A coupled Variational Encoder-Decoder - DeepONet surrogate model for the Rayleigh-Bénard convection problem

João Lucas de Sousa Almeida[1], **Pedro Roberto Barbosa Rocha**[1,2]
Allan Moreira de Carvalho,[1], **Alberto Costa Nogueira Jr.**[1]
[1]IBM Research Brazil,
[2]Pontifical Catholic University of Rio de Janeiro

Abstract

Fluid mechanics continues to advance quickly in the age of artificial intelligence, mainly due to the abundance of experimental data, field data assimilation, and high-fidelity multiscale simulations. Among the many data-driven approaches recently applied to such a discipline, ML-based reduced-order models (ROMs) have received particular attention because of their algorithmic simplicity, explainability, and computational efficiency. In this work, we have devised and implemented an ML-based ROM which combines dimensionality reduction via an Encoder-Decoder (ED) neural network with forecasting capabilities in latent space using Deep Neural Operators (DeepONets). We assessed the proposed architecture with a spatiotemporal dataset generated by the numerical solution of the Rayleigh-Bénard convection (RBC) problem. The reconstruction error of the model over the unseen datasets was lower than 10%, demonstrating the ED technique's accurate spatial representation and the neural operators' robustness in estimating future system states. This work represents a solid contribution to the fluid dynamics community with an accurate and efficient ML-based model to tackle the challenging well-known RBC problem.

1 Introduction

In recent years, machine learning (ML) has gained momentum in many fields of science and engineering, especially in fluid dynamics applications, where new algorithms and architectures have emerged and tackled increasingly complex problems. The unprecedented availability of high-performance hardware specialized in machine learning tasks, such as GPUs and TPUs, has motivated the development of a broad class of ML techniques that extract information from data in a principled way to capture the underlying fluid mechanics. Among these techniques, surrogate models have stood out as reliable and computationally efficient alternatives to classical numerical methods, such as finite volume or spectral discretizations, when repeated simulation runs are required.

Many approaches for building surrogates have been proposed so far, but the subclass of the reduced order models (ROM) has particularly attracted the attention of the research community due to its core feature: the compression of complex multidimensional datasets into affordable sets of time series allowing parsimonious and, usually, explainable inference or prediction capabilities. Two of the most popular dimensionality reduction (DR) techniques are Principal Components Analysis (PCA) and Encoder-Decoder neural networks (EDs). The PCA algorithm relies on a linear transformation of coordinates to describe most of the data variation with fewer dimensions than the original data. PCA shows many favorable properties, such as orthonormal basis representation and energy preservation. It is often seen in many fluid dynamics applications (Lui and Wolf 2019) (Costa Nogueira et al. 2020) (McQuarrie, Huang, and Willcox 2021), being quite robust for describing even chaotic (Almeida et al. 2022) and turbulent (McQuarrie, Huang, and Willcox 2021) regimes. EDs are usually more computationally expensive than PCA since their training relies heavily on gradient-descent optimization. However, as they allow nonlinear basis representation in latent space, they potentially provide a better description of the dynamical systems of interest (Oommen et al. 2022). EDs may be constructed on either Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN) or even Graph Neural Network (GNN) architectures, among others. The low-dimensional space encoded by any DR technique usually consists of a set of time series whose accurate extrapolation can become extremely challenging even for relatively simple dynamical systems.

The problem of predicting time series has been traditionally addressed using Recurrent Neural Networks (RNNs) (Pathak et al. 2017) (Vlachas et al. 2020) (Nogueira Jr et al. 2021) and, more recently, transformer-based architectures (Zeng et al. 2022). Nevertheless, neural operators as Deep Operator Networks (DeepONets) (Lu et al. 2021) and Fourier Neural Operators (FNOs) (Li et al. 2021) have also demonstrated excellent time series forecasting skills (Oommen et al. 2022) (Pathak et al. 2022). This work is inspired by the schemes presented in (Oommen et al. 2022) and combines CNN-ED for the dimensionality reduction of a large multidimensional fluid dynamics dataset with an improved version of a DeepONet (Wang, Wang, and Perdikaris 2021). The latter was fine-tuned to extrapolate latent time series of the dynamical system. All models implemented in this work used the **SimulAI** toolkit, an open-source Python package aimed to accelerate surrogate modeling using Physics-Informed machine learning (IBM 2022).

This work is organised as follows. In Section 2 we briefly describe the Rayleigh-Bénard dataset we used as test case and the simulation parameters necessary to generate it. Section 3 describes the architecture of the proposed CNN-based ED and its training procedure. In this section, we also demonstrate the effectiveness of the ED technique in reducing and reconstructing the original dataset. Section 4 depicts the DeepONet architecture and presents the way the training dataset was prepared and the characteristics of the training procedure. In the last subsection, we present and discuss the numerical results regarding the complete ROM pipeline. We conclude the paper in Section 5, summarizing the proposed model's main accomplishments and limitations, besides giving some perspectives for future work.

2 The Rayleigh-Bénard dataset

To assess the present ML-based surrogate model, we chose the Rayleigh-Bénard convection (RBC) system, which is a typical benchmark problem for emulating turbulent atmospheric circulation. We used the package Dedalus (Burns et al. 2020), a Python framework for discretizing partial differential equations (PDEs) based on the spectral element method, to generate a large spatiotemporal dataset. The full-order RBC dataset contains $400,000$ samples with time resolution $dt = 10^{-6}$, grid size of 80×80, Rayleigh number $Ra = 10^7$ and variables $[u, w, T]$, corresponding to the x-velocity, z-velocity and temperature fields, respectively. In order to enable more effective learning of the neural networks (and for enhancing the visualization experience), we normalized all state variables to the interval $[-1, 1]$.

3 Dimensionality reduction

Overview

Encoder-Decoder (ED) architectures based on neural networks are commonly used in tasks involving data representation, since they transform the original data into an intermediary space referred as latent space where modeling is usually facilitated by the lower number of dimensions. This process encodes the main features of the dynamical system of interest and subsequently re-convert the latent states into the original space. However, the encoding process can also increase the dimension of the input space (a.k.a. lifting operation) (Brunton et al. 2021). In the former case, dimensionality is reduced in the same sense as PCA, but in a more general way, since ED can support nonlinear transformations. There is a number of architecture choices to build an Encoder-Decoder model to compress dimensionality. In this work, we adopted the Convolutional Neural Networks (CNNs) since the samples of the RBC dataset can be viewed as images whose channels are the different field variables. In our perspective, CNNs seems to be the most suitable choice for composing the encoder and the decoder stages.

In order to obtain a robust latent space representation, we opted for a Convolutional Neural Network-based Variational Encoder-Decoder (CNN-VED) (Eivazi et al. 2022) since VEDs may be used as generative models in the sense that they can generate new reliable samples of data. Such a property is a valuable feature if one is interested in extending the original dataset for comprising slightly different scenarios. Furthermore, the VED statistical operation is similar to applying noise to the latent space during the training stage which works as regularizing procedure (Behrens et al. 2022). In this way, we did not use any additional penalization-like regularizer over the VED paramters $\boldsymbol{\theta}_{\mathcal{R}}$. A diagram illustrating the proposed CNN-VED architecture can be seen in Figure 1.

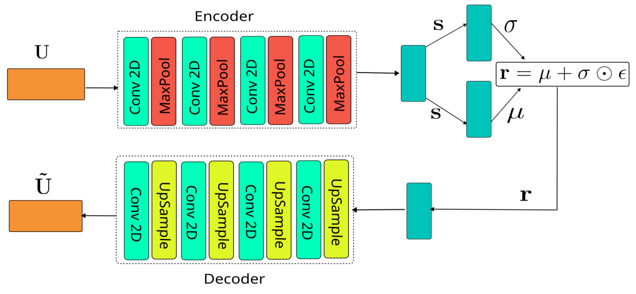

Figure 1: The CNN-VED architecture.

CNN-VED training and results

The details of the proposed CNN-VED architecture are described in Table 1 (Appendix A). All the convolution operations used padding equal to 1. It means that the input image dimensions are not modified after the convolution operations for a kernel size of 3×3 since one row or column of zeros is added on each image boundary in order to keep the dimensions unmodified. For the decoder, we used upsampling operations for increasing the image dimensions after each convolution instead of transposed convolutions, which increased convergence rate. The upsampling operations enlarge image sizes by interpolating between pixels. In the present work, we chose a re-scaling factor of 2 and bi-cubic interpolation. We empirically determined the number of latent series as $N_d = 20$ for the CNN encoder output s. The variable s is inputted in two auxiliary sub-networks in order to evaluate the mean μ and the standard deviation σ of the statistical operation used for estimating the latent variable r, given by $r = \mu + \sigma \odot \epsilon$, in which $\odot$ represents pointwise multiplication and ϵ is a random number generated by a normal distribution (Eivazi et al. 2022). Sub-networks revealed helpful to allow more flexibility for describing the variables related to the probabilistic distribution. Figure 1 depicts the CNN-VAE architecture. We used $345,000$ samples for training the VED model and discarded the first $15,000$ in order to avoid transient effects. We assessed its accuracy over the last unseen $40,000$ timesteps. Some examples of the time series generated for the training region of the dataset associated to the modes $0, 4, 9, 14$ and 19 are seen in Figure 2.

$$\mathcal{L}_{VED}(\boldsymbol{\theta}_{\mathcal{R}}) = \mathcal{L}_{rec}(\boldsymbol{\theta}_{\mathcal{R}}) + \mathcal{L}_{KL}(\boldsymbol{\theta}_{\mathcal{R}}) \tag{1}$$

$$\mathcal{L}_{rec}(\boldsymbol{\theta}_{\mathcal{R}}) = \frac{1}{N_b}\sum_{i=0}^{N_b-1}\left(\tilde{\mathbf{U}}_i - \mathbf{U}_i\right)^2$$

$$\mathcal{L}_{KL}(\boldsymbol{\theta}_{\mathcal{R}}) = -\frac{\beta}{N_b}\sum_{i=0}^{N_b-1}\sum_{j=0}^{N_d-1}(1 + log(\sigma_{ij}^2) - \mu_{ij}^2 - \sigma_{ij}^2)$$

We trained the network for $100,000$ epochs using the Adam optimizer. We set the initial learning rate as $lr = 10^{-3}$ with exponential decay of 0.9 at every $5,000$ epochs and batch size N_b of $1,000$. The entire model training was performed in a V100 Nvidia GPU. The loss function used for adjusting the VED parameters is written in Equation 1, which is composed by two terms, a squared reconstruction error $\mathcal{L}_{rec}$ and the statistics loss $\mathcal{L}_{KL}$ (Eivazi et al. 2022), also known as Kullback-Leibler (KL) loss term (Behrens et al. 2022). Regarding the parameter β, used for penalizing the KL term, we followed the literature (Behrens et al. 2022) and set it to 0.5. After the training stage, we used the relative L^2-norm (cf. Equation 2) for evaluating the compression loss ε. This norm compares the original field variables dataset $\mathbf{U}$ with the output generated by the encoding-decoding process $\tilde{\mathbf{U}}$ (cf. Figure 1). The VED we trained produced a relative compression loss of 2.17%. A visual comparison between the original and reconstructed snapshots with respect to the last time step of the testing dataset is shown in Figure 3.

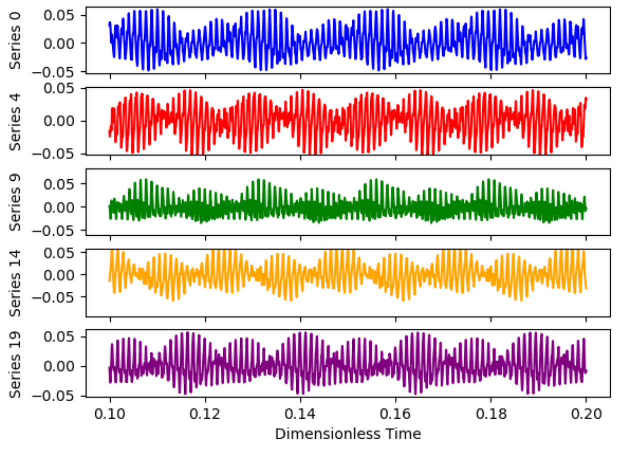

Figure 2: Examples of time series generated by CNN-VED for the training dataset.

$$\varepsilon(\mathbf{U}, \tilde{\mathbf{U}}) = 100\sqrt{\sum_{i=0}^{N-1}\left(\tilde{\mathbf{U}}_i - \mathbf{U}_i\right)^2} \Big/ \sqrt{\sum_{i=0}^{N-1}\mathbf{U}_i^2} \tag{2}$$

Once trained, the CNN-VED was applied to compress the initial full-order dataset from $(4 \times 10^5, 80, 80, 3)$ dimensions to only $(4 \times 10^5, 20)$. The latent reduced space dataset was then used to train the DeepONet to perform time-extrapolation.

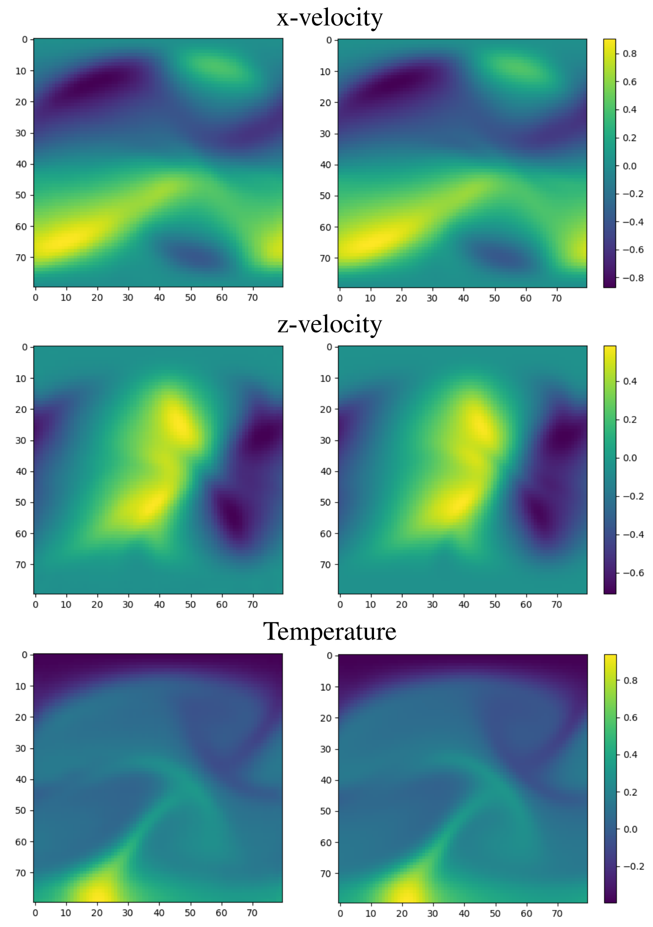

Figure 3: Examples of reconstructed (left) and ground truth (right) snapshots for the dimensionaless state variables u, w and T for the last time step of the testing dataset. Although one can notice some misrepresentations, the CNN-VED is able to depict the most important flow features.

4 DeepONets for time-extrapolation

Overview

Deep Operator Networks (DeepONets) (Lu et al. 2021) are a relatively recent class of neural network architectures aimed at better approximating function operators. The conventional DeepONet architecture is composed by two subnetworks, the branch and the trunk, each one dedicated to a specific kind of input. Trunk usually deals with inputs related to coordinates and branch with wider scope inputs as forcing terms, physical parameters and even boundary or initial conditions, which we will term simply restrictions. Broadly speaking, the trunk network works by searching suitable bases for representing a dataset and the branch network estimates the most proper weights for the linear combination of these bases. As there is no *a priori* restriction with respect to the trunk and branch architectures, a broad variety of choices has been tested since the DeepONets appearance, as MLPs (Lu et al. 2021), CNNs (Oommen et al. 2022) and even Spiking Neural Networks (SNNs) (Kahana et al. 2022).

DeepONets have been successfully employed in many research fields as modelling multiscale bubble growth (Lin et al. 2021), linear instabilities in boundary layers (Leoni et al. 2021), evolution of two-phase microstructures (Oommen et al. 2022) and multifidelity simulation (Howard et al. 2022), in which Physics-Informed DeepONets are used for learning complex operators from low-fidelity data.

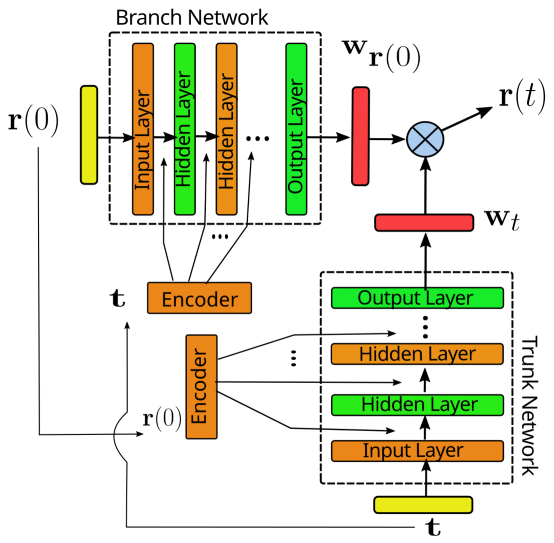

Figure 4: The so-called improved DeepONet architecture.

Recently, it was introduced the so-called improved DeepONet architecture (Wang, Wang, and Perdikaris 2021), as shown in Figure 4. This framework consists of two new subnetworks (termed as encoders) added to the original architecture in order to encode the different inputs into intermediary embeddings. The latter are shared among the hidden layers of the trunk and the branch networks, thereby enforcing a stronger communication between the two DeepONet main components. The improved DeepONet was first used in the context of long-term time-integration tasks (Wang, Wang, and Perdikaris 2021) that motivated its usage in this work as a proper time series forecasting algorithm.

Data preparation

We want to train a DeepONet that receives an initial state $u_0 = u(t_0)$ in the latent space time series and predicts any subsequent state $u(t)$ for $t \in [t_0, t_0 + \Delta t]$, with the time step size Δt preliminarily chosen and treated as a hyperparameter. After training a DeepONet capable of producing reliable predictions for the interval $[t_0, t_0 + \Delta t]$, given any initial state u_0, we can compose extrapolations by using, for instance, $u(t_0 + \Delta t)$ as the new initial state u_0 and continuously predict states for new intervals of size Δt, thus enabling long-term extrapolations.

In order to train the DeepONet, we preprocess the original dataset and create a triple $(\mathcal{D}_t, \mathcal{D}_b, \mathcal{T})$ of sets (cf. Figure 5). The set $\mathcal{D}_t$ is the trunk input and consists of a concatenation of time steps belonging to the interval $[t_0, t_0 + \Delta t]$. $\mathcal{D}_b$ is inputted to the branch network and is composed by multiple initial states whose time steps are in $\mathcal{D}_t$. $\mathcal{T}$ is the target dataset, in which every sample corresponds to a single timestep in $\mathcal{D}_t$.

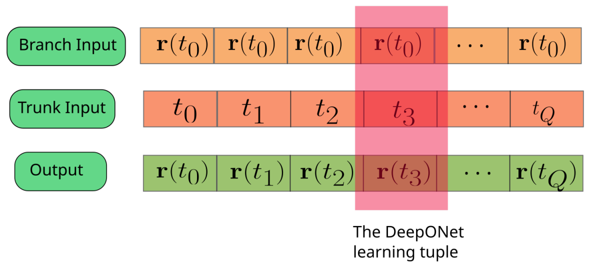

Figure 5: The triple of datasets used for training a time-integrator DeepONet.

To illustrate how the triple datasets are constructed, we can imagine a moving window of fixed size with Q slots being shifted along the latent-space time series (cf. Figure 6). All slots of this window can be considered as a "chunk" of data, whose first position is the initial state and the remaining ones are the target outputs.

The initial state is repeated for each output as a means of binding initial condition, elapsed time and corresponding target state. This approach for generating the training dataset allows a considerable flexibility since we can slide the window using an arbitrary choice for the skip size and we can arbitrarily sample states within each chunk instead of choosing them sequentially. Nevertheless, for the sake of simplicity, we adopted fixed skip sizes of value 10 and took entire chunks. Figure 6 depicts a schematic view of the datasets organization. It is worth noticing that the moving window approach for preparing the DeepONet datasets can be viewed as a data augmentation technique since we are considering time steps repeatedly in the latent-space which creates new correlations between data samples. Such data preparation strategy improves the model predictability even with limited data availability.

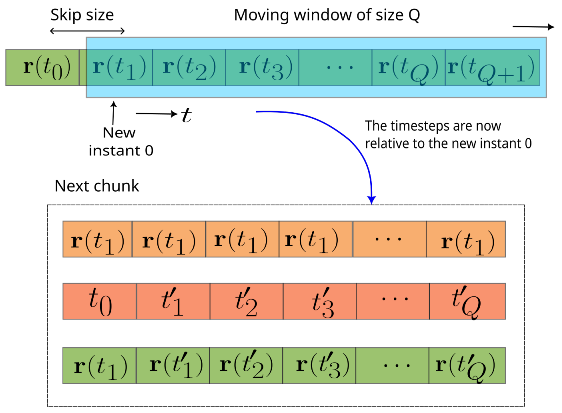

Figure 6: The concept of moving window used for constructing the datasets triple. The time axis is always relative to the initial state and each new position of the window restarts the time counting.

On top of the branch and trunk forward operations, there is a dot product computed in batches represented as $\otimes$ in Figure 4 and defined in Equation 3 below.

$$\mathbf{r}_i = \sum_{j=M_i}^{M_i+q_i} w_{r(0)_j} w_{t\,j}, \; \mathbf{r} \in \mathbb{R}^{N_v} \tag{3}$$

$$M_0 = 0; \; M_i = \sum_{j=0}^{i-1} q_j, \; 1 \leq i < N_v; \; \sum_{j=0}^{N_v-1} q_j = P$$

where r_i is the estimated system state, $w_{r(0)_j}$ and $w_{t\,j}$ are the outputs of the branch and trunk networks, respectively, q_i is the number of branch or trunk outputs in each batch (which must be equal) used for estimating the variable r_i via the $\otimes$ inner product, P is the total number of outputs of the trunk and branch networks and N_v is the maximum number of variables outputted by the DeepONet.

To keep simplicity, we choose the sizes q_i as equal. The parameters $\boldsymbol{\theta}$ of the DeepONets are sought so that they minimize the loss function defined in Equation 4, which consists of a weighted relative mean squared norm with a batch size N_b with an L^2 regularisation term added to mitigate overfitting. As the time series are well scaled, we set the coefficients α_j as 1 and the regularisation penalty λ as 10^{-12}.

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_{pred}(\boldsymbol{\theta}) + \lambda \mathcal{L}_{reg}(\boldsymbol{\theta}) \tag{4}$$

$$\mathcal{L}_{pred}(\boldsymbol{\theta}) = \sum_{j=1}^{N_v} \alpha_j \left(\sum_{i=1}^{N_b} (\tilde{r}_{i\,j} - r_{i\,j})^2 / \sum_{i=1}^{N_b} r_{i\,j}^2 \right)$$

$$\mathcal{L}_{reg}(\boldsymbol{\theta}) = \sum_{k=1}^{N_\theta} \theta_k^2$$

Training and results

Both the branch and the trunk DeepONet architectures are comprised by 7-MLP networks whose configurations are detailed in Table 2 of Appendix B. We trained the model for $200,000$ epochs using the Adam optimizer. We set initial learning rate $lr = 10^{-3}$ with exponential decay of 0.9 at every $10,000$ epochs and specified the batch sizes as $1,000$. The training stage was performed in a V100 Nvidia GPU. The chunk size Q was chosen as $1,000$ and P as 200. Considering the dataset time-resolution and the value of Q, we fixed the chunk sizes in the time axis as $\Delta t = 10^{-3}$. Once trained, we applied the DeepONet to predict the time series in latent space. We can visualize five of these time series in Figure 7.

Then, we used the decoder stage of the CNN-VED for mapping the extrapolated time series to the full-order space, as displayed in Figure 8. Using the error metric defined in Equation 2, we achieved a reconstruction error of approximately 6.2% after the time extrapolation (see Appendix C for more statistics). The extrapolation/reconstruction process took approximately 8.8 seconds in a single Nvidia V100 GPU. The CNN-VED-DeepONet revealed a quite compact model requiring less than 5 MB of disk to store the trained network with mixed precision. A reconstruction error of

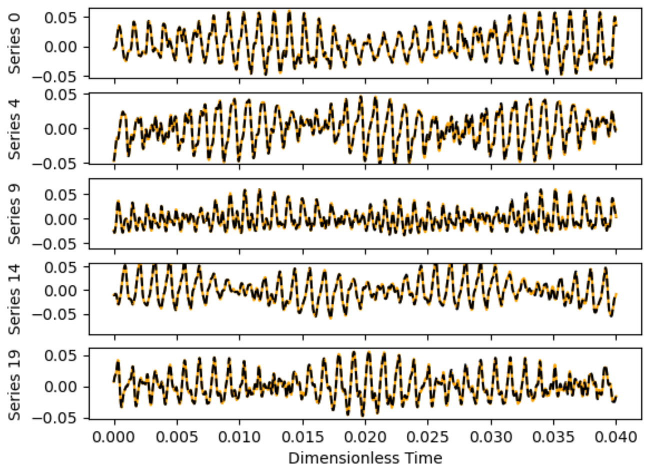

Figure 7: Latent space time series comparison between ground-truth (continuous orange lines) and DeepONet predictions (black dashed lines) for the testing dataset. For the sake of compactness, we exhibit only 5 latent time series.

such a magnitude is quite significant considering the complex multi-scale patterns generated by the turbulent flow. Experiments carried on by the same authors in an unpublished work using the non-intrusive operator inference (Peherstorfer and Willcox 2016) technique revealed a very good agreement with the level of accuracy shown in these results. It is worth mentioning that we opted for a VAE offline training to re-use the pre-trained encoder with other time-extrapolation architectures than the DeepONet, enhancing the flexibility of the proposed ROM. Although the online training, which concurrently optimizes the VAE and DeepONet parameters in a single loss function, seems attractive, it is arguable that it could produce a fine-tuned ROM with an improved prediction process. However, the training process would undoubtedly be costly.

5 Discussion, Limitations and Future Works

The proposed ML-pipeline CNN-VED-DeepONet showed accurate forecasting capabilities for the benchmark RBC problem in turbulent regime. The approximation error of the reconstructed model for the entire time interval was considerably small ($\approx 6.2\%$) for the chosen spatiotemporal dataset. DeepONet forecast preserved the topology of the time series even for long-term extrapolations. Moreover, we observed that DeepONets also admit continuous spaces as inputs, meaning that the inferred DeepONet operator $\mathcal{G}$ is naturally resolution independent and can be seamlessly trained with datastes of different quality and sizes. Although the results in this work are quite encouraging, we should keep in mind that the time series in the testing dataset are still well-behaved for the turbulent regime driven by the Rayleigh number $Ra = 10^7$. Turbulent dynamics with higher Rayleigh numbers will impose extra difficulties to build an accurate low dimension manifold representation and to capture the chaotic fluctuations in each latent time series. Thus, as a next step to improve robustness and gen-

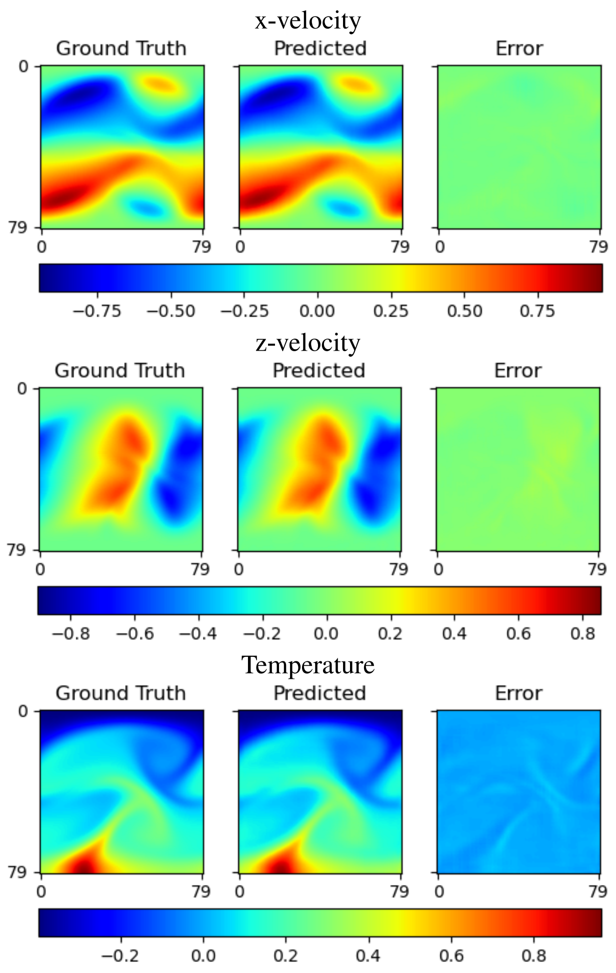

Figure 8: Ground truth vs. CNN-VAE-DeepONet predictions for the dimensionless variables u, w and T at the last time instant of the unseen dataset. The rightmost column shows the pointwise error for each state variable. It is worth noting that absolute errors are more convenient than relative percentage errors in this plot since the reference variables can take null values conveying useless visual information.

erality of the CNN-VED-DeepONet surrogates, we consider to assess this architecture using RBC test cases with higher Ra numbers and move on to tackle even more challenging problems such as oceanic and atmospheric flows.

6 Acknowledgments

We want to thank the IBM Research Brazil Lab for supporting the authors and providing the computational resources necessary to accomplish this work.

References

Almeida, J. L. d. S.; Pires, A. C.; Vaz, K. C. F.; and Nogueira Junior, A. C. 2022. Non-Intrusive Reduced Models based on Operator Inference for Chaotic Systems. *arXiv preprint arXiv:2206.01604v1*.

Behrens, G.; Beucler, T.; Gentine, P.; Iglesias-Suarez, F.; Pritchard, M.; and Eyring, V. 2022. Non-Linear Dimensionality Reduction With a Variational Encoder Decoder to Understand Convective Processes in Climate Models. *Journal of Advances in Modeling Earth Systems*, 14(8).

Brunton, S. L.; Budišić, M.; Kaiser, E.; and Kutz, J. N. 2021. Modern Koopman Theory for Dynamical Systems.

Burns, K. J.; Vasil, G. M.; Oishi, J. S.; Lecoanet, D.; and Brown, B. P. 2020. Dedalus: A flexible framework for numerical simulations with spectral methods. *Physical Review Research*, 2(2): 023068. In press.

Costa Nogueira, A.; de Sousa Almeida, J. L.; Auger, G.; and Watson, C. D. 2020. Reduced Order Modeling of Dynamical Systems Using Artificial Neural Networks Applied to Water Circulation. In Jagode, H.; Anzt, H.; Juckeland, G.; and Ltaief, H., eds., *High Performance Computing*, 116–136. Cham: Springer International Publishing. ISBN 978-3-030-59851-8.

Eivazi, H.; Le Clainche, S.; Hoyas, S.; and Vinuesa, R. 2022. Towards extraction of orthogonal and parsimonious nonlinear modes from turbulent flows. *Expert Systems with Applications*, 202: 117038. In press.

Howard, A. A.; Perego, M.; Karniadakis, G. E.; and Stinis, P. 2022. Multifidelity Deep Operator Networks.

IBM. 2022. SimulAI Toolkit: A Python package with data-driven pipelines for physics-informed machine learning. https://github.com/IBM/simulai.

Kahana, A.; Zhang, Q.; Gleyzer, L.; and Karniadakis, G. E. 2022. Spiking Neural Operators for Scientific Machine Learning.

Leoni, P. C. d.; Lu, L.; Meneveau, C.; Karniadakis, G. E.; and Zaki, T. A. 2021. DeepONet prediction of linear instability waves in high-speed boundary layers.

Li, Z.; Kovachki, N.; Azizzadenesheli, K.; Burigede, L.; Bhattacharya, K.; Stuart, A.; and Anandkumar, A. 2021. Fourier neural operator for parametric partial differential equations. In *nternational Conference on Learning Representations (ICLR)*. In press.

Lin, C.; Li, Z.; Lu, L.; Cai, S.; Maxey, M.; and Karniadakis, G. E. 2021. Operator learning for predicting multiscale bubble growth dynamics. *The Journal of Chemical Physics*, 154(10): 104118. In press.

Lu, L.; Jin, P.; Pang, G.; Zhang, Z.; and Karniadakis, G. E. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3: 218–229. In press.

Lui, H. F. S.; and Wolf, W. R. 2019. Construction of reduced-order models for fluid flows using deep feedforward neural networks. *Journal of Fluid Mechanics*, 872: 963–994. In press.

McQuarrie, S. A.; Huang, C.; and Willcox, K. E. 2021. Data-driven reduced-order models via regularised Operator Inference for a single-injector combustion process. *Journal of the Royal Society of New Zealand*, 51(2): 194–211. In press.

Nogueira Jr, A. C.; Carvalho, F. C.; Almeida, J. L. S.; Codas, A.; Bentivegna, E.; and Watson, C. D. 2021. Reservoir Computing in Reduced Order Modeling for Chaotic Dynamical Systems. In *International Conference on High Performance Computing*, 56–72. Springer, Cham.

Oommen, V.; Shukla, K.; Goswami, S.; Dingreville, R.; and Karniadakis, G. 2022. Learning two-phase microstructure evolution using neural operators and autoencoder architectures. *npj Computational Materials*, 8.

Pathak, J.; Lu, Z.; Hunt, B. R.; Girvan, M.; and Ott, E. 2017. Using machine learning to replicate chaotic attractors and calculate Lyapunov exponents from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(12): 121102.

Pathak, J.; Subramanian, S.; Harrington, P.; Raja, S.; Chattopadhyay, A.; Mardani, M.; Kurth, T.; Hall, D.; Li, Z.; Azizzadenesheli, K.; Hassanzadeh, P.; Kashinath, K.; and Anandkumar, A. 2022. FourCastNet: A Global Data-driven High-resolution Weather Model using Adaptive Fourier Neural Operators. *arXiv preprint arXiv:2202.11214*.

Peherstorfer, B.; and Willcox, K. 2016. Data-driven operator inference for nonintrusive projection-based model reduction. *Computer Methods in Applied Mechanics and Engineering*, 306: 196–215. In press.

Vlachas, P. R.; Pathak, J.; Hunt, B. R.; Sapsis, T. P.; Girvan, M.; Ott, E.; and Koumoutsakos, P. 2020. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126: 191–217. In press.

Wang, S.; Wang, H.; and Perdikaris, P. 2021. Improved architectures and training algorithms for deep operator networks. *arXiv preprint arXiv:2110.01654*.

Zeng, A.; Chen, M.; Zhang, L.; and Xu, Q. 2022. Are Transformers Effective for Time Series Forecasting? *arXiv preprint arXiv:2205.13504*.

A Neural network architecture for VAE

Table 1: The CNN-VED architecture

		Encoder			
Layer	Kernel Size / Re-scaling	Width	Activation	Input size	Output size
Conv2D	3×3	16	tanh	$80 \times 80 \times 3$	$80 \times 80 \times 16$
MaxPooling	2×2	-	-	$80 \times 80 \times 16$	$40 \times 40 \times 16$
Conv2D	3×3	32	tanh	$40 \times 40 \times 16$	$40 \times 40 \times 32$
MaxPooling	2×2	-	-	$40 \times 40 \times 32$	$20 \times 20 \times 32$
Conv2D	3×3	64	tanh	$20 \times 20 \times 32$	$20 \times 20 \times 64$
MaxPooling	2×2	-	-	$20 \times 20 \times 64$	$10 \times 10 \times 64$
Conv2D	3×3	128	tanh	$10 \times 10 \times 64$	$10 \times 10 \times 128$
MaxPooling	2×2	-	-	$10 \times 10 \times 128$	$5 \times 5 \times 128$
Reshaping	-	-	-	$5 \times 5 \times 128$	3200
		s network			
MLP	-	20	tanh	3200	20
		σ network			
MLP	-	20	tanh	20	20
		μ network			
MLP	-	20	tanh	20	20
		MLP decoder network			
MLP	-	3200	tanh	20	3200
		Decoder			
Reshaping	-	-	-	3200	$5 \times 5 \times 128$
Conv2D	3×3	64	tanh	$5 \times 5 \times 128$	$5 \times 5 \times 64$
UpSampling	$2\times$, bi-cubic	-	-	$5 \times 5 \times 64$	$10 \times 10 \times 64$
Conv2D	3×3	32	tanh	$10 \times 10 \times 64$	$10 \times 10 \times 32$
UpSampling	$2\times$, bi-cubic	-	-	$10 \times 10 \times 32$	$20 \times 20 \times 32$
Conv2D	3×3	16	tanh	$20 \times 20 \times 32$	$20 \times 20 \times 16$
UpSampling	$2\times$, bi-cubic	-	-	$20 \times 20 \times 32$	$40 \times 40 \times 16$
Conv2D	3×3	3	tanh	$40 \times 40 \times 16$	$40 \times 40 \times 3$
UpSampling	$2\times$, bi-cubic	-	-	$40 \times 40 \times 3$	$80 \times 80 \times 3$

B Neural network architecture for DeepONet

Table 2: The DeepONet architecture

Trunk				
#Layers	Width	Activation	Input size	Output size
7	100	sin	1	4000
Branch				
#Layers	Width	Activation	Input size	Output size
7	100	sin	20	4000
Encoder Trunk				
#Layers	Width	Activation	Input size	Output size
1	100	sin	1	100
Encoder Branch				
#Layers	Width	Activation	Input size	Output size
1	100	sin	20	100

C Averaged error per physical variable in relative L^2-norm over all samples of the testing dataset

	u	w	T
Mean (%)	4.27	6.93	5.26
Std (%)	2.53	4.73	2.95