EVOLVE SMOOTHLY, FIT CONSISTENTLY: LEARNING SMOOTH LATENT DYNAMICS FOR ADVECTION-DOMINATED SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a data-driven, space-time continuous framework to learn surrogate models for complex physical systems described by advection-dominated partial differential equations. Those systems have slow-decaying Kolmogorov n-width that hinders standard methods, including reduced order modeling, from producing high-fidelity simulations at low cost. In this work, we construct hypernetwork-based latent dynamical models directly on the parameter space of a compact representation network. We leverage the expressive power of the network and a specially designed consistency-inducing regularization to obtain latent trajectories that are both low-dimensional and smooth. These properties render our surrogate models highly efficient at inference time. We show the efficacy of our framework by learning models that generate accurate multi-step rollout predictions at much faster inference speed compared to competitors, for several challenging examples.

1 INTRODUCTION

High-fidelity numerical simulation of physical systems modeled by time-dependent partial differential equations (PDEs) has been at the center of many technological advances in the last century. However, for engineering applications such as design, control, optimization, data assimilation, and uncertainty quantification, which require repeated model evaluation over a potentially large number of parameters, or initial conditions, high-fidelity simulations remain prohibitively expensive, even with state-of-art PDE solvers. The necessity of reducing the overall cost for such downstream applications has led to the development of surrogate models, which captures the core behavior of the target system but at a fraction of the cost.

One of the most popular frameworks in the last decades (Aubry et al., 1988) to build such surrogates has been reduced order models (ROMs). In a nutshell, they construct lower-dimensional representations with their corresponding reduced dynamics that capture the system's behavior of interest. The computational gains then stem from the evolution of a lower-dimensional latent representation (see Benner et al. (2015) for a comprehensive review).

Many classical methods for ROMs require exact and complete knowledge of the underlying PDEs. However, that requirement is unrealistic in most real-world applications. For example, in weather and climate modeling, many physical processes are unknown or unresolved and thus are approximated. In other cases, the form of the PDE is unknown. Hence, being able to learn ROMs directly from data is highly desirable. In particular, modern learning techniques offer the opportunities of using "black-box" neural networks to parameterize the projections between the original space and the low-dimensional space as well as the dynamics in the low-dimensional space Sanchez-Gonzalez et al. (2020); Chen et al. (2021); Kochkov et al. (2020).

On the other end, neural parameterization can be too flexible such that they are not necessarily grounded in physics. Thus, they do not extrapolate well to unseen conditions, or in the case of modeling dynamical systems, they do not evolve very far from their initial point before the trajectories diverge in an nonphysical fashion. *How can we inject into neural models with the right types and amount of physical constraints?*

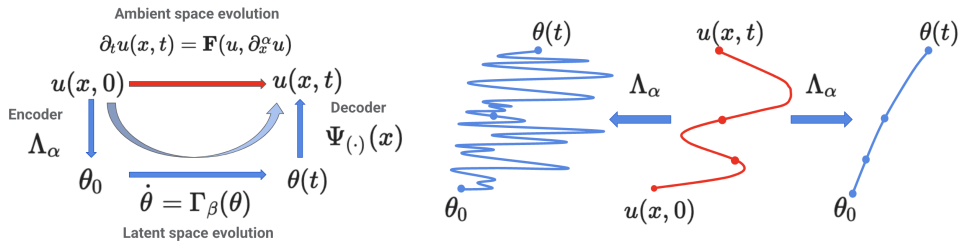

Figure 1: (left) Diagram of the latent space evolution, where u represents the state variables, and θ the latent space variables, which in this case correspond to the weights of a neural network. (right) Sketch of two possible latent space trajectories, we seek to implicitly regularize the encode-decoder to obtain trajectories as the second one, which allows us to take very long time-steps in latent space.

In this paper, we explore a hypernetwork-type surrogate model, leveraging contemporary machine learning techniques but constrained with physics-informed priors: (a) *Time-Space Continuity*: the trajectories of the model are continuous in time and space; (b) *Causality*: the present state of the model depends explicitly in the state of the model in the past; (c) *Mesh-Agnostic*: the trajectories of the model should be independent of the discretization both in space and time, so it can be sampled at any given spacial-time point (d) *Latent-Space Smoothness* if the trajectories of the model evolve smoothly the same should be true for the latent-space trajectories. The first three properties are enforced explicitly by choices of the architectures, whereas the last one is enforced implicitly at training via a consistency regularization.

Concretely, we leverage the expressive power of neural networks (Hornik et al., 1990) to represent the state of the systems through an ansatz (decoder) network that takes cues from pseudo-spectral methods (Boyd, 2001), and Neural Radiance Fields (Mildenhall et al., 2020), whose weights encodes the latent space, and are computed by an encoder-hypernetwork. This hypernetwork encoder-decoder combo is trained on trajectories of the system, which together with a consistency regularization provides smooth latent-space trajectories. The smoothness of the trajectories renders the learning of the latent-space dynamics effortless using a neural ODE (Chen et al., 2018) model whose dynamics is governed by a multi-scale network similar to a U-Net (Ronneberger et al., 2015) that extracts weight, layer and graph level features important for computing the system dynamics directly. This allows us to evolve the system *completely* in latent-space, only decoding to ambient space when required by the downstream application (such process is depicted in Fig. 1 (left)). In addition, due to the smooth nature of the latent trajectories, we can take very large time-steps when evolving the system, thus providing remarkable computational gains, particularly compared to competing methods that do not regularize the smoothness of the latent space (see Fig. 1 (right) for a cartoon).

We focus our methodology to advection-dominated problems, whose trajectories do not admit fast decaying Kolgomorov n-width (Pinkus, 2012), thus rendering them challenging for classical ROM techniques (Peherstorfer, 2020). In short, the Kolgomorov n-width is the error produced by the best approximation to a trajectory by a n-dimensional subspace, thus a slower decaying signals a richer dynamics, which are harder to accurately approximate using low-dimensional projections.

The rest of the paper is organized as follows. We briefly review existing work in solving PDEs in §2. We describe our methodology in §3, focusing on how to use consistency loss to learn smooth latent dynamics. We contrast the proposed methodology to existing methods on several advection-dominated systems in §4 and conclude in §5.

2 Related Work

The extensive literature can be roughly categorized in six main domains, starting with PDE solvers seeking to solve the underlying PDEs to purely data-driven learned surrogates, including a spectrum of different approaches in between.

Fast PDE solvers typically aim at leveraging the analytical properties of the underlying PDE to obtain low-complexity and highly parallel algorithms. Despite impressive progress (Martinsson,

2019), they are still limited by the need to mesh the ambient space, which plays an important role in the accuracy of the solution, and the stringent stability conditions for time-stepping.

Classical ROM methods typically aim at identifying low-dimensional linear approximation spaces specifically tailored to representing the snapshots of the problem at hand (in contrast with general spaces such as those in finite element or spectral methods). Approximation spaces are usually derived from data samples (Aubry et al., 1988; Chinesta et al., 2011; Barrault et al., 2004; Amsallem et al., 2012), and reduced order dynamical models are obtained by projecting the system equations onto the approximation space (Galerkin, 1915). These approaches rely on exact knowledge of the underlying PDEs and are linear in nature, although various methods have been devised to address nonlinearity in the systems (Willcox, 2006; Astrid et al., 2008; Chaturantabut & Sorensen, 2010; Geelen et al., 2022; Ayed et al., 2019).

Hybrid Physics-ML methods seek to hybridize classical numerical methods with contemporary data-driven deep learning techniques (Mishra, 2018; Bar-Sinai et al., 2019; Kochkov et al., 2021; List et al., 2022; Bruno et al., 2021; Frezat et al., 2022; Dresdner et al., 2022) These approaches seek to *learn* corrections to classical numerical schemes from high-resolution simulation data. Resulting in fast, low-resolution methods with an accuracy comparable to the ones obtained from more expensive simulations.

Operator Learning seeks to learn the differential operator directly by mimicking the analytical properties of its class, such as pseudo-differential (Hörmander, 2007) or Fourier integral operators (Hörmander, 2009), but without explicit PDE-informed components. These methods often leverage the Fourier transform (Li et al., 2021; Tran et al., 2021), the off-diagonal low-rank structure of the associated Green's function (Fan et al., 2019; Li et al., 2020a), or approximation-theoretic structures (Lu et al., 2021).

Neural Ansatz methods aim to leverage the approximation properties of neural networks (Hornik et al., 1990), by replacing the usual linear combination of handcrafted basis functions by a more general neural network ansatz. The physical prior is incorporated explicitly by enforcing the underlying PDE in strong (Raissi et al., 2019; Eivazi et al., 2021), weak (E & Yu, 2018; Gao et al., 2022), or min-max form (Zang et al., 2020). With a a few exceptions, e.g., (de Avila Belbute-Peres et al., 2021; Bruna et al., 2022), these formulation often require solving highly non-convex optimization problems at inference time.

Purely Learned Surrogates fully replace numerical schemes surrogate models purely learned from data. A number of different architectures have been explored, including multi-scale convolutional neural networks (Ronneberger et al., 2015; Wang et al., 2020), graph neural networks (Sanchez-Gonzalez et al., 2020), Encoder-Process-Decoder architectures (Stachenfeld et al., 2022), and using neural ODEs (Ayed et al., 2019).

Our current method sits between the last two categories, it inherits the advantages of both classes while avoiding their drawbacks. Our ansatz network is far more flexible than most vector-based latent-spaces used in purely ML surrogates, and once trained there is no need so solve any optimization problem at inference time, in contrast to many neural-ansatz-type methods. Also, by being mesh-agnostic our method is not subjected to mesh-induced stability constraints, which are typical in classical PDE solver and their ML-enhanced versions. Given that we seek to build surrogate models for trajectories of possibly *unknown* underlying PDE, we do not seek to replace nor compete with traditional PDE solvers.

3 METHODOLOGY

3.1 MAIN IDEA

We consider the space-time dynamics of a physical system described by PDEs of the form

$$\begin{cases} \partial_t u(x,t) &= \mathcal{F}[u(x,t)], \\ u(x,0) &= u_0. \end{cases} \tag{1}$$

where $u : \Omega \times \mathbb{R}^+ \to \mathbb{R}^d$ denotes the state of the system with respect to spatial location x and time t, $\mathcal{F}[\cdot]$ is a time-independent and potentially nonlinear differential operator dependent on u and its spatial derivatives $\{\partial_x^\alpha u\}_\alpha$, and $u_0 \in \mathcal{U}_0$ is the initial condition where $\mathcal{U}_0$ is a prescribed distribution.

Algorithm 1 Approximating $u(x, T)$

Input: initial condition: u_0**, time horizon:** T
1. Construct $\mathbf{u}_0$ following y sample or interpolating u_0 to an equi-spaced grid $\{x_i\}_{i=0}^{N-1}$
2. Compute initial latent-space condition $\theta_0 = \Lambda_\alpha(\mathbf{u}_0)$
3. Compute $\theta(T)$ by integrating $\dot{\theta} = \Gamma_\beta(\theta)$.
4. Decode the approximation $\mathbf{u}(x, T) \approx \Psi_{\theta(T)}(x)$
5. Sample the approximation to the grid $\mathbf{u}^T = \mathbf{\Psi}(\theta(T))$

Output: $\mathbf{u}^T$

For a time horizon T, we refer to $\{u(x, t),\ \text{s.t.}\ t \in [0, T]\}$ as the *rollout* from u_0, and to the partial function $u(\cdot, t^*)$ with a fixed t^* as a *snapshot* belonging to the function space $\mathcal{U} \supset \mathcal{U}_0$. We call $\mathcal{U}$ the *ambient space*.

Encoder-Decoder Architecture For any given t we approximate the corresponding snapshot of the solution u as

$$u(x, t) = \Psi_{\theta(t)}(x), \tag{2}$$

where Ψ is referred to as the *ansatz* decoder: a neural network tailored to the specific PDE to solve, so it satisfies boundary conditions and general smoothness properties, cf. Eq. (10). The *latent variables* $\theta(t) \in \Theta$ are the weights of Ψ and are time-varying quantities. Θ is referred to as the *latent space*.

To compute the *latent variables* $\theta(t)$, we use a hypernetwork encoder, which takes samples of u on a grid given by $x_i = i\Delta x$ for $i = 0, ..., N-1$, and outputs:

$$\theta(t) = \Lambda_\alpha\left(\{u(x_i, t)\}_{i=0}^{N-1}\right), \tag{3}$$

such that Eq. (2) is satisfied. Here α encompasses all the trainable parameters of the encoder.

Latent Space Dynamics We assume that θ satisfies an ODE given by

$$\begin{cases} \dot{\theta}(t) &= \Gamma_\beta(\theta(t)), \\ \theta(0) &= \Lambda_\alpha(\{u_0(x_i)\}_{i=0}^{N-1}) \end{cases} \tag{4}$$

where Γ is another hypernetwork whose trainable parameters are aggregated as β.

Algorithm 1 sketches how to solve PDEs through the latent space. For a given initial condition u_0, we uniformly sample (or interpolate unstructured samples) to a grid $\{x_i\}_{i=0}^{N-1}$ and compute its latent representation $\theta_0 = \Lambda_\alpha(\{u_0(x_i)\}_{i=0}^{N-1})$ using the encoder. We numerically solve the ODE in Eq. (4) to obtain $\theta(T)$. We then use the ansatz decoder to sample $u(x, T) \approx \Psi_{\theta(T)}(x)$, at any spatial point x. This reflects the blue computation path in Fig. 1.

To simplify the presentation, we consider only one trajectory, and assume that the sampling grid, denoted by $\mathbf{x}$, is fixed. Then $\mathbf{u}(t)$ is used to represent the vector containing the samples of $u(\cdot, t)$ on $\mathbf{x}$, or $[\mathbf{u}(t)]_i = u(x_i, t)$. Further, we conveniently denote $\mathbf{u}^n = \mathbf{u}(t^n)$. Thus, following this notation we have

$$\theta(t^n) = \Lambda_\alpha(\mathbf{u}^n), \tag{5}$$

and $\mathbf{\Psi}(\theta)$ denotes the vector approximation of u sampled on $\mathbf{x}$, i.e., $[\mathbf{\Psi}(\theta)]_i = \Psi_{\theta(t)}(x_i)$.

Consistency Regularization from Commutable Computation Our goal is to identify the structure of Ψ, Λ and Γ, together with the parameters α and β so the diagram in Fig. 1 commutes. In particular, we desire that the evolution in latent-space is equivalent to solving the PDE directly (denoted by the red path), but more computationally efficient.

Thus, a 'perfect' model of latent dynamics due to the encoder-decoder transformations should satisfy

$$\mathbf{u}(t) = \Psi(\Lambda(\mathbf{u}(t))), \qquad \dot{\mathbf{u}} = \nabla_\theta \mathbf{\Psi}(\theta)\dot{\theta}, \qquad \text{and} \qquad \dot{\theta} = \nabla_u \Lambda(\mathbf{u})\dot{\mathbf{u}}. \tag{6}$$

However, imposing the conditions entails the complete knowledge of the time derivative in the ambient space, which may not be feasible. To build the encoder/decoder independently of the time derivatives, note that we can enforce the following constraints by combining the equations above:

$$\dot{\mathbf{u}} = \nabla_\theta \mathbf{\Psi}(\theta)\nabla_u \Lambda(\mathbf{u})\dot{\mathbf{u}} \qquad \text{and} \qquad \dot{\theta} = \nabla_u \Lambda(\mathbf{u})\nabla_\theta \mathbf{\Psi}(\theta)\dot{\theta}. \tag{7}$$

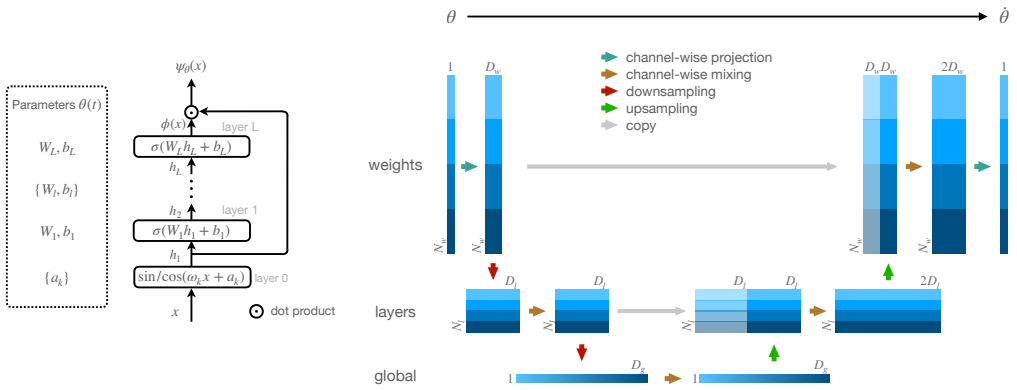

Figure 2: The Nonlinear Fourier Ansatz (left) featurizes the input with $\sin / \cos$ functions with specified frequency and trainable phases, and integrates a small MLP to parametrize functions which are periodic in $[0, L]$. The Hyper U-Net (right) architecture partitions weights according to their layers (shown in different shades) and extract/aggregate features similarly to the original U-Net. More details on the transforms represented by arrows in the Hyper U-Net are included in Appendix A.3.

These expressions are the time derivatives of

$$\mathbf{u}(t) = \Psi(\Lambda(\mathbf{u}(t))) \to \mathcal{L}_{\text{reconstruct}}(\mathbf{u}) = \|\mathbf{u}(t) - \Psi(\Lambda(\mathbf{u}(t)))\|^2 \tag{8}$$

$$\theta(t) = \Lambda(\Psi(\theta(t))) \to \mathcal{L}_{\text{consistency}}(\theta) = \|\theta(t) - \Lambda(\Psi(\theta(t)))\|^2 \tag{9}$$

where the first equation leads to the typical reconstruction loss for autoencoders. The second equation, on the other end, leads to the *consistency regularization* by imposing an additional constraint that $\theta = \Lambda(\Psi(\theta))$ whenever possible.

This symmetric pair of loss functions is intuitive from Fig. 1. To enable transforming back and forth between ambient and latent spaces with little corruption, re-encoding $\mathbf{u}(t)$ gives back $\theta(t)$ is a must if we would take either red, blue or a mixed path to rollout the trajectory from any point in time.

3.2 Architectural Choices

We provide a succinct overview of the architectures used for the ansatz, encoder and dynamics.

The Nonlinear Fourier Ansatz (NFA) As an example, in this work we consider an ansatz applicable to a wide range of advection-dominated systems:

$$\Psi_\theta(x) = \sum_{k=1}^{K} \phi_k(x) \sin(\omega_k x + a_k) + \phi_{-k}(x) \cos(\omega_k x + a_{-k}), \tag{10}$$

where $\{\omega_k\}$ follow either a linear or a dyadic profile (see Appendix A.3). This ansatz leverages low-frequency Fourier modes as an envelop, corrected by a shallow neural network in their amplitude and phase (the architecture is illustrated in Fig. 2). The ansatz is periodic in $[0, L]$ by construction, and can be further tailored to specific applications by adjusting the number of Fourier modes and their frequencies. In this case θ aggregates the phases a_k and the parameters inside MLP for ϕ_k.

Encoder Since we assume that the input data of the encoder lies in an one dimensional equispaced grid, and that the boundary conditions are periodic, we use a simple convolutional encoder with periodic padding. This is achieved by a sequence of convolutional ResNet modules with batch normalization and a sinusoidal activation function, in which the input is progressively downsampled in space. We consider either four or five levels of downsampling. Each level consists of two consecutive ResNet modules and then we downsample by two the spatial component in the layer while doubling the number of channels, thus the roughly same amount of information is carried throughout the layers. A fully connected layer at the output generates the parameters of the ansatz network.

Hyper U-Net for Learning Dynamics Since the input and output of function Γ both follow the weight structure of the same neural network Ψ, we propose a novel neural network architecture, which we refer to as the Hyper U-Net, to model the mapping between them (illustrated in Fig. 2). Analogous to the classical U-Net (Ronneberger et al., 2015), which has 'levels' based on different resolutions of a image, the Hyper U-Net operates on 'levels' based on different resolutions of the computational graph of the ansatz network. In this paper, we use three levels which are natural to modern neural networks, namely individual weights, layer and graph levels, listed in decreasing order of resolution. At each level, features are derived from the finer level in the downsampling leg of the U-Net, and later on mixed with the ones derived from coarser level in the upsampling leg. Differently from the U-Net, instead of convolutions we use locally-connected layers that do not share weights.

3.3 Multi-stage Learning Protocol

Joint training of encoder, decoder and the dynamics U-Net is difficult, due to the covariate shift among these components (ablation study in Appendix A.5). To overcome this, we develop an effective multi-stage learning procedure.

Learning to Encoder Snapshots For a given architecture of an ansatz network, we learn an encoder Λ_α that maps snapshots to the corresponding latent spaces, by optimizing the following loss

$$\mathcal{L}_{\text{enc}}(\alpha) = \sum_n \left[\mathcal{L}_{\text{reconstruct}}(\mathbf{u}^n) + \gamma \mathcal{L}_{\text{consistency}}(\theta^n)\right], \tag{11}$$

where $\theta^n = \Lambda_\alpha(\mathbf{u}^n)$. The relative strength between the two terms in the loss function, is controlled by the scalar hyperparameter γ. Empirically, the second consistency loss injects inductive bias of preferring smoother and better-conditioned trajectories, for the dynamic models to learn (see results section).

Learning to Evolve the Latent Representation After the encoder is trained, we generate training latent space trajectories by computing $\theta^n = \Lambda_\alpha(\mathbf{u}^n)$. We then learn the latent dynamics operator Γ_β from sample latent rollouts $\mathcal{D}_\theta = \{(t^0, \theta^0), (t^1, \theta^1), ...\}$ via the following loss function:

$$\mathcal{L}_{\text{ODE}}(\beta) = \sum_n^N \left\|\theta^n - \tilde{\theta}^n\right\|^2, \quad \tilde{\theta}^n = \theta^0 + \int_{t^0}^{t^n} \Gamma_\beta(\theta(t))\, dt, \tag{12}$$

where we use an explicit fourth-order Runge-Kutta scheme to compute the time integration. As our encoder identifies latent spaces with good smoothness conditions, we have the luxury to use relatively large, fixed step sizes (vs. adaptive size), which serve as an additional regularization that biases towards learning models with efficient inference. Learning Γ_β directly using this latent-space rollouts is empirically challenging, due to a highly non-convex energy landscape and gradient explosions. Instead, we split the training process in two stages: single-step pre-training, and fine-tuning using checkpointed neural ODE adjoints (Chen et al., 2018). We describe them briefly in below:

Single-step pretraining We train the latent dynamical model by first optimizing a single-step loss:

$$\mathcal{L}_{\text{single}}(\beta) = \sum_n^{N-1} \left\|\frac{\theta^{n+1} - \theta^n}{\Delta t} - \Gamma_\beta(\theta^n)\right\|^2, \tag{13}$$

which simply attempts to match Γ_β with the first order finite difference approximation of the time derivative in θ. Note that cumulative errors beyond the Δt window do not affect the learning of β.

Multi-step finetuning Although learning with single-step loss defined by Eq. (13) alone has been demonstrated to achieve success in similar contexts when optimized to high precision and combined with practical techniques such as noise injection (Pfaff et al., 2020; Han et al., 2022), we empirically observe that additional fine-tuning β using Eq. (12) as the loss function helps a great ideal to improve the accuracy and stability of our *inference rollouts*, i.e. it is important to teach the model to "look a few more steps head" during training. The fact that our encoders generate smooth latent spaces, together with the use of fixed, large-step time integration, helps to reduce the cost of the otherwise expensive adjoint gradient compute, which had been the primary drawback of such long-range neural-ODE training. We point out that there exists ample literature on multi-step training of differential equations,

e.g. Keisler (2022); Brandstetter et al. (2022). For problems considered in this work, however, this simple two-stage training has proved empirically effective.

To conclude this section, we mention in passing that the recently proposed Neural Galerkin approach (Bruna et al., 2022) can be used to learn the dynamics by fitting $\dot{\theta}$:

$$\partial_t u(x, \cdot) = \partial_t \Psi_{\theta(\cdot)}(x) = \nabla_\theta \Psi(x)\dot{\theta} = \mathcal{F}[u](x). \tag{14}$$

However, as will be demonstrated in our experiments, this approach is at times prone to bad conditioning in the stiffness matrix, and relies on having complete knowledge about $\mathcal{F}$.

4 Results

4.1 Setup

Datasets We demonstrate the efficacy of the proposed approach on three exemplary PDE systems: (1) Viscid Burgers (VB) (2) Kuramoto-Sivashinsky (KS) and (3) Kortweg-De Vries (KdV). These systems represent a diverse mix of dynamical behaviors resulting from advection, characterized by the presence of dissipative shocks, chaos and traveling waves respectively. Each is simulated on a periodic spatial domain using an Explicit-Implicit solver in time coupled with a pseudo-spectral discretization in space. We used the spectral code in `jax-cfd` (Dresdner et al., 2022) to compute the datasets. This solver provides very accurate and dispersion free ground-truth references. Detailed descriptions of the systems and the generated datasets can be found in Appendix A.1.

Benchmark Methods We compare to two recently proposed state-of-the-art methods for solving PDEs continuously in space and time: Neural Galerkin (NG) (Bruna et al., 2022) (at high and low sampling rates), DeepONet (Lu et al., 2021) and Fourier Neural Operator (FNO) (Li et al., 2020b). We also investigate variants of our model that uses the same encoder architectures but a convolution-based decoder and comparable latent space dimensions (abbreviated NDV - neural decoder variant). Details of those methods are in Appendix A.4.

Metrics For each system, we use a training set of 1000 trajectories. For evaluation, trained models are then used to generate multi-step rollouts on 100 unseen initial conditions. The rollouts are compared to the ground truths, which are obtained by directly running a high-resolution pseudo-spectral solver on the evaluation initial conditions. To quantitatively assess the predictive accuracy of the rollouts, we compute the point-wise relative root mean squared error (relRMSE)

$$\text{relRMSE}(\mathbf{u}^{\text{pred}}, \mathbf{u}^{\text{true}}) = \frac{\|\mathbf{u}^{\text{pred}} - \mathbf{u}^{\text{true}}\|}{\|\mathbf{u}^{\text{true}}\|} \tag{15}$$

as a measure for the short-term prediction quality. We also compare the spatial energy spectra (norm of the Fourier transform) of the predictions to that of the ground truths:

$$E(f_k = k/N) = \frac{1}{M} \sum_{m=1}^{M} \left| \sum_{n=0}^{N-1} u_{mn} \exp\left(-j2\pi nk/N\right) \right|, \tag{16}$$

where the average is taken for predicted snapshots u_m at a fixed long time away from the initial condition across the evaluation rollouts. This metric provides a translation-invariant assessment of the frequency features of the long-term predictions (evaluates well for having the correct features at imperfect locations). Lastly, we record the inference cost of each method in terms of the wall clock time (WCT) taken and the number of function evaluations (NFE) per simulation time unit.

4.2 Main Results and Ablation Studies

Accurate and stable long-term roll-outs Fig. 4 contrasts our method to the benchmarks. While each benchmarked method fails to produce a working surrogate for at least one of the systems (defined and summarized in Table 1), our method is able to model all three systems successfully.

Qualitatively, shocks in VB, chaos patterns in KS, and traveling waves in KdV are all present in our model rollouts in ways that are visually consistent with the ground truths. Our models produce the

Table 1: Whether the benchmark methods are able to produce a working inference model for each system. Here 'working' is defined as exceeding 100% relRMSE in 40 rollout steps (equal to twice the training range for our model).

	VB	KS	KdV
Ours-NFA	✓	✓	✓
Ours-NDV		✓	✓
NG-Lo	✓		
NG-Hi	✓	✓	
DeepONet	✓		
FNO	✓	✓	

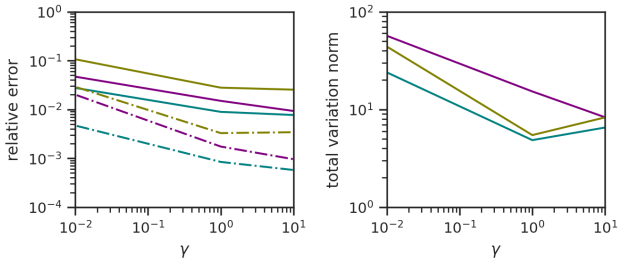

Figure 3: Relative reconstruction error (left, solid), consistency error (left, dashed) and total variation norm (right) vs. γ at different learning rates for KS: 3×10^{-4} —; 1×10^{-4} —; 3×10^{-5}; —. See metric definitions in Appendix A.5.

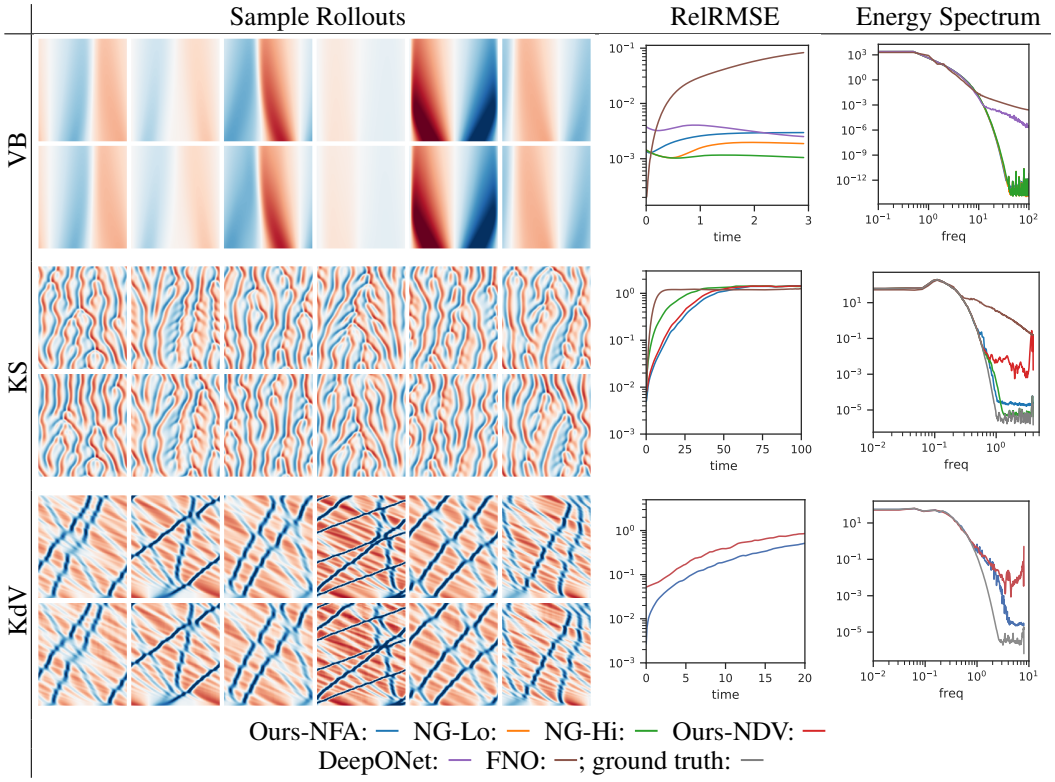

Figure 4: Sample long-term rollouts (left), point-wise error vs. time and long-range energy spectrum (right) plots. Rollouts are plotted in (x, t) plane with upper rows showing the ground truths and lower rows our corresponding predictions. Columns correspond to different initial conditions. Ranges for energy spectra plotted: VB - 3; KS - 80; KdV - 16. NG-Lo/NG-Hi are NG runs with a sampling rate of 198 and 600 respectively.

lowest point-wise errors among working benchmarks for both KS and KdV (which are the more complex examples) and come very close to the best-performing NG model for VB. In the long-range prediction regime, our models are able to remain stable for the longest period of time and exhibit statistics which perfectly match the truth in low frequencies and are the best approximating one in high frequency components.

The importance of ansatzes The right choice of the ansatz plays two roles. Firstly, learning the encoder becomes easier when the key structures are already embedded in the decoder - this is

Table 2: Comparison of inference speeds in terms of WCT and NFE statistics. WCTs are measured in microseconds. NFEs are measured by an adaptive-step integrator. Numbers are normalized to cost per unit simulation time. Solver is run at the same resolution used to generate training data. High NFE count of NG-Hi results from running into a stiff numerical regime before breaking down (around 120 rollout steps).

	VB				KS				KdV			
	WCT		NFE		WCT		NFE		WCT		NFE	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
Solver	8.4	0.5	-	-	5.2	0.3	-	-	1041	13.1	-	-
Ours-NFA	3.8	0.9	16.0	4.1	1.7	2.2	5.3	0.2	52.1	18.4	164.7	40.3
Ours-NDV	-	-	-	-	1.1	0.6	7.1	2.7	18.0	5.4	71.4	19.5
NG-Lo	30.2	8.5	27.5	10.8	167.9	21.4	114.7	13.3	-	-	-	-
NG-Hi	49.6	7.7	27.5	10.8	319.4	33.5	111.7	11.7	35670	29035	11651	9502
FNO	18.1	14.5	100.0	0.0	0.4	0.4	5.0	0.0	-	-	-	-

supported by the observation that our model easily learns to encode the shocks in VB while NDV fails to do so even allowed to operate a larger latent space. Secondly, the ansatz reduces the spurious high-frequency energy components in the state, which is helpful in improving the stability of rollouts, as these energies are likely to accumulate in a nonlinear fashion. This is the main reason why our model is able to have better errors than the closely related NDV model in KS and KdV.

Consistency regularization leads to smoothness The impact of γ in Eq. (11) is evident from the observation that (Fig. 3 left) as the strength of regularization increases the encoders resulting from minimizing the loss in Eq. (11) have lower reconstruction error. It also induces smoother latent weights trajectories, as measured in the component-wise total variation norm (Fig. 3 right). The smoothness is beneficial for downstream training to learn efficient dynamical models that permit taking large integration time steps.

Inference efficiency Statistics on the inference cost of each method are summarized in Table 2. Our models have the best efficiencies amongst the compared methods. Their low NFE counts confirm that generous time steps are being taken during time integration. Note that the aforementioned trajectory smoothness is not a sufficient condition to guarantee such efficiency, supported by the observation that the PDE-induced dynamics found by NG typically require more NFEs per unit time (can even run into prohibitively stiff regimes, like in KdV) despite starting from the same encoded initial conditions. The explicit large-step integration that we use to learn the latent dynamical models proves to be an effective regularization to alleviate this issue - the NDV models are also able to achieve efficient inference benefiting from the same factor. However, without the help of consistency regularization, this comes at a compromise in accuracy and robustness compared to our main models.

5 Conclusion

In this work we have introduced a method to learn reduced-order surrogate models for advection-dominated systems. We demonstrate that through specially tailored ansatz representations and effective regularization tools, our method is able to identify weight spaces with smooth trajectories and approximate the dynamics accurately using hyper networks. Experiments performed on challenging examples show that our proposed method achieves top performance in both accuracy and speed.

References

David Amsallem, Matthew J Zahr, and Charbel Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10): 891–916, 2012.

Patricia Astrid, Siep Weiland, Karen Willcox, and Ton Backx. Missing point estimation in models described by proper orthogonal decomposition. *IEEE Transactions on Automatic Control*, 53(10): 2237–2251, 2008. doi: 10.1109/TAC.2008.2006102.

Nadine Aubry, Philip Holmes, John L. Lumley, and Emily Stone. The dynamics of coherent structures in the wall region of a turbulent boundary layer. *Journal of Fluid Mechanics*, 192:115–173, 1988. doi: 10.1017/S0022112088001818.

Ibrahim Ayed, Emmanuel de Bezenac, Arthur Pajot, Julien Brajard, and Patrick Gallinari. Learning dynamical systems from partial observations, 2019.

Yohai Bar-Sinai, Stephan Hoyer, Jason Hickey, and Michael P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, July 2019. ISSN 0027-8424, 1091-6490. doi: 10.1073/pnas.1814058116. URL https://pnas.org/doi/full/10.1073/pnas.1814058116.

Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004. ISSN 1631-073X. doi: https://doi.org/10.1016/j.crma.2004.08.006. URL https://www.sciencedirect.com/science/article/pii/S1631073X04004248.

Peter Benner, Serkan Gugercin, and Karen Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Review*, 57(4):483–531, 2015. doi: 10.1137/130932715. URL https://doi.org/10.1137/130932715.

John P. Boyd. *Chebyshev and Fourier Spectral Methods*. Dover Books on Mathematics. Dover Publications, Mineola, NY, second edition, 2001. ISBN 0486411834 9780486411835.

Johannes Brandstetter, Daniel Worrall, and Max Welling. Message passing neural pde solvers, 2022. URL https://arxiv.org/abs/2202.03376.

Joan Bruna, Benjamin Peherstorfer, and Eric Vanden-Eijnden. Neural galerkin scheme with active learning for high-dimensional evolution equations. *arXiv:2203.01360 [math.NA]*, May 2022. URL https://arxiv.org/abs/2203.01360. arXiv: 2203.01360.

Oscar P. Bruno, Jan S. Hesthaven, and Daniel V. Leibovici. FC-based shock-dynamics solver with neural-network localized artificial-viscosity assignment. *arXiv:2111.01315 [cs, math]*, November 2021. arXiv: 2111.01315.

Claudio G Canuto, M. Yousuff Hussaini, Alfio M. Quarteroni, and Thomas A. Zang. *Spectral Methods: Evolution to Complex Geometries and Applications to Fluid Dynamics (Scientific Computation)*. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 3540307273.

Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010. doi: 10.1137/090766498. URL https://doi.org/10.1137/090766498.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/69386f6bb1dfed68692a24c8686939b9-Paper.pdf.

Wenqian Chen, Qian Wang, Jan S. Hesthaven, and Chuhua Zhang. Physics-informed machine learning for reduced-order modeling of nonlinear problems. *Journal of Computational Physics*, 446:110666, 2021. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2021.110666. URL https://www.sciencedirect.com/science/article/pii/S0021999121005611.

Francisco Chinesta, Pierre Ladeveze, and Elias Cueto. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering*, 18(4):395–404, 2011.

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19(90):297–301, 1965. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/2003354.

Filipe de Avila Belbute-Peres, Yi fan Chen, and Fei Sha. HyperPINN: Learning parameterized differential equations with physics-informed hypernetworks. In *The Symbiosis of Deep Learning and Differential Equations*, 2021.

Gideon Dresdner, Dmitrii Kochkov, Peter Norgaard, Leonardo Zepeda-Núñez, Jamie A. Smith, Michael P. Brenner, and Stephan Hoyer. Learning to correct spectral methods for simulating turbulent flows, 2022.

Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018. doi: 10.1007/s40304-018-0127-z. URL https://doi.org/10.1007/s40304-018-0127-z.

Hamidreza Eivazi, Mojtaba Tahani, Philipp Schlatter, and Ricardo Vinuesa. Physics-informed neural networks for solving Reynolds-averaged Navier-Stokes equations. *arXiv:2107.10711 [physics]*, July 2021. URL http://arxiv.org/abs/2107.10711. arXiv: 2107.10711.

Yuwei Fan, Jordi Feliu-Fabà, Lin Lin, Lexing Ying, and Leonardo Zepeda-Núñez. A multiscale neural network based on hierarchical nested bases. *Research in the Mathematical Sciences*, 6, Mar. 2019. ISSN 2197-9847. doi: 10.1007/s40687-019-0183-3.

Hugo Frezat, Julien Le Sommer, Ronan Fablet, Guillaume Balarac, and Redouane Lguensat. A posteriori learning for quasi-geostrophic turbulence parametrization. *arXiv*, April 2022.

B. G. Galerkin. Series occurring in various questions concerning the elastic equilibrium of rods and plates. *Vestnik Inzhenernov i Tekhnikov*, 19:897–908, 1915.

Han Gao, Matthew J. Zahr, and Jian-Xun Wang. Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.114502.

Rudy Geelen, Stephen Wright, and Karen Willcox. Operator inference for non-intrusive model reduction with nonlinear manifolds. *arXiv:2205.02304 [math.NA]*, May 2022. URL https://arxiv.org/abs/2205.02304. arXiv: 2205.02304.

Xu Han, Han Gao, Tobias Pfaff, Jian-Xun Wang, and Li-Ping Liu. Predicting physics in mesh-reduced space with temporal attention. *CoRR*, abs/2201.09113, 2022. URL https://arxiv.org/abs/2201.09113.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Lars Hörmander. *The Analysis of Linear Partial Differential Operators. III: Pseudo-Differential Operators*, volume 63 of *Classics in Mathematics*. Springer, Berlin, 2007.

Lars Hörmander. *The Analysis of Linear Partial Differential Operators. IV: Fourier Integral Operators*, volume 63 of *Classics in Mathematics*. Springer, Berlin, 2009.

K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

Ryan Keisler. Forecasting global weather with graph neural networks. *arXiv preprint arXiv:2202.07575*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Dmitrii Kochkov, Alvaro Sanchez-Gonzalez, Jamie Alexander Smith, Tobias Joachim Pfaff, Peter Battaglia, and Michael Brenner. Learning latent field dynamics of pdes. In *Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020)*, 2020.

Dmitrii Kochkov, Jamie A Smith, Ayya Alieva, Qing Wang, Michael P Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proc. Natl. Acad. Sci. U. S. A.*, 118 (21), May 2021. URL https://www.pnas.org/content/118/21/e2101784118.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Multipole graph neural operator for parametric partial differential equations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020a. Curran Associates Inc. ISBN 9781713829546.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020b.

Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier Neural Operator for Parametric Partial Differential Equations. *arXiv:2010.08895 [cs, math]*, May 2021. URL http://arxiv.org/abs/2010.08895. arXiv: 2010.08895.

Björn List, Li-Wei Chen, and Nils Thuerey. Learned Turbulence Modelling with Differentiable Fluid Solvers. *arXiv:2202.06988 [physics]*, February 2022. URL http://arxiv.org/abs/2202.06988. arXiv: 2202.06988.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

Per-Gunnar Martinsson. *Fast Direct Solvers for Elliptic PDEs*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2019. doi: 10.1137/1.9781611976045. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611976045.

Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

Siddhartha Mishra. A machine learning framework for data driven acceleration of computations of differential equations. *Mathematics in Engineering*, 1(1):118–146, 2018. ISSN 2640-3501. doi: 10.3934/Mine.2018.1.118. URL https://www.aimspress.com/article/doi/10.3934/Mine.2018.1.118.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 165–174, 2019.

Benjamin Peherstorfer. Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling. *SIAM Journal on Scientific Computing*, 42(5):A2803–A2836, 2020. doi: 10.1137/19M1257275. URL https://doi.org/10.1137/19M1257275.

Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

A. Pinkus. *Ergebnisse der Mathematik und ihrer Grenzgebiete. 3. Folge / A Series of Modern Surveys in Mathematics*, volume 1. pringer Berlin Heidelberg,, 2012.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, February 2019. ISSN 0021-9991. doi: 10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Hkuq2EkPf.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi (eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pp. 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W. Battaglia. Learning to Simulate Complex Physics with Graph Networks. *arXiv:2002.09405 [physics, stat]*, September 2020. URL http://arxiv.org/abs/2002.09405. arXiv: 2002.09405.

Kimberly Stachenfeld, Drummond B. Fielding, Dmitrii Kochkov, Miles Cranmer, Tobias Pfaff, Jonathan Godwin, Can Cui, Shirley Ho, Peter Battaglia, and Alvaro Sanchez-Gonzalez. Learned Coarse Models for Efficient Turbulence Simulation. *arXiv:2112.15275 [physics]*, January 2022. URL http://arxiv.org/abs/2112.15275. arXiv: 2112.15275.

Alasdair Tran, Alexander Mathews, Lexing Xie, and Cheng Soon Ong. Factorized Fourier Neural Operators. *arXiv:2111.13802 [cs]*, November 2021. arXiv: 2111.13802.

Lloyd N. Trefethen. *Spectral Methods in MATLAB*. Society for industrial and applied mathematics (SIAM), 2000.

Rui Wang, Karthik Kashinath, Mustafa Mustafa, Adrian Albert, and Rose Yu. Towards physics-informed deep learning for turbulent flow prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. URL http://dx.doi.org/10.1145/3394486.3403198.

Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science Advances*, 7(40):eabi8605, 2021. doi: 10.1126/sciadv.abi8605. URL https://www.science.org/doi/abs/10.1126/sciadv.abi8605.

K. Willcox. Unsteady flow sensing and estimation via the gappy proper orthogonal decomposition. *Computers & Fluids*, 35(2):208–226, 2006. ISSN 0045-7930. doi: https://doi.org/10.1016/j.compfluid.2004.11.006. URL https://www.sciencedirect.com/science/article/pii/S0045793005000113.

Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411:109409, 2020. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2020.109409. URL https://www.sciencedirect.com/science/article/pii/S0021999120301832.

A APPENDIX

A.1 DATASETS

In this paper we consider PDEs of the form

$$\partial_t \boldsymbol{u} = \mathbf{F}(\boldsymbol{u}) := D\boldsymbol{u} + N(\boldsymbol{u}), \tag{17}$$

plus initial and boundary conditions. Here D is a linear partial differential operator, and N is a non-linear term. For the models considered in this paper, the differential operator is typically either diffusive, $D = \partial_x^2$, dispersive, $D = \partial_x^3$, or hyper-diffusive, $D = \partial_x^4$, ; and the non-linearity is a convective term, $N = \frac{1}{2}\partial_x(\boldsymbol{u}^2) = \boldsymbol{u}\partial_x\boldsymbol{u}$. In a nutshell, diffusion tends to blur the solution, rendering more homogeneous, dispersion produced propagating waves whose velocity depends on the Fourier content of the solution, and advection, also propagate waves whose velocity depends on the local value of the solution.

In practice, PDEs are solved by discretizing in space and time, which converts the continuous PDE into a set of update rules for vectors of coefficients to approximate the state $\boldsymbol{u}$ in some discrete basis, e.g., on a grid. For time-dependent PDEs, temporal resolution must be scaled proportionally to spatial resolution to retain an accurate solution, so runtime for PDE solvers scales at best like $O(n^{d+1})$, where d is the number of spatial dimensions and n is the number of discretization points along any dimension, in general, however, dispersion errors tends to dominate the pre-asymptotic regime so one would need a large amount of point in space, and therefore a large time of time steps.

To circumvent this issue, we choose a pseudo-spectral discretization, which is known to be dispersion free, due to the *exact* evaluation of the derivatives in Fourier space, and it possess excellent approximation guarantees (Trefethen, 2000) thus, relatively few discretization points are needed to represent solutions that are smooth.

We assume that, after a rescaling in space, $\boldsymbol{u}(x,t) : [0, 2\pi]\times\mathbb{R}^+ \to \mathbb{R}$ is one-dimensional, 2π-periodic and square-integrable for all times t. Consider the Fourier coefficients $\widehat{\boldsymbol{u}}^t$ of $\boldsymbol{u}(x,t)$, truncated to lowest $K+1$ frequencies (for even K):

$$\widehat{\boldsymbol{u}}^t = (\widehat{\boldsymbol{u}}^t_{-K/2}, \ldots, \widehat{\boldsymbol{u}}^t_k, \ldots, \widehat{\boldsymbol{u}}^t_{K/2}) \quad \text{where} \quad \widehat{\boldsymbol{u}}^t_k = \frac{1}{2\pi}\int_0^{2\pi} \boldsymbol{u}(x,t)e^{-ik\cdot x}dx. \tag{18}$$

$\widehat{\boldsymbol{u}}^t$ is a vector representing the solution at time t, containing the coefficients of the Fourier basis e^{ikx} for $k \in \{-K/2, \ldots, K/2\}$. Where the integral $\widehat{\boldsymbol{u}}^t_k$ is approximated using a trapezoidal quadrature on $K+1$ points. I.e., we approximate it by sampling $\boldsymbol{u}(x,t)$ on an equispaced grid, and then we use the Fast Fourier Transform (FFT) (Cooley & Tukey, 1965) to compute these Fourier coefficients efficiently in log-linear time. Differentiation in the Fourier domain is a diagonal operator, in fact, it can be calculated by element-wise multiplication according to the identity $\partial_x\widehat{\boldsymbol{u}}_k = ik\widehat{\boldsymbol{u}}_k$. This makes applying and inverting linear differential operators trivial since it is simply element-wise operation (Trefethen, 2000).

For non-linear terms can be applied via an expensive convolution, or by using Plancherel's theorem to pivot between real and Fourier space to evaluate these terms in quasi-linear time.

In a nutshell this procedure lead to a system in Fourier domain of the form

$$\partial_t\widehat{\boldsymbol{u}}^t = \mathbf{D}\widehat{\boldsymbol{u}}^t + \mathbf{N}(\widehat{\boldsymbol{u}}^t) \tag{19}$$

where $\mathbf{D}$ encodes the D operator in Fourier domains, which is often diagonal matrix, whose entries only depend on the wavenumber k, and $\mathbf{N}$ encodes the non-linear part. This system be solved using an implicit-explicit Crack-Nicolson Runge-Kutta method of fourth order (Canuto et al., 2007). In this case, we treat the linear part implicitely and the non-linear one explicitly.

We point out that this transformation still carries some of the difficulties of the original problem, in which as k increases the conditioning number of $\mathbf{A}$ will also increase, which may require to refine the time-step in order to avoid instabilities and to satisfy the Shannon-Nyquist sampling criterion.

The code to generate the data was implemented using `jax-cfd`, where we used 512 grid points and a time step $dt = 2 \cdot 10^{-3}$. The code will released once the paper is accepted.

Viscid Burgers Equation We solve the equation given by

$$\partial_t u + u\partial_x u - \nu\partial_{xx} u = 0, \qquad \text{in } [-1,1]\times\mathbb{R}^+, \tag{20}$$

with periodic boundary conditions. We followed the set up in (Wang et al., 2021), in which $\nu = 0.01$, and the initial condition is chosen following a Gaussian process (the details can be found in the github repository associated with (Wang et al., 2021)).

We point out that the combination of the high-viscosity and the smoothness of the initial conditions results in trajectories that are fairly stable with waves travelling only small distances, and no sub-grid shocks being formed.

Kuramoto-Sivashinsky Equation We solve the equation given by

$$\partial_t u + u\partial_x u + \nu\partial_{xx} u - \nu\partial_{xxxx} u = 0, \qquad \text{in } [-L/2, L/2]\times\mathbb{R}^+, \tag{21}$$

with periodic boundary conditions, and $L = 64$. Here the domain was rescaled following in order to balance the diffusion and anti-diffusion components so the solutions are chaotic(Dresdner et al., 2022).

The initial conditions were given by

$$u_0(x) = \sum_{k=1}^{3}\sum_{j=1}^{n_c} a_j \sin(\omega_j * x + \phi_j) \tag{22}$$

where ω_j is chosen randomly from $[\pi/L, 2\pi/L, 3\pi/L]$, a_j is chosen randomly following an uniform distribution from $[-0.5, 0.5]$, and phase ϕ_j is also chosen randomly following an uniform distribution from $[0, 2\pi]$, in this case we have $n_c = 30$.

KdV Equation We solve the equation

$$\partial_t u - 6u\partial_x u + \partial_{xxx} u = 0, \qquad \text{in } [-L/2, L/2]\times\mathbb{R}^+, \tag{23}$$

with periodic boundary conditions, and $L = 32$. Here the domain was also rescaled to obtain travelling waves. We point out that this equation is highly dispersive, so an spectral methods is required to solve this equation accurately.

The initial conditions for the KdV equation were the same as in Eq. (22) but with $L = 32$ and $n_c = 10$. This choice of initial conditions allows for non-soliton solutions, with few large wave packages being propagated, and several small dispersive waves in the background.

In addition, to our knowledge this is first time that a ML method is able to compute a non-soliton solution of the KdV equation for long times.

A.2 Additional Training and Evaluation Details

For all training stages, we use the Adam optimizer (Kingma & Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

Ansatz evaluation Prior to carrying out the multi-stage learning protocol described in the main text, we first perform a preliminary evaluation of the ansatz network via batch fitting $\{\theta^n\}$ to a random selection of snapshots $\{u_i\}$ drawn from the dataset, i.e. for each i we minimize the representation loss

$$\mathcal{L}_{\text{rep}}\big(\{\theta^n\}\big) = \frac{1}{N}\sum_{n}^{N} \|\mathbf{u}^n - \mathbf{\Psi}(\theta^n)\|^2, \tag{24}$$

which provides us with an assessment of the expressive power of the particular functional form of Ψ with respect to the problem at hand. Note that this process, otherwise known as *auto-decoder training* (Park et al., 2019), can be done independently and we can easily compare amongst multiple potential candidates and rule out ones which are not suitable. In particular, as a rule of thumb, we exclude ansatz expressions that cannot achieve below 1% relative reconstruction RMSE. As an example, in Table 3 we show the metric sweep of different decoder configurations for KdV. Similar procedures are carried out to determine the ansatzes for VB and KS.

Table 3: RelRMSE of the decoder to approximate snapshots of the KdV training trajectories resulting from different hyperparameters by minimizing Eq. (24). In this case we consider a dyadic scaling of the frequencies.

Activation function	relu			sin			swish		
Features/No. freqs	3	6	9	3	6	9	3	6	9
$(4, 4)$	0.1095	0.0255	0.0151	0.0803	0.0164	0.0110	0.1198	0.0164	0.0071
$(4, 4, 4)$	0.1010	0.0249	0.0142	0.0511	0.0129	0.0085	0.0799	0.0120	0.0059
$(8, 8)$	0.0273	0.0120	0.0117	0.0343	0.0095	0.0118	0.0637	0.0094	0.0054
$(8, 8, 8)$	0.0195	0.0101	0.0094	0.0178	0.0082	0.0107	0.0333	0.0066	0.0050
$(16, 16)$	0.0113	0.0090	0.0124	0.0202	0.0100	0.0181	0.0381	0.0076	0.0073
$(16, 16, 16)$	0.0095	0.0079	0.0096	0.0115	0.0099	0.0178	0.0186	0.0057	0.0054

Table 4: Training specifications for encoder learning. Learning rate (LR) follows an exponentially decaying schedule, i.e. multiplying by a factor after a fixed period of steps.

System	γ	Batch Size	Training Steps	Base LR	LR decay	Steps Per Decay
VB	10^0	16	8M	10^{-4}	0.912	160K
KS	10^1	32	4M	3×10^{-4}	0.892	80K
KdV	10^1	16	4M	3×10^{-4}	0.945	40K

Encoder training This step acts as another round of validation to help us see if a particular ansatz have sufficient accuracy *when being encoded*. This is a stronger condition than the above-mentioned auto-decoding evaluation since the encoder adds extra regularization. We again set the bar to be 1% relative reconstruction error, which works well for us empirically.

Notably, ansatzes that have trainable frequencies (ω_k in Eq. (10)) fail this validation despite passing the auto-decoder test. This suggests that having these extra degrees of freedom in the ansatz is not as useful as hard-coding the periodic boundary condition directly, in terms of encoder learning.

Of all ansatzes that pass the 1% criterion, we choose the one with the fewest parameters, corresponding to the ones displayed in Table 6.

The encoder training specifications are summarized in Table 4.

Dynamics training The training parameters are listed in Table 5. We additionally employ gradient clipping (scaling norm to 0.25) to help stabilize training.

Device info All training and inference runs are performed on single Nvidia V100 GPUs.

A.3 Model Details

Ansatzes The hyperparameters of the ansatz used for each system are summarized in Table 6. The columns are explained as follows:

- *features*: hidden layer sizes
- *activation*: nonlinearities applied after each hidden layer of the MLP; swish (Ramachandran et al., 2018) or sin function
- *frequency profile*: if linear, $\{\omega_k\}$ follow an arithmetic sequence $\{\pi/L, 2\pi/L, 3\pi/L, ...\}$; if exponential, the sequence is geometric $\{\pi/L, 2\pi/L, 4\pi/L, ...\}$ in powers of 2
- *frequency dimensions*: each non-zero frequency has associated $\sin$ and $\cos$ functions with distinct phases; the input and output dimensions of the MLP are both $2 \times$ No. Freqs $+ \mathbf{1}[\text{Has Zero Freq}]$.

Encoder The encoder architecture consists of multiple levels of double (one downsampling and one regular) residual blocks. Each residual block follows the sequential layer structure: Conv $\rightarrow$ BatchNorm $\rightarrow$ sin activation $\rightarrow$ Conv $\rightarrow$ BatchNorm, where downsampling is applied via striding

Table 5: Training specifications for dynamics learning. Pretrain settings are the same for all systems. *Seq Length* refers to the number of steps over which to optimize the dynamical model parameters ($= 2$ for pretraining; > 2 for fine-tuning)

Stage	Batch Size	Training Steps	Seq Length	Base LR	LR decay	Steps Per Decay
Pretrain	32	4M	2	10^{-4}	0.955	40K
Fine-tune, VB	32	1M	10	3×10^{-4}	0.955	10K
Fine-tune, KS	32	1M	20	5×10^{-5}	0.955	10K
Fine-tune, KdV	32	1M	20	1×10^{-5}	0.955	10K

Table 6: Hyperparameters for ansatz networks.

System	MLP		Frequency			No. Params
	Features	Activation	No. Freqs	Profile	Has Zero Freq	
VB	(4, 4)	swish	3	dyadic	No	84
KS	(8, 8)	sin	3	linear	No	188
KdV	(8, 8)	swish	6	dyadic	Yes	297

in the first Conv layer when applicable. Overall, each level downsamples the spatial dimension and increases the number of channels, both by a factor of 2. A standard fully-connected layer is applied at the end of the network to obtain the desired output dimension.

Encoder hyperparameters differ very little across systems, with VB encoder using 5 downsampling levels, compared to 4 used in KS and KdV (see Table 7).

Hyper U-Net The transforms represented by the arrows in Fig. 2 are as follows:

- *channel-wise projection*: locally connected layer (with kernel size 1 and no bias); project weights from 1 to D_w dimensions (first layer) and back (last layer)
- *channel-wise mixing*: sequence of transformation Dense $\rightarrow$ Swish activation $\rightarrow$ Dense $\rightarrow$ Residual $\rightarrow$ LayerNorm applied to channels within the same layer (or to the global embedding channels); parameters are unshared among layers
- *downsampling*: locally connected layer compressing all weights channels within the same layer into a single layer embedding of dimension D_l; similarly, at the global level, all layer embeddings are compressed into a single global embedding of dimension D_g
- *upsampling*: the inverse of the downsampling transform mapping single size-D_l embeddings back to multiple size-D_w ones (also the size-D_g global embedding into $N_l \times D_l$ layer embeddings)
- *copy*: similar to the original U-Net, channels from the downsampling stage are concatenated during the upsampling stage

For our ansatz, we consider each MLP layer to encompass the weights and bias $\{W, b\}$ associated with the layer compute $\sigma(Wx + b)$. The phases $\{a_{\pm k}\}$ are considered a separate layer collectively.

Our Hyper U-Net architectures for different systems differ in the hyperparameters $\{D_w, D_l, D_g\}$. Their values are listed in Table 7.

A.4 Benchmark Model Details

A.4.1 Neural Galerkin

The Neural Galerkin method (Bruna et al., 2022) seeks to compute evolution equations using calculus of variations in a time-continuous fashion.

In a nutshell, the Neural Galerkin method writes an evolution equation for the weights following:

$$M(\theta)\dot{\theta} = F(\theta), \tag{25}$$

Table 7: Hyperparameters for encoder and hyper U-Net

System	Encoder		Dynamical Model (Hyper U-Net)			
	Levels	No. Params	D_w	D_l	D_g	No. Params
VB	5	186,268	4	128	512	1,846,612
KS	4	218,116	4	512	1024	17,963,932
KdV	4	218,116	4	512	1024	17,963,932

where

$$M(\theta) = \int_0^L \nabla_\theta \Psi_\theta(x) \otimes \nabla_\theta \Psi_\theta(x) dx, \text{ and } F(\theta) = \int_0^L \nabla_\theta \Psi_\theta(x) \mathcal{F}(\nabla_\theta \Psi_\theta(x)) dx \tag{26}$$

where Ψ is the neural ansatz as described in Eq. (10) and $\mathcal{F}$ is given by the underlying PDE in Eq. (17).

The matrix $M(\theta)$ is computed by sampling $\Psi_\theta(x)$. We use a uniform random sampling on the domain to approximate the integral, namely,

$$M(\theta) = \frac{1}{N_{\texttt{samples}}} \sum_j^{N_{\texttt{samples}}} \nabla_\theta \Psi_\theta(x_j) \otimes \nabla_\theta \Psi_\theta(x_j) \tag{27}$$

We also use a trapezoidal rule to compute the integrals for the KdV equation, which seemed to work better for this particular equation.

In order to provide a fair comparison we have attenuated two main issues with this method:

1. the ansatz, which is chosen *a priori*, may not be expressive enough to represent the trajectories fully. For a fair comparison we use the neural ansatz in Eq. (10) and with the same hyperparameters in Table 6, instead of the ones used in the original paper (Bruna et al., 2022).
2. $M(\theta)$ is often ill-conditioned, so we add a small shift $M_{\texttt{stab}}(\theta) = M(\theta) + \epsilon I$, for $\epsilon = 1e{-}4$. Even with with regularization the ODE is still stiff (in our experiments the condition number is around 10^8, thus, to avoid this issue we performed the computation in double precision.

Finally, the initial condition needs to be optimized at inference time, which takes a considerable amount of time. In this case we use as an initial condition the condition given by our fully trained encoder

$$\begin{cases} M_{\texttt{stab}}(\theta)\dot{\theta} &= F(\theta), \\ \theta_0 &= \Lambda_\alpha(\mathbf{u}_0). \end{cases} \tag{28}$$

The method was implemented in JAX, and we used the adaptive Dormand-Prince integrator implemented in `scipy.integrate.solve_ivp`, and we allow the routine to choose the adaptive time step. This choice was made in order to ensure a proper integration of the stiff ODE systems.

As mentioned in the main text, we were able to compute the solutions for the Burgers and Kuramoto-Sivashinsky equations. However, for the KdV equation the ODE exploded in all of the trajectories. We played with smaller tolerances, different regularization constant, and more accurate initial conditions. However, we were not able to avoid the stability issues.

A.4.2 Neural Decoder Variant

In our main model, the ansatz network plays the role of a *decoder*, which is responsible for transforming from the latent to the ambient space. The biggest difference from a typical decoder is that the ansatz essentially *pins down* the latent space and as a result the decoding map is *not parametric*. In the neural decoder variant (NDV), we consider the *typical decoder* setup and implement a convolution-based decoder, whose trainable weights parametrizes the decoding map. This allows us to contrast and study the impact of having a fixed-form ansatz that is tailored to the problem.

For NDV, we use exactly the same encoder (architecture and latent dimensions) as our main model. The decoder architecture is essentially the reverse of the encoder. It consists of a de-projection layer

Table 8: Specifications for NDV training.

Stage	Batch Size	Training Steps	Seq Length	Base LR	LR decay	Steps Per Decay
Pretrain	8	4M	2	3×10^{-4}	0.95	40K
Fine-tune	8	1M	20	5×10^{-5}	0.95	10K

(reversing the last encoder layer), followed by a sequence of upsampling blocks (alternating regular and strided convolution layers), each of which increases the resolution and decreases the number of features by a factor of 2.

The latent dynamical model is a MLP with $(2048, 2048)$ features in its hidden layers. This corresponds to around 4.5M parameters, which is smaller compared to our hyper U-Net model. We did experiment with larger models but did not observe any performance gain.

We follow the same multi-stage protocol to train NDV. The specifications are summarized in Table 8.

We explored the use of consistency regularization on the NDV as well and did not observe any clear advantage in terms of reconstruction accuracy or smoothness gain. A possible explanation is that the latent space under NDV has the freedom to scale down and let the decoder change accordingly to maintain the same level of reconstruction performance. This results in lower values in the consistency regularization without necessarily improving the conditioning of latent trajectories.

For VB system, we explored a myriad of hyperparameter combinations including layer depths/widths, activation functions and learning rates, none of which resulted in a model that can be considered generalizing well to all snapshots.

A.4.3 DeepONet

The DeepONet is a general framework to approximate operators using neural networks. In a nutshell, for an operator $\mathcal{G}$ between two function spaces such that

$$v = \mathcal{G}(w). \tag{29}$$

DeepONet seeks to approximate this map point-wise, i.e.,

$$G(w)(y) \approx v(y), \tag{30}$$

where G is the DeepONet and y are arbitrary points in the domain of v. Following the example shown above, DeepONets are usually split into two networks: the trunk network, whose inputs are samples of w, and the branch network, whose inputs are the evaluation points y.

In this case we use them to approximate the trajectory given by an initial condition u_0 at any point (x, t) for $t > 0$ and x in the domain. Namely we consider

$$u(x, t) \approx G(\mathbf{u}_0)(x, t) = \mathcal{N}_{\texttt{trunk}}(\mathbf{u}_0) \cdot \mathcal{N}_{\texttt{branch}}(x, t), \tag{31}$$

where $\mathcal{N}_{\texttt{trunk}} : \mathbb{R}^{N_{\texttt{samples}}} \to \mathbb{R}^m$ and $\mathcal{N}_{\texttt{branch}} : \mathbb{R}^2 \to \mathbb{R}^m$, are neural networks.

For both the trunk and branch networks we use fully-connected MLPs. We fit the model on the Burgers, KS and KdV datasets, each of which has $N_{\texttt{samples}} = 512$. For the Burgers data, we used rectified linear unit (ReLU) activation functions. For the KS and the KdV datasets, we found that tanh and Gaussian Error Linear Units (GELU) activation functions (Hendrycks & Gimpel, 2016) worked better, respectively.

For the branch network we modified the input slightly. Instead of feeding (x, t) directly, we compute periodic features for the spatial coordinate x using sines and cosines at different frequencies. The main rationale for this was to impose periodicity in space to the network. Thus the input to the branch network can be rewritten as

$$\tilde{x} = \begin{bmatrix} \sin(2\pi/Lx) \\ \cos(2\pi/Lx) \\ \sin(4\pi/Lx) \\ \cos(4\pi/Lx) \\ \sin(6\pi/Lx) \\ \cos(6\pi/Lx) \\ \sin(8\pi/Lx) \\ \cos(8\pi/Lx) \end{bmatrix} \tag{32}$$

Table 9: FNO hyperparameters.

System	P	K	Layer Width (SpectralConv)	Num Layers	Num Modes	Layer Width (Final Projection)
VB	8	20	64	4	8	128
KS	32	20	256	4	16	128
KdV	32	20	256	5	16	128

and the branch network is then given by

$$\mathcal{N}_{\texttt{branch}}(x, t) = \texttt{MLP}(\tilde{x}, t) \tag{33}$$

where MLP is a fully connected network. For the VB system, both the trunk and the branch network has 4 hidden layers with 256 neurons each. The full network has $528,896$ parameters in this case. For KS and the KdV data, we used 5 layers with 512 neurons each. This resulted in a total of $2,369,536$ parameters.

We trained the network using the Adam optimizer (Kingma & Ba, 2015). After a hyper parameter sweep, the learning rate that provided the best results was 10^{-4} with an exponential decay every 10000 steps by a factor of 0.977. We used a time horizon of 300 steps to train the networks in each case. We used 5000 epochs, with a batch size of 512, in which each element of the batch is a combination of $\mathbf{u}_0$ and (x, t).

A.4.4 Fourier Neural Operator

The Fourier Neural Operator (FNO) (Li et al., 2020b) is an operator learning framework centered around the operator map/layer $\mathcal{K} : v \to v'$ defined as:

$$v'(x) = \int k(x, y)v(y)\, dy + Wv(x), \tag{34}$$

where $k(x, \cdot)$ and W are parametrized by trainable parameters. The integral term (a.k.a. spectral convolution, or SpectralConv operation) is conveniently computed in the Fourier space, where low-pass filtering may be additionally applied (controlled by the number of modes). To make up the overall operator map, the input function is first raised to a higher dimension, and then goes through multiple FNO layers Eq. (34) interleaved with nonlinear activation functions, before finally projected to the desired output dimension using a local two-layer MLP.

In our benchmark experiments, we follow the training approach outlined in the original paper, using FNO to map from a stack of P past snapshots $\{u_{t-P}, ..., u_{t-1}\}$ to u_t, with the grid locations x as an additional constant feature in the input. We train the model in an autoregressive manner by iteratively rolling out K prediction steps, each step appending the current predicted snapshot to the input for predicting the next step (and removing the oldest to keep the input shape constant). L2 loss is averaged over all K prediction steps and minimized with respect to the model parameters.

For each system, we performed sweeps over all hyperparameters. The best-performing ones are listed in Table 9.

A.5 Ablation Studies

Consistency regularization In Fig. 3, the relative reconstruction error is given by Eq. (15), and the relative consistency error is defined in a similar fashion as

$$\text{relRMSE}_{\text{consistency}}(\theta) = \frac{\|\theta - \Lambda \circ \mathbf{\Psi}(\theta)\|}{\|\theta\|}, \tag{35}$$

through which we measure whether we can recover the same θ after decoding and encoding again. The maximum component-wise, 1st order total variation norm is computed as

$$\max_i \frac{1}{Q} \sum_q^Q \sum_{n=0}^{N-1} \Delta t \left|\theta^{n+1,q} - \theta^{n,q}\right|, \tag{36}$$

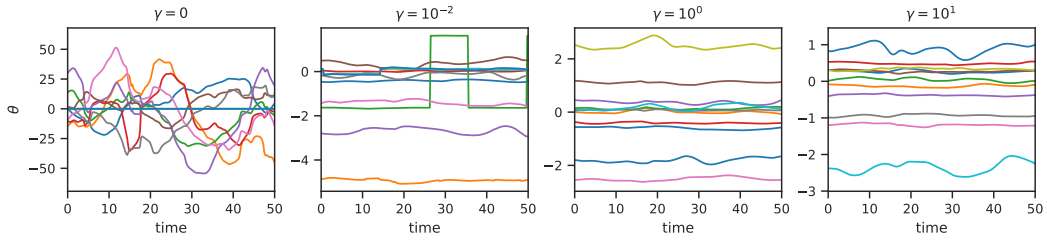

Figure 5: Latent trajectory samples resulted from training with different γ values for KS system. Trajectories plotted are for the same 10 latent/weight dimensions (identified by the same colors) randomly selected for a fixed ansatz.

where q denotes index of evaluation cases and the maximum is taken over latent dimensions i. This quantity measures how fast/slow the latent variables changes over a fixed time window.

In Fig. 5, we show samples of latent trajectories obtained using various γ values. We observe that trajectories under high γ values are visibly *smoother*.

Fine-tuning by multi-step training In Fig. 6, we compare rollouts generated by the models obtained after the pretraining and fine-tuning stages respectively, to highlight the importance of the latter. We observe that for all systems fine-tuning is able to significantly reduce the point-wise errors and, in the case of KS and KdV, remove patches in the predictions that look rather 'unphysical'.

End-to-end (e2e) training In our multi-stage training protocol, we freeze our encoders when training the dynamical model. As an ablation study, we also attempted adding an additional multi-step, e2e training stage where the parameters for both the encoder and dynamical are allowed to change, optimizing for a single rollout loss in the ambient space. Despite observing steady decrease in the training loss, this process (perhaps counter-intuitively) was in reality found to hurt the quality of rollouts significantly (errors shown in Fig. 7), especially in its stability. We suspect the reason is that without enforcing the latent evolution to remain close to highly regularized the encoder output, it is easy for the optimizer to move to a nearby local minima with equal or potentially better performance in the ambient loss. It is highly likely that the resulting dynamics, on the other hand, is not amenable to long-range rollouts in the absence of proper regularization.

A.6 Additional Metrics

We provide some additional metrics including more spatial energy spectra at various prediction ranges (Fig. 8) and sample rollouts demonstrating a richer range of patterns which are well captured by our models (Fig. 9).

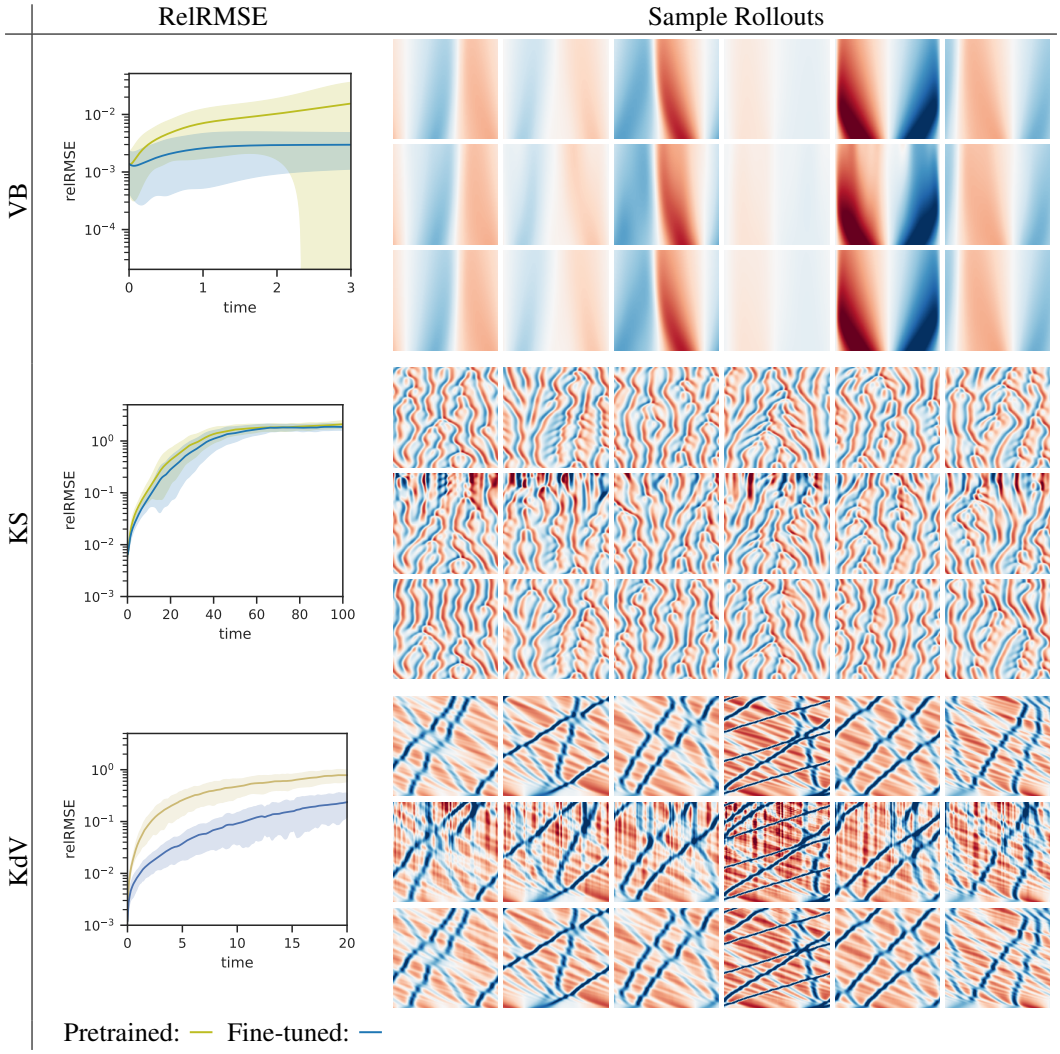

Figure 6: Relative RMSE (left) and sample rollouts (right) demonstrating the effects of fine-tuning with multi-step training. Rollouts are plotted in (x, t) plane with three rows showing the ground truths, pretrained and fine-tuned predictions respectively.

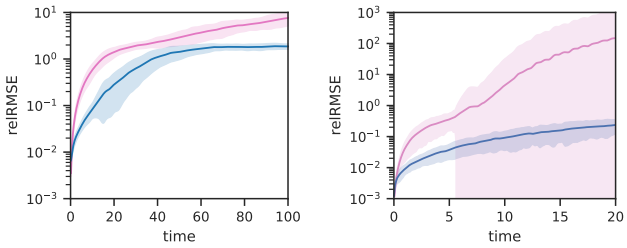

Figure 7: Relative RMSE for latent space fine-tuned (—) and e2e-trained (—) for KS (left) and KdV (right) systems.

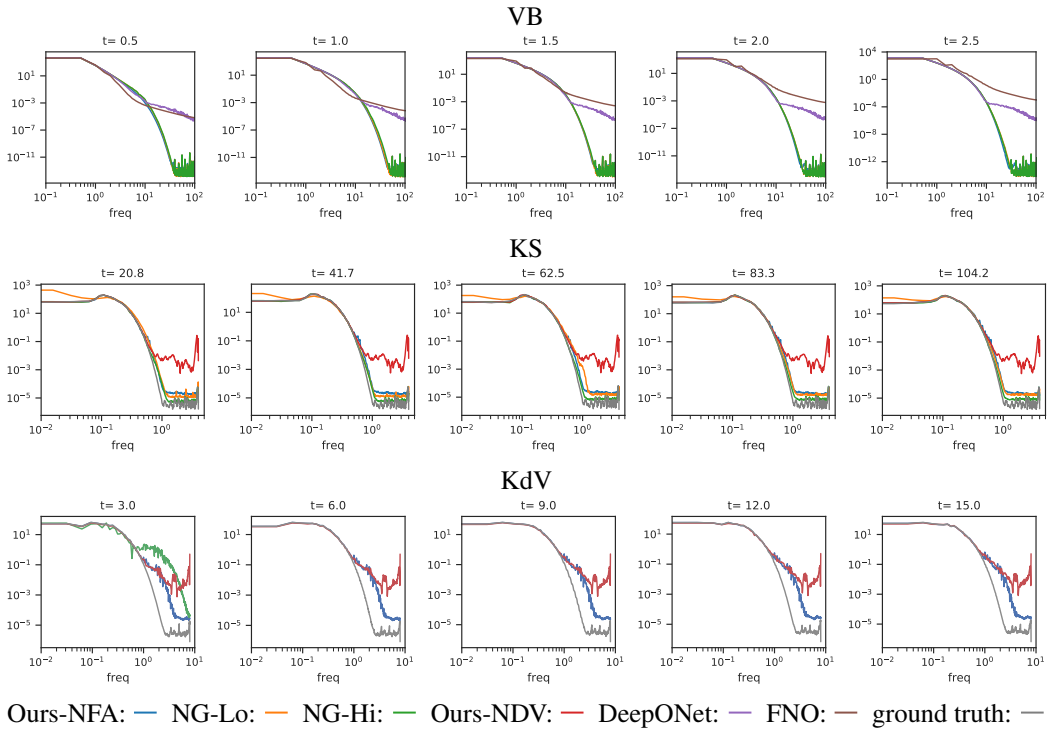

Figure 8: Spatial energy spectra, computed using Eq. (16), for different prediction ranges.

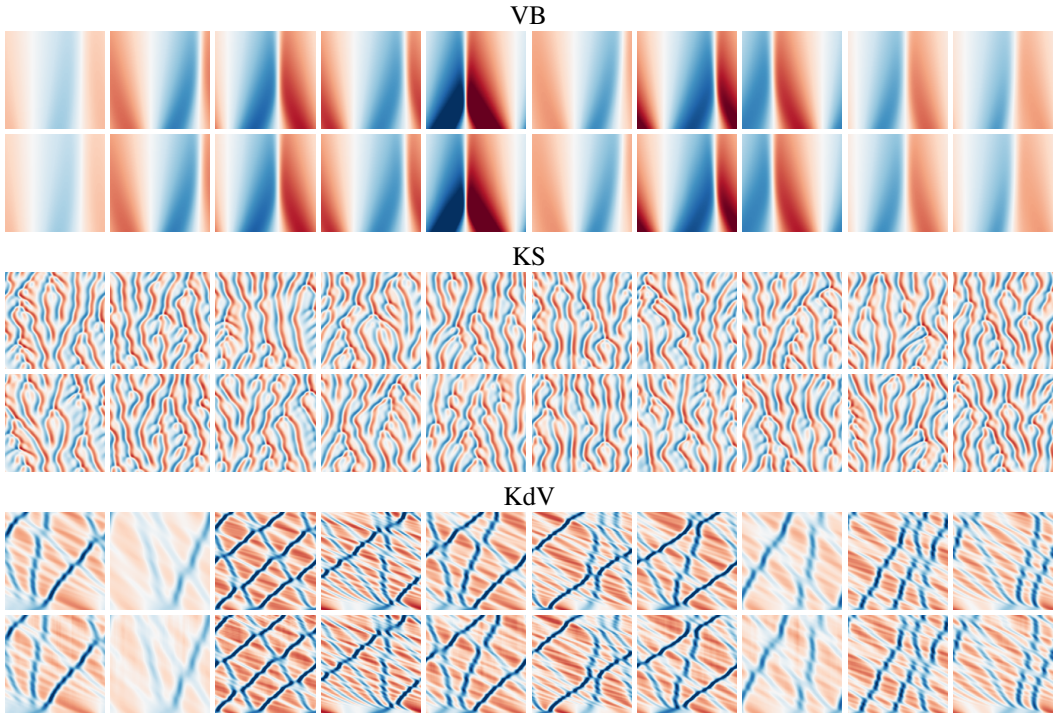

Figure 9: Additional evaluation rollouts plotted in (x, t) plane. Top rows are ground truths and bottom rows are the corresponding predictions rolled out from the same initial conditions. Prediction range: VB - 3, KS - 80, KdV - 30.