Multiscale Simulations of Complex Systems by Learning their Effective Dynamics

Pantelis R. Vlachas,[1,2] Georgios Arampatzis,[1,2] Caroline Uhler,[3] and Petros Koumoutsakos[1,2,*]

[1] *Computational Science and Engineering Laboratory, ETH Zürich, CH-8092, Switzerland*
[2] *School of Engineering and Applied Sciences, 29 Oxford Street, Harvard University, Cambridge, MA 02138, USA*
[3] *Institute for Data, Systems, and Society, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA*

(Dated: October 20, 2021)

Predictive simulations of complex systems are essential for applications ranging from weather forecasting to drug design. The veracity of these predictions hinges on their capacity to capture the effective system dynamics. Massively parallel simulations predict the system dynamics by resolving all spatiotemporal scales, often at a cost that prevents experimentation while their findings may not allow for generalisation. On the other hand reduced order models are fast but limited by the frequently adopted linearization of the system dynamics and/or the utilization of heuristic closures. Here we present a novel systematic framework that bridges large scale simulations and reduced order models to Learn the Effective Dynamics (LED) of diverse complex systems. The framework forms algorithmic alloys between non-linear machine learning algorithms and the Equation-Free approach for modeling complex systems. LED deploys autoencoders to formulate a mapping between fine and coarse-grained representations and evolves the latent space dynamics using recurrent neural networks. The algorithm is validated on benchmark problems and we find that it outperforms state of the art reduced order models in terms of predictability and large scale simulations in terms of cost. LED is applicable to systems ranging from chemistry to fluid mechanics and reduces the computational effort by up to two orders of magnitude while maintaining the prediction accuracy of the full system dynamics. We argue that LED provides a novel potent modality for the accurate prediction of complex systems.

Keywords: multiscale modeling, complex systems, equation-free, autoencoders

Some of the most important scientific advances and engineering designs are founded on the study of complex systems that exhibit dynamics spanning multiple spatiotemporal scales. Examples include protein dynamics [1], morphogenesis [2], brain dynamics [3], climate [4], ocean dynamics [5] and social systems [6]. Over the last fifty years, simulations have become a key component of these studies thanks to a confluence of advances in computing architectures, numerical methods and software. Large scale simulations have led to unprecedented insight, acting as in-silico microscopes [7] or telescopes to reveal the dynamics of galaxy formations [8]. At the same time these simulations have led to the understanding that resolving the full range of spatio-temporal scales in such complex systems will remain out of reach in the foreseeable future.

In recent years there have been intense efforts to develop efficient simulations that exploit the multiscale character of the systems under investigation [9–12]. Multiscale methods rely on judicious approximations of the interactions between processes occurring over different scales and a number of potent frameworks have been proposed including the Equation-Free framework (EFF) [10, 12–14], the Heterogeneous Multiscale Method (HMM) [11, 15, 16], and the FLow AVeraged integatoR (FLAVOR) [17]. In these algorithms the system dynamics are distinguished into fine and coarse scales or expensive and affordable simulations, respectively. Their success depends on the separation of scales that are inherent to the system dynamics and their capability to capture the transfer of information between scales. Effective applications of multiscale methodologies minimize the computational effort while maximizing the accuracy of the propagated dynamics. The EFF relies on few fine scale simulations that are used to acquire, through "restricting", information about the evolution of the coarse-grained quantities of interest. In turn various time stepping procedures are used to propagate the coarse-grained dynamics. The fine scale dynamics are obtained by judiciously "lifting" the coarse scales to return to the fine scale description of the system and repeat. When the EFF reproduces trajectories of the original system, the identified low order dynamics represent the intrinsic system dynamics, also called effective dynamics, inertial manifold [18, 19] or reaction coordinates in molecular kinetics.

While it is undisputed that the EFF, HMM, FLAVOR and related frameworks have revolutionized the field of multiscale modeling and simulation, we identify two critical issues that presently limit their potential. First, the accuracy of propagating the coarse-grained/latent dynamics hinges on the employed time integrators. Second, the choice of information transfer, in particular from coarse to fine scale dynamics in 'lifting', greatly affects the forecasting capacity of the methods.

* Corresponding author, petros@seas.harvard.edu

In the present work these two critical issues are resolved through machine learning (ML) algorithms that (i) deploy recurrent neural networks (RNNs) with gating mechanisms to evolve the coarse-grained dynamics and (ii) employ advanced (convolutional, or probabilistic) autoencoders (AE) to transfer in a systematic, data driven manner, the information between coarse and fine scale descriptions.

Over the last years, ML algorithms have exploited the ample availability of data, and powerful computing architectures, to provide us with remarkable successes across scientific disciplines [20, 21]. The particular elements of our algorithms have been employed in the modeling of dynamical systems. Autoencoders (AEs) have been used to identify a linear latent space based on the Koopman framework [22], model high-dimensional fluid flows [23, 24] or sample effectively the state space in the kinetics of proteins [25]. More recently AEs have been coupled with dynamic importance sampling [26] to accelerate multiscale simulations and investigate the interactions of RAS proteins with a plasma membrane. RNNs with gating mechanisms have been shown successful in a wide range of applications, from speech processing [27] to complex systems [28], but their effectiveness in a multiscale setting has yet to be investigated. AEs coupled with RNNs are used in [29–31] to model fluid flows. In [32], the authors build on the EFF framework, identify a PDE on a coarse representation by diffusion maps, Gaussian processes or neural networks, and utilize forward integration in the coarse representation. These previous works, however, fail to employ one or more of the following mechanisms in contrast to our framework: consider the coarse scale dynamics [23, 24], account their non-Markovian [26, 32] or non-linear nature [22], exploit a probabilistic generative mapping [23, 29–31] from the coarse to the fine scale, learn simultaneously the latent space and its dynamics in an end-to-end fashion and not sequentially [22, 23, 26, 29–32], alternate between micro and macro dynamics [22, 23, 29–32], and scale to high-dimensional systems [29, 30, 32].

Augmenting multiscale frameworks (including EFF, HMM, FLAVOR) with state of the art ML algorithms allows for evolving the coarse scale dynamics by taking into account their time history and by providing consistent lifting (decoding) and restriction (encoding) operators to transfer information between fine and coarse scales. We demonstrate that the proposed framework allows for simulations of complex multiscale systems that reduce the computational cost by orders of magnitude, to capture spatiotemporal scales that would be impossible to resolve with existing computing resources.

I. LEARNING THE EFFECTIVE DYNAMICS (LED)

We propose a framework for learning the effective dynamics (LED) of complex systems, that allows for accurate prediction of the system evolution at a significant reduced computational cost.

In the following, the high-dimensional state of a dynamical system is given by $\boldsymbol{s}_t \in \mathbb{R}^{d_{\boldsymbol{s}}}$, and the discrete time dynamics are given by

$$\boldsymbol{s}_{t+\Delta t} = \boldsymbol{F}(\boldsymbol{s}_t),$$

where Δt is the sampling period and $\boldsymbol{F}$ may be non-linear, deterministic or stochastic. We assume that the state of the system at time t can be described by a vector $\boldsymbol{z}_t \in \mathcal{Z}$, where $\mathcal{Z} \subset \mathbb{R}^{d_{\boldsymbol{z}}}$ is a low dimension manifold with $d_{\boldsymbol{z}} \ll d_{\boldsymbol{s}}$. In order to identify this manifold, an encoder $\mathcal{E}^{\boldsymbol{w}_\mathcal{E}} : \mathbb{R}^{d_{\boldsymbol{s}}} \to \mathbb{R}^{d_{\boldsymbol{z}}}$ is utilized, where $\boldsymbol{w}_\mathcal{E}$ are trainable parameters, transforming the high-dimensional state $\boldsymbol{s}_t$ to $\boldsymbol{z}_t = \mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{s}_t)$. In turn, a decoder maps back this latent representation to the high-dimensional state, i.e. $\tilde{\boldsymbol{s}}_t = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z}_t)$.

For deterministic systems, the optimal parameters $\{\boldsymbol{w}^\star_\mathcal{E}, \boldsymbol{w}^\star_\mathcal{D}\}$ are identified by minimizing the mean squared reconstruction error (MSE),

$$\boldsymbol{w}^\star_\mathcal{E}, \boldsymbol{w}^\star_\mathcal{D} = \underset{\boldsymbol{w}_\mathcal{E}, \boldsymbol{w}_\mathcal{D}}{\operatorname{argmin}} \Big(\boldsymbol{s}_t - \tilde{\boldsymbol{s}}_t \Big)^2 = \underset{\boldsymbol{w}_\mathcal{E}, \boldsymbol{w}_\mathcal{D}}{\operatorname{argmin}} \Big(\boldsymbol{s}_t - \mathcal{D}^{\boldsymbol{w}_\mathcal{D}} \big(\mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{s}_t) \big) \Big)^2.$$

Convolutional neural network [33] autoencoders (CNN-AE) that take advantage of the spatial structure of the data are embedded into LED.

For stochastic systems, $\mathcal{D}^{\boldsymbol{w}_\mathcal{D}}$ is modeled with a Mixture Density (MD) decoder [34]. Further details on the implementation of the MD decoder are provided in the SI, Section 1E, along with other components embedded in LED like AEs in SI, Section 1A, Variational AEs in SI, Section 1B, CNNs in SI, Section 1C.

We demonstrate the modularity of LED, as it can be coupled with a permutation invariant layer (see details in the SI, Section 1D), and utilized later in the modeling of the dynamics of a large set of particles governed by the advection diffusion equation (see details in the SI, Section 3A).

As a non-linear propagator in the low order manifold (coarse scale), an RNN is employed, capturing non-Markovian, memory effects by keeping an internal memory state.

The RNN is learning a forecasting rule

$$\boldsymbol{h}_t = \mathcal{H}^{\boldsymbol{w}_\mathcal{H}} \big(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t} \big), \quad \tilde{\boldsymbol{z}}_{t+\Delta t} = \mathcal{R}^{\boldsymbol{w}_\mathcal{R}} \big(\boldsymbol{h}_t \big),$$

where $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$ is an internal hidden memory state, $\tilde{\boldsymbol{z}}_{t+\Delta t}$ is a latent state prediction, $\mathcal{H}^{\boldsymbol{w}_\mathcal{H}}$ and $\mathcal{R}^{\boldsymbol{w}_\mathcal{R}}$ are the hidd ne
possi on
1F.

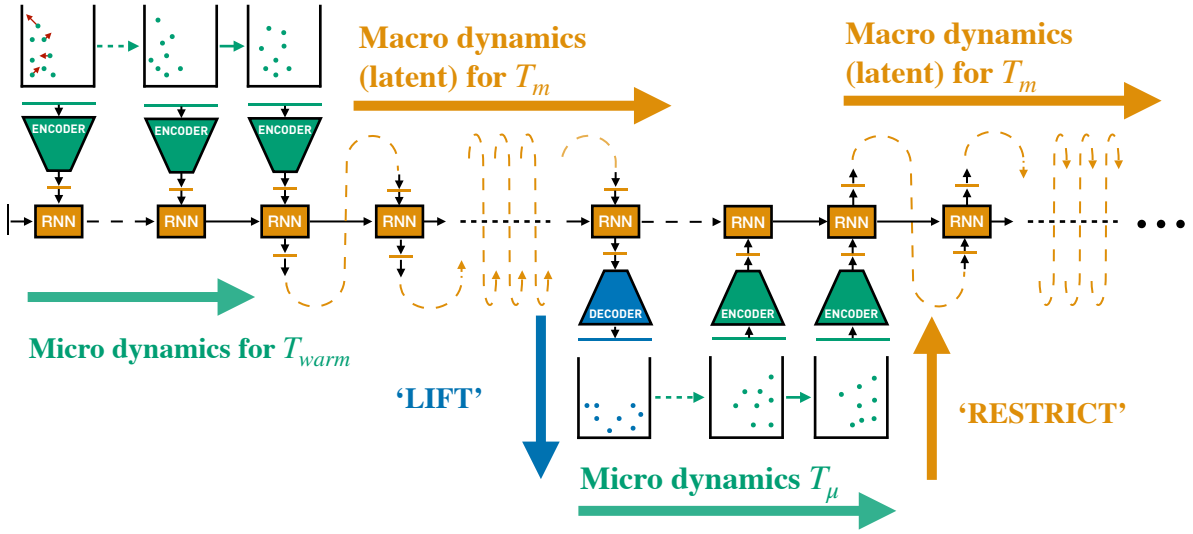


FIG. 1: Multiscale-LED: Starting from an initial condition use the equations/first principles to evolve the high-dimensional dynamics for a short period T_{warm}. During this warm-up period, the state $\boldsymbol{s}_t$ is passed through the encoder network. The outputs of the autoencoder are iteratively provided as inputs to the RNN, to warm-up its hidden state. Next, iteratively, **(1)** starting from the last latent state $\boldsymbol{z}_t$ the RNN propagates the latent dynamics for $T_m \gg T_{warm}$, **(2)** lift the latent dynamics at $t = T_{warm} + T_m$ back to the high-dimensional state, **(3)** starting from this high-dimensional state as an initial condition, use the equations/first principles to evolve the dynamics for $T_\mu \ll T_m$.

The role of the RNN is twofold. First, it is updating its hidden memory state $\boldsymbol{h}_t$, given the current state provided at the input $\boldsymbol{z}_t$ and the hidden memory state at the previous time-step $\boldsymbol{h}_{t-\Delta t}$, tracking the history of the low order state to model non-Markovian dynamics. Second, given the updated hidden state $\boldsymbol{h}_t$ the RNN forecasts the latent state at the next time-step(s) $\tilde{\boldsymbol{z}}_{t+\Delta t}$. The RNN is trained to minimize the forecasting loss $||\tilde{\boldsymbol{z}}_{t+\Delta t} - \boldsymbol{z}_{t+\Delta t}||_2^2$ by backpropagation through time [36].

The LSTM and the AE, jointly referred to as LED, are trained on data from simulations of the fully resolved (or microscale) dynamical system. The two networks can either be trained sequentially, or together. In the first case, the AE is pretrained to minimize the reconstruction loss, and then the LSTM is trained to minimize the prediction loss on the latent space (AE-LSTM). In the second case, they are seen as one network trying to minimize the sum of reconstruction and prediction losses (AE-LSTM-end2end). For large, high-dimensional systems, the later approach of end-to-end training is computationally expensive. After training, LED is employed to forecast the dynamics on unseen data, by propagating the low order latent state with the RNN and avoiding the computationally expensive simulation of high-dimensional dynamics. We refer to this mode of propagation, iteratively propagating only the latent/macro dynamics, as Latent-LED. We note that, as non-Markovian models are not self-starting, an initial small warm-up period is required, feeding the LED with data from the micro dynamics.

The LED framework allows for data driven information transfer between coarse and fine scales through the AE. Moreover it propagates the latent space dynamics without the need to upscale back to the high-dimensional state space at every time-step. As is the case for any approximate iterative integrator (here the RNN), the initial model errors will propagate. In order to mitigate potential instabilities, inspired by the Equation-Free [10], we propose the multiscale forecasting scheme in Figure 1, alternating between micro dynamics for T_μ and macro dynamics for T_m. In this way, the approximation error can be reduced at the cost of the computational complexity associated with evolving the high-dimensional dynamics. We refer to this mode of propagation as Multiscale-LED, and the ratio $\rho = T_m/T_\mu$ as multiscale ratio. In Multiscale-LED, the interface with the high-dimensional state space is enabled only at the time-steps and scales of interest. This is in contrast to [37, 38], and is easily adaptable to the needs of particular applications thus augmenting the arsenal of models developed for multiscale problems.

Training of LED models is performed with the Adam stochastic optimization method [39], and validation based early

stopping is employed to avoid overfitting. All LED models are implemented in Pytorch, mapped to a single Nvidia Tesla P100 GPU and executed on the XC50 compute nodes of the Piz Daint supercomputer at the Swiss national supercomputing centre (CSCS).

II. RESULTS

We demonstrate the application of LED in a number of benchmark problems and compare its performance with existing state of the art algorithms. In the SI Section 3D, we provide additional results on LED applied on alanine dipeptide in water. The stochastic dynamics of the molecular system are handled with an MD decoder, and an MD-LSTM in the latent space [40].

A. FitzHugh-Nagumo Model (FHN)

LED is employed to capture the dynamics of the FitzHugh-Nagumo equations (FHN) [41, 42]. The FHN model describes the evolution of an activator $u(x,t) = \rho^{ac}(x,t)$ and an inhibitor density $v(x,t) = \rho^{in}(x,t)$ on the domain $x \in [0, L]$:

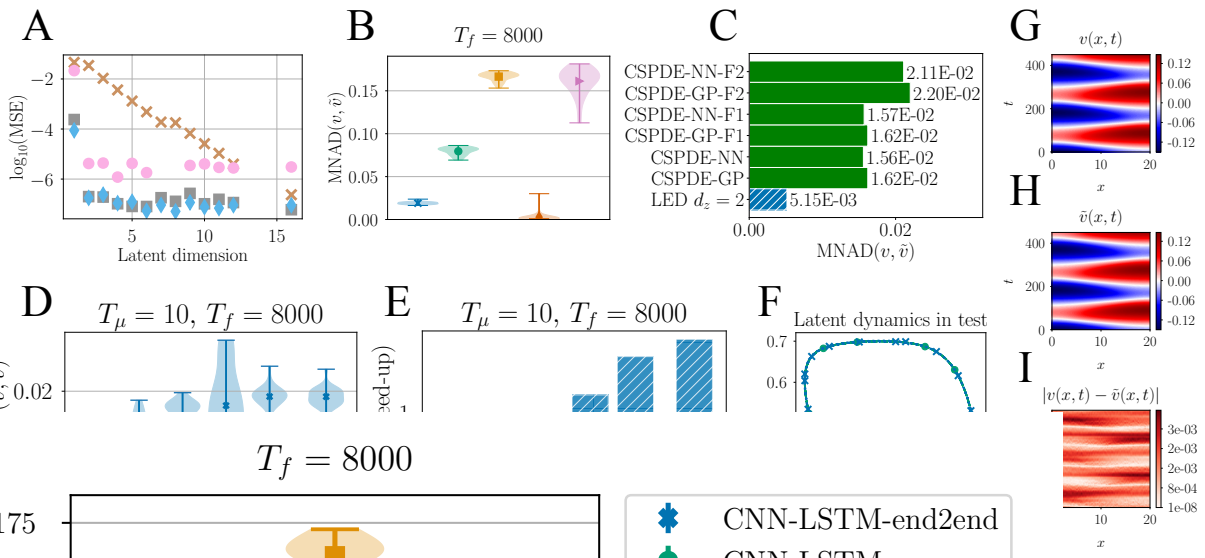
$$\begin{aligned} \frac{\partial u}{\partial t} &= D^u \frac{\partial^2 u}{\partial x^2} + u - u^3 - v, \\ \frac{\partial v}{\partial t} &= D^v \frac{\partial^2 v}{\partial x^2} + \epsilon(u - \alpha_1 v - \alpha_0). \end{aligned} \tag{1}$$

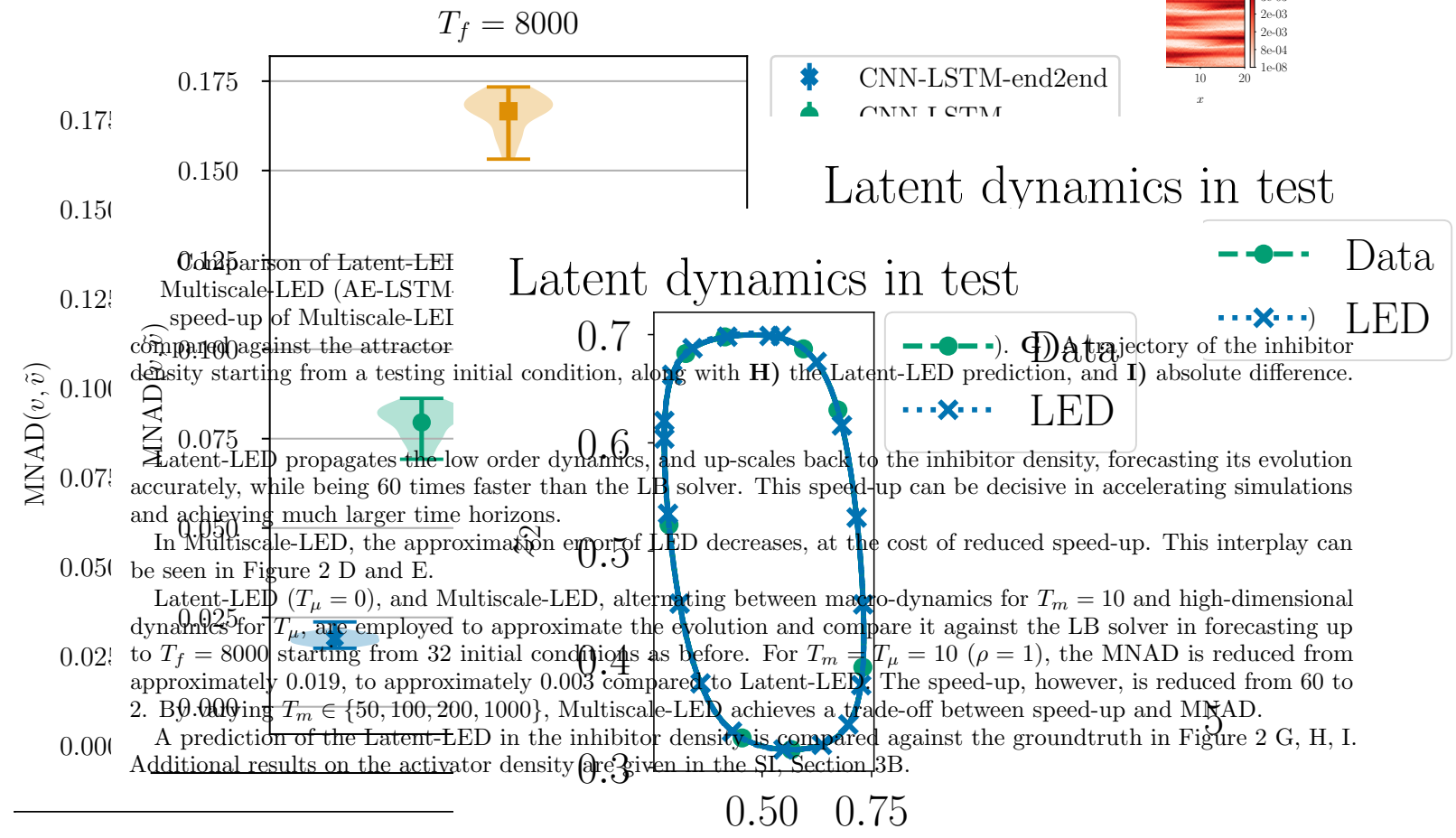
The system evolves periodically under two timescales, with the activator/inhibitor density acting as the "fast"/"slow" variable respectively. The bifurcation parameter $\epsilon = 0.006$ controls the difference in the time-scales. We choose $D^u = 1$, $D^v = 4$, $L = 20$, $\alpha_0 = -0.03$ and $\alpha_1 = 2$.

Equation (1) is discretized with $N = 101$ grid points and solved using the Lattice Boltzmann (LB) method [43], with time-step $\delta_t = 0.005$. To facilitate comparison with [32], we employ the LB method to gather data starting from 6 different initial conditions to obtain the mesoscopic solution considered here as the fine-grained solution. The data is sub-sampled, keeping every 200^{th} data point, i.e. the coarse time step is $\Delta t = 1$. Three time series with 451 points are considered for training, two time series with 451 points for validation, and 10^4 data points from a different initial condition for testing. For the identification of the latent space, we compare principal component analysis (PCA), diffusion maps, feed-forward AE, and CNN-AE, in terms of the mean squared error (MSE) of the reconstruction in the test data, plotted in Figure 2 A. The MSE is plateauing after $d_{\boldsymbol{z}} = 2$, and the AE and CNN-AE exhibit at least an order of magnitude lower MSR compared to PCA and DiffMaps. For this reason, we employ an AE with $d_{\boldsymbol{z}} = 2$ for the LED. The hyper-parameters of the networks (reported in the SI, Table 3 along with training times) are tuned based on the MSE on the validation data. The architecture of the CNN is reported in Table 5, and depicted in Figure 10 of the SI.

In Figure 2 B, we compare various propagators in forecasting of the macro (latent dynamics), starting from 32 different initial conditions in the test data, up to a horizon of $T_f = 8000$. We benchmark an AE-LSTM trained end-to-end (AE-LSTM-end2end), an AE-LSTM where the AE is pretrained (AE-LSTM), a multi-layered perceptron (AE-MLP), Reservoir Computers (AE-RC) [28, 44], and the SINDy algorithm (AE-SINDy) [45]. As a comparison metric, we consider the mean normalised absolute difference (MNAD), averaged over the 32 initial conditions. The definition of the MNAD is provided in SI, Section 2. The MNAD is computed on the inhibitor density, as the difference between the result of the LB simulation $v(x,t)$, considered as groundtruth, and the model forecasts $\hat{v}$. The warm-up period for all propagators is set to $T_{warm} = 60$. The hyper-parameters of the networks (reported in Tables 4, 6, and 7 of the SI, along with the training times) are tuned based on the MNAD on the validation data. The LSTM-end2end and the RC show the lowest test error, while the variance of the RC is larger. In the following, we consider an LSTM-end2end propagator for the LED.

LED is benchmarked against EFF variants [32] in the FHN equation in Figure 2 C. As a metric for the accuracy, the MNAD is considered, consistent with [32] to facilitate comparison. The EFF variants [32] are based on the identification of PDEs on the coarse level (CSPDE). LED is compared with CSPDEs in forecasting the dynamics of the FHN equation starting from an initial condition from the test data up to final time $T_f = 451$. CSPDE variants are utilizing Gaussian processes (GP) or neural networks (NN), features of the fine scale dynamics obtained through diffusion maps (F1 to F3) and forward integration to propagate the coarse representation in time. LED outperforms CSPDE variants by an order of magnitude. In Figure 2 F, the latent space of LED is plotted against the attractor of the data embedded in the latent space. Even for long time horizons (here $T_f = 8000$), the LED forecasts stay on the periodic attractor.





Comparison of Latent-LED, Multiscale-LED (AE-LSTM), and speed-up of Multiscale-LED compared against the attractor density starting from a testing initial condition, along with **H)** the Latent-LED prediction, and **I)** absolute difference. **G)** A trajectory of the inhibitor

Latent-LED propagates the low order dynamics, and up-scales back to the inhibitor density, forecasting its evolution accurately, while being 60 times faster than the LB solver. This speed-up can be decisive in accelerating simulations and achieving much larger time horizons.

In Multiscale-LED, the approximation error of LED decreases, at the cost of reduced speed-up. This interplay can be seen in Figure 2 D and E.

Latent-LED ($T_\mu = 0$), and Multiscale-LED, alternating between macro-dynamics for $T_m = 10$ and high-dimensional dynamics for T_μ, are employed to approximate the evolution and compare it against the LB solver in forecasting up to $T_f = 8000$ starting from 32 initial conditions as before. For $T_m = T_\mu = 10$ ($\rho = 1$), the MNAD is reduced from approximately 0.019, to approximately 0.003 compared to Latent-LED. The speed-up, however, is reduced from 60 to 2. By varying $T_m \in \{50, 100, 200, 1000\}$, Multiscale-LED achieves a trade-off between speed-up and MNAD.

A prediction of the Latent-LED in the inhibitor density is compared against the groundtruth in Figure 2 G, H, I. Additional results on the activator density are given in the SI, Section 3B.

B. Kuramoto-Sivashinsky

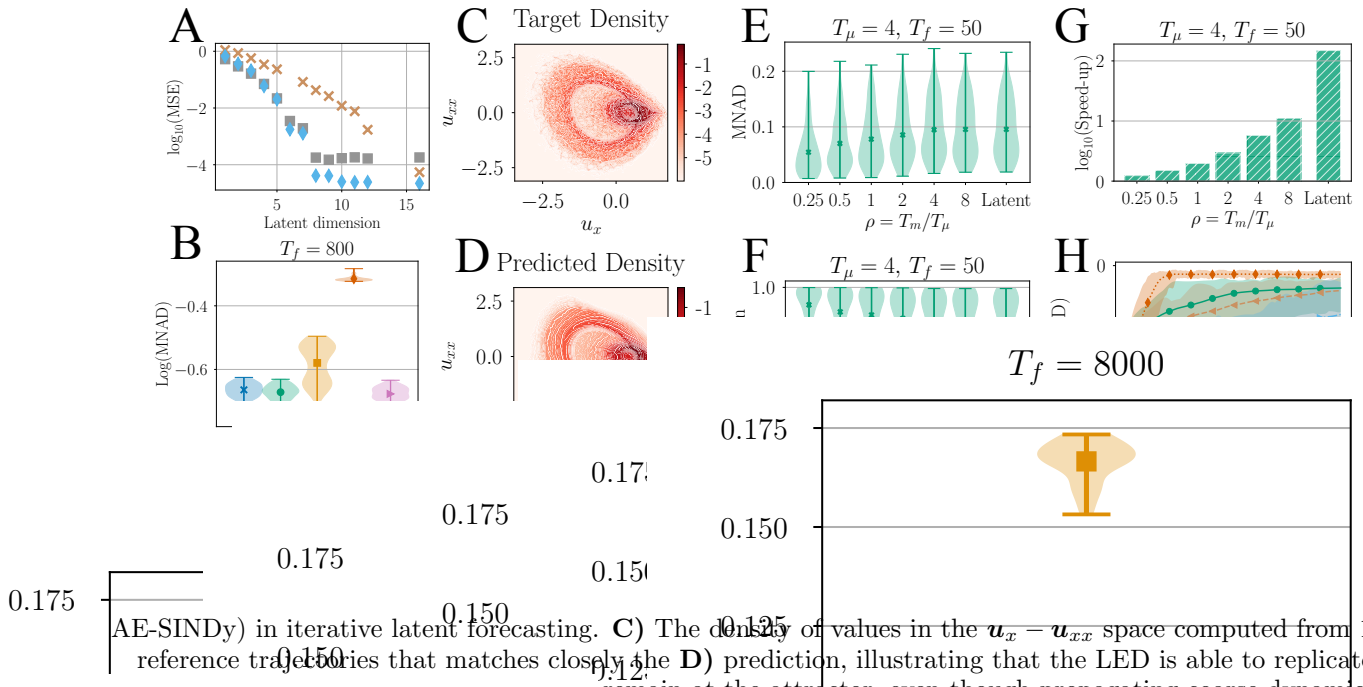
The Kuramoto-Sivashinsky (KS) [46, 47] is a prototypical partial differential equation (PDE) of fourth order that exhibits a very rich range of non-linear phenomena. In case of high dissipation and small spatial extent L (domain size), the long-term dynamics of KS can be represented on a low dimensional inertial manifold [18, 19], that attracts all neighboring states at an exponential rate after a transient period. LED is employed to learn the low order manifold of the effective dynamics in KS.

The one dimensional K-S equation is given by the PDE

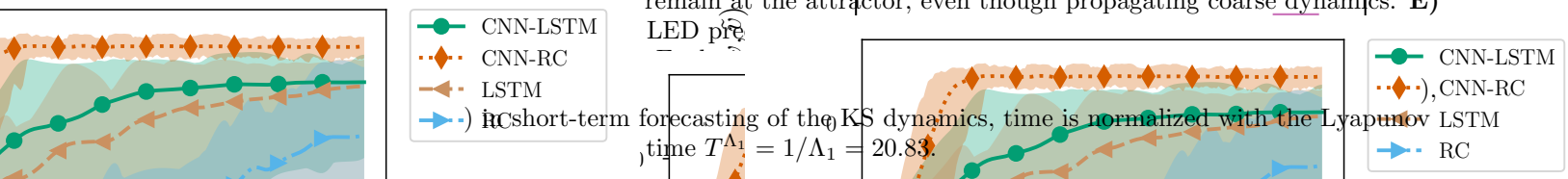
$$\frac{\partial u}{\partial t} = -\nu \frac{\partial^4 u}{\partial x^4} - \frac{\partial^2 u}{\partial x^2} - u \frac{\partial u}{\partial x}, \tag{2}$$

on the domain $\Omega = [0, L]$ with periodic boundary conditions $u(0, t) = u(L, t)$ and $\nu = 1$. The special case $L = 22$ considered in this work, is studied extensively in [48], and exhibits a structurally stable chaotic attractor, i.e. an

inertial manifold where the long-term dynamics lie. Equation (2) is discretized with a grid of size 64 points, and solved using the fourth-order method for stiff PDEs introduced in [49] with a time-step of $\delta t = 2.5 \cdot 10^{-3}$ starting from a random initial condition. The data are subsampled to $\Delta t = 0.25$ (coarse time-step of LED). $15 \cdot 10^3$ samples are used for training and another $15 \cdot 10^3$ for validation. For testing purposes, the process is repeated with a different random seed, generating another $15 \cdot 10^3$ samples.



AE-SINDy) in iterative latent forecasting. **C)** The density of values in the $\boldsymbol{u}_x - \boldsymbol{u}_{xx}$ space computed from 100 reference trajectories that matches closely the **D)** prediction, illustrating that the LED is able to replicate remain at the attractor, even though propagating coarse dynamics. **E)**



) in short-term forecasting of the KS dynamics, time is normalized with the Lyapunov time $T^{\Lambda_1} = 1/\Lambda_1 = 20.83$.

For the identification of a reasonable latent space dimension, we compare PCA, AEs, and CNNs in terms of the reconstruction MSE in the test data as a function of $d_{\boldsymbol{z}}$, plotted in Figure 3 A. MSE is plateauing after $d_{\boldsymbol{z}} = 8$ indicating arguably the dimensionality of the attractor in agreement with previous studies [18, 48], and that the CNN is superior to the AE, while orders of magnitude better than PCA. For this reason, we employ a CNN with $d_{\boldsymbol{z}} = 8$ for the autoencoding part of LED. The hyper-parameters of the networks are tuned based on the MSE on the validation data, reported in SI, Tables 8, 9 with the network training times. The CNN architecture is provided in SI Table 10, and depicted in SI Figure 12.

In Figure 3 B, we compare various propagators in predicting the macro dynamics of LED, starting from 100 test initial conditions, up to $T_f = 800$ (3200 time-steps). We employ a CNN-LSTM trained end-to-end (CNN-LSTM-end2end), a CNN-LSTM where the CNN is pretrained (CNN-LSTM), a multi-layered perceptron (CNN-MLP), Reservoir Computers (CNN-RC) [28, 44], and the SINDy algorithm (CNN-SINDy) [45]. As a comparison metric, we consider the MNAD, averaged over the 100 initial conditions. The warm-up period for all propagators is set to $T_{warm} = 60$. The hyper-parameters (reported on SI Tables 11, 12, 13, along with the training times) are tuned based on the MNAD on the validation data. While the MLP and RC propagators exhibit large errors, the LSTM, LSTM-end2end, and SINDy show comparable accuracy. In the following, we consider an LSTM propagator for the LED.

Due to chaoticity of the KS equation, iterative forecasting with LED is challenging, as initial errors propagate exponentially. In order to assess whether the iterative forecasting with LED leads to reasonable, physical predictions, we plot the density of values in the $\boldsymbol{u}_x - \boldsymbol{u}_{xx}$ space in Figure 3 C. The data come from a single long trajectory of size $T_f = 8000$ (32000 time-steps). We observe that LED, Figure 3 D, is able to qualitatively reproduce the density of the simulation.

In Figure 3 E and F we plot the MNAD, and correlation between forecasts of LED and the reference with respect to the multiscale ratio ρ. In Figure 3 G the speed-up of LED is plotted against ρ. Latent-LED is able to reproduce the long-term "climate dynamics" [28], and remain at the attractor, while being more than two orders of magnitude faster

compared to the micro solver. As ρ is increased, the error is reduced (correlation increased), at the cost of reduced speed-up.

Finally, in Figure 3 H, we compare the performance of Latent-LED (CNN-LSTM) with previous studies [28, 44], that forecast directly on the high-dimensional space. Specifically, the Latent-LED matches the performance of an LSTM (no dimensionality reduction), but shows inferior short-term forecasting ability compared to an RC (no dimensionality reduction) forecasting on the high-dimensional space. This is expected as the RC and the LSTM have full information of the state. In turn, when the RC is employed on the latent space of LED as a macro-dynamics propagator, the error grows significantly and the performance is inferior to the CNN-LSTM case.

A forecast of Latent-LED is provided in the SI, Figure 11.

C. Viscous Flow Behind a Cylinder

The flow behind a cylinder is a widely studied problem in fluids [50], that exhibits a rich range of dynamical phenomena like the transition from laminar to turbulent flow in high Reynolds numbers, and is used as a benchmark for reduced order modeling (ROM) approaches. The flow behind a cylinder in the two dimensional space is simulated by solving the incompressible Navier-Stokes equations with Brinkman penalization to enforce the no-slip boundary conditions on the surface of the cylinder [51, 52]. More details on the simulation are provided in the SI Section 3D. We consider the application of LED to two Reynolds' numbers $Re \in \{100, 1000\}$. The definition of Re is provided in the SI Equation (24).

The flow is simulated in a cluster with 12 CPU Cores, up to $T = 200$, after discarding initial transient. 250 time-steps distanced $\Delta t = 0.2$ in time (total time $T = 50$) are used for training, 250 for validation, and the rest for testing purposes. The vortex sheeding period is $T \approx 2.86$ for $Re = 100$, and $T \approx 2.22$ for $Re = 1000$.

The state of LED is $\mathbf{s}_t \equiv \{p, u_x, u_y, \omega\} \in \mathbb{R}^{4\times512\times1024}$, where ω is the vorticity field. For the autoencoding part, LED employs CNNs that take advantage of the spatial correlations. The architecture of the CNN is given in Table 14 and depicted in Figure 13 in the SI. The dimension of the latent space is tuned based on the performance on the validation dataset to $d_{\boldsymbol{z}} = 4$ for $Re = 100$ and $d_{\boldsymbol{z}} = 10$ for $Re = 1000$.

The LSTM propagator of LED is benchmarked against SINDy and RC in predicting the dynamics, starting from 10 initial conditions randomly sampled from the test data for a prediction horizon of $T = 20$ (100 time-steps). The hyper-parameters (reported on SI Tables 15, 16, 17, along with the training times) are tuned based on the MNAD on the validation data. The logarithm of the MNAD is given in Figure 5 A for $Re = 100$ and E for $Re = 1000$. For the $Re = 100$ case, the LSTM exhibits lower MNAD and lower variance compared to RC and SINDy. For the challenging $Re = 1000$ scenario, LSTM and RC exhibit lower MNAD compared to SINDy, with the LSTM being more robust (lower variance).

A prediction of the vorticity ω by Latent-LED at lead time $T = 4$ is given in Figure 4. LED captures the flow for both $Re \in \{100, 1000\}$. The error concentrates mostly around the cylinder, rendering the accurate prediction of the drag coefficient challenging. In Figure 4 D and H, the latent space of Latent-LED is compared with the transformation of the data to the latent space. The predictions stay close to the attractor even for very large horizon ($T = 20$). The Strouhal number St (defined in the SI Equation (23)) describes the periodic vortex shedding at the wake of the cylinder. By estimating the dominant frequency of the latent state using a Fourier analysis, we find that LED reproduces exactly the St of the system dynamics for both $Re \in \{100, 1000\}$ cases.

In the $Re = 100$ case, Latent-LED recovers a periodic non-linear mode in the latent space, and can forecast the dynamics accurately, as illustrated in Figure 4. In this case, approaches based on the Galerkin method or dynamic mode decomposition (DMD), construct ROM with six to eight degrees of freedom [53] that capture the most energetic spatiotemporal modes. In contrast, the latent space of LED in the $Re = 100$ case has a dimensionality of $d_{\boldsymbol{z}} = 4$. In the challenging $Re = 1000$ scenario, LED with $d_{\boldsymbol{z}} = 10$ can capture accurately the characteristic vortex street, and long-term dynamics. We note that, to the best of our knowledge, ROMs for flows past a cylinder have been so far limited to laminar periodic flows in the order of $Re = 100$ while this study advances the state of the art by one order of magnitude.

Starting from 4 initial conditions randomly sampled from the test data, six LED variants (Latent-LED, Multiscale-LED with $T_\mu = 0.4, T_m \in \{0.4, 0.8, 1.2, 2, 4\}$ for $Re = 100$, and Latent-LED, Multiscale-LED with $T_\mu = 1.6, T_m \in \{0.8, 1.6, 3.2, 6.4, 12.8\}$ for $Re = 1000$) are tested on predicting the dynamics of the flow up to $T_f = 20$, after $T_{warm} = 2$. The MNAD is plotted in Figure 5 B for $Re = 100$, and F for $Re = 1000$. The speed-up is plotted in Figure 5 D for $Re = 100$, and H for $Re = 1000$. The Latent-LED is two orders of magnitude faster than the flow solver, while exhibiting MNAD errors of 0.02 and 0.04 for $Re = 100$, and $Re = 1000$ respectively. By alternating between macro and micro, the error is reduced, at the cost of decreased speed-up.

In Figure 5 C and G, the relative error on the drag coefficient C_d (defined in SI, Equation (28)) is plotted as a function of the multiscale ratio ρ. Latent-LED exhibits a relative error of 0.04 that is reduced to approximately 0.02

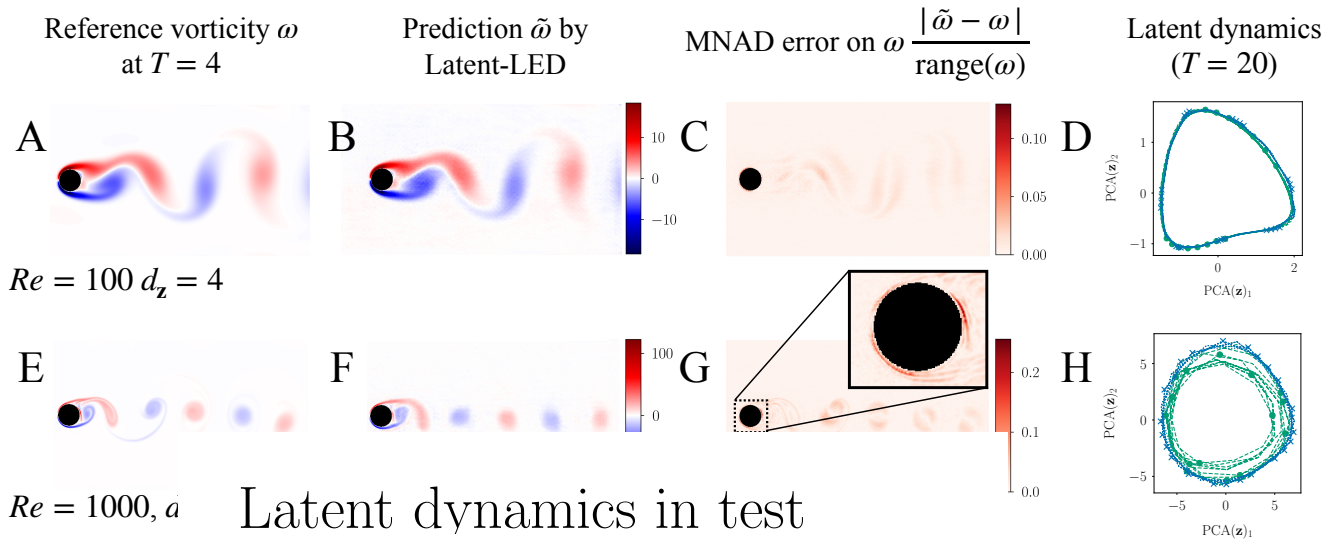
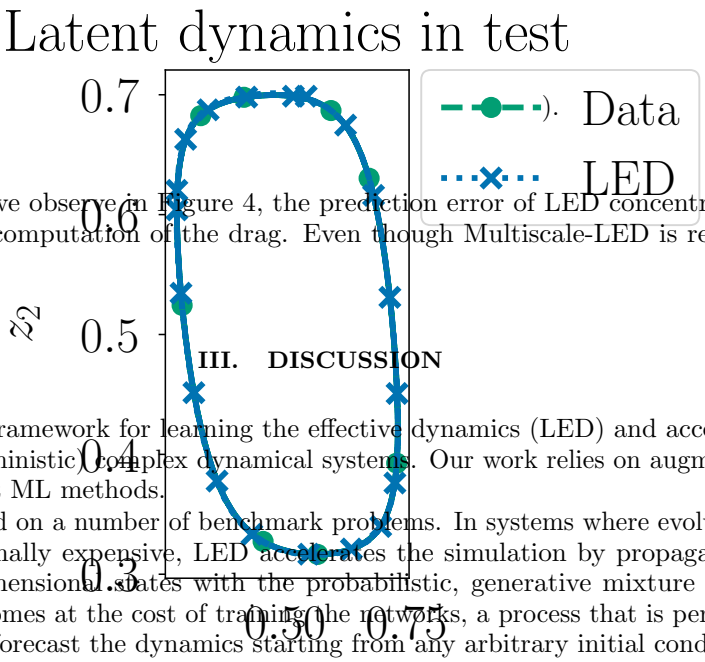


FIG. 4: **A)** The predicted by Late **G)** for $Re =$ ue vorticity field $\tilde{\omega}$) for $Re = 100$, and ctor of the data



for $\rho = 1$. For $Re = 1000$, as we observe in Figure 4, the prediction error of LED concentrates around the cylinder which leads to an inaccurate computation of the drag. Even though Multiscale-LED is reducing this error, it still remains on the order of 0.15.

III. DISCUSSION

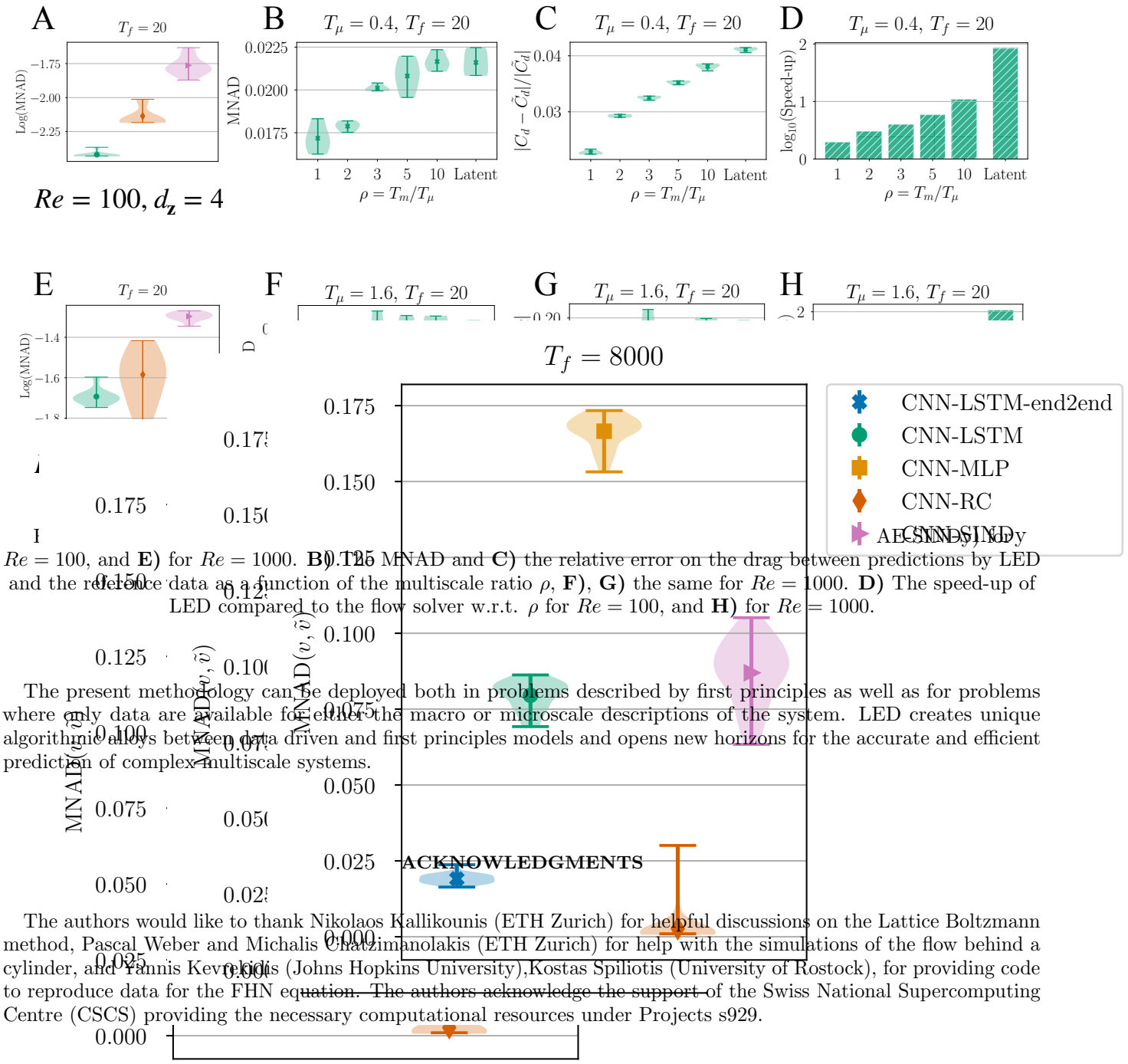
We have presented a novel framework for learning the effective dynamics (LED) and accelerate the simulations of multiscale (stochastic or deterministic) complex dynamical systems. Our work relies on augmenting the Equation-Free formalism with state of the art ML methods.

The LED framework is tested on a number of benchmark problems. In systems where evolving the high-dimensional state dynamics is computationally expensive, LED accelerates the simulation by propagating on the latent space and upscaling to the high-dimensional states with the probabilistic, generative mixture density, or deterministic convolutional, decoder. This comes at the cost of training the networks, a process that is performed once, offline. The trained model can be used to forecast the dynamics starting from any arbitrary initial condition.

The efficiency of LED was evaluated in forecasting the FitzHugh-Nagumo equation dynamics achieving an order of magnitude lower approximation error compared to other Equation-Free approaches while being two orders of magnitude faster than the Lattice Boltzmann solver. We demonstrated that the proposed framework identifies the effective dynamics of the Kuramoto-Sivashinsky equation with $L = 22$, capturing the long-term behavior ("climate dynamics"), achieving a speed-up of $S \approx 100$. Furthermore, LED captures accurately the long-term dynamics of a flow behind a cylinder in $Re = 100$ and $Re = 1000$, while being two orders of magnitude faster than a flow solver. In the SI, we demonstrate that LED can unravel and forecast the stochastic collective dynamics of 1000 particles following Brownian motion subject to advection and diffusion in the three dimensional space (SI, Section 3A). In our recent work [40] (briefly described in SI, Section 3E), we show that LED can be applied to learn the stochastic dynamics of molecular systems. We note that the present method is readily applicable to all problems where Equation-Free, HMM, and FLAVOR methodologies have been applied.

In summary, LED identifies and propagates the effective dynamics of dynamical systems with multiple spatiotemporal scales providing significant computational savings. Moreover, LED provides a systematic way of trading between speed-up and accuracy for a multiscale system by switching between propagation of the latent dynamics, and evolution of the original equations, iteratively correcting the statistical error at the cost of reduced speed-up.

The LED does not presently contain any mechanism to decide when to upscale the latent space dynamics. This is an active area of investigations. We do not expect LED to generalize to dynamical regions drastically different from those represented in the training data. Further research efforts will address this issue by adapting the training procedure.



$Re = 100$, and **E)** for $Re = 1000$. **B)** the MNAD and **C)** the relative error on the drag between predictions by LED and the reference data as a function of the multiscale ratio ρ, **F)**, **G)** the same for $Re = 1000$. **D)** The speed-up of LED compared to the flow solver w.r.t. ρ for $Re = 100$, and **H)** for $Re = 1000$.

The present methodology can be deployed both in problems described by first principles as well as for problems where only data are available for either the macro or microscale descriptions of the system. LED creates unique algorithmic alloys between data driven and first principles models and opens new horizons for the accurate and efficient prediction of complex multiscale systems.

ACKNOWLEDGMENTS

The authors would like to thank Nikolaos Kallikounis (ETH Zurich) for helpful discussions on the Lattice Boltzmann method, Pascal Weber and Michalis Chatzimanolakis (ETH Zurich) for help with the simulations of the flow behind a cylinder, and Yannis Kevrekidis (Johns Hopkins University),Kostas Spiliotis (University of Rostock), for providing code to reproduce data for the FHN equation. The authors acknowledge the support of the Swiss National Supercomputing Centre (CSCS) providing the necessary computational resources under Projects s929.

AUTHOR CONTRIBUTION

P.K. conceived the project; P.R.V., G.A., C.U., and P.K. designed and performed research; P.R.V., and G.A. contributed new analytic tools; P.R.V., G.A., and P.K. analyzed data; and P.R.V., G.A., and P.K. wrote the paper.

AUTHOR DECLARATION

The authors declare no conflict of interest.

DATA AND CODE AVAILABILITY

All code and data for the analysis associated with the current submission will become readily available upon publication in the following link: https://github.com/pvlachas/LearningEffectiveDynamics.

[1] Rackovsky, S. & Scheraga, H. A. The structure of protein dynamic space. *Proceedings of the National Academy of Sciences* **117**, 19938–19942 (2020).

[2] Gilmour, D., Rembold, M. & Leptin, M. From morphogen to morphogenesis and back. *Nature* **541**, 311–320 (2017).

[3] Robinson, P. A., Rennie, C. J., Rowe, D. L., O'Connor, S. C. & Gordon, E. Multiscale brain modelling. *Philosophical Transactions of the Royal Society B: Biological Sciences* **360**, 1043–1050 (2005).

[4] Council, N. R. *A National Strategy for Advancing Climate Modeling* (The National Academies Press, 2012).

[5] Mahadevan, A. The impact of submesoscale physics on primary productivity of plankton. *Annual review of marine science* **8**, 161–184 (2016).

[6] Bellomo, N. & Dogbe, C. On the modeling of traffic and crowds: A survey of models, speculations, and perspectives. *SIAM review* **53**, 409–463 (2011).

[7] Lee, E. H., Hsin, J., Sotomayor, M., Comellas, G. & Schulten, K. Discovery through the computational microscope. *Structure* **17**, 1295–1306 (2009).

[8] Springel, V. *et al.* Simulations of the formation, evolution and clustering of galaxies and quasars. *nature* **435**, 629–636 (2005).

[9] Car, R. & Parrinello, M. Unified approach for molecular dynamics and density-functional theory. *Physical review letters* **55**, 2471 (1985).

[10] Kevrekidis, I. G. *et al.* Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences* **1**, 715–762 (2003).

[11] Weinan, E., Engquist, B. *et al.* The heterognous multiscale methods. *Communications in Mathematical Sciences* **1**, 87–132 (2003).

[12] Kevrekidis, I. G., Gear, C. W. & Hummer, G. Equation-free: The computer-aided analysis of complex multiscale systems. *AIChE Journal* **50**, 1346–1355 (2004).

[13] Laing, C. R., Frewen, T. & Kevrekidis, I. G. Reduced models for binocular rivalry. *Journal of computational neuroscience* **28**, 459–476 (2010).

[14] Bar-Sinai, Y., Hoyer, S., Hickey, J. & Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences* **116**, 15344–15349 (2019).

[15] Weinan, E., Li, X. & Vanden-Eijnden, E. Some recent progress in multiscale modeling. In Attinger, S. & Koumoutsakos, P. (eds.) *Multiscale Modelling and Simulation*, 3–21 (Springer Berlin Heidelberg, Berlin, Heidelberg, 2004).

[16] Weinan, E., Engquist, B., Li, X., Ren, W. & Vanden-Eijnden, E. Heterogeneous multiscale methods: a review. *Communications in computational physics* **2**, 367–450 (2007).

[17] Tao, M., Owhadi, H. & Marsden, J. E. Nonintrusive and structure preserving multiscale integration of stiff odes, sdes, and hamiltonian systems with hidden slow dynamics via flow averaging. *Multiscale Modeling & Simulation* **8**, 1269–1324 (2010).

[18] Linot, A. J. & Graham, M. D. Deep learning to discover and predict dynamics on an inertial manifold. *Physical Review E* **101**, 062209 (2020).

[19] Robinson, J. C. Inertial manifolds for the kuramoto-sivashinsky equation. *Physics Letters A* **184**, 190–193 (1994).

[20] Jumper, J. *et al.* Highly accurate protein structure prediction with alphafold. *Nature* **596**, 583–589 (2021).

[21] Brunton, S. L., Noack, B. R. & Koumoutsakos, P. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics* **52** (2019).

[22] Lusch, B., Kutz, J. N. & Brunton, S. L. Deep learning for universal linear embeddings of non-linear dynamics. *Nature communications* **9**, 1–10 (2018).

[23] Geneva, N. & Zabaras, N. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *Journal of Computational Physics* **403**, 109056 (2020).

[24] Milano, M. & Koumoutsakos, P. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics* **182**, 1–26 (2002).

[25] Wehmeyer, C. & Noé, F. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of chemical physics* **148**, 241703 (2018).

[26] Bhatia, H. *et al.* Machine-learning-based dynamic-importance sampling for adaptive multiscale simulations. *Nature Machine Intelligence* **3**, 401–409 (2021).

[27] Chung, J. *et al.* A recurrent latent variable model for sequential data. *Advances in neural information processing systems* **28**, 2980–2988 (2015).

[28] Vlachas, P. R. *et al.* Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks* **126**, 191–217 (2020).

[29] Gonzalez, F. J. & Balajewicz, M. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. *arXiv preprint arXiv:1808.01346* (2018).

[30] Maulik, R., Lusch, B. & Balaprakash, P. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids* **33**, 037106 (2021).

[31] Hasegawa, K., Fukami, K., Murata, T. & Fukagata, K. Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics* **34**, 367–383 (2020).

[32] Lee, S., Kooshkbaghi, M., Spiliotis, K., Siettos, C. I. & Kevrekidis, I. G. Coarse-scale pdes from fine-scale observations via machine learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science* **30**, 013141 (2020).

[33] LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *nature* **521**, 436–444 (2015).

[34] Bishop, C. M. Mixture density networks. *Technical Report NCRG/97/004, Neural Computing Research Group, Aston University* (1994).

[35] Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).

[36] Werbos, P. J. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks* **1**, 339 – 356 (1988).

[37] Hernández, C. X., Wayment-Steele, H. K., Sultan, M. M., Husic, B. E. & Pande, V. S. Variational encoding of complex dynamics. *Physical Review E* **97**, 062412 (2018).

[38] Sultan, M. M., Wayment-Steele, H. K. & Pande, V. S. Transferable neural networks for enhanced sampling of protein dynamics. *Journal of chemical theory and computation* **14**, 1887–1894 (2018).

[39] Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (2015).

[40] Vlachas, P. R., Zavadlav, J., Praprotnik, M. & Koumoutsakos, P. Accelerated simulations of molecular systems through learning of their effective dynamics. *arXiv preprint arXiv:1312.6114* (2021).

[41] FitzHugh, R. Impulses and physiological states in theoretical models of nerve membrane. *Biophysical journal* **1**, 445 (1961).

[42] Nagumo, J., Arimoto, S. & Yoshizawa, S. An active pulse transmission line simulating nerve axon. *Proceedings of the IRE* **50**, 2061–2070 (1962).

[43] Karlin, I. V., Ansumali, S., Frouzakis, C. E. & Chikatamarla, S. S. Elements of the lattice boltzmann method i: Linear advection equation. *Commun. Comput. Phys* **1**, 616–655 (2006).

[44] Pathak, J., Hunt, B., Girvan, M., Lu, Z. & Ott, E. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters* **120**, 024102 (2018).

[45] Brunton, S. L., Proctor, J. L. & Kutz, J. N. Discovering governing equations from data by sparse identification of non-linear dynamical systems. *Proceedings of the National Academy of Sciences* **113**, 3932–3937 (2016).

[46] Kuramoto, Y. Diffusion-Induced Chaos in Reaction Systems. *Progress of Theoretical Physics Supplement* **64**, 346–367 (1978).

[47] Sivashinsky, G. I. Nonlinear analysis of hydrodynamic instability in laminar flames — I. Derivation of basic equations. *Acta Astronautica* **4**, 1177–1206 (1977).

[48] Cvitanović, P., Davidchack, R. L. & Siminos, E. On the state space geometry of the kuramoto–sivashinsky flow in a periodic domain. *SIAM Journal on Applied Dynamical Systems* **9**, 1–33 (2010).

[49] Kassam, A. & Trefethen, L. Fourth-order time-stepping for stiff pdes. *SIAM Journal on Scientific Computing* **26**, 1214–1233 (2005).

[50] Zdravkovich, M. Flow around circular cylinders; vol. i fundamentals. *Journal of Fluid Mechanics* **350**, 377–378 (1997).

[51] Rossinelli, D. *et al.* Mrag-i2d: Multi-resolution adapted grids for remeshed vortex methods on multicore architectures. *Journal of Computational Physics* **288**, 1–18 (2015).

[52] Bost, C., Cottet, G.-H. & Maitre, E. Convergence analysis of a penalization method for the three-dimensional motion of a rigid body in an incompressible viscous fluid. *SIAM Journal on Numerical Analysis* **48**, 1313–1337 (2010).

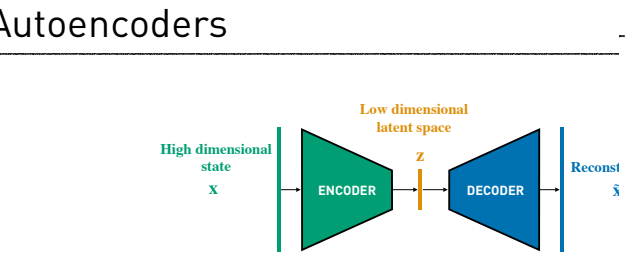
[53] Taira, K. *et al.* Modal analysis of fluid flows: Applications and outlook. *AIAA journal* **58**, 998–1022 (2020).

Supplementary Information I: Methods

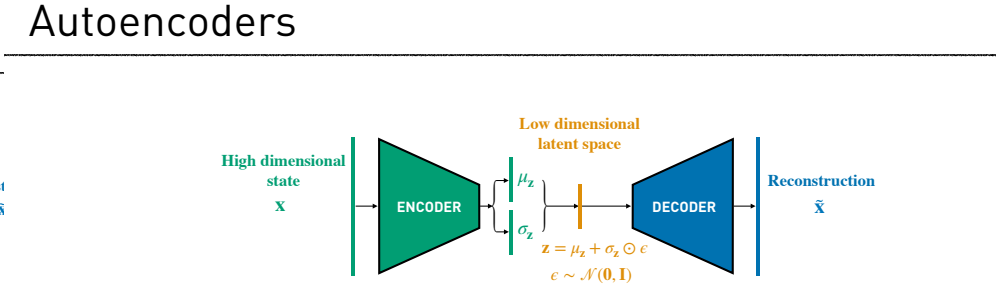
The framework to learn and propagate the effective dynamics (LED) of complex systems is composed of the models described in the following.

A. Autoencoders (AE)

Classical autoencoders are non-linear neural networks that map an input to a low dimensional latent space and then decode it to the original dimension at the output, trained to minimize the reconstruction loss $\mathcal{L} = |\boldsymbol{x} - \tilde{\boldsymbol{x}}|^2$. They were proposed in as a non-linear alternative to Principal Component Analysis (PCA). An autoencoder is depicted in Figure 1a.



(a) Classical Autoencoder (AE)



(b) Variational Autoencoder (VAE)

FIG. 1: **(a)** A schematic diagram of a classical Autoencoder (AE). A high-dimensional state $\boldsymbol{x}$ is mapped to a low dimensional feature space $\boldsymbol{z}$ by applying the encoder transformation through multiple fully connected layers. The low dimensional feature space $\boldsymbol{z}$ is expanded in the original space by the decoder. The autoencoder is trained with the loss $\mathcal{L} = ||\boldsymbol{x} - \tilde{\boldsymbol{x}}||^2$, so that the input can be reconstructed as faithfully as possible at the decoder output. **(b)** A schematic diagram of a Variational Autoencoder (VAE). Instead of modeling the latent space deterministically, the encoder outputs a mean latent representation $\boldsymbol{\mu_z}$, along with the associated uncertainty $\sigma_{\boldsymbol{z}}$. The latent space $\boldsymbol{z}$ is sampled from a normal distribution $\boldsymbol{z} \sim \mathcal{N}(\cdot|\boldsymbol{\mu_z}, \sigma_{\boldsymbol{z}}\boldsymbol{I})$, with diagonal covariance matrix.

B. Variational Autoencoders (VAE)

Research efforts on generative modeling led to the development of Variational Autoencoders (VAEs). The VAE similar to AE is composed by an encoder and a decoder. The encoder neural network, instead of mapping the input $\boldsymbol{x}$ deterministically to a reduced order latent space $\boldsymbol{z}$, produces a distribution $q(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{w}_q)$ over the latent representation $\boldsymbol{z}$, where $\boldsymbol{w}_q$ is the parametrization of the distribution given by the output of the encoder $\boldsymbol{w}_q = \mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{x})$. In most practical applications, the distribution $q(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{w}_q)$ is modeled as a factorized Gaussian, implying that $\boldsymbol{w}_q$ is composed of the mean, and the diagonal elements of the covariance matrix. The decoder maps a sampled latent representation to an output $\tilde{\boldsymbol{x}} = \mathcal{D}^{\boldsymbol{w}_\mathcal{D}}(\boldsymbol{z})$. By sampling the latent distribution $q(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{w}_q)$, for a fixed input $\boldsymbol{x}$, the autoencoder can generate samples from the probability distribution over $\tilde{\boldsymbol{x}}$ at the decoder output. The network is trained to maximize the log-likelihood of reproducing the input at the output, while minimizing the Kullback-Leibler divergence between the encoder distribution $q(\boldsymbol{z}|\boldsymbol{x}; \boldsymbol{w}_q)$ and a prior distribution, e.g. $\mathcal{N}(0, \boldsymbol{I})$. VAEs are esentially regularizing the training of AE by adding the Gaussian noise in the latent representation. In this work, a Gaussian latent distribution with diagonal covariance matrix is considered, i.e.,

$$q(\boldsymbol{z} \,|\, \boldsymbol{x}; \boldsymbol{\mu_z}, \boldsymbol{\sigma_z}) = \mathcal{N}\big(\boldsymbol{z} \,|\, \boldsymbol{\mu_z}(\boldsymbol{x}), \mathrm{diag}(\sigma_{\boldsymbol{z}}(\boldsymbol{x}))\big), \tag{1}$$

where $\boldsymbol{w}_q = (\boldsymbol{\mu_z}, \sigma_{\boldsymbol{z}})$ and the mean latent representation $\boldsymbol{\mu_z}$ and the variance $\sigma_{\boldsymbol{z}}$ vectors are the outputs of the encoder neural network $\mathcal{E}^{\boldsymbol{w}_\mathcal{E}}(\boldsymbol{x})$. The latent representation is then sampled from $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{\mu_z}, \mathrm{diag}(\sigma_{\boldsymbol{z}}))$. The decoder receives as an input the sample, and outputs the reconstruction $\tilde{\boldsymbol{x}}$. A VAE is depicted in Figure 1b.

A preliminary study, benchmarking VAEs against feedforward AEs (and Convolutional AE described later) in the FitzHugh-Nagumo equation, and the Kuramoto-Sivashinsky equation, showed no significant advantages for the cases considered in this work over feed-forward AEs. They are, however, part of the LED framework, and may be useful in other applications.

C. Convolutional Neural Networks

Convolutional neural networks (CNNs) are tailored to process image data with spatial correlations. Each layer of a CNN is processing a multidimensional input (with a channel axis, and some spatial axes) by applying a convolutional kernel or filter that slides along the input spatial axes. In other words, CNNs take into account of the structure in the data in their architecture, which is a form of a geometric prior. In this work, CNN layers are used in the Autoencoder, by introducing a bottleneck layer, reducing the dimensionality. Other dimensionality reduction techniques, like AEs, Principal Component Analysis (PCA), or Diffusion maps (DiffMaps), that are based on vectorization of input field data, do not take into account the structure of the data, i.e. when an input field is shifted by a pixel, the vectorized version will differ a lot, while the convoluted image will not.

In this work, we employ Autoencoding CNNs (and compare them with feed-forward AEs) to identify the coarse representation of the FitzHugh-Nagumo equation, the Kuramoto-Sivashinsky equation, and the incompressible Navier-Stokes flow behind a cylinder at $Re \in \{100, 1000\}$.

D. Permutation Invariance

Physical systems may satisfy specific properties like energy conservation, translation invariance, permutation invariance, etc. In order to build data-driven models that accurately reproduce the statistical behavior of such systems, these properties should be embedded in the model. In this section, the dynamics of particles of the same kind are modeled with a permutation invariance layer. This is useful in simulations of molecules, i.e. molecular dynamics, where the state of the system is described by a configuration of particles, and any permutation of these particles corresponds to the same configuration. Permutation invariance is handled here with a sum decomposition of a feature space. The exact procedure is depicted in Appendix I D.

Assume that the state of a dynamical system $\boldsymbol{s}$ is composed of N particles of the same kind, each one having specific properties or features with dimensionality $d_{\boldsymbol{x}}$, e.g. position, velocity, etc. The features of a single particle are given by the state $\boldsymbol{x} \in \mathbb{R}^{d_{\boldsymbol{x}}}$ of the particle. Raw data is provided as an input to the network, i.e. the features of all particles, stacked together in a matrix $\boldsymbol{s} \in \mathbb{R}^{N \times d_{\boldsymbol{x}}}$. A permutation of two particles represents in essence the same configuration and should be mapped to the same latent representation. This is achieved with a permutation invariant layer that first applies a non-linear transformation $\phi : \mathbb{R}^{d_{\boldsymbol{x}}} \to \mathbb{R}^{d_p}$ mapping each particles' features to a high-dimensional latent representation of dimension d_p. This mapping is applied to all particles independently leading to N such latent vectors. The mean of these vectors is taken to construct the representation of the configuration. The representation $\frac{1}{N}\sum_{i=1}^{N} \phi(\boldsymbol{x}^i)$ is finally fed to a final layer reducing the dimensionality to a low-order representation $\boldsymbol{z} \in \mathbb{R}^{d_{\boldsymbol{z}}}$, with $d_{\boldsymbol{z}} \ll d_p, N$. This is achieved by the mapping $g : \mathbb{R}^{d_p} \to \mathbb{R}^{d_{\boldsymbol{z}}}$. In this work, the permutation invariance layer is utilized in the modeling of the collective dynamics of a group of particles whose movement is governed by the advection-diffusion equation in the one and three dimensional space. Both mappings g and ϕ are implemented with neural networks, having 3 layers of 50 hidden units each, and tanh activations.

E. Mixture Density Decoder

Mixture density networks (MDNs) [34] are powerful neural networks that can model non-Gaussian, multi-modal data distributions. The outputs of MDNs are parameters of a mixture density model (mixture of probability density functions). The most generic choice of the mixture component distribution, is the Gaussian distribution. Gaussian MDNs are widely deployed in machine learning applications to model structured dynamic environments, i.e. (video) games. The effectiveness of MDNs, however, in modeling physical systems remains unexplored.

In physical systems, the state may be bounded. In this case, the choice of a Gaussian MDN is problematic due to its unbounded support. To make matters worse, most applications of Gaussian MDNs when modeling random vectors do not consider the interdependence between the vector variables, i.e. the covariance matrix of the Gaussian mixture components is diagonal, in an attempt to reduce their computational complexity. Arguably in the applications where they were successful, modeling this interdependence was not imperative. In contrast, in physical systems the variables of a state might be very strongly dependent on each other. In order to cope with these problems, the following approach is considered: Firstly, an auxiliary vector variable is considered $\boldsymbol{v}$ along with its distribution $p(\boldsymbol{v}|\boldsymbol{z})$. $\boldsymbol{v} \in \mathbb{R}^{d_{\boldsymbol{x}}}$ has the same dimensionality $d_{\boldsymbol{x}}$ as the high-dimensional state (input/output of the autoencoder). The distribution is modeled as a mixture of K **multivariate** normal distributions

$$p(\boldsymbol{v}|\boldsymbol{z}) = \sum_{k=1}^{K} \pi^k(\boldsymbol{z})\, \mathcal{N}\Big(\boldsymbol{\mu}_{\boldsymbol{v}}^k(\boldsymbol{z}), \Sigma_{\boldsymbol{v}}^k(\boldsymbol{z}) \Big), \tag{2}$$

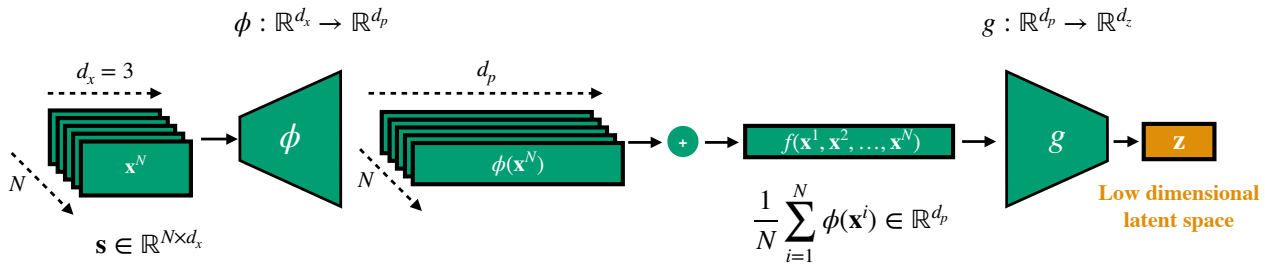


FIG. 2: Illustration of the permutation invariant encoder. The input of the network is composed of N atomic states that are permutation invariant, e.g. positions $\{\boldsymbol{x}^1, \ldots, \boldsymbol{x}^N\}$ of N particles in a particle simulation, each one with dimension $d_{\boldsymbol{x}}$, i.e. $\boldsymbol{x}^i \in \mathbb{R}^{d_{\boldsymbol{x}}}$, $\forall i \in \{1, \ldots, N\}$. A transformation $\phi(\cdot) : \mathbb{R}^{d_{\boldsymbol{x}}} \to \mathbb{R}^{d_p}$ is applied to each atomic state separately, mapping to a high-dimensional latent feature space. The mean of these latent representations of the atomic states is computed, leading to a singe latent feature that is permutation invariant with respect to the input. The final layer of the encoder maps the high-dimensional feature to a low dimensional representation $\boldsymbol{z}$, which is again permutation invariant with respect to the input, representing the encoding of the global state.

The multivariate normal distribution is parametrised in terms of a mean vector $\boldsymbol{\mu}_{\boldsymbol{v}}^k$, a positive definitive covariance matrix $\Sigma_{\boldsymbol{v}}^k$, and the mixing coefficients π^k which are functions of $\boldsymbol{z}$. The covariance matrix is parametrised by a lower-triangular matrix $L_{\boldsymbol{v}}^k$ with positive-valued diagonal entries, such that $\Sigma_{\boldsymbol{v}}^k = L_{\boldsymbol{v}}^k L_{\boldsymbol{v}}^{k\,T} \in \mathbb{R}^{d_{\boldsymbol{x}} \times d_{\boldsymbol{x}}}$ (This triangular matrix can be recovered by Cholesky factorization of the positive definite $\Sigma_{\boldsymbol{v}}^k$). The functional forms of $\pi^k(\boldsymbol{z}) \in \mathbb{R}$, $\boldsymbol{\mu}_{\boldsymbol{v}}(\boldsymbol{z}) \in \mathbb{R}^{d_{\boldsymbol{x}}}$, and the $n(n+1)/2$ entries of $L_{\boldsymbol{v}}^k$ are neural networks, their values are given by the outputs of the decoder for all mixture components $k \in \{1, \ldots, K\}$, i.e. $\boldsymbol{w}_{\mathcal{D}} = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}(\boldsymbol{z}) = \{\pi^k, \boldsymbol{\mu}_{\boldsymbol{v}}^k, L_{\boldsymbol{v}}^k\}_{1,\ldots,K}$. The positivity of the diagonal elements of $L_{\boldsymbol{v}}^k$ is ensured by a **softplus** activation function

$$f(x) = \ln(1 + \exp(x)) \tag{3}$$

in the respective outputs of the decoder. The mixing coefficients satisfy $0 \leq \pi^k < 1$ and $\sum_{k=1}^{K} \pi^k = 1$. To ensure these conditions, the respective outputs of the decoder are passed through a **softmax** activation

$$\sigma(\boldsymbol{x})_i = \frac{e^{\boldsymbol{x}_i}}{\sum_i e^{\boldsymbol{x}_i}}. \tag{4}$$

The rest (non-diagonal elements and mean vector) of the decoder outputs have linear activations, so no restriction in their sign. In total, the decoder output is composed of $K(n-1)n/2 + Kn$ single valued outputs with linear activation for the non-diagonal elements of $L_{\boldsymbol{v}}^k$ and the mean vectors $\boldsymbol{\mu}_{\boldsymbol{v}}^k$, and Kn positive outputs with softplus activation for the diagonal of $L_{\boldsymbol{v}}^k$, and K outputs with softmax activation for the mixing coefficients.

MD networks are employed in stochastic systems (e.g. molecular dynamics). In the following, we assume that the high-dimensional state of the molecular system is described by $\boldsymbol{s}_t \in \mathbb{R}^{d_{\boldsymbol{s}}}$. The decoder $\mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}$ is modeled with an MD network. The MD approximates the probability distribution of the state $\tilde{\boldsymbol{s}}_t \sim p(\cdot; \boldsymbol{w}_{\mathrm{MD}})$, where $\boldsymbol{w}_{\mathrm{MD}} = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}(\boldsymbol{z}_t)$ is the output of the decoder that parametrizes the distribution. The optimal parameters of the MD autoencoder are

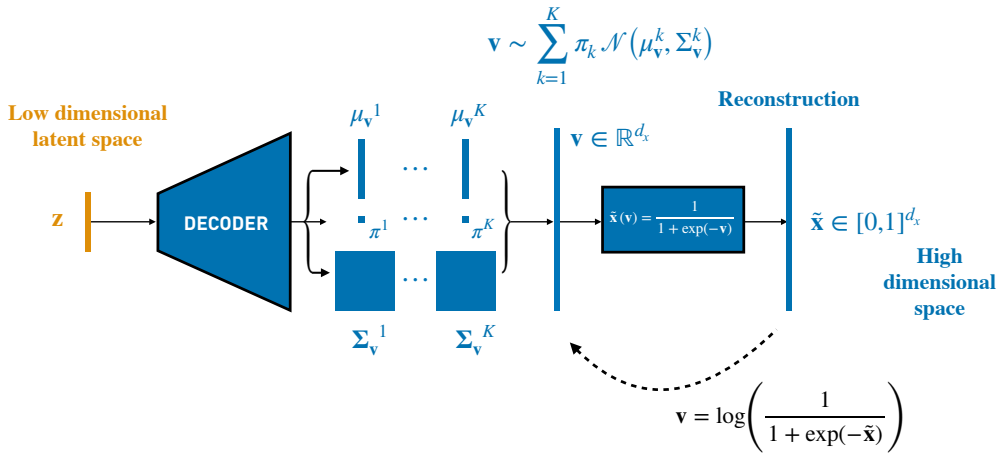


FIG. 3: A mixture density network modeling the probability density $p(\tilde{\boldsymbol{x}}|\boldsymbol{z})$, with bounded $\tilde{\boldsymbol{x}}$. The decoder maps the latent state $\boldsymbol{z}$ to the parameters of a mixture model on the latent vector $\boldsymbol{v} \in \mathbb{R}^{d_{\boldsymbol{x}}}$, which are the mixing coefficients $\pi^k \in \mathbb{R}$, mean vectors $\mu_v^k \in \mathbb{R}^{d_{\boldsymbol{x}}}$, and a lower-triangular matrix $L_{\boldsymbol{v}}^k \in \mathbb{R}^{d_{\boldsymbol{x}} \times d_{\boldsymbol{x}}}$ with positive-valued diagonal entries. From the latter, the covariance matrix is derived from $\Sigma_{\boldsymbol{v}}^k = L_{\boldsymbol{v}}^k L_{\boldsymbol{v}}^{kT}$ which is positive definite by construction. The mixture models the probability distribution of the latent state $p(\boldsymbol{v}|\boldsymbol{z})$. The targets, however, used to train the network in a supervised way are defined on the reconstruction $\tilde{\boldsymbol{x}}$. The targets are scaled to $\tilde{\boldsymbol{x}} \in [0,1]^{d_{\boldsymbol{x}}}$, and then transformed to targets for $\boldsymbol{v}$ using the inverse of the softplus activation. The MDN autoencoder is trained to maximize the likelihood $p(\boldsymbol{v}|\boldsymbol{z})$ of the transformed data $\boldsymbol{v}$.

identified by maximizing the log-likelihood of the reconstruction,

$$\boldsymbol{w}_{\mathcal{E}}^{\star}, \boldsymbol{w}_{\mathcal{D}}^{\star} = \underset{\boldsymbol{w}_{\mathcal{E}}, \boldsymbol{w}_{\mathcal{D}}}{\operatorname{argmax}} \log p\left(\boldsymbol{s}_t; \boldsymbol{w}_{\mathrm{MD}}\right),$$
$$\text{where} \quad \boldsymbol{w}_{\mathrm{MD}} = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}(\boldsymbol{z}_t) = \mathcal{D}^{\boldsymbol{w}_{\mathcal{D}}}\left(\mathcal{E}^{\boldsymbol{w}_{\mathcal{E}}}(\boldsymbol{s}_t)\right).$$

F. Long Short-Term Memory Recurrent Neural Networks (LSTM-RNNs)

In the low order manifold (coarse, latent state), a Recurrent Neural Network (RNN) is utilized to capture the non-linear, non-markovian dynamics. The forecasting rule of the RNN is given by

$$\boldsymbol{h}_t = \mathcal{H}^{\boldsymbol{w}_{\mathcal{H}}}\left(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}\right), \quad \tilde{\boldsymbol{z}}_{t+\Delta t} = \mathcal{R}^{\boldsymbol{w}_{\mathcal{R}}}\left(\boldsymbol{h}_t\right), \tag{5}$$

where $\boldsymbol{w}_{\mathcal{H}}$ and $\boldsymbol{w}_{\mathcal{R}}$ are the trainable parameters of the network, $\boldsymbol{h}_t \in \mathbb{R}^{d_{\boldsymbol{h}}}$ is an internal hidden memory state, and $\tilde{\boldsymbol{z}}_{t+\Delta t}$ is a prediction of the latent state. The RNN is trained to minimize the forecasting loss $||\tilde{\boldsymbol{z}}_{t+\Delta t} - \boldsymbol{z}_{t+\Delta t}||_2^2$, which can be written as

$$||\tilde{\boldsymbol{z}}_{t+\Delta t} - \boldsymbol{z}_{t+\Delta t}||_2^2 = ||\mathcal{R}^{\boldsymbol{w}_{\mathcal{R}}}\left(\boldsymbol{h}_t\right) - \boldsymbol{z}_{t+\Delta t}||_2^2 = ||\mathcal{R}^{\boldsymbol{w}_{\mathcal{R}}}\left(\mathcal{H}^{\boldsymbol{w}_{\mathcal{H}}}\left(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}\right)\right) - \boldsymbol{z}_{t+\Delta t}||_2^2. \tag{6}$$

This leads to

$$\boldsymbol{w}_{\mathcal{H}}, \boldsymbol{w}_{\mathcal{R}} = \underset{\boldsymbol{w}_{\mathcal{H}}, \boldsymbol{w}_{\mathcal{R}}}{\operatorname{argmin}} ||\mathcal{R}^{\boldsymbol{w}_{\mathcal{R}}}\left(\mathcal{H}^{\boldsymbol{w}_{\mathcal{H}}}\left(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}\right)\right) - \boldsymbol{z}_{t+\Delta t}||_2^2. \tag{7}$$

The RNNs are trained with Backpropagation through time (BPTT) [36]. The mappings $\mathcal{H}^{\boldsymbol{w}_{\mathcal{H}}}$ and $\mathcal{R}^{\boldsymbol{w}_{\mathcal{R}}}$, considered in this work take the form of the long short-term memory (LSTM) [35] cell. The output mapping is given by a linear transformation, i.e.

$$\tilde{\boldsymbol{z}}_{t+\Delta t} = W_{\boldsymbol{z},\boldsymbol{h}} \boldsymbol{h}_t, \tag{8}$$

where $W_{\boldsymbol{z},\boldsymbol{h}} \in \mathbb{R}^{d_{\boldsymbol{z}} \times d_{\boldsymbol{h}}}$. As a consequence, the set of trainable weights of the hidden-to-output mapping is just one matrix $\boldsymbol{w}_{\mathcal{R}} = W_{\boldsymbol{z},\boldsymbol{h}} \in \mathbb{R}^{d_{\boldsymbol{z}} \times d_{\boldsymbol{h}}}$.

The LSTM possesses two hidden states, a cell state $\boldsymbol{c}$ and an internal memory state $\boldsymbol{h}$. The hidden-to-hidden mapping

$$\boldsymbol{h}_t, \boldsymbol{c}_t = \mathcal{H}^{\boldsymbol{w}_\mathcal{H}}\big(\boldsymbol{z}_t, \boldsymbol{h}_{t-\Delta t}, \boldsymbol{c}_{t-\Delta t}\big) \tag{9}$$

takes the form

$$\begin{aligned} \boldsymbol{g}_t^f &= \sigma_f\big(W_f[\boldsymbol{h}_{t-\Delta t}, \boldsymbol{z}_t] + \boldsymbol{b}_f\big) & \boldsymbol{g}_t^i &= \sigma_i\big(W_i[\boldsymbol{h}_{t-\Delta t}, \boldsymbol{z}_t] + \boldsymbol{b}_i\big) \\ \tilde{\boldsymbol{c}}_t &= \tanh\big(W_c[\boldsymbol{h}_{t-\Delta t}, \boldsymbol{z}_t] + \boldsymbol{b}_c\big) & \boldsymbol{c}_t &= \boldsymbol{g}_t^f \odot \boldsymbol{c}_{t-\Delta t} + \boldsymbol{g}_t^i \odot \tilde{\boldsymbol{c}}_t \\ \boldsymbol{g}_t^{\boldsymbol{z}} &= \sigma_h\big(W_h[\boldsymbol{h}_{t-\Delta t}, \boldsymbol{z}_t] + \boldsymbol{b}_h\big) & \boldsymbol{h}_t &= \boldsymbol{g}_t^{\boldsymbol{z}} \odot \tanh(\boldsymbol{c}_t), \end{aligned} \tag{10}$$

where $\boldsymbol{g}_t^f, \boldsymbol{g}_t^i, \boldsymbol{g}_t^{\boldsymbol{z}} \in \mathbb{R}^{d_{\boldsymbol{h}}}$, are the gate vector signals (forget, input and output gates), $\boldsymbol{z}_t \in \mathbb{R}^{d_{\boldsymbol{z}}}$ is the latent input at time t, $\boldsymbol{h}_t \in \mathbb{R}^{d_{\boldsymbol{h}}}$ is the hidden state, $\boldsymbol{c}_t \in \mathbb{R}^{d_{\boldsymbol{h}}}$ is the cell state, while W_f, W_i, $W_c, W_h \in \mathbb{R}^{d_{\boldsymbol{h}} \times (d_{\boldsymbol{h}} + d_{\boldsymbol{z}})}$, are weight matrices and $\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c, \boldsymbol{b}_h \in \mathbb{R}^{d_{\boldsymbol{h}}}$ biases. The symbol $\odot$ denotes the element-wise product. The activation functions σ_f, σ_i and σ_h are sigmoids. The dimension of the hidden state $d_{\boldsymbol{h}}$ (number of hidden units) controls the capacity of the cell to encode history information. The set of trainable parameters of the recurrent mapping $\mathcal{H}^{\boldsymbol{w}_\mathcal{H}}$ is thus given by

$$\mathcal{H}^{\boldsymbol{w}_\mathcal{H}} = \{\boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_c, \boldsymbol{b}_h, W_f, W_i, W_c, W_h\} \tag{11}$$

Supplementary Information II: Comparison Measures

In this section, we elaborate on the metrics used to quantify the effectiveness of the proposed approach to capture the dynamics and the state statistics of the systems under study. The mean normalized absolute difference (MNAD) is used to quantify the prediction performance of a method in a deterministic system. This metric was selected to facilitate comparison of LED with equation-free variants [32]. The Wasserstein distance (WD) and the L1-Norm histogram distance (L1-NHD) are utilized to quantify the difference between distributions. These metrics are used in stochastic systems or in the comparison of state distributions.

A. Mean normalised absolute difference (MNAD)

Assume that a model is used to predict a spatiotemporal field $y(x,t)$, at discrete state x_i and time t_j locations. Predicted values from a model (neural network, etc.) are denoted with $\tilde{y}$, while the groundtruth (simulation of the equations with a solver based on first principles) with y. The normalized absolute difference (NAD) between the model output and the groundtruth is defined as

$$\mathrm{NAD}(t_j) = \frac{1}{N_x}\sum_{i=1}^{N_x}\frac{|y(x_i,t_j)-\hat{y}(x_i,t_j)|}{\max_{i,j}(y(x_i,t_j))-\min_{i,j}(y(x_i,t_j))}, \tag{12}$$

where N_x is the dimensionality of the discretized state x. The NAD depends on the time t_j. The mean NAD (MNAD) is given by the mean over time of the NAD score, i.e.

$$\mathrm{MNAD} = \frac{1}{N_T}\sum_{j=1}^{N_T}\mathrm{NAD}(t_j), \tag{13}$$

where N_T is the number of time-steps considered. The MNAD is used in the FitzHugh-Nagumo equation, and the Kuramoto-Sivashinsky equation, to quantify the prediction accuracy of LED and benchmark against other methods (e.g. other propagators on the latent space) or against other equation-free variants.

B. Pearson Correlation Coefficient

Assume as before, the spatiotemporal field $y(x,t)$, at discrete state x_i and time t_j locations. This can be vectorized in $y_{vec} = \mathrm{vec}(y(x,t)) \in \mathbb{R}^{N_x\cdot N_t\times 1}$. The same applies to the vectorized prediction $\tilde{y}_{vec} = \mathrm{vec}(\tilde{y}(x,t)) \in \mathbb{R}^{N_x\cdot N_t\times 1}$. We can compute the Pearson correlation coefficient, or simply correlation, as

$$\mathrm{Correlation} = \frac{\mathrm{COV}\left(y_{vec},\tilde{y}_{vec}\right)}{\sigma_{y_{vec}}\sigma_{\tilde{y}_{vec}}}, \tag{14}$$

where COV is the covariance, and σ is the standard deviation.

The correlation is used as a prediction performance metric in the Kuramoto-Sivashinsky equation.

C. Wasserstein Distance

The Wasserstein distance (WD), is a metric used to quantify the difference between the distribution functions of two random variables. It is defined as the integral of the absolute difference of the inverse Cumulative Distribution Functions (CDF) of the random variables. Assuming two random variables Z_1 and Z_2, with CDFs given by $\tau = F_{Z_1}(z)$ and $F_{Z_2}(z)$, with $\tau \in [0,1]$, the Wasserstein metric is defined as

$$\mathrm{WD}(Z_1,Z_2) = \int_0^1 |F_{Z_1}^{-1}(\tau) - F_{Z_2}^{-1}(\tau)|\, d\tau. \tag{15}$$

In high-dimensional problems, where the random variable is multivariate (random vector), we are reporting the mean WD of each variable after marginalization of all others.

D. L1-Norm Histogram Distance

In order to quantify the difference of the distributions of two random multivariate random variables Z_1 and Z_2, we employ in addition to the WD, the L1-Norm histogram distance. We measure this metric based on the L1 norm of the difference between the normalized histograms of the random variables computed on the same grid. The number of bins for the computation of the histograms, is selected according to Rice rule, given by $N_{bins} = \lceil 2\sqrt[3]{n} \rceil$ where n is the number of observations in the sample z. The WD and the L1-NHD are used to measure the difference between the spatial particle distributions in the Advection-Diffusion model.

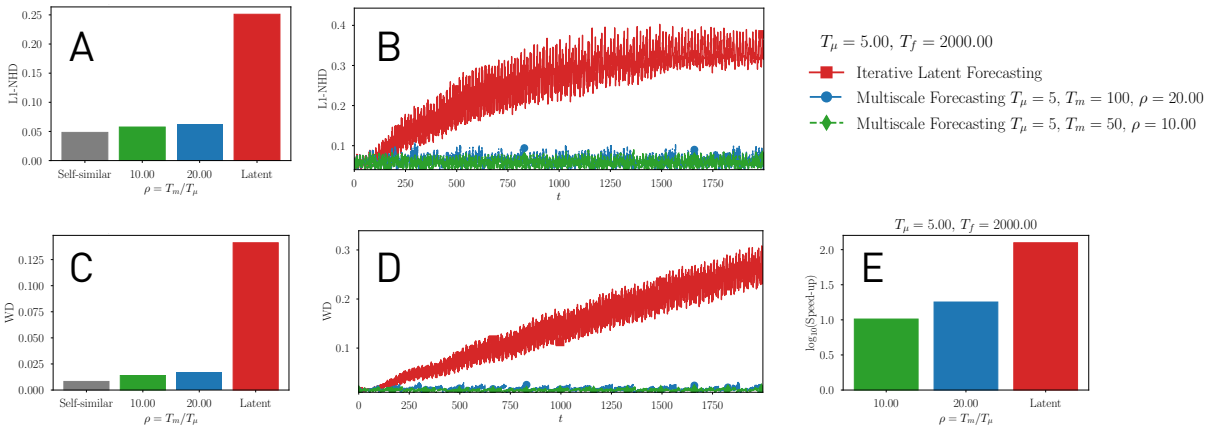


FIG. 4: **A)** The L1-Norm Histogram distance averaged over time and initial conditions. The self-similar error is plotted for reference, as errors below this level are statistically insignificant. **B)** The evolution of the L1-Norm Histogram distance in time averaged over initial conditions. **C)** The Wasserstein distance averaged over time and initial conditions. **D)** The evolution of the Wasserstein distance in time averaged over initial conditions. **E)** The speed-up of LED compared to the micro scale solver is plotted w.r.t. ρ.

A. LED for Advection-Diffusion Equation

The LED method is applied to the simulation of the advection-diffusion equation. The microscale description of the Advection-Diffusion (AD) process is modeled with a system of $N = 1000$ particles on a bounded domain $\Omega = [-L/2, L/2]^{d_{\boldsymbol{x}}}$. The particle dynamics are modeled with the stochastic differential equation (SDE)

$$\mathrm{d}\boldsymbol{x}_t = \boldsymbol{u}_t \mathrm{d}t + \sqrt{D}\,\mathrm{d}\boldsymbol{W}_t, \tag{16}$$

where $\boldsymbol{x}_t \in \Omega$ denotes the position of the particle at time t, $D \in \mathbb{R}$ is the diffusion coefficient, $\mathrm{d}\boldsymbol{W}_t \in \mathbb{R}^{d_{\boldsymbol{x}}}$ is a Wiener process, and $\boldsymbol{u}_t = \boldsymbol{A}\cos(\boldsymbol{\omega} t) \in \mathbb{R}^{d_{\boldsymbol{x}}}$ is a cosine advection (drift) term. In the following, the three dimensional space $d_{\boldsymbol{x}} = 3$ is considered, with $D = 0.1$, $\boldsymbol{A} = [1, 1.74, 0.0]^T$, $\boldsymbol{\omega} = [0.2, 1.0, 0.5]^T$, and a domain size of $L = 1$. The Péclet number quantifies the rate of advection by the rate of diffusion, i.e. $Pe = \frac{LU}{D}$. In this work, $L = 1$, $U = |\boldsymbol{A}|_2 \approx 2$, suggest a Péclet number of $Pe = 20$. Equation (16) is solved with explicit Euler integration with $\Delta t = 10^{-2}$, initial conditions $\boldsymbol{x}_0 = \boldsymbol{0}$, and reflective boundary conditions ensuring that $\boldsymbol{x}_t \in \Omega, \forall t$. The positions of the particles are saved at a coarser time-step $\Delta t = 1$. Three datasets are generated by starting from randomly selected initial conditions. The training and validation datasets consist of 500 samples each, and the test dataset consists of 4000 samples. The full state of the system is high-dimensional, i.e. $\boldsymbol{s}_t = [\boldsymbol{x}_t^1; \ldots; \boldsymbol{x}_t^N]^T \in \mathbb{R}^{N \times 3}$.

The particles concentrate on a few "meta-stable" states, and transition between them, suggesting that the collective dynamics can be captured by a few latent variables. It is not straightforward, however, to determine a-priori the number of these states and the patterns of collective motion. LED unravels this information and provides a computationally efficient multiscale model to approximate the system dynamics. An AE with a permutation invariant input layer with a latent dimension $d_{\boldsymbol{z}}$, an MD decoder and a stateful LSTM-RNN are employed to learn and forecast the dynamics on the low-dimensional manifold.

In this case, the latent dimension of LED is tuned to $d_{\boldsymbol{z}} = 8$ based on the log-likelihood on the validation data. Reducing the latent dimension further, caused a decrease in the validation log-likelihood loss, while increasing the latent dimension did not lead to any significant improvement. Regarding the rest of the LED hyper-parameters, the ϕ function consists of 3×50 layers and tanh activation, the permutation invariant space has dimension $M = 100$ with mean feature function, and the decoder g consists of a network with 3×50 layers and tanh activation, reducing the dimensionality to the desired latent state of dimension $d_{\boldsymbol{z}} = 8$. The decoder is composed of 3×50 layers, and a mixture density output layer, with 25 hidden units, and 5 kernels outputting the parameters for the mixture coefficients, the means, and the covariance matrices of the 5 kernels. The RNN propagating the dynamics in the latent space, is composed of one stateful LSTM layer with 25 nodes and was trained with BBTT with a sequence length of 100.

After training the RNN, the efficiency of LED in forecasting the dynamics is tested in 30 trajectories starting from different initial conditions randomly sampled from the testing data. The final prediction horizon is set to $T_f = 2000$. The particle spatial distribution predicted by LED is compared against the groundtruth in terms of the L1-Norm Histogram distance (L1-NHD) and the Wasserstein distance (WD). The results are shown in Figure 4. Three LED variants are considered. The first variant does not evolve the dynamics on the particle level (Latent-LED, $T_m = 0$) and its error increases with time and exhibits the highest errors on average. The second and third variants, (Multiscale-LED), evolve the low order manifold dynamics (coarse scale) for T_m time units, and the particle dynamics (fine scale) for $T_\mu = 5$ to correct iteratively for the statistical error. This effect is due to the explicit dependence of the coarse system dynamics in time, as the $\cos(\boldsymbol{\omega} t)$ advection term dominates. Two values for T_m are considered, $T_m = 50$ leading to a relative ratio of coarse to fine simulation time of $\rho = T_m/T_\mu = 10$, and another one with $T_m = 100$, leading to $\rho = 20$. This incurs additional computational cost induced by the evolution of the high-dimensional state. The warm-up time is $T_{warm} = 100$ for all variants. As the multiscale ratio $\rho = T_m/T_\mu$ is increased, spending more time in the latent propagation, the errors gradually increase. The propagation in the low dimensional latent space is far less computationally expensive compared to the evolution of the high-dimensional dynamics. As ρ is increased, greater computati
LED is abl

The effe
lower error
is isotropic

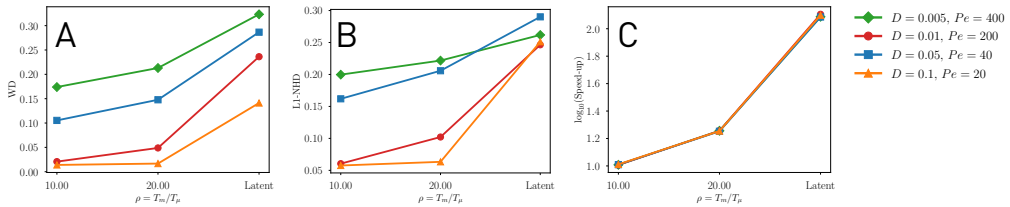
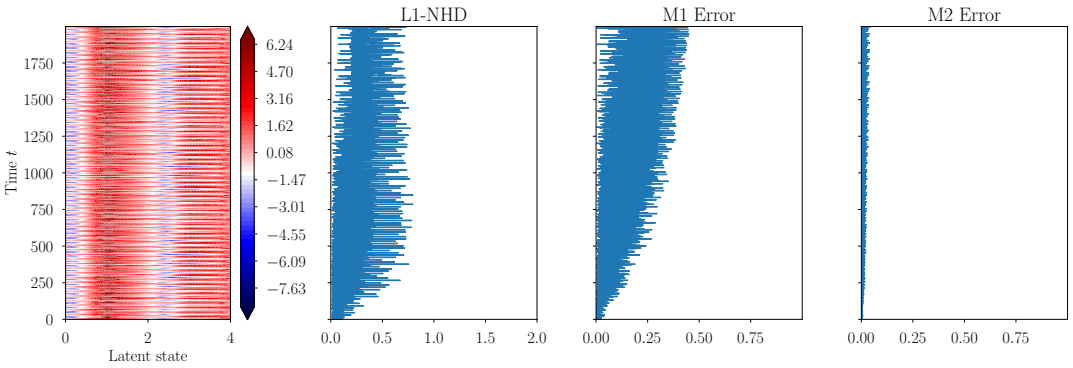


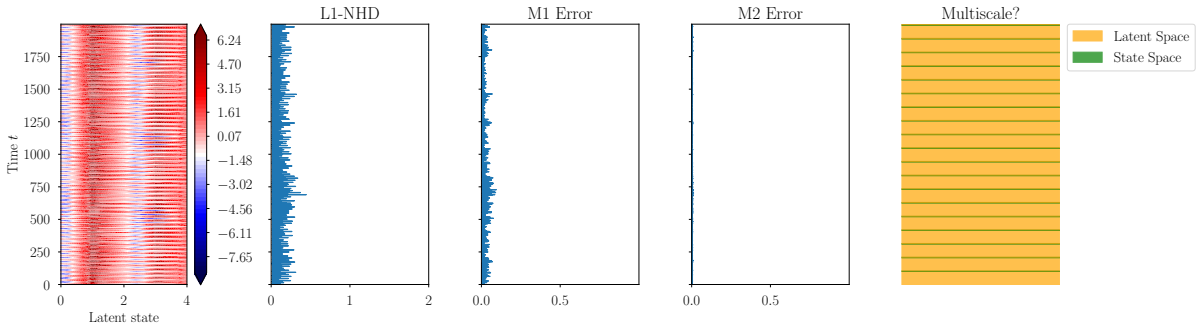
FIG. 5: Analysis of the performance of LED for different Péclet numbers $Pe \in \{20, 40, 200, 400\}$. Three LED variants are considered, Latent-LED ($T_m = 0$), and two variants of Multilscale-LED with $T_\mu = 5$ and $T_m \in \{50, 100\}$. The warm-up time is $T_{warm} = 100$ for all variants. **A)** The Wasserstein distance and **B)** L1-Norm Histogram distance between the particle spatial distributions averaged over time and initial conditions, plotted with respect to the multiscale ratio ρ. The methods consistently exhibit lower error as the Péclet number decreases. **C)** The speed-up is plotted w.r.t. ρ.

An example of the evolution of the latent state, the errors on the first two moments, and the L1-NHD between the groundtruth and the predicted spatial distribution of particles in an iterative prediction on the test data is shown in Figure 6a. The initial warm-up period of LED is set to $T_{warm} = 100$. LED captures the variance of the particle positions but due to the iterative error propagation the error on the distribution (L1-NHD and mean position) is increasing with time.

In Figure 7, the latent space of LED is clustered to identify frequently visited metastable states that can be mapped back to their respective particle configurations using the decoder.



(a) LED $T_\mu = 0$



(b) LED $T_m = 100$, $T_\mu = 5$, $\rho = 20$

FIG. 6: **(a)** LED applied on the 3-dimensional Advection-Diffusion equation, iteratively forecasting the evolution of the particles starting from an initial condition in the test data. The initial warm-up period of LED is set to $T_{warm} = 100$. An AE with a permutation invariant input layer, and a latent dimension of $d_{\boldsymbol{z}} = 8$ is utilized to coarse-grain the high-dimensional dynamics. The decoder of LED is mapping from the latent space to the particle configuration using a MD decoder. We plot the evolution of the latent state in time, along with the L1-NHD between the predicted and groundtruth particle distributions and the absolute error on the mean, and the standard deviation of the particle distributions. LED can forecast the evolution of the particle positions with low error, even though the total dimensionality of the original state describing the configuration of the $N = 1000$ particles of the system is $\boldsymbol{s}_t \in \mathbb{R}^{1000\times 3}$. The network, learned an $d_{\boldsymbol{z}} = 8$ dimensional coarse-grained representation of this configuration. However, due to the iterative prediction with LED, the error on the predicted distribution of particles is increasing with time. **(b)** Multiscale propagation in LED. To alleviate the iterative error propagation, the multiscale propagation is utilized with $T_m = 100$, $T_\mu = 5$, $\rho = 20$. Due to the iterative transition between propagation in the latent space $\boldsymbol{z}_t$ of LED for T_m and evolution of the micro-scale particle dynamics for T_μ, the effect of iterative statistical error propagation is alleviated.

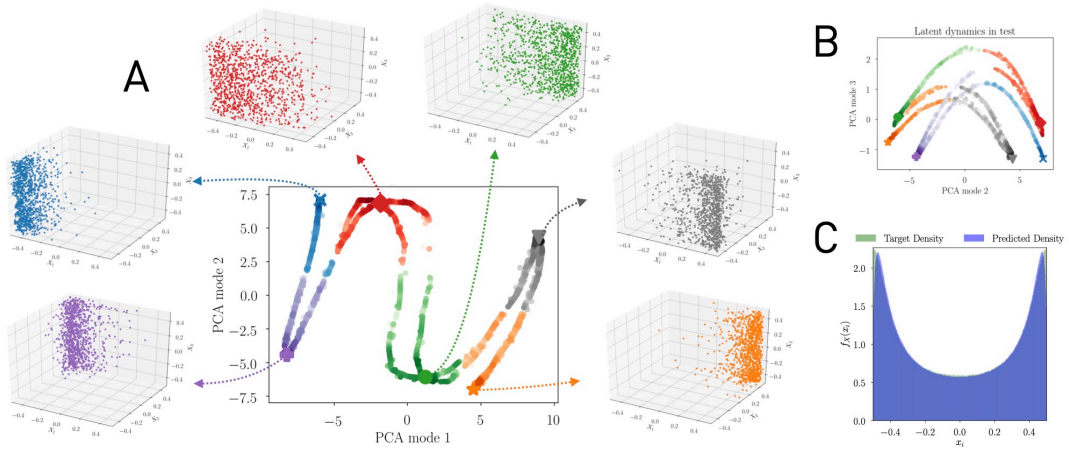


FIG. 7: **A)** Evolution of the second PCA mode of the latent state $\boldsymbol{z}_t \in \mathbb{R}^{d_{\boldsymbol{z}}=8}$, against the first mode. Higher color intensity denotes higher density. Six high density regions are identified. Spectral clustering on the PCA modes of the latent dynamics reveals the clusters. The six cluster centers are marked, while color illustrates the cluster membership. The LED probabilistic decoder is employed to map each cluster center to a realization of a high-dimensional simulation state. LED effectively unravels six meta stable states of the Advection-Diffusion equation, along with the transitions between them, representing the low order effective dynamics. **B)** Evolution of the third PCA mode against the second one, colored according to cluster assignment. **C)** Density of the particle positions from simulation plotted against the distribution of the positions predicted by LED. We remark the good agreement between the two distributions.

a. Hyper-parameter Tuning

The hyper-parameters of LED are given in Table 1 for the Autoencoder, and Table 2 for the RNN.

TABLE 1: Autoencoder hyper-parameters for Advection-Diffusion in 3-D ($d_{\boldsymbol{x}} = 3$)

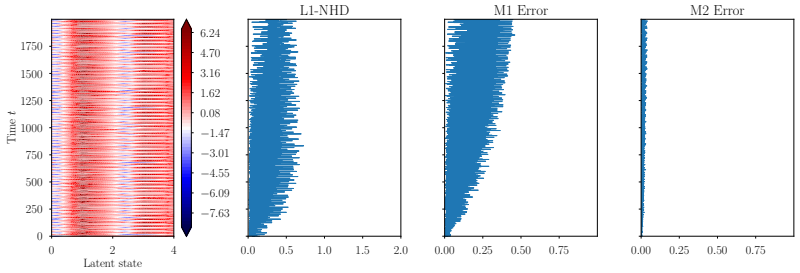
Hyper-parameter	Values
Number of AE layers	$\{3\}$
Size of AE layers	$\{50\}$
Activation of AE layers	$\tanh(\cdot)$
Latent dimension	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 18, 22, 24, 28, 32, 64\}$
Residual connections	False
Variational	True/False
Permutation Invariant Layer d_p	$\{200, 1001\}$
Number of MD kernels K	$\{5\}$
Hidden units of MD decoder	$\{50\}$
Input/Output data scaling	Min-Max in $[0, 1]$
Noise level in the data	$\{0, 1, 10\}$ (‰)
Weight decay rate	$\{0.0, 0.00001\}$
Batch size	32
Initial learning rate	0.001

TABLE 2: LED-RNN hyper-parameters for Advection-Diffusion in 3-D ($d_{\boldsymbol{x}} = 3$)

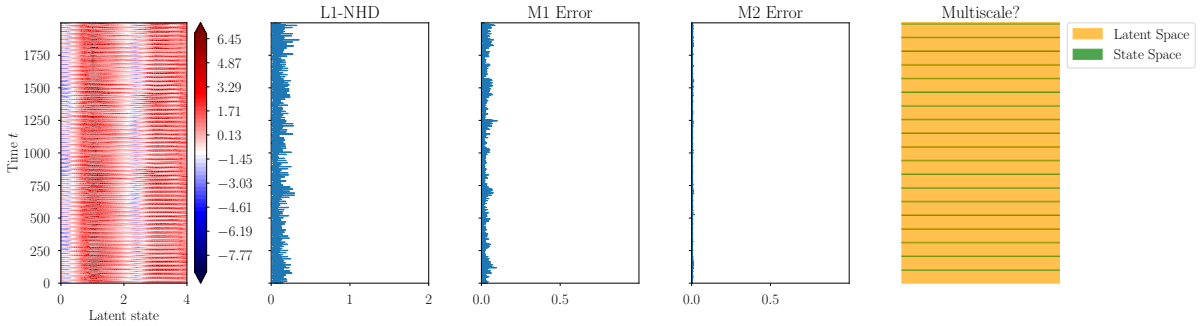
Hyper-parameter	Values
Number of AE layers	$\{3\}$
Size of AE layers	$\{50\}$
Activation of AE layers	$\tanh(\cdot)$
Latent dimension	$\{8\}$
Residual connections	False
Variational	False
Permutation Invariant Layer d_p	$\{200\}$
Number of MD kernels K	$\{5\}$
Hidden units of MD decoder	$\{50\}$
Input/Output data scaling	Min-Max in $[0, 1]$
Noise level in the data	$\{0\}$
Weight decay rate	$\{0.0\}$
Batch size	32
Initial learning rate	0.001
BBTT Sequence length	$\{100\}$
RNN cell type	$\{\text{lstm}, \text{gru}\}$
Number of RNN layers	$\{1\}$
Size of RNN layers	$\{25\}$
Activation of RNN Cell	$\tanh(\cdot)$

b. *Generalization to Different Number of Particles*

In this section, we provide additional results on the **generalization** of LED for a different **number of particles** in the simulation. Due to the permutation invariant encoder, coarse-graining the high-dimensional input of LED, the network is expected to be able to generalize to a different number of particles, since the identified coarse representation should rely on global statistical quantities, and not depend on individual positions. LED trained in configurations of $N = 1000$ particles is utilized to forecast the evolution of $N = 400$ particles evolving according to the Advection-Diffusion equation. The propagation of the errors is plotted in Figure 8. The initial warm-up period of LED is set to $T_{warm} = 100$ for all variants. We observe an excellent generalization ability of the network.



(a) LED $T_{\mu=0}$

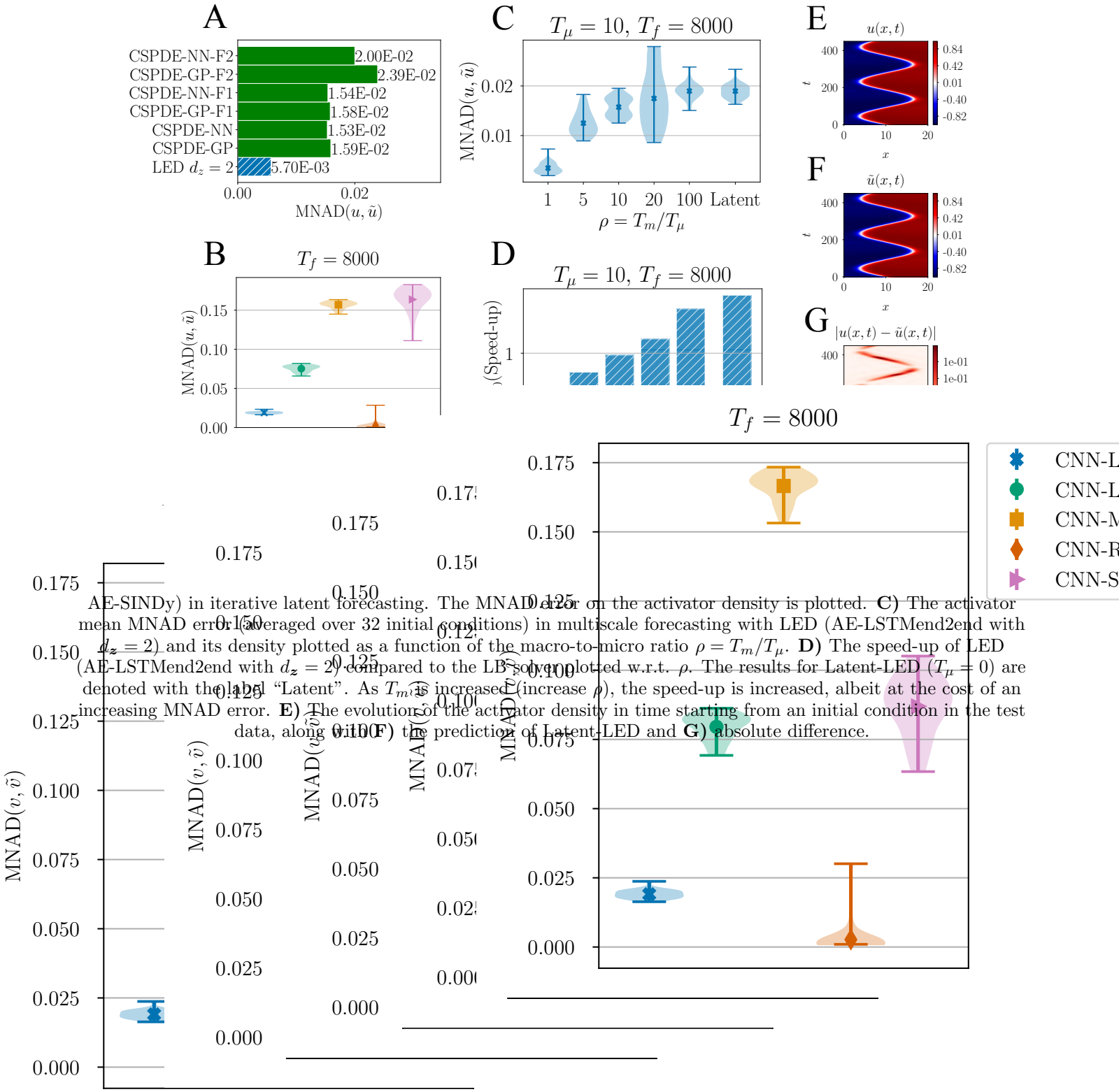


(b) LED $T_\mu = 100, \rho = 20$

FIG. 8: LED trained on particle configurations with $N = 1000$ number of particles, learned an $d_{\boldsymbol{z}} = 8$ dimensional coarse-grained representation of this configuration. We utilize two models with $T_\mu = 0$ (iterative latent propagation) and $T_\mu = 100, \rho = 20$ (multiscale forecasting) to forecast the evolution of a particle configuration composed of $N = 400$ particles to test the generalization ability of the model. The initial warm-up period is set to $T_{warm} = 100$. We plot the latent space, the L1-NHD between the densities of the particle positions, and the error on the first two moments, for both variants of LED. We observe that the LED is able to successfully generalize to the case of $N = 400$ particles.

B. FitzHugh-Nagumo Model (FHN)

The hyper-parameters of the Autoencoder are reported in Appendix III B a. Input and output are scaled to $[0, 1]$ and an output activation function of the form $1 + 0.5 \tanh(\cdot)$ is used to ensure that the data at the output lie at this range. The architecture of the CNN we employed is given in Figure 10. In this case, the inhibitor and activator density are cc



AE-SINDy) in iterative latent forecasting. The MNAD error on the activator density is plotted. **C)** The activator mean MNAD error (averaged over 32 initial conditions) in multiscale forecasting with LED (AE-LSTMend2end with $d_z = 2$) and its density plotted as a function of the macro-to-micro ratio $\rho = T_m/T_\mu$. **D)** The speed-up of LED (AE-LSTMend2end with $d_z = 2$) compared to the LB solver plotted w.r.t. ρ. The results for Latent-LED ($T_\mu = 0$) are denoted with the label "Latent". As T_m is increased (increase ρ), the speed-up is increased, albeit at the cost of an increasing MNAD error. **E)** The evolution of the activator density in time starting from an initial condition in the test data, along with **F)** the prediction of Latent-LED and **G)** absolute difference.

a. Hyper-parameters and Training Time

The hyper-parameter tuning of the autoencoder of LED and training times are reported in Table 3. PCA and Diffusion maps have very short fitting (training) times of approximately one minute. The layers of the CNN autoencoder employed in the FHN and its training times are given in Table 5. The architecture of the CNN autoencoder employed in the FHN is depicted in Figure 10.

The hyper-parameters for the LSTM and its training times are given in Table 4. For the MLP, a three layered network with CELU activations is employed. Training time for the MLP is 100 minutes. The hyper-parameters and training times for the RC are given in Table 6. The hyper-parameters and training times for SINDy are given in Table 7.

In all cases, the parameters of the best performing model on the validation data is denoted with red color.

TABLE 3: Autoencoder hyper-parameters and training times for FHN

Hyper-parameter tuning	Values
Number of AE layers	{3}
Size of AE layers	{100}
Activation of AE layers	celu(·)
Latent dimension	{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64}
Input/Output data scaling	[0, 1]
Output activation	$1 + 0.5\tanh(\cdot)$
Weight decay rate	{0.0, 0.0001}
Batch size	32
Initial learning rate	0.001

Training times [minutes]		
Min	Mean	Max

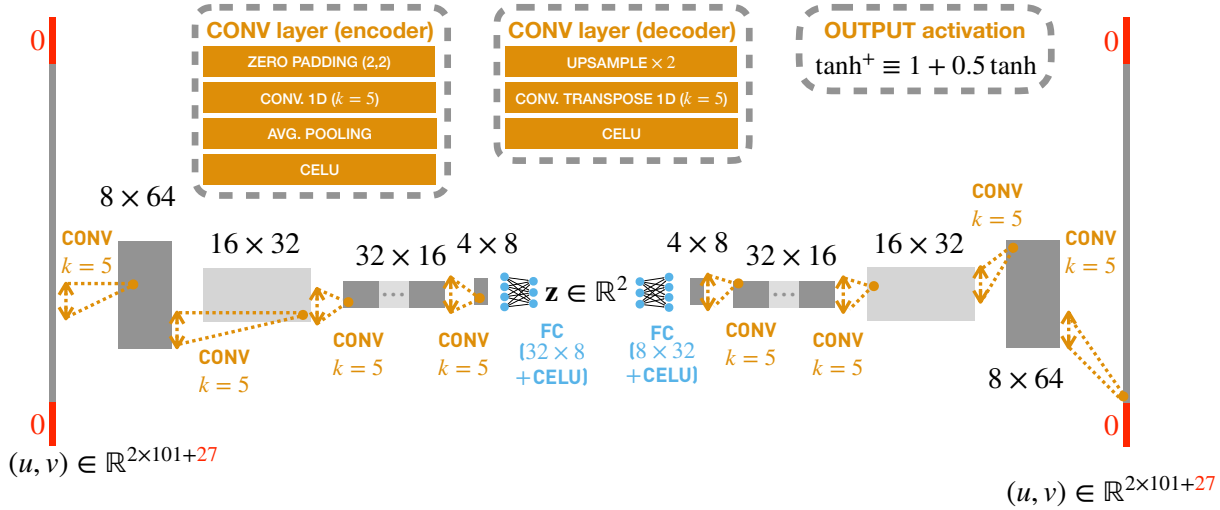


FIG. 10: The architecture of the CNN employed in the FHN equation. First, the input is padded to the closest power of two. Then, four layers of consecutive application of 1D convolutions, average pooling, CELU activations functions and dropout are used. Then an MLP is utilized to project to the low-order latent space. The output activation of the MLP is also $1 + 0.5\tanh(\cdot)$.

TABLE 4: LED-RNN hyper-parameters and training times for FHN

Hyper-parameter	Values
end2end training	True / False
Number of AE layers	{3}
Size of AE layers	{100}
Activation of AE layers	celu(·)
Latent dimension	2
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001
BPTT Sequence length	$\{20, 40, 60\}$
Output forecasting loss	True/False
RNN cell type	lstm
Number of RNN layers	1
Size of RNN layers	$\{16, 32, 64\}$
Activation of RNN Cell	tanh(·)
Output activation of RNN Cell	$1 + 0.5 \tanh(\cdot)$

Training times [minutes]	Min	Mean	Max
end2end training	2.2	2.5	2.8
only the RNN (sequential)	0.9	1.2	1.6

TABLE 5: CNN Autoencoder and training times for FHN

Layer	ENCODER
(0)	ConstantPad1d(padding=(13, 14), value=0.0)
(1)	ConstantPad1d(padding=(2, 2), value=0.0)
(2)	Conv1d(2, 8, kernel_size=(5,), stride=(1,))
(3)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(4)	CELU(alpha=1.0)
(5)	ConstantPad1d(padding=(2, 2), value=0.0)
(6)	Conv1d(8, 16, kernel_size=(5,), stride=(1,))
(7)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(8)	CELU(alpha=1.0)
(9)	ConstantPad1d(padding=(2, 2), value=0.0)
(10)	Conv1d(16, 32, kernel_size=(5,), stride=(1,))
(11)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(12)	CELU(alpha=1.0)
(13)	ConstantPad1d(padding=(2, 2), value=0.0)
(14)	Conv1d(32, 4, kernel_size=(5,), stride=(1,))
(15)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(16)	Flatten(start_dim=-2, end_dim=-1)
(17)	Linear(in_features=32, out_features=$d_{\mathbf{z}}$, bias=True)
(18)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^{d_{\mathbf{z}}}$
Layer	**DECODER**
(1)	Linear(in_features=$d_{\mathbf{z}}$, out_features=32, bias=True)
(2)	CELU(alpha=1.0)
(3)	Upsample(scale_factor=2.0, mode=linear)
(4)	ConvTranspose1d(4, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(5)	CELU(alpha=1.0)
(6)	Upsample(scale_factor=2.0, mode=linear)
(7)	ConvTranspose1d(32, 16, kernel_size=(5,), stride=(1,), padding=(2,))
(8)	CELU(alpha=1.0)
(9)	Upsample(scale_factor=2.0, mode=linear)
(10)	ConvTranspose1d(16, 8, kernel_size=(5,), stride=(1,), padding=(2,))
(11)	CELU(alpha=1.0)
(12)	Upsample(scale_factor=2.0, mode=linear)
(13)	ConvTranspose1d(8, 2, kernel_size=(5,), stride=(1,), padding=(2,))
(14)	1 + 0.5 Tanh()
(15)	Unpad()
Latent dimension $d_{\mathbf{z}}$	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$

Training times [minutes]

Min	Mean	Max
215	370	530

TABLE 6: Reservoir Computer hyper-parameters and training times (in CNN-RC) for FHN

Hyper-parameter tuning	Values
Solver	Pseudoinverse
Size	1000
Degree	10
Radius	0.99
Input scaling σ	$\{0.5, 1, 2\}$
Dynamics length	100
Regularization η	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10, 20, 30, 40, 100\}$

Training times [minutes]

Min	Mean	Max
0.15	0.18	0.19

TABLE 7: SINDy hyper-parameters and training times (in CNN-SINDy) for FHN

Hyper-parameter tuning	Values
Degree	$\{1, 2, 3\}$
Threshold	$\{0.001, 0.0001, 0.00001\}$
Library	Polynomials

Training times [minutes]

Min	Mean	Max
0.14	0.23	0.32

C. Kuramoto-Sivashinsky

A KS trajectory is plotted in Figure 11, along with the latent space evolution of Latent-LED and the predicted trajectory. We observe that the long-term climate is reproduced, although the LED is propagating an 8-dimensional latent state.

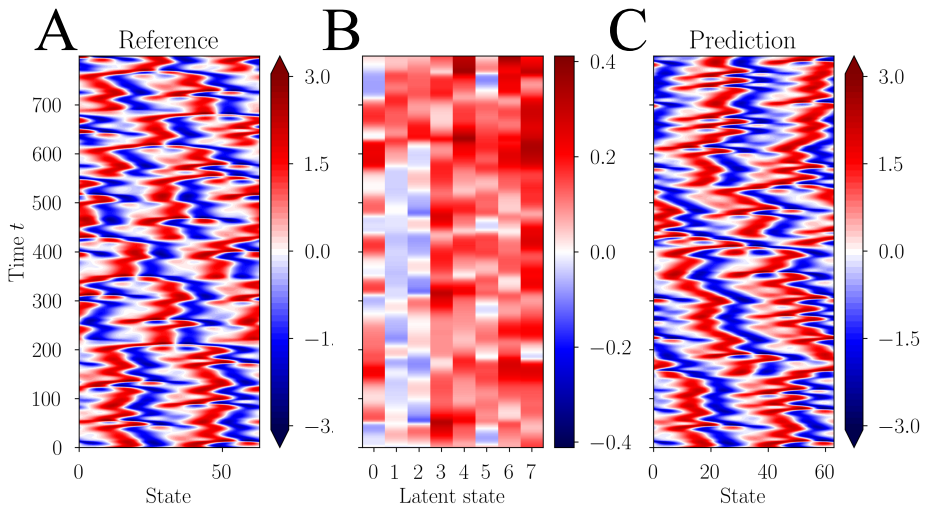


FIG. 11: **A)** Contour plot of the KS dynamics starting from an initial condition from the test data. **B)** The evolution of the $d_{\boldsymbol{z}} = 8$ dimensional latent state of Latent-LED. **C)** The predicted field by Latent-LED iteratively propagating the dynamics on a $d_{\boldsymbol{z}} = 8$ dimensional latent space, after a warm-up period $T_{warm} = 60$ ($T_\mu = 0$).

a. Hyper-parameters and Training Times

The hyper-parameter tunings and training times for the AE and CNN are given in Table 8 and Table 9 respectively . The architecture of the CNN autoencoder employed in KS is given in Table 10 along with the training times, and depicted in Figure 12. PCA fitting time is approximately one minute. The hyper-parameters and training times of the LSTM-RNN of LED are given in Table 11. The hyper-parameters and training times of the RC are given in Table 12. The hyper-parameters and training times of SINDy are given in Table 13.

TABLE 8: Autoencoder hyper-parameters for KS

Hyper-parameter tuning	Values
Number of AE layers	$\{3\}$
Size of AE layers	$\{100\}$
Activation of AE layers	$\text{celu}(\cdot)$
Latent dimension	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16, 20, 24, 28, 32, 36, 40, 64\}$
Input/Output data scaling	$[0, 1]$
Output activation	$1 + 0.5 \tanh(\cdot)$
Weight decay rate	$\{0.0, 0.0001\}$
Batch size	32
Initial learning rate	0.001

Training times [minutes]

Min	Mean	Max
160	192	311

TABLE 9: CNN hyper-parameters for KS

Hyper-parameter	Values
Convolutional	True
Kernels	Encoder: $5-5-5-5$, Decoder: $5-5-5-5$
Channels	$1-16-32-64-8-\mathbf{d}_z-8-64-32-16-1$
Batch normalization	True / False
Transpose convolution	True / False
Pooling	Average
Activation	celu$(\cdot)$
Latent dimension	$\{1,2,3,4,5,6,7,8,9,10,11,12,16,20,24,28,32,36,40,64\}$
Input/Output data scaling	$[0,1]$
Output activation	$1+0.5\tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001

Training times [minutes]

Min	Mean	Max
236	311	476

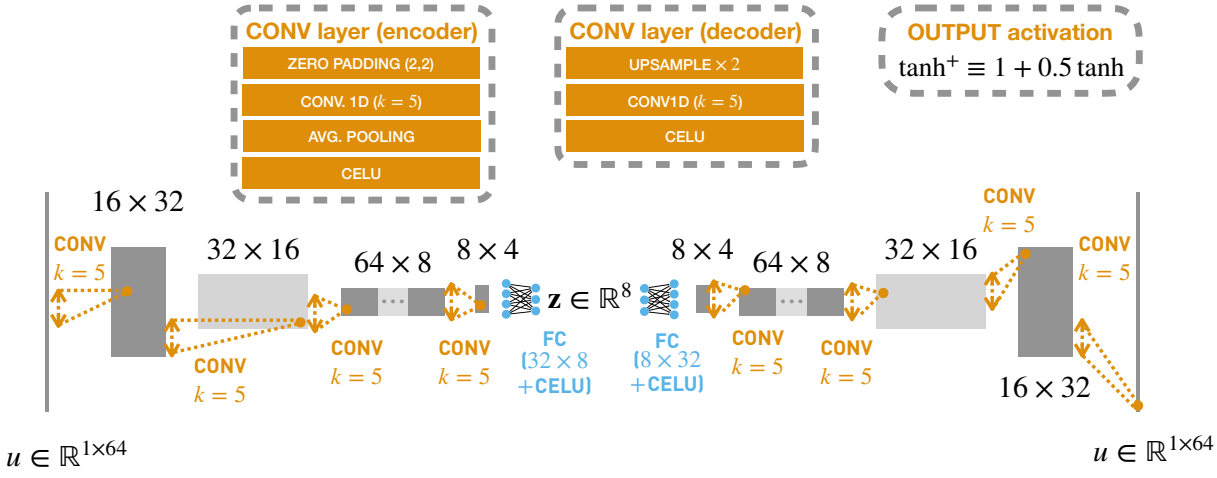


FIG. 12: The architecture of the CNN employed in KS.

TABLE 10: CNN Autoencoder for KS

Layer	ENCODER
(1)	ConstantPad1d(padding=(2, 2), value=0.0)
(2)	Conv1d(1, 16, kernel_size=(5,), stride=(1,))
(3)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(4)	CELU(alpha=1.0)
(5)	ConstantPad1d(padding=(2, 2), value=0.0)
(6)	Conv1d(16, 32, kernel_size=(5,), stride=(1,))
(7)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(8)	CELU(alpha=1.0)
(9)	ConstantPad1d(padding=(2, 2), value=0.0)
(10)	Conv1d(32, 64, kernel_size=(5,), stride=(1,))
(11)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(12)	CELU(alpha=1.0)
(13)	ConstantPad1d(padding=(2, 2), value=0.0)
(14)	Conv1d(64, 8, kernel_size=(5,), stride=(1,))
(15)	AvgPool1d(kernel_size=(2,), stride=(2,), padding=(0,))
(16)	CELU(alpha=1.0)
(17)	Flatten(start_dim =-2, end_dim = -1)
(18)	Linear(in_features =32, out_features =8, bias=True)
(19)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^8$

Layer	DECODER
(1)	Linear(in_features=8, out_features=32, bias=True)
(2)	CELU(alpha=1.0)
(3)	Upsample(scale_factor=2.0, mode=linear)
(4)	Conv1d(8, 64, kernel_size=(5,), stride=(1,), padding=(2,))
(5)	CELU(alpha=1.0)
(6)	Upsample(scale_factor=2.0, mode=linear)
(7)	Conv1d(64, 32, kernel_size=(5,), stride=(1,), padding=(2,))
(8)	CELU(alpha=1.0)
(9)	Upsample(scale_factor=2.0, mode=linear)
(10)	Conv1d(32, 16, kernel_size=(5,), stride=(1,), padding=(2,))
(11)	CELU(alpha=1.0)
(12)	Upsample(scale_factor=2.0, mode=linear)
(13)	Conv1d(16, 1, kernel_size=(5,), stride=(1,), padding=(2,))
(14)	1 + 0.5 Tanh()

TABLE 11: LED (LSTM-RNN) hyper-parameters and training times for KS

Hyper-parameter	Values
end2end training	False / True
Convolutional AE (CNN)	True
Kernels	Encoder: $5-5-5-5$, Decoder: $5-5-5-5$
Channels	$1-16-32-64-8-\mathbf{d}_z-8-64-32-16-1$
Batch normalization	False
Transpose convolution	False
Pooling	Average
Activation	celu$(\cdot)$
Latent dimension	8
Input/Output data scaling	$[0,1]$
Output activation	$1+0.5\tanh(\cdot)$
Weight decay rate	0.0
Batch size	32
Initial learning rate	0.001
BPTT Sequence length	$\{25, 50, 100\}$
Output forecasting loss	True/False
RNN cell type	lstm
Number of RNN layers	1
Size of RNN layers	$\{64, 128, 256, 512\}$
Activation of RNN Cell	tanh$(\cdot)$
Output activation of RNN Cell	$1+0.5\tanh(\cdot)$

Training times [minutes]	Min	Mean	Max
end2end training	476	978	1140
only the RNN (sequential)	960	1100	1140

TABLE 12: Reservoir Computer hyper-parameters and training times (in CNN-RC) for KS

Hyper-parameter tuning	Values
Solver	Pseudoinverse
Size	1000
Degree	10
Radius	0.99
Input scaling σ	$\{0.5, 1, 2\}$
Dynamics length	100
Regularization η	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10, 20, 30, 40, 100\}$

Training times [minutes]

Min	Mean	Max
0.25	0.35	0.38

TABLE 13: SINDy hyper-parameters and training times (in CNN-SINDy) for KS

Hyper-parameter tuning	Values
Library	Polynomials
Degree	$\{1, 2, 3\}$
Threshold	$\{0.001, 0.0001, 0.00001\}$

Training times [minutes]

Min	Mean	Max
0.13	0.62	1.59

D. Viscous Flow Behind a Cylinder

The flow behind a cylinder in the two dimensional space is simulated by solving the incompressible Navier-Stokes equations with Brinkman penalization to enforce the no-slip boundary conditions on the surface of the cylinder [51, 52], i.e.

$$\begin{aligned} \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} &= -\frac{\nabla p}{\rho} + \nu\Delta\mathbf{u} + \lambda\chi^{(s)}(\mathbf{u}^{(s)} - \mathbf{u})\,, \\ \nabla\cdot\mathbf{u} &= 0\,, \end{aligned} \tag{17}$$

where $\mathbf{u} = [u_x, u_y]^T \in \mathbb{R}^2$ is the velocity, $p \in \mathbb{R}$ is the pressure field, ρ is the density, ν is the kinematic viscocity, and λ is the penalization coefficient. The velocity-field $\mathbf{u}^{(s)} \in \mathbb{R}^2$ describes the translation of the cylinder. The numerical method of the flow solver is finite differences, with the incompressibility enforced through pressure projection. The computational domain is $\Omega = [0,1]\times[0,0.5]$, the cylinder is positioned at $(0.2, 0.5) \in \Omega$, with diameter $D = 0.075$. The cylinder is described by the characteristic function $\chi^{(s)}$, that is $\chi^{(s)} = 1$ inside the cylinder $\Omega^{(s)}$ and $\chi^{(s)} = 0$ outside $\Omega \setminus \Omega^{(s)}$. We consider the application of LED to two Reynolds' numbers $Re = 100$ and $Re = 1000$, by setting $\nu = 0.0001125$ and $\nu = 0.00001125$ respectively. The Strouhal number (defined in the SI Equation 26) is $\mathrm{St} = 0.175$, and $\mathrm{St} = 0.225$ for $Re = 100$ and $Re = 1000$ respectively. For both cases, the domain is discretized using 1024×512 grid-points and the time-step δt is adapted to ensure that the CFL number is fixed at 0.5. More details on the domain size and simulation are provided in the SI 3 D.

Equation 17 is solved for the velocity $\mathbf{u} \in \mathbb{R}^2$ and pressure field $p \in \mathbb{R}$ using the pressure projection method. First, we perform advection and diffusion of the flow field in the whole domain

$$\mathbf{u}^* = \mathbf{u}^t + \delta t\left(\nu\Delta\mathbf{u}^t - (\mathbf{u}^t\cdot\nabla)\mathbf{u}^t\right)\,. \tag{18}$$

The continuity equation requires the field to be divergence-free. This condition is imposed with the pressure projection

$$\mathbf{u}^{**} = \mathbf{u}^* - \delta t\frac{\nabla p^{t+1}}{\rho}\,. \tag{19}$$

The pressure field used here is obtained by solving the Poisson equation emerging from the divergence of Equation (19), i.e.

$$\Delta p^{t+1} = \frac{\rho}{\delta t}\nabla\cdot\mathbf{u}^*\,. \tag{20}$$

Note that adding Equation (19) and Equation (18) yields the original Equation (17) without the penalization term for Euler timestepping. The time-step is completed by applying the penalization force using $\delta t\lambda = 1$,

$$\mathbf{u}^{t+1} = \mathbf{u}^{**} + \chi^{(s),t+1}(\mathbf{u}^{(s),t+1} - \mathbf{u}^{**})\,. \tag{21}$$

We remark that the penalisation force acts as a Lagrange multiplier enforcing the translation motion of the cylinder on the fluid. The temporally discrete equations described above are solved on a grid with spacing Δx using second-order central finite differences for diffusion terms, and a third-order upwind scheme for advection terms.

For the simulated impulsively started cylinder the Reynolds-number for a cylinder with diameter D moving with velocity v in a fluid with kinematic viscosity ν is defined as

$$Re = \frac{Dv}{\nu}. \tag{22}$$

In the present simulations the cylinder moves with constant velocity $v = 0.15$ in $-x$-direction. The computational domain is chosen to be $\Omega = [0,1]\times[0,0.5]$ and moves with the center of mass of the sphere with diameter $D = 0.075$, that is fixed at $(0.2, 0.5) \in \Omega$. Here we present results for a simulation at $Re = 100$ and $Re = 1000$ by setting the kinematic viscocity to be $\nu = 0.0001125$ and $\nu = 0.00001125$ respectively. For both cases, the domain is discretized using 1024×512 gridpoints and the time-step δt is adapted to ensure that the CFL-number is fixed at 0.5.

The Strouhal number St describes the periodic vortex shedding at the wake of the cylinder. It is defined as

$$St = \frac{Df}{\nu}. \tag{23}$$

where f is the frequency of vortex shedding. In our case, $\mathrm{St} = 0.175$ for $Re = 100$, and $\mathrm{St} = 0.225$ for $Re = 1000$.

The state of the simulation is described by the velocity $\mathbf{u} \in \mathbb{R}^2$ and the pressure $p \in \mathbb{R}$ at each grid point. The drag coefficient (Cd) around the cylinder for the viscosity μ and pressure p_t is calculated as

$$\mathbf{F}_\mu = \oiint \mu(\nabla \mathbf{u} + \nabla \mathbf{u}^\intercal) \cdot \mathbf{n}\, dS, \tag{24}$$

$$\mathbf{F}_p = \oiint -p\mathbf{n}\, dS, \tag{25}$$

$$C_{d,\mu} = \frac{2 \cdot \mathbf{F}_\mu \cdot \mathbf{u}_\infty}{\varrho \cdot \|\mathbf{u}_\infty\|^3 \cdot D}, \tag{26}$$

$$C_{d,p} = \frac{2 \cdot \mathbf{F}_p \cdot \mathbf{u}_\infty}{\varrho \cdot \|\mathbf{u}_\infty\|^3 \cdot D}, \tag{27}$$

$$C_d = C_{d,\mu} + C_{d,p}, \tag{28}$$

where $\mathbf{u}_\infty = (1, 0)^\intercal$ is the free-stream velocity and $\mathbf{n}$ is the outward normal of the cylinder perimeter.

The state of the LED at every time-step is composed of four fields, the two components of the velocity field u_x, and u_y, the scalar pressure p at each grid-point, and the vorticity field ω computed a-posteriori from the velocity field, i.e. $\mathbf{s}_t = \{u_x, u_y, p, \omega\} \in \mathbb{R}^{4\times512\times1024}$. The simulation state $\mathbf{s}_t$ is saved at a coarse time resolution $\Delta t = 0.2$ for a total of 1000 coarse time-steps. There are 512 grid points along the length of the channel and 1024 gird points along the width of the channel. After discarding the initial transients, 250 time-steps are used for training (equivalent to $T = 50$ time units), the next 250 for validation (equivalent to $T = 50$ time units), and the next 500 for testing (equivalent to $T = 100$ time units).

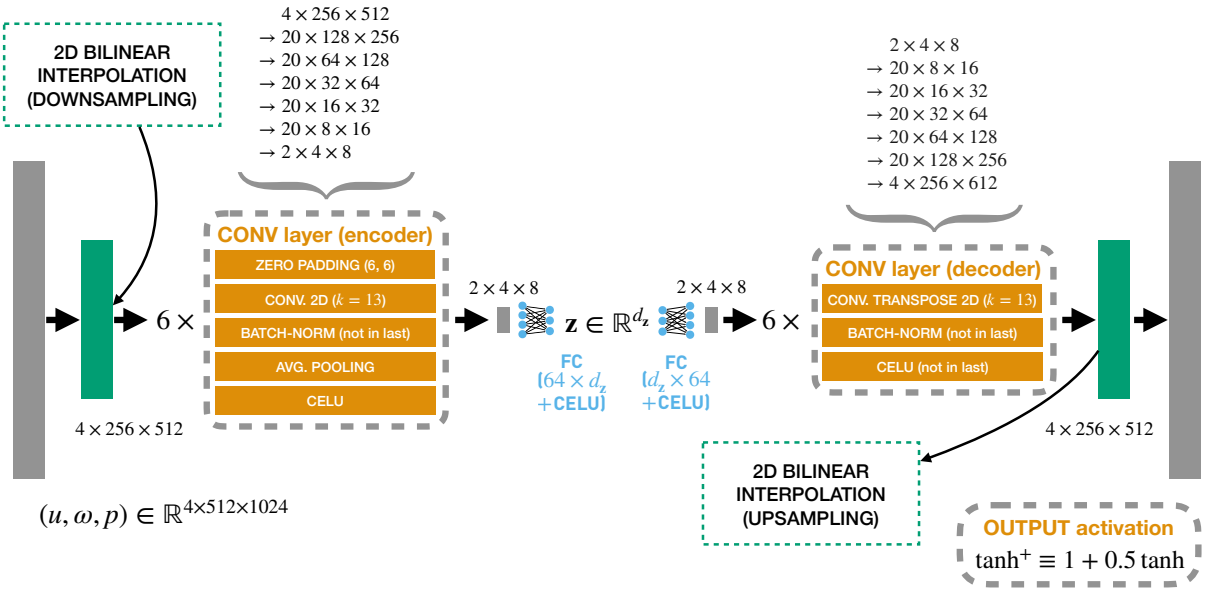


FIG. 13: The architecture of the CNN employed in the flow behind a cylinder example.

LED employs a Convolutional neural network (CNN) to identify a low dimensional latent space $\mathbf{z} \in \mathbb{R}^4$ in the $Re = 100$ scenario, and $\mathbf{z} \in \mathbb{R}^{10}$ in the $Re = 1000$ scenario. The CNN architecture is depicted in 13, and the layers are given in Table 14. We experimented with various activation functions, addition of batch-normalization layers, addition of transpose convolutional layers in the decoding part, different kernel sizes, and optimizers. The data are scaled to $[0, 1]$. The output activation function of the CNN autoencoder is set to $0.5 + 0.5 \tanh(\cdot)$, whose image range matches the data range.

The hyper-parameter tuning and training times for the LSTM-RNN of LED are given in Table 15. The hyper-parameters and training times for the RC are given in Table 16. The hyper-parameters and training times for SINDy are given in Table 17.

The hyper-parameters and training times for the RC are given in Table 16. The hyper-parameters and training times for SINDy are given in Table 17.

TABLE 14: CNN Autoencoder of LED for the flow behind a cylinder at $Re \in \{100, 1000\}$

Layer	ENCODER
(0)	interpolationLayer()
(1)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(2)	Conv2d(4, 20, kernel_size=(13, 13), stride=(1, 1))
(3)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(4)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(5)	CELU(alpha=1.0)
(6)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(7)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(8)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(9)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(10)	CELU(alpha=1.0)
(11)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(12)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(13)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(14)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(15)	CELU(alpha=1.0)
(16)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(17)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(18)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(19)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(20)	CELU(alpha=1.0)
(21)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(22)	Conv2d(20, 20, kernel_size=(13, 13), stride=(1, 1))
(23)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=True)
(24)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(25)	CELU(alpha=1.0)
(26)	ZeroPad2d(padding=(6, 6, 6, 6), value=0.0)
(27)	Conv2d(20, 2, kernel_size=(13, 13), stride=(1, 1))
(28)	AvgPool2d(kernel_size=2, stride=2, padding=0)
(29)	CELU(alpha=1.0)
(30)	Flatten(start_dim=-3, end_dim=-1)
(31)	Linear(in_features=64, out_features=$d_{\mathbf{z}}$, bias=True)
(32)	CELU(alpha=1.0)
	$\mathbf{z} \in \mathbb{R}^{d_{\mathbf{z}}}$
Layer	**DECODER**
(0)	Linear(in_features=$d_{\mathbf{z}}$, out_features=64, bias=True)
(1)	CELU(alpha=1.0)
(2)	ViewModule()
(3)	ConvTranspose2d(2, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(4)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(5)	CELU(alpha=1.0)
(6)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(7)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(8)	CELU(alpha=1.0)
(9)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(10)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(11)	CELU(alpha=1.0)
(12)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(13)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(14)	CELU(alpha=1.0)
(15)	ConvTranspose2d(20, 20, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(16)	BatchNorm2d(20, eps=1e-05, momentum=0.1, affine=0, track_running_stats=False)
(17)	CELU(alpha=1.0)
(18)	ConvTranspose2d(20, 4, kernel_size=(13, 13), stride=(2, 2), padding=(6, 6), output_padding=(1, 1))
(19)	interpolationLayer()
(20)	1 + 0.5 Tanh()
Latent dimension $d_{\mathbf{z}}$	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 16\}$

Training times [minutes]

Min	Mean	Max
1080	1081	1083

TABLE 15: LED (LSTM-RNN) hyper-parameters and training times for the flow behind a cylinder example

Hyperparameter	Values
Optimizer	Adabelief
Batch size	32
Initial learning rate	0.001
Max Epochs	1000
BPTT sequence length L	$\{10, 25\}$
Warm-up steps	10
Prediction horizon	1000
RNN Cell	LSTM
Number of RNN layers	1
Size of RNN layers	$\{32, 64\}$
Scaling	$[0, 1]$

Training times [minutes]

Min	Mean	Max
722	723	724

TABLE 16: Reservoir Computer hyper-parameters and training times (in CNN-RC) for flow behind a cylinder example

Hyper-parameter tuning	Values for $Re = 100$	Values for $Re = 1000$
Solver	Pseudoinverse	Pseudoinverse
Size	200	200
Degree	10	10
Radius	0.99	0.99
Input scaling σ	$\{0.5, 1, 2\}$	$\{0.5, 1, 2\}$
Dynamics length	100	100
Regularization η	$\{0.0, 0.001, 0.0001, 0.00001\}$	$\{0.0, 0.001, 0.0001, 0.00001\}$
Noise level per mill	$\{10\}$	$\{10\}$

Training times [minutes]

Min	Mean	Max
1.2	1.4	1.92

TABLE 17: SINDy hyper-parameters and training times (in CNN-SINDy) for flow behind a cylinder example

Hyper-parameter tuning	Values for $Re = 100$	Values for $Re = 1000$
Library	Polynomials	Polynomials
Degree	$\{1, 2, 3\}$	$\{1, 2, 3\}$
Threshold	$\{0.001, 0.0001, 0.00001\}$	$\{0.001, 0.0001, 0.00001\}$

Training times [minutes]

Min	Mean	Max
1.14	1.55	2.05

E. Alanine Dipeptide Dynamics

The efficiency of LED in capturing complex molecular dynamics is demonstrated in the dynamics of a molecule of alanine dipetide in water, a benchmark for enhanced sampling methods. The molecule is simulated with molecular dynamics with a time-step $\delta t = 1$fs, to generate data of total length 38.4ns for training, 38.4ns for validation, and 100ns for testing. The time-step of LED is set to $\Delta t = 0.1$ps. In this case, LED utilizes an MD decoder, and an MD-LSTM in the latent space, to model the stochastic, non-Markovian latent dynamics. The latent space dimension is set to $d_{\boldsymbol{z}} = 1$. As shown in Figure 14, LED identifies a meaningful one dimensional latent space, and reproduces statistics of the system. In our recent work [40], we demonstrate that LED also captures the time-scales between the meta stable states and samples realistic protein configurations while being three orders of magnitude faster than the molecular dynamics solver.

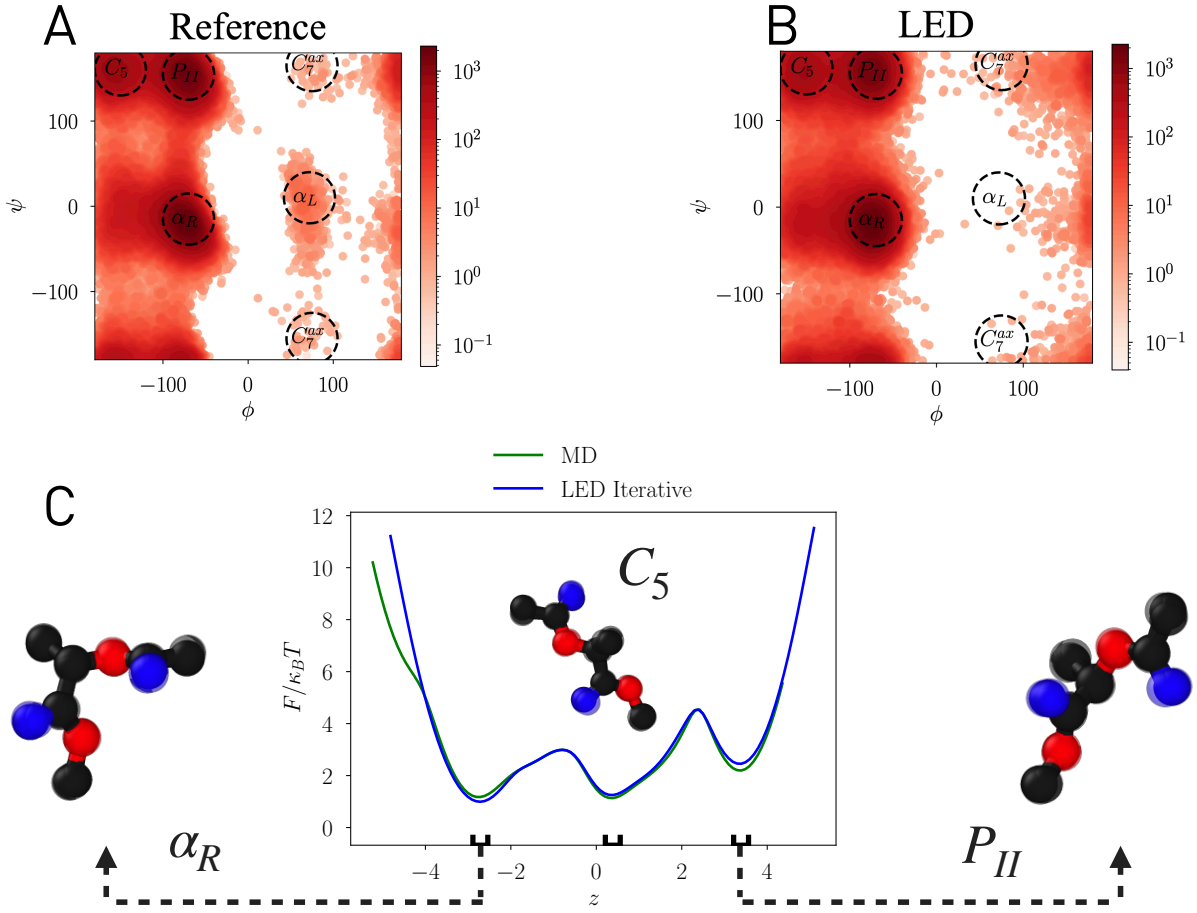


FIG. 14: **A)** Ramachadran plot of the alanine dipeptide data, i.e. state density on the space spanned by two backbone dihedral angles (ϕ, ψ). **B)** Ramachadran plot of the state evolution data predicted by LED with $T_\mu = 0$ and $d_{\boldsymbol{z}} = 1$. LED captures the three mostly visited meta-stable states $\{C_5, P_{II}, \alpha_R\}$. **C)** Projection of the state evolution data to the free energy on the one dimensional latent space unraveled by LED, i.e. $F/\kappa_B T = -\log p(z_t)$. Low energy (high probability) regions on the latent space are mapped to known metastable state configurations of alanine dipeptide.