Computer Methods in Applied Mechanics and Engineering 421 (2024) 116813

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

Respecting causality for training physics-informed neural networks

Sifan Wang [a], Shyam Sankaran [b], Paris Perdikaris [b,*]

[a] Graduate Group in Applied Mathematics and Computational Science, University of Pennsylvania, Philadelphia, PA 19104, United States of America

[b] Department of Mechanical Engineering and Applied Mechanics, University of Pennsylvania, Philadelphia, PA 19104, United States of America

ARTICLE INFO

Keywords:
Deep learning
Partial differential equations
Computational physics
Chaotic systems

ABSTRACT

While the popularity of physics-informed neural networks (PINNs) is steadily rising, to this date PINNs have not been successful in simulating dynamical systems whose solution exhibits multi-scale, chaotic or turbulent behavior. In this work we attribute this shortcoming to the inability of existing PINNs formulations to respect the spatio-temporal causal structure that is inherent to the evolution of physical systems. We argue that this is a fundamental limitation and a key source of error that can ultimately steer PINN models to converge towards erroneous solutions. We address this pathology by proposing a simple re-formulation of PINNs loss functions that can explicitly account for physical causality during model training. We demonstrate that this simple modification alone is enough to introduce significant accuracy improvements, as well as a practical quantitative mechanism for assessing the convergence of a PINNs model. We provide state-of-the-art numerical results across a series of benchmarks for which existing PINNs formulations fail, including the chaotic Lorenz system, the Kuramoto–Sivashinsky equation in the chaotic regime, and the Navier–Stokes equations. To the best of our knowledge, this is the first time that PINNs have been successful in simulating such systems, introducing new opportunities for their applicability to problems of industrial complexity.

1. Introduction

Physics-informed neural networks (PINNs) have emerged as a promising framework for synthesizing observational data and physical laws across diverse applications in science and engineering [1–8]. However, it is well known that PINNs often face severe difficulties and even fail to tackle problems whose solution exhibits highly nonlinear, multi-scale, or chaotic behavior [9,10]. Over the last few years, a series of extensions to the original formulation of Raissi et al. [11] have been proposed with the sole goal of enhancing the accuracy and robustness of PINNs in tackling increasingly more challenging problems. Such extensions include, but are not limited to, novel optimization algorithms for adaptive training [12–15], adaptive algorithms for selecting batches of training data [16,17], novel network architectures [9,12,18–20], domain decomposition strategies [21,22], new types of activation functions [23], and sequential learning strategies [16,24,25]. Although these techniques have been successful in introducing some improvements in terms of trainability and accuracy, there still exists a vast suite of problems that remain elusive to PINNs. Examples of such problems include systems whose behavior exhibits strong non-linearity, broadband energy spectra, and high sensitivity to initial conditions, such as the chaotic Kuramoto–Sivishinski equation and the Navier–Stokes equations. These are not pathological corner cases, but cases that are extremely relevant across a multitude of realistic scenarios in science and engineering. Therefore, there is a pressing need for understanding why PINNs fall short in such scenarios, and how they can be improved in order to overcome the challenges that currently limit their success to relatively simple problems.

* Corresponding author.
E-mail addresses: sifanw@sas.upenn.edu (S. Wang), shyamss@seas.upenn.edu (S. Sankaran), pgp@seas.upenn.edu (P. Perdikaris).

https://doi.org/10.1016/j.cma.2024.116813
Received 29 November 2023; Received in revised form 24 January 2024; Accepted 24 January 2024
Available online 7 February 2024

Physical systems are known to possess an inherent causal structure. Consider for example a linear wave with some initial velocity that is spreading out with a speed c across a homogeneous medium [26]. It is well-understood that, although a part of the wave may lag behind (if there is an initial velocity), no part can travel faster than speed c. This assertion encapsulates the so-called *principle of causality* that dictates how local changes in the initial/boundary data of a spatio-temporal dynamical system is reflected in its corresponding states at later times [26]. Specific to hyperbolic partial differential equations (PDEs), such as the wave equation, this principle underpins the formulation of the method of characteristics [27] that provides a rigorous set of analytical and numerical tools for efficiently tackling initial value problems. Although characterizing how information propagates in general nonlinear PDEs is a challenging task, basic principles of causality such as temporal precedence and covariation (i.e. statistical dependency between variables that are generated by coupled time evolution) are still expected to hold. This causal structure is also clearly reflected in classical numerical methods, where a PDE is typically discretized in time by sequential algorithms which ensure that the solution at time t is fully resolved before approximating the solution at time $t + \Delta t$. Strikingly, this notion of temporal dependence is absent in most continuous-time PINNs formulations (see e.g. [12,13,21,23,28–30]). In fact, we will show how continuous-time PINNs trained by gradient descent are implicitly biased towards first approximating PDE solutions at later times, before even resolving the initial conditions, therefore violating temporal causality. Consequently, it is no surprise that such formulations are fragile and often fail to simulate forward problems, especially in cases where the target solutions exhibit strong dependence on initial data (e.g. chaotic systems). Recent studies [16,24,25] have proposed remedies to this issue by empirically introducing sequential training strategies, yet a concrete justification of why such strategies appear to be effective is still missing.

This work is focused on investigating the importance of respecting physical causality during the training of continuous-time PINNs. Specifically, we reveal an implicit bias suggesting that continuous-time PINNs models can violate causality, and hence are susceptible to converge towards erroneous solutions. To address this fundamental shortcoming, we reformulate the PDE residual loss and introduce additional temporal weights, which explicitly respect the causal structure that characterizes the solution of general nonlinear PDEs. This encourages the PINNs model to learn the solution in accordance with how the information propagates in time, as the dynamics evolve throughout the spatio-temporal domain. We demonstrate that this simple modification alone is enough to introduce significant accuracy improvements, allowing us to tackle problems that have remained elusive to PINNs. Our analysis leads to the development of a *causal training* algorithm that is also accompanied by a practical quantitative criterion for assessing the training convergence of a PINNs model. Finally, we examine a collection of challenging benchmarks for which existing PINNs formulations fail, and demonstrate that the proposed *causal training* strategy leads to state-of-the-art results. Taken together, these contributions produce new understanding in how deep learning models can be used as a simulation tool for studying complex spatio-temporal systems, introducing new opportunities for their applicability to problems of industrial complexity.

The paper is structured as follows. In Section 2, we provide an overview of PINNs following the original formulation of Raissi et al. [11]. Using a simple case study, we reveal an implicit bias of continuous-time PINNs that makes them prone to violating physical causality, and thereby steering them towards erroneous solutions. To address this drawback, in Section 3 we put forth a simple re-formulation of the PINNs residual loss and propose a general *casual training* algorithm for explicitly respecting physical causality during model training. Section 4 discusses practical considerations specific to enhancing the accuracy and efficiency of PINNs. These developments are put to test in Section 5, where we demonstrate state-of-the-art results across a comprehensive collection of challenging benchmarks for which existing PINN formulations are known to fail. Finally, Section 6 provides a summary of our main findings, and touches upon remaining limitations and areas for future research.

2. Physics-informed neural networks (PINNs)

Problem setup: We begin with a brief overview of physics-informed neural networks (PINNs) [11] in the context of inferring the solutions of PDEs. Generally, we consider PDEs taking the form

$$\boldsymbol{u}_t + \mathcal{N}[\boldsymbol{u}] = 0, \quad t \in [0, T], \ \boldsymbol{x} \in \Omega, \tag{2.1}$$

subject to the initial and boundary conditions

$$\boldsymbol{u}(0, \boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega, \tag{2.2}$$

$$\mathcal{B}[\boldsymbol{u}] = 0, \quad t \in [0, T], \ \boldsymbol{x} \in \partial\Omega, \tag{2.3}$$

where $\mathcal{N}[\cdot]$ is a linear or nonlinear differential operator, and $\mathcal{B}[\cdot]$ is a boundary operator corresponding to Dirichlet, Neumann, Robin, or periodic boundary conditions. In addition, $\boldsymbol{u}$ describes the unknown latent solution that is governed by the PDE system of Eq. (2.1).

Following the original work of Raissi et al. [11], we proceed by representing the unknown solution $\boldsymbol{u}(t, \boldsymbol{x})$ by a deep neural network $\boldsymbol{u}_{\boldsymbol{\theta}}(t, \boldsymbol{x})$, where $\boldsymbol{\theta}$ denotes all tunable parameters of the network (e.g., weights and biases). Then, a physics-informed model can be trained by minimizing the following composite loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_{ic}\mathcal{L}_{ic}(\boldsymbol{\theta}) + \lambda_{bc}\mathcal{L}_{bc}(\boldsymbol{\theta}) + \lambda_r\mathcal{L}_r(\boldsymbol{\theta}), \tag{2.4}$$

where

$$\mathcal{L}_{ic}(\boldsymbol{\theta}) = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} \left| \boldsymbol{u}_{\boldsymbol{\theta}}(0, \boldsymbol{x}_{ic}^i) - \boldsymbol{g}(\boldsymbol{x}_{ic}^i) \right|^2, \tag{2.5}$$

$$\mathcal{L}_{bc}(\boldsymbol{\theta}) = \frac{1}{N_{bc}} \sum_{i=1}^{N_{bc}} \left| \mathcal{B}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_{bc}^i, \boldsymbol{x}_{bc}^i) \right|^2, \tag{2.6}$$

$$\mathcal{L}_{r}(\boldsymbol{\theta}) = \frac{1}{N_{r}} \sum_{i=1}^{N_{r}} \left| \frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial t}(t_r^i, \boldsymbol{x}_r^i) + \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_r^i, \boldsymbol{x}_r^i) \right|^2. \tag{2.7}$$

Here $\{\boldsymbol{x}_{ic}^i\}_{i=1}^{N_{ic}}$, $\{t_{bc}^i, \boldsymbol{x}_{bc}^i\}_{i=1}^{N_{bc}}$ and $\{t_r^i, \boldsymbol{x}_r^i\}_{i=1}^{N_r}$ can be the vertices of a fixed mesh or points that are randomly sampled at each iteration of a gradient descent algorithm. Notice that all required gradients with respect to input variables or network parameters $\boldsymbol{\theta}$ can be efficiently computed via automatic differentiation [31]. Moreover, the hyper-parameters $\{\lambda_{ic}, \lambda_{bc}, \lambda_r\}$ allow the flexibility of assigning a different learning rate to each individual loss term in order to balance their interplay during model training. These weights may be user-specified or tuned automatically during training [12,13].

An illustrative example: To motivate the proposed methods described in Section 3, let us study a representative case with which conventional PINN models are known to struggle. To this end, consider the one-dimensional Allen–Cahn equation

$$u_t - 0.0001 u_{xx} + 5u^3 - 5u = 0, \quad t \in [0, 1], x \in [-1, 1], \tag{2.8}$$

$$u(x, 0) = x^2 \cos(\pi x), \tag{2.9}$$

$$u(t, -1) = u(t, 1), \tag{2.10}$$

$$u_x(t, -1) = u_x(t, 1). \tag{2.11}$$

This example is difficult to directly solve with the original continuous-time formulation of Raissi et al. [11], and has been recently studied by *Wight et al.* [16] and *McClenny et al.* [14] who developed adaptive re-sampling and weighting algorithms, respectively, to improve the PINNs prediction.

Following the setup discussed in these studies [14,16], we represent the latent variable u by a fully-connected neural network $u_{\boldsymbol{\theta}}$ with tanh activation function, 4 hidden layers and 128 neurons per hidden layer. To further simplify the training objective (2.4), we also strictly impose the periodic BCs by embedding the input coordinates into Fourier expansion using Eq. (4.8) with $m = 10$ (see Section 4 for further details). Then the loss function (2.4) can be reduced to

$$\mathcal{L}(\boldsymbol{\theta}) = \lambda_{ic} \mathcal{L}_{ic}(\boldsymbol{\theta}) + \lambda_r \mathcal{L}_r(\boldsymbol{\theta}), \tag{2.12}$$

where $\mathcal{L}_{ic}(\boldsymbol{\theta})$ and $\mathcal{L}_r(\boldsymbol{\theta})$ are defined exactly the same as in Eq. (2.5) and Eq. (2.7). For simplicity, we create a uniform mesh of size 100×256 in the computational domain $[0, 1] \times [-1, 1]$, yielding $N_{ic} = 256$ initial points and $N_r = 25600$ collocation points for enforcing the PDE residual. We also choose $\lambda_{ic} = 100, \lambda_r = 1$ for better enforcing the initial condition.

We proceed by training the resulting PINN model via full-batch gradient descent using the Adam optimizer [32] for 2×10^5 iterations. As shown in Fig. 1, even when the periodic boundary conditions are enforced exactly, our conventional PINN model is unable to learn the accurate solution for this example. One can also observe that the predicted solution seems to get stuck at some intermediate state and cannot be further refined to provide an accurate approximation to the ground truth. This is consistent with the left panel of Fig. 2 where the loss functions rapidly decrease in the first few thousand training iterations, and then barely change for the rest of training, implying that the neural network gets trapped in an erroneous local minimum. Unfortunately, such problematic behavior is not a rare event, but rather a common outcome for PINNs, especially when solving transient problems [13,24].

PINNs can violate physical causality: To explore the underlying reasons behind this failed case study, let us closely examine the definition of the residual loss $\mathcal{L}_r$. Before doing so, we will slightly change our notation for convenience. Suppose that $0 = t_1 < t_2 < \cdots < t_{N_t} = T$ discretizes the temporal domain, and $\{\boldsymbol{x}_j\}_{j=1}^{N_x}$ discretizes the spatial domain Ω. For this example, $\{t_i\}_{i=1}^{N_t}$ and $\{\boldsymbol{x}_j\}_{j=1}^{N_x}$ are uniformly spaced meshes in $[0, 1]$ and $[-1, 1]$, respectively. Now for a given spatial discretization $\{\boldsymbol{x}_j\}_{j=1}^{N_x}$, we define the temporal residual loss as

$$\mathcal{L}_r(t, \boldsymbol{\theta}) = \frac{1}{N_x} \sum_{j=1}^{N_x} \left| \frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial t}(t, \boldsymbol{x}_j) + \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t, \boldsymbol{x}_j) \right|^2. \tag{2.13}$$

Then, the residual loss (2.7) can be rewritten as

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_t} \sum_{i=1}^{N_t} \mathcal{L}_r(t_i, \boldsymbol{\theta}) \tag{2.14}$$

$$= \frac{1}{N_t N_x} \sum_{i=1}^{N_t} \sum_{j=1}^{N_x} \left| \frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial t}(t_i, \boldsymbol{x}_j) + \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_i, \boldsymbol{x}_j) \right|^2. \tag{2.15}$$

Next, we discretize $\frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial t}$ using the forward Euler scheme [33]. For any $1 \le i \le N_t - 1$, $\mathcal{L}(t_i, \boldsymbol{\theta})$ can be approximated by

$$\begin{aligned} \mathcal{L}_r(t_i, \boldsymbol{\theta}) &\approx \frac{1}{N_x} \sum_{j=1}^{N_x} \left| \frac{\boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}_j) - \boldsymbol{u}_{\boldsymbol{\theta}}(t_{i-1}, \boldsymbol{x}_j)}{\Delta t} + \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_i, \boldsymbol{x}_j) \right|^2 \\ &\approx \frac{1}{\Delta t^2 \cdot |\Omega|} \int_{\Omega} |\boldsymbol{u}_{\boldsymbol{\theta}}(t_i, \boldsymbol{x}) - \boldsymbol{u}_{\boldsymbol{\theta}}(t_{i-1}, \boldsymbol{x}) + \Delta t \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t_i, \boldsymbol{x})|^2 d\boldsymbol{x}. \end{aligned} \tag{2.16}$$

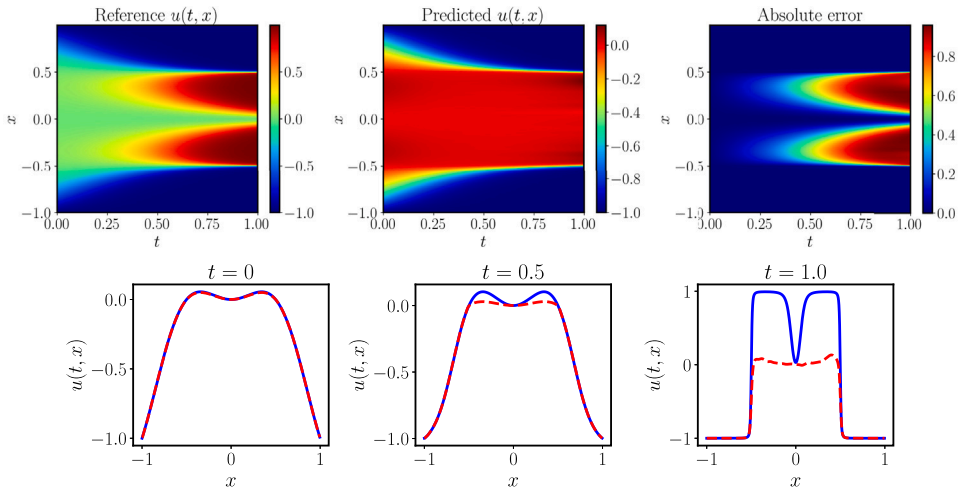

Fig. 1. *Allen–Cahn equation: Top:* Reference solution versus the prediction of a trained conventional physics-informed neural network. The resulting relative L^2 error is 49.87%. *Bottom:* Comparison of the predicted and reference solutions corresponding to the three temporal snapshots at $t = 0.0, 0.5, 1.0$.

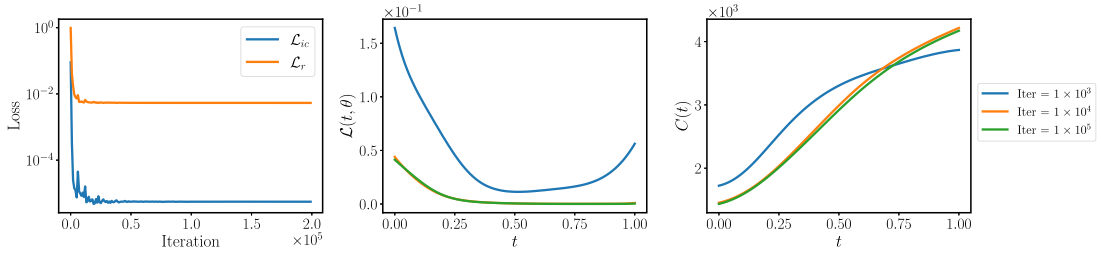

Fig. 2. *Allen–Cahn equation: Left:* Loss convergence of training a conventional physics-informed neural network for 2×10^5 iterations. *Middle:* Temporal residual loss $\mathcal{L}_r(t,\theta)$ at different iteration of the training. *Right:* Temporal convergent rate at different iteration of the training.

From the above expression, we immediately obtain that the minimization of $\mathcal{L}_r(t_i,\boldsymbol{\theta})$ should be based on the correct prediction of both $\boldsymbol{u}_{\boldsymbol{\theta}}(t_i,\boldsymbol{x})$ and $\boldsymbol{u}_{\boldsymbol{\theta}}(t_{i-1},\boldsymbol{x})$, while the original formulation of Eq. (2.7) tends to minimize all $\mathcal{L}_r(t_i,\boldsymbol{\theta})$ simultaneously. As a result, by using Eq. (2.7), the residual loss $\mathcal{L}_r(t_i,\boldsymbol{\theta})$ will be minimized even if the predictions at t_i and previous times are inaccurate. This behavior inevitably violates temporal causality, making the PINN model susceptible to learn erroneous solutions.

This conclusion is further confirmed by the middle panel of Fig. 2 where we plot the temporal residual loss of Allen–Cahn equation at different iterations of training. As expected, the residual is quite large near the initial state and rapidly decays to nearly zero after $t = 0.5$. We emphasize that the PDE temporal residual of small magnitude is meaningful only if the PINN model is well optimized and able to yield accurate predictions at the previous time steps.

An undesirable implicit bias: To provide a deeper understanding of the fact that PINNs may violate temporal causality, we analyze their training dynamics through the lens of their empirical Neural Tangent Kernel (NTK) [13,34]. Specifically, for every $\mathcal{L}_r(t,\boldsymbol{\theta})$ (Eq. (2.13)), we can define the empirical NTK $\boldsymbol{K}_{\boldsymbol{\theta}}(t) \in \mathbb{R}^{N_x \times N_x}$ whose ijth entry is given by [13]

$$\boldsymbol{K}_{\boldsymbol{\theta}}(t)_{ij} = \left\langle \frac{\partial \mathcal{R}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(t,\boldsymbol{x}_i), \frac{\partial \mathcal{R}_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}}(t,\boldsymbol{x}_j) \right\rangle, \quad i,j = 1,2,\ldots,N_x. \tag{2.17}$$

where $\mathcal{R}_{\boldsymbol{\theta}}$ is the corresponding PDE residual defined by

$$\mathcal{R}_{\boldsymbol{\theta}}(t,\boldsymbol{x}) = \frac{\partial \boldsymbol{u}_{\boldsymbol{\theta}}}{\partial t}(t,\boldsymbol{x}) + \mathcal{N}[\boldsymbol{u}_{\boldsymbol{\theta}}](t,\boldsymbol{x}). \tag{2.18}$$

As demonstrated by Wang *et al.* [13], the eigenvalues of $\boldsymbol{K}_{\boldsymbol{\theta}}(t)$ determine the convergence rate of each $\mathcal{L}_r(t,\boldsymbol{\theta})$ contributing to the total residual loss $\mathcal{L}_r(\boldsymbol{\theta})$. Specifically, larger eigenvalues implies faster convergence rate. Following [13], we introduce the definition

Definition 2.1. For any given $t \in [0,T]$, the temporal convergence rate $C(t)$ of $\mathcal{L}_r(t,\boldsymbol{\theta})$ is defined by

$$C(t) = \frac{\sum_{k=1}^{N_x} \lambda_k(t)}{N_x} = \frac{\mathrm{Trace}(\boldsymbol{K}_{\boldsymbol{\theta}}(t))}{N_x}, \tag{2.19}$$

where $\{\lambda_k(t)\}_{k=1}^{N_x}$ are the eigenvalues of $\boldsymbol{K}_{\boldsymbol{\theta}}(t)$.

Table 1
Allen–Cahn equation: Relative L^2 errors obtained by different approaches.

Method	Relative L^2 error
Original formulation of Raissi *et al.* [11]	4.98e−01
Adaptive time sampling [16]	2.33e−02
Self-attention [14]	2.10e−02
Time marching [25]	1.68e−02
Causal training (MLP)	**1.43e − 03**
Causal training (modified MLP)	**1.39e − 04**

Equipped with definition (2.19), we visualize $C(t)$ at different iterations during the training of our PINNs model for solving Allen–Cahn equation. In the right panel of Fig. 2, it can be seen that $C(t)$ is greater if t is greater, indicating that the network is biased towards minimizing the temporal residual $\mathcal{L}_r(t,\theta)$ for larger t. We emphasize that the minimization of the temporal residual loss $\mathcal{L}_r(t,\theta)$ is meaningful only if the PINN model is well optimized and able to yield accurate predictions at the previous time steps. Otherwise, even if $\mathcal{L}_r(t,\theta) = 0$, the errors can propagate through time and result in significant inaccuracies. The above analysis reveals an undesirable implicit bias of continuous-time PINN models trained via gradient descent, suggesting that such models can profoundly violate the temporal causal structure that is inherent to time-dependent PDE systems. We argue that this inherent pathology of PINNs is the key underlying reason behind their inability to simulate transient problems that exhibit strong temporal correlations and sensitivity to initial data. In the next section we put forth a remarkably simple and effective strategy for explicitly respecting physical causality during the training phase PINNs.

3 Causal training for physics-informed neural networks

A simple re-formulation: Based on our findings in the previous section, it is natural to ask how we can respect physical causality when solving PDEs with PINNs. We answer this question by introducing a simple re-formulation of the PINNs training objective that can explicitly account for the missing causal structure. To this end, we define a weighted residual loss as

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_t}\sum_{i=1}^{N_t} w_i \mathcal{L}_r(t_i,\boldsymbol{\theta}). \tag{3.1}$$

We recognize that the weights w_i should be large – and therefore allow the minimization of $\mathcal{L}_r(t_i,\boldsymbol{\theta})$ – only if all residuals $\{\mathcal{L}_r(t_k,\boldsymbol{\theta})\}_{k=1}^{i}$ before t_i are minimized properly, and vice versa. This can be achieved by expressing the weights w_i as

$$w_i = \exp\left(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}_r(t_k,\boldsymbol{\theta})\right), \text{ for } i = 2,3,\dots,N_t, \tag{3.2}$$

where ϵ will be referred to as a *causality parameter* that controls the steepness of the weights w_i (see below for a more detailed discussion). As such, the weighted residual loss can be written as

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_t}\sum_{i=1}^{N_t} \exp\left(-\epsilon \sum_{k=1}^{i-1} \mathcal{L}_r(t_k,\boldsymbol{\theta})\right) \mathcal{L}_r(t_i,\boldsymbol{\theta}). \tag{3.3}$$

Notice that w_i is inversely exponentially proportional to the magnitude of the cumulative residual loss from the previous time steps. As a consequence, $\mathcal{L}_r(t_i,\boldsymbol{\theta})$ will not be minimized unless all previous residuals $\{\mathcal{L}_r(t_k,\boldsymbol{\theta})\}_{k=1}^{i-1}$ decrease to some small value such that w_i is large enough.

The use of exponential functions is based on the requirement for a smooth, monotonically decreasing function with a range from 0 to 1, which ensures our temporal weights decay appropriately within this range. To investigate the impact of different weighting functions on model performances, we evaluate two alternatives, $1-\tanh(\epsilon x)$ and $1/1+\epsilon x$, in addition to the exponential function. The results are summarized in Figure 17, which clearly shows that exponentials are one of the best-suited choices to meet our needs.

We now employ this simple modification and revisit the Allen–Cahn case study discussed before. We proceed by training the same network by minimizing the loss of Eq. (2.4) using the weighted residual loss of Eq. (3.3) with $\epsilon = 100$, for 3×10^5 iterations of gradient descent under exactly the same hyper-parameter settings. The results of this experiment are summarized in Fig. 3. One can see that the predicted solution achieves an excellent agreement with the ground truth, yielding an approximation error of $1.43e-03$ measured in the relative L^2 norm. The left panel of Fig. 4 presents the convergence of the different loss function components, which is evidently much better than the one presented in Fig. 2. Here we note that no other modifications between the two cases exist, besides the use of the proposed weighted residual loss of Eq. (3.3).

In fact, if in conjunction with the weighted residual loss we also employ a more powerful architecture for this example, such as the modified MLP [12] described in Section 4, then we can achieve an even more accurate result with a resulting relative L^2 error of $1.39e-04$. Additional detailed visual assessments for this example are provided in Appendix D.

Finally, in Table 1 we provide the accuracy reported for this problem by existing approaches in the literature [14,16,25]. It is evident that the proposed methodology outperforms the best reported result of competing approaches by a factor of ∼10–100x. This is a strong indication of the significance and necessity of respecting causality in training PINNs.

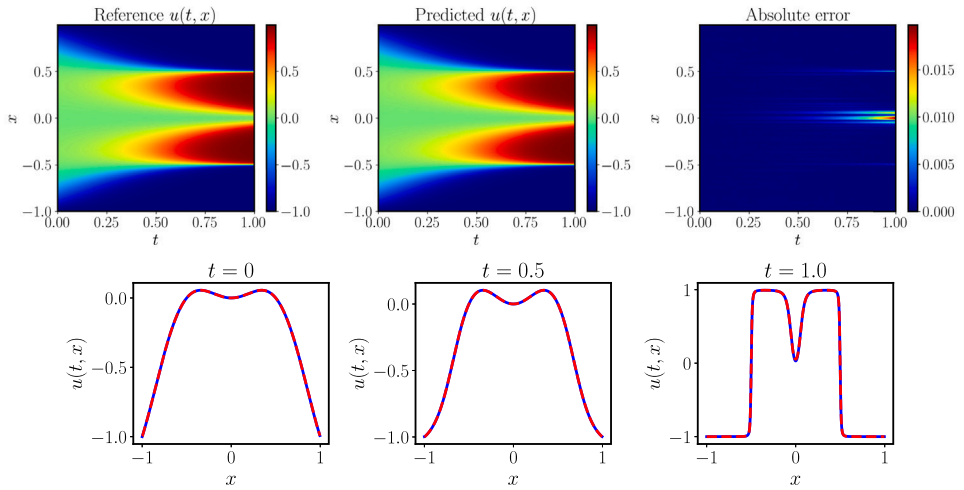

Fig. 3. *Allen–Cahn equation: Top:* Reference solution versus the prediction of a trained physics-informed neural network using Algorithm 1. The resulting relative L^2 error is $1.43e-03$. *Bottom:* Comparison of the predicted and reference solutions corresponding to the three temporal snapshots at $t = 0.0, 0.5, 1.0$.

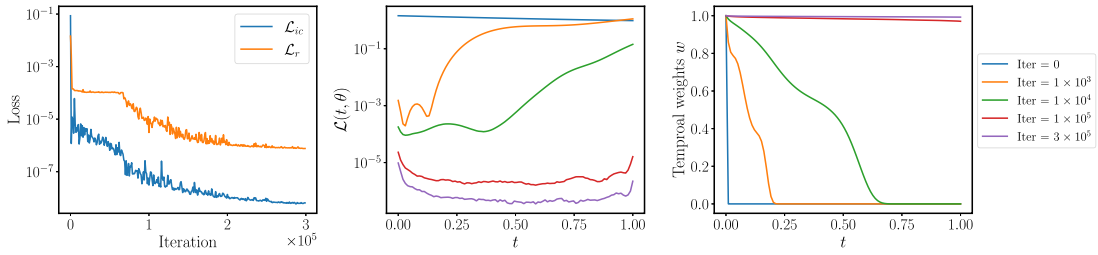

Fig. 4. *Allen–Cahn equation: Left:* Loss convergence of training a physics-informed neural network using Algorithm 1. *Middle:* Temporal residual loss $\mathcal{L}_r(t,\theta)$ at different iteration of the training. *Right:* Temporal weights at different iteration of the training.

A stopping criterion for assessing training convergence: To understand the effect of the residual weights $\{w_i\}$, we present the temporal residual loss and weights at different iterations of gradient descent in the middle and right panel of Fig. 4 and Figure 16. We observe that the initial temporal weights are all zero except for $t = 0$, implying that only $\mathcal{L}_r(t_0, \boldsymbol{\theta})$ will be minimized at the beginning of training. Throughout the rest of the training, more temporal weights are activated, and eventually, all of them converge to 1 as the PDE residual loss is properly minimized. This last observation suggests that monitoring the magnitude of the residual weights $\{w_i\}$ can provide an effective stopping criterion for assessing the convergence of a PINNs model during training. Specifically, one can choose to terminate training of $\min_i w_i > \delta$, for some chosen threshold parameter $\delta \in (0, 1)$. As we will see in Section 5, this stopping criterion not only helps to train a PINNs model faster, but it actually yields trained models with superior predictive accuracy.

Sensitivity on the causality parameter ϵ: Here we must note that the results obtained using the proposed weighted residual loss do exhibit some sensitivity to the causality parameter ϵ in Eq. (3.2). Figure 18 presents an ablation study examining the impact of varying fixed causal parameter ϵ during model training. It can be observed that choosing a very small ϵ can prevent the network from effectively minimizing the latter temporal residuals. Conversely, choosing a large ϵ value can result in a more difficult optimization problem, because the temporal residuals at earlier times have to decrease to a very small value in order to activate the latter temporal weights. This may be hard to achieve in some cases due to limited network capacity in minimizing the target residuals. In order to avoid tedious hyper-parameter tuning, we employ an annealing strategy for adjusting ϵ using an increasing sequence of values $\{\epsilon_i\}_{i=1}^k$, which gradually increases the strength with which the PDE residual constraint is enforced. As we will see in Section 5, we empirically observe that this choice yields the best results in practice.

Fitting the initial data: In the spirit of respecting causality, one may recognize that all temporal residuals should be minimized only if the network can first accurately fit the initial data. Therefore, we may treat the initial loss $\mathcal{L}_{ic}$ as a special temporal residual at $t = 0$ and incorporate it into the weighted residual loss of Eq. (3.1) in the same manner.

Causal training for PINNs: Based on the above remarks, Algorithm 1 presents a general *causal training* algorithm for PINNs. Specifically, it summarizes the proposed re-formulation of the residual and initial conditions loss, the annealing scheme for the ϵ parameter, and the stopping criterion for terminating the training upon the convergence of the temporal weights w_i. Accompanying Algorithm 1, here we present a few additional remarks worth discussing.

1. Although in this work we have limited our attention to PDEs with periodic boundary conditions that can be enforced in an exact manner (see Section 4 for more details), the proposed *causal training* algorithm can be adapted to also incorporate boundary constraints using a similar treatment to the initial conditions loss.
2. Note that the temporal weights $\{w_i\}_{i=0}^{N_t}$ are a function of the trainable parameters θ. We use `lax.stop_gradient` in our JAX [35] implementation to prevent gradient back-propagation through the computation of w_i.
3. The computational cost of the proposed algorithm is negligible compared to conventional PINNs formulations since the weights w_i are computed by directly evaluating the PINNs loss functions, whose values are already stored in the computational graph during training.
4. The proposed algorithm is not limited to fixed mesh points for evaluating the PINNs loss terms, and the collocation points can be randomly sampled at each iteration of gradient descent. The only requirement is that the sampled temporal points $\{t_i\}_{i=1}^{N_t}$ should form a non-decreasing sequence in temporal domain so that temporal causality can be respected.

Finally, the proposed algorithm is general and can be employed within any existing physics-informed machine learning pipeline, including physics-informed neural networks [11,19,21,30,36,37], physics-informed neural operators [38–41]. For a concrete example, we demonstrate how the proposed algorithm can be applied to physics-informed DeepONet to learn a solution operator for the Lorenz 96 system in Appendix H.

Algorithm 1: Causal training for physics-informed neural networks

Consider a physics-informed neural network $\boldsymbol{u}_{\theta}(t, \boldsymbol{x})$ imposed the exact boundary conditions, and the corresponding weighted loss function

$$\mathcal{L}(\theta) = \frac{1}{N_t} \sum_{i=0}^{N_t} w_i \mathcal{L}(t_i, \theta), \tag{3.4}$$

where $\mathcal{L}(t_0, \theta) = \lambda_{ic}\mathcal{L}_{ic}(\theta)$ and for $1 \leq i \leq N_t$, $\mathcal{L}(t_i, \theta)$ is defined in Equation 5. Initialize w_0 by 1 and select an increasing sequence of the causality parameter $\{\epsilon_i\}_{i=1}^{k}$. Then use S steps of a gradient descent algorithm to update the parameters θ as:

for $\epsilon = \epsilon_1, \dots, \epsilon_k$ **do**
  for $n = 1, \dots, S$ **do**
    (a) Compute and update the temporal weights by

$$w_i = \exp\left(-\epsilon \sum_{j=0}^{i-1} \mathcal{L}(t_j, \theta)\right), \text{ for } i = 1, 2, 3, \dots, N_t. \tag{3.5}$$

    Here $\epsilon > 0$ is a user-defined hyper-parameter that determines the "slope" of temporal weights. Note that we intentionally freeze the parameter w_i's during the gradient computation process such that they are not updated by gradient descent.
    (b) Update the parameters θ via gradient descent

$$\theta_{n+1} = \theta_n - \eta \nabla_{\theta} \mathcal{L}(\theta_n). \tag{3.6}$$

    if $\min_i w_i > \delta$ **then**
      break
    end
  end
end

The recommended hyper-parameters are $\lambda_{ic} = 10^3$, $\delta = 0.99$ and $\{\epsilon_i\}_{i=1}^{k} = [10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}]$.

Connection to existing approaches: The concept of causality employed in our work is distinct from and not directly related to the extensive literature on causal machine learning [42–44]. Causal machine learning primarily focuses on learning or identifying unknown causal effects from observational data. In contrast, our approach aims to incorporate the known causality structure of time-dependent PDEs into the training process of PINNs. Moreover, our proposed algorithm falls within the general framework of weighted empirical risk minimization (ERM) problems [45–47], which aim to minimize the risk or loss function by assigning weights to individual data points or model components, based on their importance or contribution to the overall model performance.

Furthermore, it is worth noting that the proposed residual weighting strategy bears some similarity to the adaptive time sampling of Wight et al. [16], since the effect of the weights w_i can be viewed as equivalent to changing the sampling density of collocation points. However, the method of Wight et al. has two main disadvantages in practice: (a) the sampling density has to be manually designed for different problems and training iterations, and (b) an accurate approximation of the designed sampling density requires a large volume of collocation points, leading to a large computational cost. Besides, we remark that our method shares the same motivation with "time-marching" or "curriculum training" strategies [16,24,48,49], in the sense of respecting temporal causality by learning the solution sequentially within separate time-windows. In fact, our *causal training* strategy should not be viewed as a replacement of time-marching approaches, but instead as a crucial enhancement to those, given the fact that violations of causality may still occur within each time window of a time-marching algorithm.

4 Practical considerations

As we will see in Section 5, high-order accuracy becomes a necessity for PINNs in order to tackle problems exhibiting sensitivity on initial data and strong spatio-temporal correlations (e.g. chaotic systems). Although PINNs are known for being incapable to achieve high-order accuracy in general, in this section we highlight a few extensions that can further enhance their performance in more challenging settings. Although these features are not deemed crucial for the successful application of Algorithm 1, we have empirically observed that, for the problems considered in this work, they can lead to further enhancements in terms of accuracy and computational efficiency.

Modified multi-layer perceptrons: In [12] Wang et al. put forth a novel architecture that was demonstrated to outperform conventional MLPs across a variety of PINNs benchmarks. Here, we will refer to this architecture as "modified MLP". The forward pass of a L-layer modified MLP is defined as follows

$$\boldsymbol{U} = \sigma(\boldsymbol{X}\boldsymbol{W}_1 + \boldsymbol{b}_1), \quad \boldsymbol{V} = \sigma(\boldsymbol{X}\boldsymbol{W}_2 + \boldsymbol{b}_2), \tag{4.1}$$

$$\boldsymbol{H}^{(1)} = \sigma(\boldsymbol{X}\boldsymbol{W}^{(1)} + \boldsymbol{b}^{(1)}), \tag{4.2}$$

$$\boldsymbol{Z}^{(l)} = \sigma(\boldsymbol{H}^{(k)}\boldsymbol{W}^{(l+1)} + \boldsymbol{b}^{(l+1)}), \quad l = 1, \dots, L-1, \tag{4.3}$$

$$\boldsymbol{H}^{(l+1)} = (1 - \boldsymbol{Z}^{(l)}) \odot \boldsymbol{U} + \boldsymbol{Z}^{(l)} \odot \boldsymbol{V}, \quad l = 1, \dots, L-1, \tag{4.4}$$

$$\boldsymbol{u}_{\boldsymbol{\theta}}(\boldsymbol{X}) = \boldsymbol{H}^{(L)}\boldsymbol{W}^{(L+1)} + \boldsymbol{b}^{(L+1)}, \tag{4.5}$$

where σ denotes a nonlinear activation function, $\odot$ denotes a point-wise multiplication, and $\boldsymbol{X}$ denotes an batch of input coordinates. All trainable parameters are given by

$$\boldsymbol{\theta} = \{\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, (\boldsymbol{W}^{(l)}, \boldsymbol{b}^{(l)})_{l=1}^{L+1}\}. \tag{4.6}$$

At first glance, this architecture seems to appear a bit complicated. However, notice that it is almost the same as a standard MLP network, with the addition of two encoders and a minor modification in the forward pass. Specifically, the inputs $\boldsymbol{X}$ are embedded into a feature space via two encoders $\boldsymbol{U}, \boldsymbol{V}$, respectively, and merged in each hidden layer of a standard MLP using a point-wise multiplication. Based on our prior experience, the modified MLP architecture is shown to be more powerful than standard MLPs in terms of minimizing the PDE residuals and capturing sharp gradients [9,12,38,39].

Exact periodic boundary conditions: Recent work by Dong et al. [50] showed how one can strictly impose periodic boundary conditions in PINNs as hard-constraints. We have empirically observed that this trick can simplify the training of PINNs and introduce some savings in terms of computational cost. To illustrate the main idea, let us consider enforcing periodic boundary conditions with period P in a one-dimensional setting. To this end, we would like to make sure that a neural network returns periodic predictions as

$$u^{(l)}(a) = u^{(l)}(a + P), \quad l = 0, 1, 2, \dots. \tag{4.7}$$

To enforce this constraint as part of the architecture itself, we construct a Fourier feature embedding of the form

$$\boldsymbol{v}(x) = (1, \cos(\omega x), \sin(\omega x), \cos(2\omega x), \sin(2\omega x), \dots, \cos(m\omega x), \sin(m\omega x)), \tag{4.8}$$

with $\omega = \frac{2\pi}{L}$, and some non-negative integer m. Then, for any network representation u_{θ}, it can be proved that any $u_{\theta}(v(x))$ exactly satisfies the periodic constraint of Eq. (4.7) (see [50] for a proof).

The same idea can be extended to higher-dimensional domains. For instance, let (x, y) denote the coordinates of a point in two dimensions, and suppose that $u(x, y)$ is a smooth periodic function to be approximated in a periodic cell $[a, a + P_x] \times [b, b + P_y]$, satisfying the following constraints

$$\frac{\partial^l}{\partial x^l} u(a, y) = \frac{\partial^l}{\partial x^l} u\left(a + P_x, y\right), \quad y \in \left[b, b + P_y\right], \tag{4.9}$$

$$\frac{\partial^l}{\partial y^l} u(x, a) = \frac{\partial^l}{\partial y^l} u\left(x, b + P_y\right), \quad x \in \left[a, a + P_x\right], \tag{4.10}$$

for $l = 0, 1, 2, \dots$, where P_x and P_y are the periods in the x and y directions, respectively. Similar to the one-dimensional setting, these constraints can be implicitly encoded in a neural network by constructing a two-dimensional Fourier features embedding as

$$\boldsymbol{v}(x, y) = \begin{bmatrix} \cos\left(\omega_x x\right)\cos\left(\omega_y y\right), \dots, \cos\left(n\omega_x x\right)\cos\left(m\omega_y y\right) \\ \cos\left(\omega_x x\right)\sin\left(\omega_y y\right), \dots, \cos\left(n\omega_x x\right)\sin\left(m\omega_y y\right) \\ \sin\left(\omega_x x\right)\cos\left(\omega_y y\right), \dots, \sin\left(n\omega_x x\right)\cos\left(m\omega_y y\right) \\ \sin\left(\omega_x x\right)\sin\left(\omega_y y\right), \dots, \sin\left(n\omega_x x\right)\sin\left(m\omega_y y\right) \end{bmatrix} \tag{4.11}$$

with $w_x = \frac{2\pi}{P_x}, w_y = \frac{2\pi}{P_y}$ and m, n being some non-negative integers. Following [50], any network representation $u_{\boldsymbol{\theta}}(\boldsymbol{v}(x, y))$ is guaranteed to be periodic in the x, y directions.

For time-dependent problems, we simply concatenate the time coordinates t with the constructed Fourier features embedding, i.e., $u_{\boldsymbol{\theta}}([t, \boldsymbol{v}(x)])$, or $u_{\boldsymbol{\theta}}([t, \boldsymbol{v}(x, y)])$. Although in this work we will only consider periodic problems, other types of boundary conditions, including Dirichlet, Neumann, Robin, etc., can also be enforced in a "hard" manner, see [51,52] for more details.

Taylor-mode automatic differentiation for high-order derivatives: Conventional forward- or reverse-mode automatic differentiation is known to incur a cost that scales exponentially – both in terms of memory and computation – with the order of differentiation. This can quickly introduce a bottleneck in cases where derivatives of order higher than two are required (see e.g. the Kuramoto–Sivashinsky benchmark considered in Section 5). To address this drawback, here we employ Taylor-mode automatic differentiation [31] in order to accelerate the computation of high-order derivatives. This is accomplished by leveraging a truncated Taylor polynomial approximation that allows for efficient computation of high-order derivatives of function compositions via the Faà di Bruno formula [31]

$$\frac{\partial^n}{\partial x_1 \cdots \partial x_n} f(g(x)) = \sum_{\sigma \in \pi\{1,\dots,n\}} f^{(|\sigma|)}(g(x)) \prod_{b \in \sigma} \frac{\partial^{|b|}}{\prod_{j \in b} \partial x_j} g(x), \tag{4.12}$$

where $\pi\{1,\dots,n\}$ is the set of all partitions of the set $\{1,\dots,n\}$. It has been shown that Taylor-mode automatic differentiation enjoys much better scaling than conventional forward-mode or reverse-mode automatic differentiation, with its benefits becoming increasingly more dramatic as the order of differentiation is increased [53]. In terms of implementation, we leverage the `jax.jet` primitive accompanying the work of Bettencourt et al. [35,53].

Parallel training: Graphics processing units (GPUs) are the prevailing hardware choice for training PINNs, however these devices are often bound by their memory capacity. For more complex simulation scenarios (e.g. the Navier–Stokes benchmark in Section 5) we have empirically observed that using larger batch sizes during training leads to enhanced convergence and predictive accuracy. However, a desirable batch size might exceed the available memory that a single GPU can offer, therefore motivating the use of data-parallelism across multiple GPU devices. In order to facilitate this, we utilize synchronous data-parallelism across multiple GPUs, with each GPU storing an identical copy of all trainable parameters. In this paradigm, a batch of training data is split into sub-batches, one for each device. Specifically, batches of spatial and temporal points used to evaluate the training loss are generated randomly and independently on each available GPU, and gradients of the training loss are then aggregated across all devices with a collective reduce-mean operation. As such, each device can then update its own local copy of all trainable model parameters at each gradient descent iteration the using global gradient signal that is broadcasted across all devices. In our implementation, this is efficiently performed leveraging the `jax.pmap` primitive in JAX [35], allowing us to seamlessly scale our code to an arbitrary number of GPUs. The parallel performance of our implementation will be assessed via strong and weak scaling studies, as discussed in Section 5.3.

5 Results

Our goal in this section is to demonstrate the effectiveness of the proposed *causal training* algorithm by providing state-of-the-art numerical results for various types of differential equations exhibiting chaotic behavior, where existing PINNs formulations are destined for failure. Specifically, we will consider the forward simulation of the chaotic Lorenz system, the Kuramoto–Sivashinsky equation, and a two-dimensional simulation of decaying turbulence governed by the incompressible Navier–Stokes equations. Although these benchmarks can all be easily tackled using conventional numerical methods, they have remained elusive to PINNs since their initial conception [28,54], and all the variants that followed the reincarnation of this framework by Raissi et al. [29].

Throughout all benchmarks, we will employ the modified MLP architecture discussed in Section 4 equipped with hyperbolic tangent activation functions (Tanh) and initialized using the Glorot normal scheme [55], unless otherwise stated. We will enforce periodic boundary conditions as hard constraints by constructing appropriate Fourier features embedding of the input, as discussed in Section 4. All networks are trained via stochastic gradient descent using the Adam optimizer with default settings [32] and an exponential learning rate decay with a decay-rate of 0.9 every 5,000 training iterations. As suggested by [16,24,25], we will also employ time-marching to reduce optimization difficulties. Specifically, we will split up the temporal domain of interest $[0, T]$ into sub-domains $[0, \Delta t], [\Delta t, 2\Delta t], \dots [T - \Delta t, T]$, and train networks to learn the solution in each sub-domain, where the initial condition is obtained from the prediction of the previously trained network. At the end of training, the resulting PINN model can produce predictions for the target solution at any continuous query location in the global spatio-temporal domain.

All hyper-parameter settings, computational costs, implementation details and validation metrics are all discussed in Appendix. The code and data accompanying this manuscript will be made publicly available at https://github.com/PredictiveIntelligenceLab/CausalPINNs.

5.1 Lorentz system

As our first example, we consider the chaotic Lorenz system. It is well known that this system exhibits strong sensitivity to its initial conditions, which can trigger divergent trajectories in finite time if the numerical predictions sought are not sufficiently accurate. The system is described by the following ordinary differential equations

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \sigma(y - x), \tag{5.1}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} = x(\rho - z) - y, \tag{5.2}$$

$$\frac{\mathrm{d}z}{\mathrm{d}t} = xy - \beta z. \tag{5.3}$$

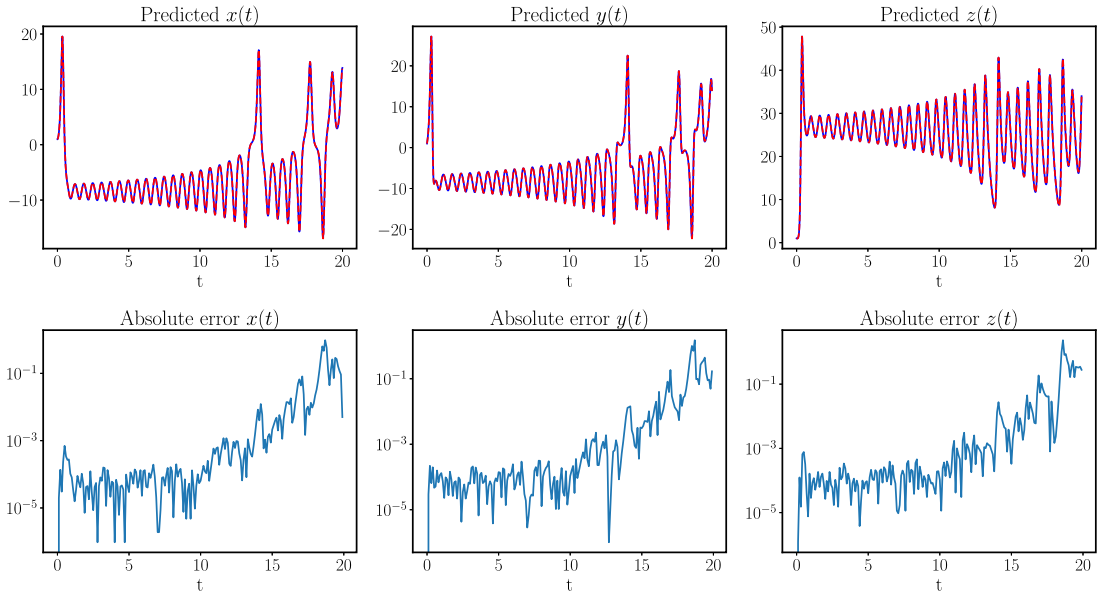

Fig. 5. *Lorentz system:* Comparison between the predicted and reference solutions.

These equations arise in studies of convection and instability in planetary atmospheric convection, where x, y, and z denote variables proportional to convective intensity, horizontal, and vertical temperature differences [56]. Parameters ρ, σ and β denote the Prandtl number, Rayleigh number, and a geometric factor, respectively. The Lorenz system is well-known to be chaotic for certain parameter values and initial conditions. Here, we consider a classical setting with $\sigma = 3, \rho = 28$, and $\beta = 8/3$. Our goal is to construct a PINNs model for learning the ODE solution up to time $T = 20$, starting from an initial condition $[x(0), y(0), z(0)] = [1, 1, 1]$ that does not lie on the system's attractor. The employed PINN model architecture and training hyper-parameters are discussed in Appendix B.

Fig. 5 shows the predicted trajectory against the reference trajectory obtained via a classical numerical solver (see Appendix B for more details), where an excellent agreement can be observed with a relative L^2 error $1.139e-02, 1.656e-02, 7.038e-03$ for the x, y, z components, respectively. Moreover, all training losses are plotted in Appendix Figure 19. We can see that the stopping criterion $\min_i w_i > \delta$ discussed in Section 3 is satisfied for the training of each time window. It is worth pointing out that the proposed stopping criterion will not only benefit the predictive accuracy, but also save lots of computational costs. To verify this, we train the network by removing the stopping criterion and training for a fixed number of iterations for each time window under exactly the same hyper-parameter setting. Interestingly, as shown in Appendix 21, the training losses can achieve slightly lower values than the ones using the stopping criterion. However, the model predictions are less accurate, as some discrepancies can be clearly observed in Appendix Figure 20. Although the reason behind this behavior still remains unclear, it appears that training the model for more iterations after the proposed stopping criterion has been met seems to give rise to over-fitting.

5.2 Kuramoto–Sivashinsky equation

The next example aims to illustrate the effectiveness of our method in tackling spatio-temporal chaotic systems. To this end, we consider one-dimensional Kuramoto–Sivashinsky equation, which has been independently derived in the context of reaction–diffusion systems [57] and flame front propagation [58]. The Kuramoto–Sivashinsky equation exhibits a wealth of spatially and temporally nontrivial dynamical behavior including chaos, and has served as a model example in efforts to understand and predict the complex dynamical behavior associated with a variety of physical systems. The equation takes the form

$$u_t + \alpha u u_x + \beta u_{xx} + \gamma u_{xxxx} = 0, \tag{5.4}$$

subject to periodic boundary conditions and an initial condition

$$u(0, x) = u_0(x). \tag{5.5}$$

Case I (regular): We start with a relatively simple scenario by setting $\alpha = 5, \beta = 0.5, \gamma = 0.005$, and a spatial domain $[-1, 1]$. The initial condition is given by $u_0(x) = -\sin(\pi x)$. Our goal is to lean the associated solution up to time $T = 1$. A detailed visual assessment of the predicted solution is presented in Fig. 6. In particular, we present a comparison between the reference and the predicted solutions at different time instants $t = 0, 0.5, 1.0$. It can be observed that the PINNs prediction achieves an excellent agreement with the reference solutions, yielding an error of $3.49e-04$ measured in the relative L^2 norm. This is further illustrated by the temporal relative L^2 error shown in the left panel of Fig. 8. Particularly, one may note that the error increases drastically by one order of

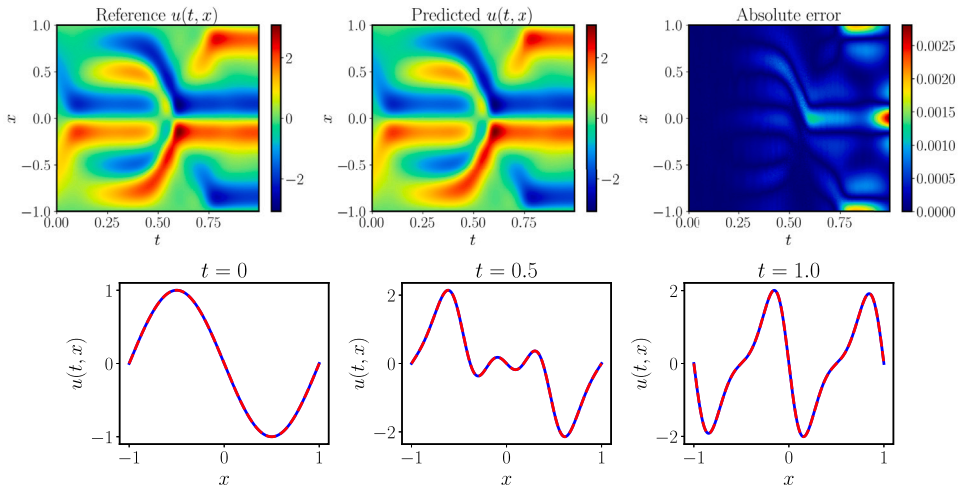

Fig. 6. *Kuramoto–Sivashinsky equation (regular): Top:* Reference solution versus the prediction of a trained physics-informed neural network using Algorithm 1. The resulting relative L^2 error is $3.49e-04$. *Bottom:* Comparison of the predicted and reference solutions corresponding to the three temporal snapshots at $t = 0, 0.5, 1.0$.

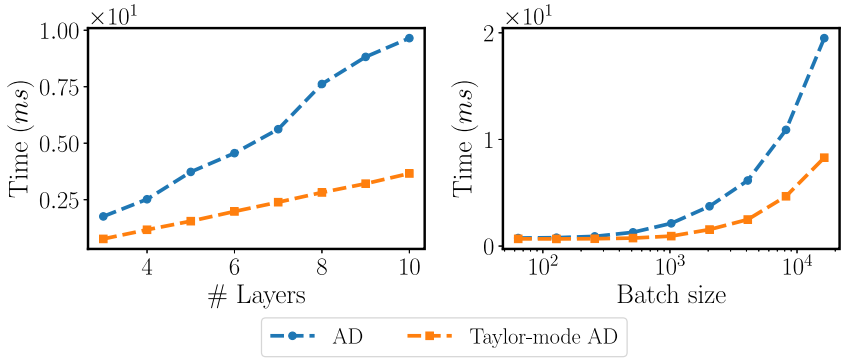

Fig. 7. *Kuramoto–Sivashinsky equation (regular): Left:* Timing of evaluating the loss function of a PINN model with different number of layers. The rest hyper-parameters are the same as in Table 3. *Right:* Timing of evaluating the forward pass of a PINN model with different batch sizes. The rest hyper-parameters are the same as in Table 3.

magnitude for $t \in [0.4, 0.6]$ where the solution happens to experience a fast transition. This behavior is consistent with the larger loss values and the larger number of training iterations required before the stopping criterion is met, as observed in Appendix Figure 22.

To highlight the computational efficiency of Taylor-mode automatic differentiation (Taylor-mode AD) discussed in , here we provide a comparison in terms of computational cost against conventional reverse-mode automatic differentiation (AD) [31]. Specifically, we consider PINN models with a different number of layers and batch sizes. As shown in Fig. 7, Taylor-mode AD provides a significant advantage in terms of computational efficiency, allowing us to accommodate larger architectures and batch sizes. As a consequence, for the same architecture and batch size, we have consistently observed a speed-up of 3–5x in the total training time required for Taylor-mode AD versus conventional AD.

Case II (chaotic): We proceed by solving a more challenging case exhibiting chaotic behavior, which remains stubbornly unsolved using existing PINNs formulations [59]. Specifically, we set $\alpha = 100/16, \beta = 100/16^2, \gamma = 100/16^4$, for a fixed spatial domain in $[0, 2\pi]$. Starting from an initial condition in the chaotic regime, we use PINNs to solve Kuramoto–Sivashinsky equation up to time $T = 0.5$. The results are summarized in Fig. 9, from which one can see that the predicted solution is in good agreement with the reference solution obtained via classical spectral methods (see Appendix F for more details). The resulting relative L^2 error over the entire spatio-temporal domain is $2.46e-02$, which is visualized in the right panel of Fig. 8. These results highly suggest that the proposed *causal training* algorithm enables PINN models to capture the intricate chaotic behavior of this system.

From a critical standpoint, it is important to acknowledge the inherent difficulties in simulating the long-time behavior of chaotic systems due to their sensitivity to initial conditions and the presence of a Lyapunov constant that sets a limit on predictability.

Fig. 10 summarizes our results starting with a simple initial state $u_0(x) = \cos(x)(1 + \sin(x))$, and simulating the dynamics up to time $T = 0.9$. One can observe that the predicted solution accurately captures the transition to chaos at around $t = 0.4$, while

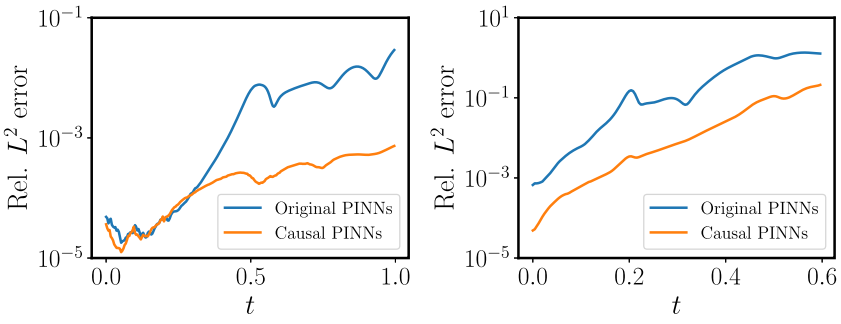

Fig. 8. *Kuramoto–Sivashinsky equation:* Relative L^2 errors of training a physics-informed neural network with and without *causal training* for regular case (left) and chaotic case (right). Both of PINNs are trained with the time-marching strategy.

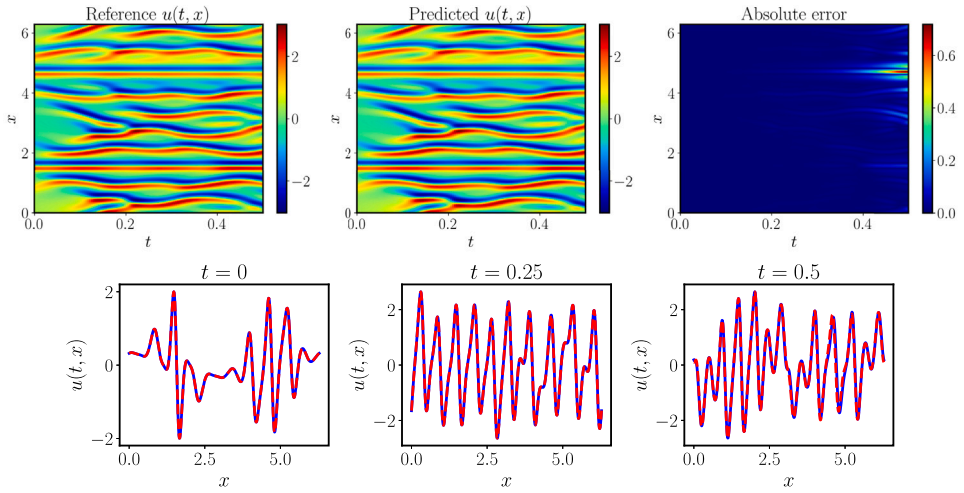

Fig. 9. *Kuramoto–Sivashinsky equation (chaotic): Top:* Reference solution versus the prediction of a trained physics-informed neural network using Algorithm 1. The resulting relative L^2 error is $2.26e-02$. *Bottom:* Comparison of the predicted and reference solutions corresponding to the three temporal snapshots at $t = 0, 0.25, 0.5$.

eventually loses accuracy after $t = 0.8$ due to the chaotic nature of the problem and the inevitable numerical error accumulation of PINNs, leading to a relative L^2 error above 10% for the final state. While it is infeasible to predict the long-time behavior of chaotic systems with absolute accuracy, our objective is to further enhance the accuracy of PINN approximations in such complex regimes over a certain time horizon. To achieve this, we plan to construct auto-regressive models via operator learning techniques as described in [39], thereby addressing the challenges associated with long-time prediction more effectively.

5.3. Navier–Stokes equation

To further emphasize the effectiveness of the proposed *causal training* algorithm for solving chaotic dynamical systems, in the last example, we consider a classical two-dimensional decaying Navier–Stokes example in a square domain with periodic boundary conditions. This problem can be modeled via the incompressible Navier–Stokes equations expressed in the velocity–vorticity formulation

$$w_t + \boldsymbol{u} \cdot \nabla w = \frac{1}{\mathrm{Re}} \Delta w, \quad \text{in } [0,T] \times \Omega, \tag{5.6}$$

$$\nabla \cdot \boldsymbol{u} = 0, \quad \text{in } [0,T] \times \Omega, \tag{5.7}$$

$$w(0,x,y) = w_0(x,y), \quad \text{in } \Omega, \tag{5.8}$$

where $\boldsymbol{u} = (u,v)$ denotes the flow velocity field, $w = \nabla \times \boldsymbol{u}$ denotes the vorticity, and Re is the Reynolds number. In addition, we set $\Omega = [0, 2\pi]^2$ and $Re = 100$. Our goal is to use PINNs to simulate the flow up to $T = 1$.

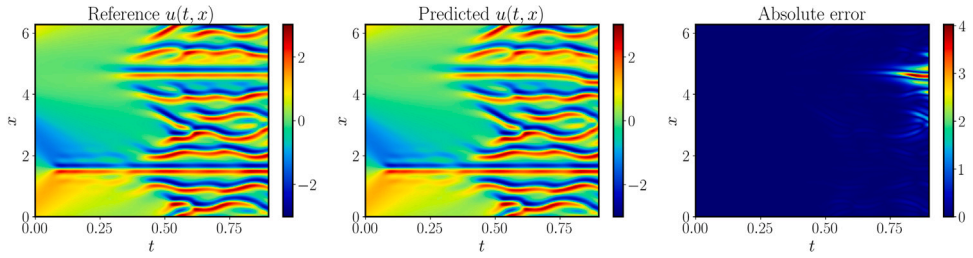

Fig. 10. *Kuramoto–Sivashinsky equation (chaotic):* Reference solution versus the prediction of a trained physics-informed neural network using Algorithm 1. The initial condition is $u_0(x) = \cos(x)(1 + \sin(x))$ An animation of the solution evolution is provided at https://github.com/PredictiveIntelligenceLab/CausalPINNs#kuramotosivashinsky-equation.

Fig. 11 presents the predicted velocity and vorticity field at $T = 1$. More detailed visual assessments are provided in Appendix G. We can see that all latent variables of interest are in good agreement with their corresponding reference solutions, yielding an error of $3.90e{-}02, 2.61e{-}02, 3.53e{-}02$ for u, v, w, respectively, over the entire spatio-temporal domain. This observation is further illustrated by the resulting errors reported in Fig. 12 and the computed energy spectrum in Fig. 13. These results highlight the remarkable effectiveness of the proposed *causal training* algorithm, successfully enabling the PINNs model to capture such complicated turbulent flow without any training data.

For this benchmark, we also report the performance of our parallel JAX implementation on a compute node equipped with 8 NVIDIA Ampere A6000 GPUs. We use an effective batch-size of 42,000 spatio-temporal points sampled in each training iteration on each GPU with a network consisting of 6 layers with 300 neurons per layer. Fig. 14 presents the scaling results obtained. To conduct a strong scaling study, we keep the problem size fixed and split the batch across several GPUs. As expected, we notice a speed-up, but the benefits deteriorate as the number of GPUs is increased beyond 4. We attribute this behavior to the fact that, for a fixed problem size, the compute load assigned to each GPU decreases as the number of devices is increased, leading to an under-utilization of each device. We have also performed a weak scaling study in which the number of points sampled per GPU is fixed. Under this setting, we report excellent parallel efficiency that remains above 99% as the number of GPUs is increased. While we have only considered data-parallelism in this study, we may be able to obtain further speed-ups by considering a combination of data- and function-parallelism techniques [60] in future studies. Fig. 14 also reports the effect of batch-size of training on the resulting L^2 accuracy for the first time-window ($t \in [0, 0.1]$). In general, we notice that an increase in batch-size results in higher accuracy of the network. This motivates the use of larger batch sizes through data-parallelism as a mechanism for enhancing the accuracy of PINNs in more challenging problems.

6. Discussion

Physical systems possess an inherent causal structure that explains the fundamental relationship between causes and effects governing their dynamic evolution. In this work, we show that physics-informed neural networks are prone to violating that structure when trained to infer the solution of time-dependent PDEs. Specifically, by studying the limiting neural tangent kernel of PINNs we reveal an implicit bias indicating a preference of PINNs to first minimize PDE residuals at later times, before even fitting the initial data. We argue that this fundamental drawback is one of the key reasons why PINNs can fail in practice. To resolve this shortcoming, we propose a novel *causal training* algorithm that can restore physical causality during the training of a PINN model by appropriately re-weighting the PDE residual loss at each iteration of gradient descent. Interestingly, this also leads to a simple stopping criterion for effectively assessing the convergence of the total training loss. We demonstrate that this simple modification alone is sufficient to achieve 10–100x improvements in accuracy compared to competing approaches, opening the path to tackling challenging problems that were not accessible to PINNs before, such as the chaotic Lorenz and Kuramoto–Sivashinsky equations, and the incompressible Navier–Stokes equations in the turbulent regime.

In this work we have solely focused on forward simulation problems, as we believe that these are the cases that most strongly expose the challenges and limitations in building PINNs models. While it is true that PINNs are currently better suited and have enjoyed far more success in tackling hybrid/inverse problems in which observational data is available, we believe that respecting causality is a crucial factor to consider when training a PINNs model, regardless of the forward/inverse nature of a given problem. To this end, in the inverse problem setting one should consider observational data as point sources of information, and ensure that PDE residuals are first adequately minimized at those locations before propagating information outwards. A more detailed exploration of this direction will be sought in future work.

We must also note that different problems are likely to pose a different causal structure. For example, in optimal control one needs to predict the state of a system by evolving its dynamics forward in time from a given initial condition, but also compute sensitivities with respect to a control input by evolving the adjoint system backwards in time from a given terminal condition that depends on the final system state. In this case, what we here refer to as "temporal causality" takes a different form for the state (forward) and the co-state (adjoint) simulations. However, our main message remains the same: respecting causality matters, and training algorithms for PINNs should be designed to respect how information propagates according to the underlying principles that govern the evolution of a given system.

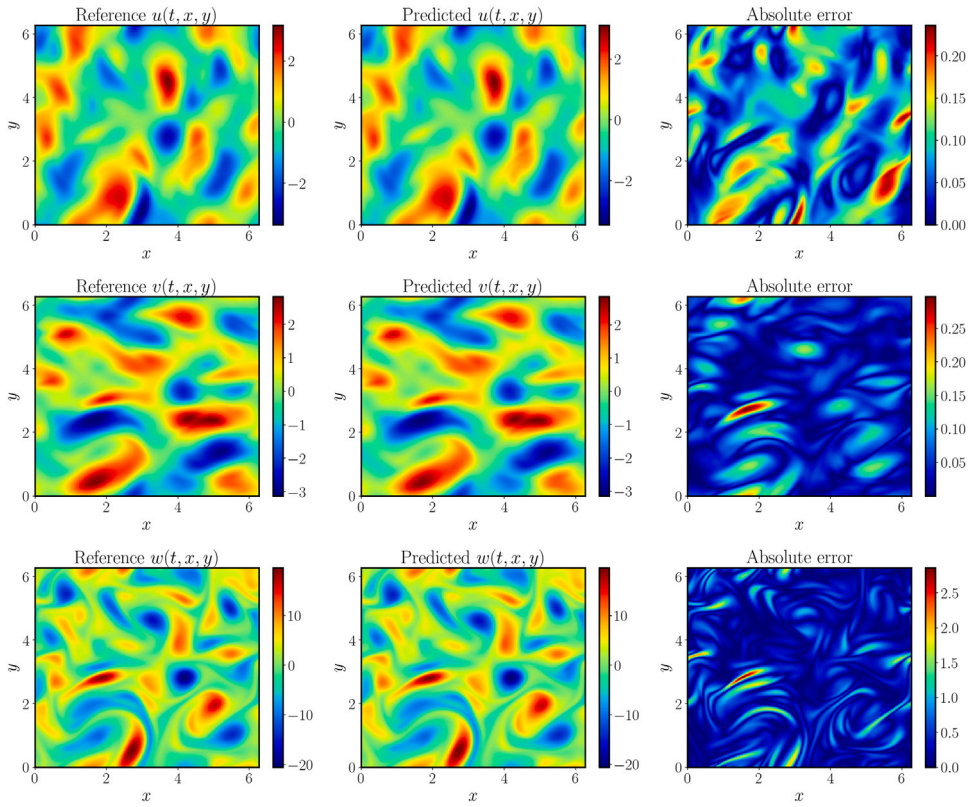

Fig. 11. *Navier–Stokes equation:* Representative snapshot of the predicted velocity and vorticity versus the corresponding reference solution at $t = 1$. An animation of the solution evolution is provided at https://github.com/PredictiveIntelligenceLab/CausalPINNs#navier-stokes-equation.

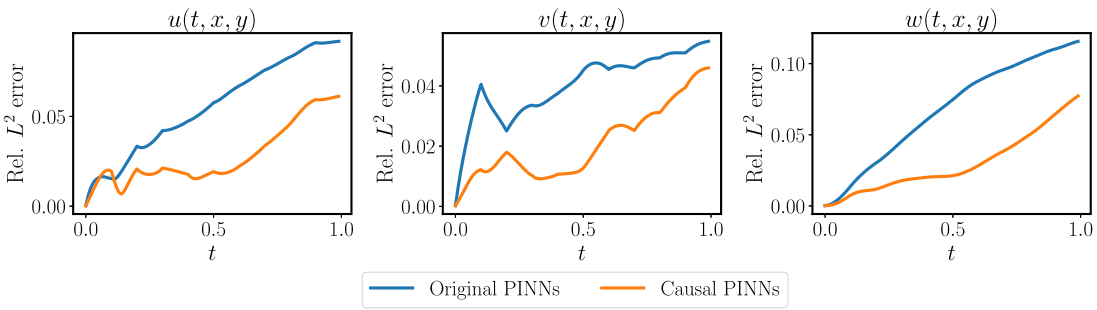

Fig. 12. Navier–Stokes: Relative L^2 errors of the u, v, w obtained by a trained physics-informed network with and without *causal training*, respectively. Both of PINNs are trained with the time-marching strategy.

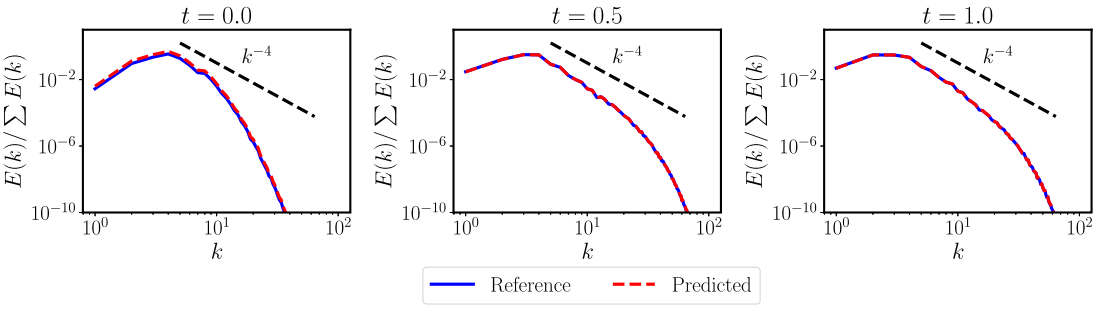

Fig. 13. *Navier–Stokes equation:* Reference versus predicted normalized kinetic energy spectra at different time snapshots $t = 0.0, 0.5, 1.0$.

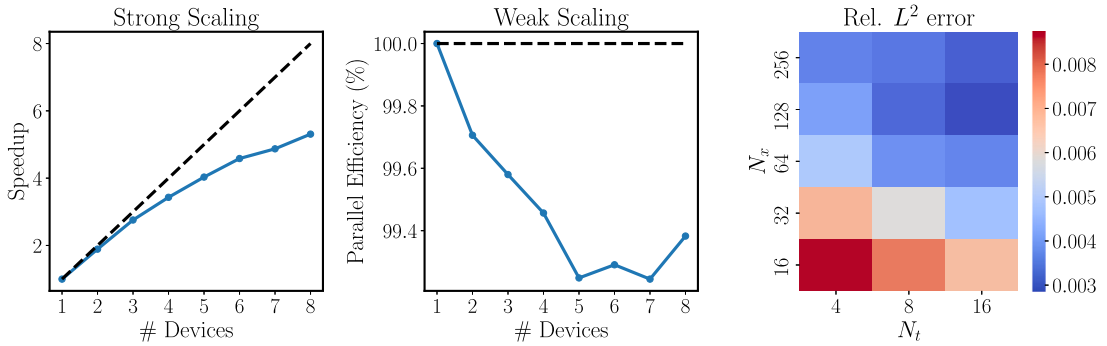

Fig. 14. *Parallel Performance: Left: Strong Scaling:* Keeping the total-batch size for the problem fixed, we evaluate the speedup obtained when the batch is split across multiple devices. *Center: Weak Scaling:* Keeping the batch-size on each GPU fixed, we report the efficiency of scaling by dividing the time taken on a single device over the time taken on n-devices. *Right: Effect of batch-size:* L^2 error for models trained till $t = 0.1$ using N_t and N_x points per iteration in the temporal and spatial domain respectively.

Given the rising prominence of PINNs across academic and industrial use cases, we consider this as a hallmark contribution that sets a new standard for what such models are capable of. We anticipate that the findings of this work will create new opportunities for the application of PINNs to more complex scenarios across diverse domains, including fluid mechanics, electromagnetics, quantum mechanics, and elasticity. However, despite the encouraging results reported here, there is a still gap between the current progress in PINNs research and real-world applications. We have to admit that viewing PINNs as a forward PDE solver is significantly more time-consuming than the traditional numerical solvers. Therefore, future research should focus on accelerating the training of PINNs. Distributed and parallel implementations can be of great help [21,61] in this direction. Another aspect with great room for improvement is related to architecture design. Even though effective modifications such as the modified MLP discussed in Section 4 and in [12] can introduce noticeable gains in accuracy, a niche architecture similar to what convolutional networks have been for vision or Transformers for language processing, is yet to be discovered for solving PDEs. To this end, we must recognize that training a PINN model is fundamentally different from solving conventional supervised learning tasks, requiring us to design more effective architectures for minimizing PDE residuals in a self-supervised manner. We believe that addressing these open questions will become an important piece of the puzzle in advancing the use of physics-informed machine learning as a reliable analysis tool in computational science and engineering.

CRediT authorship contribution statement

Sifan Wang: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Shyam Sankaran:** Writing – original draft, Visualization, Software, Investigation. **Paris Perdikaris:** Writing – review & editing, Writing – original draft, Resources, Project administration, Methodology, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The manuscript contains a link to all code and data.

Acknowledgments

We would like to acknowledge support from the US Department of Energy, United States of America under the Advanced Scientific Computing Research program (grant DE-SC0019116), the US Air Force, United States of America (grant AFOSR FA9550-20-1-0060), and US Department of Energy/Advanced Research Projects Agency, United States of America (grant DE-AR0001201). We also thank the developers of the software that enabled our research, including JAX [35], JAX-CFD [62], Matplotlib [63], and NumPy [64].

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cma.2024.116813.

References

[1] Maziar Raissi, Alireza Yazdani, George Em Karniadakis, Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations, Science 367 (6481) (2020) 1026–1030.
[2] Abhilash Mathews, Manaure Francisquez, Jerry W. Hughes, David R. Hatch, Ben Zhu, Barrett N. Rogers, Uncovering turbulent plasma dynamics via deep learning from partial observations, Phys. Rev. E 104 (2) (2021) 025205.
[3] Georgios Kissas, Yibo Yang, Eileen Hwuang, Walter R. Witschey, John A. Detre, Paris Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 358 (2020) 112623.
[4] Alireza Yazdani, Lu Lu, Maziar Raissi, George Em Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput. Biol. 16 (11) (2020) e1007575.
[5] Sifan Wang, Paris Perdikaris, Deep learning of free boundary and Stefan problems, J. Comput. Phys. 428 (2021) 109914.
[6] Khemraj Shukla, Patricio Clark Di Leoni, James Blackshire, Daniel Sparkman, George Em Karniadakis, Physics-informed neural network for ultrasound nondestructive quantification of surface breaking cracks, J. Nondestruct. Eval. 39 (3) (2020) 1–20.
[7] Yuyao Chen, Lu Lu, George Em Karniadakis, Luca Dal Negro, Physics-informed neural networks for inverse problems in nano-optics and metamaterials, Opt. Express 28 (8) (2020) 11618–11633.
[8] Francisco Sahli Costabal, Yibo Yang, Paris Perdikaris, Daniel E. Hurtado, Ellen Kuhl, Physics-informed neural networks for cardiac activation mapping, Front. Phys. 8 (2020) 42.
[9] Sifan Wang, Hanwen Wang, Paris Perdikaris, On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks, Comput. Methods Appl. Mech. Engrg. 384 (2021) 113938.
[10] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, Liu Yang, Physics-informed machine learning, Nat. Rev. Phys. (2021) 1–19.
[11] Maziar Raissi, Paris Perdikaris, George E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707.
[12] Sifan Wang, Yujun Teng, Paris Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM J. Sci. Comput. 43 (5) (2021) A3055–A3081.
[13] Sifan Wang, Xinling Yu, Paris Perdikaris, When and why PINNs fail to train: A neural tangent kernel perspective, J. Comput. Phys. 449 (2022) 110768.
[14] Levi McClenny, Ulisses Braga-Neto, Self-adaptive physics-informed neural networks using a soft attention mechanism, 2020, arXiv preprint arXiv:2009.04544.
[15] Suryanarayana Maddu, Dominik Sturm, Christian L. Müller, Ivo F. Sbalzarini, Inverse Dirichlet weighting enables reliable training of physics informed neural networks, Mach. Learn.: Sci. Technol. (2021).
[16] Colby L. Wight, Jia Zhao, Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks, 2020, arXiv preprint arXiv:2007.04542.
[17] Mohammad Amin Nabian, Rini Jasmine Gladstone, Hadi Meidani, Efficient training of physics-informed neural networks via importance sampling, Comput.-Aided Civ. Infrastruct. Eng. (2021).
[18] Jie Bu, Anuj Karpatne, Quadratic residual networks: A new class of neural networks for solving forward and inverse problems in physics involving PDEs, in: Proceedings of the 2021 SIAM International Conference on Data Mining, SDM, SIAM, 2021, pp. 675–683.
[19] Ameya D. Jagtap, Yeonjong Shin, Kenji Kawaguchi, George Em Karniadakis, Deep Kronecker neural networks: A general framework for neural networks with adaptive activation functions, Neurocomputing 468 (2022) 165–180.
[20] Senwei Liang, Liyao Lyu, Chunmei Wang, Haizhao Yang, Reproducing activation function for deep learning, 2021, arXiv preprint arXiv:2101.04844.
[21] Ameya D. Jagtap, George Em Karniadakis, Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Commun. Comput. Phys. 28 (5) (2020) 2002–2041.
[22] Ben Moseley, Andrew Markham, Tarje Nissen-Meyer, Finite basis physics-informed neural networks (FBPINNs): a scalable domain decomposition approach for solving differential equations, 2021, arXiv preprint arXiv:2107.07871.
[23] Ameya D. Jagtap, Kenji Kawaguchi, George Em Karniadakis, Adaptive activation functions accelerate convergence in deep and physics-informed neural networks, J. Comput. Phys. 404 (2020) 109136.
[24] Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, Michael W. Mahoney, Characterizing possible failure modes in physics-informed neural networks, 2021, arXiv preprint arXiv:2109.01050.
[25] Revanth Mattey, Susanta Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Comput. Methods Appl. Mech. Engrg. 390 (2022) 114474.
[26] Walter A. Strauss, Partial Differential Equations: An Introduction, John Wiley & Sons, 2007.
[27] L.C. Evans, American Mathematical Society, Partial Differential Equations, in: Graduate studies in mathematics, American Mathematical Society, ISBN: 9780821807729, 1998.
[28] Isaac E. Lagaris, Aristidis Likas, Dimitrios I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000.
[29] Maziar Raissi, Hessam Babaee, Peyman Givi, Deep learning of turbulent scalar mixing, Phys. Rev. Fluids 4 (12) (2019) 124501.
[30] Ehsan Kharazmi, Zhongqiang Zhang, George Em Karniadakis, Variational physics-informed neural networks for solving partial differential equations, 2019, arXiv preprint arXiv:1912.00873.
[31] Andreas Griewank, Andrea Walther, Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation, SIAM, 2008.
[32] Diederik P. Kingma, Jimmy Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[33] Arieh Iserles, A First Course in the Numerical Analysis of Differential Equations, (44) Cambridge University Press, 2009.
[34] Arthur Jacot, Franck Gabriel, Clément Hongler, Neural tangent kernel: Convergence and generalization in neural networks, in: Advances in Neural Information Processing Systems, 2018, pp. 8571–8580.
[35] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, Qiao Zhang, JAX: composable transformations of python+numpy programs, 2018.
[36] Lu Lu, Xuhui Meng, Zhiping Mao, George E. Karniadakis, DeepXDE: A deep learning library for solving differential equations, 2019, arXiv preprint arXiv:1907.04502.
[37] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, Sanjay Choudhry, NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework, in: International Conference on Computational Science, Springer, 2021, pp. 447–461.
[38] Sifan Wang, Hanwen Wang, Paris Perdikaris, Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets, 2021, arXiv preprint arXiv:2103.10974.
[39] Sifan Wang, Paris Perdikaris, Long-time integration of parametric evolution equations with physics-informed DeepONets, 2021, arXiv preprint arXiv:2106.05384.

[40] Sifan Wang, Hanwen Wang, Paris Perdikaris, Improved architectures and training algorithms for deep operator networks, 2021, arXiv preprint arXiv:2110.01654.
[41] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, Anima Anandkumar, Physics-informed neural operator for learning partial differential equations, 2021, arXiv preprint arXiv:2111.03794.
[42] Judea Pearl, The seven tools of causal inference, with reflections on machine learning, Commun. ACM 62 (3) (2019) 54–60.
[43] Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, Yoshua Bengio, Toward causal representation learning, Proc. IEEE 109 (5) (2021) 612–634.
[44] Jean Kaddour, Aengus Lynch, Qi Liu, Matt J. Kusner, Ricardo Silva, Causal machine learning: A survey and open problems, 2022, arXiv preprint arXiv:2206.15475.
[45] Yonghan Jung, Jin Tian, Elias Bareinboim, Learning causal effects via weighted empirical risk minimization, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), in: Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 12697–12709.
[46] Tian Li, Ahmad Beirami, Maziar Sanjabi, Virginia Smith, Tilted empirical risk minimization, 2020, arXiv preprint arXiv:2007.01162.
[47] Jonathon Byrd, Zachary Lipton, What is the effect of importance weighting in deep learning? in: International Conference on Machine Learning, PMLR, 2019, pp. 872–881.
[48] Yifan Du, Tamer A. Zaki, Evolutional deep neural network, 2021, arXiv preprint arXiv:2103.09959.
[49] Shashank Reddy Vadyala, Sai Nethra Betgeri, Naga Parameshwari Betgeri, Physics-informed neural network method for solving one-dimensional advection equation using PyTorch, Array 13 (2022) 100110.
[50] Suchuan Dong, Naxian Ni, A method for representing periodic functions and enforcing exactly periodic boundary conditions with deep neural networks, J. Comput. Phys. 435 (2021) 110242.
[51] N. Sukumar, Ankit Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, 2021, arXiv preprint arXiv:2104.08426.
[52] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, Steven G. Johnson, Physics-informed neural networks with hard constraints for inverse design, 2021, arXiv preprint arXiv:2102.04626.
[53] Jesse Bettencourt, Matthew J. Johnson, David Duvenaud, Taylor-mode automatic differentiation for higher-order derivatives in JAX, 2019.
[54] Dimitris C. Psichogios, Lyle H. Ungar, A hybrid neural network-first principles approach to process modeling, AIChE J. 38 (10) (1992) 1499–1511.
[55] Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.
[56] Edward N. Lorenz, Deterministic nonperiodic flow, J. Atmos. Sci. 20 (2) (1963) 130–141.
[57] Yoshiki Kuramoto, Toshio Tsuzuki, Persistent propagation of concentration waves in dissipative media far from thermal equilibrium, Prog. Theor. Phys. 55 (2) (1976) 356–369.
[58] Gregory I. Sivashinsky, Nonlinear analysis of hydrodynamic instability in laminar flames—I. derivation of basic equations, Acta Astronaut. 4 (11) (1977) 1177–1206.
[59] Maziar Raissi, Deep hidden physics models: Deep learning of nonlinear partial differential equations, J. Mach. Learn. Res. 19 (1) (2018) 932–955.
[60] Michael Schaarschmidt, Dominik Grewe, Dimitrios Vytiniotis, Adam Paszke, Georg Stefan Schmid, Tamara Norman, James Molloy, Jonathan Godwin, Norman Alexander Rink, Vinod Nair, et al., Automap: Towards ergonomic automated parallelism for ML models, 2021, arXiv preprint arXiv:2112.02958.
[61] Khemraj Shukla, Ameya D. Jagtap, George Em Karniadakis, Parallel physics-informed neural networks via domain decomposition, 2021, arXiv preprint arXiv:2104.10013.
[62] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, Stephan Hoyer, Machine learning–accelerated computational fluid dynamics, Proc. Natl. Acad. Sci. (ISSN: 0027-8424) 118 (21) (2021) http://dx.doi.org/10.1073/pnas.2101784118.
[63] John D. Hunter, Matplotlib: A 2D graphics environment, IEEE Ann. Hist. Comput. 9 (03) (2007) 90–95.
[64] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, et al., Array programming with NumPy, Nature 585 (7825) (2020) 357–362.

A Nomenclature

Table 2 summarizes the main symbols and notations used in this work.

Notation	Description
PDE	Partial differential equation
PINN	Physics-informed neural network
NTK	Neural Tangent Kernel
$\boldsymbol{u}(\cdot)$	solution of a PDE
$\mathcal{N}[\cdot]$	a linear or non-linear differential operator
$\mathcal{B}[\cdot]$	a boundary operator
$u_{\boldsymbol{\theta}}(\cdot)$	neural network representation of the latent PDE solution
$\boldsymbol{\theta}$	all trainable parameters of a neural network
N_t	number of temporal collocation points
N_x	number of spatial collocation points
w_i	residual weights at time t_i
ϵ	causality parameter
δ	stopping criterion threshold for terminating a training loop
$\mathcal{L}_r(t, \boldsymbol{\theta})$	temporal residual loss
$\mathcal{L}(\boldsymbol{\theta})$	aggregate training loss

Table 2: *Nomenclature*: Summary of the main symbols and notations used in this work.

B Hyper-Parameters

Table 3 summarizes the network hyper-parameters for all numerical experiments. We tuned these hyper-parameters manually, without attempting to find the absolute best hyper-parameter setting. This process can be automated in the future leveraging effective techniques for meta-learning and hyper-parameter optimization [65].

Case	Architecture	Depth	Width	N_t	N_x
Allen-Cahn	MLP	6	128	100	256
	Modified MLP	6	128	100	256
Lorentz	MLP	5	512	256	-
Kuramoto–Sivashinsky (regular)	Modified MLP	5	256	32	64
Kuramoto–Sivashinsky (chaotic)	Modified MLP	10	128	32	256
Navier-Stokes	Modified MLP	6	128	64	512

Table 3: Network architectures for each benchmark employed in this work.

C Computational Cost

Training: Table 4 summarizes the computational cost of training PINNs. The size of different models as well as network architectures are listed table 3. All networks are trained using NVIDIA RTX A6000 graphics cards.

Case	Architecture	# Time windows	Max. Iterations	Training time (iter/sec)
Allen-Cahn	MLP	1	3×10^5	120.30
	Modified MLP	1	3×10^5	58.42
Lorentz	MLP	40	1×10^5	957.41
Kuramoto–Sivashinsky (regular)	Modified MLP	10	2×10^5	164.77
Kuramoto–Sivashinsky (chaotic)	Modified MLP	5	2×10^5	28.22
Navier-Stokes	Modified MLP	10	1×10^5	68.29

Table 4: Computational cost reported timings are obtained on NVIDIA RTX A6000 graphics cards. We remark that "Max Iteration" is the maximum iteration for every tolerance ϵ in each time window. The default tolerance list is $[10^{-2}, 10^{-1}, 10^0, 10^1, 10^2]$ unless otherwise stated. The total number of iterations may vary for different examples due to the stopping criterion (see Algorithm 1).

D Allen-Cahn equation

Validation: We solve the Allen-Cahn equation using conventional spectral methods. Specifically, assuming periodic boundary conditions, we start from the initial condition $u_0(x) = x^2 \cos(\pi x)$ and integrate the system up to the final time $T = 1$. Synthetic validation data are generated using the Chebfun package [66] with a spectral Fourier discretization with 512 modes and a fourth-order stiff time-stepping scheme (ETDRK4) [67] with time-step size 10^{-5}.

Ablation study on the causal parameter ϵ: We perform an ablation study on the ϵ using Allen-Cahn equation under the same hyper-parameter setup as before. Figure 18 visualizes the resulting relative L^2 errors for different causal parameter $\epsilon \in [10^{-4}, 10^4]$, averaged over 5 random seeds. Our results demonstrate that if ϵ falls below a certain threshold, then the PINN model behaves similarly to the conventional one, while too large values of ϵ can degrade the model performance. This observation indicates the presence of a sweet spot of the causal parameter ϵ and provide strong motivation for the annealing procedure proposed in Algorithm 1.

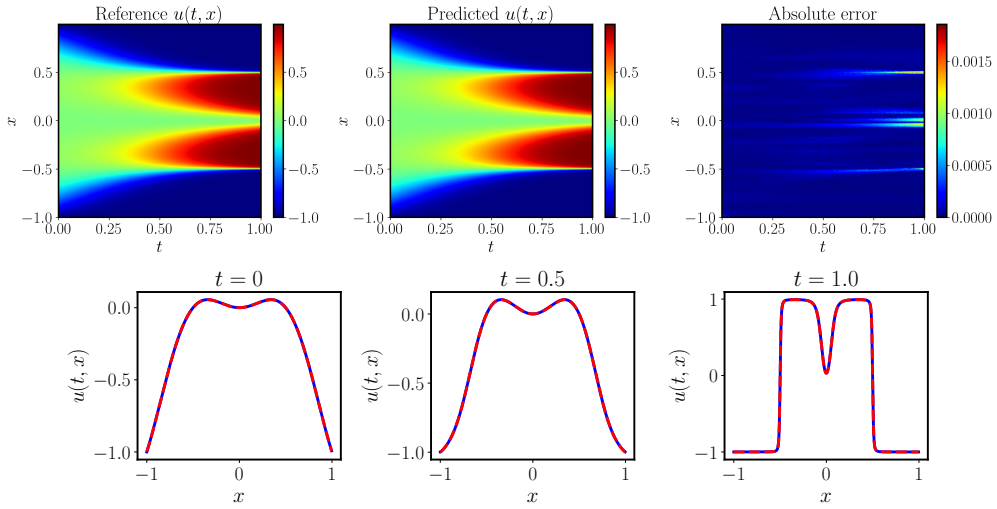

Figure 15: *Allen-Cahn equation: Top:* Exact solution versus the prediction of a trained physics-informed neural network using Algorithm 1 and modified MLP. The resulting relative L^2 error is $2.46e-04$. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots at $t = 0.0, 0.5, 1.0$.

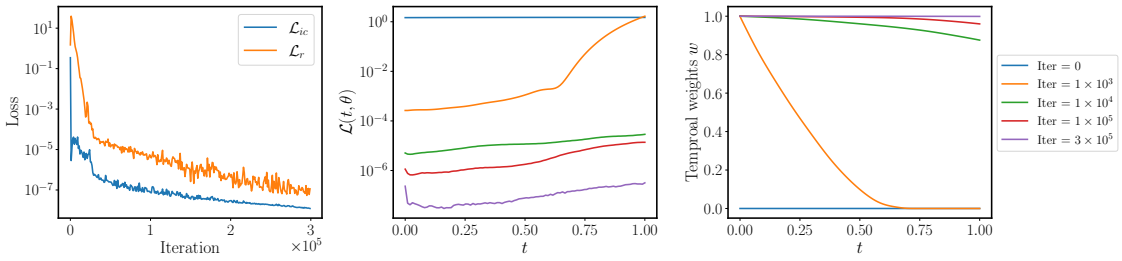

Figure 16: *Allen-Cahn equation: Left:* Loss convergence of training a physics-informed neural network using Algorithm 1. *Middle:* Temporal residual loss $\mathcal{L}(t, \boldsymbol{\theta})$ at different training iteration. *Right:* Temporal weights at different training iteration.

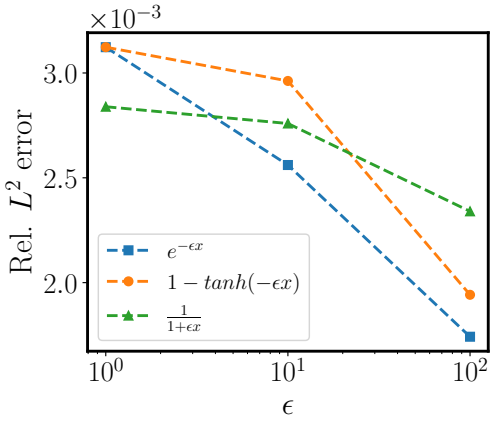

Figure 17: *Allen-Cahn equation*: Relative L^2 errors for different weighting functions and causality parameters under the same hyper-parameter setup.

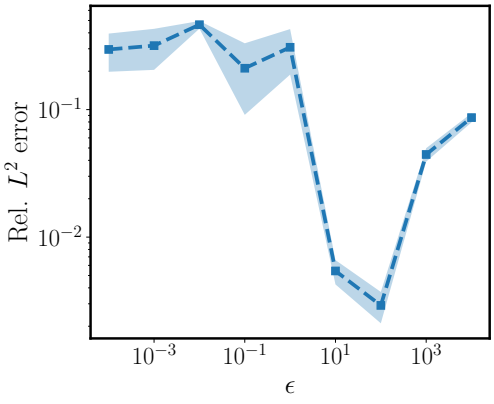

Figure 18: *Allen-Cahn equation*: Average relative L^2 errors over 5 random seeds for different fixed causal parameter $\epsilon \in [10^{-4}, 10^4]$.

E Lorentz system

Validation: The reference solution is obtained using `scipy.integrate.odeint` with default settings.

PINNs implementation: We split the whole domain $[0, 20]$ into 40 disjoint time windows of size $\Delta t = 0.5$. For each time window, we proceed by representing the latent variables of interest by a 5-layer fully-connected neural network $\boldsymbol{u_\theta}$ with 512 neurons per hidden layer

$$t \xrightarrow{\boldsymbol{u_\theta}} [x_{\boldsymbol{\theta}}, y_{\boldsymbol{\theta}}, z_{\boldsymbol{\theta}}]. \tag{E.1}$$

Since Lorentz system is highly sensitive to the initial condition, we exactly impose the initial condition by

$$\hat{x}_{\boldsymbol{\theta}}(t) = x_{\boldsymbol{\theta}}(t) \cdot t + x(0), \tag{E.2}$$
$$\hat{y}_{\boldsymbol{\theta}}(t) = y_{\boldsymbol{\theta}}(t) \cdot t + y(0), \tag{E.3}$$
$$\hat{z}_{\boldsymbol{\theta}}(t) = z_{\boldsymbol{\theta}}(t) \cdot t + z(0). \tag{E.4}$$

Then the loss function can be reduced to the residual loss

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_t} \sum_{i=1}^{N_t} w_i \left| \frac{\mathrm{d}\hat{x}_{\boldsymbol{\theta}}}{\mathrm{d}t}(t_i) - \sigma\left(\hat{y}_{\boldsymbol{\theta}}(t_i) - \hat{x}_{\boldsymbol{\theta}}(t_i)\right) \right| \tag{E.5}$$
$$+ \frac{1}{N_t} \sum_{i=1}^{N_t} w_i \left| \frac{\mathrm{d}\hat{y}_{\boldsymbol{\theta}}}{\mathrm{d}t}(t_i) - \hat{x}_{\boldsymbol{\theta}}(t_i)(\rho - \hat{z}_{\boldsymbol{\theta}}(t_i)) - \hat{y}_{\boldsymbol{\theta}}(t_i) \right| \tag{E.6}$$
$$+ \frac{1}{N_t} \sum_{i=1}^{N_t} w_i \left| \frac{\mathrm{d}\hat{z}_{\boldsymbol{\theta}}}{\mathrm{d}t}(t_i) - \hat{x}_{\boldsymbol{\theta}}(t_i)\hat{y}_{\boldsymbol{\theta}}(t_i) + \beta \hat{z}_{\boldsymbol{\theta}}(t_i) \right|, \tag{E.7}$$

where $\{t_i\}_{i=1}^{N_t}$ is a uniform grid in $[0, \Delta t]$. For this example, we set $N_t = 256$ and train the network with full-batch gradient descent. The temporal weights are updated by the proposed algorithm.

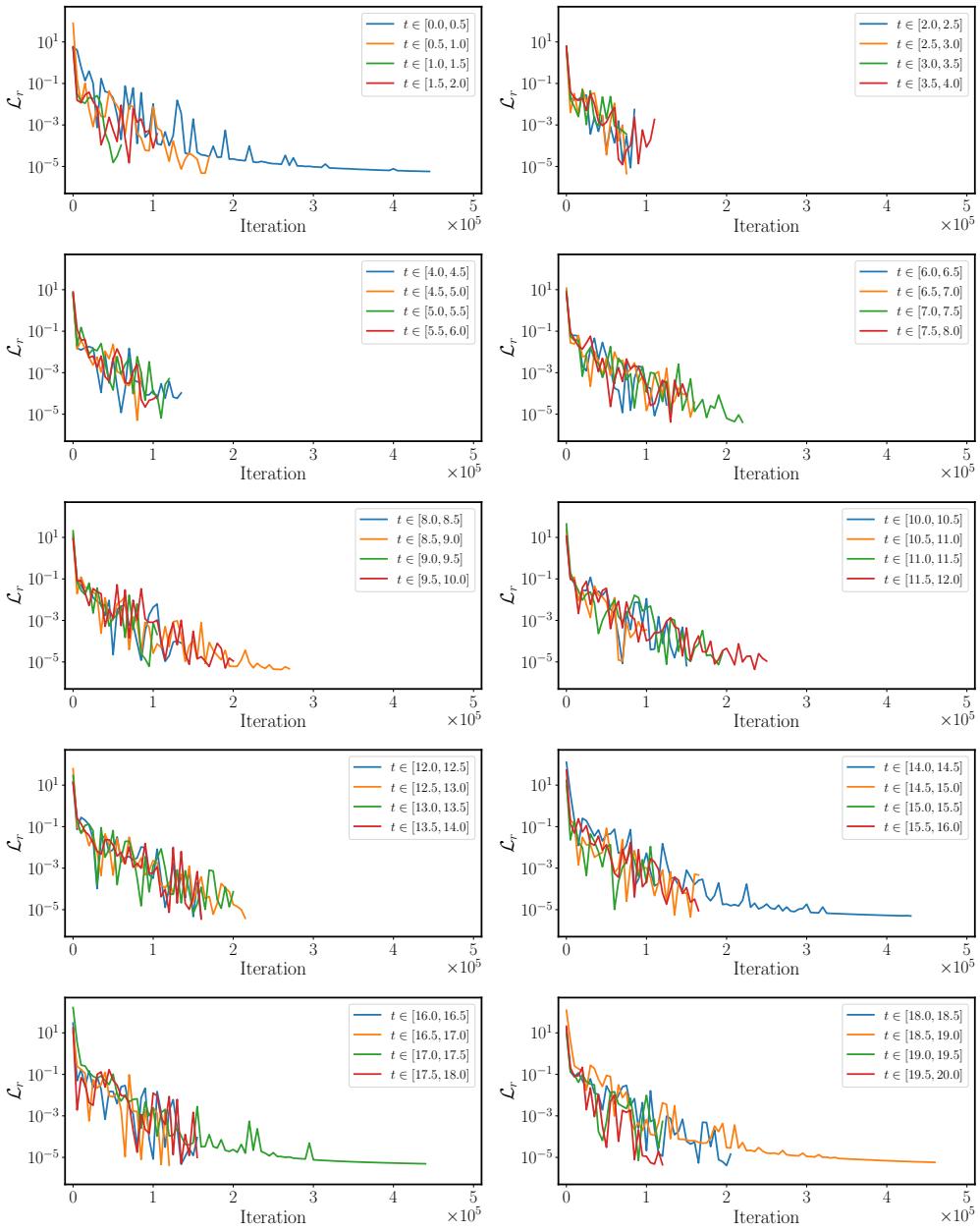

Figure 19: *Lorentz system: Left:* Loss convergence of training a physics-informed neural network using Algorithm 1 for every time window.

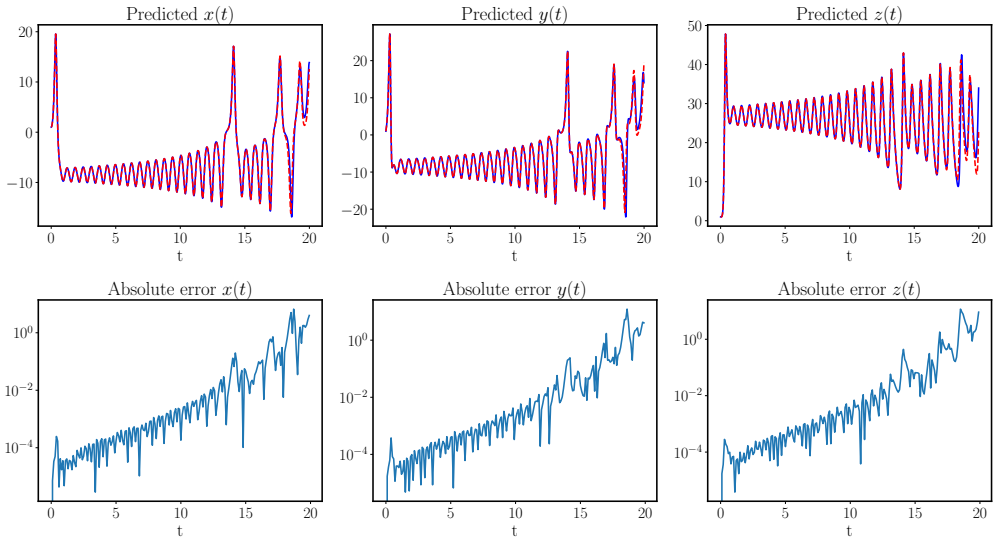

Figure 20: *Lorentz system:* Reference solutions versus the predicted solutions obtained by training a physics-informed neural network using Algorithm 1 with fixed iterations.

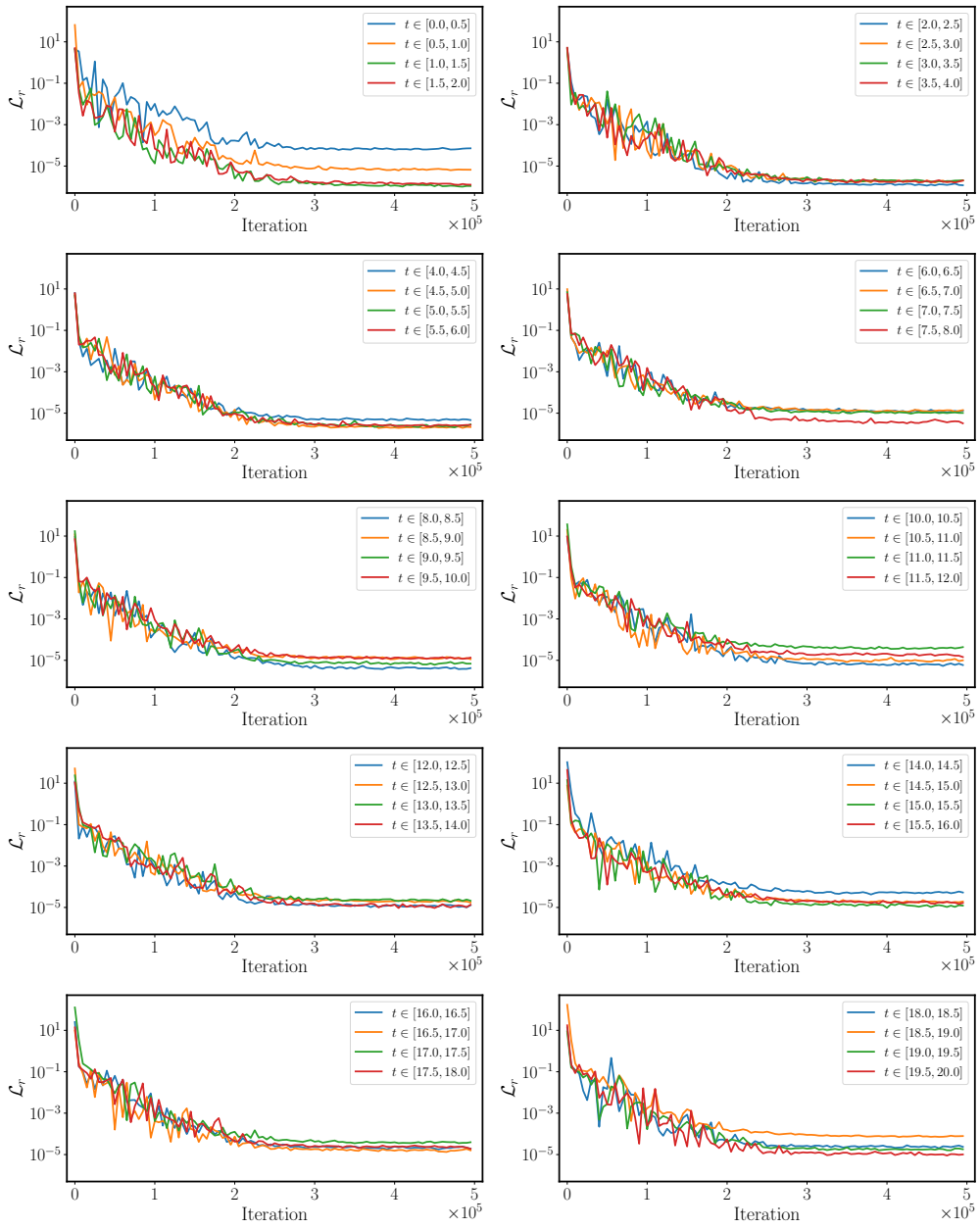

Figure 21: *Lorentz system: Left:* Loss convergence of training a physics-informed neural network using Algorithm 1 for every time window.

F Kuramoto–Sivashinsky equation

Validation: For case I (regular), we solve the Kuramoto–Sivashinsky equation using conventional spectral methods. Specifically, assuming periodic boundary conditions, we start from the initial condition $u_0(x) = -\sin(\pi x)$ and integrate the Equation 5.4 up to the final time $T = 1$. Synthetic validation data are generated using the Chebfun package [66] with a spectral Fourier discretization with 512 modes and a fourth-order stiff time-stepping scheme (ETDRK4) [67] with time-step size 10^{-5}. For case II (chaotic), we perform the same procedure with the initial condition $u_0(x) = \cos(x)(1 + \sin(x))$. Then we select the numerical solution at $t = 0.5$ as our initial condition for the PINNs simulation.

PINNs implementation: For Case I (regular), we split the temporal domain $[0, 1]$ into 10 time windows of size $\Delta t = 0.1$. Then we approximate the solution of each time window by a 5-layer modified MLP $u_{\boldsymbol{\theta}}$ with 256 neurons per hidden layer and encoded periodicity. It allows us to define the PDE residual by

$$\mathcal{R}[u_{\boldsymbol{\theta}}] = \frac{\partial u_{\boldsymbol{\theta}}}{\partial t} + \alpha u_{\boldsymbol{\theta}} \frac{\partial u_{\boldsymbol{\theta}}}{\partial x} + \beta \frac{\partial^2 u_{\boldsymbol{\theta}}}{\partial x^2} + \gamma \frac{\partial^4 u_{\boldsymbol{\theta}}}{\partial x^4}. \tag{F.1}$$

Then, we can formulate the following loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_t} \sum_{i=0}^{N_t} w_i \mathcal{L}(t_i, \boldsymbol{\theta}), \tag{F.2}$$

where

$$\mathcal{L}(t_0, \boldsymbol{\theta}) = \lambda_{ic} \frac{1}{N_x} \sum_{j=1}^{N_x} \left| u_{\boldsymbol{\theta}}(0, x_j) - u_0(x_j) \right|^2, \tag{F.3}$$

$$\mathcal{L}(t_i, \boldsymbol{\theta}) = \frac{1}{N_x} \sum_{j=1}^{N_x} \left| \mathcal{R}[u_{\boldsymbol{\theta}}](t_i, x_j) \right|^2, \text{ for } 1 \leq i \leq N_t. \tag{F.4}$$

Here we set $N_t = 32, N_x = 64$ and $\{t_i\}_{i=1}^{N_t}, \{x_j\}_{j=1}^{N_x}$ are randomly sampled in $[0, \Delta t]$ and $[-1, 1]$, respectively at each iteration of gradient descent. Particularly, we take $\lambda_{ic} = 10^3$ for better enforcing the initial condition. The network is trained by minimizing the above loss function via mini-batch gradient descent using the proposed algorithm.

For Case II (chaotic): We split the temporal domain $[0, 0.5]$ into 5 time windows of size $\Delta t = 0.1$. Then we perform the same procedure except for employing a 10-layer modified MLP with 128 neurons per hidden layer and setting $\lambda_{ic} = 10^4$.

Remark: For both cases, we employ Taylor-mode automatic differentiation [53] to accelerate the computation of high-order derivatives (see section 4).

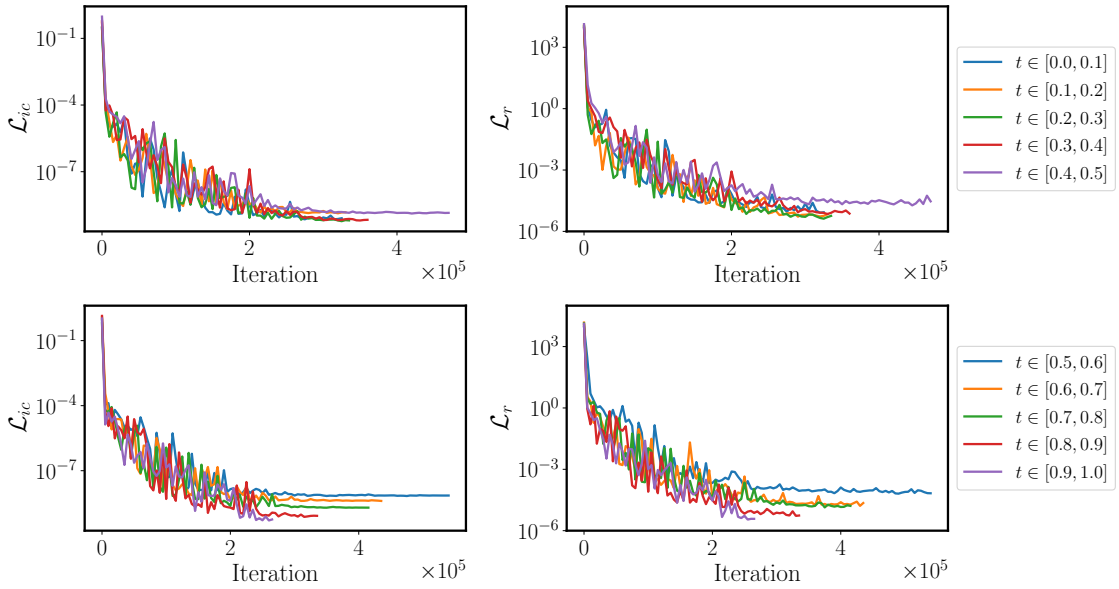

Figure 22: *Kuramoto–Sivashinsky equation (regular):* Loss convergence of training a physics-informed neural network using Algorithm 1 for every time window.

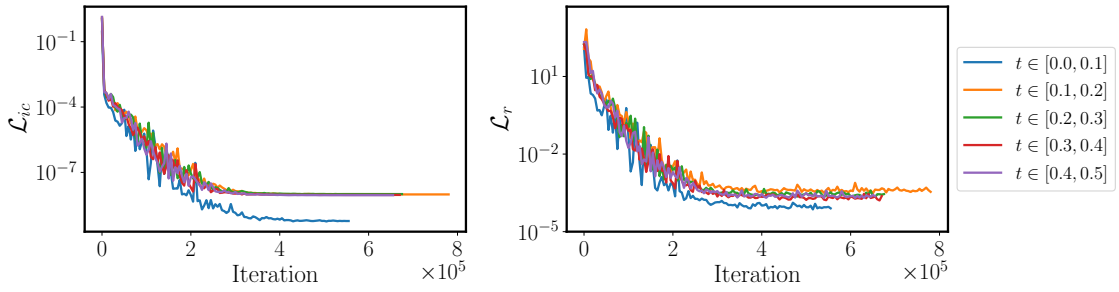

Figure 23: *Kuramoto–Sivashinsky equation (chaotic):* Loss convergence of training a physics-informed neural network using Algorithm 1 for every time window.

G Navier-Stokes equation

Validation: We simulate two-dimensional decaying turbulence in a periodic box using the JAX-CFD [62] incompressible Navier-Stokes solver. A high-resolution validation data-set is created by simulating an initial divergence free velocity field with the given maximum velocity $v_{\max} = 5$. The flow is solved using a Fourier spectral collocation method on a 1024×1024 uniform mesh with a time step of $dt = 10^{-4}$ [62].

PINNs implementation: Similar to the previous examples, the time domain $[0, 1]$ is decomposed into 10 time windows of size $\Delta t = 0.1$. We proceed by representing the velocity field by a 6-layer modified MLP with 128 neurons per hidden layer

$$[t, x, y] \xrightarrow{\boldsymbol{u_\theta}} [u_{\boldsymbol{\theta}}, v_{\boldsymbol{\theta}}]. \tag{G.1}$$

Then the vorticity can be approximated by $w_{\boldsymbol{\theta}} = \partial_x v_{\boldsymbol{\theta}} - \partial_y u_{\boldsymbol{\theta}}$ using automatic differentiation. Now we can define the PDE residual by

$$\mathcal{R}^w_{\boldsymbol{\theta}} = \frac{\partial w_{\boldsymbol{\theta}}}{\partial t} + u_{\boldsymbol{\theta}} \frac{\partial w_{\boldsymbol{\theta}}}{\partial x} + v_{\boldsymbol{\theta}} \frac{\partial w_{\boldsymbol{\theta}}}{\partial y} - \frac{1}{\mathrm{Re}} \left(\frac{\partial^2 w_{\boldsymbol{\theta}}}{\partial x^2} + \frac{\partial^2 w_{\boldsymbol{\theta}}}{\partial y^2} \right), \tag{G.2}$$

$$\mathcal{R}^c_{\boldsymbol{\theta}} = \frac{\partial u_{\boldsymbol{\theta}}}{\partial x} + \frac{\partial v_{\boldsymbol{\theta}}}{\partial y}. \tag{G.3}$$

It allows to define the loss function by

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N_t} \sum_{i=0}^{N_t} w_i \mathcal{L}(t_i, \boldsymbol{\theta}), \tag{G.4}$$

where

$$\mathcal{L}(t_0, \boldsymbol{\theta}) = \frac{\lambda_{ic}}{N_x} \sum_{j=1}^{N_x} \left| u_{\boldsymbol{\theta}}(0, x_j, y_j) - u_0(0, x_j, y_j) \right|^2 \tag{G.5}$$

$$+ \left| v_{\boldsymbol{\theta}}(0, x_j, y_j) - v_0(0, x_j, y_j) \right|^2 \tag{G.6}$$

$$+ \left| w_{\boldsymbol{\theta}}(0, x_j, y_j) - w_0(0, x_j, y_j) \right|^2 \tag{G.7}$$

and

$$\mathcal{L}(t_i, \boldsymbol{\theta}) = \frac{\lambda_w}{N_x} \sum_{j=1}^{N_x} \left| \mathcal{R}^w_{\boldsymbol{\theta}}(t_i, x_j, y_j) \right|^2 + \frac{\lambda_c}{N_x} \sum_{j=1}^{N_x} \left| \mathcal{R}^c_{\boldsymbol{\theta}}(t_i, x_j, y_j) \right|^2, \text{ for } 1 \leq i \leq N_t. \tag{G.8}$$

For this example we set $N_t = 64$, $N_x = 512$ and $\lambda_w = 1, \lambda_c = 10^2, \lambda_{ic} = 10^4$. The temporal and spatial collocation points are randomly sampled from $[0, 1]$ and $[0, 2\pi]^2$, respectively. It is worth noting that we also enforce the initial velocity field (u_0, v_0) as additional constraints for better convergence. This is not a severe restriction since the velocity field can be obtained from the vorticity by solving the associated Poisson's equation or from the network representation directly.

Furthermore, in Appendix we also present our results simulating the turbulent flow up to $T = 2$. Figure 28 presents the visualizations of the predicted velocity and vorticity field at the final state. The predictive accuracy is quantified in Figure 29. Although the resulting relative L^2 error is above 10%, our model predictions seem to be qualitatively correct against the corresponding ground truth.

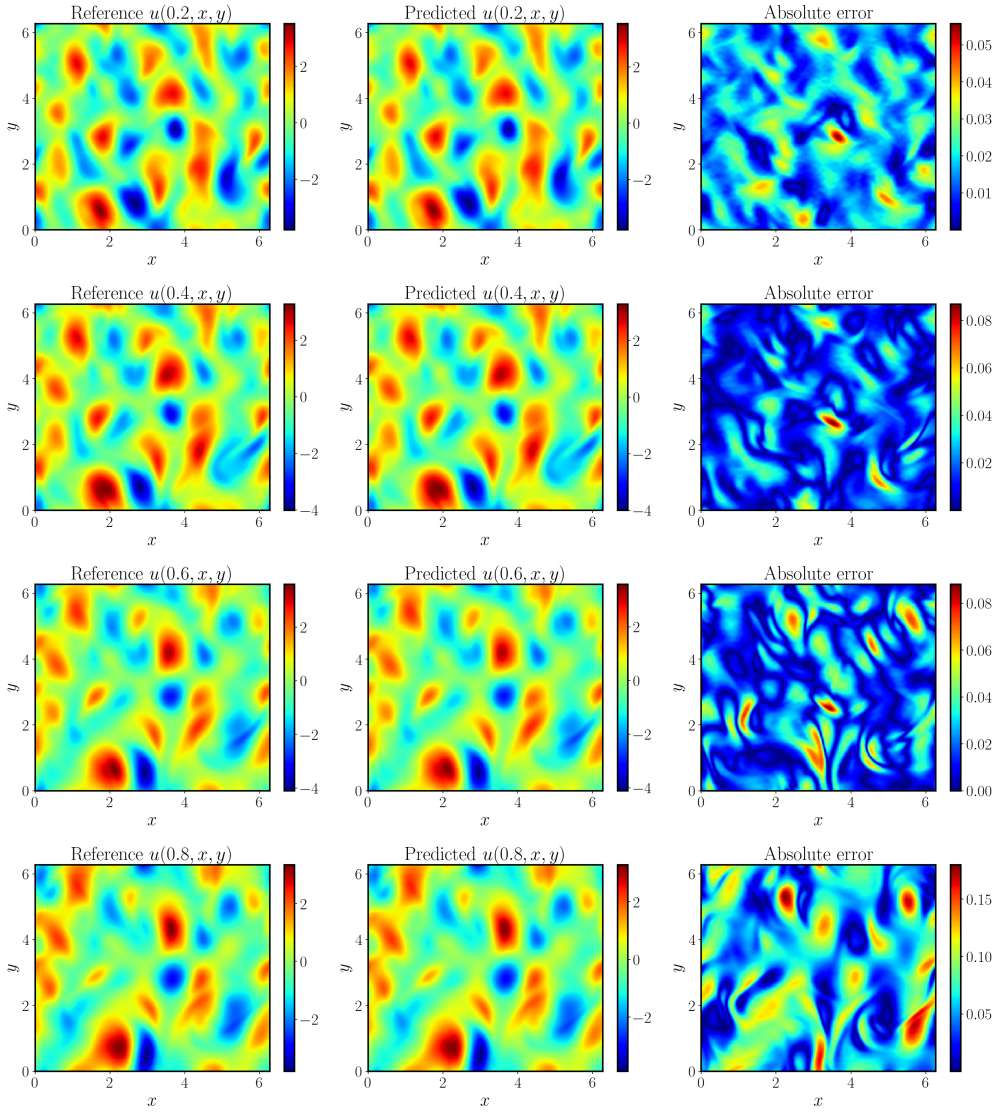

Figure 24: *Navier-Stokes:* Representative snapshots of the predicted u against the ground truth at $t = 0.2, 0.4, 0.6, 0.8$.

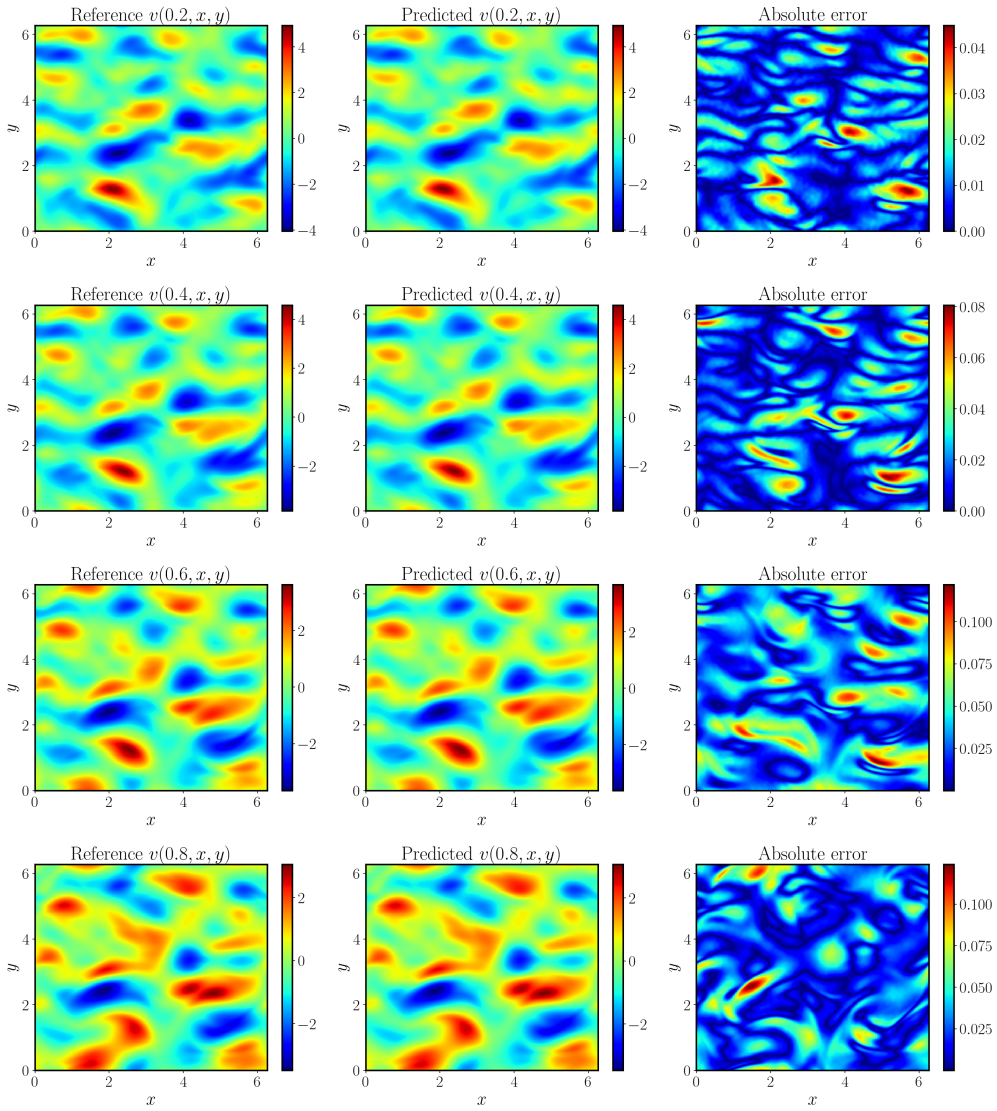

Figure 25: *Navier-Stokes:* Representative snapshots of the predicted v against the ground truth at $t = 0.2, 0.4, 0.6, 0.8$.

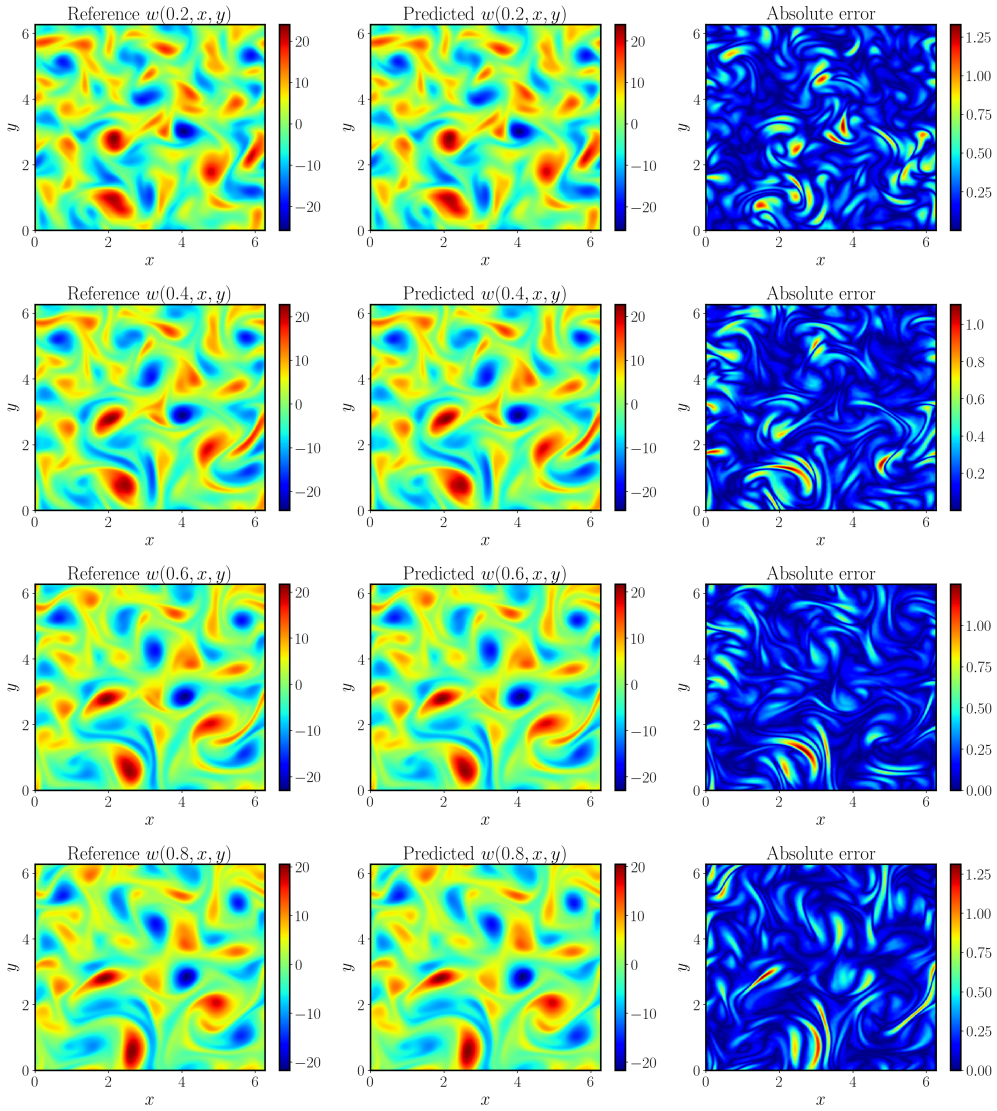

Figure 26: *Navier-Stokes:* Representative snapshots of the predicted w against the ground truth at $t = 0.2, 0.4, 0.6, 0.8$.

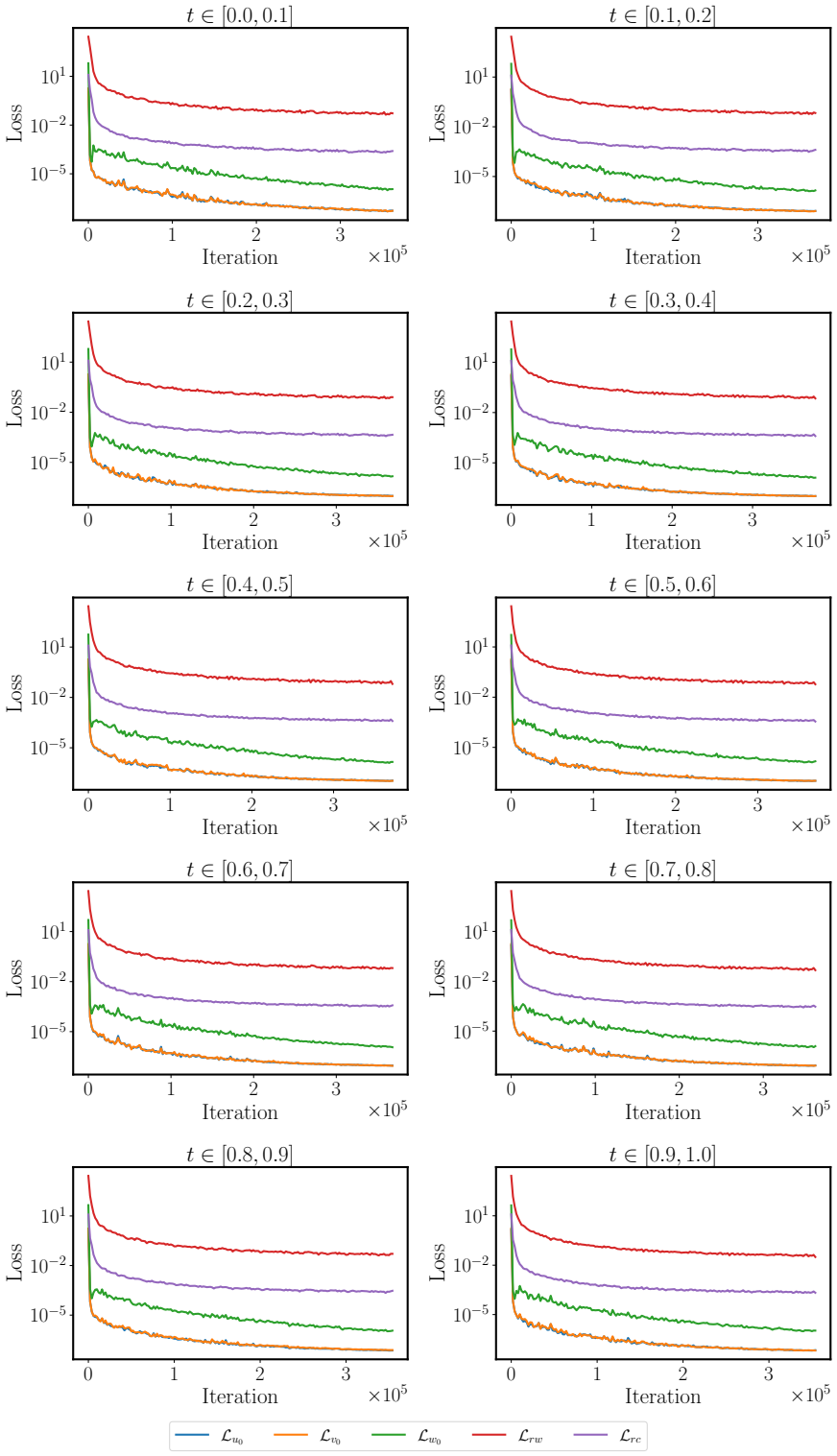

Figure 27: *Navier-Stokes:* Loss convergence of training a physics-informed neural network using Algorithm 1 for every time window.

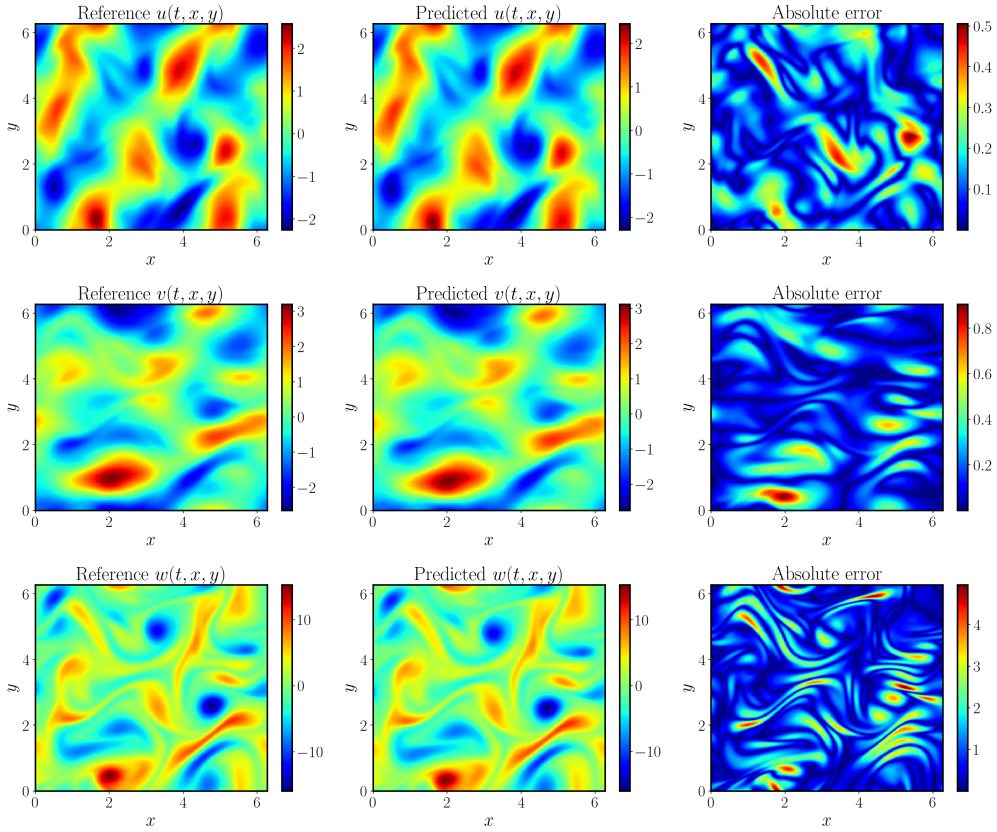

Figure 28: *Navier-Stokes:* Predicted u, v, w against the ground truth at $t = 2$.

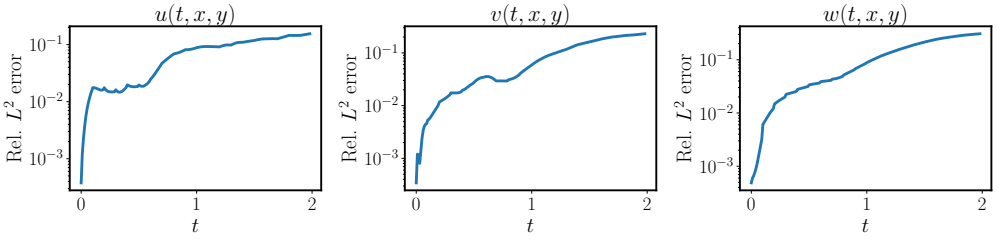

Figure 29: *Navier-Stokes:* Relative L^2 errors of u, v, w, respectively.

H High-dimensional ODE system: Lorentz 96

Here we examine the proposed algorithm for physics-informed DeepONets and observed similar improvements to PINNs. To demonstrate this, we considered a classical example of a dynamical system known as Lorenz 96, which has the form of

$$\begin{aligned}\frac{dx_i}{dt} &= (x_{i+1} - x_{i-2})\, x_{i-1} - x_i + F, \quad i = 1, \dots, D,\\ \boldsymbol{x}(0) &= \boldsymbol{u}\end{aligned}$$

where $x_{-1} = x_{D-1}, x_0 = x_D$ and $x_{D+1} = x_1$. Here x_i denotes the state of the system and F is a a forcing constant.

Our goal is to train a physics-informed DeepONet $s_\theta(\boldsymbol{u})$ to learn the solution operator from the initial condition to the corresponding solution $x(t)$ in $[0, T]$. In this example we take, $D = 64, F = 3, T = 5$. In particular, we enforce the initial condition by modifying the original model outputs as follows:

$$\hat{s}_\theta(\boldsymbol{u})(t) = t \cdot s_\theta(\boldsymbol{u}) + \boldsymbol{u}. \tag{H.1}$$

Consequently, one can train a physics-informed DeepONet by only minimizing the PDE residual loss

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{NMD} \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{k=1}^{D} \left| \mathcal{R}^{(k)}_{\boldsymbol{\theta}}[\boldsymbol{u}^{(i)}](t^{(j)}) \right|^2,$$

where $\mathcal{R}^{(k)}$ is defined as

$$\mathcal{R}^{(k)}[u](t) = \frac{ds^k_\theta(u)}{dt} - (s^{k+1}_\theta(u) - s^{k-2}_\theta(u)) s^{k-1}_\theta(u) + s^k_\theta(u) - F. \tag{H.2}$$

Here $\{\boldsymbol{u}^{(i)}\}_{i=1}^N$ represents a set of initial conditions sampled from a uniform distribution $U(0, 1)$, and $\{t^j\}_{j=1}^M$ denotes a set of evenly-spaced collocation points in [0, T]. It worth emphasising again that this model is trained without labeled data except for the given initial conditions.

Similar to PINNs case, we impose the casual structure by modifying the loss function as

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{NMD} \sum_{i=1}^{N} \sum_{j=1}^{M} w_{ij} \left[\sum_{k=1}^{D} \left| \mathcal{R}^{(k)}_{\boldsymbol{\theta}}[\boldsymbol{u}^{(i)}](t^{(j)}) \right|^2 \right],$$

with

$$w_{ij} = \exp\left(-\epsilon \sum_{l=1}^{j} \sum_{k=1}^{D} \left| \mathcal{R}^{(k)}_{\boldsymbol{\theta}}[\boldsymbol{u}^{(i)}](t^{(l)}) \right|^2 \right). \tag{H.3}$$

We employ a 7-layer DeepONet with 1024 neurons per hidden layer to represent the solution operator and train the model by minimizing the losses described in equations H.2 and H.3, respectively. The standard physics-informed DeepONet achieves a test error of 47.83% over 100 examples, while the causal physics-informed DeepONet attains a test error of 5.26%. A comparison of one representative example is visualized in Figure 30 and 31, respectively. It is evident that the prediction of the causal physics-informed DeepONet achieves a good agreement with the reference solution. This result illustrates the generality of our method for any physics-informed pipeline.

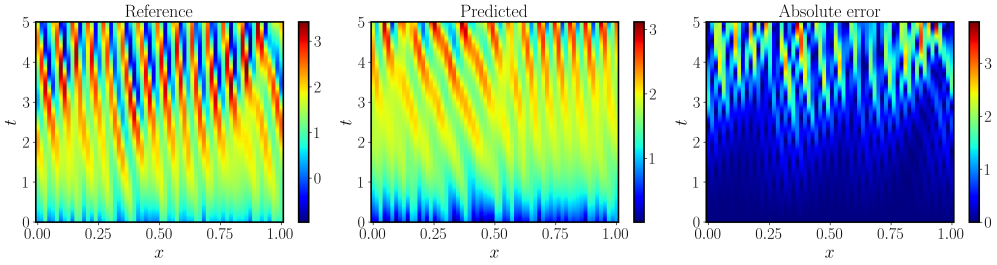

Figure 30: *Lorentz 96*: Representative predicted solution of training a conventional physics-informed DeepONet [38].

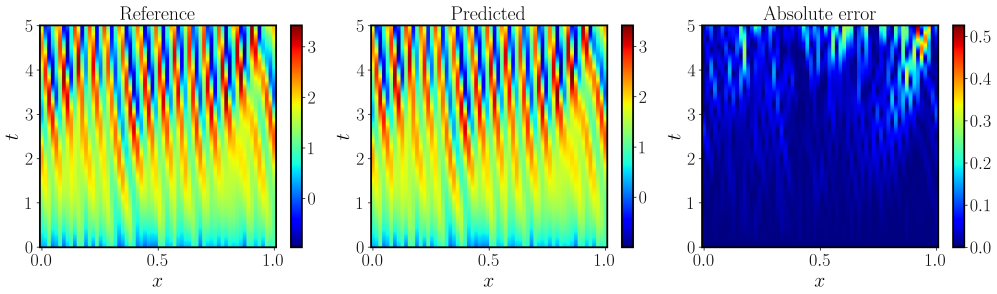

Figure 31: *Lorentz 96*: Representative predicted solution of training a physics-informed DeepONet with the proposed algorithm.