DARPA ACTM Milestone 2 Report

Quantification of extreme weather events and their future changes using Physics-Informed DeepONet modeling and functional priors

Agreement No. HR00112290029

PI: T. Sapsis (MIT), G. Karniadakis (Brown U), R. Leung (PNNL)

April 13, 2022

1 Hybrid Methodology

1.1 Why a hybrid method?

The aim of this project is to develop coarse-scale and data-informed, hybrid methods that produce accurate statistical predictions of climate models. Specifically, contemporary climate models are characterized by vast computational costs. To this end, they are not practical for informing decisions for policy making or stakeholders, e.g. answering questions such as what is the probability of a catastrophic event over a major city for the next two decades? To reduce this computational cost one can employ coarser-scale climate models. However, this will introduce important model errors which are not acceptable.

The goal of the presented hybrid methodology is to bridge this gap. Specifically, we aim to use short time climate simulations from highly accurate models, in order to machine-learn a correction that should be applied to a coarse-scale model. After we learn this correction, we can then run the coarse-scale model for an extensive period of time and subsequently correct its output using the trained scheme.

What do we gain from the presented approach? We are able for the first time to reliably and quantitatively study climate dynamics over large time scales, a previously intractable problem. In addition, we could also quantify the effect of parameters variations (e.g. CO2 concentration) and detect tipping points related to properties of extreme events. This has only been possible, so far, through the use of 'toy' problems that can only lead to qualitative conclusions which are often questionable.

1.2 Data-informed uncertainty quantification in climate models

To present the method, a simple surrogate climate model is considered. To this end, a two-layer quasi-geostrophic (QG)[1] model is employed to mimic the velocity field $\boldsymbol{v}(\boldsymbol{x},t)$ of atmospheric flows. The turbulent flow is described by the dimensionless evolution equation

$$\frac{\partial q_j}{\partial t}+\boldsymbol{v}_j\cdot\boldsymbol{\nabla}q_j+\left(\beta+k_d^2U_j\right)\frac{\partial\psi_j}{\partial x}=-\delta_{2,j}r\Delta\psi_j-\nu\Delta^s q_j, \tag{1}$$

where $j=1,2$ corresponds to the upper and lower layer respectively and the flow is defined in the horizontal domain $(x,y)\in[0,2\pi]^2$. Doubly periodic boundary conditions are assumed. A mean zonal flow of intensity $U_1=U$ and $U_2=-U$ is imposed on each layer respectively. The two layers are assumed to have the same width, k_d denotes the deformation radius, r the bottom-drag coefficient and β is the beta-plane approximation parameter. When running fine-scale simulations this term is equal to zero. The potential vorticity (PV) q_j and corresponding streamfunction ψ_j are related via the inversion formulae

$$q_j=\Delta\psi_j+\frac{k_d^2}{2}\left(\psi_{3-j}-\psi_j\right),\quad j=1,2. \tag{2}$$

The velocity field can be written as $\boldsymbol{v}_j=(U_j,0)+\boldsymbol{v}'_j$, where it is decomposed into a zonal mean and a fluctuating shear flow, with $\boldsymbol{v}'_j=(-\partial\psi_j/\partial x,\partial\psi_j/\partial y)$. For a more physical interpretation of the flow, the barotropic PV $q_t=(q_1+q_2)/2$ and the the baroclinic PV $q_c=(q_1-q_2)$ are defined respectively.

The quasigeostrophic model described by eq. (1) is used here to simulate mid latitude and high latitude atmospheric flows that are forced by an imposed shear current.[2] Example of parameter values for the two regimes are shown in table 1.

regime	β	k_d	U	r	ν	s
atmosphere, high lat.	1	4	0.2	0.2	1×10^{-13}	4
atmosphere, mid lat.	2	4	0.2	0.1	1×10^{-13}	4

Table 1: Parameter values for different atmosphere regimes.

Despite the simple structure of the model, quasi-geostrophic flows experience intermittent energy cascades. This phenomenon renders the effects of small scales to large-scale flow features crucial. As a result, a coarse-scale resolution of eq. (1) yields poor statistical predictions. For the current numerical investigation, the reference horizontal resolution is set to 128×128. The coarse-scale simulations are set to 24×24, a resolution that is inadequate to correctly capture the statistics of such flows. Both sets of simulations have the same temporal resolution. The flow is solved using spectral method and the time evolution scheme is the explicit 4th-order Runge-Kutta. Furthermore, energy interactions between small and large scales are strongly non-Gaussian in this framework. Hence, Gaussian-closure schemes lack the ability to correctly capture these contributions. Therefore, more involved reduced-order models, that appropriately model the energy transport mechanism between modes of the system, are essential.

This work aims to train a neural network that, given as input the predictions of a free running coarse-scale simulation $(\psi_1^{\text{coarse}}, \psi_2^{\text{coarse}})$ of eq. (1), it will produce a corrected time-series $\left(\text{e.g. } \psi_1^{\text{ML}}, \psi_2^{\text{ML}}\right)$ that will have the same statistics as a fine-scale reference simulation $\left(\psi_1^{\text{ref}}, \psi_2^{\text{ref}}\right)$. A diagram of this process is described in fig. 1. The corrections are carried out in a post-processing manner without the need to access the numerical solver of the system. The de-coupling of the data-informed correction process and the initial simulation phase is justified by the fact that the goal is not to make phase corrections at each time-step but retrieve the correct statistics for the current flow parameters. We use two approaches. First the streamfunction of each layer is used as input and second uses PV as the input. Although the two are related and can be derived from each other, streamfunction dynamics capture large-scale flow features while as PV captures small-scale features. Comparisons between the two approaches allow us to test the robustness of ML corrections and ascertain the accuracy of derived quantities – representing either large and (or) small scale features.

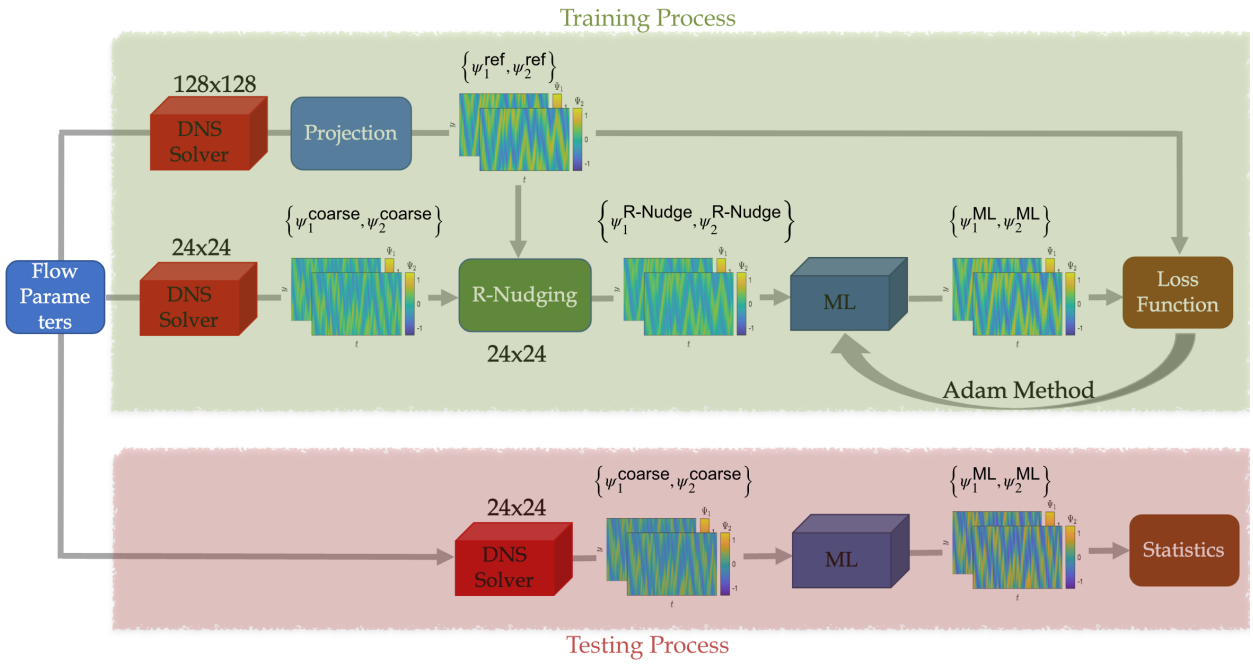

Figure 1: Schematic diagram of the proposed training and testing process.

While testing will be carried out using free running coarse-scale data, appropriate training data need to be determined first. Due to chaotic divergence, free running coarse-scale data will very quickly diverge from their fine-scale conuterpart despite having the same flow parameters and initial conditions. As a result, it is not feasible for a neural network to learn a mapping directly between $(\psi_1^{\text{coarse}}, \psi_2^{\text{coarse}})$ and $\left(\psi_1^{\text{ref}}, \psi_2^{\text{ref}}\right)$. To that end, to produce coarse-scale simulations for training, a relaxation term Q is added to the evolution eq. (1) to become

$$\frac{\partial q_j}{\partial t} + \boldsymbol{v}_j \cdot \boldsymbol{\nabla} q_j + \left(\beta + k_d^2 U_j\right) \frac{\partial \psi_j}{\partial x} = -\delta_{2,j} r \Delta \psi_j - \nu \Delta^s q_j + Q\left(q_j - q_j^{ref}\right), \tag{3}$$

where $j = 1, 2$. The term Q is called nudging tendency and it corrects the coarse-scale solution based on the fine-scale reference solution. In this study, the nudging tendency Q is given by the algebraic term

$$Q\left(q_j - q_j^{\text{ref}}\right) = -\frac{1}{\tau}\left(q_j - \mathcal{H}\left[q_j^{\text{ref}}\right]\right). \tag{4}$$

Parameter τ is a relaxation timescale to be determined, and $\mathcal{H}$ is an operator that maps q_j^{ref} to the coarse resolution. Parameter τ is chosen so as the nudged solution q_j^{nudged} satisfies two properties: (i) it stops the nudged simulation from diverging from the reference solution q_j^{ref}, i.e. allowing for the mapping between the two time histories (ii) it resembles the statistical properties of the coarse-scale free running simulation. The second property is important to ensure that the mapping learned during training, will be applicable during testing with un-nudged data $(\psi_1^{\text{coarse}}, \psi_2^{\text{coarse}})$. As can be seen in fig. 2, a value that approximately satisfies both properties is $\tau = 16$.

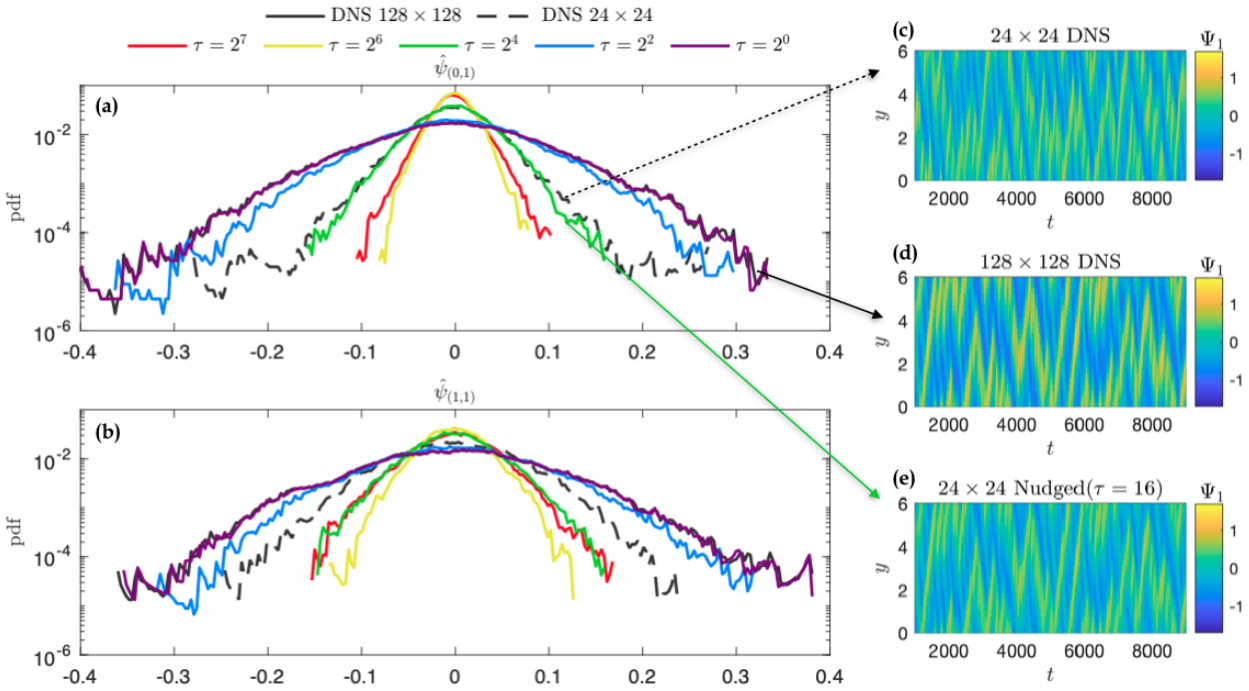

Figure 2: Pdfs of wavenumber amplitudes (a) $\hat{\psi}_{(0,1)}$ and (b) $\hat{\psi}_{(1,1)}$ of the barotropic mode for various simulations. Time evolution of the zonally-averaged profile of the top layer streamfunction for (c) coarse-scale free running simulation (d) reference fine-scale simulation (e) nudged coarse-scale simulation for $\tau = 16$. Flow parameters correspond to the mid latitude case of table 1

However, from the same figure, one can observe that the statistics of the nudged solution differ when computing the pdf of $\hat{\psi}_{(1,1)}$. This is a result of discrepancies in the energy spectrum of the nudged solution with respect to the coarse-scale solution, which can be seen in fig. 3 for the same flow parameters. These energy spectra differences lead to different statistical behaviours of testing data $(\psi_1^{\text{coarse}}, \psi_2^{\text{coarse}})$ and training data $\left(\psi_1^{\text{nudge}}, \psi_2^{\text{nudge}}\right)$. Discrepancies in the training and testing input distributions will lead to the neural network behaving differently in the two schemes.[3] These discrepancies cannot be reconciled by simply choosing an appropriate τ as algebraic nudging adds linear dissipation to the system, thus always changing the energy spectrum of the resulting flow.

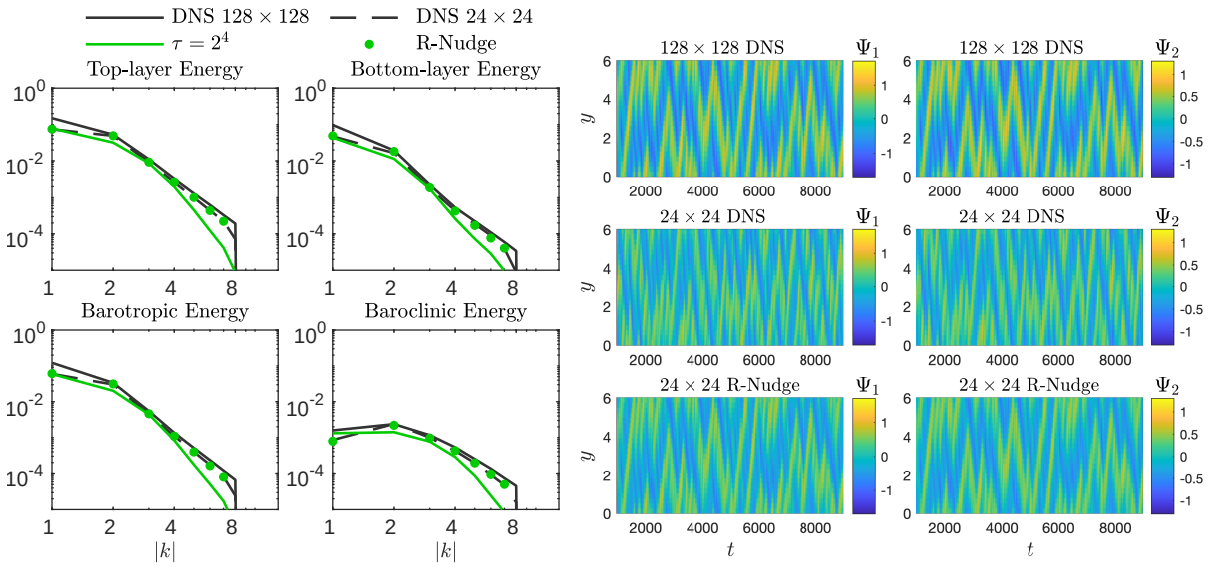

Figure 3: (a) Energy spectra of the Top layer, bottom layer, barotropic and baroclinic mode. (b) Predictions of zonally-averaged streamfunctions for reference simulation, coarse-scale free runing simulation and coarse-scale R-nudged simulation.

To mitigate this issue, a novel "spectral nudging" is developed and employed. The process is called "Reverse Spectral Nudging" with its purpose being to match the energy spectrum of the nudged solution to that of the coarse-scale solution. Hence, while traditional nudging schemes correct the coarse-scale solution with data from the reference solution, the proposed scheme further processes the nudged data by matching its energy spectrum to that of the corresponding free running coarse-scale flow. The corrected nudged data are, are termed $\left(\psi_1^{\text{R-Nudge}}, \psi_2^{\text{R-Nudge}}\right)$ and defined as

$$\psi_i^{\text{R-Nudge}}(x, y, t) = \sum_{k,l} R_{k,l} \hat{\psi}_{k,l}^{\text{nudge}}(t) e^{i(kx+ly)}, \tag{5}$$

where

$$R_{k,l} = \sqrt{\frac{\mathcal{E}_{k,l}^{\text{coarse}}}{\mathcal{E}_{k,l}^{\text{nudge}}}}, \tag{6}$$

and

$$\mathcal{E}_{k,l}^{*} = \frac{1}{T} \int_0^T \hat{E}_{k,l}^{*}(p) \mathrm{d}p, \quad * = \text{coarse, nudge, ref.} \tag{7}$$

An important property of this scheme is that the new data have exactly the energy spectrum of the free running coarse simulation, as shown in fig. 3, meaning that the training and testing data come from the same distributions. The training process is shown in fig. 1. Subfigure (b) shows that the R-nudged data still follow the reference data, allowing for a mapping

between $\left(\psi_1^{\text{R-Nudge}}, \psi_2^{\text{R-Nudge}}\right)$ and $\left(\psi_1^{\text{ref}}, \psi_2^{\text{ref}}\right)$. Furthermore, the same process can be applied directly to the reference data, creating the R-nudged data set

$$\psi_i^{\text{R-Nudge'}}(x, y, t) = \sum_{k,l} R'_{k,l} \hat{\psi}_{k,l}^{\text{ref}}(t) e^{i(kx+ly)}, \tag{8}$$

where now

$$R'_{k,l} = \sqrt{\frac{\mathcal{E}_{k,l}^{\text{coarse}}}{\mathcal{E}_{k,l}^{\text{ref}}}}, \tag{9}$$

This process does not require running additional nudged simulations, thus lowering the total cost of the training scheme. Both using as training data eq. (5) and eq. (8) will be utilized and the resulting neural networks will be compared for generalizability.

1.3 Neural Network Architecture

We use two neural network architectures to correct the coarse scale data. First, a typical recurrent neural network (RNN) is used. In particular, long short-term memory (LSTM)[4] neural networks are used. This configuration will act as a baseline for comparison with a recently proposed neural network based on operator learning called DeepONet. A particular focus of the comparison between the two architectures will be efficiency of data use, computational cost, and accuracy of generalizations. The architecture of the neural-networks is shown in fig. 4. The first architecture consists of an input fully connected layer that compresses the streamfunctions of the two layers from a $24 \times 24 \times 2$ dimension to a 60-valued vector. This layer has a tanh activation function. This vector is then passed as input to a long short-term memory (LSTM) neural network. The output of the neural network is then passed through an output fully connected neural network to produce the final data-informed corrected predictions. The output layer has a linear activation function.

LSTM neural networks incorporate non-Markovian memory effects into the reduced-order model. This ability stems from Takens embedding theorem.[5] This theorem states that given delayed embeddings of a limited number of state variables, one can still obtain the attractor of the full system for the observed variables. This approach is known to be capable of improving predictions of reduced-order models.[6–8] Hence, it is expected that RNNs can help predict the contribution of unresolved scales. Furthermore, given that the flows discussed here reach a statistical equilibrium, it is strongly motivated that Takens theorem will apply and memory terms will significantly improve the predictions.

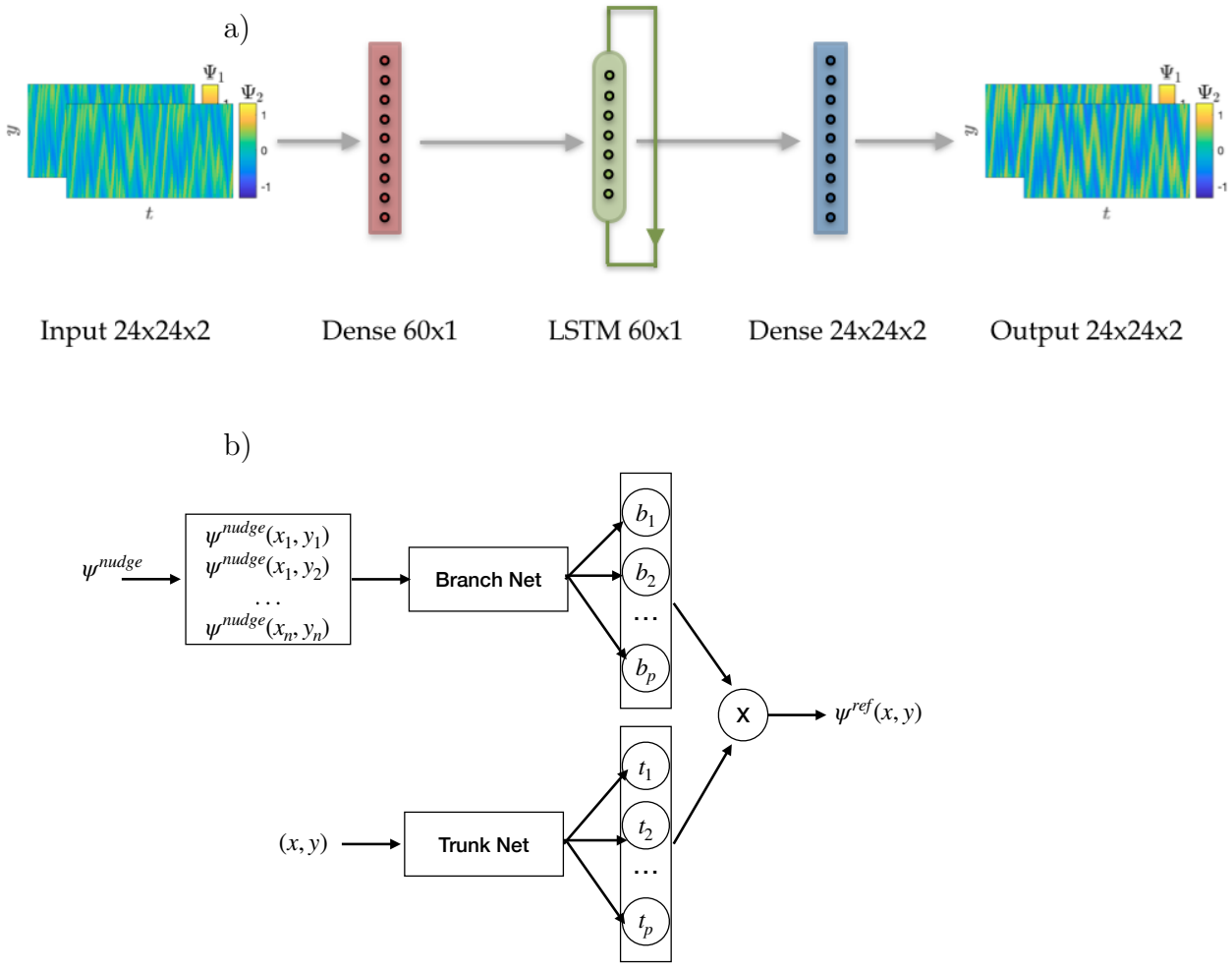

Figure 4: a) Neural network architecture using LSTM. b) DeepONet architecture

In addition to temporal nonlocality, the model is nonlocal in space. Due to the small size of the system, for a prediction at point $\boldsymbol{x}$ in space, information from the entire domain is used. In the case of E3SM dataset fully-connected layers will be replaced by convolutional layers.

The second architecture, DeepONet, is based on the generalized universal approximation theorem for operators.[9,10] The theorem states that neural networks with a single hidden layer can accurately approximate any nonlinear continuous operator. In particular, we use an unstacked DeepONet which has one trunk network and one branch network. The goal is to find the operator that maps coarse scale resolution measurements to fine scale resolution. The trunk network takes the spatial location of measurement as input and the branch network takes as input the entire flow field from coarse model at a given time. The prediction is the measurement at the given spatial location. Entire flow-field is provided as input to encode non-locality in space.

1.4 Loss Function

The first term to be used in the loss function is the L^2-error of the streamfunction predictions:

$$L = \sum_{j=1}^{2} \int_0^{L_x} \int_0^{L_y} \left(\psi_j^{\mathrm{ML}} - \psi_j^{\mathrm{ref}}\right)^2 \mathrm{d}x\mathrm{d}y. \tag{10}$$

To introduce some physical consistency in the predictions of the neural network, physical constraints are also incorporated. Let h_j^{eq} be the equilibrium depth of layer j and $h_j^{'}$ be a local variation from this depth. It then naturally follows from mass conservation of an incompressible flow that

$$\int_0^{L_x} \int_0^{L_y} h_j^{'} \mathrm{d}x\mathrm{d}y = 0, \quad j = 1, 2. \tag{11}$$

Using geostrophic balance, depth variations can be related to the streamfunction via the equations

$$\psi_1 = \frac{g}{h_0}\left(h_1^{'} + h_2^{'}\right), \quad \psi_2 = \frac{g}{h_0}\left(h_1^{'} + h_2^{'}\right) + \frac{g^{'}}{f_0} h_2^{'}. \tag{12}$$

Combining eq. (12) with the mass conservation principle eq. (11), one gets a physical constraint with respect to the streamfunctions:

$$\int_0^{L_x} \int_0^{L_y} \psi_j \mathrm{d}x\mathrm{d}y = 0, \quad j = 1, 2. \tag{13}$$

Incorporating only eq. (10) and eq. (13) in a loss function however, implies that the predictions for the two layers can be trained independently, a property that is non-physical. The main physical constraint across layers is that while the flow can exhibit velocity jumps across layers, no pressure jumps can be observed. To incorporate this cross-layer constraint, we introduce to addition terms, (i) the inversion relation eq. (2) and (ii) the system dynamics eq. (1). Incorporating all these terms, the final loss function becomes

$$\begin{aligned}
\mathcal{L} = &\sum_{i=1}^{2} \alpha_i \overbrace{\int_0^{L_x} \int_0^{L_y} \left(\psi_i^{\mathrm{ML}} - \psi_i^{\mathrm{ref}}\right)^2 \mathrm{d}x\mathrm{d}y}^{\text{Streamfunction } L^2\text{-error}} + \sum_{i=1}^{2} \beta_i \overbrace{\left| \int_0^{L_x} \int_0^{L_y} \psi_i^{\mathrm{ML}} \mathrm{d}x\mathrm{d}y \right|}^{\text{Continuity/ Mass Conservation}} \\
&+ \sum_{i=1}^{2} \gamma_i \overbrace{\int_0^{L_x} \int_0^{L_y} \left[\Delta\psi_i^{\mathrm{ML}} + \frac{k_d^2}{2}\left(\psi_{3-i}^{\mathrm{ML}} - \psi_i^{\mathrm{ML}}\right) - q_i^{\mathrm{ref}}\right]^2 \mathrm{d}x\mathrm{d}y}^{\text{Vorticity } L^2\text{-error}} \\
&+ \sum_{i=1}^{2} \delta_i \overbrace{\int_0^{L_x} \int_0^{L_y} \left(\frac{\partial q_i^{\mathrm{ML}}}{\partial t} - \frac{\partial q_i^{\mathrm{ref}}}{\partial t}\right)^2 \mathrm{d}x\mathrm{d}y}^{\text{System Dynamics}}
\end{aligned} \tag{14}$$

where the time-derivative of q_i^{ML} is determined by the equation

$$\frac{\partial q_j^{\mathrm{ML}}}{\partial t} = -\overbrace{\left(\boldsymbol{v}_j^{\mathrm{ML}} + \boldsymbol{V}_j\right)\cdot\boldsymbol{\nabla} q_j^{\mathrm{ML}}}^{\text{Advection}} + \overbrace{\left(\beta + k_d^2 U_j\right)\frac{\partial \psi_j^{\mathrm{ML}}}{\partial x}}^{\text{Coriolis}} - \overbrace{\delta_{2,j} r \Delta \psi_j^{\mathrm{ML}}}^{\text{Bottom Drag}}. \tag{15}$$

Parameters $\{\alpha_j, \beta_j, \gamma_j, \delta_j\}$ for $j = 1, 2$, are factors used to regularize the terms. A validation that the new loss function eq. (14) can be seen in fig. 5, where the predicted mass conservation error at the top layer is computed for a neural netwokr trained with eq. (10) and with eq. (14).

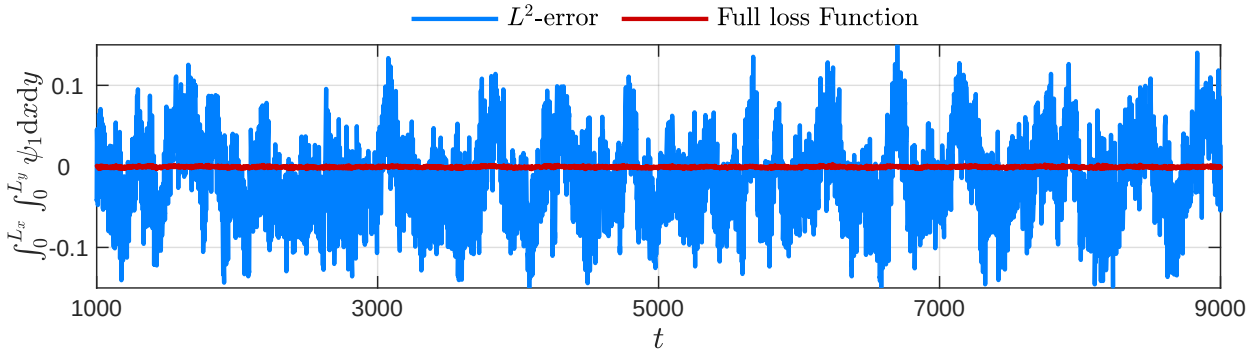

Figure 5: Mass conservation error as calculated for the layer for a neural-network trained with loss function eq. (10) (blue line) and loss function eq. (14).

1.5 Training Process

For initial results, the model is trained on the mid latitude flow parameters presented in table 1. The simulations are carried out until $T = 9000$. For training and testing the time-interval $[1000, 9000]$ is kept where the flow has achieved statistical equilibrium. The neural network is trained using the Adam method,[11] which is a first-order gradient descent method. Input and reference data are normalized so that $\psi_1, \psi_2 \in [-1, 1]$. The neural network is trained for 2000 epochs. For training, the time-interval $[1000, 4600]$ is used and the interval $[4600, 5000]$ is used for validation. R-nudged data are used as input data.

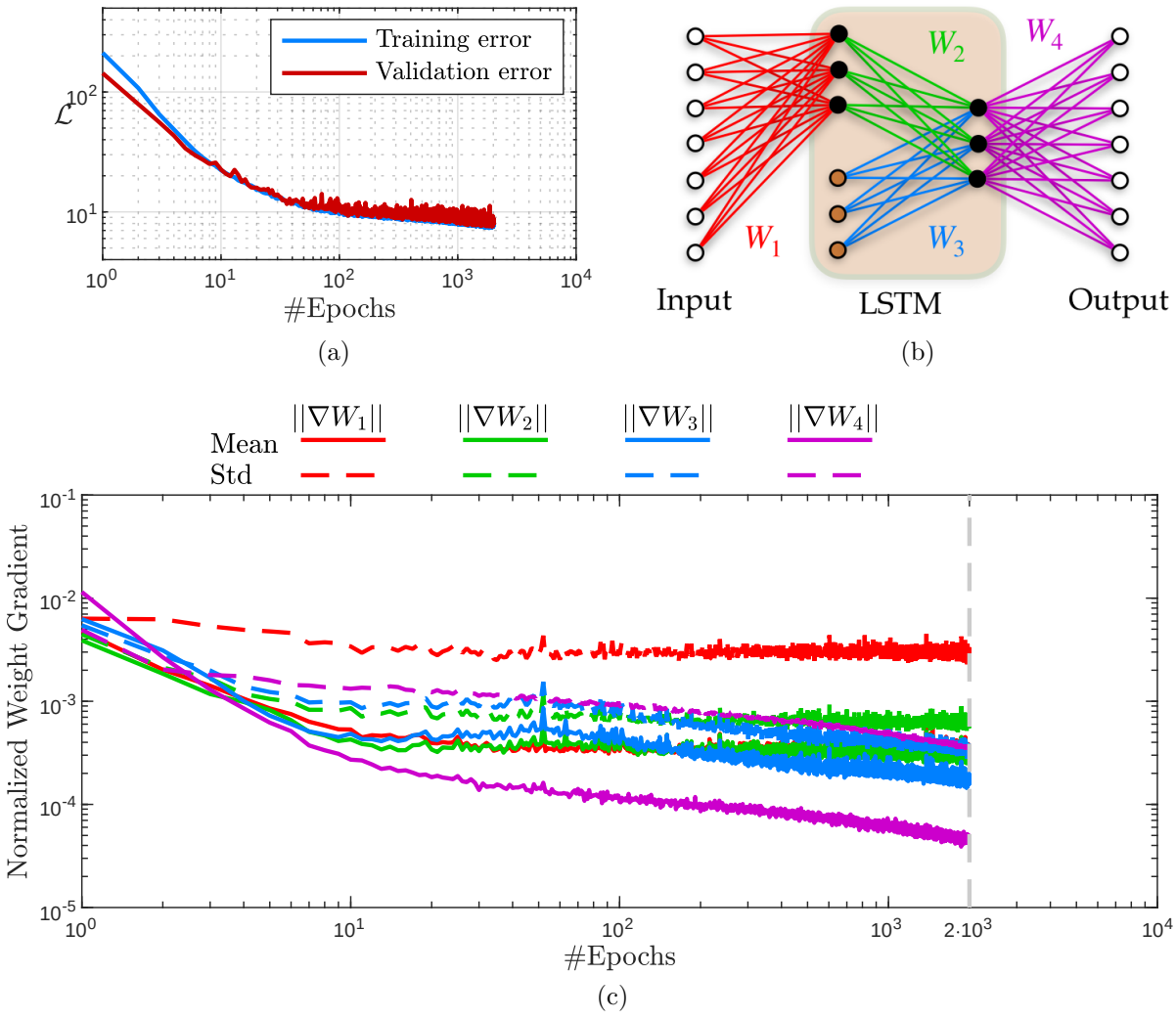

Figure 6: (a) Training and validation error for the proposed neural network; (b) Schematic diagram of the separation of weights along the architecture of the neural network; (c) Normalized norm of mean and standard deviation of weight gradients for the weights as defined in (b).

In fig. 6, subfigure (a) shows the training and validation error at the end of the training process. The close resemblance of validation and training error showcase the generalizability of the scheme. To further show that the neural network has in fact "learned" a meaningfull representation, the gradients of the weights of the architecture are computed for each batch, for each epoch. As explained in,[12] when the mean weight gradients are much larger than the corresponding standard deviations, the neural network is still applying an L^2 best fit of the training data. It is when the standard deviation becomes significantly larger than the mean that the neural network starts compressing the data, trying to "forget" features that are not usefull for its predictions. This process can be seen in subfigure (c) of fig. 6. Subfigure (b) contains an explanation of the grouping of weights along the architecture. In particular W_2 contains weights that relate the LSTM input to its output, while W_3 contains weights that

relate the LSTM memory to its output.

2 Hybrid method presented for the Energy Exascale Earth System Model (E3SM)

The time evolution of a physical quantity X_m in the Energy Exascale Earth System Model (E3SM)[13] is given by[14] :

$$\frac{\partial X_m}{\partial t} = F(X_m), \tag{16}$$

where X_m is the state variable. The nudging tendency for the state variable is given by

$$\left(\frac{\partial X_m}{\partial t}\right)_{ndg} = \begin{cases} \frac{X_p(t_p) - X_m(t)}{\tau}, & \text{if the current model time t equals a } t_{n,i}, \\ 0, & \text{if t is not at any } t_{n,i}. \end{cases} \tag{17}$$

Hence the corrected form of Eq.(1) after nudging is given by

$$\left(\frac{\partial X_m}{\partial t}\right)_{\mathbf{cor}} = \left(\frac{\partial X_m}{\partial t}\right)_{\mathbf{uncor}} + \left(\frac{\partial X_m}{\partial t}\right)_{ndg}. \tag{18}$$

For example

Our main objective is to learn the operator $\mathcal{G}$ from $\mathcal{U}$ (before nudge data) to $\mathcal{V}$ (Nudging tendency), which reads

$$\mathcal{G} : \mathcal{U}(x, y, t) \rightarrow \mathcal{V}(x, y, t + \Delta t),$$

by employing a DeepONet based neural operator. The DeepONet is comprised of two neural networks, a branch network for encoding the discrete input function space and a trunk network for encoding the domain of the output functions.[15] It has been shown that the DeepONet is driven by the principle of Universal Approximation Theorem,[16] which is an appropriate architecture for learning mapping from one functional space to another.[17–20] For our case, the input data is described by

$$Q_{\text{bf ndg}}(x, y, t), T_{\text{bf ndg}}(x, y, t), U_{\text{bf ndg}}(x, y, t), V_{\text{bf ndg}}(x, y, t),$$

where $Q_{\text{bf ndg}}(x, y, t)$, $T_{\text{bf ndg}}(x, y, t)$, $U_{\text{bf ndg}}(x, y, t)$ and $V_{\text{bf ndg}}(x, y, t)$ represents the humidity, temperature, east-west wind component and the north-south component before nudging respectively. And the output data is given by

$$Q_{NT}(x, y, t + \Delta t), T_{NT}(x, y, t + \Delta t), U_{NT}(x, y, t + \Delta t), V_{NT}(x, y, t + \Delta t),$$

where $Q_{NT}(x, y, t + \Delta t), T_{NT}(x, y, t + \Delta t), U_{NT}(x, y, t + \Delta t)$ and $V_{NT}(x, y, t + \Delta t)$ represents the nudging tendency for the same variables as discussed above. Figure 7 shows a simple block-diagram for the training process.

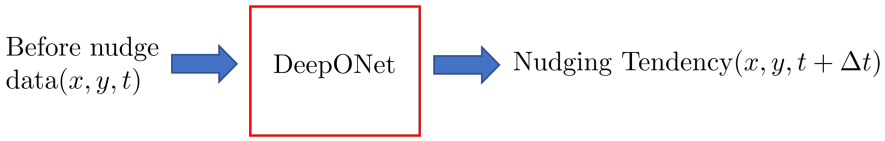

Figure 7: Schematic figure showing the basic flow of the proposed DeepONet model.

2.1 Dataset for training of DeepONet

We have received data generated by using E3SM for one year spaning from December 2009 to December 2010. The data set contains of the state variable Humidity (Q), Temperature (T), East-west wind (U) and North-south wind (V). These data are processed for the data before-nudging and after-nudging with the nudging tendency for each state variables. The shape of each data set is given by (n_x, n_y, n_z, n_t), where n_z represents the vertical atmospheric level, n_y and n_x represents the latitude and longitude respectively, and n_t represents the time when nudging is applied. For the given data-set the nudging frequency is 3 hrs.

To test our deep learning methodology, we decided to train and test the model for a particular patch on the globe. To learn the accurate operator during training, we include as much variability in the data as possible. There are three major large-scale latitudinal circulations, that effect the regional climate as well as the location of subtropical dry zones and mid-latitude jet streams.[21] These are known as the (i) Hadley cell, Ferrell cell and Polar cell and occur in the region as shown in Figure 8.

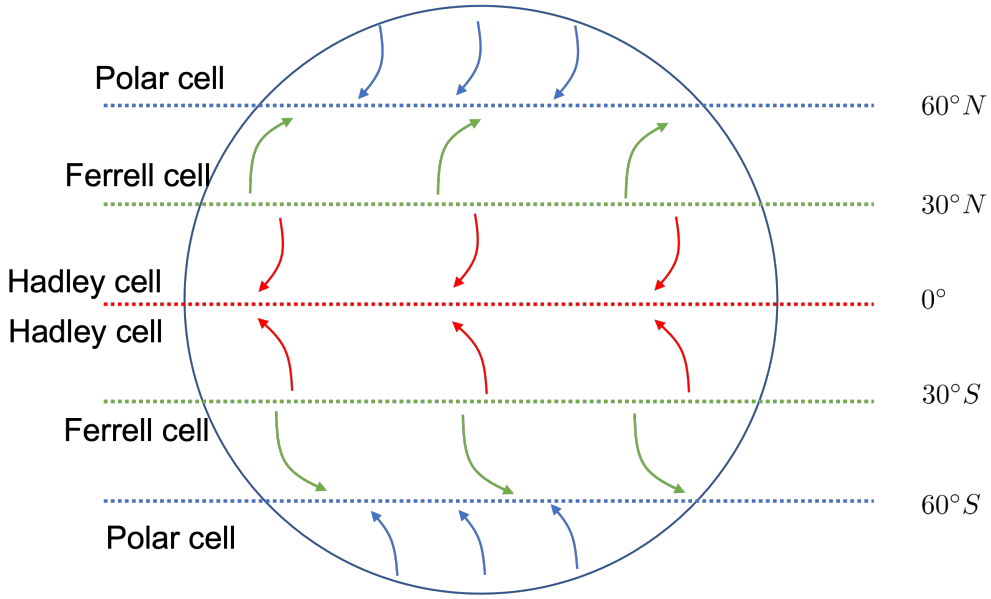

Figure 8: Schematic figure showing the regions of occurrence of Hadley, Ferrell and Polar cell over the entire globe.

Keeping the requirement of variability in mind, the following patch was selected that includes parts of all the three major cycles as described above.

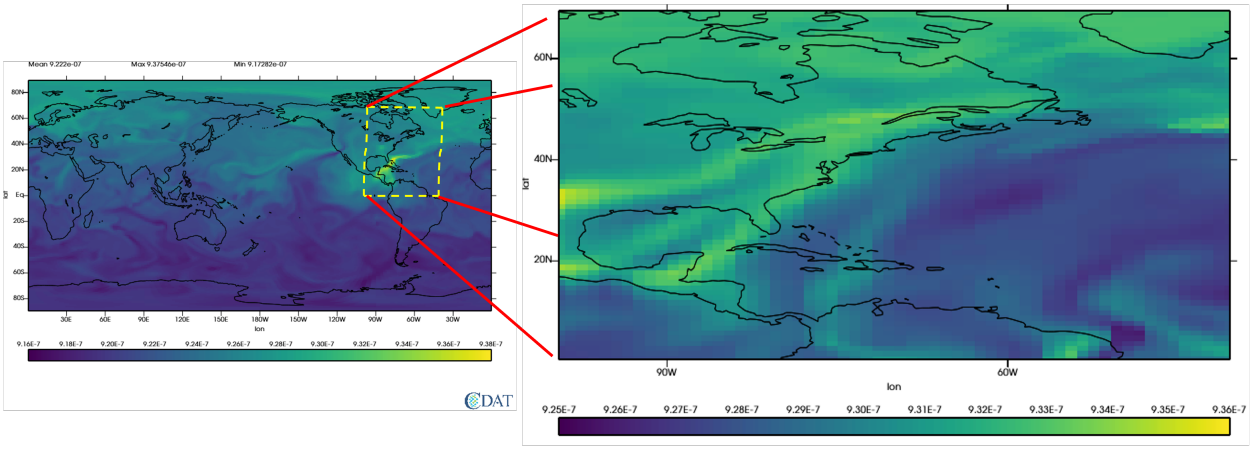

Figure 9: Schematic figure showing the patch selected for DeepONet training.

2.2 DeepONet Architecture

We propose a DeepONet with a trunk network and a branch network consisting four sub branch networks, each corresponding to one state variable. The input to each sub branch network is a sequence of snapshots. We use batch mode of training and consider the different atmospheric levels $n_z = 72$ as different realizations. Each snapshot, for example, is of the size (n_x, n_y, n_t). So we have n_z realizations, where each realization is a sequence of n_t snapshots. We rely on the inductive bias of Convolutional Neural Network (CNN) for extracting latent low dimensional features from the sequence of snapshots. For e.g., we have

- an input of shape $(n_x \times n_y \times n_z \times n_t)$,
- a filter of size f.
- a padding p and
- a stride of s.

We first rearrange the axes of the image in the form $(n_z \times n_t \times n_y \times n_x)$, in order to use n_z as the batching dimension. Then the width of the output size O_w is given by the formula

$$O_w = \frac{n_x - f + 2p}{s} + 1,$$

the height of the output size O_h is given by the formula

$$O_h = \frac{n_y - f + 2p}{s} + 1,$$

and the the depth of the output size O_d is given by the formula

$$O_d = \frac{n_t - f + 2p}{s} + 1.$$

The underlying architecture of the CNN used is shown in Figure 10. In the simplest case of

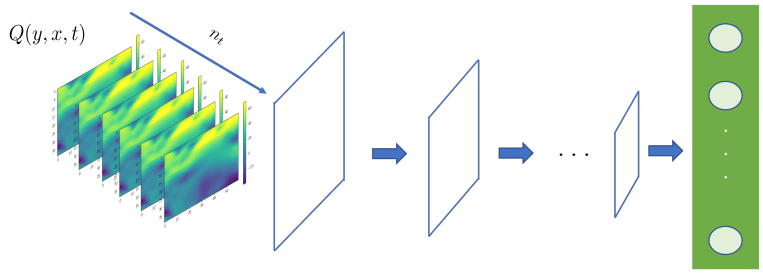

Figure 10: Schematic figure showing the general structure of the CNN used in the branch network of DeepONet for one realization of $Q(y,x,t)$.

the CNN, the output value of the layer having an input size $(n_z, n_y, n_x, [n_t]_{\text{in}})$ and output $(n_z, [n_y]_{\text{out}}, [n_x]_{\text{out}}, [n_t]_{\text{out}})$ can be described as:

$$\text{out}\left([n_z]_i, [n_t]_{\text{out }j}\right) = \text{bias}\left([n_t]_{\text{out }j}\right) + \sum_{k=0}^{[n_t]_{\text{in}}-1} \text{weight}\left([n_t]_{\text{out }j}, k\right) \star \text{input}\left([n_z]_i, k\right), \quad (19)$$

where $[n_t]_{in} =$ and $[n_t]_{out}$, is the input channel and output for the CNN layer, n_z represents the batch dimension for each realization. and n_y and n_x represents the width and height of input image. Each CNN is followed by a dense layer of the size ld_1. We then concatenate the outputs of all the sub branch networks, and reshape it to a tensor of the shape $(n_z, 4ld_1)$. This is then feed into a Fully-connected Neural Network (FNN) having an output size of ld_2. So the final shape of the tensor from the branch network is $(n_z, 4ld_2)$, which is reshaped to $(n_z, 4, ld_2)$. However, the architecture of trunk net is a vanilla FNN. The input to the trunk network are the variables (y, x, t). The output of the trunk network is a tensor of the shape $(n_t, 4ld_2)$. Then after an Einstein summation convention is performed on the output of branch net and trunk net. This results into a tensor of the shape $(n_z, 4, n_t)$, which is then compared to the corresponding nudging tendency from the E3SM model. So the loss function in general can be written as

$$\mathcal{L} = ||U_D - Nudge_U||_2 + ||V_D - Nudge_V||_2 + ||Q_D - Nudge_Q||_2 + ||Q_D - Nudge_T||_2, \quad (20)$$

where X_D is the DeepONet output and $Nudge_X$ is the nudging tendency from the E3SM model for $X = Q, T, U$ and V. The overall structure of the DeepONet is shown in Figure 11.

3 Next Steps

Our immediate next steps is to test our hybrid method with beta data, i.e. with the QG model. In fact this phase has been completed successfully and we currently compile the report (this is Milestone 4). Additional developments related to data quality and active selection of additional simulations that will increase the accuracy of the hybrid methodology will be reported in future milestone reports.

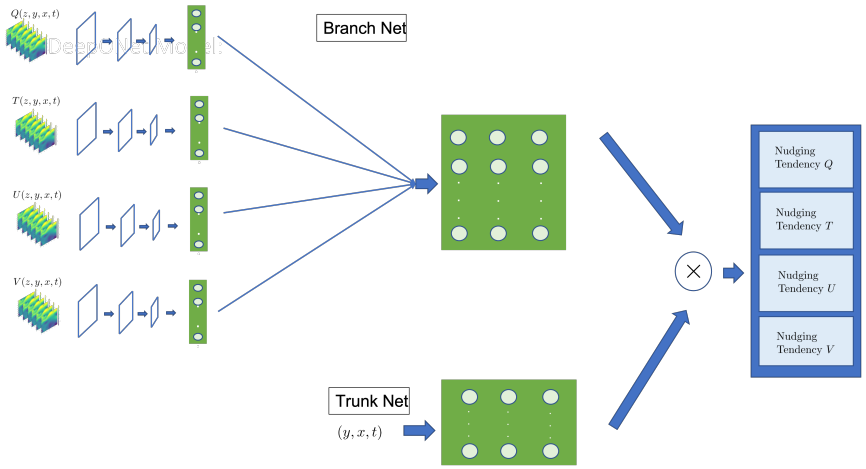

Figure 11: Schematic figure showing the general structure of the DeepONet.

References

[1] Rick Salmon. *Lectures on geophysical fluid dynamics*. Oxford University Press, 1998.

[2] Di Qi and Andrew J Majda. Predicting extreme events for passive scalar turbulence in two-layer baroclinic flows through reduced-order stochastic models. *Communications in Mathematical Sciences*, 16(1):17–51, 2018.

[3] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

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

[5] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical systems and turbulence, Warwick 1980*, pages 366–381. Springer, 1981.

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

[7] Alexis-Tzianni G Charalampopoulos and Themistoklis P Sapsis. Machine-learning energy-preserving nonlocal closures for turbulent fluid flows and inertial tracers. *Physical Review Fluids*, 7(2):024305, 2022.

[8] Zhong Yi Wan, Boyko Dodov, Christian Lessig, Henk Dijkstra, and Themistoklis P Sapsis. A data-driven framework for the stochastic reconstruction of small-scale features with application to climate data sets. *Journal of Computational Physics*, page 110484, 2021.

[9] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

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

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

[12] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[13] DOE E3SM Project. Energy exascale earth system model v1.0. [Computer Software] `https://doi.org/10.11578/E3SM/dc.20180418.36`, apr 2018.

[14] Jian Sun, Kai Zhang, Hui Wan, Po-Lun Ma, Qi Tang, and Shixuan Zhang. Impact of nudging strategy on the climate representativeness and hindcast skill of constrained eamv1 simulations. *Journal of Advances in Modeling Earth Systems*, 11(12):3911–3933, 2019.

[15] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

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

[17] Chensen Lin, Zhen Li, Lu Lu, Shengze Cai, Martin Maxey, and George Em Karniadakis. Operator learning for predicting multiscale bubble growth dynamics. *The Journal of Chemical Physics*, 154(10):104118, 2021.

[18] Minglang Yin, Enrui Zhang, Yue Yu, and George Em Karniadakis. Interfacing finite elements with deep neural operators for fast multiscale modeling of mechanics problems. *arXiv preprint arXiv:2203.00003*, 2022.

[19] Lizuo Liu and Wei Cai. Multiscale deeponet for nonlinear operators in oscillatory function spaces for building seismic wave responses. *arXiv preprint arXiv:2111.04860*, 2021.

[20] Shengze Cai, Zhicheng Wang, Lu Lu, Tamer A Zaki, and George Em Karniadakis. Deepm&mnet: Inferring the electroconvection multiphysics fields based on operator approximation by neural networks. *Journal of Computational Physics*, 436:110296, 2021.

[21] Donald J Wuebbles, David W Fahey, and Kathy A Hibbard. Climate science special report: fourth national climate assessment, volume i. 2017.