Causality-enhanced Discreted Physics-informed Neural Networks for Predicting Evolutionary Equations

Ye Li[1,3*], **Siqi Chen**[2], **Bin Shan**[1] and **Sheng-Jun Huang**[1,3]
[1]College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics
[2]College of Electronic and Information Engineering, Nanjing University of Aeronautics and Astronautics
[3]MIIT Key Laboratory of Pattern Analysis and Machine Intelligence, Nanjing, China
{yeli20, chensq, bin.shan, huangsj}@nuaa.edu.cn

Abstract

Physics-informed neural networks (PINNs) have shown promising potential for solving partial differential equations (PDEs) using deep learning. However, PINNs face training difficulties for evolutionary PDEs, particularly for dynamical systems whose solutions exhibit multi-scale or turbulent behavior over time. The reason is that PINNs may violate the temporal causality property since all the temporal features in the PINNs loss are trained simultaneously. This paper proposes to use implicit time differencing schemes to enforce temporal causality, and use transfer learning to sequentially update the PINNs in space as surrogates for PDE solutions in different time frames. The evolving PINNs are better able to capture the varying complexities of the evolutionary equations, while only requiring minor updates between adjacent time frames. Our method is theoretically proven to be convergent if the time step is small and each PINN in different time frames is well-trained. In addition, we provide state-of-the-art (SOTA) numerical results for a variety of benchmarks for which existing PINNs formulations may fail or be inefficient. We demonstrate that the proposed method improves the accuracy of PINNs approximation for evolutionary PDEs and improves efficiency by a factor of 4–40x. The code is available at https://github.com/SiqiChen9/TL-DPINNs.

1 Introduction

Evolutionary partial differential equations (PDEs) are representative of the real world, such as the Navier–Stokes equation, Cahn–Hilliard equations, wave equation, Korteweg–De Vries equation, etc., which arise from physics, mechanics, material science, and other computational science and engineering fields [Dafermos and Pokorny, 2008]. Due to the inherent universal approximation capability of neural networks and the exponential growth of data and computational resources, neural network PDE solvers have recently gained popularity [Raissi *et al.*, 2017; Han *et al.*, 2018; Khoo *et al.*, 2021; Yu and E, 2018; Sirignano and Spiliopoulos, 2018; Long *et al.*, 2018]. The most representative approach among these neural network PDE solvers is Physics-Informed Neural Networks (PINNs) [Raissi *et al.*, 2019]. PINNs have been utilized effectively to solve PDE problems such as the Poisson equation, Burgers equation, and Navier-Stokes equation [Raissi *et al.*, 2019; Lu *et al.*, 2021a; Mishra and Molinaro, 2023]. Many variants of PINNs include loss reweighting [Wang *et al.*, 2021a; Wang *et al.*, 2022b; Wang *et al.*, 2022a; Krishnapriyan *et al.*, 2021], novel optimization targets [Jagtap *et al.*, 2020; Kharazmi *et al.*, 2021], novel architectures [Jagtap *et al.*, 2020; Jagtap and Karniadakis, 2021; Wang *et al.*, 2021b] and other techniques such as transfer learning and meta-learning [Goswami *et al.*, 2020; Liu *et al.*, 2022b], have also been explored to enhance training and test accuracy.

When we apply neural networks to solve evolutionary PDEs, the most ubiquitously used PINN implementation at present is the meshless, continuous-time PINN in [Raissi *et al.*, 2019]. However, training (i.e., optimization) is still the primary challenge when employing this approach, particularly for dynamical systems whose solutions exhibit strong non-linearity, multi-scale features, and high sensitivity to initial conditions, such as the Kuramoto-Sivashinsky equation and the Navier-Stokes equations in the turbulent regime. While advanced machine learning techniques may help reducing the difficulty of training [Yang *et al.*, 2019; Hou *et al.*, 2022], more researchers try to find the reasons of training failures.

Recently Wang et al. [Wang *et al.*, 2022a] revealed that continuous-time PINNs can violate the so-called *temporal causality* property, and are therefore prone to converge to incorrect solutions. Temporal causality requires that models should be sufficiently trained at time t before approximating the solution at the later time $t + \Delta t$, while continuous-time PINNs are trained for all time t simultaneously. To enhance the temporal causality in the training process, they proposed a simple re-formulation of PINNs loss functions as shown in (1), i.e., a clever weighting technique that is inversely exponentially proportional to the magnitude of cumulative residual losses from prior times. This casual PINN method has been demonstrated to be effective for some difficult problems. However their method is sensitive to the new causality hyperparameter ϵ, and the training time is substantially longer than

*Corresponding author.

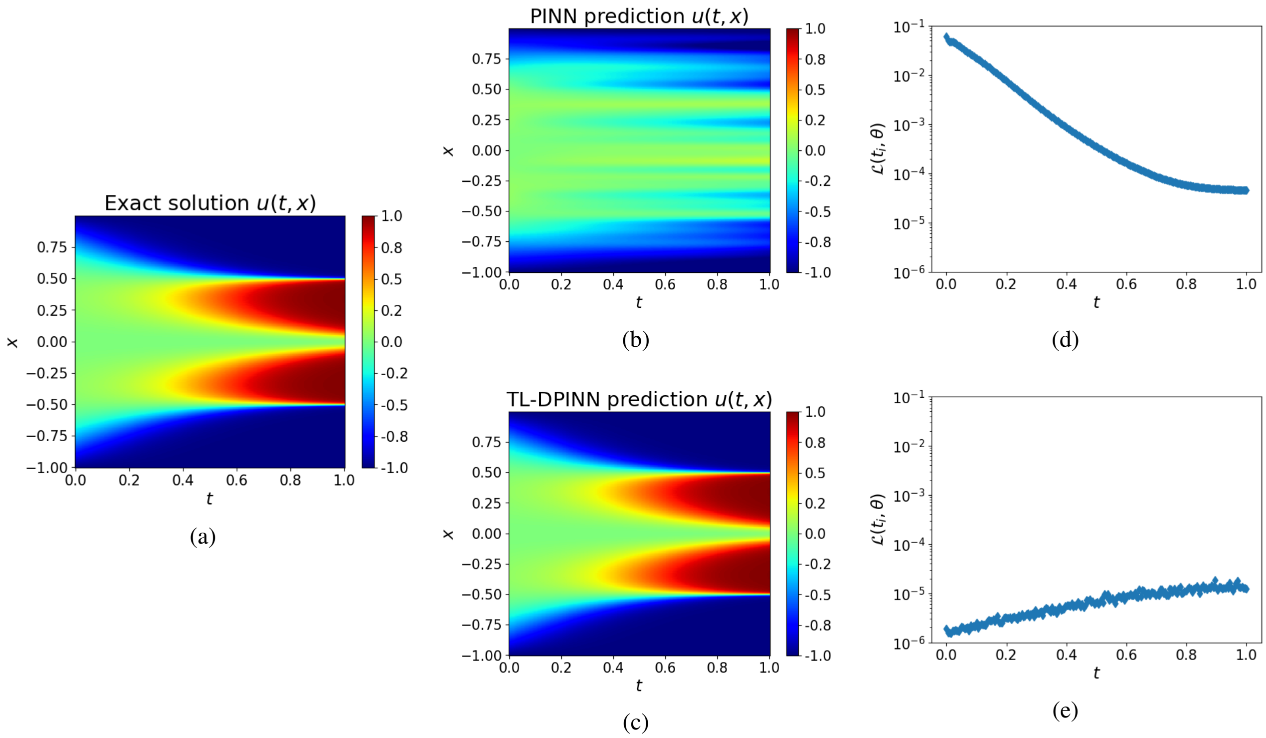


Figure 1: Allen-Cahn equation: (a)Exact solution. (b)PINN solution. (c)TL-DPINN solution. (d)PINN's temporal residual loss $\mathcal{L}_r(t_n, \theta)$, small for later time while large for initial time, violating temporal causality principle. (e)TL-DPINN's temporal residual loss $\mathcal{L}_r(t_n, \theta)$, respecting temporal causality principle and keeping small for all time.

vanilla PINNs.

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

In this paper, we introduce a new PINN implementation technique for efficiently and precisely solving evolutionary PDEs. Our technique relies on two key elements: **(a)** using discrete-time PINNs instead of continuous-time PINNs to satisfy the principle of temporal causality. Implicit time differencing schemes are stable and can enable solutions to be learned from earlier times to later times, thereby making the training process stable and accurate; and **(b)** utilizing transfer learning to accelerate PINN training in later time frames. In the following sections, we will show that our causality-enhanced discrete physics-informed neural networks with transfer learning accelerating (TL-DPINN) method is theoretically and numerically stable, accurate, and efficient.

Following is a summary of the contribution of the paper.

- Implicit time differencing with the transfer-learning tuned PINN provides more accurate and robust predictions of evolutionary PDEs' solutions while retaining a low computational cost.
- We prove theoretically the error estimation result of our TL-DPINN method, indicating that TL-DPINN solutions converge as long as the time step is small and each PINN in different time frames is well trained.
- Through extensive numerical results, we demonstrate that our method can attain state-of-the-art (SOTA) performance among various PINN frameworks in a trade-off between accuracy and efficiency.

2 Related Works

Discrete PINN. Raissi et al. [Raissi *et al.*, 2019] have applied the general form of Runge–Kutta methods with arbitrary q stages to the evolutionary PDEs. However, only an implicit Runge-Kutta scheme with $q = 100$ stages and a single large time step $\Delta t = 0.8$ are computed. Low-order methods cannot retain their predictive accuracy for large time steps. In our research, we demonstrate the capability of discrete PINNs both theoretically and experimentally, indicating that robust low-order implicit Runge-Kutta combined with PINN can obtain high-precision solutions with multiple small-sized time steps. Jagtap and Karniadakis [Jagtap and Karniadakis, 2021] propose a generalized domain decomposition framework that allows for multiple subnetworks over different subdomains to be stitched together and trained in parallel. However, it is not causal and has the same training issues as conventional PINNs. The implicit Runge-Kutta scheme combined with PINN has been used to solve simple ODE systems [Stiasny *et al.*, 2021; Moya and Lin, 2023], but not dynamic PDE systems with multi-scale or turbulent behavior over time.

Temporal Decomposition. Diverse strategies have been studied for enhancing PINN training by splitting the domain into numerous small “time-slab”. Wight and Zhao [L. Wight and Zhao, 2021] propose an adaptive time-sampling strategy to learn solutions from the previous small time domain to the whole time domain. However, collocation points are costly to add, and the computational cost rises. This time marching strategy has been enhanced further in [Krishnapriyan *et al.*, 2021; Mattey and Ghosh, 2022; McClenny and Braga-Neto, 2023]. Nevertheless, causality is only enforced on the scale of the time slabs and not inside each time slab, thus the convergence can not be theoretically guaranteed. A unified framework for causal sweeping strategies for PINNs is summarized in [Penwarden *et al.*, 2023]. Wang et al. [Wang *et al.*, 2022a] introduced a simple causal weight in the form of (1) to naturally match the principle of temporal causality with high precision. However, this significantly increased computational costs and did not guarantee convergence [Penwarden *et al.*, 2023]. Our methods can attain the same level of precision, are theoretically convergent, and are 4 to 40 times quicker.

Transfer Learning. Transfer-learning has been previously combined with various deep-learning models for solving PDEs problems, such as PINN for phase-field modeling of fracture [Goswami *et al.*, 2020], DeepONet for PDEs under conditional shift [Goswami *et al.*, 2022], DNN-based PDE solvers [Chen *et al.*, 2021], PINN for inverse problems [Xu *et al.*, 2023], one-shot transfer learning of PINN [Desai *et al.*, 2022], and training of CNNs on multi-fidelity data [Song and Tartakovsky, 2022]. Xu et al. [Xu *et al.*, 2022] proposed a transfer learning enhanced DeepONet for the long-term prediction of evolution equations. However, their method necessitates a substantial amount of training data from traditional numerical methods. In contrast, our methods are physics-informed and do not require additional training data.

3 Numerical Method

Problem Set-up Here we consider the initial-boundary value problem for a general evolutionary parabolic differential equation. The extension to hyperbolic equations are straightforward.

$$\begin{cases} u_t = \mathcal{N}(u), & x \in \Omega, t \in [0,T], \\ u(0,x) = u_0(x), & x \in \Omega, \\ u(t,x) = g(t,x), & t \in [0,T], x \in \partial\Omega, \end{cases} \tag{2}$$

where $u(t,x)$ denotes the hidden solution, t and x represent temporal and spatial coordinates respectively, $\mathcal{N}(u)$ denotes a differential operator (for example, $\mathcal{N}(u) = u_{xx}$ for the simplest Heat equation), and $\Omega \subset \mathbb{R}^D$ is an open, bounded domain with smooth boundary $\partial\Omega$. This study assumes that the equations are dissipative in the sense that $\int_\Omega u \cdot \mathcal{N}(u)dx \leq 0$ [Xu *et al.*, 2022].

Our goal is to learn $u(t,x)$ by neural network approximation. We briefly mention the basic background of PINN in Section 3.1 and then describe our TL-DPINN method in Section 3.2.

3.1 Physics-informed Neural Networks

In the original study of PINNs [Raissi *et al.*, 2019], it approximates $u(t,x)$ to (2) using a deep neural network $u_\theta(t,x)$, where θ represents the neural network's parameters (e.g., weights and biases). Consequently, the objective of a vanilla PINN is to discover the θ that minimizes the physics-based loss function:

$$\mathcal{L}(\theta) = \lambda_b\mathcal{L}_b(\theta) + \lambda_u\mathcal{L}_u(\theta) + \lambda_r\mathcal{L}_r(\theta), \tag{3}$$

where

$$\mathcal{L}_b(\theta) = \frac{1}{N_b}\sum_{i=1}^{N_b} \|u_\theta(t_b^i, x_b^i) - g(t_b^i, x_b^i)\|^2, \tag{4}$$

$$\mathcal{L}_u(\theta) = \frac{1}{N_u}\sum_{i=1}^{N_u} \|u_\theta(0, x_t^i) - u_0(x_t^i)\|^2, \tag{5}$$

$$\mathcal{L}_r(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r} \|\mathcal{R}(u_\theta(t_r^i, x_r^i)\|^2. \tag{6}$$

The t_b^i, x_b^i, x_t^i represent the boundary and initial sampling data for $u_\theta(t,x)$, whereas t_r^i, x_r^i represent the data points utilized to calculate the residual term $\mathcal{R}(u) = u_t - \mathcal{N}(u)$. The coefficients λ_b, λ_u, and λ_r in the loss function are utilized to assign a different learning rate, which can be specified by humans or automatically adjusted during training[Wang *et al.*, 2021a; Wang *et al.*, 2022b]. We note that the $\mathcal{L}_b$ term can be further omitted if we apply hard constraint in the PINN's design [Lu *et al.*, 2021b; Liu *et al.*, 2022a; Sukumar and Srivastava, 2022].

As demonstrated in [Wang *et al.*, 2022a], the vanilla PINN may violate the principle of temporal causality, as the residual loss at the later time may be minimized even if the predictions at previous times are incorrect. Figure 1 demonstrates the training result for solving the Allen-Chan equation, confirming this phenomenon. For conventional PINN, the residual loss $\mathcal{L}_r$ is large near the initial state and small for the later time while the learned solution is incorrect. Comparatively, our method's residual remains small for all $t \in [0,1]$ and captures the solution with high precision.

3.2 Causality-enhanced Discrete PINN

Discrete PINN. Since the continuous-time PINN violates temporal causality, we shift to numerical temporal differencing schemes that naturally respect temporal causality. Given a time step Δt, assume we have computed $u^n(x)$ to approximate the solution $u(n\Delta t, x)$ to (2), then we consider finding $u^{n+1}(x)$ by the Crank-Nicolson time differencing scheme:

$$\frac{u^{n+1}(x) - u^n(x)}{\Delta t} = \mathcal{N}\left[\frac{u^{n+1}(x) + u^n(x)}{2}\right]. \tag{7}$$

Instead of solving (2) in the whole space-temporal domain directly, our goal is to solve (7) from one step to the next in the space domain: $u_0(x) \mapsto u^1(x) \mapsto \cdots \mapsto u^n(x) \mapsto u^{n+1}(x) \mapsto \cdots$, so that the evolutionary dynamics can be captured over a long time horizon.

Next, we apply PINN to solve (7). It is also called discrete PINN in [Raissi *et al.*, 2019] when the Crank-Nicolson scheme is replaced by implicit high-order Runge-Kutta schemes. Assuming we have obtained a neural network $u_{\theta^n}(x)$ to approximate $u(n\Delta t, x)$ in (2), we compute

$u_{\theta^{n+1}}(x)$ by finding another new θ^{n+1} that minimize the loss functions :

$$\mathcal{L}^{n+1}(\theta^{n+1}) = \frac{\lambda_b}{N_b}\sum_{i=1}^{N_b}\left|u_{\theta^{n+1}}(x_b^i) - g(x_b^i)\right|^2 + \frac{\lambda_r}{N_r}\cdot \sum_{i=1}^{N_r}\left|\frac{u_{\theta^{n+1}}(x_r^i) - u_{\theta^n}(x_r^i)}{\Delta t} - \mathcal{N}\left[\frac{u_{\theta^{n+1}}(x_r^i) + u_{\theta^n}(x_r^i)}{2}\right]\right|^2. \tag{8}$$

These multiple PINNs $u_{\theta^n}(x)$ take x as input and output the solution values at different timestamps.

Remark 1. *We remark that there exist alternative options for time differencing beyond the second-order Crank-Nicolson scheme. Several implicit Runge-Kutta schemes, including the first-order backward Euler scheme and the fourth-order Gauss-Legendre scheme, have been found to be effective. The second-order Crank-Nicolson scheme is favored due to its optimal trade-off between computational efficiency and numerical accuracy. A comprehensive exposition of these techniques is available in Appendix A.2 of [Chen* et al.*, 2023].*

Transfer Learning. The transfer learning methodology is utilized to expedite the training procedure between two adjacent PINNs. All the PINNs $u_{\theta^n}(x)$ share the same neural network architectures with different parameters θ^n. For a small time step Δt, there are little difference between the two adjacent PINNs $u_{\theta^n}(x)$ and $u_{\theta^{n+1}}(x)$. So the parameters θ^{n+1} to be trained are very close to the trained parameters θ^n. When training $u_{\theta^{n+1}}(x)$, it is sufficient to initialize the weight parameters θ^{n+1} as the well trained parameters θ^n. Alternatively freezing a significant portion of the well-trained $u_{\theta^n}(x)$ and solely updating the weights in several hidden layers through the application of a comparable physics-informed loss function (8) are also considerable. We did experiments to fine tune all the network parameters as well as fine tuning the last 1/2/3 layers of the network in the ablation study Section 5.5.

To be more precise, we first approximate the initial condition $u_0(x)$ by the neural network $u_{\theta^0}(x)$, then learn $u_{\theta^1}(x), u_{\theta^2}(x), \ldots$ sequentially by transfer learning. We fine tune the well-trained parameters θ^n to accelerate searching the optimized parameters θ^{n+1}. The general structure of our TL-DPINN method is illustrated in Algorithm 1.

4 Theoretical Result

In this section, we analyze the TL-DPINN method and give an error estimate result to approximate the evolutionary differential equation (2). We have two reasonable assumptions as follows.

Assumption 1. *The equation* (2) *is dissipative, i.e.* $\int_\Omega u \cdot \mathcal{N}(u)dx \le 0$ *for all* $u(t,x)$. *Moreover, if* $\mathcal{N}$ *is nonlinear, then* $\int_\Omega (u_1 - u_2)\cdot(\mathcal{N}(u_1) - \mathcal{N}(u_2))dx \le 0$ *for all* $u_1(t,x)$ *and* $u_2(t,x)$.

Assumption 2. *The solution* $u(t,x)$ *to* (2) *and the neural network solution* $u_{\theta^n}(x)$ *to* (8) *are all smooth and bounded, as well as their high order derivatives.*

Algorithm 1: The training procedure of our TL-DPINN method

Input : Target evolutionary PDE (2); initial network u_θ; end time T
Output: The predicted model $u_{\theta^n}(x)$ at each timestamp t_n

```
Set hyper-parameters: timestamps number N_t,
  number of maximum training iterations M_0, M_1,
  learning rate η, threshold value ϵ ;
Step (a): learn u_{θ^0}(x) by neural network ;
for i = 1, 2, ..., M_0 do
    Compute the mean square error loss L^0(θ^0);
    Update the parameter θ^0 via gradient descent
      θ^0_{i+1} = θ^0_i − η∇L^0(θ^0_i) ;
Step (b): denote θ^0_* = θ^0_{M_0} and learn
  u_{θ^1}(x), ..., u_{θ^n}(x), ... sequentially by transfer
  learning ;
for n = 0, 1, 2, ..., N_t − 1 do
    Set θ^{n+1}_0 = θ^n_* ;
    for i = 1, 2, ..., M_1 do
        Compute loss L^{n+1}_i(θ^{n+1}_i) by (8) ;
        Update the parameter θ^{n+1} via gradient
          descent θ^{n+1}_{i+1} = θ^{n+1}_i − η∇L^{n+1}(θ^{n+1}_i) ;
        if |L^{n+1}(θ^{n+1}_{i+1}) − L^{n+1}(θ^{n+1}_i)| < ϵ then
            denote θ^{n+1}_* = θ^{n+1}_i and break ;
Return the optimized neural network parameters
  θ^1_*, θ^2_*, ..., θ^{N_t}_*.
```

The first assumption is to guarantee that the solution is not increasing over time. Consider the L^2 norm $\|u(t,\cdot)\|^2 = \int_\Omega u(t,x)^2 dx$, we multiply (2) by u and integrate in x to get $\frac{1}{2}\frac{d}{dt}\|u\|^2(t) = \int_\Omega u\cdot\mathcal{N}u dx \le 0$, so $\|u(t,\cdot)\| \le \|u_0\|$ for all $t > 0$. For the simplest Heat equation with $\mathcal{N}(u) = u_{xx}$, it is easy to verify that $\int_\Omega u\cdot\mathcal{N}(u)dx = -\int_\Omega |u_x|^2 dx \le 0$, satisfying Assumption 1.

The second assumption can be verified by the standard regularity estimate result of PDEs [Evans, 2022], and we omit it here for brevity.

Denote the symbol $\tau = \Delta t$ and $t_n = n\tau$, we show that the error can be strictly controlled by the time step τ, the training loss value $\mathcal{L}^n$ and the collocation points number N_r.

Theorem 1. *With the assumptions* (1) *and* (2) *hold, then the error between the solution* $u(t_n,x)$ *to* (2) *and the neural network solution* $u_{\theta^n}(x)$ *to* (8)*, i.e.,* $e^n(x) = u(t_n,x) - u_{\theta^n}(x)$, *can be estimated in the* L^2 *norm by*

$$\|e^n\| \le C\sqrt{1+t_n}(\tau^2 + \max_{1\le i\le n}\sqrt{\mathcal{L}^i} + N_r^{\frac{1}{4}}), \quad n = 1, ..., N_t, \tag{9}$$

where C *is a bounded constant depend on* $u(t_n,x)$ *and* $u_{\theta^n}(x)$.

The proof of Theorem 1 can be found in Appendix A.3 of [Chen *et al.*, 2023].

Method	RD Eq. L^2 error	RD Eq. time	AC Eq. L^2 error	AC Eq. time	KS Eq. L^2 error	KS Eq. time	NS Eq. L^2 error	NS Eq. time
Original PINN	4.17e-02	1397	8.23e-01	1412	1.00e+00	-	1.32e+00	-
Adaptive sampling	1.65e-02	1561	8.64e-03	1460	9.98e-01	6901	8.45e-01	25385
Self-attention	1.14e-02	1450	1.05e-01	1770	8.22e-01	5415	9.28e-01	21296
Time marching	3.98e-03	3215	2.01e-02	3715	8.02e-01	5527	8.85e-01	26200
Causal PINN	3.99e-05	7358	1.66e-03	9264	4.16e-02	22029	4.73e-02	5 days
TL-DPINN1 (ours)	**1.82e-05**	1463	**5.92e-04**	2328	**7.17e-03**	**5050**	**3.44e-02**	**12440**
TL-DPINN2 (ours)	9.34e-05	**748**	9.82e-04	**1100**	3.55e-02	5171	3.66e-02	56875

Table 1: A comparison of the relative L^2 error and training time (seconds) for different PDEs.

Equations	Depth	Width	Features M	N_t	N_r	Max epochs (M_0,M_1)	ϵ
RD Eq.	4	128	10	200	512	(10000,1000)	1e-9
AC Eq.	4	128	10	200	512	(10000,2000)	1e-10
KS Eq.	3	256	5	250	500	(10000,3000)	1e-8
NS Eq.	4	128	5	100	100	(10000,5000)	1e-5

Table 2: Detailed experimental settings of Section 5.

5 Computational Results

This section compares the accuracy and training efficiency of the TL-DPINN approach to existing PINN methods using various key evolutionary PDEs, including the Reaction-Diffusion (RD) equation, Allen-Cahn (AC) equation, Kuramoto–Sivashinsky (KS) equation, Navier-Stokes (NS) equation. All the code is implemented in JAX [Bradbury *et al.*, 2018], a framework that is gaining popularity in scientific computing and deep learning. In all examples, the activation function is $\tanh(\cdot)$ and the optimizer is Adam [Kingma and Ba, 2014]. The Fourier feature embedding and modified fully-connected neural networks used in [Wang *et al.*, 2022a] are discussed in Appendix A.4 of [Chen *et al.*, 2023].

Baselines. The Crank-Nicolson time differencing is denoted as TL-DPINN1, while the Gauss-Legendre time differencing is denoted as TL-DPINN2. Our study involves a comparison of these methods with several robust baselines: 1) original PINN [Raissi *et al.*, 2019]; 2) adaptive sampling [L. Wight and Zhao, 2021]; 3) self-attention [McClenny and Braga-Neto, 2023]; 4) time marching [Mattey and Ghosh, 2022] and 5) causal PINN [Wang *et al.*, 2022a] Table 1 summarizes a comparison of the relative L^2 error and running time (seconds) for different equations by different methods. We note that all neural networks are trained on an NVIDIA GeForce RTX 3080 Ti graphics card.

Error Metric. To quantify the performance of our methods, we apply a relative L^2 norm over the spatial-temporal domain:

$$\text{relative } L^2 \text{ error} = \sqrt{\frac{\sum_{n=1}^{N_t}\sum_{i=1}^{N_r}|u_{\theta^n}(x_i) - u(t_n,x_i)|^2}{\sum_{n=1}^{N_t}\sum_{i=1}^{N_r}u(t_n,x_i)^2}} \tag{10}$$

Neural Networks and Training Parameters. In all examples, the Fourier feature embedding is applied and the modified MLP is used. Adam optimizer with an initial learning rate of 0.001 and exponential rate decay is used. More details about the hyper-parameters of neural networks and the hyper-parameters of Algorithm 1 are presented in Table 2.

For the configuration of other five baselines, all of them have a neural network size with the same width and 1 deeper depth than that in Table 2. The collocation points number for all five baselines are configured to be $N_t \times N_r$ in Table 2. For example, a continuous original PINN has size $[2, 128, 128, 128, 128, 128, 1]$ and 200×512 collocation points on the space-time domain to compute the loss, then each discrete PINN has the smaller size $[1, 128, 128, 128, 128, 1]$ and much smaller collocation points 512 on space domain. The total parameters and computation of 200 discrete PINNs and the computation on the loss calculation are about the same with a single continuous PINN. In this configuration, we can sure that the comparison between our TL-DPINNs and other five baselines is fair, showing the discrete PINNs are efficient for practical applications.

5.1 Reaction-Diffusion Equation

This study begins with the Reaction-Diffusion (RD) equation, which is significant to nonlinear physics, chemistry, and developmental biology. We consider the one-dimensional Reaction-Diffusion equation with the following form:

$$\begin{cases} u_t = d_1 u_{xx} + d_2 u^2, \quad t \in [0,1], x \in [-1,1], \\ u(0,x) = \sin(2\pi x)(1+\cos(2\pi x)), \\ u(t,-1) = u(t,1) = 0, \end{cases} \tag{11}$$

where $d_1 = d_2 = 0.01$. The solution changes slowly over time, and Table 1 demonstrates that all methods succeed with small relative L^2 norm error in this instance. Our methods enhance accuracy by 2˜3 orders of magnitude compared to other PINN frameworks [Raissi *et al.*, 2019; L. Wight and Zhao, 2021; McClenny and Braga-Neto, 2023; Mattey and Ghosh, 2022] even with less training time. We see that our method TL-DPINN1 is more accurate than causal PINN [Wang *et al.*, 2022a] with much less computational time. We acknowledge that our methods TL-DPINN2 may be slightly less accurate than causal PINN, but the training time is only nearly $1/10$ of their method. In fact, the Causal PINN method can only achieve a relative L^2 error of 1.13e-01 if we stop early at the training time of our methods (748 seconds). Figure 2 shows the predicted solution against the reference solution, and our proposed method achieves a relative L^2 error of 1.82e-05. More computational results of the RD equation are provided in Appendix A.5 of [Chen *et al.*, 2023].

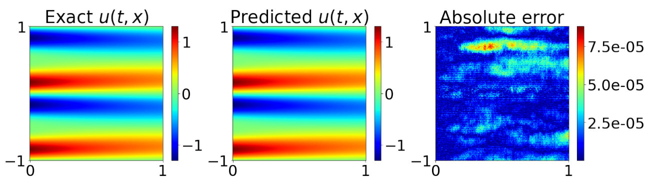


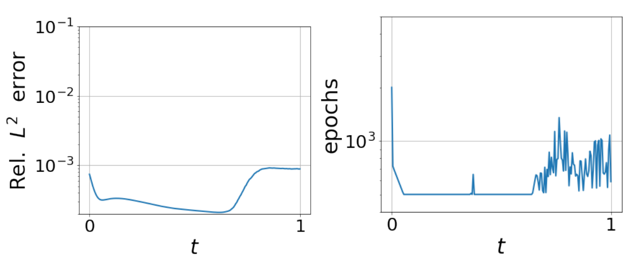
Figure 2: Comparison between the reference and predicted solutions for the Reaction-Diffusion equation, and the L^2 error is $1.82e-05$.

5.2 Allen-Cahn Equation

We consider the one-dimensional Allen-Cahn (AC) equation, a benchmark problem for PINN training [L. Wight and Zhao, 2021; Mattey and Ghosh, 2022; Wang *et al.*, 2022a]:

$$\begin{cases} u_t = \gamma_1 u_{xx} + \gamma_2 u(1-u^2), t \in [0,1], x \in [-1,1], \\ u(x,0) = u_0(x), \\ u(t,-1) = u(t,1), \quad u_x(t,-1) = u_x(t,1). \end{cases} \tag{12}$$

where $\gamma_1 = 0.0001$, $\gamma_2 = 5$ and $u_0(x) = x^2 \cos(\pi x)$. For the original PINN, the Allen-Cahn equation is hard to solve, but our approach performs well in accuracy and training efficiency. Figure 1 compares the predicted solution to the reference solution. Our technique achieves a relative L^2 error of 5.92e-04. Figure 3 shows how the L^2 error evolves and how many training epochs are needed at different timestamps. The L^2 error increases as the AC equation develops more complicated. Each timestamp's training epoch is small across the time domain, reducing training time. More computational results of the AC equation are provided in [Chen *et al.*, 2023].



(a) Relative L^2 errors (b) Training epochs

Figure 3: Training results for the Allen-Cahn equation.

5.3 Kuramoto–Sivashinsky Equation

The Kuramoto-Sivashinsky (KS) equation is used to model the diffusive–thermal instabilities in a laminar flame front. Existing PINN frameworks are challenging to solve the KS equation as the solution exhibits fast transit and chaotic behaviors [Raissi, 2018]. The KS equation takes the form

$$\begin{cases} u_t + \alpha u u_x + \beta u_{xx} + \gamma u_{xxxx} = 0, \\ u(0,x) = u_0(x), \end{cases} \tag{13}$$

with periodic boundary conditions. Here $\alpha = 5$, $\beta = 0.5$, $\gamma = 0.005$, and the initial condition $u_0(x) = -\sin(\pi x)$. Figure 4 visualizes the predicted solution against the reference solution, and our proposed method achieves a relative L^2 error of 7.17e-03. From $t = 0.4$, the reference solution begins to quickly transition, and our method is able to capture this feature. More computational results of the KS equation are provided in Appendix A.7 of [Chen *et al.*, 2023].

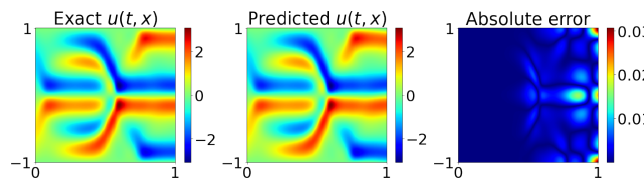


Figure 4: Comparison between the reference and predicted solutions for the Kuramoto–Sivashinsky equation, and the L^2 error is $7.17e-03$.

5.4 Navier-Stokes Equation

We consider the 2D Navier-Stokes (NS) equation in the velocity-vorticity form [Wang *et al.*, 2022a]

$$\begin{cases} w_t + \boldsymbol{u} \cdot \nabla w = \frac{1}{\mathrm{Re}} \Delta w, \quad \text{in } [0, \mathrm{T}] \times \Omega, \\ \nabla \cdot \boldsymbol{u} = 0, \quad \text{in } [0, \mathrm{T}] \times \Omega, \\ w(0,x,y) = w_0(x,y), \quad \text{in } \Omega. \end{cases} \tag{14}$$

with periodic boundary conditions. Here, $\mathbf{u} = (u, v)$ represents the flow velocity field, $w = \nabla \times u$ represents the vorticity, and Re is the Reynolds number. In addition, Ω is set to $[0, 2\pi]^2$ and Re is set to 100. Figure 5 presents the predicted solution of $w(t, x, y)$ compared to the reference solution. Our proposed method can obtain a result similar to that in [Wang *et al.*, 2022a], while the training time is only 1/58 of their method. More computational results of the NS equation are provided in Appendix A.8 of [Chen *et al.*, 2023].

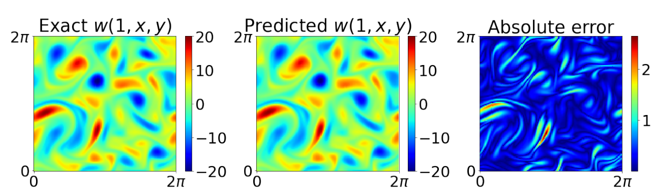


Figure 5: Comparison between the reference and predicted solutions of $w(t, x, y)$ for the Navier-Stokes equation at $t = 1.0$, and the L^2 error is $3.44e-02$.

Method	RD Eq.		AC Eq.	
	L^2 error	time	L^2 error	time
Causal PINN	3.73e-05 ± 4.66e-06	7207 ± 219	1.51e-03 ± 2.12e-04	9060 ± 341
TL-DPINN1	**1.76e-05 ± 1.06e-06**	1463 ± 53	**6.08e-04 ± 3.06e-05**	2328 ± 89
TL-DPINN2	9.89e-05 ± 8.94e-06	811 ± 122	9.29e-04 ± 8.06e-05	1291 ± 178

Table 3: Repeated test.

Method	Training efficiency (epochs/sec.)			
	Reaction-Diffusion	Allen-Cahn	Kuramoto-Sivashinsky	Navier-Stokes
Casual PINN	61.70	52.33	26.24	2.77
TL-DPINN1	**439.37**	**384.47**	**259.20**	**8.32**
TL-DPINN2	276.40	239.52	127.55	6.37

Table 4: A comparison of training efficiency for different equations.

5.5 Ablation Study

We conduct ablation studies on the relatively simpler RD Eq. and AC Eq. to ablate the main designs in our algorithm.

Time Differencing Scheme Study. Numerous time differencing schemes have been developed in the last decades. We list some commonly used schemes in [Chen *et al.*, 2023]. We do experiments on different time differencing schemes to validate that implicit time differencing schemes (2nd Crank-Nicolson or 4th Gauss-Legendre) are more stable and lead to better performance. The results are given in Table 5.

Method	RD Eq.		AC Eq.	
	L^2 error	time	L^2 error	time
Forward Euler	1.32e-03	208	9.57e-03	304
Backward Euler	2.74e-03	206	1.64e-02	444
2nd RK	1.97e-03	761	1.17e-03	1054
4th RK	2.11e-03	1187	1.31e-03	1779
TL-DPINN1	**1.82e-05**	1463	**5.92e-04**	2328
TL-DPINN2	9.34e-05	748	9.82e-04	1100

Table 5: Time differencing scheme study.

Transfer Learning Study. To see weather the transfer learning part is effective, we do ablation studies without using transfer learning. Besides, we also do experiments to fine tune the last 1/2/3 layers of the network. The results are given in Table 6. We can see that transfer learning is effective both in the efficiency and accuracy of our method.

Repeated Test. To further demonstrate the well-performance of our TL-DPINN method through accuracy and efficiency, we do 5 random runs for RD and AC Eq. by casual PINN and our method for comparison. The results are given in Table 3.

5.6 Training Efficiency

Table 4 illustrates how the computation efficiency is affected by different time discretization methods on different equations. In addition, the casual PINN method is also compared.

Method	RD Eq.		AC Eq.	
	L^2 error	time	L^2 error	time
Without TL	4.01e-04	5880	1.35e-02	9170
last layer	3.31e-04	638	1.01e-02	3624
last 2 layers	3.22e-04	221	1.01e-02	4029
last 3 layers	4.08e-04	232	1.01e-02	4685
TL-DPINN1	**1.82e-05**	1463	**5.92e-04**	2328
TL-DPINN2	9.34e-05	748	9.82e-04	1100

Table 6: Transfer learning study.

All neural networks are trained on an NVIDIA GeForce RTX 3080 Ti graphics card. We note that the total training epochs of our methods are not fixed due to the stopping criterion (see Algorithm 1). The training efficiency in Table 4 is consistent with the training time in Table 1.

6 Conclusion

In this paper, we propose a method for solving evolutionary partial differential equations via causality-enhanced discrete physics-informed neural networks with transfer learning accelerating (TL-DPINN). The discrete PINNs were thought to be time-consuming and seldom applied in the PINNs literature. We contribute to the PINN community by rediscovering the well performance of the discrete PINNs applied to solve evolutionary PDEs, both theoretically and numerically. Our method first employs a classical numerical implicit time differencing scheme to produce a series of stable propagation equations in space, and then applies PINN approximation to sequentially solve. Transfer learning is used to reduce computational costs while maintaining precision. We demonstrate the convergence property, accuracy, and computational effectiveness of our TL-DPINN method both theoretically and numerically. Our proposed method achieves state-of-the-art results among different PINN frameworks while significantly reducing the computational cost.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.62106103, No.62222605), the National Science and Technology Major Project (2020AAA0107000), the Natural Science Foundation of Jiangsu Province of China (BK20222012, BK20211517), and the project "Mathematical Models and Methods for Few-shot Learning". This work is also partially supported by High Performance Computing Platform of Nanjing University of Aeronautics and Astronautics. The first author would like to thank Prof. Zhongyi Huang for fruitful discussions on this work.

References

[Bradbury *et al.*, 2018] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[Chen *et al.*, 2021] Xinhai Chen, Chunye Gong, Qian Wan, Liang Deng, Yunbo Wan, Yang Liu, Bo Chen, and Jie Liu. Transfer learning for deep neural network-based partial differential equations solving. *Advances in Aerodynamics*, 3(1):1–14, 2021.

[Chen *et al.*, 2023] Siqi Chen, Bin Shan, and Ye Li. Efficient discrete physics-informed neural networks for addressing evolutionary partial differential equations. *arXiv preprint arXiv:2312.14608*, 2023.

[Dafermos and Pokorny, 2008] Constantine M Dafermos and Milan Pokorny. *Handbook of differential equations: evolutionary equations*. Elsevier, 2008.

[Desai *et al.*, 2022] Shaan Desai, Marios Mattheakis, Hayden Joy, Pavlos Protopapas, and Stephen J Roberts. One-shot transfer learning of physics-informed neural networks. In *ICML 2022 2nd AI for Science Workshop*, 2022.

[Evans, 2022] Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Society, 2022.

[Goswami *et al.*, 2020] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[Goswami *et al.*, 2022] Somdatta Goswami, Katiana Kontolati, Michael D Shields, and George Em Karniadakis. Deep transfer operator learning for partial differential equations under conditional shift. *Nature Machine Intelligence*, pages 1–10, 2022.

[Han *et al.*, 2018] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[Hou *et al.*, 2022] Zhengxiong Hou, Hong Shen, Xingshe Zhou, Jianhua Gu, Yunlan Wang, and Tianhai Zhao. Prediction of job characteristics for intelligent resource allocation in hpc systems: a survey and future directions. *Frontiers of Computer Science*, 16(5):165107, 2022.

[Jagtap and Karniadakis, 2021] Ameya D Jagtap and George E Karniadakis. Extended Physics-informed Neural Networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In *AAAI Spring Symposium: MLPS*, pages 2002–2041, 2021.

[Jagtap *et al.*, 2020] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[Kharazmi *et al.*, 2021] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-VPINNs: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[Khoo *et al.*, 2021] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 32(3):421–435, 2021.

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

[Krishnapriyan *et al.*, 2021] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[L. Wight and Zhao, 2021] Colby L. Wight and Jia Zhao. Solving Allen-Cahn and Cahn-Hilliard equations using the adaptive physics informed neural networks. *Communications in Computational Physics*, 29(3):930–954, 2021.

[Liu *et al.*, 2022a] Songming Liu, Hao Zhongkai, Chengyang Ying, Hang Su, Jun Zhu, and Ze Cheng. A unified hard-constraint framework for solving geometrically complex PDEs. *Advances in Neural Information Processing Systems*, 35:20287–20299, 2022.

[Liu *et al.*, 2022b] Xu Liu, Xiaoya Zhang, Wei Peng, Weien Zhou, and Wen Yao. A novel meta-learning initialization method for physics-informed neural networks. *Neural Computing and Applications*, 34(17):14511–14534, 2022.

[Long *et al.*, 2018] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. PDE-Net: Learning PDEs from data. In *International Conference on Machine Learning*, pages 3208–3216. PMLR, 2018.

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

[Lu *et al.*, 2021b] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *SIAM Journal on Scientific Computing*, 43(6):B1105–B1132, 2021.

[Mattey and Ghosh, 2022] Revanth Mattey and Susanta Ghosh. A novel sequential method to train physics-informed neural networks for Allen Cahn and Cahn Hilliard equations. *Computer Methods in Applied Mechanics and Engineering*, 390:114474, 2022.

[McClenny and Braga-Neto, 2023] Levi D McClenny and Ulisses M Braga-Neto. Self-adaptive physics-informed neural networks. *Journal of Computational Physics*, 474:111722, 2023.

[Mishra and Molinaro, 2023] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *IMA Journal of Numerical Analysis*, 43(1):1–43, 01 2023.

[Moya and Lin, 2023] Christian Moya and Guang Lin. DAE-PINN: A physics-informed neural network model for simulating differential algebraic equations with application to power networks. *Neural Computing and Applications*, 35(5):3789–3804, 2023.

[Penwarden *et al.*, 2023] Michael Penwarden, Ameya D Jagtap, Shandian Zhe, George Em Karniadakis, and Robert M Kirby. A unified scalable framework for causal sweeping strategies for Physics-Informed Neural Networks (PINNs) and their temporal decompositions. *arXiv preprint arXiv:2302.14227*, 2023.

[Raissi *et al.*, 2017] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[Raissi *et al.*, 2019] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[Raissi, 2018] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[Sirignano and Spiliopoulos, 2018] Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

[Song and Tartakovsky, 2022] Dong H Song and Daniel M Tartakovsky. Transfer learning on multifidelity data. *Journal of Machine Learning for Modeling and Computing*, 3(1), 2022.

[Stiasny *et al.*, 2021] Jochen Stiasny, Samuel Chevalier, and Spyros Chatzivasileiadis. Learning without data: Physics-informed neural networks for fast time-domain simulation. In *2021 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, pages 438–443. IEEE, 2021.

[Sukumar and Srivastava, 2022] N Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389:114333, 2022.

[Wang *et al.*, 2021a] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[Wang *et al.*, 2021b] Sifan Wang, Hanwen Wang, and Paris Perdikaris. On the eigenvector bias of Fourier feature networks: From regression to solving multi-scale PDEs with physics-informed neural networks. *Computer Methods in Applied Mechanics and Engineering*, 384:113938, 2021.

[Wang *et al.*, 2022a] Sifan Wang, Shyam Sankaran, and Paris Perdikaris. Respecting causality is all you need for training physics-informed neural networks. *arXiv preprint arXiv:2203.07404*, 2022.

[Wang *et al.*, 2022b] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[Xu *et al.*, 2022] Wuzhe Xu, Yulong Lu, and Li Wang. Transfer learning enhanced DeepONet for long-time prediction of evolution equations. *arXiv preprint arXiv:2212.04663*, 2022.

[Xu *et al.*, 2023] Chen Xu, Ba Trung Cao, Yong Yuan, and Günther Meschke. Transfer learning based physics-informed neural networks for solving inverse problems in engineering structures under different loading scenarios. *Computer Methods in Applied Mechanics and Engineering*, 405:115852, 2023.

[Yang *et al.*, 2019] Yang Yang, Da-Wei Zhou, De-Chuan Zhan, Hui Xiong, and Yuan Jiang. Adaptive deep models for incremental learning: Considering capacity scalability and sustainability. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 74–82, 2019.

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