JAMES | Journal of Advances in Modeling Earth Systems

RESEARCH ARTICLE
10.1029/2021MS002554

Key Points:

- Deep learning models trained on simulation data can learn the dynamics of Earth science simulators
- Deep learning models also learn the input–output derivatives of the state-update function, which are unavailable for many simulators
- We show on Lorenz-96 that these learned derivatives can be used directly for data assimilation and parametrization tuning

Supporting Information:

Supporting Information may be found in the online version of this article.

Correspondence to:

M. Nonnenmacher,
marcel.nonnenmacher@hereon.de

Citation:

Nonnenmacher, M., & Greenberg, D. S. (2021). Deep emulators for differentiation, forecasting, and parametrization in Earth science simulators. *Journal of Advances in Modeling Earth Systems*, *13*, e2021MS002554. https://doi.org/10.1029/2021MS002554

Received 7 APR 2021
Accepted 2 JUN 2021

Deep Emulators for Differentiation, Forecasting, and Parametrization in Earth Science Simulators

Marcel Nonnenmacher[1] **and David S. Greenberg**[1]

[1]Institute of Coastal Systems, Helmholtz-Zentrum Hereon, Geesthacht, Germany

Abstract To understand and predict large, complex, and chaotic systems, Earth scientists build simulators from physical laws. Simulators generalize better to new scenarios, require fewer tunable parameters, and are more interpretable than nonphysical deep learning, but procedures for obtaining their derivatives with respect to their inputs are often unavailable. These missing derivatives limit the application of many important tools for forecasting, model tuning, sensitivity analysis, or subgrid-scale parametrization. Here, we propose to overcome this limitation with deep emulator networks that learn to calculate the missing derivatives. By training directly on simulation data without analyzing source code or equations, this approach supports simulators in any programming language on any hardware without specialized routines for each case. To demonstrate the effectiveness of our approach, we train emulators on complete or partial system states of the chaotic Lorenz-96 simulator and evaluate the accuracy of their dynamics and derivatives as a function of integration time and training data set size. We further demonstrate that emulator-derived derivatives enable accurate 4D-Var data assimilation and closed-loop training of parametrizations. These results provide a basis for further combining the parsimony and generality of physical models with the power and flexibility of machine learning.

Plain Language Summary Many Earth science simulators are implemented as monolithic programs that calculate changes in the state of a system over time. In many cases, using or improving these simulators also requires the derivatives of their outputs with respect to inputs, which describe how future states depend on past states. These derivatives can be difficult or costly to compute. Several recent studies have applied deep learning (DL) to simulation data to construct emulators of their dynamics. Here, we use the fact that DL models can be easily and automatically differentiated to obtain approximate derivatives of the original simulator and test this idea on a simple and common chaotic model of the atmosphere. We verify in several experiments that the emulator derivatives, which require neither additional training nor extensive postprocessing to obtain, can indeed be used as a valid substitute for the derivatives of the simulator.

1. Introduction

Earth science deals with physical systems, such as the atmosphere and ocean, that are large, complex, and chaotic, with many parts and processes interacting over multiple time scales. To develop and express quantitative insights into these systems, Earth scientists build simulators (or simply "models"). These intricate software packages incorporate physical laws, mathematical approximations, empirical relationships, and discretization and integration schemes for partial differential equations (PDEs). Repeated dynamical updates of a system state are used to explore the consequences of modeling assumptions and initial conditions or compared to observations for evaluation, tuning, and forecasting. This approach has aided both fundamental understanding and practical applications such as short- and medium-range forecasting (Bauer et al., 2015), identification of potential commercial and human impacts, and validation of assumptions for long-term climate modeling (Eyring, Gleckler, et al., 2016; Maher et al., 2019).

Many techniques for working with quantitative data require derivatives of these Earth science simulators. The state-update function of a simulator, integrating the system state from one time point to the next, constitutes an input–output function, and we are interested in the derivatives of this function with respect to its inputs. However, even in cases where simulator software implements mathematical formulas that are themselves differentiable, we often find ourselves at a loss: providing and maintaining routines simulator differentiation is a laborious task, and development resources are limited. Thus, none of the 108 CMIP6

models (Eyring, Bony, et al., 2016) provide derivatives, and even advanced operational models use approximations and lower resolutions to do so (Trémolet, 2004).

Three main strategies presently exist for obtaining simulator derivatives. First, for simple simulators with low-dimensional system states, finite differences have been used (Smith et al., 1985) but are inefficient and numerically fraught and scale poorly. Second, gradient calculation routines have been implemented for specific simulators such as numerical weather prediction systems (Wedi et al., 2015; Weng & Liu, 2003), and some tools exist for automating this task (Bischof et al., 1992). These "by-hand" routines can be efficient and stable but are cumbersome to maintain, require immense effort for realistic Earth science simulators, and limit experimentation with new models, parametrizations, or discretizations. Third, the entire simulator can be reimplemented in an automatic differentiation ("autodiff") framework (Linnainmaa, 1976), which uses a fixed set of numerical operations to support automatic differentiation. While a promising long-term strategy (Holl et al., 2020; Hu et al., 2019; Rackauckas & Nie, 2017), this has not been demonstrated for realistic simulators and would discard decades of software development.

Here, we pursue a different approach for calculating missing derivatives: "translating" the simulator into an autodiff environment as an *emulator*, a deep neural network (NN) that learns from simulations. Such emulators have been studied in Earth science for their potential to provide faster and computationally cheaper numerical simulations, sensitivity analysis, and model uncertainty estimates (Dueben & Bauer, 2018; Fablet et al., 2018; Reichstein et al., 2019). We show that the trained emulator's autodiff derivatives closely match the missing derivatives of the original simulator, without requiring gradient routines or autodiff support for simulator code. This approach can leverage the mature autodiff environments developed and tested by the machine learning (ML) community (Abadi et al., 2016; Bezanson et al., 2017; Hu et al., 2019; Paszke et al., 2019; Rackauckas et al., 2018).

We develop and test our approach using the Lorenz-96 model (L96) (Lorenz, 1996), a system of nonlinear differential equations modeling atmospheric chaos and a standard benchmark for data analysis and ML tasks in Earth science (Brajard et al., 2020; Dueben & Bauer, 2018). L96 is a good test case for our purposes because it can be precisely differentiated without ML to measure the accuracy of emulator derivatives. We systematically investigate how accuracy depends on integration time and training data set size and show how emulators can learn from small system state fragments. We also demonstrate how designing the emulator's architecture to match the simulator's mathematical structure improves accuracy and data efficiency (Bocquet et al., 2019). Our work incorporates insights and techniques from previous studies using NNs to predict or emulate Earth science simulators (Chattopadhyay et al., 2019; Dueben & Bauer, 2018; Fablet et al., 2018; Scher & Messori, 2019) but is to our knowledge the first to estimate derivatives through simulation-trained emulators.

We further demonstrate the utility of emulator derivatives for two important downstream applications in Earth science. We first test strong-constraint 4D-Var (Lorenc, 1986), a simulation-based data assimilation technique (Carrassi et al., 2018; Ide et al., 1997) that estimates system states from noisy and incomplete observations. It is used in state-of-the-art weather prediction systems (Wedi et al., 2015) but requires the simulator's derivatives (Courtier & Rabier, 1997; Errico, 1997). We show that emulator derivatives can produce 4D-Var forecasts as accurately as true simulator derivatives.

We also apply emulator derivatives for learning parametrizations, corrective terms approximating physical, chemical, or biological processes not spatially resolved in the system state (Kain & Fritsch, 1993; Stensrud, 2009). Designing these requires extensive work and domain expertise (Gross et al., 2018; Hourdin et al., 2016), and DL-based replacements have garnered considerable attention (Karpatne et al., 2017; Reichstein et al., 2019; Schneider et al., 2017). Without available simulator derivatives, parametrizations are trained on their immediate inputs and desired outputs before being coupled to the simulator (Brenowitz et al., 2020; Gross et al., 2018; Seifert & Rasp, 2020). This "offline" training mode cannot account for simulator–parametrization interactions at runtime, potentially leading to unrealistic behavior and numerical instabilities (Rasp, 2020; Yuval & O'Gorman, 2020). We show that emulator derivatives allow highly accurate parametrizations to be learned in an online, "solver-in-the-loop" mode that was previously restricted to a narrow set of simulators with available derivatives (Obiols-Sales et al., 2020; Ramadhan et al., 2020; Sanchez-Gonzalez et al., 2020; Um et al., 2020).

2. Methods

2.1. Simulators

We consider a simulator with time-varying system state $\mathbf{x} \in \mathbb{R}^K$ (e.g., pressure or temperature across a spatial grid). The simulator uses an explicit, fixed-time-step numerical scheme to integrate tendencies f that are functions of the current state $\mathbf{x}_t$, resulting in a deterministic state update $\mathcal{M}$:

$$\left.\frac{d\mathbf{x}}{dt}\right|_{\mathbf{x}=\mathbf{x}(t)} = f(\mathbf{x}(t)) \tag{1}$$

$$\mathbf{x}(t+\Delta) = \mathcal{M}(\mathbf{x}(t)) \approx \mathbf{x}(t) + \int_t^{t+\Delta} f(\mathbf{x}(t'))\, dt' \tag{2}$$

with step-size $\Delta > 0$. The integration scheme can be forward Euler ($\mathcal{M}(\mathbf{x}) = \mathbf{x} + \Delta f(\mathbf{x})$) or a higher-order method (cf., Supporting Information Text S2). A *simulation* is a state sequence $\{\mathbf{x}(t_0), \mathbf{x}(t_0+\Delta), ..., \mathbf{x}(t_0+n\Delta) ..., \mathbf{x}(t_0+N\Delta)\}$ generated by repeated application of $\mathcal{M}$. For fixed step-size, we will denote the state sequence as $\{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n, ..., \mathbf{x}_N\}$ with $\mathbf{x}(t_0+n\Delta) = \mathbf{x}_n$.

2.2. Problem Statement

Our task is to obtain *simulator derivatives*. Given an initial state $\mathbf{x}_0$ and time delay $n\Delta$, we must calculate derivatives of the future state $\mathbf{x}_n$ with respect to $\mathbf{x}_0$, given by the input–output Jacobian matrix of the iterated state update $\mathcal{M}$:

$$J_{\mathcal{M}}(\mathbf{x}_0, n) = \frac{d\mathbf{x}_n}{d\mathbf{x}_0} = \prod_{m=0}^{n-1} \frac{d\mathbf{x}_{m+1}}{d\mathbf{x}_m} = \prod_{m=0}^{n-1} \left(\left.\frac{d\mathcal{M}(\mathbf{x})}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_m} \right) \tag{3}$$

We assume at least one simulation is available, and the integration scheme and Δ are known. However, to facilitate flexible application to Earth science models without extensive analysis of governing equations, discretizations or source code, we **do not** assume code or formulas are available for f, or that we can freely evaluate f or $\mathcal{M}$ on new inputs (see Section 4). Thus, we must differentiate the unknown state-update function based on a fixed data set it has generated.

2.3. Emulators

Our overall strategy is to estimate simulator Jacobians using *emulator* Jacobians. Essentially, a NN emulator learns to reproduce the simulator's dynamics through training on simulation data, and autodiff tools are used to provide the derivatives of emulator outputs with respect to the inputs.

For a simulator with a K-dimensional system state, our emulator is a NN with K inputs and outputs, trainable parameters ϕ, and input–output function $\hat{f}_\phi$ that estimates the simulator tendencies f. While we have not assumed knowledge of $f(\mathbf{x})$ for any simulation state, we can plug our emulator into the simulator's integration scheme and compare the resulting state update $\widehat{\mathcal{M}}_\phi$ to simulations. For example, forward Euler integration yields $\widehat{\mathcal{M}}_\phi(\mathbf{x}) = \mathbf{x} + \Delta\hat{f}_\phi(\mathbf{x})$.

We train the emulator (details in Supporting Information Text S3) by minimizing the objective function:

$$\mathcal{L}_{\mathrm{DYN}}(\phi) = \sum_{s,n} \|\mathbf{x}^s_{n+1} - \widehat{\mathcal{M}}_\phi(\mathbf{x}^s_n)\|_2^2 = \sum_{s,n,k} \left(x^s_{n+1,k} - \widehat{\mathcal{M}}_\phi(\mathbf{x}^s_n)_k\right)^2 \tag{4}$$

where $x^s_{n+1,k} \in \mathbb{R}$ is element k of the system state $\mathbf{x}^s_n$ at time step n in simulation s. After training $\hat{f}_\phi$, we plug it into an integration scheme to obtain $\widehat{\mathcal{M}}_\phi$. We then use autodiff to minimize $\mathcal{L}_{\mathrm{DYN}}$, applying the chain rule in Equation 3 efficiently through autodiff backpropagation.

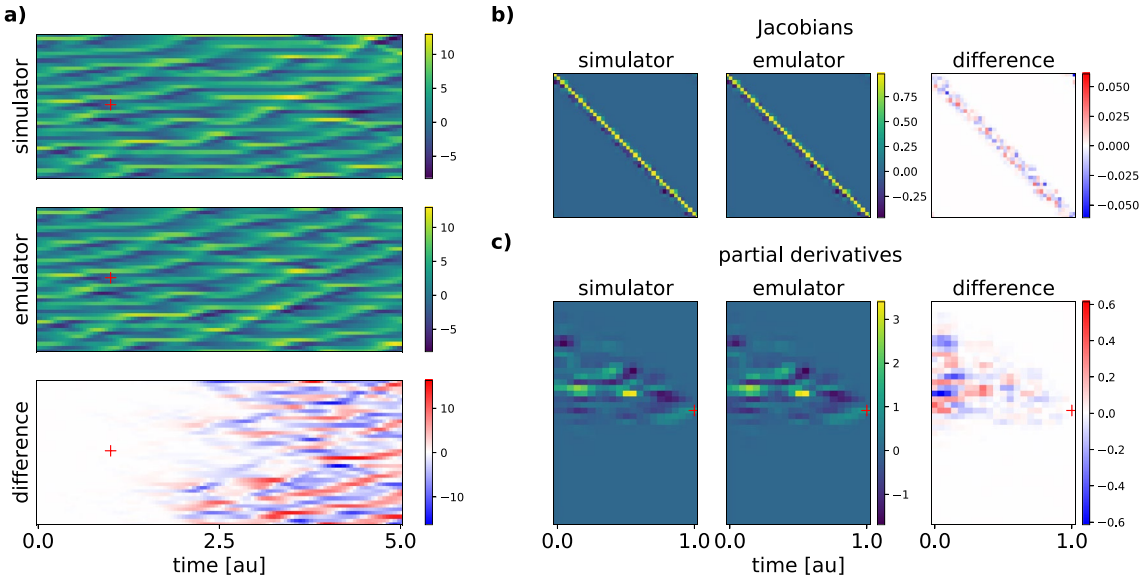

Figure 1. Emulators for differentiable dynamics. (a) System state sequences generated by numerical integration of time derivatives from a 40-dimensional L96 simulator (upper) and emulator (center) with the same initial conditions, and resulting differences (lower). The emulator was trained on 1,200 time steps at $\Delta = 0.05$. (b) Emulator outputs are differentiable functions of their inputs. Derivatives $\partial \mathbf{x}(t+\Delta) \,/\, \partial \mathbf{x}(t) \in \mathbb{R}^{40\times 40}$ for both simulator and trained emulator and the initial state from (a). (c) Sensitivity analysis demonstrating emulator differentiability over multiple time steps. Partial derivatives $\partial \mathbf{x}(1)_{20} / \partial \mathbf{x}(t)_k$ of the system state at location 20 and time $t = 1$ au (red cross in (a)), with respect to all locations k and at previous times t. Differences (right) between simulator and emulator derivatives increase with the number of differentiated time steps.

2.4. Model Architectures for Emulation

When we have no knowledge about the simulator beyond the above assumptions, fully connected deep NNs provide a generic, flexible option for $\hat{f}_\phi$. However, partial knowledge about the tendency functions f or the space of system states $\mathbf{x}$ can often be easily obtained from the simulator's description or documentation without extensive analysis of its equations or code.

In the present work, our emulator architectures incorporate knowledge about three common properties in simulators of physical systems. First, when the elements of the system state are *spatially structured* by assigning each element a location on a regular spatial grid, we use convolutions to increase efficiency, accuracy, and scalability. Second, when the simulator tendencies exhibit *local dependence*, such that $f(\mathbf{x})_k$ depends only on the system state in a local neighborhood U_k around $\mathbf{x}_k$, we impose the same structure on the emulator. To do this, we limit the number of convolutions with kernel width >1, so that the remaining convolutions operate independently at each grid point. Third, we explore the use of specialized activation functions that *mimic mathematical operations* used by the simulator (Figure S1). Full details of all networks are provided in Supporting Information Text S4, and code is available at github.com/m-dml/emulator_L96/.

3. Experiments

We trained emulators on simulation data, evaluated their accuracy, and tested them in downstream applications using the L96 simulator, a common benchmark model in Earth science.

3.1. Test Case: L96 Simulator

We generated simulations from the one-level Lorenz-96 simulator (Lorenz, 1996). This simulator exhibits wave-like patterns that interact nonlinearly while moving persistently clockwise through the spatially structured, 1-D, periodic system state (Figure 1a, top).

Each element of the state's tendency is a function of the current state $x_k = [\mathbf{x}(t)]_k$ at that location k, together with up to two neighboring state values in either direction:

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F \tag{5}$$

where F is a static parameter of the simulator and we use the periodic boundary conditions $x_{K+1} = x_1$, $x_0 = x_K$, and $x_{-1} = x_{K-1}$. We used a step-size of $\Delta = 0.05$ in a fourth-order Runge-Kutta integration scheme, so that calculating the full state $\mathbf{x}_{n+1}$ from $\mathbf{x}_n$ requires four recursive applications of f. Except where otherwise noted, we used $K = 40$ and $F = 8$, which is known to result in chaotic dynamics with a leading Lyapunov exponent of $\lambda_1 \approx 1.67$ (Brajard et al., 2020).

While the purpose of our emulation framework is to provide *unavailable* simulator derivatives, for testing and measuring accuracy we can obtain the Jacobians $J_{\mathcal{M}}(\mathbf{x}_0, n)$ of the L96 simulator through hand-written routines (details in Supporting Information Text S2).

3.2. Emulation of L96 Dynamics

We trained emulators to estimate the L96 tendency function (Equation 5) for arbitrary system states $\mathbf{x}$. In designing our emulator architectures, we sought to exploit high level conceptual properties of the simulator (Section 2.4), analogous to those that can be easily identified in more complex and realistic simulators, without tailoring our network's components excessively to the simulator. First, we used periodic 1-D convolutions to capture the system states' spatial structure. Second, to the capture local dependence, we used 3×1 convolutions in the first two layers, with all subsequent layers consisting of 1×1 convolutions that operate independently across activation channels at each spatial location. We used eight-layer networks with ReLU activations (details in Supporting Information Text S4).

We also experimented with additional specializations of the emulator's connectivity structure and activation functions to more closely match the mathematical operations of Equation 5. These specializations strongly improved performance, reduced network depth and trainable parameter count, and decreased the amount of training data required (see Supporting Information Text S5, and also Bocquet et al., 2019), but we chose not to use them for our main results since such problem-tailored architectures might not be easy to identify for more complex and realistic simulators.

3.3. Accuracy of Emulations

After using L96 simulations to train the emulator, we plugged the emulator's estimate $\hat{f}_\phi$ of the tendencies i into the RK4 integration scheme to define a state update $\widehat{\mathcal{M}}_\phi$ (Bocquet et al., 2019; Wang & Lin, 1998). Starting with an initial state for a simulation not included as emulator training data (Figure 1a, top), we applied $\widehat{\mathcal{M}}_\phi$ iteratively to generate an *emulation* (or "rollout"), a new system state sequence resembling a simulation (Figure 1a, middle). Visual inspection of emulations showed a close match to the original simulation for about one Lyapunov time, beyond which the emulation systems states generated by the emulator continued to strongly resemble simulations in their amplitude, smoothness, continuity, and wave-like appearance, but with different wave positions and amplitudes. The eventual appearance of this mismatch is inevitable even when the emulator's errors approach machine precision, since L96 is a chaotic system (Brajard et al., 2020). Nonetheless, emulations were stable for over 10^5 time steps and resulted in the same distribution of system state values as the original L96 simulation (see Figure S3).

We further quantified these errors over multiple emulations as a function of integration time (Figure 2a) for networks trained on different data set sizes (N = 1,200, 4,000, or 12,000 time steps). Initial errors were much smaller than typical state values but grew over about 3 Lyapunov times until the emulation no longer matched the simulation better than a randomly chosen system state. Emulator errors on both rollouts and single state updates decreased as a function of training data set size (Figure 2b).

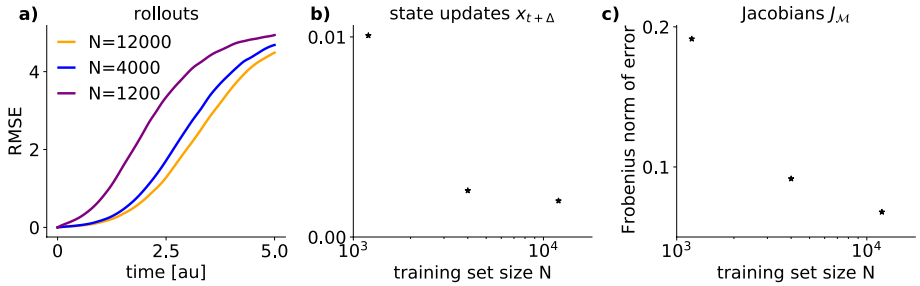

Figure 2. Accuracy of emulator-derived state updates and derivatives (averages over 1,000 initial conditions not used for training). (a) Root-mean-square error (RMSE) of numerically time-integrated emulator outputs compared to simulations, as a function of integration time and training data set size N. (b) RMSEs for the state update over a single time step, as a function of training data set size. (c) Sum of square errors for input–output Jacobians $J_{\mathcal{M}} = \partial \mathbf{x}(t + \Delta) / \partial \mathbf{x}(t)$, as a function of training data set size.

3.4. Accuracy of Emulator Derivatives

We next considered the task originally motivating emulator development: the estimation of simulator derivatives. We used the autodiff capabilities of NNs to differentiate emulator outputs with respect to their inputs and used the results as estimates of simulator derivatives. Because we have chosen a simple test case where simulator derivatives can be easily implemented by hand, we were able to compare emulator derivatives directly to their target values.

Figure 1b shows input–output Jacobians for a single time step of the simulator (right) and emulator (center). Each column corresponds to one of 40 inputs to $\mathcal{M}$ or $\widehat{\mathcal{M}}_\phi$ and each row to one output. Only values near the matrix diagonal are nonzero due to the local dependency structure: each element of f or $\hat{f}_\phi$ depends on other elements up to two positions away, and with RK4 integration each output of $\mathcal{M}$ or $\widehat{\mathcal{M}}_\phi$ depends inputs up to eight positions away. The simulator and emulator Jacobians show close agreement (Figure 1b, right), and average error decreasing as a function of training data set size (Figure 2c).

To evaluate the accuracy of emulator-derived derivatives across longer simulations, we calculated simulator and emulator derivatives of the system state at one location and time with respect to all locations up to 20 time steps in the past. The spatial and temporal structure of this dependence of future on past states was visually similar when comparing emulator and simulator derivatives (Figure 1c). As expected from the chaotic nature of L96, the size of errors relative to the derivatives grew with increasing time in the past. Due to the limited-range local dependencies built into our emulators $\widehat{\mathcal{M}}_\phi$, the size of the region with nonzero derivatives grew linearly over time into the past.

3.5. Training on Partial System States

A considerable challenge in training NNs on Earth science simulations is posed by the sheer size of their system states. For example, a global latitude/longitude grid at 0.25° spacing with 100 vertical levels contains over 10^8 locations, each of which can store a dozen or more physical quantities. In such cases, training on complete system states poses major difficulties for machine learning frameworks and the computing hardware that supports them (Kurth et al., 2018). However, this limitation can be overcome by training the emulator on partial system states. We take advantage of the fact that L96 dynamics exhibit *local dependence*, a property shared by many other simulators. For example, many Earth system models combine location-specific physical effects coupled to fluid dynamics (Gross et al., 2018). When these dynamics are integrated explicitly with discretization stencils for spatial derivatives, updates exhibit local dependence (Zängl et al., 2015).

In the specific case of L96, the time derivative $f(\mathbf{x})_k$ of the kth location in the L96 system state depends only on $x_{k-2:k+1}$, a neighborhood of four values, while for a fourth-order Runga-Kutta solver the state update $\mathcal{M}(\mathbf{x})_k$ depends on 13 values (i.e., $x_{k-8:k+4}$). By imposing the same local dependence on the emulator

computing $\hat{f}_\phi(\mathbf{x})$ through convolutional network layers, the problem of learning a function with K inputs and K outputs reduces to the much easier problem of learning a function with 13 inputs and 1 output. We investigate how to exploit this for emulator training on partial system states in Supporting Information Text S6.

3.6. 4D-Var Data Assimilation

To further test the accuracy and usefulness of emulator-derived derivatives, we applied them to data assimilation (Apte et al., 2008), aiming to identify the L96 system state sequence most consistent with noisy and incomplete observations. Observations $\mathbf{y}_n$ consist of 10 randomly locations from each $\mathbf{x}_n$ with Gaussian noise (Figure 3a, upper), reproducing the setup of Fertig et al. (2007) ($K = 40$, $\Delta = 0.0125$, $\sigma_{\text{noise}} = 1$, see supporting information Text S8).

We carried out data assimilation using the strong-constraint 4D-Var algorithm. Making use of the fact that all future states depend deterministically on $\mathbf{x}_0$, gradient-based numerical optimization is used to reduce prediction error for each $\mathbf{y}_n$ while regularizing with a prior distribution (see Supporting Information Text S8). Concretely, we seek to minimize

$$\mathcal{L}_{DA}(\mathbf{x}_0) = \sum_n \mathbf{f}_n^\top R_n^{-1} \mathbf{f}_n + \left(\mathbf{x}_0 - \mathbf{x}_f\right)^\top B^{-1} \left(\mathbf{x}_0 - \mathbf{x}_f\right) \tag{6}$$

$$\mathbf{f}_n = \mathbf{y}_n - \mathcal{H}_n\left(\mathcal{M}^{(n)}(\mathbf{x}_0)\right) \tag{7}$$

where observation error covariance matrices R_n, background error covariance matrix B, and the background state $\mathbf{x}_f$ are known, and $\mathcal{H}_n$ is known with available derivatives.

Several approaches to minimizing the loss Equation 6 exist, and not all require simulator derivatives (Desroziers et al., 2014). The gradient-based optimization of $\mathcal{L}_{DA}$ underlying the common 4D-Var algorithm however does requires derivatives of the state-update function $\mathcal{M}$, and $\frac{d\mathcal{M}(\mathbf{x})}{d\mathbf{x}}$ is unavailable for many important simulators. We therefore investigated the utility of emulation-based 4D-Var, replacing $\mathcal{M}$ by a trained emulator $\widehat{\mathcal{M}}_\phi$ and calculating $\mathcal{L}_{DA}$ in an autodiff environment. For L96, for which true simulator derivatives are available, we were able to directly compare emulator versus simulator derivatives using the same data assimilation task, algorithm, and data. Our emulators target f instead of $\mathcal{M}$, and hence they are not tied to a specific step length Δ, so we reused an emulator previously described in Section 3.2 and trained on 1,200 time steps with $\Delta = 0.05$.

4D-Var data assimilation recovered system state trajectories visually resembling true system states (Figure 3a) throughout the period of available observations, both when using state updates and derivatives from the true simulator (Figure 3c) or those from the trained emulator (Figure 3d). Errors within the observation window decreased as a function of window length (Figure 3f, and increased as expected when forecasting further into the future beyond the last observation (Figure 3g). Remarkably, we observed no difference whatsoever in forecast quality when using the emulator and its derivatives instead of the true simulator, for observations windows up to 72Δ and consecutive forecast windows up to $80\Delta = 1$au (Figures 3f and 3g). One arbitrary time unit corresponds to roughly 5 days of "weather" (Lorenz, 1996). These results demonstrate that emulation can provide missing derivatives for 4D-Var data assimilation without degrading forecast accuracy and that emulators learned for one value of Δ can be successfully applied with another.

3.7. Parametrization Learning

We further tested our emulators and their derivatives in a parametrization learning task. In Earth science, a critical and ubiquitous question ask how we can account for physical effects at spatial scales below the grid spacing of our system state. For example, how can we account for convective processes involving moisture and temperature variables $\mathbf{z}$ at spatial scales <100 m, when computational limits impose a 100 km

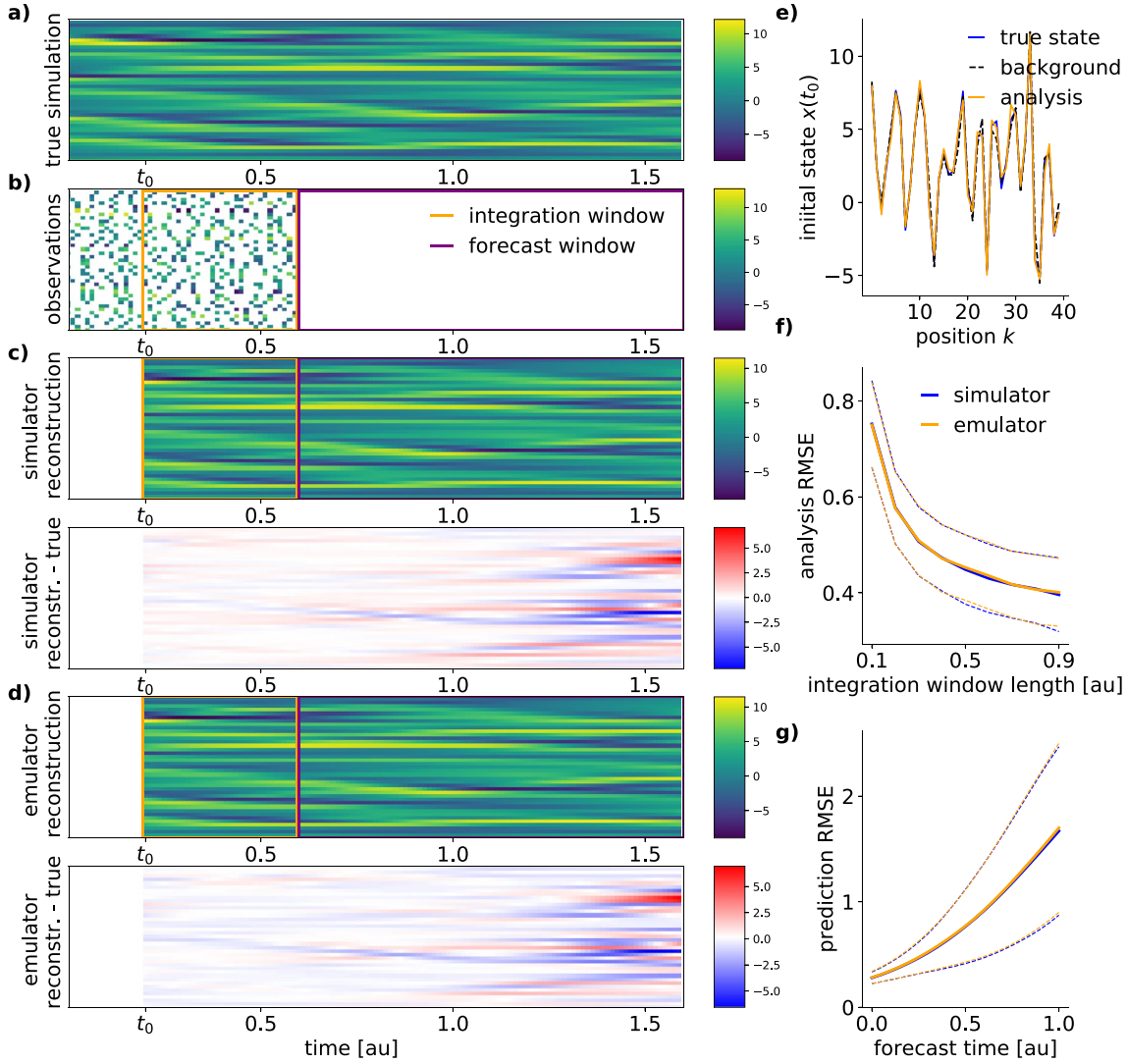

Figure 3. 4D-Var data assimilation for L96. (a) L96 simulation ($K = 40$, $\Delta = 0.0125$) providing unobserved "ground truth" system states for a forecast problem. In terms of predictability, 1 time unit corresponds to roughly 5 days of "weather." (b) Twenty-five percent of each system state is randomly observed with additive Gaussian noise. We aim to determine the state trajectory from (a) within the integration window (analysis, orange) and beyond the last measurement (forecast, purple). (c) 4D-Var reconstruction of system states from the observations in (b), using 4D-Var with the original simulator and a hand-implemented gradient calculation routine (top), with reconstruction error over time (bottom). (d) As in (c), but using state update and derivatives from an emulator trained on a separate simulation data set. (e) 4D-Var reconstruction of the initial state $\mathbf{x}(t_0)$ from the example in (a) and (b) using the emulator. (f) Mean ± 1 SD of RMSEs for $\mathbf{x}_0$ as a function of integration window lengths. (g) Mean ± 1 SD of forecast RMSEs as a function of time beyond the integration window, with window length 0.8. RMSE, root-mean-square error.

spacing on our atmospheric system state? A *parametrization* adds a corrective term to coarse-scale tendencies $\frac{d\mathbf{x}}{dt} = f(\mathbf{x})$ to mimic the effects of coupling to fine-scale variables:

$$f(\mathbf{x}) + \mathcal{B}_\psi(\mathbf{x}) \approx g(\mathbf{x},\mathbf{z})_x \tag{8}$$

where $g(\mathbf{x},\mathbf{z})_x$ denotes the tendency of the coarse variables in the full coupled model and ψ free parameters of the corrective term. $\mathcal{B}_\psi(\mathbf{x})$ typically is chosen to have local structure, meaning that for every location k, the value of $\left[\mathcal{B}_\psi(\mathbf{x})\right]_k$ depends only on x_k.

In *parametrization learning*, we seek the ψ for which Equation 8 holds as closely as possibly. One straightforward approach is to generate a fine-scale simulation $\{(\mathbf{x}_0, \mathbf{z}_0), \ldots, (\mathbf{x}_N, \mathbf{z}_N)\}$ and minimize

$\sum_n \| g(\mathbf{x}_n, \mathbf{z}_n)_x - f(\mathbf{x}_n) - \mathcal{B}_\psi(\mathbf{x}_n) \|_2^2$ over ψ. However, this "offline" training mode cannot account for coupled behaviors of $\mathcal{B}$ and f: even if Equation 8 holds closely for all $\{\mathbf{x}_n\}_{n=1}^N$ used to learn ψ, integrating $f(\mathbf{x}) + \mathcal{B}_\psi(\mathbf{x})$ will generate novel states for which Equation 8 may break down.

To address this, the parametrized model can be numerically integrated *during learning* to define a state update $\mathcal{M}_\mathcal{B}$ for dynamics $\frac{d\mathbf{x}}{dt} = f(\mathbf{x}) + \mathcal{B}_\psi(\mathbf{x})$. We minimize

$$\mathcal{L}_{PAR}(\psi) = \sum_n \sum_{r=1}^{r_{\max}} \left\| \mathbf{x}_{n+r} - \mathcal{M}_\mathcal{B}^{(r)}(\mathbf{x}_n) \right\|_2^2 \tag{9}$$

This "solver-in-the-loop" mode accounts for coupling effects and requires only coarse variables for training, halving storage requirements. However, minimizing $\mathcal{L}_{PAR}$ over multiple time steps (or one Runge-Kutta step) requires simulator derivatives.

To test emulator derivatives in a solver-in-the-loop parametrization learning task, we used a two-level L96 model with coarse and fine variables (Lorenz, 1996):

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F - h\,c\,\overline{z}_k \tag{10}$$

$$\frac{1}{c}\frac{dz_{j,k}}{dt} = -b\,z_{j+1,k}\left(z_{j+2,k} - z_{j-1,k}\right) - z_{j,k} + \frac{h}{J}x_k \tag{11}$$

with $\Delta = 0.01$, $K = 36$, $J = 10$, $F = b = c = 10$, $h = 1$ and $\overline{z}_k = \frac{1}{J}\sum_j z_{k,j}$. This model has been previously used as test for parametrization learning and is simple enough that parametrizations trained offline provide a numerically stable, reasonably accurate baseline in coupled simulations (Crommelin & Vanden-Eijnden, 2008; Orrell, 2003; Pawar & San, 2020; Rasp, 2020). Parametrizing out $\mathbf{z}$ in Equation 10, we have

$$\frac{dx_k}{dt} = -x_{k-1}\left(x_{k-2} - x_{k+1}\right) - x_k + F + \mathcal{B}_\psi(x_k) \tag{12}$$

To learn an L96 parametrization, we first trained an emulator $\hat{f}_\phi$ on 1,200 time steps of a coarse-only L96 model (Equation 5), fixed ϕ and substituted $\hat{f}_\phi$ for f in Equations 8 and 9. We then minimized $\mathcal{L}_{PAR}(\psi)$ with $r_{\max} = 10$ using autodiff on 500 time steps of fine-scale simulations as training data, with 20% held out for validation. For evaluation, we coupled the trained parametrization to the true dynamical model (Equation 12).

This "solver-in-the-loop"-trained L96 parametrization closely matched the coarse-scale variables of a two-level L96 simulation (Figure 4a). The learned parametrization also closely resembled coarse two-level L96 variables in terms of root-mean-square error (Figure 4b) and power spectral density (Figure 4c). For L96, offline- and solver-in-the-loop-trained parametrizations were strikingly similar (Figure 4d) despite different losses and training data. We emphasize that our goal was not to improve existing learned parametrizations for L96, which is sufficiently simple that offline training is adequate (Rasp, 2020). Rather, these results demonstrate that emulator derivatives allow solver-in-the-loop parametrization training without compromising accuracy.

4. Discussion

We trained differentiable emulators on full or partial states of the simple but chaotic L96 system and applied them for data assimilation and parametrization learning. Emulation extends gradient-based reasoning and analysis to important simulators lacking derivatives. By training on simulator outputs alone, we avoid painstaking analysis of formulas or simulation code, which can be complex and idiosyncratic for simulators of weather, climate, and other important Earth system phenomena.

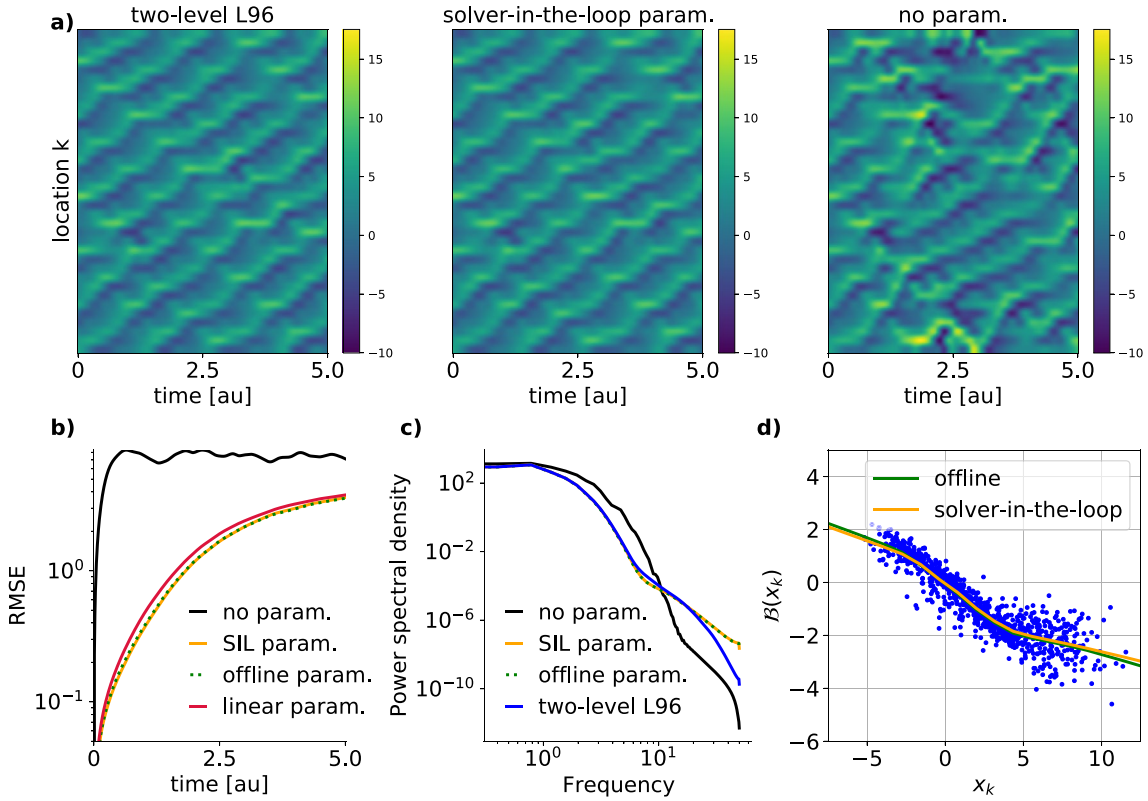

Figure 4. (a) Trajectory of coarse L96 variables from fine-scale simulations and from coarse-scale simulations with and without a parametrization aiming to capture the behavior of unresolved fine variables. We optimize a NN parametrization through our learned emulator ("solver-in-the-loop") and couple this parametrization to the true model to simulate. (b) RMSE over time between for coarse variables of a fine-scale simulation and (parametrized) coarse models, averaged over 1,000 shared initializations. (c) Power spectral density over 5 time units at $\Delta = 0.01$, averaged over all locations and 1,000 initializations. (d) Visualization of learned parametrization $\mathcal{B}(x_k)$ in comparison to a reference trained "offline" (see text). Blue dots show a subset of training data for offline parametrization training, which requires access to the unresolved fine-scale variables. NN, neural network; RMSE, root-mean-square error.

4.1. Related Work

Learning and correcting dynamics. Our results build on a growing literature describing networks that learn system dynamics (Grzeszczuk et al., 1998; Sanchez-Gonzalez et al., 2020), arising from ordinary differential equations (Chen et al., 2019; Fablet et al., 2018) and PDEs (Long et al., 2018; Rudy et al., 2019), some of which have applied numerical integration schemes during training (Wang & Lin, 1998). Several studies have used ML to learn parametrizations for L96 (Dueben & Bauer, 2018; Gagne et al., 2020; Watson, 2019) and other Earth science models but have either used offline training or employed Ensemble Kalman filtering and related approaches that do not require derivatives (Brajard et al., 2021; Pawar & San, 2020; Rasp, 2020).

A related line of research expands the task of parametrization learning to include the full dynamical model and learns updates directly from noisy and incomplete observations without a simulator, effectively combining emulator learning and data assimilation into a single task (Bocquet, 2012; Bocquet et al., 2020; Brajard et al., 2020). Long et al. (2018) approach this task by learning a PDE and discretized differential operators are learned from data. Fablet et al. (2020) learn dynamics while solving a *weak-constraint* 4D-Var problem, where minor violations of the learned dynamics are allowed, and train second network to solve the resulting optimization problem. Farchi et al. (2020) learn an ML-based correction to an existing simulator based on noisy observations but rely on available simulator derivatives. Like these studies and most Earth science simulators, we used time discretization, but network outputs can also be integrated in continuous time, resulting in a neural ordinary differential equation (Chen et al., 2019).

While building on these previous studies, our work is to our knowledge the first to design and evaluate emulators as *derivative estimators*, train them on partial system states, or systematically measure the accuracy

and performance of derivatives in downstream tasks. However, many of these approaches can be combined with ours.

Unsupervised methods. At the opposite end of the data versus physics spectrum, a new class of *unsupervised* methods train networks to solve PDEs by constructing an objective function directly from symbolic equations, without requiring any simulations for training (Raissi et al., 2019; Sirignano & Spiliopoulos, 2018). These approaches use autodiff to calculate spatial and temporal derivatives and avoid discretization in space or time but cannot change initial conditions after training. Wandel et al. (2020) instead constructs an objective function from discretized PDEs but can generalize to new initial conditions. These methods address a fundamentally different task, require careful attention to model equations, and must be validated by numerical integration.

4.2. Future Outlook

While this and other studies successfully trained emulators on simple models such as L96, it is less clear whether they can be scaled up to more complex models, larger grids, additional spatial dimensions, and more interacting variables. Key challenges include interactions across multiple space and time scales, strongly nonlinear effects, and the coupling of fluid dynamics to in-place physics. While emulation learning remains an open challenge for more complex models, a number of strategies may prove useful in these scaling up efforts.

We found that domain-specific architectural features such as spatial structure and locality help reduce the size of the function space in which we are searching for an optimal emulator. Learning tendencies within a higher-order integration scheme (Bocquet et al., 2020; Fablet et al., 2018; Wang & Lin, 1998) provides stronger locality constraints than learning state updates and reduces numerical instability (Scher & Messori, 2019). Location-dependent effects (e.g., Coriolis force) could also be captured by providing each grid coordinates as an input channel.

A concern when applying trained emulators with unfamiliar initial conditions or optimizing $\mathcal{L}_{\mathrm{DA}}$ in Equation 6 is that the input to the network may not resemble the training data, possibly degrading performance. Proposed solutions include penalizing deviations from a desired region of the space of system states (Ren et al., 2020) and adding noise to inputs during training (Sanchez-Gonzalez et al., 2020). For data assimilation, we found the regularization provided by the prior $p(\mathbf{x}_0|\mathbf{x}_f)$ to mitigate this problem.

Weather and climate models typically have modular structure with multiple interacting components, but optimizing a specific physical parametrization would also requires derivatives for the dynamical core and other parametrizations. One solution to this problem could be building modular emulators that mimic simulator subroutines for local dynamics and in-place physics. In these cases, it might be possible to train some parts of the emulator using simulator runs with certain physical parametrizations turned off.

Data Availability Statement

Code, simulated data, and results of our study can be found at Zenodo (https://doi.org/10.5281/zenodo.4638267). Simulation data for Lorenz-96 were generated with code from https://github.com/m-dml/L96sim. Code for emulator definition & training, as well as all numerical experiments, is also found at https://github.com/m-dml/emulator_L96.

Acknowledgments
M. Nonnenmacher and D. Greenberg were supported by the Helmholtz AI initiative. We thank Johanna Baehr, Bedartha Goswami, and Vadim Zinchenko for comments on the manuscript. Open access funding enabled and organized by Projekt DEAL.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., et al. (2016). *Tensorflow: A system for large-scale machine learning*. In 12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16 (pp. 265–283).

Apte, A., Jones, C. K., Stuart, A., & Voss, J. (2008). Data assimilation: Mathematical and statistical perspectives. *International Journal for Numerical Methods in Fluids*, *56*(8), 1033–1046.

Bauer, P., Thorpe, A., & Brunet, G. (2015). The quiet revolution of numerical weather prediction. *Nature*, *525*(7567), 47–55. https://doi.org/10.1038/nature14956

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98.

Bischof, C., Carle, A., Corliss, G., Griewank, A., & Hovland, P. (1992). Adifor—Generating derivative codes from Fortran programs. *Scientific Programming*, *1*(1), 11–29.

Bocquet, M. (2012). Parameter-field estimation for atmospheric dispersion: Application to the Chernobyl accident using 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, *138*(664), 664–681.

Bocquet, M., Brajard, J., Carrassi, A., & Bertino, L. (2019). Data assimilation as a learning tool to infer ordinary differential equation representations of dynamical models. *Nonlinear Processes in Geophysics*, *26*(3), 143–162.

Bocquet, M., Brajard, J., Carrassi, A., & Bertino, L. (2020). Bayesian inference of chaotic dynamics by merging data assimilation, machine learning and expectation-maximization. *Foundations of Data Science*, *2*(1), 55–80.

Brajard, J., Carassi, A., Bocquet, M., & Bertino, L. (2020). *Combining data assimilation and machine learning to emulate a dynamical model from sparse and noisy observations: A case study with the Lorenz 96 model*. arXiv preprint arXiv:2001.01520.

Brajard, J., Carrassi, A., Bocquet, M., & Bertino, L. (2021). Combining data assimilation and machine learning to infer unresolved scale parametrization. *Philosophical Transactions of the Royal Society A*, *379*(2194), 20200086.

Brenowitz, N. D., Beucler, T., Pritchard, M., & Bretherton, C. S. (2020). *Interpreting and stabilizing machine-learning parametrizations of convection*. arXiv preprint arXiv:2003.06549 [physics].

Carrassi, A., Bocquet, M., Bertino, L., & Evensen, G. (2018). Data assimilation in the geosciences: An overview of methods, issues, and perspectives. *Wiley Interdisciplinary Reviews: Climate Change*, *9*(5), e535.

Chattopadhyay, A., Hassanzadeh, P., Palem, K., & Subramanian, D. (2019). *Data-driven prediction of a multi-scale Lorenz 96 chaotic system using a hierarchy of deep learning methods: Reservoir computing, ANN, and RNN-LSTM*. arXiv preprint arXiv:1906.08829.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. (2019). *Neural ordinary differential equations*. arXiv preprint arXiv:1806.07366 [cs, stat].

Courtier, P., & Rabier, F. (1997). The use of adjoint equations in numerical weather prediction. *Atmosphere-Ocean*, *35*(suppl. 1), 303–322. https://doi.org/10.1080/07055900.1997.9687354

Crommelin, D., & Vanden-Eijnden, E. (2008). Subgrid-scale parameterization with conditional Markov chains. *Journal of the Atmospheric Sciences*, *65*(8), 2661–2675.

Desroziers, G., Camino, J.-T., & Berre, L. (2014). 4DEnVar: Link with 4D state formulation of variational assimilation and different possible implementations. *Quarterly Journal of the Royal Meteorological Society*, *140*(684), 2097–2110.

Dueben, P. D., & Bauer, P. (2018). Challenges and design choices for global weather and climate models based on machine learning. *Geoscientific Model Development*, *11*(10), 3999–4009.

Errico, R. M. (1997). What is an adjoint model? *Bulletin of the American Meteorological Society*, *78*(11), 2577–2591.

Eyring, V., Bony, S., Meehl, G. A., Senior, C. A., Stevens, B., Stouffer, R. J., & Taylor, K. E. (2016). Overview of the coupled model intercomparison project phase 6 (CMIP6) experimental design and organization. *Geoscientific Model Development*, *9*(5), 1937–1958.

Eyring, V., Gleckler, P. J., Heinze, C., Stouffer, R. J., Taylor, K. E., Balaji, V., et al. (2016). Toward improved and more routine Earth system model evaluation in CMIP. *Earth System Dynamics*, *7*(4), 813–830. https://doi.org/10.5194/esd-7-813-2016

Fablet, R., Chapron, B., Drumetz, L., Mémin, E., Pannekoucke, O., & Rousseau, F. (2020). *Learning variational data assimilation models and solvers*. arXiv preprint arXiv:2007.12941.

Fablet, R., Ouala, S., & Herzet, C. (2018). Bilinear residual neural network for the identification and forecasting of geophysical dynamics. In *2018 26th European signal processing conference (EUSIPCO)* (pp. 1477–1481).

Farchi, A., Laloyaux, P., Bonavita, M., & Bocquet, M. (2020). *Using machine learning to correct model error in data assimilation and forecast applications*. arXiv preprint arXiv:2010.12605 [physics, stat].

Fertig, E. J., Harlim, J., & Hunt, B. R. (2007). A comparative study of 4D-Var and a 4D ensemble Kalman filter: Perfect model simulations with Lorenz-96. *Tellus A: Dynamic Meteorology and Oceanography*, *59*(1), 96–100.

Gagne, D. J., Christensen, H. M., Subramanian, A. C., & Monahan, A. H. (2020). Machine learning for stochastic parameterization: Generative adversarial networks in the Lorenz'96 model. *Journal of Advances in Modeling Earth Systems*, *12*, e2019MS001896. https://doi.org/10.1029/2019MS001896

Gross, M., Wan, H., Rasch, P. J., Caldwell, P. M., Williamson, D. L., Klocke, D., et al. (2018). Physics–dynamics coupling in weather, climate, and earth system models: Challenges and recent progress. *Monthly Weather Review*, *146*(11), 3505–3544. https://doi.org/10.1175/MWR-D-17-0345.1

Grzeszczuk, R., Terzopoulos, D., & Hinton, G. (1998). NeuroAnimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on computer graphics and interactive techniques* (pp. 9–20). New York, NY: Association for Computing Machinery. https://doi.org/10.1145/280814.280816

Holl, P., Koltun, V., & Thuerey, N. (2020). *Learning to control PDEs with differentiable physics*. arXiv preprint arXiv:2001.07457.

Hourdin, F., Mauritsen, T., Gettelman, A., Golaz, J.-C., Balaji, V., Duan, Q., et al. (2016). The art and science of climate model tuning. *Bulletin of the American Meteorological Society*, *98*(3), 589–602. https://doi.org/10.1175/BAMS-D-15-00135.1

Hu, Y., Anderson, L., Li, T.-M., Sun, Q., Carr, N., Ragan-Kelley, J., & Durand, F. (2019). *DiffTaichi: Differentiable programming for physical simulation*. arXiv preprint arXiv:1910.00935.

Ide, K., Courtier, P., Ghil, M., & Lorenc, A. C. (1997). Unified notation for data assimilation: Operational, sequential and variational. *Journal of the Meteorological Society of Japan. Series II*, *75*(1B), 181–189.

Kain, J. S., & Fritsch, J. M. (1993). Convective parameterization for mesoscale models: The Kain–Fritsch scheme. In *The representation of cumulus convection in numerical models* (pp. 165–170). New York: Springer.

Karpatne, A., Ebert-Uphoff, I., Ravela, S., Babaie, H. A., & Kumar, V. (2017). *Machine learning for the geosciences: Challenges and opportunities*. arXiv preprint arXiv:1711.04708 [physics].

Kurth, T., Treichler, S., Romero, J., Mudigonda, M., Luehr, N., Phillips, E., & Houston, M. (2018). *Exascale deep learning for climate analytics*. arXiv preprint arXiv:1810.01993 [cs].

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, *16*(2), 146–160.

Long, Z., Lu, Y., Ma, X., & Dong, B. (2018). PDE-Net: Learning PDEs from data. In *International conference on machine learning, PMLR* (pp. 3208–3216).

Lorenc, A. C. (1986). Analysis methods for numerical weather prediction. *Quarterly Journal of the Royal Meteorological Society*, *112*(474), 1177–1194.

Lorenz, E. N. (1996). Predictability: A problem partly solved. In *Proc. seminar on predictability* (Vol. 1).

Maher, N., Milinski, S., Suarez-Gutierrez, L., Botzet, M., Dobrynin, M., Kornblueh, L., et al. (2019). The Max Planck Institute grand ensemble: Enabling the exploration of climate system variability. *Journal of Advances in Modeling Earth Systems*, *11*, 2050–2069. https://doi.org/10.1029/2019MS001639

Obiols-Sales, O., Vishnu, A., Malaya, N., & Chandramowlishwaran, A. (2020). CFDNet: A deep learning-based accelerator for fluid simulations. In *Proceedings of the 34th ACM international conference on supercomputing* (pp. 1–12). https://doi.org/10.1145/3392717.3392772

Orrell, D. (2003). Model error and predictability over different timescales in the Lorenz'96 systems. *Journal of the Atmospheric Sciences*, *60*(17), 2219–2228.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., & Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), Advances in neural information processing systems 32 (pp. 8026–8037). Red Hook; NY: Curran Associates, Inc.

Pawar, S., & San, O. (2020). *Data assimilation empowered neural network parameterizations for subgrid processes in geophysical flows*. arXiv preprint arXiv:2006.08901.

Rackauckas, C., Ma, Y., Dixit, V., Guo, X., Innes, M., Revels, J., & Ivaturi, V. (2018). *A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions*. arXiv preprint arXiv:1812.01892.

Rackauckas, C., & Nie, Q. (2017). Differentialequations jl—A performant and feature-rich ecosystem for solving differential equations in Julia. *Journal of Open Research Software*, *5*(1).

Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, *378*, 686–707.

Ramadhan, A., Marshall, J., Souza, A., Wagner, G. L., Ponnapati, M., & Rackauckas, C. (2020). *Capturing missing physics in climate model parameterizations using neural differential equations*. arXiv preprint arXiv:2010.12559.

Rasp, S. (2020). Coupled online learning as a way to tackle instabilities and biases in neural network parameterizations: General algorithms and Lorenz 96 case study (v1.0). *Geoscientific Model Development*, *13*(5), 2185–2196.

Reichstein, M., Camps-Valls, G., Stevens, B., Jung, M., Denzler, J., Carvalhais, N., & Prabhat (2019). Deep learning and process understanding for data-driven Earth system science. *Nature*, *566*(7743), 195–204. https://doi.org/10.1038/s41586-019-0912-1

Ren, S., Padilla, W., & Malof, J. (2020). *Benchmarking deep inverse models over time, and the neural-adjoint method*. arXiv preprint arXiv:2009.12919.

Rudy, S., Alla, A., Brunton, S. L., & Kutz, J. N. (2019). Data-driven identification of parametric partial differential equations. *SIAM Journal on Applied Dynamical Systems*, *18*(2), 643–660.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., & Battaglia, P. W. (2020). *Learning to simulate complex physics with graph networks*. arXiv preprint arXiv:2002.09405 [physics, stat].

Scher, S., & Messori, G. (2019). Generalization properties of feed-forward neural networks trained on Lorenz systems. *Nonlinear Processes in Geophysics*, *26*(4), 381–399.

Schneider, T., Lan, S., Stuart, A., & Teixeira, J. (2017). Earth system modeling 2.0: A blueprint for models that learn from observations and targeted high-resolution simulations. *Geophysical Research Letters*, *44*, 12396–12417. https://doi.org/10.1002/2017GL076101

Seifert, A., & Rasp, S. (2020). Potential and limitations of machine learning for modeling warm-rain cloud microphysical processes. *Journal of Advances in Modeling Earth Systems*, *12*, e2020MS002301. https://doi.org/10.1029/2020MS002301

Sirignano, J., & Spiliopoulos, K. (2018). DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, *375*, 1339–1364. https://doi.org/10.1016/j.jcp.2018.08.029

Smith, G. D., Smith, G. D., & Smith, G. D. S. (1985). *Numerical solution of partial differential equations: Finite difference methods*. Oxford, UK: Oxford University Press.

Stensrud, D. J. (2009). *Parameterization schemes: Keys to understanding numerical weather prediction models*. Cambridge, UK: Cambridge University Press.

Trémolet, Y. (2004). Diagnostics of linear and incremental approximations in 4D-Var. *Quarterly Journal of the Royal Meteorological Society*, *130*(601), 2233–2251.

Um, K., Holl, P., Brand, R., & Thuerey, N. (2020). *Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers*. arXiv preprint arXiv:2007.00016.

Wandel, N., Weinmann, M., & Klein, R. (2020). *Unsupervised deep learning of incompressible fluid dynamics*. arXiv preprint arXiv:2006.08762.

Wang, Y.-J., & Lin, C.-T. (1998). Runge-Kutta neural network for identification of dynamical systems in high accuracy. *IEEE Transactions on Neural Networks*, *9*(2), 294–307.

Watson, P. A. (2019). Applying machine learning to improve simulations of a chaotic dynamical system using empirical error correction. *Journal of Advances in Modeling Earth Systems*, *11*, 1402–1417. https://doi.org/10.1029/2018MS001597

Wedi, N., Bauer, P., Denoninck, W., Diamantakis, M., Hamrud, M., Kuhnlein, C., et al. (2015). *The modeling infrastructure of the integrated forecasting system: Recent advances and future challenges*. Reading, UK: European Centre for Medium-Range Weather Forecasts.

Weng, F., & Liu, Q. (2003). Satellite data assimilation in numerical weather prediction models. Part I: Forward radiative transfer and Jacobian modeling in cloudy atmospheres. *Journal of the Atmospheric Sciences*, *60*(21), 2633–2646.

Yuval, J., & O'Gorman, P. A. (2020). Stable machine-learning parameterization of subgrid processes for climate modeling at a range of resolutions. *Nature Communications*, *11*(1), 1–10.

Zängl, G., Reinert, D., Rípodas, P., & Baldauf, M. (2015). The ICON (ICOsahedral Non-hydrostatic) modeling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core. *Quarterly Journal of the Royal Meteorological Society*, *141*(687), 563–579.