SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Guiding the Unrolling Curricula of Autoregressive Neural Simulators

Airat Valiullin

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Guiding the Unrolling Curricula of Autoregressive Neural Simulators

Steuerung der Unrolling-Curricula von Autoregressiven Neuronalen Simulatoren

Author:	Airat Valiullin
Supervisor:	Björn List, MSc
Advisor:	Prof. Dr. Nils Thürey
Submission Date:	07.07.2025

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 07.07.2025 Airat Valiullin

[illegible]

Acknowledgments

I would like to sincerely thank Björn List for his invaluable guidance, thoughtful feedback, and hands-on support throughout this thesis. His suggestions, encouragement, and help in shaping the experiments were crucial to every stage of this work. I also acknowledge Prof. Thürey, whose official supervision made this thesis possible at the Chair of Computational Physics at the Technical University of Munich.

I further thank the Technical University of Munich for the opportunity to carry out this thesis. It has been a privilege to be part of this academic community, benefiting from excellent courses and the dedication of wonderful professors and teaching staff, all of which have greatly enriched my learning and research experience.

I am also deeply grateful to my friends and family for their unwavering support, patience, and encouragement: not only during my work on this thesis but throughout my studies and personal growth over the past years.

> [...] the peoples of the United Nations have [...] reaffirmed their faith in fundamental human rights, in the dignity and worth of the human person and in the equal rights of men and women and have determined to promote social progress and better standards of life in larger freedom...
>
> — *Universal Declaration of Human Rights, Preamble* [Uni48]

I dedicate this work to all those striving for justice and dignity under oppression.

Abstract

This thesis investigates training strategies for neural networks to model chaotic partial differential equations (PDEs), using the Kuramoto–Sivashinsky equation as a benchmark. An evaluation metric, the Cumulative One-Step Mean Squared Error (COSMSE), is proposed to assess short-term predictive accuracy without penalising long-term divergence inherent to chaotic systems. The work analyses the training instabilities introduced by increasing unrolling horizons and introduces an automatic scheduler that adaptively extends these horizons based on model performance. Experimental results demonstrate that this scheduler achieves performance on par with manually designed curricula. The techniques developed offer practical tools for more reliable learning of chaotic dynamics.

Contents

1 Introduction

1.1 Numerical and Neural Solvers of Partial Differential Equations

Partial Differential Equations (PDEs) are essential tools for modelling dynamic systems with applications in science and engineering. While essential for prediction and understanding, many real-world PDEs have no reachable analytical solutions. Classical numerical methods, such as finite difference, finite element, and finite volume methods, offer high accuracy and parallelisable algorithms, but are often computationally expensive per simulation as well as being heavily mesh-dependent.

Recently, deep learning has gained traction as a complementary approach for solving PDEs [FDC20; Koc+21]. A wide range of neural PDE solvers has emerged, in which neural networks are trained to approximate either the solution function or the solution operator of a PDE. These models can be mesh-free, data-efficient, and, once trained, provide fast inference. One early example is the Physics-Informed Neural Network (PINN) framework [RPK19], which incorporates the physical structure of PDEs directly into the loss function. Building on such ideas, later methods have demonstrated competitive accuracy in domains such as computational fluid dynamics [Wan+20; Bot+25], molecular dynamics [Raz+21; Doe+21], and even geometry-sensitive tasks like mesh optimisation, drug discovery, and cloth simulation [Pfa+21; Ask+23; Pen+23].

1.2 Limitations of Neural Solvers

Despite promising results, training neural PDE solvers remains challenging. Models can be sensitive to optimisation techniques, and it is difficult to ensure stable and accurate long-term behaviour. Properly encoding boundary conditions and maintaining physical consistency are persistent open problems, among many [Lip+23; GZW22; SPP24]. Practical experience often reveals additional complications, including poor generalization when applied to new problem settings [CB22].

To address some of these limitations, hybrid approaches have been developed that integrate neural components with classical solvers. These models seek to unite the flexibility and speed of learned systems with the robustness and interpretability of traditional numerical schemes [Rue+20; LWX21; Jin+25].

1.3 Research Question

This thesis studies *unrolling*, a technique where gradients are propagated through entire chains, and the network trains on sequences of its own autoregressive predictions, rather than single steps, promoting stable, long-term inference, rather similar to real deployment [Lis+24; Lip+23; BWW23; LCT22; Koc+21]. However, efficient application of unrolling requires meticulous design of unrolling curricula tailored to the specific dynamics of the system. These curricula define schedules for the unrolling horizon length, deciding when and how to increase the number of prediction steps during training to gradually expose the model to longer sequences and more challenging learning scenarios.

Aiming for more adaptive and effective neural PDE training, we propose automating training curricula in real time by guiding unrolling horizon switches using a custom performance metric. In Chapter 2, we introduce the dynamical system used as a benchmark for our numerical experiments — the Kuramoto–Sivashinsky equation — as well as the metric — Cumulative One-Step Mean Squared Error, — and explain our choice by comparing it to a traditional alternative. Chapter 3 outlines the experimental methodology, including the network architectures (and the rationale behind the chosen design), learning setup, unrolling implementation, data structure, and optimisation objective. Since switching unrolling horizons causes significant error spikes, Chapter 4 explores simple techniques towards mitigating this disruptive behaviour. In Chapter 5, we propose an algorithm to automate unrolling horizon switches, highlighting its strengths and limitations. Finally, Chapter 6 summarises our findings, discusses the shortcomings of our algorithm, and considers how the proposed method may extend to other dynamical systems.

2 Physical System

This chapter introduces the physical system at the core of our study: the Kuramoto–Sivashinsky equation.

We begin by describing this equation, explaining why its chaotic nature makes it a valuable and challenging test for our methods. Following this, we define the metrics used to judge the performance of our models and explore why traditional error measures can be misleading for chaotic systems and introduce an alternative metric that better captures a model's ability to learn the underlying physical behaviour. Finally, we compare these metrics to show how they behave for both a well-trained and a poorly-trained model, demonstrating the importance of choosing the right tool for evaluation.

2.1 The Kuramoto–Sivashinsky Equation

The system of interest is the *Kuramoto–Sivashinsky (KS)* equation, a nonlinear partial differential equation (PDE) widely used as a benchmark in studies of chaotic dynamics and data-driven modelling. Originally introduced as a simplified model for instabilities in laminar flame fronts, it has since become a canonical system for exploring spatio-temporal chaos, pattern formation, and the development of machine learning methods for complex dynamical systems [BHW22; PWM22; OAH20; GKK24; Che+20; ÖM25; RK17].

We consider the one-dimensional KS equation in the form:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} = 0, \tag{2.1}$$

defined on a spatial domain $x \in [0, 2\pi L]$ and time interval $t \in [0, +\infty)$.

The range of domain sizes studied in this work is given by:

$$\mathcal{X} = \{2\pi L \mid L \in \tilde{\mathcal{X}}\}, \quad \text{where } \tilde{\mathcal{X}} := [6, 15] \subset \mathbb{N}. \tag{2.2}$$

Unless stated otherwise, we refer to the *domain size* by the integer multiple $L \in \tilde{\mathcal{X}}$ of 2π in this work.

Previous studies investigated the time evolution of the KS system, characterised its chaotic attractor using Lyapunov exponents (see Section 5.1.1), and explored the impact

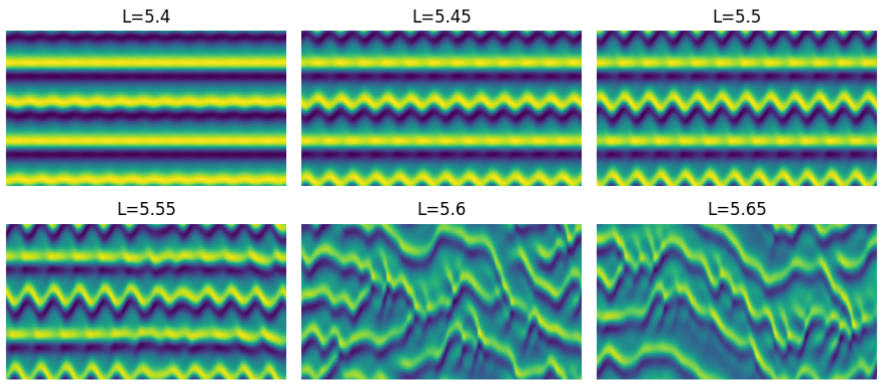


Figure 2.1: The effect of the domain size L on the dynamics of the KS equation. The domain size serves as a bifurcation parameter in the onset of the chaotic behaviour.

of the domain size (which serves as a bifurcation parameter) on the onset and nature of chaos. These analyses have revealed transitions between steady, periodic, and chaotic regimes depending on the system size (see Fig. 2.1) [Eds+19; Toh87; NST85].

These interesting dynamics arise from a balance of competing physical processes inherent in the PDE:

- large-scale instability induced by the second derivative term u''_{xx} (often referred to as a *negative viscosity* or destabilising diffusion);
- small-scale dissipation due to the fourth-order *hyper-diffusion* term $u^{(IV)}_{xxxx}$;
- non-linear *advection* term uu'_x which redistributes energy across scales.

The non-linear term acts as a stabilising mechanism by transferring energy from unstable large-scale modes to more stable small-scale structures. In Fourier space, the destabilising second derivative corresponds to a positive coefficient for low wave numbers, which contributes to the emergence of chaotic spatio-temporal dynamics.

As a result, the KS equation presents a particularly challenging test bed for machine learning approaches seeking to model chaotic autoregressive systems.

To generate the training dataset, we numerically solve the KS equation using a second-order exponential time-differencing Runge–Kutta scheme (ETD2RK) [CM02]. This method is particularly well-suited for stiff PDEs such as the KS equation. We take the solution produced by this integrator as the ground truth reference throughout this study.

2.2 Metrics

Multiple metrics have been considered for the evaluation and comparison of the networks. They use the ground truth solver to calculate one scalar value marking performance: accuracy, rollout stability or conservation of physical quantities such as like energy or momentum.

For instance, specifically in the context of quantifying the stability of an autoregressive rollout, some works use *high-correlation times* — that is, the first autoregressive step when the Pearson correlation between the ground truth and the prediction drops below a threshold (e.g. 0.9) — as the target metric [Lip+23]. A Mean Squared Error threshold is also a typical candidate for such quantification, although manually setting a value for a boundless metric can be seen as too arbitrary.

In this section, we let $\mathcal{NN}$ be an autoregressive neural network operator, and $\mathcal{S}$ be the ground truth (numerical) solver operator such that

$$\mathcal{NN},\ \mathcal{S} : \mathbb{R}^N \to \mathbb{R}^N,$$

where it must be noted that since we later discretize the spatial domain, the continuous function $u(t,x)$ is represented at each time step by a vector $\boldsymbol{u_i} := \boldsymbol{u}(t_i) \in \mathbb{R}^N$ of its values at N grid points.

2.2.1 Pointwise Mean Squared Error

We define the *Pointwise Mean Squared Error (PwMSE)* metric as:

$$\mathrm{PwMSE}(\mathcal{NN}) = \frac{1}{M} \sum_{i=1}^{M} \mathrm{MSE}(\boldsymbol{u_i}, \boldsymbol{\hat{u}_i}),$$

where $\boldsymbol{u}$ and $\boldsymbol{\hat{u}}$ represent the ground truth and the network-predicted trajectories, respectively, both generated from the same initial value $\boldsymbol{u}_{\text{init}}$:

$$\begin{cases} \boldsymbol{u_0} = \boldsymbol{\hat{u}_0} =: \boldsymbol{u}_{\text{init}}, \\ \boldsymbol{u_{i+1}} = \mathcal{S}(\boldsymbol{u_i}), \\ \boldsymbol{\hat{u}_{i+1}} = \mathcal{NN}(\boldsymbol{\hat{u}_i}), \\ i \in [0, M-1], \end{cases} \tag{2.3}$$

and the mean squared error between two real vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^N$ is defined as

$$\mathrm{MSE}(\boldsymbol{a}, \boldsymbol{b}) = \frac{1}{N} \sum_{j=1}^{N} (a_j - b_j)^2. \tag{2.4}$$

However, for chaotic systems, the PwMSE metric may fail to adequately distinguish between high- and low-quality network predictions. Due to the inherent sensitivity of chaotic dynamics, even minor discretisation errors or numerical approximations will inevitably cause the predicted trajectory to diverge exponentially from the ground truth, regardless of the model's fidelity to the underlying physics. Consequently, PwMSE can misleadingly penalize otherwise accurate models due to this expected long-term divergence. A more robust evaluation metric is therefore needed, one that is less sensitive to trajectory separation while still capturing the model's ability to reproduce the system's dynamics.

2.2.2 Cumulative One-Step Mean Squared Error

The *Cumulative One-Step Mean Squared Error (COSMSE)* quantifies a neural network's ability to reproduce the local dynamics of a system over a rollout. For a network's rollout of length M, the metric sums the mean squared error (MSE) between the network's prediction at each time step and the state produced by applying the ground-truth solver to the previous network output.

Formally, for a rollout of length M, COSMSE is computed as:

$$\mathrm{COSMSE}(\mathcal{NN}) = \frac{1}{M-1} \sum_{i=1}^{M-1} \mathrm{MSE}\left(\hat{\boldsymbol{u}}_{\boldsymbol{i}}, \mathcal{S}(\hat{\boldsymbol{u}}_{\boldsymbol{i-1}})\right),$$

where $\hat{\boldsymbol{u}}_{\boldsymbol{i}}$ denotes the network's predicted state at step i, $\mathcal{S}(\hat{\boldsymbol{u}}_{\boldsymbol{i-1}})$ is the state obtained by applying the solver to the previous predicted state, and $\mathrm{MSE}(\cdot\,,\,\cdot)$ is defined in (2.4). Each term in the sum evaluates only the immediate, one-step discrepancy between the network's output and the solver's next step, without considering how far the predicted trajectory has drifted overall.

In simpler terms, COSMSE measures how well the network approximates the solver at *each individual step* along the network trajectory, rather than evaluating the pointwise accuracy of a full multi-step trajectory. Because each comparison resets the reference to the ground-truth solver applied to the network's own prior output, this metric isolates single-step prediction errors and does not accumulate discrepancies over time. As a result, COSMSE remains meaningful even in chaotic systems where small initial errors grow exponentially in full rollouts.

Unless stated otherwise, both reported PwMSE and COSMSE values are averaged across all domain sizes $\tilde{\mathcal{X}}$ in this work.

PwMSE

$$\begin{tikzcd}
& & u_0 \arrow[dl, "\mathcal{NN}"'] \arrow[dr, "\mathcal{S}"] & & \\
& \hat{u}_1 \arrow[dl, "\mathcal{NN}"'] \arrow[rr, leftrightarrow, "MSE"] & & u_1 \arrow[dr, "\mathcal{S}"] & \\
\hat{u}_2 \arrow[rrrr, leftrightarrow, "MSE"] & & & & u_2 \\
& & \vdots & &
\end{tikzcd}$$

COSMSE

$$\begin{tikzcd}
& & u_0 \arrow[dl, "\mathcal{NN}"'] \arrow[dr, "\mathcal{S}"] & \\
& \hat{u}_1 \arrow[dl, "\mathcal{NN}"'] \arrow[dr, "\mathcal{S}"] \arrow[rr, leftrightarrow, "MSE"] & & \mathcal{S}(u_0) \\
\hat{u}_2 \arrow[rr, leftrightarrow, "MSE"] & & \mathcal{S}(\hat{u}_1) & \\
& \vdots & &
\end{tikzcd}$$

Figure 2.2: A schematic of PwMSE and COSMSE over time steps. PwMSE compares the network's predictions $\hat{u}_i$ to a fixed ground-truth trajectory u_i; COSMSE compares each prediction $\hat{u}_i$ to the result of applying the true solver $\mathcal{S}$ to the previous prediction $\hat{u}_{i-1}$.

2.2.3 Comparison

Figure 2.2 contrasts COSMSE with the more conventional pointwise MSE (PwMSE) and shows schematically how they are computed.

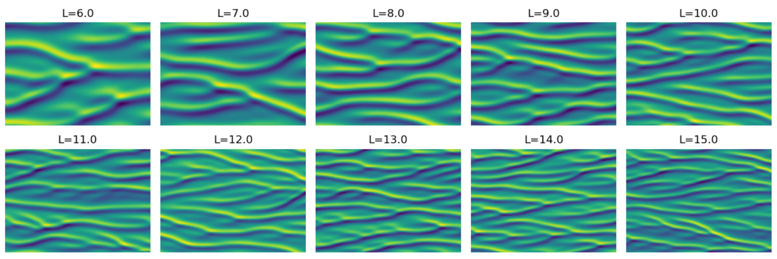
We now investigate how the two metrics behave for a poorly- and a well-trained model. Both models share the same architecture and initialization seed; however, the undertrained network was stopped after only 100 epochs (or 4000 optimisation steps), while the well-trained one was trained fully following the procedure described later in Section 3.4.

Figure 2.3 visually compares their predicted trajectories. The undertrained network performs worse, exhibiting non-physical drift and significant amplitude errors, whereas the well-trained network produces a trajectory that is qualitatively and statistically similar to the ground truth.

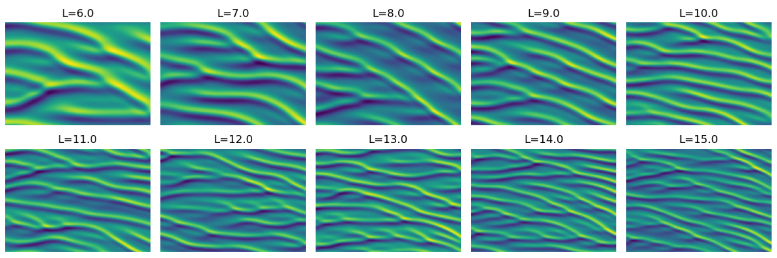
We further compare the evolution of both metrics across 80 autoregressive inference steps. Later in this work, unless stated otherwise, such metrics will always be evaluated across 80 steps.

The difference in model quality becomes most apparent when analysing how each metric evolves over time, as shown in Figure 2.4.

In the PwMSE metric (Figure 2.4a), both the well-trained and undertrained models show an increasing error over the rollout. This is expected in chaotic systems: small inaccuracies in early predictions grow exponentially, eventually leading to large de-



(a) Well-trained network



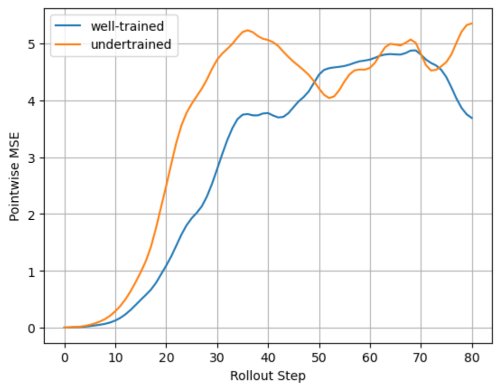
(b) Undertrained network

Figure 2.3: 80-step inference trajectories for the two networks. The undertrained network shows significant drift and amplitude shift.

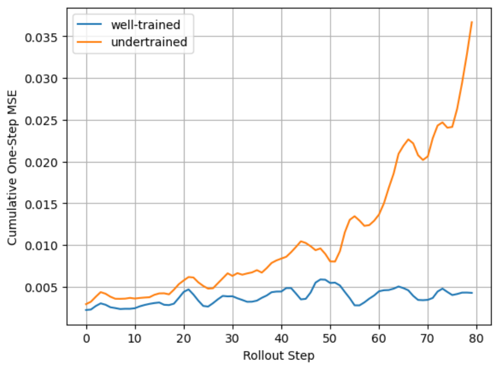
viations from the ground-truth trajectory. Importantly, PwMSE alone does not allow us to clearly distinguish between a model that accurately captures the system's local dynamics and one that does not. Both models appear to perform poorly over the full rollout, even though their underlying causes of error differ. Notably, the PwMSE of the undertrained model grows faster during the initial rollout steps, suggesting worse short-term predictive accuracy; however, as errors accumulate (after around 40 rollout steps), both models eventually exhibit similarly large deviations, making it difficult to distinguish them. One could introduce a threshold to focus only on this early, more informative phase, but doing so would require additional tuning and may not generalise across different systems or rollout lengths.

In contrast, the Cumulative One-Step Mean Squared Error (COSMSE) metric, illustrated in Figure 2.4b, shows a different pattern. For the undertrained model, the one-step errors grow significantly over time, indicating that the model struggles to reproduce the correct local dynamics even at a single prediction step. However, the well-trained model maintains a consistently *low and bounded* COSMSE throughout the rollout. This demonstrates that, despite the overall trajectory divergence visible in PwMSE, the well-trained model has successfully learned the system's local dynamics and is able to reproduce it even later in the rollout.

Thus, while PwMSE primarily reflects the inevitable accumulation of errors in chaotic systems, COSMSE isolates the model's true ability to capture the underlying dynamics, revealing whether it correctly models the system's local evolution and providing a clearer qualitative evaluation of neural solvers.



(a) Pointwise MSE



(b) Cumulative One-Step MSE

Figure 2.4: Evolution of error metrics over inference steps. COSMSE more clearly distinguishes between the well-trained and undertrained models, remaining bounded for the former. In contrast, PwMSE increases rapidly for both, making it less reliable for assessing model quality in chaotic regimes.

From Figure 2.4, it is clear that COSMSE provides a clearer separation between the two models in terms of their long-term predictive ability. These results are summarized in Table 2.1: the relative difference between the two models is higher for COSMSE, indicating a better sensitivity to model quality.

Table 2.1: Total error metrics over 80 inference steps.

Metric	Well-Trained	Undertrained	Relative Increase
PwMSE	2.946e1	3.591e1	1.22
COSMSE	3.746e-3	1.087e-2	2.90

3 Methodology

This chapter outlines the methodological framework used to train neural networks for the modelling of spatio-temporal dynamics (e.g. of the Kuramoto–Sivashinsky equation). The approach is data-driven, yet tightly coupled with the underlying physics.

We begin by detailing the construction of the dataset and the numerical solver used to generate reference trajectories. We then describe the neural architecture employed, its domain-awareness, and the optimization setup. A key part of our methodology is the use of unrolling, a training technique that encourages temporal stability by feeding model outputs as inputs over multiple steps. This is especially crucial for long-horizon prediction in chaotic autoregressive systems, where minor errors can quickly accumulate. Finally, we summarize the training procedure and infrastructure used for the experiments.

3.1 Dataset

Training data were generated by numerically simulating the KS equation using the second order Runge–Kutta numerical scheme [CM02] across a set of domain sizes L defined in (2.2).

For each domain L, the spatial interval $[0, 2\pi L]$ was discretised uniformly using $N = 96$ grid points:

$$x_i = \frac{2\pi L}{N} \cdot i, \quad i = 0, 1, \dots, N-1.$$

The ground truth trajectories were computed using an exponential second-order Runge–Kutta (RK2) integrator [CM02], chosen for its accuracy and efficiency in stiff PDEs.

A unique periodic initial condition was used for each domain, given by:

$$u_0(x) = \cos\left(\frac{x}{L}\right) + 0.2\cos\left(\frac{x}{L}\right)\left(1 - 2\sin\left(\frac{x}{L}\right)\right). \tag{3.1}$$

In each simulation, the first 20,000 burn-in steps were discarded to ensure that the retained data reflects the long-term behaviour on the chaotic attractor. This extended burn-in phase effectively eliminates the influence of the initial condition on the observed trajectories. Nevertheless, care must be taken to avoid choosing an initial condition

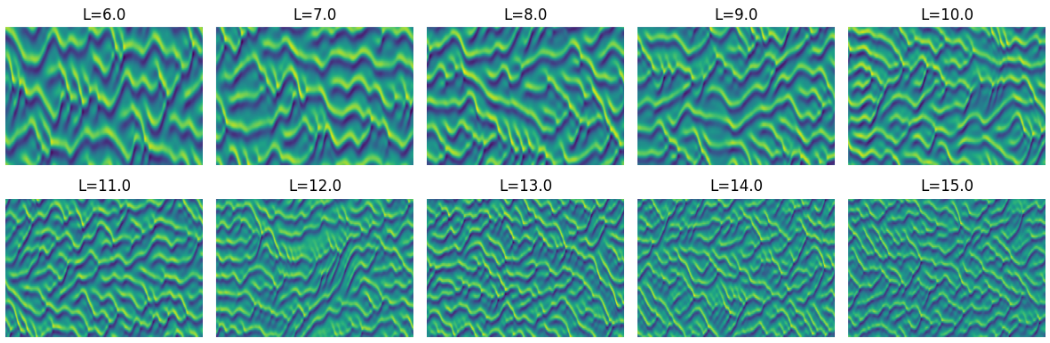


Figure 3.1: Spatio-temporal plot of the solver trajectories of the Kuramoto–Sivashinsky equation for $L \in \tilde{\mathcal{X}}$, 500 time steps each, with $\Delta t = 0.5$

from a pathological region of the function space (such as a basin of attraction for a fixed point or a non-chaotic periodic orbit) that would prevent the system from reaching the chaotic regime.

For training the domain-aware network (see Section 3.2 for details), we generated ten trajectories — one for each $L \in \tilde{\mathcal{X}}$ — with each trajectory consisting of 500 time steps after burn-in. An illustration of the resulting dataset is shown in Fig. 3.1.

3.2 Network Architecture

The final architecture emerged from iterative exploration. It builds upon a baseline of three circular 1D convolutional layers with ReLU activations and kernel size 7, designed to respect the PDE's translational invariance and periodic boundary conditions through circular padding. To enable generalisation across varying domain sizes, an additional input channel was introduced: a constant-valued vector encoding the domain size L, repeated across spatial positions. This modification allowed the network to adapt its computation to the geometry while maintaining the simplicity and efficiency of a fully convolutional structure. All subsequent experiments employed this domain-aware architecture, with hyperparameters summarized in Table 3.1.

The model was trained using the AdamW optimizer [KB17], which modifies Adam by decoupling the weight decay step from the adaptive gradient calculation for improved regularization [LH19]. Mean Squared Error (MSE) (defined in (2.4)), which is a common loss function in machine learning problems, was used to evaluate the difference between the two states.

Table 3.1: Network and optimization hyperparameters used in all further experiments

Hyperparameter	**Value**
Batch size	100
Hidden channels	32
Kernel size	7
Network depth (layers)	5
Optimizer	AdamW
Loss function	MSE
Learning rate	10^{-3}
Exponential scheduler decay factor (γ)	0.99
Early stopping patience	5
Weight decay	10^{-5}
Trainable parameters	22,304

3.3 What is Unrolling

In autoregressive simulators, training-time *unrolling* refers to applying the model recursively over several time steps during training, instead of optimising it only on single input-output pairs ("one-step"), as is common in standard autoregressive neural networks. Reflecting real-world deployment, this approach forces the network to learn predictions over longer horizons by feeding its own outputs back in. In the fully differentiable variant that we use in this work, gradients are propagated through the chain of model applications (backpropagation through time, or BPTT), allowing the loss accumulated over multiple steps to guide learning via gradient descent. Unrolled training has been shown to significantly improve the performance of neural simulators of dynamical systems, even amid chaotic regimes, by reducing distribution shift and improving long-range consistency [Lis+24; ST24; LCT22].

One formalizes the technique as follows. Let:

- $\boldsymbol{u}^{\boldsymbol{t}} \in \mathbb{R}^d$: the system state at time t
- $\mathcal{NN}_\theta : \mathbb{R}^d \to \mathbb{R}^d$: a neural network parametrized by θ
- $H \in \mathbb{N}$: unrolling horizon length
- $\mathcal{L}$: a differentiable loss function
- $(\boldsymbol{u}^{\boldsymbol{0}}, \boldsymbol{u}^{\boldsymbol{1}}, \dots, \boldsymbol{u}^{\boldsymbol{H}})$: a ground truth trajectory

Then, we define repeated application of the network as

$$\hat{\boldsymbol{u}}^{\mathbf{0}} := \boldsymbol{u}^{\mathbf{0}}, \quad \hat{\boldsymbol{u}}^{\boldsymbol{t+1}} := \mathcal{NN}_\theta(\hat{\boldsymbol{u}}^{\boldsymbol{t}}), \quad \forall t = 0, \dots, H-1.$$

To simplify our notation, let $\mathcal{NN}_\theta^{(t)}$ denote the composition of the model applied t times:

$$\mathcal{NN}_\theta^{(t)} := \underbrace{\mathcal{NN}_\theta \circ \cdots \circ \mathcal{NN}_\theta}_{t}.$$

Then, the total loss over a trajectory starting at u^0 is defined as

$$\mathcal{L}^{\text{unr}}(\theta) := \frac{1}{H}\sum_{t=1}^{H} \mathcal{L}(\hat{\boldsymbol{u}}^{\boldsymbol{t}}, \boldsymbol{u}^{\boldsymbol{t}}) = \frac{1}{H}\sum_{t=1}^{H} \mathcal{L}\left(\mathcal{NN}_\theta^{(t)}(\boldsymbol{u}^{\mathbf{0}}), \boldsymbol{u}^{\boldsymbol{t}}\right).$$

Given a dataset of N trajectories with the unrolling horizon H

$$\{\{\boldsymbol{u}_{(i)}^{\boldsymbol{t}}\}_{t=0}^{H}\}_{i=1}^{N},$$

the optimisation objective is given by

$$\theta^* := \underset{\theta}{\text{argmin}}\, \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{(i)}^{\text{unr}}(\theta) = \underset{\theta}{\text{argmin}}\, \frac{1}{NH}\sum_{i=1}^{N}\sum_{t=1}^{H} \mathcal{L}\left(\mathcal{NN}_\theta^{(t)}(\boldsymbol{u}_{(i)}^{\mathbf{0}}), \boldsymbol{u}_{(i)}^{\boldsymbol{t}}\right).$$

As a result, this helps networks maintain accuracy when making multi-step forecasts (e.g. weather or fluid simulations) and exposes the network to its own predictions during training, which prevents failures when small errors accumulate.

3.4 Optimisation Procedure

The training datasets consist of pairs of tensors (i.e. multi-dimensional arrays) representing system states at consecutive time steps, with an additional input channel encoding the domain size.

For the one-step version of the dataset (i.e. $H = 1$), each training sample is a tuple (x, y) where:

- $x \in \mathbb{R}^{\mathrm{N}\times 2}$: the input state at time t, concatenated with a constant channel encoding the domain size, and transposed to shape $(\mathrm{N}, 2)$;
- $y \in \mathbb{R}^{1\times \mathrm{N}}$: the target state at time $t+1$.

For unrolling with horizon H, each sample becomes a tuple (x, Y), where:

- $x \in \mathbb{R}^{N \times 2}$: the initial state of the trajectory window, again domain-size encoded and transposed;
- $Y \in \mathbb{R}^{H \times N}$: a stack of H ground-truth system states following x, without any additional encoding, used for computing multi-step losses against the model's autoregressive outputs.

Here and in the following, $N = 96$, which denotes the spatial resolution — i.e. the number of discretization points in the domain for representing the system state.

All models were trained using a standardised procedure, terminating when either of the following conditions was met: (1) the number of training epochs reached 700, or (2) an early stopping criterion was triggered. The early stopping mechanism monitored the test loss over a sliding window of epochs. If the relative change in test loss across a number of last epochs (the so-called patience) fell below a fixed threshold — specifically, if the average relative change between five consecutive values was less than $5 \cdot 10^{-5}$ — training was terminated. This approach ensured computational efficiency without sacrificing model performance, especially in settings where convergence occurred well before the epoch limit. Later, however, during the experiments involving guided automatic unrolling horizon switches in Chapter 5.3, the early stopping criterion was disabled to allow uninterrupted adaptation across horizon changes.

All numerical experiments were carried out using PyTorch — a high-performing, GPU-accelerated machine learning Python framework [Pas+17; Pas+19]. They were run on NVIDIA GTX 1080 GPUs provided by the university, using a custom CLI-based training script for reproducible and automated execution.

4 Mitigating Instabilities from Switching the Unrolling Horizons

When the training curriculum switches from a one-step prediction (with the unrolling horizon length $H = 1$) to a multi-step, unrolled prediction, the training error often exhibits sudden increase. This phenomenon may arise because the optimisation task changes abruptly: during the one-step phase, the model learns to minimize the error of a single prediction based on ground-truth inputs, and when the horizon is extended, the model's own (likely still inaccurate) outputs are fed back as inputs for multiple consecutive steps — this increase in the complexity of the loss landscape can lead to large, destabilizing gradient updates, resulting in the observed error spike.

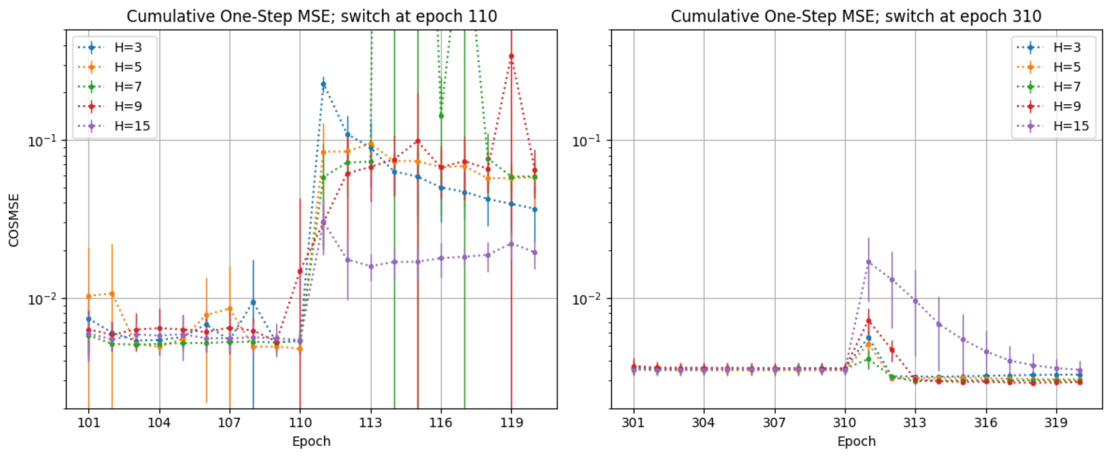


Figure 4.1: Effect of Changing Unrolling Horizon on COSMSE: transitions to different unrolling horizon lengths H at epochs 110 and 310. Extending the one-step phase reduces switch instabilities and facilitates recovery.

Furthermore, this training instability persists even when common optimisation hyperparameters are adjusted. The following sections describe our attempts to mitigate this error jump by modifying Adam's β_1 parameter, applying gradient clipping and implementing the learning rate cutting and scheduling.

Note that the initial training phase using a one-step horizon is essential for achieving

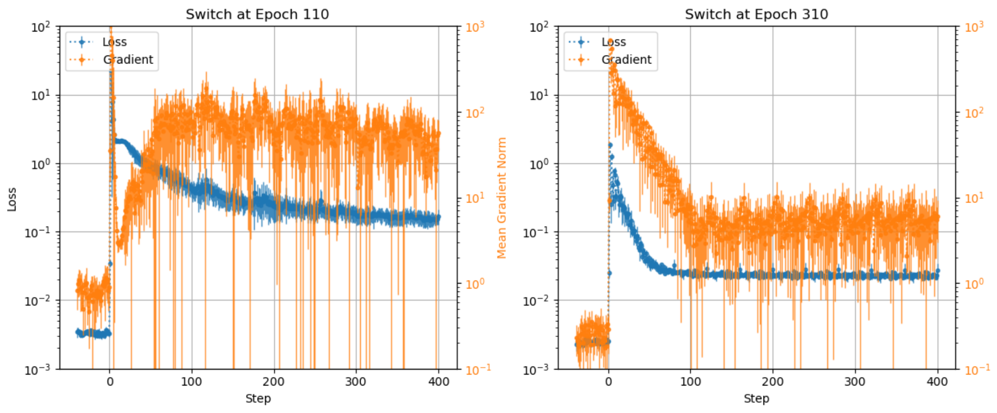


Figure 4.2: Loss and gradient norm explode immediately after horizon switches.

strong final performance in all cases. As shown in Figure 4.1, switching earlier (left) introduces instabilities regardless of the unrolling horizon, whereas delaying the switch (extending the one-step training phase) yields steady improvements in the COSMSE (right). The figure also illustrates how the new unrolling horizon length influences the network's ability to adapt to the new objective, with the increased errors and subsequent recovery. Moreover, Figure 4.2 illustrates how both the test loss and the gradient norm increase following the switch, often pushing the model into an unrecoverable state. Notably, this effect is less pronounced when the switch occurs later, at epoch 310 rather than at epoch 110, suggesting that a sufficiently long one-step phase is necessary to ensure a stable transition.

In this section's further experiments, unless stated otherwise, each curve represents an average and standard deviation of a metric over 5 initialisation seeds. The "switch" refers to the change of the unrolling horizon length H from 1 to a fixed value of 7.

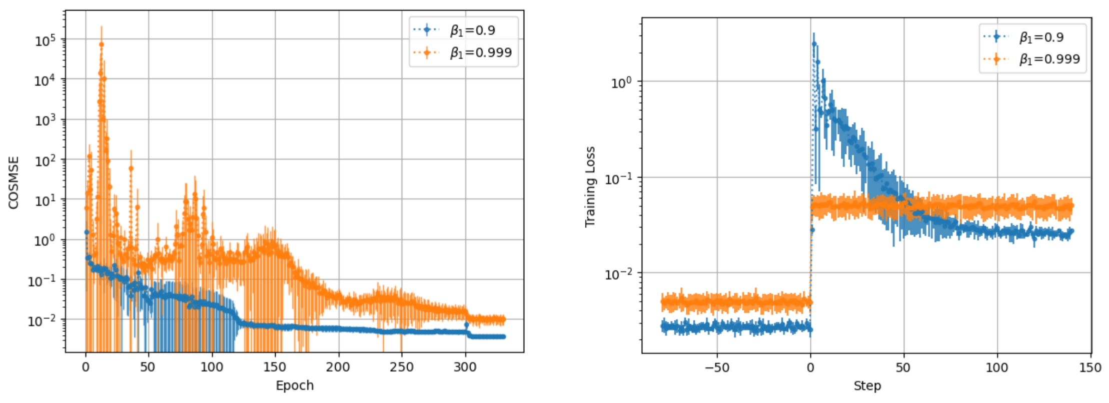
4.1 Momentum in Adam Optimizer

To prevent error explosions following the switch, we experimented with modifying the Adam optimizer's β_1 parameter, which controls the extent to which past gradients influence the current momentum estimate: higher values of β_1 retain past momentum for longer and slow down the update dynamics [KB17]. The intuition was that increasing β_1 (that is, bringing it closer to 1) could potentially dampen the destabilizing gradient update responsible for the blow-up.

We first tested a static modification, where β_1 remains fixed throughout the entire training run. Figure 4.3 compares two such scenarios: $\beta_1 = 0.9$ (default) and $\beta_1 = 0.999$.

Panel 4.3a shows the evolution of the COSMSE metric across epochs, and Panel 4.3b zooms into the training loss immediately before and after the switch.

In Figure 4.3a, we observe that a larger β_1 significantly worsens generalisation performance, as evidenced by the higher and more erratic COSMSE values throughout training. In contrast, Figure 4.3b demonstrates that increasing β_1 does succeed in mitigating the sharp increase in training loss directly after the switch, suggesting a dampening effect on the instability. However, this gain in stability appears to come at the cost of model quality, as the loss fails to decrease further and the COSMSE remains high.



(a) COSMSE

(b) Training Loss

Figure 4.3: Stepwise Training Loss and COSMSE for Adam's $\beta_1 \in \{0.9, 0.999\}$; the β_1 modification is applied throughout the entire training.

Given these results, a natural next step is to restrict the modification of β_1 to the switch point itself. That is, training is conducted with the default value (e.g., $\beta_1 = 0.9$) up until the switch, and a different $\beta_1 \in \{0.99, 0.999\}$ is applied starting exactly at the switch step.

As shown in Figure 4.4, this isolated adjustment has no significant effect on the training error dynamics: the spike in loss immediately after the switch still occurs. This suggests that simply altering β_1 at the switch step is insufficient to mitigate the destabilisation, likely due to a mismatch between accumulated momentum and the sudden change in optimisation dynamics.

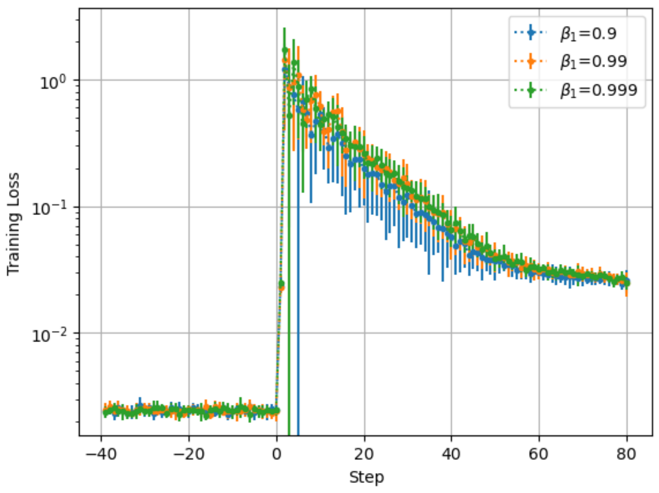


Figure 4.4: Stepwise Training Loss for Adam's $\beta_1 \in \{0.9, 0.99, 0.999\}$; here β_1 is only modified from the switch onward.

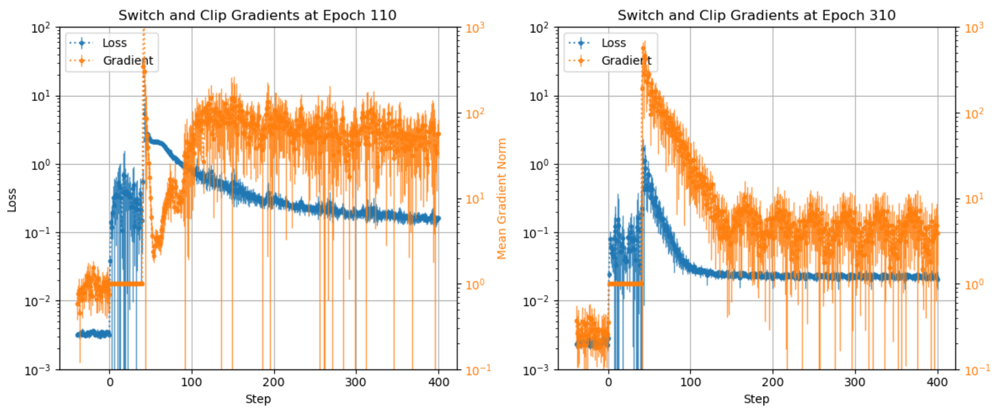
4.2 Gradient Clipping

To counteract the error spike that consistently follows an increase in the unrolling horizon, we applied gradient clipping immediately after the switch: specifically, the gradient norm was clipped to a maximum of 1 during a short post-switch window. Experiments were performed with switches at epochs 110 and 310, corresponding to the left and right panels of Figure 4.5, respectively. Once again, the earlier switch leads to a more violent gradient explosion and slower recovery, underscoring that the training trajectory is more fragile at earlier epochs. The x-axis denotes optimizer steps, with step 0 aligned to the switch point for both cases.

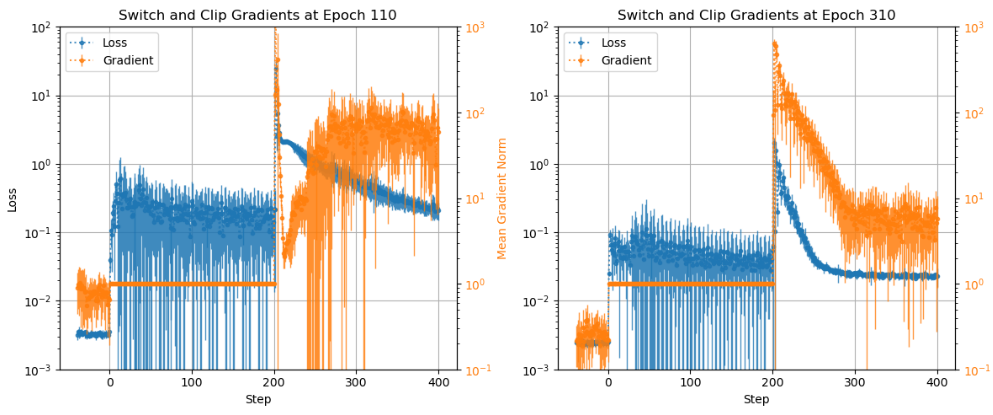
We begin with clipping gradients for just one epoch after the switch (Figure 4.5a). While this temporarily suppresses the gradient explosion, it fails to prevent the subsequent loss spike, and the instability re-emerges as soon as clipping is removed.

To test whether a longer stabilization period helps, we extend clipping to five post-switch epochs (Figure 4.5b). This delays the onset of instability, but does not eliminate it. After clipping ends, the gradients rapidly regain high magnitudes and the loss escalates, mirroring the unmitigated scenario.

These results suggest that gradient clipping does not resolve the underlying instability triggered by the horizon switch. The optimisation trajectory appears to retain memory of the transition, and clipping alone, regardless of duration, cannot fully absorb the resulting shock.



(a) Clipping to norm 1 during the first post-switch epoch; explodes once clipping is removed.



(b) Clipping for 5 post-switch epochs; failure is delayed but still occurs after clipping ends.

Figure 4.5: Effect of Gradient Clipping on Training Stability with respect to the Training Loss After Increasing the Unrolling Horizon.

4.3 Learning Rate Cutting

To mitigate the gradient explosion that occurs immediately after increasing the unrolling horizon, we also experimented with reducing the learning rate by a factor of 10 (i.e., `lr := lr * 0.1`) at the moment of the switch.

Figure 4.6 shows the results for a fixed curriculum consisting of 300 epochs of 1-step training, followed by 30 epochs at 7 steps, and then 15-step training thereafter. Two configurations were compared: one with the standard learning rate throughout, and another with a tenfold reduction applied immediately before the first horizon switch. The zoomed-in plots confirm that lowering the learning rate successfully prevents the immediate post-switch error blow-up. However, this modification did not lead to a meaningful long-term improvement in the evaluation metric (COSMSE), calling into question whether the associated slowdown in training is justified.

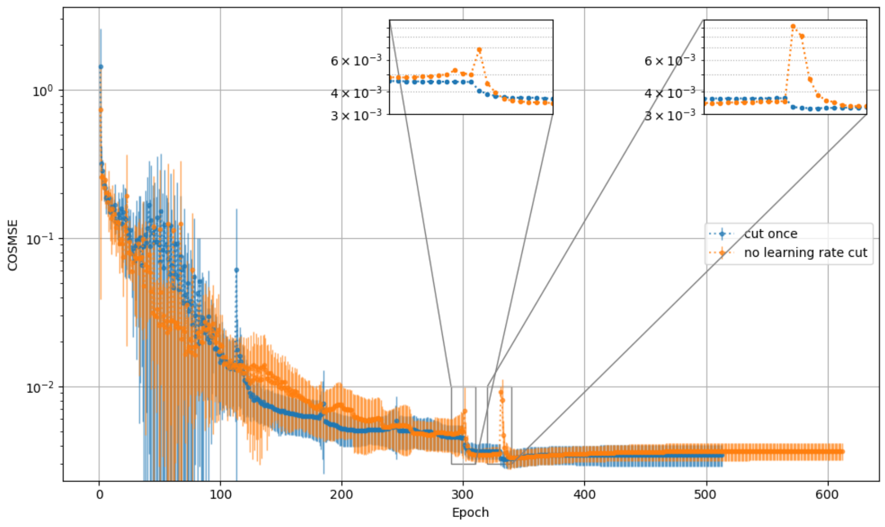


Figure 4.6: Learning Rate Reduction at First Curriculum Switch: Error Blow-up.

To further explore the trade-off between stability and training speed, we hypothesised that gradually restoring the learning rate to its original value after the switch might retain the stabilising effect of the initial reduction while eventually allowing faster convergence. This approach did indeed suppress the large gradient and loss spike seen immediately after the switch. However, as shown in Figure 4.7, even a slight increase in the learning rate during this recovery phase led to worse generalisation on the COSMSE metric. In other words, although the model remained relatively numerically stable during training, it converged to solutions with poorer performance. This suggests that aggressive learning rate reduction is effective for preventing training instability, but resuming higher learning rates (even gradually) may push the optimiser out of stable regions: either into poor local minima or into regimes where it can no longer adapt effectively to the longer unrolling horizon.

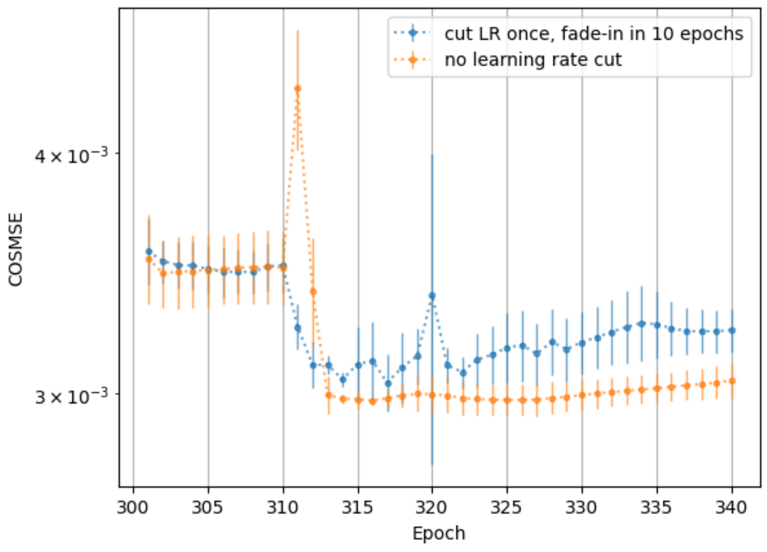


Figure 4.7: Learning Rate Fade-in: Effect on Generalisation. Learning rate mitigates the error blow up but deteriorates further learning.

4.4 Mitigating Instabilities: Conclusion

In this chapter, we investigated the training instabilities that arise when switching from a single-step to a multi-step unrolling horizon. The abrupt change in the optimisation objective consistently triggers a sharp increase in training error and gradient magnitudes, often destabilising the model. Our experiments demonstrated that a longer initial phase of one-step training is crucial for building a robust foundation, making the subsequent transition to a longer horizon smoother. We explored several common techniques to mitigate the post-switch instability, including modifying the Adam optimizer's momentum, applying gradient clipping, and reducing the learning rate.

Adjusting Adam's β_1 parameter to increase momentum did dampen the initial error spike, but at the significant cost of overall generalisation performance.

Similarly, applying gradient clipping provided only a temporary measure: the instability and error explosion re-emerged as soon as the clipping was removed, indicating that it does not address the underlying cause.

Cutting the learning rate at the switch point proved to be the most effective method for preventing the immediate error blow-up. However, this stability came with a trade-off, as the model's long-term performance, measured by COSMSE, did not improve and sometimes worsened, especially when attempting to gradually restore the original learning rate.

Ultimately, while these methods can suppress the symptoms of instability, they do not offer a complete solution. This suggests that the core issue lies in the fundamental shift in the loss landscape, which is not easily overcome by simple adjustments to the optimisation algorithm.

5 Guiding Unrolling Curricula

The preceding chapter demonstrated that common optimisation techniques are insufficient to resolve the instabilities arising from manually switching the unrolling horizon. This highlights a critical need for a more principled approach to structuring the training process. We therefore return to the primary objective of this thesis: to automatise the training curriculum by developing a method that determines both when to increase the unrolling horizon and what the new horizon length, H, should be. Such a method requires a guiding metric that reliably signals the model's readiness to handle longer, unrolled predictions.

Our initial hypothesis centred on using a metric that reflects the intrinsic chaotic timescale of the dynamics being learned. The maximal Lyapunov exponent (LE), which quantifies a system's sensitivity to initial conditions, appeared to be a natural candidate. A model that has accurately learned the system's LE might be presumed ready for a longer prediction horizon. However, our experiments revealed the LE to be an impractical guide for training. We observed that the network's LE converges to the true value very early in training, long before the model's predictive performance is adequate.

Given the failure of the LE as a reliable indicator, our focus shifted to a metric more directly aligned with the training objective: the Cumulative One-Step Mean Squared Error (COSMSE). Unlike the LE, COSMSE provides a continuous and stable measure of the model's predictive accuracy, making it a more robust proxy for its rollout behaviour. While the ultimate goal is to automatise both the timing and the length of the horizon switch, we simplify the problem for now. In the remainder of this chapter, we use a fixed sequence of horizon lengths and investigate the effectiveness of COSMSE as a data-driven trigger for deciding precisely *when* to advance the curriculum from one stage to the next.

5.1 Candidate Functions for Guiding Unrolling Horizon Switches

5.1.1 Lyapunov Exponent

The Lyapunov exponent (LE) of a dynamical system characterises the rate of separation between two initially nearby trajectories. Given an initial separation vector $\vec{\delta}_0$, the distance between the trajectories evolves approximately according to:

$$|\vec{\delta}(t)| \approx e^{\lambda t} |\vec{\delta}_0|, \tag{5.1}$$

where λ is the Lyapunov exponent (see Figure 5.1). It is a fundamental tool for analysing complex and chaotic dynamical systems [PP16].

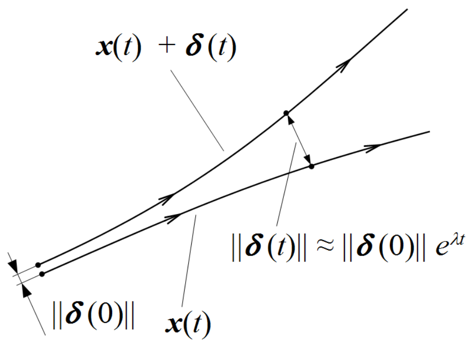

$\boldsymbol{x}(t) + \boldsymbol{\delta}(t)$
$\|\boldsymbol{\delta}(t)\| \approx \|\boldsymbol{\delta}(0)\| \, e^{\lambda t}$
$\|\boldsymbol{\delta}(0)\|$
$\boldsymbol{x}(t)$

Figure 5.1: A schematic illustrating the definition of the Lyapunov Exponent. Source: Yapparina (2015). CC0. [Yap15] `https://commons.wikimedia.org/wiki/File:Orbital_instability_(Lyapunov_exponent).png`

Since the rate of separation may depend on the orientation of the initial separation vector, the maximal Lyapunov exponent, the largest exponent across all possible orientations, is typically used as the primary indicator of a system's chaoticity.

For a discrete-time operator $\mathcal{F}$ evolving as $\boldsymbol{u}_{i+1} = \mathcal{F}(\boldsymbol{u}_i)$, the maximal Lyapunov exponent λ quantifies the average exponential growth rate of small perturbations and can be estimated using the Benettin algorithm [Pch20]:

$$\lambda = \frac{1}{T} \sum_{t=1}^{T} \ln \|\gamma^t\| \tag{5.2}$$

Here, $\gamma^t := J^t_{\mathcal{F}} \boldsymbol{v}^{t-1}$ is the evolved perturbation at time t, and $J^t_{\mathcal{F}} := D\mathcal{F}(\boldsymbol{u}^{t-1})$ is the Jacobian of $\mathcal{F}$ at $\boldsymbol{u}^{t-1}$, and $\boldsymbol{v}^0$ is an initial unit-norm perturbation. After each step, γ^t is normalised to produce $\boldsymbol{v}^t := \gamma^t / \|\gamma^t\|$ and to ensure numerical stability.

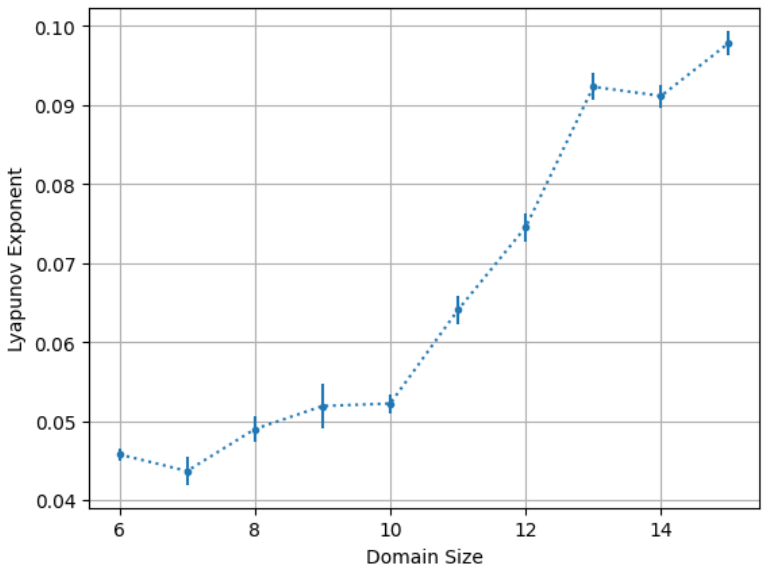


Figure 5.2: The relationship between domain size and the solver's Lyapunov Exponent for the Kuramoto–Sivashinsky equation, averaged over $n = 10$ simulations. LE is larger for bigger systems.

In essence, the maximal LE of an autoregressive operator quantifies the average exponential rate at which nearby trajectories diverge. This value reveals how sensitive the system is to its initial state and thus how quickly its complex dynamics become unpredictable. For the Kuramoto–Sivashinsky equation, for instance, a larger physical domain size corresponds to a higher maximal LE, indicating that larger systems can support more complex and chaotic dynamics (Figure 5.2; see also Section 2.1 and Figure 2.1). In the following experiments, the maximal LE is estimated over 500 steps after discarding an initial transient period of 100 steps.

Despite its theoretical appeal, Figure 5.3 demonstrates that the LE is not a suitable metric for guiding the training curriculum. The network's LE converges to the solver's reference value within approximately 10 epochs, yet the model itself remains severely undertrained. This rapid convergence creates a dangerously misleading signal of model readiness. For example, Figure 5.4 shows an 80-step rollout from a model after just 10 one-step epochs. While its LE matches the target, its rollout exhibits non-physical artefacts and its COSMSE is poor (0.09, compared to approx. 0.002 for a well-trained model).

In addition to this fundamental misalignment with predictive accuracy, the LE has practical drawbacks. Its estimation via the Benettin algorithm imposes significant

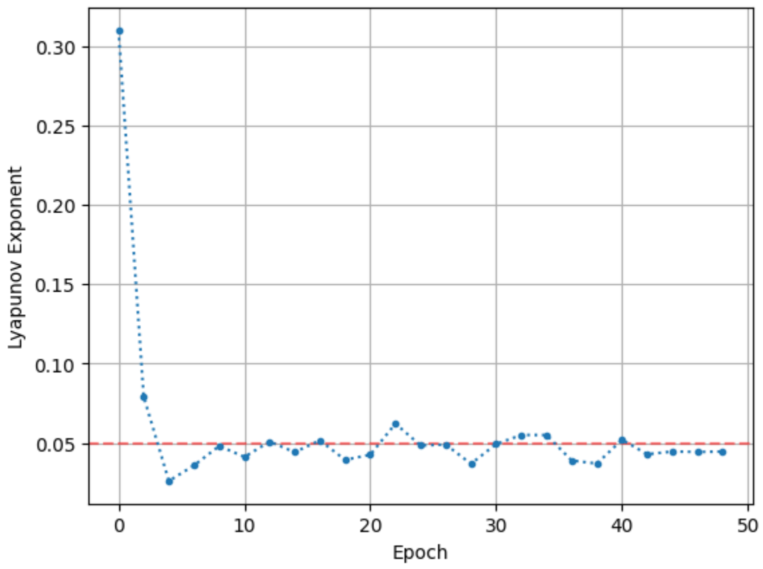


Figure 5.3: The network's maximal Lyapunov exponent during training for a domain of size $L = 8.0$. The horizontal red line at $y \approx 0.053$ indicates the reference LE of the numerical solver.

computational overhead, making frequent evaluation during training impractical. Furthermore, the LE calculation exhibits high variance and sensitivity to noise, which complicates analysis and further undermines its reliability as a curriculum scheduler.

5.1.2 Cumulative One-Step Mean Squared Error

As defined in Section 2.2.2, the Cumulative One-Step Mean Squared Error (COSMSE) measures the model's average one-step prediction error across an entire trajectory. In stark contrast to the Lyapunov exponent, COSMSE offers a practical and reliable metric for guiding the training curriculum for several reasons.

First, unlike the LE, which measures an abstract dynamical property, COSMSE directly quantifies the model's predictive error — the very quantity the training process seeks to minimise. This ensures it is fundamentally aligned with the model's objective. Second, it is computationally efficient to evaluate, allowing for frequent monitoring with minimal overhead. Third, its characteristic behaviour after a horizon switch — a sharp increase followed by a gradual recovery as the model adapts — provides a clear and interpretable signal of the model's state. This drop-and-recovery pattern directly reflects the model's struggle and subsequent adaptation to a more difficult predictive task. Finally, by aggregating error over full trajectories, COSMSE is stable and normally

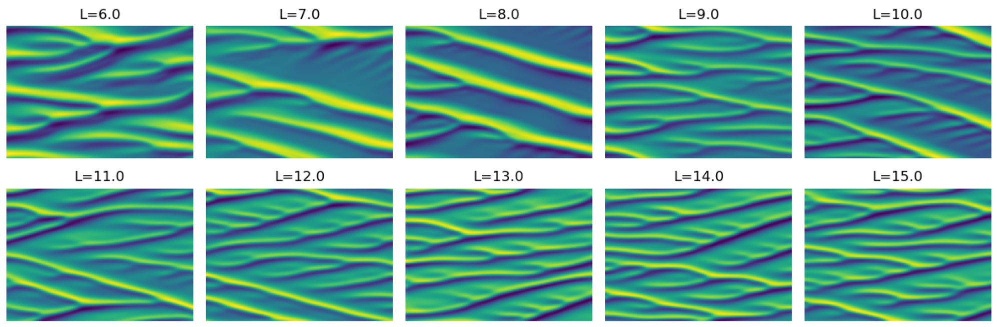


Figure 5.4: Spatiotemporal plot of a rollout from a network trained for only 10 one-step epochs. Although its LE has converged, the rollout shows clear non-physical amplitude shifts and drift artefacts, indicating an undertrained model.

exhibits low variance across runs, making it a robust indicator of the training progress. These properties make it a far more suitable guide for deciding when to advance the curriculum.

5.2 Fixing Unrolling Horizon Lengths

While COSMSE is well-suited for tracking training progress and deciding when to switch horizons, it is less effective for determining the next horizon length itself. The optimal choice of H depends on factors beyond immediate predictive error, such as long-term stability or task-specific temporal scales, which COSMSE alone does not capture. Therefore, to isolate the problem of timing, we predefine a fixed sequence of unrolling horizon lengths and use COSMSE solely to decide when to transition between them.

We consider the following manually defined curriculum configurations, with experimental results shown in Figure 5.5:

1. A fixed one-step setup, where no unrolling is applied throughout training ($H = 1$).
2. A stepwise progression with horizon lengths $H \in \{1, 5, 15, 25\}$, with switches occurring after 300, 100, and 100 epochs, respectively.
3. A geometric progression with horizon lengths $H \in \{1, 3, 9, 27\}$, using the same epoch schedule as above.

These experiments verify that training autoregressive neural operators without any unrolling produces models that, although stable in one-step predictions, perform worse when generating long, accurate rollouts. In-axis windows of the Figure 5.5 also show that, in contrast to the error spikes caused by premature switching, the error *drops* slightly after each switch. In the remainder of this chapter, we explore how this observation can be leveraged to design a scheduling strategy that minimizes the resulting COSMSE error.

While certain curricula clearly lead to superior models, the process of manually selecting both the horizon sequence (e.g., $1-5-15-25$ vs $1-3-9-27$) and the phase durations is tedious. To streamline the development of an automatic scheduler, we standardise the unrolling sequence to $1-5-15-25$, which proved effective from this numerical experiment, and focus on the new primary challenge: allowing the scheduler to automatically determine the optimal moment to switch to the next length in this sequence.

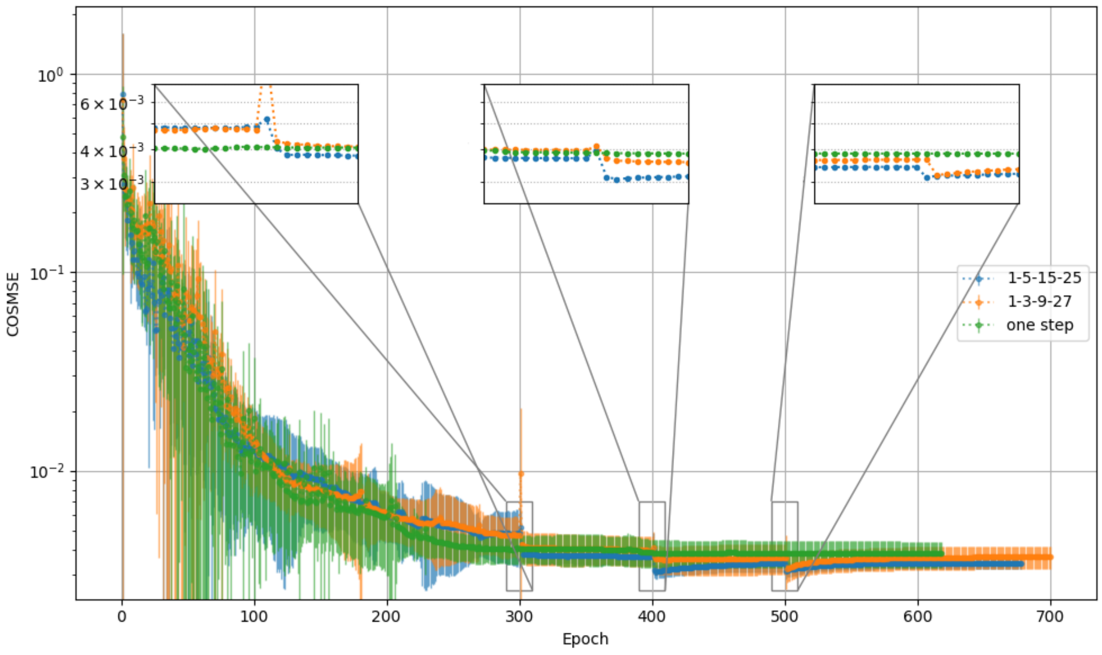


Figure 5.5: Comparison of COSMSE for two fixed unrolling curricula and a constant one-step training regime. The results are averaged over $n = 10$ runs.

5.3 Automatic Unrolling Scheduler

A manually defined curriculum requires extensive trial-and-error to find a schedule that is neither too aggressive nor too conservative. Fixed schedules risk switching too early, before the model has adapted, or too late, wasting computation on a learning plateau or risking to overtrain.

To address this, we propose a scheduler that adapts to the learning dynamics by monitoring COSMSE and triggering transitions based on statistically grounded signals of performance stagnation or degradation. This creates a responsive and robust curriculum progression without manually tuned switch points. The scheduler's decision is based on the following set of conditions, evaluated periodically (at the end of each epoch) during training. Let $\{e_k\}_{k=-10}^{-1}$ represent the sequence of the last ten recorded COSMSE values, where e_{-1} is the most recent. A switch to the next horizon is triggered if and only if all four of the following conditions are met:

1. The initial one-step training phase of at least $E_{\min}$ epochs has been completed (Initial Training Phase).
2. A "patience" period of at least P epochs (counting starts after $E_{\min}$ epochs) has elapsed since the current unrolling horizon was activated (Patience Period).
3. The predefined sequence of unrolling horizons has not yet been exhausted (Curriculum Progression).
4. At least one of the following two sub-conditions on the COSMSE sequence e is met (Switch Conditions Due to Degradation):
   (a) The sequence e is monotonically increasing.
   (b) The most recent value e_{-1} shows a significant relative increase of more than 3% compared to ten epochs prior (i.e., $e_{-1}/e_{-10} > 1.03$).

The rationale for this design is to advance the model to a more challenging horizon once it shows signs of saturation or degradation on the current one. Condition 1 ensures the model builds a stable foundation on simple one-step predictions, a necessity established in Chapter 4. Condition 2 enforces a minimum adaptation period, preventing premature switches before the model has had a reasonable time to converge on the current task. Condition 3 simply ensures the scheduler follows the predefined curriculum path (horizon lengths). Finally, Condition 4 acts as the core trigger: a consistently worsening error (4a) or a sharp, significant decline in performance (4b) indicates that further training on the current horizon is becoming counter-productive. Together, these conditions guide the model along a challenging but stable learning trajectory. For these experiments, standard training termination criteria (Section 3.4) are disabled; training concludes once the final horizon in the sequence is completed.

This scheduler introduces two key hyperparameters: the minimum one-step training length $E_{\min}$ and the patience P. We fix $E_{\min} = 200$ epochs and focus our analysis on P, as it directly controls the scheduler's responsiveness. Because the number of parameter updates per epoch depends on the dataset size, we express patience in terms of optimiser steps, $\tilde{P}$, where one epoch corresponds to 40 steps ($\tilde{P} = 40P$).

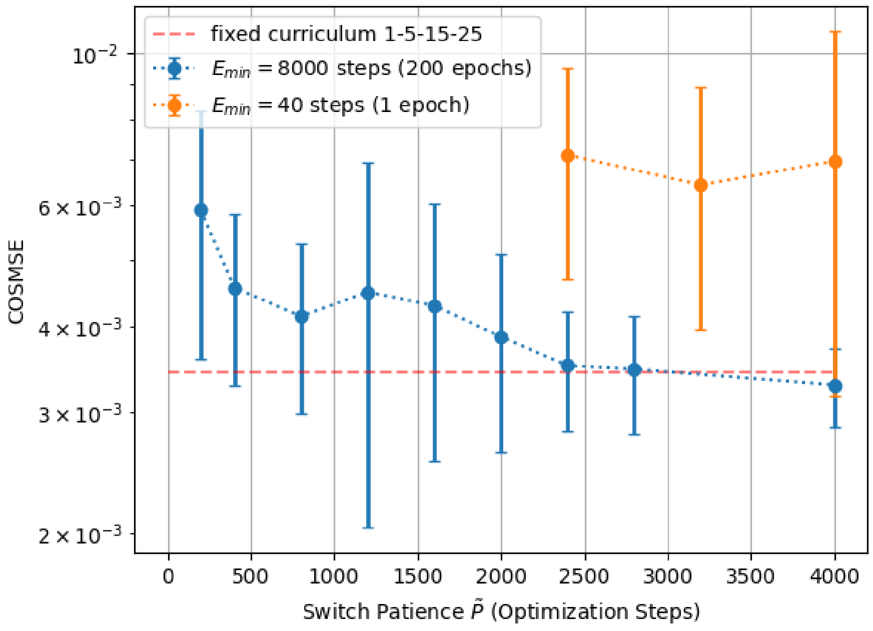


Figure 5.6: The effect of varying switch patience $\tilde{P}$ (measured in optimizer steps) on the final COSMSE. The red dashed line indicates the final COSMSE achieved by the fixed curriculum $1-5-15-25$, which is comparable to the performance of our scheduler. Values represent the mean and standard deviation across multiple runs. Small patience values result in greater error variance, while increasing patience beyond a certain point yields diminishing improvements in loss.

As shown in Figure 5.6, increasing the switch patience $\tilde{P}$ generally reduces the final COSMSE, though with diminishing returns. This suggests that giving the model more time to adapt at each stage leads to better overall performance. Moreover, the large error bars for smaller $\tilde{P}$ values indicate that short patience can lead to premature, unstable switching and inconsistent outcomes. In contrast, higher patience values yield more reliable convergence, albeit at the cost of a slower curriculum. For practical application, one would select the smallest patience $\tilde{P}^*$ at the "elbow" of this curve, balancing training efficiency with model performance (here, $\tilde{P}^* = 2000$ steps or $P^* = 50$ epochs).

Finally, we tested the hypothesis that the scheduler could function without a mandated one-step phase by setting $\tilde{E}_{\text{min}} = 40$ steps ($E_{\text{min}} = 1$ epoch), effectively removing it. The results were conclusive: for reasonably large patience values, this configuration produced a final COSMSE approximately an order of magnitude higher than with a fixed 8000-step (200-epoch) one-step phase (also shown in Figure 5.6). It also showed much greater variance across runs. This result invalidates the hypothesis that the scheduler can operate without a foundational one-step phase and reinforces a core conclusion from our earlier experiments: an initial period of dedicated one-step training is indispensable for achieving stable and high-performing autoregressive models.

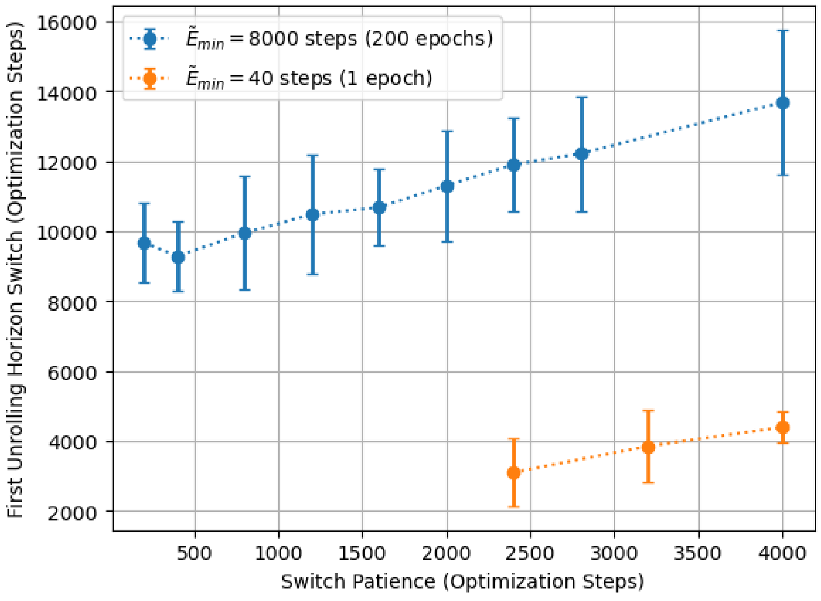


Figure 5.7: First unrolling horizon switch vs. switch patience, shown for a minimal one-step phase of $E_{\text{min}} = 1$ and $E_{\text{min}} = 200$ epochs, respectively. When training begins with virtually no one-step phase ($E_{\text{min}} = 1$), the switch happens prematurely, immediately following $\tilde{P}$ steps. In contrast, a substantial one-step phase ($E_{\text{min}} = 200$) delays the switch, allowing the network more time to adapt to the one-step learning objective.

Furthermore, Figure 5.7 illustrates how the timing of the first unrolling switch depends on the patience $\tilde{P}$ and the initial one-step training length E_{min}. In its current form, the scheduler initiates the switch almost immediately after the combined threshold $E_{\text{min}} + P$ is reached, indicating that its behaviour is tightly coupled to these hyperparameters. However, this also highlights a potential direction for refinement: the scheduler could be modified to decouple its switching logic from strict epoch counts and instead rely more heavily on the dynamics of the observed error signal. For example, adaptive or smoothed statistical thresholds could be used to delay or accelerate switching based on actual learning behaviour rather than fixed time windows. Such modifications would make the scheduler more flexible and responsive, reducing the need for careful tuning of E_{min} and $\tilde{P}$.

To summarize, in this chapter, we explored candidate metrics for guiding the training curriculum of unrolling-based models and found that the maximal Lyapunov exponent, despite its theoretical appeal, fails as a practical signal for model readiness. In contrast, the Cumulative One-Step Mean Squared Error (COSMSE) emerges as a stable and well-interpretable metric that enables data-driven decisions about when to advance the unrolling horizon. By using COSMSE to guide unrolling horizon switches during training, we achieved results comparable to manual curriculum selection (see Figure 5.6), which typically relies on intuition, experience, and a deeper understanding of the underlying physical system.

6 Conclusion

This thesis has explored new approaches for training neural networks to solve chaotic partial differential equations (PDEs), with the Kuramoto–Sivashinsky (KS) equation as a challenging and representative test case. Solving chaotic PDEs with machine learning is particularly difficult due to their sensitivity to initial conditions, instability over long time horizons, and rich dynamical structures. Despite these challenges, we demonstrated that neural networks can learn to approximate the system's dynamics when supported by careful training design and evaluation strategies. This work contributes three main ideas to this effort.

First, we introduced the Cumulative One-Step Mean Squared Error (COSMSE) metric to evaluate a neural network's ability to model the short-term dynamics of a chaotic system. Unlike traditional rollout-based metrics such as pointwise MSE, which compare a predicted trajectory to a fixed reference and are highly sensitive to chaotic divergence, COSMSE auto-regressively evaluates the model's local accuracy: at each time step, it compares the model's prediction to the next state obtained by applying a numerical solver to the previous prediction. This provides a more stable and informative signal for training and evaluation, as it isolates the model's ability to replicate the system's dynamical characteristics without being penalised for inevitable long-term divergence.

Second, we analysed the instabilities that arise during training phase transitions — particularly when increasing the unrolling horizon in curriculum learning. Our findings show that error spikes during these transitions are a fundamental feature of learning in chaotic systems, not only artifacts of poor hyperparameter choices. These spikes occur because the model, initially trained on short rollouts, struggles to generalise to longer horizons where error accumulation becomes significant. Recognising and managing these instabilities is essential for stable training.

Third, we designed and tested an automatic scheduling algorithm that switches the unrolling horizon during training based on model performance. This scheduler enables the training process to gradually increase in difficulty without manual tuning. By using COSMSE as a guide, the system adapts when the model becomes sufficiently accurate at shorter horizons, allowing it to gradually progress to more challenging ones. We showed that this automated approach achieves performance comparable to manually tuned curricula while removing the need for heuristic schedule design. Most notably, the automatic scheduler comes close to the effectiveness of manually defined curricula while reducing development effort and improving training reproducibility.

However, this study has limitations. All experiments were conducted on the KS equation, a canonical but low-dimensional chaotic PDE. While the techniques presented here are likely applicable to other chaotic or stiff systems, their effectiveness on more complex problems — such as turbulent Navier-Stokes flows, multi-scale systems, or high-dimensional PDEs with conservation laws — remains to be demonstrated. Generalisation beyond this setting will require careful adaptation and further empirical validation.

This thesis opens several promising directions for future research. One key avenue is investigating how optimal curriculum schedules relate to the system's intrinsic timescales, such as Lyapunov exponents or decorrelation times. These indicators could potentially be used not only to guide the timing of unrolling horizon transitions, but also to adapt the magnitude of each step, dynamically adjusting how much the unrolling length increases based on the system's local stability properties. Another is to evaluate the proposed methods on PDEs with physical constraints (e.g., mass or energy conservation), or with more intricate geometries. In addition, building theoretical tools to explain the success of specific training strategies could inform the design of more general and principled learning frameworks for scientific systems.

In summary, this work addresses fundamental challenges in training neural networks for chaotic PDEs by developing better evaluation metrics and introducing strategies to manage model adaptability. The techniques developed here provide a solid foundation for further work on training neural solvers of dynamical systems — particularly those that exhibit chaotic behaviour. Broader application and deeper theoretical analysis will be key steps toward reliable neural surrogates in scientific computing.

Source Code and Data

The source code, data generation pipelines, and trained model weights associated with this thesis are publicly available on GitLab:
`https://gitlab.lrz.de/00000000014B568C/masters-thesis.git`.

List of Figures

List of Tables

Bibliography

[Ask+23] H. Askr, E. Elgeldawi, H. Aboul Ella, Y. A. M. M. Elshaier, M. M. Gomaa, and A. E. Hassanien. "Deep learning in drug discovery: an integrative review and future challenges". In: *Artificial Intelligence Review* 56.7 (2023), pp. 5975–6037. ISSN: 1573-7462. DOI: `10.1007/s10462-022-10306-1`.

[BHW22] J. C. Baez, S. Huntsman, and C. Weis. *The Kuramoto-Sivashinsky Equation*. 2022. arXiv: `2210.01711 [math.AP]`.

[Bot+25] T. Botarelli, M. Fanfani, P. Nesi, and L. Pinelli. "Using Physics-Informed neural networks for solving Navier-Stokes equations in fluid dynamic complex scenarios". In: *Engineering Applications of Artificial Intelligence* 148 (2025), p. 110347. ISSN: 0952-1976. DOI: `https://doi.org/10.1016/j.engappai.2025.110347`.

[BWW23] J. Brandstetter, D. Worrall, and M. Welling. *Message Passing Neural PDE Solvers*. 2023. arXiv: `2202.03376 [cs.LG]`.

[CB22] P.-Y. Chuang and L. A. Barba. *Experience report of physics-informed neural networks in fluid simulations: pitfalls and frustration*. 2022. arXiv: `2205.14249 [physics.flu-dyn]`.

[Che+20] Y.-C. Chen, C. Shi, J. M. Kosterlitz, X. Zhu, and P. Ao. "Global potential, topology, and pattern selection in a noisy stabilized Kuramoto–Sivashinsky equation". In: *Proceedings of the National Academy of Sciences* 117.38 (2020), pp. 23227–23234. DOI: `10.1073/pnas.2012364117`. eprint: `https://www.pnas.org/doi/pdf/10.1073/pnas.2012364117`.

[CM02] S. Cox and P. Matthews. "Exponential Time Differencing for Stiff Systems". In: *Journal of Computational Physics* 176.2 (2002), pp. 430–455. ISSN: 0021-9991. DOI: `https://doi.org/10.1006/jcph.2002.6995`.

[Doe+21] S. Doerr, M. Majewski, A. Pérez, A. Krämer, C. Clementi, F. Noe, T. Giorgino, and G. De Fabritiis. "TorchMD: A Deep Learning Framework for Molecular Simulations". In: *Journal of Chemical Theory and Computation* 17.4 (2021). PMID: 33729795, pp. 2355–2363. DOI: `10.1021/acs.jctc.0c01343`. eprint: `https://doi.org/10.1021/acs.jctc.0c01343`.

[Eds+19] R. A. Edson, J. E. Bunder, T. W. Mattner, and A. J. Roberts. *Lyapunov exponents of the Kuramoto-Sivashinsky PDE*. 2019. arXiv: `1902.09651 [math.DS]`.

[FDC20] M. Frank, D. Drikakis, and V. Charissis. "Machine-Learning Methods for Computational Science and Engineering". In: *Computation* 8 (Mar. 2020), p. 15. DOI: `10.3390/computation8010015`.

[GKK24] H. Gao, S. Kaltenbach, and P. Koumoutsakos. "Generative learning for forecasting the dynamics of high-dimensional complex systems". In: *Nature Communications* 15.1 (2024). Published 2024 Oct 16, p. 8904. DOI: `10.1038/s41467-024-53165-w`.

[GZW22] H. Gao, M. J. Zahr, and J.-X. Wang. "Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems". In: *Computer Methods in Applied Mechanics and Engineering* 390 (2022), p. 114502. ISSN: 0045-7825. DOI: `https://doi.org/10.1016/j.cma.2021.114502`.

[Jin+25] T. Jin, G. Maierhofer, K. Schratz, and Y. Xiang. *A fast neural hybrid Newton solver adapted to implicit methods for nonlinear dynamics*. 2025. arXiv: `2407.03945 [math.NA]`.

[KB17] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: `1412.6980 [cs.LG]`.

[Koc+21] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. "Machine learning–accelerated computational fluid dynamics". In: *Proceedings of the National Academy of Sciences* 118.21 (May 2021). ISSN: 1091-6490. DOI: `10.1073/pnas.2101784118`.

[LCT22] B. List, L.-W. Chen, and N. Thuerey. "Learned turbulence modelling with differentiable fluid solvers: physics-based loss functions and optimisation horizons". In: *Journal of Fluid Mechanics* 949 (Sept. 2022). ISSN: 1469-7645. DOI: `10.1017/jfm.2022.738`.

[LH19] I. Loshchilov and F. Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: `1711.05101 [cs.LG]`.

[Lip+23] P. Lippe, B. S. Veeling, P. Perdikaris, R. E. Turner, and J. Brandstetter. *PDE-Refiner: Achieving Accurate Long Rollouts with Neural PDE Solvers*. 2023. arXiv: `2308.05732 [cs.LG]`.

[Lis+24] B. List, L.-W. Chen, K. Bali, and N. Thuerey. *Differentiability in Unrolled Training of Neural Physics Simulators on Transient Dynamics*. 2024. arXiv: `2402.12971 [physics.comp-ph]`.

[LWX21] C. Lv, L. Wang, and C. Xie. *A hybrid physics-informed neural network for nonlinear partial differential equation*. 2021. arXiv: `2112.01696 [math.NA]`.

[NST85] B. Nicolaenko, B. Scheurer, and R. Temam. "Some global dynamical properties of the Kuramoto-Sivashinsky equations: Nonlinear stability and attractors". In: *Physica D: Nonlinear Phenomena* 16.2 (1985), pp. 155–183. ISSN: 0167-2789. DOI: `https://doi.org/10.1016/0167-2789(85)90056-9`.

[OAH20] S. Okuno, K. Aihara, and Y. Hirata. "Forecasting high-dimensional dynamics exploiting suboptimal embeddings". In: *Scientific Reports* 10 (Jan. 2020), p. 664. DOI: `10.1038/s41598-019-57255-4`.

[ÖM25] E. Özalp and L. Magri. "Stability analysis of chaotic systems in latent spaces". In: *Nonlinear Dynamics* 113 (Feb. 2025), pp. 13791–13806. DOI: `10.1007/s11071-024-10712-w`.

[Pas+17] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. *Automatic differentiation in PyTorch*. 2017.

[Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. arXiv: `1912.01703 [cs.LG]`.

[Pch20] A. N. Pchelintsev. "An Accurate Numerical Method and Algorithm for Constructing Solutions of Chaotic Systems". In: *Journal of Applied Nonlinear Dynamics* 9.2 (June 2020), pp. 207–221. ISSN: 2164-6473. DOI: `10.5890/jand.2020.06.004`.

[Pen+23] T. Peng, W. Wu, J. Liu, L. Li, J. Miao, X. Hu, R. He, and L. Li. "PGN-Cloth: Physics-based graph network model for 3D cloth animation". In: *Displays* 80 (2023), p. 102534. ISSN: 0141-9382. DOI: `https://doi.org/10.1016/j.displa.2023.102534`.

[Pfa+21] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia. *Learning Mesh-Based Simulation with Graph Networks*. 2021. arXiv: `2010.03409 [cs.LG]`.

[PP16] A. Pikovsky and A. Politi. *Lyapunov Exponents: A Tool to Explore Complex Dynamics*. Cambridge University Press, 2016.

[PWM22] B. Pachev, J. P. Whitehead, and S. A. McQuarrie. “Concurrent MultiParameter Learning Demonstrated on the Kuramoto–Sivashinsky Equation”. In: *SIAM Journal on Scientific Computing* 44.5 (Sept. 2022), A2974–A2990. ISSN: 1095-7197. DOI: 10.1137/21m1426109.

[Raz+21] T. M. Razakh, B. Wang, S. Jackson, R. K. Kalia, A. Nakano, K.-i. Nomura, and P. Vashishta. “PND: Physics-informed neural-network software for molecular dynamics applications”. In: *SoftwareX* 15 (2021), p. 100789. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2021.100789.

[RK17] M. Raissi and G. Karniadakis. “Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations”. In: (Aug. 2017). DOI: 10.48550/arXiv.1708.00588.

[RPK19] M. Raissi, P. Perdikaris, and G. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2018.10.045.

[Rue+20] L. von Rueden, S. Mayer, R. Sifa, C. Bauckhage, and J. Garcke. *Combining Machine Learning and Simulation to a Hybrid Modelling Approach: Current and Future Directions*. Ed. by M. R. Berthold, A. Feelders, and G. Krempl. Cham, 2020.

[SPP24] F. Sahli Costabal, S. Pezzuto, and P. Perdikaris. “Delta-PINNs: Physics-informed neural networks on complex geometries”. In: *Engineering Applications of Artificial Intelligence* 127 (2024), p. 107324. ISSN: 0952-1976. DOI: https://doi.org/10.1016/j.engappai.2023.107324.

[ST24] P. Schnell and N. Thuerey. *Stabilizing Backpropagation Through Time to Learn Complex Physics*. 2024. arXiv: 2405.02041 [cs.LG].

[Toh87] S. Toh. “Statistical Model with Localized Structures Describing the Spatio-Temporal Chaos of Kuramoto-Sivashinsky Equation”. In: *Journal of the Physical Society of Japan* 56.3 (1987), pp. 949–962. DOI: 10.1143/JPSJ.56.949. eprint: https://doi.org/10.1143/JPSJ.56.949.

[Uni48] United Nations. *Universal Declaration of Human Rights, Preamble*. Available at https://www.un.org/en/about-us/universal-declaration-of-human-rights. 1948.

[Wan+20] R. Wang, K. Kashinath, M. Mustafa, A. Albert, and R. Yu. *Towards Physics-informed Deep Learning for Turbulent Flow Prediction*. 2020. arXiv: 1911.08655 [physics.comp-ph].

[Yap15] Yapparina. *Orbital instability (Lyapunov exponent)*. Wikimedia Commons. Creative Commons Zero (CC0) License. Available at `https://commons.wikimedia.org/wiki/File:Orbital_instability_(Lyapunov_exponent).png`. 2015.