RESEARCH ARTICLE

WILEY

Non-intrusive reduced-order modeling using convolutional autoencoders

Rakesh Halder[1] | Krzysztof J. Fidkowski[1] | Kevin J. Maki[2]

[1]Department of Aerospace Engineering, University of Michigan, Ann Arbor, Michigan, USA

[2]Department of Naval Architecture and Marine Engineering, University of Michigan, Ann Arbor, Michigan, USA

Correspondence

Rakesh Halder, Department of Aerospace Engineering, University of Michigan, 1320 Beal Avenue, Ann Arbor, MI 48108, USA.
Email: rhalder@umich.edu

Abstract

The use of reduced-order models (ROMs) in physics-based modeling and simulation almost always involves the use of linear reduced basis (RB) methods such as the proper orthogonal decomposition (POD). For some nonlinear problems, linear RB methods perform poorly, failing to provide an efficient subspace for the solution space. The use of nonlinear manifolds for ROMs has gained traction in recent years, showing increased performance for certain nonlinear problems over linear methods. Deep learning has been popular to this end through the use of autoencoders for providing a nonlinear trial manifold for the solution space. In this work, we present a non-intrusive ROM framework for steady-state parameterized partial differential equations that uses convolutional autoencoders to provide a nonlinear solution manifold and is augmented by Gaussian process regression (GPR) to approximate the expansion coefficients of the reduced model. When applied to a numerical example involving the steady incompressible Navier–Stokes equations solving a lid-driven cavity problem, it is shown that the proposed ROM offers greater performance in prediction of full-order states when compared to a popular method employing POD and GPR over a number of ROM dimensions.

KEYWORDS

autoencoders, deep learning, machine learning, model reduction, non-intrusive, proper orthogonal decomposition

1 | INTRODUCTION

Physics-based modeling and simulation has become an essential tool in many engineering and science applications, allowing for highly accurate representations of physical systems which may be otherwise difficult to evaluate. Physics-based models consist of a set of governing equations, which are often found in the form of parametrized partial differential equations (PDEs) discretized over a computational domain, where a set of design parameters μ controls properties such as the boundary conditions, geometry of the computational domain, or physical properties. In industrial processes such as design optimization, a large number of designs needs to be evaluated and the accuracy, or fidelity of these models must be high. High-fidelity simulations of large scale models are very computationally intensive, requiring large amounts of

memory and computational time. This large computational cost can render many-query processes such as design optimization infeasible.

The use of reduced-order models (ROMs) is a common approach for drastically lowering this computational cost. ROMs create a surrogate model using training data from a set of computed simulations of the high-fidelity full-order model that allows for accurate, rapid, real-time evaluation of simulations at unseen design parameters. ROMs seek to reduce the dimensionality of full-order models, which contain a large number of degrees of freedom. This involves a compression phase in which a *reduced basis* (RB) of the solution space is obtained, from which accurate approximations of full-order solutions can be obtained through a set of *expansion coefficients*. ROMs consist of two stages: a computationally intensive offline stage where high-fidelity solutions are evaluated to obtain data snapshots and a low-dimensional surrogate model is trained, and an online stage where the surrogate model can be rapidly evaluated to approximate solutions at unseen design parameters.

A commonly used method for obtaining the reduced basis is the proper orthogonal decomposition (POD),[1,2] which utilizes the singular value decomposition (SVD) to obtain a low-rank trial subspace composed of a number of linearly independent basis vectors, a linear combination of which is used to approximate unrealized solutions. Projection-based RB methods[3,4] project the physics of the governing equations onto the low-rank trial subspace and solve a low-dimensional version of the full-order model. Although projection-based RB methods have been shown to offer robust performance, their computational cost remains large for certain nonlinear problems that have a non-affine dependence on the inputs for quantities such as residuals in computational fluid dynamics (CFD) models.[5] Methods such as the discrete empirical interpolation method (DEIM)[6] allow for an affine representation of operators to avoid full-order evaluations, although realizing this is often difficult and intractable for some nonlinear problems.

In *non-intrusive* ROMs, an alternative to projection-based ROMs, the governing physics are only used to generate solution snapshots of the high-fidelity model in the offline stage and are not projected onto a lower dimension in the online stage. A regression model is required to interpolate over the expansion coefficients of the training data to approximate them for unseen design parameters. Some popular interpolation methods include Gaussian process regression (GPR)[7,8] and neural networks.[9,10] Although RB methods using POD are widely used and generally offer good performance, they often produce inaccurate results for certain nonlinear problems, such as those dominated by advection, and require ROMs of large dimension to produce results with acceptable accuracy.[11] For nonlinear problems characterized by different physical regimes, localized POD subspaces[12] are often employed to mitigate this issue.

Whereas POD produces a trial subspace that is linear, recent methods have attempted to compute low-dimensional nonlinear *trial manifolds* that are more adept at handling nonlinear problems. Many recent advances have utilized machine learning and artificial intelligence (AI) methods to this end, which have been at the forefront of massive recent breakthroughs in numerous fields such as computer vision, natural language processing, and recommender systems.[13-15] The use of machine learning methods has become ubiquitous in many domains, significantly improving and even beating the performance of existing methods.

There are many machine learning methods for producing low-dimensional representations of high-dimensional data, many of which do so non-linearly as opposed to POD. Deep learning[16] approaches have been utilized to develop ROMs that provide efficient nonlinear trial manifolds of physical systems. Convolutional autoencoders (CAEs), a type of neural network, have been used in ROMs and have been shown to outperform POD-based methods.[17,18] Convolutional autoencoders are adept at learning data that are spatially distributed, including the solutions to PDEs discretized over a computational domain. Autoencoder neural networks consist of two parts: an encoder, which maps high-dimensional inputs to a low-dimensional code, and a decoder, which maps the low-dimensional code to an approximation of the high-dimensional input. In the context of ROMs, the code is analogous to the expansion coefficients that map back to the full-order solution space. When using autoencoders for ROMs, the entire network is trained in the offline stage, while only the decoder is used in the online stage for rapid evaluation of unseen solutions. To the best of our knowledge, there have been two attempts at using convolutional autoencoders for non-intrusive ROMs; one utilizing them for vehicle aerodynamic simulation[19] and another for natural convection in porous media.[20] The first found that using autoencoders only offers a very slight improvement over POD-based methods. The second involves unsteady problems with a limited number of design variables and does not elaborate on the projection errors provided by autoencoders and their relation to interpolation accuracy, nor does it involve a cross-validation analysis of the results. In this work, we propose a non-intrusive ROM framework consisting of a nonlinear trial manifold produced by a CAE that is augmented by GPR to handle the interpolation of expansion coefficients in the design parameter space. This ROM framework is referred to as CAE-GPR and its performance is compared to that of POD-GPR when applied to a problem which solves the incompressible Navier–Stokes equations over a number of ROM dimensions.

2 | FULL-ORDER MODEL

The full-order model (FOM) in this work is considered to be the solution $x(\mu) \in \mathbb{R}^N$ of a state variable in a system governed by a set of steady-state parameterized PDEs discretized over a computational domain $\Omega \in \mathbb{R}^d$. We consider design parameters $\mu \in \mathcal{D}$ that define both the computational domain and parameters of the governing equations. Here $\mathcal{D} \subseteq \mathbb{R}^p$ denotes the parameter space such that $x: \mathcal{D} \to \mathbb{R}^N$. The set of PDEs governing the FOM is solved numerically over Ω to generate a solution $x(\mu)$. The computational cost of numerically solving the system increases with its dimension N, which is in proportion with the fineness of Ω. Accurate or useful solutions of systems often require large values of N, resulting in large computational costs for a single solution. In processes such as design optimization, the need to evaluate the solutions for many different designs in real-time becomes infeasible if numerous FOMs have to be solved. This large computational cost motivates the use of reduced-order models, where a small number of FOMs are solved and used to create a computationally inexpensive surrogate model that can deliver accurate approximations in real time.

3 | LINEAR REDUCED BASIS METHOD

This section gives an overview of the proper orthogonal decomposition (POD), a popular method for constructing a linear reduced basis, which allows for the construction of full-order solutions as a linear combination of independent basis vectors. The basis vectors are formed from a collection of training solutions over the parameter space. A snapshot matrix is assembled from these training solutions, from which the underlying structure of the solution space can be extracted.[1] Reduced-order models essentially obtain low-dimensional representations of any solution lying on the solution manifold for which there exists a mapping back to the solution space. In classification, a popular choice for linear dimensionality reduction is principal component analysis (PCA), to which the proper orthogonal decomposition is closely related.

3.1 | Proper orthogonal decomposition

Linear RB reduced-order models rely on training data obtained from a set of n solution snapshots calculated at chosen design points in the parameter space. A snapshot matrix, $S \in \mathbb{R}^{N \times n}$, is assembled

$$S \in \mathbb{R}^{N \times n} = [x^1, x^2, \ldots, x^n] = [x(\mu^1), x(\mu^2), \ldots, x(\mu^n)]. \tag{1}$$

Denote by $\mathcal{M}$ a subspace of the column space of S. We assume that $\mathcal{M}$ provides a good approximation of the solution manifold for $\mu \in \mathcal{D}$ if there are a sufficient number of solution snapshots in S which correspond to a judiciously chosen subset of design parameters in $\mathcal{D}$. $\mathcal{M}$ is the span of k orthonormal basis vectors, $[\psi^1, \psi^2, \ldots, \psi^k] \in \mathbb{R}^N$, where $k \ll N$. The basis is chosen such that each solution snapshot x^i in S can be well-approximated as a linear combination of the basis vectors

$$x^i \approx a_1^i \psi^1 + a_2^i \psi^2 \cdots + a_k^i \psi^k, \tag{2}$$

where a^i is the set of basis coefficients, or expansion coefficients, for a given solution snapshot. The truncated singular value decomposition of S, contains two orthonormal matrices $U \in \mathbb{R}^{N \times n}$ and $V \in \mathbb{R}^{n \times n}$, as well as a diagonal matrix $\Sigma \in \mathbb{R}^{n \times n}$

$$S = U\Sigma V^T. \tag{3}$$

Here, U contains a set of n left singular vectors that form an orthonormal basis for the column space of S, V contains a set of n right singular vectors that form an orthonormal basis for the row space of S, and $\text{diag}(\Sigma) \in \mathbb{R}^n = [\sigma_1, \sigma_2, \ldots, \sigma_n]$ contains the singular values corresponding to the singular vectors in descending order, $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. The first k left singular vectors of U are chosen to be the basis vectors forming the POD basis, $\Psi \in \mathbb{R}^{N \times k} = [\psi^1, \psi^2, \ldots, \psi^k]$. Often, the singular values associated with the basis vectors decay very quickly and only the first k singular vectors are chosen to form
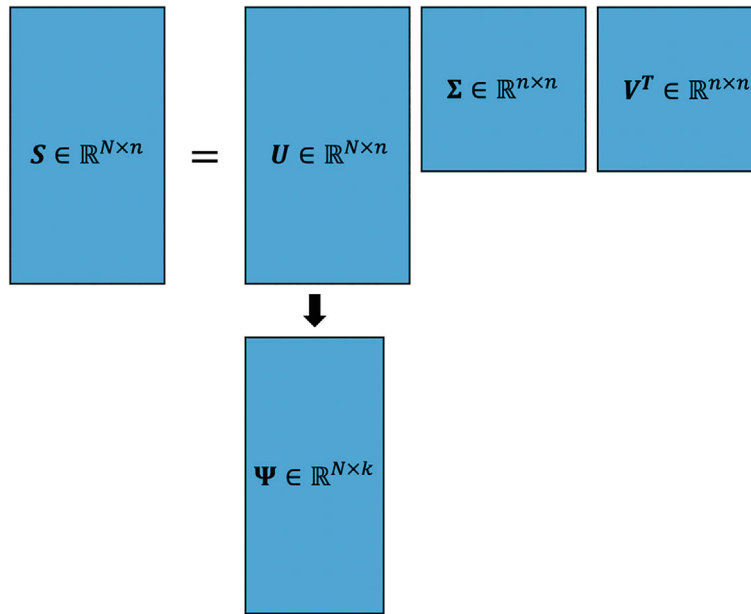
FIGURE 1 Schematic of the proper orthogonal decomposition (POD), where the snapshot matrix S is decomposed using the singular value decomposition (SVD) and the POD basis Ψ is obtained from U.

the POD basis to preserve only the most dominant basis vectors. To determine the value of k, the relative information content of the subspace is evaluated

$$E(k) = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{n} \sigma_j^2}, \tag{4}$$

and k is chosen such that $E(k) \geq \gamma$, where $\gamma \in [0,1)$ is chosen somewhat arbitrarily, usually to a value $\gamma \geq 0.95$.[21] Using the POD basis, full-order solutions at unseen design parameters $x(\mu^*)$ can be approximated

$$x(\mu^*) \approx \Psi a^* = a_1^* \psi^1 + a_2^* \psi^2 \cdots + a_k^* \psi^k, \tag{5}$$

where a^* can be estimated through a computational model that takes μ^* as an input (Figure 1).

3.2 | Projection error

A measure of quality of the POD basis is its ability to reconstruct solution snapshots x^i in S with a high degree of accuracy. We first calculate the projection of x^i onto Ψ

$$\hat{x}^i = \Psi \Psi^T x^i. \tag{6}$$

A measure of the relative error over all of the solution snapshots in S is measured through the quantity

$$\epsilon_{\text{POD}} = \sum_{i=1}^{n} \frac{\left\| x^i - \hat{x}^i \right\|^2}{\left\| x^i \right\|^2}. \tag{7}$$

The Schmidt–Eckart–Young theorem[22] states that the POD basis consisting of the first k left singular vectors found from the SVD of S minimizes this error amongst all orthonormal bases of rank k.

4 | NONLINEAR MANIFOLD CONSTRUCTION USING CONVOLUTIONAL AUTOENCODERS

This section describes the use of deep convolutional autoencoders in constructing a nonlinear trial manifold of simulation data. Unlike a linear reduced basis, which constructs solutions as a linear combination of known and calculated basis vectors, nonlinear manifolds use a mapping function $g(\boldsymbol{a})$, which may not be known explicitly, to approximate a mapping between the expansion coefficients and the full-order solution

$$g(\boldsymbol{a}) : \mathbb{R}^k \rightarrow \mathbb{R}^N. \tag{8}$$

There are many popular methods for nonlinear dimensionality reduction such as Isomap[23] that provide a low-dimensional embedding of high-dimensional data. Nonlinear methods have been shown to offer better performance in classification tasks when compared to linear methods such as PCA,[24] highlighting the advantage of using nonlinear methods to create low-dimensional representations of data. However, most nonlinear dimensionality reduction methods do not provide a mapping back to the high-dimensional solution space which restricts their use in ROMs. Deep convolutional autoencoders, which do provide an approximate mapping $g(\boldsymbol{a})$, have been utilized in projection-based ROMs where they have been shown to outperform POD-based methods.[17] Non-intrusive methods have also used deep CAEs to construct nonlinear manifolds that efficiently learn the dynamics of physical systems.[25]

4.1 | Artificial neural networks

An artificial neural network (ANN) is a computational model that is able to learn from a training data set $\mathcal{T} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ refer to the inputs and outputs respectively. ANNs are inspired by biological neural networks existing in animal brains.[26] ANNs are widely used and versatile models for regression and classification problems. Feedforward neural networks are a type of ANN in which information always propagates in only one direction, creating a direct mapping between inputs and outputs. Feedforward neural networks are composed of an input layer, a number of hidden layers, and an output layer. Neurons comprising these layers are associated with weights and biases, trainable parameters which are optimized during the model training stages. Figure 2 shows the architecture of a simple feedforward neural network with an input layer, two hidden layers, and an output layer. There are connections between each possible pair of neurons between layers, with each connection carrying a weight term and each neuron carrying a bias term with the exception of those in the input layer. Such a network is referred to as fully connected, or a multilayer perceptron (MLP). Hidden layers in MLPs are also referred to as fully connected layers. Each hidden layer state $\boldsymbol{h}_j$ is computed from the state in the previous layer, $\boldsymbol{h}_{j-1}$, along with its weights $\boldsymbol{W}_j$ and biases $\boldsymbol{b}_j$ as well as an activation function $\phi(x)$

$$\boldsymbol{h}_j = \phi\left(\boldsymbol{W}_j \boldsymbol{h}_{j-1} + \boldsymbol{b}_j\right). \tag{9}$$

The role of activation functions is to introduce nonlinearities into the model, allowing for complex functional relationships to arise. In addition to being able to learn the training data well, neural networks should provide reasonable accuracy for unknown data of the same class, a property referred to as generalization.[26] A commonly used activation function is the rectified linear unit (ReLU),[27] which has been shown to offer better performance and ability to generalize when compared to other common activation functions[28,29]

$$\phi(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \tag{10}$$

For inputs less than 0, the ReLU activation function returns a valuation and gradient of zero, effectively rendering certain neurons inactive. This can be problematic for network training if a large percentage of neurons exhibit this behavior and is commonly referred to as the dying ReLU problem. The leaky ReLU[30] activation function mitigates this issue, by incorporating a small positive constant α for negative inputs.
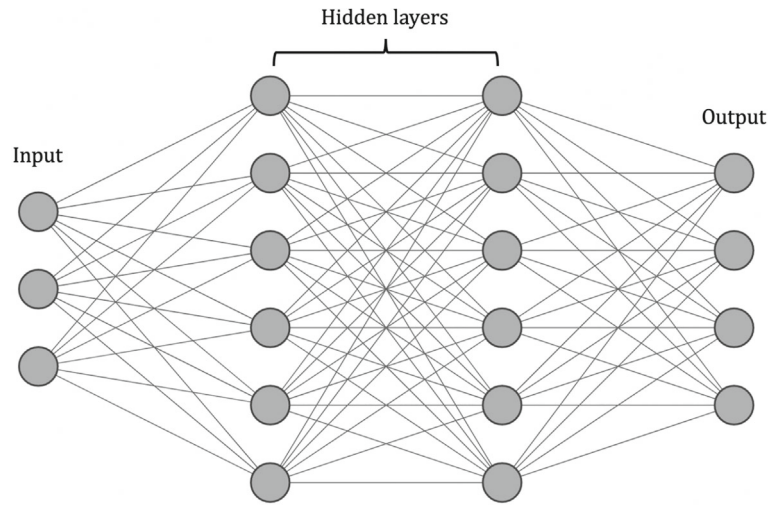
FIGURE 2 Architecture of a multilayer perceptron with a 3-dimensional input, six neurons in two fully connected layers, and four neurons in the output layer.

$$\phi(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}. \tag{11}$$

4.2 | Training neural networks

4.2.1 | Backpropagation

Feedforward networks are trained using a differentiable loss function, $\mathcal{L}\left(\mathcal{T}, (\boldsymbol{W}, \boldsymbol{b})\right)$, which calculates a measure of error between the state found in the output layer and the correct output values from the training data. The loss function serves as an objective function in an optimization problem, where its gradients with respect to the weights and biases are calculated through backpropagation,[31] an algorithm utilizing automatic differentiation. Common optimizers used in training neural networks include stochastic gradient descent (SGD) and Adam.[32] Optimizers perform a number of training epochs over $\mathcal{T}$ in an attempt to minimize the loss function. The weights and biases update at the end of epoch n according to

$$\left(\boldsymbol{W}^{n+1}, \boldsymbol{b}^{n+1}\right) = \left(\boldsymbol{W}^n, \boldsymbol{b}^n\right) - \eta \mathcal{G}\left(\frac{\partial \mathcal{L}\left(\mathcal{T}, \left(\boldsymbol{W}^n, \boldsymbol{b}^n\right)\right)}{\partial \left(\boldsymbol{W}^n, \boldsymbol{b}^n\right)}\right), \tag{12}$$

where η is the learning rate, a hyperparameter controlling the optimizer's step size and $\mathcal{G}$ is a function of the loss function's gradient dependent upon the chosen optimizer. The gradient of the loss function can be calculated using a single training sample as it is when using SGD, using the average gradient of the entire training set, or by using averages of a number of randomly selected mini-batches from $\mathcal{T}$. Using mini-batches when training neural networks has been shown to improve the ability to generalize in addition to providing stable convergence.[33] The mini-batch size b is chosen based on the size of $\mathcal{T}$ to strike a balance between performance and computational cost.

The predictive performance of a neural network initially increases with the number of training epochs but starts to stall and then decrease as the network parameters become overly tuned towards the training data and fail to generalize, a problem referred to as overfitting. Regularization methods[34] exist to prevent overfitting. One method is to use early stopping, where the loss on a validation data set $\mathcal{V}$ is monitored during training. If $\mathcal{L}\left(\mathcal{V}, (\boldsymbol{w}, \boldsymbol{b})\right)$ fails to drop for a prescribed number of epochs, training is stopped.

The initial set of weights and biases that are used can also effect the final performance of a neural network. A commonly used weight initialization scheme for layers using ReLU activation functions is the He normal[35] initializer, which samples weights from a normal distribution centered around 0. It is a common practice to initialize the biases in each layer to 0. There is no standard and accepted approach to choosing the number of hidden layers and the number of nodes

in each layer when designing MLPs. An optimal choice depends upon a number of factors, including the number of training samples, the dimensionality of the inputs and outputs, the choice of activation functions, and the complexity of the function which is being approximated. The number of hidden layers and nodes to use is often found through a trial and error approach involving model validation techniques such as cross-validation. In general, the total number of trainable parameters in a network is directly related to its capacity to learn functions. Neural networks become deeper as more hidden layers are added. However, network configurations with a large number of trainable parameters tend to overfit to the training data and fail to generalize unless regularization techniques are used. In addition, large networks are computationally expensive to train. In spite of these downfalls, deeper network architectures have become increasingly popular for complex learning tasks in multiple domains as they offer better performance.[16] Although more than two hidden layers are not required for many learning tasks, some functions are not adequately approximated by networks containing two hidden layers and using deeper networks can drastically improve performance.[36,37]

4.2.2 | Data normalization

Similar to many other machine learning algorithms, neural networks often require that the training data be normalized in order to ensure adequate performance.[38] Data normalization allows the optimizer to learn the optimal network parameters at a much faster rate. One way to normalize the training data is to apply min–max scaling to each feature in the data matrix D containing either the inputs or outputs

$$d' = \frac{d - \min(d_j)}{\max(d_j) - \min(d_j)}, \tag{13}$$

where j is the feature index. Min–max scaling results in the data being transformed into the range [0,1]. After training, new input data are also normalized while an inverse transformation is applied to predicted outputs.

4.3 | Autoencoders

Autoencoders are a type of feedforward neural network that aim to learn to reconstruct inputs in the output layer, $g : x \rightarrow \hat{x}$ where $x \approx \hat{x}$. Autoencoders use an architecture composed of two individual feedforward neural networks. The encoder $g_{enc} : \mathbb{R}^N \rightarrow \mathbb{R}^k$ where $k \ll N$ maps a high-dimensional input x into the low-dimensional code a. The decoder $g_{dec} : \mathbb{R}^k \rightarrow \mathbb{R}^N$ maps the code back to an approximation of the high-dimensional input $\hat{x}$. The combination of the two results in

$$g : \hat{x} = g_{dec} \circ g_{enc}(x). \tag{14}$$

Autoencoders have been shown to provide robust low-dimensional representations of high dimensional data.[39] In general, higher dimensional codes are able to produce more efficient representations of data, but continuously increasing the size of the code will provide diminishing or even negative returns. Once an autoencoder is sufficiently trained and $g(x) \approx x$ for all inputs over $\mathcal{T}$, the corresponding low-dimensional codes can be passed to the decoder $g_{dec}(a)$ to obtain accurate approximations $\hat{x}$ for all data in $\mathcal{T}$. States existing outside of the training set x^* can also be well-approximated if a good approximation of the low-dimensional code a^* can be found. In the context of ROMs, the code is equivalent to the set of expansion coefficients that map from a low-dimensional representation to the high-dimensional full-order solution. Similarly, the projection of a full-order solution onto the nonlinear manifold provided by the autoencoder is given by $\hat{x}$. Training is conducted on the combination of the encoder and decoder, while after training the encoder is often no longer useful and only the decoder is used. Figure 3 shows a sample architecture of a symmetric MLP autoencoder with two hidden layers between the input/output layers and code. Since MLP autoencoders are fully connected, the total number of trainable parameters in the network can grow very large when the dimension of the input, N, is high. As the number of trainable parameters increases, the amount of training data required to sufficiently train the network to make reasonably accurate predictions also grows large. This is contrary to the objective of model reduction, which aims to make predictions using a limited amount of training data.
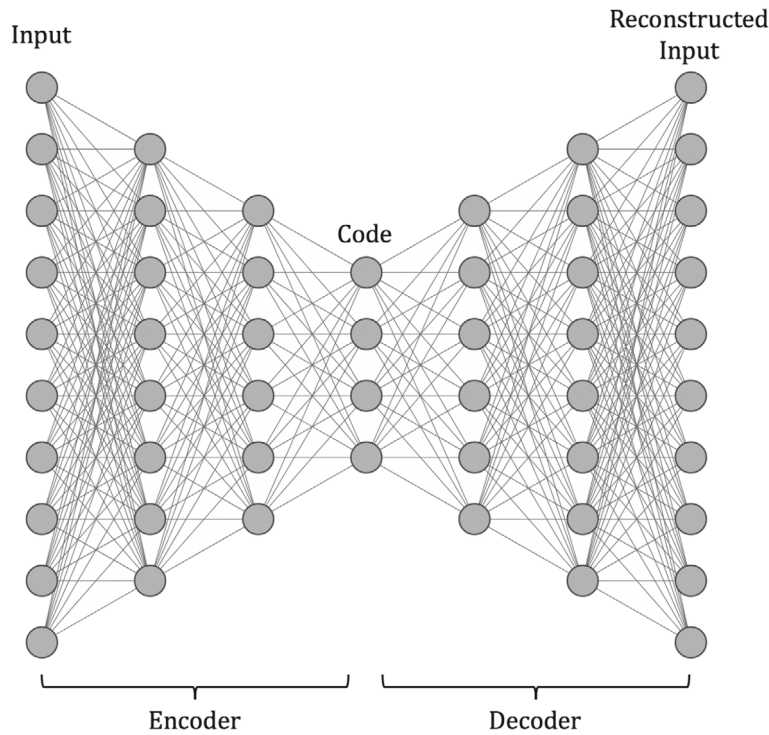
FIGURE 3   Architecture of a symmetric MLP autoencoder with two fully connected layers between the input/output and code

4.3.1 | Convolutional autoencoders

There exist neural network architectures that make use of *parameter sharing*, where rather than weight combinations existing for each pair of neurons between layers, multiple neurons share a single weight. *Convolutional autoencoders* effectively implement parameter sharing to limit the total number of trainable parameters in the network. This is done through the use of convolutional layers, which provide feature maps of input data that are spatially arranged.[40] Convolutional layers use a number of filters to convolve over spatially distributed input data, with each filter having its own set of weights. Pooling layers are also used in convolutional networks to summarize the features in input data through operations including averaging and maximization. Convolutional layers are widely used in the field of computer vision, dealing with spatially distributed data such as images.[13,41] CAEs can also be a useful tool for states that arise from numerically solving discretized PDEs as they tend to be spatially distributed. Data with multiple states, that is, components of velocity or levels of red, green, and blue in images, can also be handled well by CAEs through the use of a number of input channels. More details on convolutional layers can be found in a work by Dumoulin and Visin.[42] A combination of convolutional, pooling, and fully connected layers is used to construct CAEs, as shown in a schematic of an encoder section of a CAE in Figure 4. Spatially distributed data arising from the solutions of discretized PDEs often vary smoothly through the computational domain. CAEs are highly adept at handling data that are naturally spatially distributed by learning spatially invariant features, allowing them to outperform other neural network architectures.[43,44]

The input and output layers of CAEs usually consist of 2-dimensional (2D) states in each channel. Training data must be reshaped before being input into the network through the use of a reshape operator

$$\boldsymbol{R} : \mathbb{R}^{N \times n_c} \rightarrow \mathbb{R}^{n_y \times n_x \times n_c}, \tag{15}$$

where n_y refers to the number of data points in the vertical direction and n_x the number of data points in the horizontal direction. For solutions to physical problems, the number of points in each direction can be chosen based on the required resolution of the full-order state. The reshape operator is applied to each separate state that occupies the n_c input channels. An inverse reshape operator is used to reshape state output data in each output channel into the original vector format

$$\boldsymbol{R}^{-1} : \mathbb{R}^{n_y \times n_x \times n_c} \rightarrow \mathbb{R}^{N \times n_c}. \tag{16}$$
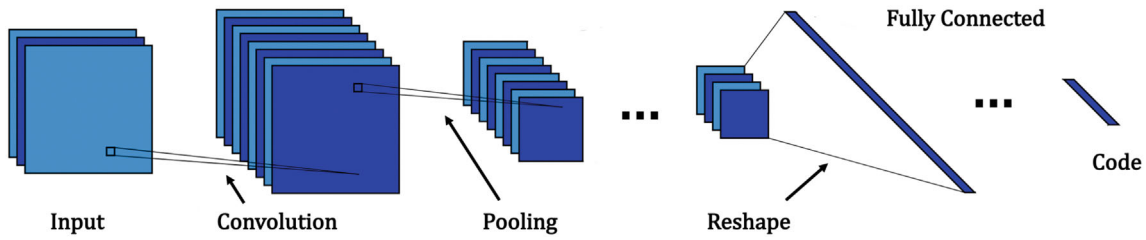
FIGURE 4 Architecture of the encoder of a convolutional autoencoder (CAE) consisting of convolutional, pooling, and fully connected layers.

5 | EXPANSION COEFFICIENT PREDICTION USING GAUSSIAN PROCESS REGRESSION

Non-intrusive ROMs require a regression model that can accurately predict the expansion coefficients $\boldsymbol{a}$ of unrealized solutions given their design parameters $\boldsymbol{\mu}$. The regression model is created in the offline stage utilizing the training data. In particular, we use a regression model to approximate a mapping $f(\boldsymbol{\mu})$ that outputs the expansion coefficients. A commonly used regression model in non-intrusive ROMs is GPR, a supervised learning method used for predictions of continuous outputs. GPR is also referred to as *Kriging*, and had one of its first uses in the field of geostatistics.[45] The regression model is constructed using training data composed of inputs $\bar{\boldsymbol{z}} = [\boldsymbol{z}^1, \boldsymbol{z}^2 \cdots \boldsymbol{z}^n]$ and outputs $\bar{\boldsymbol{y}} = [y^1, y^2 \cdots y^n]$, where each input $\boldsymbol{z}^i \in \mathcal{P} \subset \mathbb{R}^p$ belonging to an input domain $\mathcal{P}$ corresponds to a single output $y^i \in \mathbb{R}$. GPR infers a probability distribution over functions conditioned on the training data which is used for predictions at new inputs. A brief introduction to GPR is given in Section 5.1, and the work of Rasmussen et al.[46] can be referred to for a more complete overview.

In ROMs, an individual regression model is used for each coefficient in $\boldsymbol{a}$, leading to k different regression models,

$$f_i(\boldsymbol{\mu}) : \mathbb{R}^p \rightarrow \mathbb{R}, i \in [1, 2, \ldots k]. \tag{17}$$

As a result, ROMs using GPR tend to allow the number of expansion coefficients to be large as long as it does not degrade the quality of the trial manifold. GPR provides reasonable accuracy, is computationally inexpensive, does not require many training samples, and is easy to implement, making it a popular choice in non-intrusive ROMs. While the use of neural networks in non-intrusive ROMs has become more widespread,[9,10] finding a sufficient neural network architecture to use for regression is a non-trivial task. Even though using neural networks may offer better performance, in this work our goal is to highlight the advantages of using a nonlinear trial manifold compared to a linear reduced basis. As GPR offers more flexibility, we choose to use it as our regression model.

5.1 | Gaussian process regression

A Gaussian process (GP) is a set of random variables, of which any finite number follow a joint Gaussian distribution. In GPR, it is assumed that data are generated according to a GP with mean function m and covariance function κ,

$$f(\boldsymbol{z}) \sim \text{GP}\left(m(\boldsymbol{z}), \kappa(\boldsymbol{z}, \boldsymbol{z}^*)\right), \tag{18}$$

with some added Gaussian noise $\delta \sim \mathcal{N}(0, \xi_y^2)$,

$$y = f(\boldsymbol{z}) + \delta. \tag{19}$$

Using a finite amount of training data $\{\bar{\boldsymbol{z}}, \bar{\boldsymbol{y}}\}$, a prior joint Gaussian on the data and predictions at points $\boldsymbol{z}^*$ are given by

$$
\begin{bmatrix} \bar{\boldsymbol{y}} \\ f(\boldsymbol{z}^*) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} m(\bar{\boldsymbol{z}}) \\ m(\bar{\boldsymbol{z}}) \end{bmatrix}, \begin{bmatrix} \kappa(\bar{\boldsymbol{z}},\bar{\boldsymbol{z}}) + \xi_y^2 I & \kappa(\bar{\boldsymbol{z}},\boldsymbol{z}^*) \\ \kappa(\boldsymbol{z}^*,\bar{\boldsymbol{z}}) & \kappa(\boldsymbol{z}^*,\boldsymbol{z}^*) \end{bmatrix} \right). \tag{20}
$$

Using the properties of conditional Gaussian distributions, the conditional expectation of $f(\boldsymbol{z}^*)$ is given as

$$
\mathbb{E}(f(\boldsymbol{z}^*)|\bar{\boldsymbol{y}}) = \kappa(\boldsymbol{z}^*,\bar{\boldsymbol{z}})(\kappa(\bar{\boldsymbol{z}},\bar{\boldsymbol{z}}) + \xi_y^2 I)^{-1} \left(\bar{\boldsymbol{y}} - m(\bar{\boldsymbol{z}}) \right), \tag{21}
$$

where I is the identity matrix. In practice, the mean function m is set to the mean of the training outputs,

$$
m(\bar{\boldsymbol{z}}) = \frac{\sum_{i=1}^n y^i}{n}, \tag{22}
$$

and the inputs are scaled before training to obtain their standard score $\mathcal{Z}$

$$
\mathcal{Z}_j^i = \frac{z_j^i - m(\boldsymbol{z}_j)}{s_j}, \tag{23}
$$

where i and j refer to indices of the observation and input entry, respectively, and s is the sample standard deviation. There are many kernels that can be chosen for the covariance function. A very common one is the radial basis function (RBF) kernel

$$
\kappa(\boldsymbol{z},\boldsymbol{z}^*) = \exp\left(-\frac{d(\boldsymbol{z},\boldsymbol{z}^*)^2}{2l^2}\right). \tag{24}
$$

Another choice of kernel, and the one that will be used in this work, is the Matern kernel,

$$
\kappa(\boldsymbol{z},\boldsymbol{z}^*) = \frac{1}{\Gamma(\nu)2^{\nu-1}} \left(\frac{\sqrt{2\nu}}{l} d(\boldsymbol{z},\boldsymbol{z}') \right)^\nu K_\nu, \tag{25}
$$

where d is the Euclidean distance function, Γ is the gamma function, and K_ν is the modified Bessel function of the second kind. The set of hyperparameters θ of the Matern kernel are l and ν, which control the length scale and smoothness respectively. The predictive performance of the regression model is sensitive to the values of the hyperparameters. Gradient-based optimizers are often used to maximize the marginal log-likelihood of the training data to obtain an optimal set of hyperparameters θ_{opt}

$$
\theta_{\text{opt}} = \underset{\theta}{\text{argmax}} \ \log \ p(\bar{\boldsymbol{y}}|\bar{\boldsymbol{z}},\theta) = -\frac{1}{2}\bar{\boldsymbol{y}}^T(\kappa(\bar{\boldsymbol{z}},\bar{\boldsymbol{z}}) + \xi_y^2 I)^{-1} - \frac{1}{2}\log \ |\kappa(\bar{\boldsymbol{z}},\bar{\boldsymbol{z}}) + \xi_y^2 I| - \frac{n}{2}\log \ 2\pi. \tag{26}
$$

6 | OFFLINE AND ONLINE STAGES

This section describes the offline training and online evaluation stages of both the POD and CAE based ROMs with GPR as a regression model. The combined models consisting of both the offline and online stages are referred to respectively as POD-GPR and CAE-GPR. The offline stage is run first and is computationally expensive, while the online stage allows for rapid prediction of full-order models. Both models share a step of obtaining full-order snapshots of solutions evaluated at a set of design parameters $\boldsymbol{\mathcal{U}}_{\text{train}}$ and assembling them into a snapshot matrix $\boldsymbol{S}$. The offline stage of the POD-GPR method involves calculating a truncated SVD of the snapshot matrix to find the POD basis $\boldsymbol{\Psi}$. The set of expansion coefficients $\boldsymbol{A}_{\text{train}}$ is obtained for the training set using the POD basis and k GPR models $\boldsymbol{\mathcal{F}} = [f_1(\boldsymbol{\mu}), f_2(\boldsymbol{\mu}), \dots f_k(\boldsymbol{\mu})]$ trained on $\boldsymbol{\mathcal{T}} = \{\boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{A}_{\text{train}}\}$. After training, the GPR models are saved for use in the online stage, where unseen parameters $\boldsymbol{\mu}^*$ are evaluated to approximate the expansion coefficients $\tilde{\boldsymbol{a}}^*$. Matrix-vector multiplication of the POD-basis and expansion coefficients is then used to obtain an approximate solution $\tilde{\boldsymbol{x}}$. The POD-GPR method is outlined in Algorithm 1.

ALGORITHM 1. Offline and online stages of POD-GPR method

1: **function** PODGPR_ OFFLINE($\mathcal{U}_{\text{train}}$)
2:   Compute high-fidelity solutions for $\mu \in \mathcal{U}_{\text{train}}$ by solving FOM and assemble into S
3:   Calculate truncated SVD of snapshot matrix to obtain POD basis Ψ
4:   Calculate expansion coefficients for training data $A_{\text{train}} = \left(\Psi^T S\right)^T$
5:   Train k GPR models $F = [f_1(\mu), f_2(\mu), \dots f_k(\mu)]$ for each expansion coefficient in $\{\mathcal{U}_{\text{train}}, A_{\text{train}}\}$
6:   **return** $(\Psi, \mathcal{F})$
7: **end function**

1: **function** PODGPR_ ONLINE($\mu^*, \Psi, \mathcal{F}$)
2:   Evaluate expansion coefficients $\tilde{a}^* = \mathcal{F}(\mu^*)$
3:   Predict full-order solution $\tilde{x}^* = \Psi\tilde{a}^*$
4:   **return** $\tilde{x}^*$
5: **end function**

ALGORITHM 2. Offline and online stages of CAE-GPR method

1: **function** CAEGPR_ OFFLINE($\mathcal{U}_{\text{train}}, \mathcal{U}_{\text{val}}, C, R$)
2:   Compute high-fidelity solutions for $\mu \in \mathcal{U}_{\text{train}}, \mathcal{U}_{\text{val}}$ by solving FOM and assemble into $S_{\text{train}}, S_{\text{val}}$
3:   Apply reshape operator R to $S_{\text{train}}, S_{\text{val}}$ to obtain $X_{\text{train}}, X_{\text{val}}$
4:   Train convolutional autoencoder with architecture C on $\{X_{\text{train}}, X_{\text{train}}\}$ while monitoring loss on $\{X_{\text{val}}, X_{\text{val}}\}$
5:   Calculate expansion coefficients for training data $A_{\text{train}} = g_{\text{enc}}(X_{\text{train}})$
6:   Train k GPR models $F = [f_1(\mu), f_2(\mu), \dots f_k(\mu)]$ for each expansion coefficient in $\{\mathcal{U}_{\text{train}}, A_{\text{train}}\}$
7:   **return** $(g_{\text{dec}}, \mathcal{F})$
8: **end function**

1: **function** CAEGPR_ ONLINE($\mu^*, g_{\text{dec}}, \mathcal{F}, R^{-1}$)
2:   Evaluate expansion coefficients $\tilde{a}^* = \mathcal{F}(\mu^*)$
3:   Predict full-order solution $\tilde{X}^* = g_{\text{dec}}(\tilde{a}^*)$
4:   Apply inverse reshape operator R^{-1} to $\tilde{X}^*$ to obtain $\tilde{x}^*$
5:   **return** $\tilde{x}^*$
6: **end function**

The CAE-GPR method also involves evaluating additional full-order solutions at a set of validation design parameters $\mathcal{U}_{\text{val}}$ which are used to monitor the validation loss during training so early stopping can be implemented as a regularization method. A convolutional autoencoder architecture C is also required and the decoder g_{dec} is saved for use in the online stage. Approximate expansion coefficients $\tilde{a}$ are passed to the decoder to obtain approximate solutions $\tilde{X}$. The reshape operator R is also required to make the training and validation data compatible with the architecture of the CAE in the offline stage, while the inverse reshape operator R^{-1} is needed in the online stage to reshape approximated solutions to their original format. The CAE-GPR method is outlined in Algorithm 2. Training the convolutional autoencoder makes the offline stage of CAE-GPR more expensive than POD-GPR, while the online costs for both models are similar.

7 | NUMERICAL RESULTS

This section compares the performance of the POD-GPR and CAE-GPR methods on a geometrically and physically parameterized lid-driven cavity problem which simulates laminar flow using the steady incompressible Navier–Stokes equations with OpenFOAM,[47] an open-source toolbox for multiphysics simulation. The FOM quantities of interest are u and v, the components of the velocity in the horizontal and vertical directions respectively. The autoencoder is constructed using TensorFlow,[48] and scikit-learn[49] is used to implement GPR. 500 design parameters are generated using Latin hypercube sampling,[50] a statistical method that aims to maximize the distance and minimize the correlation amongst produced

samples. The metric of performance used to compare the ROMs is the relative l^2 error ϵ_{ROM} between the FOM state $\boldsymbol{x}$ and the ROM approximated state $\tilde{\boldsymbol{x}}$

$$\epsilon_{\text{ROM}} = \frac{\left\| \boldsymbol{x}^i - \tilde{\boldsymbol{x}}^i \right\|^2}{\left\| \boldsymbol{x}^i \right\|^2}. \tag{27}$$

Similarly, the relative projection error ϵ_{Proj} for both methods is also reported between the FOM state and the projected state $\hat{\boldsymbol{x}}$ to assess how accurately the expansion coefficients are interpolated as well as to provide a lower bound for the ROM prediction errors.

$$\epsilon_{\text{Proj}} = \frac{\left\| \boldsymbol{x}^i - \hat{\boldsymbol{x}}^i \right\|^2}{\left\| \boldsymbol{x}^i \right\|^2}. \tag{28}$$

A five-fold cross-validation approach is used to assess the performance of the ROM over the entire dataset, creating five folds of the dataset containing 400 training samples and 100 testing/validation samples. These 100 samples are split evenly into 50 testing and 50 validation samples, which are used to monitor the autoencoder loss during training. While every sample in the dataset is used for training, only half are used for prediction. An average cross-validation error is reported for both the ROM prediction and projection errors over all of the prediction points. The validation samples are not used for the POD-GPR ROM, which uses an individual ROM for both u and v.

The CAE that is used has two input channels, one for each of the velocity components. The expansion coefficients are used to approximate both u and v. The encoder consists of a combination of convolutional, pooling, and fully connected layers. The decoder consists of fully connected and transpose-convolutional layers. The full details of the network architecture can be found in Appendix. Min–max scaling is used independently on u and v before training and the CAE outputs are then scaled back to their original range after prediction. A maximum number of 7500 training epochs are used, and early stopping is enforced if the validation loss fails to decrease over 500 epochs. A mini-batch size of $b = 8$ is used for training and the mean squared error loss function is used. The Adam optimizer is used with an initial learning rate of $\eta = 3 \times 10^{-4}$. Each layer has its weights initialized with the He normal initializer. All of the layers with the exception of the output use a leaky ReLU activation function with $\alpha = 0.25$. A performance comparison with other activation functions is given in Appendix. The output layer uses the sigmoid activation function, which scales into the range [0,1], ensuring that the outputs can be scaled back to their original range for prediction

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{29}$$

As the FOM is inexpensive to solve, offline computational costs related to simulation and POD-GPR are not reported as well as all online costs. Training and validation losses against the number of epochs at selected folds of the data for different values of k are presented in Appendix in addition to computational costs.

7.1 | Steady incompressible Navier–Stokes equations

Steady incompressible laminar flow is simulated using simpleFoam, a standard OpenFOAM solver, by solving the Navier–Stokes equations,

$$\int_S \vec{U} \cdot \mathrm{d}\vec{S} = 0, \tag{30}$$

$$\int_S \vec{U}\vec{U} \cdot \mathrm{d}\vec{S} + \int_V \nabla p \mathrm{d}V - \nu \int_S (\nabla \vec{U} + \nabla \vec{U}^T) \cdot \mathrm{d}\vec{S} = 0, \tag{31}$$

where $\vec{U} = [u, v]$ is the velocity vector and u and v are the velocity components in the x and y directions respectively, $\vec{S}$ is the face-area vector, V is the volume; ν is the kinematic viscosity, and p is the pressure. The continuity and momentum equations are discretized over the computational domain by using the finite-volume method (FVM). Both equations are

coupled through the semi-implicit method for pressure-linked equations (SIMPLE) algorithm[51] along with Rhie–Chow interpolation.[52] The SIMPLE algorithm is iteratively repeated until a residual tolerance of 1×10^{-6} is reached for both $\vec{U}$ and p.

7.2 | Lid-driven cavity problem

The numerical example used in this work is a physically and geometrically parameterized lid-driven cavity flow, a popular benchmark problem for CFD solvers. Three parameters control the computational domain Ω and one parameter controls the kinematic viscosity through the Reynolds number. A version of this problem has previously appeared in a work by Hesthaven and Ubbiali.[9] Figure 5 shows the boundary conditions on each edge $\Gamma_i, i \in [1, 2, 3, 4]$ of the domain; $u, v = 0$ on all of the edges except Γ_1, where $u = 1, v = 0$. The pressure gradient, ∇p, is set to 0 on all of the edges. The reference pressure is set to 0 on the bottom left corner of the domain. The parameterization of the geometry is also shown, involving three parameters which change the length of the horizontal (μ_1) and slanting edges (μ_2) as well as the slanting angle (μ_3). The Reynolds number, Re (μ_4), is the fourth parameter, and is related to the kinematic viscosity ν as

$$Re = \frac{\max(\mu_1, \mu_2)}{\nu(\boldsymbol{\mu})}. \tag{32}$$

The design parameter combinations are generated using Latin hypercube sampling with the following bounds for each parameter

$$\mu_1 \in [1, 2],$$
$$\mu_2 \in [1, 2],$$
$$\mu_3 \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right],$$
$$\mu_4 \in [100, 600].$$

The computational mesh consists of 64×64 cells uniformly distributed in the x and y directions and one cell spanning the z direction, resulting in $N = 4096$ and a reshape operator $\boldsymbol{R}$ with $n_y, n_x = 64$ and $n_c = 2$. The full-order states of u and v are used to compare the performance of the POD-GPR and CAE-GPR methods, contours of which are shown in Figure 6 at three different sets of design parameters. A sharp gradient in u exists at the top of the domain, and a vortex moves throughout the cavity as the design parameters change. This vortex is also shown moving throughout the cavity shown in the contours of v, varying in shape and size with the design parameters. The relationship between both u and v and $\boldsymbol{\mu}$ is shown to be highly nonlinear, making this a difficult prediction problem in the context of ROMs.
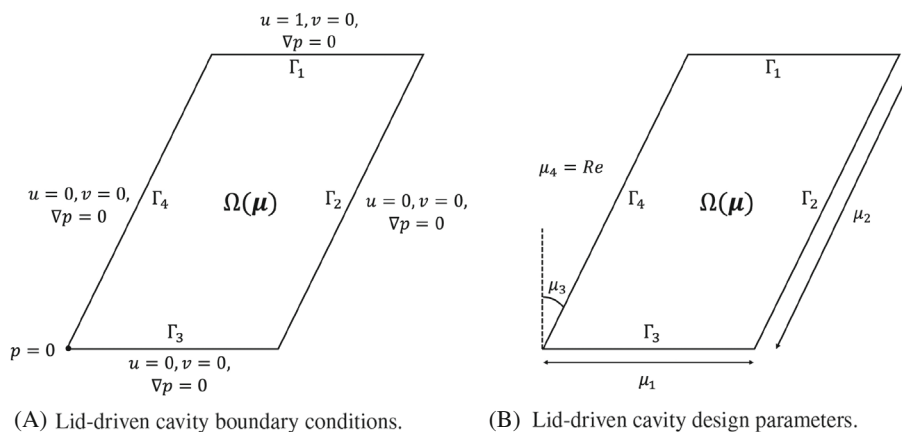

(A) Lid-driven cavity boundary conditions.   (B) Lid-driven cavity design parameters.

FIGURE 5   Schematics describing the lid-driven cavity problem

(A) $\mu = (1.5, \frac{3}{\sqrt{3}}, -\frac{\pi}{6}, 200)$ (B) $\mu = (1.5, 1.5, 0, 300)$ (C) $\mu = (1.5, \frac{3}{\sqrt{3}}, \frac{\pi}{6}, 400)$

(D) $\mu = (1.5, \frac{3}{\sqrt{3}}, -\frac{\pi}{6}, 200)$ (E) $\mu = (1.5, 1.5, 0, 300)$ (F) $\mu = (1.5, \frac{3}{\sqrt{3}}, \frac{\pi}{6}, 400)$
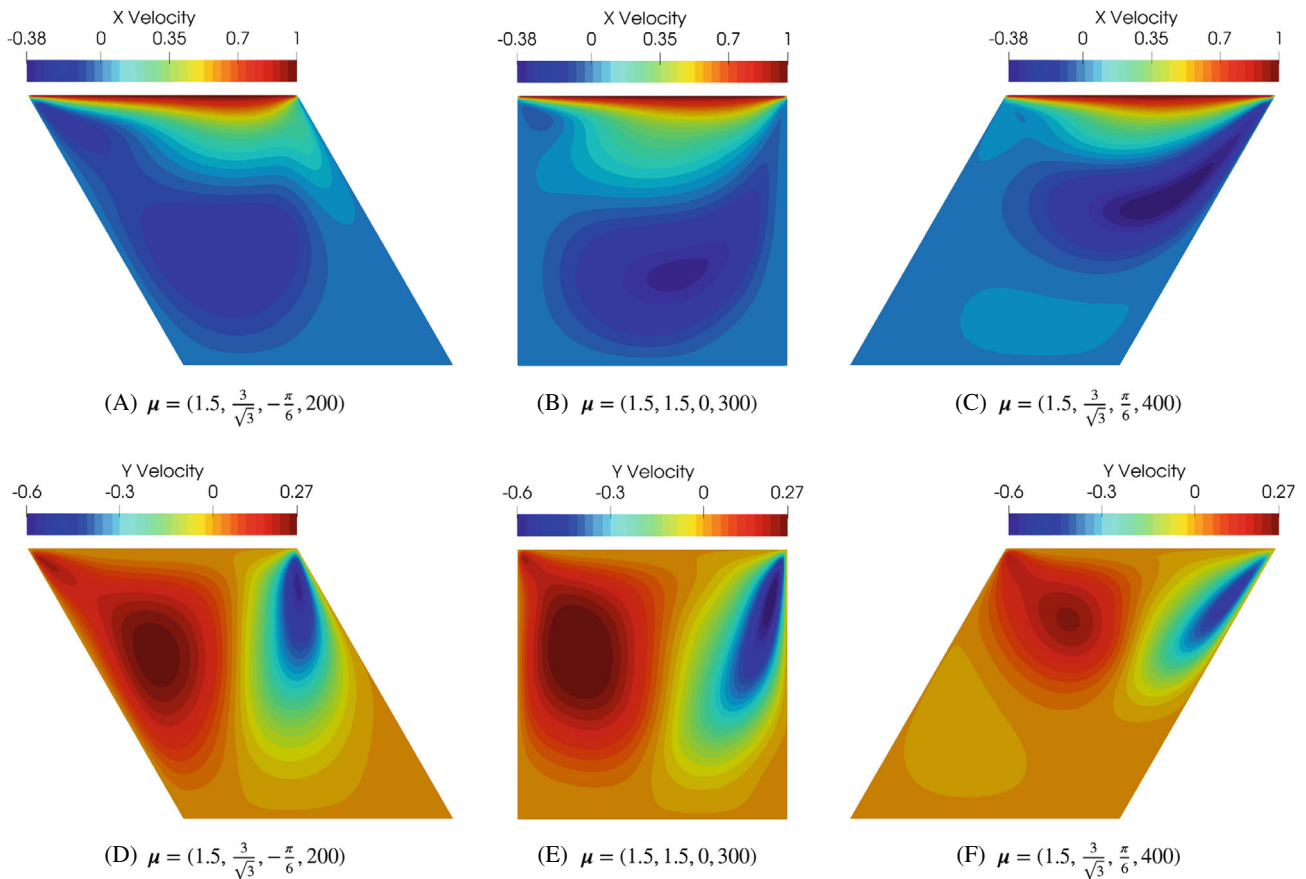
FIGURE 6 Contours of u (top) and v (bottom) for the lid-driven cavity problem at three different sets of design parameters.
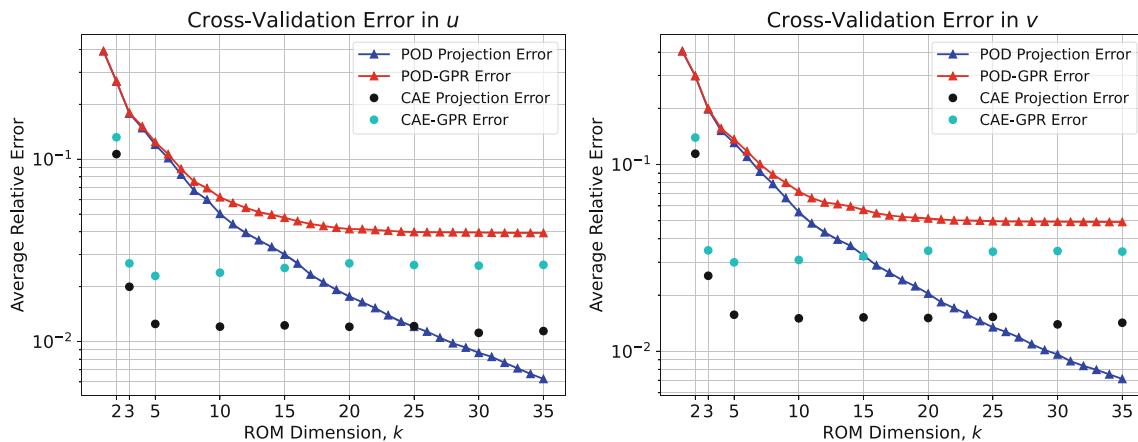

FIGURE 7 Plots of the cross-validation prediction and projection errors in u and v for both ROMs at different values of k.

The POD-GPR ROM is evaluated for both projection and prediction errors at ROM dimensions $k \in [1, 2, \ldots 35]$ while the CAE-GPR ROM is similarly evaluated at $k \in [2, 3, 5, 10, \ldots 35]$. Figure 7 shows the cross-validation relative errors for u and v; for evaluated ROM dimensions $k > 2$, the CAE-GPR ROM exhibits higher predictive performance over the data set. The projection error provided by the CAE is high at $k = 2$ and comparable to that of POD, and it decays rapidly until $k = 5$, after which it does not vary much. Similar results are presented in a work by Lee and Carlberg.[17] Very low values of k decrease the capacity of the network to learn meaningful low-dimensional representations of the training data. As k increases, the relative difference in the projection and prediction errors grows. This is an expected result since
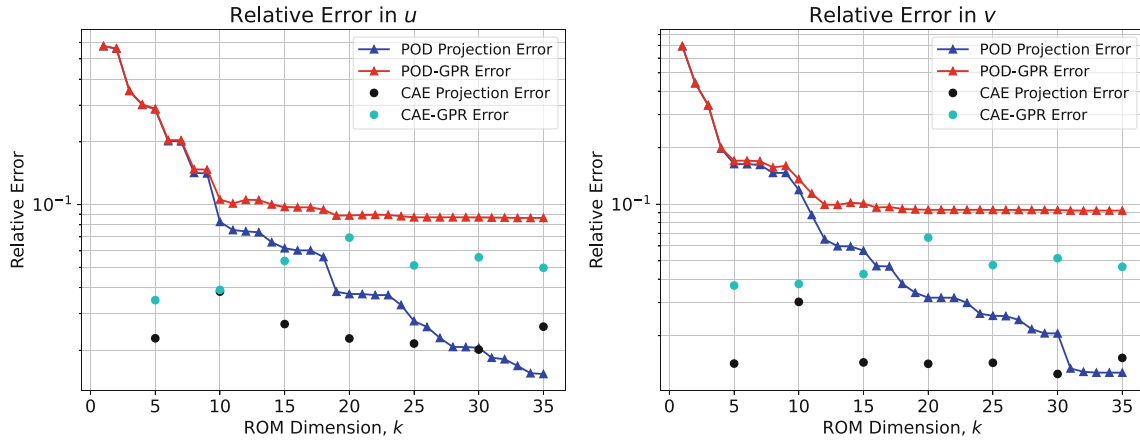
FIGURE 8 Plots of the prediction and projection errors in u and v for both ROMs at $\boldsymbol{\mu} = (1.167, 1.997, -0.4665, 555.5)$ at different values of k.

(A) $\boldsymbol{u}$, Ground Truth

(B) $\boldsymbol{u}$, POD-GPR Difference

(C) $\boldsymbol{u}$, CAE-GPR Difference

(D) $\boldsymbol{v}$, Ground Truth

(E) $\boldsymbol{v}$, POD-GPR Difference
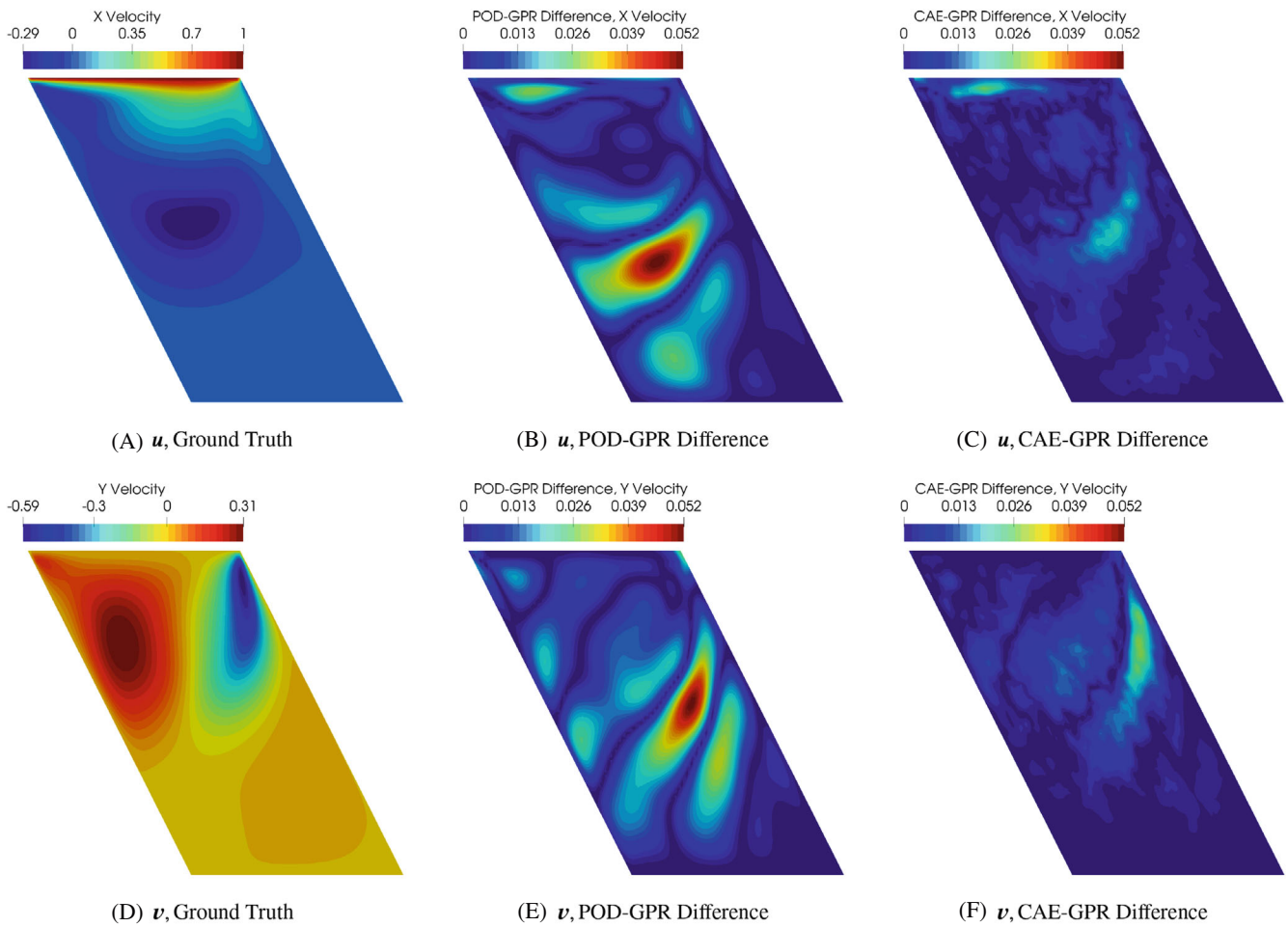
(F) $\boldsymbol{v}$, CAE-GPR Difference

FIGURE 9 ROM comparison of u and v at $\boldsymbol{\mu} = (1.167, 1.997, -0.4665, 555.5)$, with $k = 5$ for CAE-GPR and $k = 35$ for POD-GPR.
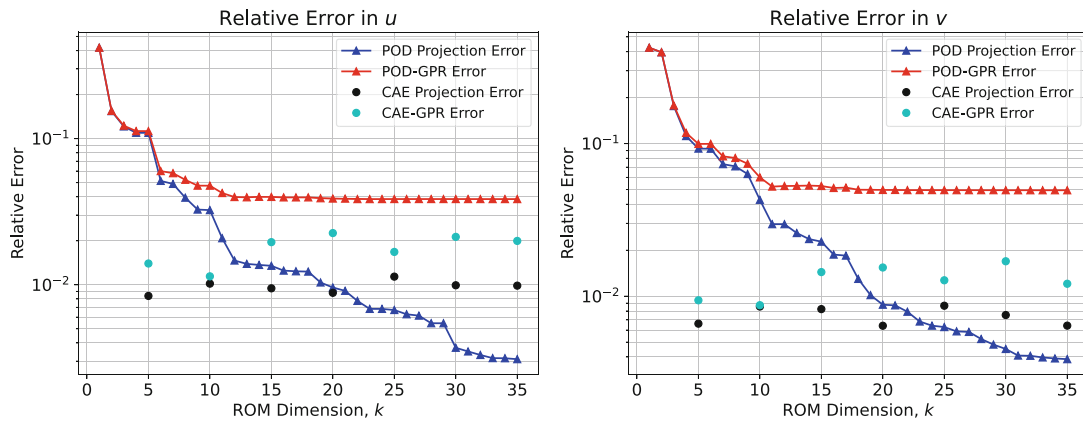
FIGURE 10 Plots of the prediction and projection errors in u and v for both ROMs at $\boldsymbol{\mu} = (1.963, 1.789, 0.5890, 308.5)$ at different values of k.
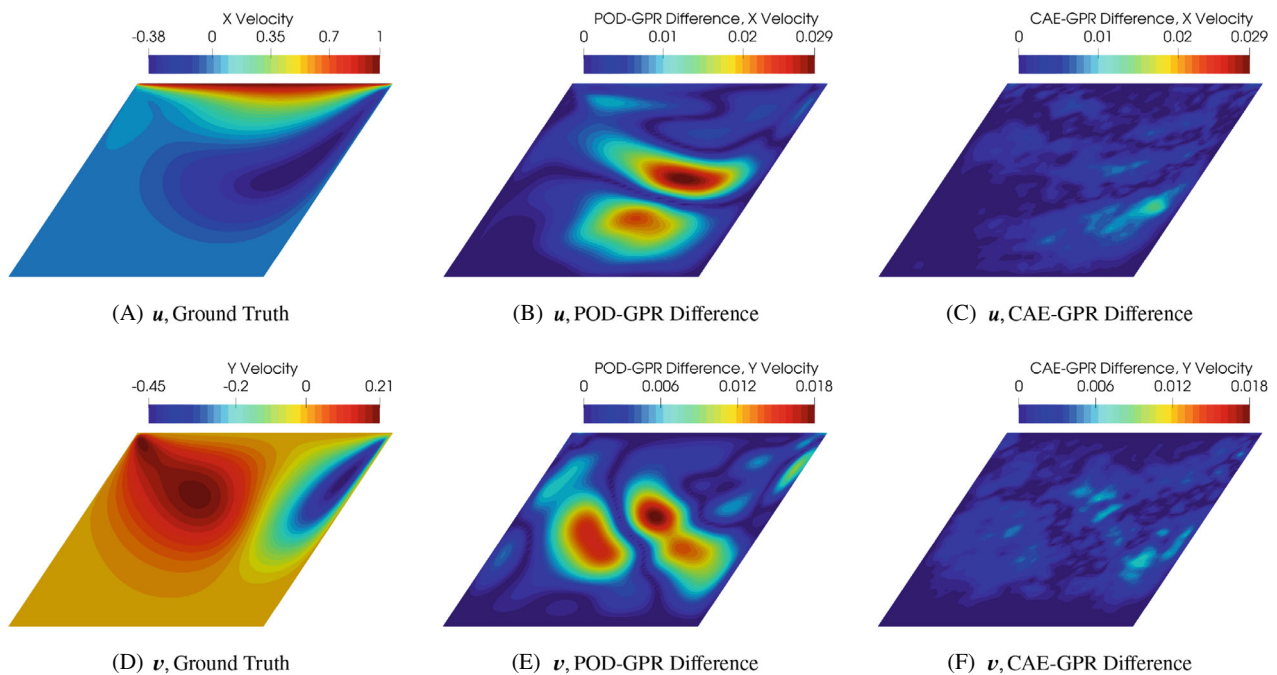

FIGURE 11 ROM comparison of u and v at $\boldsymbol{\mu} = (1.963, 1.789, 0.5890, 308.5)$, with $k = 5$ for CAE-GPR and $k = 35$ for POD-GPR.

prediction errors in the individual expansion coefficients will have a cascading effect. The cross-validation projection error from POD continues to decay after $k = 35$, while the POD-GPR prediction error flattens out at around $k = 20$. The cross-validation projection error produced by the CAE for both u and v is lower than that of POD until around $k = 25$; even with a higher projection error, the predictive performance offered by CAE-GPR exceeds that of POD-GPR for $k \geq 25$. The CAE offers a set of expansion coefficients that are more easily interpolated when using GPR compared to POD, which sees its predictive performance stall after k reaches a certain value, a commonly found result for POD-GPR based ROMs.[21] In addition to giving better performance in terms of both projection and prediction for low values of k, the use of a nonlinear trial manifold for ROM construction offers a more robust relationship between the design parameters and expansion coefficients. The difference in projection and prediction errors from POD-GPR is almost 0 at low values of k, but rapidly increases as k grows, suggesting that the individual expansion coefficients become harder to interpolate as the corresponding singular values decay.

Figure 8 shows the relative error plot for both ROMs at $k \in [5, 10, \ldots 35]$ for a single design parameter $\boldsymbol{\mu}_1 = (1.167, 1.997, -0.4665, 555.5)$, while Figure 9 shows the contour plots of the FOM as well as the absolute error

plots for CAE-GPR at $k = 5$ and POD-GPR at $k = 35$. Results at another design parameter instance at $\boldsymbol{\mu}_2 = (1.963, 1.789, 0.5890, 308.5)$ are shown in Figures 10 and 11. The generalized results from the cross-validation also hold here; for a greater projection error, CAE-GPR provides a lower prediction error. It is also shown at these design parameters that the prediction error curve of POD-GPR flattens out. There is more volatility in both the projection and prediction errors for CAE-GPR, although POD-GPR still never outperforms it in predicting u and v. For $k = 10$ at both design parameters, the difference in the projection and prediction errors is very small, and almost 0 for u at $\boldsymbol{\mu}_1$ and v at $\boldsymbol{\mu}_2$. This is similar to the behavior exhibited by POD-GPR for low values of k, showing that CAE-GPR is also capable of producing highly accurate estimates of the expansion coefficients. The error contours at these design parameters highlight the increased predictive performance given by CAE-GPR over POD-GPR. While the error contours given by POD-GPR exhibit distinct bands of high error, the contours produced by CAE-GPR are generally more uniform and dispersed throughout the domain. At the chosen design parameters, there is a significant decrease in relative error; at $\boldsymbol{\mu}_1$, the percent decreases in relative error of u and v from POD-GPR to CAE-GPR are 42.4% and 49.8% respectively, while at $\boldsymbol{\mu}_2$ they are 48.3% and 75.6% respectively.

8 | CONCLUSION

This work presents a non-intrusive reduced-order model framework utilizing nonlinear trial manifolds through the use of convolutional autoencoders. A deep learning approach, CAEs learn efficient low-dimensional representations of data through an encoder and decoder connected by a code. CAEs provide a set of expansion coefficients through the low-dimensional code, similar to the basis coefficients provided by POD-based methods. A nonlinear relationship exists between the expansion coefficients and full-order states when using autoencoders, in contrast with the linear relationship when using POD. Given steady-state solutions of PDEs parameterized by a set of design parameters, GPR can be used to approximate the expansion coefficients at unseen points in the design space for both approaches (referred to as CAE-GPR and POD-GPR). CAE-GPR involves a more expensive offline stage due to the high computational cost associated with training deep neural networks and requires that solutions have appropriate spatial arrangement.

When applied to a geometrically and physically parameterized lid-driven cavity problem solved using the steady incompressible Navier–Stokes equations, it is shown that CAE-GPR offers higher performance in predicting the components of the velocity field when compared to POD-GPR over a range of ROM dimension. The low-dimensional code provided by CAEs is shown to be more easily interpolated than the basis coefficients obtained from POD. For a greater manifold projection error, CAE-GPR provides lower error in predictions of full-order states. It is also shown that CAE-GPR has the ability to provide highly accurate estimates of the expansion coefficients, providing prediction errors that are very close to projection errors. Although other works[17,53,54] have shown that CAEs outperform POD for some ROMs, the analyses presented compare performance at only a small number of unseen parameter instances or only show improvement for low-dimensional ROMs. In this work, we have provided a comprehensive cross-validation analysis over a large number of ROM dimensions which shows that using CAEs can offer significantly better performance than POD for non-intrusive ROMs. A similar work[19] using CAEs for non-intrusive ROMs applied to external vehicle aerodynamics has shown only a very small improvement in performance compared to POD; problems that are not highly non-linear like the one presented in this work may not benefit from using deep learning for ROMs. Future work will extend this ROM framework to larger problems where unstructured meshes are required and investigate constructing nonlinear trial manifolds using variational autoencoders (VAEs), which have been shown to provide a more interpretable low-dimensional code.

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

ORCID
Rakesh Halder https://orcid.org/0000-0003-2540-9411
Krzysztof J. Fidkowski https://orcid.org/0000-0002-5106-136X

REFERENCES

1. Berkooz G, Holmes PJ, John L. The proper orthogonal decomposition in the analysis of turbulent flows. *Annu Rev Fluid Mech*. 2003;25:539-575. doi:10.1146/annurev.fl.25.010193.002543
2. Holmes P, Lumley JL, Berkooz G, Mattingly JC, Wittenberg RW. Low-dimensional models of coherent structures in turbulence. *Phys Rep*. 1997;287:337-384.
3. Carlberg K, Bou-Mosleh C, Farhat C. Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. *Int J Numer Methods Eng*. 2011;86(2):155-181.
4. Carlberg K, Barone M, Antil H. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. *J Comput Phys*. 2017;330:693-734. doi:10.1016/j.jcp.2016.10.033
5. Carlberg K. Adaptive H-refinement for reduced-order models. *Int J Numer Methods Eng*. 2015;102(5):1192-1210.
6. Chaturantabut S, Sorensen DC. Nonlinear model reduction via discrete empirical interpolation. *SIAM J Sci Comput*. 2010;32(5): 2737-2764.
7. Guo M, Hesthaven JS. Reduced order modeling for nonlinear structural analysis using Gaussian process regression. *Comput Methods Appl Mech Eng*. 2018;341:807-826.
8. Dupuis R, Jouhaud J-C, Sagaut P. Surrogate modeling of aerodynamic simulations for multiple operating conditions using machine learning. *AIAA J*. 2018;56(9):3622-3635. doi:10.2514/1.J056405
9. Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *J Comput Phys*. 2018;363:55-78. doi:10.1016/j.jcp.2018.02.037
10. Jacquier P, Abdedou A, Delmas V, Soulaimani A. Non-intrusive reduced-order modeling using uncertainty-aware deep neural networks and proper orthogonal decomposition: application to flood modeling. *J Comput Phys*. 2021;424:109854. doi:10.1016/j.jcp.2020.109854
11. Ohlberger M, Rave S. Reduced basis methods: success, limitations and future challenges. Proceedings of the Conference Algoritmy; 2016.
12. Amsallem D, Zahr MJ, Farhat C. Nonlinear model order reduction based on local reduced-order bases. *Int J Numer Methods Eng*. 2012;92(10):891-916.
13. Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. *Adv Neural Inf Process Syst*. 2012;25:1097-1105.
14. Hirschberg J, Manning CD. Advances in natural language processing. *Science*. 2015;349(6245):261-266.
15. Zhang S, Yao L, Sun A, Tay Y. Deep learning based recommender system: a survey and new perspectives. *ACM Comput Surv (CSUR)*. 2019;52(1):1-38.
16. LeCun Y, Bengio Y, Hinton G. Deep learning. *Nature*. 2015;521(7553):436-444. doi:10.1038/nature14539
17. Lee K, Carlberg K. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J Comput Phys*. 2020;404:108973.
18. Fresca S, Dede' L, Manzoni A. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *J Sci Comput*. 2021;87:61.
19. Markus M, Othmer C, Radespiel R. Variational autoencoders for model order reduction in vehicle aerodynamics; 2021:3049.
20. Kadeethum T, Ballarin F, Choi Y, O'Malley D, Yoon H, Bouklas N. Non-intrusive reduced order modeling of natural convection in porous media using convolutional autoencoders: comparison with linear subspace techniques. *Adv Water Resour*. 2022;160:104098.
21. Mrosek M, Othmer C, Radespiel R. Reduced-order modeling of vehicle aerodynamics via proper orthogonal decomposition. *SAE Int J Passenger Cars - Mech Syst*. 2019;12(3):225-236. doi:10.4271/06‐12‐03‐0016
22. Eckart C, Young G. The approximation of one matrix by another of lower rank. *Psychometrika*. 1936;1:211-218.
23. Tenenbaum JB, De Silva V, Langford JC. A global geometric framework for nonlinear dimensionality reduction. *Science*. 2000;290(5500):2319-2323.
24. Lee G, Rodriguez C, Madabhushi A. Investigating the efficacy of nonlinear dimensionality reduction schemes in classifying gene and protein expression studies. *IEEE/ACM Trans Comput Biol Bioinform*. 2008;5(3):368-384.
25. Gonzalez FJ, Balajewicz M. Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems. arXiv preprint arXiv:1808.01346, 2018.
26. Kriesel David *A Brief Introduction to Neural Networks*. 2007. https://www.dkriesel.com/en/science/neural_networks
27. Nair V, Hinton GE. Rectified linear units improve restricted Boltzmann machines; 2010.
28. Dahl GE, Sainath TN, Hinton GE. Improving deep neural networks for LVCSR using rectified linear units and dropout; 2013: 8609-8613.
29. Zeiler MD, Ranzato M, Monga R, et al. On rectified linear units for speech processing. Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing; 2013:3517-3521.
30. Xu B, Wang N, Chen T, Li M. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853, 2015.
31. Rumelhart DE, Hinton GE, Williams RJ. Learning representations by back-propagating errors. *Nature*. 1986;323(6088):533-536. doi:10.1038/323533a0
32. Kingma DP, Ba J. Adam: a method for stochastic optimization. Proceedings of the 3rd International Conference for Learning Representations; 2015; San Diego.
33. Masters D, Luschi C. Revisiting small batch training for deep neural networks. arXiv preprint arXiv:1804.07612, 2018.

34. Ying X. An overview of overfitting and its solutions. *J Phys Conf Ser*. 2019;1168:022022. doi:10.1088/1742‐6596/1168/2/022022

35. He Kaiming, Zhang Xiangyu, Ren Shaoqing, Sun Jian. Delving deep into rectifiers: surpassing human-level performance on imagenet classification; 2015:1026-1034.

36. Telgarsky M. Benefits of depth in neural networks; 2016:1517-1539; PMLR.

37. Eldan R, Shamir O. The power of depth for feedforward neural networks. In: Feldman V, Rakhlin A, Shamir O, eds. *Proceedings of the 29th Annual Conference on Learning Theory, Proceedings of Machine Learning Research*. Vol 49. Columbia University; 2016: 907-940.

38. Sola J, Sevilla J. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Trans Nucl Sci*. 1997;44:1464-1468.

39. Hinton GE, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. *Science*. 2006;313(5786):504-507. doi:10.1126/science.1127647

40. LeCun Y, Haffner P, Bottou L, Bengio Y. Object recognition with gradient-based learning; 1999:319-345; Springer.

41. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

42. Dumoulin V, Visin F. A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285, 2016.

43. Lecun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. *Proc IEEE*. 1998;86(11):2278-2324. doi:10.1109/5.726791

44. Goodfellow I, Bengio Y, Courville A. *Deep Learning*. MIT Press; 2016 http://www.deeplearningbook.org

45. Krige D. *A Statistical Approach to Some Mine Valuation and Allied Problems on the Witwatersrand*. PhD thesis. University of the Witwatersrand; 1951.

46. Rasmussen CE. Gaussian processes in machine learning; 2003:63-71; Springer.

47. Weller HG, Tabor G, Jasak H, Fureby C. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Comput Phys*. 1998;12(6):620-631.

48. Abadi M, Agarwal A, Barham P, et al. TensorFlow: large-scale machine learning on heterogeneous systems; 2015. tensorflow.org.

49. Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: machine learning in python. *J Mach Learn Res*. 2011;12:2825-2830.

50. Jin R, Chen W, Sudjianto A. An efficient algorithm for constructing optimal design of computer experiments. *J Stat Plan Infer*. 2005;134(1):268-287. doi:10.1016/j.jspi.2004.02.014

51. Patankar SV, Spalding DB. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *Int J Heat Mass Transf*. 1972;15(10):1787-1806. doi:10.1016/0017‐9310(72)90054‐3

52. Rhie CM, Chow WL. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA J*. 1983;21(11):1525-1532. doi:10.2514/3.8284

53. Xu J, Duraisamy K. Multi-level convolutional autoencoder networks for parametric prediction of spatio-temporal dynamics. *Comput Methods Appl Mech Eng*. 2020;372:113379.

54. Maulik R, Lusch B, Balaprakash P. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Phys Fluids*. 2021;33(3):037106.

APPENDIX A. CONVOLUTIONAL AUTOENCODER ARCHITECTURE

The CAE architecture used for the lid-driven cavity problem ROM is listed in Table A1. Zero padding is used for all convolutional and max-pooling layers. All leaky ReLU activation functions use a value of $\alpha = 0.25$. There are a relatively small number of convolutional and pooling layers in the network; we found that adding more of them did not improve the network performance, although their absence (using an MLP) causes a large decrease in performance. Compared to images which can be very noisy, the physical states in the presented problem vary smoothly, and fewer convolutional layers are required to learn features. We also found that having a fully connected layer on either side of the code was important for network performance, although this does drastically increase the number of network parameters. This makes the use of early stopping as a regularization method important to ensure that the network does not overfit. The learning task at

TABLE A1 Detailed convolutional autoencoder (CAE) architecture used for the lid-driven cavity problem ROM.

Layer	Number of filters	Kernel size	Stride value	Activation function	Size of output
Input					$64 \times 64 \times 2$
Convolutional	64	3×3	1×1	Leaky ReLU	$64 \times 64 \times 64$
Max-pooling		2×2	2×2		$32 \times 32 \times 64$
Convolutional	32	3×3	1×1	Leaky ReLU	$32 \times 32 \times 32$
Max-pooling		2×2	2×2		$16 \times 16 \times 32$
Reshape					8192
Fully connected				Leaky ReLU	128
Code				Leaky ReLU	k
Fully connected				Leaky ReLU	128
Fully connected				Leaky ReLU	8192
Reshape					$16 \times 16 \times 32$
Convolutional transpose	32	3×3	2×2	Leaky ReLU	$32 \times 32 \times 32$
Convolutional transpose	64	3×3	2×2	Leaky ReLU	$64 \times 64 \times 64$
Convolutional transpose	2	3×3	1×1	Sigmoid	$64 \times 64 \times 2$

hand requires that reconstructions of data be highly accurate, and networks with more parameters allow for more robust functional relationships to arise.

APPENDIX B. CONVOLUTIONAL AUTOENCODER TRAINING

Figure B1 shows the training and validation losses against the number of epochs for selected folds of the training and validation data for $k = 5, 10, 25, 30$. Early stopping is used as a regularization method, and the validation loss fails to drop for 500 epochs well before the maximum number of 7500 epochs at $k = 5, 10, 25$. At $k = 30$, training stops after 7491 epochs. While the training loss continues to decline slowly in all of the plots, the validation loss shows asymptotic behavior. By monitoring the validation loss and using early stopping, the network is prevented from overfitting the training data. Training is performed on an NVIDIA TITAN RTX GPU. The average wall time and number of epochs for training the CAE over all of the data folds is shown in Table B1; in general, increasing k leads to higher computational costs. Both the number of trainable parameters and capacity of the network to learn are affected by the size of the code.

TABLE B1 Average computational costs over all data folds for training the CAE

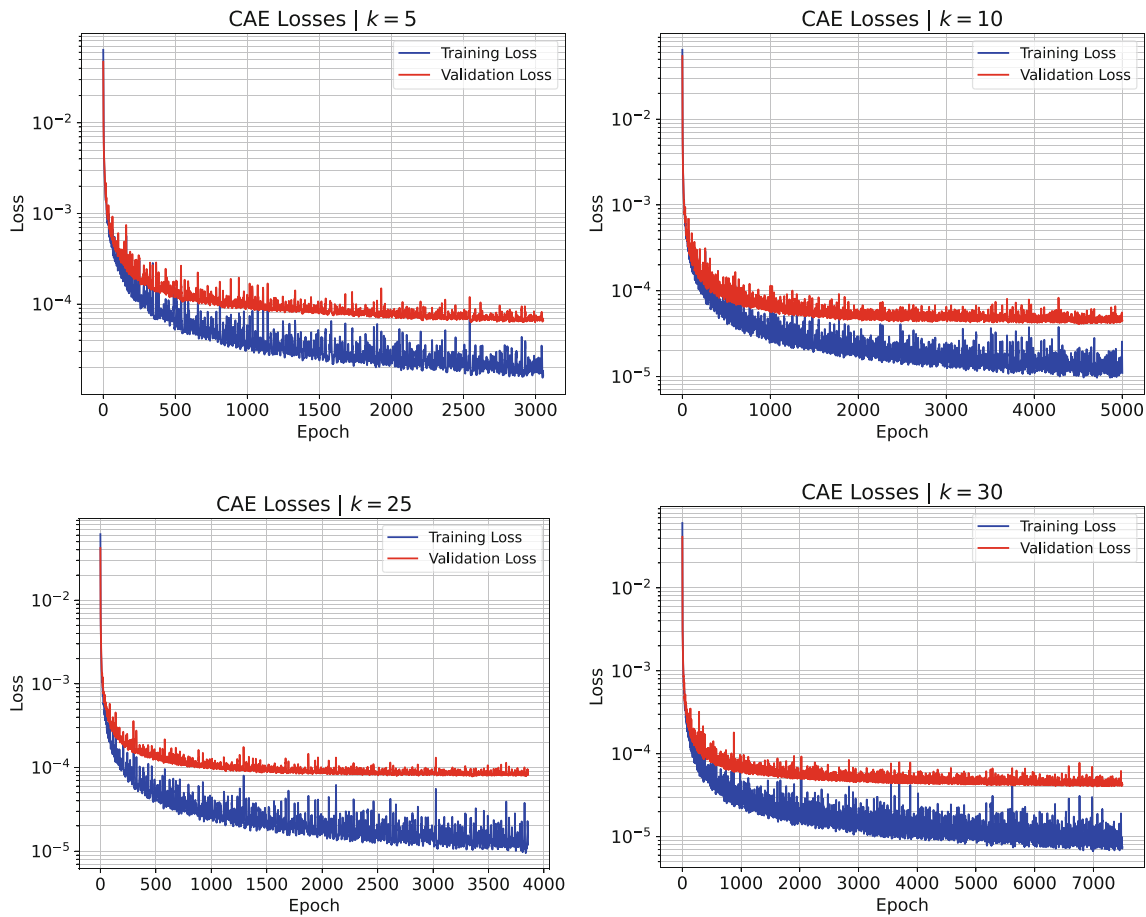| ROM dimension, k	Average wall time (s)	Average number of epochs
5	884	3533
10	883	3564
15	867	3441
20	1123	4465
25	971	3865
30	1283	4675
35	1301	4519

F I G U R E B1   Plots of the training and validation losses at different ROM dimensions k for a selected fold of the training and validation data.

APPENDIX C. ACTIVATION FUNCTION PERFORMANCE

In addition to the leaky ReLU activation function, we also present the projection errors obtained when using the ReLU activation function as well as the tanh activation function, another popular choice for neural networks which is given as

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{C1}$$

This is done by replacing all of the leaky ReLU activation functions ($\alpha = 0.25$) in the network in Table A1 with either ReLU or tanh and following the same network training procedure. Figure C1 shows the cross-validation projection errors obtained using networks with the three different activation functions for ROM dimensions $k \in [5, 10, \ldots 35]$ and POD. It is shown that the network using the ReLU activation function is outperformed by POD for the chosen values of k, and that leaky ReLU offers slightly improved performance over tanh in reconstructing u and v. Using the leaky ReLU activation function overcomes the dying ReLU problem, which is shown to lead to networks with very poor predictive performance. The initial randomization of the network weights and biases may lead to poor performance when using the ReLU activation function; for $k = 15$, the maximum value of ϵ_{Proj} for u at a single test fold is 4.228×10^{-1}, while the second highest value is 2.372×10^{-2}, which is similar to the errors provided by using leaky ReLU or tanh. The poor performance given by using ReLU is not restricted to single folds of the data either; for a specific fold over the selected k, ϵ_{Proj} for u has a maximum of 4.233×10^{-1} at $k = 5$ and minimum of 2.1028×10^{-2} at $k = 25$. This implies that using the ReLU activation function leads to network performance being sensitive to the initialization of the weights and biases, in contrast to using tanh or leaky ReLU which offer more consistent performance.
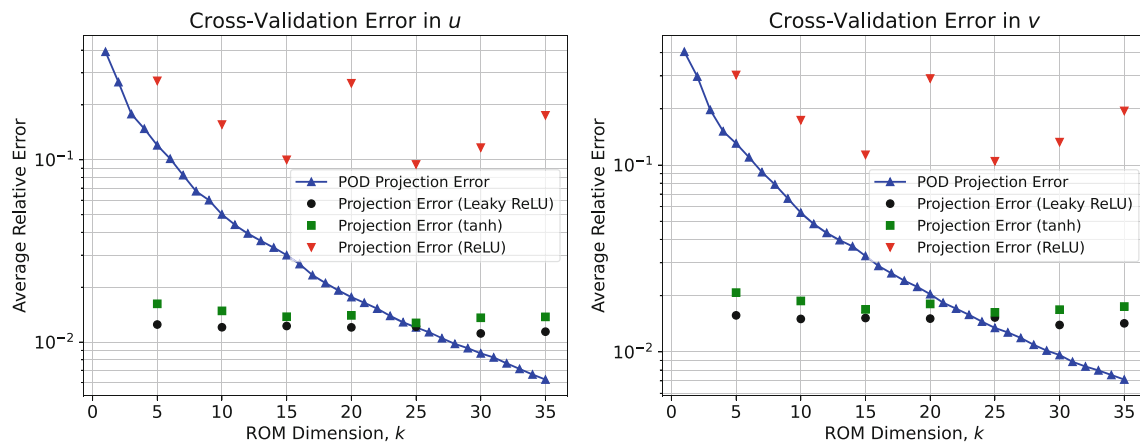
FIGURE C1 Plots of the cross-validation projection errors in u and v for POD using different activation functions.