors in equation (B.9). Here, we have created a training data-set compris-  
869 ing of  $N_n = 199$  and  $N_{n+1} = 201$  spatial points by randomly sampling the



exact solution at time instants  $t^n = 0.1$  and  $t^{n+1} = 0.9$ , respectively. The training data are shown in the top and middle panel of figure B.9. The neural network architecture used here consists of 4 hidden layers with 50 neurons each, while the number of Runge-Kutta stages is empirically chosen to yield a temporal error accumulation of the order of machine precision  $\epsilon$  by setting<sup>6</sup>

$$q = 0.5 \log \epsilon / \log(\Delta t), \quad (\text{B.11})$$

where the time-step for this example is  $\Delta t = 0.8$ . The bottom panel of figure B.9 summarizes the identified parameters  $\lambda = (\lambda_1, \lambda_2)$  for the cases of noise-free data, as well as noisy data with 1% of Gaussian uncorrelated noise corruption. For both cases, the proposed algorithm is able to learn the correct parameter values  $\lambda_1 = 1.0$  and  $\lambda_2 = 0.01/\pi$  with remarkable accuracy, despite the fact that the two data snapshots used for training are very far apart, and potentially describe different regimes of the underlying dynamics.

A sensitivity analysis is performed to quantify the accuracy of our predictions with respect to the gap between the training snapshots  $\Delta t$ , the noise levels in the training data, and the *physics-informed neural network* architecture. As shown in table B.8, the proposed algorithm is quite robust to both  $\Delta t$  and the noise corruption levels, and it returns reasonable estimates for the unknown parameters. This robustness is mainly attributed to the flexibility of the underlying implicit Runge-Kutta scheme to admit an arbitrarily high number of stages, allowing the data snapshots to be very far apart in time, while not compromising the accuracy with which the nonlinear dynamics of equation (B.10) are resolved. This is the key highlight of our discrete time formulation for identification problems, setting it apart from competing approaches [9, 50]. Lastly, table B.9 presents the percentage error in the identified parameters, demonstrating the robustness of our estimates with respect to the underlying neural network architecture. Despite the overall positive results, the variability observed in tables B.8 and B.9 is still largely unexplained and naturally motivates a series of questions provided in the concluding remarks section of this paper.

---

<sup>6</sup>This is motivated by the theoretical error estimates for implicit Runge-Kutta schemes suggesting a truncation error of  $\mathcal{O}(\Delta t^{2q})$  [45].

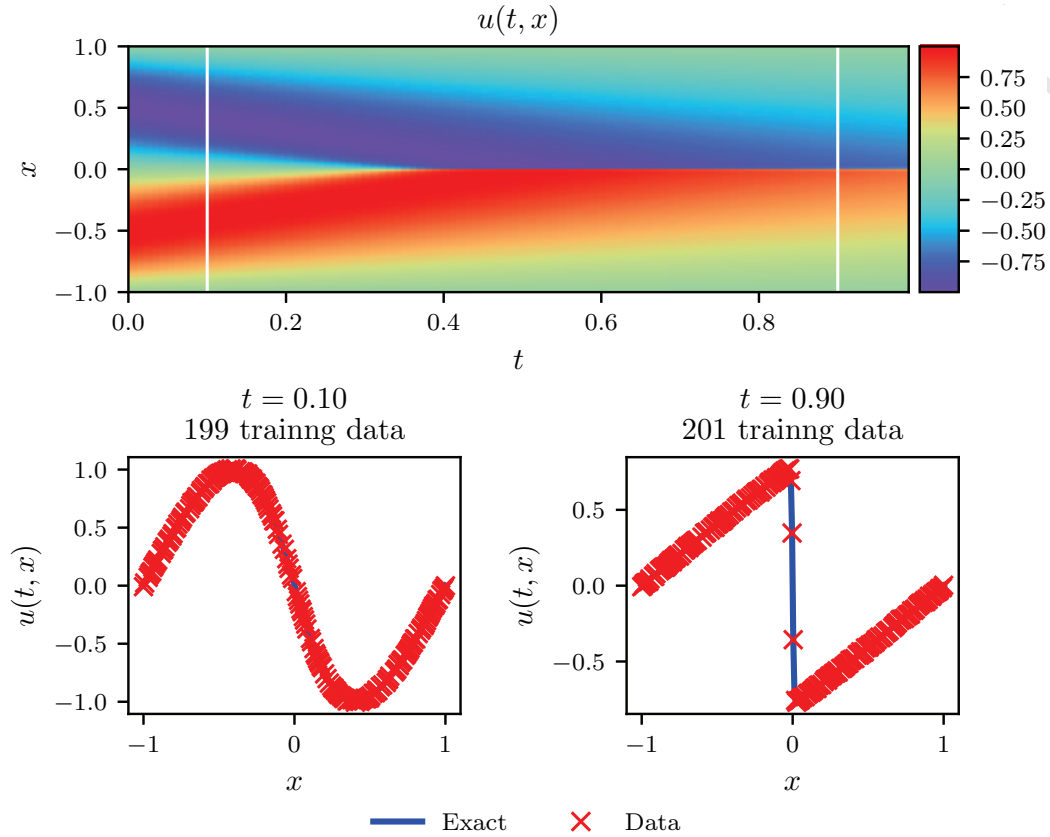


Figure B.9: *Burgers equation*: Top: Solution  $u(t, x)$  along with the temporal locations of the two training snapshots. Middle: Training data and exact solution corresponding to the two temporal snapshots depicted by the dashed vertical lines in the top panel. Bottom: Correct partial differential equation along with the identified one obtained by learning  $\lambda_1, \lambda_2$ .

## References

- [1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

$\Delta t$ \ noise	% error in $\lambda_1$				% error in $\lambda_2$			
	0%	1%	5%	10%	0%	1%	5%	10%
0.2	0.002	0.435	6.073	3.273	0.151	4.982	59.314	83.969
0.4	0.001	0.119	1.679	2.985	0.088	2.816	8.396	8.377
0.6	0.002	0.064	2.096	1.383	0.090	0.068	3.493	24.321
0.8	0.010	0.221	0.097	1.233	1.918	3.215	13.479	1.621

Table B.8: *Burgers' equation*: Percentage error in the identified parameters  $\lambda_1$  and  $\lambda_2$  for different gap size  $\Delta t$  between two different snapshots and for different noise levels.

Layers \ Neurons	% error in $\lambda_1$			% error in $\lambda_2$		
	10	25	50	10	25	50
1	1.868	4.868	1.960	180.373	237.463	123.539
2	0.443	0.037	0.015	29.474	2.676	1.561
3	0.123	0.012	0.004	7.991	1.906	0.586
4	0.012	0.020	0.011	1.125	4.448	2.014

Table B.9: *Burgers' equation*: Percentage error in the identified parameters  $\lambda_1$  and  $\lambda_2$  for different number of hidden layers and neurons in each layer.

- 904 [2] B. M. Lake, R. Salakhutdinov, J. B. Tenenbaum, Human-level concept  
905 learning through probabilistic program induction, *Science* 350 (2015)  
906 1332–1338.
- 907 [3] B. Alipanahi, A. Delong, M. T. Weirauch, B. J. Frey, Predicting the se-  
908 quence specificities of DNA-and RNA-binding proteins by deep learning,  
909 *Nature biotechnology* 33 (2015) 831–838.
- 910 [4] M. Raissi, P. Perdikaris, G. E. Karniadakis, Inferring solutions of dif-  
911 ferential equations using noisy multi-fidelity data, *Journal of Computa-  
912 tional Physics* 335 (2017) 736–746.
- 913 [5] M. Raissi, P. Perdikaris, G. E. Karniadakis, Machine learning of linear  
914 differential equations using Gaussian processes, *Journal of Computa-  
915 tional Physics* 348 (2017) 683 – 693.
- 916 [6] H. Owhadi, Bayesian numerical homogenization, *Multiscale Modeling  
917 & Simulation* 13 (2015) 812–828.

- 918 [7] C. E. Rasmussen, C. K. Williams, Gaussian processes for machine learn-  
919 ing, volume 1, MIT press Cambridge, 2006.
- 920 [8] M. Raissi, P. Perdikaris, G. E. Karniadakis, Numerical Gaussian pro-  
921 cesses for time-dependent and non-linear partial differential equations,  
922 arXiv preprint arXiv:1703.10230 (2017).
- 923 [9] M. Raissi, G. E. Karniadakis, Hidden physics models: Machine  
924 learning of nonlinear partial differential equations, arXiv preprint  
925 arXiv:1708.00588 (2017).
- 926 [10] H. Owhadi, C. Scovel, T. Sullivan, et al., Brittleness of Bayesian infer-  
927 ence under finite information in a continuous world, Electronic Journal  
928 of Statistics 9 (2015) 1–79.
- 929 [11] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks  
930 are universal approximators, Neural networks 2 (1989) 359–366.
- 931 [12] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, J. M. Siskind, Au-  
932 tomatic differentiation in machine learning: a survey, arXiv preprint  
933 arXiv:1502.05767 (2015).
- 934 [13] C. Basdevant, M. Deville, P. Haldenwang, J. Lacroix, J. Ouazzani,  
935 R. Peyret, P. Orlandi, A. Patera, Spectral and finite difference solu-  
936 tions of the Burgers equation, Computers & fluids 14 (1986) 23–41.
- 937 [14] S. H. Rudy, S. L. Brunton, J. L. Proctor, J. N. Kutz, Data-driven  
938 discovery of partial differential equations, Science Advances 3 (2017).
- 939 [15] I. E. Lagaris, A. Likas, D. I. Fotiadis, Artificial neural networks for  
940 solving ordinary and partial differential equations, IEEE Transactions  
941 on Neural Networks 9 (1998) 987–1000.
- 942 [16] D. C. Psychogios, L. H. Ungar, A hybrid neural network-first principles  
943 approach to process modeling, AIChE Journal 38 (1992) 1499–1511.
- 944 [17] J.-X. Wang, J. Wu, J. Ling, G. Iaccarino, H. Xiao, A comprehensive  
945 physics-informed machine learning framework for predictive turbulence  
946 modeling, arXiv preprint arXiv:1701.07102 (2017).

- [18] Y. Zhu, N. Zabaras, Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification, arXiv preprint arXiv:1801.06879 (2018).
- [19] T. Hagge, P. Stinis, E. Yeung, A. M. Tartakovsky, Solving differential equations with unknown constitutive relations as recurrent neural networks, arXiv preprint arXiv:1710.02242 (2017).
- [20] R. Tripathy, I. Bionis, Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification, arXiv preprint arXiv:1802.00850 (2018).
- [21] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long-short term memory networks, arXiv preprint arXiv:1802.07486 (2018).
- [22] E. J. Parish, K. Duraisamy, A paradigm for data-driven predictive modeling using field inversion and machine learning, *Journal of Computational Physics* 305 (2016) 758–774.
- [23] K. Duraisamy, Z. J. Zhang, A. P. Singh, New approaches in turbulence and transition modeling using data-driven techniques, in: 53rd AIAA Aerospace Sciences Meeting, p. 1284.
- [24] J. Ling, A. Kurzawski, J. Templeton, Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, *Journal of Fluid Mechanics* 807 (2016) 155–166.
- [25] Z. J. Zhang, K. Duraisamy, Machine learning methods for data-driven turbulence modeling, in: 22nd AIAA Computational Fluid Dynamics Conference, p. 2460.
- [26] M. Milano, P. Koumoutsakos, Neural network modeling for near wall turbulent flow, *Journal of Computational Physics* 182 (2002) 1–26.
- [27] P. Perdikaris, D. Venturi, G. E. Karniadakis, Multifidelity information fusion algorithms for high-dimensional systems and massive data sets, *SIAM J. Sci. Comput.* 38 (2016) B521–B538.
- [28] R. Rico-Martinez, J. Anderson, I. Kevrekidis, Continuous-time nonlinear signal processing: a neural network based approach for gray box

- 978 identification, in: Neural Networks for Signal Processing [1994] IV. Pro-  
 979 ceedings of the 1994 IEEE Workshop, IEEE, pp. 596–605.
- 980 [29] J. Ling, J. Templeton, Evaluation of machine learning algorithms for  
 981 prediction of regions of high reynolds averaged navier stokes uncertainty,  
 982 Physics of Fluids 27 (2015) 085103.
- 983 [30] H. W. Lin, M. Tegmark, D. Rolnick, Why does deep and cheap learning  
 984 work so well?, Journal of Statistical Physics 168 (2017) 1223–1247.
- 985 [31] R. Kondor, N-body networks: a covariant hierarchical neural net-  
 986 work architecture for learning atomic potentials, arXiv preprint  
 987 arXiv:1803.01588 (2018).
- 988 [32] R. Kondor, S. Trivedi, On the generalization of equivariance and con-  
 989 volution in neural networks to the action of compact groups, arXiv  
 990 preprint arXiv:1802.03690 (2018).
- 991 [33] M. Hirn, S. Mallat, N. Poilvert, Wavelet scattering regression of quan-  
 992 tum chemical energies, Multiscale Modeling & Simulation 15 (2017)  
 993 827–863.
- 994 [34] S. Mallat, Understanding deep convolutional networks, Phil. Trans. R.  
 995 Soc. A 374 (2016) 20150203.
- 996 [35] D. C. Liu, J. Nocedal, On the limited memory BFGS method for large  
 997 scale optimization, Mathematical programming 45 (1989) 503–528.
- 998 [36] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.
- 999 [37] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv  
 1000 preprint arXiv:1412.6980 (2014).
- 1001 [38] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, Y. LeCun, The  
 1002 loss surfaces of multilayer networks, in: Artificial Intelligence and Statis-  
 1003 tics, pp. 192–204.
- 1004 [39] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural net-  
 1005 works via information, arXiv preprint arXiv:1703.00810 (2017).
- 1006 [40] T. A. Driscoll, N. Hale, L. N. Trefethen, Chebfun guide, 2014.

- 1007 [41] M. Stein, Large sample properties of simulations using latin hypercube  
1008 sampling, *Technometrics* 29 (1987) 143–151.
- 1009 [42] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization  
1010 of machine learning algorithms, in: *Advances in neural information*  
1011 *processing systems*, pp. 2951–2959.
- 1012 [43] H.-J. Bungartz, M. Griebel, Sparse grids, *Acta numerica* 13 (2004)  
1013 147–269.
- 1014 [44] I. H. Sloan, H. Woźniakowski, When are quasi-monte carlo algorithms  
1015 efficient for high dimensional integrals?, *Journal of Complexity* 14 (1998)  
1016 1–33.
- 1017 [45] A. Iserles, *A first course in the numerical analysis of differential equa-*  
1018 *tions*, 44, Cambridge University Press, 2009.
- 1019 [46] T. Von Kármán, *Aerodynamics*, volume 9, McGraw-Hill New York,  
1020 1963.
- 1021 [47] G. Karniadakis, S. Sherwin, *Spectral/hp element methods for computa-*  
1022 *tional fluid dynamics*, Oxford University Press, 2013.
- 1023 [48] T. Dauxois, Fermi, Pasta, Ulam and a mysterious lady, *arXiv preprint*  
1024 *arXiv:0801.1590* (2008).
- 1025 [49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S.  
1026 Corrado, A. Davis, J. Dean, M. Devin, et al., *Tensorflow: Large-scale*  
1027 *machine learning on heterogeneous distributed systems*, *arXiv preprint*  
1028 *arXiv:1603.04467* (2016).
- 1029 [50] S. L. Brunton, J. L. Proctor, J. N. Kutz, Discovering governing equa-  
1030 tions from data by sparse identification of nonlinear dynamical systems,  
1031 *Proceedings of the National Academy of Sciences* 113 (2016) 3932–3937.