On Learning the Dynamical Response of Nonlinear Control Systems with Deep Operator Networks

Guang Lin, Christian Moya, and Zecheng Zhang

June 15, 2022

Abstract
We propose a Deep Operator Network (DeepONet) framework to learn the dynamic response of continuous-time nonlinear control systems from data. To this end, we first construct and train a DeepONet that approximates the control system’s local solution operator. Then, we design a numerical scheme that recursively uses the trained DeepONet to simulate the control system’s long/medium-term dynamic response for given control inputs and initial conditions. We accompany the proposed scheme with an estimate for the error bound of the associated cumulative error. Furthermore, we design a data-driven Runge-Kutta (RK) explicit scheme that uses the DeepONet forward pass and automatic differentiation to better approximate the system’s response when the numerical scheme’s step size is sufficiently small. Numerical experiments on the predator-prey, pendulum, and cart pole systems confirm that our DeepONet framework learns to approximate the dynamic response of nonlinear control systems effectively.

1 Introduction
High-fidelity numerical schemes are the prevalent computational tools for simulating and predicting complex dynamical systems, such as climate modeling, robotics, or the modern power grid. However, these numerical schemes may be prohibitively expensive for control, optimization, and uncertainty quantification tasks. These tasks often require a large number of forward simulations, which consume considerable computational resources. As a result, there is growing interest in developing tools that can accelerate the numerical simulation of complex dynamical systems without compromising accuracy.

By providing faster alternatives to traditional numerical schemes, machine learning-based computational tools hold the promise to accelerate the rate of innovation for complex dynamical systems. Hence, a recent wave of machine learning-based tools has demonstrated the potential of using observational data to construct fast surrogates of complex systems [11, 25]. These tools aim at (1) learning the governing equations of a dynamical system or (2) learning to predict the system’s response from data.

On the one hand, several works [3, 4, 29, 30] use observational data to discover the unknown governing equations of the underlying system. For example, Bruton et al. [3] proposed identifying nonlinear systems using sparse schemes and a large set of dictionaries. The authors

---

*The authors are listed alphabetically.
†G. Lin is with the Department of Mathematics and Mechanical Engineering, Purdue University, West Lafayette, IN, 47907 USA, e-mail: guanglin@purdue.edu
‡C. Moya and Z. Zhang are with the Department of Mathematics, Purdue University, West Lafayette, IN, 47907 USA, Christian e-mail: cmoyacal@purdue.edu, Zecheng email: zecheng.zhang.math@gmail.com
then extended their work in [4] to identify input/output mappings, describing control systems.

In [30], the authors used sparse approximation schemes to recover the governing partial or ordinary differential equations describing unknown systems. On the other hand, there is growing interest in learning to predict the next state response of a dynamical system using time-series data [26, 27, 25, 24]. For instance, the authors in [11, 7] trained a transformer with data from the early stages of the time dependent partial differential equations to predict the solution recursively of the future stages. Qin et al. [26] used a recurrent residual neural network (ResNet) to approximate the mapping from the current state to the next state of an unknown autonomous system. Similarly, in [27], the authors used feed-forward neural networks (FNN) as a building block of a multi-step scheme that predicts the response of an autonomous system. In [25], the authors extended their previous work [26] to nonautonomous systems with time-dependent inputs. To this end, they parametrized the input locally within small time intervals.

Most of the works mentioned above require large amounts of training data to avoid overfitting. However, obtaining this data may be prohibitively expensive for complex dynamical systems. Furthermore, for next state methods using traditional neural networks, the predicted response may drift and accumulate errors if one fails to avoid overfitting. Thus it becomes imperative to derive efficient deep learning-based frameworks that can handle the infinite-dimensional nature of the problem of predicting the response of the control systems studied in this paper.

Recently, in the seminal paper [21], the authors proposed the Deep Operator Network (DeepONet) framework to learn nonlinear operators (i.e., mappings from a function space to another function space). They designed DeepONet based on the universal approximation theorem of operators [6], which states that nonlinear operators (e.g., the solution operator of a nonlinear control system) can be approximated using streams of scattered data. Compared to traditional neural networks, DeepONet exhibits small generalization error and learns with a limited amount of training data, as demonstrated in many applications areas, such as power engineering [22], multi-physics problems [5], and turbulent combustion problems [28]. Extensions to the original DeepONet [21] have enabled incorporating physics-informed models [34], handling noisy data [18], or designing novel optimization methods [18, 19]. As a result, in this paper, we focus on extending the original DeepONet framework to learn the solution operator of control systems. Such a data-driven operator framework can then be used to design control policies for continuous control systems.

In particular, one of our motivations behind learning to approximate the solution operator of a nonlinear control system is its application in Model-Based Reinforcement Learning (MBRL) [35, 31]. In MBRL, one learns to approximate the system to control (from data) and then uses the learned model to seek for an optimal policy without extensive interaction with the actual system. A common approach is to learn a discrete-time forward model that predicts the next state using the current state and selected control action. Such an MBRL framework has delivered successful and efficient results in discrete-time problems such as games [16]. However, most control systems in science and engineering (e.g., robotics [23], unmanned vehicles [14], or laminar flows [12]) are continuous. Of course, one can always discretize the continuous dynamics and apply discrete-time MBRL using traditional neural networks (e.g., see [2]). However, if one fails to handle the inherent epistemic uncertainty [9], such a strategy may lead to error accumulation (due to model bias) and poor asymptotic performance. To alleviate these drawbacks, we build on the original DeepONet [21] to design an effective and efficient framework that can learn the solution operator of a nonlinear control system, which one can then apply within the framework of continuous MBRL [10].

Formally, the objectives of this paper are twofold.

1. **Approximation of the local solution operator:** We aim at deriving a deep learning-based
framework that learns to map (1) the current state of the control system and (2) a local approximation of the control input to the next state of the control system.

2. **Long/Medium-term simulation:** We aim at designing an efficient scheme that uses the deep learning-based framework to simulate (over a given long/medium-term horizon) the system’s response to a control input.

Our contributions are summarized below.

1. We first design (in Section 2.1) a deep-operator network (DeepONet) framework that approximates the *local* solution operator of a control (non-autonomous) dynamical system. To this end, the proposed DeepONet first observes (1) the current state of the system and (2) a local description of the control/input signal. Then, the DeepONet predicts the next state of the system at any given resolution.

2. We then (in Section 2.2) describe a numerical scheme that uses the trained DeepONet to simulate the response of the control system over a long/medium-term horizon. The resulting scheme will effectively simulate trajectories of continuous plants (with unknown dynamics) driven by external input signals (continuous or sampled).

3. We provide (in Section 2.3) an estimate of the cumulative error for the proposed DeepONet-based numerical scheme. Our estimated error bounds are tighter than those presented in [25].

4. We also propose (in Section 3) a novel data-driven Runge-Kutta (RK) method that can improve the accuracy of the long-term simulation. To estimate the required vector-field for Runge-Kutta, we employ automatic differentiation to the forward pass of our DeepONet.

5. We then provide (in Section 3.1) a theoretical guarantee illustrating that RK has an improved cumulative error when the simulation step-size is less than a pre-specified constant.

6. Finally, we test (in Section 4) the efficacy of the proposed framework on the predator-prey system with control (also presented in [25]) and the pendulum and cart-pole systems, which are widely used by the Reinforcement Learning community [2].

We organize the rest of this paper as follows. In Section 2, we formulate the problem of approximating the local solution operator of a nonlinear control system. Section 2.1 details the Deep Operator Framework (DeepONet) that approximates such an operator. The recursive scheme that simulates the nonlinear control system over a given long/medium-term horizon is detailed in Section 2.2. Then, in Section 2.3, we present the proposed scheme’s cumulative error estimate. In Section 3, we introduce a data-driven Runge-Kutta scheme and estimate its corresponding error bound. In Section 4, we test the effectiveness of the proposed scheme using a series of numerical experiments. Finally, Section 5 discusses our future work and concludes the paper.

## 2 Problem Formulation

We consider the problem of learning from data the solution operator of the continuous-time nonlinear control system

$$\frac{d}{dt} x(t) = f(x(t), u(t)), \quad t \in [a, b]$$

$$x(a) = x_0,$$

(1)
where \( x(t) \in \mathcal{X} \subseteq \mathbb{R}^n \) is the state vector, \( u(t) \in \mathcal{U} \subseteq \mathbb{R}^p \) the control input vector, and \( f : \mathcal{X} \times \mathcal{U} \to \mathcal{X} \) an unknown function. Additional assumptions on the control input function \( u \) and the function \( f \) will be discussed later. Also, throughout this paper, we assume \( u(t) \) is a scalar control input, i.e., \( p = 1 \). Extending the proposed framework to the vector-valued case is straightforward.

**Solution Operator.** Let \( \mathcal{F} \) denote the solution operator (also known as flow map) of (1). \( \mathcal{F} \) takes as inputs the initial condition \( x(a) = x_0 \in \mathcal{X} \) and the sequence of control functions \( u_{[a,t)} := \{ u(\tau) \in \mathcal{U} : \tau \in [a,t) \} \) and outputs the state \( x(t) \in \mathcal{X} \) at time \( t \in [a,b] \). We compute the solution operator via

\[
\mathcal{F}(x_0, u_{[a,t)}) (t) \equiv x(t) = x_0 + \int_a^t f(x(s), u(s))ds.
\] (2)

**Approximate System.** In practice, to learn the operator \( \mathcal{F} \), we only have access to a approximate/discretized representation of the control input function \( u(t) \). Let \( u_m \) denote this approximate representation, which yields the following approximate system

\[
\frac{d}{dt} \tilde{x}(t) = f(\tilde{x}(t), u_m), \quad t \in [a,b]
\]
\[
\tilde{x}(a) = x(a),
\] (3)

whose solution operator is

\[
\mathcal{F}(x_0, u_m) (t) \equiv \tilde{x}(t) = x_0 + \int_a^t f(\tilde{x}(s), u_m)ds.
\] (4)

In the above, with a slight abuse of notation, we denoted as \( u_m \) the input discretized using \( m \geq 1 \) sensors or interpolation points within the interval \( [a,b] \), and the approximate state function as \( \tilde{x}(t) \).

**Remark 1.** In practice, the approximate system (3) with solution operator (4) can represent, for example, sampled-data control systems [36] or semi-Markov Decision Processes [10]. As a result, the methods introduced in this paper can be used to design optimal control policies within the framework of model-based reinforcement learning [35].

**2.1 Learning the Solution Operator**

To learn the solution operator \( \mathcal{F} \), we use the Deep Operator Network (DeepONet) framework introduced in [21]. In [21], the authors used a DeepONet \( G_{\theta} \) to learn a simplified version of the solution operator \( \mathcal{F} \) with \( x(a) = 0 \), that is,

\[
G(u)(t) \equiv x(t) = \int_a^t f(x(s), u(s))ds, \quad t \in [a,b].
\]

The DeepONet \( G_{\theta} \) takes as inputs the (1) control \( u(t) \) discretized using \( m \) interpolation points (known as sensors in [20]) and (2) time \( t \in [a,b] \), and outputs the state \( x(t) \). Clearly, this DeepONet prediction is one-shot, that is, it requires knowledge of the input in the whole interval \([a,b]\). For small values of \( b \), the DeepONet’s \( G_{\theta} \) prediction is very accurate. However, as we increase \( b \), the accuracy deteriorates. To improve accuracy, one can increase the number of interpolation points. This, however, makes the DeepONet’s training more challenging.

To alleviate this drawback, in this paper, we take a different approach. First, we train a DeepONet \( \mathcal{F}_{\theta} \), with vector of trainable parameters \( \theta \), to learn the local solution operator of the control
Learning the Local Solution Operator. We let $\mathcal{P}$ denote the possibly irregular and arbitrary time partition

$$\mathcal{P} : a = t_0 < t_1 < \ldots < t_M = b,$$

where $h_n := t_{n+1} - t_n$ for all $n = 0, 1, \ldots, M - 1$ and let $h := \max_n h_n$. Then, within the local interval $[t_n, t_{n+1}] \equiv [t_n, t_n + h_n]$, the solution operator is given by

$$\mathcal{F}(\tilde{x}(t_n), u_m^n(h_n)) \equiv \tilde{x}(t_n + h_n)
= \tilde{x}(t_n) + \int_{t_n}^{t_n + h_n} f(\tilde{x}(s), u_m^n)ds.
$$

(5)

In the above, with a slight abuse of notation, we use $u_m^n$ to denote the local discretized representation of the control input function $u_m$, within the interval $[t_n, t_{n+1}]$, using $n_s \geq 1$ interpolation/sensor points or basis.

The DeepONet $\mathcal{F}_\theta$. We design next a Deep Operator Network (DeepONet) $\mathcal{F}_\theta$, with vector of trainable parameters $\theta$, to approximate the local solution operator $\mathcal{F}$. Figure 1 illustrates the proposed DeepONet $\mathcal{F}_\theta$ that has two neural networks: the Branch Net and the Trunk Net.

The Branch Net maps the vector that concatenates the (1) current state $\tilde{x}(t_n)$ and (2) control input function $u_m^n \in \mathbb{R}^{n_s}$, discretized using the mesh of local sensors $t_n = d_0^n < d_1^n < \ldots < d_{n_s-1} = t_{n+1}$, to the branch output feature vector $\beta \in \mathbb{R}^{nq}$. On the other hand, the Trunk Net maps the scalar step size $h_n \in (0, h]$ to the trunk output feature vector $\tau \in \mathbb{R}^{nq}$. We compute the DeepONet’s output for the $i$th component of the state vector $\tilde{x}(t_{n+1})$ using the dot product:

$$\mathcal{F}^{(i)}_\theta(\tilde{x}(t_n), u_m^n(h_n)) = \sum_{k=1}^{nq} \beta_{(i-1)q+k} \tau_{(i-1)q+k}.$$ 

Remark 2. Sensor locations. One of the problems of the original DeepONet [20] is that the sensor locations, $t_n = d_0^n < d_1^n < \ldots < d_{n_s-1} = t_{n+1}$, are fixed. If we fix these sensor locations,
Algorithm 1: DeepONet-based Numerical Scheme

1. **Require:** initial state vector $x(a) = x_0$, partition $P$, control input $u^{n_m}_n$, for $n = 1, \ldots, M - 1$, and trained DeepONet $F_{\theta^*}$.

2. initialize $\tilde{x}(t_0) = x_0$

3. for $n = 0, \ldots, M - 1$ do

4. update the independent variable $t_{n+1} = t_n + h_n$

5. update the state vector using the DeepONet’s forward pass

\[
\tilde{x}(t_{n+1}) = F_{\theta^*}(\tilde{x}(t_n), u^{n_m}_n)(h_n).
\]

6. end

7. **Return:** predicted response $\{\tilde{x}_n \equiv \tilde{x}(t_n) : t_n \in P\}$.

then we cannot predict the response of the control system using the arbitrary and irregular partition $P$. To enable predicting with $P$, we let the input to the branch $u^{n_m}_n$ be the concatenation of the discretized input $(u(d^n_0), \ldots, u(d^n_{n_s-1}))$ with the corresponding (flexible) relative sensor locations $(t_n - d^n_0, \ldots, t_n - d^n_{n_s-1})$.

**Remark 3.** The case of $n_s = 1$ sensors, i.e., the piece-wise constant approximation of $u$. If we let $n_s = 1$ sensors, then the discretized control input function, within the interval $[t_n, t_{n+1}]$, corresponds to the singleton $u^{n_m}_n \equiv u(t_n)$. Such a case is the most challenging to learn because it introduces the largest delay error that propagates over time. However, it also represents one of our target applications: continuous model-based reinforcement learning with semi-Markov decision processes. We will show (in Section 4) that our DeepONet can handle effectively having only $n_s = 1$ sensor.

**Training the DeepONet $F_{\theta}$.** We train the proposed DeepONet $F_{\theta}$ model by minimizing the loss function

\[
L(\theta; D) = \frac{1}{N} \sum_{i=1}^{N} ||x^i(t_n + h_n^i) - F_{\theta}(x^i(t_n), u^{i,n_m^i}(h_n^i))||^2
\]

gerater over of $N$ training data triplets

\[
D = \{(x^i(t_n), u^{i,n_m^i}, h_n^i, x^i(t_n + h_n^i)) \}_{i=1}^{N}
\]
generated by the unknown ground truth local solution operator $F$.

2.2 Predicting the System’s Response for Long/Medium-Term Horizons

We predict the response of the control system over a long/medium-term horizon (i.e., within the interval $[a, b]$, with $b \gg 1$) using the DeepONet-based numerical scheme detailed in Algorithm 1. Algorithm 1 takes as inputs the (1) initial condition $x(a) = x_0$, (2) partition $P$, (3) discretized representation of the control input $u^{n_m}_n$, for $n = 1, \ldots, M - 1$, and (4) trained DeepONet $F_{\theta^*}$. Then, Algorithm 1 outputs the predicted response of the control system over the partition $P$, i.e., $\{\tilde{x}_n \equiv \tilde{x}(t_n) : t_n \in P\}$. Let us conclude this section by estimating a bound for the cumulative error of the proposed DeepONet-based numerical scheme described in Algorithm 1.
2.3 Error Bound for the DeepONet-based Numerical Scheme

**Assumptions.** We let the control input function \( u \in V \subset C[a,b] \) where \( V \) is compact. We also assume the unknown vector field \( f : \mathcal{X} \times \mathcal{U} \to \mathcal{X} \) is Lipschitz in \( x \) and \( u \), i.e.,

\[
\|f(x_1, u) - f(x_2, u)\| \leq C_1\|x_1 - x_2\|, \\
\|f(x_1, u) - f(x, u_2)\| \leq C_1\|u_1 - u_2\|,
\]

where \( C_1 > 0 \) is a constant, and \( x_1, x_2, u_1, u_2 \) are in some proper space. Such an assumption is common in engineering as \( f \) is often differentiable with respect to \( x \) and \( u \). The following Lemma, presented in [25], provides us with an alternative form to the local solution operator (4).

Such a form will be used later when estimating the error bound.

**Lemma 2.1.** Consider the local solution operator \( \Phi(F(x_n), u_m^0)(h_n) \). Then, there exists a function \( \Phi : \mathbb{R}^n \times \mathbb{R}^{n_x} \times \mathbb{R} \to \mathbb{R}^n \), which depends on \( f \), such that

\[
\tilde{x}(t_{n+1}) = \Phi(F(x_n), u_m^0)(h_n) = \Phi(\tilde{x}(t), u_m^0, h_n),
\]

for any \( t_n \in \mathcal{P} \). In the above, \( u_m^0 \in \mathbb{R}^{n_x} \) locally characterizes \( u_m \) on the mesh \( t_n = d_{n-1}^0 < d_n^0 < \ldots < d_{n-1}^n = t_{n+1} \).

We now provide the error estimation of the proposed DeepONet prediction scheme detailed in Algorithm 1. For the ease of notation, we denote \( x_n = x(t_n) \) and \( \tilde{x}_n = \tilde{x}(t_n) \) for all \( t_n \in \mathcal{P} \).

**Lemma 2.2.** For any \( t_n \in \mathcal{P} \), we have

\[
\|x_n - \tilde{x}_n\| \leq \frac{1 - \tilde{C}e}{1 - C} \tilde{e}(u_m) := \tilde{C}\tilde{e}_n,
\]

where \( \tilde{C} = e^{C_1h} \), \( \tilde{e}(u_m) = \max_n \tilde{e}_n(u_m) \), and \( \tilde{e}_n(u_m) = C_1h_n\kappa_n(m)e^{C_1h_n} \).

**Proof.** Let \( a = t_n \) and \( s \in [t_n, t_{n+1}] \) in the solution operators (2) and (4). Then, subtracting (4) from (2) gives

\[
\|x(s) - \tilde{x}(s)\| \leq \|x_n - \tilde{x}_n\| + \int_{t_n}^{s} \|f(x(t), u(t)) - f(\tilde{x}(t), u_m)\|dt \\
\leq \|x_n - \tilde{x}_n\| + C_1 \int_{t_n}^{s} \|u(t) - u_m\|dt + C_1 \int_{t_n}^{s} \|x(t) - \tilde{x}(t)\|dt \\
\leq \|x_n - \tilde{x}_n\| + C_1h_n\kappa_n(m) + C_1 \int_{t_n}^{s} \|x(t) - \tilde{x}(t)\|dt,
\]

where \( \kappa_n(m) \) is the local approximation error of the control input within the interval \( [t_n, t_{n+1}] \):

\[
\max_{s \in [t_n, t_{n+1}]} |u(s) - u_m| \leq \kappa_n(m),
\]

such that

\[
\kappa_n(m) \to 0 \text{ when the number of sensors } m \to \infty.
\]

We refer the interested reader to equation (4) of [20] for details of such an approximation for the control input. Set \( s = t_{n+1} \) and apply Gronwall’s inequality, we then have,

\[
\|x_{n+1} - \tilde{x}_{n+1}\| \leq \|x_n - \tilde{x}_n\|e^{C_1h_n} + \underbrace{C_1h_n\kappa_n(m)e^{C_1h_n}}_{\tilde{e}_n(u_m)}.
\]

Taking \( \tilde{e}(u_m) = \max_n \tilde{e}_n(u_m) \) gives

\[
\|x_{n+1} - \tilde{x}_{n+1}\| \leq \|x_n - \tilde{x}_n\|e^{C_1h_n} + \tilde{e}(u_m).
\]

The bound then follows immediately due to \( x(t_0) = \tilde{x}_0 \). \( \square \)
Before we estimate the cumulative error of the DeepONet assisted solution $\tilde{x}_n$, we review the universal approximation theorem of neural network for high-dimensional functions [3]. To this end, given $h_n$, we define the following vector-valued continuous function $\varphi : \mathbb{R}^n \times \mathbb{R}^{n_k} \rightarrow \mathbb{R}^n$

$$\varphi(y_n, u_n^m) = F(y_n, u_n^m)(h_n) = \Phi(y_n, u_n^m, h_n),$$

where $y_n \in \mathbb{R}^n$. Then, by the universal approximation theorem, for $\varepsilon(u_m) > 0$, there exist $W_1 \in \mathbb{R}^{K \times (n+n_k)}, b_1 \in \mathbb{R}^K, W_2 \in \mathbb{R}^{n \times K}$ and $b_2 \in \mathbb{R}^n$ such that

$$\left\| \varphi(y_n, u_n^m) - \left( W_2 \sigma(W_1[y_n, u_n^m]_n^T + b_1) + b_2 \right) \right\| < \varepsilon(u_m). \quad (9)$$

Here, the two-layer network represents the DeepONet for a given $h_n$, i.e.,

$$\left( W_2 \sigma(W_1[y_n, u_n^m]_n^T + b_1) + b_2 \right) \equiv F_{\theta^*}(y_n, u_n^m).$$

The next Lemma then estimates the cumulative error between the DeepONet assisted solution $\tilde{x}$ (obtained via Algorithm 1) and the solution $\tilde{x}$ of the approximate system that satisfies (6).

**Lemma 2.3.** Assume $\Phi$ is Lipschitz in $x$ with Lipschitz constant $C_2$. Suppose the DeepONet is well trained so that the network satisfies (9). Then, we have following estimate:

$$\|\tilde{x}_n - \tilde{x}_n\| \leq \tilde{C} \varepsilon(u_m), \quad (10)$$

where $\tilde{C} = \frac{1-C_2^2}{1-C_2}$.

**Proof.** It follows from the universal approximation theorem of neural networks [9] and $\Phi$ being Lipschitz that,

$$\|\tilde{x}_{n+1} - \tilde{x}_{n+1}\| = \|F_{\theta^*}(\tilde{x}_n, u_n^m) - \Phi(\tilde{x}_n, u_n^m, h_n)\|$$

$$= \|F_{\theta^*}(\tilde{x}_n, u_n^m) - \varphi(\tilde{x}_n, u_n^m)\| + \|\Phi(\tilde{x}_n, u_n^m, h_n) - \Phi(\tilde{x}_n, u_n^m, h_n)\|$$

$$\leq \varepsilon(u_m) + C_2 \|\tilde{x}_n - \tilde{x}_n\|.$$

The result follows immediately from $\tilde{x}_0 = \tilde{x}_0$. 

The following theorem summarizes the error of the proposed DeepONet scheme.

**Theorem 2.4.** For any $t_n \in P$, we have

$$\|x_n - \tilde{x}_n\| \leq \tilde{C} \varepsilon(u_m) + \tilde{C} \varepsilon(u_m), \quad (11)$$

where $\tilde{C}, \tilde{C}$ and $\tilde{C}(u_m)$ are, respectively, the constants defined in [7], [10], and [7].

We conclude this section by observing that the error bound found in this section is tighter than the error bound found in [25], which behaves like $te^{ct}$ where $c$ is a positive constant.

### 3 Data-Driven Runge-Kutta (RK) Prediction Scheme

In this section, we propose a *data-driven* Runge-Kutta explicit scheme [15] that predicts the new state vector $\hat{x}(t_n + h_n)$ using the current state value $\tilde{x}(t_n)$, i.e.,

$$\hat{x}(t_n + h_n) = \hat{x}(t_n) + \frac{h_n}{2} (k_1 + k_2),$$

where $k_1 = \Phi(y_n, u_n^m, h_n)$ and $k_2 = \Phi(y_n, u_n^m, h_n)$. The error bound for this proposed scheme is given by

$$\|x_n - \hat{x}_n\| \leq \hat{C} \varepsilon(u_m) + \hat{C} \varepsilon(u_m),$$

where $\hat{C}$ is a constant.
Algorithm 2: Data-Driven Runge-Kutta (RK) Scheme

1. **Require:** initial state vector $x(a) = x_0$, partition $\mathcal{P}$, control input $u^n_m$, for $n = 1, \ldots, M - 1$, and trained DeepONet $\mathcal{F}_{\theta^*}$.
2. initialize $\tilde{x}(t_0) = x_0$
3. for $n = 0, \ldots, M - 1$ do
   4. update the independent variable $t_{n+1} = t_n + h_n$
   5. use the DeepONet’s forward pass to compute
      \[ \tilde{x}(t_{n+1}) = \mathcal{F}_{\theta^*} (\tilde{x}(t_n), u^n_m)(h_n). \] (14)
   6. use automatic differentiation to estimate the vector field $f$ at $t = t_n$ and $t = t_{n+1}$
      \[ k_1 = \frac{d}{dt} (\mathcal{F}_{\theta^*} (\tilde{x}(t_n), u^n_m)(0)) = f(\tilde{x}(t_n), u^n_m), \] (13a)
      \[ k_2 = \frac{d}{dt} \tilde{x}(t_{n+1}) = f(\tilde{x}(t_{n+1}), u^n_m). \] (13b)
   7. update the state vector with the improved Euler (RK-2) step
      \[ \tilde{x}(t_{n+1}) = \tilde{x}(t_n) + \frac{h_n}{2}(k_1 + k_2). \] (14)
4. end
5. **Return:** predicted response $\{\tilde{x}(t_n) : t_n \in \mathcal{P}\}$.

Here $k_1$ and $k_2$ are, respectively, the estimates of $f$ at $t_n$ and $t_{n+1}$. We compute these estimates (see equation (13)) using (1) the forward pass of trained DeepONet $\mathcal{F}_{\theta^*}$ and (2) automatic differentiation. Note that in (13b), we use the notation $\tilde{x}(t_{n+1})$ for the estimate of the state at $t_{n+1}$ obtained using the DeepONet’s $\mathcal{F}_{\theta^*}$ forward pass.

We detail the proposed data-driven RK explicit scheme in Algorithm 2. Two remarks about our Algorithm are provided next. (1) For simplicity, we only present our scheme for the improved Euler method or RK-2 [15]. However, we remark that we can extend our idea to any RK explicit scheme. (2) If $u^n_{m+1}$ is available at $t_n$, then we can compute $k_2$ as follows
\[ k_2 = \frac{d}{dt} (\mathcal{F}_{\theta^*}(\tilde{x}(t_n), u^n_{m+1})(h_n)). \] (12)

Then, equations (13a) and (12) will work as a predictor-corrector scheme with updated input information. Other strategies can also be adopted within the proposed RK scheme. However, we let the design of such strategies for our future work.

### 3.1 Error Bound for the Data-Driven Runge-Kutta Scheme

Here we derive a conditional improved error bound estimate for $\hat{x}(t_n)$. To that end, we start by rephrasing the universal approximation theorem of neural network for high-dimensional functions [8], which we introduced in Section 2.3. For $\tilde{e}(u_m) - C_4 h^2_n > 0$, there exist $W_1 \in \mathbb{R}^{K \times (n+n_u)}$, $b_1 \in \mathbb{R}^K$, $W_2 \in \mathbb{R}^{n \times K}$ and $b_2 \in \mathbb{R}^n$ such that
\[ \| \varphi(y_n, u^n_m) - \left( W_2 \sigma(W_1[y_n, u^n_m]^\top + b_1) + b_2 \right) \| < \epsilon, \] (15)
where \( \epsilon := \hat{e}(u_m) - C_4 h^2 \) and \( C_4 > 0 \) is constant. As before, the two-layer network represents the DeepONet \( F_{\theta^*} \) for a given \( h_n \).

The next Lemma estimates the error between \( \hat{x} \), predicted using the RK scheme (Algorithm 2), and \( \hat{x} \), obtained using the solution operator [4] of the approximate system (3).

**Lemma 3.1.** Assume \( \Phi \) is Lipschitz in \( x \) with Lipschitz constant \( C_2 \). Suppose the DeepONet is well trained so that (15) holds. Then, we have the estimate

\[
\| \hat{x}(t_n) - \hat{x}(t_n) \| \leq \hat{C} \hat{e}(u_m),
\]

where \( \hat{C} = \frac{C_1 h}{2} \frac{1 - C_n}{1 - C_3} \), and \( C_3 = \left( 1 + (1 + C_2) \frac{C_1 h}{2} \right) \).

**Proof.** We denote \( \bar{x}_n = \hat{x}(t_n) \), \( \bar{x}_n = \hat{x}(t_n) \), and \( \bar{x}_n = \hat{x}(t_n) \). Then, it follows from \( f \) being Lipschitz that,

\[
\begin{align*}
\| \bar{x}_{n+1} - \bar{x}_{n+1} \| & = \| \bar{x}_n + \int_{t_n}^{t_{n+1}} f(\hat{x}(s), u_m^n) ds - \hat{x}_n - \frac{h_n}{2} (f(\bar{x}_{n+1}, u_m^n) + f(\bar{x}_n, u_m^n)) \\
& \leq \| \bar{x}_n - \hat{x}_n \| + \| f(\bar{x}_{n+1}, u_m^n) - f(\bar{x}_n, u_m^n) \| + \| f(\bar{x}_n, u_m^n) - f(\bar{x}_n, u_m^n) \| + O(h^3) \\
& \leq \| \bar{x}_n - \hat{x}_n \| + \frac{C_1 h_n}{2} \| \bar{x}_n - \hat{x}_n \| + \frac{C_1 h_n}{2} \| \bar{x}_{n+1} - \bar{x}_{n+1} \| + O(h^3) \\
& \leq \left( 1 + \frac{C_1 h_n}{2} \right) \| \bar{x}_n - \hat{x}_n \| + \frac{C_1 h_n}{2} \| \bar{x}_{n+1} - \bar{x}_{n+1} \| + O(h^3). \tag{17}
\end{align*}
\]

An estimate of the above term \( \| \bar{x}_{n+1} - \bar{x}_{n+1} \| \) is

\[
\begin{align*}
\| \bar{x}_{n+1} - \bar{x}_{n+1} \| & = \| \Phi(\bar{x}_n, u_m^n, h_n) - F_{\theta^*}(\bar{x}_n, u_m^n) \| \\
& \leq \| \Phi(\bar{x}_n, u_m^n, h_n) - \Phi(\bar{x}_n, u_m^n, h_n) + \phi(\bar{x}_n, u_m^n) - F_{\theta^*}(\bar{x}_n, u_m^n) \| \\
& \leq C_2 \| \bar{x}_n - \hat{x}_n \| + \hat{e}(u_m^n) - C_4 h^2, \tag{18}
\end{align*}
\]

where \( C_4 > 0 \) is a constant. Substituting (18) back into (17) yields

\[
\| \bar{x}_{n+1} - \bar{x}_{n+1} \| \leq \left( 1 + (1 + C_2) \frac{C_1 h_n}{2} \right) \| \bar{x}_n - \hat{x}_n \| + \frac{C_1 h_n}{2} \hat{e}(u_m^n). \tag{19}
\]

Recursive estimation and \( \hat{x}_0 = \bar{x}_0 \) gives the desired error bound (16). \( \square \)

Two remarks about the error bound are as follows. (1) Note that the proposed data-driven RK scheme provides an improved error bound (16) when compared to the bound obtained (10). More specifically, the growth factor \( C_3 \) here behaves like \( C_2 \) in (10). However, when \( h_n < \frac{C_1}{C_3} \frac{2C_2 - 2}{C_4 + 1} \), one can derive a smaller factor. (2) We can extend the proof provided here for the RK-2 scheme to any other RK explicit scheme. We let such a analysis for our future work. Let us conclude this section with the following theorem that summarizes the error of the proposed data-driven RK scheme.

**Theorem 3.2.** For any \( t_n \in \mathcal{P} \), we have

\[
\| x(t_n) - \hat{x}(t_n) \| \leq \hat{C} \hat{e}(u_m) + \hat{C} \hat{e}(u_m), \tag{20}
\]

where \( \hat{C} \), \( \hat{C} \) and \( \hat{e}(u_m) \) are, respectively, the constants defined in [7], [16], and [7].
4 Numerical Experiments

To evaluate our framework, we test the DeepONet on three continuous control tasks: the predator-prey dynamics with control (in Section 4.1), the pendulum swing-up (in Section 4.2), and the cart-pole system (in Section 4.3). For all three tasks, we use only $n_s = 1$ sensors. The reasons for selecting only one sensor are two-fold. First, we want to show that DeepONet is effective even when the input signal is encoded with minimal information. For reference, in [25], the authors encoded the input signals (used in their experiments) with at least $n_s = 3$ interpolation points (sensors). Second, the $n_s = 1$ sensor scenario resembles the scenario of sampled-data control systems [36] or reinforcement learning tasks [32] with continuous action space.

Training dataset. For each one of the three continuous control tasks, we generate the training dataset $D_{\text{train}}$ as follows. We use Runge-Kutta (RK-4) [15] to simulate $N_{\text{train}}$ trajectories of size two. For each trajectory, the input to RK-4 is the initial condition $x(t_n)$ uniformly sampled from $\mathcal{X}$ and the control input $u(t_n)$ uniformly sampled from the set $\mathcal{U}$. The output from the RK-4 algorithm is the state $x(t_n+h)$, where $h$ is uniformly sampled from the interval $[0,0.25]$. Such a procedure gives the dataset:

$$D_{\text{train}} = \{(x_i(t_n), u_i(t_n)), h_i, x_i(t_n+h_i)\}_{i=1}^{N_{\text{train}}}.$$  

Training protocol and neural networks. We implemented our framework using JAX [1]. The neural networks for the Branch and Trunk Nets are the modified fully-connected networks proposed in [33] and used in our previous paper [22]. We trained the parameters of the networks using Adam [17]. Moreover, we selected (1) the default hyper-parameters for the Adam algorithm and (2) an initial learning rate of $\eta = 0.001$ that exponentially decays every 2000 epochs.

4.1 The Predator-Prey Dynamics with Control

To evaluate our framework, we first consider the following Lotka-Volterra Predator-Prey system with input signal $u(t)$:

$$\dot{x}_1 = x_1 - x_1 x_2 + u(t)$$
$$\dot{x}_2 = -x_2 + x_1 x_2.$$  

(21)

The system [21] was also studied in [25] where the authors encoded $u(t)$ using three interpolation points. To train our DeepONet, we generated $N_{\text{train}} = 2000$ trajectories with the initial condition $x_i(t_n)$ (resp. input signal $u_i(t_n)$) sampled from the state space $\mathcal{X} := [0,5]^2$ (resp. input space $\mathcal{U} := [0,5]$).

We use the trained DeepONet to predict the predator-prey (21) system’s response to the input signal $u(t) = \sin(t/3) + \cos(t) + 2$ within a partition $\mathcal{P} \subset [0,100]$ (s) with the constant step size $h = 0.1 \equiv t_{n+1} - t_n$, for all $t_n,t_{n+1} \in \mathcal{P}$. Figure 2 compares the DeepONet’s long-term prediction with the true trajectory. Note that for both states, $x = (x_1, x_2)^T$, the predicted trajectory agrees very well with the true trajectory. The $L_2$-relative errors for $x_1$ and $x_2$ are, respectively, 2.42% and 0.93%.

4.2 Pendulum Swing-Up

Let us now consider the following pendulum swing-up control system:

$$\ddot{\theta} \left(\frac{1}{4}ml^2 + 1\right) + \frac{1}{2}mlg \sin \theta = u(t) - b\dot{\theta},$$  

(22)

We will publish the code in GitHub after publication.
where \( x = (\theta, \dot{\theta})^\top \in \mathcal{X} \) is the state vector, \( \theta \) the pendulum’s angle, \( \dot{\theta} \) the angular velocity, and \( u(t) \in \mathcal{U} \) the control torque. We set the parameters to the following values. The pendulum’s mass is \( m = 1 \) (kg), the length is \( l = 1 \) (m), the moment of inertia of the pendulum around the midpoint is \( I = \frac{1}{12} ml^2 \), and the friction coefficient is \( b = 0.01 \) (sNm/rad).

We train the proposed framework using \( N_{\text{train}} = 5000 \) trajectories. Each trajectory is generated using (1) an initial condition \( x_i(t_n) \) sampled from the state space \( \mathcal{X} := [-\pi, \pi] \times [-8, 8] \) and (2) a control torque \( u_i(t_n) \) sampled from the input space \( \mathcal{U} := [-2, 2] \).

**Stable response.** We first use the trained DeepONet to predict the pendulum’s response to the control input \( u_1(t) = -0.8\dot{\theta}(t) \) within the partition \( \mathcal{P} \subset [0, 10] \) with step size \( h = 0.1 \). Such an input yields state trajectories \( \{(\theta(t_n), \dot{\theta}(t_n)) : t_n \in \mathcal{P}\} \) that settle to an asymptotic equilibrium point. Figure 3 illustrates an excellent agreement between the predicted and the actual trajectory.

To test the predictive power of the proposed framework, we compute the average and standard deviation (st. dev.) of the \( L_2 \)-relative error between the predicted and actual response of the
|         | angle $\theta(t)$ | angular velocity $\dot{\theta}(t)$ |
|---------|-------------------|------------------------------------|
| mean $L_2$ | 1.056 %          | 3.356 %                            |
| st.dev. $L_2$ | 2.509 %        | 8.234%                             |

Table 1: The average and standard deviation (st.dev.) of the $L_2$–relative error between the predicted and actual response trajectories of the pendulum system (22) to (1) the control torque $u_1(t) = -0.8\dot{\theta}(t)$ and (2) 100 initial condition uniformly sampled from the set $X_o := \{\theta, \dot{\theta} : \theta \in [-\pi/2, \pi/2], \dot{\theta} = 0\}$.

Figure 4: Comparison of the DeepONet prediction with the actual trajectories of the pendulum (22) system’s state $x = (\theta(t), \dot{\theta}(t))^\top$ response to the input signal $u(t) = \sin(t/2)$ within the partition $P \subset [0, 10]$ (s) of constant step size $h = 0.1$.

Oscillatory response. We now test the trained DeepONet for the control torque $u_2(t) = \sin(t/2)$ within the partition $P \subset [0, 10]$ (s) with constant step size $h = 0.1$. Figure 4 depicts the excellent agreement between the predicted and actual oscillatory trajectory.

Let us now consider a different partition $P'$ with a smaller step size $h = 0.00025$. Figure 5 shows that the proposed DeepONet fails to keep up with the oscillatory response. To improve our prediction, we employ the proposed data-driven Runge-Kutta (RK) method described in Algorithm 2. Figure 5 depicts agreement between the actual trajectory and the trajectory predicted using the proposed data-driven RK method. The corresponding $L_2$-relative errors are 7.73% and 11.34% for the angle $\theta(t)$ and the angular velocity $\dot{\theta}(t)$, respectively.
Figure 5: Comparison of the DeepONet prediction with the data-driven RK prediction, and the actual trajectories of the pendulum system’s state $x = (\theta(t), \dot{\theta}(t))^\top$ response to the input signal $u(t) = \sin(t/2)$ within the partition $\mathcal{P}' \subset [0, 10]$ (s) of constant step size $h = 0.00025$.

4.3 Cart-Pole

Finally, we consider the following cart-pole system [13] with control:

$$\ddot{\theta} = \frac{g \sin \theta + \cos \theta \left( \frac{-u(t) - m_p \dot{\theta}^2 \sin \theta}{m_c + m_p} \right)}{l \left( \frac{4}{3} - \frac{m_p \cos^2 \theta}{m_c + m_p} \right)}$$

$$\ddot{p} = \frac{u(t) - b \dot{p} + m_p l (\dot{\theta}^2 \sin \theta - \dot{\theta} \cos \theta)}{m_c + m_p}. \tag{23}$$

In the above, the state of the cart-pole system is $x = (\theta, \dot{\theta}, p, \dot{p})^\top \in \mathcal{X}$, where $\theta$ is the angle of the pendulum, $\dot{\theta}$ the angular velocity of the pendulum, $p$ and $\dot{p}$ are, respectively, the position and the velocity of the cart. The control input $u \in \mathcal{U}$ is the horizontal force that makes the cart move to the left or the right. We selected the parameters of the cart-pole system as follows. The pendulum’s length is $l = 0.5$ (m), the pendulum’s mass is $m_p = 0.5$ (kg), the cart’s mass is $m_c = 0.5$ (kg), and the friction coefficient is $b = 0.01$ (sN/m/rad).

To train our DeepONet, we generated $N_{\text{train}} = 20000$ trajectories with the initial condition $x_i(t_n)$ (resp. control input $u_i(t_n)$) sampled from the state space $\mathcal{X} := [-2\pi, 2\pi] \times [-\pi, \pi] \times [-2, 2] \times [-1, 1]$ (resp. $\mathcal{U} := [-5, 5]$).

We use a trained DeepONet to predict the cart-pole system’s response to the time-dependent input signal $u(t) = t/100$ within a partition $\mathcal{P} \subset [0, 10]$ (s) with the constant step size $h = 0.1 \equiv t_{n+1} - t_n$, for all $t_n, t_{n+1} \in \mathcal{P}$. We depict in Figure 6 a comparison between the predicted and true state trajectories. We can observe agreement between the predicted and true state trajectories with $L_2$-relative errors of 0.008%, 1.028%, 0.478%, and 0.296% for $\theta$, $\dot{\theta}$, $p$, and $\dot{p}$. From these results, we may conclude that the proposed DeepONet framework can effectively predict the response of a nonlinear control system, such as the cart-pole, to given control inputs and different initial conditions.
Figure 6: Comparison of the DeepONet prediction with the actual trajectory of the cart-pole system’s state $x = (\theta(t), \dot{\theta}(t), p(t), \dot{p}(t))^T$ response to the input signal $u(t) = t/100$ within the partition $\mathcal{P} \subset [0, 10]$ (s) of constant step size $h = 0.1$.

5 Conclusion

We introduced a Deep Operator Network (DeepONet) framework to learn (from data) the dynamic response of nonlinear control systems. Our framework approximates locally, using the DeepONet, the control system’s solution operator. Then, it predicts the system’s response for long/medium-term horizons using the trained DeepONet recursively. We estimated the error bound for such a DeepONet-based numerical scheme. To improve the predictive accuracy when the step size of the scheme is small, we designed and theoretically validated a data-driven Runge-Kutta (RK) scheme that uses estimates of the vector field computed with the DeepONet forward pass and automatic differentiation. We validated our framework using three continuous control tasks. In our future work, we aim to extend the proposed framework to (1) reduced-order, (2) stochastic, and (3) networked control systems. From an application perspective, our goal is to use the proposed DeepONet framework for model-based reinforcement learning. In particular, we aim to use the proposed work for learning semi-Markov decision processes, which, in turn, can be used to learn suboptimal policies offline.

References

[1] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milhe, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
[2] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. arXiv preprint arXiv:1606.01540, 2016.

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

[4] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Sparse identification of nonlinear dynamics with control (sindyc). IFAC-PapersOnLine, 49(18):710–715, 2016.

[5] S. Cai, Z. Wang, L. Lu, T. A. Zaki, and G. E. Karniadakis. Deeppm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. Journal of Computational Physics, 436:110296, 2021.

[6] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Transactions on Neural Networks, 6(4):911–917, 1995.

[7] E. Chung, W. T. Leung, S.-M. Pun, and Z. Zhang. A multi-stage deep learning based algorithm for multiscale model reduction. Journal of Computational and Applied Mathematics, 394:113506, 2021.

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

[9] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian processes for data-efficient learning in robotics and control. IEEE transactions on pattern analysis and machine intelligence, 37(2):408–423, 2013.

[10] J. Du, J. Futoma, and F. Doshi-Velez. Model-based reinforcement learning for semi-markov decision processes with neural odes. Advances in Neural Information Processing Systems, 33:19805–19816, 2020.

[11] Y. Efendiev, W. T. Leung, G. Lin, and Z. Zhang. Hei: hybrid explicit-implicit learning for multiscale problems. arXiv preprint arXiv:2109.02147, 2021.

[12] D. Fan, L. Yang, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. Proceedings of the National Academy of Sciences, 117(42):26091–26098, 2020.

[13] R. V. Florian. Correct equations for the dynamics of the cart-pole system. Center for Cognitive and Neural Studies (Coneural), Romania, 2007.

[14] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. arXiv preprint arXiv:1912.01603, 2019.

[15] A. Iserles. A first course in the numerical analysis of differential equations. Number 44. Cambridge university press, 2009.

[16] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, et al. Model-based reinforcement learning for atari. arXiv preprint arXiv:1903.00374, 2019.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.
[18] G. Lin, C. Moya, and Z. Zhang. Accelerated replica exchange stochastic gradient langevin diffusion enhanced bayesian deeponet for solving noisy parametric pdes. *arXiv preprint arXiv:2111.02484*, 2021.

[19] G. Lin, Y. Wang, and Z. Zhang. Multi-variance replica exchange stochastic gradient mcmc for inverse and forward bayesian physics-informed neural network. *arXiv preprint arXiv:2107.06330*, 2021.

[20] L. Lu, P. Jin, and G. E. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[21] L. Lu, P. Jin, G. Pang, Z. Zhang, and G. E. Karniadakis. Learning nonlinear operators via deepnet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[22] C. Moya, S. Zhang, M. Yue, and G. Lin. Deepnet-grid-uq: A trustworthy deep operator framework for predicting the power grid’s post-fault trajectories. *arXiv preprint arXiv:2202.07176*, 2022.

[23] A. Nagabandi, K. Konolige, S. Levine, and V. Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[24] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Generalizing koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930, 2018.

[25] T. Qin, Z. Chen, J. D. Jakeman, and D. Xiu. Data-driven learning of nonautonomous systems. *SIAM Journal on Scientific Computing*, 43(3):A1607–A1624, 2021.

[26] T. Qin, K. Wu, and D. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[27] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

[28] R. Ranade, K. Gitushi, and T. Echekki. Generalized joint probability density function formulation in turbulent combustion using deepnet. *arXiv preprint arXiv:2104.01996*, 2021.

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

[30] Y. Sun, L. Zhang, and H. Schaeffer. Neupde: Neural network based ordinary and partial differential equations for modeling time-dependent data. In *Mathematical and Scientific Machine Learning*, pages 352–372. PMLR, 2020.

[31] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine learning proceedings 1990*, pages 216–224. Elsevier, 1990.

[32] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[33] S. Wang, Y. Teng, and P. Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.
[34] S. Wang, H. Wang, and P. Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed deeponets. *Science advances*, 7(40):eabi8605, 2021.

[35] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

[36] B. Wittenmark, K. J. Åström, and K.-E. Årzen. Computer control: An overview. *IFAC Professional Brief*, 1:2, 2002.