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

Sandeep Pandey,1 Philipp Teutsch,2 Patrick Mäder,2,3 and Jörg Schumacher1,4
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, NY 11201, USA.

(Dated: 1 March 2022)

A combined convolutional autoencoder–recurrent neural network machine learning model is presented to analyse 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 Pr = 7 and Rayleigh number Ra = 10^7. Two recurrent neural networks are applied for the temporal advancement of flow data in the reduced latent data space, a reservoir computing model in the form of an echo state network and a recurrent gated unit. Thereby, the present 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 which suggests the present model might be applicable as a reduced dynamical model that delivers transport fluxes and their variations to the coarse grid cells of larger-scale computational models, such as global circulation models for the 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. The fundamental physical question in these flows is the one of the local and global characteristics of turbulent heat transfer which 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 which is enclosed by two impermeable parallel plates at distance H, known as the Rayleigh-Bénard convection case. The bottom plate is uniformly heated at a constant temperature T = T₀ + ∆T and the top plate is cooled at T = T₀. For temperature differences ∆T > 0 being large enough, the buoyancy-triggered fluid motion is turbulent. Convective turbulence is sustained by characteristic coherent structures which 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 and numerical studies. They are connected with the local convective heat flux which is given by

\[ j_{\text{conv}}(x,t) = u_z(x,t)\theta(x,t), \quad (1) \]

with

\[ \theta(x,t) = T(x,t) - \langle T(z)\rangle_{\Lambda}, \quad (2) \]

Here, \( u_z \) is the vertical velocity component and \( \theta \) the deviation of the total temperature field \( T \) from the mean profile \( \langle T(z)\rangle_{\Lambda} \). 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 this central transport quantity and its statistical properties directly without solving the underlying nonlinear equations by means of recurrent neural networks. 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 analyse, model and control turbulent flows. 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, have 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 ∼ 10^8 is resolved with about 6 billion spectral collocation points on an unstructured spectral element mesh for each of the involved fields, such as the three velocity components, temperature, and pressure. It amounts to 181 GByte of raw data per snapshot. The estimate does not incorporate the temporal dynamics that is typically stored as a sequence of such highly
Reduced-order models (ROM) 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)\[39,40\], Dynamic Mode Decomposition (DMD)\[41\], nonlinear Laplacian spectral analysis\[42\], or expansions in modes and eigenfunctions of the Koopman operator\[43–45\] to mention only a few. Particularly, the POD is still a popular workhorse for projection-based reduced models\[25–27\] and has been combined more recently also with ML algorithms\[46–49\]. With the increase of the vigor of turbulence (which is in line with an increase of 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 which is a 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 \(j_{\text{conv}}(x,t)\). 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 \(j_{\text{conv}}(x,t)\) 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 is discussed in detail. We discuss two CAE-RNN architectures which 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 \(j_{\text{conv}}(x,t)\). The two chosen RNN architectures are as follows:

- **Echo state networks (ESNs)** belong to the class of reservoir computing models\[53\] and have been used to model the nonlinear dynamics of the Lorenz 63 or Lorenz 96 models, chaotic acoustic models, and two-dimensional convection without and with phase changes\[54,55\]. In the latter cases, the ESN was combined with a data reduction by POD\[47,48\] 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\[59\] 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\[59,60\] by a convolutional encoder/decoder network, which will be better suited for higher-dimensional simulation data. Differently to previous studies, we feed the derived field \(j_{\text{conv}}(x,t)\) 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 here even further to a compression by 99.7% while losing only about 5% of the variance.

Finally, we mention that RNN-based encoder-decoder architectures are often used as comparison model for ESN models in similar applications\[52,55\]. Encoders and decoders have been successfully employed for de-noising\[56–60\]. anomaly detection\[59\] and to dimensionality reduction\[40,41\] 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\[51\]. For example, the spatio-temporal dynamics of fluid flows past cylinders and airfoils\[61,62\] or for turbulent channel flow\[63\] were successfully predicted and analyzed with CAEs. It should be mentioned that Koopman methods have been recently combined with RNNs\[55\]. Neural network algorithms have been also used to fit the global laws of turbulent heat and momentum transfer\[59\].

The outline of the manuscript is as follows. Section II presents the Boussinesq equations of turbulent convection and the DNS in brief that generate the data base. Section III discusses 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 section V. Further specific details on the architecture of the CAE and the training are summarized in appendices 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\[64\] in the two-dimensional case. The Boussinesq equations (3)–(5), which couple the velocity components \((u_i, u_j)\) and temperature \(T\), are solved in a closed rectangular cell of an aspect ratio \(L/H = 6\). They are given in dimensionless form by

\[
\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = \frac{\partial p}{\partial x_i} + \sqrt{\Pr} \frac{\partial^2 u_i}{\partial x^2_j} + T \delta_{i3},
\]

\[
\frac{\partial T}{\partial t} + u_j \frac{\partial T}{\partial x_j} = \frac{1}{\sqrt{\text{RaPr}}} \frac{\partial^2 T}{\partial x^2_j}.
\]
The pressure field is denoted by \( p \) and \( i, j \in \{x, z\} \). The horizontal coordinate is given by \( x \), the vertical one by \( z \). The dimensionless Rayleigh number \( Ra \) is a measure of the vigor of convective turbulence, set to \( Ra = 10^4 \) here. The dimensionless Prandtl number \( Pr \) which is the ratio of momentum to thermal diffusion was fixed to \( Pr = 7 \), as for thermal convection in water. All equations are made dimensionless by the cell height \( H \), free-fall velocity \( U_f = \sqrt{g \alpha \Delta T H} \) \( (g \) is the acceleration due to the gravity and \( \alpha \) the thermal expansion coefficient) and the temperature difference between the bottom and top plates \( \Delta T > 0 \). The simulation apply \( 48 \times 16 \) spectral elements. On each element, the 4 fields are expanded in polynomials of order 11 in each space dimension. For the machine learning analysis, these fields are interpolated spectrally 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 on 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 \( u_z \) (right column) that are combined to the derived convective heat flux field \( j_{\text{conv}} \) (middle column). The latter field is characterized by sharp ridges which suggest a strongly localized convective heat flux. Exactly these ridges will be extracted as the dominant features on which \( \Delta T \) 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 spatio-temporal data, a recurrent neural network (RNN) to forecast dynamics in the reduced order space (also known as latent space), and a 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.

A. Generation of compressed snapshot representation

An autoencoder is a machine learning model consisting of an encoder and a decoder part, 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 labelling effort for a training dataset is required. That is, in theory an autoencoder encodes input data \( I \) into \( e = \text{encode}(I) \) where \( e \in \mathbb{R}^L \). It decodes \( e \) back into \( \hat{I} = \text{decode}(e) \) subsequently. In practice, however, there will be deviations, such that \( \hat{I} \approx I \) (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) \(^{59}\).

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 \( I \in \mathbb{R}^{C_{\text{in}} \times N_x \times N_z} \) with \( C_{\text{in}} \) the number of input channels. Here,
\( C_{in} = 1 \) and thus \( I \in \mathbb{R}^{N_i \times N_z} \). This input is convoluted with a kernel \( k_m \) as shown in Eq. (6), where \( m \in [1, C_{out}] \) with the number of output channels \( C_{out} \). Furthermore, \( b_m \) is the bias term of kernel \( k_m \), and \( \psi \) 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,

\[
\text{Conv}(m, I) = \psi \left( \sum_{i=1}^{C_{in}} k_m * I_i \right) + b_m \text{ for } m \in [1, C_{out}] . \tag{6}
\]

In the encoder, one or multiple convolutional layers are typically followed by a pooling layer using a window of configurable size and sliding with a configurable step across the spatial dimensions of the layer’s 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 so that it reduces the dimensionality of input \( I \). The eventual output of the encoder is the input data represented in latent space \( \mathbb{R}^l \) where \( l \ll N_i \times N_z \). The CAE’s decoder follows a mostly analogous design. The main difference is the use of upsampling layers rather then max-pooling layers to decode data in latent space back to the input domain shape. The effective processing of complex turbulence data requires an architecture of multiple convolutional and accompanying upsampling layers in order to gain a compressed 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 \( c_t \) from a given snapshot \( I_t \). The compressed snapshot becomes input to a subsequent RNN that forecasts the next reduced representation \( \hat{c}_{t+1} \), which is then decoded back into the input domain by the CAE’s decoder.

**B. Forecast of compressed snapshots**

We study two types of RNNs for forecasting compressed snapshots in the latent domain, echo state networks (ESN) and gated recurrent units (GRU).

1. **Echo state network**

The ESN has previously shown great potential in modeling sequential turbulent flow data \( \text{[35][36]} \). 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).

The reservoir is represented as an adjacency matrix \( \mathbf{W}^r_\eta \) whose initialization depends on a vector of hyperparameters \( \eta \) (cf. Section 4) and is used to encode the ESN’s input into a hidden representation, the reservoir, that accumulates information of previous inputs. This reservoir is furthermore updated per time step with new input. More specifically, at each time step \( t \) the ESN’s input \( c_t \) influences the computation of an updated reservoir state \( r_t \in \mathbb{R}^N \) which is computed by

\[
r_t = (1 - \alpha) r_{t-1} + \alpha \cdot \text{tanh} \left( \mathbf{W}^{(in)}_\eta c_t + \mathbf{W}^r_\eta r_{t-1} \right) , \tag{7}
\]

where \( N \) denotes the size of the reservoir with \( N \gg M_x \times M_z \) by \( r_t = \mathbf{W}^{(in)} c_t \) and \( \alpha \) denotes a leakage rate determining the blending of previous state and current input. The random matrices \( \mathbf{W}^{(in)} \) and \( \mathbf{W}^r_\eta \) are initialized at the beginning of the training and remain unchanged thereafter. The ESN’s output at \( t \) is obtained by

\[
\hat{c}_{t+1} = \mathbf{W}^{(out)} r_t , \tag{8}
\]

where the components of \( \mathbf{W}^{(out)} \) are the only trained parameters, optimized by a mean squared error cost function with a ridge regression regularization in a non-iterative manner.
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 $\rho(W^{(r)})$, the leakage rate $\alpha$, and the Tikhonov regularization parameter $\beta$ in the cost function. Thus $\eta = (N, D, \rho, \alpha, \beta)$. This training is inherently fast compared to other types of RNN. After successful training and hyper-parameter optimization, the ESN runs as an autonomous dynamical system, i.e., a forecasted compressed snapshot $c_{t+1}$ can be used as the next ESN input to forecast $c_{t+2}$ and so on.

2. Gated recurrent unit

As already mentioned in the introduction, 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

$$\rho_t = \sigma(W^{(r)} \cdot [h_t, c_t]),$$

and

$$\omega_t = \sigma(W^{(o)} \cdot [h_t, c_t]),$$

where $\rho_t$ denotes the reset gate vector and $\omega_t$ the update gate vector at time step $t$; $W^{(r)}$ and $W^{(o)}$ are the corresponding weight matrices applied to a vector formed by concatenating the input vector $c_t$ at time step $t$ with the hidden state vector $h_t$. The sigmoid activation function $\sigma$ 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, $\rho_t$ is element-wise multiplied with $h_t$ to “reset” individual values of the old state when computing the updated intermediate state $\hat{h}_t$ as

$$\hat{h}_t = \tanh(W \cdot [\rho_t \cdot h_t, c_t]),$$

where $W$ is an additional weight matrix and tanh is used as activation function. Analogously, $\omega_t$ is element-wise multiplied with $h_t$ and $(1 - \omega_t)$ with $\hat{h}_t$ to add current input information to the updated cell state $h_{t+1}$, which is also the cell’s output, given by

$$h_{t+1} = (1 - \omega_t) \cdot h_t + \omega_t \cdot \hat{h}_t.$$ 

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

We organize two GRU cells in an encoder-decoder architecture 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, $\hat{c}_{t+1}$. Following an initial externally triggered step, the decoder progresses auto-regressively: it consumes its output of the previous iteration $\hat{c}_{t+1}$ as an input for the computation of an updated latent representation which is used to forecast the next output $\hat{c}_{t+2}$.

IV. RESULTS AND DISCUSSION

Before we turn to the training, validation, and test phases, we list a few more details on the DNS data record. In total, we used 2,400 snapshots divided into three subsets. The first subset consisted of 1,000 snapshots exclusively used for training, thus, called training set. The second subset, called validation set, consisted of 500 snapshots used to cross-validate the training and to warrant the robustness and generalization ability of a given model in the runtime. The third subset contained the remaining 900 snapshots used as an independent, unseen, and non-trained test data set. We trained CAE and GRU on NVIDIA GeForce GTX 1060 and RTX 2080Ti GPUs, respectively, and the ESN on a CPU with 16 GBytes 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. Tab. [IV] in appendix A). The decoder consists of an additional cropping layer to gain the original data snapshot 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 1 shows the optimized parameters, the value search range per parameter, and the eventually 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 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. More details on the BO procedure are detailed in appendix B. After obtaining the optimized parameters, we trained the CAE and simultaneously cross-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. 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 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 for 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.

We started the BO at 2 random points for β which enable the calculation of the posterior distribution as shown in Fig. 6(a). The third observation is at β = 18.6 and the acquisition function in the form of an upper confidence bound (UCB) predicts the next query point at β = 16.4 because it becomes the area for exploitation as κ = 1. In Figure 6(b), it can be seen that the uncertainty becomes zero for β = 16.4 (assuming no noise) and that the acquisition function suggests a next point, β = 18.9. This iteration proceeds until we reach an
optimum or a predefined number of iterations. Figure 6(c) illustrates the status after 8 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 5 hyperparameters in vector $\vec{\eta}$ 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. Due to the high-dimensionality of our input data, the ESN is prone to overfitting which is visible in the cross-validation phase. To mitigate this problem, we added a negative penalty term to the cost function.

![Graph showing the performance of the Bayesian optimization model](image)

**FIG. 6.** Exemplary progress of the Bayesian optimization of the ESN. In panels (a) to (c), we plot the negative mean squared error (MSE) in the top together with the upper confidence bound in panel (c) for the search of the optimal Tikhonov regularization parameter $\beta$. (a) Iteration No. 1 after initialization with 2 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.

| Parameter                  | Search range   | Optimized value |
|----------------------------|----------------|-----------------|
| Reservoir size             | 100–5,000      | 2.992           |
| 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            |

**TABLE II.** Optimized parameters obtained from BO for the ESN. Here, 5 points were randomly chosen to initialize the prior, 50 iterations were used for the BO and $\kappa = 1$.

### 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. Tab. III). Differently to 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 (patience), i.e., we used a reduce learning rate on plateau scheduler. The reduction is performed by multiplying the current learning rate with a factor $\gamma$. 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.

| Parameter           | Search range | Optimized value |
|---------------------|--------------|-----------------|
| Initial learning rate | 0.0006, 0.0003, 0.001, 0.0006, 0.0003, 0.0001, 0.00006, 0.00003, 0.00001 | 0.001 |
| Batch size          | 32, 64, 128, 256, 512 | 128 |
| Hidden state size   | 128, 256, 384, 512, 1024 | 512 |

**TABLE III.** Hyperparameters used for the encoder-decoder GRU training

### 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 cross-validation, it is necessary now to assess their generalization ability by a comparison with unseen test data. These are taken out of the 900 remaining snapshots. Thereby, the ground truth DNS data are fed into the encoder, which generates lower-dimensional compressed data vectors of dimension $l = 40$, see again Sec. III A. These latent vectors are then used as an ini-
FIG. 7. Result from the blind test. (a)–(d) Instantaneous convective turbulent heat flux snapshots at a given time. (e)–(h) Mean convective turbulent heat flux field averaged over 900 snapshots. (a,e) DNS, i.e., ground truth. (b,f) CAE, i.e., ground truth for the ESN and GRU. (c,g) Prediction from the ESN with subsequent decoding. (d,h) Prediction from the GRU with subsequent decoding.

FIG. 8. Comparison mean convective turbulent heat flux and fluctuation over the 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.

A deviation of the mean profiles of the convective heat flux, $\langle u_z \theta(z) \rangle_{x,t}$, 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.

FIG. 8(b) shows fairly well overlapping profiles for the fluctuations of the convective heat flux for all 3 cases that re-
main close to the ground truth. Therefore

$$\langle u, \theta \rangle^t = u, \theta - \langle u, \theta (z) \rangle_{x, t},$$

(13)
is defined. We have also quantified the loss of information of the CAE application which is based on the integrated convective flux which is given by

$$\Phi = \int_A \langle \langle u, \theta \rangle^t \rangle^2, \, dA.$$  

(14)

The loss follows by

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

(15)

with \(i = \{\text{CAE, ESN, GRU}\} \). The results are \(L_{\text{CAE}} = 5.3\%\), \(L_{\text{ESN}} = 4.2\%\), and \(L_{\text{GRU}} = 6.7\%\).

E. Probability density function of convective heat flux

Figure 9 illustrates a further comparison of the two ML algorithms with the original DNS results. The probability density function (PDF) of convective turbulent heat flux \(\langle u, \theta \rangle^t\), see eq. (13), 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.

Figure 10 depicts the temporal autocorrelation functions of the local convective heat flux, see eq. (1), which were extracted from the middle of the convection domain at \(z = 1/2\). The autocorrelation is defined by

$$C(k\Delta t) = \sum_{k=0}^{M} \langle j_{\text{conv}}(t) j_{\text{conv}}(t + k\Delta t) \rangle_{x, z=1/2, t},$$

(16)

where \(\Delta t = 0.125\) is the time interval between subsequent output snapshots given in free-fall time units \(H/U_f\). All algorithms, CAE, ESN, and GRU follow the typical trend where the autocorrelation decays gradually to zero with oscillations about the zero axis. The results for the DNS and the CAE show a qualitatively similar variation including the oscillation frequency of fluctuations and the amplitude. This is not the case for the ESN and GRU cases. The corresponding autocorrelations are found to oscillate at a much higher frequency. Both, ESN and GRU show also a slight mismatch regarding the amplitude of the local maxima and minima. This specific analysis demonstrates thus the limitations of the combined CAE-RNN application. The reproduction of the time correlations in the latent space is challenging.

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 2 principal modes which carry more than 65% of variance. Figure 11 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.

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 \(\sigma\). Figure 12 shows that 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. Fig. 12, a,c) while a higher noise level is beyond the filtering abilities of the CAE (cp. Fig. 12, d,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 the present 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 amounts of added noise. Our investigation demonstrate 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 simulation and experiments inherently contain a large number of high-dimensional data records which make 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 data set at the beginning of the reduction, as for example in case of a POD snapshot algorithm. Therefore, we presented a two-level neural network architecture which can reduce the dimensionality of data by using a non-linear 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
Fig. 12. Noise resilience of the CAE tested by means of contours of 2 individual temperature snapshots. (a,d) Prediction by the CAE for original temperature field without noise. (b,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,f) Prediction from the CAE for the corresponding noisy fields.

Fig. 13. Comparison of (a) the mean and (b) the fluctuation profiles over the cell height for 2 different level of random noise. Data are compared with the noise-free run.

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. The developed model provides a dynamical 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 which have been successfully used in the solution of problems in quantum many-particle dynamics. These investigations are currently in progress and will be reported elsewhere.

ACKNOWLEDGMENTS

The work is supported by the Deutsche Forschungsgemeinschaft with Grant No. SCHU 1410/30-1 and in parts by the project “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.
DATA AVAILABILITY STATEMENT

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.

Appendix B: Hyperparameter tuning by Bayesian optimization

The training of an ML algorithm relies on a cost function which depends on different hyperparameters $\mathbf{x} = (x_1, ..., x_n)$. 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 (near) optimum solution.\[60\] 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

$$\mathbf{x}^* = \arg \max_{x \in \mathcal{X}} f(x), \quad (B1)$$

by a relatively small number of iterations.\[60\] As the name suggests, BO utilizes the Bayes rule,

$$p(f(x)|x) \sim p(x|f(x))p(f). \quad (B2)$$

The a-posteriori probability of a hyperparameter model $f(x) = (f(x_1), ..., f(x_n))$ given the hyperparameters $\mathbf{x}$ is similar to the likelihood of $\mathbf{x}$ given $f$, denoted as $p(x|f)$, 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 $\mu(x)$ and a covariance matrix $K(x_i, x_j)$. Here, we use a Matérn kernel for the covariance matrix which is given by\[60\]

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

In this equation, $d(\cdot, \cdot)$ is the Euclidean distance, $K_\nu$ is a modified Bessel function of the second kind, and $\Gamma$ is the gamma function. The parameter $\nu$, 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\[11\] UCB at iteration step $t$ is then given by

$$x_+ = \arg \max_x (\mu(x) + \kappa \sigma(x)), \quad (B3)$$

where $\mu$ is the mean and $\sigma$ the standard deviation. Here, $\kappa$ is a UCB model coefficient which is provided in the main text when UCB is applied.

\[11\] P. Kadanoff, “Turbulent heat flow: Structures and scaling,” Phys. Today 54, 34–39 (2001).

\[12\] 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).

\[13\] F. Chilla and J. Schumacher, “New perspectives in turbulent Rayleigh-Bénard convection,” Eur. Phys. J. E 35, 58 (2012).

\[14\] J. Schumacher and K. R. Sreenivasan, “Colloquium: Unusual dynamics of convection in the Sun,” Rev. Mod. Phys. 92, 041001 (2020).

\[15\] M. K. Verma, Physics of Buoyant Flows (World Scientific, 2018).

\[16\] Q. Zhou, C. Sun, and K.-Q. Xia, “Morphological evolution of thermal plumes in turbulent Rayleigh-Bénard convection,” Phys. Rev. Lett. 98, 074501 (2007).

\[17\] S. Möller, C. Resagk, and C. Cierpika, “Long-time experimental investigation of turbulent superstructures in Rayleigh-Bénard convection by non-invasive simultaneous measurements of temperature and velocity fields,” Exp. Fluids 62, 64 (2021).

\[18\] O. Shishkina and C. Wagner, “Analysis of sheet-like thermal plumes in turbulent Rayleigh-Bénard convection,” J. Fluid Mech. 599, 383–404 (2008).

\[19\] M. S. Emran and J. Schumacher, “Conditional statistics of thermal dissipation rate in turbulent Rayleigh-Bénard convection,” Eur. Phys. J. E 35, 108 (2012).

\[20\] M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects.” Science 349, 255–260 (2015).

\[21\] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” Nature 521, 436–444 (2015).

\[22\] J. N. Kutz, “Deep learning in fluid dynamics,” J. Fluid Mech. 814, 1–4 (2017).

\[23\] K. Duraisamy, G. Iaccarino, and H. Xiao, “Turbulence modeling in the age of data,” Annu. Rev. Fluid Mech. 51, 357–377 (2019).

\[24\] M. P. Brenner, J. D. Eldredge, and J. B. Freund, “Perspective on machine learning for advancing fluid mechanics,” Phys. Rev. Fluids 4, 100501 (2019).

\[25\] S. Brunton, B. R. Noack, and P. Koumoutsakos, “Machine learning for fluid mechanics,” Annu. Rev. Fluid Mech. 52, 477–508 (2020).

\[26\] S. Pandey, J. Schumacher, and K. R. Sreenivasan, “A perspective on machine learning in turbulent flows,” J. Turbul. 21, 567–584 (2020).

\[27\] P. P. Vieweg, J. D. Scheel, and J. Schumacher, “Supergranule aggregation for constant heat flux-driven turbulent convection,” Phys. Rev. Res. 3, 013231 (2021).

\[28\] 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.

\[29\] 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).

\[30\] P. J. Schmid, “Dynamic mode decomposition of numerical and experimental data,” J. Fluid Mech. 656, 5–28 (2010).

\[31\] D. Giannakis and A. J. Majda, “Nonlinear Laplacian spectral analysis for time series with intermittency and low-frequency variability,” PNAS 109, 2222–2227 (2012).

\[32\] 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).

\[33\] 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).

\[34\] J. Moelis, T. R. Smith, P. Holmes, and H. Faisst, “Models for turbulent plane Couette flow using the proper orthogonal decomposition,” Phys. Fluids 14, 2393–2397 (2002).

\[35\] 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).

\[36\] 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).

\[37\] L. Soucasse, B. Podvin, P. Rivière, and A. Soufiani, “Reduced-order mod-
### TABLE IV. Detailed structure of the convolutional autoencoder. 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 × 60 × 1 | Decoder input | 5 × 1 × 8 |
| 2D Conv-E1 | 320 × 60 × 256 | 2D Conv-D1 | 5 × 1 × 8 |
| Max Pool-E1 | 160 × 30 × 256 | 2D Upsamp-D1 | 10 × 2 × 8 |
| 2D Conv-E2 | 160 × 28 × 128 | 2D Conv-D2 | 10 × 2 × 32 |
| Max Pool-E2 | 80 × 15 × 128 | 2D Upsamp-D2 | 20 × 4 × 32 |
| 2D Conv-E3 | 80 × 15 × 64 | 2D Conv-D3 | 20 × 4 × 32 |
| Max Pool-E3 | 40 × 8 × 64 | 2D Upsamp-D3 | 40 × 8 × 32 |
| 2D Conv-E4 | 40 × 8 × 32 | 2D Conv-D4 | 40 × 8 × 64 |
| Max Pool-E4 | 20 × 4 × 32 | 2D Upsamp-D4 | 80 × 16 × 32 |
| 2D Conv-E5 | 20 × 4 × 32 | 2D Conv-D5 | 80 × 16 × 128 |
| Max Pool-E5 | 10 × 2 × 32 | 2D Upsamp-D5 | 160 × 32 × 128 |
| 2D Conv-E6 | 10 × 2 × 38 | 2D Conv-D6 | 160 × 32 × 256 |
| Max Pool-E6 | 5 × 1 × 8 | 2D Upsamp-D6 | 320 × 64 × 256 |
| Output with 2D Conv | 320 × 64 × 1 | Output with Cropping | 320 × 60 × 1 |
P. F. Fischer, “An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations,” J. Comput. Phys. 133, 84–101 (1997).

D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” preprint arXiv:1412.6980 (2014).

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 in Geophysics 27, 373–389 (2020).

D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” preprint arXiv:1409.0473 (2014).

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” preprint arXiv:1412.3555 (2014).

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 (2010) pp. 249–256.

E. Fonda, A. Pandey, J. Schumacher, and K. R. Sreenivasan, “Deep learning in turbulent convection networks,” PNAS 116, 8667–8672 (2019).

L. Zanna and T. Bolton, “Data-driven equation discovery of ocean mesoscale closures,” Geophys. Res. Lett. 47, e2020GL088376 (2020).

S. Bony, H. Schulz, J. Vial, and B. Stevens, “Sugar, gravel, fish, and flow- ers: Dependence of mesoscale patterns of trade-wind clouds on environmental conditions,” Geophys. Res. Lett. 47, e2019GL085988 (2020).

R. Orús, “Tensor networks for complex quantum systems,” Nat. Rev. Physics 1, 538–550 (2019).

J. Bergstra and Y. Bengio, “Random search for hyper-parameter optimization,” J. Mach. Learn. Res. 13, 281–305 (2012).

M. Feurer and F. Hutter, “Hyperparameter optimization,” in Automated Machine Learning (Springer, Cham, 2019) pp. 3–33.

M. G. Genton, “Classes of kernels for machine learning: A statistics perspective,” J. Mach. Learn. Res. 2, 299–312 (2001).

N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, “Gaussian process optimization in the bandit setting: No regret and experimental design,” preprint arXiv:0912.3995 (2009).