Neural Networks for Nonlinear Dynamical Systems

Yuying Liu

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2022

Reading Committee:

J. Nathan Kutz, Chair

Steven L. Brunton

Tim Leung

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Neural Networks for Nonlinear Dynamical Systems

Yuying Liu

Chair of the Supervisory Committee:
Robert Bolles and Yasuko Endo Professor J. Nathan Kutz
Department of Applied Mathematics

Nonlinear dynamical systems are ubiquitous in many fields of sciences and engineering. Throughout the history, differential equations were widely accepted as effective tools for describing the evolution of such systems in continuous time, with the earliest example dating back to the mid-1600s when Isaac Newton discovered his laws of motion and universal gravitation, and combined them to explain the planetary motion. However, two difficulties often rise in practice. First, identifying governing law of a system requires tremendous insights. The dynamics are often so complex that even to build up a coarse-grained model is a nontrivial task. Second, obtaining solutions from the governing equation is also challenging, most of which need to resort to numerical simulations. And high-fidelity simulations are often expensive to conduct. Thanks to the recent advances in deep learning and the abundance of measurement data, now we have alternative ways to address these two challenges. In this thesis, we present several neural network architectures for nonlinear dynamical systems. We begin by introducing neural network-based hierarchical time-steppers (HiTS) which enables highly accurate and efficient numerical integration over a long period of time. Then we turn to multi-resolution convolutional autoencoder (MrCAE) which integrates and leverages three highly successful mathematical architectures: (i) multigrid methods, (ii) convolutional autoencoders and (iii) transfer learning, providing an adaptive, hierarchical architecture that capitalizes on a progressive training approach for multiscale spatio-temporal data. Finally, we introduce physics-informed Koopman network (PIKN)

which leverages automatic differentiation to impose the underlying physical laws via soft penalty constraints so that to identify the intrinsic, linearized coordinates for nonlinear dynamics. The first two architectures are purely data-driven, aiming to provide an alternative way to model nonlinear dynamical systems and run efficient simulations. The third architecture is physics-informed, providing a data-efficient framework to numerically approximate the Koopman operator, enabling fast simulations.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First of all, I am deeply thankful to my advisors Nathan Kutz and Steve Brunton, not only for their professional support and research guidance, but also for their mental support and sparkling personality which truly comforts, encourages and inspires me. Working with them has been a huge pleasure, in fact, I dare to say that this 4-year study at the University of Washington is by far the most cultivating and enriching experience I've ever had. I am forever grateful.

I also would like to thank the other members in my committee Tim Leung and Kevin Jamieson who are able to squeeze some time for joining and mentoring. I am very appreciative of the thought-provoking questions and discussion at my general and final exams. I would also like to extend my gratitude to Lauren Lederer, Tony Garcia, Bernard Deconinck and Matt Lorig for their assistance in navigating the logistical side of graduate school.

A big thank to my numerous mentors/hosts in the industries. Indeed, internship experience has been another big part of my PhD journey. I want to thank Colin Ponce at Lawrence Livermore National Lab for being supportive on the research direction I chose and helping me finish my first PhD paper. I want to thank Josh Proctor at Bill & Melinda Gates Foundation for investing so much time helping me understand the contexts of the project and grant me huge flexibility on model innovation. I want to thank Saleh Nabi at Mitsubishi Electric Research Laboratories for being willing to go through all the details of the research papers and my code line-by-line when I get stuck. And I want to thank Geoff Anders at Google X, the moonshot factory, for being able to onboard me even if my residence is so short due to my graduation timeline.

A big thank also goes to my collaborators: Mollie Van Gordon, David Barajas-Solano, Alex Tartakovsky, Floris van Breugel, Bing Brunton, Hassen Mansour, Charlotte Leroy, Hongxu Ma, Jeff Bush, Alexandre Szenicer, Purva Pruthi, Greg Bronevetsky, Alexander Muhr, etc. and also to the research group members at UW: Sam Rudy, Brian DeSilva, Kathleen Champion, Bethany Lusch, Craig Gin, Henning Lange, Seth Hirsh, Zhe Bai, Chang Sun, Charlie Fieseler, Krithika Manohar, Charles DeLahunt, Shaowu Pan, Kardier-dan Kaheman, etc. I always feel lucky to have had the opportunity to work with such an exceptional group of collaborators and I have learned a lot from you guys.

I also owe my thanks to my family and friends. In particular, I want to thank my parents Xueming Liu and Liqin Zang and my grandma Fengmei Shi for their unfailing love and support on my education throughout 28 years. I want to thank my roommates Mingrui Zhang and Yi Chu, they've always been there to cheer me on through my successes and to listen patiently when I'm stressed. My wonderful officemates Olga Dorabiala, Kelly Liu, Iris Shi, Doris Voina and Matthew Farrell, it's so great to have you work by my side and have some cheerful small talks when I feel exhausted! My magical cohorts: Diya Sashidhar, Jorge Cisneros, Katherine Lacy Owens, Micah Henson, Nora Gilbertson, Roman Levin, Tyler Chen and Ying-Jen Yang, I never feel lonely throughout this 4-year long journey because of your company! And my unbeatable teammate Theodore Zhao, it's so great to have you by my side in many datathon and hackthon competitions. I feel so excited at the night we won the prize money from Citadel.

Finally, I want to thank for all the people who once helped me, encouraged me, loved me in my life. From a dynamical system point of view, any action of kindness, albeit insignificant, may trigger dramatically different consequences in one's future life. Maybe I've forgotten your names, but I believe everything I've experienced brings me where I am, who I am now. So thank you. And that's also why life is so beautiful.

DEDICATION

This dissertation is dedicated to my beloved grandma who passed away on May 17, 2020. I promised to make her proud by being a kind, decent and capable person. And I hope the efforts I made up to this point have partially fulfilled that promise.

Chapter 1

INTRODUCTION

Nonlinear dynamical systems are ubiquitous. From the expansion of the universe, to the evolution of life, and to the globalization of human economics, a series of interconnected discoveries were made, revealing the hidden face of nature while also blurring the traditional boundaries between natural science, social science, cognitive science, mathematics and philosophy. Symmetry and complexity are two prominent features of these systems. Starting from symmetry breaking, an inanimate matter without purpose or design, can spontaneously create exquisite beauty, leading to emergence of new order or structure, all based on the most basic laws. To further understand the mechanism of these systems, one way is to do quantitative study using mathematical models, often in the form of differential equations. Maxwell's equations, for example, describe the behavior of electric and magnetic fields. Navier-Stokes equations provide a mathematical description of viscous fluid flow. Einstein field equations, relate the local spacetime curvature with the local energy, supplementing Newton's law of universal gravitation. All these models are unreasonably effective and provide high-fidelity results if properly simulated which is still a mystery to science, as once commented by a great physicist Eugene Wigner: *"The miracle of the appropriateness of the language of mathematics for the formulation of the laws of physics is a wonderful gift which we neither understand nor deserve. We should be grateful for it and hope that it will remain valid in future research and that it will extend, for better or for worse, to our pleasure, even though perhaps also to our bafflement, to wide branches of learning."* Historically when data were scarce, these models were very popular as they were hypotheses-driven (and hence biased), people collected data mainly for validating purposes. However, there are certain downsides for this approach. First, the majority of real-world interacting systems are far too complicated to model in their entirety. Identifying the most important part and excluding the rest is the key to success, which requires tremendous insights. The

second level of compromises come from mathematical manipulations. Small changes of the equations (that reflects reality) may lead to dramatic changes in the mathematical methods for solving them, restricting the usage of the model. There's often a sweet spot between elegancy and usability.

Thanks to the advancement of information and sensor technologies, the presence of massive data of different fields and ever-increasing rise of computational power now open a gate to new modeling techniques: data-driven modeling. In these models, data are not only used for validation, but also for extracting non-trivial and yet hard-to-find patterns. Statistical and machine learning algorithms are the real workhorses behind the scene. At the forefront of these algorithms are the deep neural networks (DNNs) which have long been deemed as the gold standard by practitioners. They not only achieve superior performance in static tasks such as image classification, but also show great potential for future state prediction of nonlinear dynamical systems. However, a key limitation of DNNs, similar to other data-driven models as well, is the lack of interpretability and transparency. This often leads to the users' misuse of models and lack of confidence. It is also observed that these models seldom truly generalize to unseen scenarios. Another perhaps equally severe problem is that DNNs are data-hungry and computationally demanding: for some serious learning tasks, a DNN usually demands millions of training data points and takes months of training on hundreds of GPUs to fully digest the information. Imposing known physical constraints often alleviates these limitations.

1.1 Organization

This thesis addresses several challenges associated with leveraging deep learning (or neural networks) for nonlinear dynamical system applications. In particular, we focus on promoting the strengths and avoiding the weaknesses of both deep learning and classical approaches.

In Chapter 2, we introduce Hierarchical multiscale deep learning Time-Steppers (HiTS), which learn flow map of different time scales of a dynamical system from data. Then a hierarchical multiscale time-stepping scheme can be used to conduct numerical integration which is both accurate and highly efficient. This scheme provides important advantages over current time-stepping algorithms, including (i) circumventing numerical stiffness due

to disparate time-scales, (ii) improved accuracy in comparison with leading neural-network architectures, (iii) efficiency in long-time simulation/forecasting due to explicit training of slow time-scale dynamics, and (iv) a flexible framework that is parallelizable and may be integrated with traditional numerical time-stepping algorithms. In this work, the key to the success is that deep neural networks can represent complex flow maps associated with large time steps, complementing the traditional numerical time-stepping algorithms which are restricted by a local Taylor series expansion.

Chapter 3 introduces Multi-resolution Convolutional Autoencoder (MrCAE) architecture that integrates and leverages three highly successful tools or methods: (i) multigrid methods, (ii) convolutional autoencoders and (iii) transfer learning. The method provides an adaptive, hierarchical architecture that capitalizes on a progressive training approach for multiscale spatio-temporal data. This framework allows for inputs across multiple scales: starting from a compact (small number of weights) network architecture and low-resolution data, our network progressively deepens and widens itself in a principled manner to encode new information in the higher resolution data based on its current performance of reconstruction. Basic transfer learning techniques are applied to ensure information learned from previous training steps can be rapidly transferred to the larger network. As a result, the network can dynamically capture different scaled features at different depths of the network. In a word, this work leverages multi-scale features from data and an adaptive architecture, allowing for a more transparent procedure and an interpretable neural network model.

In Chapter 4, we turn to Koopman operator theory. This theory recently receives increasing attention due to its promise to linearize the nonlinear dynamics. We propose a physics-informed Koopman networks (PIKNs) that can be used to automatically identify Koopman eigenvalues and eigenfunctions, which is the key to a successful linearization. Unlike other proposed methods in the literature, we combine the strengths of physics-informed neural networks (PINNs) and autoencoder-based Koopman networks, reducing the need of large training data-sets while keeping the generalization property for future state predictions. And since the network performs (Koopman) operator learning, it can be used to solve multiple instances once it is trained. This framework clearly illustrates how we can build in physical laws into deep learning by leveraging automatic differentiation.

Chapter 2

HIERARCHICAL DEEP LEARNING OF MULTISCALE NEURAL NETWORK TIME-STEPPERS

2.1 abstract

Nonlinear differential equations rarely admit closed-form solutions, thus requiring numerical time-stepping algorithms to approximate solutions. Further, many systems characterized by multiscale physics exhibit dynamics over a vast range of timescales, making numerical integration expensive. In this work, we develop a hierarchy of deep neural network time-steppers to approximate the dynamical system flow map over a range of time-scales. The model is purely data-driven, enabling accurate and efficient numerical integration and forecasting. Similar ideas can be used to couple neural network-based models with classical numerical time-steppers. Our hierarchical time-stepping scheme provides advantages over current time-stepping algorithms, including (i) capturing a range of timescales, (ii) improved accuracy in comparison with leading neural network architectures, (iii) efficiency in long-time forecasting due to explicit training of slow time-scale dynamics, and (iv) a flexible framework that is parallelizable and may be integrated with standard numerical time-stepping algorithms. The method is demonstrated on numerous nonlinear dynamical systems, including the Van der Pol oscillator, the Lorenz system, the Kuramoto–Sivashinsky equation, and fluid flow pass a cylinder; audio and video signals are also explored. On the sequence generation examples, we benchmark our algorithm against state-of-the-art methods, such as LSTM, reservoir computing, and clockwork RNN.

2.2 Introduction

Scientific computing has revolutionized nearly every scientific discipline, allowing for the ability to model, simulate, engineer, and optimize a complex system's design and performance. This capability has been especially important in nonlinear, multiscale systems

where recourse to analytic and perturbation methods are limited. For instance, modern high-fidelity simulations enable researchers to design aircraft, simulate the evolution of galaxies, quantify atmospheric and ocean interactions for weather forecasting, and model high-dimensional neuronal networks of the brain. Thus, given a set of governing equations, typically spatio-temporal *partial differential equations* (PDEs), discretization in time and space form the foundational algorithmic structure of scientific computing [91, 51, 123]. Discretization is required to accurately resolve all relevant spatial and temporal scales in order to produce a high-fidelity representation of the dynamics. Such resolution can be prohibitively expensive, as resolving physics on fast time scales limits simulation times and the ability to model slow timescale processes [62, 39]. Time-stepping schemes are typically based on Taylor series expansions, which are local in time and have a numerical accuracy determined by the step size Δt. However, there is a growing effort to develop *deep neural networks* (DNNs) to learn time-stepping schemes unrestricted by local Taylor series constraints [184, 210, 198, 126]. We build on the flow map viewpoint of dynamical systems [275, 35, 198] in order to learn *hierarchical time-steppers* (HiTSs) that explicitly exploit the multiscale flow map structure of a dynamical system over a disparate range of time-scales. In leveraging features at different timescales, we can produce an accurate and efficient computational scheme that can provide exceptional efficiency in long-time simulation/forecasting and that can be integrated with classical time-stepping algorithms.

Numerical discretization has been extensively studied since the earliest days of scientific computing. Numerical analysis has provided rigorous estimates of error bounds for the diversity of discretization schemes developed over the past few decades [91, 51, 123]. Spatial discretization predominantly involves finite element, finite difference, or spectral methods. Multigrid methods have been extensively developed in physics-based simulations where coarse grained models must be progressively refined in order to achieve a required numerical precision while remaining tractable [163, 244]. The resulting discretized dynamics may be generically represented as a nonlinear dynamical system of the form

$$\frac{d}{dt}\boldsymbol{x}(t) = \boldsymbol{f}(\boldsymbol{x}(t), t) \tag{2.1}$$

in terms of a state $\mathbf{x} \in \mathbb{R}^D$ (typically $D \gg 1$). The dynamics are then integrated with

a time-stepping algorithm. As with spatial discretization, there is a wide range of techniques developed for time-stepping, including explicit and implicit schemes, which have varying degrees of stability and accuracy. These schemes approximate the discrete-time flow map [81, 265]

$$\boldsymbol{x}(t+\Delta t) = \boldsymbol{F}(\boldsymbol{x}(t), \Delta t) \triangleq \int_t^{t+\Delta t} \boldsymbol{f}(\boldsymbol{x}(\tau), \tau)\, d\tau, \tag{2.2}$$

often through a Taylor-series expansion. Runge-Kutta, for which the Euler method is a subset, is one of the standard time-stepping schemes used in practice. Generically, it takes the form

$$\mathbf{x}_{n+1} \approx \tilde{\boldsymbol{F}}_{\Delta t}(\mathbf{x}_n) \triangleq \mathbf{x}_n + \Delta t \sum_{j=1}^{k} b_j \mathbf{h}_j \tag{2.3}$$

where

$$\mathbf{h}_j = \boldsymbol{f}\left(\mathbf{x}_n + g\left(\sum_{k=1}^{j-1} \alpha_{j,k}\mathbf{h}_k\right), t_n + c_j \Delta t\right) \tag{2.4}$$

and $\mathbf{x}_n = \mathbf{x}(t_n) = \mathbf{x}(n\Delta t)$. Note that the contributions $\mathbf{h}_j$ are hierarchically computed in the Runge-Kutta scheme. The weightings b_j, c_j and $\alpha_{j,k}$ are derived from Taylor series expansions in order to minimize error. For instance, the classic fourth-order Runge-Kutta scheme, for which $k = 4$ above, has a local truncation error of $\mathcal{O}(\Delta t^5)$, which leads to a global time-stepping error of $\mathcal{O}(\Delta t^4)$. Euler stepping, for which $k = 1$, has local and global time-stepping errors of $\mathcal{O}(\Delta t)$ and $\mathcal{O}(\Delta t^2)$ respectively. Importantly, the error is explicitly related to the time-step Δt, making such time-discretization schemes *local* in nature.

In contrast to schemes such as Runge-Kutta, that approximate the flow map with a local Taylor series, it is possible to directly construct an approximate flow map $\hat{\boldsymbol{F}}$ using DNN architectures. There are several approaches to modeling flow-map time-steppers using neural networks, which will be reviewed below. The approach taken in this work is to develop a hierarchy of approximate flow maps $\hat{\boldsymbol{F}}_j(\mathbf{x}, \Delta t_j)$ to facilitate the accurate and efficient simulation of multiscale systems over a range of time-scales; similar flow map composition schemes have been demonstrated to be highly effective for simulating differential equations without neural networks [275, 35, 147]. Flow maps also provide a robust framework for model discovery of multiscale physics [29, 28]. Various existing DNN architectures can be integrated into this hierarchical framework. Flow map approximations based on the Taylor series are typically only valid for small time steps, as the flow map becomes arbitrarily

complex for large time steps in chaotic systems. However, DNN architectures are not limited by this small time step constraint, as they may be sufficiently descriptive to approximate exceedingly complex flow map functions [96].

Neural networks have been used to model dynamical systems for decades [75, 166]. They are computational models that are composed of multiple layers and are used to learn representations of data [129, 76]. Due to their remarkable performance on many data-driven tasks [121, 241, 165, 224, 49, 230], various new architectures that favor interpretability and promote physical insight have been recently proposed, leading to many successful applications. In particular, it has been shown that neural networks may be used in conjunction with classical methods in numerical analysis to obtain discrete time-steppers [207, 184, 210, 198, 213, 117]. Other applications include reduced order modeling [180, 247], multiscale modeling [99, 254, 58, 139], scientific computing [206, 11, 228, 57], coordinate transformations [149, 41], attractor reconstructions [187, 146], operator learning [144, 248] and forecasting [181, 188, 246, 264]. Neural network models are increasingly popular for two reasons. First, the universal approximation theorem guarantees that arbitrary continuous functions can be approximated by neural networks with sufficiently many hidden units [53]. Second, neural networks themselves can be viewed as discretizations of continuous dynamical systems [258, 262, 153, 42, 45, 82, 195], which makes them suitable for studying dynamics. Among all architectures, *recurrent neural networks* (RNNs) are natural candidates for temporal sequence modeling; however, training has proven to be especially difficult due to the notorious exploding/vanishing gradient problem [93, 15]. To alleviate this problem, new architectures have been proposed [94, 100, 185], for example, augmenting the network with explicit memory using gating mechanism, resulting in the *long short term memory* (LSTM) algorithm [94], or adding skipped connections to the network, leading to *residual networks* (ResNet) [85].

In this work, we expand on [198] and employ a deep *residual network* (ResNet) as the basic building block for modeling the flow-map dynamics (2.2). Unconstrained by the typical form of a Taylor-series based time-stepping scheme (2.3), a multiscale modeling perspective is taken to strengthen the performance of our proposed multiscale, flow-map models. Our multiscale HiTS algorithm consists of ResNet models trained to perform hierarchical time-

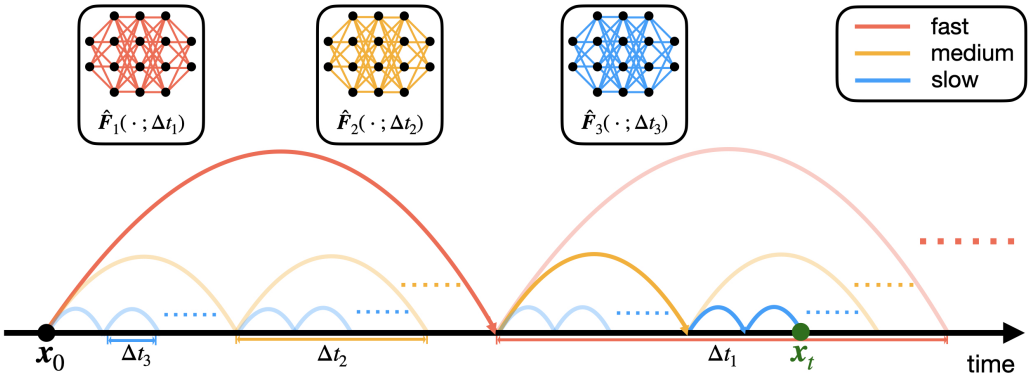

Figure 2.1: **Multiscale hierarchical time-stepping scheme**. Here, we employ neural network time-steppers over three time scales. The red model takes large steps, leaving the finer time-stepping to the yellow and blue models. The dark path shows the sequence of maps from $\boldsymbol{x}_0$ to $\boldsymbol{x}_t$.

stepping tasks. The contributions of this work are summarized as follows:

- We propose a novel method to couple neural network HiTSs trained across different time scales, shown in Fig. 2.1, resulting in more accurate future state forecasts without losing computational efficiency.

- Neural network HiTSs may be coupled with classical numerical time-steppers. This hybrid time-stepping scheme can be naturally parallelized, accelerating classical numerical simulation algorithms.

- By coupling models across different scales, each individual model only needs to be trained over a short period without being exposed to the exploding/vanishing gradient problem, enabling faster training.

- Despite the structural simplicity, the coupled model can still be used to capture long-term dependencies, achieving state-of-the-art performance on sequence generation.

The paper is organized as follows. We motivate the proposed approach and present the methodology in Sec. 2.3. Our approach is then tested on several benchmark problems in

Sec. 2.4. In Sec. 2.5, we conclude and discuss future directions. Our code is publicly available at `https://github.com/luckystarufo/multiscale_HiTS`.

2.3 Multiscale Time-Stepping with Deep Learning

Here we outline our multiscale hierarchical time-stepping based on deep learning, illustrated in Fig. 2.1. Our approach constructs a hierarchy of flow maps, $\hat{\boldsymbol{F}}_j(\mathbf{x}, \Delta t_j)$, each approximated with a deep neural network. This enables accurate and efficient simulations with fine temporal resolution over long time scales, as compared in Fig. 2.2. We begin with a motivating example, followed by a summary of notation and description of the training data. Then we will introduce our multiscale hierarchical time-stepping scheme, including descriptions on how to vectorize operations and create hybrid time-steppers by combining with classical numerical methods.

2.3.1 Motivating example

To explore the effect of time step size on simulation performance, we consider the following simple linear differential equation for a harmonic oscillator

$$\dot{x} = y \tag{2.5a}$$

$$\dot{y} = -x. \tag{2.5b}$$

We individually train 8 neural networks with step sizes Δt of 0.008, 0.016, 0.032, 0.064, 0.128, 0.256, 0.512 and 1.024 on 500 sampled trajectories and test them on 100 new trajectories. Linear interpolation is used to estimate the state at time steps that are not directly obtained from the neural network time-stepping scheme. To evaluate the forecasting performance, we calculate the averaged error at each time step against the ground truth analytical solution. In this experiment, training and testing data are both sampled from the region $\{(x, y)|\ x^2 + y^2 \leq 1\}$, and we only train one step forward for each neural network time-stepper.

Results are plotted in Fig. 2.2, where it is clear that the proposed multiscale time-stepper outperforms all fixed step models. Networks equipped with small time steps offer accurate short-term predictions, although error accumulates at each step and quickly dominates. Networks with large time steps can handle long-term predictions, although they

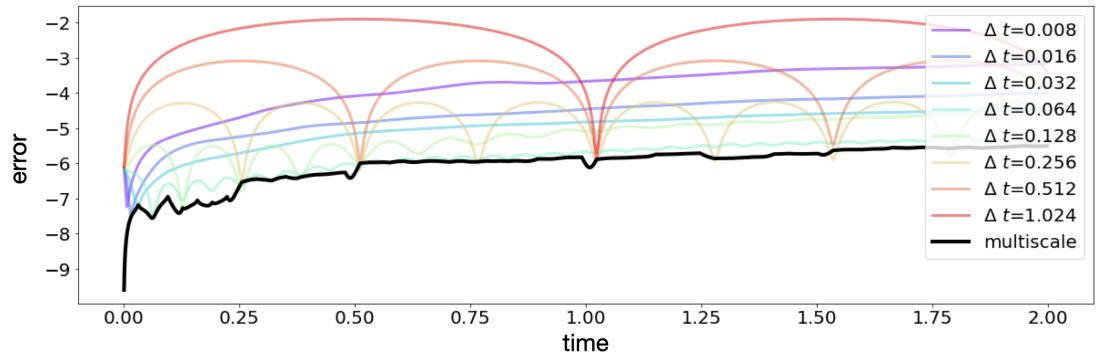

Figure 2.2: **Performance of multiscale HiTS on harmonic oscillator example**. This figure shows the time-stepping performance of different neural network time-steppers. 100 testing trajectories are used for benchmarking each time-stepper and the mean squared errors at each step are plotted in the base-10 logarithmic scale. The black curve represents our proposed multiscale scheme whereas other colors represent time-steppers at particular scales.

fail to provide information between steps. Time-steppers with some intermediate step sizes (eg. $\Delta t = 0.064$) perform well in general as they balance these two factors. However, by leveraging all time steppers across each scale, it is possible to create a multiscale HiTS, given by the black curve, that is both accurate and efficient over long time scales and with fine temporal resolution. Details of the methodology will be presented in the remainder of this section.

2.3.2 Notation and training data

For training data, we collect n trajectories sampled at p instances with time step Δt:

$$S^{(i)} = \{(\boldsymbol{x}_t^{(i)}, \boldsymbol{x}_{t+\Delta t}^{(i)}, \boldsymbol{x}_{t+2\Delta t}^{(i)}, \cdots, \boldsymbol{x}_{t+p\Delta t}^{(i)})\} \tag{2.6}$$

for $i = 0, 1, \cdots, n-1$. This data is used to train neural network flow map approximations, $\hat{\boldsymbol{F}}_j(\mathbf{x}, \Delta t_j)$, for $j = 1, 2, \cdots, m$.

We follow [198] and employ the residual network as our fundamental building block of our deep learning architecture. Residual networks were first proposed in [85] and have

gained considerable prominence since. Specifically, our network only models the difference between time steps $\boldsymbol{x}_{n+1}$ and $\boldsymbol{x}_n$, so that it is technically the flow map minus the identity:

$$\hat{\boldsymbol{x}}_{t+\Delta t_j} = \boldsymbol{x}_t + \hat{\boldsymbol{F}}_j(\mathbf{x}, \Delta t_j) \tag{2.7}$$

where

$$\hat{\boldsymbol{F}}_j(\mathbf{x}, \Delta t_j) = \sigma_M \left(\mathbf{A}_{M-1} \left(\cdots \sigma_1 \left(\mathbf{A}_0 \right) \cdots \right) \right) \tag{2.8}$$

is a feed-forward neural network. The network is parameterized by the linear operators $\mathbf{A}_j$, and σ_j are nonlinear activation functions that are chosen to be rectified linear units (ReLU). The extra addition creates a skipped connection from the inputs to the outputs. Here, the architecture $\hat{\boldsymbol{F}}_j$ learns the increment in states between each Δt_j step. It is possible to compose the networks to take multiple steps forward in time: $\hat{\boldsymbol{F}}_j^{(k)} = \underbrace{\hat{\boldsymbol{F}}_j \circ \hat{\boldsymbol{F}}_j \circ \cdots \circ \hat{\boldsymbol{F}}_j}_{k \text{ times}}$. Finally, we formulate our training objective function as

$$MSE = \frac{1}{np} \sum_{i=1}^{n} \sum_{k=1}^{p} (\hat{\boldsymbol{x}}_{t+k\Delta t_j}^{(i)} - \boldsymbol{x}_{t+k\Delta t_j}^{(i)})^2 \tag{2.9}$$

which is the classical mean squared loss function.

2.3.3 Multiscale hierarchical time-stepping scheme

Multiscale modeling is ubiquitous in modern physics-based simulation models. Computational challenges arise in these simulations since coarse-grained macroscale models are usually not accurate enough and microscale models are too expensive to be used in practice [61]. By coupling macroscopic and microscopic models, we hope to take advantage of the simplicity and efficiency of the macroscopic models, as well as the accuracy of the microscopic models [259]. Many efforts have been made towards this goal, resulting in many algorithms that exploit multiscale structure in space and time, including the multi-grid method [30], the fast multipole method [79], adaptive mesh refinement [19], domain decomposition [242], multi-resolution representation [55], multi-resolution dynamic mode decomposition [125], etc. Mathematical algorithms such as heterogeneous multiscale modeling (HMM) [256, 260] and the equation-free approach [114] attempt to develop general guidelines and provide principled methods for this field. In this work, however, we develop data-driven models

using neural networks, which have different considerations than physics-based simulation models. Regardless, the goal is still to produce accurate and efficient computational models.

Coupling neural network time-steppers across different time scales is rather straightforward, as shown in Fig. 2.1. One can clearly see the time-steppers with small Δt are responsible for the accurate time-stepping results over short periods, while the models with larger Δt steps are used to 'reset' the predictions over longer periods, preventing error accumulations from the short-time models. Mathematically, suppose we have $\hat{\mathbf{F}}_j(\cdot;\Delta t_j)$ approximating the true flow map of time step size $\Delta t_j = 2^{m-j}\Delta t$ $(j = 1, 2, \cdots, m)$ where Δt represents the unit step size. To forecast the state, for example after a time period of $q\Delta t$ $(q < 2^m)$, we would need to step forward q_j times with model j for $j = 1, 2, \cdots, m$, where $q = 2^{m-1}q_1 + 2^{m-2}q_2 + \cdots + 2^0 q_m$ is the binary representation of q. All other intermediate time steps will be obtained via linear interpolation. So instead of stepping forward in a sequential manner (i.e. using the smallest scaled model to step forward for q steps), we have reduced the total number of steps to order $\mathcal{O}(\log_2 q)$. There are additional benefits of this multiscale coupling in time. First, training each individual network is simpler, as it is possible to use trajectories with small p so that each model may focus on its own range of interest, circumventing the problem of exploding/vanishing gradients. Second, the framework is flexible, so that for forecasting it is possible to vectorize the computations or utilize parallel computing technologies, enabling fast time-stepping schemes. Moreover, it can be easily combined with classical numerical time-steppers, resulting in hybrid schemes, boosting the performance of simulation algorithms.

The proposed multiscale coupling procedure may resemble those used in multiscale simulations (e.g., HMM). However, the data-driven microscopic cannot provide accurate long-time forecasts on its own, so the coupling here is not only for efficiency but also to improve the long-time fidelity of the model. It should also be noted that before coupling neural network time-steppers across different time scales, cross validation is used to filter the models. This is practically helpful because qualities of different models may vary and we want to couple the best set of models. Specifically, suppose we have m neural network models $\{\hat{\boldsymbol{F}}_1, \hat{\boldsymbol{F}}_2, \cdots, \hat{\boldsymbol{F}}_m\}$, ordered by their associated step sizes. We first determine the upper bound index u so that ensembling $\{\hat{\boldsymbol{F}}_1, \hat{\boldsymbol{F}}_2, \cdots, \hat{\boldsymbol{F}}_u\}$ has the best time-stepping perfor-

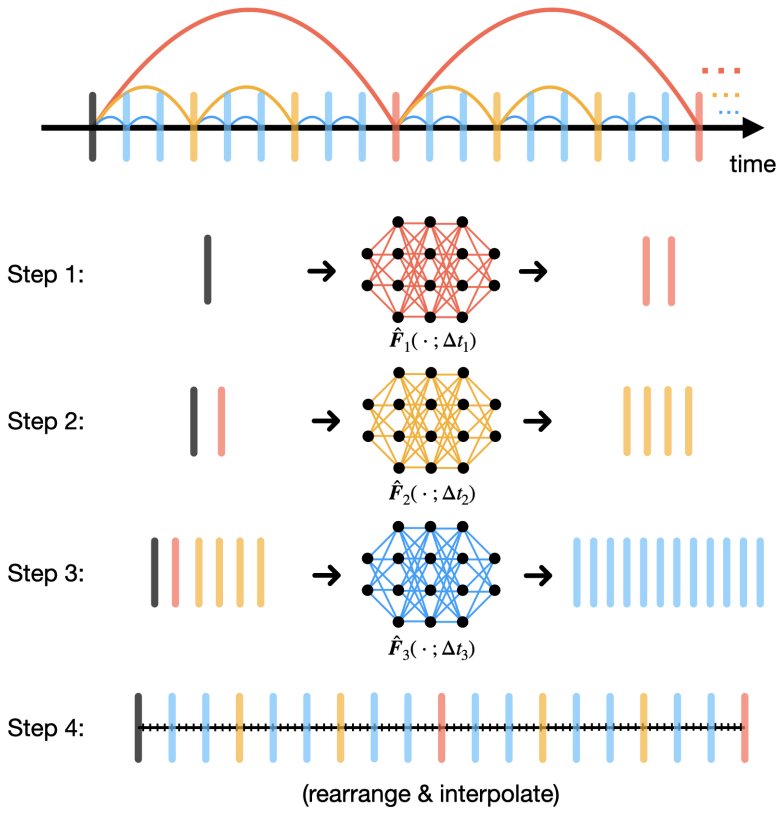

Figure 2.3: **Vectorized computation**. The three neural networks in this diagram are used sequentially, ordered by their associated step sizes from large to small. For each network, we stack all currently existing states and step forward (in the beginning, we only use the initial state), resulting in vectorized computations. These newly generated states are further fed to the next neural network in queue. Once we finish using all networks, the states will be rearranged in terms of chronological order and intermediate time steps will be obtained via interpolation.

mance among $\{\hat{\boldsymbol{F}}_1, \hat{\boldsymbol{F}}_2, \cdots, \hat{\boldsymbol{F}}_k\}$ for all k. Next, we seek a lower bound index l so that $\{\hat{\boldsymbol{F}}_l, \hat{\boldsymbol{F}}_{l+1}, \cdots, \hat{\boldsymbol{F}}_u\}$ performs best among $\{\hat{\boldsymbol{F}}_k, \hat{\boldsymbol{F}}_{k+1}, \cdots, \hat{\boldsymbol{F}}_u\}$ for all k.

Vectorization. The diagram for vectorized computations is illustrated in Fig. 2.3. The basic idea is to start by using the time-steppers with the largest Δt step and generate the future states corresponding with this stepper. We then stack the new states with

Algorithm 1 Vectorized multiscale hierarchical time-stepping

Input: a set of neural network time-steppers $\hat{\boldsymbol{F}}s$

Output: a list of predicted states Xs sorted in chronological order

function VectorizedMultiScaleTimeStepping($\hat{\boldsymbol{F}}s$)
  $Sort(\hat{\boldsymbol{F}}s)$; ▷ models are ordered with decreasing step sizes
  $Xs = List()$; ▷ create an empty list
  $Append(\boldsymbol{x}_0, Xs)$; ▷ append initial state to the list
  for i in $1, 2, \cdots, m$: **do**
    K_i := the number of steps forward with $\hat{\boldsymbol{F}}_i$;
    $X_{cur} = Stack(Xs)$; ▷ stack together all states in Xs
    $X_{next} = \hat{\boldsymbol{F}}_i.Forward(X_{cur}, K_i)$; ▷ step forward K_i steps with $\hat{\boldsymbol{F}}_i$ and X_{cur}
    $Append(X_{next}, Xs)$; ▷ update the list with new states
  end for
  $Rearrange(Xs)$;
  $Interpolate(Xs)$; **return** Xs
end function

the original states and feed them to the next-level neural network time-stepper. After we proceed through all time-steppers, we rearrange the states and use interpolation to fill in the state at all intermediate time steps. Details are given in Algorithm 1.

Hybrid time-steppers. The flexibility of our multiscale coupling approach makes it possible to combine these time-steppers with classical numerical time-stepping algorithms. The algorithm is detailed in Algorithm 2 and the concept is illustrated in Fig. 2.4. This approach provides an innovation in the computational paradigm of numerical simulations. If one were to use classical numerical time-steppers (eg. Runge-Kutta method) alone, opportunities for vectorized computations or parallel computations are limited due to the serialized nature: one cannot march forward to the future without knowledge about the past. Our hybrid scheme, on the other hand, bypasses the difficulty by utilizing large scaled neural network time-steppers, enabling vectorized computations (or parallel computations) at the bottom

Algorithm 2 Hybrid time-stepping scheme

Input: a set of neural network time-steppers $\hat{\boldsymbol{F}}s$

Output: a list of predicted states Xs sorted in chronological order

function HybridTimeStepping($\hat{\boldsymbol{F}}s$)
  $Sort(\hat{\boldsymbol{F}}s)$; (models are ordered with decreasing step sizes)
  $Xs = List()$; ▷ create an empty list
  // Use neural network time-steppers for large steps
  $Append(\boldsymbol{x}_0, Xs)$; ▷ append initial state to the list
  for i in $1, 2, \cdots, u$: **do** ▷ u is the number of HiTSs to use
    K_i := the number of steps forward with $\hat{\boldsymbol{F}}_i$;
    $X_{cur} = Stack(Xs)$; ▷ stack together all states in Xs
    $X_{next} = \hat{\boldsymbol{F}}_i.Forward(X_{cur}, \hat{\boldsymbol{F}}_i)$; ▷ step forward K_i steps with $\hat{\boldsymbol{F}}_i$ and X_{cur}
    $Append(X_{next}, Xs)$; ▷ update the list with new states
  end for
  // Use classic numerical time-steppers for fine steps
  K := the number of steps forward with runge-kutta time stepper;
  $X_{cur} = Stack(Xs)$; (stack together all states in Xs)
  $X_{next} = RK4(X_{cur}, K)$; ▷ step forward K steps with runge-kutta time-stepper and X_{cur}
  $Append(X_{next}, Xs)$; ▷ update the list with new states
  $Rearrange(Xs)$; **return** Xs
end function

level where we use classical numerical time-steppers for accurate simulations.

2.4 Numerical Experiments

We will now provide a thorough exploration of our proposed multiscale hierarchical time-steppers. We begin by comparing against single time-scale neural network time-steppers and Runge-Kutta algorithms on simple dynamical systems. We then explore this approach on more sophisticated examples in spatiotemporal physics and sequence generation.

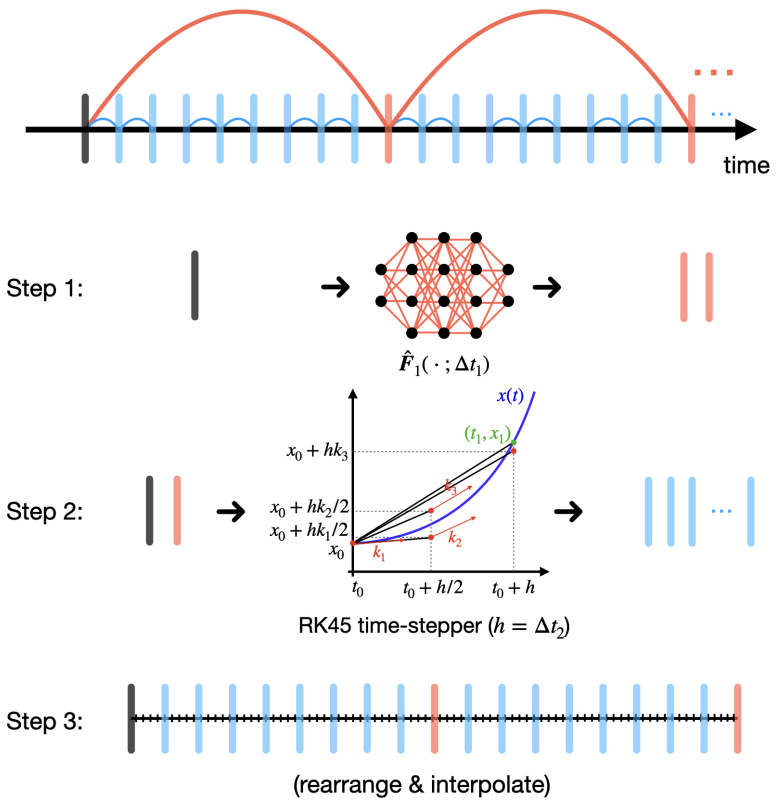

Figure 2.4: **Hybrid time-stepper**. A hierarchy of coarse neural network time-steppers generate states that are fed to a fourth order Runge-Kutta solver for fine-scale time-stepping.

2.4.1 Benchmark on time-stepping

We first benchmark the multiscale neural network HiTS against the single time-scale neural network time-steppers on five simple nonlinear dynamical systems: a nonlinear system with a hyperbolic fixed point, a damped cubic oscillator, the Van der Pol oscillator, a Hopf normal form, and the Lorenz system. For each example, we train 11 single time-scale neural network time-steppers with separate time steps, and then combine them into a multiscale neural network HiTS with the methods described in Section 2.3.3. More details about these numerical experiments can be found in Appendix A.1.1.

Fig. 2.5 shows that the multiscale scheme outperforms all single time-scale schemes in terms of accuracy, as shown by the black curve in the last column. On the sampled testing trajectory, in the third column, our multiscale scheme achieves nearly perfect time-stepping

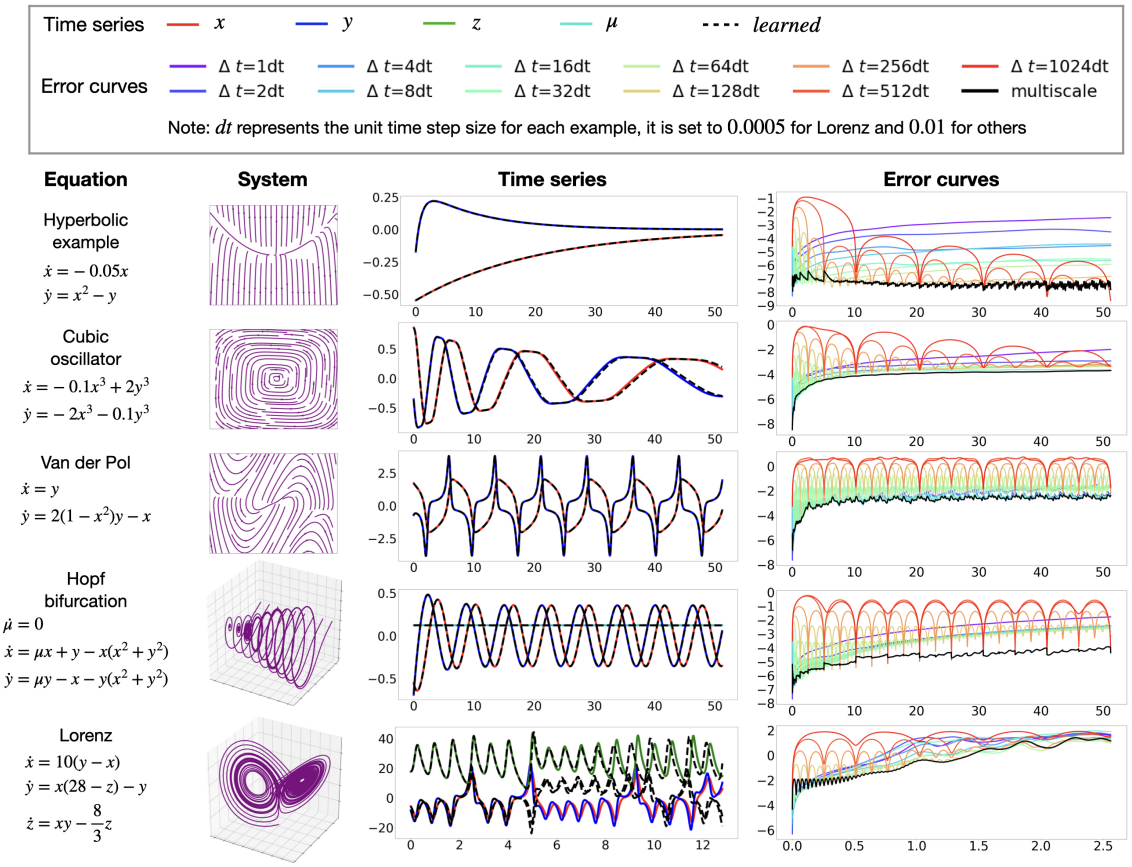

Figure 2.5: **Performance of multiscale HiTSs on nonlinear systems**. In the first two columns, systems of equations and phase portraits are visualized. In the third column, we visualize the predictions of multiscale time-steppers on a testing trajectory. In the last column, mean squared errors at each step are visualized in the base-10 logarithmic scale for different time-steppers.

for a time period of 51.20 for the first four nonlinear systems. For the Lorenz system, discrepancy occurs in the forecast after about 5 time units, when the trajectory switches lobes. Indeed, the error plot suggests the time-stepper becomes unreliable even after a single time unit, due to the intrinsic chaotic dynamics. The challenges of chaos are exacerbated by measurement noise on the data, as shown in the Appendix A, although there are many applications where low noise make this a viable approach, even for chaotic dynamics. Many works have shown success in predicting the Lorenz system; however, the tasks considered are

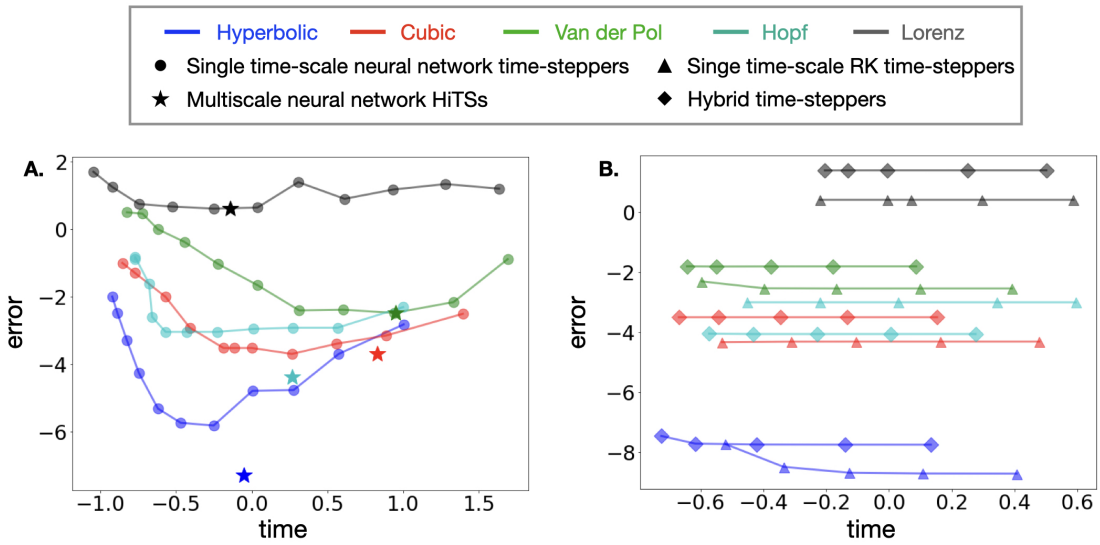

Figure 2.6: **Accuracy vs computational efficiency plot**. **A**. Comparison between multiscale neural network time-stepper and all single time-scale neural network time-steppers. **B**. Comparison between hybrid time-stepper and Runge-Kutta time-steppers with uniform step sizes. In both plots, horizontal and vertical axes represent time and integrated $\mathcal{L}_2$ error respectively, visualized in the base-10 logarithmic scale.

different from that explored in this work, and they usually fall into the following categories:

- The model is trained with data from a single trajectory and then tested with the same initial state [213, 207].

- The resulting model offers accurate short-term predictions while generating similar dynamical statistics for long times [38, 187].

- The model predicts the Lorenz system with regular adjustments or intermittent external forcing or under a data assimilation framework [32, 188].

In contrast, we train the model with batches of trajectories and test with unseen initial conditions; we also test long-term predictions without any external inputs. In fact, almost all previous models belong to single-scaled models in our discussion.

The trade-off between computational accuracy and efficiency are visualized in Fig. 2.6A. Here, we report the $\mathcal{L}_2$ error, averaged over all time steps and test trajectories. The single

time-scale scheme curves have a "U" shape for each example, indicating that accuracy first improves and then deteriorates as we spend more time in the computation. This finding is consistent with [207], which states that there is a problem-dependent sweet-spot for the hyperparameter Δt. Our multiscale HiTS always achieves the best accuracy, usually with a reasonable computational efficiency, due to the vectorized computations of array programming. For the cubic oscillator, Van der Pol oscillator, and Lorenz system, there seems to exist a single-scale neural network time-stepper with higher efficiency and competitive accuracy compared to our multiscale scheme. This is due to the greedy method we use for the cross validation process: we always prefer models with higher accuracy at the cost of efficiency. However, one can adjust the balance between accuracy and efficiency depending on the objective. The hyperparameter tuning of Δt is implicitly conducted in the cross validation process before coupling the various scales.

Though the multiscale scheme improves the computational efficiency of neural network time-steppers, they still cannot easily match the efficiency of most classic numerical algorithms (see Appendix A.2.3 for more details). Indeed, evaluating a neural network model (forward propagation) typically requires more computational effort than applying a classic discretization scheme that only involves a few evaluations of the known vector field. A hybrid time-stepper, on the other hand, may break this bottleneck by combining large-scale neural network time-steppers with classic numerical time-steppers to make the computations inherently parallelizable. Therefore, we benchmark the hybrid time-steppers against the classic numerical time-steppers in Fig. 2.6B. In particular, we use a fourth-order Runge–Kutta integrator, with 7 different sizes of uniform time step. For simplicity, the hybrid time-steppers are constructed from the same fourth-order Runge-Kutta (RK4) time-steppers with one large-scale neural network time-stepper at the highest level. More details can be found in Appendix A.1.2.

Fig. 2.6B shows the trade-off between accuracy and efficiency for our hybrid time-stepper and the classic RK4 scheme. Our hybrid time-stepper offers an efficiency gain over the Runge-Kutta time-stepper with the same minimal step size. This suggests the marginal benefits of enabling vectorized computation is potentially greater than the costs of evaluating a neural network time-stepper. But the accuracy of hybrid time-steppers are usually slightly

lower than the purely numerical time-steppers with rare exceptions (e.g., the Hopf normal form example), as the global error are usually dominated by the error of neural network time-steppers, which are model agnostic and purely data-driven, limiting their ability to produce high-fidelity simulations.

2.4.2 Benchmark on sequence generation

In addition to integrating simple low-dimensional dynamical systems, here we show that it is possible to forecast the state of more complex, high-dimensional dynamical systems. In the field of machine learning, this is often termed *sequence generation*. Importantly, we benchmark our architecture against state-of-the-art networks, including long short-term memory networks (LSTMs) [94], echo state networks (ESNs) [187], and clockwork recurrent neural networks (CW-RNNs). Here, our goal is to train different architectures that can generate the target sequence as accurately as possible. The sequences we explore include a simulated solution of the Kuramoto–Sivashinsky (KS) equation, a music excerpt from Bach's Fugue No. 1 In C Major, BWV 846, a simulation of fluid flow past a circular cylinder at Reynolds number 100 [233, 50], and a video frame of blooming flowers. Within each individual experiment, the various architectures have nearly the same number of parameters; more details about the data preprocessing and choice of parameters are described in Appendix A.1.3.

From Fig. 2.7, one can visually see that the multiscale HiTS provides the best sequence generation results, and these results are confirmed by the integrated $\mathcal{L}_2$ errors shown in Table 2.1. We also see that the LSTM and CW-RNN can learn the first few steps accurately, whereas the ESN tends to smooth the signals. It should be noted that our sequence generation task are very different from the tasks considered in [186], where they use observations of the system's past evolution to predict future states. An ESN is usually trained by finding a set of output weights through linear regression, if the reservoir is large enough, it can in principle reproduce the observed dynamics perfectly, which would make it an inappropriate benchmark. An ESN with the same number of parameters as the other architectures often cannot fully describe the dynamics, resulting in coarse or smoothed approximations.

In a nutshell, the competing architectures fail to capture the long-time behaviors, as error

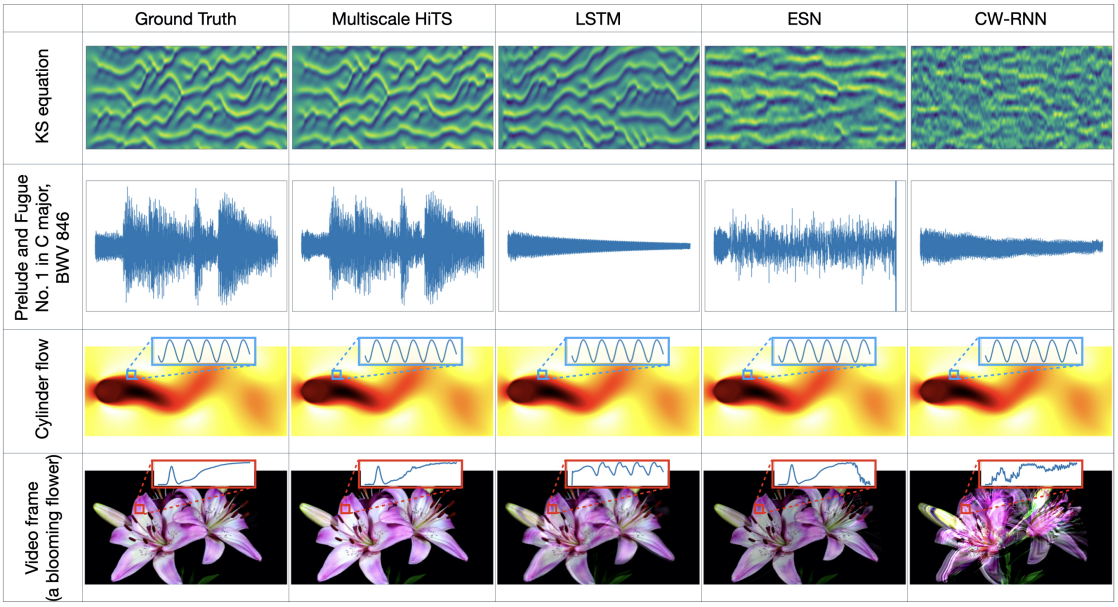

Figure 2.7: **Outputs of different network architectures (column) on each training sequence (row)**. We use different visualization schemes to show the results: for the KS equation and the music excerpt, we plot the time series evolution, that is, the horizontal axes represent time; For the cylinder flow and the video frame, since each state is a 2D array, we choose to visualize the last frame of our reconstruction, however, we also visualize the time evolution of some states averaged over a small patch of pixel values. For a video that shows the performance, visit: `https://youtu.be/2psX5efLhCE`.

accumulation is inevitable for serialized computations. However, our proposed framework should not be viewed as a replacement for these state-of-the-art methods, as they take a different approach and philosophy for sequence generation. RNNs tend to uncover the full dynamics with the help of memory in their internal states, whereas our scheme performs reconstruction using the time-stepping schemes learned at different time scales, though these schemes may only be accurate for a few steps. Instead, our multiscale framework should be used to strengthen these existing approaches, as it can utilize data-driven models across different scales, avoiding local error accumulations and potentially boosting the accuracy and efficiency.

Sequences	HiTS	LSTM	ESN	CW-RNN
Fluid flow	1.23e-7	9.20e-8	2.22e-8	6.79e-7
Video frame	7.09e-5	1.44e-2	1.62e-3	8.05e-2
Music data	8.59e-7	4.65e-5	2.69e-4	4.70e-5
KS equation	1.04e-3	3.50e+0	3.51e+0	3.59e+0

Table 2.1: The integrated $\mathcal{L}_2$ error between the generated sequence and the exact sequence.

2.5 Conclusions & Discussion

In this work, we have demonstrated an effective and general data-driven time-stepper framework based on synthesizing multiple deep neural networks for hierarchical time-steppers (HiTSs) trained at multiple temporal scales. Our approach outperforms neural networks trained at a single scale, providing an accurate and flexible approach for integrating nonlinear dynamical systems. We have carefully explored this multiscale HiTS approach on several illustrative dynamical systems as well as for a number of challenging high-dimensional problems in sequence generation. In the sequence generation examples, our approach outperforms state-of-the-art neural network architectures, including LSTMs, ESNs, and CW-RNNs. Our method explicitly takes advantage of dynamics on different scales by learning flow-maps for those different scales. The coupled model still maintains computational efficiency thanks to the vectorized computations of array programming. Moreover, exactly due to this coupling scheme, each individual network can focus on their intrinsic ranges of interest, bypassing the exploding/vanishing gradient problem for training recurrent neural networks. In addition, we demonstrate the joint use of our neural network time-steppers with the classical time-steppers, resulting in a new computational paradigm: numerical simulation algorithms are now parallelizable rather than serialized in nature, leading to performance boosts in computational speed.

This work highlights fundamental differences between physics-based simulation models and data-driven models. In the former, the error of the time-stepping constraints are determined strictly by Taylor series expansions which are local in nature and limit the time-step

Δt. The latter is a more general flow-map construction that can be trained for any time step Δt. Thus the error is not limited by a local Taylor expansion, but rather by pairs of training data mapping the solution to a future Δt. In the end, we show our proposed scheme is capable of learning long-term dependencies on some more realistic data sets, achieving state-of-the-art performance.

However, some cautionary remarks are warranted: as with all DNN architectures, obtaining reliable large scaled time-steppers comes at the cost of significant training because of the increasing complexity (see Appendix A.2.4). Specifically, one usually needs to acquire large enough data sets to train the appropriate deep NN architecture. Deriving a rigorous *a priori* error bound using neural networks is rather involved compared to traditional time-stepping algorithms [150]. Deriving error bounds requires a rigorous formulation of the function space and introduction of an appropriate norm. One remedy for this is to evaluate the trained model on some unseen test trajectories so that the generalization error can be empirically controlled before putting into practical use. Lastly, we have assumed full observation of the system states, although full-state measurements are often unavailable in real-world applications. When the system states are partially observed, techniques such as time-delay embedding [32, 4, 109, 182] may be applied as a preprocessing step to address the presence of latent variables, which is related to the closure problem.

This work also suggests a number of open questions that motivate further investigation. In particular, nearly every sub-field within numerical integration can be revisited from this perspective. For example, bringing the idea of adaptive step sizes into this framework may potentially lead to even more efficient and accurate time-stepping schemes. Although this important element may be naturally addressed, it is not yet built in to the proposed methodology in its present form. In addition, as mentioned in Appendix A.2.4, the increments of the flow maps mostly exhibit obvious multiscale features. Since deep learning is essentially an interpolation method [156], to leverage more effective training, new sampling strategies may be proposed to address this problem [60, 28]. This is of crucial importance for deep learning, as neural networks are data-hungry and the curse of dimensionality makes it impossible to generate largely enough data sets for high dimensional systems. It is also important to introduce principled approaches to deal with uncertainty quantification within

this framework, as real-world systems may be stochastically forced. Similarly, models at different scales may have different associated levels of confidence, which the current multiscale modelling approach does not address. These are both important areas of ongoing research. Bayesian statistical techniques provide one potential approach to enhance the functionality of these methods, by making distribution, rather than point, forecasts.

Chapter 3

MULTIRESOLUTION CONVOLUTIONAL AUTOENCODER

3.1 abstract

We propose a *multi-resolution convolutional autoencoder* (MrCAE) architecture that integrates and leverages three highly successful mathematical architectures: (i) multigrid methods, (ii) convolutional autoencoders and (iii) transfer learning. The method provides an adaptive, hierarchical architecture that capitalizes on a progressive training approach for multiscale spatio-temporal data. This framework allows for inputs across multiple scales: starting from a compact (small number of weights) network architecture and low-resolution data, our network progressively deepens and widens itself in a principled manner to encode new information in the higher resolution data based on its current performance of reconstruction. Basic transfer learning techniques are applied to ensure information learned from previous training steps can be rapidly transferred to the larger network. As a result, the network can dynamically capture different scaled features at different depths of the network. The performance gains of this adaptive multiscale architecture are illustrated through a sequence of numerical experiments on synthetic examples and real-world spatial-temporal data.

3.2 Introduction

The multiscale spatio-temporal dynamics observed in many complex systems poses significant challenges for modeling and prediction. Although we are often primarily interested in macroscale phenomena, the microscale dynamics must also be modeled and understood, as it plays an important role in driving the macroscale behavior. Indeed, a given system may have multiple fast and slow time scales as well as a number of micro to macro spatial scales that interact to produce the complex dynamics observed. This makes modeling multiscale systems particularly difficult unless the time scales are disambiguated in a principled way.

Coarse graining and mesh-refinement (multigrid methods) are two principled mathematical techniques for addressing multiscale behavior. In the former, the microscale physics are averaged over to produce an effective macroscale model, while in the latter, computational models are refined where the macroscale variables produce large errors. In this paper, we present a *multi-resolution convolutional autoencoder* (MrCAE), a decomposition scheme inspired by multigrid computational methods to progressively refine a multiscale description of spatio-temporal data. It leverages and exploits aspects of multigrid methods and transfer learning to produce an effective multi-scale analysis tool for characterizing large scale spatio-temporal data. The multiscale spatio-temporal dynamics observed in many complex systems poses significant challenges for modeling and prediction. Although we are often primarily interested in macroscale phenomena, the microscale dynamics must also be modeled and understood, as it plays an important role in driving the macroscale behavior. Indeed, a given system may have multiple fast and slow time scales as well as a number of micro to macro spatial scales that interact to produce the complex dynamics observed. This makes modeling multiscale systems particularly difficult unless the time scales are disambiguated in a principled way. Coarse graining and mesh-refinement (multigrid methods) are two principled mathematical techniques for addressing multiscale behavior. In the former, the microscale physics are averaged over to produce an effective macroscale model, while in the latter, computational models are refined where the macroscale variables produce large errors. In this paper, we present a *multi-resolution convolutional autoencoder* (MrCAE), a decomposition scheme inspired by multigrid computational methods to progressively refine a multiscale description of spatio-temporal data. It leverages and exploits aspects of multigrid methods and transfer learning to produce an effective multi-scale analysis tool for characterizing large scale spatio-temporal data.

Multigrid methods [163, 244] have been extensively developed for physics-based simulation models where coarse grained models must be progressively refined in order to achieve a required numerical precision while keeping the simulation tractable. Multigrid architectures provide a principled method for targeting the refinement process, constituting a mature field with wide spread applications in the engineering and physical sciences. In contrast, coarse graining methods attempt to construct a macroscale physics model by progressive construc-

tion of coarse grained variables and their dynamics. Mathematical algorithms such as the *heterogeneous multiscale modeling* (HMM) [261, 257] and *equation-free method* [115] provide principled methods for multiscale systems. Additional work has focused on testing for the presence of multiscale dynamics so that analyzing and simulating multiscale systems is more computationally efficient [67, 69].

Data-driven methods, specifically neural networks (NNs), have emerged as an attractive alternative for characterizing multi-scale physics [75, 271, 255, 161, 149, 41, 84, 200, 209]. The structure of *convolutional neural networks* (CNNs) are especially relevant for multiscale data as the convolutional window extracts features of the data at an appropriate, coarse- or fine-grained resolution. As a result, CNNs have demonstrated exceptional performance in image processing tasks [76]. Indeed, in the wake of AlexNet [121], many aspects of CNN design have been thoroughly studied individually, such as the spatial filters [231, 227], non-linear activation functions [269], width and depth of the network [276], skip connections [86], batch normalization [98], etc [3]. As more complex neural architectures and designs arise, NNs have become increasingly popular, which enables the process of automating architecture engineering by jointly considering all design factors [237, 138, 281]. However, the design choices of NN algorithms are highly automated and often uninterpretable, which makes integrating domain knowledge difficult. This motivates the innovations of the current work; here, we propose a multi-resolution convolutional autoencoder (MrCAE) that begins by processing on downsampled (ie "coarse") data in order to capture large-scale, low-dimensional structure, and then progressively refines both the data and the neural network while employing transfer learning in building each stage's neural network. This progress-refinement approach provides a principled framework for leveraging multiresolution and transfer learning ideas and an in-depth understanding over modern end-to-end architectures, enabling one to build multiscale models that result in more compact networks and more compact encodings than those produced by traditional CAE methods.

Importantly, our architecture can be used to leverage data in an intelligent way: querying data that targets the refinement process. Thus a hierarchy of models is constructed with less data, producing highly compact networks and encodings. The method is well aligned with the arguments in [46]: *'Initially, a small model may be preferred, in order to prevent*

overfitting and to reduce the computational cost of using the model. Later, a large model may be necessary to fully utilize the large dataset.'.

Many encoder-decoder based models have been proposed for physical applications due to their ability to impose physics-based constraints. This includes the linearization of the dynamics in order to construct Koopman embeddings [149, 73], dimensionality reduction for parsimonious model discovery via sparse regression [41], and forecasting [213], for instance. In these applications, encoder-decoder architectures are used to project data into some new coordinate systems which leverage physical constraints of the intrinsic dimension of the dynamical evolution. Here, we use it to capitalize on the spatial, multi-scale features manifest in many complex systems. So we envision our proposed architecture to be a critical piece of many future multi-scale neural network architectures. Additionally, our work is intimately connected to reduced order modeling [16, 123, 36, 199, 90, 234] whose primary goal is to exploit tailored low-rank features extracted from training data, rather than represent dynamics in terms of some universal basis.

Our paper is organized as follows: the multi-resolution network architecture is proposed in Section 3.3, the progressive training algorithm is presented in Section 3.4, experimental results are in Section 3.5, and the conclusions and discussions follow in Section 3.6. Our code is publicly available at `https://github.com/luckystarufo/MrCAE`.

3.3 Network Architecture

Section 3.3 details the overall architecture of our MrCAE. It consists of several levels of autoencoders. Each level is associated with the data of a particular resolution. The network is built up recursively: in the base level, a small autoencoder is trained to encode and reconstruct the coarsest data. We then refine the data so that our inputs are of a higher resolution and embed the existing architecture into a new autoencoder (the transfer learning step) to learn the new, refined data. In this way, we construct a hierarchy of trained neural networks. The mathematical details of the construction are given in the subsections that follow.

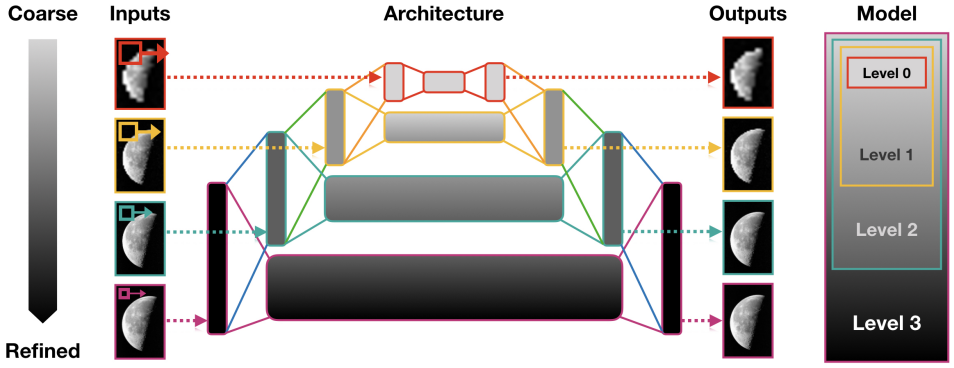

Figure 3.1: **A Schematic Overview of the Network Architecture**. In this example, there are 4 levels for the architecture which are colored in red, yellow, cyan and purple respectively. They are recursively built up to process data across different resolutions — architectures built for processing coarser data are later embedded into the next-level architectures to ensure knowledge transfer.

3.3.1 Hierarchical Construction

The technical details for constructing each level of the network is demonstrated in Section 3.3: we perform a deepening operation (except for the base level) followed by a sequence of widening operations.

The *deepening operation* inserts a convolutional/deconvolutional layer between the current and previous level inputs/outputs. This is the transfer learning step [1]. We denote the inserted convolutional filter as $c^{(k)}$ and the deconvolutional filter as $d^{(k)}$, and let $f_{\boldsymbol{\theta_{k-1}}}$ be the existing network at level $(k-1)$, $g^{(k)}$ be the network after we apply this deepening operation. Mathematically, we have

$$g^{(k)} = d^{(k)} \circ f_{\boldsymbol{\theta_{k-1}}} \circ c^{(k)} \tag{3.1}$$

In our setup, we make the resolution differ by a factor of 2 along each spatial dimension between two adjacent levels, which is typical of a multi-resolution analysis [123]. As a

[1] Transfer learning refers to the idea of using trained results from a task to improve the learning of another task. Here, we are using the trained results of reconstructing coarse-grained data to benefit the training on higher resolution data.

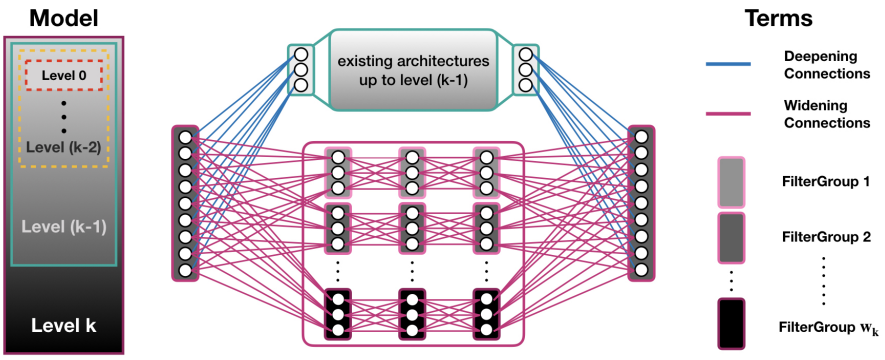

Figure 3.2: **Build up Network Architecture at Level** k. Within level k, we perform one deepening operation and a sequence of widening operations. The deepening operation (shown in blue) is the transfer learning step, creating a convolutional filter that connects the new input (fine) to the previous level input (coarse). Widening operations (shown in purple) are performed sequentially by allocating more convolutional filters so that it can capture new, higher-resolution features.

result, convolutional/deconvolutional filters with a kernel size 3×3 and stride size 2 are used to ensure compatible dimensions. Similar to [46], the two inserted layers are properly initialized to ensure knowledge transfer. More technical details about the initialization are covered in the Appendix B.1. There is another, perhaps more intuitive, way to understand the design: data at coarser levels can be regarded as obtained by applying a sequence of hard-coded down-sampling (convolutional) operations to the finest data and vice versa. When we proceed to a new level, the network replaces one of these hard-coded operators with trainable convolutional/deconvolutional filters.

The *widening operation* expands the network capacity in order to capture the new, finer-grained features of the higher resolution data. Specifically, it adds a group of new convolutional pathways through the network, and then continues training. See the Filter-Groups in Figure 3.3. Note that each new pathway is separate, not adding any connections to old pathways; this reduces the risks of overfitting. Mathematically speaking, the widen-

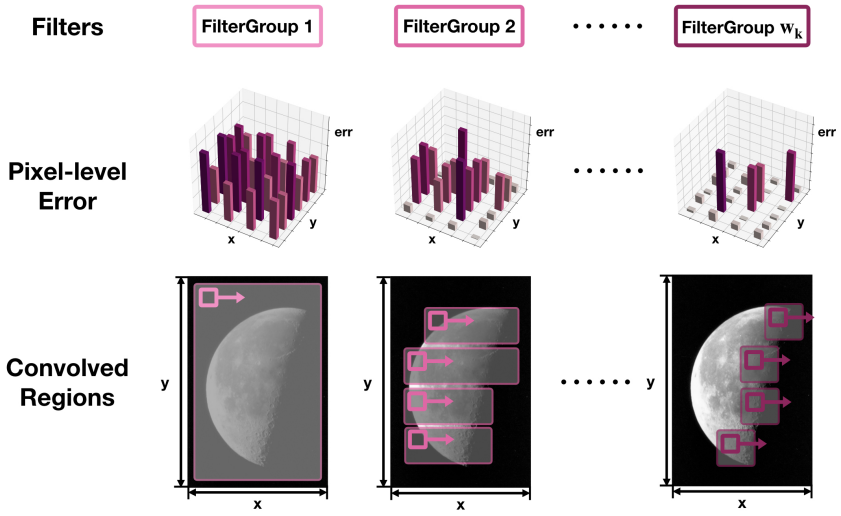

Figure 3.3: **Adaptive Filters (I)**. For each widening operation shown in Section 3.3, new convolutional filters are only applied to the regions that are still poorly resolved. The progressive refinement ensures a parsimonious use of the parameters.

ing operations at level k create a list of new pathways for the network, we denote them as $w_1^{(k)}, w_2^{(k)}, \cdots, w_{k_n}^{(k)}$. Then we have,

$$f_{\boldsymbol{\theta_k}} = g^{(k)} + w_1^{(k)} + \cdots + w_{w_k}^{(k)}. \tag{3.2}$$

3.3.2 Adaptive Filters

The way we perform a sequence of widening operations is shown in Section 3.3. Suppose we have performed j groups of widening operations and finished the subsequent training. We proceed by computing the mean squared error of a 3×3 region around each pixel. Our $(j+1)^{th}$ group of filters will be only applied to the regions where the reconstructions exhibit high error; details are explained in Section 3.4. Then a subsequent training is performed for the new architecture. We do this recursively until all regions can be reconstructed within some error threshold/bound. This process bears a close resemblance to mesh refinement [19] of multigrid methods [163, 244] and therefore is highly adaptive and ensures a parsimonious use of parameters.

It is worth noting that we do not use nonlinear activation functions, in contrast to the classical designs in which convolutional filters are usually followed by a Rectified Linear Units (ReLU). We find it actually gives us a performance boost in terms of the training progress while also making the filters highly interpretable. In our design, the highly adaptive filters play the role of the nonlinear activation functions: by explicitly specifying where to apply the filters, we are forcing those filters to be activated in the relevant regions while leaving them inhibited in other regions. Additionally, this implies that the resulting, trained network is fully linear, enabling extremely fast matrix-based representations.

3.4 Training

3.4.1 Framework

To accompany our adaptive architecture, we have developed a progressive training framework which is outlined in Algorithm 3. The training is performed whenever the architecture changes, thus it can effectively utilize the new pathways for representing new data. Early stopping criteria are implemented as well, however, we reference the full training details in Appendix B.2 in order to maintain clarity and simplicity for the description of the architecture.

3.4.2 Loss Function

Our loss function is designed to capture the general, low-rank dynamics as well as identifying outlier/singularities. It is of the same functional form across each level. However, different resolution data are used to calculate this quantity. Suppose we are training the network at level k ($0 \leq k < N$) with data set $D^{(k)} \in R^{m_k \times n_k \times T_k}$ (m_k, n_k, T_k are the width, height of each snapshot and the number of snapshots), and $\widehat{D}^{(k)} := f_{\boldsymbol{\theta_k}}(D^{(k)})$ is the corresponding reconstruction through the network, where $\boldsymbol{\theta_k}$ are the parameters of the current network $f_{\boldsymbol{\theta_k}}$ at level k[2]. Let i, j, t to be the indices of the row, column and snapshot. We formulate

[2]In fact, at level k, the network keeps growing itself as we perform the deepening operation and the sequence of widening operations, which gives rise to different networks g_k, $f^{(1)}_{\boldsymbol{\theta_k}}, \cdots, f^{(k_n)}_{\boldsymbol{\theta_k}}$ respectively. Here, without causing ambiguity, we refer them all as $f_{\boldsymbol{\theta_k}}$.

Algorithm 3 Progressive Training

Define N to be the number of levels

Define E to be the maximum training epochs

Define training data with increasing resolutions $\{D^{(0)}, \cdots, D^{(N-1)}\}$

Initialize a model object M

for i in $0, 1, \cdots, N-1$: **do**

  Perform deepening operation on M;

  Perform E epochs training;

  while Reconstruction at this level is not fully resolved **do**

    Perform widening operation on M;

    Perform E epochs training;

  end while

end forreturn M

our loss function $\mathcal{L}(\boldsymbol{\theta_k}, D^{(k)})$ to be the following:

$$\mathcal{L}(\boldsymbol{\theta_k}, D^{(k)}) = \omega \mathcal{L}_{mse}(\boldsymbol{\theta_k}, D^{(k)}) + (1-\omega)\mathcal{L}_{max}(\boldsymbol{\theta_k}, D^{(k)}) \tag{3.3}$$

where

$$\mathcal{L}_{mse}(\boldsymbol{\theta_k}, D^{(k)}) = \frac{1}{m_k n_k T_k} \sum_{i=1}^{m_k} \sum_{j=1}^{n_k} \sum_{t=1}^{T_k} (D_{i,j,t}^{(k)} - \widehat{D}_{i,j,t}^{(k)})^2 \tag{3.4}$$

$$\mathcal{L}_{max}(\boldsymbol{\theta_k}, D^{(k)}) = \max_{i,j} \frac{1}{T_k} \sum_{t=1}^{T_k} (D_{i,j,t}^{(k)} - \widehat{D}_{i,j,t}^{(k)})^2 \tag{3.5}$$

The first term in Eq. (3.3) is the classic mean squared error (MSE) to ensure an overall satisfactory reconstruction. The second term captures the worst case scenario which primarily corresponds to the highly variable dynamics over some regions in the spatio-temporal data. The loss function is a weighted sum of these two terms modulated by a control parameter ω ($0 \leq \omega \leq 1$). Based on our experience, we find setting $\omega = 0.5$ is suitable for dynamics that have a clear separation of spatial scales, so that the network pays more attention to the singularities or highly variable regions. But for those without persistent spatial patterns, we recommend setting ω close to 1.

3.4.3 Measuring Error

In addition to the loss function in Eq. (3.3), we define two other important quantities. The first quantity is developed for tracking the overall training progress. The loss function in Eq. (3.3) only gives us the information of how it performs at a specific level. Nothing can be said about the reconstructions at higher levels. In this case, knowledge transfer between adjacent levels cannot be evaluated. What we need is a metric that reflects the overall training progress. Therefore, we define a metric that estimates the reconstruction error with respect to the finest resolution data at each level:

$$\mathcal{L}^{global}(\boldsymbol{\theta}, D^{(N-1)}) = \omega \mathcal{L}^{global}_{mse}(\boldsymbol{\theta}, D^{(N-1)}) + (1-\omega)\mathcal{L}^{global}_{max}(\boldsymbol{\theta}, D^{(N-1)}) \tag{3.6}$$

$$\mathcal{L}^{global}_{mse} = \frac{1}{m_{N-1}n_{N-1}T_{N-1}} \sum_{i=1}^{m_{N-1}} \sum_{j=1}^{n_{N-1}} \sum_{t=1}^{T_{N-1}} (D^{(N-1)}_{i,j,t} - \mathcal{U}^{(N-k-1)}(\widehat{D}^{(k)}_{i,j,t}))^2 \tag{3.7}$$

$$\mathcal{L}^{global}_{max} = \max_{i,j} \frac{1}{T_{N-1}} \sum_{t=1}^{T_{N-1}} (D^{(N-1)}_{i,j,t} - \mathcal{U}^{(N-k-1)}(\widehat{D}^{(k)}_{i,j,t}))^2 \tag{3.8}$$

where

$$\mathcal{U}^{(s)} = \underbrace{\mathcal{U} \circ \mathcal{U} \circ \cdots \circ \mathcal{U}}_{s} \tag{3.9}$$

is a composition of a sequence of up-sampling operation $\mathcal{U}$ which is chosen to be a bi-linear interpolation with a kernel size 3×3 and a stride size 2. This is similar to the prolongation operator from multigrid methods. It should be noted that this metric is only proposed for the purpose of validating the effectiveness of knowledge transfer across different levels of the model, it would be absent in the realistic settings where we have a growing data set so that the finest data $D^{(N-1)}$ may not be accessible initially.

The other metric is used to provide feedback for the adaptive filters. Within a widening operation, the new group of convolutional filters are only applied to the regions that still need to be further refined. Technically, this is achieved by applying the filters to all regions of the inputs first (and therefore obtaining the convolved features), then determining the irrelevant ones. To tell if a convolved feature should be masked out or not, we first calculate the mean squared error of each pixel along the time axis, followed by performing a down-sampling operation $\mathcal{A}$ (local average) since the width and height of the convolved features

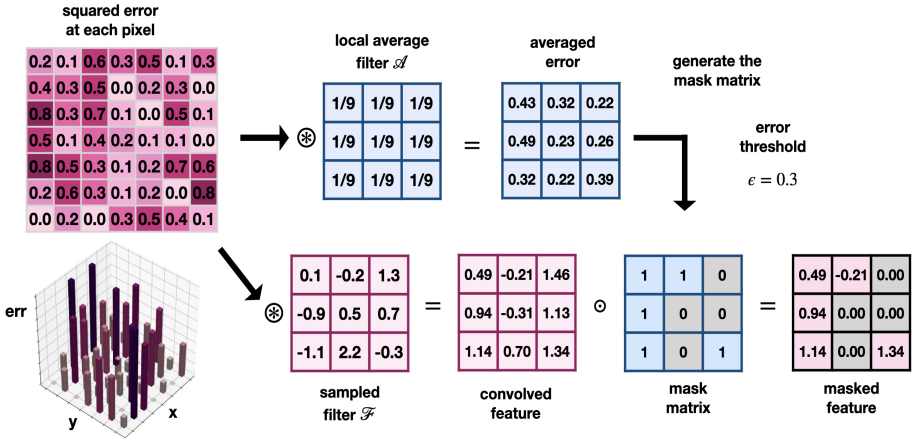

Figure 3.4: **Adaptive Filters (II)**. This picture illustrates the implementation details of applying adaptive filters. It is done by calculating the mask (shown at the top) and convolved features of all regions (shown at the bottom) followed by a point-wise multiplication.

are both reduced by a factor of 2 compared to the original inputs. Finally, we threshold it by a prescribed tolerance ϵ to obtain the mask $\mathcal{M}$:

$$\mathcal{M} = \mathbb{1}\Big\{\mathcal{A}\Big[\frac{1}{T_k}\sum_{t=1}^{T_k}(D^{(k)}_{:,:,t} - \widehat{D}^{(k)}_{:,:,t})^2\Big]_{i,j} \geq \epsilon\Big\} \tag{3.10}$$

The process is illustrated in Section 3.4.3.

3.5 Experimental results

In this section, we test the performance of our MrCAE on a range of data displaying increasing levels of complexity. Several unsteady fluid flow fields are analyzed, as fluid dynamics exhibit a range of multiscale behavior and have been the focus of intense modeling efforts, with and without machine learning [234, 37]. In the first part, we show the performance of our network on each individual case. Specifically, we show the reconstructions on some testing snapshots and heat maps of the regions being refined across each level and the error plot with respect to Eq. (3.6) that reflects the overall training progress. In the second part, we benchmark our network against a structurally similar network: residual encoder-decoder

network (RED-Net[160]). We show our architecture scales better in terms of the number of network parameters and the size of encoding.

All data are resized so that each snapshot has the shape $(2^p - 1) \times (2^q - 1)$ (with some positive integers p and q) before data ingestion into the network, all snapshots are shuffled randomly and split in to training set (70%), validation set (20%) and testing set (10%).

3.5.1 Individual experiments

Two oscillatory modes

For the first example (Section 3.5.1), we consider two nonlinear spatial modes driven by sinusoidal temporal modes with different frequencies:

$$\Phi(x, y, t) = u(x, y) \cos(\omega_0 t) + v(x, y) \cos(\omega_1 t + \frac{\pi}{4}) \tag{3.11}$$

where

$$\begin{aligned} u(x, y) &= \cosh(\frac{x+1}{\sigma_0}) \cosh(\frac{y-1}{\sigma_0}) \\ v(x, y) &= \frac{1}{(2\pi\sigma_1)^{1/2}} \exp(-\frac{(x-1)^2 + (y+1)^2}{2\sigma_1^2}) \end{aligned} \tag{3.12}$$

We generate it in the domain of $[-5, 5] \times [-5, 5]$ using 127×127 grids, and 500 snapshots are uniformly collected from the time interval $[0, 8\pi]$. We set $\omega_0 = 0.5$, $\omega_1 = 4.0$, $\sigma_0 = 10.0$ and $\sigma_1 = 0.25$ in our experiment.

Section 3.5.1a shows the synthetic spatio-temporal data. We visualize the two different spatial modes and their associated temporal modes. 6 sampled snapshots are drawn from the test set which are marked in red. This is an example where small spatial (high-frequency) content is modulated by a large, slow-varying background mode.

For this example, we construct a network of 3 levels, with each level having 1, 2 and 3 groups of adaptive filters (a.k.a. widening operations) correspondingly. The output reconstructions across different levels shown in Section 3.5.1b become sharper and sharper as the network grows. The regions that adaptive filters being applied are shown in Section 3.5.1c. One can clearly see the network faithfully picks up the high-frequency content with more convolutional filters while leaving other regions less parametrized which is consistent with

our intuition. We can also see the proposed error metric on the validation set keeps decreasing throughout the training process except for the starting phase of each operation, which is the effects of random initialization, suggesting an effective knowledge transfer.

Two oscillatory modes with one drifting

For the second example, we also consider two nonlinear modes driven by different sinusoidal temporal modes. However, only one of the nonlinear modes is purely spatial, the other one is modulated by time. This synthetic example is meant to replicate the effects of traveling waves or the drifting dynamics that usually appears in the spatial-temporal data. Mathematically, we have:

$$\Phi(x, y, t) = u(x)\cos(\omega_0 t) + v(x, t)\cos(\omega_1 t + \frac{\pi}{4}) \tag{3.13}$$

where

$$\begin{aligned} u(x, y) &= \cosh(\frac{x+1}{\sigma_0})\cosh(\frac{y-1}{\sigma_0}) \\ v(x, y, t) &= \frac{1}{(2\pi\sigma_1)^{1/2}}\exp(-\frac{(x-3+0.5t)^2 + (y+3-0.5t)^2}{2\sigma_1^2}) \end{aligned} \tag{3.14}$$

We generate the data in the domain of $[-5, 5] \times [-5, 5]$ using 127×127 grids, and 500 snapshots are uniformly collected from the time interval $[0, 4\pi]$. We set $\omega_0 = 0.5$, $\omega_1 = 4.0$, $\sigma_0 = 10.0$, $\sigma_1 = 0.25$ and $v = 1.0$ in our experiment. Section 3.5.1 shows the two spatial-temporal modes, sampled test snapshots, reconstructions across different levels, refined regions and the logarithmic error plot.

In this example, we construct a network of 3 levels, with 3, 3 and 3 groups of adaptive filters applied on each level. Similarly, we see the network keeps refining itself by growing new pathways, our adaptive filters successfully capture the high-frequency contents drifting along the diagonal and knowledge learned in the previous level can be successfully transferred to the next level.

Channel flow

The channel flow dataset is acquired from the Johns Hopkins Turbulence Database [136, 190, 77]. Data was generated by solving the Navier-Stokes equations in a domain of size

$8\pi \times 2 \times 3\pi$ using $2048 \times 512 \times 1536$ nodes. For more details, refer to [110]. We sample the slice of $z = 1.5\pi$ and down-sample the other two spatial dimensions both by a factor of 8, so that the spatial dimensions of our snapshots are 255×63. 500 snapshots are collected with $\Delta t = 0.052$ between each snapshot.

In this example, we set up a network of 4 levels with each level having 0, 3, 3, 4 groups of adaptive filters. (At level 0, no adaptive filters are used because all pixels can be well-reconstructed with the deepening operation alone.) Section 3.5.1 shows the increasingly refined outputs at each level and the heat maps of regions that the adaptive filters apply. Notably, we see from the heat maps that the network spends more effort processing the regions near the boundary which intuitively makes sense: there are more detailed small-scale vortical behaviors arising within these regions. The plot of the decreasing error on the validation set also justifies the knowledge transfer of each operation.

Forced isotropic turbulence

The forced isotropic turbulence data set is also acquired from the Johns Hopkins Turbulence Database [136, 190, 274]. The data is generated from a direct numerical simulation of forced isotropic turbulence on a $1024 \times 1024 \times 1024$ periodic grid, using a pseudo-spectral parallel code. The range of spatial dimensions x, y and z are $[0, 2\pi]$. We sample the slice of $z = \pi$ and down-sample the other two spatial dimensions both by a factor of 8 so that each snapshot is of size 127×127. 503 snapshots are collected with $\Delta t = 0.02$ between each snapshot. This is an example without a clear separation of scales, although it still possesses rich microscopic dynamics across all regions.

For this example, we set up a network of 4 levels with each level having 0, 4, 4, 4 groups of adaptive filters. (At level 0, no adaptive filters are used because all pixels can be well-reconstructed with the deepening operation alone.) As shown in Section 3.5.1, the network still offers reasonably well and progressively refined reconstructions of sampled test snapshots and the error plot also shows the effectiveness in knowledge transfer. But the heat maps of refined regions do not exhibit clear spatial patterns due to the isotropic nature of the data. In this case, one should be very cautious when attempting to use the

hierarchical representations encoded in the network since neural networks are fundamentally interpolation methods [156] and lacking of interpretations often suggests the representations are hard to generalize.

Sea surface temperature

In this example, we consider the global sea surface temperature (SST) data. The NOAA OI SST V2 data set is open source and can be downloaded at `http://www.esrl.noaa.gov/psd/`. The data is collected each month and spans a 20 year period from 1990 to 2010. In this example, we set up a network of 4 levels with each level having 2, 2, 4, 4 groups of adaptive filters. Section 3.5.1 shows the increasingly refined reconstructions at higher and higher levels and the heat maps of regions where the adaptive filters apply. One observation is that network explores the middle regions more exhaustively. We conjecture that the underlying reason could be the temperature at the North Pole (top) and the South Pole (bottom) are less variant. In addition, one can see the boundaries of the highly intensive regions align well with edges of the continents which suggests the transition from the land to the ocean.

3.5.2 Benchmarks

We benchmark MrCAE against the RED-Net[160], which is structurally similar to ours. We run experiments of the network with and without rectified nonlinear units (ReLU) which results in two candidate comparisons: RED-Net(Linear) and RED-Net(+ReLU). We also include our MrCAE without adaptive filters, but equipped with the rectified nonlinear units, which is named MrCAE(+ReLU).

Although RED-Nets were not primarily proposed to do progressive training with data of increasing spatio-temporal resolution, for fair comparison, we also use different resolution data to train them and use the metric in Section 3.4.3 to evaluate their performance.

The procedure is as follows: throughout the training of our MrCAE network, we obtain a sequence of network architectures at different hierarchical levels. For each specific architecture, we train a corresponding one (with the same network depth and the same number

of filters across each layer) for all other networks used for comparison. The key differences among the four candidate networks are as follows:

- **number of parameters**: MrCAEs have sparse connections which requires less parameters whereas connections in RED-Nets are dense. This is due to the fact that different group of filters are independently patched in MrCAEs.

- **size of encoding**: MrCAE(PR) explicitly utilizes a sparse coding scheme whereas the other three types of networks don't have this feature.

Results are shown in Section 3.5.2 and Section 3.5.2. Axes in both plots are in logarithmic scales.

Number of parameters

In Section 3.5.2, we show that MrCAEs use significantly fewer parameters in comparison with similar architectures of RED-Nets. In other words, due to the sparse connections, it scales better when the network grows. Moreover, the error curves for MrCAEs seem to steadily decrease as the network grows which is highly desirable. But for RED-Nets, the curves are quite variable — more parameters does not necessarily offer better results. And despite the parsimonious use of parameters, MrCAEs can achieve reasonable accuracy and sometimes, in the toy examples where the dynamics are not that complex, outperform other networks. We conjecture the reasons are of two folds: First, this spatio-temporal data has a clear separation of scales and our adaptive filters are primarily designed to efficiently leverage this feature. And second, the progressive training process may have some guiding effects for the network parameters which lowers the risks of getting stuck in some local minima.

Size of encoding

In Section 3.5.2, we show the validation error versus the size of encoding. By utilizing the adaptive filters, we are able to hierarchically encode different regions: some filters are only applied to highly variable regions. Therefore in practice, with similar architecture,

MrCAE(PR) can always generate a smaller encoding size which leads to a better compression ratio. (It should be noted that unlike other types of networks, in principle, we should count not only the encoded information but also the corresponding positions which is similar to storing sparse matrices when we calculate size of encoding for MrCAE. However, since the position information is shared across all the snapshots and in practice there would be a large number of snapshots, the size of the position information becomes negligible, so we don't include it in our calculation.) In the isotropic turbulence example, this gain is less obvious because of its isotropic nature: there's very little structured spatio-temporal data in this application.

3.6 Discussion

In summary, we have equipped the highly-successful convolutional autoencoder (CAE) with both adaptive filters and multiscale modeling capabilities, giving rise to a flexible workflow which allows for improved interpretability, control of the modeling framework, and in-depth understanding over modern end-to-end architectures. Our proposed MrCAE architecture integrates and leverages three highly successful mathematical architectures: (i) multigrid methods, (ii) convolutional autoencoders and (iii) transfer learning. Instead of training an extremely large black box model end-to-end, our model progressively utilizes a refinement strategy to build a hierarchical structure which leverages increased data due to improved spatial resolution. As a consequence, it enables automatic data augmentation across different spatial resolutions, and outputs insightful intermediate models and results for distinct spatial scales. These intermediate models can either guide the next-level architecture design or be put into immediate use which may be favored by online algorithms.

In addition, we develop a masking mechanism for the use of adaptive convolutional filters which resembles the mesh refinement process. It fully exploits the spatial patterns (which often shows up in many interested spatio-temporal physical systems) in the data set and therefore the size of encoding is small, the use of parameters is parsimonious in nature and the interpretation for each convolutional filter is clear. Though it is highly effective in resolving reconstruction errors, cautions must be taken when there is no clear spatio-temporal separation of scales in the data set.

There are many ongoing challenges and promising directions that motivate future works. Our network is currently very shallow. However, many successful applications achieved by neural networks rely heavily on its depth, as these architectures are reported to create more abstractions and therefore finding more efficient representations [76]. Moreover, our original objective for building up this framework is to do multi-scale forecasting. By adopting such an encoder-decoder architecture, now we are able to build different forecasting tools for different parts of the hierarchical representations. Then, through the decoder, we are able to map it back to the original space. It will be very interesting to see how different forecasting algorithms will be built into the network and how they will affect the encoder-decoder architecture.

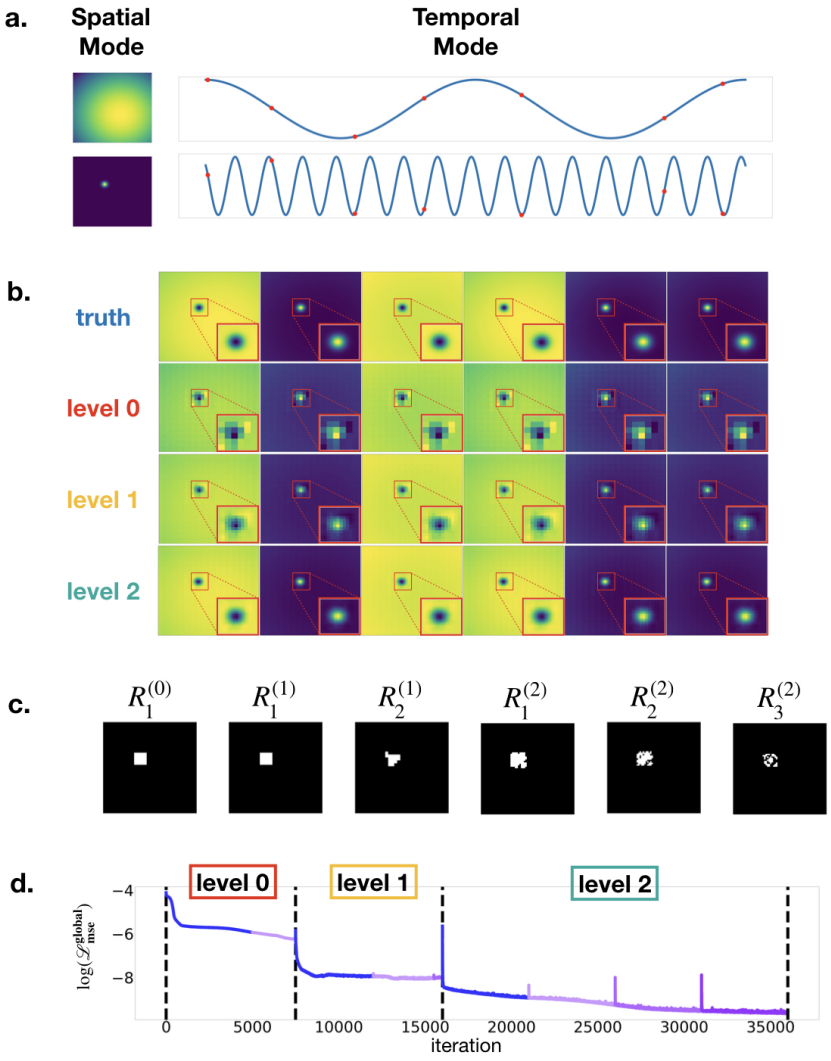

Figure 3.5: **a**. Two separated spatial and temporal modes for the example of two oscillatory modes. Red dots correspond to the sampled test snapshots. **b**. Original data and the reconstructions of the sampled test snapshots across each level of the network. **c**. Regions that different groups of adaptive filters apply across different levels of the architecture. Here, $R_j^{(k)}$ represents the regions that the j^{th} group of adaptive filters being applied at level k. **d**. Logarithmic error plot (in terms of the metric presented in Section 3.4.3) on the validation set throughout the training.

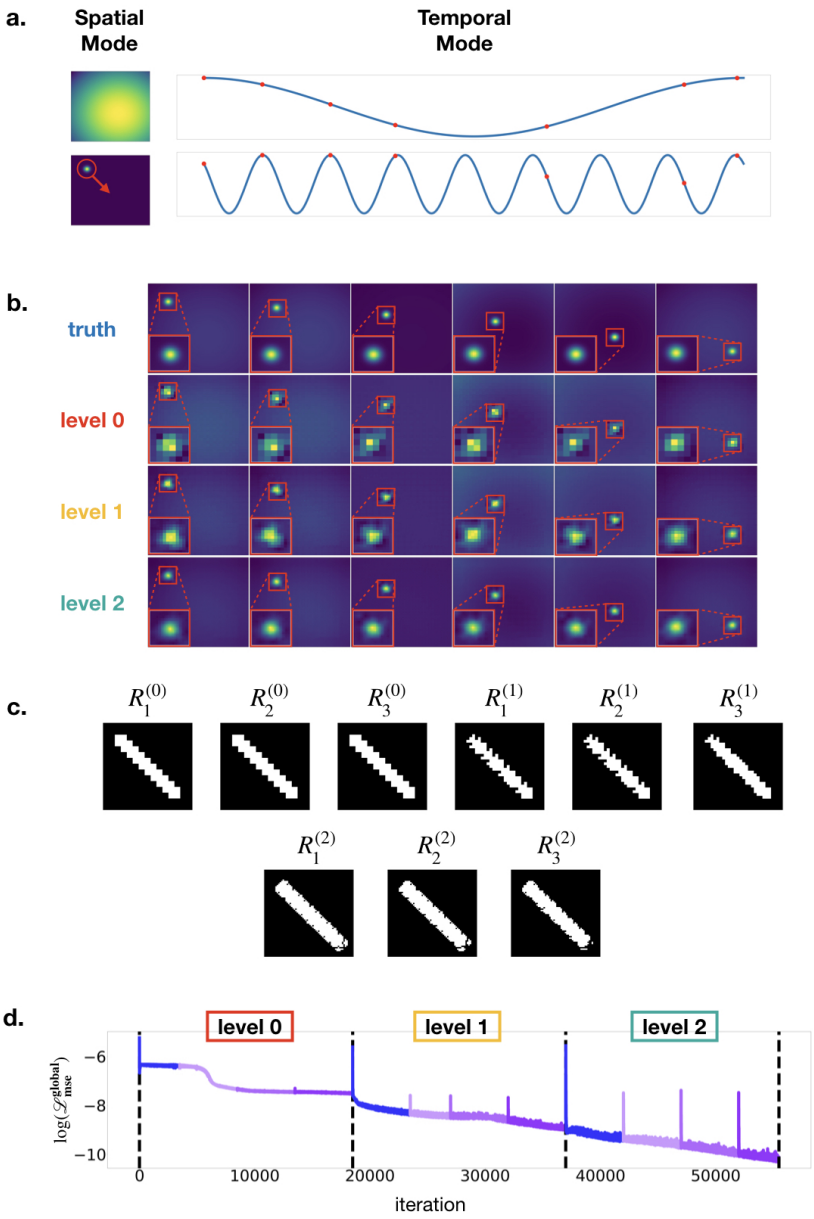

Figure 3.6: From top to bottom: **a**. Two separated spatial and temporal modes for the example of two oscillatory modes with one drifting. Red dots correspond to the sampled test snapshots. **b**. Original data and the reconstructions of the sampled test snapshots across each level of the network. **c**. Regions that different groups of adaptive filters apply across different levels of the architecture. Here, $R_j^{(k)}$ represents the regions that the j^{th} group of adaptive filters being applied at level k. **d**. Logarithmic error plot (in terms of the metric presented in Section 3.4.3) on the validation set throughout the training.

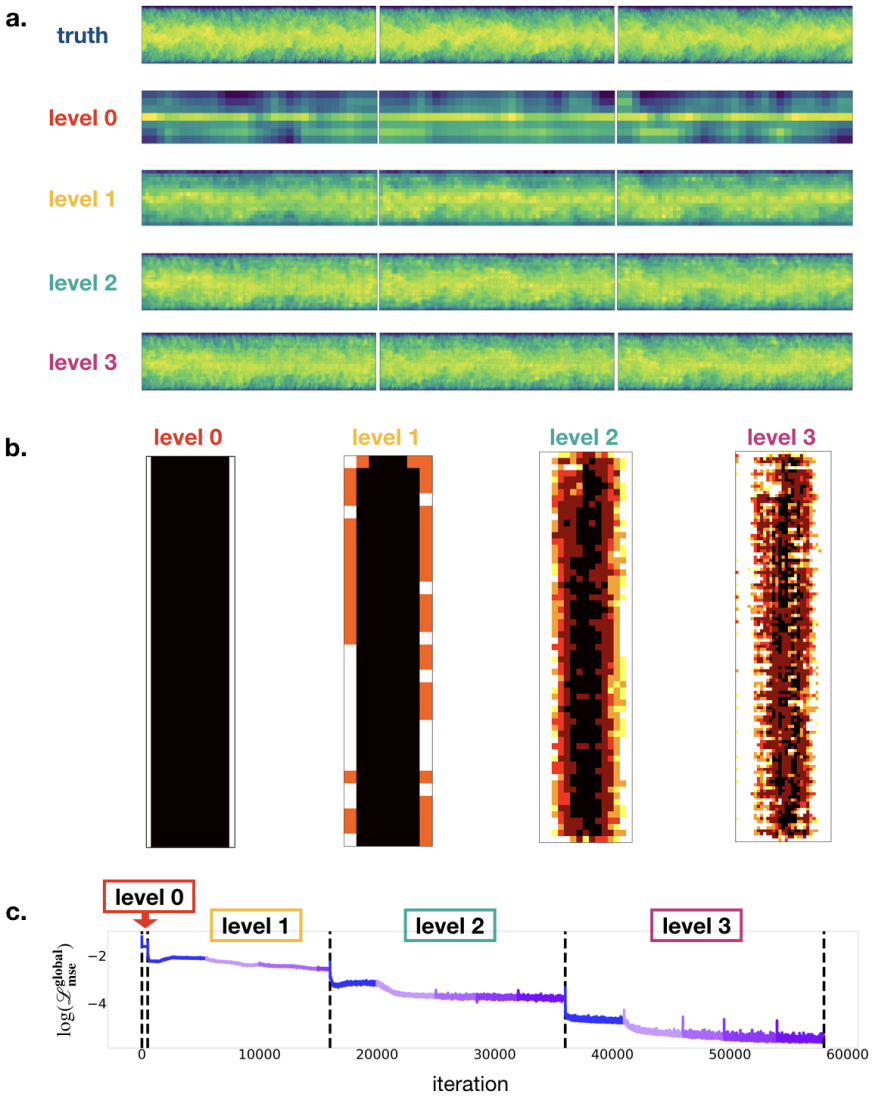

Figure 3.7: **a**. Original data and reconstructions of the channel flow example across different levels of the network over the sampled test snapshots. **b**. Heat maps that reflect the regions that adaptive filters apply across different levels of the architecture. The brighter the pixel, the more filters are applied. **c**. Logarithmic error plot (in terms of the metric presented in Section 3.4.3) on the validation set throughout the training.

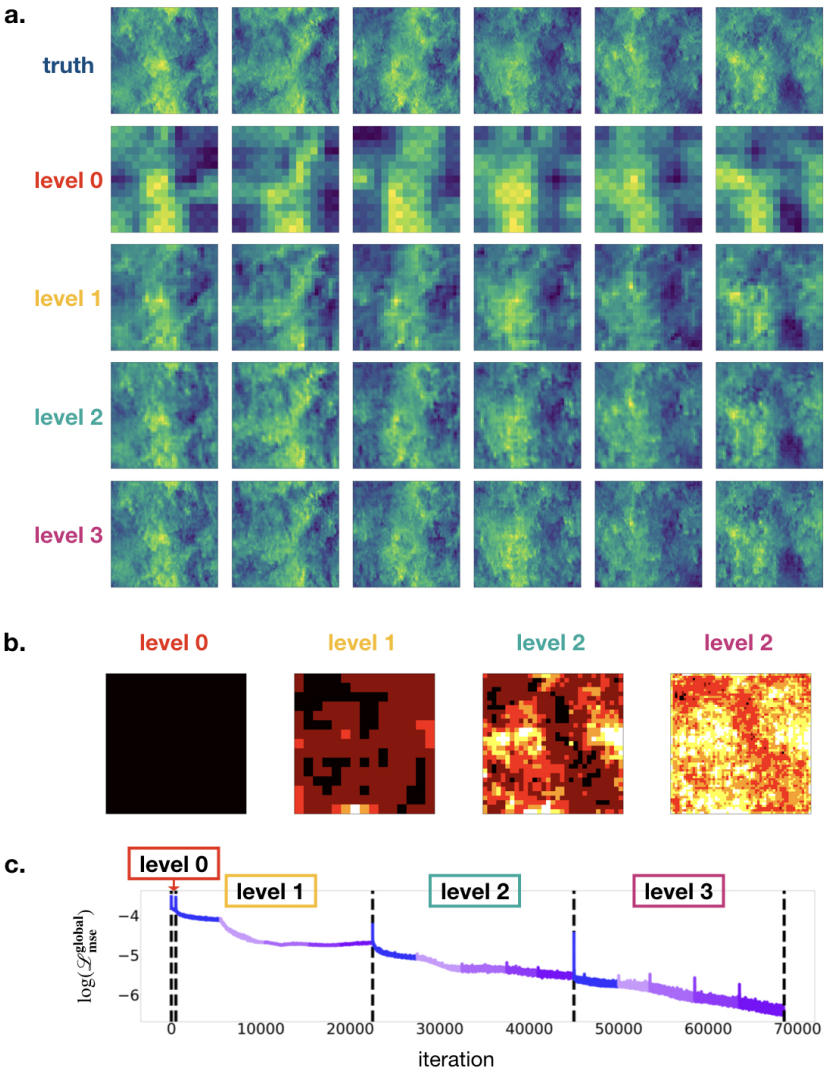

Figure 3.8: **a**. Original data and the reconstructions of the forced isotropic turbulence example across different levels of the network over the sampled test snapshots. **b**. Heat maps that reflect the regions that adaptive filters apply across different levels of the architecture. The brighter the pixel, the more filters are used. **c**. Logarithmic error plot (in terms of the metric presented in Section 3.4.3) on the validation set throughout the training.

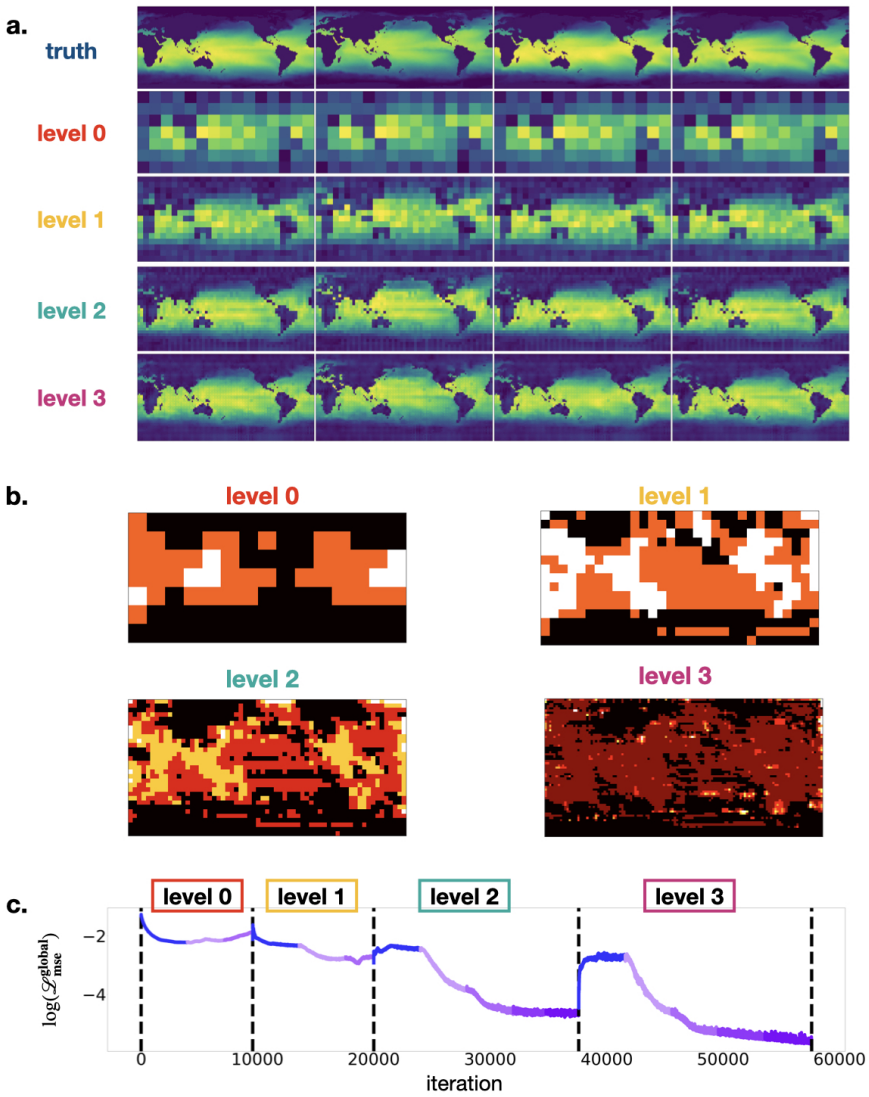

Figure 3.9: **a**. Original data and reconstructions of the sea surface temperature example across different levels of the network over the sampled test snapshots. **b**. Heat maps that reflect the regions that adaptive filters apply across different levels of the architecture. The brighter the pixel, the more filters are used. **c**. Logarithmic error plot (in terms of the metric presented in Section 3.4.3) on the validation set throughout the training.

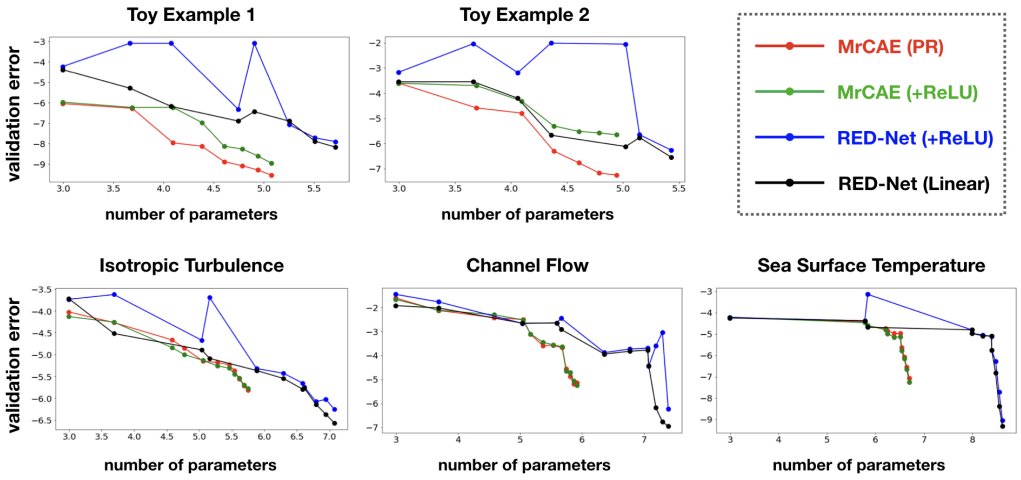

Figure 3.10: **MrCAEs require less parameters.** Comparison of the validation error versus number of parameters for our method versus other state-of-the-art methods. Both the number of parameters and the validation error are on logarithmic scales. In this figure, each point represents a particular architecture: we progressively train the networks of all kinds so that their depths and widths increase along the curves. The number of parameters of MrCAEs scale better as the network expands while exhibiting a steady decrease in validation

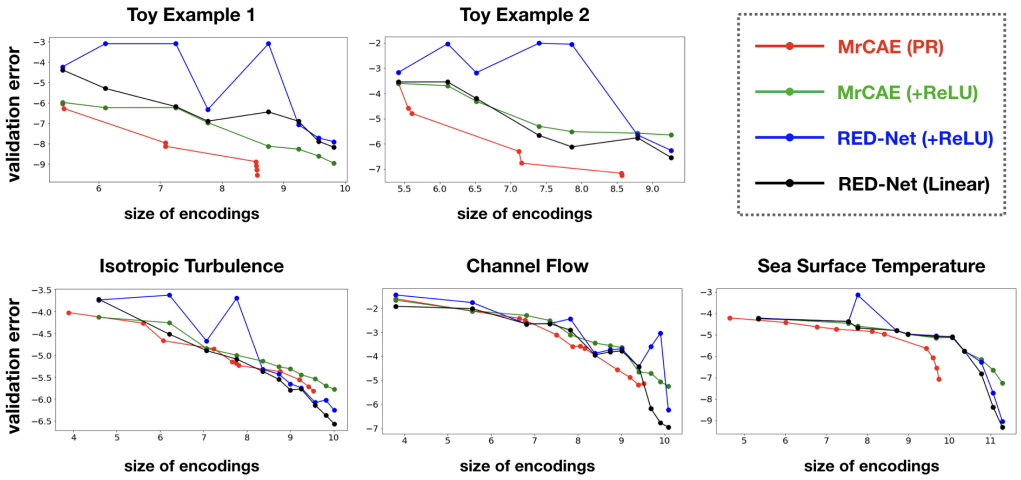

Figure 3.11: **MrCAE(PR) offers better compression ratio**. Comparison of the validation error versus size of encoding for our method versus other state-of-the-art methods. Both the size of encoding and the validation error are on logarithmic scales. MrCAE provides smaller sizes of encoding over similar architectures due to the exploitation of spatial patterns in data sets.

Chapter 4

PHYSICS-INFORMED KOOPMAN NETWORK

4.1 abstract

Koopman operator theory is receiving increased attention due to their promise to linearize the nonlinear dynamics. Neural networks that were developed to represent Koopman operators are of great success thanks to their capability of approximating arbitrarily complex functions. However, despite their great potential, they typically require large training data-sets either from measurements of a real system or high-fidelity simulations. In this work, we propose a novel architecture inspired by the physics-informed neural network, which leverages automatic differentiation to impose the underlying physical laws via soft penalty constraints during model training. We demonstrate that it not only reduces the need of large training data-sets, but also maintains high effectiveness in approximating Koopman eigenfunctions.

4.2 Introduction

Nonlinear dynamical systems give rise to a rich diversity of complex phenomenons. Dealing with nonlinearity is the central task of many areas of sciences and engineering such as climate science, neuroscience, ecology, finance, and epidemiology. A classic yet popular view of dynamical systems is based on the state space models, where the behavior of multiple trajectories can be simultaneously studied and categorized in a qualitative manner. However, the geometry in state space becomes more complex when the dynamics is nonlinear, making the system hard to analyze, predict and control. In 1931, Koopman introduced his operator theoretic perspective of dynamical systems, complementing the traditional geometric perspectives. In his theoretic framework, a Koopman operator is defined which acts on observation functions (observables) in a function space. Under this operator, the evolution of the observables are linear although the function space maybe infinite-dimensional. As a

consequence, approximating the Koopman operator and seeking its eigenfunctions become the key to linearize the nonlinear dynamics.

The leading computational methods for approximating Koopman operator is dynamic mode decomposition (DMD). It is a data-driven approach in that it can be used in absence of any physical models. Using the snapshots of state measurements of a system, the DMD algorithm seeks a best linear operator that approximately advances these states. There are many variants of dynamic mode decomposition, however, most of them require an a priori, judicious selection of the observables; and there is no guarantee that these observables span a invariant Koopman subspace. Another limitation is that a typical application of DMD starts from a single trajectory and thererfore the approximation of the Koopman operator is only restricted to these measurements. As a result, it often fails to generalize to other trajectories unless the system is ergodic so that the training trajectory visits close to any other points in the state space.

To address these challenges, neural networks were proposed to approximate the Koopman operator. In terms of architecture, autoencoder was the most commonly used building block: through an encoding transformation $\mathbf{z} = \varphi(\mathbf{x})$, the inputs were first lifted to a latent space where the dynamics can be propagated linearly, then a decoder network was built to map the latent states back to the original space. To enforce linearity in the latent space, a constraint of the form $\|\varphi(\mathbf{x}_{k+p}) - \mathbf{K}^p\varphi(\mathbf{x}_k)\|$ was added to the loss function, where $\mathbf{K}$ is the matrix that approximates the Koopman operator. The neural network approach generalizes well mainly due to three reasons: (i) it automatically selects the observables; (ii) it can approximate arbitrarily complex functions and (iii) multiple trajectories were used for training. A successfully trained neural network can identify coordinate transformations that make strongly nonlinear dynamics approximately linear and therefore enabling nonlinear prediction, estimation, and control. However, the weakness is also obvious: one would need to acquire a large enough data-set to successfully train a neural network which is neither efficient nor practical.

On another line of research, physics-informed neural networks (PINNs)[203] were introduced in 2019. They can seamlessly integrate the measurement data and physical governing laws by penalizing the residuals of the differential equation in the loss function using automatic

differentiation. Although obtaining more measurement data is generally beneficial, a large data-set is not a necessity. However, these data-efficient architectures also come with costs: (i) the underlying physics needs to be known so that to be imposed via soft penalty constraints during model training (i.e. it's not purely data-driven); (ii) PINNs can only solve one solution instance at a time and (iii) the solution is not accurate outside the training time horizon.

In this work, we propose physics-informed Koopman networks (PIKNs) which combines the strengths of both PINNs and autoencoder-based Koopman networks. More specifically, we reduces the need of large training data-sets for identifying Koopman eigenvalues and eigenfunctions while keeping the generalization property for future state predictions. And since the network performs (Koopman) operator learning, it can be used to solve multiple instances once it is trained. The paper is organized as follows. In Section 4.3, we briefly introduce Koopman operator theory and our methodology. Then we mention some related works in Section 4.4 and highlight the connections and differences. Our approach is then tested on several benchmark problems in Section 4.5. In Section 4.6, we conclude and discuss future directions. Our code is publicly available at `https://github.com/luckystarufo/PIKN`.

4.3 Method

4.3.1 Koopman operator theory

We begin with an autonomous ordinary differential equation 4.1 defined on a finite-dimensional space $\mathcal{X} \subseteq \mathbb{R}^n$.

$$\frac{d}{dt}\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t)) \tag{4.1}$$

And we can define the time-t flow map operator as $\mathbf{F}^t : \mathcal{X} \to \mathcal{X}$ which advances states forward along the trajectory by a time t, so that

$$\mathbf{x}(t_0 + t) = \mathbf{F}^t(\mathbf{x}(t_0)) \tag{4.2}$$

In 1931, B.O.Koopman provided an alternative description for dynamical systems in terms of evolution of functions of possible measurements $\mathbf{y} = g(\mathbf{x})$ of the states, where $g : \mathcal{X} \to \mathbb{C}$ is called a measurement function which belongs to some set of functions $\mathcal{G}(\mathcal{X})$. For most

of the cases, this set of functions $\mathcal{G}(\mathcal{X})$ is not defined *a priori*. Hilbert spaces such as $L^2(\mathcal{X}, d\mu)$ or reproducing kernel Hilbert spaces (RKHS) are common choices for modern applied analysis. In all cases, however, $\mathcal{G}(\mathcal{X})$ is of significantly higher dimension than $\mathcal{X}$, so we are trading dimensionality for linearity.

The family of Koopman operators $\mathcal{K}^t : \mathcal{G}(\mathcal{X}) \to \mathcal{G}(\mathcal{X})$, parameterized by t are given by

$$\mathcal{K}^t g(\mathbf{x}) = g(\mathbf{F}^t(\mathbf{x})) \tag{4.3}$$

One can easily check $\mathcal{K}^t$ is linear, and the evolution in equation 4.3 is analogous to any linear systems except it is infinite-dimensional.

If the dynamics in equation 4.1 is sufficiently smooth, one can also define the infinitesimal generator $\mathcal{L}$ of the Koopman operator family as

$$\mathcal{L}g := \lim_{t\to 0} \frac{\mathcal{K}^t g - g}{t} = \lim_{t\to 0} \frac{g \circ \mathbf{F}^t - g}{t} \tag{4.4}$$

From the definition, we can easily see

$$\mathcal{L}g(\mathbf{x}(t)) = \lim_{\tau\to 0} \frac{g(\mathbf{x}(t+\tau)) - g(\mathbf{x}(t))}{\tau} = \frac{d}{dt} g(\mathbf{x}(t)) \tag{4.5}$$

On the other hand, we also have

$$\frac{d}{dt} g(\mathbf{x}(t)) = \nabla g \cdot \frac{d}{dt}\mathbf{x}(t) = \nabla g \cdot \mathbf{f}(\mathbf{x}(t)) \tag{4.6}$$

Therefore, we conclude

$$\mathcal{L}g = \nabla g \cdot \mathbf{f} \tag{4.7}$$

This is why the generator $\mathcal{L}$ was called the Lie operator historically: it is the Lie derivative of g along the vector field $\mathbf{f}(\mathbf{x})$ when the dynamics is given by equation 4.1. Moreover, this important consequence will be the key for the physics-informed Koopman networks (PIKNs).

Applied Koopman analysis attempts to seek key measurement functions that behave linearly in time. Eigenfunctions of the Koopman operator are natural candidates. A Koopman eigenfunction $\varphi(\mathbf{x})$ corresponding to an eigenvalue λ^t satisfies

$$\mathcal{K}^t \varphi(\mathbf{x}) = \lambda^t \varphi(\mathbf{x}) \tag{4.8}$$

It is simple to show that Koopman eigenfunctions $\varphi(\mathbf{x})$ that satisfies equation 4.8 for $\lambda \neq 0$ are also eigenfunctions of the Lie operator, although with a different eigenvalue $\mu = \log(\lambda^t)/t = \log(\lambda)$. In general, eigenvalues and eigenfunctions are both complex-valued. Once we have a set of such eigenfunctions $\{\varphi_1, \varphi_2, \cdots, \varphi_M\}$, observables that can be formed as a linear combination of these eigenfunctions, i.e., $g \in span\{\varphi_k\}_{k=1}^M$ have a particularly simple evolution under the Koopman operator

$$g(\mathbf{x}) = \sum_{k=1}^{M} c_k \varphi_k(\mathbf{x}) \quad \Longrightarrow \quad \mathcal{K}^t g(\mathbf{x}) = \sum_{k=1}^{M} c_k \lambda_k^t \varphi_k(\mathbf{x}) \tag{4.9}$$

This also implies $span\{\varphi_k\}_{k=1}^M$ is an invariant subspace under the Koopman operator $\mathcal{K}^t$ and $\{\varphi_1, \varphi_2, \cdots, \varphi_M\}$ can be viewed as the new coordinates on which the dynamics evolve. Although the promise of Koopman operator theory is tempting, there are certain challenges as well, for example,

- The Koopman operator of a system may not admit a discrete spectrum. Certain systems fundamentally fail to fit into this framework.

- Some asymptotic methods can be used to approximate certain eigenfunctions for simple dynamics (eg. polynomial nonlinear dynamics), however, there is no analytical procedure to seek for the eigenvalue and eigenfunction pairs in general.

- Some computational methods (eg. DMD) can be used to approximate eigenfunctions using snapshots of measurements, but the approximation is only restricted to those measurements, sometimes leading to spurious modes. And the identified basis may not span a Koopman invariant subspace.

In addition, since our ultimate goal is to study nonlinear dynamical systems using linear theory, we do not need to restrict ourselves to equation 4.9. Instead, we can generalize it as

$$\begin{gathered} g(\mathbf{x}) = \psi(\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_M(\mathbf{x}); \omega) \\ \Downarrow \\ \mathcal{K}^t g(\mathbf{x}) = \psi(\lambda_1^t \varphi_1(\mathbf{x}), \lambda_2^t \varphi_2(\mathbf{x}), \ldots, \lambda_M^t \varphi_M(\mathbf{x}); \omega) \end{gathered} \tag{4.10}$$

where ψ is an arbitrary transformation parameterized by ω. In fact, [149, 74] all share this viewpoint.

4.3.2 Auto-encoder based architecture

Neural networks are computational models that are composed of multiple layers and used to learn representations of data. Importantly, the universal approximation theorem guarantees that a neural network with sufficiently many hidden units and a linear output layer is capable of representing any arbitrary function. Auto-encoder is a special type of neural network which is particularly suitable for our application and widely used in the literature [149, 74]. There are two separate parts of the network: an encoder and a decoder. The encoder learns the representation of the relevant Koopman eigenfunctions which provides intrinsic coordinates that linearize the dynamics. And the decoder seeks an inverse transformation to reconstruct the original measurements. Ideally, if we define $\phi : \mathbf{x} \to (\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_M(\mathbf{x}))^T$, then up to a constant, the encoder learns this transformation ϕ and the decoder learns the ψ as shown in equation 4.10. (Alternatively, if we specify a linear decoder, then the learning regime would correspond to equation 4.9.) Therefore without losing generality, we will name the encoding/decoding transformation as ϕ and ψ, respectively.

Within the auto-encoder, the dynamics is supposed to be linear. Therefore in previous works, a squared matrix $\mathbf{K}$ is often used to drive the evolution of the dynamics, and the choice of its size usually requires certain insightful observations or a process of trial and error. Sometimes there is no invariant, finite-dimensional Koopman subspace that captures the evolution of all the measurements, then that matrix $\mathbf{K}$ will only be an approximation of the true underlying linear operator.

There are different ways to train the auto-encoder architecture in the literature [149, 74], however, all of them would require a large amount of measurement data. Normally, the training data-set X is arranged as a 3D tensor, with its dimensions to be (i) number of sequences (with different initial states), (ii) number of snapshots and (iii) dimensionality of the measurements, respectively. Then the constraint of linear dynamics can be enforced by a loss term resembling $\|\phi(\mathbf{x}_{n+1}) - \mathbf{K}\phi(\mathbf{x}_n)\|$, or more generally, linearity is enforced over multiple steps $\|\phi(\mathbf{x}_{n+p}) - \mathbf{K}^p\phi(\mathbf{x}_n)\|$, generating recurrencies in the neural network architecture. We will see, however, such large data-sets are not necessary (but beneficial) for the physics-informed Koopman networks (PIKNs).

4.3.3 Physics-informed Koopman network

In physics-informed Koopman networks (PIKNs), we do not enforce linearity using snapshots of measurement data, we take advantage of equation 4.7: the linearity constraint is enforced through minimizing the quantity $\|\nabla\varphi_k(\mathbf{x})\cdot\mathbf{f}-\mu_k\varphi_k(\mathbf{x})\|,\quad \forall k=1,2,3,\cdots,M$. Or more generally, a squared matrix L is used to approximate the Lie operator $\mathcal{L}$ and we minimize $\|L\phi(\mathbf{x})-\nabla\phi(\mathbf{x})\cdot\mathbf{f}\|$. Finding the eigenvalue and eigenfunction pairs of the Lie operator corresponds to performing eigendecomposition to the matrix L.

For ODE

To be more specific, let's first consider an ordinary differential equation of the form in equation 4.1. We start from sampling a set of collocation points $X:=\{\mathbf{x}_1,\mathbf{x}_2,\cdots,\mathbf{x}_N\}$. The key to note here is that this set of collocation points do not need to come from any trajectories of the dynamics but simply some randomly generated points, avoiding a bulk of simulations or measurement data collections. The objective of the network is to identify a few key coordinates $\mathbf{z}=\phi(\mathbf{x})$ spannned by a set of Koopman eigenfunctions $\varphi_k(\mathbf{x}):\mathcal{R}^n\rightarrow\mathcal{R},\ k=1,2,\cdots,M$ along with a dynamical system $\dot{\mathbf{z}}=L\mathbf{z}$. So our objective function is going to be

$$\mathcal{J}_{linear}=\frac{1}{N}\sum_{i=1}^{N}(\omega_1\|L\phi(\mathbf{x}_i)-\nabla\phi(\mathbf{x}_i)\cdot\mathbf{f}(\mathbf{x}_i)\|^2+\omega_2\|\mathbf{x}_i-C\mathbf{z}_i\|^2) \tag{4.11}$$

if the decoder is linear (where C represents the reconstruction coefficients), or for a more general decoder

$$\mathcal{J}_{nonlinear}=\frac{1}{N}\sum_{i=1}^{N}(\omega_1\|L\phi(\mathbf{x}_i)-\nabla\phi(\mathbf{x}_i)\cdot\mathbf{f}(\mathbf{x}_i)\|^2+\omega_2\|\mathbf{x}_i-\psi(\mathbf{z_i})\|^2) \tag{4.12}$$

Here, ω_1 and ω_2 are the weights for each loss term. The first term encourages linear dynamics within the latent space and the second term makes it a valid auto-encoder. One can further enforce L to have a diagonal structure, then those diagonal elements will approximate the Koopman eigenvalues and the corresponding outputs of the encoder will approximate the Koopman eigenfunctions, respectively. But this constraint is not necessary as it is equivalent to performing eigendecomposition of a general-structured L after training.

For PDE

For application to partial differential equations of the form

$$\mathbf{u}_t = \mathbf{f}(\mathbf{u}, \mathbf{u}_\mathbf{x}, \dots) \tag{4.13}$$

the goal is still to seek coordinates $\mathbf{v} = \phi(\mathbf{u})$ that has linear evolution $\dot{\mathbf{v}} = L\mathbf{v}$ and can be used to reconstruct the original measurements $\hat{\mathbf{u}} = \psi(\mathbf{v})$. The big difference is, however, the input/output of the network are functions of spatial variables $\mathbf{u}(\mathbf{x}, \mathbf{t})$ instead of the spatial variables $\mathbf{x}(t)$ itself as in the ODE cases. Therefore, we need to sample a set of "collocation points" in an appropriate function space, which is usually restricted by some known boundary conditions. In addition, the training set also includes spatial derivatives of $\mathbf{u}$ as shown in equation 4.13. Those will be obtained either from analytical derivations or some numerical methods. All the functions will be represented on some equally-spaced grids in our work.

4.3.4 Data integration

Like PINNs, we can easily integrate information from measurement data. Suppose we have snapshots of measurements $X_{data} := \{\mathbf{x}(t_0), \mathbf{x}(t_1), \cdots, \mathbf{x}(t_p)\}$ for an ODE system or $U_{data} := \{\mathbf{u}(\mathbf{x}, t_0), \mathbf{u}(\mathbf{x}, t_1), \cdots, \mathbf{u}(\mathbf{x}, t_p)\}$ for a PDE, by adding extra loss terms

$$\begin{aligned}
\mathcal{J}_{data} &= \frac{1}{p}\sum_{j=0}^{p}(\omega_3\|e^{L\Delta t_j}\phi(\mathbf{x}(t_0)) - \phi(\mathbf{x}(t_j))\|^2 + \omega_4\|\mathbf{x}(t_j) - \psi(\mathbf{z}(t_j))\|^2) \quad \text{(for ODE)} \\
\mathcal{J}_{data} &= \frac{1}{p}\sum_{j=0}^{p}(\omega_3\|e^{L\Delta t_j}\phi(\mathbf{u}(\mathbf{x}, t_0)) - \phi(\mathbf{u}(\mathbf{x}, t_j))\|^2 + \omega_4\|\mathbf{u}(\mathbf{x}, t_j) - \psi(\mathbf{v}(\mathbf{x}, t_j))\|^2) \quad \text{(for PDE)}
\end{aligned} \tag{4.14}$$

we can penalize the network predictions that do not match the real measurements, where $\Delta t_j = t_j - t_0, \quad \forall j = 1, 2, \cdots, p$. And $e^{L\Delta t} := U^T e^{\Lambda\Delta t} U$ where $L = U^T \Lambda U$ is the eigen-decomposition of L. Again, the first term is the linearity loss and the second term is the reconstruction loss. And integration of information from data highly resembles the DMD algorithms.

4.4 Related Work

4.4.1 Dynamic Mode Decomposition

Dynamic mode decomposition (DMD), which was originally introduced by Schmid[220], is the leading computational method to approximate the Koopman operator from data[124, 245]. Rowley et al. were the first to establish the connection between DMD and Koopman operator theory[211]. One of the major advantages of DMD is its simple formulation in terms of linear regression. For this reason, many methodological innovations have been introduced, for example, Jovanovic et al.[105] uses sparsity promoting optimzation to identify the fewest DMD modes; [23, 64] accelerate DMD using randomized linear algebra; extended DMD[266] suggests to include nonlinear measurements; higher order DMD[127] acts on delayed coordinates and generates more complex models; multiresolution DMD[125] deals with multiscale systems that exhibit transient or intermittent dynamics; Proctor et al.[196] extended the algorithm to disambiguate the natural dynamics and actuation; algorithms include total least-squares DMD[88], forward-bacward DMD[56] and variable projection[5] improve the performance of DMD over noise sensitivity. These methods are now widely applied to many fields in science and engineering such as fluid dynamics[6, 12, 13, 168], epidemiology[197], neuroscience[31], finance[159], plasma physics[238], robotics[17, 18] and video processing[80, 63, 20]. For a more comprehensive review, one can refer to [33].

4.4.2 Deep Learning for Linear Embeddings

Hand-crafted basis functions or measurements from DMD sometimes fail to represent the complex Koopman eigenfunctions. Neural networks turn out to be more effective in approximating them, leading to linear embeddings of the nonlinear dynamics[236, 132, 149, 273]. They have achieved great successes in long-term dynamic predictions[126], fluid control[169] and also recently be extended to account for uncertainties[170], modeling PDEs[74] and jointly used with optimal control[83, 2]. There are also innovations on the side of neural network architectures, for example, neural ODEs are used for dictionary learning[239] and graphical neural networks are used to learn compositional Koopman operators[137]. However, all the listed works are purely data-driven and do not address the issue of data

efficiency.

4.4.3 Physics-Informed Neural Network

Physics-informed neural networks (PINNs) were first proposed in[206, 203] and have received lots of attention due to its flexibility to integrate measurement data and the physics (governing equations). They have been applied to various classes of PDEs[183, 65, 277] and extended to deal with uncertainties[272, 278, 280, 229, 270]. Another line of research of PINNs focus on its training and performance. For example, domain decomposition is considered in some variations of PINNs[116, 103, 101], leading to parallel implementations[226, 89]; multi-fidelity framework[164], dynamic weights of the loss function[249] and hard constraints[145] have also been thoroughly studied, in order to achieve stable training results. Theoretical insights into the convergence of PINNs are presented in[225, 167, 251]. Recently, PINNs have also been jointly used with DeepONets[250], entering the realm of operator learning. Similar to our work, the key motivation of using PINNs there is to eliminate the need of large data-sets for training DeepONets.

4.5 Experiments

In this section, we demonstrate our approach by applying it to several ODE and PDE examples. We start with easy examples to showcase the basic usage and verify its correctness. Then we proceed to more challenging examples to reveal the benefits and potential pitfalls of this approach.

4.5.1 Simple nonlinear system with discrete spectrum

First, we consider a simple nonlinear system with a single fixed point and a discrete eigenvalue spectrum:

$$\begin{aligned} \dot{x_1} &= \mu x_1 \\ \dot{x_2} &= \lambda(x_2 - x_1^2) \end{aligned} \tag{4.15}$$

For $\lambda < \mu < 0$, the system exhibits a slow attracting manifold given by $x_2 = x_1^2$. In our experiment, we use $\mu = -0.1$ and $\lambda = -1$. This example is simple and widely adopted

as a benchmark for Koopman/DMD related algorithms because it's possible to explicitly define a three-dimensional Koopman invariant subspace (spanned by Koopman eigenvalue and eigenfunction pairs) that contains the state variables x_1 and x_2:

$$\begin{aligned} \frac{d}{dt}\begin{bmatrix} \varphi_\mu \\ \varphi_{2\mu} \\ \varphi_\lambda \end{bmatrix} &= \begin{bmatrix} \mu & 0 & 0 \\ 0 & 2\mu & 0 \\ 0 & 0 & \lambda \end{bmatrix}\begin{bmatrix} \varphi_\mu \\ \varphi_{2\mu} \\ \varphi_\lambda \end{bmatrix} \\ \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & b & 1 \end{bmatrix}\begin{bmatrix} \varphi_\mu \\ \varphi_{2\mu} \\ \varphi_\lambda \end{bmatrix} \end{aligned} \tag{4.16}$$

where $\phi(\mathbf{x}) = [\varphi_\mu, \varphi_{2\mu}, \varphi_\lambda] = [x_1, x_1^2, x_2 - bx_1^2]$ and $b = \frac{\lambda}{\lambda - 2\mu}$ is a constant. If nonlinear decoding is allowed, a two-dimensional Koopman invariant subspace spanned only by $\{\varphi_\mu, \varphi_\lambda\}$ is sufficient because

$$\begin{aligned} x_1 &= \varphi_\mu \\ x_2 &= \varphi_\lambda + b\varphi_\mu^2 \end{aligned} \tag{4.17}$$

Therefore, we should at least expect our network to find one three-dimensional Koopman invariant subspace with a linear decoder or a two-dimensional Koopman invariant subspace with a nonlinear decoder. In Fig 4.1, we show the results of trained PIKNs with and without a linear decoder in two columns respectively. The networks have accurately identified the Koopman eigenvalues and eigenfunctions[1] within the corresponding invariant subspace. Both models show very strong predictive power while the one equipped with a linear decoder provides more robust results. Specific details about the network architecture and training procedure are provided in the Appendix.

4.5.2 Heat equation

The first PDE we consider is the one-dimensional heat equation:

$$u_t = u_{xx}, \qquad x \in (-\pi, \pi) \tag{4.18}$$

[1]We don't expect eigenfunctions to be exactly $[x_1, x_1^2, x_2 - bx_1^2]$ because each one is still a valid eigenfunction (associated with the same eigenvalue) when being multiplied by some factors. Therefore, this statement is correct up to some constant.

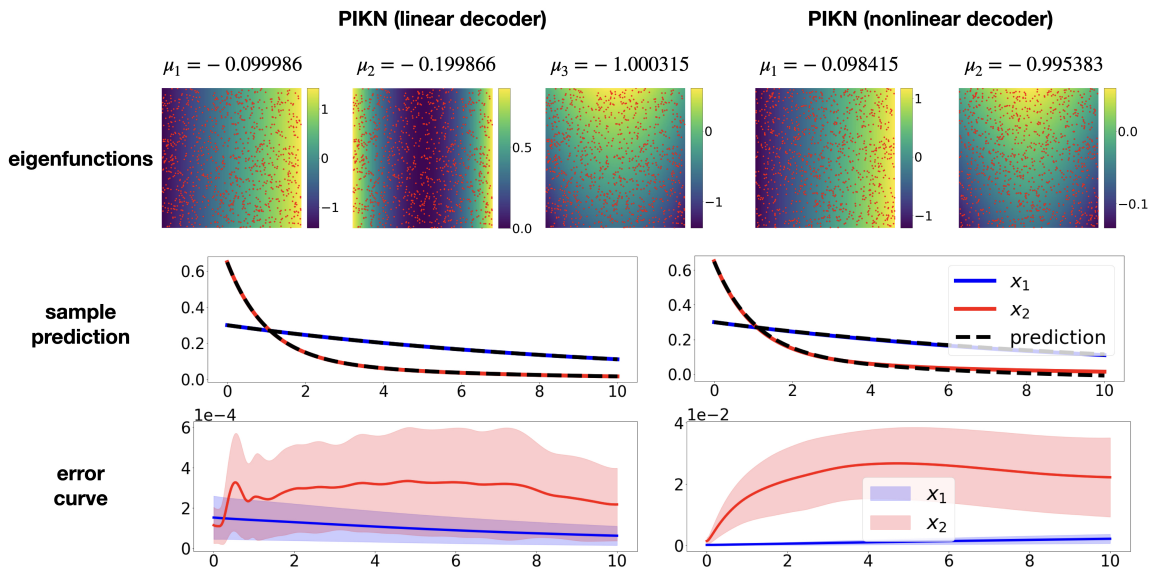

Figure 4.1: The two columns show the PIKN results w/o a linear decoder network, respectively. The first row shows the eigenvalues and eigenfunctions identified by PIKNs where the red dots represent the collocation points for training; the second row shows a 10000-step forward prediction of the dynamics starting from a initial state $(x_1, x_2) = (0.3, 0.65)$ and with $\Delta t = 0.001$; the third row visualizes the mean absolute error over 1000 different trajectories with initial conditions uniformly sampled from $[-1, 1] \times [-1, 1]$, the shaded region covers one standard deviation away from the mean error.

with periodic boundary conditions. The heat equation can be linearized by the Fourier transform and the spectrum is discrete due to the periodic boundary conditions. Theoretically, the discrete-time eigenvalues are

$$\mu_k = -k^2, \quad k = 0, \pm 1, \pm 2, \dots \tag{4.19}$$

To approximately represent u and u_{xx}, we discretize the spatial domain with $n = 64$ equally spaced grids. As a consequence, the value of k only ranges from -32 to 31. Therefore, we expect our network to at least mimic a discrete Fourier transform and its inverse transform, identifying the right eigenvalues after training. In addition, we study the effects of data

integration. More specifically, we compare the results obtained from the networks that is purely physics-informed, purely data-driven and a mixture of both which we call it hybrid model. More details of the experiments can be found in the Appendix.

Indeed, Fig 4.2 shows that the transformation coefficients collide with the frequencies of the corresponding Fourier modes. The phase difference is expected because Discrete Fourier transformation is not unique for diagonaling the heat equation. The networks nearly identify all the correct eigenvalues of the heat equation, however, the purely physics-informed network fails to discover the low-frequency modes. In addition, we observe it identifies an eigenvalue with a small positive real part (around 0.00005) which is not visualized in this plot. On the contrary, the purely data-driven network misses the high-frequency modes but all identified eigenvalues have negative real parts; the hybrid model presents the most satisfying accuracy among all. The reason behind it might be the multiscale feature of the system:

- For data-driven model, we use simulation data which are integrated over time. Slow dynamics (which are associated with the slow frequencies) are more persistent which dominate in the overall loss.

- For physics-informed model, we use the spatial derivatives of the states which are more sensitive to the high-frequency transient modes.

- The hybrid model leverages these two time scales and achieves a nice balance for identifying all eigenvalues.

Fig 4.3 shows the error curves of the networks trained in different ways. The left plot shows physics-informed network is good at short-term predictions while the data-driven network is more promising in long-term predictions. The hybrid network merges the best part of these two and consistently offer the most accurate predictions. We also find from the right plot that the performance of the hybrid model is not sensitive to the number of simulation data used for training. More importantly, training with the largest number of simulation data does not necessarily gives the best prediction results. This is probably because in principle

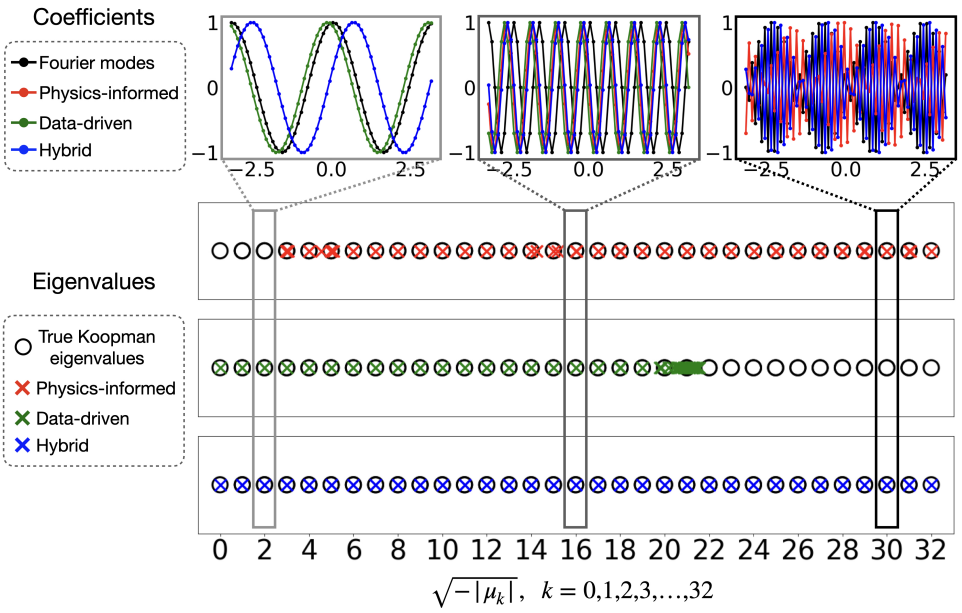

Figure 4.2: The eigenvalues (with negative real part) of the matrix L from different neural networks are plotted along with the exact, discrete-time eigenvalues of the heat equation at the bottom. The top row shows the coefficients of the linear transformation corresponding to the selected eigenvalues.

the physics-informed training is sufficient for finding all modes, adding simulation data can only help better discover the persistent modes. This suggests that if the network is physics-informed, data demand from simulation is low although data integration is beneficial for identifying a more accurate model.

4.5.3 Burger's equation

For the next example, we consider the nonlinear PDE known as the Burger's equation

$$u_t = -uu_x + u_{xx},\ x \in (-\pi, \pi) \tag{4.20}$$

with homogeneous boundary conditions. It is known that the Burger's equation can be linearized through Cole-Hopf transformation which was discovered in 1950 and 1951 by Eberhard Hopf[95] and Julian Cole[48], independently of one another, and later noticed by

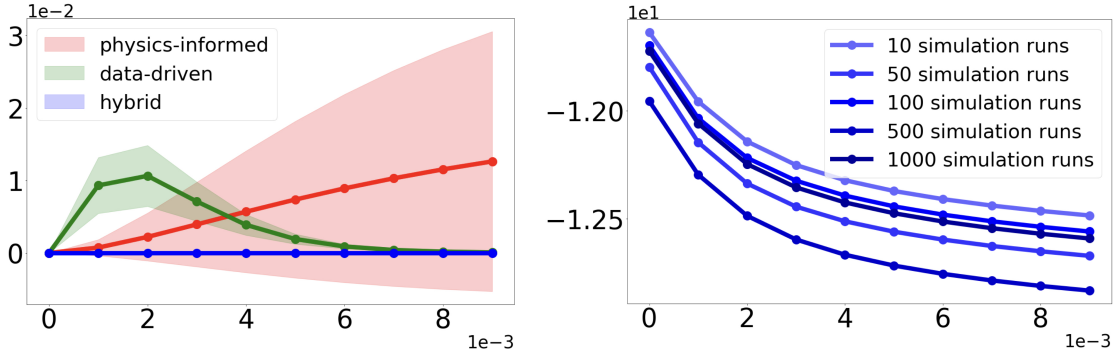

Figure 4.3: The left plot shows the predictive power of three different neural networks. The prediction task is evaluated on 1000 newly simulated test trials. The shaded region highlights one standard deviation from the mean squared error. The right plot shows the error curves of 5 hybrid models. These models differ on the number of simulated trajectories used for training. And the error is visualized in a logarithmic scale of base 10. Both plots are 10-step forward predictions with $\Delta t = 0.001$.

Kutz et al [174] and exploited by many others[178, 8]. The Cole-Hopf transformations are defined as:

$$\begin{aligned} u := C(v) &= -\frac{2v_x}{v} \\ v := H(u) &= \frac{e^{-\frac{1}{2}\int_{-\pi}^{x} u(s,t)ds}}{\int_{-\pi}^{\pi} e^{-\frac{1}{2}\int_{-\pi}^{x} u(s,t)ds} dx} \end{aligned} \tag{4.21}$$

If $u(x,t)$ satisfies the Burger's equation 4.20 with a homogeneous boundary condition $u(-\pi,t) = u(\pi,t) = 0$ and an initial condition $u(x,0) = u_0(x)$, then $v(x,t) = H(u(x,t))$ solves the heat equation:

$$\begin{aligned} &v_t = v_{xx}, \ \ x \in (-\pi, \pi) \\ &v_x(-\pi,t) = v_x(\pi,t) = 0 \\ &v(x,0) := v_0(x) = H(u_0(x)) \end{aligned} \tag{4.22}$$

The solution of it, by using a standard separation of variables approach, can be derived as

$$v(x,t) = C_0 + \sum_{k=1}^{\infty} C_k \cos(kx) e^{-k^2 t} \tag{4.23}$$

where $C_0 = \frac{1}{2\pi}\int_{-\pi}^{\pi} v_0(x)dx$ and $C_k = \frac{1}{\pi}\int_{-\pi}^{\pi} v_0(x)\cos(kx)dx$.
Conversely, it is also shown in [8] that if $v(x,t)$ solves the above heat equation and further satisfies

$$\begin{aligned} &v(x,t) > 0,\ x \in (-\pi,\pi), t > 0 \\ &\int_{-\pi}^{\pi} v(x,t)dx = 1 \end{aligned} \tag{4.24}$$

then $u(x,t) = C(v(x,t))$ is also a solution for the original Burger's equation. To satisfy these constraints, we can require $C_0 = \frac{1}{2\pi}$ and $\sum_{k=1}^{\infty} |C_k| < \frac{1}{2\pi}$. More details and the experimental setups can be found in the Appendix.
In fig 4.4, the identified eigenvalues for Burger's equation of each training strategy are plotted along with the exact Koopman eigenvalues. We see that the data-driven model is still able to effectively identify the low frequencies (i.e. $\mu = -0.942, -3.965$) while gradually losing track on high-frequency modes. The reason behind might still be that high-frequency modes are transient and therefore is negligible during training. The physics-informed learning regime, however, is proved to be challenging. Though it shows potential in identifying some high-frequency modes (eg. $\mu = -62.223, -80.005$), it's not clear how to interpret them especially in the case that three other positive eigenvalues are also observed ($\mu = 9.74, 4.65, 2.01$). The hybrid model seems to combine both features and reveal a more appealing spectrum, however, positive eigenvalues are also observed ($\mu = 3.668, 2.732$). The modes (or latent representation) of a sampled input is visualized against the exact Koopman modes up to some scaling, we see there is no good alignment, indicating the learned transformations are not Cole-Hopf. Indeed, Cole-Hopf transformations may not be only one to linearize Burger's equation, similar to the problem arised in section C.2.
Therefore, we turn to study the models provided by different training strategies. Fig 4.5 shows the error curves of future state predictions of networks trained in different ways. The left plot shows data-driven model is unbeatable, whereas the physics-informed network suffers from a lower reconstruction accuracy as well as bad long-term prediction accuracy

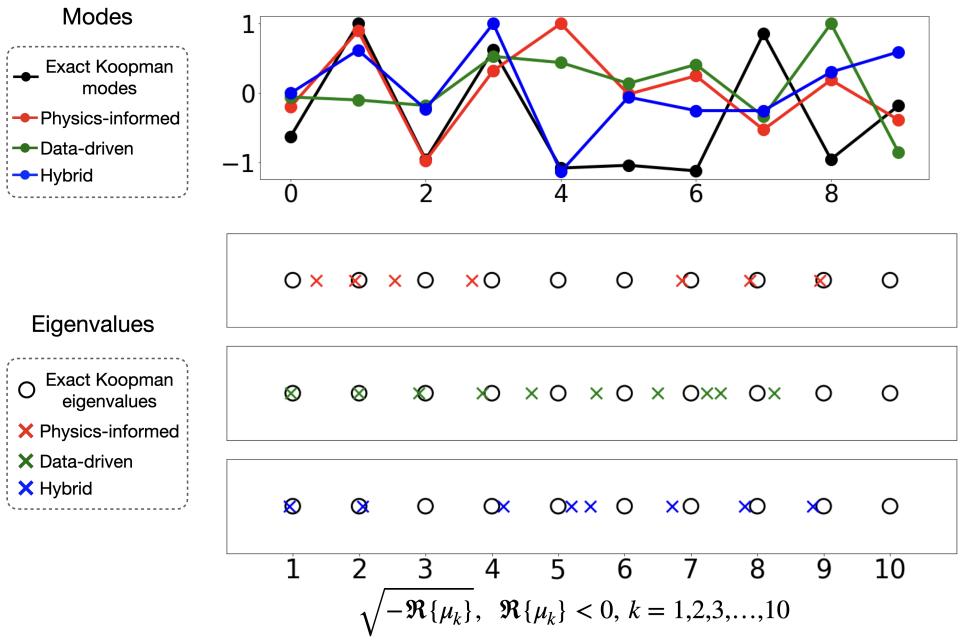

Figure 4.4: The first row shows the identified modes (or latent representation) of a sampled test input from different network architectures along with the exact Koopman modes. The second row shows eigenvalues (with negative real part) from different neural networks along with the exact, discrete-time eigenvalues of the Burger's equation identified through Cole-Hopf.

due to the existence of the positive eigenvalues. The situation is similar for the hybrid model with slightly weaker effects. Although the data-driven model is superior for Burgers', we notice that the performance deteriorates badly when the availability of training data is limited. The right plot shows the performance of data-driven models and hybrid models trained with different number of trajectories. As number of training trajectories goes below 500, hybrid models are preferable for predictions, indicating the value of adding physical constraints to the training.

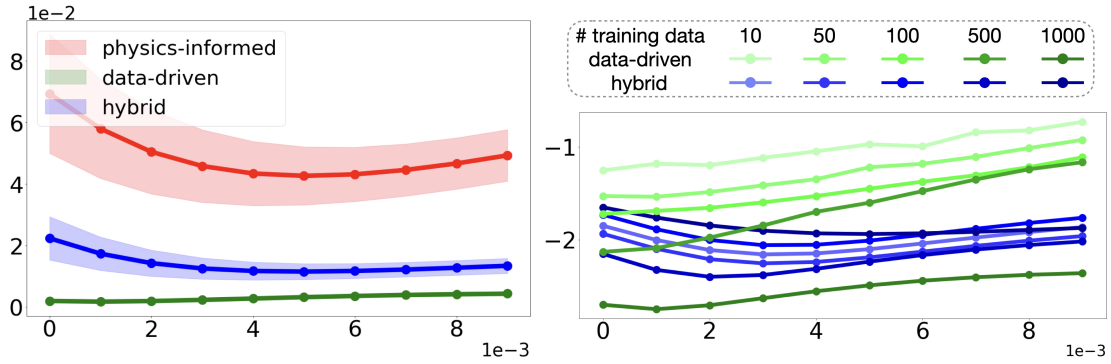

Figure 4.5: The left plot shows the predictive power of three different neural networks. The prediction task is evaluated on 1000 newly simulated test trials. The shaded region highlights one standard deviation from the mean squared error. The right plot shows the error curves of data-driven models and hybrid models trained with different numbers of simulated trajectories. The error is visualized in a logarithmic scale of base 10. Both plots are 10-step forward predictions with $\Delta t = 0.001$.

4.6 Discussion and Conclusion

In this work, we have demonstrated an effective deep learning framework for identifying Koopman eigenvalue and eigenfunction pairs for reconstructing the original nonlinear dynamics. In order to validate our method, we have carefully gone through three examples on which the analytical form of the Koopman eigen-decomposition can be derived. Our results show that (i) through automatic differentiation, the lie equation can be imposed via soft penalty to regularize the network, reducing the need of large training data sets as being used in previous works; (ii) since the framework is under the scope of operator learning, our model can be used for future state predictions on unseen initial states.

This work also suggests a number of open questions that motivate further investigation. For example, choosing the dimension of the latent space is tricky in practice as there could be trade-offs between computation and accuracy, studying reduced-order models would shed light on this matter; in addition, many nonlinear systems may have different Koopman

eigen-decompositions over different domains[179], being able to identify the boundaries of these domains would greatly expand the scope of applications of this approach. It is also interesting to see how these model can benefit the control problems, as in the real world, it is the ultimate goal of studying nonlinear dynamics.

BIBLIOGRAPHY

[1] Assyr Abdulle, E Weinan, Björn Engquist, and Eric Vanden-Eijnden. The heterogeneous multiscale method. *Acta Numerica*, 21:1–87, 2012.

[2] Mostafa Al-Gabalawy. Deep learning for koopman operator optimal control. *ISA transactions*, 2021.

[3] Md Zahangir Alom, Tarek M Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S Awwal, and Vijayan K Asari. The history began from alexnet: A comprehensive survey on deep learning approaches. *arXiv preprint arXiv:1803.01164*, 2018.

[4] Hassan Arbabi and Igor Mezic. Ergodic theory, dynamic mode decomposition, and computation of spectral properties of the koopman operator. *SIAM Journal on Applied Dynamical Systems*, 16(4):2096–2126, 2017.

[5] Travis Askham and J Nathan Kutz. Variable projection methods for an optimized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 17(1):380–416, 2018.

[6] Shervin Bagheri. Koopman-mode decomposition of the cylinder wake. *Journal of Fluid Mechanics*, 726:596–623, 2013.

[7] Bhavik R Bakshi and George Stephanopoulos. Wave-net: a multiresolution, hierarchical neural network with localized learning. *AIChE Journal*, 39(1):57–81, 1993.

[8] Mikhael Balabane, Miguel Alfonso Mendez, and Sara Najem. Koopman operator for burgers's equation. *Physical Review Fluids*, 6(6):064401, 2021.

[9] Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

[10] Andrzej Banburski, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Bob Liang, Jack Hidary, and Tomaso Poggio. Theory iii: Dynamics and generalization in deep networks. *arXiv preprint arXiv:1903.04991*, 2019.

[11] 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, 2019.

[12] Jérémy Basley, Luc R Pastur, Nathalie Delprat, and François Lusseyran. Space-time aspects of a three-dimensional multi-modulated open cavity flow. *Physics of Fluids*, 25(6):064105, 2013.

[13] Gabriele Bellani. *Experimental studies of complex flows through image-based techniques*. PhD thesis, KTH Royal Institute of Technology, 2011.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[16] 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.

[17] Erik Berger, Mark Sastuba, David Vogt, Bernhard Jung, and Heni Ben Amor. Dynamic mode decomposition for perturbation estimation in human robot interaction. In *The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, pages 593–600. IEEE, 2014.

[18] Erik Berger, Mark Sastuba, David Vogt, Bernhard Jung, and Heni Ben Amor. Estimation of perturbations in robotic behavior using dynamic mode decomposition. *Advanced Robotics*, 29(5):331–343, 2015.

[19] Marsha J Berger, Phillip Colella, et al. Local adaptive mesh refinement for shock hydrodynamics. *Journal of computational Physics*, 82(1):64–84, 1989.

[20] Chongke Bi, Ye Yuan, Jiawan Zhang, Yun Shi, Yiqing Xiang, Yuehuan Wang, and RongHui Zhang. Dynamic mode decomposition based video shot detection. *IEEE Access*, 6:21397–21407, 2018.

[21] Stephen A Billings. *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons, 2013.

[22] Robert Byron Bird, Robert Calvin Armstrong, and Ole Hassager. Dynamics of polymeric liquids. vol. 1: Fluid mechanics. 1987.

[23] Diana Alina Bistrian and Ionel Michael Navon. Randomized dynamic mode decomposition for nonintrusive reduced order modelling. *International Journal for Numerical Methods in Engineering*, 112(1):3–25, 2017.

[24] Josh Bongard and Hod Lipson. Automated reverse engineering of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 104(24):9943–9948, 2007.

[25] Lorenzo Boninsegna, Feliks Nüske, and Cecilia Clementi. Sparse learning of stochastic dynamical equations. *The Journal of chemical physics*, 148(24):241723, 2018.

[26] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*, 2014.

[27] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[28] Jason J Bramburger, Daniel Dylewsky, and J Nathan Kutz. Sparse identification of slow timescale dynamics. *arXiv preprint arXiv:2006.00940*, 2020.

[29] Jason J Bramburger and J Nathan Kutz. Poincaré maps for multiscale physics discovery and nonlinear floquet theory. *Physica D: Nonlinear Phenomena*, page 132479, 2020.

[30] Achi Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of computation*, 31(138):333–390, 1977.

[31] Bingni W Brunton, Lise A Johnson, Jeffrey G Ojemann, and J Nathan Kutz. Extracting spatial–temporal coherent patterns in large-scale neural recordings using dynamic mode decomposition. *Journal of neuroscience methods*, 258:1–15, 2016.

[32] S. L. Brunton, B. W. Brunton, J. L. Proctor, E. Kaiser, and J. N. Kutz. Chaos as an intermittently forced linear system. *Nature Communications*, 8(19):1–9, 2017.

[33] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[34] S. L. Brunton and B. R. Noack. Closed-loop turbulence control: Progress and challenges. *Applied Mechanics Reviews*, 67:050801–1–050801–48, 2015.

[35] S. L. Brunton and C. W. Rowley. Fast computation of FTLE fields for unsteady flows: a comparison of methods. *Chaos*, 20:017503, 2010.

[36] Steven L Brunton and J Nathan Kutz. *Data-driven science and engineering: Machine learning, dynamical systems, and control*. Cambridge University Press, 2019.

[37] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics*, 52:477–508, 2020.

[38] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[39] George D Byrne and Alan C Hindmarsh. Stiff ode solvers: A review of current and coming attractions. *Journal of Computational physics*, 70(1):1–62, 1987.

[40] Richard Car and Mark Parrinello. Unified approach for molecular dynamics and density-functional theory. *Physical review letters*, 55(22):2471, 1985.

[41] Kathleen Champion, Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Data-driven discovery of coordinates and governing equations. *Proceedings of the National Academy of Sciences*, 116(45):22445–22451, 2019.

[42] Bo Chang, Lili Meng, Eldad Haber, Frederick Tung, and David Begert. Multi-level residual networks from dynamical systems view. *arXiv preprint arXiv:1710.10348*, 2017.

[43] Haibin Chang and Dongxiao Zhang. Identification of physical processes via combined data-driven and data-assimilation methods. *Journal of Computational Physics*, 393:337–350, 2019.

[44] S Chen, SA Billings, CFN Cowan, and PM Grant. Non-linear systems identification using radial basis functions. *International Journal of Systems Science*, 21(12):2513–2539, 1990.

[45] Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583, 2018.

[46] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[47] T Ciodaro, D Deva, JM De Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. In *Journal of physics: conference series*, volume 368, page 012030. IOP Publishing, 2012.

[48] Julian D Cole. On a quasi-linear parabolic equation occurring in aerodynamics. *Quarterly of applied mathematics*, 9(3):225–236, 1951.

[49] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537, 2011.

[50] T. Colonius and K. Taira. A fast immersed boundary method using a nullspace approach and multi-domain far-field boundary conditions. *Computer Methods in Applied Mechanics and Engineering*, 197:2131–2146, 2008.

[51] Samuel Daniel Conte and Carl De Boor. *Elementary numerical analysis: an algorithmic approach*. SIAM, 2017.

[52] Richard Courant and David Hilbert. *Methods of Mathematical Physics: Partial Differential Equations*. John Wiley & Sons, 2008.

[53] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[54] Roger Daley. *Atmospheric data analysis*. Number 2. Cambridge university press, 1993.

[55] Ingrid Daubechies. *Ten lectures on wavelets*, volume 61. Siam, 1992.

[56] Scott TM Dawson, Maziar S Hemati, Matthew O Williams, and Clarence W Rowley. Characterizing and correcting for the effect of sensor noise in the dynamic mode decomposition. *Experiments in Fluids*, 57(3):42, 2016.

[57] Michael Dellnitz, Eyke Hüllermeier, Marvin Lücke, Sina Ober-Blöbaum, Christian Offen, Sebastian Peitz, and Karlson Pfannschmidt. Efficient time stepping for numerical integration using reinforcement learning. *arXiv preprint arXiv:2104.03562*, 2021.

[58] Craig C Douglas. Mgnet: a multigrid and domain decomposition network. *ACM SIGNUM Newsletter*, 27(4):2–8, 1992.

[59] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[60] Daniel Dylewsky, Molei Tao, and J Nathan Kutz. Dynamic mode decomposition for multiscale nonlinear physics. *Physical Review E*, 99(6):063311, 2019.

[61] W. E and J. Lu. Multiscale modeling. *Scholarpedia*, 6(10):11527, 2011. revision #91540.

[62] Wayne H Enright, TE Hull, and Bengt Lindberg. Comparing numerical methods for stiff systems of ode: s. *BIT Numerical Mathematics*, 15(1):10–48, 1975.

[63] N Benjamin Erichson, Steven L Brunton, and J Nathan Kutz. Compressed dynamic mode decomposition for background modeling. *Journal of Real-Time Image Processing*, 16(5):1479–1492, 2019.

[64] N Benjamin Erichson, Lionel Mathelin, J Nathan Kutz, and Steven L Brunton. Randomized dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 18(4):1867–1891, 2019.

[65] Zhiwei Fang and Justin Zhan. A physics-informed neural network framework for pdes on 3d surfaces: Time independent problems. *IEEE Access*, 8:26328–26335, 2019.

[66] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

[67] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A computational method to extract macroscopic variables and their dynamics in multiscale systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1816–1846, 2014.

[68] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A computational method to extract macroscopic variables and their dynamics in multiscale systems. *SIAM Journal on Applied Dynamical Systems*, 13(4):1816–1846, 2014.

[69] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A trajectory-free framework for analysing multiscale systems. *Physica D: Nonlinear Phenomena*, 328:34–43, 2016.

[70] Gary Froyland, Georg A Gottwald, and Andy Hammerlindl. A trajectory-free framework for analysing multiscale systems. *Physica D: Nonlinear Phenomena*, 328:34–43, 2016.

[71] Han Gao, Luning Sun, and Jian-Xun Wang. Phygeonet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state pdes on irregular domain. *Journal of Computational Physics*, 428:110079, 2021.

[72] Michael Gavin. The stock market and exchange rate dynamics. *Journal of international money and finance*, 8(2):181–200, 1989.

[73] Craig Gin, Bethany Lusch, Steven L Brunton, and J Nathan Kutz. Deep learning models for global coordinate transformations that linearize PDEs. *arXiv preprint arXiv:1911.02710*, 2019.

[74] Craig Gin, Bethany Lusch, Steven L Brunton, and J Nathan Kutz. Deep learning models for global coordinate transformations that linearise pdes. *European Journal of Applied Mathematics*, 32(3):515–539, 2021.

[75] R Gonzalez-Garcia, R Rico-Martinez, and IG Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & chemical engineering*, 22:S965–S968, 1998.

[76] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[77] J Graham, K Kanov, XIA Yang, M Lee, N Malaya, CC Lalescu, R Burns, G Eyink, A Szalay, RD Moser, et al. A web services accessible database of turbulent channel flow and its use for testing a new integral wall model for LES. *Journal of Turbulence*, 17(2):181–215, 2016.

[78] Emad M Grais, Hagen Wierstorf, Dominic Ward, and Mark D Plumbley. Multi-resolution fully convolutional neural networks for monaural audio source separation. In *International Conference on Latent Variable Analysis and Signal Separation*, pages 340–350. Springer, 2018.

[79] Leslie Greengard and Vladimir Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 135(2):280–292, 1997.

[80] Jacob Grosek and J Nathan Kutz. Dynamic mode decomposition for real-time background/foreground separation in video. *arXiv preprint arXiv:1404.7592*, 2014.

[81] John Guckenheimer and Philip Holmes. *Nonlinear oscillations, dynamical systems, and bifurcations of vector fields*, volume 42. Springer Science & Business Media, 2013.

[82] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

[83] Yiqiang Han, Wenjian Hao, and Umesh Vaidya. Deep learning of koopman representation for control. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1890–1895. IEEE, 2020.

[84] Juncai He and Jinchao Xu. Mgnet: A unified framework of multigrid and convolutional neural network. *Science china mathematics*, 62(7):1331–1354, 2019.

[85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[86] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[87] Moritz Helmstaedter, Kevin L Briggman, Srinivas C Turaga, Viren Jain, H Sebastian Seung, and Winfried Denk. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174, 2013.

[88] Maziar S Hemati, Clarence W Rowley, Eric A Deem, and Louis N Cattafesta. Debiasing the dynamic mode decomposition for applied koopman spectral analysis of noisy datasets. *Theoretical and Computational Fluid Dynamics*, 31(4):349–368, 2017.

[89] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. Nvidia simnet™: An ai-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pages 447–461. Springer, 2021.

[90] Jan S Hesthaven, Gianluigi Rozza, Benjamin Stamm, et al. *Certified reduced basis methods for parametrized partial differential equations*, volume 590. Springer, 2016.

[91] Francis Begnaud Hildebrand. *Introduction to numerical analysis*. Courier Corporation, 1987.

[92] BL Ho and Rudolf E Kálmán. Effective construction of linear state-variable models from input/output functions. *at-Automatisierungstechnik*, 14(1-12):545–548, 1966.

[93] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[94] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[95] Eberhard Hopf. The partial differential equation u sub t+ uu sub x= mu sub xx. Technical report, INDIANA UNIV AT BLOOMINGTON, 1950.

[96] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[97] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[98] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[99] Jörn-Henrik Jacobsen, Edouard Oyallon, Stéphane Mallat, and Arnold WM Smeulders. Multiscale hierarchical convolutional networks. *arXiv preprint arXiv:1703.04140*, 2017.

[100] Herbert Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach*, volume 5. GMD-Forschungszentrum Informationstechnik Bonn, 2002.

[101] Ameya D Jagtap and George Em Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. *Communications in Computational Physics*, 28(5):2002–2041, 2020.

[102] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[103] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020.

[104] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[105] Mihailo R Jovanović, Peter J Schmid, and Joseph W Nichols. Sparsity-promoting dynamic mode decomposition. *Physics of Fluids*, 26(2):024103, 2014.

[106] Jer-Nan Juang. System identification. *fvms*, 5:119–134, 1993.

[107] Jer-Nan Juang and Richard S Pappa. An eigensystem realization algorithm for modal parameter identification and model reduction. *Journal of guidance, control, and dynamics*, 8(5):620–627, 1985.

[108] Jer-Nan Juang, Minh Phan, Lucas G Horta, and Richard W Longman. Identification of observer/kalman filter markov parameters-theory and experiments. *Journal of Guidance, Control, and Dynamics*, 16(2):320–329, 1993.

[109] Mason Kamb, Eurika Kaiser, Steven L Brunton, and J Nathan Kutz. Time-delay observables for koopman: Theory and applications. *SIAM Journal on Applied Dynamical Systems*, 19(2):886–917, 2020.

[110] Kalin Kanov, Randal Burns, Cristian Lalescu, and Gregory Eyink. The johns hopkins turbulence databases: an open simulation laboratory for turbulence research. *Computing in Science & Engineering*, 17(5):10–17, 2015.

[111] Robert E Kass, Shun-Ichi Amari, Kensuke Arai, Emery N Brown, Casey O Diekman, Markus Diesmann, Brent Doiron, Uri T Eden, Adrienne L Fairhall, Grant M Fiddyment, et al. Computational neuroscience: Mathematical and statistical perspectives. *Annual review of statistics and its application*, 5:183–214, 2018.

[112] Tsung-Wei Ke, Michael Maire, and Stella X Yu. Multigrid neural architectures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6665–6673, 2017.

[113] I. G. Kevrekidis, C. W. Gear, J. M. Hyman, P. G. Kevrekidis, O. Runborg, and C. Theodoropoulos. Equation-free, coarse-grained multiscale computation: Enabling microscopic simulators to perform system-level analysis. *Communications in Mathematical Science*, 1(4):715–762, 2003.

[114] Ioannis G Kevrekidis, C William Gear, James M Hyman, Panagiotis G Kevrekidid, Olof Runborg, Constantinos Theodoropoulos, et al. Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences*, 1(4):715–762, 2003.

[115] Ioannis G Kevrekidis, C William Gear, James M Hyman, Panagiotis G Kevrekidid, Olof Runborg, Constantinos Theodoropoulos, and others. Equation-free, coarse-grained multiscale computation: Enabling mocroscopic simulators to perform system-level analysis. *Communications in Mathematical Sciences*, 1(4):715–762, 2003.

[116] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *Computer Methods in Applied Mechanics and Engineering*, 374:113547, 2021.

[117] Byungjoo Kim, Bryce Chudomelka, Jinyoung Park, Jaewoo Kang, Youngjoon Hong, and Hyunwoo J Kim. Robust neural networks inspired by strong stability preserving runge-kutta methods. In *European Conference on Computer Vision*, pages 416–432. Springer, 2020.

[118] Byungsoo Kim, Vinicius C Azevedo, Nils Thuerey, Theodore Kim, Markus Gross, and Barbara Solenthaler. Deep fluids: A generative network for parameterized fluid simulations. In *Computer Graphics Forum*, volume 38, pages 59–70. Wiley Online Library, 2019.

[119] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[120] Juš Kocijan, Agathe Girard, Blaž Banko, and Roderick Murray-Smith. Dynamic systems identification with gaussian processes. *Mathematical and Computer Modelling of Dynamical Systems*, 11(4):411–424, 2005.

[121] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[122] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[123] J Nathan Kutz. *Data-driven modeling & scientific computation: methods for complex systems & big data*. Oxford University Press, 2013.

[124] J Nathan Kutz, Steven L Brunton, Bingni W Brunton, and Joshua L Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems*. SIAM, 2016.

[125] J Nathan Kutz, Xing Fu, and Steven L Brunton. Multiresolution dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 15(2):713–735, 2016.

[126] Henning Lange, Steven L Brunton, and Nathan Kutz. From fourier to koopman: Spectral methods for long-term time series prediction. *arXiv preprint arXiv:2004.00574*, 2020.

[127] Soledad Le Clainche and José M Vega. Higher order dynamic mode decomposition. *SIAM Journal on Applied Dynamical Systems*, 16(2):882–925, 2017.

[128] Yann Lecun. Une procedure d'apprentissage pour reseau a seuil asymmetrique (a learning scheme for asymmetric threshold networks). In *Proceedings of Cognitiva 85, Paris, France*, pages 599–604, 1985.

[129] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[130] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[131] Michael KK Leung, Hui Yuan Xiong, Leo J Lee, and Brendan J Frey. Deep learning of the tissue-regulated splicing code. *Bioinformatics*, 30(12):i121–i129, 2014.

[132] Qianxiao Li, Felix Dietrich, Erik M Bollt, and Ioannis G Kevrekidis. Extended dynamic mode decomposition with dictionary learning: A data-driven adaptive spectral decomposition of the koopman operator. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(10):103111, 2017.

[133] Qianxiao Li, Cheng Tai, et al. Stochastic modified equations and adaptive stochastic gradient algorithms. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2101–2110. JMLR. org, 2017.

[134] Qianxiao Li, Cheng Tai, and E Weinan. Dynamics of stochastic gradient algorithms. *arXiv preprint arXiv:1511.06251*, 2015.

[135] Qianxiao Li, Cheng Tai, and E Weinan. Stochastic modified equations and dynamics of stochastic gradient algorithms i: Mathematical foundations. *Journal of Machine Learning Research*, 20(40):1–40, 2019.

[136] Yi Li, Eric Perlman, Minping Wan, Yunke Yang, Charles Meneveau, Randal Burns, Shiyi Chen, Alexander Szalay, and Gregory Eyink. A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence. *Journal of Turbulence*, (9):N31, 2008.

[137] Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. *arXiv preprint arXiv:1910.08264*, 2019.

[138] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[139] Yuying Liu, Colin Ponce, Steven L Brunton, and J Nathan Kutz. Multiresolution convolutional autoencoders. *arXiv preprint arXiv:2004.04946*, 2020.

[140] Jean-Christophe Loiseau and Steven L Brunton. Constrained sparse galerkin regression. *Journal of Fluid Mechanics*, 838:42–67, 2018.

[141] Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

[142] Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. *arXiv preprint arXiv:1710.09668*, 2017.

[143] Richard W Longman and Jer-Nan Juang. Recursive form of the eigensystem realization algorithm for system identification. *Journal of Guidance, Control, and Dynamics*, 12(5):647–652, 1989.

[144] 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.

[145] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G Johnson. Physics-informed neural networks with hard constraints for inverse design. *arXiv preprint arXiv:2102.04626*, 2021.

[146] Zhixin Lu, Brian R Hunt, and Edward Ott. Attractor reconstruction by machine learning. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(6):061104, 2018.

[147] D. M. Luchtenburg, S. L. Brunton, and C. W. Rowley. Long-time uncertainty propagation using generalized polynomial chaos and flow map composition. *Journal of Computational Physics*, 274:783–802, 2014.

[148] Mantas Lukoševičius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[149] Bethany Lusch, J Nathan Kutz, and Steven L Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nature communications*, 9(1):1–10, 2018.

[150] Chao Ma, Qingcan Wang, et al. A priori estimates of the population risk for residual networks. *arXiv preprint arXiv:1903.02154*, 2019.

[151] Chao Ma, Lei Wu, et al. A priori estimates for two-layer neural networks. *arXiv preprint arXiv:1810.06397*, 2018.

[152] Chao Ma, Lei Wu, et al. Barron spaces and the compositional function spaces for neural network models. *arXiv preprint arXiv:1906.08039*, 2019.

[153] Chao Ma, Lei Wu, et al. Machine learning from a continuous viewpoint. *arXiv preprint arXiv:1912.12777*, 2019.

[154] Junshui Ma, Robert P Sheridan, Andy Liaw, George E Dahl, and Vladimir Svetnik. Deep neural nets as a method for quantitative structure–activity relationships. *Journal of chemical information and modeling*, 55(2):263–274, 2015.

[155] Suryanarayana Maddu, Dominik Sturm, Christian L Müller, and Ivo F Sbalzarini. Inverse dirichlet weighting enables reliable training of physics informed neural networks. *Machine Learning: Science and Technology*, 2021.

[156] Stéphane Mallat. Understanding deep convolutional networks. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150203, 2016.

[157] Stephane Georges Mallat. Multiresolution representations and wavelets. 1988.

[158] Niall M Mangan, J Nathan Kutz, Steven L Brunton, and Joshua L Proctor. Model selection for dynamical systems via sparse regression and information criteria. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2204):20170009, 2017.

[159] Jordan Mann and J Nathan Kutz. Dynamic mode decomposition for financial trading strategies. *Quantitative Finance*, 16(11):1643–1655, 2016.

[160] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using convolutional auto-encoders with symmetric skip connections. *arXiv preprint arXiv:1606.08921*, 2016.

[161] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. VAMPnets: Deep learning of molecular kinetics. *Nature Communications*, 9(5), 2018.

[162] Andreas Mardt, Luca Pasquali, Hao Wu, and Frank Noé. Vampnets for deep learning of molecular kinetics. *Nature communications*, 9(1):1–11, 2018.

[163] Stephen F McCormick. *Multigrid methods*. SIAM, 1987.

[164] Xuhui Meng and George Em Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. *Journal of Computational Physics*, 401:109020, 2020.

[165] Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 196–201. IEEE, 2011.

[166] Michele Milano and Petros Koumoutsakos. Neural network modeling for near wall turbulent flow. *Journal of Computational Physics*, 182(1):1–26, 2002.

[167] Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes. *arXiv preprint arXiv:2006.16144*, 2020.

[168] Yoshinori Mizuno, Daniel Duke, Callum Atkinson, and Julio Soria. Investigation of wall-bounded turbulent flow using dynamic mode decomposition. In *Journal of Physics: Conference Series*, volume 318, page 042040. IOP Publishing, 2011.

[169] Jeremy Morton, Freddie D Witherden, Antony Jameson, and Mykel J Kochenderfer. Deep dynamical modeling and control of unsteady fluid flows. *arXiv preprint arXiv:1805.07472*, 2018.

[170] Jeremy Morton, Freddie D Witherden, and Mykel J Kochenderfer. Deep variational koopman models: Inferring koopman observations for uncertainty-aware dynamics modeling and control. *arXiv preprint arXiv:1902.09742*, 2019.

[171] Michael Muehlebach and Michael I Jordan. A dynamical systems perspective on nesterov acceleration. *arXiv preprint arXiv:1905.07436*, 2019.

[172] Michael Muehlebach and Michael I Jordan. Optimization with momentum: Dynamical, control-theoretic, and symplectic perspectives. *arXiv preprint arXiv:2002.12493*, 2020.

[173] Navid Naderi and Babak Nasersharif. Multiresolution convolutional neural network for robust speech recognition. In *2017 Iranian Conference on Electrical Engineering (ICEE)*, pages 1459–1464. IEEE, 2017.

[174] J Nathan Kutz, Joshua L Proctor, and Steven L Brunton. Applied koopman theory for partial differential equations and data-driven modeling of spatio-temporal systems. *Complexity*, 2018, 2018.

[175] Frank Noé and Feliks Nuske. A variational approach to modeling slow processes in stochastic dynamical systems. *Multiscale Modeling Simulation*, 11(2):635–655, 2013.

[176] Feliks Nüske, Bettina G Keller, Guillermo Pérez-Hernández, Antonia SJS Mey, and Frank Noé. Variational approach to molecular kinetics. *Journal of chemical theory and computation*, 10(4):1739–1752, 2014.

[177] Hans C Öttinger. *Stochastic processes in polymeric fluids: tools and examples for developing simulation algorithms*. Springer Science & Business Media, 2012.

[178] Jacob Page and Rich R Kerswell. Koopman analysis of burgers equation. *Physical Review Fluids*, 3(7):071901, 2018.

[179] Jacob Page and Rich R Kerswell. Koopman mode expansions between simple invariant solutions. *Journal of Fluid Mechanics*, 879:1–27, 2019.

[180] Shaowu Pan and Karthik Duraisamy. Data-driven discovery of closure models. *SIAM Journal on Applied Dynamical Systems*, 17(4):2381–2413, 2018.

[181] Shaowu Pan and Karthik Duraisamy. Long-time predictive modeling of nonlinear dynamical systems using neural networks. *Complexity*, 2018, 2018.

[182] Shaowu Pan and Karthik Duraisamy. On the structure of time-delay embedding in linear models of non-linear dynamical systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 30(7):073135, 2020.

[183] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[184] Eric J Parish and Kevin T Carlberg. Time-series machine-learning error models for approximate solutions to parameterized dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 365:112990, 2020.

[185] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[186] Jaideep Pathak, Brian Hunt, Michelle Girvan, Zhixin Lu, and Edward Ott. Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach. *Physical review letters*, 120(2):024102, 2018.

[187] Jaideep Pathak, Zhixin Lu, Brian R Hunt, Michelle Girvan, and Edward Ott. Using machine learning to replicate chaotic attractors and calculate lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12):121102, 2017.

[188] Jaideep Pathak, Alexander Wikner, Rebeckah Fussell, Sarthak Chandra, Brian R Hunt, Michelle Girvan, and Edward Ott. Hybrid forecasting of chaotic processes: Using machine learning in conjunction with a knowledge-based model. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 28(4):041101, 2018.

[189] Xi Peng, Rogerio S Feris, Xiaoyu Wang, and Dimitris N Metaxas. Red-net: A recurrent encoder–decoder network for video-based face alignment. *International Journal of Computer Vision*, 126(10):1103–1119, 2018.

[190] Eric Perlman, Randal Burns, Yi Li, and Charles Meneveau. Data exploration of turbulence simulations using a database cluster. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, page 23. ACM, 2007.

[191] Minh Phan, Lucas G Horta, Jer-Nan Juang, and Richard W Longman. Linear system identification via an asymptotically stable observer. *Journal of Optimization Theory and Applications*, 79(1):59–86, 1993.

[192] Minh Phan, Jer-Nan Juang, and Richard Longman. Identification of linear multivariable systems from a single set of data by identification of observers with assigned real eigenvalues. In *32nd Structures, Structural Dynamics, and Materials Conference*, page 949, 1991.

[193] Tomaso Poggio, Andrzej Banburski, and Qianli Liao. Theoretical issues in deep networks: Approximation, optimization and generalization. *arXiv preprint arXiv:1908.09375*, 2019.

[194] Tomaso Poggio, Hrushikesh Mhaskar, Lorenzo Rosasco, Brando Miranda, and Qianli Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[195] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in neural information processing systems*, pages 3360–3368, 2016.

[196] Joshua L Proctor, Steven L Brunton, and J Nathan Kutz. Dynamic mode decomposition with control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[197] Joshua L Proctor and Philip A Eckhoff. Discovering dynamic patterns from infectious disease data using dynamic mode decomposition. *International health*, 7(2):139–145, 2015.

[198] Tong Qin, Kailiang Wu, and Dongbin Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[199] Alfio Quarteroni, Gianluigi Rozza, et al. *Reduced order methods for modeling and computational reduction*, volume 9. Springer, 2014.

[200] M Raissi, P Perdikaris, and GE 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, 2019.

[201] Maziar Raissi. Deep hidden physics models: Deep learning of nonlinear partial differential equations. *The Journal of Machine Learning Research*, 19(1):932–955, 2018.

[202] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.

[203] Maziar Raissi, Paris Perdikaris, and George 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, 2019.

[204] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, 335:736–746, 2017.

[205] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.

[206] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10561*, 2017.

[207] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[208] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.

[209] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

[210] Francesco Regazzoni, Luca Dedè, and Alfio Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *J. Comp. Phys.*, 397:108852, 2019.

[211] Clarence W Rowley, Igor Mezić, Shervin Bagheri, Philipp Schlatter, and Dan S Henningson. Spectral analysis of nonlinear flows. *Journal of fluid mechanics*, 641:115–127, 2009.

[212] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[213] Samuel H Rudy, J Nathan Kutz, and Steven L Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396:483–506, 2019.

[214] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[215] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for lvcsr. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8614–8618. IEEE, 2013.

[216] Hayden Schaeffer. Learning partial differential equations via data discovery and sparse optimization. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 473(2197):20160446, 2017.

[217] Hayden Schaeffer and Scott G McCalla. Sparse model selection via integral terms. *Physical Review E*, 96(2):023302, 2017.

[218] Hayden Schaeffer, Giang Tran, and Rachel Ward. Extracting sparse high-dimensional dynamics from limited data. *SIAM Journal on Applied Mathematics*, 78(6):3279–3295, 2018.

[219] Hayden Schaeffer, Giang Tran, Rachel Ward, and Linan Zhang. Extracting structured dynamical systems using sparse optimization with very few samples. *arXiv preprint arXiv:1805.04158*, 2018.

[220] Peter J Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of fluid mechanics*, 656:5–28, 2010.

[221] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[222] Michael D Schmidt, Ravishankar R Vallabhajosyula, Jerry W Jenkins, Jonathan E Hood, Abhishek S Soni, John P Wikswo, and Hod Lipson. Automated refinement and inference of analytical models for metabolic networks. *Physical biology*, 8(5):055011, 2011.

[223] CB Scott and Eric Mjolsness. Multilevel artificial neural network training for spatially correlated learning. *SIAM Journal on Scientific Computing*, 41(5):S297–S320, 2019.

[224] A Senior, V Vanhoucke, P Nguyen, T Sainath, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 2012.

[225] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *arXiv preprint arXiv:2004.01806*, 2020.

[226] Khemraj Shukla, Ameya D Jagtap, and George Em Karniadakis. Parallel physics-informed neural networks via domain decomposition. *arXiv preprint arXiv:2104.10013*, 2021.

[227] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[228] Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.

[229] Luning Sun and Jian-Xun Wang. Physics-constrained bayesian neural network for fluid flow reconstruction with sparse and noisy data. *Theoretical and Applied Mechanics Letters*, 10(3):161–169, 2020.

[230] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[231] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[232] Ellad B Tadmor, Michael Ortiz, and Rob Phillips. Quasicontinuum analysis of defects in solids. *Philosophical magazine A*, 73(6):1529–1563, 1996.

[233] K. Taira and T. Colonius. The immersed boundary method: a projection approach. *Journal of Computational Physics*, 225(2):2118–2137, 2007.

[234] Kunihiko Taira, Steven L Brunton, Scott Dawson, Clarence W Rowley, Tim Colonius, Beverley J McKeon, Oliver T Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S Ukeiley. Modal analysis of fluid flows: An overview. *AIAA Journal*, 55(12):4013–4041, 2017.

[235] Kunihiko Taira, Maziar S Hemati, Steven L Brunton, Yiyang Sun, Karthik Duraisamy, Shervin Bagheri, Scott Dawson, and Chi-An Yeh. Modal analysis of fluid flows: Applications and outlook. *AIAA Journal*, 58(3):998–1022, 2020.

[236] Naoya Takeishi, Yoshinobu Kawahara, and Takehisa Yairi. Learning koopman invariant subspaces for dynamic mode decomposition. *arXiv preprint arXiv:1710.04340*, 2017.

[237] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[238] Roy Taylor, J Nathan Kutz, Kyle Morgan, and Brian A Nelson. Dynamic mode decomposition for plasma diagnostics and validation. *Review of Scientific Instruments*, 89(5):053501, 2018.

[239] Hiroaki Terao, Sho Shirasaka, and Hideyuki Suzuki. Extended dynamic mode decomposition with dictionary learning using neural ordinary differential equations. *Nonlinear Theory and Its Applications, IEICE*, 12(4):626–638, 2021.

[240] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[241] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in neural information processing systems*, pages 1799–1807, 2014.

[242] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2006.

[243] Giang Tran and Rachel Ward. Exact recovery of chaotic systems from highly corrupted data. *Multiscale Modeling & Simulation*, 15(3):1108–1129, 2017.

[244] Ulrich Trottenberg, Cornelius W Oosterlee, and Anton Schuller. *Multigrid*. Elsevier, 2000.

[245] Jonathan H Tu, Clarence W Rowley, Dirk M Luchtenburg, Steven L Brunton, and J Nathan Kutz. On dynamic mode decomposition: Theory and applications. *arXiv preprint arXiv:1312.0041*, 2013.

[246] Pantelis R Vlachas, Wonmin Byeon, Zhong Y Wan, Themistoklis P Sapsis, and Petros Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2213):20170844, 2018.

[247] Zhong Yi Wan, Pantelis Vlachas, Petros Koumoutsakos, and Themistoklis Sapsis. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PloS one*, 13(5), 2018.

[248] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed deeponets. *arXiv preprint arXiv:2106.05384*, 2021.

[249] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[250] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *arXiv preprint arXiv:2103.10974*, 2021.

[251] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449:110768, 2022.

[252] Wen-Xu Wang, Rui Yang, Ying-Cheng Lai, Vassilios Kovanis, and Celso Grebogi. Predicting catastrophes in nonlinear dynamical systems by compressive sensing. *Physical review letters*, 106(15):154101, 2011.

[253] Arieh Warshel and Michael Levitt. Theoretical studies of enzymic reactions: dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. *Journal of molecular biology*, 103(2):227–249, 1976.

[254] Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of chemical physics*, 148(24):241703, 2018.

[255] Christoph Wehmeyer and Frank Noé. Time-lagged autoencoders: Deep learning of slow collective variables for molecular kinetics. *The Journal of Chemical Physics*, 148(241703):1–9, 2018.

[256] E Weinan. *Principles of multiscale modeling*. Cambridge University Press, 2011.

[257] E Weinan. *Principles of multiscale modeling*. Cambridge University Press, 2011.

[258] E Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[259] E Weinan and Bjorn Engquist. Multiscale modeling and computation. *Notices of the AMS*, 50(9):1062–1070, 2003.

[260] E Weinan, Bjorn Engquist, Xiantao Li, Weiqing Ren, and Eric Vanden-Eijnden. Heterogeneous multiscale methods: a review. *Communications in computational physics*, 2(3):367–450, 2007.

[261] E Weinan, Bjorn Engquist, and others. The heterogeneous multiscale methods. *Communications in Mathematical Sciences*, 1(1):87–132, 2003.

[262] E Weinan, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10, 2019.

[263] Paul John Werbos. *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, volume 1. John Wiley & Sons, 1994.

[264] Steffen Wiewel, Moritz Becher, and Nils Thuerey. Latent space physics: Towards learning the temporal evolution of fluid flow. In *Computer Graphics Forum*, volume 38, pages 71–82. Wiley Online Library, 2019.

[265] Stephen Wiggins. *Introduction to applied nonlinear dynamical systems and chaos*, volume 2. Springer Science & Business Media, 2003.

[266] Matthew O Williams, Ioannis G Kevrekidis, and Clarence W Rowley. A data–driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25(6):1307–1346, 2015.

[267] Zongmin Wu and Ran Zhang. Learning physics by data for the motion of a sphere falling in a non-newtonian fluid. *Communications in Nonlinear Science and Numerical Simulation*, 67:577–593, 2019.

[268] Hui Y Xiong, Babak Alipanahi, Leo J Lee, Hannes Bretschneider, Daniele Merico, Ryan KC Yuen, Yimin Hua, Serge Gueroussov, Hamed S Najafabadi, Timothy R Hughes, et al. The human splicing code reveals new insights into the genetic determinants of disease. *Science*, 347(6218):1254806, 2015.

[269] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[270] Liu Yang, Xuhui Meng, and George Em Karniadakis. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. *Journal of Computational Physics*, 425:109913, 2021.

[271] Liu Yang, Dongkun Zhang, and George Em Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

[272] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.

[273] Enoch Yeung, Soumya Kundu, and Nathan Hodas. Learning deep neural network representations for koopman operators of nonlinear dynamical systems. In *2019 American Control Conference (ACC)*, pages 4832–4839. IEEE, 2019.

[274] PK Yeung, DA Donzis, and KR Sreenivasan. Dissipation, enstrophy and pressure statistics in turbulence simulations at high reynolds numbers. *Journal of Fluid Mechanics*, 700:5–15, 2012.

[275] L. Ying and E. J. Candès. The phase flow method. *Journal of Computational Physics*, 220:184–215, 2006.

[276] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[277] Dongkun Zhang, Ling Guo, and George Em Karniadakis. Learning in modal space: Solving time-dependent stochastic pdes using physics-informed neural networks. *SIAM Journal on Scientific Computing*, 42(2):A639–A665, 2020.

[278] Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

[279] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[280] Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[281] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[282] Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Luján, Valentin Sulzer, Ashutosh Bharambe, et al. Neuralpde: Automating physics-informed neural networks (pinns) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

Appendix A

HIERARCHICAL DEEP LEARNING OF MULTISCALE NEURAL NETWORK TIME-STEPPERS

A.1 Methods and data

A.1.1 Multiscale neural network HiTS

The setup of our multiscale neural network HiTS relies on a sequence of successfully trained, single time-scale neural network time-steppers. Here, we document the working details. For all systems, we first specify a domain of interest $\mathcal{D}$ in the state space, over which we uniformly sample the initial states of training, validating, and testing trajectories. For all systems except for the Lorenz system, the time steps of the single time-scale neural network time-steppers are chosen to be 0.01, 0.02, 0.04, 0.08, 0.16, 0.32, 0.64, 1.28, 2.56, 5.12 and 10.24.

Systems	Samples (train/ validate/test)	Sampled region $\mathcal{D}$ in state space	Network architectures	Cross-validated NNTSs
Hyperbolic	1600/320/320	$[-1,1]^2$	[2, 128, 128, 128, 2]	3 - 10
Cubic oscillator	3200/320/320	$[-1,1]^2$	[2, 256, 256, 256, 2]	1 - 4
Van der Pol	3200/320/320	$[-2,2] \times [-4,4]$	[2, 512, 512, 512, 2]	2 - 5
Hopf bifurcation	3200/320/320	$[-0.2,0.6] \times [-1,2] \times [-1,1]$	[3, 128, 128, 128, 3]	2 - 11
Lorenz	6400/640/640	$[-0.1,0.1]^3$	[3, 1024, 1024, 1024, 3]	6 - 8

Table A.1: Parameters and setups for the neural network time-steppers.

Validating and testing trajectories are different from the training trajectories and they last 51.20 time units. For the Lorenz system, the process is slightly different. We simulate 3 long trajectories to form training, validating, and testing data sets (one for each). For each of these 3 trajectories, we cut off the initial 5 time units to make sure the collected data are on the attractor. 11 neural network time-steppers are trained for the Lorenz system as well; however, they have time steps of 0.0005, 0.001, 0.002, 0.004, 0.008, 0.016, 0.032, 0.064, 0.128, 0.256 and 0.512 due to the inherently chaotic dynamics. The validating and testing trajectories last 2.56 time units. For convenience, we term these single time-scale neural network time-steppers NNTS 0 - NNTS 10 within each individual experiment. In our experiments, all trajectory data are simulated with the **odeint** function provided by the **Scipy** package and are considered as the ground truth. Upon training, the number of forward steps p is set to 5, and we use the Python API for the **PyTorch** framework and the Adam optimizer with a learning rate of $1e-3$. The training ends when the maximum epoch is reached (which is set to 100000) or when the mean squared error on one-step prediction is lower than $1e-8$. Upon evaluation, we use the procedure presented in 2.3.3, coupling the cross-validated HiTSs to perform testing. The number of training/validating/testing samples, regions of interest, network architectures, and indices of cross validated neural network time-steppers are shown in Table A.1. We note that here the network size is fixed across all scales for each particular example, although this isn't necessary. In fact, we observe that width and depth of the network are not sensitive parameters for this method as long as the network capacity is large enough to represent the flow map. In practice, we suggest that the training error is a good indicator of how well the neural network represents the flow map. We recommend larger, both deeper and wider, networks for approximating more complex flow maps, including maps with larger temporal gaps and those coming from more complex systems. This complexity is why we have created larger networks for learning the Lorenz system. In addition, the ResNet is adapted as the base structure and the ReLU activation function is used, which are in alignment with [85]; however, fully-connected networks also perform well, wince they can also be used for universal function approximations. Other nonlinear activation functions, such as exponential linear units were also tested, although there were no significant changes of the results.

Sequences	**HiTS**	**LSTM**	**ESN**	**CW-RNN**
Fluid flow	15388	15408	15400	15622
Video frame	31216	30976	31360	31560
Music data	1020416	1018368	1024000	1042328
KS equation	4069760	4091232	4070400	4076232

Table A.2: Number of parameters for different architectures of the sequence generation experiments.

A.1.2 Hybrid time-steppers

We benchmark our proposed hybrid time-stepper against a commonly used numerical time-stepper, a fourth-order Runge-Kutta integrator with uniform step size. For the first four nonlinear system examples, we simulate with step sizes of 0.01, 0.02, 0.04, 0.08 and 0.16. For the Lorenz system, we shrink the sizes accordingly to 0.0005, 0.001, 0.002, 0.004 and 0.008. For convenience, we term them RK 0 - RK 4 for each individual experiment. For simplicity, our hybrid time-steppers are constructed to be the same Runge-Kutta time-steppers (i.e. RK 0 - RK 4) associated with one single time-scale neural-network time-stepper. The step size of this neural network time-stepper is set to be 0.512 for Lorenz and 10.24 for others. Similarly, we term them Hybrid 0 - Hybrid 4 for convenience.

A.1.3 Details for sequence generation examples

In Section 2.4.2, we benchmark our multiscale HiTS against long short-term memory networks (LSTMs), echo state networks (ESNs), and clockwork recurrent neural networks (CW-RNNs) on sequence generation tasks over four data sets: the Kuramoto–Sivashinsky (KS) equation, a music snippet from Fugue No. 1 In C Major, BWV 846, an animation of fluid flow passing a cylinder, and a video frame of blooming flowers.

Information for these data sets and preprocessing steps are as follows:

- For the KS equation, 4001 snapshots on 512 evenly distributed spatial points over the interval $(0, 16\pi)$ are simulated with a spectral method. These data are used to train

Sequences	**Network architectures in multiscale HiTS**
Fluid flow	[22, 256, 22], [22, 64, 22], [22, 16, 22], [22, 4, 22]
Video frame	[64, 128, 64], [64, 64, 64], [64, 32, 64], [64, 16, 64]
Music data	[128, 2048, 128], [128, 1024, 128], [128, 512, 128], [128, 256, 128], [128, 128, 128]
KS equation	[512, 2048, 512], [512, 1024 ,512], [512, 512, 512], [512, 256, 512], [512, 128, 512]

Table A.3: Network architectures in multiscale HiTS for sequence generation experiments.

different network architectures.

- For the music data, we sample at 1102 Hz for 2 seconds. Then, a time-delay embedding with 128 delays is applied to produce a richer feature space, resulting in a training tensor of size 128×2077.

- For the simulated fluid flow example, data is generated using the immersed boundary projection method (IBPM) [233, 50] at a Reynolds number of 100; details on the numerical simulation may be found in [33]. We apply principal component analysis (PCA) to the streamwise velocity field u, and use the dynamics on the first 22 modes to train the neural networks.

- The video frame of blooming flowers consists of 175 image snapshots of 540×960 pixels and 3 channels (RGB). Similar to the fluid flow data, we preprocess it by applying PCA and work with its reduced representation in the first 64 PC dimensions.

For each sequence, we setup different types of neural network architectures so that they have about the same number of trainable parameters; see Table A.2 for the summary of number of network parameters. The specific procedures are as follows:

- We first set up our multiscale neural network time-steppers. For the fluid flow and video frame examples, we use 4 neural network time-steppers with step sizes 1, 4, 16, and 64, respectively. For the music data, we use 5 time-steppers and the corresponding step sizes are 1, 5, 25, 125, and 625. For the KS example, we use 5 time-steppers with step sizes of 1, 6, 36, 216, and 1296. For each individual example, one step size equals to the time interval between two adjacent snapshots. For simplicity, all HiTSs only have one hidden layer, and detailed network architectures are shown in Table A.3.

- Once the multiscale HiTSs are trained for each data set, we calculate the total number of parameters and use this information to create the other three types of architectures by carefully choosing the number of hidden units.

- For the CW-RNN, it has two other hyper-parameters: the number of internal modules and their associated clock rates. We set these parameters to match those in multiscale HiTSs for fair comparison.

A.2 Additional results

A.2.1 Computation time for neural network time-steppers

As a supplement to Fig. 2.6A, Table A.4 shows the computational time for all single time-scale neural network time-steppers along with the timings for our multiscale time-steppers. It is not surprising to see that the computation accelerates as the step size grows. The multiscale scheme is more efficient than the finest scale neural network time-stepper in use (see Table A.1 for the coupled NNTSs), which suggests the benefit of vectorized computation.

A.2.2 Noisy measurements

In Table A.5, Table A.6, Table A.7, Table A.8, Table A.9, Table A.10 and Fig. A.1, we study the accuracy of neural network time-steppers with different temporal gaps and the robustness of our results with respect to noise in the observations of the system states. Gaussian random noise with different variances are independently applied to each component of the dynamics. The variances are set to be 0% (noise free), 1%, 2%, 5%, 10% and 20% of the

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	10.06s	24.74s	48.95s	9.98s	42.98s
NNTS 1	3.75s	7.68s	21.48s	3.69s	19.01s
NNTS 2	1.89s	3.62s	8.74s	1.87s	8.51s
NNTS 3	1.01s	1.84s	4.00s	1.03s	4.10s
NNTS 4	0.56s	1.00s	2.06s	0.59s	2.03s
NNTS 5	0.34s	0.77s	1.09s	0.37s	1.09s
NNTS 6	0.24s	0.65s	0.60s	0.27s	0.56s
NNTS 7	0.18s	0.39s	0.36s	0.22s	0.30s
NNTS 8	0.15s	0.27s	0.24s	0.21s	0.18s
NNTS 9	0.13s	0.17s	0.19s	0.17s	0.12s
NNTS 10	0.12s	0.14s	0.15s	0.17s	0.09s
multiscale	0.89s	6.73s	7.96s	1.84s	0.73s

Table A.4: Computation time for multiscale and all single time-scale neural network time-steppers.

variance of that component averaged over all trajectories across the data sets. We observe that larger noise corruption levels generally lead to inferior accuracy, but our multiscale scheme consistently works better than any single time-scale NNTSs. Similar to the results shown in [207], neural networks with larger temporal gaps tend to show more robustness and the optimal value of the temporal gap is usually problem-dependent, indicating the temporal gap is a crucial parameter for data-driven modeling. By leveraging our proposed multiscale coupling strategy, it is automatically taken into account.

A.2.3 More benchmark results against Runge-Kutta time-steppers

Fig. A.2 shows the mean squared errors of different schemes versus simulation time. Again, they are plotted in a base-10 logarithmic scale. Our multiscale schemes provide competitive results compared to the Runge-Kutta schemes in terms of accuracy. This is especially notable for the purely data-driven, neural network-based multiscale time-stepping scheme:

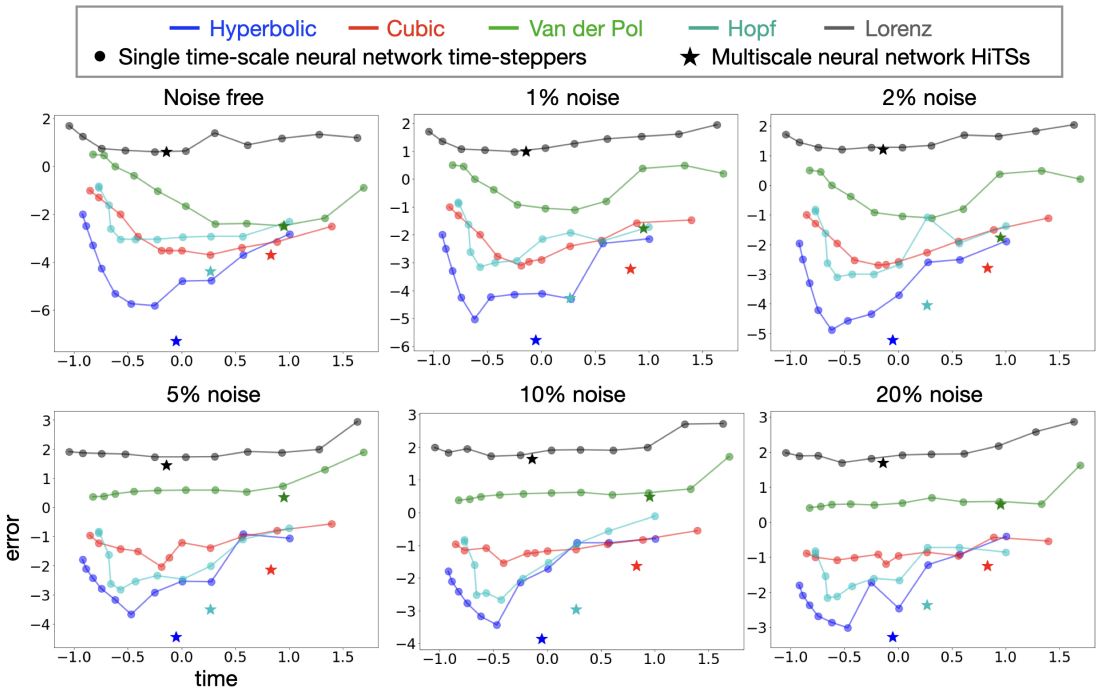

Figure A.1: Accuracy and efficiency trade-offs under 0%, 1%, 2%, 5%, 10% and 20% noise corruptions

the true model of the underlying dynamics is unavailable, which is in sharp contrast to the classical simulation setting. In other words, neural network time-steppers can identify the dynamics and generate accurate predictions at the same time. Hybrid time-steppers, on the other hand, also provide competitive accuracy. And the accuracy level is dominated by either the numerical time-stepper or the neural network time-stepper, whichever is worse. Therefore, the hybrid time-steppers usually bear a slightly lower accuracy than the purely numerical time-steppers.

Table A.11 and Table A.12 shows the computational time and accuracy of the various algorithms which are supplements for Fig. 2.6. For Runge-Kutta time-steppers, computations speed up as the time step increases. Both our multiscale neural network hierarchical time-stepping scheme and hybrid time-stepping scheme provide efficient computations. In particular, the hybrid time-steppers outperform the corresponding RK integrators, as the

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$1.5e-3$	$3.1e-3$	$1.3e-1$	$4.9e-3$	$1.6e+1$
NNTS 1	$2.0e-4$	$7.0e-4$	$6.9e-3$	$1.2e-3$	$2.2e+1$
NNTS 2	$1.7e-5$	$4.0e-4$	$3.3e-3$	$1.2e-3$	$1.5e+1$
NNTS 3	$1.6e-5$	$2.0e-4$	$4.1e-3$	$1.1e-3$	$8.0e+0$
NNTS 4	$1.5e-6$	$3.0e-4$	$3.9e-3$	$9.0e-4$	$2.5e+1$
NNTS 5	$1.8e-6$	$3.0e-4$	$2.2e-2$	$9.0e-4$	$4.4e+0$
NNTS 6	$4.8e-6$	$3.0e-4$	$9.3e-2$	$9.0e-4$	$4.1e+0$
NNTS 7	$5.4e-5$	$1.2e-3$	$4.2e-1$	$2.5e-3$	$4.7e+0$
NNTS 8	$5.0e-4$	$9.8e-3$	$1.0e+0$	$2.4e-2$	$5.6e+0$
NNTS 9	$3.2e-3$	$5.0e-2$	$2.9e+0$	$1.5e-1$	$1.8e+1$
NNTS 10	$1.0e-2$	$1.0e-1$	$3.2e+0$	$1.3e-1$	$5.1e+1$
multiscale	$5.1e-8$	$2.0e-4$	$3.2e-3$	$4.1e-5$	$4.1e+0$

Table A.5: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with noise free data.

vectorized computations accelerate these simulations. The hybrid time-steppers generally beat the multiscale neural network time-steppers in terms of the speed. This speed-up may stem from the computation on each individual step: evaluating a large neural network model (forward propagation) requires more computational effort than applying the Runge-Kutta scheme, which only involves four evaluations of the vector field.

A.2.4 Remarks on learning flow maps

Though there is evidence that our schemes can boost the performance of neural network time-steppers, as well as classical numerical simulation algorithms, each individual result heavily depends on the system under study. Theoretically, there is nothing preventing us from fitting the flow maps with arbitrarily long time periods; however, this is not always possible. For example, in the Lorenz system, one can clearly see the performance of NNTS 10 is unsatisfactory: the one-step prediction error is on the order of $\mathcal{O}(1)$. The reason could

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$7.2e-3$	$3.4e-2$	$1.6e+0$	$1.9e-2$	$9.0e+1$
NNTS 1	$4.9e-3$	$2.7e-2$	$3.1e+0$	$5.9e-3$	$4.1e+1$
NNTS 2	$5.0e-4$	$6.4e-3$	$2.4e+0$	$1.2e-2$	$3.4e+1$
NNTS 3	$7.8e-5$	$3.9e-3$	$1.6e-1$	$7.0e-3$	$2.8e+1$
NNTS 4	$7.3e-5$	$1.3e-3$	$7.8e-2$	$1.2e-3$	$1.9e+1$
NNTS 5	$5.8e-5$	$1.1e-3$	$9.0e-2$	$1.0e-3$	$1.3e+1$
NNTS 6	$9.5e-6$	$8.0e-4$	$1.2e-1$	$7.0e-4$	$9.7e+0$
NNTS 7	$5.7e-5$	$1.7e-3$	$4.2e-1$	$2.4e-3$	$1.1e+1$
NNTS 8	$5.0e-4$	$1.0e-2$	$1.0e+0$	$2.4e-2$	$1.2e+1$
NNTS 9	$3.2e-3$	$5.0e-2$	$2.9e+0$	$1.5e-1$	$2.3e+1$
NNTS 10	$1.0e-2$	$1.0e-1$	$3.2e+0$	$1.3e-1$	$5.2e+1$
multiscale	$1.7e-6$	$6.0e-4$	$1.7e-2$	$5.4e-5$	$9.8e+0$

Table A.6: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with 1% Gaussian noise.

either be that the current architecture is not large enough to encode the map, or the training data set is not large enough to fully capture the complexity of the corresponding flow map. Chaotic dynamics will generally present this challenge, as the flow map complexity grows exponentially with time. As presented in Section 2.3.2, we are fitting the $\Delta t-$lag flow map minus the identity with the neural network

$$\hat{\boldsymbol{F}}(\boldsymbol{x}_t, \Delta t) \approx \boldsymbol{x}_{t+\Delta t} - \boldsymbol{x}_t. \tag{A.1}$$

To see what these increments look like, for the first four nonlinear systems, we have visualized $\boldsymbol{x}_{j\Delta t} - \boldsymbol{x}_0$, for $j = 1, 2, \cdots 64$ with $\boldsymbol{x}_0's$ uniformly sampled from their associated region on interest $\mathcal{D}$, shown in Table A.1. As for the Lorenz system, the process is similar but the initial states are sampled from the region $[-9, -7] \times [6, 8] \times \{27\}$ because this region is close to the strange attractor, and Δt is set to 0.008 for Lorenz and 0.16 for others. Results are shown in Fig. A.3.

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$1.3e-2$	$7.8e-2$	$1.6e+0$	$4.2e-2$	$1.1e+2$
NNTS 1	$3.1e-3$	$3.1e-2$	$3.1e+0$	$1.1e-2$	$6.8e+1$
NNTS 2	$2.5e-3$	$1.3e-2$	$2.4e+0$	$8.5e-2$	$4.5e+1$
NNTS 3	$2.0e-4$	$5.3e-3$	$1.6e-1$	$2.1e-3$	$4.9e+1$
NNTS 4	$4.5e-5$	$2.6e-3$	$7.8e-2$	$1.0e-3$	$2.2e+1$
NNTS 5	$2.7e-5$	$2.1e-3$	$9.0e-2$	$1.0e-3$	$1.9e+1$
NNTS 6	$1.3e-5$	$2.0e-3$	$1.2e-1$	$8.0e-4$	$1.9e+1$
NNTS 7	$6.1e-5$	$3.0e-3$	$4.2e-1$	$2.4e-3$	$1.6e+1$
NNTS 8	$5.0e-4$	$1.1e-2$	$1.0e+0$	$2.4e-2$	$1.9e+1$
NNTS 9	$3.2e-3$	$5.2e-2$	$2.9e+0$	$1.5e-1$	$2.8e+1$
NNTS 10	$1.1e-2$	$1.0e-1$	$3.2e+0$	$1.3e-1$	$5.2e+1$
multiscale	$6.1e-6$	$1.6e-3$	$1.7e-2$	$9.0e-5$	$1.6e+1$

Table A.7: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with 2% Gaussian noise.

In the examples of hyperbolic fixed point and Hopf bifurcation, the plots do not visually grow more complex as time proceeds. For the Van der Pol oscillator, complexity grows in the beginning stage but later it tends to maintain that level of complexity. For the Cubic oscillator example, the complexity keeps growing with time; however, there are clear structures, as more spirals are generated. The most interesting case is the Lorenz system: not only does the complexity grow, but earlier stripe patterns (period doubling) also disappear as time proceeds, leading to tremendous difficulties in training. To fully capture this growing complexity, one may need an exponentially growing data set and network architecture. In addition, one seemingly common feature from all plots is that the multiscale effects become more pronounced as time proceeds, leading to the emergence of patterns. Whether we can utilize this feature to perform more effective training may be a promising direction that motivates future work.

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$8.7e-2$	$2.7e-1$	$7.8e+1$	$1.9e-1$	$8.8e+2$
NNTS 1	$1.2e-1$	$1.6e-1$	$2.0e+1$	$7.9e-2$	$9.8e+1$
NNTS 2	$2.8e-3$	$9.7e-2$	$5.3e+0$	$9.7e-3$	$7.6e+1$
NNTS 3	$2.9e-3$	$4.1e-2$	$3.4e+0$	$3.4e-3$	$8.3e+1$
NNTS 4	$1.2e-3$	$6.2e-2$	$3.9e+0$	$4.5e-3$	$5.5e+1$
NNTS 5	$2.1e-4$	$1.9e-2$	$3.9e+0$	$2.9e-3$	$5.4e+1$
NNTS 6	$6.8e-4$	$9.1e-3$	$3.8e+0$	$1.5e-3$	$5.4e+1$
NNTS 7	$1.6e-3$	$3.1e-2$	$3.5e+0$	$2.4e-3$	$6.8e+1$
NNTS 8	$3.7e-3$	$3.8e-2$	$2.9e+0$	$2.4e-2$	$7.1e+1$
NNTS 9	$7.8e-3$	$6.0e-2$	$2.4e+0$	$1.5e-1$	$7.4e+1$
NNTS 10	$1.6e-2$	$1.1e-1$	$2.3e+0$	$1.3e-1$	$8.2e+1$
multiscale	$3.6e-5$	$7.2e-3$	$2.2e+0$	$3.1e-4$	$2.8e+1$

Table A.8: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with 5% Gaussian noise.

A.2.5 Van der Pol oscillator example

The multiscale nature of a problem often leads to numerical stiffness when it comes to scientific computing. Many classic numerical solvers become unstable unless the step size is set to be extremely small, which in turn, poses severe computational challenges. Here, we explore how this would affect our multiscale hierarchical neural network time-stepping scheme. We explore this by varying the parameter $\mu = 2, 5, 10, 20, 50, 100$ of the Van der Pol system:

$$\begin{aligned} \dot{x} &= \mu(x - \frac{1}{3}x^3 - y) \\ \dot{y} &= \frac{1}{\mu}x \end{aligned} \tag{A.2}$$

We choose to work with this form so that the state variables evolve approximately on the same scale across different values of μ. A higher value of μ indicates stiffer dynamics. For this set of experiments, all setups and network architectures are exactly the same to those used in Appendix A.1A.1.1 for the Van der Pol example. Shown in Table A.13 are the

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$1.6e-1$	$2.8e-1$	$5.2e+1$	$7.8e-1$	$5.3e+2$
NNTS 1	$1.2e-1$	$1.5e-1$	$5.3e+0$	$2.7e-1$	$5.1e+2$
NNTS 2	$1.2e-1$	$1.1e-1$	$4.1e+0$	$1.1e-1$	$9.9e+1$
NNTS 3	$1.9e-2$	$7.6e-2$	$3.5e+0$	$3.0e-2$	$8.1e+1$
NNTS 4	$7.4e-3$	$6.7e-2$	$4.2e+0$	$9.9e-3$	$8.4e+1$
NNTS 5	$3.7e-4$	$5.9e-2$	$4.0e+0$	$2.2e-3$	$8.2e+1$
NNTS 6	$6.8e-4$	$5.7e-2$	$3.8e+0$	$3.5e-3$	$5.7e+1$
NNTS 7	$1.7e-3$	$2.9e-2$	$3.5e+0$	$3.1e-3$	$5.3e+1$
NNTS 8	$3.9e-3$	$8.3e-2$	$3.1e+0$	$2.5e-2$	$9.0e+1$
NNTS 9	$8.0e-3$	$7.1e-2$	$2.6e+0$	$1.5e-1$	$6.9e+1$
NNTS 10	$1.6e-2$	$1.1e-1$	$2.4e+0$	$1.3e-1$	$9.8e+1$
multiscale	$1.4e-4$	$2.4e-2$	$3.1e+0$	$1.1e-3$	$4.4e+1$

Table A.9: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with 10% Gaussian noise.

computation time and the integrated $\mathcal{L}_2$ error from $t = 0$ to $t = 50$ with varying parameters of μ's. We also present the best integrated $\mathcal{L}_2$ error achieved by all single scaled neural network time-steppers for comparison purposes. We observe that the computation time are similar across different groups, suggesting unchanged efficiency. This is not surprising as the computation speed only rely on the size of the neural networks used in the proposed framework. However, the accuracy deteriorates as μ increases, indicating the challenge lies on the training side: the more stiff the problem, the more complex the flow maps would be, which is consistent with our conjectures from Appendix A.2A.2.4. However, they still outperform the single scaled neural network time-steppers thanks to the corrections from larger scaled networks.

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
NNTS 0	$3.9e-1$	$2.9e-1$	$4.2e+1$	$1.4e-1$	$7.4e+2$
NNTS 1	$1.2e-1$	$3.7e-1$	$3.3e+0$	$1.9e-1$	$3.8e+2$
NNTS 2	$6.1e-2$	$1.1e-1$	$3.9e+0$	$1.9e-1$	$1.5e+2$
NNTS 3	$3.5e-3$	$1.4e-1$	$3.8e+0$	$2.2e-2$	$8.9e+1$
NNTS 4	$1.9e-2$	$1.1e-1$	$5.0e+0$	$2.5e-2$	$8.7e+1$
NNTS 5	$9.7e-4$	$6.4e-2$	$3.5e+0$	$1.5e-2$	$8.3e+1$
NNTS 6	$1.4e-3$	$1.2e-1$	$3.1e+0$	$7.6e-3$	$6.5e+1$
NNTS 7	$2.1e-3$	$9.7e-2$	$3.3e+0$	$7.0e-3$	$5.0e+1$
NNTS 8	$4.3e-3$	$8.3e-2$	$3.2e+0$	$2.9e-2$	$7.9e+1$
NNTS 9	$8.3e-3$	$1.0e-1$	$2.8e+0$	$1.5e-1$	$7.8e+1$
NNTS 10	$1.6e-2$	$1.3e-1$	$2.6e+0$	$1.3e-1$	$9.6e+1$
multiscale	$5.4e-4$	$5.7e-2$	$3.2e+0$	$4.3e-3$	$5.0e+1$

Table A.10: Integrated $\mathcal{L}_2$ between the predicted and the exact measurement with 20% Gaussian noise.

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
RK 0	$2.55s$	$3.18s$	$2.82s$	$3.93s$	$3.87s$
RK 1	$1.28s$	$1.60s$	$1.30s$	$2.21s$	$1.97s$
RK 2	$0.74s$	$1.01s$	$0.78s$	$1.07s$	$1.17s$
RK 3	$0.46s$	$0.64s$	$0.42s$	$0.60s$	$0.98s$
RK 4	$0.30s$	$0.43s$	$0.30s$	$0.35s$	$0.60s$
Hybrid 0	$1.36s$	$2.25s$	$1.37s$	$1.89s$	$3.17s$
Hybrid 1	$0.72s$	$0.96s$	$0.69s$	$1.01s$	$1.78s$
Hybrid 2	$0.37s$	$0.55s$	$0.42s$	$0.59s$	$0.99s$
Hybrid 3	$0.24s$	$0.30s$	$0.29s$	$0.37s$	$0.74s$
Hybrid 4	$0.19s$	$0.23s$	$0.22s$	$0.26s$	$0.62s$
multiscale	$0.89s$	$6.73s$	$7.96s$	$1.84s$	$0.73s$

Table A.11: Computation time of various schemes on testing trajectories.

Systems	Hyperbolic	Cubic oscillator	Van der Pol	Hopf bifurcation	Lorenz
RK 0	$1.8e-9$	$4.8e-5$	$2.0e-3$	$9.8e-4$	$2.5e+0$
RK 1	$1.9e-9$	$4.8e-5$	$2.0e-3$	$9.8e-4$	$2.5e+0$
RK 2	$2.0e-9$	$4.8e-5$	$2.0e-3$	$9.8e-4$	$2.5e+0$
RK 3	$3.1e-9$	$4.8e-5$	$2.1e-3$	$9.8e-4$	$2.5e+0$
RK 4	$1.8e-8$	$4.6e-5$	$4.1e-3$	$9.9e-4$	$2.5e+0$
Hybrid 0	$1.7e-8$	$2.8e-4$	$2.0e-2$	$8.6e-5$	$2.5e+1$
Hybrid 1	$1.8e-8$	$2.8e-4$	$2.0e-2$	$8.5e-5$	$2.4e+1$
Hybrid 2	$1.8e-8$	$2.8e-4$	$2.0e-2$	$8.5e-5$	$2.4e+1$
Hybrid 3	$1.9e-8$	$2.8e-4$	$2.0e-2$	$8.5e-5$	$2.4e+1$
Hybrid 4	$3.4e-8$	$2.8e-4$	$2.0e-2$	$8.7e-5$	$2.4e+1$
multiscale	$5.1e-8$	$2.0e-4$	$3.2e-3$	$4.1e-5$	$4.1e+0$

Table A.12: Integrated $\mathcal{L}_2$ error of various schemes on testing trajectories.

μ	2	5	10	20	50	100
computation time (HiTS)	$7.96s$	$8.13s$	$8.02s$	$7.98s$	$8.07s$	$8.10s$
$\mathcal{L}_2$ error (multiscale)	$5.9e-5$	$3.7e-4$	$7.3e-3$	$6.2e-2$	$2.2e-2$	$3.3e-2$
$\mathcal{L}_2$ error (best NNTS)	$5.1e-4$	$2.2e-3$	$8.7e-3$	$1.2e-1$	$8.6e-2$	$6.3e-2$

Table A.13: Computation time and integrated $\mathcal{L}_2$ error of multiscale hierarchical neural network time-stepping scheme under varying levels of stiffness of Van der Pol oscillator.

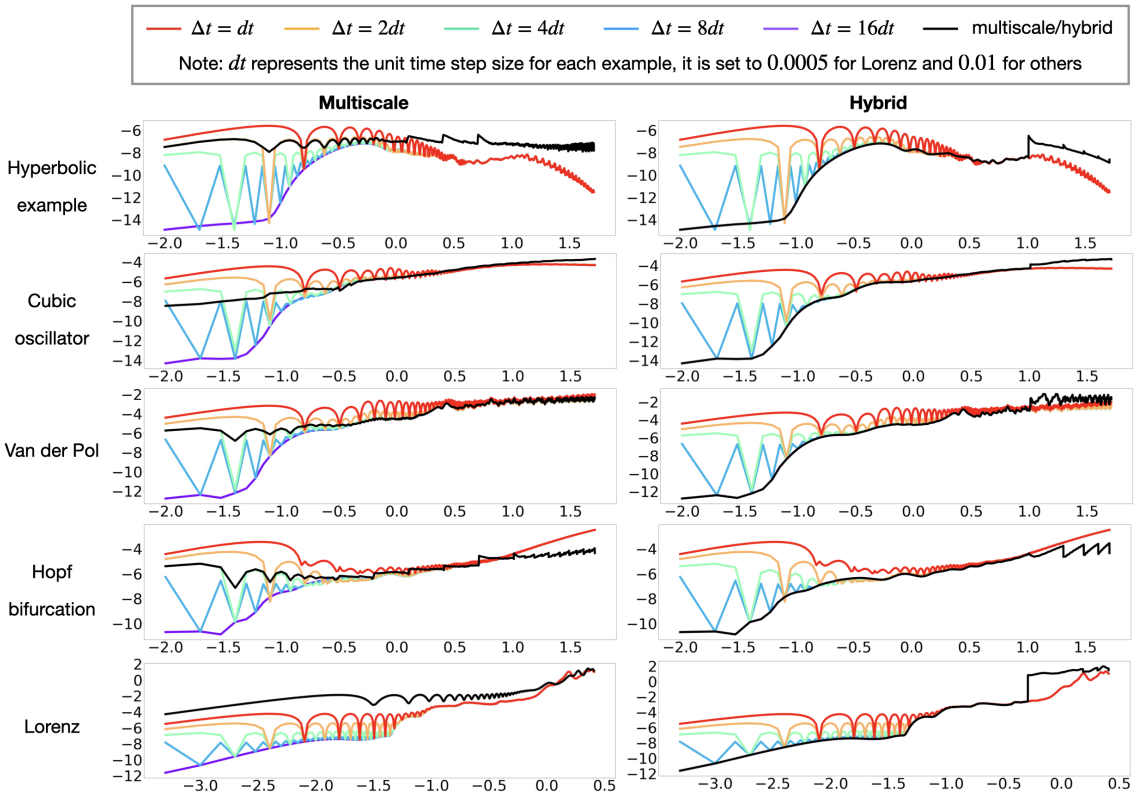

Figure A.2: **Stepwise error comparisons with Runge-Kutta integration**. For each plot, the horizontal axis represents simulation time and vertical axis represents $\mathcal{L}_2$ error. Both axes are visualized in base-10 logarithmic scale. The left column shows the comparisons between multiscale time-steppers and Runge-Kutta time-steppers. The right column shows the comparisons between hybrid time-steppers and Runge-Kutta time-steppers. For both columns, mean squared errors at each time step are plotted.

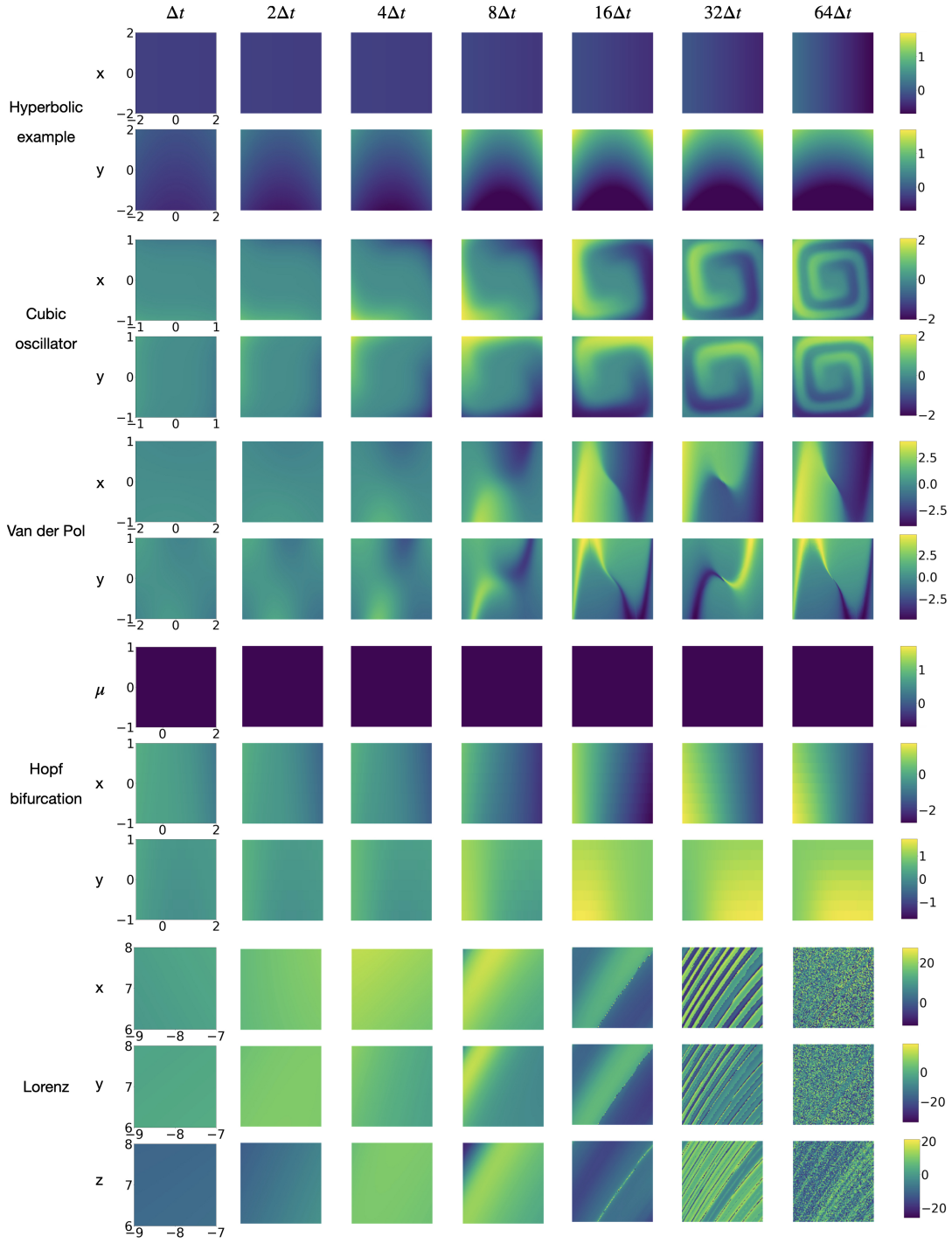

Figure A.3: **Increments of flow maps**. We visualize $\boldsymbol{x}_{j\Delta t} - \boldsymbol{x}_0$, for $j = 1, 2, \cdots 64$ where $\boldsymbol{x}$ is the state of each example. For the Lorenz system, we set $\Delta t = 0.008$ and the region we visualize is $[-9, -7] \times [6, 8] \times \{27\}$ whereas for other examples, $\Delta t = 0.16$ and the regions are in Table A.1.

Appendix B

MULTIRESOLUTION CONVOLUTIONAL AUTOENCODER

B.1 Remarks on filter initialization

As we perform the deepening operation or widening operation, we are essentially adding properly initialized convoltuional and deconvolutional filters to the existing network.

For a deepening operation, one convolutional filter C_0 and one deconvolutional filter D_0 are added to the network to connect the inputs and outputs from two adjacent levels. Both of the two filters are of kernel size 3×3 and applied with stride size 2 without padding, and they are initialized with the following matrices:

$$C_0 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} \epsilon_{11}^{(c)} & \epsilon_{12}^{(c)} & \epsilon_{13}^{(c)} \\ \epsilon_{21}^{(c)} & \epsilon_{22}^{(c)} & \epsilon_{23}^{(c)} \\ \epsilon_{31}^{(c)} & \epsilon_{32}^{(c)} & \epsilon_{33}^{(c)} \end{bmatrix} \tag{B.1}$$

$$D_0 = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix} + \begin{bmatrix} \epsilon_{11}^{(d)} & \epsilon_{12}^{(d)} & \epsilon_{13}^{(d)} \\ \epsilon_{21}^{(d)} & \epsilon_{22}^{(d)} & \epsilon_{23}^{(d)} \\ \epsilon_{31}^{(d)} & \epsilon_{32}^{(d)} & \epsilon_{33}^{(d)} \end{bmatrix} \tag{B.2}$$

In a nutshell, C_0 is initialized with a down-sampling operator while D_0 is initialized with an interpolation operator so that the transition is smooth between each level. We also add some uniformly distributed random noise ($\epsilon_{ij}^{(c)}, \epsilon_{ij}^{(d)} \in [-\epsilon, \epsilon]$) to break the symmetry.

To perform a widening operation, we create a group of convolutional filters and deconvolutional filters where the channel number is specified by the user. Also for the sake of ensuring a smooth transition, we initialize all entries of these filters with independent, uniformly distributed random variables from $[-\epsilon, \epsilon]$ so that immediately after the change of the architecture, reconstructions won't be affected by much.

B.2 Remarks on training

In addition to specifying the tolerances (threshold for residuals on all pixels) for different training phases as shown in Algorithm 3, we set a maximum number of epochs to stop the training. We also implement a stopping criterion as the convergence slowns down: if the relative error decrease every ten epochs is less than 0.001, we stop the training and move to the next operation to expand the network capacity.

Appendix C

PHYSICS-INFORMED KOOPMAN NETWORK

C.1 Experimental setups

For all networks in the experiments, L is set to a zero matrix initially and all other parameters are randomly initialized using the default initializer of Pytorch. The exponential linear unit (ELU) is used as the nonlinear activation function as we expect the transformations in PIKN to be relatively smooth.

Simple nonlinear system with discrete spectrum

In this example, we have trained two different types of PIKN: a PIKN with a linear decoder and a PIKN with a nonlinear decoder. In both experiments, 1000 collocation points were uniformly sampled from $[-1, 1] \times [-1, 1]$ for training, an Adam optimizer with a learning rate of $1e-4$ has been applied. The total number of training epochs is set to 50000 and the weights in the loss function are set to $\omega_1 = \omega_2 = 1$. The encoder part of both architectures are the same: a 2-layer fully-connected neural network with hidden layer containing 50 neurons. The major differences between the two architectures are as follows:

- The linear decoder is simply a linear layer without a bias term whereas the nonlinear decoder is symmetric to the encoder: a 2-layer fully-connected neural network with hidden width 50.

- For the PIKN with a linear decoder, we have a three-dimensional latent space whereas for the PIKN with a nonlinear decoder, it is two-dimensional.

Heat equation

In the PIKN architecture for Heat equation, encoder and decoder are both linear and the latent dimension is set to 64, the same as the number of spatial grids. Number of training

epochs is 100000 and an Adam optimizer has been applied for the training algorithm. In this set of experiments, we use an adaptive learning rate: initially set to 0.01, it keeps decreasing by a factor of 0.5 if no improvements are made over the recent 5000 epochs, until it hits the minimal value of 0.000001.

With the network architecture and training parameters fixed, we train it in three different ways. Namely, we set different values for the weights $\omega_1, \omega_2, \omega_3, \omega_4$ in the loss function, leading to three different training regimes:

- $\omega_1 = 0.0001, \omega_2 = 1, \omega_3 = 0, \omega_4 = 0$: the network is purely physics-informed.
- $\omega_1 = 0, \omega_2 = 0, \omega_3 = 1, \omega_4 = 1$: this corresponds to a purely data-driven learning.
- $\omega_1 = 0.0001, \omega_2 = 1, \omega_3 = 1, \omega_4 = 1$: this represents the scenario where we train a physics-informed network with data integration. For simplicity, we call it hybrid training.

Notice that the value of ω_1 is set on a different order compared to other weights, it is because this loss term involves calculation of numerical derivatives which usually has a greater amplitude. This is a well-known issue for physics-informed neural network and has been thoroughly studied. Adaptive re-weighting schemes were proposed to fix it [249, 155, 282]. In our case, however, we find that choosing a fixed, small value $\omega_1 = 0.0001$ is sufficient for achieving a fast convergence[1].

For the physics-informed learning, we create 1000 trial functions $\{u^{(1)}, u^{(2)}, \ldots, u^{(1000)}\}$ for training purpose. Each trial function $u^{(j)}$ is a superposition of the Fourier modes that satisfy the periodic boundary conditions. In our case, this amounts to using $\sin(kx)$ and $\cos(kx)$ for $k = 0, 1, 2, \ldots$ as basis functions. The value of k is restricted to be no greater than 32 due to the choice of our grid spatial resolution. The spatial derivatives of the trail functions $\{u_{xx}^{(1)}, u_{xx}^{(2)}, \ldots, u_{xx}^{(1000)}\}$ are calculated using numerical spectral method.

In the data-driven regime, we use simulation data obtained from a solver based on spectral

[1] Another way to get around this is to learn the pseudo-inverse of L instead. Then the loss term reads $\|\phi(\mathbf{x_i}) - L^{\dagger}\nabla\phi(\mathbf{x_i}) \cdot \mathbf{f}(\mathbf{x_i})\|$. In that case ω_1 and ω_2 can be both set to 1 because the two loss terms are approximately on the same scale.

method. We run the simulation with 1000 different initial states. For each run, the initial state of u is obtained through the same procedure as we obtain the trial functions $u^{(k)}$. Then we sample snapshots of the state u with a temporal gap $\Delta t = 0.01$ for 5 steps (i.e. $p = 5$).

For the hybrid training regime, both of the above two data sets are used. However, to study the effects of the amount of simulation data, we conduct 10, 50, 100, 500, 1000 simulation runs in 5 separate groups of experiments.

Burger's equation

The architecture for Burger's equation consists of a nonlinear encoder and a nonlinear decoder. Both encoder and decoder are 3-layer fully connected neural network with hidden width 512. The latent dimension is set to the same as the number of modes we use for data generation, which will be clarified shortly. Spatial grids and training parameters (learning rates and number of training epochs) are set to be the same as in the heat equation experiments.

To generate the data, we take advantage of the Cole-Hopf transformations as previously discussed, resulting in the following procedure:

- Choose a number of modes K (i.e. $C_k = 0$ for $k > K$).
- Create a list of $\{\tilde{C}_1, \tilde{C}_2, \cdots, \tilde{C}_K\}$ with each $\tilde{C}_k$ uniformly sampled from $(-1, 1)$.
- Obtain a number α uniformly sampled from $(0, 1)$.
- Create a list of $\{C_1, C_2, \cdots, C_K\}$ with each $C_k = \frac{\alpha \tilde{C}_k}{\sum_{k=1}^{K} |\tilde{C}_k|}$ (so that $\sum_{k=1}^{K} |C_k| = \alpha < 1$)
- Generate $v(x, t) = \frac{1}{2\pi} + \sum_{k=1}^{K} C_k \cos(kx) e^{-k^2 t}$.
- Obtain $u(x, t) = C(v(x, t))$.

This data generation process guarantees the existence of at least one transformation (Cole-Hopf transformation followed by a Fourier transformation) that linearizes the nonlinear Burger's equation with eigenvalues $\mu_k = -k^2$ $(k = 1, 2, ..., K^2)$. In our experiments, we set $K = 10$.

We also study the three different learning regimes, with different combinations of $\omega_1, \omega_2, \omega_3$ and ω_4. The only difference from the setup of the heat equation experiments is that ω_1 is set to 0.01.

For physics-informed learning, we create 1000 trial functions $\{u^{(1)}, u^{(2)}, \ldots, u^{(1000)}\}$ for training. Each trial function $u^{(j)}$ is a snapshot at a random time point $t^{(j)} \in [0, 0.1)$ of the $u(x,t)$ generated from the above procedure and projected onto the spatial grids. To generate $\{u_x^{(1)}, u_x^{(2)}, \ldots, u_x^{(1000)}\}$ and $\{u_{xx}^{(1)}, u_{xx}^{(2)}, \ldots, u_{xx}^{(1000)}\}$, we take advantage of the $u = C(v) = -\frac{2v_x}{v}$ again, so that

$$\begin{aligned} u_x &= 2(\frac{v_x}{v})^2 - 2\frac{v_{xx}}{v} \\ u_{xx} &= 6\frac{v_x v_{xx}}{v^2} - 2\frac{v_{xxx}}{v} - 4(\frac{v_x}{v})^3 \end{aligned} \tag{C.1}$$

where v_x, v_{xx} and v_{xxx} are easy to derive.

For data-driven learning, we also obtain data from the above procedure. We sample snapshots of the state $u(x,t)$ with a temporal gap $\Delta t = 0.02$ for 5 steps. For the hybrid regime, both types of data are used.

To study the effects of number of training trajectories for data-driven learning and hybrid learning, we conduct 5 groups of experiments for each, using 1000, 500, 100, 50, 10 training trajectories respectively.

C.2 Additional results

Potential pitfalls of using nonlinear transformations

For each architecture, we have run the training algorithm for 10 times and the results are robust in the sense that the identified eigenvalues are all centered around the real ones we derived analytically, which are presented in Table C.1. One can see our PIKNs faithfully recover the desired Koopman eigenvalues. The one with the linear decoding provides slightly more robust results, indicating the benefits of adding more known constraints.

We notice that, however, the PIKN with nonlinear decoder doesn't always identify the eigenvalue $\mu_1 = -0.1$. If we significantly change the initialization of the network parameters, sometimes other values emerge. This indicates other transformations exist to linearize the dynamics and reconstruct the state variables. As an example, one can easily check $\varphi_\mu^\beta = x_1^\beta$ for all $\beta \in \mathbb{N}$ are all valid Koopman eigenfunctions associated with eigenvalues $\mu\beta$ and can be used for reconstruction. The lesson here is that by using a nonlinear decoder, we not only increase the flexibility of the Koopman operator theory framework, but also dramatically increase the searching space, which may lead to different learning outcomes.

	PIKN(linear decoder)	**PIKN(nonlinear decoder)**
Experiment 1	$\mu_1 = -0.10000689$, $\mu_2 = -0.19914131$, $\mu_3 = -0.9996788$	$\mu_1 = -0.09756267$, $\mu_2 = -0.99973810$
Experiment 2	$\mu_1 = -0.10009335$, $\mu_2 = -0.19947967$, $\mu_3 = -1.0004123$	$\mu_1 = -0.09644943$, $\mu_2 = -0.99867420$
Experiment 3	$\mu_1 = -0.10005733$, $\mu_2 = -0.19892442$, $\mu_3 = -1.0003881$	$\mu_1 = -0.09592330$, $\mu_2 = -0.99915651$
Experiment 4	$\mu_1 = -0.10008135$, $\mu_2 = -0.20019206$, $\mu_3 = -0.9986593$	$\mu_1 = -0.09784412$, $\mu_2 = -1.00298023$
Experiment 5	$\mu_1 = -0.09990316$, $\mu_2 = -0.20018657$, $\mu_3 = -0.9991975$	$\mu_1 = -0.09837312$, $\mu_2 = -1.00187280$
Experiment 6	$\mu_1 = -0.09997816$, $\mu_2 = -0.19966313$, $\mu_3 = -1.0005931$	$\mu_1 = -0.09764695$, $\mu_2 = -1.00517617$
Experiment 7	$\mu_1 = -0.10009032$, $\mu_2 = -0.19991782$, $\mu_3 = -1.0000535$	$\mu_1 = -0.09866422$, $\mu_2 = -0.99622254$
Experiment 8	$\mu_1 = -0.09998395$, $\mu_2 = -0.19948480$, $\mu_3 = -0.9996783$	$\mu_1 = -0.09856838$, $\mu_2 = -1.00204348$
Experiment 9	$\mu_1 = -0.09996726$, $\mu_2 = -0.19905518$, $\mu_3 = -1.0005406$	$\mu_1 = -0.09929386$, $\mu_2 = -0.99679565$
Experiment 10	$\mu_1 = -0.10004932$, $\mu_2 = -0.19994377$, $\mu_3 = -0.9993069$	$\mu_1 = -0.09756234$, $\mu_2 = -1.00279380$
summary	$\boldsymbol{\mu_1 = -0.10 \pm 6.22e-04}$, $\boldsymbol{\mu_2 = -0.20 \pm 4.37e-04}$, $\boldsymbol{\mu_3 = -1.00 \pm 5.99e-05}$	$\boldsymbol{\mu_1 = -0.10 \pm 2.74e-03}$, $\boldsymbol{\mu_2 = -1.00 \pm 9.68e-04}$

Table C.1: Eigenvalues identified by PIKNs at all training runs. The first column represents the PIKN with a linear decoder and the second column is the one with a nonlinear decoder.