# Neural Dynamical Systems: Balancing Structure and Flexibility in Physical Prediction

Viraj Mehta[1], Ian Char[2], Willie Neiswanger[2], Youngseog Chung[1], Andrew Nelson[3], Mark Boyer[3],
Egemen Kolemen[3], and Jeff Schneider[1]

[1]Robotics Institute, Carnegie Mellon University
[2]Machine Learning Department, Carnegie Mellon University
[3]Princeton Plasma Physics Laboratory

*Abstract*— We introduce Neural Dynamical Systems (NDS), a method of learning dynamical models in various gray-box settings which incorporates prior knowledge in the form of systems of ordinary differential equations. NDS uses neural networks to estimate free parameters of the system, predicts residual terms, and numerically integrates over time to predict future states. A key insight is that many real dynamical systems of interest are hard to model because the dynamics may vary across rollouts. We mitigate this problem by taking a trajectory of prior states as the input to NDS and train it to dynamically estimate system parameters using the preceding trajectory. We find that NDS learns dynamics with higher accuracy and fewer samples than a variety of deep learning methods that do not incorporate the prior knowledge and methods from the system identification literature which do. We demonstrate these advantages first on synthetic dynamical systems and then on real data captured from deuterium shots from a nuclear fusion reactor. Finally, we demonstrate that these benefits can be utilized for control in small-scale experiments.

## I. INTRODUCTION

The use of function approximators for dynamical system modeling has become increasingly common. This has proven quite effective when a substantial amount of real data is available relative to the complexity of the model being learned [11, 21, 9]. These learned models are used for downstream applications such as model-based reinforcement learning [31, 35] or model-predictive control (MPC) [41].

Model-based control techniques are exciting as we may be able to solve new classes of problems with improved controllers. Problems like dextrous robotic manipulation [30], game-playing [38], and nuclear fusion are increasingly being approached using model-based reinforcement learning techniques. However, learning a dynamics model using, for example, a deep neural network can require large amounts of data. This is especially problematic when trying to optimize real physical systems, where data collection can be expensive. As an alternative to data-hungry machine learning methods, there is also a long history of fitting models to a system using techniques from system identification, some of which include prior knowledge about the system drawn from human understanding [32, 23, 39]. These models, especially in the gray-box setting, are typically data-efficient and often contain interpretable model parameters. However,

Email for Correspondence: virajm@cs.cmu.edu

they are not well suited for the situation where the given prior knowledge is approximate or incomplete in nature. They also do not generally adapt to the situation where trajectories are drawn from a variety of parameter settings at test time. This is an especially crucial point as many systems of interest exhibit path-dependent dynamics, which we aim to recover on the fly.

In total, system identification methods are sample efficient but inflexible given changing parameter settings and incomplete or approximate knowledge. Conversely, deep learning methods are more flexible at the cost of many more samples. In this paper, we aim to solve both of these problems by biasing the model class towards our physical model of dynamics. Physical models of dynamics are often given in the form of systems of ordinary differential equations (ODEs), which are ubiquitious and may have free parameters that specialize them to a given physical system. We develop a model that uses neural networks to predict the free parameters of an ODE system from the previous timesteps as well as residual terms added to each component of the system. To train this model, we integrate over the ODE and backpropagate gradients from the prediction error. This particular combination of prior knowledge and deep learning components is effective in quickly learning the dynamics and allows us to adjust system behavior in response to a wide variety of dynamic parameter settings. Even when the dynamical system is partially understood and only a subset of the ODEs are known, we find that our method still enjoys these benefits. We apply our algorithm to learning models in three synthetic settings: a generic model of ballistics, the Lorenz system [24], and a generalized cartpole problem, which we use for control as well. We also learn a high-level model of plasma dynamics for a fusion tokamak from real data.

The contributions of this paper are
- We introduce Neural Dynamical Systems (NDS), a new class of model for learning dynamics that can incorporate prior knowledge about the system.
- We show that these models naturally handle the issue of partial or approximate prior knowledge, irregularly spaced data, and system dynamics that change across instantiations, which generalizes the typical system iden-

tification setting. We also show that these advantages extend to control settings.

- We demonstrate this model's effectiveness on a real dynamics problem relevant to nuclear fusion and on synthetic problems where we can compare against a ground truth model.

## II. RELATED WORK

There is a long tradition of forecasting physical dynamics with machine learning. [13] lays out ideas from classical statistics for predicting spatiotemporal data. In the deep learning world, recurrent neural networks and long short-term memory networks have been used for a long time to address sequential problems [20]. However, these models struggle with continuous-time data and do not allow for the introduction of physical priors. Recently, there has been work in learning provably stable dynamics models by constraining the neural network to have a stable and jointly learned Lyapunov function [25]. We see opportunities to use these techniques in follow up work.

*a) Neural Ordinary Differential Equations:* As most numerical ODE solvers are algorithms involving differentiable operations, it has always been possible in principle to backpropagate through the steps of these solvers dating back to at least [37]. However, since each step of the solver involves calling the derivative function, naïve backpropagation incurs an $O(n)$ memory cost in the number of ODE solution steps, where $n$ is the number of derivative calls made by the solver. [8] demonstrated that by computing gradients through the adjoint sensitivity method, the memory complexity of backpropagating through a family of ODE solvers can be reduced to $O(1)$ for a fixed network, as opposed to the naive $O(n)$. However, this work only used generic neural networks as the derivative function and did not consider dynamics. They also provide a PyTorch package which we have built off of in our work.

There has been some work using neural ordinary differential equations to solve physical problems. [33] used a fully-connected neural ODE with an RNN encoder and decoder to model Navier-Stokes problems. [36] used a neural network integrated with a Runge-Kutta method for noise reduction and irregularly sampled data. There has also been work learning the structure of dynamical systems, first with a convolutional-deconvolutional warping scheme inspired by the solutions to advection-diffusion PDEs [2], then with a Neural ODE which was forced to respect boundary conditions and a partial observation mechanism [1]. There was also work on constraining a neural network model to respect separable conservative Hamiltonian dynamics, but like all of the aforementioned methods, this does not incorporate information about the dynamics of a particular system [10]. In general, none of these methods incorporate prior knowledge as explicitly as including appropriate equations in the statistical model. Furthermore, many of these methods focus on a specific problem, whereas we give a way to apply specific knowledge about a variety of problems.

An interesting approach that builds on a previous version of this paper [26] is developed in [18], where it is shown that

iteratively solving a Lagrangian formulation of the problem for a fixed set of parameters and varying Lagrange multiplier results in predictions which maximally use the physical model. Our problem setting is a generalization of theirs in that we allow the parameters to vary among rollouts and predict them on the fly.

A further generalization is [34], which treats the whole problem of differential equations with missing components as a framework for modeling a very wide range of settings. We give a more prescriptive architecture for our class of problems and extend the analysis to a setting with real data as well as showing an application of our model for control.

*b) Machine Learning for Nuclear Fusion::* As far back as 1995, [28] showed that by approximating the differential operator with a (single-layer, in their case) neural network, one could fit simple cases of the Grad-Shafranov equation for magnetohydrodynamic equilibria. Recently, work has shown that plasma dynamics are amenable to neural network prediction. In particular, [22] used a convolutional and LSTM-based architecture to predict possible plasma disruptions (when a plasma instability grows large and causes a loss of plasma containment and pressure).

There has also been work in the field of plasma control: a neural network model of the neutral beam injection for the DIII-D tokamak has been deployed in order to diagnose the effect of controls on shots conducted at the reactor [3]. Additionally, [4] used classic control techniques and a simpler model of the dynamics to develop a controller that allows characteristics of the tokamak plasma to be held at desired levels.

## III. PROBLEM SETTING

Typically, a dynamical system $\dot{x} = f_\phi(x, u, t)$ with some parameters $\phi$ is the conventional model for system identification problems. Here, state is $x \in \mathcal{X}$, control is $u \in \mathcal{U}$, and time is $t \in \mathbb{R}$. The objective is to predict future states given past states, past and future controls, and prior knowledge of the form of $f$. We denote $x(\phi, t, \mathbf{u}, x_0) = x_0 + \int_0^t f_\phi(x, u, t) dt$ as the state obtained by integrating our dynamical system around $f$ to time $t$.

We consider in this work a more general setting and address the problem of prediction and control over a *class of dynamical systems*, which we define as the set $\{\dot{x} = f_\phi(x, u, t) \mid \phi \in \Phi\}$, where $\Phi$ is the space of parameters for the dynamical system (e.g. spring constant or terminal velocity). We can generate a trajectory from a class by sampling a $\phi \sim P(\Phi)$ for some distribution $P$ and choosing initial conditions and controls. In real data, we can view nature as choosing, but not disclosing, $\phi$. For a particular example $j$, we sample $\phi \sim P(\Phi)$ and $x_0 \sim P(X_0)$ and are given controls $\mathbf{u}$ indexed as $u(t)$ and input data $\{x(\phi, t_i, \mathbf{u}, x_0)\}_{i=0}^{T}$ during training. At test time, we give a shorter, prefix time series $\{x(\phi, t_i, \mathbf{u}, x_0)\}_{i=0}^{T'}$ but assume access to future controls. Then the prediction objective for a class of systems for $N$ examples for timesteps $\{t_i\}_{T'+1}^T$ is

$$\hat{x} = \arg\min_{\hat{x}} \mathbb{E}_{\substack{x_0 \sim P(X_0) \\ \phi \sim P(\Phi)}} \left[ \sum_{i=T'+1}^{T} ||x(\phi, t_i, \mathbf{u}, x_0) - \hat{x}_{t_i}||_2^2 \right].$$

This objective differs from the traditional one in that implicitly, identifying $\phi$ for each trajectory needs to be done from the problem data in order to be able to predict the data generated by $f_\phi$.

Similarly, the control problem is

$$\mathbf{u} = \min_{\mathbf{u}} \mathbb{E}_{\phi \sim P(\Phi), x_0 \sim P(X_0)} \left[ \int_0^t c(u(t), x(t)) dt \right],$$

$$\text{s.t. } x(t) = x_0 + \int_0^t f_\phi(x, u, t) \, dt$$

for some cost functional $c$. We will primarily explore the prediction problem in this setting, but as secondary considerations, we explore robustness to noise, the ability to handle irregularly spaced input data, and the ability to recover the parameters $\phi$ which generated the original trajectories. We will also consider the control problem in a simple setting.

## IV. METHODS

We build up the description of our proposed method by first describing the two methods that inspire it: gray box system identification through optimization [23], and using a Neural ODE [8] to predict future states in a dynamical system.

To apply grey box optimization [23] to a dynamical system $\dot{x} = f_\phi(x, u, t)$ for problem data $\{x_{t_i}\}_{t=0}^{T'}$, we would use nonlinear least squares [12] to find

$$\hat{\phi} = \arg \min_{\hat{\phi}} \sum_i \left\| \int_0^{t_i} f_\phi(x, u(t), t) \, dt - \int_0^{t_i} f_{\hat{\phi}}(x, u(t), t) dt \right\|. \tag{1}$$

This makes good use of the prior knowledge component of our system but is prone to compounding errors through integration and does not leverage data that may have come from alternate system parameters.

A data driven approach would be to minimize the same objective with a fully connected neural ODE [8] $h_\theta$ in place of $f$. However, we find that this procedure requires large amounts of training data and doesn't leverage any prior knowledge we might have, though it is flexible to classes of dynamical systems.

We define a Neural Dynamical System by taking the advantages of both these methods in the setting where we know the full and correct ODEs and then show how to generalize it to situations where only some ODEs are known or they are approximate. Specifically, a *Neural Dynamical System* (NDS) is a class of dynamical systems where a neural network predicts some part of $f_\phi(x, u, t)$, usually parameters $\phi$ or a term which is added to $f$.

*a) NDS with Full System Dynamics:* Consider a class of dynamical systems as defined in Section III where $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $\phi \in \mathbb{R}^{d_p}$, $d_h, d_c \in \mathbb{N}$ and let $\theta$, $\vartheta$, $\tau$ be trainable neural network weights. Let $h_\theta(x_{t_{1:T'}}, u_{t_{1:T}})$ be a neural net mapping state history and control sequence to the $d_p$ parameters of the system $\hat{\phi}$ and an embedding $b_h \in \mathbb{R}^{d_h}$. Also let $c_\vartheta(x_t, u_t)$ be a similar network taking a single state and control that outputs an embedding $b_c \in \mathbb{R}^{d_c}$. Finally,

let $d_\tau(b_h, b_c)$ be a network which takes the two output embeddings from the previous network and outputs residual terms $\hat{r}$. Intuitively, we would like to use the observed history to estimate our system parameters, and some combination of the observed history and current observation to estimate residuals, which influences the design of our model, the neural dynamical system (a visualization of which is shown in Figure 1), written

$$
\begin{aligned}
\dot{x} &= \underbrace{g_{\hat{\phi}}(x_t, u_t, t)}_{\text{Prior knowledge}} + \hat{r} & \hat{\phi}, b_h &= \underbrace{h_\theta(x_{t_{1:T'}}, u_{t_{1:T}})}_{\text{History encoder}} \\
b_c &= \underbrace{c_\vartheta(x_t, u_t)}_{\text{Context encoder}} & \hat{r} &= \underbrace{d_\tau(b_h, b_c)}_{\text{Residual prediction}}
\end{aligned} \tag{2}
$$

where $g$ are domain-specific ODEs which are the input 'domain knowledge' about the system being modeled. Note that if the prior knowledge $g$ is identically zero, this method reduces to the Neural ODE predictions we discussed at the beginning of this section. We also study an ablated model, NDS0, which lacks the residual component $\hat{r}$ and context encoder network $d_\tau$. We note here that the context encoder is intended to potentially correct for model misspecification and noise but in the noiseless case with a model which is perfect, it may not be necessary. We explore this throughout Section V.

*Example 1: Lorenz system.* To illustrate the full construction, we operate on the example of the the Lorenz system: a chaotic dynamical system originally defined to model atmospheric processes [24]. The system has 3-dimensional state (which we'll denote by $x, y, z$), 3 parameters, $\rho, \sigma$, and $\beta$, and no control input. The system is given by

$$\dot{x} = \sigma(y - x) \quad \dot{y} = x(\rho - z) - y \quad \dot{z} = xy - \beta z. \tag{3}$$

For a given instantiation of the Lorenz system, we have values of $\phi = [\beta, \sigma, \rho]$ that are constant across the trajectory. So, we can instantiate $h_\theta$ which outputs $\hat{\phi} = [\hat{\beta}, \hat{\sigma}, \hat{\rho}]$. We use the DOPRI5 method [14] to integrate the full neural dynamical system in Equation 2, with $g$ given by the system in Equation 3 using the adjoint method of [8]. We use the state $x_{T'}$ as the initial condition for this integration. This gives a sequence $\{\hat{x}_t\}_{t=T'}^T$, which we evaluate and supervise with a loss of the form

$$\mathcal{L}_{\theta, \vartheta, \tau}(\{\hat{x}_{t_i}\}_{i=T'+1}^T, \{x_{t_i}\}_{t=T'+1}^T) = \sum_{t=T'+1}^T ||x_{t_i} - \hat{x}_{t_i}||_2^2. \tag{4}$$

Because of the way we generate our input data, this is equivalent to Equation III. We assume in our setting with full dynamics that the true dynamics lie in the function class established in Equation 2. By the method in [8] we can backpropagate gradients through this loss into the parameters of our NDS. Then algorithms in the SGD family will converge to a local minimum of our loss function.

*b) NDS with Partial System Dynamics:* Suppose we only had prior knowledge about some of the components of our system and none about others. We can easily accomodate
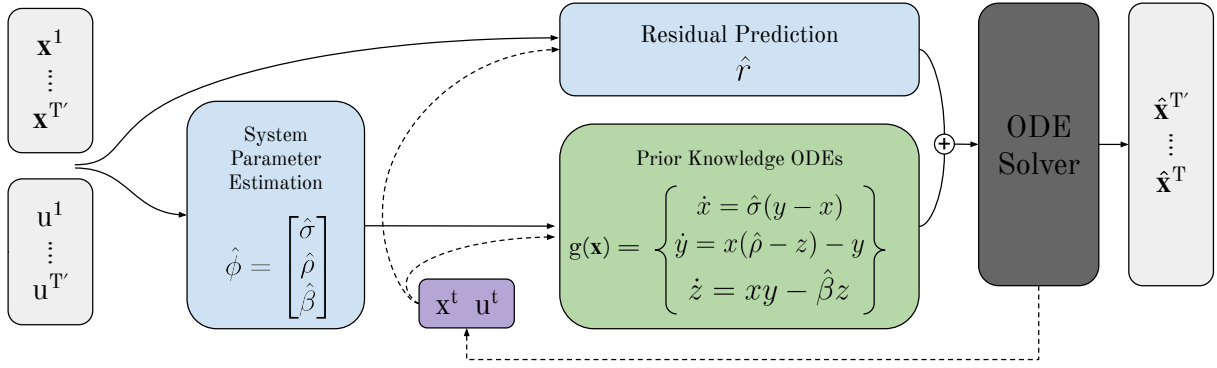
Fig. 1. **An example Neural Dynamical System.** Here, blue boxes are fully connected neural networks, gray boxes are problem data and output, the green box is the prior knowledge dynamical system, the purple box is data output by ODE solver to query derivatives, and the black box is an ODE solver. The ODEs and system parameters are problem dependent, but here we consider the Lorenz system (defined in Example 1) as an example. Our notation for $x$ is unfortunately overloaded by our method and the Lorenz system—the $x$ from our method is bolded in the figure.

this incomplete information by simply 'zeroing out' the function. This looks like

$$g_\phi(x, u, t) = \begin{bmatrix} \begin{cases} g_\phi^{(i)}(x, u, t) & \text{if } g_\phi^{(i)} \text{ is known,} \\ 0 & \text{else.} \end{cases} \end{bmatrix} \quad (5)$$

substituted into equation 2. In this setup, the residual term essentially makes the unknown dimensions unstructured Neural ODEs, which still can model time series well [33].

*c) NDS with Approximate System Dynamics:* For Neural Dynamical Systems to be useful, they must handle situations where the known model is approximate. This is transparently handled by our formulation of Neural Dynamical Systems: the parameters of the approximate model $\hat\phi$ are predicted by $h_\theta(x_{1:T'}, u_{1:T'})$ and the residuals $\hat r$ are predicted by $d_\tau(b_h, b_c)$. This is the same as in the case where we have the correct dynamics, but we remove the assumption of a perfect model.

*Example 2: Nuclear Fusion System.* In this paper, we apply this technique to plasma dynamics in a tokamak. In a tokamak, two quantities of interest are the stored energy of the plasma, which we denote $E$ and its rotational frequency, $\omega$. The neutral beams and microwave heating allow us to add power ($P$) and torque ($T$) to the plasma. The plasma also dissipates energy and rotational momentum via transport across the boundary of the plasma, radiative cooling, and other mechanisms. While the detailed evolution of these quantities is described by turbulent transport equations, for the purposes of control and design studies, physicists often use reduced, volume-averaged models. The simple linear model (up to variable parameters) used for control development in [4] is used in this work.

$$\dot E = P - \frac{E}{\tau_e} \quad \dot\omega = \frac{T}{n_i m_i R_0} - \frac{\omega}{\tau_m} \quad (6)$$

Here, $n_i$ is ion density, $m_i$ is ion mass, and $R_0$ is the tokamak major radius. We use the constant known values for these. $\tau_e$ and $\tau_m$ are the confinement times of the plasma energy and angular momentum, which we treat as variable parameters (because they are!). These are predicted by the neural network in our model. We again use the model in Equation 2 to give us a neural dynamical system which can

learn the real dynamics starting from this approximation in Section V-C.

## V. EXPERIMENTS

In the following experiments, we aim to show that our methods improve predictions of physical systems by including prior dynamical knowledge. These improvements hold even as we vary the configurations between structured and fluid settings. We show that our models learn from less data and are more accurate, that they handle irregularly spaced data well, and that they learn the appropriate parameters of the prior knowledge systems even when they only ever see trajectories.

We use L2 error as our evaluation measure for predictive accuracy as given by Equation 4, though in cases where the absolute errors aren't very interpretable, we normalize for ease of comparison. We also evaluate our model's ability to predict the system parameters by computing the L2 error, i.e. $\sum_{i=1}^{n} ||\hat\phi_i - \phi_i||_2^2$. For synthetic examples, we consider the Lorenz system in (3) and a simple Ballistic system modeling projectile motion under a variety of drag conditions. We learn over trajectories $\{(x_{t_i}, u_{t_i}, t_i)\}_{i=1}^{T}$ where the $x_{t_i}$ are generated by numerically integrating $\dot x_\phi(x, u, t)$ using scipy's odeint function [40], with $x_0$ and $\phi$ uniformly sampled from $\mathcal{X}$ and $\Phi$, and $u_{t_i}$ given. Note that $\phi$ remains fixed throughout a single trajectory. Details on the ranges of initial conditions and parameters sampled are in the appendix. We evaluate the synthetic experiments on a test set of 20,000 trajectories that is fixed for a particular random seed generated in the same way as the training data. We use a timestep of $0.5$ seconds for the synthetic trajectories. On the Ballistic system this allows us to see trajectories that do not reach their peak and those that start to fall. Since the Lyapunov exponent of the Lorenz system is less than 3, in 16 predicted timesteps we get both predictable and unpredictable data [16]). We believe it is important to look at the progress of the system across this threshold to understand whether the NDS model is robust to chaotic dynamics — since the Lorenz system used for structure is itself chaotic, we want to make sure that the system does not blow up over longer timescales.

We note that ReLU activations were chosen for all feedforward and recurrent architectures, while in the Neural-ODE-based architectures, we follow the recommendations of [8] and use the Softplus. The sizes and depths of the baselines were chosen after moderate hyperparameter search.

We can view the Partial NDS and NODE as ablations of the Full NDS model which remove some and all of the prior knowledge, respectively. Each model takes 32 timesteps of state and control information as input and are trained on predictions for the following 16 timesteps. The ODE-based models are integrated from the initial conditions of the last given state. All neural networks are all trained with a learning rate of $3 \times 10^{-3}$, which was seen to work well across models. We generated a training set of 100,000 trajectories, test set of 20,000 trajectories, and validation set of 10,000 trajectories. Training was halted if validation error did not improve for 3 consecutive epochs.

### A. Comparison Methods

We compare our models with other choices along the spectrum of structured to flexible models from both machine learning and system identification. In our paper, we compared the following methods in our experiments:

- **Full NDS**: A Neural Dynamical System with the full system dynamics for the problem being analyzed. The full construction of this model is given by Equation 2. For the functions $h_\theta, c_\vartheta, d_\tau$, we use fully connected networks with 2 layers, Softplus activations, 64 hidden nodes in each layer, and batch normalization.
- **Partial NDS**: A Neural Dynamical System with partial system dynamics for the problem being analyzed. These follow Equation 5 as applied to Equation 2. For the Ballistic system, we only provide equations for $\dot{x}$ and $\ddot{x}$, excluding the information about vertical motion from our network. For the Lorenz system, we only provide equations for $\dot{x}$ and $\dot{y}$, excluding information about motion in the $z$ direction. For the Cartpole system, we only provide information about $\dot{\theta}$ and $\ddot{\theta}$. These equations were chosen somewhat arbitrarily to illustrate the partial NDS effectiveness. We use similar neural networks here as for the Full NDS.
- **NDS0**: A Full NDS with residual terms removed. This serves as an ablation which shows the use of the residual terms.
- **Fully Connected** (FC): A Fully-Connected Neural Network with 4 hidden layers containing 128 nodes with ReLU activations and batch normalization.
- **Fully Connected Neural ODE** (FC NODE): A larger version of the Neural ODE as given in [8], we use 3 hidden layers with 128 nodes, batch norm, and Softplus activations for $\dot{x}$. This can be interpreted as a version of our NDS with no prior knowledge, i.e. $g(x) = 0$.
- **LSTM**: A stacked LSTM with 8 layers as in [17]. The data is fed in sequentially and we regress the outputs of the LSTM against the true values of the trajectory.
- **Gray Box Optimization** (GBO): We use MATLAB's gray-box system identification toolbox [23] along with

the prior knowledge ODEs to fit the parameters $\hat{\phi}$ as an alternative to using neural networks. This algorithm uses trust-region reflective nonlinear least squares with finite differencing [12] to find the parameter values which minimize the error of the model rollouts over the observed data.

- **Sparse Identification of Nonlinear Systems** (SR): We use the method from [6] to identify the dynamical systems of interest. This method uses sparse symbolic regression to learn a linear mapping from basis functions of the state $x_t$ and control $u_t$ to the derivatives $\dot{x}_t$ computed by finite differences. Our synthetic systems are in the span of the polynomial basis that we used.
- **APHYNITY**: We use the method from [18], which fits a min-error parameter but then has an additional neural network component to model unknown dynamics.

### B. Synthetic Experiments

We first present results on a pair of synthetic physical systems where the data is generated in a noiseless and regularly spaced setting.

*a) Sample Complexity and Overall Accuracy:* In order to test sample complexity in learning or fitting, we generated data for a full training dataset of 100,000 trajectories. We then fit our models on different fractions of the training dataset: $1, 0.25, 0.05, 0.01$. We repeated this process with 5 different random seeds and computed the $L2$ error of the model over a the various dataset splits seen by the model in Table I. The error regions are the standard error of the errors over the various seeds.

We also see that with small amounts of data, the NDS models greatly outperform the Neural ODE, but with the full dataset, their performances get closer. This makes sense as the Neural ODE is likely able to infer the structure of the system with large amounts of data. Also, the Fully Connected Neural ODE outperforms the other baselines, which we posit is due to the fact that it implicitly represents that this system as a continuous time dynamical process and should change in a continuous fashion. From a sample-complexity perspective it makese sense that the better initialization of NDS should matter most when data is limited. A table of the full results of all experiments can be seen in Table I.

We notice that the NDS0 slightly outperforms the NDS with higher variance on these systems. Since it has a perfect model of the system, the residual components aren't necessary for the model to perform well, however, there is no way the network can 'correct' for a bad estimate.

Curiously, we see on the ballistic system that the partial NDS slightly outperforms the full NDS in the small data setting, but they converge to similar performance with slightly more data. A potential explanation for this is that errors propagate through the dynamical model when the parameters are wrong, while the partial systems naturally dampen errors since, for example, $\dot{z}$ only depends on the other components through a neural network. Concretely this might look like a full NDS predicting the wrong Rayleigh number $\sigma$ which might give errors to $y$ which would then propagate to $x$ and
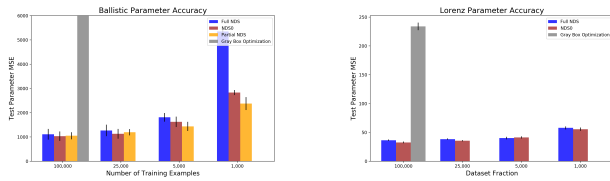
Fig. 2. $L2$ **distance between** $\phi$ **and** $\hat{\phi}$**.** As the NDS are trained under the usual $L2$ supervision, the parameters $\hat{\phi}$ of the system approach the correct values.

$y$. Conversely, this wouldn't happen as easily in a partial NDS because there are neural networks intermediating the components of the system. We also see APHYNITY (which fits a min-error parameter and is otherwise very similar to NDS) performs worse than NDS in this setting.

*b) Parameter Learning without Explicit Supervision:* For experiments in Figure 2, we stored the parameter estimates $\hat{\phi}$ for the NDS and gray box models and compared them to the true values to see how they perform in identification rather than prediction. None of these models were ever supervised with the true parameters. We see in Figure 2 that the NDS is better able to estimate the parameter values than the gray-box method for both systems tested. We believe this is because our method is able to leverage many trajectories to infer the parameters whereas the gray-box method only uses the single trajectory.

*C. Fusion Experiments*

We explored the concept of approximate system dynamics in a simplified fusion system. We predict the state of the tokamak as summarized by its stored energy and rotational frequency given the time series of control input in the form of injected power and torque. As mentioned in Section IV, we have a simplified physical model given by Equation 6 that approximately gives the dynamics of these quantities and how they relate to one another through time. Though there is a lot of remaining work to apply this model in a real experiment, approaches merging theoretical models with data to make useful predictions can be embedded into useful controller designs and improve the state of fusion.

Our full dataset consisted of 17,686 trajectories, which we randomly partitioned into 1000 as a test set and 16,686 as a training set.[1] The data are measured from the D-IIID tokamak via magnetic reconstruction [15] and charge-exchange recombination spectroscopy [19]. Similar to our synthetic experiments, we cut each trajectory into overlapping 48 timestep sections and train on 32 timesteps to predict 16 timesteps. We compare with the same models as in the previous section, but using our Fusion Neural Dynamical System as described in Equation 2 with $g$ given by Equation 6. As we discussed above, the dynamics in this equation are approximate. To illustrate this, we have included the accuracy

[1]Data is loaded and partially processed within the OMFIT framework [27]. We used the "SIGNAL_PROCESSING" module which has recently been developed for this task and is publicly available on the "profile_prediction_data_processing" branch of the OMFIT source code. Details of the preprocessing are in the Appendix.

of the naive dynamics with no learning on our data with fixed confinement times $\tau_e = \tau_m = 0.1$s as the Nominal Fusion Model in Table II. We use a larger fully connected network with 6 layers with 512 hidden nodes to attempt to capture the added complexity of the problem.

*a) Sample Complexity and Overall Accuracy:* When comparing our NDS models, the machine learning baselines, the system ID baselines, and a nominal model from [3], we see that the Fusion NDS model performs best by a large margin. Although the fully connected neural ODE performs competitively, it fails to reach the same performance. We speculate that the dynamical model helps with generalization whereas the fully connected network may overfit the training data and fail to reach good performance on the test set. Here the NDS0 is unable to perform well compared to the NDS, as the approximate dynamics mean that the model error is somewhat catastrophic for predictions. We see however that the NDS0 outperforms the Nominal Physics Model as it is able to estimate the parameters for each example rather than fixing values of the parameters for the whole dataset. Similarly, APHYNITY outperforms the nominal model but underperforms the NDS0, suggesting that the online parameter estimation component is necessary for good performance.

We see these results as highly encouraging and will continue exploring uses of NDS in fusion applications.

*D. Control Experiment*

We also explored the use of these models for control purposes using model-predictive control [7]. For this purpose, we modified the Cartpole problem from [5] so that there are a variety of parameter values for the weight of the cart and pole as well as pole length. Typically, a 'solved' cartpole environment would imply a consistent performance of 200 from a control algorithm. However, there are three factors that make this problem more difficult. First, in order to allow each algorithm to identify the system in order to make appropriate control decisions, we begin each rollout with 8 random actions. The control never fails at this point but would certainly fail soon after if continued. Second, the randomly sampled parameters per rollout make the actual control problem more difficult as the environment responds less consistently to control. For example, MPC (for an explanation of MPC, see Appendix B) using the typical Cartpole environment as a model results in rewards of approximately 37. Third, all training data for these algorithms uses random actions with no exploration, which has been seen to degrade the performance of most model-based RL or control algorithms [29].

We then trained dynamics models on this 'EvilCartpole' enviroment for each of our comparison algorithms on datasets of trajectories on the environment with random actions. At that point, we rolled out trajectories on our EvilCartpole environment using MPC with control sequences and random shooting with 1,000 samples and a horizon of 10 timesteps. The uncertainties are standard errors over 5 separately trained models.

| System | Lorenz | | | | Ballistic | | | |
|---|---|---|---|---|---|---|---|---|
| Samples | 100,000 | 25,000 | 5,000 | 1,000 | 100,000 | 25,000 | 5,000 | 1,000 |
| FC | $1.06 \pm 0.002$ | $1.39 \pm 0.003$ | $1.694 \pm 0.002$ | $4.692 \pm 0.6$ | $7.8 \pm 1.4$ | $8.3 \pm 1.6$ | $9.2 \pm 2.5$ | $13.4 \pm 3.4$ |
| FC NODE | $1.205 \pm 0.01$ | $1.233 \pm 0.01$ | $1.27 \pm 0.01$ | $1.917 \pm 0.12$ | $2.53 \pm 0.2$ | $2.6 \pm 0.4$ | $4.8 \pm 0.7$ | $9.7 \pm 1.3$ |
| Full NDS | $\mathbf{1.004 \pm 0.06}$ | $\mathbf{1.087 \pm 0.06}$ | $1.14 \pm 0.05$ | $1.42 \pm .05$ | $1.2 \pm 0.1$ | $1.4 \pm 0.2$ | $1.5 \pm 0.2$ | $4.23 \pm 0.3$ |
| Partial NDS | $1.036 \pm 0.03$ | $\mathbf{1.064 \pm 0.1}$ | $\mathbf{1.12 \pm 0.06}$ | $\mathbf{1.39 \pm 0.04}$ | $\mathbf{1.05 \pm 0.1}$ | $\mathbf{1 \pm 0.03}$ | $\mathbf{1.48 \pm 0.02}$ | $\mathbf{1.9 \pm 0.15}$ |
| NDS0 | $\mathbf{1 \pm 0.03}$ | $\mathbf{1.075 \pm 0.1}$ | $1.13 \pm 0.11$ | $1.36 \pm 0.17$ | $1.2 \pm 0.06$ | $\mathbf{1.35 \pm 0.14}$ | $1.45 \pm 0.18$ | $4.3 \pm 0.6$ |
| APHYNITY | $1.08 \pm 0.03$ | $1.17 \pm 0.05$ | $1.23 \pm 0.04$ | $1.61 \pm 0.07$ | $1.7 \pm 0.1$ | $1.75 \pm 0.13$ | $1.94 \pm 0.18$ | $6.2 \pm 0.4$ |
| LSTM | $4.98 \pm 0.01$ | $5.98 \pm 0.3$ | $5.99 \pm 0.3$ | $6.13 \pm 0.4$ | $8.5 \pm 1.6$ | $9.2 \pm 1.5$ | $10.1 \pm 2.1$ | $14.9 \pm 1.9$ |
| SR | $2.3 \pm 0.6$ | n/a | n/a | n/a | $3.5 \pm 0.3$ | n/a | n/a | n/a |
| GBO | $2.8 \pm 0.4$ | n/a | n/a | n/a | $2.94 \pm 0.3$ | n/a | n/a | n/a |

TABLE I

SAMPLE COMPLEXITY RESULTS AS DISCUSSED IN SECTION V-B. HERE, THE VALUES ARE NORMALIZED BY THE SMALLEST REPORTED VALUE FOR COMPARISON PURPOSES.

| Model | $L2$ Error on the Fusion Test Set |
|---|---|
| FC | $4.02 \pm 0.27$ |
| FC NODE | $1.71 \pm 0.11$ |
| Nominal Fusion Model | $2.89$ |
| NDS with Approximate Dynamics | $\mathbf{1 \pm 0.06}$ |
| NDS0 | $1.85 \pm 0.09$ |
| APHYNITY | $1.21 \pm 0.15$ |
| LSTM | $5.23 \pm 0.43$ |
| SR | $5.26 \pm 0.35$ |
| GBO | $2.98$ |

TABLE II

THE PERFORMANCE OF OUR COMPARISON MODELS ON THE NUCLEAR FUSION PROBLEM, AS DISCUSSED IN SECTION V-C. WE AGAIN NORMALIZE BY THE SMALLEST VALUE FOR EASE OF COMPARISON.

| | MSE of Model | | MPC Returns | |
|---|---|---|---|---|
| # Train | 5K | 1K | 5K | 1K |
| FC | $0.031 \pm 0.009$ | $0.058 \pm 0.018$ | $52 \pm 3$ | $41 \pm 4$ |
| FC NODE | $0.028 \pm 0.011$ | $0.049 \pm 0.013$ | $55 \pm 4$ | $46 \pm 3$ |
| LSTM | $0.081 \pm 0.023$ | $0.092 \pm 0.025$ | $23 \pm 6$ | $25 \pm 8$ |
| Full NDS | $\mathbf{0.020 \pm 0.006}$ | $\mathbf{0.029 \pm 0.007}$ | $\mathbf{72 \pm 4}$ | $\mathbf{60 \pm 3}$ |
| Partial NDS | $\mathbf{0.022 \pm 0.009}$ | $\mathbf{0.033 \pm 0.011}$ | $\mathbf{69 \pm 8}$ | $55 \pm 6$ |
| NDS0 | $\mathbf{0.023 \pm 0.013}$ | $\mathbf{0.028 \pm 0.014}$ | $\mathbf{71 \pm 11}$ | $57 \pm 8$ |
| SR | $0.037 \pm 0.023$ | $0.041 \pm 0.015$ | $65 \pm 4$ | $56 \pm 4$ |
| GBO | $0.046 \pm 0.019$ | n/a | $49 \pm 5$ | n/a |

TABLE III

MODELING AND CONTROL ON THE EVILCARTPOLE SYSTEM.

As shown in Table III, the NDS algorithms outperform all baselines on the cartpole task for both the modeling and control objectives. We see that all algorithms degrade in performance as the amount of data is limited. We notice however that with larger amounts of data (we performed other experiments with 25,000 and 100,000 samples) the Fully Connected and Neural ODE models perform as well as the NDS models. We hypothesize that this is due to the fact that the cartpole dynamics are ultimately not that complicated and with sufficient data unstructured machine learning algorithms can learn the appropriate dynamics to reach a modestly performing controller as well as NDS.

## VI. CONCLUSION

In conclusion, we give a framework that merges theoretical dynamical system models with deep learning by backpropagating through a numerical ODE solver. This framework succeeds even when there is a partial or approximate model of the system. We show there is an empirical reduction in sample complexity and increase in accuracy on two synthetic systems and on a real nuclear fusion dataset. In the future, we wish to expand upon our work to make more sophisticated models in the nuclear fusion setting as we move toward practical use. We also hope to explore applications of this framework in other area which have ODE-based models of systems.

REFERENCES

[1] I. Ayed et al. "Learning Dynamical Systems from Partial Observations". In: *arXiv:1902.11136 [physics]* (Feb. 2019).

[2] E. de Bezenac, A. Pajot, and P. Gallinari. "Deep Learning for Physical Processes: Incorporating Prior Scientific Knowledge". In: *arXiv:1711.07970 [cs, stat]* (Jan. 2018). arXiv: 1711.07970.

[3] M. D. Boyer, S. Kaye, and K. Erickson. "Real-time capable modeling of neutral beam injection on NSTX-U using neural networks". en. In: *Nuclear Fusion* 59.5 (Mar. 2019), p. 056008. ISSN: 0029-5515.

[4] M. D. Boyer et al. "Feedback control of stored energy and rotation with variable beam energy and perveance on DIII-D". en. In: *Nuclear Fusion* 59.7 (May 2019), p. 076004. ISSN: 0029-5515.

[5] G. Brockman et al. *OpenAI Gym.* 2016. arXiv: 1606. 01540 [cs.LG].

[6] S. L. Brunton, J. L. Proctor, and J. N. Kutz. "Discovering governing equations from data: Sparse identification of nonlinear dynamical systems". en. In: (Sept. 2015). DOI: 10.1073/pnas.1517384113.

[7] E. F. Camacho and C. B. Alba. *Model predictive control.* Springer Science & Business Media, 2013.

[8] R. T. Q. Chen et al. "Neural Ordinary Differential Equations". In: *Neurips* (2018). arXiv: 1806.07366.

[9] S. Chen, S. A. Billings, and P. M. Grant. "Nonlinear system identification using neural networks". In: *International Journal of Control* 51.6 (Jan. 1990), pp. 1191–1214. ISSN: 0020-7179. DOI: 10.1080/00207179008934126.

[10] Z. Chen et al. "Symplectic Recurrent Neural Networks". In: *arXiv:1909.13334 [cs, stat]* (Sept. 2019). arXiv: 1909.13334.

[11] K. Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *arXiv:1805.12114 [cs, stat]* (May 2018). arXiv: 1805.12114.

[12] T. F. Coleman and Y. Li. "An Interior Trust Region Approach for Nonlinear Minimization Subject to Bounds". In: *SIAM Journal on Optimization* 6.2 (May 1996), pp. 418–445. (Visited on 05/23/2020).

[13] N. Cressie and C. K. Wikle. *Statistics for Spatio-Temporal Data*. en. Google-Books-ID: 4L_dCgAAQBAJ. John Wiley & Sons, Nov. 2015. ISBN: 978-1-119-24304-5.

[14] J. R. Dormand and P. J. Prince. "A family of embedded Runge-Kutta formulae". en. In: *Journal of Computational and Applied Mathematics* 6.1 (Mar. 1980), pp. 19–26. (Visited on 01/31/2020).

[15] J. R. Ferron et al. "Real time equilibrium reconstruction for tokamak discharge control". en. In: *Nuclear Fusion* 38.7 (July 1998), pp. 1055–1066.

[16] J. Frøyland and K. H. Alfsen. "Lyapunov-exponent spectra for the Lorenz model". In: *Physical Review A* 29.5 (May 1984), pp. 2928–2931.

[17] A. Graves. "Generating Sequences With Recurrent Neural Networks". en. In: (Aug. 2013). URL: https://arxiv.org/abs/1308.0850v5.

[18] V. Le Guen et al. *Augmenting Physical Models with Deep Networks for Complex Dynamics Forecasting*. 2021. arXiv: 2010.04456 [stat.ML].

[19] S. R. Haskey et al. "Active spectroscopy measurements of the deuterium temperature, rotation, and density from the core to scrape off layer on the DIII-D tokamak (invited)". In: *Review of Scientific Instruments* 89.10 (Aug. 2018), p. 10D110.

[20] S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". EN. In: *Neural Computation* (Nov. 1997).

[21] M. Janner et al. "When to Trust Your Model: Model-Based Policy Optimization". In: *Neurips* (2019). arXiv: 1906.08253.

[22] J. Kates-Harbeck, A. Svyatkovskiy, and W. Tang. "Predicting disruptive instabilities in controlled fusion plasmas through deep learning". en. In: *Nature* 568.7753 (Apr. 2019), pp. 526–531. ISSN: 1476-4687.

[23] Lennart Ljung et al. "Developments in The Math-Works System Identification Toolbox". en. In: *IFAC Proceedings Volumes* 42.10 (2009), pp. 522–527. ISSN: 14746670.

[24] E. N. Lorenz. "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences* 20.2 (Mar. 1963), pp. 130–141.

[25] G. Manek and J. Z. Kolter. "Learning Stable Deep Dynamics Models". en. In: (Jan. 2020).

[26] V. Mehta et al. "Neural Dynamical Systems". en. In: Feb. 2020. URL: https://openreview.net/forum?id=Rsmqn9R2Mg (visited on 03/16/2021).

[27] O. Meneghini et al. "Integrated modeling applications for tokamak experiments with OMFIT". en. In: *Nuclear Fusion* 55.8 (July 2015), p. 083008. (Visited on 01/25/2020).

[28] B. Ph. van Milligen, V. Tribaldos, and J. A. Jiménez. "Neural Network Differential Equation and Plasma Equilibrium Solver". In: *Physical Review Letters* 75.20 (Nov. 1995), pp. 3594–3597.

[29] M. C. Mozer and J. Bachrach. "Discovering the Structure of a Reactive Environment by Exploration". In: *Advances in Neural Information Processing Systems* 2. 1990, pp. 439–446.

[30] A. Nagabandi et al. "Deep Dynamics Models for Learning Dexterous Manipulation". In: *arXiv:1909.11652 [cs]* (Sept. 2019). arXiv: 1909.11652.

[31] A. Nagabandi et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *arXiv:1708.02596 [cs]* (Dec. 2017). arXiv: 1708.02596. URL: http://arxiv.org/abs/1708.02596 (visited on 04/24/2020).

[32] O. Nelles. *Nonlinear System Identification: From Classical Approaches to Neural Networks and Fuzzy Models*. en. Springer Science & Business Media, Mar. 2013. ISBN: 978-3-662-04323-3.

[33] G. D. Portwood et al. "Turbulence forecasting via Neural ODE". In: *arXiv:1911.05180 [physics]* (Nov. 2019). arXiv: 1911.05180.

[34] C. Rackauckas et al. "Universal Differential Equations for Scientific Machine Learning". In: *CoRR* abs/2001.04385 (2020). arXiv: 2001.04385. URL: https://arxiv.org/abs/2001.04385.

[35] S. Ross and J. A. Bagnell. "Agnostic System Identification for Model-Based Reinforcement Learning". In: *arXiv:1203.1007 [cs, stat]* (July 2012).

[36] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton. "Deep learning of dynamics and signal-noise decomposition with time-stepping constraints". en. In: *Journal of Computational Physics* 396 (Nov. 2019), pp. 483–506.

[37] C. Runge. "Ueber die numerische Auflösung von Differentialgleichungen". de. In: *Mathematische Annalen* 46.2 (June 1895), pp. 167–178. (Visited on 06/03/2020).

[38] J. Schrittwieser et al. "Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model". In: *arXiv:1911.08265 [cs, stat]* (Nov. 2019). arXiv: 1911.08265.

[39] B. Sohlberg and E. W. Jacobsen. "GREY BOX MODELLING – BRANCHES AND EXPERIENCES". en. In: *IFAC Proceedings Volumes*. 17th IFAC World Congress 41.2 (Jan. 2008), pp. 11415–11420.

[40] P. Virtanen et al. "SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python". In: *arXiv:1907.10121 [physics]* (July 2019). arXiv: 1907.10121.

[41] T. Wang and J. Ba. "Exploring Model-based Planning with Policy Networks". In: *arXiv:1906.08649 [cs, stat]* (June 2019).