Temporal Weighted Loss Curriculum Learning for Neural Operators

O E Coker*[1], P K Jimack[1], A Khan[2], H Wang[3]

[1] School of Computer Science, University of Leeds, Leeds, UK, LS2 9JT
[2] School of Civil Engineering, University of Leeds, Leeds, UK, LS2 9JT
[3] AI Centre, Department of Computer Science, University College London, London, UK, WC1E 6BT

Keywords: Curriculum learning, Time-dependent PDEs

Abstract:

Neural Operators offer promise for efficient solutions to time-dependent partial differential equations, but face challenges in long-term prediction due to complex dynamics, gradient accumulation, and error propagation. To address these limitations, we propose a novel curriculum learning strategy, temporal weighted loss. This method mitigates overfitting to early dynamics by dynamically adjusting the weights applied to the loss across the temporal sequence, prioritising initial time steps during early training while gradually emphasising later steps. This approach enhances model generalisation and prediction accuracy for extended time horizons, demonstrating improved performance compared to baseline curriculum learning techniques.

1. Introduction

Neural operators offer a promising approach to address the computational cost of traditional numerical methods for solving time-dependent partial differential equations (PDEs), especially modelling fluid flow. They can learn the dynamics within the data during the training phase and predict the solution to unseen conditions during inference. It offers the potential to achieve significant speed-up over the classical approach, especially in tasks like design optimisation, which require evaluation over several design parameters. However, long-term prediction of time-dependent PDEs associated with fluid flow using neural operators is a challenging problem. Firstly, the long-term dynamics of the PDEs can be complex, where multiple temporal and spatial length scales must be captured, as well as the unsteady long-term behaviour. Secondly, training neural operators is challenging due to the gradient accumulation problem for long-rollout training that can lead to numerical errors and out-of-memory issues [1]. Third, predicting the dynamics over time with large autoregressive rollouts (i.e., the number of times the model is used for the next time step prediction) during inference can lead to the propagation of errors that cause the divergence of the predicted solution from the true solution, and this error often grows exponentially [2, 3].

To address this challenge, proposed solutions generally focus on learning bias, which modifies the training strategy to improve stability. For example, adding a noise removal process (refinement steps) to the output during training [4] to improve accuracy and injecting noise into the input to improve stability [5]. A promising approach is curriculum learning, a training strategy that trains from ”easy” to ”hard” samples. The most common curriculum learning approach slowly increases the number of autoregressive rollouts during training (AR curriculum). However, for PDEs with dynamical changes over time, the training can overfit to early dynamics, leading to poor predictions on later dynamics. Another curriculum learning strategy called the AR-TF curriculum [6] improves accuracy by gradually transitioning from autoregressive rollout for earlier (easy) time steps to teacher-forcing for later (hard) time steps. However, the teacher-forcing step adds Gaussian noise to the input function, which is not representative of the error added by the model during the autoregressive inference stage.

In this work, we investigate curriculum learning for different neural operators (FNO [7] and U-NO [8]) in the context of time-dependent PDEs and propose a more accurate training strategy, called **Temporal Weighted Loss** (TWL). It is a form of curriculum learning that can explicitly prioritise learning earlier time steps in the time sequence without neglecting the later time steps, thereby enhancing the model performance. We achieve this by assigning a weight value to the loss at each time step of the autoregressive rollout, with a weighting function controlling the value of the weights. During the early stages of training, later time steps can be assigned lower weights, effectively reducing their contribution to the overall loss. As training progresses, the weight value for the latter time steps increases, eventually leading to equal weighting (i.e., weight ≈ 1) at the end of training (figure 4.1). Our curriculum leads to better accuracy and generalisation, and we demonstrate this on a selection of 1D time-dependent PDE problems: Advection, Burgers’ and Korteweg–De Vries equations.

*Corresponding Author, O E Coker scoc@leeds.ac.uk

2. Background

In this paper, we formulate our learning by considering explicit time-stepping for a general class of time-dependent PDEs $t = [0, T]$, $x \in D$ with the boundary condition B in the following form;

$$\begin{aligned} \partial_t u(t, x) &= \mathsf{F}_\Psi\,(u(t, x)) \qquad (t, x) \in [0, t_s] \times D, \\ u(0, x) &= u_0(x) \qquad x \in D, \\ u(t, x) &= B \qquad (t, x) \in (0, t_s] \times \partial D, \end{aligned} \tag{2.1}$$

where $u : [0, t_s] \times D \subset \mathrm{R}^d$, with initial condition $u_0(x)$ at $t = 0$ and boundary conditions $u(t, x) = B$ on the boundary ∂D. F_Ψ is a spatial differential operator.

Therefore, the aim is to learn a forward-explicit/autoregressive model that takes an input function $u(t)$ and predicts the next solution at $u(t + \Delta t)$. We define a nonlinear parametric autoregressive model as F_θ with parameters θ

$$u\,(t + \Delta t) = \mathsf{F}_\theta\,(\Delta t,\, u(t)) \quad \theta \in \mathrm{R}^p, \tag{2.2}$$

where θ are the parameters of the model in R^p. The learning problem aims to find the optimum θ^*.

Training is carried out using a finite collection of time evolutions of the PDE, made up of different initial conditions, where each evolution is represented as an evenly spaced time sequence and each time sequence is a uniformly distributed spatial resolution. The error is estimated using an empirical test-train method, where the difference between ground truth and prediction is minimised using a fixed number of training samples, and the error is evaluated from test samples that have not been seen during training.

3. Related Work

Operator learning models such as Fourier Neural Operator [7] and U-shaped Neural Operator [8] have shown huge success in solving different PDEs [9]. However, there is little work on the effectiveness of these methods for autoregressive prediction. The standard architecture fails to accurately predict the long-term dynamics of complex PDEs and they generally require extensive adaptation or/and modification to improve accuracy. In general, there are three approaches to encourage stable long-term prediction of PDEs. The first is the addition of extra components or appending to existing architecture. Examples include accounting for long-term temporal dependencies using learning loss in the form of PINNs [10], Recurrent Neural Networks for temporal dependencies [11], transfer learning to enhance stability [12] and denoising techniques to improve prediction [13]. The second approach uses training strategies to encourage stability such as; adversarial attack to improve robustness [5], temporal bundling that simultaneously predicts multiple time steps to reduce the number of forward passes through the model [14], latent evolution coupled with a latent loss [15] and a push forward trick that adds noise generated by the model itself [14]. The last approach modifies existing methods with inductive biases to improve long-term prediction, such as attention mechanisms to capture important temporal dependencies [16], U-Net for accurate prediction of small-scale dynamics [17] or implicit neural network that guarantees stability at the cost of accuracy [18].

4. Temporal Weighted Loss Curriculum

We present our approach, temporal weighted loss (TWL), a curriculum learning training strategy for autoregressive rollouts of time-dependent PDEs. The core idea of temporal weighted loss is to allow the model to attend to all time steps of the data during each training phase but at varying weights. In autoregressive rollout, errors from earlier time steps accumulate and propagate to later time steps, potentially leading to an exponentially increasing error over time. Therefore, we hypothesise that for many PDEs, prioritising earlier time steps will reduce the overall propagation of errors and enhance training stability. Additionally, for problems with changing dynamics (i.e., transitioning from smooth to sharp solutions or vice versa), it is crucial to consider all dynamics during the entire training phase to prevent overfitting to early dynamics while avoiding unstable training associated with a full rollout training due to the large gradient accumulation of long-term prediction. Based on this, we propose a strategy that alleviates the limitations of small- and full-rollout training. We define our approach as a form of curriculum learning, as it prioritises easy (short-rollout or early time steps with low errors) samples before hard (long-rollout or later time steps with high errors) samples, and this definition is consistent with [6].

To achieve this during training, we control the loss contributed by each predicted time step from autoregressive rollouts. For time series data u^t with $T + 1$ time steps, where $t \in 0, 1, 2, \cdots, T$, we assign a weight w^t to the loss l^t (difference between true and prediction) at each time step, such that $w^t \in [0, 1]$,

*Corresponding Author, O E Coker scoc@leeds.ac.uk

for $t > 0$ (see figure 4.1). Therefore, the total loss is a sum of the weighted loss $\hat{l}_t$ at each time step, where the weighted loss is a product of its corresponding weight and true loss. For an autoregressive training rollout of R time steps at any $t \in T - R$, the total loss L that is backpropagated is defined as;

$$L = \sum_{r=1}^{R} \hat{l}^{t+r} = \sum_{r=1}^{R} w^{t+r} \odot l^{t+r}. \tag{4.1}$$

The value of each weight w^t is determined using a predefined weighting function W, which depends on the current and the total epoch. We chose to define the weighting function as;

$$w^t = \exp\left(-Wt^2\right), \qquad W(ep, epoch) = \exp\left(-\frac{5.0\, T^{0.277}}{epoch} ep\right), \tag{4.2}$$

where T is the total number of time steps, ep is the current epoch, and $epoch$ is the total epoch, respectively.

The first part of equation 4.2 creates a bell-shaped curve, centred on $t = 0$. The value of W determines the width of the curve, and the width increases with increasing ep. We define an empirical equation for W to control the rate of increase in width with change in T. Figure 4.1 shows a visualisation of the weighting function during training. In this case, the function assigns higher and lower weights to earlier and later time steps, respectively, in the early stages of training. In the latter stages of training, all the weights are close to one ($w^t \approx 1$).

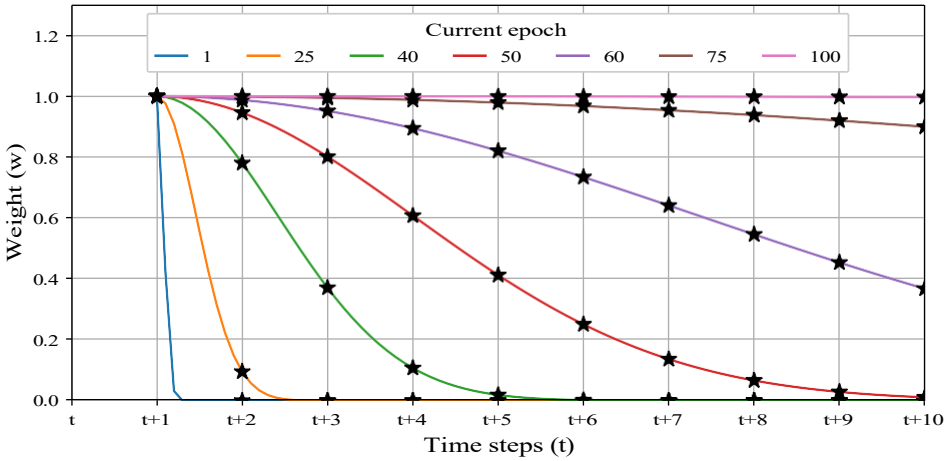


Figure 4.1: Schematic of the weighting function during training: For a Data D^t with $T + 1$ time step ($T = 10$), a weight w is assigned to each time step (black markers) for $t > 0$, which controls the loss contributed by each time step. The weighting function (curves) which determines the weight values, changes during training, and at the end of training (epoch = 100), the weighing function sets all weights to 1, i.e., $w^t \approx 1$.

4.1. Training Algorithm

A pseudocode of the training algorithm is shown in table 4.1. We train using randomly sampled batches, and the backpropagated loss is a weighted sum of all predicted time steps. The model weights are updated by *iteration* times per epoch. For the other curricula, we compared our approach to using sequential sampling (SS) (see tables E.1 and E.2 for the SS + AR and SS + AR-TF curriculum, respectively). In sequential sampling, for each epoch, the training batches are rolled out over time from the initial condition ($t = 0$) to the final time ($t = T$). Therefore, the samples within each training batch follow the same time sequence, and the backpropagated loss is composed of the losses from the entire rollout; i.e., the model weights are updated once per epoch.

*Corresponding Author, O E Coker scoc@leeds.ac.uk

Algorithm 1: Algorithm of the temporal weighted loss (TWL) Curriculum

Input: Epochs *epoch*, iteration per epoch *iteration*, temporal weights $\{w^t\}_{t=0,\cdots,T}$, training samples: $\{u^t\}_{t=0,\cdots,T}$, fixed training rollout R, model with parameters θ and loss function **criterion**.

for $ep = 1$ **to** *epoch* **do**
  Calculate w **for** $t = 0, \cdots, T$ using equation 4.2 → Weights for all time steps at current epoch.
  for *iter* = 1 **to** *iteration* **do**
    $t \in$ **RandomNumber**$(0,\ T - R)$ →Draw a random starting point btw 0 and $T - R$.
    Total loss = 0
    for r **in Range** $(0,\ R)$ **do** → Indicies of the first input timestep of each rollout.
      $\tilde{u}^{t+r+1} =$ **model** $\tilde{u}^{t+r}; \theta$
      $l^{t+r+1} =$ **criterion** $\tilde{u}^{t+r+1}, u^{t+r+1}$
      $\hat{loss} =$ **sum** $w^{t+r+1} \odot l^{t+r+1}$ → Weighted sum of loss for each predicted time steps.
      Total loss = Total loss + $\hat{loss}$
    end
    $\theta \leftarrow$ Optimiser (Total Loss)
  end
end

Table 4.1: Pseudocode: Temporal Weighted Loss Curriculum Algorithm

4.2. Empirical Analysis

In figure 4.2, we examined how different training strategies affected the model's ability to overfit to a single batch size of 32 training samples. We used the viscous Burgers (B-1) [19], which features a solution that transitions from sharp, high-amplitude dynamics to smooth, low-amplitude dynamics. We also tested a case where B-1 is flipped (B1-Reverse), i.e., the solution at $t = 0$ becomes $t = T$ and the solution at $t = T$ becomes $t = 0$. We compared four training approaches: sequential sampling (SS), where the training batch is sampled in a fixed order; random sampling (RS), where the training batch is sampled in a random order, random sampling with our curriculum (RS + TWL); and random sampling with our reversed curriculum (RS + TWL-R). TWL prioritises earlier time steps, while TWL-R prioritises later time steps during the early phase of training.

The results showed that random sampling outperformed sequential sampling. Furthermore, random sampling with our curriculum (RS + TWL) achieved the lowest training rollout error. For this specific PDE, the focus on earlier time steps (TWL) was more effective than focusing on later time steps (TWL-R) during the initial training phase. In particular, TWL consistently achieved the lowest error for both B-1 and B1-Reverse. This suggests that prioritising earlier time steps during training could significantly impact the overall performance of the model. Importantly, the order of the dynamic transition - whether from smooth to sharp or sharp to smooth — did not significantly affect the model's final performance.

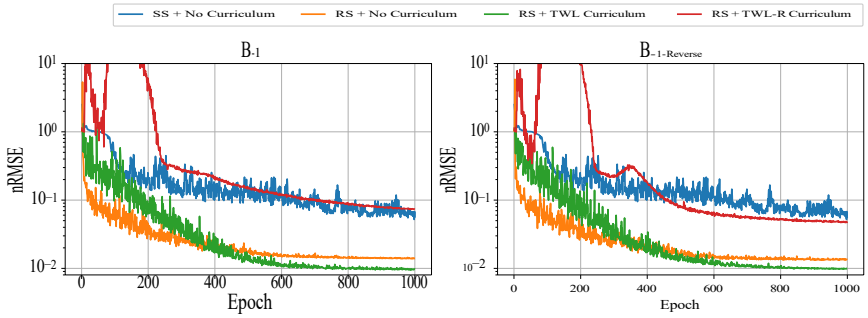


Figure 4.2: Training rollout error of different training strategies using a single training batch of size 32 for 1000 epochs. B-1 (sharp-to-smooth dynamics *left*) and B-1-Reverse (smooth-to-sharp dynamics *right*).

*Corresponding Author, O E Coker scoc@leeds.ac.uk

5. Experiments

To evaluate our proposed curriculum learning approach, we performed experiments on four one-dimensional (1D) time-dependent PDEs: Advection (A-1) and viscous Burgers (B-1) from the PDEBench dataset [19], invscid Burgers with forcing term (B-2) from [14], and Korteweg–De Vries (KdV-1) from [20]. The dynamics of B-1 and B-2 evolve over time sequences of 200 and 250 steps, respectively. Specifically, B-1 transitions from sharp, high-amplitude solutions to smooth, low-amplitude solutions, while B-2 exhibits the opposite behaviour, transitioning from smooth, low-amplitude solutions to sharp, high-amplitude solutions. The dynamics of A-1 and KdV-1 do not change over their respective time sequences of 200 and 640 time steps (first column in figure C.2). See appendix B for details on the training setup.

We compared our Temporal Weighted Loss Curriculum (RS + TWL and RS + TWL-R) with other baseline curricula using the FNO model. TWL and TWL-R prioritise earlier and later time steps in the early phase of training, respectively. Table 5.1 presents the accuracy and table 5.2 shows the generalisation performance. The results show that RS+TWL achieved the lowest normalised root mean squared error (nRMSE) for three of the four PDEs, indicating better accuracy. We compared our curriculum with: **1)** the SS+AR curriculum, which involves a purely autoregressive rollout and a curriculum that linearly varies the number of rollouts during training; **2)** the SS+TF-AR curriculum [6], a combination of teacher-forcing (TF) and autoregressive rollout (AR); and **3)** a baseline random sampling approach (RS) without any curriculum, using a fixed training rollout.

Generalisation was evaluated using the absolute difference between training and test error. Table 5.2 shows that our curriculum has better generalisation for three of the four PDEs. We observed the same trend with the U-NO model in appendix D. The comprehensive evaluation across 1D time-dependent PDEs and various baseline curricula confirms that Temporal Weighted Loss (TWL) effectively achieves competitive long-term rollout.

TRAINING STRATEGY	A-1	B-1	B-2	KdV-1
SS + AR Curriculum	0.0112	0.0104	0.2885	0.3644
SS + AR-TF Curriculum [6]	0.0080	0.0114	0.2718	0.4712
RS + No Curriculum	0.0063	0.0034	<u>0.2000</u>	**0.1411**
RS + TWL Curriculum - **ours**	**0.0036**	**0.0019**	**0.1889**	0.1472
RS + TWL-R Curriculum - **ours**	<u>0.0039</u>	<u>0.0023</u>	0.2115	<u>0.1422</u>

Table 5.1: **Accuracy**: Normalised root mean squared error (nRMSE) using Fourier Neural Operator (FNO): The numbers in **Bold** and <u>underline</u> are the **smallest** and <u>second smallest</u> error in the respective column.

TRAINING STRATEGY	A-1	B-1	B-2	KdV-1
SS + AR Curriculum	0.0004	0.0006	0.0189	0.0217
SS + AR-TF Curriculum [6]	<u>0.0003</u>	0.001	0.0081	0.0279
RS + No Curriculum	**0.0001**	0.0001	0.0006	0.0249
RS + TWL Curriculum - **ours**	0.0004	**0.00007**	<u>0.0031</u>	**0.0158**
RS + TWL-R Curriculum - **ours**	0.0004	<u>0.00008</u>	**0.0019**	<u>0.0216</u>

Table 5.2: **Generalisation**: Absolute difference between training and test error using Fourier Neural Operator (FNO): The numbers in **Bold** and <u>underline</u> are the **smallest** and <u>second smallest</u> error in the respective column.

6. Conclusion

In summary, our Temporal Weighted Loss Curriculum (TWL) demonstrated superior performance across a range of 1D time-dependent PDEs. By prioritising early time steps during training, TWL can achieve higher accuracy (lower nRMSE) and better generalisation compared to baseline curricula. These findings underscore the effectiveness of our curriculum learning approach in improving long-term rollout performance for complex PDEs. However, further work is required to explore more diverse and complex PDEs in two- and three-dimensional spaces. Also, we have yet to investigate our approach on other classes of neural networks like convolution, graph, and attention-based networks.

*Corresponding Author, O E Coker scoc@leeds.ac.uk

7. Appendix

The appendix contains supplementary materials to complement the results in section 5. We provide more details about the PDEs investigated in A and the training setup in B. FNO results in C showing the propagation of errors over time for the curricula investigated. We show results using another neural operator (U-NO) in D and present the pseudocode of the baseline curricula in E.

A. Partial Differential Equations (PDEs)

A.1. Advection Equation (A-1)

$$\begin{aligned} &\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0, \quad x \in (0,1), t \in (0,2], \\ &u(0,x) = u_0(x), \quad x \in (0,1), \end{aligned} \tag{A.1}$$

where c is a constant advection speed. The exact solution of the system is given as: $u(t,x) = u_0(x - \beta t)$. The dataset consists of time series prediction at 201 uniformly discretised points in time $t \in [0, 2]$ and 1024 uniformly discretised points in space $x \in [0, 1]$. The initial condition is generated by a superposition of sinusoidal waves;

$$u_0(x) = \sum_{k_i = k_1, \ldots, k_N} A_i \sin(k_i x + \phi_i), \tag{A.2}$$

where $k_i = 2\pi n_i / l_x$ are the wave numbers, N is the number of waves, A_i is a random number that is uniformly selected in $[0, 1]$, l_x is the domain size and ϕ_i is a randomly chosen phase change in $(0, 2\pi)$.

A.2. Burgers' Equation (B-1)

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{A.3}$$

Numerical solutions are generated using a 2nd-order upwind difference scheme and central difference scheme for the advection and diffusing term, respectively.

A.3. Burgers' Equation (B-2)

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \delta(t,x) \\ &u(0,x) = \delta(0,x), \qquad \delta(t,x) = \sum_{j=1} A_j \sin(\omega_j t + 2\pi_j x/L + \phi_j). \end{aligned} \tag{A.4}$$

Numerical groundtruth is generated using a 5th-order WENO scheme for the convection term and 4th-order finite difference stencils for the remaining terms. The temporal solver is an explicit Runge-Kutta 4 solver with adaptive timestepping.

A.4. Korteweg–De Vries Equation (KdV-1)

Korteweg–De Vries (KdV) is a nonlinear scalar-valued PDE in space and time, which exhibits solutions of solitary weakly-interacting wave-pulses.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0. \tag{A.5}$$

B. Training setup

As discussed in Sec.5, we used datasets provided by PDE Benchmark [19] for Advection (A-1) and Burgers with viscosity (B-1). Burgers without viscosity (B-2) were provided by [14], and Korteweg–De Vries (KdV-1) is from [20]. The resolution of the PDEs is provided in table B.1. We used 1024 training samples and the results are based on 128 test samples. The results were trained on a Fourier Neural Operator (FNO) [7] of 141, 258 parameters, with 4 layers, 16 modes and 32 hidden dimensions. For all models, we use a multi-timestep prediction approach, that is, all models use 10 consecutive time steps (history) to predict the next 10 consecutive time steps (future).

We use Adam optimiser[21] and an l_2 loss function for all results. Models using sequential sampling (SS) were trained for 300 epochs at 9 rollouts per epoch, while models using Random Sampling (RS) were trained for 50 epochs at 180 iterations per epoch and 3 rollouts per iteration. This gives a total training rollout of 2700 for both samplings, ensuring a fair comparison. The learning rate was set as 3×10^{-3}, and uses a StepLR scheduler that decays the learning rate by 0.9 for every 2 step-size epoch. The mini-batch size we used was 32 for all results. The training was performed on NVIDIA L40S 48GB GPU.

Dataset	Space (x)	Time (t)
A-1	256	201
B-1	256	201
B-2	200	250
KdV-1	200	640

Table B.1: Space and time resolution of datasets.

*Corresponding Author, O E Coker scoc@leeds.ac.uk

C. Fourier Neural Operator (FNO)

Figure C.1 shows the propagation of errors over time for different curricula, and our curriculum (RS+TWL) has the smallest error over time for three of the four PDEs shown. A visualisation of the difference between ground truth and prediction is shown in figure C.2, where the dynamics of the four PDEs are in the first column.

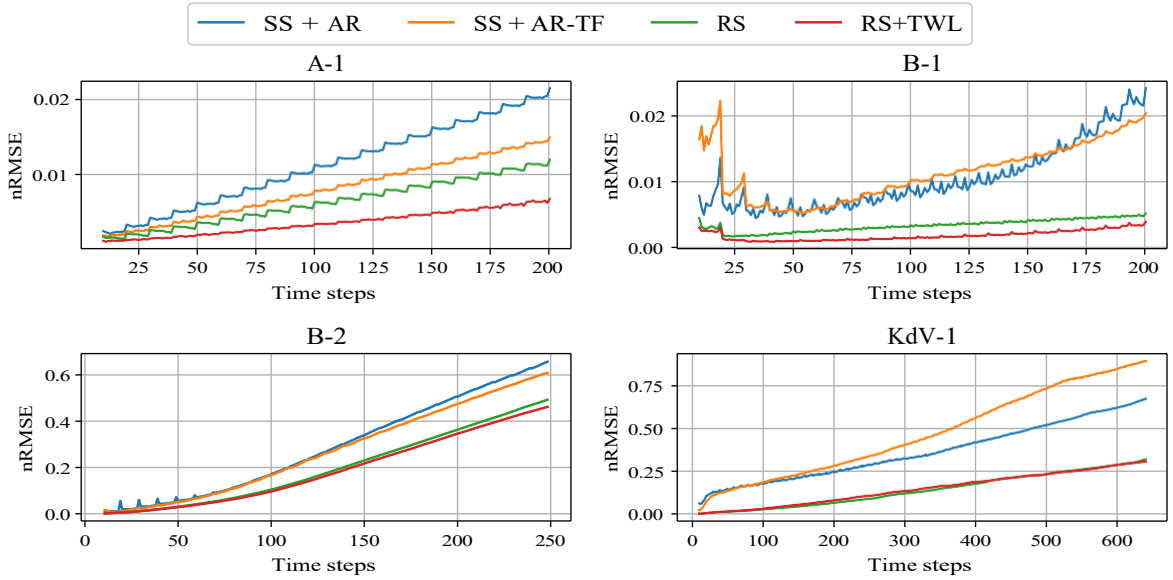
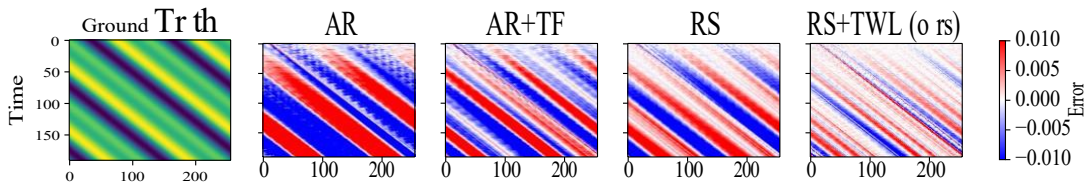
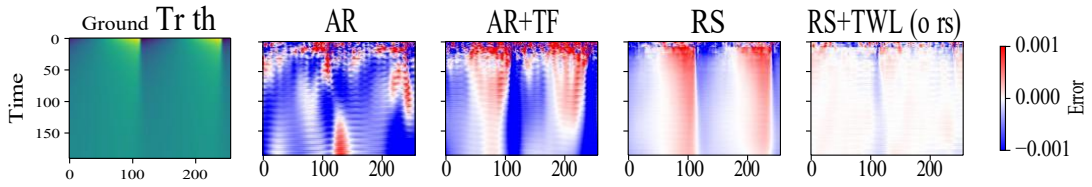


Figure C.1: Time evolution of the nRMSE for all four training strategies and equations: Advection A-1 (top-left), Burgers' equation with viscosity N-1 (top-right), Burgers' equation without viscosity B-2 (bottom-left) and Korteweg–De Vries KdV-1 (bottom-right).

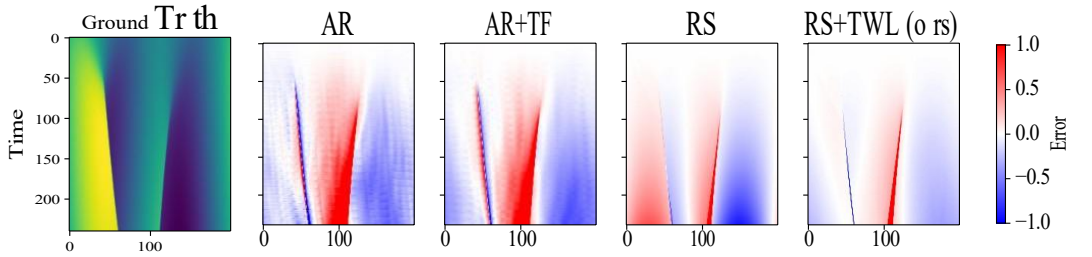
*Corresponding Author, O E Coker scoc@leeds.ac.uk



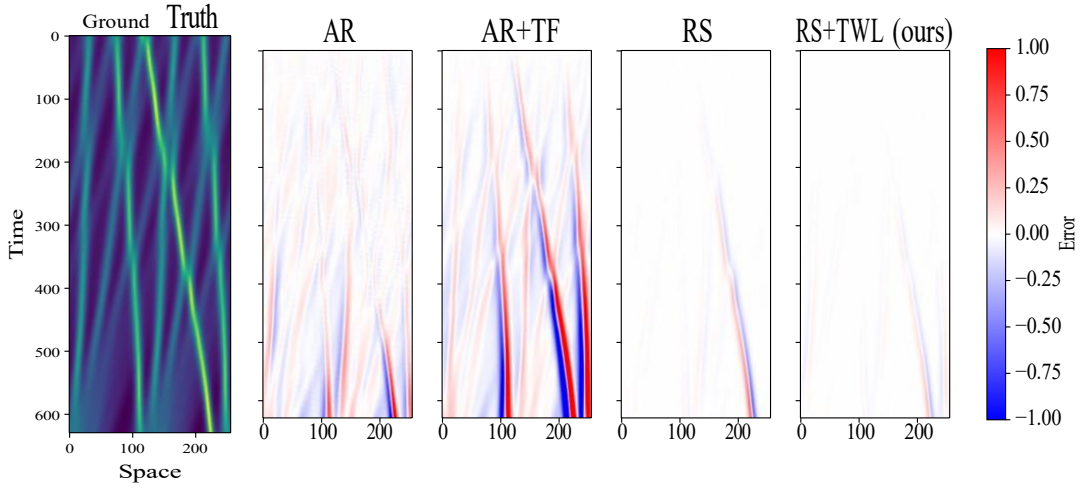
(a) Advection Equation (A-1)



(b) Viscous Burgers' Equation (B-1)



(c) Inviscid Burgers' Equation (B-2)



(d) Korteweg–De Vries (KdV-1)

Figure C.2: A visualisation of samples of the 1D PDEs are shown in the first column (time: y-axis, space: x-axis). The remaining columns show the error (truth-prediction) for all four training strategies investigated. RS+TWL outperforms other strategies with the lowest errors for all PDEs.

D. U-shaped Neural Operator (U-NO)

The U-shaped Neural Operator (U-NO) model [8] is based on 7 layers with 16 hidden dimensions (395, 738 parameters). As presented in tables D.1 and D.2, which show accuracy and generalisation performance respectively, our proposed curriculum learning strategy achieved the lowest error for the PDEs with changing dynamics (B-1 and B-2). These results strongly support our hypothesis regarding the effectiveness of our approach for such PDEs. Also, we achieved the lowest generalisation error for three out of four PDEs.

*Corresponding Author, O E Coker scoc@leeds.ac.uk

TRAINING STRATEGY	A-1	B-1	B-2	KdV-1
SS + AR Curriculum	-	-	-	-
SS + AR-TF Curriculum [6]	-	-	-	-
RS + No Curriculum	**0.0062**	0.0059	0.3646	**0.3160**
RS + TWL Curriculum - **ours**	<u>0.0076</u>	**0.0019**	<u>0.3580</u>	<u>0.3326</u>
RS + TWL-R Curriculum - **ours**	0.0085	<u>0.0144</u>	**0.2582**	0.3635

Table D.1: **Accuracy**: Normalised root mean squared error (nRMSE): The numbers in **Bold** and <u>underline</u> are the **smallest** and <u>second smallest</u> error in the respective column.

TRAINING STRATEGY	A-1	B-1	B-2	KdV-1
SS + AR Curriculum	-	-	-	-
SS + AR-TF Curriculum [6]	-	-	-	-
RS + No Curriculum	**0.0014**	<u>0.0003</u>	0.2727	0.0304
RS + TWL Curriculum - **ours**	<u>0.0027</u>	**0.0001**	<u>0.2861</u>	<u>0.0104</u>
RS + TWL-R Curriculum - **ours**	0.0027	<u>0.0003</u>	**0.0702**	**0.0040**

Table D.2: **Generalisation**: Absolute difference between training and test error using Fourier Neural Operator (FNO): The numbers in **Bold** and <u>underline</u> are the **smallest** and <u>second smallest</u> error in the respective column.

E. Training Algorithms

The pseudocode of baseline training strategies algorithms are presented. Sequential Sampling with Autoregressive rollout (SS + AR) in table E.1, Sequential sampling with autoregressive rollout and teacher-forcing (SS + AR-TF) in table E.2, and Random sampling without curriculum in table E.3.

Algorithm 2: Algorithm of the Autoregressive Rollout (SS + AR) curriculum)

Input: total epoch *epoch*, temporal index t, final time step T, random noise ϵ, training samples: $\{u^t\}_{t=0,\dots,T}$, **model** with parameters θ and loss function **criterion**.

for $ep = 1$ **to** *epoch* **do**
  Total loss = 0
  Calculate T_{trans} following: $T_{\text{trans}} = \frac{T-1}{epoch-1} ep + (1 - \frac{T-1}{epoch-1})$
  for $t = 1$ **to** T_{trans} **do** → Sequential Sampling
    $\tilde{u}^{t+1}$ = **model** $\tilde{u}^t; \theta$ → Autoregressive rollout (AR)
    loss = **criterion** $\tilde{u}^{t+1}, u^{t+1}$
    Total loss = Total loss + loss
  end
  $\theta \leftarrow$ Optimiser (Total Loss)
end

Table E.1: Pseudocode: AR Curriculum Algorithm

*Corresponding Author, O E Coker scoc@leeds.ac.uk

Algorithm 3: Algorithm of the Autoregressive Rollout and Teacher-Forcing (SS + AR+TF curriculum)

Input: total epoch *epoch*, temporal index t, final time step T, random noise ϵ,
training samples: $\{u^i\}_{t=0,\cdots,T}$, **model** with parameters θ and loss function **criterion**.

```
for ep = 1 to epoch do
    Total loss = 0
    Calculate t_trans following Eq...
    for t = 1 to T do                          → Sequential sampling
        if t ≤ t_trans do
            ũ^{t+1} = model ũ^t; θ             → Autoregressive rollout (AR)
        else
            ũ^{t+1} = model u^t; θ             → Teacher-Forcing (TF)
        loss = criterion ũ^{t+1}, u^{t+1}
        Total loss = Total loss + loss
    end
    θ ← Optimiser (Total Loss)
end
```

Table E.2: Pseudocode: AR+TF Curriculum Algorithm

Algorithm 4: Algorithm of the Random sampling with fixed rollout (RS)

Input: total epoch *epoch*, iteration per epoch *iteration*, training samples: $\{u^i\}_{t=0,\cdots,T}$,
fixed training rollout R, **model** with parameters θ and loss function **criterion**.

```
for ep = 1 to epoch do
    for iter = 1 to iteration do
        Total loss = 0
        t ∈ RandomNumber(0, T − R)        →Draw a random starting point btw 0 and T − R
        for r in Range (0, R) do          → Indicies of the first input timestep of each rollout.
            ũ^{t+r+1} = model u^{t+r}; θ
            loss = criterion ũ^{t+r+1}, u^{t+r+1}
            Total loss = Total loss + loss
        end
        θ ← Optimiser (Total Loss)
    end
end
```

Table E.3: Pseudocode: Random Sampling with fixed rollout and no curriculum Algorithm

*Corresponding Author, O E Coker scoc@leeds.ac.uk

References

[1] B. List, L.-W. Chen, K. Bali, and N. Thuerey, "Differentiability in unrolled training of neural physics simulators on transient dynamics," *Computer Methods in Applied Mechanics and Engineering*, vol. 433, p. 117441, 2025.

[2] F. Gonzalez, F.-X. Demoulin, and S. Bernard, "Towards long-term predictions of turbulence using neural operators," *arXiv preprint arXiv:2307.13517*, 2023.

[3] M. McCabe, P. Harrington, S. Subramanian, and J. Brown, "Towards stability of autoregressive neural operators," *arXiv preprint arXiv:2306.10619*, 2023.

[4] P. Lippe, B. Veeling, P. Perdikaris, R. Turner, and J. Brandstetter, "Pde-refiner: Achieving accurate long rollouts with neural pde solvers," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[5] A. Sanchez-Gonzalez, K. Stachenfeld, D. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, and P. Battaglia, "Learning general-purpose cnn-based simulators for astrophysical turbulence," in *ICLR 2021 SimDL Workshop*, 2021.

[6] M. Takamoto, F. Alesiani, and M. Niepert, "Learning neural pde solvers with parameter-guided channel attention," in *International Conference on Machine Learning*, pp. 33448–33467, PMLR, 2023.

[7] Z. Li, N. B. Kovachki, K. Azizzadenesheli, B. liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," in *International Conference on Learning Representations*, 2021.

[8] M. A. Rahman, Z. E. Ross, and K. Azizzadenesheli, "U-NO: U-shaped neural operators," *Transactions on Machine Learning Research*, 2023.

[9] L. Lu, X. Meng, S. Cai, Z. Mao, S. Goswami, Z. Zhang, and G. E. Karniadakis, "A comprehensive and fair comparison of two neural operators (with practical extensions) based on fair data," *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114778, 2022.

[10] S. Wang and P. Perdikaris, "Long-time integration of parametric evolution equations with physics-informed deeponets," *Journal of Computational Physics*, vol. 475, p. 111855, 2023.

[11] K. Michałowska, S. Goswami, G. E. Karniadakis, and S. Riemer-Sørensen, "Neural operator learning for long-time integration in dynamical systems with recurrent neural networks," in *2024 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2024.

[12] W. Xu, Y. Lu, and L. Wang, "Transfer learning enhanced deeponet for long-time prediction of evolution equations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 10629–10636, 2023.

[13] P. Lippe, B. Veeling, P. Perdikaris, R. Turner, and J. Brandstetter, "Pde-refiner: Achieving accurate long rollouts with neural pde solvers," *Advances in Neural Information Processing Systems*, vol. 36, pp. 67398–67433, 2023.

[14] J. Brandstetter, D. E. Worrall, and M. Welling, "Message passing neural PDE solvers," in *International Conference on Learning Representations*, 2022.

[15] T. Wu, T. Maruyama, and J. Leskovec, "Learning to accelerate partial differential equations via latent global evolution," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2240–2253, 2022.

[16] W. Peng, Z. Yuan, and J. Wang, "Attention-enhanced neural network models for turbulence simulation," *Physics of Fluids*, vol. 34, no. 2, 2022.

[17] Z. Li, W. Peng, Z. Yuan, and J. Wang, "Long-term predictions of turbulence by implicit u-net enhanced fourier neural operator," *Physics of Fluids*, vol. 35, no. 7, 2023.

[18] L. Migus, J. Salomon, and P. Gallinari, "Stability of implicit neural networks for long-term forecasting in dynamical systems," in *ICLR 2023 Workshop on Physics for Machine Learning*, 2023.

[19] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflu¨ger, and M. Niepert, "Pdebench: An extensive benchmark for scientific machine learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1596–1611, 2022.

[20] J. Brandstetter, M. Welling, and D. E. Worrall, "Lie point symmetry data augmentation for neural PDE solvers," in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 2241–2256, PMLR, 17–23 Jul 2022.

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

*Corresponding Author, O E Coker scoc@leeds.ac.uk