Direct data-driven forecast of local turbulent heat flux in Rayleigh–Bénard convection

Cite as: Phys. Fluids **34**, 045106 (2022); https://doi.org/10.1063/5.0087977
Submitted: 11 February 2022 • Accepted: 14 March 2022 • Published Online: 05 April 2022

Sandeep Pandey, Philipp Teutsch, Patrick Mäder, et al.

Direct data-driven forecast of local turbulent heat flux in Rayleigh–Bénard convection

Cite as: Phys. Fluids **34**, 045106 (2022); doi: 10.1063/5.0087977
Submitted: 11 February 2022 · Accepted: 14 March 2022 ·
Published Online: 5 April 2022

Sandeep Pandey,[1] Philipp Teutsch,[2] Patrick Mäder,[2,3] and Jörg Schumacher[1,4,a)]

AFFILIATIONS

[1]Institute of Thermodynamics and Fluid Mechanics, Technische Universität Ilmenau, D-98684 Ilmenau, Germany
[2]Institute for Practical Computer Science and Media Informatics, Technische Universität Ilmenau, D-98684 Ilmenau, Germany
[3]Faculty of Biological Sciences, Friedrich-Schiller-Universität Jena, D-07745 Jena, Germany
[4]Tandon School of Engineering, New York University, New York, New York 11201, USA

[a)]Author to whom correspondence should be addressed: joerg.schumacher@tu-ilmenau.de

ABSTRACT

A combined convolutional autoencoder–recurrent neural network machine learning model is presented to directly analyze and forecast the dynamics and low-order statistics of the local convective heat flux field in a two-dimensional turbulent Rayleigh–Bénard convection flow at Prandtl number $\mathrm{Pr} = 7$ and Rayleigh number $\mathrm{Ra} = 10^7$. Two recurrent neural networks are applied for the temporal advancement of turbulent heat transfer data in the reduced latent data space, an echo state network, and a recurrent gated unit. Thereby, our work exploits the modular combination of three different machine learning algorithms to build a fully data-driven and reduced model for the dynamics of the turbulent heat transfer in a complex thermally driven flow. The convolutional autoencoder with 12 hidden layers is able to reduce the dimensionality of the turbulence data to about 0.2% of their original size. Our results indicate a fairly good accuracy in the first- and second-order statistics of the convective heat flux. The algorithm is also able to reproduce the intermittent plume-mixing dynamics at the upper edges of the thermal boundary layers with some deviations. The same holds for the probability density function of the local convective heat flux with differences in the far tails. Furthermore, we demonstrate the noise resilience of the framework. This suggests that the present model might be applicable as a reduced dynamical model that delivers transport fluxes and their variations to coarse grids of larger-scale computational models, such as global circulation models for atmosphere and ocean.

I. INTRODUCTION

Turbulent thermal convection processes form one fundamental class of flows that are found in numerous natural and technological applications ranging from astrophysical scales in stellar interiors to sub-meter lengths in heat exchangers.[1–4] The fundamental physical question in these flows is the one on the local and global mechanisms of turbulent heat transfer that is typically significantly enhanced by the turbulent fluid motion in comparison to a purely diffusive transport in a quiescent medium. In its simplest configuration, a thermal convection flow consists of a fluid layer that is enclosed by two impermeable parallel plates at distance H, known as the Rayleigh–Bénard convection (RBC) case. The bottom plate is uniformly heated at a constant temperature $T = T_0 + \Delta T$, and the top plate is cooled at $T = T_0$.[5] For temperature differences $\Delta T > 0$ being large enough, the buoyancy-triggered fluid motion is turbulent. Convective turbulence is sustained by characteristic coherent structures that are denoted as thermal plumes. These unstable fragments of the thermal boundary layer permanently rise from the bottom or fall from the top into the bulk region of the convection layer and thus inject kinetic energy into the flow. Thermal plumes are also the local building blocks of the global heat transfer; their morphology has been studied in several experimental[6,7] and numerical studies.[8,9] They are connected with the local convective heat flux, which is given by

$$j_{\mathrm{conv}}(\boldsymbol{x}, t) = u_z(\boldsymbol{x}, t)\theta(\boldsymbol{x}, t), \tag{1}$$

with

$$\theta(\boldsymbol{x}, t) = T(\boldsymbol{x}, t) - \langle T \rangle_{A,t}. \tag{2}$$

Here, u_z is the vertical velocity component and θ is the deviation of the total temperature field T from the mean $\langle T \rangle_{A,t}$, where $\langle \cdot \rangle_{A,t}$ represents a combined average with respect to simulation domain area content A and time t. The analysis of this flux requires the joint solution of the coupled Boussinesq equations for the velocity and temperature fields. Here, we want to model the dynamics of this central transport quantity and its statistical properties directly by recurrent neural networks (RNNs) without solving the underlying nonlinear equations for velocity and temperature fields. This results in a significant simplification and data reduction and sets the major motivation for the present work.

Machine learning (ML) methods have caused a change of paradigms to analyze, model, and control turbulent flows.[10–16] This evolution is driven by the growing technological capabilities of numerical and laboratory experiments to generate high-dimensional, highly resolved data records at increasing Reynolds or Rayleigh number that can reproduce many aspects of fully developed turbulent flows in great detail. Flow features, such as the thermal plumes in the present case, are then to be classified, dynamically modeled, or connected to statistical moments for parametrizations and other reduced descriptions. To illustrate the typically resulting demands of a data analysis better, let us consider a concrete example of a three-dimensional direct numerical simulation (DNS) of a turbulent thermal convection flow. A layer of height H with an aspect ratio of $60H : 60H : H$ at a Rayleigh number $Ra \sim 10^8$ is resolved with about 6×10^9 spectral collocation points on an unstructured spectral element mesh for each of the involved fields, such as the three velocity components, temperature, and pressure.[17] It amounts to 181 GB of raw data per snapshot. The estimate does not incorporate the temporal dynamics that is typically stored as a sequence of such highly resolved snapshots. This underlines clearly the necessity to process and reduce data in completely new ways to uncover the main physical processes, such as the characteristic structures that form the backbone of the turbulent heat transfer.

Reduced-order models (ROMs) are derived to approximate the dynamics of the most energetic degrees of freedom or the large-scale flow and predict low-order turbulence statistics such as mean or fluctuation profiles. Most of these models are data-driven and can be generated in several ways, e.g., by Proper Orthogonal Decomposition (POD),[18,19] Dynamic Mode Decomposition,[20] nonlinear Laplacian spectral analysis,[21] or expansions in modes and eigenfunctions of the Koopman operator[22,23] to mention only a few. Particularly, the POD is still a popular workhorse for projection-based reduced models[24–27] and has been combined more recently also with ML algorithms.[28–31] With the increase in the vigor of turbulence (which is in line with an increase in Reynolds or Rayleigh number), the number of necessary POD modes in a ROM grows quickly. As a consequence, limitations of these models are reached quickly even with efficient algorithms such as the snapshot method.[32] This circumstance calls for alternative ways to reduce simulation snapshots, being further motivation for our present work.

In the present work, we combine a convolutional autoencoder (CAE) with recurrent neural networks (RNNs) to obtain a ML-based equation-free dynamical model for the convective heat flux field $j_{\mathrm{conv}}(\boldsymbol{x}, t)$. The convolutional encoder reduces the high-dimensional simulation snapshots of the convective heat flux to a low-dimensional feature space. In this latent space, the RNNs are trained and then run autonomously to advance the dynamics of $j_{\mathrm{conv}}(\boldsymbol{x}, t)$ with respect to time. A subsequent convolutional decoder transforms the resulting latent space data back into high-dimensional data snapshots of the flux. The choice of the hyperparameters of the RNNs will be explained more detailed further below. We discuss two CAE-RNN architectures that will predict the low-order statistics, such as the mean and fluctuation profiles of the convective heat flux, very well and are even able to reproduce the probability density function (PDF) of $j_{\mathrm{conv}}(\boldsymbol{x}, t)$. The two chosen RNN architectures are as follows:

- Echo state networks (ESNs) belong to the class of reservoir computing models[33] and have been used to describe the nonlinear dynamics of the Lorenz 63 or Lorenz 96 models, chaotic acoustic models, and two-dimensional convection without and with phase changes.[34–38] In the latter cases, the ESN was combined with a data reduction by POD,[37,38] such that the temporal dynamics of the POD expansion coefficients was modeled.
- Gated Recurrent Units (GRUs) with a specific sequence to sequence (seq2seq) architecture form the second RNN architecture that is studied in the present work. More precisely, we apply an encoder–decoder GRU[39] that was originally designed for natural language processing, but is now also applied to predict continuous time series data.[40,41]

Here, we thus substitute the data reduction and expansion that was formerly done by means of a snapshot POD[37,38] by a convolutional encoder/decoder network, which will be better suited for higher-dimensional simulation data. Different from previous studies, we feed the derived field $j_{\mathrm{conv}}(\boldsymbol{x}, t)$ directly into the ML algorithm. Throughout this work, we will remain in the two-dimensional Rayleigh–Bénard convection setup to demonstrate our concepts. For example in Ref. 37, we were able to compress our data by 92% while losing 17% of the turbulent variance. This is improved now even further to a compression by 99.7% while losing only about 5% of the variance.

A similar approach, to the present one, has been successfully taken by Gonzalez and Balajewicz, who combined a long short-term memory network with a CAE to study the one-dimensional viscous Burgers equation, interacting point vortices in a plane, and a two-dimensional lid-driven cavity flow.[42] Furthermore, RNN-based encoder–decoder architectures are also used as a comparison model for ESNs in similar applications.[43–46] Encoders and decoders have been successfully employed for de-noising data,[47,48] anomaly detection,[49] and to dimensionality reduction[50,51] in many other fields. Encoder and decoder networks can incorporate a variety of nonlinear activation functions, thus taking advantage of higher-order representations in connection with a deep network architecture. Particularly for multi-dimensional datasets, such architectures can reduce the training effort due to its parameter sharing and sparse connectivity.[52] For example, the spatiotemporal dynamics of fluid flows past cylinders and airfoils[53–55] or for turbulent channel flows[56] were successfully predicted and analyzed with CAEs. In the field of turbulent convection, neural network algorithms have been used to fit the global laws of turbulent heat and momentum transfer better.[57]

Finally, we want to mention the connection between the linear Koopman operator, which describes the evolution of observables of a nonlinear dynamical systems, and deep learning methods. Eivazi *et al.* combined Koopman methods with RNNs.[58] Lusch *et al.* used convolutional neural networks to discover representations of the eigenfunctions of the Koopman operator from data for linear higher-dimensional embeddings of the nonlinear dynamics.[59] Otto and Rowley combined an autoencoder and a linear recurrent dynamics in the latent space to learn low-dimensional subspaces of observables that are invariant with respect to the Koopman operator.[60]

The outline of the manuscript is as follows. Section II presents the Boussinesq equations of turbulent convection and the numerical simulation model in brief that generate the database. Section III describes in detail the building blocks of our CAE-ESN and CAE-GRU networks including the hyperparameter tuning. Section IV discusses the training procedures and the results of both model runs with test data. We summarize the work and give a brief outlook at the end in Sec. V. Further specific details of the architecture of the CAE and the training are summarized in Appendixes A and B, respectively.

II. SIMULATION DATA OF TWO-DIMENSIONAL TURBULENT CONVECTION

The turbulent convection data are generated by a DNS using nek5000 spectral element solver[61] in the two-dimensional case. The Boussinesq equations (3)–(5), which couple the velocity components (u_x, u_z) and temperature T, are solved in a closed rectangular cell of an aspect ratio $L/H = 6$. They are given in dimensionless form by

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{3}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \sqrt{\frac{\mathrm{Pr}}{\mathrm{Ra}}}\frac{\partial^2 u_i}{\partial x_j^2} + T\delta_{i,z}, \tag{4}$$

$$\frac{\partial T}{\partial t} + u_j \frac{\partial T}{\partial x_j} = \frac{1}{\sqrt{\mathrm{RaPr}}}\frac{\partial^2 T}{\partial x_j^2}. \tag{5}$$

The pressure field is denoted by p and $i, j \in \{x, z\}$. The horizontal coordinate is given by x and the vertical one by z. The dimensionless Rayleigh number Ra is a measure of the vigor of convective turbulence, set to $\mathrm{Ra} = 10^7$ here. The dimensionless Prandtl number Pr, which is the ratio of momentum to thermal diffusion, was fixed to $\mathrm{Pr} = 7$, as for thermal convection in water. The parameters are given by

$$\mathrm{Ra} = \frac{g\alpha\Delta T H^3}{\nu\kappa} \quad \text{and} \quad \mathrm{Pr} = \frac{\nu}{\kappa}. \tag{6}$$

Here, g is the acceleration due to the gravity, α the thermal expansion coefficient, ν the kinematic viscosity, κ the thermal diffusivity, and ΔT the temperature difference between the bottom and top plates. All equations are made dimensionless by the cell height H, free fall velocity $U_f = \sqrt{g\alpha\Delta T H}$, and $\Delta T > 0$. Note that $0 \le T \le 1$ and thus $-0.5 \le \theta \le 0.5$.

The simulation domain is covered by 48×16 spectral elements. On each element, the four fields are expanded in polynomials of order 11 in each space dimension. For the machine learning analysis, these fields are obtained by a spectral interpolation on a uniform grid consisting of $N_x \times N_z = 320 \times 60$ points. As a result, we have 2400 snapshots and thus a total of $2400 \times 320 \times 60$ data points. Snapshots were sampled at every 0.125 free fall time units, H/U_f. This data generation process using the DNS took approximately 123 CPU-hours including the initialization with random perturbations. More details of the simulation and the boundary conditions can be found in Ref. 37.

Figure 1 shows snapshots of the two-dimensional convection fields. We display the temperature (left column) and vertical velocity component (middle column) that are combined to the derived convective heat flux field j_{conv} (right column). The latter field is characterized by sharp ridges that suggest a strongly localized convective heat flux. Exactly these ridges will be extracted as the dominant features by the CAE. Following from definitions (1) and (2), a positive local convective heat flux j_{conv} is present for both, $u_z > 0$ and $\theta > 0$ as well as $u_z < 0$ and $\theta < 0$; compare panels (a) and (c). We note that the Nusselt number, a global dimensionless measure of the turbulent heat transfer, follows $\mathrm{Nu} = 1 + \sqrt{\mathrm{RaPr}}\langle j_{\mathrm{conv}}\rangle_{A,t} \ge 1$. Again, the symbol $\langle\cdot\rangle_{A,t}$ stands for a combined average with respect to time t and area $A = L \times H$.

III. BUILDING BLOCKS OF THE END-TO-END PIPELINE

Figure 2 illustrates the building blocks and workflow in our end-to-end pipeline. It consists of a convolutional encoder for compressing the two-dimensional spatiotemporal data, a recurrent neural network (RNN) to forecast dynamics in the reduced order space (also known as latent space), and an accompanying convolutional decoder decompressing the two-dimensional convective heat flux field from the forecasted reduced order data. We study two types of RNNs for forecasting, an ESN and a GRU-based network. Once trained, the presented purely data-driven approach can process simulation snapshots on the fly. In contrast, the previously applied POD snapshot analysis[37,38] requires knowledge of the entire training dataset *ab initio* to extract the POD modes from the collection of simulation snapshots and thus to obtain subsequently the time series of the POD expansion coefficients as the RNN input. The following subsections discuss the individual building blocks of our CAE–RNN approach in detail.

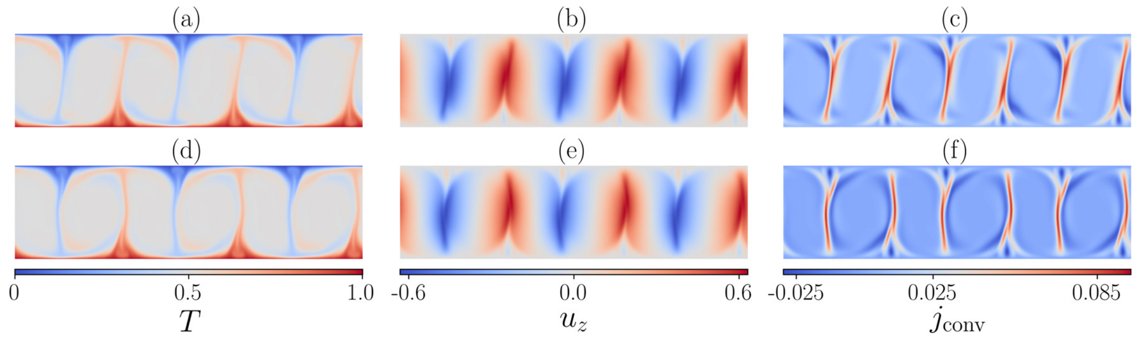


FIG. 1. Contours of the two-dimensional convection fields for two time instances. (a) and (d) Total temperature T, see also Eq. (2). (b) and (e) Vertical velocity component u_z. (c) and (f) Resulting turbulent convective heat flux j_{conv} as given by Eq. (1).

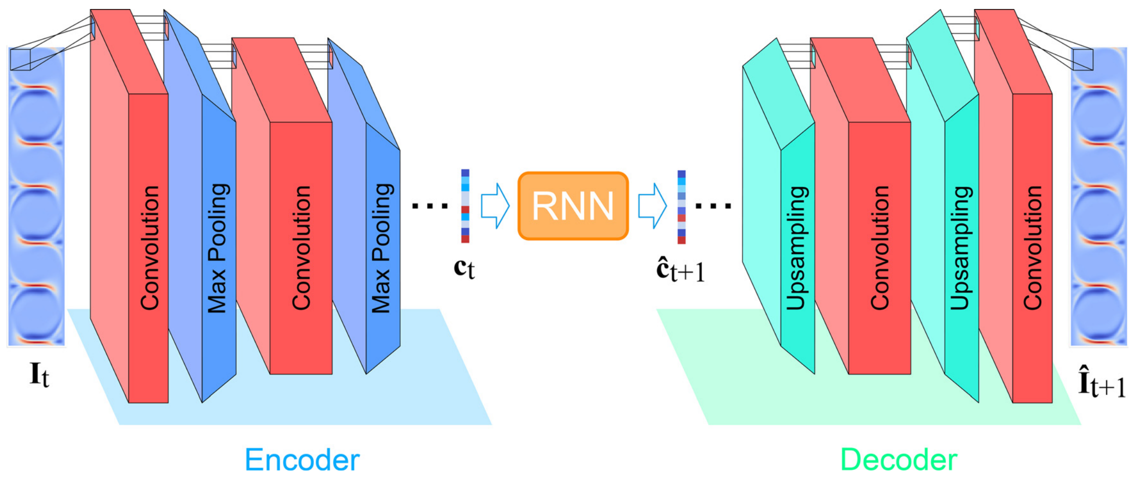


FIG. 2. Illustration of proposed end-to-end pipeline for the forecasting of convective heat flux dynamics. The convolutional autoencoder receives the high-dimensional field data of the direct numerical simulations at time t and has been trained for a task-specific order reduction. A trained RNN consumes the reduced order dynamics compressed by the encoder and forecasts future dynamics in the reduced order space at time $t+1$. The compressed dynamics is input for the decoder that accompanies the CAE to predict a fully resolved flow field at time $t+1$. Note that the RNN can also be run in latent space for several time steps $p>1$ from time t to $t+p$.

A. Generation of compressed snapshot representation

An autoencoder is a machine learning model consisting of an encoder and a decoder module, see Fig. 2. The purpose of the encoder is to compress its input into a trained latent space. The accompanying decoder takes data in this latent representation and reconstructs its representation in the original input domain. From a learning perspective, autoencoders are self-supervised, i.e., the network's input is also used as expected output and no additional labeling effort for a training dataset is required. That is, in theory an autoencoder encodes input data $\boldsymbol{I}$ into $\boldsymbol{c} = \text{encode}(\boldsymbol{I})$ where $\boldsymbol{c} \in \mathbb{R}^l$. It decodes $\boldsymbol{c}$ back into $\hat{\boldsymbol{I}} = \text{decode}(\boldsymbol{c})$ subsequently. In practice, however, there will be deviations, such that $\hat{\boldsymbol{I}} \approx \boldsymbol{I}$ (cf. Fig. 2). An L_2 norm is used as an objective function, and the training uses a gradient descent method, e.g., with adaptive momentum (Adam).[62]

A CAE utilizing convolutional layers rather than fully connected layers is specifically suitable for coping with the complexity of high-dimensional input data. A typical convolutional layer combines a convolution operation on the input data with trainable kernels and an activation function to induce a nonlinearity. Consider, three-dimensional input data $\boldsymbol{I} \in \mathbb{R}^{C_{\text{in}} \times N_x \times N_z}$ with C_{in} the number of input channels. Here, $C_{\text{in}} = 1$ (as we load the convective heat flux field only into the network) and thus $\boldsymbol{I} \in \mathbb{R}^{N_x \times N_z}$. This input is convoluted with a kernel k_m as shown in Eq. (7), where $M = [1, C_{\text{out}}] \subset \mathbb{N}$ with the number of output channels C_{out}. Furthermore, b_m is the bias term of kernel k_m, and ψ a nonlinear activation function. Zero-padding is typically applied to a layer's input to prevent information loss at the edges of the input data,

$$\text{Conv}(m, \boldsymbol{I}) = \psi\left(b_m + \sum_{i=1}^{C_{\text{in}}} k_m * \boldsymbol{I}_i\right) \quad \text{for } m \in M. \tag{7}$$

In the encoder, multiple convolutional layers are typically followed by pooling layers. A window of configurable size and sliding is used with a configurable step across the input. Thereby, the input per window is aggregated into a single output element with a selectable aggregation function. This aggregation function of the common max-pooling layer retains only the maximum element per window. The pooling layer's step size is typically chosen such that it reduces the dimensionality of the input $\boldsymbol{I}$. The eventual output of the encoder consists of data in the latent space $\mathbb{R}^l$ where $l \ll N_x \times N_z$. The CAE's decoder follows a mostly analogous design. The main difference is the use of upsampling instead of max-pooling layers to decode the data back to input domain size. An upsampling step, also known as unpooling, doubles the data dimension. Nearest-neighbour linear interpolation is therefore used in the present work. The effective processing of complex turbulence data requires a deep architecture consisting of multiple convolutional and accompanying upsampling layers in order to obtain a representation without substantial loss of information (cf. Table IV in Appendix A). In the end-to-end scenario and after training all networks, we utilize the CAE's encoder to derive a compressed form $\boldsymbol{c}_t$ from a given snapshot $\boldsymbol{I}_t$. The compressed snapshot becomes input to a subsequent RNN that forecasts the next reduced representation $\hat{\boldsymbol{c}}_{t+1}$, which is then decoded back into the input domain by the decoder unit of the CAE.

B. Forecast of compressed snapshots in the latent space

We study two types of RNNs for forecasting the dynamics of compressed snapshots in the latent space, echo state networks (ESNs) and gated recurrent units (GRUs).

1. Echo state network

The ESN has previously shown great potential in modeling sequential turbulent flow data.[35,63] ESNs consist of an input layer; a sparsely occupied and randomly parametrized network of recurrent and direct connections, the reservoir; and an output layer, see Fig. 3 (top).

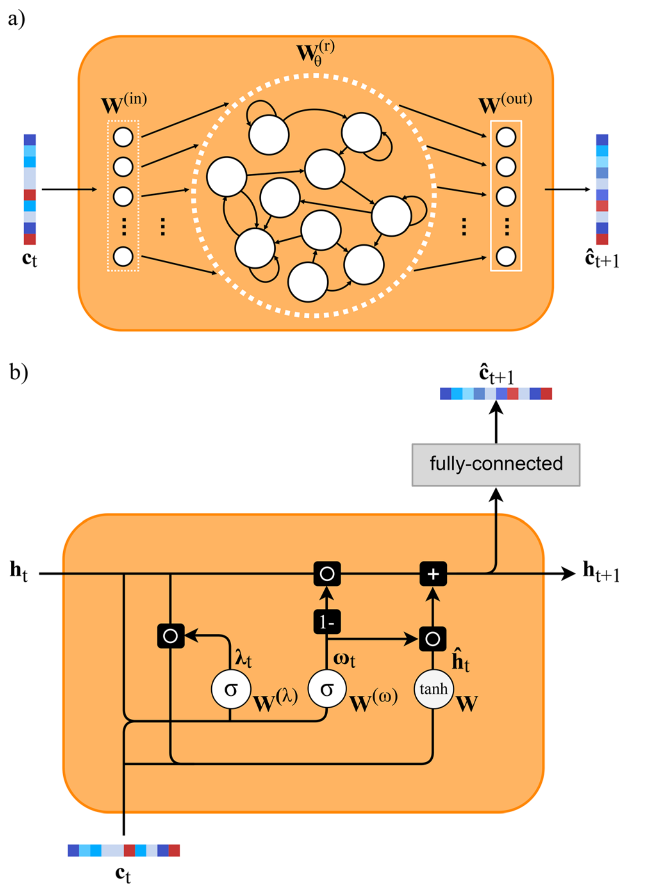


FIG. 3. (a) Echo state network (ESN) architecture consisting of an input layer, the reservoir, and an output layer. (b) Gated recurrent unit (GRU) cell consisting of a reset gate and an update gate.

The reservoir is represented as an $N \times N$ adjacency matrix $\mathbf{W}_\eta^{(r)}$ whose initialization depends on a vector of hyperparameters η (cf. Sec. IV) and is used to encode the ESN's input into a hidden representation, the reservoir state, that accumulates information of previous inputs. N is also called the reservoir dimension and chosen typically much larger than the dimension of the latent space, $N \gg l$. This reservoir is furthermore updated per time step with new input. More specifically, at each time step t the ESN's input $\boldsymbol{c}_t$ influences the computation of an updated reservoir state $\boldsymbol{r}_t \in \mathbb{R}^N$. This step is given by

$$\boldsymbol{r}_t = (1-\alpha)\boldsymbol{r}_{t-1} + \alpha \tanh\left(\mathbf{W}^{(\text{in})}\boldsymbol{c}_t + \mathbf{W}_\eta^{(r)}\boldsymbol{r}_{t-1}\right). \tag{8}$$

The parameter α denotes a leakage rate determining the blending of previous state and current input. The random matrices $\mathbf{W}^{(\text{in})}$ and $\mathbf{W}_\eta^{(r)}$ are initialized at the beginning of the training and remain unchanged thereafter. The ESN's output at time t is obtained by

$$\hat{\boldsymbol{c}}_{t+1} = \mathbf{W}^{(\text{out})}\boldsymbol{r}_t. \tag{9}$$

In the ESN case, the final output layer is trained only; no backpropagation in several epochs is required as for convolutional networks. This implies that the components of the output matrix $\mathbf{W}^{(\text{out})}$ have to be optimized to give a minimal cost function that quantifies the difference between training data and ESN output. The cost function C is given by

$$C[\mathbf{W}^{(\text{out})}] = \frac{1}{n_{\text{tr}}}\sum_{t=1}^{n_{\text{tr}}}\left(\mathbf{W}^{(\text{out})}\boldsymbol{r}_t - \boldsymbol{c}_t\right)^2 + \beta\sum_{i=1}^{l}||w_i^{\text{out}}||_2^2, \tag{10}$$

and has to be minimized corresponding to

$$\mathbf{W}_*^{(\text{out})} = \arg\min C\left(\mathbf{W}^{(\text{out})}\right). \tag{11}$$

Here, w_i^{out} denotes the ith row of $\mathbf{W}^{(\text{out})}$ and $||\cdot||_2$ the L_2 norm. The number of training samples is n_{tr}. Equations (10) and (11) are known as ridge regression with the parameter β, also known as the Tikhonov regularization parameter. The last term suppresses large values of the rows of the output matrix. This regression problem is solved by

$$\mathbf{W}_*^{(\text{out})} = \mathbf{Y}\mathbf{S}^{\text{T}}\left(\mathbf{S}\mathbf{S}^{\text{T}} + \beta\,\mathbf{Id}\right)^{-1}, \tag{12}$$

where $(\cdot)^{\text{T}}$ denotes the transposed and **Id** the identity matrix. Here, **Y** and **S** are matrices where the nth column is the target output $\boldsymbol{c}_t$ and the reservoir output $\hat{\boldsymbol{c}}_t$, respectively.

The hyperparameters of the ESN are the reservoir size N, the node density D, which is the percentage of active nodes, the spectral radius of the reservoir $\rho(\mathbf{W}_\eta^{(r)})$, the leakage rate α, and the Tikhonov regularization parameter β in the cost function C. Thus, the vector of hyperparameters is $\boldsymbol{\eta} = (N, D, \rho, \alpha, \beta)$. This training is inherently fast compared to other types of RNN.[33,35] After successful training and hyper-parameter optimization, the ESN runs as an autonomous dynamical system, i.e., a forecasted compressed snapshot $\boldsymbol{c}_{t+1}$ can be used as the next ESN input to forecast $\boldsymbol{c}_{t+2}$ and so on. This is also known as the closed-loop scenario.

2. Gated recurrent unit

As already mentioned in Sec. I, we also study a gated recurrent unit (GRU) in an encoder–decoder architecture as an RNN, used for complex sequence analysis and forecasting problems. GRU is an advanced RNN cell that uses gates to control which information becomes part of the maintained cell state and which previously acquired information can be forgotten. Due to this mechanism, the GRU effectively mitigates vanishing and exploding gradient problems typically faced when training RNNs with long training sequences. Figure 3 (bottom) shows the interplay of the components of the GRU.

The operation of the reset gate and the update gate is denoted as

$$\boldsymbol{\lambda}_t = \sigma(\boldsymbol{W}^{(\lambda)} \cdot [\boldsymbol{h}_t, \boldsymbol{c}_t]) \tag{13}$$

and

$$\boldsymbol{\omega}_t = \sigma(\boldsymbol{W}^{(\omega)} \cdot [\boldsymbol{h}_t, \boldsymbol{c}_t]), \tag{14}$$

where $\boldsymbol{\lambda}_t$ denotes the reset gate vector and $\boldsymbol{\omega}_t$ the update gate vector at time step t; $\boldsymbol{W}^{(\lambda)}$ and $\boldsymbol{W}^{(\omega)}$ are the corresponding weight matrices applied to a vector formed by concatenating the input vector $\boldsymbol{c}_t$ at time step t with the hidden state vector $\boldsymbol{h}_t$. The sigmoid activation function σ ensures an output range between 0 and 1. The output is used in an element-wise vector multiplication to determine how much of other vector element's value to preserve. More specifically, $\boldsymbol{\lambda}_t$ is

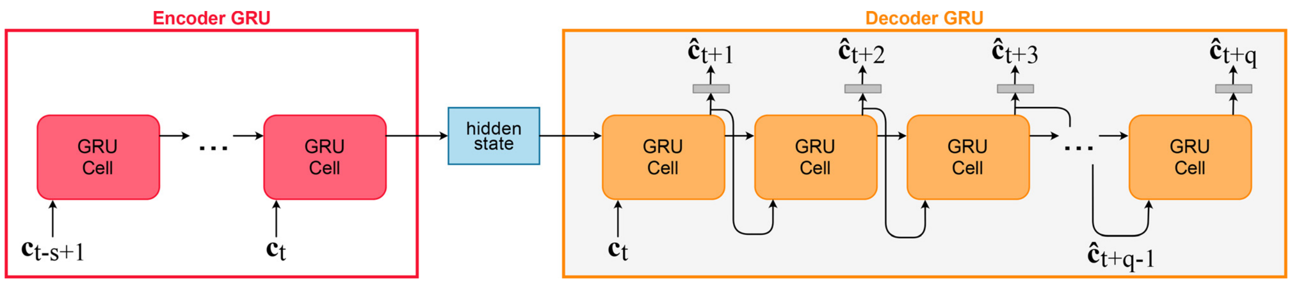


FIG. 4. Forward data flow of an encoder–decoder network based on a gated recurrent unit.

element-wise multiplied with $\boldsymbol{h}_t$ to "reset" individual values of the old state when computing the updated intermediate state $\hat{\boldsymbol{h}}_t$ as

$$\hat{\boldsymbol{h}}_t = \tanh(\boldsymbol{W} \cdot [\boldsymbol{\lambda}_t \odot \boldsymbol{h}_t, \boldsymbol{c}_t]), \tag{15}$$

where $\boldsymbol{W}$ is an additional weight matrix and tanh is used as activation function. The element-wise Hadamard product is denoted by $\odot$ in (15). Analogously, $\boldsymbol{\omega}_t$ is element-wise multiplied with $\boldsymbol{h}_t$ and $(1 - \boldsymbol{\omega}_t)$ with $\hat{\boldsymbol{h}}_t$ to add current input information to the updated cell state $\boldsymbol{h}_{t+1}$, which is also the cell's output, given by

$$\boldsymbol{h}_{t+1} = (1 - \boldsymbol{\omega}_t) \odot \boldsymbol{h}_t + \boldsymbol{\omega}_t \odot \boldsymbol{h}_t. \tag{16}$$

The output is then fed forward through a final fully connected layer with a linear activation to forecast the next compressed snapshot $\hat{\boldsymbol{c}}_{t+1}$.

We organize two GRU cells in an encoder–decoder architecture[64,65] as detailed in Fig. 4. The encoder processes a sequence of compressed input snapshots while building the hidden state that represents a latent representation thereof. This latent representation is then passed to the decoder which uses the encoded accumulated information up to the previous time step to forecast the next compressed snapshot of the sequence, $\hat{\boldsymbol{c}}_{t+1}$. Following an initial externally triggered step, the decoder progresses auto-regressively; it consumes its output of the previous iteration $\hat{\boldsymbol{c}}_{t+1}$ as an input for the computation of an updated latent representation that is used to forecast the next output $\hat{\boldsymbol{c}}_{t+2}$.

IV. RESULTS AND DISCUSSION

Before we turn to the training, validation, and test phases, we list a few more details of the DNS data record. In total, we used 2400 snapshots divided into three subsets. The first subset consisted of 1000 snapshots exclusively used for training, thus, called training set. The second subset, called validation set, consisted of 500 snapshots used to evaluate an unbiased estimation of training while performing hyperparameter tuning of a given model in the runtime. This validation is done right after the training with each of the chosen hyperparameter sets in order to find the optimal $\boldsymbol{\eta}$ for the subsequent test phase; it helps to mitigate over- and under-fitting. The third subset contained the remaining 900 snapshots used as an independent, unseen, and non-trained dataset, known as test dataset. The final model is evaluated for this test data, providing an unbiased estimation on blind dataset. We trained CAE and GRU on NVIDIA GeForce GTX 1060 and RTX 2080TI GPUs, respectively, and the ESN on a CPU with 16 GB of memory.

A. Training of the convolutional autoencoder

We first developed a multi-layer CAE consisting of 12 layers for each, encoder and decoder (cf. Table IV in Appendix A). The decoder consists of an additional cropping layer to gain the original data dimension of 320×60. This final layer uses a sigmoid activation to ensure a normalized output. We optimized the network hyperparameters using a Bayesian optimization (BO). Table I shows the optimized parameters, the value search range per parameter, and the discovered optimum within this range. Eventually, we used convolutions with a kernel size of 5×5 giving them a larger receptive field. For the latent representation between the encoder and decoder, we found a size of 40 elements to best suit our application. The latent space will thus be 40-dimensional. All network weights were initialized following a Glorot uniform distribution.[66] More details of the BO procedure are found in Appendix B. After obtaining the optimized parameters, we trained the CAE and simultaneously validated the model against the validation dataset at the end of every epoch (cf. Fig. 5). The training has converged after about 30 epochs; it is however continued until it satisfies the early stopping criterion. In addition, we observe that both training and validation error decrease coherently, which indicates the robustness of the model on unseen validation data.

B. Training of the echo state network

After obtaining a trained CAE delivering compressed representations of input data, we proceeded with the training of the ESN for the prediction of the temporal evolution of the convection flow in the latent space. Details of the corresponding training procedure have been given already in Sec. III B 1. We employed the same dataset and the same splitting as discussed above. We used the mean squared error (MSE) between the predicted and ground truth modes as an objective function for ESN training. We also monitored the MSE between the

TABLE I. Optimized parameters obtained from the Bayesian optimization for the convolutional autoencoder. The optimization process was started with five random initial points and thereafter 25 iterations were used with a factor $\kappa = 1$, see Eq. (B3) in Appendix B.

Parameter	Search range	Optimized value
Kernel size	(1×1)–(5×5)	5×5
Latent vector size	$\{20k \mid k \in \{1, 2, \ldots, 3\}\}$	40
Learning rate	0.0001–0.001	0.000 58
Batch size	$\{8l \mid l \in \{1, 2, 3\}\}$	16

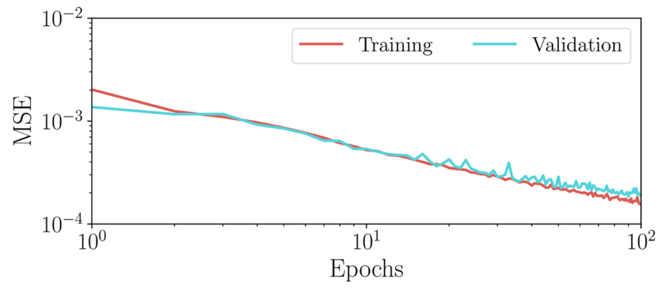


FIG. 5. The loss function in the form of a mean squared error (MSE) vs the number of training epochs in double-logarithmic plot.

predicted and the original turbulent convective heat flux as an additional metric for training success. This was realized by continuously feeding the predicted test modes to the decoder and gather them in ensembles. Again, we optimized the network's hyperparameter using a BO.

Figure 6 illustrates an example of the BO progress in three different iterations. In the example, we solely optimize the regularization parameter β while keeping the other hyperparameters constant. In these figures, the actual (unknown, black-box) objective function is shown as the red curve. One can observe the complex nature along with an incapability of a grid-search if the grid is too coarse to capture the optima. Here, we used the MSE as a cost function and maximized the negative of MSE.

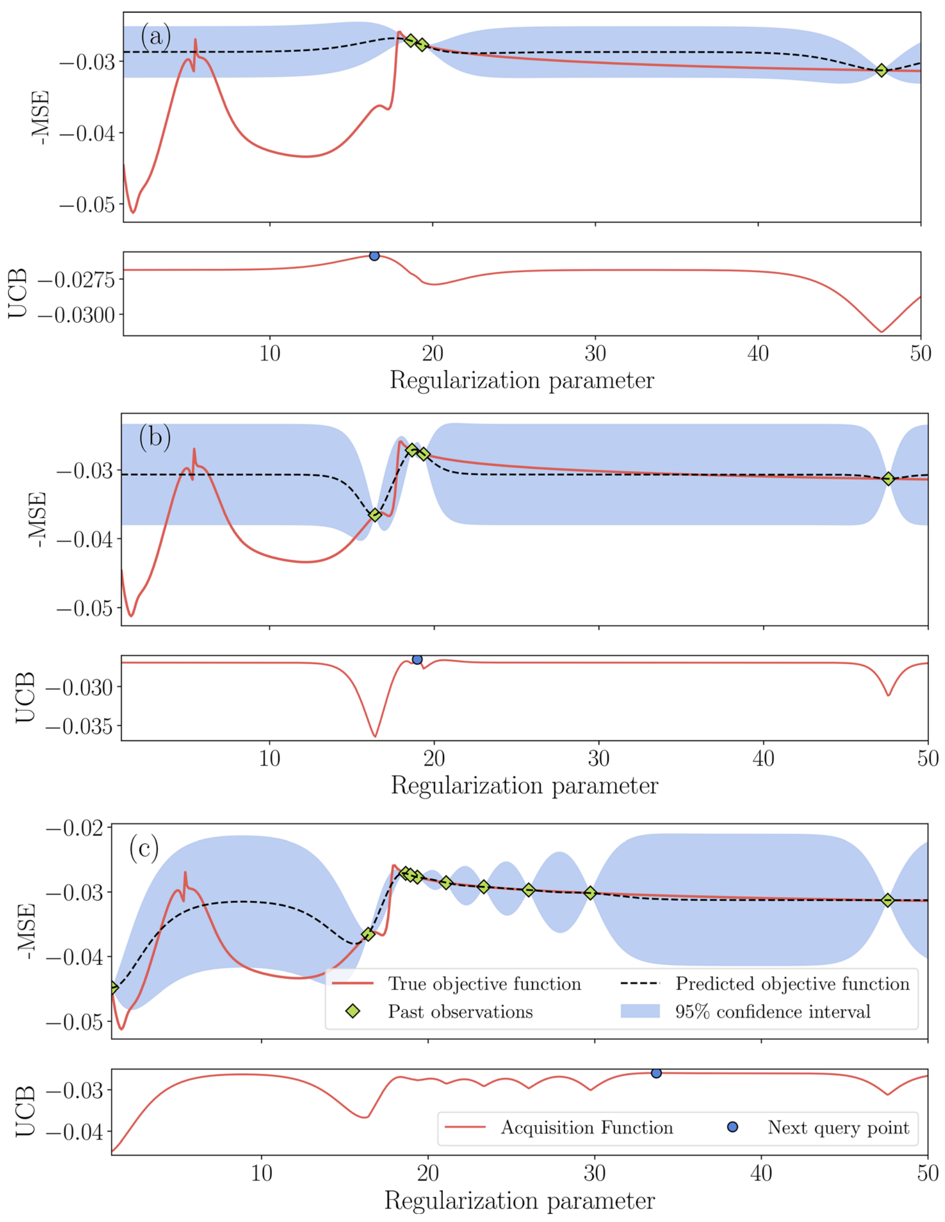


FIG. 6. Exemplary progress of the Bayesian optimization of the ESN. In panels (a)–(c), we plot the negative mean squared error (MSE) in the top together with the upper confidence bound in the corresponding bottom panel for the search of the optimal Tikhonov regularization parameter β. (a) Iteration No. 1 after initialization with two random points is shown. (b) Iteration No. 2 is shown. (c) Iteration No. 8 is shown. For this demonstrative case, we took $\kappa = 1$ (see Appendix B), and the following ESN hyperparameters: $\alpha = 0.96$, $\rho = 0.92$, $D = 0.2$, and $N = 100$ from the given ranges.

TABLE II. Optimized parameters obtained from BO for the ESN. Here, five points were randomly chosen to initialize the prior, 50 iterations were used for the BO and $\kappa = 1$.

Parameter	Search range	Optimized value
Reservoir size	100–5000	2992
Spectral radius	0.90–0.99	0.97
Reservoir density	0.05–0.20	0.09
Scaling	True, false	False
Leakage rate	0.5–0.9	0.50
Regularization parameter	0–600	4.89

We started the BO at two random points for β, which enable the calculation of the posterior distribution as shown in Fig. 6(a). The third observation is at $\beta = 18.6$ and the acquisition function in the form of an upper confidence bound (UCB) predicts the next query point at $\beta = 16.4$ because it becomes the area for exploitation as $\kappa = 1$. In Fig. 6(b), it can be seen that the uncertainty becomes zero for $\beta = 16.4$ (assuming no noise) and that the acquisition function suggests a next point, $\beta = 18.9$. This iteration proceeds until we reach an optimum or a predefined number of iterations. Figure 6(c) illustrates the status after eight iterations, showing that the model is still not converged, but continues to explore a region with higher uncertainty and eventually yielding an optimum. Here, the factor $\kappa = 1$ in the UCB is taken, which forces the algorithm to exploit regions with higher mean; see again Appendix B. In our BO for the ESN, we optimized the five hyperparameters that are summarized in vector $\boldsymbol{\eta}$ plus the scaling. They are known to have a significant effect on the performance of the network. Table II summarizes the optimized hyperparameters obtained after 50 BO iterations.

C. Training of the gated recurrent unit

We further went on to train the encoder–decoder GRU as an alternative to the ESN. Both will predict the temporal evolution of the convection flow in the latent space. Again, we searched for the optimal hyperparameters of the network in our given application scenario. More specifically, we optimized here the learning rate, the batch size, and the hidden state size via a grid-search (cf. Table III). Different from the ESN, the GRU is trained by means of a stochastic gradient descent (SGD) method, equally to the CAE. Therefore, we use snapshots from the training, validation and test set, respectively, to generate samples. Each sample consists of 50 input snapshots and 100 target snapshots. Within the training process, we dynamically adapt the learning rate once the training loss did not improve for a given number of epochs (known as patience), i.e., we reduce the learning rate. The reduction is performed by multiplying the current learning rate with a factor γ. We set the number of patience epochs to 20 and $\gamma = 0.6$. Finally, we applied an early stopping to determine the number of training epochs. Here, we choose a patience of 100 epochs to ensure the convergence of the model.

TABLE III. Hyperparameters used for the encoder–decoder GRU training.

Parameter	Search range	Optimized value
Initial learning rate	0.006, 0.003, 0.001, 0.0006, 0.0003, 0.0001, 0.000 06, 0.000 03, 0.000 01	0.001
Batch size	32, 64, 128, 256, 512	128
Hidden state size	128, 256, 384, 512, 1024	512

D. Convective heat flux fields and mean profiles

With two levels of machine learning models combined, both of which have reached low errors in training and validation, it is necessary now to assess their generalization ability by a comparison with unseen test data snapshots.

Test data samples for the RNNs are generated by feeding the corresponding ground truth DNS data into the CAE encoder unit, which generates lower-dimensional compressed data vectors of dimension $l = 40$ (cf. Sec. III A). For the GRU, a test sample consists of 950 compressed representations. From these representations, the encoder uses the first 50 as an input to build its hidden state; the decoder predicts the remaining 900 autoregressively. The ESN does not require this extra initialization of its hidden state since we use a model that is initialized already and can succeed from the training–validation sequence

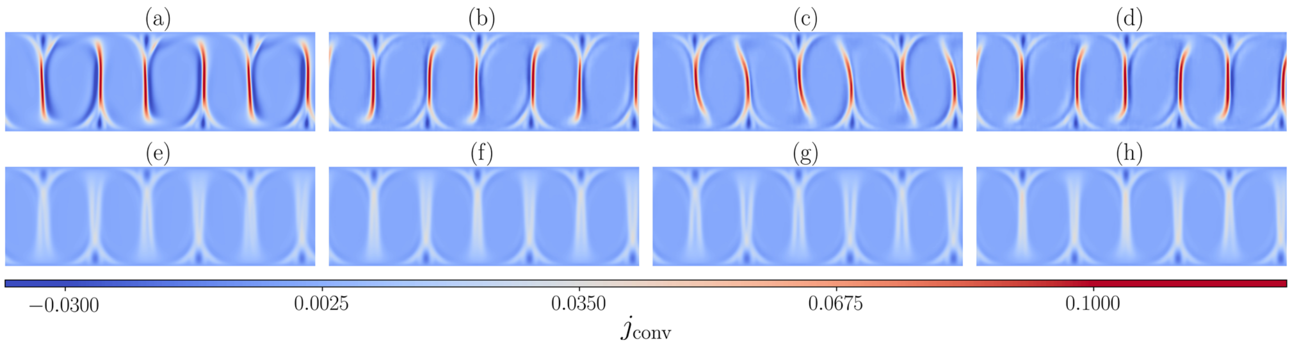


FIG. 7. Result from the blind test. (a)–(d) Instantaneous convective turbulent heat flux snapshots at output time step No. 51 out of 900 (which corresponds to $t = 6.25$ free fall time units). (e)–(h) Mean convective turbulent heat flux field averaged over all 900 snapshots. (a) and (e) DNS data that are the ground truth for the CAE. (b) and (f) Fields that have been processed by the CAE, i.e., encoded and subsequently decoded. Note that the encoded DNS data are considered as the ground truth for ESN/GRU since they operate in the latent space. (c) and (g) Prediction from the ESN with subsequent decoding. (d) and (h) Prediction from the GRU with subsequent decoding.

to predict 900 compressed representations. Thus, the test sample of the ESN consists of 900 latent vectors in total. Afterwards, the 900 predicted latent data vectors are passed through the decoder to reconstruct the corresponding convective heat flux fields. Thereby, the ESN and the GRU are evaluated as autoregressive prediction models. To evaluate the CAE itself, we feed 900 target snapshots into the encoder and then reconstruct them directly via the decoder, thereby analyzing the bare reconstruction error.

Figure 7 depicts the qualitative comparison between the DNS results, the reconstruction by the CAE, and the autonomous prediction by ESN and GRU. A good agreement can be observed especially for the high-magnitude plume regions and the general flow structure in the form of circulating convection rolls, refer to Figs. 7(a)–7(d). In the former regions, the convective heat flux is locally largest, caused by sinking colder fluid or rising hotter fluid. In a 3D convection case, these regions would form a dynamically evolving skeleton as shown and analyzed in Fonda *et al.*[67]

When looking at the instantaneous and time-averaged fields in Fig. 7, small deviations can be noticed due to the highly reduced dimensionality of the latent space with $l = 40$. This is an expected error for such kind of data compression. Nevertheless, one can conclude that the reconstructed fields agree qualitatively and even quantitatively fairly well with the ground truth. Unlike in cases, where a POD has been applied for data reduction,[37,38] the mean fields are not separated here; these fields were thus also dependent upon the ML predictions.

A deviation of the obtained mean profiles of the convective heat flux, $\langle j_{\mathrm{conv}} \rangle_{x,t}$, from the ground truth is seen in Fig. 8(a) away from the wall. This deviation is situated above the thermal boundary layer in the plume mixing zone for the high Prandtl number of Pr = 7 that is chosen here. In this region, the dynamics is characterized by bursting thermal plumes that detach randomly at different positions from the wall and get dispersed by the turbulence. Clearly, the reproduction of this intermittent time dynamics of the convective turbulent transport is most challenging for the ML algorithms.

Figure 8(b) shows fairly well overlapping profiles for the fluctuations of the convective heat flux for all three cases that remain close to the ground truth. In more detail, these profiles are obtained as a root mean square average of the field

$$j'_{\mathrm{conv}} = (u_z\theta)' = u_z\theta - \langle u_z\theta(z)\rangle_{x,t}. \tag{17}$$

We have also quantified the loss of information of the CAE-RNN application, which is based on the integrated convective flux, which is given by

$$\Phi = \int_A \langle (u_z\theta)'^2 \rangle_t \, dA. \tag{18}$$

The loss follows by

$$L_i = \frac{|\Phi_i - \Phi_{\mathrm{DNS}}|}{\Phi_{\mathrm{DNS}}}, \tag{19}$$

with $i = \{\mathrm{CAE}, \mathrm{ESN}, \mathrm{GRU}\}$. The results are $L_{\mathrm{CAE}} = 5.3\%$, $L_{\mathrm{ESN}} = 4.2\%$, and $L_{\mathrm{GRU}} = 6.7\%$. Here, L_{CAE} denotes the loss that results from encoding and decoding the ground truth sequence without any forecasting. L_{ESN} and L_{GRU} denote the loss between the original DNS and the decoding of latent representations that were previously forecasted by the ESN and the GRU, respectively.

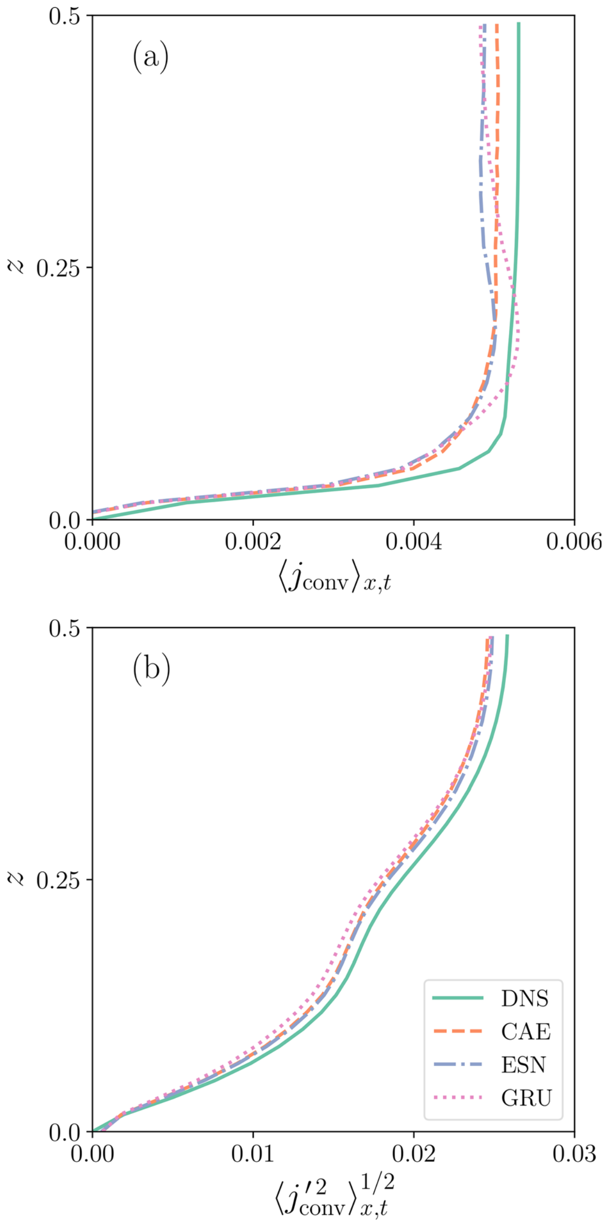


FIG. 8. Comparison of the mean convective turbulent heat flux and the corresponding fluctuation profiles over half the cell height. Due to the top-down symmetry in RBC, we took an additional mean over both halves of the layer. (a) Mean convective turbulent heat flux profile, and (b) Convective turbulent heat flux fluctuation profile. Linestyles in the legend hold for both panels.

Figure 9 illustrates the temporal evolution of the area-averaged values of the convective heat flux. All algorithms show a similar variation in time for the mean convective heat flux. However, we observe a quantitative mismatch, in particular for the ESN case. Qualitatively, the time variation of this average is picked up by all three algorithms fairly well.

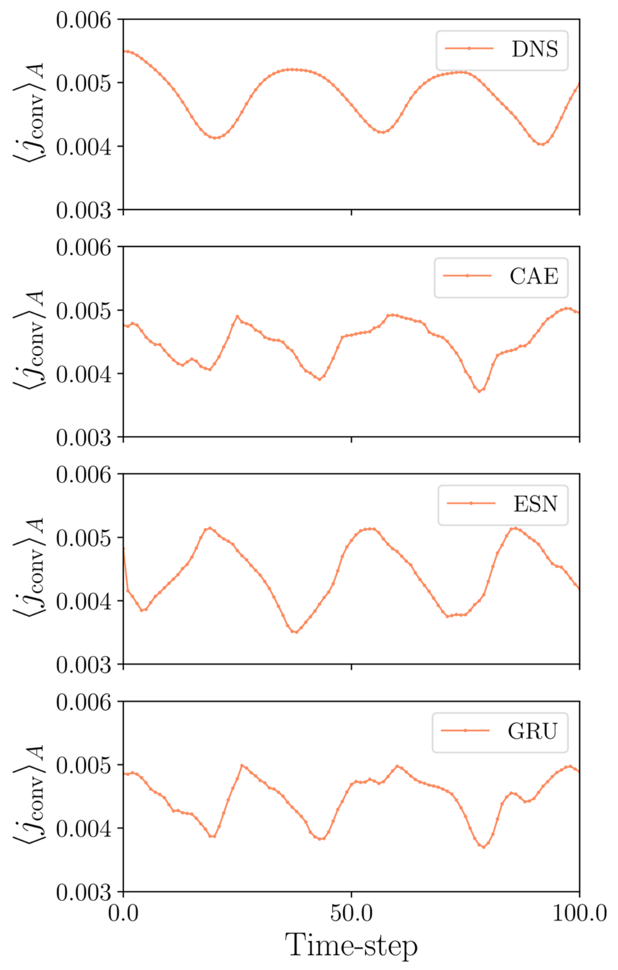


FIG. 9. Comparison of the temporal evolution of the area-averaged convective turbulent heat flux $\langle j_{\rm conv}\rangle_A$ over the first 100 time steps that correspond to a time interval of 12.5 free fall time units.

E. Probability density function of convective heat flux

Figure 10 illustrates a further comparison of the two ML algorithms with the original DNS results. The probability density function (PDF) of convective turbulent heat flux ($j'_{\rm conv}$), see Eq. (17), contains the full statistical information of the fluctuations at all orders. It is extracted here at three different locations in the channel. The PDFs of all methods show a good overlap around the mean. Deviations are observed in the tails, particularly in the positive tails where the most intense rising and falling plume events appear. The differences between pure CAE application and the combination with the ESN or GRU remain small. A physically important property of this PDF is that has to be skewed to positive values since heat is transported from the bottom to the top on average. This property is reproduced well by all our ML methods. The pronounced positive tails correspond physically to both, rising warmer-than-average and falling colder-than average thermal plumes.[8,9]

F. Individually extracted modes and noise resilience

After directly comparing the flow fields with the ML prediction, we further verified the evolution of individual modes from the ESN evolution. This was done with the help of a principal component analysis (PCA). The PCA algorithm enables a further down-scaling from 40 modes in the latent space to two principal modes, which carry more than 65% of the variance. Figure 11 shows a scatterplot of the two primary PCA components for the three algorithms. We display the result of the CAE, i.e., ground truth for the subsequent RNN application together with the outputs of ESN and GRU for unseen test data. All data points are clustered in the same ranges and overlap. This warrants the learning and generalization ability of the ESN and GRU when compared to the CAE for the present dynamical system at hand.

Finally, we examined the robustness of the CAE against noise. Robustness is desired especially when one wants to use a CAE in an experimental facility or for a simulation model with parametrizations and closures. The objective of this step is to evaluate how the trained CAE will be affected by noise at the small scale. We investigated two distinct noise levels sampled from a random distribution with a zero mean and two different standard deviations σ. Figure 12 shows that

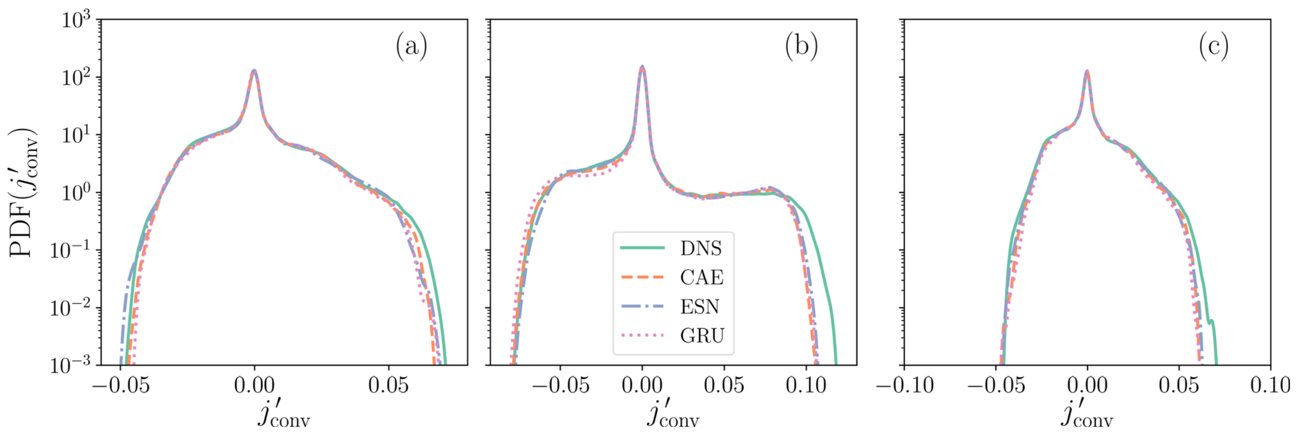


FIG. 10. Comparison of probability density functions (PDFs) for convective turbulent heat flux at three different locations in the wall normal direction z. (a) $z=0.16$. (b) $z=0.50$. (c) $z=0.84$. The legend holds for all three panels.

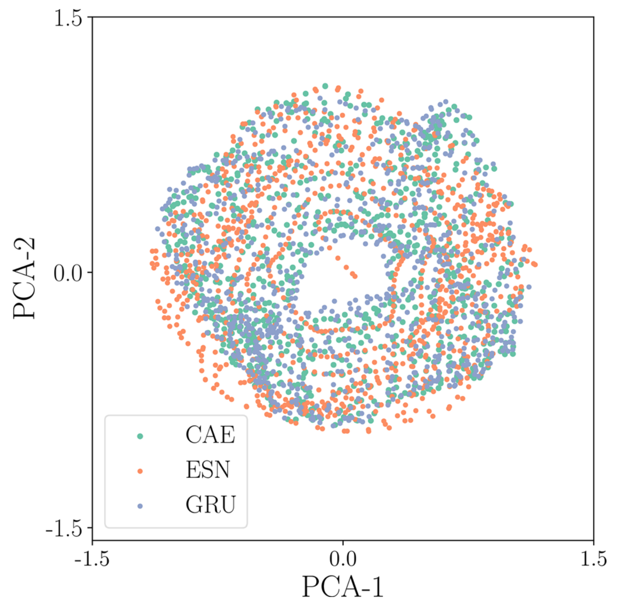


FIG. 11. Scatterplot visualization of the principal components obtained by a principal component analysis that was applied to the 40 modes in the latent space. The two primary components are denoted by PCA-1 and PCA-2.

the CAE is affected by the noise. We apply the analysis to a temperature field snapshot here. However, at the lower noise level these effects remain subdominant and the CAE can filter them out [cf. Figs. 12(a) and 12(c)]. while a higher noise level is beyond the filtering abilities of the CAE [cf. Figs. 12(d) and 12(f)].

Figure 13 substantiates this finding. Small levels of noise leave the mean and fluctuation profiles nearly unchanged. Higher levels of additive noise lead to stronger deviations in the low-order statistics as demonstrated in both panels of the figure.

V. CONCLUSIONS AND OUTLOOK

In this work, we have investigated two recurrent machine learning (ML) algorithms that model and forecast the local convective heat flux—the central quantity for the characterization of the mean turbulent heat transfer from the bottom to the top—in a two-dimensional turbulent Rayleigh–Bénard flow directly. This implies that this derived property, which is the product of the temperature and the vertical velocity component, together with its low-order statistics is modeled without applying the underlying highly nonlinear Boussinesq equations of motion.

In both models, a convolutional autoencoder (CAE) is applied first to reduce the high-dimensional data records, which are obtained from direct numerical simulations of the turbulent flow, in a low-dimensional latent space. The dynamics in the latent space is advanced by means of either an echo state network (ESN), one implementation of a reservoir computing model, or a gated recurrent unit (GRU). We find that both ML algorithms performed well and are able to reproduce mean profiles of the convective heat flux and its fluctuations fairly well. This includes even the reproduction of the whole probability density function. It is furthermore tested how resilient the models are with respect to a small amount of added noise. Our investigation demonstrates that the model is robust to such a noise with a small standard deviation of $\sigma \lesssim 0.02$ using the advantages of the CAE architecture for data reduction. The latter is often used for de-noising in image analysis.

Typical fluid dynamics data from turbulence simulations and experiments inherently contain a large number of degrees of freedom, which makes them unsuitable for a direct input into a machine learning algorithm. A reduction step, as applied here, is consequently necessary. Our CAE algorithm can be used on the fly (while the DNS is running and writing data records) and does not require the complete turbulence dataset at the beginning of the reduction, as it would be the case for a POD snapshot algorithm. Therefore, we presented a two-level neural network architecture that can reduce the dimensionality of data by using a nonlinear convolutional autoencoder and the latent vector can be autoregressively predicted by a recurrent network. Due to a large number of hyperparameters in the ESN case, we took a Bayesian optimization, which enables an efficient search of optimized hyperparameters by a relatively small number of iteration steps in comparison to a conventional grid search.

A potential application field of our CAE-RNN approach could be the modeling of mesoscale convective fluxes of heat (or moisture and salinity) in global circulation models of the atmosphere and ocean. These models are typically built on coarse computational grids that span the globe and require parametrizations of locally strongly varying unresolved fluxes.[68,69] The developed model provides a dynamical

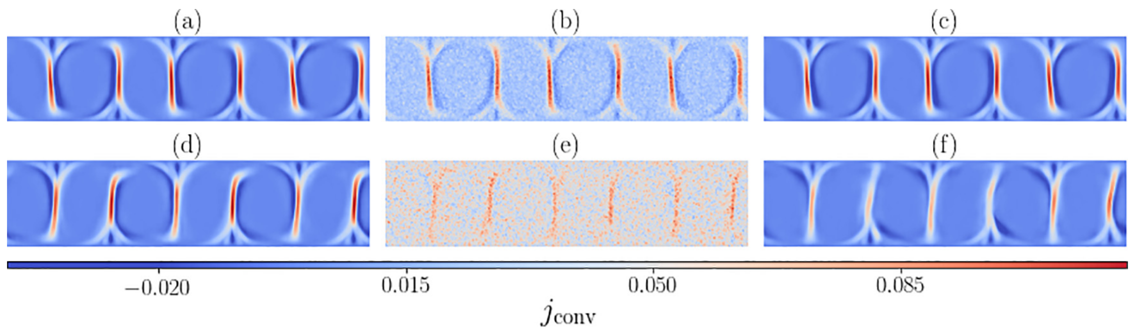
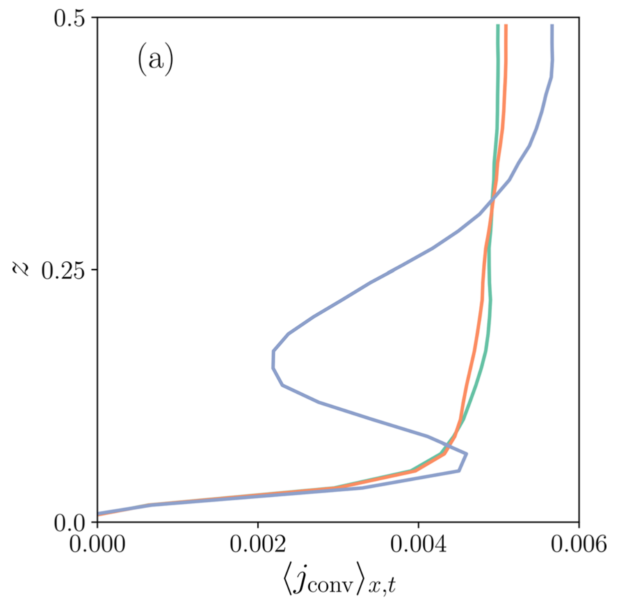


FIG. 12. Noise resilience of the CAE tested by means of contours of two individual temperature snapshots. (a) and (d) Prediction by the CAE for original temperature field without noise. (b) and (e) Original temperature field with added random noise. In panel (b), the standard deviation is $\sigma = 0.025$, in panel (e), the distribution is broader with $\sigma = 0.05$. (c) and (f) Prediction from the CAE for the corresponding noisy fields.



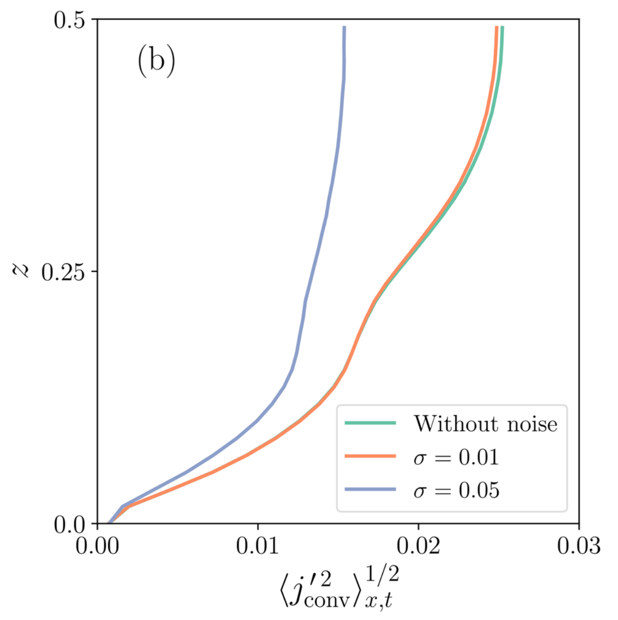


FIG. 13. Comparison of (a) the mean and (b) the fluctuation profiles of the convective heat flux over half the cell height for two different levels of random noise. Data are compared with the noise-free run. The profiles are the arithmetic mean over both halves of the layers.

ROM that delivers the low-order statistics of the turbulent transport in convection flows.

The extension to three-dimensional data records is required and possible with the given tools. It will however face new additional challenges. Three-dimensional data records will require higher-dimensional latent spaces and deeper networks for both, encoder and decoder. This suggests a possible decomposition of the weight matrix of the corresponding convolutional networks into matrix product states that have been successfully used in the solution of problems in quantum many-particle dynamics.[70] These investigations are currently in progress and will be reported elsewhere.

ACKNOWLEDGMENTS

This work was supported by the Deutsche Forschungsgemeinschaft with Grant No. SCHU 1410/30-1 and in part by Project No. P2018-02-001, "DeepTurb-Deep Learning in and of Turbulence," which is funded by the Carl Zeiss Foundation. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author upon reasonable request.

APPENDIX A: CONVOLUTIONAL AUTOENCODER ARCHITECTURE

The following Table IV details the architecture of the encoder–decoder network that was used in this work.

TABLE IV. Detailed structure of the convolutional autoenocoder. The table summarizes the encoder and decoder architectures (Conv = convolution, Max Pool = max pooling, Upsamp = upsampling). Symbol E2 denotes for example encoder hidden layer No. 2.

Encoder		Decoder	
Layer	Output size	Layer	Output size
Encoder input	$320 \times 60 \times 1$	Decoder input	$5 \times 1 \times 8$
2D conv-E1	$320 \times 60 \times 256$	2D conv-D1	$5 \times 1 \times 8$
Max pool-E1	$160 \times 30 \times 256$	2D upsamp-D1	$10 \times 2 \times 8$
2D conv-E2	$160 \times 28 \times 128$	2D conv-D2	$10 \times 2 \times 32$
Max pool-E2	$80 \times 15 \times 128$	2D upsamp-D2	$20 \times 4 \times 32$
2D conv-E3	$80 \times 15 \times 64$	2D conv-D3	$20 \times 4 \times 32$
Max pool-E3	$40 \times 8 \times 64$	2D upsamp-D3	$40 \times 8 \times 32$
2D conv-E4	$40 \times 8 \times 32$	2D conv-D4	$40 \times 8 \times 64$
Max pool-E4	$20 \times 4 \times 32$	2D upsamp-D4	$80 \times 16 \times 64$
2D conv-E5	$20 \times 4 \times 32$	2D conv-D5	$80 \times 16 \times 128$
Max pool-E5	$10 \times 2 \times 32$	2D upsamp-D5	$160 \times 32 \times 128$
2D conv-E6	$10 \times 2 \times 38$	2D conv-D6	$160 \times 32 \times 256$
Max pool-E6	$5 \times 1 \times 8$	2D upsamp-D6	$320 \times 64 \times 256$
		Output with 2 D conv	$320 \times 64 \times 1$
		Output with cropping	$320 \times 60 \times 1$

APPENDIX B: HYPERPARAMETER TUNING BY BAYESIAN OPTIMIZATION

The training of an ML algorithm relies on a cost function which depends on different hyperparameter vectors $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$. In the ESN case, the hyperparameter vector consists of $(N, D, \rho, \alpha, \beta)$. Grid and random search procedures are often used, and they proved to provide a (nearly) optimal solution.[71] As a downside, these methods are based on a parameter space which is predefined in the form of a multi-dimensional grid of parameter vectors. A more favorable alternative is the Bayesian optimization (BO), a global optimization method that automatically finds the optimal hyperparameter vector

$$\boldsymbol{\eta}_* = \arg\max_{\boldsymbol{\eta}} f(\boldsymbol{\eta}), \tag{B1}$$

by a relatively small number of iterations.[72] As the name suggests, BO utilizes the Bayes rule,

$$p(f(\boldsymbol{\eta})|\boldsymbol{\eta}) \sim p(\boldsymbol{\eta}|f(\boldsymbol{\eta}))p(f). \tag{B2}$$

The *a posteriori* probability of a hyperparameter model $f(\boldsymbol{\eta}) = (f(\eta_1), \ldots, f(\eta_n))$ given the hyperparameters $\boldsymbol{\eta}$ is similar to the likelihood of $\boldsymbol{\eta}$ given *f*, denoted as $p(\boldsymbol{\eta}|f)$, and the *a priori* probability *p*(*f*). Here, *p*(*f*) contains our obtained knowledge from prior iterations which is not discarded. Two main ingredients are necessary:

(1) BO typically utilizes a Gaussian process (GP) to model *p*(*f*) which is characterized by a mean $\mu(\boldsymbol{\eta})$ and a covariance matrix $k(\eta_i, \eta_j)$. Here, we use a Matérn kernel for the covariance matrix, which is given by[73]

$$k(\eta_i, \eta_j) = \frac{1}{\Gamma(\nu)2^{\nu-1}}\left[\frac{\sqrt{2\nu}}{l}d(\eta_i, \eta_j)\right]^{\nu} K_\nu\left(\frac{\sqrt{2\nu}}{l}d(\eta_i, \eta_j)\right).$$

In this equation, $d(\cdot,\cdot)$ is the Euclidean distance, K_ν is a modified Bessel function of the second kind, and Γ is the gamma function. The parameter ν, which controls the smoothness of the learned function, is set to $\nu = 1.5$.

(2) BO needs furthermore an acquisition function to determine the hyperparameter vectors that are going to be evaluated by *f* in the next iteration. This is a trade-off between the exploration, i.e., to sample at a high uncertainty region and exploitation, i.e., querying a high mean region. In this work, we have used upper confidence bound (UCB) as an acquisition function.[74] UCB at iteration step *t* is then given by [see also Eq. (B1)]

$$\boldsymbol{\eta}_* = \arg\max_{\boldsymbol{\eta}}(\mu_t(\boldsymbol{\eta}) + \kappa\sigma_t(\boldsymbol{\eta})), \tag{B3}$$

where μ_t is the mean and σ_t the standard deviation. Here, κ is a UCB model coefficient that is provided in the main text when UCB is applied.

REFERENCES

[1] L. P. Kadanoff, "Turbulent heat flow: Structures and scaling," Phys. Today **54**(8), 34–39 (2001).
[2] G. Ahlers, S. Grossmann, and D. Lohse, "Heat transfer and large scale dynamics in turbulent Rayleigh-Bénard convection," Rev. Mod. Phys. **81**, 503–537 (2009).
[3] F. Chillà and J. Schumacher, "New perspectives in turbulent Rayleigh-Bénard convection," Eur. Phys. J. E **35**, 58 (2012).
[4] J. Schumacher and K. R. Sreenivasan, "Colloquium: Unusual dynamics of convection in the Sun," Rev. Mod. Phys. **92**, 041001 (2020).
[5] M. K. Verma, *Physics of Buoyant Flows* (World Scientific, 2018).
[6] Q. Zhou, C. Sun, and K.-Q. Xia, "Morphological evolution of thermal plumes in turbulent Rayleigh-Bénard convection," Phys. Rev. Lett. **98**, 074501 (2007).
[7] S. Moller, C. Resagk, and C. Cierpka, "Long-time experimental investigation of turbulent superstructures in Rayleigh-Bénard convection by noninvasive simultaneous measurements of temperature and velocity fields," Exp. Fluids **62**, 64 (2021).
[8] O. Shishkina and C. Wagner, "Analysis of sheet-like thermal plumes in turbulent Rayleigh-Bénard convection," J. Fluid Mech. **599**, 383–404 (2008).
[9] M. S. Emran and J. Schumacher, "Conditional statistics of thermal dissipation rate in turbulent Rayleigh-Bénard convection," Eur. Phys. J. E **35**, 108 (2012).
[10] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science **349**, 255–260 (2015).
[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature **521**, 436–444 (2015).
[12] J. N. Kutz, "Deep learning in fluid dynamics," J. Fluid Mech. **814**, 1–4 (2017).
[13] K. Duraisamy, G. Iaccarino, and H. Xiao, "Turbulence modeling in the age of data," Annu. Rev. Fluid Mech. **51**, 357–377 (2019).
[14] M. P. Brenner, J. D. Eldredge, and J. B. Freund, "Perspective on machine learning for advancing fluid mechanics," Phys. Rev. Fluids **4**, 100501 (2019).
[15] S. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," Annu. Rev. Fluid Mech. **52**, 477–508 (2020).
[16] S. Pandey, J. Schumacher, and K. R. Sreenivasan, "A perspective on machine learning in turbulent flows," J. Turbul. **21**, 567–584 (2020).
[17] P. P. Vieweg, J. D. Scheel, and J. Schumacher, "Supergranule aggregation for constant heat flux-driven turbulent convection," Phys. Rev. Res. **3**, 013231 (2021).
[18] J. L. Lumley, "The structure of inhomogeneous turbulent flows," in *Atmospheric Turbulence and Radio Wave Propagation*, edited by A. M. Yaglom and V. I. Tatarski (Nauka, Moscow, 1967), pp. 166–178.
[19] G. Berkooz, P. Holmes, and J. L. Lumley, "The proper orthogonal decomposition in the analysis of turbulent flows," Annu. Rev. Fluid Mech. **25**, 539–575 (1993).
[20] P. J. Schmid, "Dynamic mode decomposition of numerical and experimental data," J. Fluid Mech. **656**, 5–28 (2010).
[21] D. Giannakis and A. J. Majda, "Nonlinear Laplacian spectral analysis for time series with intermittency and low-frequency variability," Proc. Natl. Acad. Sci. **109**, 2222–2227 (2012).
[22] C. W. Rowley, I. Mezić, S. Bagheri, P. Schlatter, and D. S. Henningson, "Spectral analysis of nonlinear flows," J. Fluid Mech. **641**, 115–127 (2009).
[23] D. Giannakis, A. Kolchinskaya, D. Krasnov, and J. Schumacher, "Koopman analysis of the long-term evolution in a turbulent convection cell," J. Fluid Mech. **847**, 735–767 (2018).
[24] J. Moehlis, T. R. Smith, P. Holmes, and H. Faisst, "Models for turbulent plane Couette flow using the proper orthogonal decomposition," Phys. Fluids **14**, 2493–2507 (2002).
[25] B. R. Noack, P. Papas, and P. A. Monkewitz, "The need for a pressure-term representation in empirical Galerkin models of incompressible shear flows," J. Fluid Mech. **523**, 339–365 (2005).
[26] J. Bailon-Cuba and J. Schumacher, "Low-dimensional model of turbulent Rayleigh-Bénard convection in a Cartesian cell with square domain," Phys. Fluids **23**, 077101 (2011).
[27] L. Soucasse, B. Podvin, P. Rivière, and A. Soufiani, "Reduced-order modelling of radiative transfer effects on Rayleigh–Bénard convection in a cubic cell," J. Fluid Mech. **898**, A2 (2020).
[28] S. Pawar, S. M. Rahman, H. Vaddireddy, O. San, A. Rasheed, and P. Vedula, "A deep learning enabler for nonintrusive reduced order modeling of fluid flows," Phys. Fluids **31**, 085101 (2019).
[29] S. A. Renganathan, R. Maulik, and V. Rao, "Machine learning for nonintrusive model order reduction of the parametric inviscid transonic flow past an airfoil," Phys. Fluids **32**, 047110 (2020).
[30] Z. Deng, Y. Chen, Y. Liu, and K. C. Kim, "Time-resolved turbulent velocity field reconstruction using a long short-term memory (LSTM)-based artificial intelligence framework," Phys. Fluids **31**, 075108 (2019).

[31] S. M. Rahman, S. Pawar, O. San, A. Rasheed, and T. Iliescu, "Nonintrusive reduced order modeling framework for quasigeostrophic turbulence," Phys. Rev. E **100**, 053306 (2019).
[32] L. Sirovich, "Turbulence and the dynamics of coherent structures. Part I: Coherent structures," Q. Appl. Math. **45**, 561 (1987).
[33] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," Science **304**, 78–80 (2004).
[34] Z. Lu, J. Pathak, B. R. Hunt, M. Girvan, R. Brockett, and E. Ott, "Reservoir observers: Model-free inference of unmeasured variables in chaotic systems," Chaos **27**, 041102 (2017).
[35] P. R. Vlachas, J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics," Neural Networks **126**, 191–217 (2020).
[36] F. Huhn and L. Magri, "Gradient-free optimization of chaotic acoustics with reservoir computing," Phys. Rev. Fluids **7**, 014402 (2022).
[37] S. Pandey and J. Schumacher, "Reservoir computing model of two-dimensional turbulent convection," Phys. Rev. Fluids **5**, 113506 (2020).
[38] F. Heyder and J. Schumacher, "Echo state network for two-dimensional moist Rayleigh-Bénard convection," Phys. Rev. E **103**, 053107 (2021).
[39] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," arXiv:1406.1078 (2014).
[40] S. Du, T. Li, and S.-J. Horng, "Time series forecasting using sequence-to-sequence deep learning framework," in *2018 9th International Symposium on Parallel Architectures, Algorithms and Programming (PAAP)* (IEEE, 2018), pp. 171–176.
[41] M. Sangiorgio and F. Dercole, "Robustness of LSTM neural networks for multi-step forecasting of chaotic time series," Chaos Solitons Fractals **139**, 110045 (2020).
[42] F. J. Gonzalez and M. Balajewicz, "Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems," arXiv:1808.01346 (2018).
[43] A. A. M. Al-Saffar, H. Tao, and M. A. Talab, "Review of deep convolution neural network in image classification," in *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)* (IEEE, 2017), pp. 26–31.
[44] Z. Han, J. Zhao, H. Leung, K. F. Ma, and W. Wang, "A review of deep learning models for time series prediction," IEEE Sens. J. **21**, 7833–7848 (2021).
[45] F. M. Bianchi, S. Scardapane, S. Løkse, and R. Jenssen, "Reservoir computing approaches for representation and classification of multivariate time series," IEEE Trans. Neural Networks Learn. Syst. **32**, 2169–2179 (2021).
[46] S. Qian, Y. Yu, L. Li, and Y. Chang, "An attention-based GRU encoder decoder for hostload prediction in a data center," in *2021 International Conference on Computer Communication and Artificial Intelligence (CCAI)* (IEEE, 2021), pp. 121–125.
[47] X. Lu, Y. Tsao, S. Matsuda, and C. Hori, "Speech enhancement based on deep denoising autoencoder," in *Interspeech* (ISCA, 2013), Vol. 2013, pp. 436–440.
[48] L. Gondara, "Medical image denoising using convolutional denoising autoencoders," in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)* (IEEE, 2016), pp. 241–246.
[49] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Association for Computing Machinery, Halifax, Canada, 2017), pp. 665–674.
[50] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science **313**, 504–507 (2006).
[51] Y. Wang, H. Yao, and S. Zhao, "Auto-encoder based dimensionality reduction," Neurocomputing **184**, 232–242 (2016).
[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016).
[53] J. Xu and K. Duraisamy, "Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics," Comput. Methods Appl. Mech. **372**, 113379 (2020).
[54] N. Omata and S. Shirayama, "A novel method of low-dimensional representation for temporal behavior of flow fields using deep autoencoder," AIP Adv. **9**, 015006 (2019).
[55] T. Murata, K. Fukami, and K. Fukagata, "Nonlinear mode decomposition with convolutional neural networks for fluid dynamics," J. Fluid Mech. **882**, A13 (2020).
[56] K. Fukami, T. Nakamura, and K. Fukagata, "Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data," Phys. Fluids **32**, 095110 (2020).
[57] S. Bhattacharya, M. K. Verma, and A. Bhattacharya, "Predictions of Nusselt and Reynolds numbers in turbulent convection using machine-learning models," Phys. Fluids **34**, 025102 (2022).
[58] H. Eivazi, L. Guastoni, P. Schlatter, H. Azizpour, and R. Vinuesa, "Recurrent neural networks and Koopman-based frameworks for temporal predictions in a low-order model of turbulence," Int. J. Heat Fluid Flow **90**, 108816 (2021).
[59] B. Lusch, J. N. Kutz, and S. L. Brunton, "Deep learning for universal linear embeddings of nonlinear dynamics," Nat. Commun. **9**, 4950 (2018).
[60] S. E. Otto and C. W. Rowley, "Linearly recurrent autoencoder networks for learning dynamics," SIAM J. Appl. Dyn. Syst. **18**, 558–593 (2019).
[61] P. F. Fischer, "An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations," J. Comput. Phys. **133**, 84–101 (1997).
[62] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 (2014).
[63] A. Chattopadhyay, P. Hassanzadeh, and D. Subramanian, "Data-driven predictions of a multiscale Lorenz 96 chaotic system using machine-learning methods: Reservoir computing, artificial neural network, and long short-term memory network," Nonlinear Processes Geophys. **27**, 373–389 (2020).
[64] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," arXiv:1409.0473 (2014).
[65] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," arXiv:1412.3555 (2014).
[66] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Society for Artificial Intelligence and Statistics, Sardinia, Italy, 2010) pp. 249–256.
[67] E. Fonda, A. Pandey, J. Schumacher, and K. R. Sreenivasan, "Deep learning in turbulent convection networks," Proc. Natl. Acad. Sci. **116**, 8667–8672 (2019).
[68] L. Zanna and T. Bolton, "Data-driven equation discovery of ocean mesoscale closures," Geophys. Res. Lett. **47**, e2020GL088376, https://doi.org/10.1029/2019GL085988 (2020).
[69] S. Bony, H. Schulz, J. Vial, and B. Stevens, "Sugar, gravel, fish, and flowers: Dependence of mesoscale patterns of trade-wind clouds on environmental conditions," Geophys. Res. Lett. **47**, e2019GL085988, https://doi.org/10.1029/2019GL085988 (2020).
[70] R. Orús, "Tensor networks for complex quantum systems," Nat. Rev. Phys. **1**, 538–550 (2019).
[71] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," J. Mach. Learn. Res. **13**, 281–305 (2012), available at http://jmlr.org/papers/v13/bergstra12a.html.
[72] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning* (Springer, Cham, 2019), pp. 3–33.
[73] M. G. Genton, "Classes of kernels for machine learning: A statistics perspective," J. Mach. Learn. Res. **2**, 299–312 (2001), available at https://www.jmlr.org/papers/v2/genton01a.html.
[74] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," arXiv:0912.3995 (2009).