DATA-DRIVEN MODEL REDUCTION USING WELDNET: WINDOWED AUTOENCODERS FOR LEARNING DYNAMICS

BIRAJ DAHAL* AND OTHERS

Abstract. Many problems in science and engineering involve time-dependent, high dimensional datasets arising from complex physical processes, which are costly to simulate. In this work, we propose **WeldNet**: **W**indowed **E**ncoders for **L**earning **D**ynamics, a data-driven nonlinear model reduction framework to build a low-dimensional surrogate model for complex evolution systems. Given time-dependent training data, we split the time domain into multiple overlapping windows, within which nonlinear dimension reduction is performed by auto-encoders to capture latent codes. Once a low-dimensional representation of the data is learned, a propagator network is trained to capture the evolution of the latent codes in each window, and a transcoder is trained to connect the latent codes between adjacent windows. The proposed windowed decomposition significantly simplifies propagator training by breaking long-horizon dynamics into multiple short, manageable segments, while the transcoders ensure consistency across windows. In addition to the algorithmic framework, we develop a mathematical theory establishing the representation power of WeldNet under the manifold hypothesis, justifying the success of nonlinear model reduction via deep autoencoder-based architectures. Our numerical experiments on various differential equations indicate that WeldNet can capture nonlinear latent structures and their underlying dynamics, outperforming both traditional projection-based approaches and recently developed nonlinear model reduction methods.

Key words. model reduction, manifold learning, scientific machine learning, dynamical systems

MSC codes. 68Q25, 68R10, 68U05

1. Introduction.

Many real-world applications in science and engineering involve large-scale, complex, and costly data simulations or inversions of physical processes. However, the high-dimensional nature of these models often creates overwhelming demands on computational resources. Model reduction plays a crucial role in addressing this challenge, which helps to reduce the data dimension and problem size [21, 6, 5, 52, 46, 9, 42].

Linear model reduction techniques for differential equations and dynamical systems have been well-established in literature [5, 6, 52]. Classical projection-based model reduction approaches have demonstrated great success when the underlying model is linear and low-dimensional. Representative projection-based model reduction methods include Proper Orthogonal Decomposition (POD) [7, 40, 52], reduced-basis techniques [41, 46], the Principal Component Analysis (PCA) approach [33], rational interpolation [19, 3], Galerkin projection [23, 45], etc. Most projection-based model reduction methods rely on projecting high-dimensional models onto a low-dimensional linear subspace.

In real-world applications, many objects exhibit low-dimensional nonlinear structures [49, 44]. Simple transformations, such as translations or rotations, place these objects on low-dimensional nonlinear manifolds, which cannot be efficiently captured by linear subspaces. Meanwhile, many physical processes are inherently nonlinear, including fluid dynamics, nonlinear optical phenomena, and shallow water wave propagation. When linear model reduction methods are applied to reduce evolutionary equations, their optimal performance is quantified by the Kolmogorov n-width [39]. For diffusion-dominated problems, the Kolmogorov n-width decays rapidly as the subspace dimension increases, enabling linear methods to achieve significant success with well-established justification [2, 8]. Unfortunately, many differential equations,

*School of Mathematics, Georgia Institute of Technology, Atlanta, GA, USA bdahal6@gatech.edu.

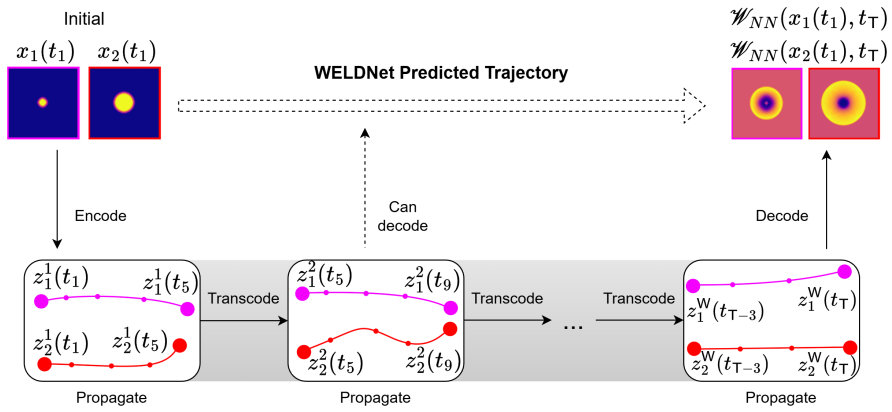

FIG. 1. *WeldNet: Initial conditions are encoded, propagated within windows, transcoded between windows, and decoded. In this example, there are* W *windows and* T *time steps.* $z_j^i(t_k)$ *denotes the latent space representation of* $x_j(t_k)$ *according to window* i.

particularly advection-dominated problems, display a slow decay of the Kolmogorov n-width. In these cases, linear model reduction methods require a sufficiently large reduced dimensionality to achieve acceptable accuracy [34], which demonstrates the limitations of linear model reduction methods in handling nonlinear structures.

The limitations of linear methods drive the development of nonlinear model reduction techniques, which aim to leverage low-dimensional nonlinear structures in data and models [11, 12, 55]. As deep learning has gained popularity in recent years, deep learning methods have been increasingly applied to address nonlinear model reduction [36, 54, 26, 17, 18, 17]. Many works utilize autoencoder [4] for dimension reduction, and then learn the unknown physical process on latent variables by a neural network [22, 37, 53, 36, 54, 26, 17, 18, 17, 16, 29]. However, when the solution manifold of evolutionary equations has evolved for a long time, a global dimension reduction may not well capture the effective latent parameters at all time points.

In this work, we propose Windowed Encoders for Learning Dynamics with Neural Networks (WeldNet) for nonlinear model reduction. WeldNet uses encoder networks to perform nonlinear dimension reduction and learn the evolution operation in the encoded domain. The time domain is divided into several sequentially overlapping subintervals called "windows". In each window, we train an autoencoder to learn the latent codes and a propagator to evolve the latent codes in time. Transcoder networks are trained to connect two adjacent windows on their overlap, allowing an initial condition to be encoded, evolved to the terminal time, and then decoded. After training, WeldNet can be used to approximate the trajectory at any time given an initial condition. Figure 1 diagrams our WeldNet method of trajectory learning for time-dependent data.

The separation between dimension reduction and trajectory learning allows the exploitation of existing nonlinear low-dimensional evolutionary structures in the data. Figure 2 indicates the notation for each component of WeldNet. We consider the trajectory manifold of an evolutionary process, denoted by $\mathcal{M}([0,T])$, which collects all evolution trajectories in the time interval $[0,T]$. We split this time interval into W windows. For each window $i \in \{1, 2, \ldots, \mathsf{W}\}$, $\mathscr{E}^i$ and $\mathscr{D}^i$ denote the encoder and decoder networks, $\mathscr{P}^i$ denotes the propagator network for the evolution of latent

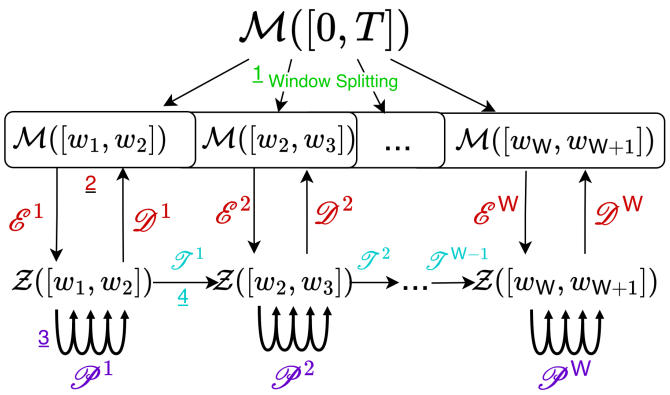

FIG. 2. *Components of WeldNet: 1) Window splitting, 2) Autoencoder, 3) Propagator, 4) Transcoder.*

codes, and $\mathscr{T}_i$ (for $i < \mathsf{W}$) denotes the transcoder network from the ith window to the $(i+1)$th window.

We validate WeldNet on a few examples, including the Burgers' equation, transport equation, Korteweg–De Vries (KdV) equation and 2D shallow-water equation. Furthermore, this paper provides a mathematical theory on the representation power of WeldNet to justify the success of nonlinear model reduction by deep learning under manifold hypothesis. Suppose the evolution operator satisfies a regularity assumption and all trajectories of this evolutionary process lie on a low-dimensional manifold. We prove that, WeldNet can approximate the trajectories of this evolutionary process up to arbitrary accuracy, if the encoder, decoder and propagator network architectures are properly set up. Our theory justifies the representation power of WeldNet for a large class of evolutionary processes, and provides a theoretical foundation for nonlinear model reduction using auto-encoder-based methods.

The main contributions of this paper are:

1. **WeldNet framework.** We propose a windowed autoencoder–propagator architecture for nonlinear model reduction. The method performs dimension reduction and latent-space trajectory learning by dividing the time domain into overlapping windows connected via transcoder networks.
2. **Efficient trajectory prediction.** WeldNet enables end-to-end approximation of system trajectories by encoding an initial condition, evolving the latent representations across windows, and decoding the result at any desired time.
3. **Theoretical foundation.** Under the manifold hypothesis, we establish a representation theory to show that WeldNet can express evolutionary processes with low-dimensional structures to arbitrarily high accuracy when the encoder, decoder, and propagator networks are appropriately designed.
4. **Empirical validation.** The effectiveness of WeldNet is demonstrated on several nonlinear PDEs, including the Burgers', transport, KdV, and 2D shallow-water equations.

Organization. We first introduce our WeldNet model in Section 2. A representation theory of WeldNet is presented in Section 3, and comprehensive numerical experiments are given in Section 4. Finally, we conclude in Section 5.

Notations. For any $n \in \mathbb{N}$, we denote $[n] = \{1, \dots, n\}$. For $\mathbf{x} \in \mathbb{R}^n$, we denote $\|\mathbf{x}\|_{\mathbb{R}^n} = \|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty = \max_{i\in[n]} x_i$ where x_i is the ith component of $\mathbf{x}$. For any $\mathbf{x} \in \mathbb{R}^n$, we denote the ReLU function $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ by $\sigma(\mathbf{x}) = (\max(x_i, 0))_{i=1}^n$. For any function $f : A \to B$ defined on sets A, B, we denote $\|f\|_{L^\infty(A;\, B)} = \sup_{\mathbf{x}\in A} \|f(\mathbf{x})\|_B$ where $\|\cdot\|_B$ is a norm on B. Given a finite set A, we denote the cardinality of A by $|A|$. We use $\bigcirc_{k=1}^K f_k$ to denote the composition $f_K \circ f_{K-1} \circ \cdots \circ f_1$. In particular, $\bigcirc_{k=1}^K f$ denotes the composition of f for K times.

2. WeldNet for Model Reduction.

In this section, we present our WeldNet model, which operates on time-dependent trajectory data collected from an evolutionary process. In science and engineering applications, the solution trajectory in an evolutionary process often depends on few parameters in the initial condition or in the evolution equation [56, 26]. When the trajectories of this evolutionary process lie on a low-dimensional manifold, WeldNet parametrizes the trajectory manifold by a low-dimensional latent code and builds a surrogate evolutionary model in the latent space.

We consider an evolutionary process in $\mathbb{R}^D$, whose initial states can be (locally) parameterized by a small number of parameters. Suppose the initial states are supported on a set $\mathcal{M}(0) \subseteq \mathbb{R}^D$ We are interested in learning an evolutionary process driven by an unknown continuous **time-evolution operator** $\mathscr{F} : \mathcal{M}(0) \times [0, T] \to \mathbb{R}^D$. For simplicity, we denote

$$\mathbf{x}(t) = \mathscr{F}(\mathbf{x}(0), t).$$

We denote $\mathcal{M}(t) = \mathscr{F}(\mathcal{M}(0), t) = \{\mathscr{F}(\mathbf{x}(0), t) :\ \mathbf{x}(0) \in \mathcal{M}(0)\}$ as the initial data evolved by t time units, and $\mathcal{M}([a, b]) = \{\mathcal{M}(t) :\ t \in [a, b]\}$ denotes the collection of state manifold in the time interval $[a, b]$. We call the set $\mathcal{M}(0)$ the initial manifold and the set $\mathcal{M}([0, T])$ the **trajectory manifold**. We will assume that $\mathcal{M}([0, T])$ is a d-dimensional smooth manifold smoothly isometrically embedded in $\mathbb{R}^D$. For convenience, we extend the domain of the time-evolution operator $\mathscr{F}$ to cover all of $\mathcal{M}([0, T])$ and all relevant times, such that for all $\mathbf{x}(s) = \mathscr{F}(\mathbf{x}(0), s) \in \mathcal{M}(s)$, we assume that the dynamic satisfies $\mathscr{F}(\mathbf{x}(s), t) = \mathscr{F}(\mathbf{x}(0), s + t) = \mathbf{x}(s + t)$ for $t \in [0, T - s]$.

In addition, we extend the domain of $\mathscr{F}$ to all of $\mathcal{M}([0, T]) \times [0, T]$ in the following way: if $\mathbf{x} \in \mathcal{M}(s)$ then define $\mathscr{F}(\mathbf{x}, t) = \mathscr{F}(\mathbf{x}, \max(T - s, t))$. Note that this extension preserves the Lipschitz constant.

2.1. Data Collection.

In applications, one can measure the trajectory of the above evolutionary process and collect trajectory data for multiple initial states. Suppose N initial states are randomly sampled from a probability measure $\rho(0)$ supported on $\mathcal{M}(0)$:

$$\{\mathbf{x}_1(0), \dots, \mathbf{x}_N(0)\} \overset{\text{iid}}{\sim} \rho(0).$$

Let $0 = t_1 < t_2 < \cdots < t_{\mathsf{T}} = T$ be a time grid, denoted $\mathbb{T} = \{t_1, \dots, t_{\mathsf{T}}\}$.

$$\{\mathbf{x}_n\}_{n=1}^N := \{(\mathbf{x}_n(t_k))_{k=1}^{\mathsf{T}}\}_{n=1}^N \in \mathbb{R}^{N\times\mathsf{T}\times D},\ \text{where } \mathbf{x}_n(t_k) = \mathscr{F}(\mathbf{x}_n(0), t_k).$$

Given this dataset, our goal is to construct a low-dimensional surrogate model $\mathscr{W}_{model}$: $\mathcal{M}(0) \times \mathbb{T} \to \mathbb{R}^D$ such that

$$\mathscr{W}_{model}(\mathbf{x}(0), t_k) \approx \mathscr{F}(\mathbf{x}(0), t_k),\ \forall \mathbf{x}(0) \in \mathcal{M}(0),\ t_k \in \mathbb{T}.$$

After training, one can predict the solution trajectory for a new initial condition sampled from $\rho(0)$ by evolving the surrogate model in the low-dimensional latent space.

Notation	Explanation
$[w_i, w_{i+1}]$	Time range in window i
$\mathbb{T}$	Time grid: $\{t_1, \ldots, t_{\mathsf{T}}\}$
$\mathbb{T}^i$	$\mathbb{T} \cap [w_i, w_{i+1}]$
$\boldsymbol{T}_i$	Size of $\mathbb{T} \cap (w_i, w_{i+1}]$
$\mathscr{F}$	Evolution operator
$\mathscr{E}^i_*/\mathscr{D}^i_*$	Window i's oracle encoder/decoder
$\mathscr{P}^i_*$	Window i's oracle propagator $= \mathscr{E}^i_* \circ \mathscr{F}(\cdot, \Delta t) \circ \mathscr{D}^i_*$.
$\mathscr{T}^i_*$	Window i's oracle transcoder $= \mathscr{E}^{i+1}_* \circ \mathscr{E}^i_*$.
$\mathscr{E}^i_{\mathrm{NN}}/\mathscr{D}^i_{\mathrm{NN}}/\mathscr{P}^i_{\mathrm{NN}}/\mathscr{T}^i_{\mathrm{NN}}$	Window i's oracle encoder/decoder/propagator/transcoder
$\mathcal{M}([a, b])$	Trajectory manifold from time a to b
$\mathcal{Z}([a, b])$	Latent space from time a to b

TABLE 1
Notation overview.

2.2. WeldNet Training. Training a WeldNet model involves four stages, shown in Figure 2: 1) Window splitting, 2) Autoencoder training, 3) Propagator training, 4) Transcoder training. We provide details of each stages below. We also provide an overview of notations is provided in Table 1.

1) Window splitting. A WeldNet model with W windows divides the time domain into W sequentially overlapping windows, with the i-th window denoted by $[w_i, w_{i+1}]$ and $\bigcup_{i=1}^{\mathsf{W}} [w_i, w_{i+1}] = [0, T]$. Suppose $\{w_i\}_{i=1}^{\mathsf{W}+1} \subset \{t_k\}_{k=1}^{\mathsf{T}}$, i.e., the endpoints of each window are on the time grid.

2) Autoencoder and Propagator training. WeldNet first trains W autoencoders and W propagators, where the ith autoencoder aims to learn a representation of the manifold $\mathcal{M}([w_i, w_{i+1}])$, and the ith propagator will predict the displacement required to evolve the latent code from one time discretization point to the subsequent one in latent manifold.

Fixing a latent space dimension $d \in \mathbb{N}$, We denote the domain of the decoders and propagators, also known as the **latent space**, by $\mathcal{Z}([0, T]) = [0, 1]^d \times [0, T]$. We also use the notation $\mathcal{Z}(t) = [0, 1]^d \times \{t\}$ and notation $\mathcal{Z}([a, b]) = [0, 1]^d \times [a, b]$. Note that we always incorporate time in our latent codes to track the time information.

The autoencoder reconstruction loss for the ith window is

$$\mathbf{L}^i_{\mathrm{ae}}(\mathscr{E}^i, \mathscr{D}^i) = \frac{1}{N|\mathbb{T} \cap [w_i, w_{i+1})|} \sum_{n=1}^{N} \sum_{t_k \in \mathbb{T} \cap [w_i, w_{i+1}]} \left\| \mathscr{D}^i(\mathscr{E}^i(\mathbf{x}_n(t_k))) - \mathbf{x}_n(t_k) \right\|^2_{\mathbb{R}^D}, \tag{2.1}$$

where $(\mathscr{E}^i, \mathscr{D}^i)$ is an encoder-decoder pair in the ith window. The propagator loss for the ith window (denoted $\mathbf{L}^i_{\mathrm{prop}}(\mathscr{E}^i, \mathscr{P}^i)$) is

$$\frac{1}{N|\mathbb{T} \cap [w_i, w_{i+1})|} \sum_{n=1}^{N} \sum_{t_k \in \mathbb{T} \cap [w_i, w_{i+1})} \left| \mathscr{P}^i(\mathscr{E}^i(\mathbf{x}_n(t_k))) - \mathscr{E}^i(\mathbf{x}_n(t_{k+1})) \right|^2_{\mathbb{R}^{d+1}}, \tag{2.2}$$

where $\mathscr{P}^i$ denotes an one-step propagator for the latent code in the ith window.

We train the encoder, decoder, and propagator together using the following objective function:

$$(\mathscr{E}^i_{\mathrm{NN}}, \mathscr{D}^i_{\mathrm{NN}}, \mathscr{P}^i_{\mathrm{NN}}) \in \underset{\mathscr{E} \in \mathcal{F}^{\mathscr{E}}_{\mathrm{NN}}, \mathscr{D} \in \mathcal{F}^{\mathscr{D}}_{\mathrm{NN}}, \mathscr{P} \in \mathcal{F}^{\mathscr{P}}_{\mathrm{NN}}}{\arg\min} \mathbf{L}^i_{\mathrm{ae}}(\mathscr{E}^i, \mathscr{D}^i) + \lambda \mathbf{L}^i_{\mathrm{prop}}(\mathscr{E}^i, \mathscr{P}^i), \tag{2.3}$$

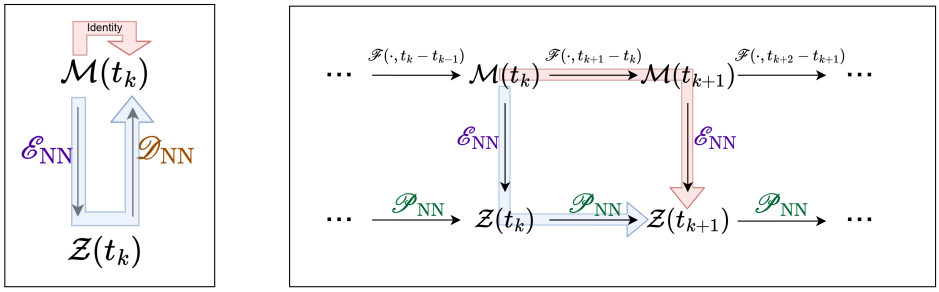

FIG. 3. *Diagram of the autoencoder loss (left) and propagator loss (right), where the loss is the MSE between the output of the red arrow and the blue arrow.*

where $\mathcal{F}^{\mathscr{D}}_{\text{NN}}$, $\mathcal{F}^{\mathscr{E}}_{\text{NN}}$, and $\mathcal{F}^{\mathscr{P}}_{\text{NN}}$ are network classes for decoder, encoder, and propagator, respectively, and $\lambda > 0$ is a hyperparameter. In this work, we use $\lambda = 0.1$ for all experiments. Figure 3 diagrams the autoencoder and propagator losses (the loss is the mean squared error between applying the functions in the red path and applying the functions in the blue path).

3) Propagator finetuning. After we train the autoencoder and propagator together, we then finetune the propagator in order to reduce the accumulation of error that occurs when applying $\mathscr{P}^i$ to propagate a latent code over multiple time steps. Specifically, we freeze the encoder and train the propagator with the objective (where $k(w_i)$ is the time index of w_i, i.e. the first time step in window i):

(2.4)
$$\mathscr{P}^i_{\text{NN}} = \underset{\mathscr{P}^i \in \mathcal{F}^{\mathscr{P}}_{\text{NN}}}{\arg\min} \frac{1}{N\boldsymbol{T}_i} \sum_{n=1}^{N} \sum_{s=1}^{\boldsymbol{T}_i} \left\| \left(\bigcirc_{k=1}^{s} \mathscr{P}^i \right) \left(\mathscr{E}^i \left(\mathbf{x}_n(t_{k(w_i)}) \right) \right) - \mathscr{E}^i \left(\mathbf{x}_n(t_{k(w_i)+s}) \right) \right\|^2_{\mathbb{R}^{d+1}},$$

where we denote $\boldsymbol{T}_i = |\mathbb{T} \cap (w_i, w_{i+1}]|$.

The loss for the finetuning of propagator network is illustrated in the Figure 4.

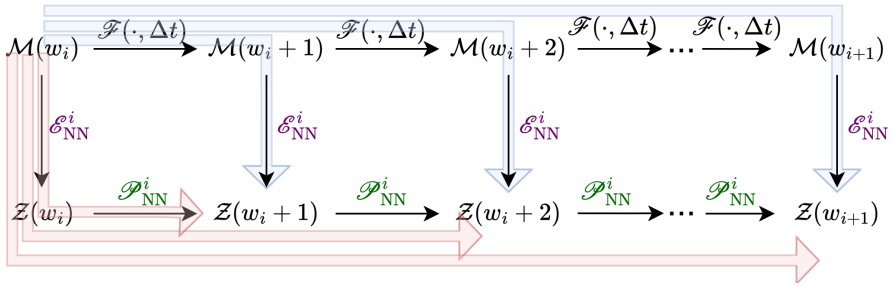

FIG. 4. *Diagram for propagator accumulation loss. The loss is the average of the mean squared error between the blue and red arrows that point to the same symbol.*

4) Transcoder training. We have trained autoencoders and propagators on each time window separately. In order to connect the windows, we train transcoder networks on the overlap between the windows. The goal of the ith transcoder is to connect the codes at the end of window i with the codes at the beginning of window $i+1$. To train the transcoder, we will propagate codes from the beginning of window

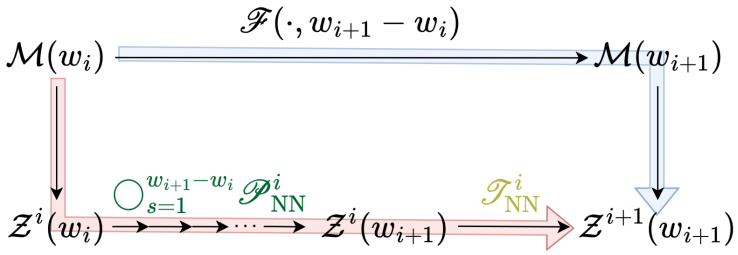

FIG. 5. *Diagram of the transcoder loss, where the loss is the MSE between the output of the red arrow and the blue arrow.*

i to the end. Specifically, we use the objective $\mathscr{T}^i_{\mathrm{NN}} =$

(2.5)
$$\arg\min_{\mathscr{T}^i \in \mathcal{F}^{\mathscr{T}}_{\mathrm{NN}}} \frac{1}{N} \sum_{n=1}^{N} \left\| \mathscr{T}^i \left(\left(\bigcirc_{s=1}^{|\mathbb{T} \cap (w_i, w_{i+1}]|} \mathscr{P}^i \right) (\mathscr{E}^i_{\mathrm{NN}}(\mathbf{x}_n(w_i))) \right) - \mathscr{E}^{i+1}_{\mathrm{NN}}(\mathbf{x}_n(w_{i+1})) \right\|^2_{\mathbb{R}^{d+1}}$$

where $\mathcal{F}^{\mathscr{T}}_{\mathrm{NN}}$ denotes the network class for transcoders. Figure 5 diagrams the loss function for the transcoder.

2.3. WeldNet Inference. After WeldNet is trained, one can predict the trajectory given a new initial state $\mathbf{x}(0)$. The inference involves encoding the initial state to the latent code, evolving the latent code in time, and decoding at the final time. This inference procedure can be represented by our WeldNet model, denoted by $\mathscr{W}_{\mathrm{NN}}$, such that, for any $\mathbf{x}(0) \in \mathcal{M}(0)$ and $t_k \in \mathbb{T} \cap (w_i, w_{i+1}]$, WeldNet gives rise to (where $k(w_i)$ is the time grid index of the time w_i)

(2.6)
$$\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) := \mathscr{D}^i_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{k-k(w_i)} \mathscr{P}^i_{\mathrm{NN}} \circ \bigcirc_{j=1}^{i-1} (\mathscr{T}^j_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{|\mathbb{T} \cap (w_j, w_{j+1}]|} \mathscr{P}^j_{\mathrm{NN}}) \circ \mathscr{E}^1_{\mathrm{NN}}(\mathbf{x}(0)).$$

Figure 1 shows the process of encoding, propagating, and decoding for inference with WeldNet.

3. Approximation Theory. In this section, we prove the approximation ability of WeldNet for a large class of evolutionary operators. The low-dimensional structure in the trajectory manifold results in the existence of oracle maps for charts, time-evolution operator, and transition maps which can be approximated by the autoencoder, propagator, and transcoder networks respectively.

3.1. Preliminaries.

3.1.1. Neural Networks. In this work, we consider ReLU networks as follows:

DEFINITION 3.1 (Feedforward Neural Network (FNN)). *Let $L \in \mathbb{N}$ and suppose $\mathfrak{W}_1, \ldots, \mathfrak{W}_{L+1}$ are weight matrices and $\mathfrak{b}_1, \ldots, \mathfrak{b}_{L+1}$ are bias vectors with $\mathfrak{W}_\ell \in \mathbb{R}^{d_{\ell-1} \times d_\ell}$ and $\mathfrak{b}_\ell \in \mathbb{R}^{d_\ell}$ for all $\ell \in [L+1]$. Let $\theta = (\mathfrak{W}_1, \mathfrak{b}_1, \ldots, \mathfrak{W}_{L+1}, \mathfrak{b}_{L+1})$. We define a **feedforward neural network (FNN)** with weights θ, denoted f_{NN}, as a function of the form:*

$$f_{\mathrm{NN}}(x) = \mathfrak{W}_{L+1}\sigma(\mathfrak{W}_L\sigma(\cdots \mathfrak{W}_1 x + \mathfrak{b}_1) \cdots + \mathfrak{b}_L) + \mathfrak{b}_{L+1}.$$

*We say the **depth** and **width** of f_{NN} are L and $W = \max_\ell d_\ell$ respectively.*

A FNN with depth L and width W is therefore a composition of $L+1$ functions, all but the last one of which involves an affine transform with output dimension at most W. With the (maximum) depth and width specified, we can define a function class of feedforward neural networks.

DEFINITION 3.2 (FNN Class). *Let $d_{in}, d_{out}, L, W \in \mathbb{N}$. The class of feedforward neural networks (**FNN class**) with d_{in} inputs, d_{out} outputs, depth at most L, and width at most W is denoted $\mathcal{F}_{\mathrm{NN}}(d_{in}, d_{out}, L, W)$.*

3.1.2. Manifolds.

DEFINITION 3.3 (Lipschitz Function). *Let A, B be metric spaces with metrics d_A and d_B respectively, and let $L > 0$. A function $f : A \to B$ is called L-**Lipschitz** if*

$$\sup_{x \neq y \in A} \frac{\|f(x) - f(y)\|_{d_B}}{\|x - y\|_{d_A}} \leq L,$$

*and we define the **Lipschitz constant** of f, denoted $\mathrm{Lip}_A(f)$, as the smallest L for which $f : A \to B$ is L-Lipschitz.*

DEFINITION 3.4 (Manifold). *A d-dimensional **manifold** $\mathcal{M}$ is a topological space endowed with a collection of **charts** $(\phi_i : U_i \subseteq \mathcal{M} \to \tilde{U}_i \subseteq \mathbb{R}^d)_{i \in \mathcal{I}}$ (for some index set $\mathcal{I}$) such that for all $i \in \mathcal{I}$, ϕ_i is a homeomorphism between the open sets U_i and $\tilde{U}_i$, and we have $\bigcup_{i \in \mathcal{I}} U_i = \mathcal{M}$. We say $\mathcal{M}$ is a **smooth** manifold if for all charts ϕ_i and ϕ_j on $\mathcal{M}$, the function*

$$\phi_j \circ \phi_i^{-1} : \tilde{U}_i \cap \tilde{U}_j \subseteq \mathbb{R}^d \to \tilde{U}_i \cap \tilde{U}_j \subseteq \mathbb{R}^d$$

*is smooth in the usual Euclidean sense. Finally, for any $f : \mathcal{M} \to \mathbb{R}^k$, we say that f has **injective derivative** if for each chart $\phi_i : U_i \to \tilde{U}_i$, we have the function $f \circ \phi_i^{-1}$ is differentiable in the Euclidean sense, and its derivative is nonzero on its domain.*

DEFINITION 3.5 (Smooth Embedding). *We say that a manifold is **smoothly embedded** in $\mathbb{R}^D$ if $\mathcal{M} \subseteq \mathbb{R}^D$ and the inclusion function $\iota : \mathcal{M} \to \mathbb{R}^D$ is a smooth homeomorphism onto its image with injective derivative.*

More details on manifolds can be found in standard texts on differential geometry [25]. Next, we introduce a regularity/curvature parameter for embedded submanifolds of Euclidean space known as reach. Informally speaking, the reach of a manifold describes the curvature of the embedded manifold in the ambient Euclidean space. It is the radius of the largest "ball" (interior of a hypersphere) you can roll around the manifold without crossing it.

DEFINITION 3.6 (Reach [15]). *Let $\mathcal{M} \subseteq \mathbb{R}^D$ be an isometrically embedded manifold. The reach of $\mathcal{M}$, denoted $\boldsymbol{\tau}(\mathcal{M})$, is*

$$\begin{aligned} &\inf\{r > 0 : \exists x \neq y \in \mathcal{M}, v \in \mathbb{R}^D \\ &\quad \textit{such that } r = \|x - v\| = \|y - v\| = \textit{dist}(v, \mathcal{M})\}. \end{aligned}$$

Linear subspaces have reach ∞, and the reach of a hypersphere equals to its radius. It is well-known that compact embedded manifolds have positive reach [50]. The larger the reach, the easier the manifold is to be represented with neural networks, and that will affect the bounds in neural network approximation theory [10, 30].

3.2. Main Theorems. According to Section 2.1, the time-evolution operator $\mathscr{F}$ is defined on the set $\mathcal{M}([0,T]) \times [0,T]$. We will assume that $\mathscr{F}$ is Lipschitz for all times:

Assumption 1. Assume $\mathscr{F}$ is Lipschitz, i.e. $\mathrm{Lip}(\mathscr{F}) :=$

$$\sup_{\substack{t\in[0,T],\\ s\in[0,T-t]}} \mathrm{Lip}_{\mathcal{M}(t)}(\mathscr{F}(\cdot,s)) = \sup_{\substack{t\in[0,T],\\ s\in[0,T-t]}} \sup_{\substack{\mathbf{x}(t)\neq\mathbf{y}(t)\\ \in\mathcal{M}(t)}} \frac{\|\mathscr{F}(\mathbf{x}(t),s) - \mathscr{F}(\mathbf{y}(t),s)\|}{\|\mathbf{x}(t)-\mathbf{y}(t)\|} < \infty.$$

In this work, we will consider evolutionary processes whose initial conditions are sampled from an embedded d-dimensional manifold. This means that each initial condition can be described (locally) by d parameters. More specifically, we will assume that the collection of trajectories at various segmented times form a $(d+1)$-dimensional manifold, with d-dimensions in parameter space and one dimension given by time. For convenience, we will use the notation $\mathcal{Z}(t) = [0,1]^d \times \{t\}$ and $\mathcal{Z}([a,b]) = [0,1]^d \times [a,b]$.

Assumption 2 (Segmented Manifold). Suppose there exist $0 = s_1 < s_2 < \cdots < s_{\mathsf{S}+1} = T$ such that for all $i \in [\mathsf{S}]$, the subset $\mathcal{M}([s_i, s_{i+1}])$ is a compact $(d+1)$-dimensional smooth manifold smoothly embedded in $\mathbb{R}^D$ with reach $\boldsymbol{\tau}(\mathcal{M}([s_i,s_{i+1}])) > 0$. In addition, for all $i \in [\mathsf{S}]$ we assume there is a smooth function $\mathscr{D}^i_*$ that maps from a Euclidean space to the manifold with smooth inverse $\mathscr{E}^i_* = (\mathscr{D}^i_*)^{-1}$:

$$\mathscr{E}^i_* : \mathcal{M}([s_i,s_{i+1}]) \to \mathcal{Z}([s_i,s_{i+1}]), \quad \mathscr{D}^i_* : \mathcal{Z}([s_i,s_{i+1}]) \to \mathcal{M}([s_i,s_{i+1}]),$$

such that for any $t \in [s_i, s_{i+1}]$, $\mathcal{M}(t) = \mathscr{D}^i_*(\mathcal{Z}(t))$.

In Assumption 2, $\mathscr{E}^i_*$ and $\mathscr{D}^i_*$ serve as the oracle encoder and decoder for the trajectory manifold for each segment $i \in [\mathsf{S}]$. We denote the Lipschitz constant of the oracle encoder on each time segment by $\mathrm{Lip}_{\mathscr{E}^i} = \sup_{t\in[s_i,s_{i+1}]} \mathrm{Lip}_{\mathcal{M}(t)}(\mathscr{E}^i)$ and the oracle decoder in each time segment by $\mathrm{Lip}_{\mathscr{D}^i} = \sup_{t\in[s_i,s_{i+1}]} \mathrm{Lip}_{\mathcal{Z}(t)}(\mathscr{D}^i)$.

We will first establish the result where we set the windows to be exactly aligned with the manifold segments in Assumption 2. In other words, we will choose $\mathsf{W} = \mathsf{S}$ and define $0 < w_1 < \cdots < w_{\mathsf{W}+1} = T$ such that $w_i = s_i$ for all $i \in [\mathsf{W}]$. We construct encoder, decoder, and propagator networks over each window (which is the same as a segment). In order to translate between adjacent windows, we use transcoder networks. The inference by WeldNet for any initial state $\mathbf{x}(0) \in \mathcal{M}(0)$ can be represented in (2.6).

We introduce a latent time-evolution map, defined on the domain $\mathcal{Z}([w_i, w_{i+1}])$ denoted as $\mathscr{P}^i_*$ such that $\mathscr{P}^i_*(\mathbf{z},t) = \mathscr{E}^i_*(\mathscr{F}(\mathscr{D}^i_*(\mathbf{z}),t))$ for $t \le w_{i+1} - w_i$, which gives the evolution of latent code for $\mathscr{F}$ within window i: $\mathscr{D}^i_* \circ \mathscr{P}_* \circ \mathscr{E}^i_* = \mathscr{F}$. The evolution of $\mathscr{P}^i_*$ operates in latent space, yet it matches the dynamics of the time-evolution operator $\mathscr{F}$ on a section of the trajectory manifold. We will call $\mathscr{F}$ the time-evolution operator, and $\mathscr{P}^i_*$ the **oracle propagator** for the ith window.

We are primarily interested in the case where the latent codes follow an ODE within each segment:

Assumption 3 (Latent Dynamics). Using the notation in Assumption 2, suppose for all $i \in [\mathsf{S}]$, there is a Lipschitz function $g^i : \mathcal{Z}([s_i,s_{i+1}]) \to \mathbb{R}^{d+1}$ such that for all $\mathbf{z}(s_i) \in \mathcal{Z}(s_i)$, the latent code $\mathbf{z}(t) = \mathscr{P}^i_*(\mathbf{z}(s_i), t - s_i)$ for $t \in [s_i, s_{i+1}]$ satisfies $\frac{\partial \mathbf{z}}{\partial t}(t) = g^i(\mathbf{z}(t))$.

The ODE in Assumption 3 models the dynamics of the latent code. In particular, the $(d+1)$th coordinate of the latent code $\mathbf{z}(t)$ is time, i.e. $\mathbf{z}(t)_{d+1} = t$ and therefore

the $(d+1)$th coordinate of g^i is 1, i.e. $g^i_{d+1} = 1$. We denote the Lipschitz constant of g^i (for the ith segment) as $\mathrm{Lip}(g^i) := \sup_{t\in[s_i,s_{i+1}]} \mathrm{Lip}_{\mathcal{Z}(t)}(g^i)$ and the Lipschitz constant of $g = \{g^i\}_{i=1}^{\mathsf{S}}$ (among all segments) as $\mathrm{Lip}(g) = \sup_{i\in[\mathsf{W}]} \mathrm{Lip}(g^i)$.

Our first result (presented in Lemma 3.7 below) gives an approximation guarantee for WeldNet in the latent dynamics setting, assuming that the windows are set to be equal to the manifold segments.

LEMMA 3.7. *Suppose Assumptions 1, 2, and 3 hold. Assume the time grid $\mathbb{T}$ is uniform such that $t_{k+1} - t_k = \Delta t$ for all $k \in [\mathsf{T}-1]$. Let $\epsilon > 0$, and suppose there are $\mathsf{W} = \mathsf{S}$ windows such that for all $i \in [\mathsf{W}]$, we have $w_i = s_i$ and each window has number of time steps $|\mathbb{T}^i| > 1 + \mathrm{Lip}(g)\|g\|_{L^\infty}(w_{i+1} - w_i)^2 e^{\mathrm{Lip}(g)(w_{i+1}-w_i)}\epsilon^{-1}$. Then for each $i \in [\mathsf{W}]$ there exist an encoder network $\mathscr{E}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(D, d+1, L_{\mathscr{E}^i}, W_{\mathscr{E}^i})$, a decoder network $\mathscr{D}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, D, L_{\mathscr{D}^i}, W_{\mathscr{D}^i})$, a propagator network $\mathscr{P}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{P}^i}, W_{\mathscr{P}^i})$, and a transcoder network (for $i < \mathsf{W}$) $\mathscr{T}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{T}^i}, W_{\mathscr{T}^i})$ such that for any $k \in [\mathsf{T}]$, the WeldNet $\mathscr{W}_{\mathrm{NN}}$ given in (2.6) guarantees*

$$\sup_{\mathbf{x}(0)\in\mathcal{M}(0)} \|\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) - \mathscr{F}(\mathbf{x}(0), t_k)\|_{\mathbb{R}^D} < \epsilon.$$

The network parameters are

$$L_{\mathscr{E}^i} = O\left(\log^2(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{E}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{D}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{D}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{P}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{P}^i} = O\left((\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{T}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{T}^i} = O\left((\mathsf{S}/\epsilon)^{d}\right)$$

where we hide constants depending on (for all $i \in [\mathsf{W}]$) $\|g_i\|_{L^\infty}$, $\mathrm{Lip}(g_i)$, $\log(D)$, d, $\boldsymbol{\tau}(\mathcal{M}([s_i, s_{i+1}]))$, $\max_i \mathrm{Lip}(\mathscr{E}^i_*)$, $\max_i \mathrm{Lip}(\mathscr{D}^i_*)$, $\mathrm{Lip}(\mathscr{F})$, the volume of $\mathcal{M}([s_i, s_{i+1}])$, and $\sup_{\mathbf{x}\in\mathcal{M}([s_i,s_{i+1}])} \|\mathbf{x}\|_{\mathbb{R}^D}$.

Lemma 3.7 provides a representation guarantee of WeldNet, and the network size scales crucially with the intrinsic dimension d instead of the data dimension D, which demonstrates the efficiency of model reduction. The proof is deferred to Appendix B.1.

Lemma 3.7 requires the number of windows to be equal to the number of segments. One major advantage of WeldNet is the ability to train with a number of windows potentially higher than the number of segments. Fortunately, the approximation guarantee of Lemma 3.7 can be extended to this scenario, as long as the windows partition each segment. This is the subject of the next theorem, which follows from Lemma 3.7.

DEFINITION 3.8. *We say a sequence $\{w_i\}_{i=1}^{\mathsf{W}+1}$ with $0 = w_1 < w_2 < \cdots < w_{\mathsf{W}+1} = T$ **subdivides the segments** $\{s_j\}_{j=1}^{\mathsf{S}+1}$ if for all $i \in [\mathsf{W}]$, there is a unique $j \in [\mathsf{S}]$ such that $[w_i, w_{i+1}] \subseteq [s_j, s_{j+1}]$. We denote by $\pi : [\mathsf{W}] \to [\mathsf{S}]$ that assigns each window index to the unique segment index it lies within.*

THEOREM 3.9. *Suppose Assumptions 1, 2, and 3 hold. Assume the time grid $\mathbb{T}$ is uniform such that $t_{k+1} - t_k = \Delta t$ for all $k \in [\mathsf{T}-1]$. Let $\epsilon > 0$ and suppose there are $\mathsf{W} \geq \mathsf{S}$ windows so that $0 = w_1 < w_2 < \cdots < w_{\mathsf{W}+1}$ subdivides the segments $\{s_j\}_{j=1}^{\mathsf{S}+1}$. Additionally, suppose for all $i \in [\mathsf{W}]$, each window has number of time steps $|\mathbb{T}^i| > 1 + \mathrm{Lip}(g)\|g\|_{L^\infty}(w_{i+1} - w_i)^2 e^{\mathrm{Lip}(g)(w_{i+1}-w_i)}\epsilon^{-1}$. Then for each $i \in [\mathsf{W}]$ there exist an*

encoder $\mathscr{E}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(D, d+1, L_{\mathscr{E}^i}, W_{\mathscr{E}^i})$*, a decoder* $\mathscr{D}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, D, L_{\mathscr{D}^i}, W_{\mathscr{D}^i})$*, a propagator* $\mathscr{P}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{P}^i}, W_{\mathscr{P}^i})$*, and a transcoder (for* $i < \mathsf{W}$*)* $\mathscr{T}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{T}^i}, W_{\mathscr{T}^i})$ *such that for any* $k \in [\mathsf{T}]$*, the WeldNet* $\mathscr{W}_{\mathrm{NN}}$ *given in* (2.6) *guarantees*

$$\sup_{\mathbf{x}(0)\in\mathcal{M}(0)} \|\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) - \mathscr{F}(\mathbf{x}(0), t_k)\|_{\mathbb{R}^D} < \epsilon.$$

The network parameters are

$$L_{\mathscr{E}^i} = O\left(\log^2(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{E}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{D}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{D}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{P}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{P}^i} = O\left((\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{T}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{T}^i} = O\left((\mathsf{S}/\epsilon)^{d}\right)$$

where we hide constants depending on (for all $i \in [\mathsf{W}]$*)* $\|g_{\pi(i)}\|_{L^\infty}$*,* $\mathrm{Lip}(g_{\pi(i)})$*,* $\log(D)$*,* d*,* $\boldsymbol{\tau}(\mathcal{M}([s_{\pi(i)}, s_{\pi(i)+1}]))$*,* $\max_i \mathrm{Lip}(\mathscr{E}^{\pi(i)}_*)$*,* $\max_i \mathrm{Lip}(\mathscr{D}^{\pi(i)}_*)$*,* $\mathrm{Lip}(\mathscr{F})$*, the volume of* $\mathcal{M}([s_{\pi(i)}, s_{i+1}])$*, and* $\sup_{\mathbf{x}\in\mathcal{M}([w_i, w_{i+1}])} \|\mathbf{x}\|_{\mathbb{R}^D}$*.*

Theorem 3.9 is a generalization of Lemma 3.7 (they both rely on Assumptions 1, 2, and 3), and its proof is deferred to Appendix B.2. Without Assumption 3, we can still establish an approximation error guarantee for WeldNet, but with a different construction for the propagator network. The proof of the following theorem is deferred to Appendix B.3.

THEOREM 3.10. *Suppose Assumption 1 and 2 hold. Let* $\epsilon > 0$*, and suppose there are* $\mathsf{W} \geq \mathsf{S}$ *windows so that* $0 = w_1 < w_2 < \cdots < w_{\mathsf{W}+1}$ *subdivides the segments. Suppose the time grid* T *is uniform such that* $t_{k+1} - t_k = \Delta t$ *for all* $k \in [\mathsf{T}-1]$*. For all* $i \in [\mathsf{S}]$*, let* $\overline{\boldsymbol{T}_s} = |\mathbb{T} \cap [s_i, s_{i+1}]|$ *denote the number of time steps in segment* i*. Then for each* $i \in [\mathsf{W}]$*, there exist an encoder network* $\mathscr{E}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(D, d+1, L_{\mathscr{E}^i}, W_{\mathscr{E}^i})$*, a decoder network* $\mathscr{D}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, D, L_{\mathscr{D}^i}, W_{\mathscr{D}^i})$*, a propagator network* $\mathscr{P}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{P}^i}, W_{\mathscr{P}^i})$*, and a transcoder network (for* $i < \mathsf{W}$*)* $\mathscr{T}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{T}^i}, W_{\mathscr{T}^i})$ *such that for any* $k \in [\mathsf{T}]$*, the WeldNet* $\mathscr{W}_{\mathrm{NN}}$ *given in* (2.6) *guarantees*

$$\sup_{\mathbf{x}(0)\in\mathcal{M}(0)} \|\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) - \mathscr{F}(\mathbf{x}(0), t_k)\|_{\mathbb{R}^D} < \epsilon.$$

The network parameters are

$$L_{\mathscr{E}^i} = O\left(\log^2(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{E}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{D}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{D}^i} = O\left(D\,(\mathsf{S}/\epsilon)^{d+1}\right)$$

$$L_{\mathscr{P}^i} = O\left(\overline{\boldsymbol{T}_{\pi(i)}}\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{P}^i} = O\left(\left(\overline{\boldsymbol{T}_{\pi(i)}}/\epsilon\right)^{d}\right)$$

$$L_{\mathscr{T}^i} = O\left(\log(\mathsf{S}/\epsilon)\right),\ W_{\mathscr{T}^i} = O\left((\mathsf{S}/\epsilon)^{d+1}\right)$$

where we hide constants in O *depending on (for all* $i \in [\mathsf{W}]$*)* d*,* $\log(D)$*,* $\boldsymbol{\tau}(\mathcal{M}([0,T]))$*,* $\max_i \mathrm{Lip}(\mathscr{E}^i_*)$*,* $\max_i \mathrm{Lip}(\mathscr{D}^i_*)$*,* $\mathrm{Lip}(\mathscr{F})$*, volume of* $\mathcal{M}([0,T])$*, and* $\sup_{\mathbf{x}\in\mathcal{M}([0,T])} \|\mathbf{x}\|_{\mathbb{R}^D}$*.*

Remark. The main difference between Theorem 3.10 and Theorem 3.9 is that the former does not assume Assumption 3, which makes the approximation of the propagator network more challenging, resulting in the large propagator size. We briefly remark that that the idea of the proof of Theorem 3.10 does not require a uniform time grid size, so it can be extended to the case of non-uniform time grids (in which case the size of the propagator will depend on the minimum and maximum time grid spacing).

4. Experiments.

4.1. Data Generation.

For simplicity, we start from an evolutionary ordinary differential equation (ODE)

$$\begin{aligned} \partial_t \mathbf{x} &= \mathfrak{F}(\mathbf{x}(t), t), \\ \mathbf{x}(0) &= \mathbf{x}_0 \end{aligned} \tag{4.1}$$

with solution $\mathbf{x}(t) : [0, T] \to \mathbb{R}^D$, where $T > 0$ is the end time. We will measure the solution of the ODE in (4.1) at discrete time points; let $0 = t_1 < t_2 < \ldots < t_{\mathsf{T}} = T$ be the discretized time locations for some fixed $\mathsf{T} \in \mathbb{N}$. The data for the ODE represents a trajectory of (4.1) with initial condition $\mathbf{x}(t_1) = \mathbf{x}_0$: $\{\mathbf{x}(t_1), \mathbf{x}(t_2), \ldots, \mathbf{x}(t_{\mathsf{T}})\} \subseteq \mathbb{R}^D$.

In the data-driven framework, we collect trajectory data from multiple initial conditions. Let $\{\mathbf{x}_n(t_1)\}_{n=1}^N$ be N sets of initial conditions. The collection of discretized trajectory data with these initial conditions is denoted by

$$\{\mathbf{x}_n(t_1), \mathbf{x}_n(t_2), \ldots, \mathbf{x}_n(t_{\mathsf{T}})\}_{n=1}^N, \tag{4.2}$$

which serves as the training data (and can be thought of as N elements of $\mathbb{R}^{\mathsf{T} \times D}$).

We next consider a partial differential equation (PDE)

$$\begin{aligned} \partial_t \mathfrak{u} &= \mathfrak{F}(\mathfrak{u}(\mathbf{x}, t), t), \\ \mathfrak{u}(\mathbf{x}, 0) &= f(\mathbf{x}), \quad \forall \mathbf{x} \in \Omega, \end{aligned} \tag{4.3}$$

with solution $\mathfrak{u} : \Omega \times [0, \mathsf{T}] \to \mathbb{R}$ where $\Omega \subset \mathbb{R}^{d_\Omega}$ is the spatial domain. We will discretize the solution of the PDE in the spatial and temporal domain. Let $\mathbb{X} = (X_1, \ldots, X_D) \subseteq \Omega$ be a discretization set in the spatial domain and consider the same temporal discretization as before. The discretized data for the PDE solution $\mathfrak{u}$ is denoted $\{\mathfrak{u}(\mathbb{X}, t_1), \mathfrak{u}(\mathbb{X}, t_2), \ldots, \mathfrak{u}(\mathbb{X}, t_T)\} \subseteq \mathbb{R}^D$, which represents a discretized trajectory of the PDE in (4.3) with the initial condition f.

Given initial conditions $\{f_n\}_{n=1}^N$ and solution trajectories $\{\mathfrak{u}_n\}_{n=1}^N$ (for all $n \in [N]$, $\mathfrak{u}_n$ solves (4.3) with initial condition $f = f_n$), the trajectory data in this case has the same form as (4.2) if we use the notation $\mathbf{x}_n(t_k) = \mathfrak{u}_n(\mathbb{X}, t_k)$, that is, $\mathbf{x}_n(t_k) \in \mathbb{R}^D$ with ith component given by $\mathfrak{u}_n(X_i, t_k)$.

Many differential equations in applications can be complex to simulate - the discretization set size grows exponentially in the spatial dimension d_Ω. When the spatial dimension d_Ω is higher than one, a large number of discretization points are needed. Our goal is to build a low-dimensional surrogate model based on training data of the form (4.2) for ODEs, PDEs, and other evolutionary processes. After training, one can predict the solution trajectory for a new initial condition by evolving the surrogate model in a low-dimensional space.

4.2. Benefit of Windowed Approach.

In this section, we illustrate the benefit of using a windowed autoencoder approach on a simple but instructive example.

Consider the one-dimensional transport equation for $T = 0.3$, given by

$$u_t = -u_x; \quad u(x,0) = g(x),\ x \in (0,1), \tag{4.4}$$

with zero Dirichlet boundary conditions. We consider the weak version of this PDE, so the initial condition g does not have to be differentiable everywhere.

We consider initial conditions containing two hats. Let $\sigma(x) = \max(x, 0)$, and fix $\epsilon = 0.05$. For any α and t, we define the "hat" function with height α, location s, and width ϵ as

$$H_{\alpha,s,\epsilon}(x) = 2\alpha/\epsilon \left(\sigma(x-s) - 2\sigma\left(x - s - \frac{\epsilon}{2}\right) + \sigma(x-s-\epsilon)\right).$$

We consider the following one-parameter set of initial conditions:

$$\hat{g}_{\text{tscale}} = \{H_{a,0.1,0.05}(x) + H_{2.5,0.2,0.05}(x) : a \in [1,4]\} \tag{4.5}$$

We discretize the spatial domain $[0, 1]$ with 512 equally spaced points. We consider a probability measure on the set $\hat{g}_{\text{tscale}}$ obtained by sampling the parameter a uniformly from $[1, 4]$ and then outputting the discretized initial condition corresponding to that value of a. Formally, we define a function $G_{\text{tscale}} : [1,4] \to \mathbb{R}^{512}$ that maps from parameter (a) to initial condition. This allows us to construct the initial measure $\rho_{\text{tscale}}(0) = (G_{\text{tscale}})_\sharp \mathbf{Unif}([1,4])$. We sample by $\rho_{\text{tscale}}(0)$ by first choosing $1 \le a \le 4$ uniformly and outputting the corresponding (discretized) initial condition for a as defined in (4.5).

The trajectory manifold $\mathcal{M}_{\text{tscale}}$ is obtained by collecting solutions from the PDE in (4.4) with initial conditions from $\rho_{\text{tscale}}(0)$. For simplicity, we omit the time interval notation for the manifolds in this section. We collect 500 trajectories with 301 time steps until $T = 0.3$, so our data is of the form $((\mathbf{x}_n(t_k)))_{k=1}^{51})_{n=1}^{500}$, as detailed in Section 2.1.

We will train WeldNet models on this dataset with one window, two windows, and four windows, and we will attempt to roughly equalize the number of trainable parameters and number of training epochs. Specifically, we will implement each component of WeldNet using 3 layer neural networks so that we implement the one-window, two-window, and four-window models with networks of width 1000, 500, and 250 (respectively) and train each component for 1200, 600, and 300 epochs (respectively). The total training time for each model in minutes is 42.28, 15.73, and 8.31 (respectively).

In this section, we will use a latent space dimension of 2 for the autoencoders - equal to the intrinsic dimension of the data (see Table 2). This results are in a lower performance than a latent space dimension of 4 that we use later in the paper. We use this reduced latent space dimension to show that using windows with the estimated intrinsic dimension (instead of using a higher latent space dimension) can overcome the error accumulation issues of a single-window autoencoder model, and due to the ease of visualizing two dimensional latent spaces.

We train autoencoders and propagators together for all models. We show the latent space of each model in Figures 6, 7, and 8 for the one-window, two-window, and four-window models respectively. We show each window individually, and we show the points clouds colored by their parameter in the left plot and time in the right plot. The latent space for the one window model in Figure 6 involves pinches and twists, while the latent spaces (in each window) for the two-window and four-window are much more uniform. We note that even though the latent space in the

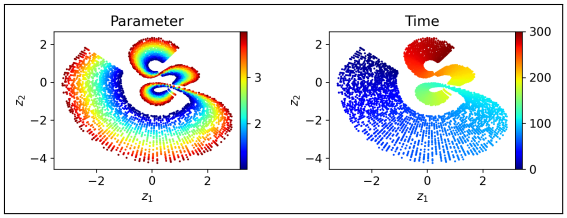

FIG. 6. *Latent space of illustrative example with one-window model.*

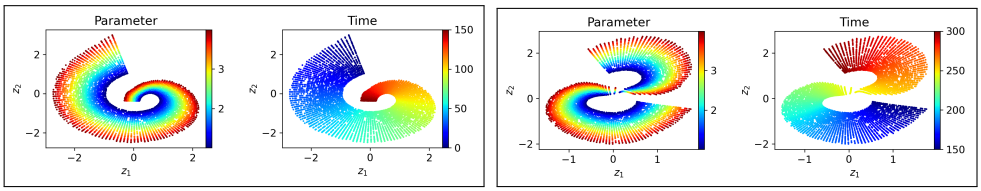

FIG. 7. *Latent space of illustrative example with two-window model.*

second window of the two-window model has a compressed area in Figure 7, the propagator is still able to learn these latent dynamics.

Next, we train propagators for the one-window model and the propagators and transcoders for the two-window and four-window models. We compare the projection and operator error of the models over all times in Figure 9. The projection error is the error of the autoencoder reconstruction on the test data (i.e. the representation or approximation error of the autoencoder in representing the solution manifold). The operator error is the prediction error of a solution given an initial condition, which depends on both the projection error and the error of the propagator network in learning the dynamics. We observe severe error accumulation in the one-window model. This problem is resolved by using more windows.

4.3. Implementation Details. The datasets for the Burgers' equation and KdV equation trajectory manifolds were generated using the chebfun package [13] in Matlab. The datasets for the transport equation trajectory manifolds were generated using the analytic solution of the equation.

4.3.1. Model Descriptions. In this section, we detail the architectures of the components of WeldNet and comparison models. Suppose the data is D-dimensional.

FF-WeldNet and Conv-WeldNet uses ReLU networks with 3-hidden layers and width 200 for the propagators and transcoders. FF-WeldNet uses feedforward ReLU networks with 3-hidden layers and width 500 for the encoder and decoder. Conv-WeldNet uses a 4 layer convolutional encoder with the convolution layers having channel size 8, 16, 32, 32, and kernel size 8, 8, 8, 4, respectively, with stride 2, and symmetric zero padding of 1, and it uses a symmetric architecture of convolution transpose layers for the decoder.

Note that a one-window WeldNet model is a normal latent dynamics model which has already been considered in the literature (e.g. see [26]), so we will use the label "AENet" to refer to that. The goal of our work is to show the benefit of multiple windows, so the performance of a one-window WeldNet model or an AENet should only be considered as a baseline.

To illustrate the advantage of dimension reduction, we train a model consisting of a propagator operating directly on the D-dimensional data. We call this approach

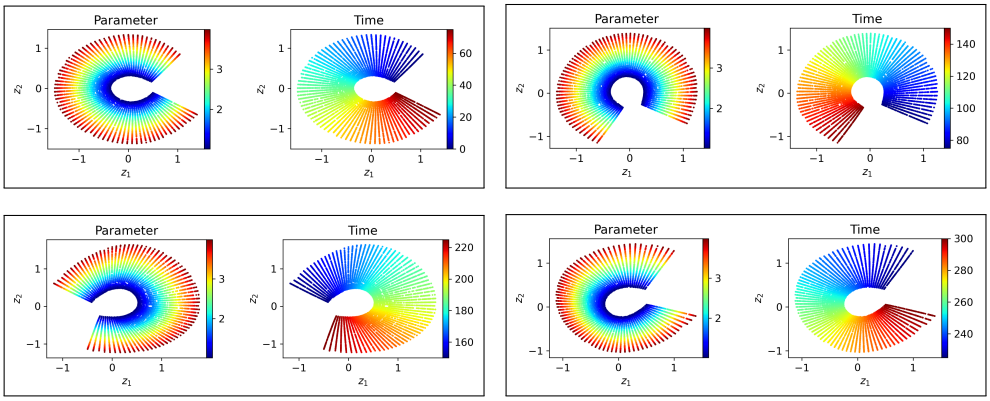

FIG. 8. *Latent space of illustrative example with four-window model.*

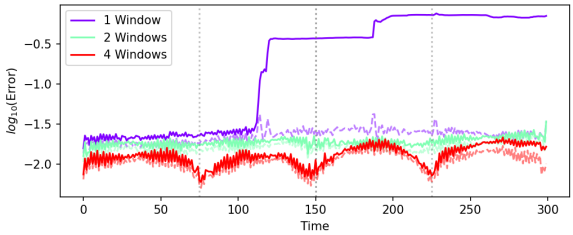

FIG. 9. *Projection (dashed) and Operator (solid) error for the example in Section 4.2.*

High Dimensional Propagator (HDP). Specifically, we train (depth 3) ReLU networks with input and output dimension equal to D ($D = 512$ for most of our examples). We use a width of 1000 since HDP only consists of a single network (while WeldNet and other models have several networks). We found that using residual/displacement based training as done for the the WeldNet propagator (as described in Section 4.3.2) leads to training instability. Thus, we will train the HDP to predict the high dimensional state at the next time directly, i.e. given an input of $x(t) \in \mathcal{M}(t) \subseteq \mathbb{R}^D$, the HDP should predict $x(t + \Delta t) \in \mathbb{R}^D$.

The Time-Input feedforward networks are three-hidden layer ReLU networks with width 1000. If the data is D-dimensional, the Time-Input feedforward network has $D+1$ input neurons and D output neurons. Given an input $u(0)$ (representing a data point initial condition) and a time t, the Time-Input network outputs the evolution of that input by t time units, i.e. it should approximately be $u(t)$.

We also compare to a model called Latent Deep Operator Network (Latent-DON) [24]. Similar to a one-window WeldNet, Latent-DON trains a single autoencoder over all times and then trains a latent prediction model that uses a Deep Operator Network (DON) architecture. DON is an operator learning method that takes time as an input and tries to predict the evolution of an initial condition by that given time, similar to time-input models, but in latent space. Given an input $u(0) \in \mathbb{R}^D$ and a time t, Latent-DON first projects to a k-dimensional latent code $z(0) \in \mathbb{R}^k$ using the encoder, and then uses the latent code and time as inputs to the DON to try to predict the evolution of that latent code by t time units. We then decode the predicted evolved

latent code, and this value should approximately be $u(t)$. See Appendix C.2 for specific details on the DON architecture used in Latent-DON.

We remark that the Time-Input and Latent-DON models have an easier learning task than WeldNet, because they are only tested to predict the evolution of the initial conditions by a given time, while WeldNet is capable of evolving data from any start time to any end time on the time grid.

LDNet is a model reduction based method for learning dynamical systems. The method trains a dynamics network that evolves latent codes and a reconstruction network that maps from latent code to function predictions on a grid. Specifically, each initial condition is assigned a latent code consisting of f fixed dimensions and k dynamic dimensions that start at 0. The fixed dimensions are given by the codes assigned to the initial conditions (which need to be known ahead of time). The dynamics network is trained to evolve the dynamic portion of each latent code, with $f + k$ input neurons and k output neurons. The latent code at a given time can be evolved by the dynamics network iteratively. The output prediction is then obtained by passing the evolved latent code through the reconstruction network. This means that the reconstruction network has $f + k$ input neurons and D output neurons. We use a width of 500 for the dynamics and reconstruction networks, and we use the ReLU activation function for both networks.

Remark. This description of LDNet is reliant on a fixed size D for the data and reconstruction network, while the original implementation of LDNet in [43] uses a grid-independent reconstruction network. We use a modified grid based version for a more direct comparison to our grid based WeldNet models, and due to training issues with the grid independent version. See Appendix C.1 for a comparison between the original LDNet and the grid based LDNet that we will subsequently use in this paper.

Weak-form Latent Space Dynamics Identification, WLaSDI, is a reduced-order modeling technique recently proposed in [51]. WLaSDI uses a one-hidden layer autoencoder with 1024 hidden neurons in the encoder and 6168 hidden neurons in the decoder with the ***silu*** activation function, $\boldsymbol{silu}(x) = \frac{x}{1+e^{-x}}$. The latent dynamics are then learned by solving a least-squares problem involving a dictionary of trial functions; specifically we apply the region-based local dynamics identification algorithm discussed in Section 3.3.2 of [51].

All WeldNet, Time-Input, HDP, Latent-DON, and LDNet models are trained with a batch size of 32, learning rate of 1e-4, with an adaptive learning schedule that decay by a factor of 0.3 with a patience of 15 epochs, to a minimum learning rate of 1e-6. We use PyTorch [38] to implement the neural networks, and we use PyTorch's imlementation of the AdamW optimization algorithm with default settings for training the models.

For WLaSDI, the autoencoder is trained with a batch size of 20, learning rate of 1e-4, with an adaptive learning rate schedule that decays by a factor of 0.1 with a patience of 10 epochs, using the implementation in [51].

4.3.2. Displacement Networks. When training the propagators, we noticed that using a "displacement" or residual architecture for the propagator and transcoder significantly aids in training. Specifically, we implement the propagator network $\mathscr{P}_{\text{NN}}$ as a function of the form:

$$\mathscr{P}_{\text{NN}}(\mathbf{z}) = \mathbf{z} + f_{\text{NN}}(\mathbf{z}),$$

where f_{NN} is a trainable neural network. We implement the transcoder in the same way. While changing the propagator and transcoder to this form does not affect

the approximation theory, we found experimentally that this change resulted in a much lower loss in the training. On the other hand, the loss explodes even for low learning rate when this method is used for the HDP model, so we only employ this displacement network strategy for WeldNet. Note that LDNet uses a displacement form for its dynamics network as part of its original specification, so we also use this strategy there.

4.4. Numerical Results. We test WeldNet on several numerical examples, similar to the illustrative example in Section 4.2, demonstrating the efficacy of our nonlinear dimension reduction and trajectory learning model. For all examples, we collect 500 trajectories with 301 time steps, so our data is of the form $((\mathbf{x}_n(t_k)))_{k=1}^{301})_{n=1}^{500}$, as detailed in Section 2.1.

We consider three versions of WeldNet based on the architecture for the autoencoder: FF-Weld (feedforward networks), Conv-Weld (convolutional networks), and PCA-WeldNet (Principal Component Analysis for linear dimension reduction). Note that PCA-WeldNet is analogous to the classic POD or linear model reduction methods.

We compare with Feedforward Neural Networks, LDNets in [31] and Weak-form Latent Space Dynamics Identification (WLaSDI) as described in [51]. We use a latent dimension of **4** for WeldNet and WLaSDI, and we use 1 fixed dimension and 3 dynamic dimensions for LDNet.

The numerical test error of all the trained models at various times can be found in Appendix D.

4.4.1. Burgers' Equation. We consider the viscous Burgers' equation with $\nu = 1/1000$ and with periodic boundary conditions for $t \in [0, 1]$.

$$u_t = \nu u_{xx} - u u_x; \quad u(x, 0) = g(x),\ x \in (0, 1). \tag{4.6}$$

We consider initial conditions generated by combining two complex base waves. Let w_0, w_1 be functions sampled from the Gaussian Random Field $N(0, 7^4(-\frac{d^2}{dx^2} + 7^2 I)^{-2.5})$ on $[0, 1)$. For any $a \in [-0.9, 0.9]$, consider the two sets of initial conditions:

$$\hat{g}_{\text{bscale}} = \{a w_0(x) + \sqrt{1 - a^2} w_1(x) : a \in [-0.9, 0.9]\}, \tag{4.7}$$

$$\hat{g}_{\text{bshift}} = \{0.5 w_0(x - h) + \sqrt{0.75} w_1(x - h) : h \in [0, 1]\}. \tag{4.8}$$

The trajectory manifolds $\mathcal{M}_{\text{bscale}}$ and $\mathcal{M}_{\text{bshift}}$ are obtained by collecting solutions from the PDE in (4.6) with initial conditions from $\hat{g}_{\text{bscale}}$ and $\hat{g}_{\text{bshift}}$ respectively. We collect 500 data trajectories with 301 time steps. We display the estimated intrinsic dimensionality of these datasets by Maximum Likelihood Estimation (MLE) [28] at various times and as a whole in Table 2 in Section 4.5 for details on this computation). The intrinsic dimension for each time is approximately 1, and the dimension of the datasets for all times is approximately 2, matching our expectation of 1 parameter dimension plus 1 time dimension.

We compare the performance of our WeldNet models (for $\mathsf{W} = 4$) with other models in Figures **??** and 11 in terms of final time Prediction error. We use a latent space dimension of **four** for this example. The x-axis is the parameter value for a or h (respectively), and the y-axis shows the prediction error at the final time for the initial condition corresponding to that parameter value. In Figures **??** and 11, FF-WeldNet significantly outperforms HDP, highlighting the advantages of model reduction: low-dimensional latent propagators are substantially easier to learn than high-dimensional

ones. FF-WeldNet also significantly outperforms PCA-WeldNet, which demonstrates the advantage of nonlinear dimension reduction. On this problem, FF-WeldNet and LDNet achieve the best result.

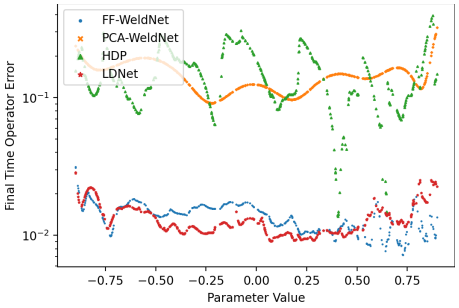

FIG. 10. *Error vs Parameter value for $\mathcal{M}_{bscale}$ about the trajectory manifold of the Burgers' equation* (4.6) *with initial conditions in* (4.7).

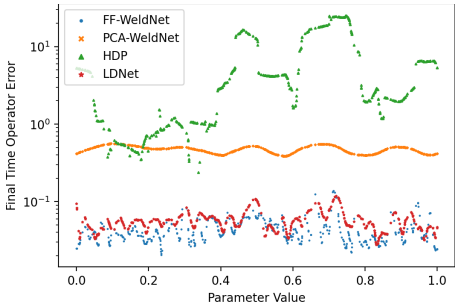

FIG. 11. *Error vs Parameter value for $\mathcal{M}_{bshift}$ about the trajectory manifold of the Burgers' equation* (4.6) *with initial conditions in* (4.8).

For both of the trajectory manifolds $\mathcal{M}_{\text{bscale}}$ and $\mathcal{M}_{\text{bshift}}$, we can compare the reconstruction error and the operator/prediction error at each time. Figures 12 and 13 are line plots of error versus time for FF-WeldNet, Conv-WeldNet, and PCA-WeldNet for $\mathcal{M}_{\text{bscale}}$ and $\mathcal{M}_{\text{bshift}}$, respectively. For this example, FF-WeldNet and Conv-WeldNet with latent dimension 4 outperforms PCA-WeldNet with latent dimension 4, since the solution manifold are nonlinear in these examples.

4.4.2. Transport Equation. We next consider the transport equation given by (4.4). We consider initial conditions given by $\hat{g}_{\text{tscale}}$ in (4.5) and

$$\hat{g}_{\text{tshift}}(x) = \{H_{1,0.1}(x) + H_{2.5,0.2+0.1h}(x) : h \in [0,3]\} \tag{4.9}$$

and collect the trajectory manifold $\mathcal{M}_{\text{tscale}}$ and $\mathcal{M}_{\text{tshift}}$ from the corresponding condition sets. We use a latent space dimension of **four** for this example. We compare the performance of four window WeldNet models with other models in terms of final time prediction error in Figures 14 and 15 respectively. WLaSDI has a very high relative operator error, so it does not appear in the figure to allow for clarity of scaling. The transport equation is a representative example of advection-dominated PDEs, which generates a highly nonlinear solution manifold. In Figures 14 and 15, FF-WeldNet with latent dimension 4 outperforms PCA-WeldNet with latent dimension 4, HDP and LDNet.

For both of the trajectory manifolds $\mathcal{M}_{\text{tscale}}$ and $\mathcal{M}_{\text{tshift}}$, we can compare the reconstruction error and the operator error at each time. Figures 16 and 17, are line plots of error versus time for FF-WeldNet, Conv-WeldNet, and PCA-WeldNet for $\mathcal{M}_{\text{tscale}}$ and $\mathcal{M}_{\text{tshift}}$, respectively. In Figures 16 and 17, Conv-WeldNet achieves the lowest error, followed by FF-WeldNet. Both models substantially outperform PCA-WeldNet for the same latent space dimension.

4.4.3. KdV Equation. We also consider the Korteweg–De Vries (KdV) equation for $T = 0.01$, given by

$$u_t = -u_{xxx} - uu_x, \quad u(x,0) = g(x),\ x \in (0,6). \tag{4.10}$$

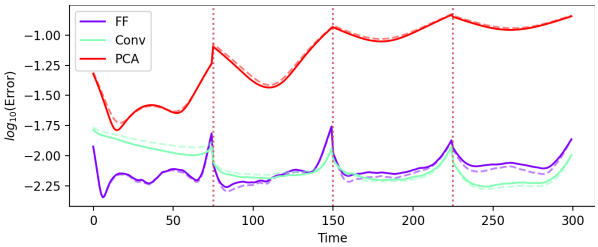

FIG. 12. *Autoencoder error (dashed) and Test error for* 4*-window WeldNet models for* $\mathcal{M}_{bscale}$*. FF refers to FF-WeldNet, Conv refers to Conv-WeldNet, and PCA refers to PCA-WeldNet. The latent space dimension is* 4*.*

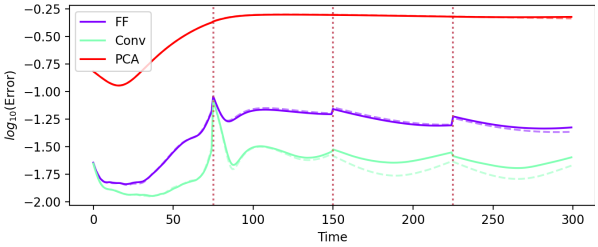

FIG. 13. *Autoencoder error (dashed) and Test error for* 4*-window WeldNet models for* $\mathcal{M}_{bshift}$*.*

We describe our sets of initial conditions. Given a constant c, we define the soliton of size c as $\phi_c(x) = c/2\text{sech}\left(\sqrt{c}x/2\right)^2$. We consider initial conditions:

$$\hat{g}_{\text{kscale}} = \{\phi_{a^2}(x-1) + \phi_{36}(x-2) : a \in [6, 18]\}, \tag{4.11}$$

$$\hat{g}_{\text{kshift}} = \{\phi_{36}(x-1) + \phi_{36}(x-2-h) : h \in [0, 0.4]\}, \tag{4.12}$$

and collect the trajectory manifold $\mathcal{M}_{\text{kscale}}$ and $\mathcal{M}_{\text{kshift}}$ from the corresponding condition sets. We compare the performance of four-window WeldNet models (with latent space dimension four) with other models in terms of final time operator error in Figure 18 and 19 respectively. WLaSDI has a very high relative operator error, so most of it is cut off from the figure to allow for clarity of scaling. For both examples, FF-WeldNet is the best model, outperforming PCA-WeldNet, HDP, and LDNet for almost all parameter values.

For both of the trajectory manifolds $\mathcal{M}_{\text{kscale}}$ and $\mathcal{M}_{\text{kshift}}$, we can compare the reconstruction error and the operator error at each time. Figures 20 and 21, are line plots of error versus time for FF-WeldNet, Conv-WeldNet, and PCA-WeldNet for $\mathcal{M}_{\text{kscale}}$ and $\mathcal{M}_{\text{kshift}}$, respectively. FF-WeldNet and Conv-WeldNet perform similarly on this problem, and they both greatly outperform PCA-WeldNet, showing the advantage of nonlinear dimension reduction.

4.4.4. Shallow-Water Equations. We now consider an example with two spatial dimensions. Consider the shallow-water equations over the spatial domain $[-2.5, 2.5]^2$ and time domain $[0, 1]$:

$$\partial_t h + \partial_x(hu) + \partial_y(hv) = 0, \tag{4.13}$$

$$\partial_t(hu) + \partial_x\left(u^2h + \tfrac{1}{2}g_r h^2\right) + \partial_y(uvh) = -g_r h\,\partial_x b, \tag{4.14}$$

$$\partial_t(hv) + \partial_y\left(v^2h + \tfrac{1}{2}g_r h^2\right) + \partial_x(uvh) = -g_r h\,\partial_y b, \tag{4.15}$$

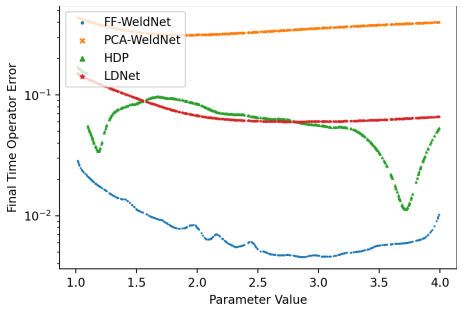

FIG. 14. *Error vs Parameter value for* $\mathcal{M}_{tscale}$ *about the trajectory manifold of the transport equation* (4.4) *with initial conditions in* (4.7).

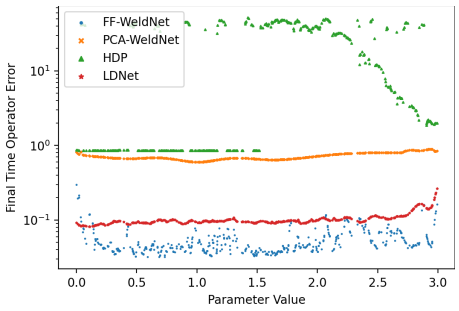

FIG. 15. *Error vs Parameter value for* $\mathcal{M}_{tshift}$ *about the trajectory manifold of the transport equation* (4.4) *with initial conditions in* (4.8).

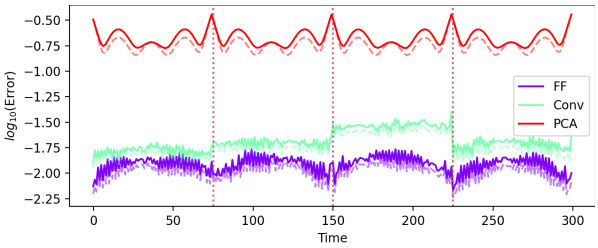

FIG. 16. *Autoencoder error (dashed) and Test error for* 4*-window WeldNet models for* $\mathcal{M}_{tscale}$.

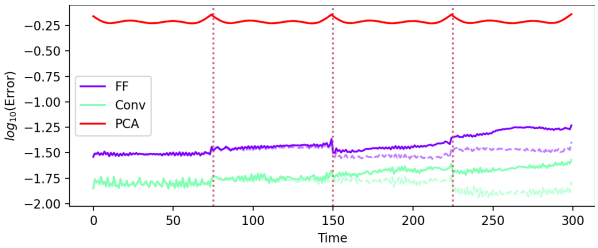

FIG. 17. *Autoencoder error (dashed) and Test error for* 4*-window WeldNet models for* $\mathcal{M}_{tshift}$.

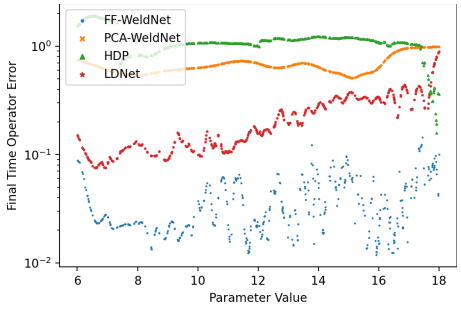

FIG. 18. *Error vs Parameter value for* $\mathcal{M}_{kscale}$ *about the trajectory manifold of the KdV equation* (4.10) *with initial conditions in* (4.11).

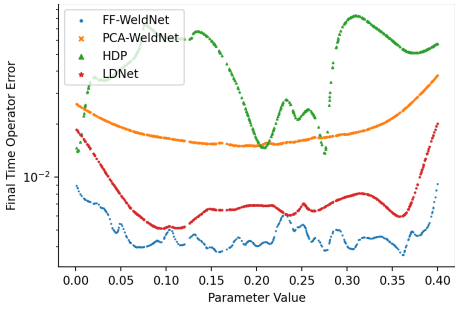

FIG. 19. *Error vs Parameter value for* $\mathcal{M}_{kshift}$ *about the trajectory manifold of the KdV equation* (4.10) *with initial conditions in* (4.12).

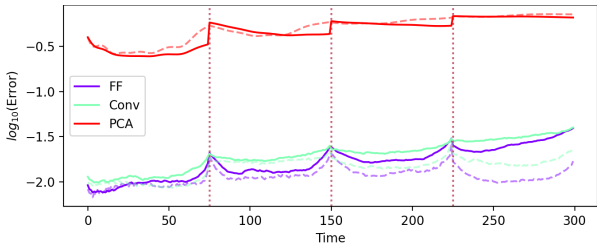

FIG. 20. *Autoencoder error (dashed) and Test error for WeldNet models for* $\mathcal{M}_{kscale}$.

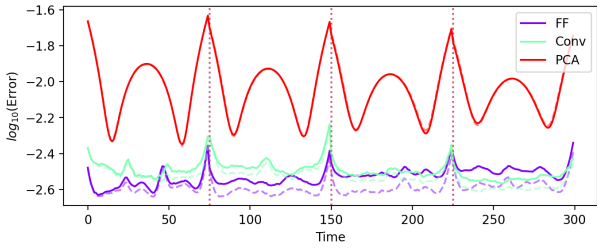

FIG. 21. *Autoencoder error (dashed) and Test error for WeldNet models for* $\mathcal{M}_{kshift}$.

where u is horizontal velocity, v is vertical velocity, h is the water depth, g_r is the gravitational acceleration, and b is a scalar field known as bathymetry. We impose zero dirichlet boundary conditions for u and v.

We consider the following set of initial conditions representing centered bumps of varying radii (provided by the PDEBench dataset [48, 47]):

$$\hat{g}_{\text{shallow}} = \left\{ h(x,y) = 1 + \mathbb{1}[\sqrt{x^2+y^2} < r] : r \in [0.3, 0.7] \right\} \tag{4.16}$$

and collect the trajectory manifold $\mathcal{M}_{\text{shallow}}$. Note that the initial condition is discontinuous, and the location of the discontinuity depends on the sample. For this example only, we use a 128×128 grid for each sample, and we use 101 time steps. We collect 81 examples.

Note that the inputs are sampled on a size 16384 grid. This makes the size of feedforward networks prohibitively high for training. While more sophisticated architectures such as two-dimensional convolutions exist, we will instead reduce the dimension from 16384 to 128 using PCA, and then train our models to predict the evolution of the PCA modes. We compute the top 128 principal components (over all times) using the training data only, and this results in a relative projection error of 0.11%.

Now we have 81 samples with 101 time steps and 128 features each. We train each of our models on this dataset. Even though the compression to 128 features is very accurate, it can be challenging to learn the dynamics. We compare the performance of the four-window WeldNet model (with latent space dimension of 4) with other models in terms of final time operator error in Figure 22. WeldNet greatly outperforms all other models by almost an entire order of magnitude.

The lower error for WeldNet stems from the fact that the trajectory manifold is harder to represent at earlier times than later times, and WeldNet is able to separate the latent dynamics learning in each of its segments. The transcoder is able to lower the error as a code is evolved between windows. On the other hand, there is no

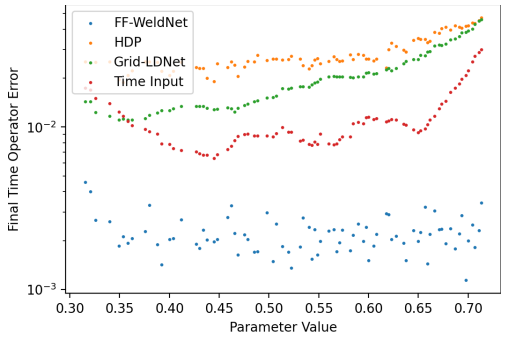

FIG. 22. *Error vs Parameter value for* $\mathcal{M}_{shallow}$.

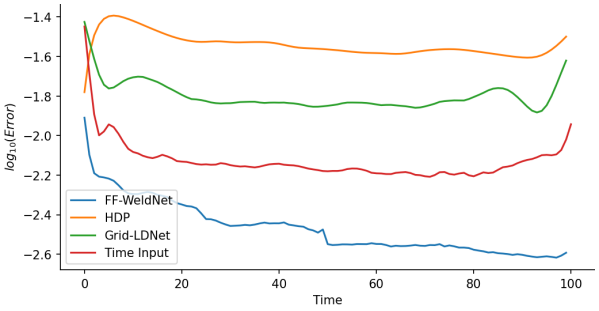

FIG. 23. *Error vs Parameter value for* $\mathcal{M}_{shallow}$.

error reduction mechanism in the other dynamics learning models, so the error only increases with time. We can see this clearly in Figure 23, a plot of the operator error versus time for various models. This plot shows that FF-WeldNet has improved performance for later times, while the error stagnates or increases with time on the other models. This shows the advantage of having multiple windows; a decoupled representation of this trajectory manifold outperforms a fully coupled one.

4.5. Intrinsic Dimension Estimation. We estimate the intrinsic dimensionality of our datasets using the Scikit-dimension [1] Python package and display it in Table 2. Specifically, we use the MLE algorithm [28] with the "Haro" integral approximation [20] and the TwoNN algorithm [14] with the parameter discard_fraction= 0. We use a random size 50 000 subset for computing the intrinsic dimension of all times together. By construction, the initial conditions for all considered datasets are generated by sampling one random scalar, so we expect the intrinsic dimensionality of each time slice to be approximately 1 and the intrinsic dimensionality of all times together to be approximately 2. This justifies the use of dimension reduction methods for these datasets and indicates that a small latent space dimension can effectively capture the data.

For the shallow water dataset, the relatively high intrinsic dimensionality for the initial time and all times together likely comes from the discontinuity of the initial condition in (4.16). The estimated intrinsic dimensionality for a continuous approximation to the initial conditions can be computed, and we empirically found that it was about 1 regardless of the specific construction used (as long as the functions are continuous).

Dataset	t=0	t=60	t=120	t=180	t=240	t=300	All t
bscale	0.973	0.974	0.974	0.976	0.978	0.977	1.89
bshift	0.998	1.019	1.128	1.091	1.065	1.048	1.685
tscale	0.976	0.976	0.976	0.976	0.976	0.976	1.118
tshift	1.013	1.013	1.012	1.013	1.013	1.012	1.973
kscale	0.945	0.945	0.945	0.945	0.945	0.945	1.803
kshift	0.995	0.995	0.995	0.995	0.995	0.995	1.901
shallow	2.184	1.282	1.257	1.264	1.244	1.231	2.593

TABLE 2

Estimated intrinsic dimensionality of all datasets for various times and as a whole. MLE is used for all datasets except for kscale, for which TwoNN was used (see Appendix 4.5 for details).

4.6. Propagator Ablation Study. WeldNet first trains the autoencoder and propagator together using the propagator displacement loss in (2.2) (where the name displacement comes from being one step of applying propagator) and then finetunes the propagator using the accumulation loss in (2.4). Alternatively, we could train the autoencoder and propagator separately, or we could use only a single loss function for the propagator. We compare our Weldnet model training algorithm against three variants.

(i) "WeldNet" (original) - train autoencoder and propagator together with displacement loss, then finetune the propagator with accumulation loss.
(ii) "Together / Displacement" - same as (i) but finetune with displacement loss.
(iii) "Separate / Accumulate" - train autoencoder first, then train propagator with accumulation loss.
(iv) "Separate / Displacement" - same as (iii) but with the displacement loss.

We train autoencoder, propagator, and transcoder models using each of the above four training algorithms. For algorithms (i) and (ii), we jointly train the autoencoder and propagator for 300 epochs and then finetune the propagator for 150 epochs. For algorithms (iii) and (iv), we train the autoencoder for 300 epochs and then the propagator for 450 epochs. We train any transcoders for 300 epochs. All other settings are identical to the main paper. We use the $kshift$ dataset (for the KdV equation) described in Section 4.4.3.

Figures 24 and 25 compare the relative test error of one-window and two-window models (respectively) trained with each algorithm. The final time test error for each one-window model is (in order): 0.87%, 1.28%, 1.31%, 20.4%, and the average test error for each two-window model is (in order): 0.42%, 0.45%, 0.50%, 1.3%. While increasing the number of windows reduces the gap between the models, the best model is (i) which is the training algorithm used in the main paper.

5. Conclusion. This paper introduces WeldNet, a data-driven framework for nonlinear model reduction that constructs low-dimensional surrogate models for complex evolutionary systems. The architecture consists of three key components: autoencoders, propagators, and transcoders. These components operate together to model dynamics in reduced latent spaces. By transferring the evolution of high-dimensional systems into these latent spaces and decomposing the time domain into overlapping windows, WeldNet effectively captures intricate nonlinear structures while simplifying long-time propagation into a sequence of short, tractable segments.

In addition to the algorithmic design, a representation theory is developed to establish the approximation capability of WeldNet under the manifold hypothesis,

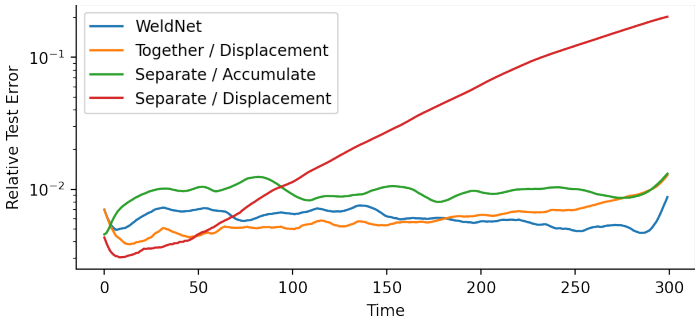

FIG. 24. *Test error vs time for variant training algorithms on $\mathcal{M}_{kshift}$ (1 window models).*

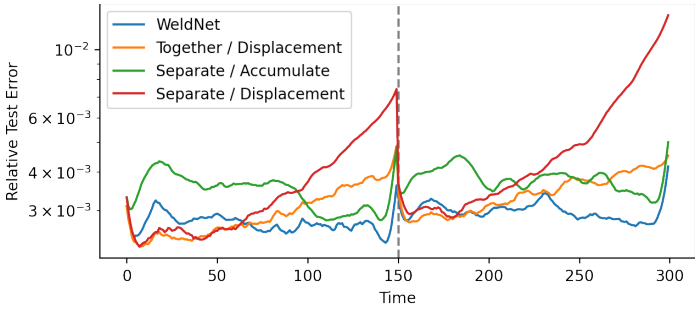

FIG. 25. *Test error vs time for variant training algorithms on $\mathcal{M}_{kshift}$ (2 window models).*

thereby providing a mathematical foundation for nonlinear model reduction via deep learning. Extensive numerical experiments on a variety of differential equations demonstrate the robustness and accuracy of the proposed approach. Across all tested scenarios, WeldNet consistently achieves smaller prediction errors than both classical projection-based techniques and recently developed nonlinear reduced-order models. In addition, WeldNet is more computationally efficient to train than other methods.

Overall, this work shows that windowed latent-space learning offers a powerful and principled strategy for modeling complex dynamical systems, and it opens promising ways for future research in nonlinear model reduction in scientific machine learning.

REFERENCES

[1] J. Bac, E. M. Mirkes, A. N. Gorban, I. Tyukin, and A. Zinovyev, *Scikit-dimension: a python package for intrinsic dimension estimation*, Entropy, 23 (2021), p. 1368.
[2] M. Bachmayr and A. Cohen, *Kolmogorov widths and low-rank approximations of parametric elliptic pdes*, Mathematics of Computation, 86 (2017), pp. 701–724.
[3] U. Baur, C. Beattie, P. Benner, and S. Gugercin, *Interpolatory projection methods for parameterized model reduction*, SIAM Journal on Scientific Computing, 33 (2011), pp. 2489–2518.
[4] M. Belkin and P. Niyogi, *Laplacian eigenmaps for dimensionality reduction and data representation*, Neural computation, 15 (2003), pp. 1373–1396.
[5] P. Benner, S. Gugercin, and K. Willcox, *A survey of projection-based model reduction methods for parametric dynamical systems*, SIAM review, 57 (2015), pp. 483–531.
[6] P. Benner, M. Ohlberger, A. Cohen, and K. Willcox, *Model reduction and approximation: theory and algorithms*, SIAM, 2017.
[7] G. Berkooz, P. Holmes, and J. L. Lumley, *The proper orthogonal decomposition in the analysis of turbulent flows*, Annual review of fluid mechanics, 25 (1993), pp. 539–575.

[8] P. Binev, A. Cohen, W. Dahmen, R. DeVore, G. Petrova, and P. Wojtaszczyk, *Convergence rates for greedy algorithms in reduced basis methods*, SIAM journal on mathematical analysis, 43 (2011), pp. 1457–1472.
[9] S. L. Brunton and J. N. Kutz, *Data-driven science and engineering: Machine learning, dynamical systems, and control*, Cambridge University Press, 2022.
[10] M. Chen, H. Jiang, W. Liao, and T. Zhao, *Efficient approximation of deep relu networks for functions on low dimensional manifolds*, Advances in neural information processing systems, 32 (2019), pp. 8174–8184.
[11] R. R. Coifman, I. G. Kevrekidis, S. Lafon, M. Maggioni, and B. Nadler, *Diffusion maps, reduction coordinates, and low dimensional representation of stochastic systems*, Multiscale Modeling & Simulation, 7 (2008), pp. 842–864.
[12] M. Crosskey and M. Maggioni, *Atlas: a geometric approach to learning high-dimensional stochastic systems near manifolds*, Multiscale Modeling & Simulation, 15 (2017), pp. 110–156.
[13] T. A. Driscoll, N. Hale, and L. N. Trefethen, *Chebfun guide*, 2014.
[14] E. Facco, M. d'Errico, A. Rodriguez, and A. Laio, *Estimating the intrinsic dimension of datasets by a minimal neighborhood information*, Scientific reports, 7 (2017), p. 12140.
[15] H. Federer, *Curvature measures*, Transactions of the American Mathematical Society, 93 (1959), pp. 418–491.
[16] N. Franco, A. Manzoni, and P. Zunino, *A deep learning approach to reduced order modelling of parameter dependent partial differential equations*, Mathematics of Computation, 92 (2023), pp. 483–524.
[17] S. Fresca, L. Dede, and A. Manzoni, *A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes*, Journal of Scientific Computing, 87 (2021), pp. 1–36.
[18] F. J. Gonzalez and M. Balajewicz, *Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems*, arXiv preprint arXiv:1808.01346, (2018).
[19] S. Gugercin, A. C. Antoulas, and C. Beattie, *H_2 model reduction for large-scale linear dynamical systems*, SIAM journal on matrix analysis and applications, 30 (2008), pp. 609–638.
[20] G. Haro, G. Randall, and G. Sapiro, *Translated poisson mixture model for stratification learning*, International Journal of Computer Vision, 80 (2008), pp. 358–374.
[21] J. S. Hesthaven, G. Rozza, B. Stamm, et al., *Certified reduced basis methods for parametrized partial differential equations*, vol. 590, Springer, 2016.
[22] J. S. Hesthaven and S. Ubbiali, *Non-intrusive reduced order modeling of nonlinear problems using neural networks*, Journal of Computational Physics, 363 (2018), pp. 55–78.
[23] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley, *Turbulence, coherent structures, dynamical systems and symmetry*, Cambridge university press, 2012.
[24] K. Kontolati, S. Goswami, G. Em Karniadakis, and M. D. Shields, *Learning nonlinear operators in latent spaces for real-time predictions of complex dynamics in physical systems*, Nature Communications, 15 (2024), p. 5101.
[25] J. M. Lee, *Introduction to Riemannian manifolds*, vol. 2, Springer, 2018.
[26] K. Lee and K. T. Carlberg, *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*, Journal of Computational Physics, 404 (2020), p. 108973.
[27] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, SIAM, 2007.
[28] E. Levina and P. Bickel, *Maximum likelihood estimation of intrinsic dimension*, Advances in neural information processing systems, 17 (2004).
[29] H. Liu, B. Dahal, R. Lai, and W. Liao, *Generalization error guaranteed auto-encoder-based nonlinear model reduction for operator learning*, Applied and Computational Harmonic Analysis, 74 (2025), p. 101717.
[30] H. Liu, A. Havrilla, R. Lai, and W. Liao, *Deep nonparametric estimation of intrinsic data structures by chart autoencoders: Generalization error and robustness*, Applied and Computational Harmonic Analysis, 68 (2024), p. 101602.
[31] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via deeponet based on the universal approximation theorem of operators*, Nature Machine Intelligence, 3 (2021), pp. 218–229.
[32] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis, *Learning nonlinear operators via deeponet based on the universal approximation theorem of operators*, Nature Machine Intelligence, 3 (2021), pp. 218–229, https://doi.org/10.1038/s42256-021-00302-5.

826 [33] B. Moore, *Principal component analysis in linear systems: Controllability, observability, and*
827 *model reduction*, IEEE transactions on automatic control, 26 (1981), pp. 17–32.
828 [34] M. Ohlberger and S. Rave, *Reduced basis methods: Success, limitations and future chal-*
829 *lenges*, arXiv preprint arXiv:1511.02021, (2015).
830 [35] K. Oono and T. Suzuki, *Approximation and non-parametric estimation of resnet-type con-*
831 *volutional neural networks*, in International conference on machine learning, PMLR, 2019,
832 pp. 4922–4931.
833 [36] S. E. Otto and C. W. Rowley, *Linearly recurrent autoencoder networks for learning dynam-*
834 *ics*, SIAM Journal on Applied Dynamical Systems, 18 (2019), pp. 558–593.
835 [37] T. O'Leary-Roseberry, U. Villa, P. Chen, and O. Ghattas, *Derivative-informed projected*
836 *neural networks for high-dimensional parametric maps governed by pdes*, Computer Meth-
837 ods in Applied Mechanics and Engineering, 388 (2022), p. 114199.
838 [38] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin,
839 N. Gimelshein, L. Antiga, et al., *Pytorch: An imperative style, high-performance deep*
840 *learning library*, Advances in neural information processing systems, 32 (2019).
841 [39] A. Pinkus, *N-widths in Approximation Theory*, vol. 7, Springer Science & Business Media,
842 2012.
843 [40] R. Pinnau, *Model reduction via proper orthogonal decomposition*, Model order reduction: the-
844 ory, research aspects and applications, (2008), pp. 95–109.
845 [41] C. Prud'Homme, D. V. Rovas, K. Veroy, L. Machiels, Y. Maday, A. T. Patera, and
846 G. Turinici, *Reliable real-time solution of parametrized partial differential equations:*
847 *Reduced-basis output bound methods*, J. Fluids Eng., 124 (2002), pp. 70–80.
848 [42] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced basis methods for partial differential*
849 *equations: an introduction*, vol. 92, Springer, 2015.
850 [43] F. Regazzoni, S. Pagani, M. Salvador, L. Dede', and A. Quarteroni, *Learning the in-*
851 *trinsic dynamics of spatio-temporal processes through latent dynamics networks*, Nature
852 Communications, 15 (2024), p. 1834.
853 [44] S. T. Roweis and L. K. Saul, *Nonlinear dimensionality reduction by locally linear embedding*,
854 science, 290 (2000), pp. 2323–2326.
855 [45] C. W. Rowley, T. Colonius, and R. M. Murray, *Model reduction for compressible flows*
856 *using pod and galerkin projection*, Physica D: Nonlinear Phenomena, 189 (2004), pp. 115–
857 129.
858 [46] G. Rozza, D. B. P. Huynh, and A. T. Patera, *Reduced basis approximation and a posteriori*
859 *error estimation for affinely parametrized elliptic coercive partial differential equations:*
860 *application to transport and continuum mechanics*, Archives of Computational Methods in
861 Engineering, 15 (2008), p. 229.
862 [47] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and
863 M. Niepert, *Pdebench: An extensive benchmark for scientific machine learning*, Advances
864 in Neural Information Processing Systems, 35 (2022), pp. 1596–1611.
865 [48] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, and
866 M. Niepert, *PDEBench Datasets*, 2022, https://doi.org/10.18419/DARUS-2986, https:
867 //doi.org/10.18419/DARUS-2986.
868 [49] J. B. Tenenbaum, V. d. Silva, and J. C. Langford, *A global geometric framework for*
869 *nonlinear dimensionality reduction*, science, 290 (2000), pp. 2319–2323.
870 [50] C. Thäle, *50 years sets with positive reach–a survey.*, Surveys in Mathematics and its Appli-
871 cations, 3 (2008), pp. 123–165.
872 [51] A. Tran, X. He, D. A. Messenger, Y. Choi, and D. M. Bortz, *Weak-form latent space*
873 *dynamics identification*, Computer Methods in Applied Mechanics and Engineering, 427
874 (2024), p. 116998.
875 [52] S. Volkwein, *Model reduction using proper orthogonal decomposition*, Lecture Notes, Institute
876 of Mathematics and Scientific Computing, University of Graz., 1025 (2011).
877 [53] Q. Wang, J. S. Hesthaven, and D. Ray, *Non-intrusive reduced order modeling of unsteady*
878 *flows using artificial neural networks with application to a combustion problem*, Journal of
879 computational physics, 384 (2019), pp. 289–307.
880 [54] Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, and Y. Guo, *Model identification*
881 *of reduced order fluid dynamics systems using deep learning*, International Journal for
882 Numerical Methods in Fluids, 86 (2018), pp. 255–268.
883 [55] F. X.-F. Ye, S. Yang, and M. Maggioni, *Nonlinear model reduction for slow–fast stochastic*
884 *systems near unknown invariant manifolds*, Journal of Nonlinear Science, 34 (2024), p. 22.
885 [56] K. Zeng, C. E. P. De Jesús, A. J. Fox, and M. D. Graham, *Autoencoders for discovering*
886 *manifold dimension and coordinates in data from complex dynamical systems*, Machine
887 Learning: Science and Technology, 5 (2024), p. 025053.

Appendix A. Helper Approximation Results. In this section, we introduce some approximation results for ReLU networks. They are later used to prove our main results.

First, we establish that ReLU networks (with a given size) can implement the identity function.

PROPOSITION A.1. *Let $d, L, W \in \mathbb{N}$, with $W \geq 2d$. Then there is a $f_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d, d, L, W)$ such that $f_{\text{NN}}(\mathbf{x}) = \mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$.*

Proof of Proposition A.1. We denote by I_d the identity function on $\mathbb{R}^d$. First, suppose $L = 1$ and $W = 2d$. Consider the weights and biases $\mathfrak{W}_1 = \begin{bmatrix} I_d \\ -I_d \end{bmatrix}$, $\mathfrak{b}_1 = \vec{0}_{2d}$, $\mathfrak{W}_2 = \begin{bmatrix} I_d & -I_d \end{bmatrix}$, and $\mathfrak{b}_2 = \vec{0}_d$. Then the one-hidden-layer ReLU network with the above weights and biases is in $\mathcal{F}_{\text{NN}}(d, d, 1, 2d)$ and exactly implements the identity (since $\sigma(x) - \sigma(-x) = x$ for $\sigma(x) = \max\{x, 0\}$).

Next, if $L > 1$ and $W = 2d$, define $\mathfrak{W}_1 = \begin{bmatrix} I_d \\ -I_d \end{bmatrix}$ and $\mathfrak{W}_{L+1} = \begin{bmatrix} I_d & -I_d \end{bmatrix}$. Also for any $1 < \ell < L+1$, let $\mathfrak{W}_\ell = I_{2d}$. Let all bias vectors be zero. Then the L-hidden-layer ReLU network with the above weights and biases is in $\mathcal{F}_{\text{NN}}(d, d, L, 2d)$ and exactly implements the identity (as the output of the Lth hidden layer is the same as the output of the first hidden layer).

Finally, suppose $W > 2d$. It is easy to see that augmenting weight matrices $\mathfrak{W}_1, \cdots, \mathfrak{W}_L$ defined above with zeros in all new rows and columns (with the original weight matrix being in the top left corner) does not affect output of the neural network. Thus we have constructed an L-hidden-layer ReLU network with width exactly W that implements the identity in $\mathbb{R}^d$. ∎

PROPOSITION A.2. *Let $f_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, L, W)$ be a feedforward ReLU network. Suppose $W > d_{out}$.*

1. *For any $L' > L$, there exists a FNN $g_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, L', 2W)$ with depth exactly L' such that $g_{\text{NN}}(\mathbf{x}) = f_{\text{NN}}(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^{d_1}$.*
2. *Let $A = \prod_{i=1}^{d_{out}} [a_i, b_i]$ be a cube. Then there is a $h_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, L + 2, \max\{d_{out}, W\})$ such that for all $\mathbf{x} \in \mathbb{R}^{d_{in}}$, $h_{\text{NN}}(\mathbf{x})$ is vector valued and $= (h_{\text{NN}}^{(1)}(\mathbf{x}), \ldots, h_{\text{NN}}^{(d_{out})}(\mathbf{x})) \in \mathbb{R}^{d_{out}}$ with $h_{\text{NN}}^{(i)}(\mathbf{x}) = \min\{\max\{f_{\text{NN}}^{(i)}(\mathbf{x}), a_i\}, b_i\}$.*
3. *Let $f_{\text{NN}}^1, \ldots, f_{\text{NN}}^m$ be FNNs such that $f_{\text{NN}}^j \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, L_j, W_j)$ for all $j \in [m]$. Then there is a $g_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, \sum_{j=1}^m L_j, 2d_{in} + 2d_{out} + \max_j W_j)$ such that for all $\mathbf{x} \in \mathbb{R}^{d_{in}}$, we have $g_{\text{NN}}(\mathbf{x}) = \sum_{j=1}^m f_{\text{NN}}^j(\mathbf{x})$.*

Proposition A.2 gives rise to basic properties of feedforward ReLU networks. The first property is that, any neural network can be realized by a deeper one; The second property says that, feedforward ReLU networks allows one to truncate or restrict its output to be in the set of $A = \prod_{i=1}^{d_2} [a_i, b_i]$; The third property allows one to concatenate several feedforward ReLU networks and then output its summation.

Proof of Proposition A.2. **Part 1**: Denote the number of neurons in the last hidden layer of f_{NN} by r (note $r \leq W$). Let L_f be the number of layers of f_{NN}, and by the hypothesis we have $L_f < L'$.

First, consider the function $\phi_1 : \mathbb{R}^r \to \mathbb{R}^{2r}$ defined by $\phi_1(x_1, \ldots, x_r) = [\sigma(x_1), \sigma(-x_1), \ldots, \sigma(x_r), \sigma(-x_r)]$. Clearly, ϕ_1 can be implemented by a single ReLU layer. Next, consider the function ϕ_2 on $\mathbb{R}^{2r}$ defined by $\phi_2(x_1, \ldots, x_{2r}) = [\sigma(x_1), \ldots, \sigma(x_{2r})]$. Clearly, ϕ_2 can also be implemented by a single ReLU layer. Note that $\bigcirc_{i=1}^n \phi_2 = \phi_2$ for any $n \in \mathbb{N}$. We will construct g_{NN} from f_{NN} by appending a hidden layer

implementing ϕ_1, appending $L' - L_f - 1$ hidden layers implementing ϕ_2, and then adjusting the weights in the final layer of f_{NN}.

Denote the final weight matrix of f_{NN} by $\mathfrak{W} \in \mathbb{R}^{r \times d_{out}}$. We will adjust this matrix to handle inputs of size $2r$. Define the matrix $\hat{\mathfrak{W}} \in \mathbb{R}^{2r \times d_{out}}$ by $\hat{\mathfrak{W}}_{i,j} = (-1)^{j+1}\mathfrak{W}_{i,\lceil j/2 \rceil}$, for all $i \in [d_{out}], j \in [2r]$. This means $\hat{\mathfrak{W}}(x_1, x_2, \ldots, x_{2r-1}, x_{2r}) = \mathfrak{W}(x_1 - x_2, \ldots, x_{2r-1} - x_{2r})$.

Note that for all $i \in [r]$, we have that $\sigma(x_i) - \sigma(-x_i) = x_i$. This means (where we use denote multiplication vector multiplication by parenthesis)

$$\hat{\mathfrak{W}}\left((\bigcirc_{i=1}^{L'-L_f-1}\phi_2) \circ \phi_1(x_1, \ldots, x_r)\right)$$
$$= \hat{\mathfrak{W}}(\sigma(x_1), \sigma(-x_1), \ldots, \sigma(x_r), \sigma(-x_r)) = \mathfrak{W}(x_1 \ldots, x_r).$$

Finally, we can describe g_{NN}. Before the final weight matrix of f_{NN}, we will append one layer implementing ϕ_1 and $L' - L_f - 1$ layers implementing ϕ_2 to the network. Then, we change the final weight matrix to $\hat{\mathfrak{W}}$. Thus, g_{NN} and f_{NN} implement the same function, and $g_{\text{NN}} \in \mathcal{F}(d_{in}, d_{out}, L', 2W)$.

Part 2: Note that the function $b - \sigma(b - a - \sigma(x - a)) = \min(\max(x, a), b)$.

Let $\mathfrak{b} \in \mathbb{R}^{d_{out}}$ be the bias vector for the final layer. Define a new bias vector $\tilde{\mathfrak{b}} \in \mathbb{R}^{d_{out}}$ by $\tilde{\mathfrak{b}}_i = \mathfrak{b}_i - a_i$, for all $i \in [d_{out}]$.

If we denote $\mathbf{a} = [a_1, \ldots, a_{d_{out}}]$ and $\mathbf{b} = [b_1, \ldots, b_{d_{out}}]$, then note that for all $x \in \mathbb{R}^{d_{in}}$,

$$-\sigma(-(\sigma(f_{\text{NN}}(x) - \mathbf{a}) + \mathbf{b} - \mathbf{a}) + \mathbf{b} = \min(\max(f_{\text{NN}}(x), \mathbf{a}), \mathbf{b}), \tag{A.1}$$

where σ and $\max, \min$ are applied componentwise. We can implement the LHS of Equation (A.1) starting from f_{NN} using the following steps (where $I_{d_{out}}$ is the identity matrix for $\mathbb{R}^{d_{out}}$):

1. Modify the last bias vector of f_{NN} to $\tilde{\mathfrak{b}}$. This implements $f_{\text{NN}}(x) - \mathbf{a}$.
2. Add a new layer with weight matrix equal to $-I_{d_{out}}$ and bias vector equal to $\mathbf{b} - \mathbf{a}$. This implements $-\sigma(f_{\text{NN}}(x) - \mathbf{a}) + \mathbf{b} - \mathbf{a}$.
3. Add a new layer with weight matrix equal to $-I_{d_{out}}$ (identity) and bias equal to $\mathbf{b}$. This implements the LHS of (A.1). ∎

Thus we have constructed a neural network h_{NN} by modifying the bias vector in the final layer of f_{NN} and adding two more layers.

Part 3: We add extra layers that store the running sum. For every network f^j_{NN}, we add $2d_{in}$ extra neurons that store the positive and negative values of the inputs every layer. For the final layer of each network, we store the positive and negative values of the output with $2d_{out}$ neurons.

We then concatenate each of these networks. We collect the running sum in the $2d_{out}$ neurons, and we preserve the input in the $2d_{in}$ extra intermediate neurons. The depth of this concatenated network is the sum of the depths of each network, and the width is the maximum width of the networks plus $2d_{out} + 2d_{in}$ neurons.

Each layer will need $2d_{out}$ neurons to capture the positive and negative values of the d_{out} outputs in the sum. Specifically, we modify the final layer of f_{NN} to have double the number of output neurons as d_{out}

Now, let's consider to approximate Hölder functions using ReLU networks. For

any function $f : \Omega \to \mathbb{R}$, we define it 1-Hölder norm:

$$\|f\|_{\mathcal{H}^1(\Omega;\mathbb{R})} = \max\left\{\sup_{x\in\Omega}|f(x)|,\ \sup_{x,y\in\Omega}\frac{|f(x)-f(y)|}{\|x-y\|_2}\right\}$$

and

$$\mathcal{H}^1(\Omega;\mathbb{R}) = \left\{f : \|f\|_{\mathcal{H}^1(\Omega;R)} < \infty\right\}.$$

PROPOSITION A.3. *Let $f^* \in \mathcal{H}^1([0,1]^d;\mathbb{R})$. Then, $\forall \mathsf{M} \in \mathbb{N}$, there is a $f_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d,1,L,W)$ such that*

$$\|f^* - f_{\mathrm{NN}}\|_{L^\infty([0,1]^d;\mathbb{R})} \le C\mathsf{M}^{-\frac{1}{d}}.$$

Here $L = O(\log(\mathsf{M}))$ and $W = O(\mathsf{M})$, where the big O and C hides constants only depending on d and $\|f^*\|_{\mathcal{H}^1([0,1]^d;\mathbb{R})}$.

Proof. This is a restatement of [35, Lemma 7]. ∎

We extend Proposition A.3 to functions with vector-valued outputs.

PROPOSITION A.4. *Let $f^* \in \mathcal{H}^1([0,1]^{d_{in}}; A)$, where $A = \mathbb{R}^{d_{out}}$ is the Euclidean space, or $A = \prod_{i=1}^{d_{out}}[a_i,b_i]$ is a cube. Then, $\forall \mathsf{M} \in \mathbb{N}$, $\exists f_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d_{in},d_{out},L,W)$ such that*

$$\|f^* - f_{\mathrm{NN}}\|_{L^\infty([0,1]^{d_{in}};\mathbb{R}^{d_{out}})} \le C\mathsf{M}^{-\frac{1}{d_{in}}}.$$

In addition, the range of f_{NN} is contained in A. Here $L = O(\log(\mathsf{M}))$ and $W = O(\mathsf{M})$, where big O and C hide constants depending on d_{in}, d_{out}, and $\|f^*\|_{\mathcal{H}^1([0,1]^{d_{in}};\mathbb{R}^{d_{out}})}$.

Proof of Proposition A.4. For all $i \in [d_{out}]$, we denote the ith component function of f^* by f^i. Then since $f^i \in \mathcal{H}^1([0,1]^{d_{in}},\mathbb{R})$, by Proposition A.3 there exists an FNN $f^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d_{in},1,L_i,W_i)$ such that $\|f^i - f^i_{\mathrm{NN}}\|_{L^\infty([0,1]^{d_{in}};\mathbb{R})} \le C_i\mathsf{M}^{-\frac{1}{d_{in}}}$ for some C_i depending on d and $\|f^i\|_{\mathcal{H}^1([0,1]^{d_{in}};\mathbb{R})}$. Let $L = \max_i L_i$, and $W = \sum_{i=1}^{d_{out}} W_i$. We will add extra layers implementing the identity so that each f^i_{NN} has exactly L layers. This is done by applying Proposition A.2 Part 1, which will double the width of each f^i_{NN}, leaving the term inside the big O notation unchanged.

Now, define f_{NN} as the network implementing each of the f^i_{NN} as component functions. Then $f_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d_{in},d_{out},L,W)$ with $L = O(\log(\mathsf{M}))$ and $W = O(\mathsf{M})$. For all $x \in [0,1]^{d_{in}}$ we estimate

$$\begin{aligned}
\|f^*(x) - f_{\mathrm{NN}}(x)\|_{\mathbb{R}^{d_{out}}} &= \left(\sum_{i=1}^{d_{out}}(f^*_i(x) - f^i_{\mathrm{NN}}(x))^2\right)^{\frac{1}{2}} \\
&\le \left(\sum_{i=1}^{d_{out}} C_i^2\mathsf{M}^{-\frac{2}{d_{in}}}\right)^{\frac{1}{2}} \\
&\le \left(d_{out}\left(\max_i C_i^2\right)\mathsf{M}^{-\frac{2}{d_{in}}}\right)^{\frac{1}{2}} \\
&= \sqrt{d_{out}\left(\max_i C_i^2\right)}\mathsf{M}^{-\frac{1}{d_{in}}}.
\end{aligned}$$

Finally, if $A = \prod_{i=1}^{d_{out}}[a_i,b_i]$, we replace each f^i_{NN} by $\sigma(-\sigma(-f^i_{\mathrm{NN}} + b_i - a_i) + a_i)$ so that for any $x \in [0,1]^{d_{in}}$, $f^i_{\mathrm{NN}}(x) \in [a_i,b_i]$ and thus $f_{\mathrm{NN}}(x) \in A$. ∎

We use Proposition A.4 to approximate target functions up to ϵ accuracy. Setting the approximation error as ϵ in Proposition A.4 gives rise to the following proposition.

PROPOSITION A.5. *Let $\epsilon > 0$ and $f^* \in \mathcal{H}^1([0,1]^{d_{in}}; A)$, where $A = \mathbb{R}^{d_{out}}$ is Euclidean space or $A = \prod_{i=1}^{d_{out}}[a_i, b_i]$ is a cube. Then $\exists f_{\text{NN}} \in \mathcal{F}_{\text{NN}}(d_{in}, d_{out}, L, W)$ such that*

$$\|f^* - f_{\text{NN}}\|_{L^\infty([0,1]^{d_{in}}; \mathbb{R}^{d_{out}})} < \epsilon.$$

In addition, the range of f_{NN} is contained in A. Here $L = O(\log(\epsilon^{-1}))$ and $W = O(\epsilon^{-d_{in}})$, where the big O hides constants depending only on d_{in}, d_{out}, and $\|f^*\|_{\mathcal{H}^1([0,1]^{d_{in}}; \mathbb{R}^{d_{out}})}$.

Proof of Proposition A.5. This follows directly from Proposition A.4. ∎

WeldNet uses a composition of neural networks (i.e. propagators and transcoders) that are sequentially trained. In order to control the approximation error of such a composition, we use the following proposition.

PROPOSITION A.6 (Composition). *Let $f_1, \dots, f_n$ be Lipschitz functions, such that $f_j : A_{j-1} \to A_j$ for all $j \in [n]$. Suppose for each $j \in [n]$ and $\epsilon_j > 0$, there is an approximating function $\hat{f}_j : A_{j-1} \to A_j$ such that*

$$\sup_{x \in A_{j-1}} \|f_j(x) - \hat{f}_j(x)\| < \epsilon_j.$$

Then

$$\sup_{x \in A_0} \|(\bigcirc_{j=1}^n f_j)(x) - (\bigcirc_{j=1}^n \hat{f}_j)(x)\| < \sum_{j=1}^n \text{Lip}(\bigcirc_{\ell=j+1}^n f_\ell)\epsilon_i. \tag{A.2}$$

Proof of Proposition A.6. We have

$$\begin{aligned}
&\sup_{x \in A_0} \|(\bigcirc_{j=1}^n f_j)(x) - (\bigcirc_{j=1}^n \hat{f}_\ell)(x)\| \\
&\quad \le \sup_{x \in A_0} \sum_{j=1}^n \|(\bigcirc_{\ell=j+1}^n f_\ell \circ f_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x) - (\bigcirc_{\ell=j+1}^n f_\ell \circ \hat{f}_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x)\| \\
&\quad \le \sum_{j=1}^n \sup_{x \in A_0} \|(\bigcirc_{\ell=j+1}^n f_\ell \circ f_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x) - (\bigcirc_{\ell=j+1}^n f_\ell \circ \hat{f}_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x)\| \\
&\quad \le \sum_{j=1}^n \text{Lip}(\bigcirc_{\ell=j+1}^n f_\ell) \sup_{x \in A_0} \|(f_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x) - (\hat{f}_j \circ \bigcirc_{\ell=1}^{j-1} \hat{f}_\ell)(x)\| \\
&\quad \le \sum_{j=1}^n \text{Lip}(\bigcirc_{\ell=j+1}^n f_\ell) \sup_{y \in A_{j-1}} \|f_j(y) - \hat{f}_j(y)\| \\
&\quad \le \sum_{j=1}^n \text{Lip}(\bigcirc_{\ell=j+1}^n f_\ell)\epsilon_j.
\end{aligned}$$

∎

Appendix B. Proof of Main Results.

B.1. Proof of Lemma 3.7. In this section, we prove Lemma 3.7. In the following, the windows are the same as the segments. First, we prove lemmas that establish approximation results for the autoencoders, propagators, and transcoders.

LEMMA B.1 (Autoencoder Approximation). *Suppose Assumption 2 holds, and let* $0 < \epsilon_1, \epsilon_2 < \min\{1, \boldsymbol{\tau}(\mathcal{M}([0,T]))/2\}$. *Then for any* $i \in [\mathsf{S}]$, *there exists* $\mathscr{E}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(D, d+1, L_{\mathscr{E}^i}, W_{\mathscr{E}^i})$ *and* $\mathscr{D}^i_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, D, L_{\mathscr{D}^i}, W_{\mathscr{D}^i})$ *with parameters*

$$L_{\mathscr{E}^i} = O(\log^2(\epsilon_1^{-1})), \quad W_{\mathscr{E}^i} = O(D\epsilon_1^{-(d+1)})$$
$$L_{\mathscr{D}^i} = O(\log(\epsilon_2^{-1})), \quad W_{\mathscr{D}^i} = O(D\epsilon_2^{-(d+1)})$$

such that

$$\sup_{\mathbf{x}\in\mathcal{M}([s_i,s_{i+1}])} \|\mathscr{E}_{\mathrm{NN}}(\mathbf{x}) - \mathscr{E}^i_*(\mathbf{x})\|_{\mathbb{R}^{d+1}} \le \epsilon_1 \text{ and } \sup_{\mathbf{z}\in\mathcal{Z}([s_i,s_{i+1}])} \|\mathscr{D}_{\mathrm{NN}}(\mathbf{z}) - \mathscr{D}_*(\mathbf{z})\|_{\mathbb{R}^D} \le \epsilon_2.$$

Moreover, the range of $\mathscr{E}_{\mathrm{NN}}$ *is contained in* $\mathcal{Z}([0,T])$. *The constants hidden in* O *depend on (for each* $i \in [\mathsf{S}]$*)* $\log D$, d, $\boldsymbol{\tau}(\mathcal{M}([s_i, s_{i+1}]))$, $s_{i+1} - s_i$, $\mathrm{Lip}(\mathscr{E}^i_*)$, $\mathrm{Lip}(\mathscr{D}^i_*)$, *the volume of* $\mathcal{M}([s_i, s_{i+1}])$, *and* $\sup_{\mathbf{x}\in\mathcal{M}([s_i,s_{i+1}])} \|\mathbf{x}\|_{\mathbb{R}^D}$.

Proof of Lemma B.1. This is follows from [30, Lemma 6 and 8], with one additional step to restrict the range of $\mathscr{E}_{\mathrm{NN}}$. This is done by applying Proposition A.2(Part 2), which increases the number of layers by 2. ∎

LEMMA B.2 (Propagator Approximation with Latent Dynamics). *Suppose there is a Lipschitz function* $g : \mathcal{Z}([a,b]) \to \mathbb{R}^{d+1}$ *such that for all* $\mathbf{z}(a) \in \mathcal{Z}(a)$, *if we denote* $\mathbf{z}(t) = \mathscr{P}_*(\mathbf{z}(a), t-a)$ *for all* $t \in [a,b]$, *then these latent codes satisfy:*

$$\frac{\partial \mathbf{z}}{\partial t}(t) = g(\mathbf{z}(t)). \tag{B.1}$$

Denote $\mathrm{Lip}(g) = \sup_{t\in[a,b]} \mathrm{Lip}_{\mathcal{Z}(t)}(g(\mathbf{z}(t)))$ *and* $T = b - a$. *Suppose* $a = t_1 < \cdots < t_{\mathsf{T}-1} < t_{\mathsf{T}} = b$ *is an equally spaced time grid, i.e.* $t_{k-1} - t_k = \Delta t$ *is the same for all* $k \in [\mathsf{T}-1]$. *For any* $\epsilon > 0$ *and* $\mathsf{T} > 1 + \mathrm{Lip}(g)T^2 e^{\mathrm{Lip}(g)T}\epsilon^{-1}$, *then there is a* $\mathscr{P}_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L, W)$, *with range contained in* $\mathcal{Z}([a,b])$, *such that for any* $k \in [\mathsf{T}-1]$ *and* $i \in [\mathsf{T}-k]$,

$$\sup_{\mathbf{z}(t_k)\in\mathcal{Z}(t_k)} \| \left(\bigcirc_{j=1}^{i} \mathscr{P}_{\mathrm{NN}}\right)(\mathbf{z}(t_k)) - \mathscr{P}_*(\mathbf{z}(t_k), t_{k+i} - t_k)\|_{\mathbb{R}^{d+1}} < \epsilon.$$

Here $L = O\left(\log\left(\frac{1}{\epsilon}\right)\right)$, $W = O\left(\epsilon^{-(d+1)}\right)$, *where the constants hidden in* O *depend on* d, $\mathrm{Lip}(g)$, $\|g\|_{L^\infty(\mathcal{Z}([0,T]))}$, *and* T.

Proof of Lemma B.2. Since g is Lipschitz, by Proposition A.5 there is a $g_{\mathrm{NN}} \in g_{\mathrm{NN}}(d+1, d+1, L_g, W_g)$ such that

$$\sup_{\mathbf{z}\in\mathcal{Z}([a,b])} \|g_{\mathrm{NN}}(\mathbf{z}) - g(\mathbf{z})\|_{\mathbb{R}^{d+1}} < \delta := \frac{\epsilon}{6Te^{\mathrm{Lip(g)T}}}.$$

Here $L_g = cO\left(\log\left(\frac{1}{\epsilon}\right)\right)$ and $W_g = O\left(\epsilon^{-(d+1)}\right)$. We define $\mathscr{P}_{\mathrm{NN}} : \mathcal{Z}([a,b]) \to \mathcal{Z}([a,b])$ component-wise. For all $i \in [d]$, define the ith component of $\mathscr{P}_{\mathrm{NN}}$ as

$$\mathscr{P}^i_{\mathrm{NN}}(\mathbf{z}) = \max(0, \min(\mathbf{z}_i + \Delta t(g_{\mathrm{NN}}(\mathbf{z}))^i)$$

and the $(d+1)$th component as $\mathscr{P}^{d+1}_{\mathrm{NN}}(\mathbf{z}_1, \ldots, \mathbf{z}_{d+1}) = \mathbf{z}_{d+1} + \Delta t$. Then note we can implement $\mathscr{P}_{\mathrm{NN}}$ as a neural network by modifying the last layer of $\mathscr{F}_{\mathrm{NN}}$ and then adding two layers using Proposition A.2(Part 2). Then $\mathscr{P}_{\mathrm{NN}} \in g_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{P}}, W_{\mathscr{P}})$ with $L_{\mathscr{P}} = O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ and $W_{\mathscr{P}} = O\left(\epsilon^{-(d+1)}\right)$.

We next use classical argument for the convergence of the Euler method for ODEs to finish the proof (c.f. [27] Chapter 6). Let $\mathbf{z}(a) \in \mathcal{Z}(a)$. For all $k \in [\mathsf{T}]$, define $\mathbf{z}(t_k) = \mathscr{P}_*(\mathbf{z}(a), t_k - a)$, and define $\hat{\mathbf{z}}_k$ using the iterative formula

$$\hat{\mathbf{z}}_k = \mathscr{P}_{\mathrm{NN}}(\hat{\mathbf{z}}_{k-1}), \quad \hat{\mathbf{z}}_1 = \mathbf{z}(a_i).$$

Note that

$$\begin{aligned}\mathbf{z}(t_{k+1}) &= \mathbf{z}(t_k) + \Delta t \frac{\partial \mathbf{z}}{\partial t}(t_k) + \int_{t_k}^{t_{k+1}} \left(\frac{\partial \mathbf{z}}{\partial t}(s) - \frac{\partial \mathbf{z}}{\partial t}(t_k) \right) ds \\ &= \mathbf{z}(t_k) + \Delta t g(\mathbf{z}(t_k)) + \int_{t_k}^{t_{k+1}} (g(\mathbf{z}(s)) - g(\mathbf{z}(t_k)))\, ds.\end{aligned}$$

This means

$$\begin{aligned}\left\| \frac{1}{\Delta t}(\mathbf{z}(t_{k+1}) - \mathbf{z}(t_k)) - g(\mathbf{z}(t_k)) \right\|_{\mathbb{R}^d} &= \frac{1}{\Delta t} \left\| \int_{t_k}^{t_{k+1}} (g(\mathbf{z}(s)) - g(\mathbf{z}(t_k)))\, ds \right\|_{\mathbb{R}^{d+1}} \\ &\le \frac{1}{\Delta t} \int_{t_k}^{t_{k+1}} \|g(\mathbf{z}(s)) - g(\mathbf{z}(t_k))\|_{\mathbb{R}^{d+1}} ds \\ &\le \frac{1}{\Delta t} \int_{t_k}^{t_{k+1}} \mathrm{Lip}(g) \|\mathbf{z}(s) - \mathbf{z}(t_k)\|_{\mathbb{R}^{d+1}} ds \\ &\le \frac{\mathrm{Lip}(g)}{\Delta t} \int_{t_k}^{t_{k+1}} \|g\|_{L^\infty(\mathcal{Z}([a,b]))} (s - t_k) ds \\ &\le \frac{\mathrm{Lip}(g)}{\Delta t} \|g\|_{L^\infty(\mathcal{Z}([a,b]))} \frac{(\Delta t)^2}{2} \le C \Delta t,\end{aligned}$$

where $C = \mathrm{Lip}(g)\|g\|_{L^\infty(\mathcal{Z}([a,b]))}/2$.

Now, let $\mathbf{e}_k = \frac{1}{\Delta t}(\mathbf{z}(t_{k+1}) - \mathbf{z}(t_k)) - g_{\mathrm{NN}}(\mathbf{z}(t_k))$ be the step k truncation error for Euler method of solving the ODE (B.1) while the governing equation is g_{NN} instead of g. We have

$$\mathbf{z}(t_{k+1}) = \mathbf{z}(t_k) + \Delta t g_{\mathrm{NN}}(\mathbf{z}(t_k)) + \Delta t \mathbf{e}_k.$$

This local truncation error satisfies the error bound:

$$\begin{aligned}\|\mathbf{e}_k\|_{\mathbb{R}^{d+1}} &= \left\| \frac{1}{\Delta t}(\mathbf{z}(t_{k+1}) - \mathbf{z}(t_k)) - g_{\mathrm{NN}}(\mathbf{z}(t_k)) \right\|_{\mathbb{R}^{d+1}} \\ &\le \left\| \frac{1}{\Delta t}(\mathbf{z}(t_{k+1}) - \mathbf{z}(t_k)) - g(\mathbf{z}(t_k)) \right\|_{\mathbb{R}^{d+1}} + \|g(\mathbf{z}(t_k)) - g_{\mathrm{NN}}(\mathbf{z}(t_k))\|_{\mathbb{R}^{d+1}} \\ &\le C\Delta t + \delta.\end{aligned}$$

Then we decompose

$$\begin{aligned}\mathbf{z}(t_{k+1}) - \hat{\mathbf{z}}_{k+1} &= \mathbf{z}(t_k) + \Delta t g_{\mathrm{NN}}(\mathbf{z}(t_k)) + \Delta t \mathbf{e}_k - \hat{\mathbf{z}}_k - \Delta t g_{\mathrm{NN}}(\hat{\mathbf{z}}_k) \\ &= (\mathbf{z}(t_k) - \hat{\mathbf{z}}_k) + \Delta t (g_{\mathrm{NN}}(\mathbf{z}(t_k)) - g_{\mathrm{NN}}(\hat{\mathbf{z}}_k)) + \Delta t \mathbf{e}_k.\end{aligned}$$

This means

$$\|\mathbf{z}(t_{k+1}) - \hat{\mathbf{z}}_{k+1}\|_{\mathbb{R}^{d+1}} \le \|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k\|_{\mathbb{R}^{d+1}} + \Delta t \|g_{\mathrm{NN}}(\mathbf{z}(t_k)) - g_{\mathrm{NN}}(\hat{\mathbf{z}}_k)\|_{\mathbb{R}^{d+1}} + \Delta t \|\mathbf{e}_k\|_{\mathbb{R}^{d+1}}.$$

Now $\|g_{\mathrm{NN}}(\mathbf{z}(t_k)) - g_{\mathrm{NN}}(\hat{\mathbf{z}}_k)\|_{\mathbb{R}^{d+1}} \le \|g_{\mathrm{NN}}(\mathbf{z}(t_k)) - g(\mathbf{z}(t_k))\|_{\mathbb{R}^{d+1}} + \|g(\mathbf{z}(t_k)) - g(\hat{\mathbf{z}}_k)\|_{\mathbb{R}^{d+1}} + \|g(\hat{\mathbf{z}}_k) - g_{\mathrm{NN}}(\hat{\mathbf{z}}_k)\|_{\mathbb{R}^{d+1}} \le 2\delta + \mathrm{Lip}(g)\|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k\|_{\mathbb{R}^{d+1}}$. So we have the inequality

$$\|\mathbf{z}(t_{k+1}) - \hat{\mathbf{z}}_{k+1}\|_{\mathbb{R}^{d+1}} \le (1 + \Delta t \mathrm{Lip}(g)) \|\mathbf{z}(t_k) - \hat{\mathbf{z}}_k\|_{\mathbb{R}^{d+1}} + \Delta t(2\delta + \|\mathbf{e}_k\|_{\mathbb{R}^{d+1}}).$$

Since $\|\mathbf{z}(t_1) - \hat{\mathbf{z}}_1\|_{\mathbb{R}^d} = 0$, this implies

$$\|\mathbf{z}(t_{k+1}) - \hat{\mathbf{z}}_{k+1}\|_{\mathbb{R}^{d+1}}$$
$$\le \Delta t \sum_{i=1}^{k} (1 + \Delta t \mathrm{Lip}(g))^{k-i} (2\delta + \|\mathbf{e}_i\|_{\mathbb{R}^{d+1}}) \le e^{\mathrm{Lip}(g)T} \Delta t \sum_{i=1}^{k} (2\delta + \|\mathbf{e}_i\|_{\mathbb{R}^{d+1}}).$$

Therefore (since $k\Delta t < T$ and $\|\mathbf{e}_k\|_{\mathbb{R}^{d+1}} \le C\Delta t + \delta$)

$$\|\mathbf{z}(t_{k+1}) - \hat{\mathbf{z}}_{k+1}\|_{\mathbb{R}^{d+1}} \le e^{\mathrm{Lip}(g)T} \Delta t \sum_{i=1}^{k} (3\delta + C\Delta t)$$
$$= e^{\mathrm{Lip}(g)T} (k\Delta t)(3\delta + C\Delta t) \le 3Te^{\mathrm{Lip}(g)T}\delta + CTe^{\mathrm{Lip}(g)T}\Delta t.$$

Now since $\Delta t = \frac{T}{\mathsf{T}-1}$, if we have $\mathsf{T} - 1 > \frac{2CT^2 e^{\mathrm{Lip}(g)T}}{\epsilon}$, then

$$3Te^{\mathrm{Lip}(g)T}\delta + CTe^{\mathrm{Lip}(g)T}\Delta t$$
$$\le \frac{3Te^{\mathrm{Lip}(g)T}\epsilon}{6Te^{\mathrm{Lip}(g)T}} + \frac{CT^2 e^{\mathrm{Lip}(g)T}}{\mathsf{T}-1} \le \frac{\epsilon}{2} + (CT^2 e^{\mathrm{Lip}(g)T}) \frac{\epsilon}{2CT^2 e^{\mathrm{Lip}(g)T}} = \epsilon. \square$$

LEMMA B.3 (Transcoder Approximation). *Suppose Assumption 2 holds. Fix an $i \in [\mathsf{S}]$, and for any $\epsilon > 0$, there exists a $\mathscr{T}_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L_{\mathscr{T}}, W_{\mathscr{T}})$ with parameters*

$$L_{\mathscr{T}} = O(\log(\epsilon^{-1})), \quad W_{\mathscr{T}} = O(\epsilon^{-d})$$

such that

$$\sup_{\mathbf{z}(s_{i+1}) \in \mathcal{Z}(s_{i+1})} \left\| \mathscr{T}_{\mathrm{NN}}(\mathbf{z}(s_{i+1})) - \mathscr{E}_*^{i+1}(\mathscr{D}_*^i(\mathbf{z}(s_{i+1}))) \right\|_{\mathbb{R}^{d+1}} < \epsilon$$

Moreover, the range of $\mathscr{T}_{\mathrm{NN}}$ is contained in $\mathcal{Z}(s_{i+1})$. Here, $L_{\mathscr{T}} = O\left(\log\left(\frac{1}{\epsilon}\right)\right)$ and $W_{\mathscr{T}} = O\left(\epsilon^{-\frac{1}{d}}\right)$. The constants hidden in O depend on d, $\tau_{\mathcal{M}}$, $\mathrm{Lip}_{\mathcal{M}(s_{i+1})}(\mathscr{E}_*^{i+1})$, $\mathrm{Lip}_{\mathcal{Z}(s_{i+1})}(\mathscr{D}_*^i)$, the volume of $\mathcal{M}(s_{i+1})$, and $\sup_{x \in \mathcal{M}(s_{i+1})} \|x\|_{\mathbb{R}^D}$.

Proof of Lemma B.3. Consider the oracle transcoder $\mathscr{T}_* = \mathscr{E}_*^{i+1} \circ \mathscr{D}_*^i$, which satisfies the Lipschitz condition:

$$\mathrm{Lip}_{\mathcal{Z}(s_{i+1})}(\mathscr{T}_*) = \mathrm{Lip}_{\mathcal{Z}(s_{i+1})}(\mathscr{E}_*^{i+1} \circ \mathscr{D}_*^i) \le \mathrm{Lip}_{\mathcal{Z}(s_{i+1})}(\mathscr{E}_*^{i+1}) \mathrm{Lip}_{\mathcal{M}(s_{i+1})}(\mathscr{D}_*^i) < \infty.$$

The proof then follows by applying Proposition A.5 to $\mathscr{T}_*$. The width of the neural network size is exponential in d instead of $d+1$, because the function $\mathscr{T}_*$ leaves the $(d+1)$th component unchanged (as that component is the time), so it can be exactly represented by a neural network with no need for approximation. $\square$

Now we prove Lemma 3.7. We refer to Table 1 for some important notations to be used in the proof.

Proof of Lemma 3.7. For any $i \in [\mathsf{S}]$, denote $\boldsymbol{T}_i = |\mathbb{T} \cap (s_i, s_{i+1}]|$. Suppose t_k is the jth element of $\mathbb{T} \cap (s_i, s_{i+1}]$, i.e. it is in window/segment number i. We will construct encoder, decoder, propagator, and transcoder networks such that for any t_k,

(B.2)
$$\left\| \mathscr{D}^i_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{j} \mathscr{P}^i_{\mathrm{NN}} \circ \bigcirc_{w=1}^{i-1} \left(\mathscr{T}^w_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_{\mathrm{NN}} \right) \circ \mathscr{E}^1_{\mathrm{NN}} - \mathscr{F}(\cdot, t_k) \right\|_{L^\infty(\mathcal{M}(0))} < \epsilon.$$

We denote t^w_ℓ as the ℓth element of $\mathbb{T} \cap [s_w, s_{w+1}]$. Note that the evolution operator to t_k is

$$\begin{aligned}
&\mathscr{F}(\cdot, t_k) \\
&= \bigcirc_{\ell=1}^{j} \mathscr{F}(\cdot, t^i_{\ell+1} - t^i_\ell) \circ \bigcirc_{w=1}^{i-1} \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{F}(\cdot, t^w_{\ell+1} - t^w_\ell) \\
&= \mathscr{D}^i_* \circ \bigcirc_{\ell=1}^{j} (\mathscr{E}^i_* \circ \mathscr{F}(\cdot, t^i_{\ell+1} - t_\ell) \circ \mathscr{D}^i_*) \circ \\
&\quad \bigcirc_{w=1}^{i-1} \left(\mathscr{E}^{w+1}_* \circ \mathscr{D}^w_* \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} (\mathscr{E}^w_* \circ \mathscr{F}(\cdot, t^w_{\ell+1} - t^w_\ell) \circ \mathscr{D}^w_*) \right) \circ \mathscr{E}^1_*.
\end{aligned}$$

We will approximate each term of the composition in this expression by neural networks, and then apply Proposition A.6.

For all $w \in [\mathsf{S}]$, consider the encoders and decoders $\mathscr{E}^w_*$ and $\mathscr{D}^w_*$. By Lemma B.1, there is a neural network $\mathscr{E}^w_{\mathrm{NN}}$ with $O\left(\log^2\left(\frac{\mathsf{S}}{\epsilon}\right)\right)$ layers and width $O\left(D\left(\frac{\mathsf{S}}{\epsilon}\right)^{d+1}\right)$ and a neural network $\mathscr{D}^w_{\mathrm{NN}}$ with $O\left(\log\left(\frac{\mathsf{S}}{\epsilon}\right)\right)$ layers and width $O\left(D\left(\frac{\mathsf{S}}{\epsilon}\right)^{d+1}\right)$ such that

(B.3)
$$\begin{aligned}
\|\mathscr{E}^w_{\mathrm{NN}} - \mathscr{E}^w_*\|_{L^\infty(\mathcal{M}([s_w, s_{w+1}]))} &< \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D}_*)\mathrm{Lip}(\mathscr{F})}, \\
\|\mathscr{D}^w_* - \mathscr{D}^w_{\mathrm{NN}}\|_{L^\infty(\mathcal{Z}([s_w, s_{w+1}]))} &< \frac{\epsilon}{2\mathsf{S}+1},
\end{aligned}$$

where $\mathrm{Lip}(\mathscr{D}_*) = \max_w \mathrm{Lip}(\mathscr{D}^w_*)$. Note that the range of $\mathscr{E}^w_{\mathrm{NN}}$ is contained in $\mathcal{Z}([s_w, s_{w+1}])$.

Next, for all $w \in [\mathsf{W}]$ consider $\mathscr{P}^w_*(\mathbf{z}, t) = \mathscr{E}^w_*(\mathscr{F}(\mathscr{D}^w_*(\mathbf{z}), t))$ which is called the oracle propagator. Since $\mathscr{P}^w_*$ is Lipschitz as a composition of Lipschitz functions, by Lemma B.2, there is a neural network $\mathscr{P}^w_{\mathrm{NN}}$ with $O\left(\log\left(\frac{\mathsf{S}}{\epsilon}\right)\right)$ layers and width $O\left(\left(\frac{\mathsf{S}}{\epsilon}\right)^{d+1}\right)$ such that

(B.4)
$$\| \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_{\mathrm{NN}} - \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_*(\cdot, t^w_{\ell+1} - t^w_\ell) \|_{L^\infty(\mathcal{Z}(s_w))} < \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})}.$$

The composition above makes sense, since the range of $\mathscr{P}^w_{\mathrm{NN}}$ is contained in $\mathcal{Z}([s_w, s_{w+1}])$ by construction.

Finally, for all $w \in [\mathsf{S}-1]$, consider $\mathscr{E}^{w+1}_* \circ \mathscr{D}^w_*$. By Lemma B.3, there is a neural network $\mathscr{T}^w_{\mathrm{NN}}$ with $O\left(\log\left(\frac{\mathsf{S}}{\epsilon}\right)\right)$ layers and width $O\left(\left(\frac{\mathsf{S}}{\epsilon}\right)^{d}\right)$ such that

(B.5)
$$\|\mathscr{T}^w_{\mathrm{NN}} - \mathscr{E}^{w+1}_* \circ \mathscr{D}^w_*\|_{L^\infty(\mathcal{Z}(s_{w+1}))} < \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})},$$

with the range of $\mathscr{T}^w_{\mathrm{NN}}$ contained in $\mathcal{Z}(s_{w+1}) = [0,1]^d \times \{s_{w+1}\}$. Theorem 3.9 then follows by applying Proposition A.6 to the expression (where we indicate matching terms with the same letter and subscript):

$$\| \underbrace{\mathscr{D}^i_{\mathrm{NN}}}_{a} \circ \underbrace{\bigcirc_{\ell=1}^{j} \mathscr{P}^i_{\mathrm{NN}}}_{b} \circ \bigcirc_{w=1}^{i-1} (\underbrace{\mathscr{T}^w_{\mathrm{NN}}}_{c_w} \circ \underbrace{\bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_{\mathrm{NN}}}_{d_w}) \circ \underbrace{\mathscr{E}^1_{\mathrm{NN}}}_{e} -$$

$$\underbrace{\mathscr{D}_*^i}_{a} \circ \underbrace{\bigcirc_{\ell=1}^{j} \mathscr{P}_*^w(\cdot, t_{\ell+1}^i - t_\ell)}_{b} \circ \bigcirc_{w=1}^{i-1} \underbrace{(\mathscr{E}_*^{w+1} \circ \mathscr{D}_*^w)}_{c_w} \circ \underbrace{\bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}_*^w(\cdot, t_{\ell+1}^w - t_{\ell+1}^w))}_{d_w} \circ \underbrace{\mathscr{E}_*^1}_{e} \|,$$

where the norm is taken over the space $L^\infty(\mathcal{M}(0))$. Note that we can do this because the range of each neural network in the composition is contained in the domain of the subsequent neural network (by construction). To compute the bound given in Proposition A.6, we compute Lipschitz constants of compositions of functions. Both terms in the difference involve $2+2(i-1)+1 = 2i+1$ functions. Each term in Equation (A.2) is the product of the approximation error of a network (encoder, propagator, transcoder, or decoder), and the Lipschitz constant of the oracle maps that come later in the approximation. We will show that each term in the sum is less than $\frac{\epsilon}{2i+1}$, so that the total error is less than ϵ. Recall $t_{\ell+1}^i = t_k$ and $a_1 = t_1$.

(a) **Decoder**. The decoder $\mathscr{D}_*^i$ corresponds to the $(2i+1)$th and last term of the sum in Equation (A.2). The Lipschitz factor is over an empty composition, which is defined to be 1 by convention. Thus the decoder term contributes an error of $\frac{\epsilon}{2\mathsf{S}+1}$ by Equation (B.3), which is less than $\frac{\epsilon}{2i+1}$ since $i \le \mathsf{S}$.

(b) **Final Propagator**. The final propagator is $\mathscr{P}_*^i$, and it corresponds to the second to last term of the sum in Equation (A.2). The Lipschitz factor is the Lipschitz constant of the decoder $\mathscr{D}_*^i$. Thus this term contributes an error of (by the approximation in (B.4))

$$\mathrm{Lip}_{\mathcal{Z}^i(t_k)}(\mathscr{D}_*^i) \cdot \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})} \le \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{F})} \le \frac{\epsilon}{2i+1}.$$

(c) **Transcoder**. For any window $m \in [i-1]$, the mth transcoder corresponds to a middle term in the sum in Equation (A.2). For each of these terms, the error is again given by the approximation error of the corresponding transcoder network, scaled by the Lipschitz constant of the neural networks that come after. The Lipschitz constant of the composition of all functions after the mth transcoder are

$$\begin{aligned}
&\mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{P}_*^i(\cdot, t_{\ell+1}^i - t_\ell^i)) \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1} \circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{P}_*^w(\cdot, t_{\ell+1}^w - t_\ell^w)\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{E}_*^i \circ \mathscr{F}(\cdot, t_{\ell+1}^i - t_\ell^i) \circ \mathscr{D}_*^i) \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1} \circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{E}_*^w \circ \mathscr{F}(\cdot, t_{\ell+1}^w - t_\ell^w) \circ \mathscr{D}_*^w)\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \mathscr{E}_*^i \circ \mathscr{F}(\cdot, t_{j+1}^i - s_i) \circ \mathscr{D}_*^i \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1} \circ \mathscr{D}_*^w \circ \mathscr{E}_*^w \circ \mathscr{F}(\cdot, s_{w+1} - s_w) \circ \mathscr{D}_*^w\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i) \circ \mathscr{D}_*^i \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1} \circ \mathscr{F}(\cdot, s_{w+1} - s_w) \circ \mathscr{D}_*^w\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i) \circ \mathscr{D}_*^i \circ \mathscr{E}_*^i \circ \mathscr{F}(\cdot, s_i - s_{m+1}) \circ \mathscr{D}_*^{m+1}\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i) \circ \mathscr{F}(\cdot, s_i - s_{m+1}) \circ \mathscr{D}_*^{m+1}\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_{m+1}) \circ \mathscr{D}_*^{m+1}\right) \le \mathrm{Lip}_{\mathcal{M}(s_{m+1})}(\mathscr{F}(\cdot, t_k - s_{m+1}))\mathrm{Lip}_{\mathcal{Z}^{m+1}(s_{m+1})}(\mathscr{D}_*^{m+1}).
\end{aligned}$$

∎

Thus, by scaling the approximation guarantee in Equation (B.5), we see that each transcoder term contributes error at most

$$\begin{aligned}
&\mathrm{Lip}_{\mathcal{M}(s_{m+1})}(\mathscr{F}(\cdot, t_k - s_{m+1}))\mathrm{Lip}_{\mathcal{Z}^{m+1}(s_{m+1})}(\mathscr{D}_*^{m+1}) \frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})}\\
&\quad \le \frac{\epsilon}{2\mathsf{S}+1} \le \frac{\epsilon}{2i+1}.
\end{aligned}$$

(d) **Intermediate Propagators**. For any window $m \in [i-1]$, the mth propagator corresponds to a middle term in the sum in Equation (A.2), similar to the transcoder case. The Lipschitz constant of the composition of all functions after the mth oracle propagator are

$$
\begin{aligned}
&\mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{P}_*^i(\cdot, t_{\ell+1}^i - t_\ell^i)\circ \mathscr{D}_*^i) \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{P}_*^w(\cdot, t_{\ell+1}^w - t_\ell^w)\right)\circ \mathscr{E}_*^{m+1}\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{E}_*^i\circ \mathscr{F}(\cdot, t_{\ell+1}^i - t_\ell^i)\circ \mathscr{D}_*^i) \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{E}_*^w\circ \mathscr{F}(\cdot, t_{\ell+1}^w - t_\ell^w)\circ \mathscr{D}_*^w)\right)\circ \mathscr{E}_*^{m+1}\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{D}_*^i \circ \mathscr{E}_*^i\circ \mathscr{F}(\cdot, t_{j+1}^i - s_i)\circ \mathscr{D}_*^i \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \mathscr{E}_*^w\circ \mathscr{F}(\cdot, s_{w+1} - s_w)\circ \mathscr{D}_*^w\right)\circ \mathscr{E}_*^{m+1}\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i)\circ \mathscr{D}_*^i \circ \bigcirc_{w=m+1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{F}(\cdot, s_{w+1} - s_w)\circ \mathscr{D}_*^w\right)\circ \mathscr{E}_*^{m+1}\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i)\circ \mathscr{D}_*^i \circ \mathscr{E}_*^i\circ \mathscr{F}(\cdot, s_i - s_{m+1})\circ \mathscr{D}_*^{m+1}\circ \mathscr{E}_*^{m+1}\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_i)\circ \mathscr{F}(\cdot, s_i - s_{m+1})\circ \mathscr{D}_*^m\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}\left(\mathscr{F}(\cdot, t_k - s_{m+1})\circ \mathscr{D}_*^m\right) \le \mathrm{Lip}_{\mathcal{M}(s_{m+1})}(\mathscr{F}(\cdot, t_k - s_{m+1}))\mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}(\mathscr{D}_*^m).
\end{aligned}
$$

Thus, by scaling the approximation guarantee in Equation (B.4), we see that each intermediate propagator term contributes error at most

$$
\begin{aligned}
&\mathrm{Lip}_{\mathcal{M}(s_{m+1})}(\mathscr{F}(\cdot, t_k - s_{m+1}))\mathrm{Lip}_{\mathcal{Z}^m(s_{m+1})}(\mathscr{D}_*^m)\frac{\epsilon}{(2\mathsf{W}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})}\\
&\quad\le \frac{\epsilon}{2\mathsf{W}+1} \le \frac{\epsilon}{2i+1}.
\end{aligned}
$$

(e) **Encoder**. The encoder term $\mathscr{E}^1$ corresponds to the first term of the sum in Equation (A.2). The Lipschitz factor can be computed as:

$$
\begin{aligned}
&\mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{P}_*^i(\cdot, t_{\ell+1}^i - t_\ell^i)\circ \mathscr{D}_*^i) \circ \bigcirc_{w=1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{P}_*^w(\cdot, t_{\ell+1}^w - t_\ell^w)\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{D}_*^i \circ \bigcirc_{\ell=1}^{j}(\mathscr{E}_*^i\circ \mathscr{F}(\cdot, t_{\ell+1}^i - t_\ell^i)\circ \mathscr{D}_*^i) \circ \bigcirc_{w=1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w}(\mathscr{E}_*^w\circ \mathscr{F}(\cdot, t_{\ell+1}^w - t_\ell^w)\circ \mathscr{D}_*^w)\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{D}_*^i \circ \mathscr{E}_*^i\circ \mathscr{F}(\cdot, t_{j+1}^i - s_i)\circ \mathscr{D}_*^i \circ \bigcirc_{w=1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{D}_*^w \circ \mathscr{E}_*^w\circ \mathscr{F}(\cdot, s_{w+1} - s_w)\circ \mathscr{D}_*^w\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{F}(\cdot, t_k - s_i)\circ \mathscr{D}_*^i \circ \bigcirc_{w=1}^{i-1}\left(\mathscr{E}_*^{w+1}\circ \mathscr{F}(\cdot, s_{w+1} - s_w)\circ \mathscr{D}_*^w\right)\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{F}(\cdot, t_k - s_i)\circ \mathscr{D}_*^i \circ \mathscr{E}_*^i\circ \mathscr{F}(\cdot, s_i - s_1)\circ \mathscr{D}_*^1\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{F}(\cdot, t_k - t_1)\circ \mathscr{F}(\cdot, s_i - s_1)\circ \mathscr{D}_*^1\right)\\
&= \mathrm{Lip}_{\mathcal{Z}^1(t_1)}\left(\mathscr{F}(\cdot, t_k - t_1)\circ \mathscr{D}_*^1\right) \le \mathrm{Lip}_{\mathcal{M}(t_1)}(\mathscr{F}(\cdot, t_k - t_1))\mathrm{Lip}_{\mathcal{Z}^1(t_1)}(\mathscr{D}_*^1).
\end{aligned}
$$

Thus, by scaling the approximation guarantee in Equation (B.3), we see that the encoder term contributes error at most

$$
\begin{aligned}
&\mathrm{Lip}_{\mathcal{M}(t_1)}(\mathscr{F}(\cdot, t_k - t_1))\mathrm{Lip}_{\mathcal{Z}^1(t_1)}(\mathscr{D}_*^1)\frac{\epsilon}{(2\mathsf{S}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})}\\
&\quad\le \frac{\epsilon}{2\mathsf{S}+1} \le \frac{\epsilon}{2i+1}.
\end{aligned}
$$

B.2. Proof of Theorem 3.9. In this section, we prove Theorem 3.9, which follows quickly from Lemma 3.7. The key is to construct a WeldNet model with more windows that implements exactly the same function as the WeldNet constructed before.

Proof of Theorem 3.9. Let $\pi : [\mathsf{W}] \to [\mathsf{S}]$ be the function that indicates (with the index) which segment each window falls inside. Consider the S-window WeldNet from

Lemma 3.7 denoted $\overline{\mathscr{W}_{\mathrm{NN}}}$ with components (for all $s \in [\mathsf{S}]$) $\overline{\mathscr{E}^s_{\mathrm{NN}}}$, $\overline{\mathscr{D}^s_{\mathrm{NN}}}$, $\overline{\mathscr{P}^s_{\mathrm{NN}}}$, and (for $s < \mathsf{S}$) $\overline{\mathscr{T}^s_{\mathrm{NN}}}$ such that for any $k \in [\mathsf{T}]$ with k being the jth element of $\mathsf{T} \cap [s_i, s_{i+1})$),

$$\sup_{\mathbf{x}(0)\in\mathcal{M}(0)} \left\| \overline{\mathscr{W}_{\mathrm{NN}}}(\mathbf{x}(0), t_k) - \mathscr{F}(\mathbf{x}(0), t_k) \right\|_{\mathbb{R}^D} < \epsilon.$$

We now construct a W-window WeldNet that implements the same function as the S-window $\overline{\mathscr{W}_{\mathrm{NN}}}$. For each $i \in [\mathsf{W}]$, consider the neural networks

$$\mathscr{E}^i_{\mathrm{NN}} = \overline{\mathscr{E}^{\pi(i)}_{\mathrm{NN}}}, \mathscr{D}^i_{\mathrm{NN}} = \overline{\mathscr{D}^{\pi(i)}_{\mathrm{NN}}}, \mathscr{P}^i_{\mathrm{NN}} = \overline{\mathscr{P}^{\pi(i)}_{\mathrm{NN}}},$$

and for each $i \in [\mathsf{W}]$ such that $w_{i+1} = s_{j+1}$ for some $j \in [\mathsf{S}]$ (i.e windows that end at the end of a segment), we define $\mathscr{T}^i_{\mathrm{NN}} = \overline{\mathscr{T}^j_{\mathrm{NN}}}$, and we construct every other transcoder to be the identity function in $\mathbb{R}^{d+1}$ given by Proposition A.1.

Since for all $\mathbf{x}(0) \in \mathcal{M}(0)$, we have $\|\overline{\mathscr{W}_{\mathrm{NN}}}(\mathbf{x}(0), t_k) - \mathscr{F}(\mathbf{x}(0), t_k)\|_{\mathbb{R}^D} < \epsilon$, we can complete the proof by showing that $\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) = \overline{\mathscr{W}_{\mathrm{NN}}}(\mathbf{x}(0), t_k)$. We denote $\boldsymbol{T}_i = |\mathbb{T} \cap (w_i, w_{i+1}]|$ and $\overline{\boldsymbol{T}_i} = |\mathbb{T} \cap (s_i, s_{i+1}]|$. Suppose that t_k is the jth element of $\mathbb{T} \cap (w_i, w_{i+1}]$ and also that t_k is the j'-th element of $\mathbb{T} \cap (s_{i'}, s_{i'+1}]$ (which means $\pi(i) = i'$). Then we have

$$\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) = \mathscr{D}^i_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{j} \mathscr{P}^i_{\mathrm{NN}} \circ \bigcirc_{w=1}^{i-1} \left(\mathscr{T}^w_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_{\mathrm{NN}} \right) \circ \mathscr{E}^1_{\mathrm{NN}}(\mathbf{x}(0)) \text{ and}$$

$$\overline{\mathscr{W}_{\mathrm{NN}}}(\mathbf{x}(0), t_k) = \overline{\mathscr{D}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left(\overline{\mathscr{T}^s_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\overline{\boldsymbol{T}_s}} \overline{\mathscr{P}^s_{\mathrm{NN}}} \right) \circ \overline{\mathscr{E}^1_{\mathrm{NN}}}(\mathbf{x}(0)).$$

Note that $\mathscr{E}^1_{\mathrm{NN}} = \overline{\mathscr{E}^{\pi(1)}_{\mathrm{NN}}}$ since the first window is inside of the first segment. Next, note that $\mathscr{D}^i_{\mathrm{NN}} = \overline{\mathscr{D}^{\pi(i)}_{\mathrm{NN}}} = \overline{\mathscr{D}^{i'}_{\mathrm{NN}}}$ and $\mathscr{P}^i_{\mathrm{NN}} = \overline{\mathscr{P}^{\pi(i)}_{\mathrm{NN}}} = \overline{\mathscr{P}^{i'}_{\mathrm{NN}}}$. Finally, we will decompose the composition of propagators and transcoders to be over each segment before being over each window.

Let n^m_s denote the index of the mth window in the sth segment, and let $\boldsymbol{n}_s$ denote the index of the final window in the sth segment. Note that all transcoders except for the last window in each segment implements the identity, i.e. $\mathscr{T}^{n^m_s}_{\mathrm{NN}} = I_{d+1}$ if $m < |\pi^{-1}(\{s\})|$, where $\pi^{-1}(\{s\})$ is the pre-image of s by π, which is the set of all window indices inside of Segment s, and the last transcoder within segment s of $\mathscr{W}_{\mathrm{NN}}$ is equal to the sth transcoder of $\overline{\mathscr{W}_{\mathrm{NN}}}$, i.e. $\mathscr{T}^{\boldsymbol{n}_s}_{\mathrm{NN}} = \overline{\mathscr{T}^s_{\mathrm{NN}}}$. Also by construction $\mathscr{P}^{n^m_s}_{\mathrm{NN}} = \overline{\mathscr{P}^s_{\mathrm{NN}}}$. Then we decompose the non encoder/decoder terms of the above composition

$$
\begin{aligned}
&\bigcirc_{\ell=1}^{j} \mathscr{P}^{i}_{\mathrm{NN}} \circ \bigcirc_{w=1}^{i-1} \left(\mathscr{T}^{w}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^{w}_{\mathrm{NN}} \right) \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i}_{\mathrm{NN}}} \circ \bigcirc_{w=1}^{i-1} \left(\mathscr{T}^{w}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^{w}_{\mathrm{NN}} \right) \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\bigcirc_{m=1}^{|\pi^{-1}(s)|} \left(\mathscr{T}^{n_s^m}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{n_s^m}} \mathscr{P}^{n_s^m}_{\mathrm{NN}} \right) \right] \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\bigcirc_{m=1}^{|\pi^{-1}(s)|} \left(\mathscr{T}^{n_s^m}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{n_s^m}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right) \right] \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\left(\mathscr{T}^{\boldsymbol{n}_s}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{\boldsymbol{n}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right) \circ \bigcirc_{m=1}^{|\pi^{-1}(s)|-1} \left(\mathscr{T}^{n_s^m}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{n_s^m}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right) \right] \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\overline{\mathscr{T}^{s}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{\boldsymbol{n}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \circ \bigcirc_{m=1}^{|\pi^{-1}(s)|-1} \left(I_{d+1} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{n_s^m}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right) \right] \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\overline{\mathscr{T}^{s}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_{\boldsymbol{n}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \circ \bigcirc_{m=1}^{|\pi^{-1}(s)|-1} \bigcirc_{\ell=1}^{\boldsymbol{T}_{n_s^m}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right] \\
&= \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\overline{\mathscr{T}^{s}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\overline{\boldsymbol{T}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right].
\end{aligned}
$$

Thus,

$$
\begin{aligned}
\mathscr{W}_{\mathrm{NN}}(\mathbf{x}(0), t_k) &= \mathscr{D}^{i}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{j} \mathscr{P}^{i}_{\mathrm{NN}} \circ \bigcirc_{w=1}^{i-1} \left(\mathscr{T}^{w}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^{w}_{\mathrm{NN}} \right) \circ \mathscr{E}^{1}_{\mathrm{NN}} \\
&= \mathscr{D}^{i}_{\mathrm{NN}} \circ \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\overline{\mathscr{T}^{s}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\overline{\boldsymbol{T}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right] \circ \mathscr{E}^{1}_{\mathrm{NN}} \\
&= \overline{\mathscr{D}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{j} \overline{\mathscr{P}^{i'}_{\mathrm{NN}}} \circ \bigcirc_{s=1}^{i'-1} \left[\overline{\mathscr{T}^{s}_{\mathrm{NN}}} \circ \bigcirc_{\ell=1}^{\overline{\boldsymbol{T}_s}} \overline{\mathscr{P}^{s}_{\mathrm{NN}}} \right] \circ \overline{\mathscr{E}^{1}_{\mathrm{NN}}} \\
&= \overline{\mathscr{W}_{\mathrm{NN}}}(\mathbf{x}(0), t_k).
\end{aligned}
$$

Now note that the size of all autoencoders, all propagators, and $\mathsf{S}-1$ transcoders of $\mathscr{W}_{\mathrm{NN}}$ are exactly equal to the size of (corresponding) components in $\overline{\mathscr{W}_{\mathrm{NN}}}$. For the $\mathsf{W}-\mathsf{S}$ identity transcoders of $\mathscr{W}_{\mathrm{NN}}$, we can use any network size as a result of Proposition A.1. This completes the proof. ∎

B.3. Proof of Theorem 3.10. In this section, we prove Theorem 3.10. First, we prove a version of Lemma B.2 but without the dynamics assumption.

LEMMA B.4 (Propagator Approximation in General Case). *Let $\mathscr{P}_* : \mathcal{Z}([0,T]) \times [0,T] \to \mathcal{Z}([0,T])$ be a Lipschitz function such that*

- *For all $x \in \mathcal{Z}([0,T])$, $\mathscr{P}_*(x,0) = x$,*
- *For all $x \in \mathcal{Z}([0,T]), t \in [0,T], s \in [0,T-t]$, $\mathscr{P}_*(\mathscr{P}_*(x,t),s) = \mathscr{P}_*(x,t+s)$.*

Let $0 = t_1 < t_2 < \cdots < t_{\mathsf{T}} = T$ be a grid for $[0,T]$ of T points, and $\epsilon > 0$. Then there is a function $\mathscr{P}_{\mathrm{NN}} \in \mathcal{F}_{\mathrm{NN}}(d+1, d+1, L, W)$ with parameters

$$
L = O\left(\mathsf{T} \log\left(\frac{\mathsf{T}}{\epsilon} \right) \right), \quad W = O\left(\left(\frac{\mathsf{T}}{\epsilon} \right)^{d} \right),
$$

such that for all $k \in [\mathsf{T}-1]$ and $i \in [\mathsf{T}-k]$, we have

$$
\sup_{\mathbf{z}(t_k) \in \mathcal{Z}(t_k)} \| \left(\bigcirc_{j=1}^{i} \mathscr{P}_{\mathrm{NN}} \right)(\mathbf{z}(t_k)) - \mathscr{P}_*(\mathbf{z}(t_k), t_{k+i} - t_k) \|_{\mathbb{R}^{d+1}} \leq \varepsilon.
$$

Moreover, the range of $\mathscr{P}_{\mathrm{NN}}$ is contained in $\mathcal{Z}([0,1])$. The constants hidden in O depend on d and $\mathrm{Lip}_*(\mathscr{P}) = \sup_{j \in [\mathsf{T}-1], i \leq \mathsf{T}-j} \mathrm{Lip}_{\mathcal{Z}(t_j)}(\mathscr{P}_*(\cdot, t_{j+i} - t_j))$.

Proof of Lemma B.4. For all $j \in [\mathsf{T}-1]$, define $p_*^j : [0,1]^d \to [0,1]^d$ by $p_*^j(\mathbf{v}) = [\mathscr{P}_*((\mathbf{v}, t_j), t_{j+1} - t_j)]_{1,\dots,d}$ for all $\mathbf{v} \in [0,1]^d$. In other words, p_*^j returns the first d-components (i.e. without time) of the code propagated from time t_j to t_{j+1}, and note that this is Lipschitz.

By Proposition A.5, there is a neural network $p_{\text{NN}}^j \in \mathcal{F}_{\text{NN}}(d, d, L_j, W_j)$ such that $\|p_{\text{NN}}^j - p_*^j\|_{L^\infty([0,1]^d)} < \frac{\epsilon}{(\mathsf{T}-1)\text{Lip}(\mathscr{P}_*)}$, with $O\left(\log\left(\frac{\mathsf{T}}{\epsilon}\right)\right)$ layers and width $O\left(\left(\frac{\mathsf{T}}{\epsilon}\right)^d\right)$. Moreover, the range of of p_{NN}^j can be restricted to $[0,1]^d$ by Proposition A.2(Part 2).

For all $k \in [\mathsf{T}-1]$ and $i \in [\mathsf{T}-k]$, note that $\bigcirc_{j=k}^{k+i-1} p_*^j = \bigcirc_{j=k}^{k+i-1}(\mathscr{P}_*(\cdot, t_{j+1} - t_j)) = \mathscr{P}_*(\cdot, t_{k+i} - t_k)$. By Proposition A.6, we have

$$\begin{aligned}
&\sup_{\mathbf{v}\in[0,1]^d} \left\| \left(\bigcirc_{j=k}^{k+i-1} p_{\text{NN}}^j\right)(\mathbf{v}) - \mathscr{P}_*((\mathbf{v}, t_k), t_{k+i} - t_k) \right\| \\
&\quad \le \sum_{j=k}^{k+i-1} \text{Lip}_{\mathcal{Z}(t_{j+1})}\left(\bigcirc_{\ell=j+1}^{k+i-1} \mathscr{P}_*(\cdot, t_{\ell+1} - t_\ell)\right) \|p_{\text{NN}}^j - p_*^j\|_{L^\infty([0,1]^d)} \\
&\quad = \sum_{j=k}^{k+i-1} \text{Lip}_{\mathcal{Z}(t_{j+1})}\left(\mathscr{P}_*(\cdot, t_{k+i} - t_{j+1})\right) \|p_{\text{NN}}^j - p_*^j\|_{L^\infty([0,1]^d)} \\
&\quad \le \sum_{j=k}^{k+i-1} \frac{\text{Lip}_{\mathcal{Z}(t_{j+1})}(\mathscr{P}_*(\cdot, t_{k+i} - t_{j+1}))}{(\mathsf{T}-1)\text{Lip}(\mathscr{P}_*)}\epsilon \le \sum_{j=k}^{k+i-1} \frac{\epsilon}{\mathsf{T}} = \frac{i}{\mathsf{T}}\epsilon < \epsilon,
\end{aligned}$$

We finish the proof by constructing a single neural network, denoted $\mathscr{P}_{\text{NN}}$, which will exactly represent each of the p_{NN}^j, selecting the correct network based on a time parameter. For all $j \in [\mathsf{T}-1]$, let $\delta_j = \min\{|t_{j+1} - t_j|, |t_j - t_{j-1}|\}$ (and $\delta_1 = |t_2 - t_1|$) represent the distance between the time point t_j and the closest other time point.

We will use the notation $\mathbf{v}$ to represent the first d-dimensions of latent code in $\mathcal{Z}([a,b])$, and we define the function $\tilde{p}_{\text{NN}}^j : [0,1]^d \times [0,T] \to [0,1]^d$ (where we recall that σ is the ReLU function)

$$\tilde{p}_{\text{NN}}^j((\mathbf{v}, t)) := \sigma\left(\left(p_{\text{NN}}^j((\mathbf{v}, t)), t_j\right) - \frac{\sqrt{d}}{\delta_j}(t - t_j)\mathbf{1}_{d+1}\right).$$

Here, $\mathbf{1}_{d+1} \in \mathbb{R}^{d+1}$ is the vector of all 1s. Note that $\tilde{p}_{\text{NN}}^j \in \mathcal{F}_{\text{NN}}(d+1, d+1, 1 + L_j, W_j)$, and for all $\mathbf{v} \in [0,1]^d$ and $k \in [\mathsf{T}-1]$.

$$\tilde{p}_{\text{NN}}^j((\mathbf{v}, t_k)) = \begin{cases} \left(p_{\text{NN}}^k((\mathbf{v}, t_k)), t_{k+1}\right), & j = k, \\ \mathbf{0}_{d+1}, & j \neq k. \end{cases}$$

Finally, we define the propagator network for all $\mathbf{z} \in \mathbb{R}^{d+1}$ as

$$\mathscr{P}_{\text{NN}}(\mathbf{z}) = \sum_{j=1}^{\mathsf{T}} \tilde{p}_{\text{NN}}^j(\mathbf{z}).$$

Then for all $k \in [\mathsf{T}-1]$, we have that $\mathscr{P}_{\text{NN}}((\mathbf{v}, t_k)) = p_{\text{NN}}^k(\mathbf{z})$. In addition, we can

see that for all $j \in [\mathsf{T}-1]$, we have

$$\begin{aligned}
\left(\bigcirc_{j=1}^{k} \mathscr{P}_{\mathrm{NN}}\right)(\mathbf{v}, t_1) &= \left(\bigcirc_{j=1}^{k-1} \mathscr{P}_{\mathrm{NN}}\right)(\tilde{p}^1_{\mathrm{NN}}(\mathbf{v}, t_1)) \\
&= \left(\bigcirc_{j=1}^{k-1} \mathscr{P}_{\mathrm{NN}}\right)(p^1_{\mathrm{NN}}(\mathbf{v}), t_2) \\
&= \left(\bigcirc_{j=1}^{k-2} \mathscr{P}_{\mathrm{NN}}\right)(\tilde{p}^2_{\mathrm{NN}}(p^1_{\mathrm{NN}}(\mathbf{v}), t_2)) \\
&= \left(\bigcirc_{j=1}^{k-2} \mathscr{P}_{\mathrm{NN}}\right)(p^2_{\mathrm{NN}}(p^1_{\mathrm{NN}}(\mathbf{v})), t_3) \\
&= \cdots \\
&= p^k_{\mathrm{NN}}(p^{k-1}_{\mathrm{NN}}(\cdots(p^1_{\mathrm{NN}}(\mathbf{v}))), t_{k+1}).
\end{aligned}$$

We finish by computing the size of $\mathscr{P}_{\mathrm{NN}}$. According to Proposition A.2 Part 3, the number of layers of $\mathscr{P}_{\mathrm{NN}}$ is the sum of the number of layers of each $\tilde{p}_{\mathrm{NN}}$, so it is $\sum_{t=j}^{\mathsf{T}} \left(1 + O\left(\log\left(\frac{\mathsf{T}}{\epsilon}\right)\right)\right) = O\left(\mathsf{T}\log\left(\frac{\mathsf{T}}{\epsilon}\right)\right)$, and the width is given by $d + 1 + 2\max_j O\left(\left(\frac{\mathsf{T}}{\epsilon}\right)^d\right) = O\left(\left(\frac{\mathsf{T}}{\epsilon}\right)^d\right)$. ∎

Now we prove Theorem 3.10. In the proof, we first derive a result analogous to Lemma 3.7, and then apply the argument in Theorem 3.9.

Proof of Theorem 3.10. We first assume that $\mathsf{W} = \mathsf{S}$, i.e. the windows are equal to the segments. This is done by following proof of Lemma 3.7, but using Lemma B.4 instead of Lemma B.2 to construct $\mathscr{P}_{\mathrm{NN}}$. We use the exact same construction as Lemma 3.7 for the autoencoders and transcoders.

For all $w \in [\mathsf{W}]$, consider the oracle propagator $\mathscr{P}^w_*(x,t) = \mathscr{E}^w_*(\mathscr{F}(\mathscr{D}^w_*(x), t))$. By Lemma B.4, there is a neural network $\mathscr{P}^w_{\mathrm{NN}}$ with $O\left(\mathbf{T}_w \log\left(\frac{\mathsf{W}\mathbf{T}_w}{\epsilon}\right)\right)$ layers and width $O\left(\left(\frac{\mathsf{W}\mathbf{T}_w}{\epsilon}\right)^d\right)$ such that

$$\| \bigcirc_{\ell=1}^{\boldsymbol{T}_w} \mathscr{P}^w_{\mathrm{NN}} - \bigcirc_{\ell=1}^{\mathbf{T}_w} \mathscr{P}^w_*(\cdot, t^w_{\ell+1} - t^w_\ell)\|_{L^\infty(\mathcal{Z}(a_w))} < \frac{\epsilon}{(2\mathsf{W}+1)\mathrm{Lip}(\mathscr{D})\mathrm{Lip}(\mathscr{F})}.$$

The composition above makes sense, since the range of $\mathscr{P}^w_{\mathrm{NN}}$ is contained in $\mathcal{Z}([s_w, s_{w+1}])$ by construction. This establishes the result for the case that $\mathsf{W} = \mathsf{S}$. To handle the case that $\mathsf{W} > \mathsf{S}$, we can use the same argument as in Theorem 3.9 to complete the proof. ∎

Appendix C. Comparison Model Details.

C.1. LDNet. We implemented LDNet which was proposed in [43], but we modified the architecture to be grid-dependent. Specifically, since WeldNet has inputs and outputs on a fixed grid, for a more direct comparison we implemented a grid-dependent LDNet such that the reconstruction network has to predict the values on the grid. The original LDNet implementation uses a reconstruction network that inputs the query location and outputs the value of the output field at that location. In other words, we would have to call the reconstruction network D times to output values on a size D grid. Training the grid based LDNet implementation is subsequently much faster and requires much lower computational resources (such as memory); the original LDNet implementation has significant memory requirements for our data. We found that the original model sizes used in [43] were too small to perform well on our test problems, so we increased the network size to be comparable to the models in LDNet (e.g. width 500 networks), but this lead to a significant memory requirement.

We present a comparison between the original LDNet implementation and the grid based LDNet model. The original model was trained on an NVIDIA A100 GPU

	Original-LDNet	Grid-LDNet
bscale	3.81% / 5.12%	0.78% / 1.36%
bshift	6.13% / 6.87%	10.4% / 6.38%
tscale	4.12% / 4.90%	4.72% / 8.29%
tshift	0.75% / 0.89%	5.98% / 10.3%
kscale	>100% / >100%	11.1% / 22.7%
kshift	0.95% / 1.89%	0.43% / 0.72%

TABLE 3

Comparison of middle and final time relative test errors for original LDNet (with 101 time steps) and grid LDNet (with 301 time steps). Each cell is formatted as MiddleError / FinalError.

with 80 GBs of memory, while the grid based model was trained on an NVIDIA RTX6000 (i.e. we train with the same resources as WeldNet and other models). Due to memory limitations, we trained the original LDNet using data on a time grid that was **three** times as coarse as the time used for grid LDNet. With all of these changes, the total training time on the Burgers' scale dataset (with initial conditions from (4.6)) for the original LDNet model is 1335.3s, while the grid LDNet took 1343.6s to train (on a dataset with 3x as many time steps).

Table 3 shows a comparison of the midway (i.e. middle of the time grid) and final time test errors between the original LDNet and the grid LDNet models. We used the tanh activation function for the reconstruction network for the Burgers' and KdV examples, but we used a ReLU activation for the transport examples. Superior performance of the original LDNet model is observed on the transport examples, but it is matched or outperformed with the grid LDNet. Recall that the original LDNet is trained to predict the evolution of 101 time steps but the grid LDNet is trained to predict 301 time steps. Clearly, a grid based LDNet performs roughly similarly or better to the original LDNet implementation for most of our problems using much less computational resources. We note that the original LDNet implementation has the advantage of being grid independent, so the output predictions can be queried at all locations regardless of the grid it was trained on. This is also advantageous for some datasets such as for the two-hat-shaped initial conditions we used for the transport equation. However, for our purpose of grid based surrogate modeling, a grid based LDNet is faster to train and performs better on lower resources (and on a more fine time grid), so we use that for comparison in the rest of the paper.

C.2. DeepONet. Latent-DON consists of an autoencoder (which is identical to the autoencoder used in a one-window WeldNet model) and a DeepONet that predicts the evolution of a latent code from its initial time to a given future time. DeepONet is an operator learning architecture [32] that can be used in this case as time can be considered as the "input" to the operator. Specifically, a DeepONet is a function of the form (for some $p \in \mathbb{N}$):

$$\phi(\vec{z})(t) = \tau_{\mathrm{NN}}(t) \cdot B_{\mathrm{NN}}(\vec{z}) \tag{C.1}$$

Here, B_{NN} is a neural network with input dimension k (i.e. the latent space dimension) and output dimension pk which is then reshaped to be dimension $p \times k$. On the other hand, τ_{NN} is a neural network with input dimension 1 and output dimension p, so the product in (C.1) outputs a vector of dimension k.

This is the formulation used in [24]. We implement it in Pytorch such that all components of latent DeepONet (i.e. encoder, decoder, branch net, trunk net) are

	Weld-1	Weld-2	Weld-2*	Weld-4	Weld-4*
Time	668.5s	547.7s	373.2s	528.1s	267.6s

TABLE 4

Total training time for each model on $\mathcal{M}_{bscale}$ data for WeldNet models. $*$ indicates training with multiple GPUs.

	LDON	LDNet	Time Input	HDP	WLaSDI
Time	65.2s	1355.3s	52.0s	56.1s	2285s

TABLE 5

Total training time for each model on $\mathcal{M}_{bscale}$ data for comparison models.

width 400 and depth 3 feedforward ReLU networks. We use $p = 10$ for the Latent-DON comparison in this work.

Appendix D. Error Tables. For each dataset, we compute the relative test errors at different times and show them in Table 6-12. The best model error in the final time is bolded. A dash "–" indicates a relative test over of 10 or higher.

We also display the total training time for each model on the Burgers' scale dataset in Table 4 for WeldNet models and Table 5 for comparison models. The total training times for other models are similar. We use NVIDIA RTX 6000 GPUs. For WeldNet-2 and WeldNet-4, we use one GPU per window since the training of each window's models is completely independent from the other windows, but we use only one GPU for other models (there is no easy way to distribute the training among multiple GPUs for those models).

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-1	0.0091	0.0095	0.0111	0.0133	0.0145	0.0178	0.0220	0.0239	0.0241	0.0343
FF-Weld-2	0.0068	0.0045	0.0050	0.0053	0.0058	0.0122	0.0088	0.0089	0.0100	0.0101
FF-Weld-4	0.0063	0.0071	0.0059	0.0071	0.0175	0.0073	0.0081	0.0082	0.0084	0.0137
PCA-WeldNet-1	0.1875	0.1177	0.1078	0.1176	0.1307	0.1561	0.1798	0.1990	0.2174	0.2404
PCA-WeldNet-2	0.1077	0.0420	0.0656	0.0578	0.0909	0.1675	0.1408	0.1204	0.1220	0.1551
PCA-WeldNet-4	0.0445	0.0254	0.0292	0.0524	0.0440	0.1145	0.0885	0.1223	0.1213	0.1133
Conv-Weld-1	0.0099	0.0070	0.0064	0.0065	0.0074	0.0072	0.0076	0.0076	0.0080	0.0091
Conv-Weld-2	0.0090	0.0056	0.0055	0.0064	0.0066	0.0121	0.0082	0.0084	0.0090	0.0094
Conv-Weld-4	0.0159	0.0119	0.0101	0.0067	0.0069	0.0089	0.0063	0.0074	0.0057	**0.0060**
Latent-DON	0.0059	0.0065	0.0069	0.0072	0.0078	0.0083	0.0092	0.0091	0.0092	0.0153
LDNet	0.0440	0.0373	0.0370	0.0375	0.0381	0.0374	0.0369	0.0360	0.0388	0.0512
Grid-LDNet	0.0063	0.0070	0.0072	0.0071	0.0078	0.0083	0.0097	0.0097	0.0095	0.0136
TimeInput	0.0258	0.0135	0.0116	0.0126	0.0123	0.0126	0.0145	0.0159	0.0178	0.0176
HDP	0.0209	0.0313	0.0403	0.0553	0.0825	0.0967	0.1116	0.1313	0.1323	0.1433
WLaSDI	1.045	1.042	1.055	1.072	1.097	1.137	1.192	1.263	1.359	1.486

TABLE 6
Test Errors for Burgers' Scale

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-2	0.0232	0.0181	0.0246	0.0427	0.0455	0.0484	0.0363	0.0287	0.0247	0.0242
FF-Weld-4	0.0151	0.0364	0.0570	0.0669	0.0623	0.0557	0.0492	0.0536	0.0468	0.0472
FF-AENet	0.0218	0.0310	0.0468	0.0542	0.0444	0.0364	0.0312	0.0284	0.0289	0.0340
Conv-Weld-2	0.0245	0.0147	0.0170	0.0202	0.0205	0.0252	0.0178	0.0155	0.0171	**0.0201**
Conv-Weld-4	0.0199	0.0116	0.0153	0.0237	0.0278	0.0295	0.0229	0.0253	0.0227	0.0205
Conv-AENet	0.0328	0.0204	0.0291	0.0442	0.0554	0.0627	0.0738	0.0867	0.0998	0.1119
PCA-WeldNet-2	0.1500	0.1628	0.3493	0.4804	0.4964	0.4950	0.4884	0.4806	0.4727	0.4651
PCA-WeldNet-4	0.1484	0.1568	0.3466	0.4830	0.4980	0.4949	0.4882	0.4805	0.4736	0.4719
PCA-AENet	0.1521	0.2058	0.3729	0.4967	0.5127	0.5107	0.5060	0.5009	0.4983	0.5021
Latent-DON	0.0209	0.0291	0.0400	0.0460	0.0386	0.0320	0.0272	0.0248	0.0261	0.0318
LDNet	0.0254	0.0399	0.1043	0.1158	0.1041	0.0919	0.0811	0.0722	0.0657	0.0638
TimeInput	0.0291	0.0222	0.0323	0.0548	0.0550	0.0451	0.0366	0.0303	0.0267	0.0265
HDP	0.2155	0.4972	0.8060	1.2015	1.7128	2.3725	3.2132	4.2975	5.6602	7.3479
WLaSDI	0.782	0.839	0.854	0.835	0.844	0.910	0.978	1.045	1.199	1.380

TABLE 7
Test Errors for Burgers' Shift

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-2	0.0095	0.0125	0.0103	0.0139	0.0120	0.0085	0.0140	0.0118	0.0113	0.0142
FF-Weld-4	0.0136	0.0112	0.0116	0.0141	0.0125	0.0160	0.0135	0.0094	0.0141	**0.0102**
FF-AENet	0.0215	0.0204	0.0223	0.0196	0.0195	0.0224	0.0245	0.0217	0.0249	0.0362
Conv-Weld-2	0.0221	0.0234	0.0235	0.0248	0.0181	0.0198	0.0248	0.0250	0.0225	0.0268
Conv-Weld-4	0.0168	0.0163	0.0155	0.0212	0.0202	0.0248	0.0311	0.0292	0.0190	0.0203
Conv-AENet	0.0298	0.0427	0.0346	0.0516	0.0445	0.0439	0.0271	0.0324	0.0300	0.0234
PCA-WeldNet-2	0.5815	0.4339	0.4880	0.5128	0.5090	0.5813	0.4336	0.4875	0.5101	0.5010
PCA-WeldNet-4	0.2867	0.1783	0.2379	0.2582	0.1697	0.2860	0.1790	0.2388	0.2571	0.1705
PCA-AENet	0.8214	0.5968	0.5885	0.6678	0.6158	0.6106	0.6205	0.6837	0.6580	0.5872
Latent-DON	0.0156	0.0155	0.0218	0.0217	0.0165	0.0193	0.0174	0.0204	0.0207	0.0318
LDNet	0.0359	0.0416	0.0492	0.0459	0.0472	0.0476	0.0453	0.0504	0.0463	0.0829
TimeInput	0.0580	0.0668	0.0591	0.0669	0.0679	0.0610	0.0640	0.0703	0.0661	0.0921
HDP	0.0189	0.0348	0.0767	0.1062	0.1366	0.1480	0.1629	0.1946	0.1881	0.0697
WLaSDI	1.210	1.671	1.521	3.540	9.388	5.028	7.136	–	–	–

TABLE 8
Test Errors for Transport Scale

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-2	0.0283	0.0295	0.0276	0.0308	0.0305	0.0405	0.1727	0.1273	0.0631	0.0599
FF-Weld-4	0.0310	0.0297	0.0353	0.0372	0.0432	0.0357	0.0374	0.0457	0.0559	0.0587
FF-AENet	0.7640	0.8940	0.9187	0.9386	0.9492	0.9243	0.8009	0.8958	0.8003	1.1303
Conv-Weld-2	0.0207	0.0186	0.0168	0.0189	0.0187	0.0208	0.0213	0.0221	0.0188	0.0210
Conv-Weld-4	0.0173	0.0151	0.0154	0.0186	0.0183	0.0180	0.0213	0.0231	0.0200	0.0227
Conv-AENet	0.0312	0.0279	0.0330	0.0293	0.0276	0.0275	0.0254	0.0320	0.0321	0.0319
PCA-WeldNet-2	0.7819	0.7177	0.6850	0.6988	0.7210	0.7889	0.7401	0.6998	0.7150	0.7326
PCA-WeldNet-4	0.6773	0.6055	0.5905	0.6087	0.6159	0.6777	0.6055	0.5906	0.6095	0.6176
PCA-AENet	0.8650	0.8727	0.8775	0.9071	0.8685	0.8806	0.8619	0.8546	0.8281	0.8190
Latent-DON	0.1120	0.1088	0.1519	0.1396	0.1161	0.1438	0.1252	0.1412	0.1473	0.3016
LDNet	0.0503	0.0521	0.0565	0.0589	0.0598	0.0616	0.0609	0.0618	0.0664	0.1029
TimeInput	0.1435	0.1065	0.0966	0.1035	0.1085	0.1059	0.1188	0.1048	0.0955	0.1031
HDP	0.3034	2.3470	5.1519	8.4939	–	–	–	–	–	–
WLaSDI	6.216	–	–	–	–	–	–	–	–	–

TABLE 9
Test Errors for Transport Shift

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-2	0.0160	0.0249	0.0329	0.0273	0.0281	0.0277	0.0366	0.0332	0.0338	0.0371
FF-Weld-4	0.0096	0.0101	0.0127	0.0128	0.0240	0.0166	0.0187	0.0216	0.0272	0.0395
FF-AENet	0.0795	0.1304	0.1255	0.1307	0.1696	0.1798	0.2025	0.2515	0.4120	0.6350
Conv-Weld-2	0.0261	0.0231	0.0225	0.0235	0.0240	0.0232	0.0288	0.0283	0.0308	0.0316
Conv-Weld-4	0.0106	0.0112	0.0128	0.0173	0.0186	0.0227	0.0206	0.0234	0.0283	**0.0317**
Conv-AENet	0.0187	0.0514	0.0864	0.1098	0.1423	0.1818	0.2392	0.2971	0.3628	0.3851
PCA-WeldNet-2	0.5672	0.3454	0.3709	0.3762	0.4345	0.6621	0.6137	0.5912	0.5849	0.6024
PCA-WeldNet-4	0.3723	0.2473	0.2717	0.5046	0.4217	0.5999	0.5589	0.5352	0.6763	0.6782
PCA-AENet	0.6709	0.4938	0.4819	0.4796	0.4769	0.4968	0.5399	0.5860	0.6234	0.6611
Latent-DON	0.2379	0.1528	0.1858	0.1949	0.1685	0.1378	0.1225	0.1438	0.1914	0.3889
LDNet	0.0735	0.0793	0.0890	0.1011	0.1105	0.1262	0.1393	0.1550	0.1734	0.2272
TimeInput	0.1673	0.1505	0.1362	0.1506	0.1570	0.1700	0.1767	0.1878	0.2060	0.2472
HDP	0.0770	0.1302	0.1688	0.2621	0.3905	0.6109	0.8120	0.9843	1.0881	1.1590
WLaSDI	0.993	1.417	2.495	5.950	–	–	–	–	–	–

TABLE 10
Test Errors for KdV Scale

Model	30	60	90	120	150	180	210	240	270	300
FF-Weld-2	0.0028	0.0027	0.0028	0.0025	0.0027	0.0031	0.0030	0.0030	0.0030	**0.0028**
FF-Weld-4	0.0026	0.0028	0.0028	0.0027	0.0041	0.0030	0.0032	0.0033	0.0032	0.0046
FF-AENet	0.0071	0.0069	0.0065	0.0069	0.0063	0.0061	0.0058	0.0051	0.0053	0.0087
Conv-Weld-2	0.0044	0.0034	0.0036	0.0032	0.0036	0.0042	0.0034	0.0031	0.0031	0.0032
Conv-Weld-4	0.0040	0.0030	0.0031	0.0034	0.0031	0.0038	0.0030	0.0030	0.0028	**0.0028**
Conv-AENet	0.0050	0.0053	0.0046	0.0042	0.0046	0.0042	0.0039	0.0046	0.0047	0.0053
PCA-WeldNet-2	0.0444	0.0233	0.0192	0.0215	0.0230	0.0482	0.0210	0.0177	0.0233	0.0186
PCA-WeldNet-4	0.0199	0.0119	0.0055	0.0051	0.0098	0.0168	0.0105	0.0057	0.0059	0.0090
PCA-AENet	0.1276	0.0832	0.0672	0.0551	0.0476	0.0542	0.0629	0.0587	0.0473	0.0628
Latent-DON	0.0047	0.0039	0.0044	0.0044	0.0035	0.0039	0.0038	0.0032	0.0032	0.0051
LDNet	0.0049	0.0041	0.0044	0.0045	0.0043	0.0043	0.0042	0.0038	0.0037	0.0072
TimeInput	0.0130	0.0071	0.0081	0.0099	0.0067	0.0074	0.0074	0.0064	0.0073	0.0071
HDP	0.0115	0.0226	0.0212	0.0253	0.0273	0.0321	0.0319	0.0332	0.0360	0.0426
WLaSDI	0.464	0.463	0.476	0.499	0.521	0.520	0.523	0.520	0.505	0.497

TABLE 11
Test Errors for KdV Shift

Model	10	20	30	40	50	60	70	80	90	100
FF-Weld-2	0.0051	0.0048	0.0043	0.0042	0.0041	0.0030	0.0030	0.0030	0.0032	0.0034
FF-Weld-4	0.0051	0.0045	0.0037	0.0034	0.0031	0.0027	0.0026	0.0027	0.0026	0.0028
FF-AENet	0.0064	0.0056	0.0051	0.0051	0.0045	0.0047	0.0045	0.0042	0.0036	0.0043
Latent-DON	0.0080	0.0063	0.0052	0.0046	0.0044	0.0047	0.0047	0.0049	0.0045	0.0061
LDNet	0.0086	0.0074	0.0072	0.0072	0.0066	0.0065	0.0063	0.0063	0.0073	0.0095
TimeInput	0.0155	0.0129	0.0121	0.0120	0.0118	0.0119	0.0121	0.0115	0.0140	0.0194
HDP	0.0389	0.0312	0.0298	0.0294	0.0274	0.0262	0.0267	0.0269	0.0250	0.0316

TABLE 12
Test Errors for Shallow Water