Scheduled Temporal Loss Weighting for Neural Operators

Oluwaseun E. Coker
Department of Computer Science
University of Leeds
Leeds, LS2 9JT.
scoc@leeds.ac.uk

He Wang
AI Centre, Department of Computer Science
University College London
London, WC1E 6BT.
he_wang@ucl.ac.uk

Amirul Khan
School of Civil Engineering,
University of Leeds
Leeds, LS2 9JT.
A.Khan@leeds.ac.uk

Peter K. Jimack
Department of Computer Science
University of Leeds
Leeds, LS2 9JT.
P.K.Jimack@leeds.ac.uk

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, scheduled temporal loss weighting. 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. This approach enhances model generalisation and prediction accuracy, demonstrating improved performance compared to baseline techniques.

1 Introduction

Neural operators are a promising deep learning architecture for solving time-dependent partial differential equations (PDEs), offering significant speed-up over traditional methods, particularly in applications like design optimisation [11, 8]. They learn the dynamics from data during training and can then predict the solution to new conditions during inference. However, long-term prediction of time-dependent PDEs with neural operators presents several challenges. Firstly, long-term PDE dynamics can be complex, involving multiple temporal and spatial scales and unsteady behaviour that must be captured by the model. Secondly, training neural operators for long-rollout predictions is difficult due to the accumulation of gradients during the backpropagation step, which can lead to numerical errors and out-of-memory issues [7]. Thirdly, predicting the dynamics over time with large autoregressive rollouts (i.e., the number of times the model is called 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 [4, 9].

To address these challenges of long-term prediction, solutions often focus on learning bias to enhance training stability. For instance, techniques involve injecting noise into the input to improve the model's robustness to noise or removing noise from the output during training to improve accuracy [6, 12]. A promising approach is curriculum learning, which trains a model from "easy" to "hard" data samples, imitating the meaningful learning order in human curricula. The most common curriculum learning approach gradually increases the number of autoregressive rollouts during training (AR-CL). However, for PDEs with changing dynamics over time, this can cause training overfitting to early dynamics, leading to poor performance on later dynamics. Other strategies that transition between autoregressive rollout (AR) and teacher-forcing (TF) with Gaussian noise are also not ideal, as the noise is not representative of the model's actual error during inference[16].

In this work, we investigate curriculum learning for neural operators (in particular, Fourier Neural Operator [5]) in the context of time-dependent PDEs and propose a more accurate training strategy, called **Scheduled Temporal Loss Weighting** (STLW). It is a form of curriculum learning that explicitly prioritise learning the earlier (easy) time steps in the time sequence without neglecting the later (hard) time steps, thereby enhancing model performance. We achieve this during training by assigning a weight value to the loss at each time step during autoregressive rollout (AR), with a weighting function controlling the value of the weights. During the early stages of training, later (hard) time steps can be assigned lower weights, effectively reducing their contribution to the overall loss. As training progresses, the weight value for the later time steps increases, eventually leading to equal weighting (i.e., weight ≈ 1) at the end of training. Our curriculum leads to improved accuracy and stability, and we demonstrate this on a selection of 1D time-dependent PDE problems, including the advection, Burgers', and Korteweg–De Vries equations.

2 Background

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

$$\partial_t u(t,x) = \mathcal{F}^{\dagger}\left(u(t,x)\right) \qquad (t,x) \in (0,T] \times D \tag{1}$$

where $u : (0,T] \times D \subset \mathbb{R}^d$, with initial condition $u_0(x)$ at $t = 0$ and boundary conditions $u(t,x) = B$ on the boundary ∂D. $\mathcal{F}^{\dagger}$ 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 $\mathcal{F}_\theta$ with parameters θ

$$u\left(t + \Delta t\right) = \mathcal{F}_\theta\left(\Delta t, u(t)\right) \quad \theta \in \mathbb{R}^p, \tag{2}$$

where θ are the parameters of the model in $\mathbb{R}^p$.

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 has 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

Bengio et al. [1] first proposed curriculum learning (CL) as a method to train neural networks, hypothesising that it affects the speed of convergence and generalisation. This approach trains a model from "easy" to "hard" data, imitating the meaningful learning order in human curricula. CL is formulated by defining a "curriculum metric" and "scheduler". The metric decides the relative ease of the data sample, while the scheduler controls how the data is processed during training (see Appendix A.2.2 and A.2.1 for more details on curriculum learning). CL is broadly classified into predefined and automatic. The predefined approach relies on prior knowledge to design the metric and scheduler, which is the focus of this work. Most prior knowledge is generally based on the estimation of complexity, diversity, and noise.

Limited research is available on applying curriculum learning for PDEs, and most studies are based on the transition between baseline training strategies, i.e., easy teacher-forcing (TF) and hard autoregressive rollout (AR) or vice-versa, which we denote as TF-AR-CL or AR-TF-CL, respectively. Teutsch and Mäder [15] investigated different CL schedulers between autoregressive rollout and teacher forcing for a time sequence using a recurrent neural network. Their work demonstrated that training with CL yields better outcomes than without CL; however, obtaining the optimal scheduler is not trivial and can require an expensive grid search. Vlachas and Koumoutsakos [16] further demonstrated the benefits of CL for longer rollouts, effectively reducing error propagation over time. Takamoto et al. [14] used curriculum learning during training, proposing a new scheduler that reduced errors during autoregressive inference. Most of these works use teacher-forcing despite it being suboptimal for long-term prediction. In our work, we avoid teacher forcing and formulate our curriculum approach solely on autoregressive rollout.

4 Scheduled Temporal Loss Weighting (STLW)

We present our approach, scheduled temporal loss weighting (STLW), a curriculum learning training strategy for long-term autoregressive rollouts of time-dependent PDEs. Our core idea is to allow the model to attend to all time steps of the data during each training phase, but at different strengths. In autoregressive inference, errors from earlier time steps accumulate and propagate to later time steps, potentially leading to an exponentially increasing error over time (see derivation in Appendix A.1.1). For PDEs with changing dynamics over long time; for example, transitioning from smooth to sharp solutions or vice versa, we hypothesise that giving higher priority to earlier timesteps will reduce the overall propagation of errors and enhance training stability, however, it is crucial to attend to all dynamics during the entire training phase to avoid overfitting to early dynamics while avoiding unstable training associated with a long rollout training due to the large gradient accumulation. Based on this, we propose a strategy that alleviates the limitations of both short- and long-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.

4.1 Implementation

To achieve our training strategy, we control the loss contributed by each predicted time step during autoregressive rollouts. For time series data with T time steps and $t \in 0, 1, 2, \cdots, T-1$, we assign a weight w to the loss l (difference between true and prediction) at each time step, such that $w \in [0, 1]$, for $t > 0$. In Figure 5, we show a schematic of our approach, where the colour intensity at each time step (circle) represents the weight value, and as the training progresses, the colour intensity of the later time step increases, i.e., the weight value increases. Therefore, the total loss that is optimised (backpropagated) is the sum of the weighted loss $\hat{l}$ at each time step, where the weighted loss is a product of its corresponding weight and true loss (see Appendix A.4.2 for training algorithm pseudocode). In summary, for each training epoch e, with an autoregressive training rollout of $\widehat{T}$, where $\widehat{T} \ll T$, for any $t \in T - \widehat{T}$, the total weighted loss $\widehat{L}$ that is backpropagated is;

$$\widehat{L}_t = \sum_{\hat{t}=1}^{\widehat{T}} \hat{l}_{t+\hat{t}} = \sum_{\hat{t}=1}^{\widehat{T}} w_{e,\,t+\hat{t}} \cdot l(u_{t+\hat{t}} - \tilde{u}_{t+\hat{t}}), \qquad \hat{t} \in 0, 1, \cdots, \widehat{T}. \tag{3}$$

We apply a weight to each timestep, and we can define the curriculum metric $\bar{w}$ as the mean of all weights w. The mean weight at each epoch e is defined as; $\bar{w}_e = \frac{1}{T} + \left(\sum_{t=1}^{T} w_{e,\,t}\right)$. The easiest task ($\bar{w}_e = 1/T$) is when all weights are fully damped ($w_{e,t} = 0$), while the most difficult task ($\bar{w}_e = 1$) is when the weights are fully undamped ($w_{e,t} = 1$). Therefore, we can define a scheduler that transitions from $\bar{w} = 1/T$ to $\bar{w} = 1$. For the Scheduler, we define a smooth exponential function that controls the weights by initially dampening the later time steps, slowly increasing the weights with increasing epoch (see more details in Appendix A.4.1).

5 Experiment and Result

Table 1 shows the mean normalised Root Mean Squared Error (nRMSE) for four PDE's datasets: Advection (Av) and viscous Burgers (vB) [13], inviscid Burgers with forcing term (iB) [3], and Korteweg–De Vries (KdV) [2]. We compare our approach, scheduled temporal loss weighting (AR-STLW-CL) curriculum, against baseline training strategies (teacher forcing (TF), teacher forcing with noise (TF+N), and fixed auto-regressive rollout (AR)) and existing curriculum learning strategies (CL) methods; deterministic (D) (TF-AR-D-CL, AR-TF-D-CL and AR-CL), and probabilistic (P) variants (TF-AR-P-CL and AR-TF-P-CL). For each dataset, the best (lowest nRMSE) strategy is highlighted in bold and the second-best is underlined, respectively.

We first examine the baseline strategies: TF, TF+N and AR. Our findings show that AR consistency outperforms TF across three out of four datasets, i.e., Av, v-B and KdV, while TF outperforms AR on the i-B dataset. Therefore, we use the best baseline for each dataset to obtain the relative improvement of the curriculum strategies in the "rel. Impro." column. All curriculum strategies, with one exception, improved upon the best baseline strategy. Across all datasets and metrics, AR-STLW-CL emerged as the best-performing curriculum, followed by AR-CL as the second-best. Specifically, AR-STLW-CL achieved the lowest error (highest relative improvement) for three of the four datasets v-B, i-B and KdV, with a relative improvement of 62.5%, 12.0% and 40.3%, respectively. AR-CL has the second-lowest error for three datasets (Av, i-B and KdV), with relative improvement of 16.8%, 11.4% and 36.9%, respectively.

	STRATEGY	**Best Curriculum**	**nRMSE**		
		$\alpha_{start} \rightarrow \alpha_{end}$	$mean^{\pm s.d} \downarrow$	rel. impro. $\uparrow$	last 10% $\downarrow$
Advection (Av)	TF	0.0	$0.0375^{\pm 1.7e-2}$	-	0.0883
	TF+N	0.0	$0.0165^{\pm 5.6e-3}$	-	0.0446
	AR	1.0	$0.00353^{\pm 8.7e-5}$	-	0.00536
	AR-CL	1.0	$\underline{0.00293}^{\pm \underline{8.9e-5}}$	16.8%	$\underline{0.00450}$
	TF-AR-D-CL	$0.0 \nearrow 1.0$	$\mathbf{0.00272}^{\pm 9.5e-5}$	**22.6%**	**0.00423**
	TF+N-AR-D-CL	$0.0 \nearrow 1.0$	$0.00296^{\pm 1.4e-4}$	15.9%	0.00455
	AR-TF-P-CL	$1.0 \searrow 0.0$	$0.00538^{\pm 1.1e-3}$	-52.4%	0.0146
	TF-AR-P-CL	$0.0 \nearrow 1.0$	$0.00294^{\pm 9.4e-5}$	16.5%	0.00473
	AR-STLW-CL (**ours**)	$1.0\ ,\ A_c = 5.0$	$0.00303^{\pm \mathbf{0.0}}$	14.1%	0.00485
Viscous Burgers' (v-B)	TF	0.0	$0.0632^{\pm 4.2e-2}$	-	0.104
	TF+N	0.0	$0.0125^{\pm 2.3e-2}$	-	0.0154
	AR	1.0	$0.00749^{\pm 1.1e-3}$	-	0.00771
	AR-CL	1 .0	$0.00376^{\pm \mathbf{1.3e-4}}$	49.7%	0.00623
	TF-AR-D-CL	$0 \nearrow 1$	$\mathbf{0.00280}^{\pm 4.8e-4}$	62.5%	**0.00363**
	TF+N-AR-D-CL	$0.0 \nearrow 1.0$	$\underline{0.00291}^{\pm 3.5e-4}$	$\underline{61.1\%}$	$\underline{0.00422}$
	AR-TF-P-CL	$1.0 \searrow 0.0$	$0.0112^{\pm 8.2e-4}$	$-44.8.6\%$	0.0142
	TF-AR-P-CL	$0.0 \nearrow 1.0$	$0.00413^{\pm \underline{2.1e-4}}$	56.3%	0.00501
	AR-STLW-CL (**ours**)	$1,\ A_c = 5.0$	$\mathbf{0.00280}^{\pm 4.6e-4}$	**62.5%**	0.00566
Inviscid Burgers' (i-B)	TF	0.0	$0.274^{\pm 3.9e-3}$	-	0.625
	TF+N	0.0	$0.278^{\pm 8.1e-3}$	-	0.628
	AR	1.0	$0.306^{\pm 8.7e-3}$	-	0.683
	AR-CL	1.0	$\underline{0.243}^{\pm 3.4e-3}$	$\underline{11.4\%}$	$\underline{0.559}$
	TF-AR-D-CL	$0.0 \nearrow 1.0$	$0.252^{\pm 4.4e-3}$	8.2%	0.571
	TF+N-AR-D-CL	$0.0 \nearrow 1.0$	$0.250^{\pm 3.1e-3}$	8.7%	0.573
	AR-TF-P-CL	$1.0 \searrow 0.0$	$0.291^{\pm 1.4e-2}$	-6.1%	0.660
	TF-AR-P-CL	$0.0 \nearrow 1$	$0.266^{\pm \mathbf{1.7e-3}}$	3.0%	0.603
	AR-STLW-CL (**ours**)	$1.0,\ A_c = 5.0$	$\mathbf{0.241}^{\pm \underline{2.8e-3}}$	**12.0%**	**0.558**
Korteweg–De Vries (KdV)	TF	0.0	$0.541^{\pm 9.5e-2}$	-	0.896
	TF+N	0.0	$0.347^{\pm 1.0e-1}$	-	0.623
	AR	1.0	$0.189^{\pm 5.2e-3}$	-	0.311
	AR-CL	1.0	$\underline{0.119}^{\pm 5.2e-3}$	$\underline{36.9\%}$	$\underline{0.226}$
	TF-AR-D-CL	$0.0 \nearrow 1.0$	$0.159^{\pm \mathbf{4.1e-3}}$	15.9%	0.290
	TF+N-AR-D-CL	$0.0 \nearrow 1.0$	$0.145^{\pm \underline{4.7e-3}}$	23.3%	0.267
	AR-TF-P-CL	$1.0 \searrow 0.0$	$0.226^{\pm 2.0e-2}$	-19.1%	0.411
	TF-AR-P-CL	$0.0 \nearrow 1.0$	$0.189^{\pm 7.2e-3}$	0%	0.342
	AR-STLW-CL (**ours**)	$1.0,\ A_c = 5.0$	$\mathbf{0.113}^{\pm 5.0e-3}$	**40.3%**	**0.214**

Table 1: **Accuracy**: Normalised root mean squared error (nRMSE) using Fourier Neural Operator (FNO): The numbers in **Bold** and $\underline{\text{underline}}$ are the **smallest** and $\underline{\text{second smallest}}$ error in the respective column.

6 Discussion and Conclusion

We introduced a novel curriculum training strategy, called scheduled temporal loss weighting (AR-STLW-CL), which assigns variable weights to the loss at each time step during training. Our evaluation against eight different training strategies showed that our approach is highly effective (see Appendix for more results). We observed that AR-TF (i.e., starting with AR and finishing with TF) was the worst-performing strategy, which is not a surprise since the baseline teacher-forcing method is less effective than autoregressive rollout. This finding contrasts with the work of Teutsch and

Mäder [15], who found that increasing the amount of TF during training led to the best performance. This discrepancy likely stems from the nature of the PDEs investigated; their study focused on chaotic systems, which benefit from a teacher-forcing approach. Furthermore, our deterministic approach outperformed its probabilistic counterpart, a finding that also contrasts with their study.

References

[1] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[2] J. Brandstetter, M. Welling, and D. E. Worrall. Lie point symmetry data augmentation for neural pde solvers. In *International Conference on Machine Learning*, pages 2241–2256. PMLR, 2022.

[3] J. Brandstetter, D. E. Worrall, and M. Welling. Message passing neural PDE solvers. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=vSix3HPYKSU`.

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

[5] 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. URL `https://openreview.net/forum?id=c8P9NQVtmnO`.

[6] 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*, 36, 2024.

[7] 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*, 433:117441, 2025. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2024.117441. URL `https://www.sciencedirect.com/science/article/pii/S0045782524006960`.

[8] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

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

[10] J. Mikhaeil, Z. Monfared, and D. Durstewitz. On the difficulty of learning chaotic dynamics with rnns. *Advances in neural information processing systems*, 35:11297–11312, 2022.

[11] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[12] 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.

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

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

[15] P. Teutsch and P. Mäder. Flipped classroom: effective teaching for time series forecasting. *arXiv preprint arXiv:2210.08959*, 2022.

[16] P. R. Vlachas and P. Koumoutsakos. Learning on predictions: Fusing training and autoregressive inference for long-term spatiotemporal forecasts. *Physica D: Nonlinear Phenomena*, 470: 134371, 2024.

A Appendices

A.1 Baseline Training Strategy

Training a neural operator for long-term prediction presents significant challenges. Consider a time t in a dataset with $T+1$ evenly spaced time samples ($0 \leq t \geq T$) and an autoregressive model can be trained to predict T timesteps during inference. Therefore, we need an initial condition (or ground truth solution) $u_{t=0}$ to predict the next time step $\tilde{u}_{t+1}$, which is fed back to the model to predict $\tilde{u}_{t+2}$, and so on until $\tilde{u}_T$ solution is obtained, corresponding to T predictions. This inference approach is referred to as autoregressive prediction.

To achieve autoregressive inference, the autoregressive-rollout (AR) training strategy is ideal for training a model. This approach predicts T timesteps by doing T rollouts, i.e., passing the predicted solution into the model to predict the next time step. Then the total loss used to optimise the weights of the model becomes the sum of losses at all T predictions. Since the loss is a function of the current prediction, and the current prediction is a function of the previous prediction, the computational graph is complex, leading to gradient accumulation. Therefore, full autoregressive rollout on a large T can be computationally prohibitive due to memory constraints and computational cost.

To mitigate this issue, a common strategy involves training on shorter sequences of length $\widehat{T}+1$, where $\widehat{T}$ is significantly smaller than T ($\widehat{T} \ll T$). To effectively utilise the available data despite the smaller sequence, we can sample multiple batches of $\widehat{T}+1$-length subsequences, each with a different starting time $t=0$ within the original T-length sequence. Based on this, we can define two baseline training strategies as seen in Figure 1: Teacher-Forcing (TF) and Autoregressive Rollout (AR).

Teacher-forcing (TF) is a one-rollout training strategy ($\widehat{T}=1$) that only predicts the next time step, i.e., for any $t \in T-\widehat{T}$, teacher forcing takes the ground truth solution and predicts the next time step ($t+1$). This approach is very computationally efficient because it avoids gradient accumulation caused by multi-rollout ($\widehat{T}>1$). However, this approach can be unstable during inference due to the amplification of errors or noise in the prediction that the model has not seen during training. This can lead to solution divergence after long rollouts. To address this, Gaussian noise of varying levels can be added to the ground truth solution to improve robustness to errors. We refer to this approach as teacher forcing with noise (TF+N).

Fixed $\widehat{T}+1$-length autoregressive rollout (AR) is similar to full autoregressive rollout, however we predict for a much shorter $\widehat{T}+1$ subsequence, where $\widehat{T}$ is greater than one but much less than T ($1<\widehat{T} \ll T$). This approach enables a reduction in computational cost compared to the full autoregressive rollout of $T+1$ sequences, while also mitigating the discrepancy between the training and testing phases inherent in teacher forcing.

A.1.1 Gradient propagation during training

For a training batch of $\widehat{T}$ timesteps from a dataset of $T+1$ time samples, we can define our training procedure as;

$$\theta^* = argmin \left[\sum_{t=0}^{T-\widehat{T}} \sum_{\widehat{t}=1}^{\widehat{T}} l\left(u_{t+\widehat{t}},\, \tilde{u}_{t+\widehat{t}}\right) \right]. \tag{4}$$

We sequentially iterate through each timestep from $t=0$ to $T-\widehat{T}$. The initial solution u_t is fed into the model to autoregressively predict the next $\widehat{T}$ steps, where the total loss is the sum of all losses at each time step $\hat{t}$. For each t with $\widehat{T}$ steps, the loss is defined as;

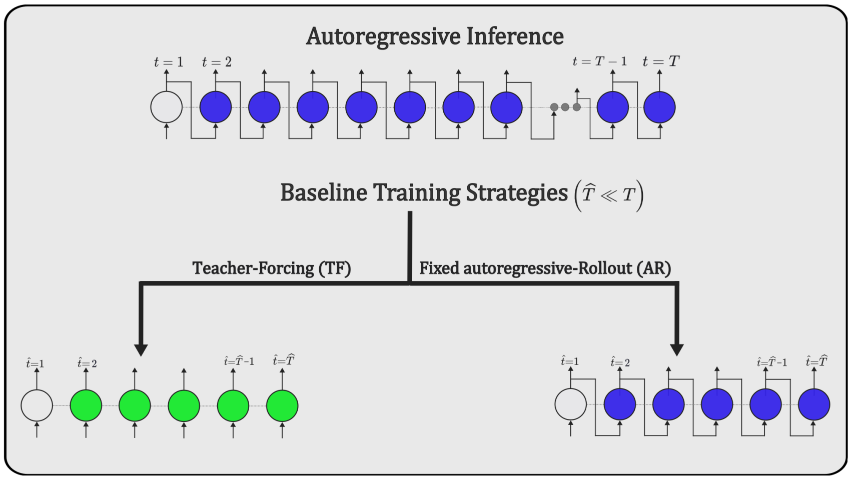


Figure 1: Baseline training strategies of $\widehat{T}$ time step: Teacher forcing (one-rollout) and autoregressive rollout (multi-rollout), where T represents the total time steps of the data and $\widehat{T}$ represent the training batch time steps.

$$L_t = \sum_{\hat{t}=1}^{\widehat{T}} l_{\hat{t}} = l(u_{\hat{t}},\, \tilde{u}_{\hat{t}}) \tag{5}$$

If $\widehat{T} = T$, and $\hat{t} = t$, we can rewrite Equation 5 as;

$$L_t = \sum_{t=1}^{T} l_t = l(u_t,\, \tilde{u}_t) \tag{6}$$

The gradients associated with this loss $\frac{\partial L_t}{\partial \theta}$ are obtained by backpropagating the loss at each step, and can be expressed as;

$$\frac{\partial l_t}{\partial \theta} = \sum_{tt=1}^{t} \left[\frac{\partial l_t}{\partial \tilde{u}_t} \left(\prod_{ttt=1}^{t-tt} \frac{\partial \tilde{u}_{t-ttt+1}}{\partial \tilde{u}_{t-ttt}} \right) \frac{\partial \tilde{u}_{tt}}{\partial \theta} \right]. \tag{7}$$

For example, when $T = 3$, the total loss is:

$$L_3 = l_1 + l_2 + l_3 \tag{8}$$

we can write the loss gradient of each loss l_1, l_2 and l_3 with respect to the model's parameter θ using equation 7 as follows;

$$\frac{\partial l_1}{\partial \theta} = \frac{\partial l_1}{\partial \tilde{u}_1} \frac{\tilde{u}_1}{\partial \theta}, \tag{9}$$

$$\frac{\partial l_2}{\partial \theta} = \frac{\partial l_2}{\partial \tilde{u}_2} \frac{\tilde{u}_2}{\partial \theta} + \frac{\partial l_2}{\partial \tilde{u}_2} \left(\frac{\partial \tilde{u}_2}{\partial \tilde{u}_1} \right) \frac{\partial \tilde{u}_1}{\partial \theta}, \tag{10}$$

$$\frac{\partial l_3}{\partial \theta} = \frac{\partial l_3}{\partial \tilde{u}_3} \frac{\tilde{u}_3}{\partial \theta} + \frac{\partial l_3}{\partial \tilde{u}_3} \left(\frac{\partial \tilde{u}_3}{\partial \tilde{u}_2} \right) \frac{\partial \tilde{u}_2}{\partial \theta} + \frac{\partial l_3}{\partial \tilde{u}_3} \left(\frac{\partial \tilde{u}_3}{\partial \tilde{u}_2} \frac{\partial \tilde{u}_2}{\partial \tilde{u}_1} \right) \frac{\tilde{u}_1}{\partial \theta}. \tag{11}$$

Observing the gradient formulation, the behaviour of the loss gradient is significantly dependent on the series of product Jacobians, which is composed of two consecutive solutions i.e., $u_{t-ttt-1}$ and u_{t-ttt}. In other words, there is a strong relationship between the loss gradient and the dynamics of the PDE. For chaotic systems, [10] showed that in the geometric mean, the eigenvalues of the Jacobians are greater than 1;

$$\left\| \prod_{ttt=1}^{t-tt} \frac{\partial \tilde{u}_{t-ttt+1}}{\partial \tilde{u}_{t-ttt}} \right\| > 1. \tag{12}$$

This means that as T increases, the magnitude of the gradient also increases, and for large T, this can lead to unstable training and gradient divergence. In the limit as $T \to \infty$, the magnitude of the Jacobian is infinity;

$$\lim_{T\to\infty} \left\| \frac{\partial u_T}{\partial u_1} \right\| = \infty. \tag{13}$$

This leads to the exploding gradient phenomena that makes training difficult. Furthermore, their theoretical results indicate that the challenges of training autoregressive models cannot be easily avoided through specially designed architectures; therefore, handling the gradients during training is critical.

A.2 Curriculum Training Strategy

Curriculum learning is a training strategy that can improve the stability and accuracy of training autoregressive models by mimicking the natural process of learning [1]. The aim of curriculum learning is to begin training with easy samples and progress to more challenging samples, thereby increasing the level of difficulty during the training phase. Formulating a curriculum requires defining the curriculum metric and scheduler, where the metric defines the level of difficulty of a sample and the scheduler controls the rate of difficulty during training.

In the context of sequence prediction, the easy task is the teacher-forcing training (TF), i.e., to predict the solution $\tilde{u}_{\hat{t}}$, the recursive model $(\mathcal{F}_\theta)^{(\hat{t})}$ takes the ground truth solution $u_{\hat{t}-1}$. The difficult task is autoregressive training (AR), i.e., to predict $\tilde{u}_{\hat{t}}$, the model $(\mathcal{F}_\theta)^{(\hat{t})}$ takes the previous prediction $\tilde{u}_{\hat{t}-1}$, thereby exposing the model to its prediction during training (see Figure 1).

Therefore, the curriculum metric is defined as the proportion of TF to AR for a fixed training rollout $\widehat{T}$. This means that if there is more AR than TF in a training rollout, the task is more difficult, and conversely, the case where TF exceeds AR. Several works have proposed curriculum strategies to define the transition between TF and AR. We expand on this approach at two levels: per-training and per-epoch.

A.2.1 Scheduler: Per-training level

For each epoch of training (one pass through the dataset) e, we can define a parameter α_e that defines an AR ratio, i.e., the number of AR divided by the training rollout $\widehat{T}$, such that $0 \leq \alpha_e \leq 1$. This means that $\alpha = 1$ corresponds to full AR and $\alpha = 0$ corresponds to full TF for all rollouts, respectively. Therefore, we can define a curriculum or transition function $C : \mathbb{N} \to [\alpha_{e_{start}}, \alpha_{e_{end}}]$ that controls the value of α_e from $\alpha_{e_{start}}$ to $\alpha_{e_{end}}$. C can be any analytical function (linear, sigmoid and exponential).

This offers two training options, which are increasing and decreasing α as seen in Figure 2 for different analytical functions:

TF-AR curriculum ($\alpha_{e_{start}} \leq \alpha_e \leq \alpha_{e+1} \leq \alpha_{e_{end}}$)

For increasing α ($\alpha_{e_{start}} < \alpha_{e_{end}}$), the curriculum transitions from a high number of TF (low AR) to a high number of AR (low TF).

AR-TF curriculum ($\alpha_{e_{start}} \geq \alpha_e \geq \alpha_{e+1} \geq \alpha_{e_{end}}$)

For decreasing α ($\alpha_{e_{start}} > \alpha_{e_{end}}$), we transition from a high number of AR (low TF) to a high number of TF (low AR).

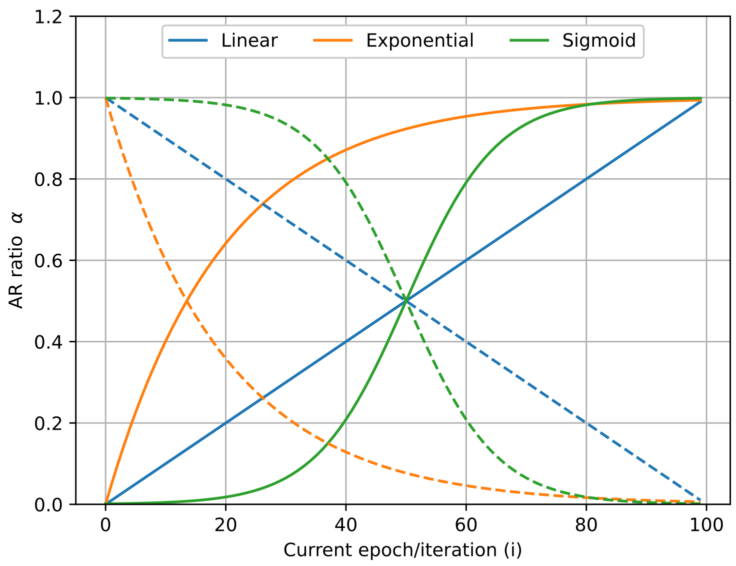


Figure 2: Transition function at training scale: Analytical functions for the transition function C, showing their increasing (straight line) and decreasing (dashed line) variant.

A.2.2 Metric: Per-epoch level

After the transition function C is defined that determines α at each epoch, we need to decide how to distribute AR and TF across the $\widehat{T}$ rollouts for each iteration e, to match α_e. For time step t, we can formalise the input to the model as a function of a decision function ϕ, where if $\phi = 1$, the function passes the previous prediction, otherwise, the ground truth solution;

$$\tilde{u}_{t+1} = \begin{cases} (\mathcal{F}_\theta)^{(t+1)}(\tilde{u}_t), & \text{if } \phi_{e,t} = 1 \quad \text{AR} \\ (\mathcal{F}_\theta)^{(t+1)}(u_t), & \text{otherwise} \quad \text{TF.} \end{cases} \tag{14}$$

We can broadly classify the approach to choosing between AR and TF into two categories: deterministic and probabilistic. Figure 3 is a diagram of these approaches during training.

Deterministic (D): This approach explicitly defines which rollout $(\mathcal{F}_\theta)^{(t)}$ uses AR and TF, by defining a transition point from TF to AR or vice-versa. In the case of the TF-AR curriculum, the AR ratio α_e determines the number of TF steps before switching to AR, and vice versa for the AR-TF curriculum. Therefore, we can define the decision function ϕ as:

$$\phi_{e,t} = \begin{cases} 1, & \text{if } \alpha_e \geq \frac{t}{\widehat{T}} \quad \text{AR} \\ 0, & \text{otherwise.} \end{cases} \tag{15}$$

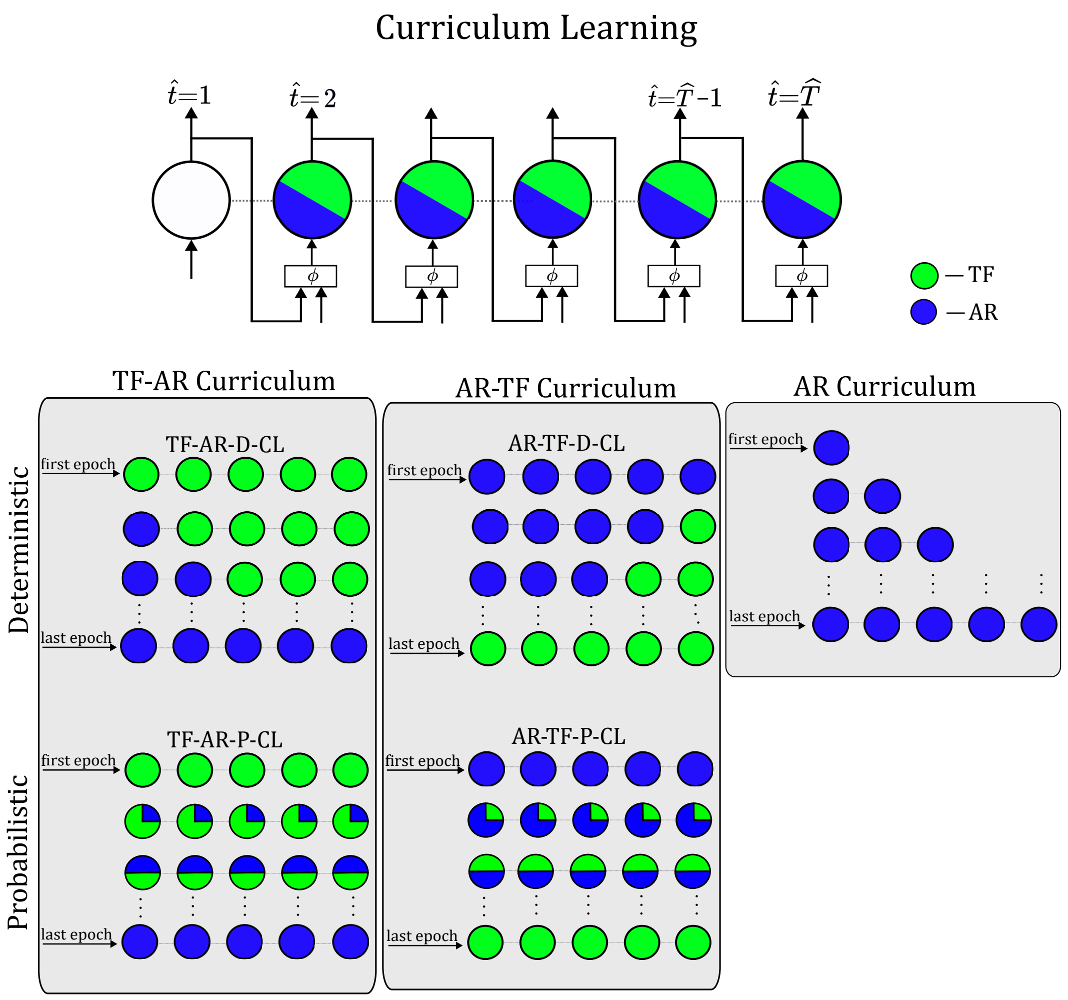


Figure 3: Iteration Scale: Deterministic and probabilistic variants of combining teacher-forcing (TF, green) and autoregressive rollout (AR, blue) .

Probabilistic (P): This approach interprets the AR ratio α_e as a probability measure for $(\mathcal{F}_\theta)^{(t)}$ for $\widehat{t} \in \widehat{T}$ being a AR step. The decision function $\phi_{e,t}$ is defined as a discrete random variable that is drawn from a Bernoulli distribution;

$$\phi_e \sim \textbf{Bernoulli}(\alpha_e) \tag{16}$$

Based on the training and epoch scales, we can define different curricula, as shown in Figure 3. This includes choosing between increasing or decreasing α, which corresponds to the TF-AR and AR-TF curricula, respectively. The per-epoch level option includes deterministic (D) or probabilistic (P) options that determine how to choose between TF and AR for each time step.

A.3 Autoregressive rollout Curriculum (AR-CL)

Another curriculum option is to use only AR, but vary the training rollout $\widehat{T}$. In this case, we can redefine our curriculum metric, since we do not use TF. The classification between easy and hard problems is now defined by the number of training rollouts $\widehat{T}$. Due to the effect of increasing $\widehat{T}$ on the loss gradient as discussed in Section A.1.1, we can state that;

As training rollout $\widehat{T}$ increases, the magnitude of the loss gradient increases,
leading to increased training difficulty.

The AR ratio α corresponds to the percentage of maximum training rollout $\widehat{T}$. Therefore, we define an increasing α case called AR curriculum, which increases the number of training rollouts, as seen in Figure 3.

A.4 Scheduled Temporal Loss Weighting (AR-STLW-CL)

A.4.1 Weighting (Transition) function

The weighted loss at each predicted time step is determined by the weight value at each time step w_t and its corresponding loss l_t. Therefore, we use a predefined weighting function C to determine the weight. We can use analytical functions (linear or exponential) to determine the distribution of weight values at each per-epoch level and per-training level.

For each training, we define a transition function C that determines the weight value for each timestep at each epoch.

$$w_{e,\,t} = C(e,t) \qquad t \in 0, 1, \cdots, T \tag{17}$$

At each epoch e, we defined the weights such that $w_{e,\,t-1} < w_{e,\,t} < w_{e,\,t+1}$. After conducting an empirical investigation, we chose to use an exponential function, as it was the most effective.

$$C(e,t) = \exp\left(-C_e\, t^2\right). \tag{18}$$

We define C_e as;

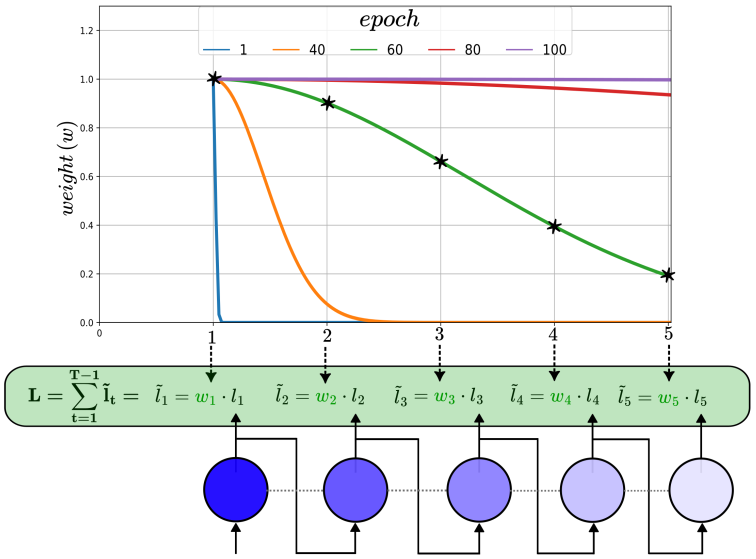
$$C_e(e) = \exp\left(-A\, e\, \frac{T}{E}\right), \tag{19}$$

E is the total number of epoch (i.e, $e = 1, 2, 3, \cdots E$). T and E are fixed values based on training data and training time, respectively. A is a constant that accounts for different values of T. We obtain an empirical function by fitting an exponential curve to the plot A versus T.

$$A = \exp\left(5.0\, T^{\,-0.723}\right) \tag{20}$$

Figure 4 shows a visualisation of the weighting function during training ($E = 100$). 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 approximately one ($w_t \approx 1$).

In summary, the transition function C (18) determines the weight values over the total timesteps (T). We chose an exponential function that forms a Gaussian curve, centred on $t = 1$. The value of C_e (19) determines the width of the curve. As epoch e increases, the width of the curve (C_e) increases based on an exponential function that depends on the total timesteps T and the total number of epochs E. Finally, we define an empirical function A to account for training data with different T values, thereby maintaining a smooth transition.



epoch
1
40
60
80
100
weight (w)
1
2
3
4
5
$\mathbf{L} = \sum_{t=1}^{T-1} \tilde{\mathbf{l}}_t = \tilde{l}_1 = w_1 \cdot l_1 \quad \tilde{l}_2 = w_2 \cdot l_2 \quad \tilde{l}_3 = w_3 \cdot l_3 \quad \tilde{l}_4 = w_4 \cdot l_4 \quad \tilde{l}_5 = w_5 \cdot l_5$

Figure 4: A schematic of the scheduled temporal loss Weighting (STLW) Curriculum. For a data with a subsequence of length T ($T = 6$), a weight w is assigned to each time step for $t > 0$, which controls the loss l contributed by each time step (for example, at epoch $= 60$, the weight for each time step is highlighted in black markers). 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., $w \approx 1$.

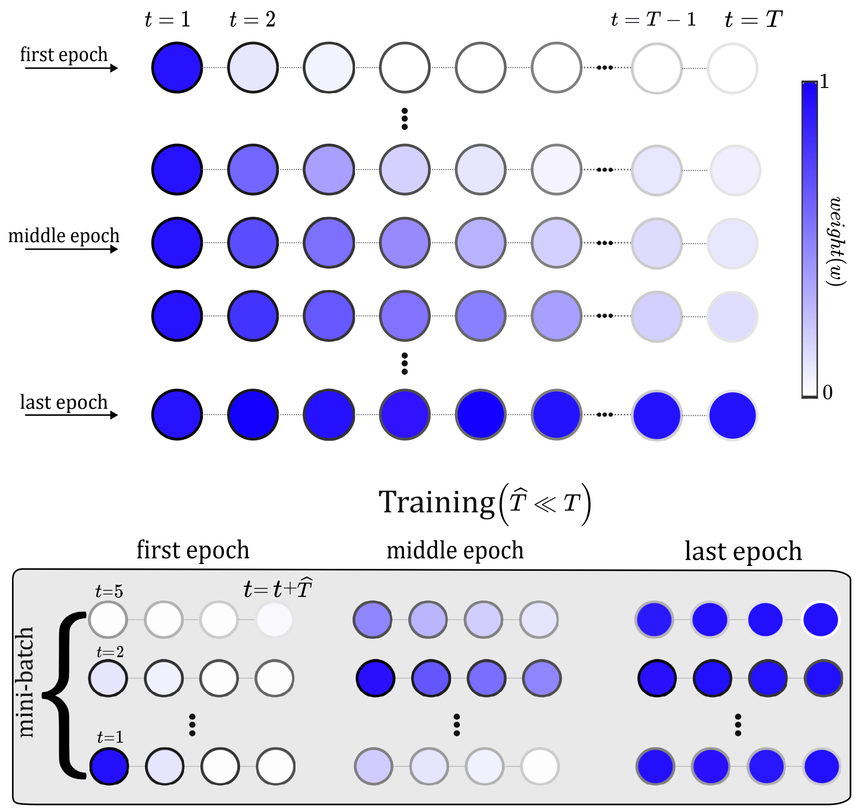


Figure 5: STLW: A schematic of the Scheduled Temporal Loss Weighting (STLW) Curriculum during training. Time steps are represented as a circle with some colour intensity. The colour intensity represents the weight value. In AR-STLW-CL, as the epoch increases, the weight value of the later time step increases.

A.4.2 Training Algorithm

A pseudocode of the training algorithm is shown in Table 2. We train using mini-batches of randomly sampled initial timesteps over a number of iterations per epoch. The backpropagated loss is the sum of the weighted losses at the predicted time steps, and the model weights are updated *iteration* times per epoch.

```
Algorithm 1: Algorithm of the STLW Curriculum
Input: total epoch $E$, iteration per epoch $iteration$,
temporal weights $\{w_{e,\,t}\}_{e=1,\cdots,E,\quad t=0,\cdots,T}$, training samples: $\{u_t\}_{t=0,\cdots,T}$,
fixed training rollout $\widehat{T}$, model with parameters $\theta$ and loss function criterion.

for $e = 1$ to $E$ do
    Calculate $w_{e,\,t}$ for $t = 0, \cdots, T$ using eq. 20    $\rightarrow$ weights for time steps at e.
    for $iter = 1$ to $iteration$ do
        $t \in$ RandomNumber($0,\ T - R$)    $\rightarrow$Random initial time btw 0 and $T - R$.
        Total loss = 0
        for $\hat{t}$ in Range $\left(0,\ \widehat{T}\right)$ do    $\rightarrow$ Indicies of the first step of each rollout.
            $\tilde{u}_{t+\hat{t}+1} =$ model $\left(\tilde{u}_{t+\hat{t}}; \theta\right)$
            $l_{t+\hat{t}+1} =$ criterion $\left(\tilde{u}_{t+\hat{t}+1}, u_{t+\hat{t}+1}\right)$
            $\widehat{L} =$ sum $\left(w_{e,\,t+\hat{t}+1} \cdot l_{t+\hat{t}+1}\right)$    $\rightarrow$ Sum of weighted losses.
            Total loss = Total loss + $\widehat{L}$
        end
        $\theta \leftarrow$ Optimiser (Total Loss)
    end
end
```

Table 2: Pseudocode: Scheduled Temporal Weighted Loss Curriculum Algorithm

A.4.3 Gradient Propagation

For our approach, we can rewrite the standard optimisation (4) for the training procedure as;

$$\theta^* = argmin \left[\sum_{t=0}^{T-\widehat{T}} \sum_{\widehat{t}=1}^{\widehat{T}} w_{e,\,t+\hat{t}} \cdot l\left(u_{t+\widehat{t}},\ \tilde{u}_{t+\widehat{t}}\right) \right]. \tag{21}$$

For convenience, we consider $\widehat{T} = T$, thus $\hat{t} \rightarrow t$. The weighted loss can be written as;

$$\widehat{L}_t = \sum_{t=1}^{T} \hat{l}_t = w_{e,\,t} \cdot l(u_t,\ \tilde{u}_t) \tag{22}$$

We can rewrite the loss gradient in equation 7 as;

$$\begin{aligned} \frac{\partial \hat{l}_t}{\partial \theta} &= \sum_{tt=1}^{t} \left[\frac{\partial \hat{l}_t}{\partial \tilde{u}_t} \left(\prod_{ttt=1}^{t-tt} \frac{\partial \tilde{u}_{t-ttt+1}}{\partial \tilde{u}_{t-ttt}} \right) \frac{\partial \tilde{u}_{tt}}{\partial \theta} \right] \\ \frac{\partial \hat{l}_t}{\partial \theta} &= w_{e,\,t} \left(\sum_{tt=1}^{t} \left[\frac{\partial l_t}{\partial \tilde{u}_t} \left(\prod_{ttt=1}^{t-tt} \frac{\partial \tilde{u}_{t-ttt+1}}{\partial \tilde{u}_{t-ttt}} \right) \frac{\partial \tilde{u}_{tt}}{\partial \theta} \right] \right) \\ \frac{\partial \hat{l}_t}{\partial \theta} &= w_{e,\,t} \frac{\partial l_t}{\partial \theta}. \end{aligned} \tag{23}$$

When we compare our weighted loss gradient with the standard formulation in 7, we observe that the weights are backpropagated through the optimisation process, and the weighted loss gradient is a product of the standard loss gradient and the scheduled weight as seen above. Therefore, our approach can be seen as a scheduled weighted loss gradient. Initially, a higher priority (low damping) is assigned to small t while a lower priority (high damping) is assigned to large t.

A.5 STLW as a learning rate Scheduler

Learning rate (η) is a crucial hyperparameter for optimisation using gradient descent. It helps to control the step size, prevent overshooting and ensure convergence, and our method can be seen as a form of a learning rate. Firstly, gradient descent can be written as;

$$\theta_{e+1} = \theta_e - \eta_e \frac{\partial L}{\partial \theta}. \tag{24}$$

It is common practice to use a scheduler (linear, exponential, or cosine decay) to vary the learning rate during training, as this can improve training stability and accuracy. Typically, the learning rate decreases with epoch e. Our method can be written in the form;

$$\begin{aligned}\theta_{e+1} &= \theta_e - \eta_e \left(w_{e,\,t} \frac{\partial L_t}{\partial \theta} \right), \\ \theta_{e+1} &= \theta_e - \widehat{\eta}_{e,t} \frac{\partial L_t}{\partial \theta}, \qquad \widehat{\eta}_{e,t} = \eta_e \, w_{e,\,t}.\end{aligned} \tag{25}$$

Using this formulation, we can view our approach as having an extra dimension to the standard learning rate. This allows us to take different step sizes depending on the time t, enabling us to leverage the benefits of using a learning rate scheduler. Despite this formulation, we chose to classify our approach as a form of curriculum learning since the weights increase during training, which is generally the opposite direction of learning rates.

A.6 STLW as a Continuous AR Curriculum

Our approach can also be viewed as a continuous form of the discrete AR-Curriculum (Figure 6). Training with fixed-AR where the training rollout $\widehat{T}$ is large is prone to instabilities during training; thus, using a discrete curriculum improves stability by varying the number of training rollout. However, the discrete curriculum approach can be prone to overfitting to early dynamics in the early phase of training if the PDE of interest exhibits changing dynamics. Therefore, a continuous approach can help alleviate the issue with a discrete curriculum, where it does not completely ignore the later dynamics (time steps) but limits its effect through a weighted loss approach.

$$\begin{aligned}&\text{Fixed AR:} \quad \widehat{L}_t = \sum_{t=1}^{T} \hat{l}_t = w_{e,t} \cdot l(u_t - \tilde{u}_t), \qquad w_{e,\,t} = 1 \\ &\text{Discrete AR-CL:} \quad \widehat{L}_t = \sum_{t=1}^{\widehat{T}} \hat{l}_t = w_{e,t} \cdot l(u_t - \tilde{u}_t), \qquad w_{e,\,t} = [0,1] \\ &\text{Continuous AR-CL:} \quad \widehat{L}_t = \sum_{t=1}^{T} \hat{l}_t = w_{e,\,t} \cdot l(u_t - \tilde{u}_t), \qquad w_{e,\,t} \in (0,1)\end{aligned} \tag{26}$$

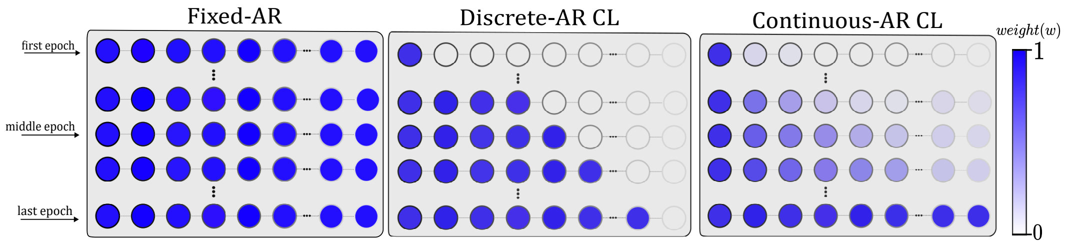


Figure 6: A schematic of the scheduled temporal loss Weighting (STLW) Curriculum. A weight is applied to each timestep, represented by the colour bar

A.7 Dataset

To evaluate our proposed curriculum learning approach, we performed experiments on four one-dimensional (1D) time-dependent PDEs that exhibit different evolution of the solution function: Advection (Av) and viscous Burgers (vB) from the PDEBench dataset Takamoto et al. [13], inviscid Burgers with forcing term (iB) from Brandstetter et al. [3], and Korteweg–De Vries (KdV) from Brandstetter et al. [2].

The Advection equation (Av) is a first-order hyperbolic PDE that models the linear transport of a quantity by bulk motion of a fluid. The PDE will test the ability of our curriculum strategies to evolve simple dynamics over a time sequence of 200 timesteps. This PDE can be considered to be the easiest of all the PDEs investigated.

The Burgers' equation (vB and iB) dynamics is governed by the balance between advection and dissipation. Specifically, vB transitions from sharp, high-amplitude solutions to smooth, low-amplitude solutions, while iB with a forcing term exhibits the opposite behaviour, transitioning from smooth, low-amplitude solutions to sharp, high-amplitude solutions. These PDEs will test the ability of our curriculum strategies to capture the changing dynamics of the solution function of v-B and i-B over time sequences of 200 and 250 steps, respectively. The dynamics of the Korteweg-de Vries (KdV) equation is an equation that models nonlinear dispersive nondissipative waves present on shallow water surfaces. This dynamics is more challenging than Av; however, this PDE will test the ability of our curriculum strategies over a longer sequence of 640 timesteps (first column in Figure.

A.8 Training

All models were trained and evaluated using the Fourier Neural Operator (FNO) with 16 Fourier modes, 32 hidden dimensions, and 4 layers. We explored the modes and hidden dimensions of the model to obtain a trade-off between accuracy and speed. For the training optimiser, we investigated with different starting learning rates and schedules. For training data, we investigated batch size, training rollout T to determine the best configuration for the experiments.

Based on this investigation, unless otherwise stated, we use the Adam optimiser, a batch size of 64 and a training rollout of $\widehat{T}$=24. We use an initial learning rate of 10^{-3} with a step-size scheduler that reduces the learning rate by a factor of 0.95 after each epoch. We randomly sample the starting point for each trajectory within a batch, repeating this process 100 times. In detail, for a single trajectory of T timesteps, we subsample $\widehat{T}$ timesteps, and perform this operation 100 times for each epoch (iteration). We train for a total of 100 epochs using an l_2 loss function. Finally, we run the model five times for each dataset and training strategy, using different random weight initialisations to obtain the mean and standard deviation.

We provide plots of the training and validation loss curves during training. In Figure 7, we show the baseline training strategies (TF, TF+N and AR) loss curves for a one-rollout loss and full-rollout for two datasets (vB and KdV). The shaded area is the standard deviation of the validation loss. We observe that TF and TF+N achieve lower errors for one-rollout loss, in other words, short-term prediction. The smaller shaded region of TF+N shows that adding noise improves the robustness of the teacher-forcing strategy. However, teacher-forcing strategies become relatively unstable during training for the full rollout loss (i.e., long-term prediction), with autoregressive rollout having better stability and accuracy.

Our experiments of the AR strategy with and without curriculum learning (CL) demonstrate that CL training outperforms the baseline strategy on both one-rollout and full-rollout losses, as shown in Figure 8, i.e., AR-CL and AR-STLW-CL has smaller error over AR for both losses. Interestingly, AR-CL achieves smaller errors than AR-STLW-CL for one-rollout loss, while for full-rollout loss, AR-STLW-CL achieves a smaller error than AR-CL. This supports our claim that AR-STLW-CL is more suitable for long-term prediction.

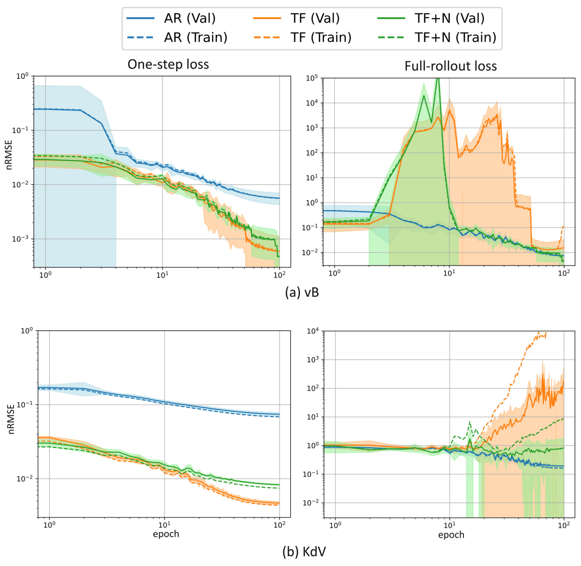


Figure 7: Training dynamics of baseline strategies. The training (dashed line) and validation (straight line) losses are shown for each strategy.

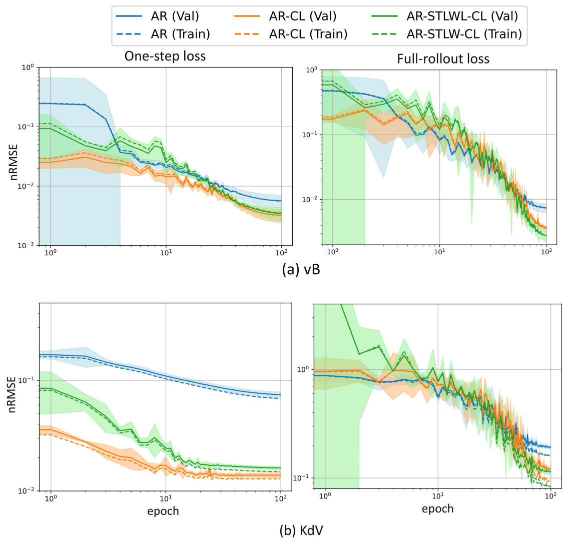


Figure 8: Training dynamics of curriculum strategies. The training (dashed line) and validation (straight line) losses are shown for each strategy.

A.9 Long-term Accuracy and Stability

To assess long-term accuracy, we compute the nRMSE over the final 10% of the timesteps, presented in the "last 10 $\%$" column of Table 1. Excluding the probabilistic approaches, curriculum learning consistently improved accuracy at later timesteps. AR-STLW-CL demonstrated the lowest error for two datasets (i-B and KdV), while TF-AR-D-CL achieved the lowest error for Av and v-B. We also investigated the stability of the curriculum strategies by examining the standard deviation across five different runs for each curriculum and dataset. Our result revealed that curriculum learning strategies, excluding the probabilistic ones, show a smaller standard deviation, indicating enhanced stability and robustness. Specifically, AR-STLW-CL showed the smallest standard deviation for Av, AR-CL for v-B, TF-AR-P-CL for i-B and TF-AR-D-CL for KdV.

A.10 Error Propagation Over Time

Figure 9 shows the error propagation over time for the top 3 curricula for each dataset. These plots clearly demonstrate the benefits of using curriculum strategies across all datasets. While AR-STLW-CL performs best with the smallest nRMSE, the graph indicates that its gains over the other two top curricula are less pronounced over time.

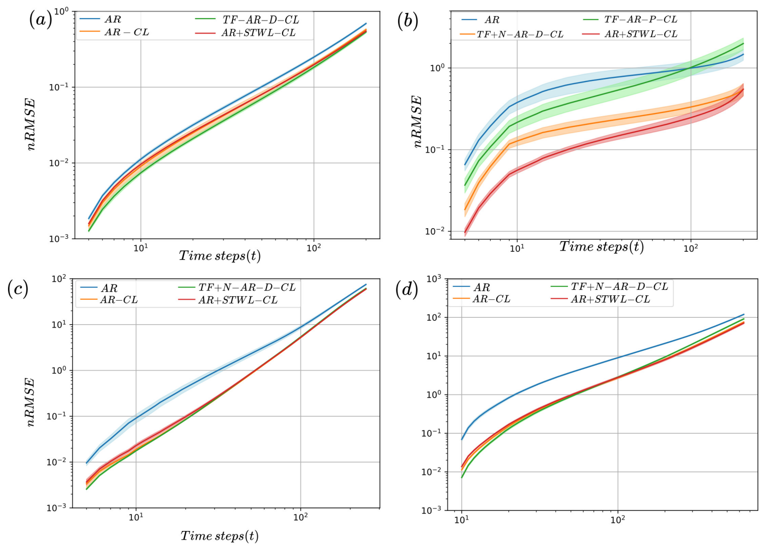


Figure 9: Error propagation over time compared with three strategies (Best baseline and Top 2 Curriculum), respectively, to each PDE. (a) Av (b)iB (b)vB (c) E1 (d)KdV .

A.11 Training Rollout Analysis

In Section A.6, we established that our approach can be conceptualised as a continuous form of the discrete autoregressive rollout (AR-CL). While discrete AR-CL progressively increases the number of training rollouts $\widehat{T}$, our continuous approach applies weights to each rollout. Therefore, we investigate the impact of varying the number of training rollouts $\widehat{T}$, on the performance of fixed-AR, discrete AR-CL, and continuous AR-CL (our approach).

Figure 10 presents the nRMSE result for the training rollouts $\widehat{T}$ at 2, 4, 8, 16 and 32. We compare the performance of the three training strategies across each dataset. Our observations indicate that AR-STLW-CL consistently exhibits smaller errors and superior stability per rollout, demonstrating more effective scaling with an increasing number of training rollouts compared to both AR and discrete AR-CL. Specially:

- For the Av dataset, AR-STLW-CL remains stable at small $\widehat{T}$.
- In the vB dataset, while a general trend shows the error reaching a minimum at $\widehat{T}$=8, AR-STLW-CL consistently achieves smaller errors across the training rollouts values.
- For iB and KdV, although AR-CL tends to increase after $\widehat{T} = 8$, both discrete AR-CL and AR-STLW-CL show a reduction in error, with AR-STLW-CL ultimately achieving smaller errors.

In summary, AR-STLW-CL demonstrates better stability across varying training rollouts and consistently achieves lower errors per rollout, highlighting its robustness and effectiveness.

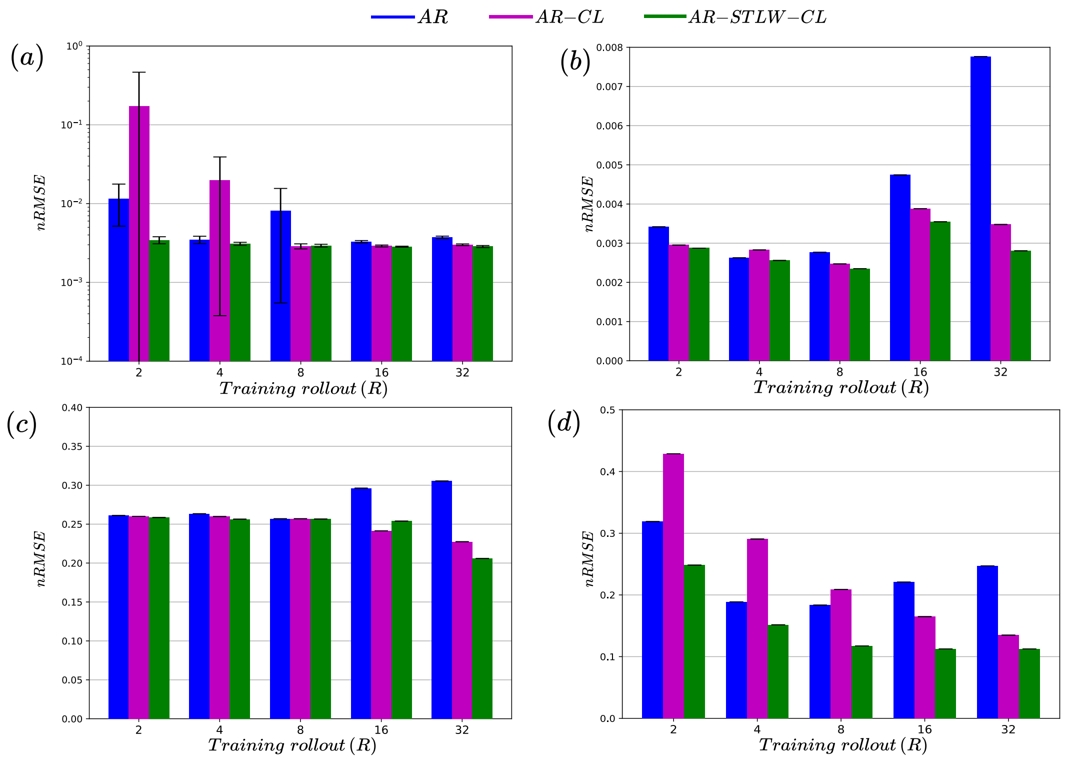


Figure 10: Effect of increasing the training rollout $\widehat{T}$ from 2 to 32 against the error of three training strategies (Best baseline and top 2 Curriculum). (a) Av (b)iB (b)vB (c) iB (d)KdV.

A.12 Inference Rollout Analysis

To asses the efficacy of our training strategy (AR-STLW-CL) for long-term prediction, we investigated its performance against other strategies by evaluating various inference rollout values (T). Table 3 presents the mean nRMSE and standard deviation for different inference rollouts for the Advection (Av) and Korteweg-De Vries (KdV) datasets. For each dataset, each reported value corresponds to a distinct model configured with a different output size to achieve the specified inference rollouts. For Av, we investigate T = 40 and T = 200, while for KdV, we investigate T = 128, T = 320, and T = 640.

The results demonstrate that AR-STLW-CL consistently shows the lowest error compared to other strategies across nearly all inference rollouts, with the exception of the smallest rollout ($T = 40$). This finding further underscores the advantage of our curriculum strategy in achieving better long-term accuracy and stability.

	STRATEGY	Inference rollout T		
		40		200
Advection (Av)	TF	$0.0375^{\pm 1.76e-2}$		$0.486^{\pm 4.53e-2}$
	TF+N	$0.0165^{\pm 5.68e-3}$		NaN
	AR	$0.00353^{\pm 8.70e-5}$		$859.6^{\pm 1913.4}$
	AR-CL	$\underline{0.00293}^{\pm 8.98e-5}$		$\underline{0.0551}^{\pm 1.01e-1}$
	TF-AR-D-CL	$\mathbf{0.00272}^{\pm 9.58e-5}$		$10.9^{\pm 23.1}$
	TF+N-AR-D-CL	$0.00296^{\pm 1.43e-4}$		$1.057^{\pm 1.93}$
	AR-TF-P-CL	$0.00538^{\pm 1.18e-3}$		$0.196^{\pm 1.12e-1}$
	TF-AR-P-CL	$0.00294^{\pm 9.48e-5}$		$29.9^{\pm 65.7}$
	AR-STLW-CL (**ours**)	$0.00303^{\pm 0.0}$		$\mathbf{0.0135}^{\pm 8.72e-3}$
		Inference rollout T		
		128	320	640
Korteweg–De Vries (KdV)	TF	$0.541^{\pm 9.50e-2}$	-	-
	TF+N	$0.347^{\pm 1.01e-1}$	$0.408^{\pm 1.14e-2}$	$0.810^{\pm 1.37e-1}$
	AR	$0.189^{\pm 5.29e-3}$	$0.298^{\pm 6.62e-2}$	$0.518^{\pm 1.94e-1}$
	AR-CL	$\underline{0.119}^{\pm 5.29e-3}$	$0.196^{\pm 6.608e-3}$	$249449.2^{\pm 557783.3}$
	TF-AR-D-CL	$0.159^{\pm 4.13e-3}$	$0.244^{\pm 1.42e-2}$	$0.366^{\pm 1.29e-2}$
	TF+N-AR-D-CL	$0.145^{\pm 4.74e-3}$	-	-
	AR-TF-P-CL	$0.226^{\pm 2.0e-2}$	-	-
	TF-AR-P-CL	$0.189^{\pm 7.29e-3}$	-	-
	AR-STLW-CL (**ours**)	$\mathbf{0.113^{\pm 5.02e-3}}$	$\mathbf{0.170}^{\pm \mathbf{1.09e-2}}$	$\mathbf{0.307}^{\pm \mathbf{2.85e-2}}$

Table 3: **Inference rollout**: Normalised root mean squared error (nRMSE) with standard deviation. The numbers in **Bold** and <u>underline</u> are the **smallest** and <u>second smallest</u> error in the respective column.

A.13 STLW as a learning rate scheduler

Our curriculum approach, AR-STLW-CL, can be viewed as a two-dimensional learning rate (Lr) scheduler. Unlike traditional schedulers that only vary the learning rate with respect to the training epoch (e), AR-STLW-CL, applies a weight (w) to the learning rate, where w is a function of both the epoch (e) and the prediction time ($t \in T$). You can find a more detailed explanation in Section A.5).

To validate this hypothesis, we examine the test loss curves shown in Figure 11. The figure compares different training approaches at a fixed learning rate of 1e-4 (left) and 1e-5 (right). In each sub-figure, we show training strategy of fixed learning rate with and without our curriculum, i.e., AR+STLW-CL+fixed LR and AR+fixed Lr, respectively. For comparison, we also include discrete AR-CL with fixed learning rate (AR-CL+fixed Lr) and a standard training approach with variable learning rate, i.e., AR+var Lr.

When comparing fixed learning with and without our curriculum, i.e., AR+STLW-CL+fixed Lr and AR+fixed Lr, our curriculum significantly improves the final test loss. When we compare our approach against the standard approach with a variable learning rate (AR+Var Lr), we observe that the curve has similar behaviour, even though the latter strategy ultimately leads to a smaller test loss. This similarity in the curves suggests that both variable learning rates (with a standard scheduler) and our weighted loss approach (STLW) have a comparable effect on the training phase. This becomes even more apparent when contrasting it with the distinct curve behaviour of AR-CL+Fixed Lr, where the number of training rollouts is varied instead.

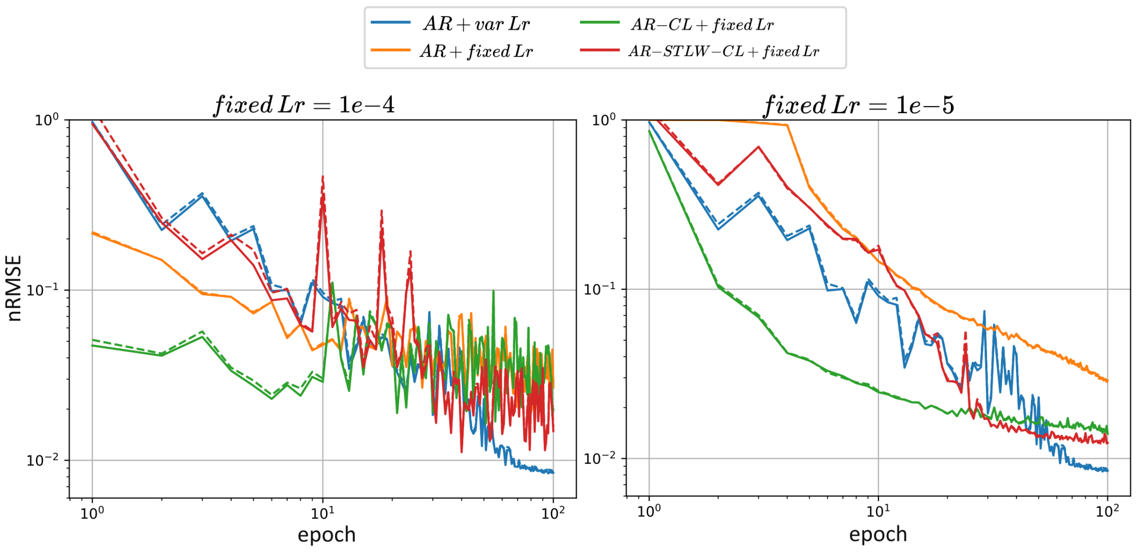


Figure 11: Training with a fixed and variable (Scheduler) learning rate (Lr) on different curricula using Adam Optimiser.