Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 415 (2023) 116204
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Adaptive learning of effective dynamics for online modeling of complex systems

Ivica Kičić[a], Pantelis R. Vlachas[a,b], Georgios Arampatzis[a,b], Michail Chatzimanolakis[a,b], Leonidas Guibas[c], Petros Koumoutsakos[b,*]

[a] *Computational Science and Engineering Laboratory, ETH Zürich, CH-8092, Switzerland*
[b] *School of Engineering and Applied Sciences, Harvard University, 29 Oxford Street, Cambridge, MA 02138, USA*
[c] *Department of Computer Science, Stanford University, 353 Serra Hall, CA 94035, USA*

Received 4 April 2023; received in revised form 9 June 2023; accepted 21 June 2023
Available online xxxx

Dataset link: https://github.com/cselab/adaled

Abstract

Predictive simulations are essential for applications ranging from weather forecasting to material design. The veracity of these simulations hinges on their capacity to capture the effective system dynamics. Massively parallel simulations predict the systems dynamics by resolving all spatiotemporal scales, often at a cost that prevents experimentation. On the other hand, reduced order models are fast but often limited by the linearization of the system dynamics and the adopted heuristic closures. We propose a novel systematic framework that bridges large scale simulations and reduced order models to extract and forecast adaptively the effective dynamics (AdaLED) of multiscale systems. AdaLED employs an autoencoder to identify reduced-order representations of the system dynamics and an ensemble of probabilistic recurrent neural networks (RNNs) as the latent time-stepper. The framework alternates between the computational solver and the surrogate, accelerating learned dynamics while leaving yet-to-be-learned dynamics regimes to the original solver. AdaLED continuously adapts the surrogate to the new dynamics through online training. The transitions between the surrogate and the computational solver are determined by monitoring the prediction accuracy and uncertainty of the surrogate. The effectiveness of AdaLED is demonstrated on three different systems - a Van der Pol oscillator, a 2D reaction–diffusion equation, and a 2D Navier–Stokes flow past a cylinder for varying Reynolds numbers (400 up to 1200), showcasing its ability to learn effective dynamics online, detect unseen dynamics regimes, and provide net speed-ups. To the best of our knowledge, AdaLED is the first framework that couples a surrogate model with a computational solver to achieve online adaptive learning of effective dynamics. It constitutes a potent tool for applications requiring many computationally expensive simulations.

Keywords: Adaptive reduced-order modeling; Computer simulations; Machine learning; Online real-time learning; Continuous learning; Navier–Stokes equations

* Corresponding author.
E-mail addresses: ivicakicic@gmail.com (I. Kičić), pvlachas@ethz.ch (P.R. Vlachas), garampat@ethz.ch (G. Arampatzis), michaich@ethz.ch (M. Chatzimanolakis), guibas@cs.stanford.edu (L. Guibas), petros@seas.harvard.edu (P. Koumoutsakos).

https://doi.org/10.1016/j.cma.2023.116204
0045-7825/

1. Introduction

Simulations of complex systems have transformed our predictive capabilities in areas ranging from health and epidemiology [1] to physics [2], meteorology [3], and fluid mechanics. Large-scale, simulations are prominent in fields where experiments may be unavailable, such as astrophysics and climate sciences, or where they require expensive infrastructure, equipment, and personnel.

The predictive fidelity of the simulations depends on their capacity to resolve all relevant spatiotemporal scales of the physical phenomenon under study. However, high fidelity implies high computational cost, which hinders experimentation and optimization. Many scientific and engineering tasks, such as parameter and design optimization [4], multi-objective optimization [5], reinforcement learning (RL) [6–8], and high-throughput computing [9], require a large number of system evaluations. While using highly accurate simulations for these tasks is desirable, it can also be cost-prohibitive and potentially infeasible. As a result, numerous research efforts have focused on developing accurate and efficient surrogate models that can replicate and accelerate simulations.

While computationally costly simulations are essential in resolving all scales of a complex system, key quantities of interest can be often described by a coarse-grained, averaged behavior. Selecting the proper degrees of freedom for such coarse grained representations is a long standing problem in science and engineering. Furthermore, appropriate combinations of coarse-grained and fine-scale simulations, predictions offer the potential for accelerated simulations at a controlled accuracy. Pioneering hybrid methods include the Equation-Free Framework (EFF) [10–12], the Heterogeneous Multiscale Method (HMM) [13,14], and the FLow AVeraged integratoR (FLAVOR) [15]. Hybrid methods distinguish between a detailed high-dimensional physical space (*micro scale*) which is expensive to simulate, and a coarse-grained, reduced-order, or latent space (*macro scale*). More specifically, in EFF, a system is first advanced in the expensive micro-scale for a given time. Then, a transition is made into the macro scale using a compression mechanism, such as Principal Component Analysis, Dynamic Mode Decomposition [16], or diffusion maps [17]. EFF then employs time-stepping schemes such as Euler or Runge–Kutta to advance the macro-scale dynamics. After several time steps, the macro-scale dynamics are mapped back onto the fine scale for detailed simulation. By alternating between the micro-scale and the macro-scale dynamics at timescales of interest, EFF can achieve significant computational savings. However, the generalization of EFF to complex high-dimensional systems has been limited by the proper information transfer between micro and macro and the use of inefficient macro-scale propagator.

In recent years there have been numerous efforts to develop reduced-order models and accelerate complex simulations using machine learning (ML) [18–23]. In a previous works, we extended the EFF with ML algorithms that learn the time integrators and the transfer operators in a data-driven manner. The resulting framework of Learning the Effective Dynamics (LED) [24] of complex dynamical systems employs convolutional autoencoders (CAEs) for the identification of the micro-to-macro and macro-to-micro mappings and recurrent neural networks (RNNs) to propagate the macro dynamics. The autoencoder and the RNN are trained offline using data from the micro propagator, i.e., the original simulator. LED has been applied to a variety of dynamical systems [25], from fluid flows to molecular simulations [26].

We note that frameworks related to LED include the Latent Evolution of Partial Differential Equations (LE-PDE) [27]. Meanwhile, CAEs coupled with Long Short-Term Memory networks (LSTMs) have been applied in modeling complex flows [28–34]. Other autoencoders (AEs), coupled with LSTM networks have been employed for surrogate modeling of high-dimensional dynamical systems [35], e.g., unsteady flows over a circular cylinder [36,37], or structural modeling of a two-story building [38]. Other notable works are based on Proper Orthogonal Decomposition (POD) [39–43], local approximations with POD [44,45], Dynamic Mode Decomposition (DMD) [16], and Dynamics Identification (ID) [46,47]. Adaptive extensions that utilize low-rank updates are proposed in [48,49]. In [50,51], the authors propose online adaptive versions of the DMD algorithm.

However, to the best of our knowledge, the above mentioned frameworks do not entail one or more of the following characteristics:

1. They lack continuous training, which limits their ability to adapt to changing dynamics or to generalize to regions underrepresented in the initial training data.
2. They do not quantify prediction uncertainty or monitor prediction error.
3. They overlook that a surrogate should only be used when it is reliable.
4. They do not exploit the capability of restarting a computer simulation from any time point.

5. They do not control the balance between speed-up and accuracy.
6. They do not account for variations in parameters of the fine-scale dynamics.

Although multiple works present robust surrogate modeling frameworks, the need for real-time applications and tasks involving non-stationary dynamics or state-space exploration calls for continual learning frameworks to address the problem of distribution shift [52].

Adaptivity and continual learning in forming surrogates are key components of the present paper. The proposed method of *Adaptive Learning of Effective Dynamics* (AdaLED) accelerates computationally expensive simulations by learning an online surrogate model that replaces the original simulator only when its predictions are sufficiently reliable. More specifically, AdaLED extends the LED framework with uncertainty quantification. The surrogate monitors its prediction accuracy and provides a confidence level for its predictions. In turn, the surrogate is utilized only in state-space regions (parts of the trajectories) that it has learned and is confident about its prediction. Otherwise, the computational solver is employed to simulate dynamics unknown to the surrogate model. Finally, we extend LED with continuous learning capabilities to address the distribution shift problem in environments/dynamical systems with time-varying dynamics.

We demonstrate that AdaLED can adaptively learn complex dynamics, produce reliable surrogate model predictions, and accelerate computationally expensive simulations while maintaining high accuracy. AdaLED provides control over the accuracy-speed trade-offs by adaptively specifying error thresholds for the simulation.

The paper is organized as follows: in Section 2, we present a detailed description of the AdaLED framework while in Sections 3 and 4, we demonstrate the efficiency and efficacy of AdaLED on the Van der Pol oscillator and a 2D reaction–diffusion system, respectively. In Section 5, we show how AdaLED can accelerate a 2D Navier–Stokes simulation of flow past a cylinder at varying Reynolds numbers. Section 6 concludes the paper. Technical information on the neural networks and handling of very high-dimensional fluid flow states are provided in the appendix.

2. Method

We consider the evolution of a dynamical system at a micro/fine scale, with a state denoted by $\mathbf{x}_t \in \mathbb{R}^{d_x}$ at time t. The state is advanced by δt using a *micro propagator* $\mathcal{F}$, so that

$$\mathbf{x}_{t+\delta t} = \mathcal{F}(\mathbf{x}_t, \mathbf{f}_t), \tag{1}$$

where $\mathbf{f}_t \in \mathbb{R}^{d_f}$ is the time-varying external forcing that affects the dynamics. Apart from $\mathbf{x}_t$, the simulation provides access to quantities of interest denoted by $\mathbf{q}_t \in \mathbb{R}^{d_q}$, $\mathbf{q}_t = \mathcal{Q}(\mathbf{x}_t)$. We postulate that the effective system dynamics can be approximated by lower-dimensional latent states $\mathbf{z}_t \in \mathcal{M}_z$, where $\mathcal{M}_z \subset \mathbb{R}^{d_z}$ (with $d_z \ll d_x$) is a low-order manifold of the system state space. Here, we identify the latent space by employing an encoder $\mathcal{E}^{\boldsymbol{\theta}_\mathcal{E}} : \mathbb{R}^{d_x} \to \mathbb{R}^{d_z}$ with trainable parameters $\boldsymbol{\theta}_\mathcal{E}$. The encoder maps micro states $\mathbf{x}_t$ to latent (macro) states $\mathbf{z}_t = \mathcal{E}^{\boldsymbol{\theta}_\mathcal{E}}(\mathbf{x}_t)$. In the other direction, a decoder $\mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}} : \mathbb{R}^{d_z} \to \mathbb{R}^{d_x}$, with trainable parameters $\boldsymbol{\theta}_\mathcal{D}$, maps the latent state $\mathbf{z}_t$ to the micro state $\tilde{\mathbf{x}}_t = \mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\mathbf{z}_t)$. The optimal parameters $\boldsymbol{\theta}^*_\mathcal{E}$ and $\boldsymbol{\theta}^*_\mathcal{D}$ minimize an application-specific reconstruction loss $\ell(\mathbf{x}_t, \tilde{\mathbf{x}}_t)$:

$$(\boldsymbol{\theta}^*_\mathcal{E}, \boldsymbol{\theta}^*_\mathcal{D}) = \underset{\boldsymbol{\theta}_\mathcal{E}, \boldsymbol{\theta}_\mathcal{D}}{\arg\min}\, \ell\left(\mathbf{x}_t, \tilde{\mathbf{x}}_t\right) = \underset{\boldsymbol{\theta}_\mathcal{E}, \boldsymbol{\theta}_\mathcal{D}}{\arg\min}\, \ell\left(\mathbf{x}_t, \mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\mathcal{E}^{\boldsymbol{\theta}_\mathcal{E}}(\mathbf{x}_t))\right). \tag{2}$$

A non-linear macro propagator $(\mathcal{H}^{\boldsymbol{\theta}_M}, \mathcal{Z}^{\boldsymbol{\theta}_M}, \mathcal{Q}^{\boldsymbol{\theta}_M}, \mathcal{S}^{\boldsymbol{\theta}_M})$, with parameters $\boldsymbol{\theta}_M$ and an internal hidden state $\mathbf{h}_t$ capturing non-Markovian effects, is trained to predict the system dynamics in the macro scale:

$$\mathbf{h}_{t+\Delta t} = \mathcal{H}^{\boldsymbol{\theta}_M}(\mathbf{z}_t, \mathbf{q}_t, \mathbf{f}_t, \mathbf{h}_t), \quad \tilde{\mathbf{z}}_{t+\Delta t} = \mathbf{z}_t + \mathcal{Z}^{\boldsymbol{\theta}_M}(\mathbf{h}_{t+\Delta t}), \quad \tilde{\mathbf{q}}_{t+\Delta t} = \mathbf{q}_t + \mathcal{Q}^{\boldsymbol{\theta}_M}(\mathbf{h}_{t+\Delta t}). \tag{3}$$

where Δt is the time step of the macro propagator, with Δt being an integer multiple of δt. The macro propagator is trained with backpropagation through time [53] to minimize the combined mean square error (MSE) loss $\|\tilde{\mathbf{z}}_{t+\Delta t} - \mathbf{z}_{t+\Delta t}\| + \|\tilde{\mathbf{q}}_{t+\Delta t} - \mathbf{q}_{t+\Delta t}\|$. Optional weights can be added to each loss component to control their relative importance.

We note that the present framework also predicts physical quantities of interest while evolving the latent space dynamics. Such quantities of interest $\tilde{\mathbf{q}}_{t+\Delta t}$ could be computed by reverting to the fine-scale representation of the system dynamics, i.e., $\tilde{\mathbf{q}}_{t+\Delta t} = \mathcal{Q}(\mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\tilde{\mathbf{z}}_{t+\Delta t}))$. However, this approach has two drawbacks: (i) it requires evaluation of the relatively expensive decoder $\mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}$ (ii) the function $\mathcal{Q}$ might not be explicitly available. Consequently, the

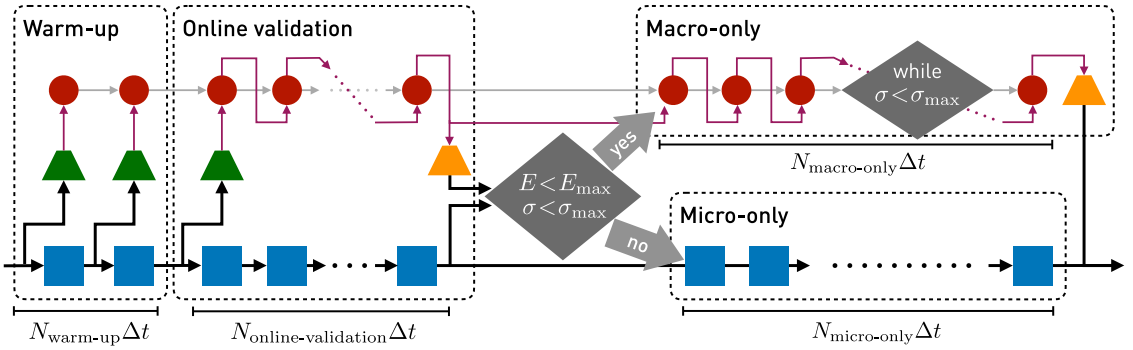

Fig. 1. The stages of the inference, the AdaLED cycle. ■ denotes the micro propagator, ● the macro propagator, ▲ the encoder, ▲ the decoder, **black** line the micro (high-dimensional) state $\mathbf{x}_t$, purple the macro (latent) state $\mathbf{z}_t$, and gray the hidden state $\mathbf{h}_t$ of the macro propagator. Quantities of interest $\mathbf{q}_t$ and external forcing $\mathbf{f}_t$ are hidden for brevity. Depending on the prediction error E_t of the model and uncertainty σ_t at the end of the online validation stage, either the macro-only or the micro-only stage is performed. The macro-only stage is performed as long as the uncertainty is below the threshold (possibly for 0 steps) or limited to a given number of steps. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

macro propagator is trained to predict $\tilde{\mathbf{q}}_{t+\Delta t}$ directly. In addition, the macro propagator outputs the uncertainty $\sigma_{t+\Delta t} \in \mathbb{R}$, i.e.

$$\sigma_{t+\Delta t} = \mathcal{S}^{\boldsymbol{\theta}_M}(\mathbf{h}_{t+\Delta t}). \tag{4}$$

The uncertainty is used to robustly control the transitions between the micro and the macro propagator, as explained later in the text.

To achieve significant acceleration of the simulations, the macro propagator operates with a time step $\Delta t \gg \delta t$. Here, the encoder and decoder are the two halves of a convolutional autoencoder, and the macro propagator is an ensemble of probabilistic recurrent neural networks (PRNNs) (see Section 2.3). We note that AdaLED can incorporate various encoders and decoders and accommodate any macro propagator that can estimate the uncertainty of its own predictions. We will refer to the combination of the encoder, decoder, and propagator as the Machine-Learned Model (MLM).

2.1. AdaLED cycle (inference)

Inference in AdaLED proceeds in an iterative fashion. In each iteration, AdaLED assesses the accuracy of the MLM and temporarily shifts the simulation from the micro to the macro scale if the accuracy is sufficiently high. This alternation between the scales allows the micro propagator to correct errors introduced by the MLM and guide the simulation back to the manifold $\mathcal{M}_z$. Additionally, the short cycles enable the MLM to replace the simulation in sections of trajectories that it has learned so far. Other sections that are underrepresented in the training data or require more extended training are left to the micro propagator. Finally, frequent evaluation of the micro propagator also enables continuous gathering of training data.

Each computational cycle in AdaLED consists of three stages: (i) the warm-up stage, (ii) the online validation stage, and (iii) either the micro-only or the macro-only stage (Fig. 1). In the *warm-up* stage, both micro and macro propagators are running. In each time step, the micro state $\mathbf{x}_t$ is passed through the encoder $\mathcal{E}^{\boldsymbol{\theta}_\mathcal{E}}$ and fed into the macro propagator in order to warm up its hidden state $\mathbf{h}_t$ (starting from $\mathbf{h}_t = \mathbf{0}$). In the *online validation* stage, micro and macro propagators run independently, in order to estimate the macro propagator's prediction accuracy for the current section of the system trajectory. At the end of the online validation stage, the final latent state $\tilde{\mathbf{z}}_t$ is decoded back to the high-dimensional space, and an application-specific MLM prediction error $E_t = E(\mathbf{x}_t, \mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\tilde{\mathbf{z}}_t))$ between the micro state (the ground truth) and the MLM's prediction is computed. If either the MLM prediction error E_t or the uncertainty σ_t of the macro propagator are above the transition thresholds $E_{\max}$ and $\sigma_{\max}$, respectively, the macro prediction is discarded, and the simulation continues with the micro-only stage. However, if both are below error thresholds, the prediction $\tilde{\mathbf{z}}_t$ of the macro propagator is accepted, and the simulation continues with the macro-only

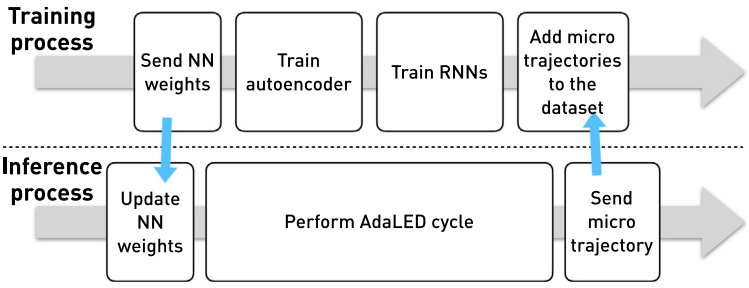

Fig. 2. The inference and training loops. For performance reasons, training and inference are optionally performed in separate processes. Such division naturally extends to multiprocess training, multiprocess simulations, and to multiple simulations.

stage. Cycles are described as *accepted* or *rejected*, depending on whether the macro prediction was accepted or not.

In accepted cycles, the online validation stage is followed by the *macro-only* stage, where the micro propagator is paused, and the only computation is done in the latent state using the inexpensive macro propagator. This stage continues as long as the prediction uncertainty σ_t is below the threshold $\sigma_{\max}$. Once the threshold is violated, the prediction for that step is dropped. Then, the latent state from the previous time step is decoded to the high-dimensional state and passed to the micro propagator. An important assumption is that the micro propagator can be reinitialized to an arbitrary state. Additionally, this stage is optionally limited to $N^{\max}_{\text{macro-only}}$ steps. In Fig. 1, values N_X represent the number of time steps in the stage X.

2.2. Dataset and training

The trajectories $\mathbf{x}_t$ produced by the micro propagator are sliced into trajectories of L time steps and stored in a dynamic dataset of capacity $D \gg 1$. The length L is set equal to the number of recorded states in accepted cycles: $L = 1 + N_{\text{warm-up}} + N_{\text{online-validation}}$. Once the dataset is filled, when adding a new trajectory, an existing trajectory selected uniformly at random is deleted. Randomly removing trajectories ensures that old trajectories are preserved for a long time, alleviating the problem of catastrophic forgetting [54], i.e., neural network predictions deteriorating in continuous learning for samples that they have seen in the past.

The autoencoder and the macro propagator are trained separately, one after the other, on a random subset of the dataset. Training is performed continuously, either after each AdaLED cycle or asynchronously in parallel with AdaLED cycles (Fig. 2). For inference, during one AdaLED cycle, the autoencoder and macro propagator parameters are fixed.

2.3. Estimation of prediction uncertainty

In the macro stage of AdaLED, the trajectory predicted by the macro propagator will eventually diverge from the ground truth, that the micro propagator would have produced, at a rate that depends on the complexity of the system dynamics [19]. In this study, rather than manually selecting the number of macro steps, we adopt a robust mechanism for estimating the duration of reliable coarse-grained predictions. To achieve this, we use probabilistic networks and network ensembles [55].

The input data are denoted by $\mathbf{x} \in \mathbb{R}^{N_x}$, and the output data (targets) of a network whose prediction uncertainty we want to estimate by $\mathbf{y} \in \mathbb{R}^{N_y}$. The output of the system is predicted via two networks. The first network is parameterized with $\boldsymbol{\theta}_{\boldsymbol{\mu}}$ and outputs the mean $\boldsymbol{\mu}^{\boldsymbol{\theta}_{\boldsymbol{\mu}}}(\mathbf{x})$. This network is trained to minimize the MSE between the target and the mean output, i.e.,

$$\ell_{\text{MSE}}(\boldsymbol{\theta}_{\boldsymbol{\mu}}, \mathbf{x}, \mathbf{y}) = \frac{1}{N_y} \sum_{i=1}^{N_y} \left(\boldsymbol{\mu}^{\boldsymbol{\theta}_{\boldsymbol{\mu}}}(\mathbf{x})_i - y_i\right)^2. \tag{5}$$

A second network with parameters $\boldsymbol{\theta}_{\boldsymbol{\sigma}}$ outputs the variance $\boldsymbol{\Sigma}^{\boldsymbol{\theta}_{\boldsymbol{\sigma}}}(\mathbf{x}) = \text{diag}(\boldsymbol{\sigma}^{\boldsymbol{\theta}_{\boldsymbol{\sigma}}}(\mathbf{x})^2)$ of a Gaussian distribution with mean $\boldsymbol{\mu}^{\boldsymbol{\theta}_{\boldsymbol{\mu}}}$, i.e., $p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}^{\boldsymbol{\theta}_{\boldsymbol{\mu}}}(\mathbf{x}), \boldsymbol{\Sigma}^{\boldsymbol{\theta}_{\boldsymbol{\sigma}}}(\mathbf{x}))$ [55,56], where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\theta}_{\boldsymbol{\sigma}}\}$. A diagonal covariance matrix is

considered here for simplicity. The details of the neural architecture are shown in Appendix A. This second network is trained to minimize the negative log-likelihood loss (NLL):

$$\begin{aligned}\ell_{\text{NLL}}(\boldsymbol{\theta}_{\boldsymbol{\sigma}}, \mathbf{x}, \mathbf{y}) &= -\log p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}) \\ &= \frac{1}{2}\log\left(\boldsymbol{\sigma}^{\boldsymbol{\theta}_{\sigma}}(\mathbf{x})\right)^2 + \frac{\left(\mathbf{y} - \boldsymbol{\mu}^{\boldsymbol{\theta}_{\mu}}(\mathbf{x})\right)^2}{2\left(\boldsymbol{\sigma}^{\boldsymbol{\theta}_{\sigma}}(\mathbf{x})\right)^2} + \text{const.}\end{aligned} \tag{6}$$

The networks are trained together, and can be viewed as a single network with parameters $\boldsymbol{\theta}$, while the weights $\boldsymbol{\theta}_{\boldsymbol{\mu}}$ are considered fixed in the computation of the NLL loss. The total sample loss can be written as:

$$\ell(\boldsymbol{\theta}, \mathbf{x}, \mathbf{y}) = \ell_{\text{MSE}}(\boldsymbol{\theta}_{\boldsymbol{\mu}}, \mathbf{x}, \mathbf{y}) + \ell_{\text{NLL}}(\boldsymbol{\theta}_{\boldsymbol{\sigma}}, \mathbf{x}, \mathbf{y}), \quad \boldsymbol{\theta} = \{\boldsymbol{\theta}_{\boldsymbol{\mu}}, \boldsymbol{\theta}_{\boldsymbol{\sigma}}\}. \tag{7}$$

This combination of MSE and NLL losses with decoupled gradients for $\boldsymbol{\theta}_{\boldsymbol{\mu}}$ achieved higher accuracy than solely the NLL loss.

Moreover, we consider an ensemble of K such probabilistic networks, each randomly initialized with its own parameters $\boldsymbol{\theta}^k$, $k \in \{1, \ldots, K\}$ and trained separately on the same data to minimize the loss $\ell(\boldsymbol{\theta}^k, \mathbf{x}, \mathbf{y})$. For a given input $\mathbf{x}$, the outputs $\boldsymbol{\mu}^{(k)} = \boldsymbol{\mu}^{\boldsymbol{\theta}^k_{\mu}}(\mathbf{x})$ and $\boldsymbol{\sigma}^{(k)} = \boldsymbol{\sigma}^{\boldsymbol{\theta}^k_{\sigma}}(\mathbf{x})$ are combined into the final prediction $\boldsymbol{\mu}(\mathbf{x})$ and uncertainty $\boldsymbol{\sigma}(\mathbf{x})$ of the ensemble as follows [55]:

$$\begin{aligned}\boldsymbol{\mu}(\mathbf{x}) &= \frac{1}{K}\sum_k \boldsymbol{\mu}^{(k)}(\mathbf{x}), \\ \boldsymbol{\sigma}^2(\mathbf{x}) &= \underbrace{\frac{1}{K}\sum_k \left(\boldsymbol{\sigma}^{(k)}\right)^2(\mathbf{x})}_{\boldsymbol{\sigma}^2_{\text{ind}}(\mathbf{x})} + \underbrace{\frac{1}{K}\sum_k \left(\boldsymbol{\mu}^{(k)}\right)^2(\mathbf{x}) - \boldsymbol{\mu}^2(\mathbf{x})}_{\boldsymbol{\sigma}^2_{\text{std}}(\mathbf{x})}.\end{aligned} \tag{8}$$

The term $\boldsymbol{\sigma}_{\text{ind}}$ in Eq. (8) refers to the prediction uncertainties as estimated by each network individually. On the other hand, the term $\boldsymbol{\sigma}_{\text{std}}$ measures the disagreement of the ensemble in the prediction of $\mathbf{y}$. Thus, this term provides an estimate of the training inaccuracy for the given input $\mathbf{x}$. For unseen states $\mathbf{x}$ or states underrepresented in the training data, we expected $\boldsymbol{\sigma}_{\text{ind}}$ and $\boldsymbol{\sigma}_{\text{std}}$ to be larger than for frequently seen states. From the vector uncertainty $\boldsymbol{\sigma}$, the scalar uncertainty σ is defined as either $\sigma = \|\boldsymbol{\sigma}\|_2$ or $\sigma = \|\boldsymbol{\sigma}\|_2/N_y$.

2.4. AdaLED hyper-parameters

The error thresholds E_{max} and σ_{max} are application-specific and determine the trade-off between speed-up and accuracy. The latent state dimension d_z of the autoencoder should be chosen based on the system dynamics and its effective degrees of freedom [24]. The remaining hyperparameters, such as network size and the number of layers, can be determined through small-scale experiments and hyperparameter tuning.

3. Case study: Van der Pol oscillator

We first demonstrate the capabilities of AdaLED on the Van der Pol oscillator (VdP), a system used as a benchmark for a variety of multiscale frameworks [10,13,15]. In contrast to these frameworks, we do not distinguish a priori between fast and slow dynamics. Instead, we arbitrarily change the oscillator limit cycle and oscillation time scale, which is controlled by the damping parameter μ, to demonstrate that AdaLED can adapt to these changes. In this case study, no autoencoder is used, i.e., the encoder and the decoder are identity operators.

The Van der Pol oscillator [57,58] is a non-linear damped oscillator governed by the following equations:

$$\begin{aligned}\frac{\mathrm{d}x}{\mathrm{d}t} &= \mu\left(x - \frac{1}{3}x^3 - y\right), \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= \frac{1}{\mu}x,\end{aligned} \tag{9}$$

where $\mu = \mu(t) > 0$ is a time-varying system parameter that controls the system's non-linearity and damping.

The micro propagator is an ODE integrator based on the Euler method that integrates Eq. (9) with a time step of $\delta t = 0.001$ starting from a random initial condition $(x_0, y_0) \sim \mathcal{U}([-5, 5]^2)$. The macro propagator is an ensemble of

five LSTMs. Their architecture is explained in Appendix A. The ensemble is trained to predict the dynamics with a macro time step of $\Delta t = 0.1$. Each LSTM takes the tuple $(x(t), y(t), \mu(t))$ as input and outputs the means and variances $(\mu_{\Delta x}(t), \mu_{\Delta y}(t), \sigma^2_{\Delta x}(t), \sigma^2_{\Delta y}(t))$ for residuals $\Delta x(t) = x(t + \Delta t) - x(t)$ and $\Delta y(t) = y(t + \Delta t) - y(t)$.

For a constant μ, the system enters a limit cycle whose shape depends on μ (Fig. B.1). For the values of μ and Δt considered here, a single limit cycle is completed in about 60 to 90 macro time steps Δt.

AdaLED cycles are configured to have 7 warm-up steps and 25 online validation steps. The maximum number of macro and micro steps is selected for each AdaLED cycle uniformly at random between 80 and 120 steps in order to avoid synchronizing the AdaLED cycle with the system limit cycle. We observed that such synchronization causes the training dataset to be filled with data from the same limit cycle region while the rest of the cycle is underrepresented. For a given cycle c, we define the *macro utilization* $\eta_c = N^c_{\text{macro-only}}/N^c$ as the fraction of steps performed in the macro-only stage, where $N^c_{\text{macro-only}}$ is the number of macro-only steps and N^c the total number of steps of the cycle c. In rejected cycles, $\eta_c = 0$. The total macro utilization η is defined as $\eta = (\sum_c N^c_{\text{macro-only}})/(\sum_c N^c)$. Given the selected stage durations, the maximum attainable total macro utilization is $\eta \approx 75.6\%$.

The maximum capacity of the dataset is set to 1280 trajectories. The training is performed at a fixed rate of 2 trajectories for each simulation macro step.

The prediction error is defined as $E = \sqrt{(x_{\text{micro}} - x_{\text{macro}})^2 + (y_{\text{micro}} - y_{\text{macro}})^2}$ and the prediction uncertainty as $\sigma = \sqrt{\sigma^2_{\Delta x} + \sigma^2_{\Delta y}}$. The AdaLED transition thresholds are set to $E_{\max} = 0.10$ and $\sigma_{\max} = 0.10$. We note that the error threshold $E_{\max}$ refers to the accumulated error after 25 online validation steps and not a single-step prediction error.

3.1. Results

We analyze the performance of AdaLED on three different cases of $\mu(t)$:

- $\mu_{\text{ALT}}(t)$, a piecewise constant function alternating between values $\mu = 1.0$ and $\mu = 3.0$ every 50 000 time steps,
- $\mu_{\text{RAND}}(t)$, a piecewise constant function with multiple randomly selected values $\mu \in [1.0, 3.0]$, and
- $\mu_{\text{BROWN}}(t)$, a piecewise constant function $\mu_{\text{RAND}}(t)$ augmented with Brownian-like noise (details explained in Appendix B).

The system is integrated for 400 000 time steps. The hyper-parameters of the LSTMs are tuned in a preliminary study reported in Appendix B.1.

The macro utilization for all three cases, together with the functions $\mu(t)$, are shown in Fig. 3. The case $\mu_{\text{ALT}}(t)$ is depicted in the top figure. The macro utilization at the start of the run is equal to zero, which is expected since the macro propagator is untrained and produces inaccurate predictions. After about 8000 time steps, the prediction error reaches the desired threshold and AdaLED starts accepting the prediction of the macro propagator. As a result, the macro utilization increases gradually to 60%. At the time step 50 000, the value of $\mu(t)$ suddenly changes, putting the system into an unseen regime. AdaLED correctly detects the change and starts rejecting the predictions of the macro propagator, which suddenly became unreliable. However, soon after, the macro propagator learns the new regime and its predictions are accepted again, leading to an increase in macro utilization, which approaches the maximum of 75.6%. After 50 000 more time steps, the system switches back to $\mu = 3$. The macro propagator has already observed and learned this regime, as demonstrated by the fact that AdaLED resumes uninterruptedly with a very high acceptance rate.

The results for the case $\mu_{\text{RAND}}(t)$, shown in the middle plot of Fig. 3, display similar behavior as the previous case. At the beginning of the simulation, all cycles are rejected until the network learns the corresponding regime. This trend repeats after the first two changes in μ. From the third change onwards, we observe that AdaLED can interpolate between previously seen values of μ and can continue to produce accurate predictions despite changes on $\mu_{\text{RAND}}(t)$. This demonstrates the ability of AdaLED to learn and interpolate on unseen dynamical regimes adaptively.

Finally, the more complex case of a noisy $\mu_{\text{BROWN}}(t)$ is depicted in the bottom plot in Fig. 3. AdaLED again gradually learns to replace the micro propagator and adaptively learns the different dynamical regimes. However, here it takes longer for the macro propagator to reach the highest acceptance rate compared to the cases μ_{ALT} and

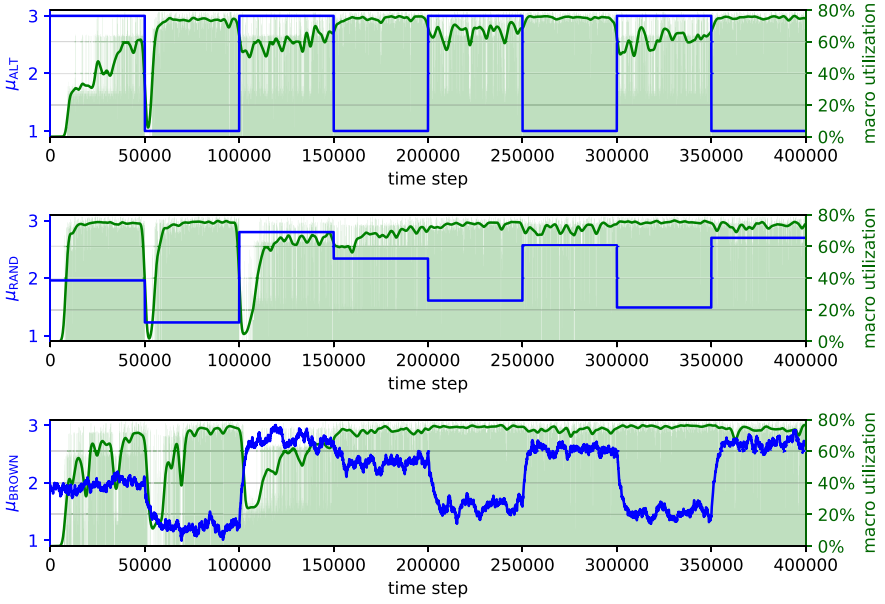

Fig. 3. Macro utilization (fraction of steps performed in macro-only stage) for the Van der Pol oscillator case study for three different variants of $\mu(t)$. The light green histograms show the macro utilization of each individual AdaLED cycle, and the dark green line the macro utilization smoothed using a Gaussian blur (for visualization purposes only). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

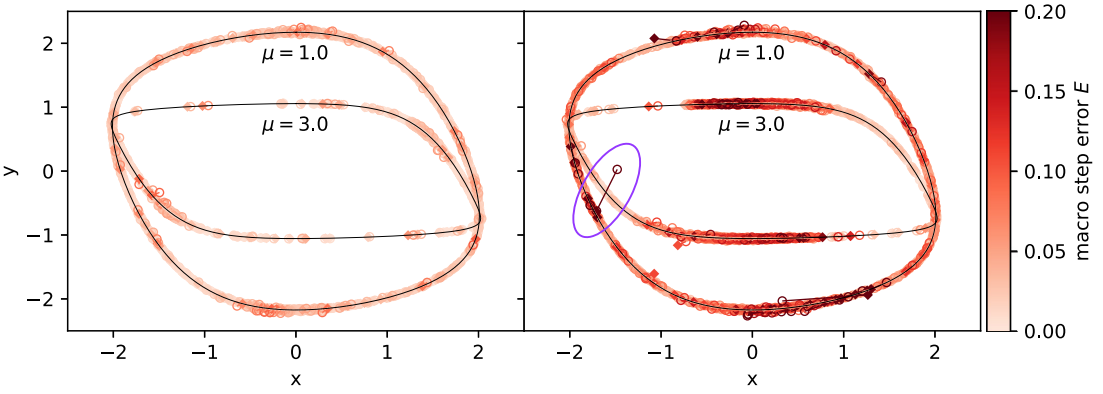

Fig. 4. Visualization of the prediction error for each accepted cycle in the case $\mu_{\mathrm{ALT}}(t)$ of the Van der Pol oscillator case study. Left: error at the end of the online validation stages, right: error at the end of macro-only stages. Diamonds denote micro states and empty circles the macro states, colored with respect to the error E (Euclidean distance). The AdaLED cycles with endpoints outside of limit cycles correspond to those during which the value of $\mu_{\mathrm{ALT}}(t)$ changed. The first such change at the time step 50 000 (encircled pair) is visualized in detail in Fig. 5.

μ_{RAND}. We argue that this is due to the increased difficulty of the learning task, as data are generated from various limit cycles with varying time scales.

The online validation errors and final testing errors (prediction error at the end of the macro-only stage) for each accepted cycle in case $\mu_{\mathrm{ALT}}(t)$ are depicted in Fig. 4. The testing error is calculated by running the micro propagator even during the macro-only stage, solely for evaluation purposes. In production runs, the micro propagator is inactive during the macro-only stages.

We observe that errors are generally small and almost every cycle ends very close to the limit cycles. Exceptions to this trend occur during sudden changes of $\mu(t)$. It is important to note that the online validation errors (left plot) can be directly controlled by adjusting the value of $E_{\max}$. On the other hand, the testing error (right plot) can be

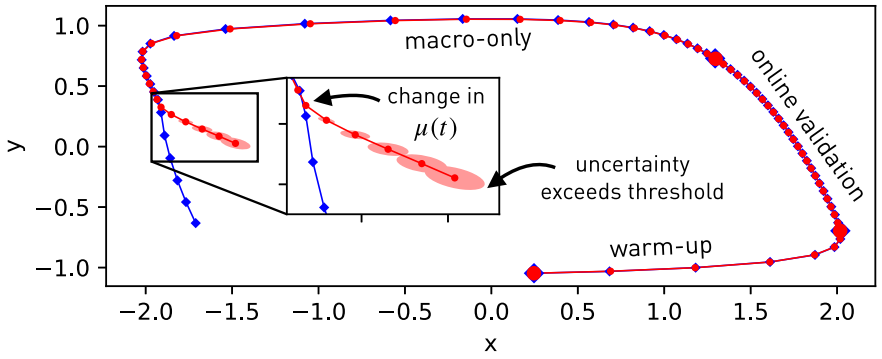

Fig. 5. AdaLED cycle around the time step 50 000 at which $\mu_{\text{ALT}}(t)$ switches from value 3.0 to 1.0. Blue diamonds ◆ denote the micro states, red circles ● the macro states, and red ellipses the ensemble covariance. The plot shows that, due to the different responses of individual LSTMs in the ensemble to the value of $\mu(t)$, AdaLED quickly detected the change in dynamics and stopped the execution of the macro propagator.

controlled only indirectly through E_{max} and σ_{max}, and thus serves as a measure of the robustness and quality of the surrogate model. This highlights the advantage of AdaLED compared to other multiscale frameworks, as it allows for control over the error thresholds a priori.

The behavior of the system during the first change of μ_{ALT}, happening at time step 50 000, is visualized in Fig. 5. Prior to that change, the LSTMs were trained only on trajectories with $\mu = 3$. As a result, the predictions of individual networks in the ensemble are alike and accurate for this particular regime. However, once the value of $\mu_{\text{ALT}}(t)$ changes, different LSTMs in the ensemble respond differently to $\mu = 1$, since the predictions for the unseen regimes are arbitrary and depend on weight initialization. This causes the predictions to diverge and the uncertainty to increase (as defined in Eq. (8)). When the uncertainty crosses the threshold σ_{max} after seven steps, the cycle terminates. The 7th step is rejected and therefore excluded from the plot.

Additional results on the dependence of the testing error on the uncertainty threshold σ_{max} and the ensemble size K are presented in Appendix B.2.

4. Case study: Reaction–diffusion equation

Here, we test AdaLED on the lambda-omega reaction–diffusion system [59,60] governed by:

$$\begin{aligned} \frac{\partial u}{\partial t} &= [1-(u^2+v^2)]u + \beta(u^2+v^2)v + d_1\nabla^2 u, \\ \frac{\partial v}{\partial t} &= -\beta(u^2+v^2)u + [1-(u^2+v^2)]v + d_2\nabla^2 v \end{aligned} \tag{10}$$

for $-10 \leq x, y \leq 10$, where $\beta = 1.0$ is the reaction parameter and $d_1 = d_2 = d = d(t)$ the time-varying diffusion parameters. The equation is integrated on a 96×96 uniform grid using the Runge–Kutta–Fehlberg method of fourth order with a time step of $\Delta t = 0.05$. Thus, the state of the system is fully described by a tensor $\mathbf{w} = (\mathbf{u}, \mathbf{v}) \in \mathbb{R}^{2\times 96\times 96}$. The system exhibits a spiral wave whose shape depends on the parameter d.

We evaluate the performance of AdaLED with the diffusion parameter d alternating between 0.1 and 0.2 every 20 000 time steps. AdaLED cycles are configured to have 5 warm-up steps, 18 online validation steps, 10 to 15 micro-only steps, and 400 to 500 macro-only steps. The AdaLED transition thresholds are set to $E_{\text{max}} = 0.002$ and $\sigma_{\text{max}} = 0.002$. We use the mean square error (MSE) between the macro and the micro state $\mathbf{w}$ as the error metric, i.e., $E = E(\mathbf{w}, \tilde{\mathbf{w}}) = \|\mathbf{w} - \tilde{\mathbf{w}}\|_2^2/(2 \cdot 96 \cdot 96)$. The simulation is run for 200 000 time steps. Other details of the system, the neural networks, and the training are listed in Appendix C.

The macro utilization η and test errors E are shown in Fig. 6. Similar to the Van der Pol oscillator case study, we observe that the macro utilization is initially zero, as the networks are not yet trained. However, after approximately 10 000 time steps, the autoencoder learns to reconstruct the state, and the LSTMs learn to predict the dynamics. Consequently, AdaLED begins to accept the macro prediction. When the diffusion parameter d changes from 0.1 to 0.2, AdaLED recognizes the unreliability of the macro predictions and switches back to the simulator. Once the

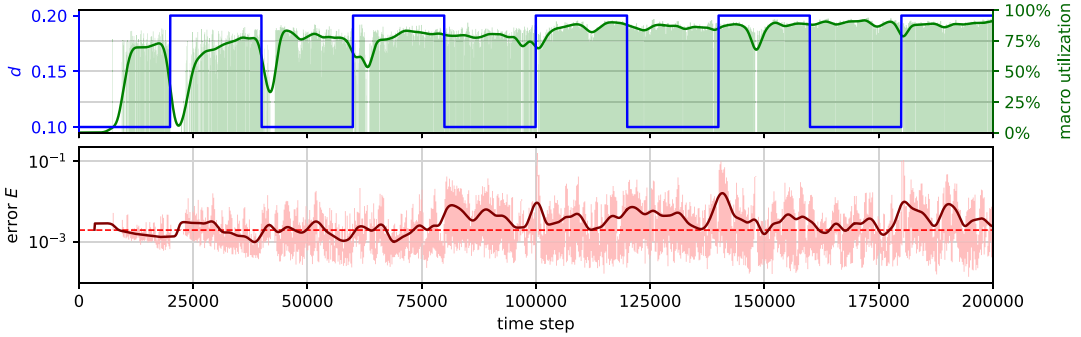

Fig. 6. Performance on AdaLED on the reaction–diffusion case study with a time-varying diffusion parameter d (blue line). Top: macro utilization (fraction of steps performed in macro-only stage). The light green histograms show the macro utilization of each individual AdaLED cycle, and the dark green line shows the macro utilization smoothed using a Gaussian blur (for visualization purposes only). Bottom: test error of the reconstructed state $\mathbf{w}$. The per-step errors (faded red) alternate between low values at the beginning of the macro-only stage and higher errors at the end of the macro-only stage. The dark red denotes the smoothed test error, and the dashed red the online validation threshold $E_{\max}$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

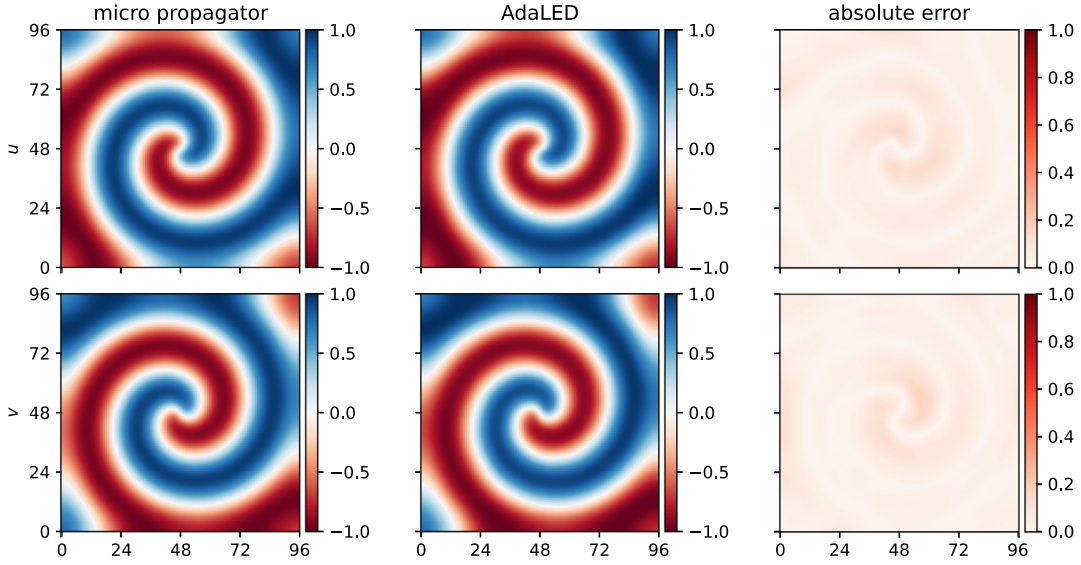

Fig. 7. Snapshot of the simulation at time step 200000, 305 steps into the macro-only stage. Left: the micro propagator (ground truth), middle: the AdaLED Machine-Learned Model, right: absolute error. The mean square error amounts to $E = 0.0021$ (relative error of 0.0045).

new regime is learned, AdaLED resumes using the macro propagator. The bottom plot of Fig. 6 shows the test errors E. Overall, the macro utilization reaches $75\% \pm 1\%$, with an MSE of 0.0055 ± 0.0013 (relative MSE of 0.012 ± 0.003). The reported confidence levels are based on the variance calculated from ten repeated simulation runs.

Fig. 7 shows a snapshot of the simulation at the time step 200 000, 305 time steps into the macro-only stage. The predicted and expected states are in agreement, demonstrating the accuracy of the macro propagator even after performing the simulation for a significant number of time steps in the latent space. The micro states during macro-only stages are retrieved for testing purposes by continuing the micro simulation even during the macro-only stage.

Fig. 8. Schematic view of the multiresolution AE. To accelerate the training and reduce the storage and memory requirements, the AE operates on two downsampled variants of the velocity field. The yellow region denotes the blending mask used for reconstructing the full resolution field. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5. Case study: 2D flow past a cylinder

Finally, we employ AdaLED to accelerate a 2D Direct Numerical Simulation (DNS) of the flow past a circular cylinder at varying Reynolds numbers. AdaLED is trained to forecast the velocity field, representing the state of the flow. Forecasting the complete simulation state enables the alternation between macro (latent) and micro scale (the DNS). In addition to predicting the state, AdaLED is also tasked with predicting the force exerted by the fluid on the cylinder. The force serves as the quantity of interest $\mathbf{q}(t)$ that we want to have access to at all time steps of the simulation.

The system is governed by the incompressible Navier–Stokes equations and the no-slip boundary condition is enforced via the Brinkman penalization [61]:

$$\nabla \cdot \mathbf{u} = 0, \tag{11}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \lambda(\mathbf{u}^s - \mathbf{u})\chi, \tag{12}$$

where $\mathbf{u} = \mathbf{u}(\mathbf{x}, t)$ is the fluid velocity field, $\rho = 1$ the fluid density, $p = p(\mathbf{x}, t)$ the pressure, $\nu = 10^{-4}$ the kinematic viscosity, $\lambda = 10^6$ Brinkman penalization coefficient, $\mathbf{u}^s = \mathbf{u}^s(t)$ the velocity of the cylinder and $\chi = \chi(\mathbf{x}, t)$ the characteristic function of the cylinder, equal to 1 inside the cylinder and 0 outside it. The equation is solved on a $[0, 1] \times [0, 0.5]$ domain with open boundary conditions. A solid cylinder of diameter $d = 0.075$ is fixed at the coordinate $(0.2, 0.25)$ relative to the simulation domain. The cylinder and the simulation domain are moving horizontally at the speed of $u_x^s(t) = \mathrm{Re}(t)\nu/d$ relative to the fluid, with Reynolds number $\mathrm{Re}(t)$ (the external forcing) varying between $\mathrm{Re} = 400$ and $\mathrm{Re} = 1200$. In this range, for a fixed Re, a vortex street forms behind the cylinder.

Eqs. (11) and (12) are solved using a pressure projection method [62] on an adaptive Cartesian mesh of maximum resolution of 1024×512 cells. For the purpose of this study, the adaptive, non-uniform mesh is interpolated to the maximum resolution of 1024×512 cells and is thus treated as a uniform mesh when used by AdaLED.

In an effort to reduce the computational demands of the high-dimensional grid and speed up the training process of AdaLED, we propose a novel multiresolution physics-based AE. The proposed AE takes advantage of the characteristics of the flow and uses a reduced-resolution grid far from the cylinder where the flow exhibits simpler features compared to the vicinity of the cylinder. Concretely, the AE operates on two downsampled grids: a half-resolution grid spanning the entire domain and a small full-resolution patch around the cylinder (Fig. 8). By utilizing this multiresolution approach, we are able to reduce the storage and memory requirements and speed up the training. Furthermore, the AE outputs the stream function instead of the velocity field [63]. This physics-inspired architecture ensures zero divergence of the velocity field (as per Eq. (11)). An additional physics-based vorticity loss is added to improve performance. The specifics are outlined in Appendix D.1.

In addition to the latent state $\mathbf{z}(t)$, which is necessary to recreate the system dynamics and support macro-to-micro transitions, the MLM also outputs the force $\mathbf{F}_{\mathrm{cyl}}(t)$ exerted by the fluid on the cylinder as the quantity of interest $\mathbf{q}(t)$. The relative importance of $\mathbf{F}_{\mathrm{cyl}}$ and $\mathbf{z}$ in the macro propagator's training loss and uncertainty estimation can be controlled by scaling the force with some factor α_F.

The macro propagator is an ensemble of five probabilistic LSTMs. Each LSTM is trained to predict the mean and the variance of the residuals $\Delta\mathbf{z}(t) = \mathbf{z}(t + \Delta t) - \mathbf{z}(t)$ and $\Delta\mathbf{q}(t) = \mathbf{q}(t + \Delta t) - \mathbf{q}(t)$, $\mathbf{q}(t) = \alpha_F \mathbf{F}_{\mathrm{cyl}}(t)$. The

acceptance criterion is based on a relative reconstruction error E of the velocity field

$$E(\tilde{\mathbf{u}}, \mathbf{u}) = \frac{\|\mathbf{u} - \tilde{\mathbf{u}}\|_2^2}{\|\mathbf{u}\|_2^2}, \quad \|\mathbf{u}\|_2^2 = \sum_{ij} \mathbf{u}_{ij}^2, \tag{13}$$

where $\mathbf{u}$ is the full-resolution velocity field from the micro propagator, and $\tilde{\mathbf{u}} = \mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\tilde{\mathbf{z}})$ the prediction of the ML model. The autoencoder itself is trained on a different loss function, explained in Appendix D.1. The total uncertainty σ of the prediction of the macro propagator is defined as the standard deviation of the uncertainty vectors $\sigma = \sqrt{\left((\boldsymbol{\sigma}_{\Delta \mathbf{z}}(t))^2 + (\boldsymbol{\sigma}_{\Delta \mathbf{q}})(t)^2\right)/(d_z + 2)}$.

The time step of AdaLED is set to $\Delta t = 0.005$, resulting in approximately 60 AdaLED time steps per vortex street period for Re $= 1000$. The internal time step of the micro propagator δt is, for simplicity, fixed throughout the simulation. For simulations with Re(t) of up to 1000, $\delta t = 0.005/18$, and for simulations with Re(t) of up to 1200, $\delta t = 0.005/21$, resulting in a Courant number of $\sim$0.4.

We use AdaLED cycles of 4 warm-up steps, 12 online validation steps, between 400 and 500 macro steps, and between 9 and 14 micro steps. Both limits are chosen uniformly at random for each cycle to avoid synchronizing AdaLED cycles with vortex street periods. The capacity of the dataset is set to 256 trajectories. To maximize the speed-up of AdaLED, training is performed on a separate compute node in parallel with the inference and the micro propagator, as depicted in Fig. 2. Experiments were conducted on the Piz Daint supercomputer on two XC50 nodes, each equipped with one 12-core Intel Xeon E5-2690 CPU running at 2.6 GHz and one Nvidia P100 16 GB GPU. The simulations were performed using the `CubismAMR` software [64].

5.1. Results

In Section 5.1.1, we demonstrate the effectiveness of AdaLED in accelerating the simulation of the flow past the cylinder without sacrificing accuracy. In Section 5.1.2, we highlight the importance of adaptive training in systems with changing dynamics. Finally, in Section 5.1.3, we conduct an ablation study to evaluate the advantage of the multiresolution autoencoder.

5.1.1. Effectiveness of AdaLED

We perform the simulation for a total of 300 000 time steps, with Reynolds number transitioning cyclically between Re $= 600$, Re $= 750$, and Re $= 900$ every 5000 time steps. The hyper-parameters, listed in Table 2, are tuned according to the performance on a shorter simulation, as reported in Section 5.1.3. For this simulation, the error and uncertainty thresholds are set to $E_{\max} = 0.017$ and $\sigma_{\max}^2 = 0.00035$, respectively. We note that these are the key AdaLED hyper-parameters that can be adjusted to balance accuracy and acceleration as desired.

The Re(t) profile, the macro utilization, and the errors are displayed in Fig. 9. The errors correspond to validation errors during the online validation phase and test errors during the macro-only stage. To calculate these errors, we compare the micro states $\mathbf{u}_t$ with the reconstructed states $\tilde{\mathbf{u}}_t = \mathcal{D}^{\boldsymbol{\theta}_\mathcal{D}}(\tilde{\mathbf{z}}_t)$ produced by the MLM. We use the velocity field error metric E from Eq. (13) to quantify the error in the velocity field. For the force $\mathbf{F}_{\text{cyl}}$ error, we define a normalized error E_F as follows:

$$E_F = E_F(\mathbf{F}'_{\text{cyl}}, \mathbf{F}_{\text{cyl}}) = \frac{\|\mathbf{F}'_{\text{cyl}} - \mathbf{F}_{\text{cyl}}\|}{F_{\text{cyl}}^{\text{avg}}}, \tag{14}$$

where $F_{\text{cyl}}^{\text{avg}} = \left\langle \|\mathbf{F}_{\text{cyl}}\| \right\rangle \approx 0.079$ is the average magnitude of the force. For testing purposes, to retrieve the micro states $\mathbf{u}_t$, we continue running the simulation even in the macro-only stages. The training of the MLM is temporarily suspended during this time.

In Fig. 9, we observe the same trend as in the previous two case studies. Initially, the macro utilization is zero. After the networks become sufficiently trained, the framework starts to accept the predictions of the MLM. As training continues, the errors and uncertainties decrease, resulting in an increase in macro utilization. During the macro-only stage, the testing error E and E_F remain low, averaging to 1% and 5%, respectively. The errors can be further decreased at the cost of reduced speed-up.

In Fig. 10, we present a closer look at how error and uncertainty change over a selected section of the trajectory, specifically during the transition from Re $= 900$ to Re $= 600$. The acceptance of the macro prediction is determined

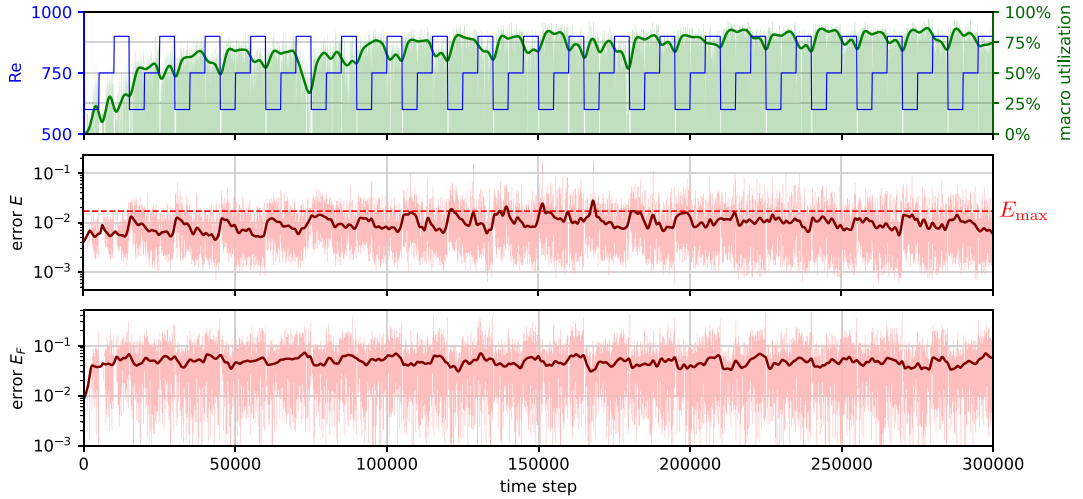

Fig. 9. AdaLED performance on a flow behind cylinder simulation for $\mathrm{Re}(t) \in \{600, 750, 900\}$ (Section 5.1.1). Top: Reynolds number $\mathrm{Re}(t)$ profile and the macro utilization η. Middle and bottom: validation errors of the velocity (E, Eq. (13)) and force on the cylinder (E_F, Eq. (14)). The per-step errors (faded red) alternate between low values at the beginning of the macro-only stage and higher errors at the end of the macro-only stage. The errors for velocity stay close to 1% on average (dark red) and close to 5% for the force (with a cross-correlation of 0.99). The errors refer only to macro-only steps. A detailed view of errors in a short simulation section is shown in Fig. 10. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

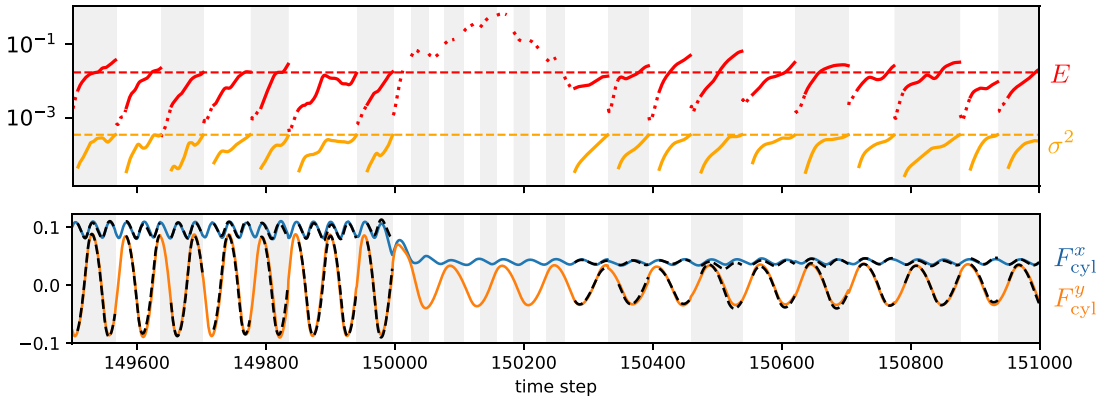

Fig. 10. A detailed view of the part of the simulation of flow behind the cylinder from Section 5.1.1 and Fig. 9, during the $\mathrm{Re} = 900$ to $\mathrm{Re} = 600$ transition. Top: velocity validation error E and the squared uncertainty σ^2 (dotted for warm-up and online validation stages, solid for macro-only) and their thresholds $E_{\max}$ and $\sigma^2_{\max}$ (dashed). Bottom: horizontal force on the cylinder (blue), vertical force (orange), and the macro's prediction (dashed black). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

by the error E and its threshold $E_{\max}$. Before the Re transition, the online validation error remains below the threshold, and AdaLED accepts the macro prediction. However, during the transition period, which lasts for a few hundred time steps, the MLM cannot reliably predict the dynamics. Hence, AdaLED switches to the micro propagator instead. Once the transition period ends, AdaLED resumes utilizing the macro propagator. It should be noted that the error E may exceed the threshold $E_{\max}$ at times. The threshold $E_{\max}$ represents the maximum error at the end of the online validation stage, so it should be set to a value lower than the desired maximum tolerable error.

In contrast to the error E that controls whether AdaLED enters the macro-only stage, the uncertainty σ and the threshold $\sigma_{\max}$ control its duration. Once σ exceeds $\sigma_{\max}$, the macro-only stage is stopped.

Table 1
Average execution times per time step for the simulation setup from Section 5.1.1, together with the speed-up and the macro utilization η.

Operation	Execution time [ms]	Comment
without AdaLED		
micro propagator	**969**	
with AdaLED		
micro propagator	1055	larger internal mesh
overhead of AdaLED	13	autoencoder, LSTMs, data handling...
training	N/A	running on another compute node
average	**339**	**2.9× speed-up** ($\eta = 69\%$)
average (last 15k time steps)	**225**	**4.3× speed-up** ($\eta = 80\%$)

The accuracy of the predicted force $\mathbf{F}_{\text{cyl}}$ is illustrated in the bottom plot of Fig. 10. We observe a good agreement between the two force profiles. A visual representation of the latent trajectory can be found in Fig. D.4 in Appendix D.5.

A snapshot of the simulation during the macro-only stage is visualized in Fig. 11. We observe that AdaLED reproduces the state of the simulation accurately and captures the characteristics of the flow with high accuracy. Notably, errors concentrate on the fine-scale structures of the flow. Arguably, the double arcs in the error profile indicate that the error can be partially attributed to the macro propagator advancing the dynamics at an incorrect speed.

The execution time of the standalone simulation and the AdaLED-accelerated simulation is compared in Fig. 12. The green area represents the time saved by using AdaLED. The total macro utilization over the whole run (300 000 time steps) is 69%, achieving a speed-up of approximately 2.9×. After the training converges, the macro utilization reaches 80% in the last 15 000 time steps, resulting in a speed-up of 4.3×. This implies that the trained MLM can be applied to other simulations with $\text{Re} \in \{600, 750, 900\}$ achieving similar performance.

We remark that the reported 2.9–4.3× speed-up is modest. This is a result of the frequent transition to the micro model, which is a trade-off we take to ensure that the (rather stringent) accuracy threshold is satisfied. The constraint of accuracy is unique to the present algorithm over other surrogate techniques (including the original LED [24]). In order to increase the speed-up, we either need to improve the accuracy of the macro model (to have longer macro-only stages), reduce the duration of the online validation stage, or even remove the warm-up stage.

The execution time breakdown of each simulation time step is presented in Table 1. By itself, the micro propagator (without any use of AdaLED) takes on average 969 ms per time step for the given profile of $\text{Re}(t)$ (larger Re are slower to simulate). When used within AdaLED, the micro propagator is slower, down to 1055 ms per step (+9%). This is due to the fact that we employ a solver [64,65] with adaptive mesh refinement. Namely, after the first micro-to-macro transition, the imperfect autoencoder reconstruction causes a slight increase in the mesh size. We report speed-ups compared to the original simulation without AdaLED to guarantee a fair comparison. The overhead of AdaLED (autoencoder, macro propagator, data handling, logging and diagnostics) is small in comparison, averaging to only 13 ms per time step. This refers to the total time spent outside the micro propagator, divided by the number of steps.

As mentioned earlier, the training is performed on a separate compute node. Therefore, it causes no overhead to the micro propagator or the networks. Although this increases the computational cost, the training cost becomes negligible once AdaLED is used to accelerate large tasks that require many simulations in parallel. We note that the trained surrogate can be saved and reused for simulations with similar dynamics.

We conclude that for computationally expensive CFD simulations, the overhead of AdaLED itself is minimal. In this case, the speed-up is determined solely by the macro utilization. Higher speed-ups can be achieved by affording higher errors (increasing the error thresholds) or employing more accurate models. The latter can be achieved through network architecture improvements, more effective training procedures, larger ensembles, or more extensive tuning.

The results of a run with a different Reynolds number profile are shown in Appendix D.6.

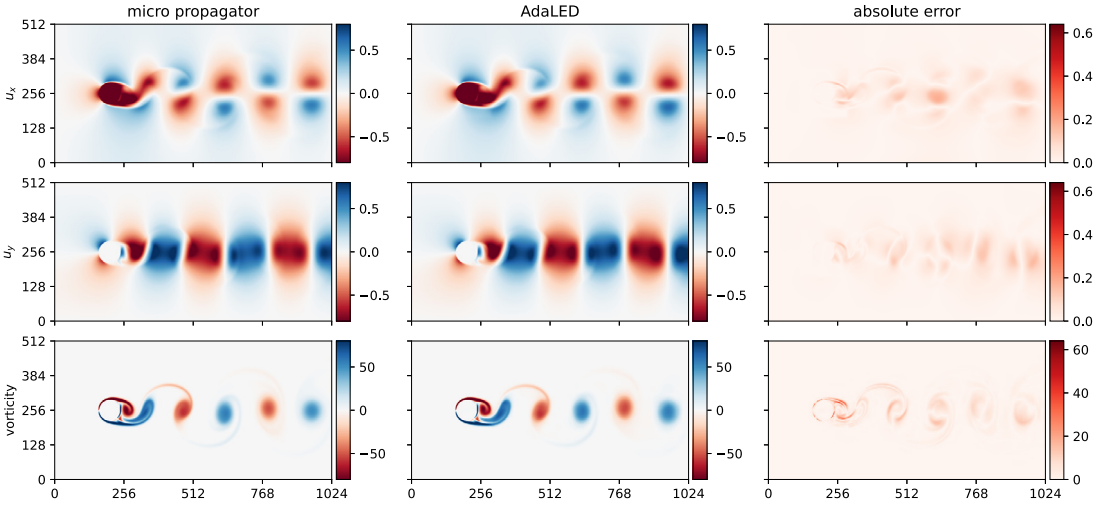

Fig. 11. Snapshot of the time step 212500 (Re = 600) of the simulation from Section 5.1.1, 50 time steps into the macro-only stage, with a relative error of $E \approx 0.014$ (Eq. (13)). Left: micro propagator state $\mathbf{u}_t$ and vorticity ω_t, middle: the prediction of the surrogate (MLM) and full-resolution reconstruction, right: absolute error.

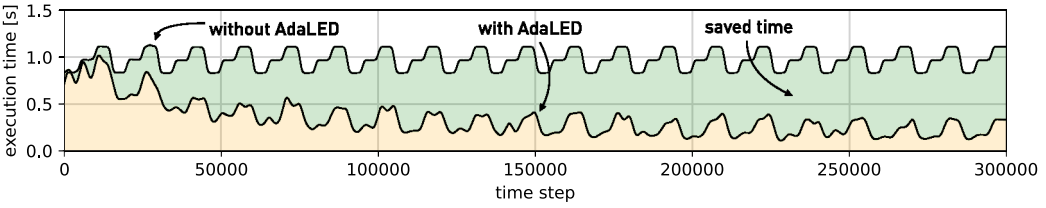

Fig. 12. Smoothed time step execution time with and without AdaLED for the simulation setup from Section 5.1.1. The speed-up factor converges to 4.3× (Table 1). The periodic changes in the execution time correspond to the periodic changes of the Reynolds number (see the top plot in Fig. 9).

Table 2
Parameter search space for the multiresolution autoencoder study, and the selected parameter set.

Parameter	Search space		Selected
multiresolution?	no	yes	yes
inner resolution	N/A	{256 × 256, 224 × 224}	224 × 224
# of CNN layers	{5, 6}	{4, 5}	4
channels/layer	{16, 20, 24}		16
d_z (per resolution)	{4, 8, 12, 16, 24}		8
AE learning rate	LogUniform(0.0001, 0.001)		0.00047
LSTM learning rate	LogUniform(0.0003, 0.003)		0.00126
scaling α_F	LogUniform(0.03, 30.0)		7.2
$E_{\max}$	LogUniform(0.001, 0.1)		0.017
$\sigma^2_{\max}$	LogUniform(0.00001, 0.1)		0.00035

5.1.2. The importance of adaptivity

In the following, we demonstrate the importance of adaptivity, i.e., constantly training throughout the whole simulation and adapting to new states and trajectories, compared to pretraining or training only until a given point in time. We analyze two profiles of time-varying Reynolds numbers Re(t). In the first, Re(t) switches between values 500, 750, and 1000 in a zig-zag fashion throughout the whole simulation. In the second, Re(t) starts as the first profile but switches to a different regime (400, 600, 800, 1000, and 1200) in the second half of the simulation, to emulate a system that enters a new regime late in the simulation. For each profile, two setups are tested: one

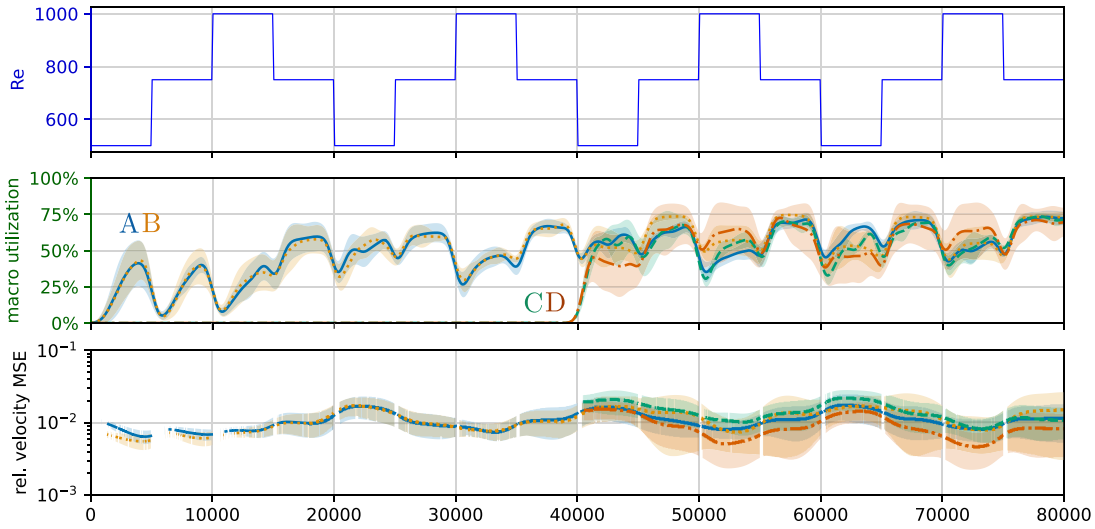

Fig. 13. The adaptivity and transition delay study from Section 5.1.2 for the Re(t) with a fully repeating profile. Lines denote: adaptive without delay (A ——), non-adaptive without delay (B ·····), adaptive with delay (C – – –), non-adaptive with delay (D –·–·–). Top: Reynolds number profile, middle: macro utilization η, bottom: smoothed relative MSE of the velocity in macro-only stages. Shaded regions, where available, denote the standard deviation along five repeated runs of the same setup.

with training enabled all the time (*adaptive*) and one with training enabled only at the first half of the simulation (*non-adaptive*).

Apart from testing adaptivity, we test how disabling micro-to-macro transitions in the first half of the simulation affects the quality of the MLM in the second half. Namely, we expect that delaying initial transitions and providing more time for training may help improve the accuracy and macro utilization in the later stages of the simulation. Thus, for each Re(t) profile, we test in total four setups: adaptive without delay (A; default AdaLED behavior), non-adaptive without delay (B), adaptive with delay (C), and non-adaptive with delay (D). For each setup, five runs with different random seeds are performed to obtain the variance in performance.

The macro utilization and relative MSE on the velocity for the first Re(t) profile are visualized in Fig. 13. We observe that training only in the first half with transitions disabled (the setup D) achieves higher accuracy in the second half of the simulation compared to other setups. In fact, the non-adaptive setup D exhibits higher macro utilization and lower error compared to the adaptive setup C. This is expected as the training dataset from the first half of the run already contains all the information needed to forecast effectively the dynamics in the second half (the profiles are similar). As a consequence, there is no need for online training.

However, we observe a different phenomenon when the system regime changes over time, as shown in Fig. 14. Here, the macro utilization in non-adaptive setups B and D drops to zero when the system enters the previously unseen Re(t) = 1200 regime, whereas the adaptive setups A and C eventually adapt and achieve macro utilization of 25 to 35%.

We note here that the available training time for setups D and C is higher. While setups A and B accelerated the simulation from the start and had only 6 h for training during the first half of the simulation, setups D and C took 11 h for the first half and thus had almost twice as much time for training before being tested in the second half. We argue that this phenomenon is an important characteristic property of online surrogates, i.e., the higher speed-up they achieve, the less time they have for training.

5.1.3. Multiresolution autoencoders

In this section, we perform an ablation study to analyze the benefit of the multiresolution convolutional autoencoder. We compare three cases: (i) single resolution, (ii) multiresolution with a 256 × 256 patch around the cylinder, and (iii) multiresolution with a 224 × 224 patch. For each case, we perform 80 runs with randomized thresholds $E_{\max}$ and $\sigma_{\max}$, learning rates, force scaling α_F, latent state size d_z, number of CNN channels, and the number of layers. The hyper-parameter search space is listed in Table 2. The CNN architecture, the remainder of

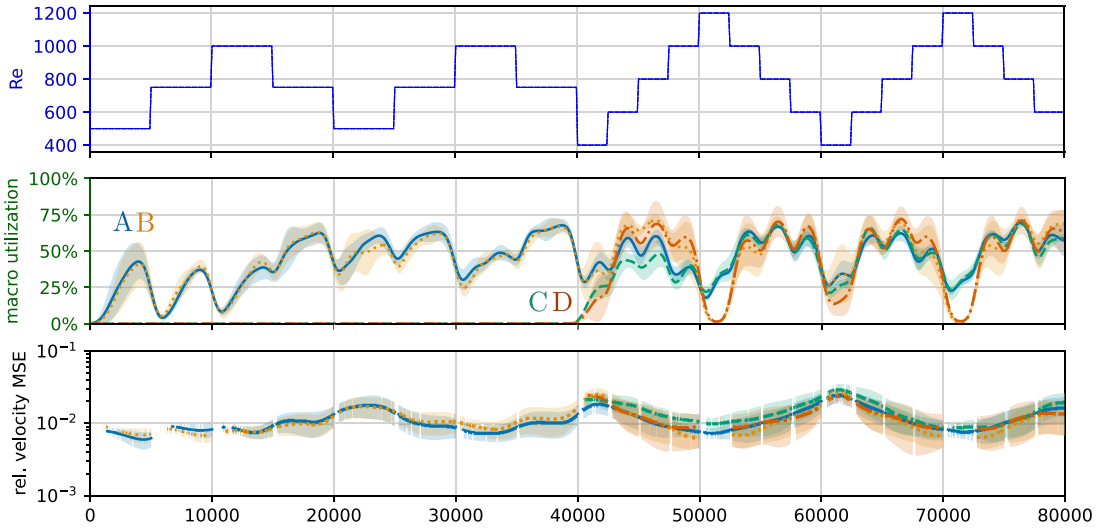

Fig. 14. Analogous of Fig. 13, for the Re(t) that changes the profile in the second half of the simulation. The drop in performance in the second half is clearly visible for the non-adaptive cases (B ····· and D -·-·).

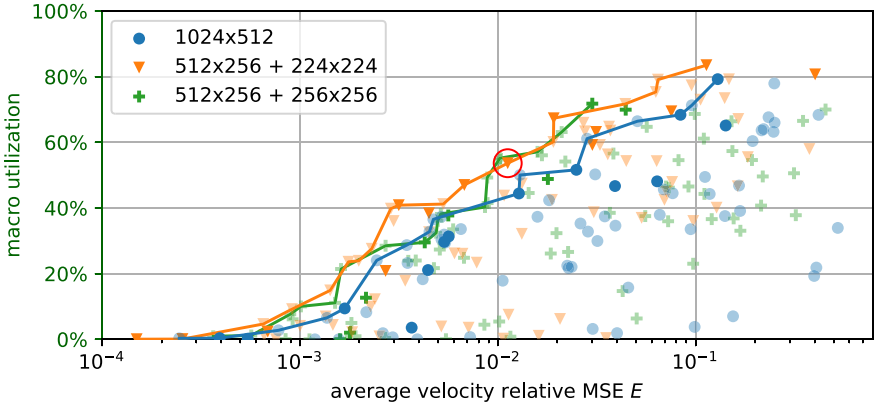

Fig. 15. Multiresolution autoencoder multi-objective study (Section 5.1.3), optimizing for total macro utilization η and velocity error E. The lines represent the Pareto fronts for each autoencoder setup. Darker symbols denote samples that are also optimal in the η–E_F sense (Fig. 16). The encircled sample is the reference parameter set used in the rest of the study.

hyper-parameters, and the breakdown of training execution time are described in Appendix D.4. The simulation setup matches the one from Section 5.1.1 (Reynolds number Re(t) cycles between 600, 750, and 900 every 5000 time steps), with a shorter running time of 60 000 time steps. On average, a single simulation run takes approximately 16 h to complete. Three performance metrics are considered: total macro utilization η, average velocity relative MSE E (Eq. (13)), and the average force error E_F (Eq. (14)). The averages include only the macro-only stages.

The results of the comparison are shown in Figs. 15 and 16, where the macro utilization is plotted against the average errors E and E_F, respectively. We observe a clear advantage of the multiresolution approach compared to the single-resolution autoencoder in terms of both macro utilization and accuracy. The encircled point in the plots refers to the hyper-parameter set used in Sections 5.1.1 and 5.1.2, which resulted in a macro utilization of 54% in 60 000 time steps (69% when run for 300 000 time steps). The performance of this hyper-parameter set on a different Reynolds number profile is shown in Appendix D.6. It is important to note that the results are subject to random variations, with errors varying by about ±5% (relative) and macro utilization varying by ±3% (absolute), depending on the random seed.

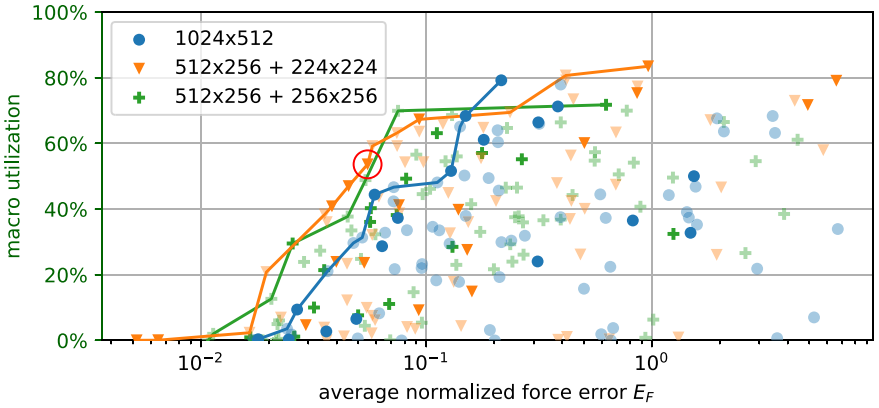

Fig. 16. Analogous of Fig. 15, with x-axis denoting the average normalized cylinder force root MSE E_F instead of the velocity field error E. Here, darker symbols denote optimal samples in the η–E sense.

6. Discussion

We present AdaLED, a framework that employs CAEs and an ensemble of RNN-LSTMs to learn data-driven, online, adaptive machine-learned models to accelerate the simulations of complex systems. The model is trained in parallel with the original simulation (micro propagator) and can adapt online to newly discovered dynamics. More importantly, the model monitors its accuracy and prediction uncertainty and replaces the micro propagator only when its accuracy is high, and its uncertainty is low. This mechanism enables the acceleration of simulations for sections of the state space that are learned, even if the model cannot or is not yet fully trained to faithfully reproduce the whole complex state space dynamics. In regions of the state that are underrepresented or not part of the training data, AdaLED utilizes the original computational solver.

We demonstrate AdaLED in three benchmark problems: the Van der Pol oscillator dynamics with varying parametric nonlinearity $\mu \in [1, 3]$, a 2D reaction–diffusion equation with varying diffusion parameter $d \in [0.1, 0.2]$, and flows past a circular cylinder at varying $\mathrm{Re} \in [400, 1200]$. On these benchmarks, we demonstrate its ability to train a machine-learned model progressively, exploit its predictions only when they are reliable, and detect when a system enters an unseen regime in the phase space. The trained model demonstrates high accuracy in all dynamic regimes seen during training and does not suffer from catastrophic forgetting.

On the flow past a cylinder at varying Re, AdaLED, starting from untrained networks, reproduces the dynamics of vastly different dynamical regimes, achieving a net speed-up of $2.9\times$ for a 3-day-long simulation. This speed-up is achieved at the cost of a mean square error of only $\sim$1% on the velocity field, $\sim$5% root mean square error of the force on the cylinder and cross-correlation of 0.99. The speed-up can be increased further at the cost of lower accuracy by increasing $E_{\max}$ and $\sigma_{\max}$. We emphasize the advantage of AdaLED compared to other frameworks to control this trade-off between speed-up and accuracy. To our knowledge, AdaLED is the first method that can efficiently learn to propagate the high-dimensional dynamics of a complex flow at various regimes using a single surrogate, offering a robust accuracy vs. speed-up trade-off.

Our findings suggest that AdaLED is a potent adaptive algorithm for adaptively constructing and interfacing surrogates that accelerate complex multiscale simulations. We believe that AdaLED can be employed as a black box accelerator that takes advantage of repeating patterns in computation-heavy tasks. In the future, we plan to investigate its acceleration capabilities on reinforcement learning tasks and model parameter optimizations, where multiple simulations can share the same surrogate.

Moreover, we argue that the proposed framework is a valuable contribution to the digital twin literature [66,67]. AdaLED combines data assimilation, real-time monitoring, and online adaptive data-driven learning to build a surrogate. The proposed framework is directly applicable if the simulation of the physical system is possible from any initial condition at will. Otherwise, it can be applied only with minor modifications (by turning off the restarting of the micro-scale solver). This way, the framework can be employed to learn a digital replica of a physical system, i.e., the digital twin. The surrogate's response under different conditions and parametrizations can be tested at will,

avoiding the cost and computational burden of experiments or fully resolved simulations and the risk of exposing the original system to adverse conditions [22,68].

Application-wise, AdaLED can benefit from more advanced autoencoders, such as variational autoencoders [69], autoencoders that take into account temporal correlations [70], or non-uniform autoencoders based on space-filling curves [71] and octrees [72,73]. A topic of ongoing research is to utilize the latter to help scale AdaLED to 3D fluid flows. Moreover, all latent state variables are currently treated as equally important. The method could benefit from compression techniques that can estimate the relevance of each latent dimension in the reconstruction [74]. Furthermore, by removing the autoencoders and employing Convolutional RNNs [75], we can enable the application of the framework in dynamical systems that do not require the existence of a low-dimensional manifold in the dynamics.

Recently proposed hierarchical deep learning time-steppers [76], reduced-order propagators on the latent space [77], and Autoformer networks [78] demonstrate promising results in PDEs and other complex time-series data. These algorithms can be employed as efficient macro propagators. Having a very fast macro propagator opens space for more advanced techniques, such as planning optimal actions in reinforcement learning [79].

Likewise, if the application allows it, we could detect dynamic regimes underrepresented in the data by simulating many steps in advance and looking at the future prediction uncertainty to determine if we should perform a macro-to-micro transition early. An additional network could be trained to estimate the decoder reconstruction error given the current latent state. Combined with the macro propagator, the macro-to-micro transition criteria could be based on the joint uncertainty of the ensemble and this reconstruction error.

AdaLED attempts to solve a non-convex optimization problem in an online fashion. Benchmarking against linear online learning frameworks like DMD [50,51] is left for future work. Moreover, linear online learning frameworks can be extended with the uncertainty estimation mechanism and online validation phases of AdaLED to enhance the robustness and increase speed-up.

Furthermore, we plan to investigate improved scheduling and refined control of AdaLED cycles and the micro–macro transitions to reduce the total number of time steps performed in the micro-scale and to provide more control over the trade-off between the adaptivity versus speed-up.

Finally, recent research [80] suggests that even though deep ensembles improve upon other uncertainty quantification methods [81–83], they may still become over-confident outside the training domain. In this direction, AdaLED can benefit from novel algorithms for uncertainty quantification of supervised learning algorithms [84].

Declaration of competing interest

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

Code availability

The source code will be made readily available at https://github.com/cselab/adaled upon publication.

Acknowledgments

We acknowledge support from The European High-Performance Computing Joint Undertaking (EuroHPC) Grant DCoMEX (956201-H2020-JTI-EuroHPC-2019-1), the Air Force Office of Scientific Research (MURI grant no. FA9550-21-1-005) and the Department of Energy (Grant DE-SC0022199). We gratefully acknowledge the service and computing resources from the Swiss National Supercomputing Centre (CSCS) under projects s930 and s1160.

We would like to thank Pascal Weber (ETHZ) for several useful discussions.

Appendix A. Macro propagator LSTMs

The macro propagator of AdaLED is an ensemble of multi-layer long short-term memory (LSTM) [85] recurrent neural networks (RNNs). Given an input state $\boldsymbol{\xi}_t \in \mathbb{R}^{d_\xi}$, the current hidden state $\mathbf{h}_t \in \mathbb{R}^{d_h}$ and the cell state $\mathbf{c}_t \in \mathbb{R}^{d_h}$,

each layer of each network computes the next hidden state $\mathbf{h}_{t+\Delta t}$ and the cell state $\mathbf{c}_{t+\Delta t}$ as follows (layer and ensemble notation omitted for brevity):

$$\begin{aligned}
\mathbf{i}_{t+\Delta t} &= \sigma(\mathbf{W}_i[\boldsymbol{\xi}_t, \mathbf{h}_t] + \mathbf{b}_i),\\
\mathbf{f}_{t+\Delta t} &= \sigma(\mathbf{W}_f[\boldsymbol{\xi}_t, \mathbf{h}_t] + \mathbf{b}_f),\\
\mathbf{g}_{t+\Delta t} &= \tanh\left(\mathbf{W}_g[\boldsymbol{\xi}_t, \mathbf{h}_t] + \mathbf{b}_g\right),\\
\mathbf{o}_{t+\Delta t} &= \sigma(\mathbf{W}_o[\boldsymbol{\xi}_t, \mathbf{h}_t] + \mathbf{b}_o),\\
\mathbf{c}_{t+\Delta t} &= \mathbf{f}_{t+\Delta t} \odot \mathbf{c}_t + \mathbf{i}_{t+\Delta t} \odot \mathbf{g}_{t+\Delta t},\\
\mathbf{h}_{t+\Delta t} &= \mathbf{o}_{t+\Delta t} \odot \tanh\left(\mathbf{c}_{t+\Delta t}\right),
\end{aligned} \tag{A.1}$$

where $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$, and $\mathbf{o}_t$ are input, forget, cell, and output gates, respectively. Matrices $\mathbf{W}_i$, $\mathbf{W}_f$, $\mathbf{W}_g$, $\mathbf{W}_o \in \mathbb{R}^{d_h \times (d_h + d_z)}$, and vectors $\mathbf{b}_i$, $\mathbf{b}_f$, $\mathbf{b}_g$, $\mathbf{b}_o \in \mathbb{R}^{d_h}$ are trainable parameters. Square brackets $[\dots]$ denote concatenation, σ the sigmoid function, and $\odot$ element-wise multiplication.

The input state $\boldsymbol{\xi}_t^{(1)}$ of the first layer is equal to $\boldsymbol{\xi}_t^{(1)} = [\mathbf{z}_t, \mathbf{q}_t, \mathbf{f}_t]$, where $\mathbf{z}_t \in \mathbb{R}^{d_z}$ is the latent state, $\mathbf{q}_t \in \mathbb{R}^{d_q}$ the quantities of interest, and $\mathbf{f}_t \in \mathbb{R}^{d_F}$ the external forcing. In the Van der Pol oscillator (VdP) study, $\mathbf{z}_t = \mathbf{z}(t) = [x(t), y(t)]$, and $\mathbf{f}_t = f(t) = \mu(t)$. In the reaction–diffusion case study, $\mathbf{z}_t$ is the output of the autoencoder, and $f_t = d(t)$. In the CFD study, $\mathbf{z}(t)$ is the output of the (multiresolution) autoencoder, $\mathbf{q}(t) = \alpha_F \mathbf{F}_{\text{cyl}}(t)$ the scaled force on the cylinder, and $f(t)$ the normalized Reynolds number $f(t) = \widetilde{\text{Re}}(t) = 2.4\text{Re}(t)/\text{Re}_{\max} - 1.2$. In studies with $\text{Re} \leq 1000$, $\text{Re}_{\max} = 1000$, and in studies with Re up to 1200, $\text{Re}_{\max} = 1200$. For other layers $l \geq 2$, the input state $\boldsymbol{\xi}_t^{(l)}$ is equal to the previous layer's hidden state $\mathbf{h}_{t+\Delta t}^{(l-1)}$.

The residuals $\Delta\mathbf{w}(t) = \mathbf{w}(t + \Delta t) - \mathbf{w}(t)$, $\mathbf{w}(t) = [\mathbf{z}(t), \mathbf{q}(t)]$ and uncertainties $\boldsymbol{\sigma}_{\Delta w}(t)$ are given by

$$\begin{aligned}
\Delta\mathbf{w}(t) &= \mathbf{W}_w \mathbf{h}_{t+\Delta t}^{(L)} + \mathbf{b}_w,\\
\boldsymbol{\sigma}_{\Delta\mathbf{w}}^2(t) &= \text{SoftPlus}_\epsilon(\mathbf{W}'_\sigma\ \text{CELU}(\mathbf{W}_\sigma \mathbf{h}_{t+\Delta t}^{(L)} + \mathbf{b}_\sigma) + \mathbf{b}'_\sigma),
\end{aligned} \tag{A.2}$$

where $\mathbf{h}^{(L)}$ is the hidden state of the final layer L. Matrices $\mathbf{W}_w \in \mathbb{R}^{(d_z+d_q)\times d_h}$, $\mathbf{W}_w \in \mathbb{R}^{d_\sigma \times d_h}$ and $\mathbf{W}'_\sigma \in \mathbb{R}^{(d_z+d_q)\times d_\sigma}$, and biases $\mathbf{b}_w, \mathbf{b}'_\sigma \in \mathbb{R}^{d_z+d_q}$ and $\mathbf{b}_\sigma \in \mathbb{R}^{d_\sigma}$, with $d_\sigma = 100$, are trainable parameters. Functions $\text{CELU}(x) = \max(0, x) + \min(0, \exp(x) - 1)$ and $\text{SoftPlus}_\epsilon(x) = \log(1 + \exp(x)) + \epsilon$, with $\epsilon = 10^{-6}$, are nonlinearities. The parameters for $\boldsymbol{\sigma}_{\Delta\mathbf{w}}$ are trained separately from the rest, as explained in Section 2.3. Concretely, the parameters $\mathbf{W}_\sigma$, $\mathbf{W}'_\sigma$, $\mathbf{b}_\sigma$ and $\mathbf{b}'_\sigma$ constitute the trainable parameters $\boldsymbol{\theta}_\sigma$, whereas other parameters constitute $\boldsymbol{\theta}_\mu$.

In the ensemble, each LSTM network, augmented with the additional layers for computing $\Delta\mathbf{w}(t)$ and $\boldsymbol{\sigma}_{\Delta w}^2(t)$, is trained separately with its own trainable parameters, starting from its own randomly initialized values. The networks are trained using the Adam optimizer [86] with backpropagation through time [53] to minimize the trajectory sum of per-state losses described in Section 2.3. Finally, in the VdP study, $L = 3$ layers with hidden state size of $d_h = 32$ (per layer) were used, amounting to 8176 trainable parameters in total. In the reaction–diffusion case study, $L = 2$ layers with $d_h = 64$ were used, amounting to 60 308 parameters (for $d_z = 8$). In the CFD case study, $L = 2$ layers with $d_h = 32$ were used, amounting to 20 944 parameters (for $d_z = 16$).

Appendix B. Details of the van der pol oscillator case study

The Van der Pol oscillator enters a limit cycle given a fixed μ. The limit cycles for different values of μ are shown in Fig. B.1.

The details of the three cases of $\mu(t)$ are as follows. In the $\mu_{\text{ALT}}(t)$ case, μ alternates between values $\mu = 1$ and $\mu = 3$ every 50 000 time steps. In the $\mu_{\text{RAND}}(t)$ case, μ changes between values 1.96, 1.23, 2.80, 2.34, 1.61, 2.57, 1.49, 3.00, 1.69, and 1.00, at the same rate as $\mu_{\text{ALT}}(t)$. Finally, the $\mu_{\text{BROWN}}(t)$ profile is computed by smoothing $\mu_{\text{RAND}}(t)$ and adding Brownian noise to it:

$$\begin{aligned}
&\mu_{\text{BROWN}}(t) = \mu_{\text{RAND}}(t) + \alpha(\mu_{\text{BROWN}}(t - \Delta t) - \mu_{\text{RAND}}(t)) + \beta\epsilon(t), \quad \epsilon(t) \sim \mathcal{U}([-1, 1]),\\
&\mu_{\text{BROWN}}(0) = \mu_{\text{RAND}}(0),
\end{aligned} \tag{B.1}$$

with $\alpha = \exp(-\Delta t/200)$, $\Delta t = 0.1$ and $\beta = 0.005$.

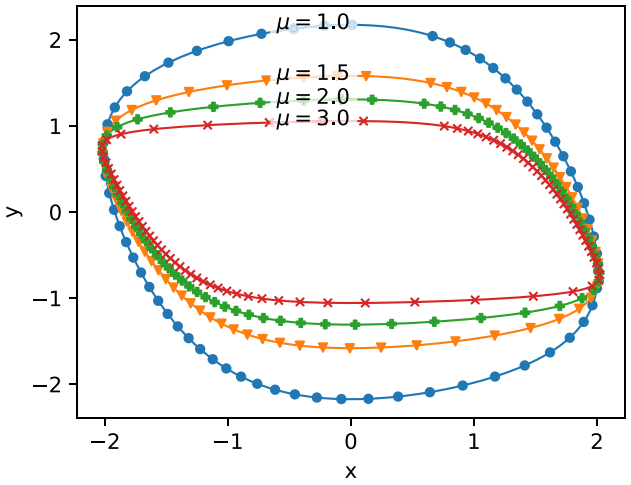

Fig. B.1. The limit cycles of the Van der Pol oscillator.

Table B.1
Van der Pol oscillator hyper-parameter study and final parameters in **bold**. Percentages on the right denote the highest achieved macro utilization.

Parameter	Search space	Comment
learning rate	LogUniform(0.0002, 0.02)	**0.002**
batch size	{8, **16**, 32, 64}	
LSTM hidden state size d_h	{8, 16, **32**}	
number of LSTM layers L	{1, 2, **3**, 4}	
adversarial training?	{**no**, yes}	no significant effect
$\mu(t)$ as part of input?	{no, **yes**}	∼54% vs ∼70%

B.1. Hyper-parameter study

The search space of hyper-parameters and their selected values are shown in Table B.1. The study was performed on 1536 samples of hyper-parameter values, randomly selected in the listed ranges, optimizing for the total macro utilization and average online validation error E. Each simulation was run for 200 000 time steps, with μ alternating between 1.5 and 2.5 every 25 000 time steps. Thresholds of $E_{\max} = 0.14$ and $\sigma_{\max} = 0.14$ were used. The highest macro utilization achieved was 70%. As the final hyper-parameter set (Table B.1), we selected a Pareto-optimal sample that achieves 68% macro utilization and average online validation error E of 0.008. We further explored adversarial training [55,87], but it did not affect the results noticeably. Moreover, we tested how significantly better the network is with $\mu(t)$ as part of the input compared to not having access to $\mu(t)$. The results show that the macro propagator the acceptance rate and the total utilization are still high (∼54%) without providing $\mu(t)$ to the network. However, naturally, in that case, the uncertainty of the macro propagator's prediction is insensitive to changes of $\mu(t)$ during the macro-only stage.

B.2. Dependence of error on thresholds and ensemble size

The testing error E can be decreased with a stricter uncertainty threshold $\sigma_{\max}$ or with a larger ensemble size K. To analyze the extent of their effect on E, we run the $\mu_{\text{ALT}}(t)$ case for varying $\sigma_{\max}$ and K. The threshold $E_{\max}$ is fixed to 0.1. The top plot in Fig. B.2 shows the distributions of mean cycle prediction errors $E^c_{\max}$ (the average over macro-only steps of a cycle c) for varying $\sigma_{\max}$ and K. We notice that, for sufficiently small $\sigma_{\max}$, the error drops

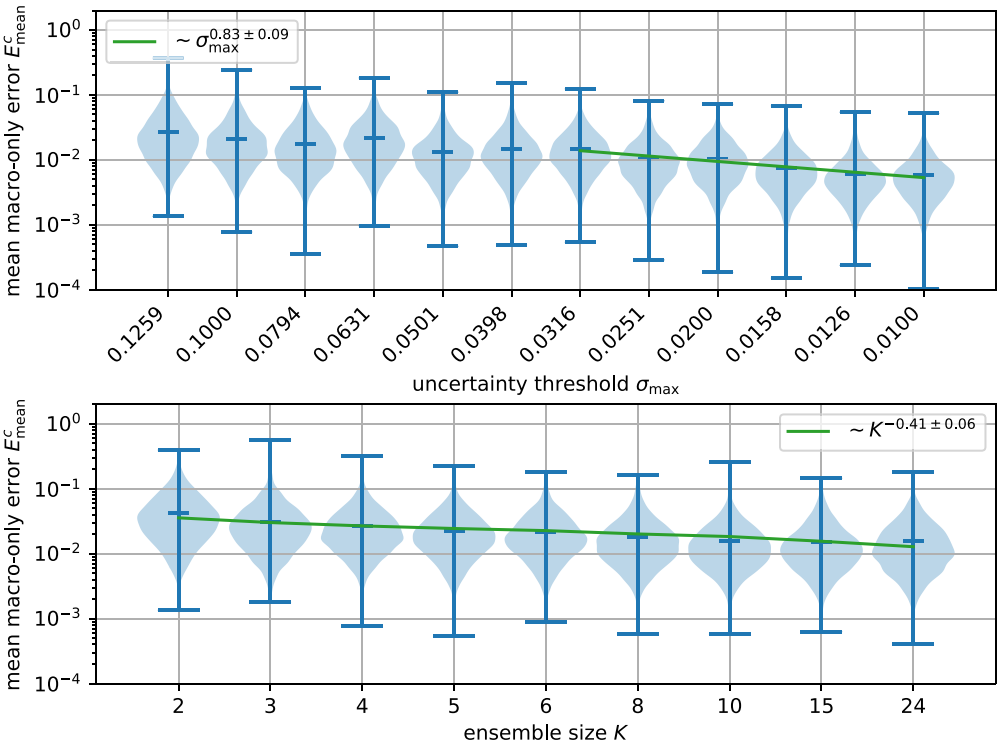

Fig. B.2. Dependence of the prediction error on the uncertainty threshold $\sigma_{\max}$ (top) and the ensemble size K (bottom), in the Van der Pol oscillator case study (Appendix B.2). Each violin plot represents one run and shows the distribution of mean macro-only prediction errors along the AdaLED cycles.

approximately linearly with respect to $\sigma_{\max}$. This trend can be explained through dimensionality analysis. Namely, E and $\sigma_{\max}$ are quantities of the same units. The bottom plot of Fig. B.2 shows that E drops approximately as $1/\sqrt{K}$, which is in accordance with the central limit theorem. Concretely, if we assume that each individual LSTM produces the correct trajectory up to the noise of zero bias, then the noise cancels out at the rate of $1/\sqrt{K}$.

It should be noted, however, that although stricter $\sigma_{\max}$ improves error, it decreases macro utilization η. For example, for $\sigma_{\max} = 0.1$, 0.02 and 0.01, the macro utilization η is 60%, 29% and 18%, respectively. Likewise, increasing the ensemble size from $K = 5$ to $K = 20$ reduces the error by $\sim 2\times$, but increases the total training time by $4\times$, assuming a fixed number of epochs. Thus, depending on the situation and objectives, decreasing $\sigma_{\max}$ and increasing K may or may not be favorable.

Appendix C. Details of the reaction–diffusion study

The initial condition of the system is given by [59]

$$\begin{aligned} u(x, y, 0) &= \tanh\left(\sqrt{x^2+y^2}\cos\left(\text{atan2}(y, x) - \sqrt{x^2+y^2}\right)\right), \\ v(x, y, 0) &= \tanh\left(\sqrt{x^2+y^2}\sin\left(\text{atan2}(y, x) - \sqrt{x^2+y^2}\right)\right). \end{aligned} \tag{C.1}$$

Eq. (10) is integrated in time using the fourth-order Runge–Kutta-Fehlberg integration scheme. A second-order centered stencil with zero von Neumann boundary conditions is used for the diffusion term.

The capacity of the dataset is set to 1024 trajectories, each having 24 time steps. The hyper-parameters, including the latent space dimension, RNN hidden state size, number of layers, batch size, learning rate, the amount of training

Table C.1
The architecture of the convolutional autoencoder for the reaction–diffusion case study. All convolutional layers use `padding_mode=replicate`. Batch normalization layers use the default parameters from PyTorch.

ID	Shape	Layer
	$2 \times 96 \times 96$	Input
1	$16 \times 96 \times 96$	Conv(2, 16, kernel_size=5, padding=2)
1	$16 \times 96 \times 96$	BatchNorm()
2	$16 \times 48 \times 48$	AvgPool(kernel_size=2, stride=2)
3	$16 \times 48 \times 48$	CELU()
4	$16 \times 48 \times 48$	Conv(16, 16, kernel_size=5, padding=2)
1	$16 \times 48 \times 48$	BatchNorm()
5	$16 \times 24 \times 24$	AvgPool(kernel_size=2, stride=2)
6	$16 \times 24 \times 24$	CELU()
7	$16 \times 24 \times 24$	Conv(16, 16, kernel_size=5, padding=2)
1	$16 \times 24 \times 24$	BatchNorm()
8	$16 \times 12 \times 12$	AvgPool(kernel_size=2, stride=2)
9	$16 \times 12 \times 12$	CELU()
13	$16 \times 12 \times 12$	Conv(16, 16, kernel_size=5, padding=2)
14	$16 \times 6 \times 6$	AvgPool(kernel_size=2, stride=2)
15	$16 \times 6 \times 6$	CELU()
16	576	Flatten()
17	$d_z = 8$	Linear()
18	$d_z = 8$	Tanh()
	$d_z = 8$	$\mathbf{z}$
1	576	Linear()
2	$16 \times 6 \times 6$	ViewLayer()
3	$16 \times 12 \times 12$	Upsample(scale_factor=2.0, mode=bilinear))
4	$16 \times 12 \times 12$	Conv(16, 16, kernel_size=3, padding=1)
1	$16 \times 12 \times 12$	BatchNorm()
8	$16 \times 12 \times 12$	CELU()
9	$16 \times 24 \times 24$	Upsample(scale_factor=2.0, mode=bilinear))
10	$16 \times 24 \times 24$	Conv(16, 16, kernel_size=5, padding=2)
1	$16 \times 24 \times 24$	BatchNorm()
11	$16 \times 24 \times 24$	CELU()
12	$16 \times 48 \times 48$	Upsample(scale_factor=2.0, mode=bilinear))
13	$16 \times 48 \times 48$	Conv(16, 16, kernel_size=5, padding=2)
1	$16 \times 48 \times 48$	BatchNorm()
14	$16 \times 48 \times 48$	CELU()
15	$16 \times 96 \times 96$	Upsample(scale_factor=2.0, mode=bilinear))
16	$2 \times 96 \times 96$	Conv(16, 2, kernel_size=5, padding=2)
17	$2 \times 96 \times 96$	Tanh()
	50K	total number of parameters

per AdaLED cycle, and the thresholds $E_{\max}$ and $\sigma_{\max}$ are all hand-tuned. The autoencoder is composed of an encoder and a decoder, each a 4-layer convolutional neural network with 16 channels per layer. The autoencoder architecture is provided in Table C.1. The values $(\mathbf{u}, \mathbf{v})$, which span the range $[-1, 1]$, are downscaled by a factor of 1.1 before entering the encoder and upscaled back at the end of the decoder. A learning rate of 0.001 and a batch size of 64 are used for both the autoencoder and RNNs. The training is performed in partial epochs, with the autoencoder trained on 6.25% of states in the dataset and the RNNs trained on 12.5% of stored trajectories after each AdaLED cycle.

Appendix D. Details of the flow behind the cylinder study

To trigger vortex shedding, we add a short symmetry-breaking vertical movement at the start of the simulation:

$$u_y^s(t) = e^{-\alpha t} \sin(\beta t)\, u_{y0}^s, \tag{D.1}$$

where $\alpha = 100$, $\beta = 200$ and $u_{y0}^s = 0.05d$.

The simulations were performed using `CubismAMR` [64], an adaptive mesh refinement (AMR) CPU–GPU hybrid C++ code for solving the incompressible Naiver-Stokes equations. To use it within AdaLED and from Python, we added Python bindings [88], APIs for controlling the execution of the simulation, and APIs for exporting and importing the state. The existing coarse-fine AMR interpolation schemes were reused for exporting and importing the state as a uniform grid.

The force $\mathbf{F}_{\text{cyl}}$ that the fluid exerts on the cylinder is given by the sum of the pressure and viscous forces and is provided by `CubismAMR`:

$$\begin{aligned} \mathbf{F}_{\text{cyl}} &= \mathbf{F}_{\text{p}} + \mathbf{F}_{\text{v}}, \\ \mathbf{F}_{\text{p}} &= \oiint -p\mathbf{n}\,\mathrm{d}S\,, \\ \mathbf{F}_{\text{v}} &= \nu\rho \oiint (\nabla\mathbf{u} + \nabla\mathbf{u}^{\intercal}) \cdot \mathbf{n}\,\mathrm{d}S\,, \end{aligned} \tag{D.2}$$

where S is the surface of the cylinder, and $\mathbf{n}$ the outward normal vector.

D.1. Autoencoder for the CFD state

To achieve high speed-ups, the macro propagator is not operating on the high-dimensional micro state $\mathbf{u} \in \mathbb{R}^{d_v}$, $d_v = 512 \times 1024 \times 2 \approx 10^6$ directly, but on a smaller low-dimensional latent state $\mathbf{z} \in \mathbb{R}^{d_z}$, with $d_z \sim 10$. The assumption is that this transition can indeed be performed: while we need high resolution to simulate the flow dynamics accurately and to acquire accurate forces $\mathbf{F}_{\text{cyl}}$, the actual dimensionality of the dynamics may be low.

To compress the velocity field $\mathbf{u}$ to the latent state $\mathbf{z}$, we use an autoencoder based on convolutional neural networks (CNNs). Instead of training the autoencoder to naively reproduce $\mathbf{u}$ by utilizing a simple (relative) MSE of $\mathbf{u}$, we take into account the characteristics of the fluid dynamical system: (i) physically essential quantities are also the spatial derivatives of the velocity (see Eq. (12)) and the vorticity $\omega = (\nabla \times \mathbf{u})_z$, (ii) the flow is incompressible, hence the divergence must be zero ($\nabla \cdot \mathbf{u} = 0$), (iii) we assume that the flow far from the cylinder requires smaller reconstruction accuracy compared to the flow around the cylinder.

If the autoencoder is trained only to minimize the MSE of $\mathbf{u}$ while ignoring the value of derivatives, the reconstructed $\mathbf{u}$ would have high spatial noise, resulting in inaccurate local derivatives and locally high vorticity $\omega = \omega_z = (\nabla \times \mathbf{u})_z$. This noisy vorticity would cause unnecessary mesh refinement in the CFD solver used in this study [64], which uses adaptive non-uniform mesh and magnitude of local vorticity as the mesh refinement and coarsening criterion. We extend the loss function with a relative L_1 vorticity reconstruction error to ensure the reconstructed vorticity is low where it originally is low. This helps reduce the mesh size by about 10%–15% compared to having no vorticity loss, and reduces the performance degradation that would partially cancel out the benefit of AdaLED.

Non-zero divergence $\nabla \cdot \mathbf{u}$ can be prevented entirely as a hard constraint by predicting the stream function ψ instead of the velocity field $\mathbf{u}$ [63]. The velocity field $\mathbf{u}$ is then given as:

$$u_x = \frac{\partial \psi}{\partial y}, \quad u_y = -\frac{\partial \psi}{\partial x}. \tag{D.3}$$

In its discretized form, the derivatives for computing $\mathbf{u}$ from ψ and ω from $\mathbf{u}$ are computed using the 2nd order accurate centered stencil. For example, for a field f and a grid spacing of Δx, the x-derivative is given as:

$$\left.\frac{\partial f}{\partial x}\right|_{ij} = \frac{f_{i,j+1} - f_{i,j-1}}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{D.4}$$

Thus, taking derivatives removes one cell from each side of each dimension. To account for that, ψ is predicted with one cell of padding.

Finally, we want to prioritize reducing the reconstruction loss in the vicinity of the cylinder because this part affects the force $\mathbf{F}_{\text{cyl}}$ and because any error there will propagate to the rest of the flow. Moreover, the vortex street behind the cylinder is relatively smooth and does not require high resolution. We take advantage of these two observations and use a multiresolution autoencoder with two encoder–decoder pairs: One for the whole domain downsampled to half the resolution ($\mathbf{u}^{(1)}$), and one focusing on the subdomain around the cylinder at full resolution ($\mathbf{u}^{(2)}$). The two pairs operate independently and their compressed latent state $\mathbf{z}^{(k)} \in \mathbb{R}^{d_z^{(k)}}$ are concatenated into the final latent state $\mathbf{z} \in \mathbb{R}^{d_z}$, $d_z = d_z^{(1)} + d_z^{(2)}$. The details are explained in the following section.

D.2. Multiresolution autoencoders

When building an autoencoder for 2D (or 3D) arrays, in cases where different parts of the flow exhibit different features and where not all parts require the same level of accuracy, we may benefit from combining multiple autoencoders operating at different spatial resolutions into a single one. This enables us to reduce memory requirements and improve computational efficiency and training accuracy (accuracy is positively affected by improved processing speed and potentially from the benefits of a specialized architecture). This section describes how such *multiresolution autoencoders* can be constructed. For simplicity, we focus on autoencoders reconstructing a 2D scalar array $\boldsymbol{\phi} \in \mathbb{R}^{H\times W}$ using two encoder–decoder pairs. The method can be easily generalized to vector arrays, to more than two encoder–decoder pairs, and higher-dimensional arrays.

For each encoder–decoder pair AE_k, $k \in \{1, 2\}$, we define a downsampling operation $\mathbf{D}^{(k)} : \mathbb{R}^{H\times W} \to \mathbb{R}^{H^{(k)}\times W^{(k)}}$ that converts the full-resolution array $\boldsymbol{\phi}$ into a smaller array $\boldsymbol{\phi}^{(k)} = \mathbf{D}^{(k)}(\boldsymbol{\phi})$ that AE_k will operate on. In this case study, AE_1 is used to reconstruct the whole domain at half the resolution ($H^{(1)} = H/2$, $W^{(1)} = W/2$), whereas AE_2 is used for the detailed part of some size $H^{(2)} \times W^{(2)}$ around the cylinder. Functions $\mathbf{D}^{(1)}$ and $\mathbf{D}^{(2)}$ are thus given as:

$$\begin{aligned} D_{ij}^{(1)}(\boldsymbol{\phi}) &= \frac{1}{4}\left(\boldsymbol{\phi}_{2i,2j} + \boldsymbol{\phi}_{2i,2j+1} + \boldsymbol{\phi}_{2i+1,2j} + \boldsymbol{\phi}_{2i+1,2j+1}\right), && 0 \le i < H^{(1)}, 0 \le j < W^{(1)} \\ D_{ij}^{(2)}(\boldsymbol{\phi}) &= \boldsymbol{\phi}_{i_0+i,\, j_0+j}, && 0 \le i < H^{(2)}, 0 \le j < W^{(2)} \end{aligned} \tag{D.5}$$

where i_0 and j_0 are offsets of $\boldsymbol{\phi}^{(2)}$ with respect to $\boldsymbol{\phi}$ (indexing is 0-based).

The total reconstruction loss is defined as a weighted sum of the reconstruction losses of each individual AE_k:

$$\ell(\tilde{\boldsymbol{\phi}}, \boldsymbol{\phi}) = w^{(1)}\ell^{(1)}(\tilde{\boldsymbol{\phi}}^{(1)}, \boldsymbol{\phi}^{(1)}) + w^{(2)}\ell^{(2)}(\tilde{\boldsymbol{\phi}}^{(2)}, \boldsymbol{\phi}^{(2)}), \tag{D.6}$$

where $w^{(k)}$ are weight factors. Since the AE_ks are independent, the weights $w^{(k)}$ effectively determine the relative learning rate between them. For simplicity, we take $w^{(1)} = w^{(2)} = 1$.

The individual losses $l^{(k)}$ take into consideration that we do not want to waste the limited expressiveness of AE_1 on reconstructing the part that AE_2 is already focusing on. Furthermore, to reduce the boundary effects (at which the reconstruction might be poor), we also want to exclude the edges from the reconstruction loss of AE_2. We, thus, use weighted (relative) reconstruction losses:

$$\ell^{(k)}\left(\tilde{\boldsymbol{\phi}}^{(k)}, \boldsymbol{\phi}^{(k)}\right) = \frac{\sum_{ij} \alpha_{ij}^{(k)} \left(\tilde{\boldsymbol{\phi}}_{ij}^{(k)} - \boldsymbol{\phi}_{ij}^{(k)}\right)^2}{\sum_{ij} \alpha_{ij}^{(k)} \left(\boldsymbol{\phi}_{ij}^{(k)}\right)^2 + W^{(k)}H^{(k)}\epsilon_\phi}$$

for the relative MSE loss, or

$$\ell^{(k)}\left(\tilde{\boldsymbol{\phi}}^{(k)}, \boldsymbol{\phi}^{(k)}\right) = \frac{\sum_{ij} \alpha_{ij}^{(k)} \left|\tilde{\boldsymbol{\phi}}_{ij}^{(k)} - \boldsymbol{\phi}_{ij}^{(k)}\right|}{\sum_{ij} \alpha_{ij}^{(k)} \left|\boldsymbol{\phi}_{ij}^{(k)}\right| + W^{(k)}H^{(k)}\epsilon_\phi}$$

for the relative L_1 loss. Here, field $\tilde{\boldsymbol{\phi}}^{(k)}$ denotes the autoencoder reconstruction, $\boldsymbol{\phi}^{(k)}$ the input and the target, and $\epsilon_\phi > 0$ a normalization offset for preventing diverging gradients. The weight factors $\boldsymbol{\alpha}^{(k)}$ are selected such that the center of the cylinder does not affect the loss of AE_1 and that the edge of the second subdomain does not affect the loss of AE_2:

$$\begin{aligned} \boldsymbol{\alpha}^{(1)} &= 1 - \mathbf{D}^{(1)}(\mathbf{S}(d^{(1)})), \\ \boldsymbol{\alpha}^{(2)} &= \mathbf{D}^{(2)}(\mathbf{S}(d^{(2)})), \end{aligned} \tag{D.7}$$

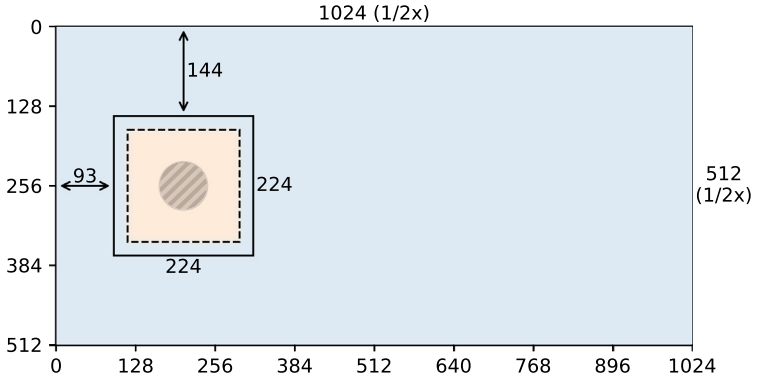

Fig. D.1. The geometry of the multiresolution autoencoder with resolution of AE_2 equal to 224 × 224. The solid inner box represents the subdomain that AE_2 operates on, the dashed line the distance $d^r = 22$ from the inner subdomain boundary (where $\boldsymbol{\beta}^{(1)} = \boldsymbol{\beta}^{(2)} = 0.5$). Colors represent the blending contributions ($\boldsymbol{\beta}^{(1)} > 0.5$ marked as blue, $\boldsymbol{\beta}^{(2)} > 0.5$ with orange), and circle the cylinder. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

where $\mathbf{S} : \mathbb{R} \to \mathbb{R}^{H\times W}$ is a 2D smoothed rectangular function:

$$\begin{aligned} S_{ij}(d) &= S\left(\frac{\min\{i' - i_0, i_1 - i'\} - d}{s}\right) \\ &\times S\left(\frac{\min\{j' - j_0, j_1 - j'\} - d}{s}\right), \\ i' &= i + \frac{1}{2}, \qquad \text{(for cell-centered values)} \\ j' &= j + \frac{1}{2}, \\ S(x) &= \frac{1}{1 + e^{-x}}, \quad \text{(sigmoid)} \end{aligned} \tag{D.8}$$

where (i_0, j_0) and $(i_1, j_1) = (i_0 + H^{(2)}, j_0 + W^{(2)})$ are the start and the end bounds of AE_2. Parameter $s > 0$ is a smoothing factor, and $d^{(1)}, d^{(2)} > 0$ the spatial margins.

The final reconstruction of $\boldsymbol{\phi} \in \mathbb{R}^{H\times W}$ from $\boldsymbol{\phi}^{(1)}$ and $\boldsymbol{\phi}^{(2)}$ is as well done in a weighted manner:

$$\begin{aligned} \boldsymbol{\phi} &= \boldsymbol{\beta}^{(1)} \odot \mathbf{U}(\boldsymbol{\phi}^{(1)}) + \boldsymbol{\beta}^{(2)} \odot \mathbf{U}(\boldsymbol{\phi}^{(2)}), \\ \boldsymbol{\beta}^{(1)} &= 1 - \mathbf{S}(d^r) \\ \boldsymbol{\beta}^{(2)} &= \mathbf{S}(d^r) \end{aligned} \tag{D.9}$$

where $\mathbf{U}^{(k)} : \mathbb{R}^{H^{(k)}\times W^{(k)}} \to \mathbb{R}^{H\times W}$ are upsampling operations, d^r the reconstruction margin, and operator $\odot$ the element-wise multiplication. In this case study, $\mathbf{U}^{(1)}$ is the upsampling operation with bilinear interpolation and $\mathbf{U}^{(2)}$ is a zero-padding operation.

We tested two variants of AE_2 that operate on different resolutions: 256 × 256 and 224 × 224. As shown in Section 5.1.3, the latter variant exhibited slightly better performance for large macro utilizations and was selected as part of the reference parameter set. The parameters are shown in Table D.1, and the geometry is visualized in Fig. D.1. By using margins $d^{(1)} > d^r > d^{(2)}$, we ensure that both AE_1 and AE_2 accurately reconstruct the part where the smoothed blending occurs ($0 < \beta_{ij}^{(k)} < 1$).

The multiresolution approach decreases the memory and storage requirements by 2.9× (for the 224 × 224 variant) and accelerates the training by approximately the same factor at a small cost of accuracy degradation. Concretely, the relative mean square error between the original velocity field $\mathbf{u}$ and the downsampled–upsampled $\mathbf{u}'$ is $\|\mathbf{u}' - \mathbf{u}\|_2^2 / \|\mathbf{u}\|_2^2 \approx 10^{-6}$.

Table D.1
Geometry and loss function parameters for multiresolution autoencoders, and memory usage per single $\mathbf{u}$ field in single precision.

Parameter	**Variant 1**		Variant 2
original resolution		1024×512	
AE_1 resolution		512×256	
AE_2 resolution	256×256		$\mathbf{224 \times 224}$
AE_2 begin	(77, 128)		(93, 144)
AE_2 end	(333, 384)		(317, 368)
weight $w^{(1)}$		1.0	
weight $w^{(2)}$		1.0	
margin $d^{(1)}$		30.0	
margin d^r		22.0	
margin $d^{(2)}$		14.0	
smoothing s		3.0	
original size of $\mathbf{u}$		4.00 MB	
reduced size of $\mathbf{u}$	1.50 MB		1.38 MB

D.3. Autoencoder summary and loss function

The discussion above is summarized in the following. The state $\mathbf{u}$ is stored in the dataset in two downsampled versions $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$, each handled by its own encoder–decoder pair AE_k. For each AE_k, $k \in \{1, 2\}$, the encoder k takes the downsampled velocity field $\mathbf{u}^{(k)} \in \mathbb{R}^{H^{(k)} \times W^{(k)} \times 2}$ as input and compresses it into a latent state $\mathbf{z}^{(k)} \in \mathbb{R}^{d_z^{(k)}}$. The decoder k takes the latent state $\mathbf{z}^{(k)}$ and decompresses it into the scalar field $\tilde{\psi}^{(k)}$ (with one cell of padding on each side of each dimension). Then, the reconstructed velocity $\tilde{\mathbf{u}}^{(k)}$ is computed from $\tilde{\psi}^{(k)}$ using Eq. (D.3). The reconstruction loss of AE_k is defined as a weighted sum of the relative MSE loss of $\mathbf{u}^{(k)}$ and the relative L_1 loss of vorticity $\boldsymbol{\omega}^{(k)}$ (notation (k) omitted in the following for brevity):

$$\begin{aligned} \ell^{(k)}(\tilde{\mathbf{u}}, \mathbf{u}) = \lambda_u \frac{\sum_{ij} \alpha_{ij} (\tilde{\mathbf{u}}_{ij} - \mathbf{u}_{ij})^2}{\sum_{ij} \alpha_{ij} \mathbf{u}_{ij}^2 + W^{(k)} H^{(k)} \epsilon_u} \\ + \lambda_\omega \frac{\sum_{ij} \alpha'_{ij} \|\tilde{\boldsymbol{\omega}}_{ij} - \boldsymbol{\omega}_{ij}\|_1}{\sum_{ij} \alpha'_{ij} \|\boldsymbol{\omega}_{ij}\|_1 + (W^{(k)} - 2)(H^{(k)} - 2)\epsilon_\omega}, \end{aligned} \tag{D.10}$$

$$\begin{aligned} \tilde{\boldsymbol{\omega}} = (\nabla \times \tilde{\mathbf{u}})_z, \quad \boldsymbol{\omega} = (\nabla \times \mathbf{u})_z, \\ \boldsymbol{\alpha}' = \boldsymbol{\alpha}_{1..H-2;1..W-2} \in \mathbb{R}^{(H-2)\times(W-2)}, \end{aligned} \tag{D.11}$$

where $\lambda_u = 1$ and $\lambda_\omega = 0.03$ are the weight factors, and $\epsilon_u = 0.01$ and $\epsilon_\omega = 0.7$ the normalization offsets used to avoid exploding gradients when training on initial states where $\mathbf{u} \approx \mathbf{0}$. The effect of λ_ω is visualized in Fig. D.2. Numbers ϵ_u and ϵ_ω were selected to match $\approx$25% of the mean $(\mathbf{u}_{ij}^{(1)})^2$ and the mean $\left|\omega_{ij}^{(1)}\right|$, respectively, for the developed flow at $\mathrm{Re} = 500$. The total loss $\ell(\dots)$ is defined as the weighted sum of the losses of AE_ks:

$$\ell(\tilde{\mathbf{u}}^{(1)}, \tilde{\mathbf{u}}^{(2)}, \mathbf{u}^{(1)}, \mathbf{u}^{(2)}) = w^{(1)} \ell^{(1)}(\tilde{\mathbf{u}}^{(1)}, \mathbf{u}^{(1)}) + w^{(2)} \ell^{(2)}(\tilde{\mathbf{u}}^{(2)}, \mathbf{u}^{(2)}), \tag{D.12}$$

where $w^{(1)} = w^{(2)} = 1$ are relative weights between AE_ks. Either when computing the online validation error E in Eq. (13) or when performing macro-to-micro transition, the full resolution velocity $\mathbf{u}$ is reconstructed by merging $\mathbf{u}^{(1)}$ and $\mathbf{u}^{(2)}$ as described in Eq. (D.9). The merging must be performed on velocities $\mathbf{u}$ and not on the stream function ψ. This is because the stream functions are defined up to an unspecified additive constant, making their merging impossible. Furthermore, by smoothly blending between two upscaled velocity fields (Eq. (D.9)), we ensure the spatial derivatives of $\mathbf{u}$ are smooth.

D.4. Hyper-parameters, the CNN architecture and training

Apart from the hyper-parameters listed in the autoencoder study in Section 5.1.2, other parameters, such as the batch size, were hand-tuned and are listed in Table D.2.

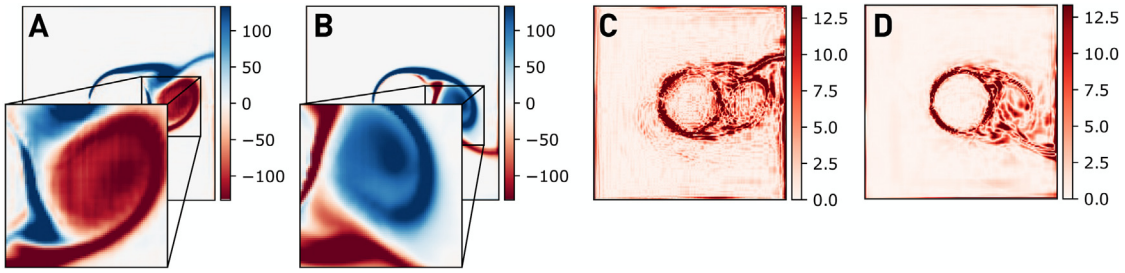

Fig. D.2. Reconstructed vorticity field (A and B) and its absolute error (C and D) for the AE_2, for $\lambda_\omega = 0$ (without vorticity loss, A and C) and $\lambda_\omega = 0.03$ (with vorticity loss, B and D).

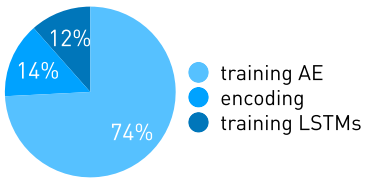

Fig. D.3. Fraction of the execution time of stages of the training in the flow behind the cylinder case.

Table D.2
Hand-tuned network hyper-parameters for the flow behind the cylinder study. Other parameters are listed in Section 5.1.3.

Hyper-parameter	Value
autoencoder batch size	8
LSTM batch size	8
LSTM hidden state size	32
number of LSTM layers	2
single vs double precision	single

The basis of the $\mathbf{u}$ autoencoder are two convolutional autoencoders, each operating on one downsampled array $\mathbf{u}^{(k)}$. The two autoencoders share the same architecture but are trained separately. Their final CNN architecture after the hyper-parameter study is shown in Table D.3.

The training is performed continuously in parallel with the simulation and AdaLED inference. In each epoch, the networks are trained on 12.5% of the dataset. An epoch consists of training the autoencoder, encoding the states to build a temporary dataset for LSTMs, and finally, training the LSTMs. The relative execution time of the three training stages is shown in Fig. D.3.

D.5. Latent trajectory

A section of the macro trajectory from the simulation from Section 5.1.1 is shown in Fig. D.4. The first 16 lines correspond to the latent states $\mathbf{z}(t)$ and the last two to the force $\mathbf{F}_{\text{cyl}}(t)$ (scaled with a factor of $\alpha_F = 7.2$). The total uncertainty σ is defined as the root square mean of all 18 uncertainties. It can be seen that the majority of the uncertainty comes from low-amplitude latent variables. The possibility of using weighted uncertainties, depending on the importance of each variable, is a topic of future research.

Table D.3
The architecture of the convolutional autoencoders for the flow behind the cylinder case study. All convolutional layers use `padding_mode=replicate`.

ID	AE #1 shape	AE #2 shape	Layer
	$2 \times 256 \times 512$	$2 \times 224 \times 224$	Input
1	$16 \times 256 \times 512$	$16 \times 224 \times 224$	Conv(2, 16, kernel_size=5, padding=2)
2	$16 \times 128 \times 256$	$16 \times 112 \times 112$	AvgPool(kernel_size=2, stride=2)
3	$16 \times 128 \times 256$	$16 \times 112 \times 112$	CELU()
4	$16 \times 128 \times 256$	$16 \times 112 \times 112$	Conv(16, 16, kernel_size=5, padding=2)
5	$16 \times 64 \times 128$	$16 \times 56 \times 56$	AvgPool(kernel_size=2, stride=2)
6	$16 \times 64 \times 128$	$16 \times 56 \times 56$	CELU()
7	$16 \times 64 \times 128$	$16 \times 56 \times 56$	Conv(16, 16, kernel_size=5, padding=2)
8	$16 \times 32 \times 64$	$16 \times 28 \times 28$	AvgPool(kernel_size=2, stride=2)
9	$16 \times 32 \times 64$	$16 \times 28 \times 28$	CELU()
13	$16 \times 32 \times 64$	$16 \times 28 \times 28$	Conv(16, 16, kernel_size=3, padding=1)
14	$16 \times 16 \times 32$	$16 \times 14 \times 14$	AvgPool(kernel_size=2, stride=2)
15	$16 \times 16 \times 32$	$16 \times 14 \times 14$	CELU()
16	8192	3136	Flatten()
17	$d_z^{(1)} = 8$	$d_z^{(2)} = 8$	Linear()
18	$d_z^{(1)} = 8$	$d_z^{(2)} = 8$	Tanh()
	$d_z^{(1)} = 8$	$d_z^{(2)} = 8$	$\mathbf{z}^{(i)}$
1	8192	3136	Linear()
2	$16 \times 16 \times 32$	$16 \times 14 \times 14$	ViewLayer()
3	$16 \times 32 \times 64$	$16 \times 28 \times 28$	Upsample(scale_factor=2.0, mode=bilinear))
4	$16 \times 32 \times 64$	$16 \times 28 \times 28$	Conv(16, 16, kernel_size=3, padding=1)
8	$16 \times 32 \times 64$	$16 \times 28 \times 28$	CELU()
9	$16 \times 64 \times 128$	$16 \times 56 \times 56$	Upsample(scale_factor=2.0, mode=bilinear))
10	$16 \times 64 \times 128$	$16 \times 56 \times 56$	Conv(16, 16, kernel_size=5, padding=2)
11	$16 \times 64 \times 128$	$16 \times 56 \times 56$	CELU()
12	$16 \times 128 \times 256$	$16 \times 112 \times 112$	Upsample(scale_factor=2.0, mode=bilinear))
13	$16 \times 128 \times 256$	$16 \times 112 \times 112$	Conv(16, 16, kernel_size=5, padding=2)
14	$16 \times 128 \times 256$	$16 \times 112 \times 112$	CELU()
15	$16 \times 256 \times 512$	$16 \times 224 \times 224$	Upsample(scale_factor=2.0, mode=bilinear))
16	$1 \times 258 \times 514$	$1 \times 224 \times 224$	Conv(16, 1, kernel_size=5, padding=3)
17	$2 \times 256 \times 512$	$2 \times 224 \times 224$	StreamFnToVelocity() (Eq. (D.3))
	171K	85K	total number of parameters

D.6. Generalization to other Reynolds number profiles

The hyper-parameters and thresholds used in simulations reported in Section 5.1.1 were fine-tuned for that specific Reynolds number profile of cycling between Re $=$ 600, 750 and 900, updated every 5000 time steps, as described in Section 5.1.3 (the hyper-parameter study used shorter simulations than the production runs). To test the generalization of hyper-parameters and thresholds to another Reynolds number profile, we simulate with Reynolds number alternating between 500 and 1000 every 10 000 time steps. The macro utilization η, velocity field error E, and the cylinder force error E_F are shown in Fig. D.5. Compared to the macro utilization of $\eta = 69\%$ (speed-up of $2.9\times$) in Section 5.1.1, here, the achieved utilization is 58% (speed-up of $2.1\times$). As before, the average velocity and cylinder force errors E and E_F are 1% and 5%. In this case, changing the setup resulted in smaller speed-ups. Thus, to achieve optimal performance, the hyper-parameters (particularly learning rates and thresholds) may have to be additionally fine-tuned if the simulation setup is updated.

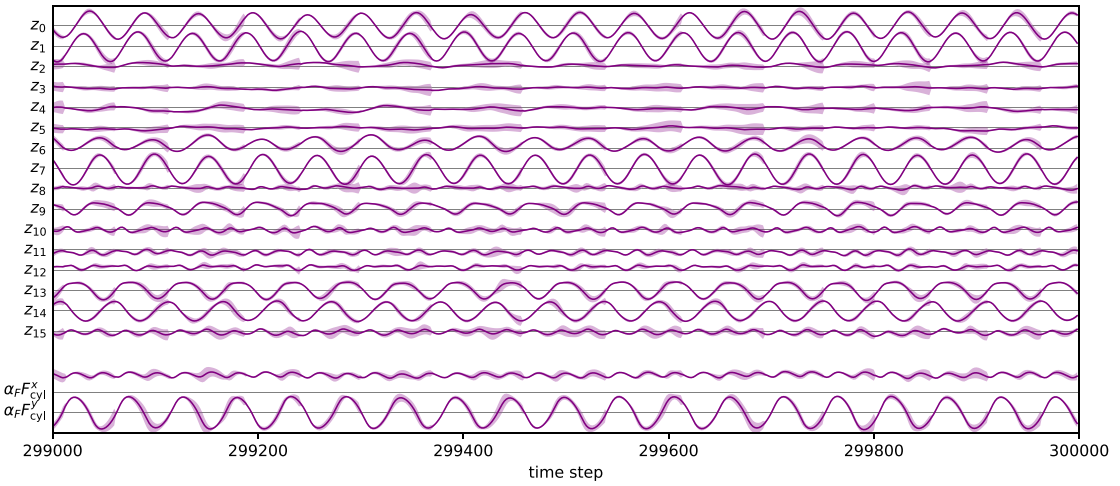

Fig. D.4. The trajectory $\mathbf{z}(t)$ and $\alpha_F \mathbf{F}_{\text{cyl}}(t)$ from the simulation from Section 5.1.1, as predicted by the ensemble. The solid line represents the ensemble mean prediction, and the faded region is the prediction uncertainty (the ensemble's standard deviation). For clarity, the uncertainties are enhanced by 8×. The numbers range between approx. −0.5 and 0.5. See Appendix D.5.

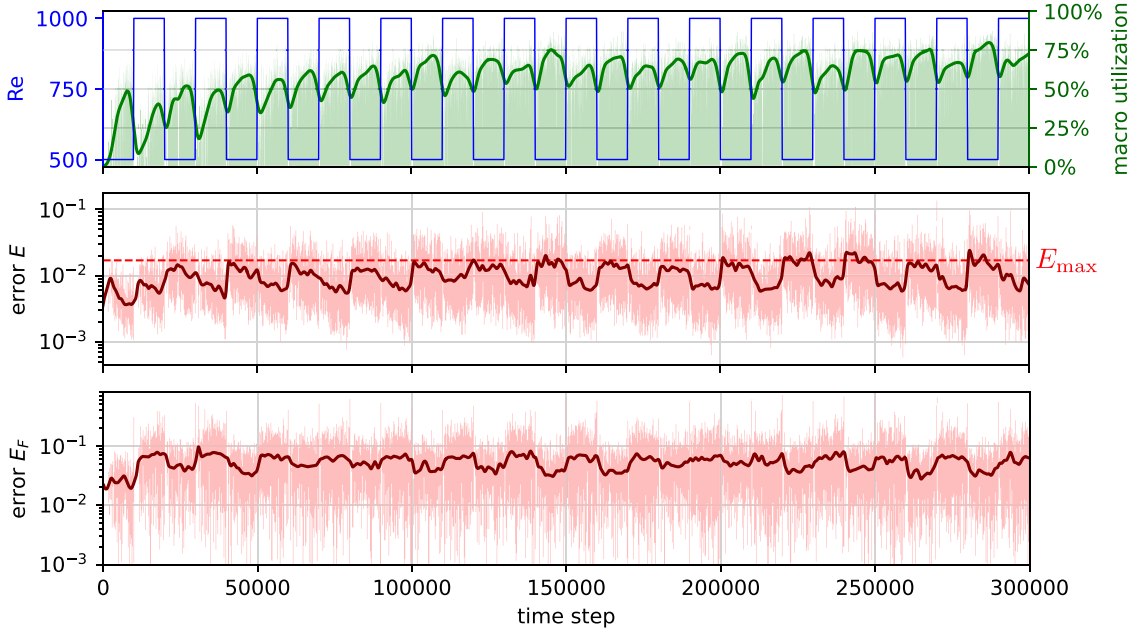

Fig. D.5. AdaLED performance on a flow behind cylinder simulation for $\text{Re}(t) \in \{500, 1000\}$, analogous to Fig. 9. Top: Reynolds number $\text{Re}(t)$ profile and the macro utilization η. Middle and bottom: validation errors of the velocity (E, Eq. (13)) and force on the cylinder (E_F, Eq. (14)). The per-step errors (faded red) alternate between low values at the beginning of the macro-only stage and higher errors at the end of the macro-only stage. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

References

[1] F. Lateef, Simulation-based learning: Just like the real thing, J. Emerg. Trauma Shock 3 (4) (2010) 348.

[2] V. Springel, S.D. White, A. Jenkins, C.S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, et al., Simulations of the formation, evolution and clustering of galaxies and quasars, Nature 435 (7042) (2005) 629–636.

[3] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, et al., Exascale deep learning for climate analytics, in: SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE, 2018, pp. 649–660.
[4] O. Ghattas, K. Willcox, Learning physics-based models from data: perspectives from inverse problems and model reduction, Acta Numer. 30 (2021) 445–554, http://dx.doi.org/10.1017/S0962492921000064.
[5] W. Gong, Q. Duan, J. Li, C. Wang, Z. Di, Y. Dai, A. Ye, C. Miao, Multi-objective parameter optimization of common land model using adaptive surrogate modeling, Hydrol. Earth Syst. Sci. 19 (5) (2015) 2409–2425.
[6] S. Verma, G. Novati, P. Koumoutsakos, Efficient collective swimming by harnessing vortices through deep reinforcement learning, Proc. Natl. Acad. Sci. 115 (23) (2018) 5849–5854.
[7] G. Novati, H.L. de Laroussilhe, P. Koumoutsakos, Automating turbulence modelling by multi-agent reinforcement learning, Nat. Mach. Intell. 3 (1) (2021) 87–96.
[8] S.S. Du, S.M. Kakade, R. Wang, L.F. Yang, Is a good representation sufficient for sample efficient reinforcement learning? 2019, arXiv preprint arXiv:1910.03016.
[9] M. Taufer, E. Deelman, R.F.d. Silva, T. Estrada, M. Hall, M. Livny, A roadmap to robust science for high-throughput applications: The developers' perspective, in: 2021 IEEE International Conference on Cluster Computing, CLUSTER, 2021, pp. 807–808, http://dx.doi.org/10.1109/Cluster48925.2021.00068.
[10] I.G. Kevrekidis, C.W. Gear, J.M. Hyman, P.G. Kevrekidis, O. Runborg, C. Theodoropoulos, et al., Equation-free, coarse-grained multiscale computation: enabling microscopic simulators to perform system-level analysis, Commun. Math. Sci. 1 (4) (2003) 715–762.
[11] C.R. Laing, T. Frewen, I.G. Kevrekidis, Reduced models for binocular rivalry, J. Comput. Neurosci. 28 (3) (2010) 459–476.
[12] Y. Bar-Sinai, S. Hoyer, J. Hickey, M.P. Brenner, Learning data-driven discretizations for partial differential equations, Proc. Natl. Acad. Sci. 116 (31) (2019) 15344–15349.
[13] E. Weinan, B. Engquist, et al., The heterognous multiscale methods, Commun. Math. Sci. 1 (1) (2003) 87–132.
[14] E. Weinan, B. Engquist, X. Li, W. Ren, E. Vanden-Eijnden, Heterogeneous multiscale methods: a review, Commun. Comput. Phys. 2 (3) (2007) 367–450.
[15] M. Tao, H. Owhadi, J.E. Marsden, Nonintrusive and structure preserving multiscale integration of stiff ODEs, SDEs, and Hamiltonian systems with hidden slow dynamics via flow averaging, Multiscale Model. Simul. 8 (4) (2010) 1269–1324.
[16] J.N. Kutz, S.L. Brunton, B.W. Brunton, J.L. Proctor, Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems, SIAM, 2016.
[17] R.R. Coifman, S. Lafon, Diffusion maps, Appl. Comput. Harmon. Anal. 21 (1) (2006) 5–30.
[18] P.R. Vlachas, J. Pathak, B.R. Hunt, T.P. Sapsis, M. Girvan, E. Ott, P. Koumoutsakos, Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics, Neural Netw. 126 (2020) 191–217.
[19] P.R. Vlachas, W. Byeon, Z.Y. Wan, T.P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. 474 (2213) (2018) 20170844.
[20] Z.Y. Wan, P. Vlachas, P. Koumoutsakos, T. Sapsis, Data-assisted reduced-order modeling of extreme events in complex dynamical systems, PLoS One 13 (5) (2018) e0197704.
[21] S. Brunton, B. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, 2019, arXiv preprint arXiv:1905.11075.
[22] R. Vinuesa, S.L. Brunton, Enhancing computational fluid dynamics with machine learning, Nat. Comput. Sci. 2 (6) (2022) 358–366.
[23] D. Kochkov, J.A. Smith, A. Alieva, Q. Wang, M.P. Brenner, S. Hoyer, Machine learning–accelerated computational fluid dynamics, Proc. Natl. Acad. Sci. 118 (21) (2021) e2101784118.
[24] P.R. Vlachas, G. Arampatzis, C. Uhler, P. Koumoutsakos, Multiscale simulations of complex systems by learning their effective dynamics, Nat. Mach. Intell. 4 (4) (2022) 359–366.
[25] P.R. Vlachas, Learning and Forecasting the Effective Dynamics of Complex Systems Across Scales (Ph.D. thesis), ETH Zurich, 2022.
[26] P.R. Vlachas, J. Zavadlav, M. Praprotnik, P. Koumoutsakos, Accelerated simulations of molecular systems through learning of effective dynamics, J. Chem. Theory Comput. 18 (1) (2021) 538–549.
[27] T. Wu, T. Maruyama, J. Leskovec, Learning to accelerate partial differential equations via latent global evolution, 2022, arXiv preprint arXiv:2206.07681.
[28] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer Graphics Forum, Vol. 38, No. 2, Wiley Online Library, 2019, pp. 71–82.
[29] F.J. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, 2018, arXiv preprint arXiv:1808.01346.
[30] K. Fukami, K. Hasegawa, T. Nakamura, M. Morimoto, K. Fukagata, Model order reduction with neural networks: Application to laminar and turbulent flows, SN Comput. Sci. 2 (6) (2021) 1–16.
[31] K. Stachenfeld, D.B. Fielding, D. Kochkov, M. Cranmer, T. Pfaff, J. Godwin, C. Cui, S. Ho, P. Battaglia, A. Sanchez-Gonzalez, Learned coarse models for efficient turbulence simulation, 2021, arXiv preprint arXiv:2112.15275.
[32] N. Geneva, N. Zabaras, Modeling the dynamics of PDE systems with physics-constrained deep auto-regressive networks, J. Comput. Phys. 403 (2020) 109056.
[33] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Phys. Fluids 33 (3) (2021) 037106.
[34] K. Hasegawa, K. Fukami, T. Murata, K. Fukagata, Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes, Theor. Comput. Fluid Dyn. 34 (4) (2020) 367–383.
[35] P. Pant, R. Doshi, P. Bahl, A. Barati Farimani, Deep learning for reduced order modelling and efficient temporal evolution of fluid simulations, Phys. Fluids 33 (10) (2021) 107101.

[36] H. Eivazi, H. Veisi, M.H. Naderi, V. Esfahanian, Deep neural networks for nonlinear model order reduction of unsteady flows, Phys. Fluids 32 (10) (2020) 105104.
[37] X. Zhang, T. Ji, F. Xie, H. Zheng, Y. Zheng, Unsteady flow prediction from sparse measurements by compressed sensing reduced order modeling, Comput. Methods Appl. Mech. Engrg. 393 (2022) 114800.
[38] T. Simpson, N. Dervilis, E. Chatzi, Machine learning approach to model order reduction of nonlinear systems via autoencoder and LSTM networks, 2021, arXiv preprint arXiv:2109.11213.
[39] P. Wu, J. Sun, X. Chang, W. Zhang, R. Arcucci, Y. Guo, C.C. Pain, Data-driven reduced order model with temporal convolutional neural network, Comput. Methods Appl. Mech. Engrg. 360 (2020) 112766.
[40] S. Fresca, A. Manzoni, POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, Comput. Methods Appl. Mech. Engrg. 388 (2022) 114181.
[41] B. Peherstorfer, K. Willcox, Data-driven operator inference for nonintrusive projection-based model reduction, Comput. Methods Appl. Mech. Engrg. 306 (2016) 196–215.
[42] P. Benner, M. Ohlberger, A. Cohen, K. Willcox, Model Reduction and Approximation: Theory and Algorithms, SIAM, 2017.
[43] D. Galbally, K. Fidkowski, K. Willcox, O. Ghattas, Non-linear model reduction for uncertainty quantification in large-scale inverse problems, Internat. J. Numer. Methods Engrg. 81 (12) (2010) 1581–1608.
[44] K. Vlachas, K. Tatsis, K. Agathos, A.R. Brink, E. Chatzi, A local basis approximation approach for nonlinear parametric model order reduction, J. Sound Vib. 502 (2021) 116055.
[45] K. Vlachas, K. Tatsis, K. Agathos, A.R. Brink, D. Quinn, E. Chatzi, Parametric model order reduction for localized nonlinear feature inclusion, in: Advances in Nonlinear Dynamics, Springer, 2022, pp. 373–383.
[46] W.D. Fries, X. He, Y. Choi, LaSDI: Parametric latent space dynamics identification, Comput. Methods Appl. Mech. Engrg. 399 (2022) 115436.
[47] X. He, Y. Choi, W.D. Fries, J. Belof, J.-S. Chen, gLaSDI: Parametric physics-informed greedy latent space dynamics identification, 2022, arXiv preprint arXiv:2204.12005.
[48] B. Peherstorfer, K. Willcox, Dynamic data-driven reduced-order models, Comput. Methods Appl. Mech. Engrg. 291 (2015) 21–41.
[49] B. Peherstorfer, K. Willcox, Online adaptive model reduction for nonlinear systems via low-rank updates, SIAM J. Sci. Comput. 37 (4) (2015) A2123–A2150.
[50] H. Zhang, C.W. Rowley, E.A. Deem, L.N. Cattafesta, Online dynamic mode decomposition for time-varying systems, SIAM J. Appl. Dyn. Syst. 18 (3) (2019) 1586–1609.
[51] M.S. Hemati, M.O. Williams, C.W. Rowley, Dynamic mode decomposition for large and streaming datasets, Phys. Fluids 26 (11) (2014) 111701.
[52] Q. Wang, O. Fink, L. Van Gool, D. Dai, Continual test-time domain adaptation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2022, pp. 7201–7211.
[53] P.J. Werbos, Generalization of backpropagation with application to a recurrent gas market model, Neural Netw. 1 (4) (1988) 339–356.
[54] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A.A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al., Overcoming catastrophic forgetting in neural networks, Proc. Natl. Acad. Sci. 114 (13) (2017) 3521–3526.
[55] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, 2016, arXiv preprint arXiv:1612.01474.
[56] D.A. Nix, A.S. Weigend, Estimating the mean and variance of the target probability distribution, in: Proceedings of 1994 Ieee International Conference on Neural Networks, Vol. 1, ICNN'94, IEEE, 1994, pp. 55–60.
[57] B. Van der Pol, On "relaxation-oscillations", Lond. Edinb. Dublin Philos. Mag. J. Sci. 2 (11) (1926) 978–992.
[58] D. Kaplan, L. Glass, Understanding Nonlinear Dynamics, Springer Science & Business Media, 1997, pp. 240–244.
[59] K. Champion, B. Lusch, J.N. Kutz, S.L. Brunton, Data-driven discovery of coordinates and governing equations, Proc. Natl. Acad. Sci. 116 (45) (2019) 22445–22451.
[60] D. Floryan, M.D. Graham, Data-driven discovery of intrinsic dynamics, Nat. Mach. Intell. 4 (12) (2022) 1113–1120.
[61] P. Angot, C.-H. Bruneau, P. Fabrie, A penalization method to take into account obstacles in incompressible viscous flows, Numer. Math. 81 (4) (1999) 497–520.
[62] A.J. Chorin, Numerical solution of the Navier-Stokes equations, Math. Comp. 22 (104) (1968) 745–762.
[63] A.T. Mohan, N. Lubbers, D. Livescu, M. Chertkov, Embedding hard physical constraints in neural network coarse-graining of 3D turbulence, 2020, arXiv preprint arXiv:2002.00021.
[64] M. Chatzimanolakis, P. Weber, P. Koumoutsakos, CubismAMR – a C++ library for distributed block-structured adaptive mesh refinement, 2022, arXiv preprint arXiv:2206.07345.
[65] M. Chatzimanolakis, P. Weber, P. Koumoutsakos, Vortex separation cascades in simulations of the planar flow past an impulsively started cylinder up to, J. Fluid Mech. 953 (2022) R2.
[66] A. Rasheed, O. San, T. Kvamsdal, Digital twin: Values, challenges and enablers from a modeling perspective, IEEE Access 8 (2020) 21980–22012.
[67] M.G. Kapteyn, J.V. Pretorius, K.E. Willcox, A probabilistic graphical model foundation for enabling predictive digital twins at scale, Nat. Comput. Sci. 1 (5) (2021) 337–347.
[68] R. Vinuesa, S.L. Brunton, B.J. McKeon, The transformative potential of machine learning for experiments in fluid mechanics, 2023, arXiv preprint arXiv:2303.15832.
[69] D.P. Kingma, M. Welling, Auto-encoding variational bayes, 2013, arXiv preprint arXiv:1312.6114.
[70] L. Girin, S. Leglaive, X. Bie, J. Diard, T. Hueber, X. Alameda-Pineda, Dynamical variational autoencoders: A comprehensive review, 2020, arXiv preprint arXiv:2008.12595.

[71] C.E. Heaney, Y. Li, O.K. Matar, C.C. Pain, Applying convolutional neural networks to data on unstructured meshes with space-filling curves, 2020, arXiv preprint arXiv:2011.14820.
[72] G. Riegler, A. Osman Ulusoy, A. Geiger, Octnet: Learning deep 3D representations at high resolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3577–3586.
[73] M. Tatarchenko, A. Dosovitskiy, T. Brox, Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2088–2096.
[74] K. Fukami, T. Nakamura, K. Fukagata, Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data, Phys. Fluids 32 (9) (2020) 095110.
[75] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, W.-c. Woo, Convolutional LSTM network: A machine learning approach for precipitation nowcasting, Adv. Neural Inf. Process. Syst. 28 (2015).
[76] Y. Liu, J.N. Kutz, S.L. Brunton, Hierarchical deep learning of multiscale differential equation time-steppers, Phil. Trans. R. Soc. A 380 (2229) (2022) 20210200.
[77] S. Pawar, S. Rahman, H. Vaddireddy, O. San, A. Rasheed, P. Vedula, A deep learning enabler for nonintrusive reduced order modeling of fluid flows, Phys. Fluids 31 (8) (2019) 085101.
[78] H. Wu, J. Xu, J. Wang, M. Long, Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting, Adv. Neural Inf. Process. Syst. 34 (2021) 22419–22430.
[79] T.M. Moerland, J. Broekens, C.M. Jonker, Model-based reinforcement learning: A survey, 2020, arXiv preprint arXiv:2006.16712.
[80] A. de Mathelin, F. Deheeger, M. Mougeot, N. Vayatis, Deep anti-regularized ensembles provide reliable out-of-distribution uncertainty quantification, 2023, arXiv preprint arXiv:2304.04042.
[81] M. Valdenegro-Toro, D.S. Mori, A deeper look into aleatoric and epistemic uncertainty disentanglement, in: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPRW, IEEE, 2022, pp. 1508–1516.
[82] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, Adv. Neural Inf. Process. Syst. 30 (2017).
[83] J. Nixon, B. Lakshminarayanan, D. Tran, Why are bootstrapped deep ensembles not better? in: "I Can't Believe It's Not Better!"NeurIPS 2020 Workshop, 2020.
[84] R. Egele, R. Maulik, K. Raghavan, B. Lusch, I. Guyon, P. Balaprakash, Autodeuq: Automated deep ensemble with uncertainty quantification, in: 2022 26th International Conference on Pattern Recognition, ICPR, IEEE, 2022, pp. 1908–1914.
[85] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[86] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.
[87] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, 2014, arXiv preprint arXiv:1412.6572.
[88] W. Jakob, J. Rhinelander, D. Moldovan, pybind11 – seamless operability between C++11 and Python, 2017, https://github.com/pybind/pybind11.