Journal of Computational Physics 521 (2025) 113526

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Polytopic autoencoders with smooth clustering for reduced-order modeling of flows

Jan Heiland [a,b], Yongho Kim [a,b,*]

[a] Department of Mathematics, Otto-von-Guericke University Magdeburg, Universitätsplatz 2, 39106, Magdeburg, Germany
[b] Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, 39106, Magdeburg, Germany

ARTICLE INFO

Dataset link: https://doi.org/10.5281/zenodo.10491870

MSC:
65F45
68T07

Keywords:
Convolutional autoencoders
Clustering
Convex polytope
Model order reduction
Linear parameter-varying (LPV) systems
Polytopic LPV system

ABSTRACT

With the advancement of neural networks, there has been a notable increase, both in terms of quantity and variety, in research publications concerning the application of autoencoders to reduced-order models. We propose a polytopic autoencoder architecture that includes a lightweight nonlinear encoder, a convex combination decoder, and a smooth clustering network. Supported by several proofs, the model architecture ensures that all reconstructed states lie within a polytope, accompanied by a metric indicating the quality of the constructed polytopes, referred to as polytope error. Additionally, it offers a minimal number of convex coordinates for polytopic linear-parameter varying systems while achieving acceptable reconstruction errors compared to proper orthogonal decomposition (POD). To validate our proposed model, we conduct simulations involving two flow scenarios with the incompressible Navier-Stokes equation. Numerical results demonstrate the guaranteed properties of the model, low reconstruction errors compared to POD, and the improvement in error using a clustering network.

1. Introduction

The solution of high-dimensional dynamical systems of the form

$$\dot{\mathbf{v}}(t) = f(\mathbf{v}(t)), \quad \text{with } \mathbf{v}(t) \in \mathbb{R}^n, \quad \text{for time } t > 0, \tag{1}$$

in simulations often demands substantial computational resources and even becomes infeasible due to hardware constraints. In response to these challenges, researchers have used model order reduction methods in diverse fields such as engineering, medicine, and chemistry (e.g., [1–5]). The promise and procedure of model order reduction is to design a model or reduced dimension to enhance computational efficiency while maintaining a desired level of accuracy.

Proper Orthogonal Decomposition (POD) [6], a classical model order reduction method, has gained wide acceptance due to its linearity, its optimality (as a linear projection for given measurements of the state), and the advantages of the POD modes as an orthogonal basis derived from a *truncated singular value decomposition* (e.g., [7]) of given data. POD provides the reduced coordinates in a low-dimensional space through a linear projection

$$\mathbf{v}_r(t) = \mathbf{V}\mathbf{v}(t)$$

* Corresponding author at: Max Planck Institute for Dynamics of Complex Technical Systems, Sandtorstraße 1, 39106, Magdeburg, Germany.
E-mail addresses: heiland@mpi-magdeburg.mpg.de (J. Heiland), ykim@mpi-magdeburg.mpg.de (Y. Kim).

https://doi.org/10.1016/j.jcp.2024.113526
Received 22 January 2024; Received in revised form 17 September 2024; Accepted 20 October 2024

Available online 23 October 2024

and reconstructs the states using a linear lifting

$$\tilde{\mathbf{v}}(t) = \mathbf{V}^\top \mathbf{v}_r(t),$$

where $\mathbf{V} \in \mathbb{R}^{r \times n}$ is the matrix including the so-called *leading r* POD modes, where $\mathbf{v}(t)$ is an n dimensional state, where $\mathbf{v}_r(t) \in \mathbb{R}^r$ are the reduced coordinates (by virtue of $r \ll n$), and where $\tilde{\mathbf{v}}(t)$ is the reconstructed state that approximates $\mathbf{v}(t)$.

The linearity of the POD comes with many algorithmic advantages but also means a natural limitation in terms of accuracy versus reduction potential which is commonly known as the *Kolmogorov n-width* [8]. For our intended application of very low-dimensional approximations, one, therefore, has to resort to nonlinear model reduction schemes.

In this realm, general *autoencoders* (e.g., [9]) have been widely used either in conjunction with POD or as a substitute for POD, driven by two main reasons. Firstly, the universality: autoencoders can be constructed in various architectures and contexts such as convolutional autoencoders, denoising autoencoders, and variational autoencoders; e.g., [10, Ch. 14] or [11]. Secondly, the efficiency: the training of autoencoders is a classical machine learning tasks and, thus, comes with state-of-the-art implementations in all machine learning toolboxes. Accordingly, the research reports on autoencoders for reduced-order dynamical systems have reached a significant amount both in numbers and diversity in recent years; e.g., [12–19].

Particularly, convolutional autoencoders for model order reduction have been developed due to the translation invariance and sparse connectivity of convolutions; e.g., [15–17,19]. For these reasons and what has been reported so far, we choose to utilize convolutional autoencoders in this paper.

As another attempt to overcome the limits of linear model order reduction schemes, one approach is to use local POD bases for state reconstruction on subregions of the given data set; cp. e.g., [20–23]. To define the subsets, often referred to as *clusters*, one commonly refers to clustering methods such as *k-means clustering* [24] and fuzzy c-means clustering [25] and problem-specific measures.

In a previous work [26], we have considered the direct combination of the two approaches – clustering and separate, i.e., local autoencoders for each cluster. The result was a highly-performant autoencoder for very low-dimensional parametrizations of incompressible flows. However, the involved identification of the local bases is a highly nonlinear and even noncontinuous process and, thus, not suited for use in dynamical systems.

In the present work, following up on reported efforts in developing differential clustering algorithms (e.g., [27]), we propose a fully differentiable clustering network suitable for large-scale dimensional systems. In addition, since the model including the clustering network is fully differentiable, the clustering model parameters and other model parameters can be trained simultaneously through a joint loss function; e.g., [28,29,27].

Another target of the presented research is motivated from the fact that, typically, the states of dynamical systems are confined to a bounded subset of the state space which is not contradicting but in a sense dismissing the premise of linear model order reduction that the states reside in a linear space. This additional feature is like naturally ensured in our proposed architecture by defining the reconstruction as a convex combination of supporting vectors associated with the smoothly selected clusters. Furthermore, with standard tools of neural networks and with identifying *pseudo-labels* for the training, we can control the number of involved supporting vectors which translates into very low-dimensional local bases.

As an intended side effect, we reason that the convex combination-based decoding readily defines an affine parametrization within a polytope as it can be exploited for efficient nonlinear controller design in the context of *linear parameter-varying* (LPV) systems; see [30] for robust controller design for polytopic LPV systems, and for applications, refer to [31–34].

The paper is structured as follows: In Section 2, we introduce the motivation and basic ideas for the application of autoencoders, convex polytopes, and clustering. In Section 3, we introduce notions and notations and state basic properties of convex polytopes. In Section 4, we introduce our proposed model, *Polytopic Autoencoders (PAEs)* and define polytope error. In Section 5, we assess the reconstruction performance of PAEs in comparison to POD and CAEs. Additionally, we examine the outcomes achieved with pretrained PAEs. Section 6 serves as the conclusion of our study, where we summarize our findings and provide insights into potential future research directions.

2. Motivation and basic ideas

In view of making the connections to *autoencoders* and to the intended applications in *linear-parameter varying* (LPV) approximations to dynamical systems as in (1) later, we note the different nomenclature: In an autoencoder context, the reduced-order coordinates $\boldsymbol{\rho}$ are referred to as *latent variable*, whereas in an LPV context, we will consider the $\boldsymbol{\rho}$ a parametrization of the states.

General nonlinear autoencoders of type

$$\mathbf{v} \to \mu(\mathbf{v}) = \boldsymbol{\rho} \to \varphi(\boldsymbol{\rho}) = \tilde{\mathbf{v}} \approx \mathbf{v} \tag{2}$$

have been successfully employed for parametrizing dynamical systems like (1) on a very low-dimensional (if compared to, e.g., POD) manifold; see e.g., [35,14,17].

It has been noted, however, that the low-dimensional approximation of (1) via

$$\dot{\mathbf{v}}(t) \approx \dot{\tilde{\mathbf{v}}}(t) = \frac{d\varphi}{d\boldsymbol{\rho}} \dot{\boldsymbol{\rho}}(t) = f(\varphi(\boldsymbol{\rho}(t)));$$ (3)

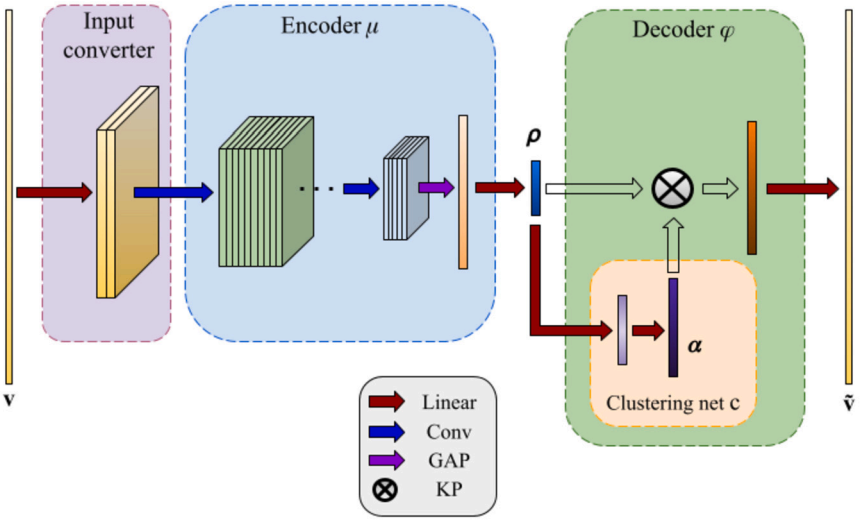

Fig. 1. Polytopic Autoencoder (PAE): "Linear" corresponds to a linear layer, "Conv" refers to a convolutional layer, "GAP" stands for global average pooling, and "KP" is the Kronecker product. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

apart from the nonlinear reconstruction $\tilde{\mathbf{v}}(t) = \varphi(\boldsymbol{\rho}(t))$ requires the repeated evaluation of the Jacobian $\frac{d\varphi}{d\rho}$ which can be computationally demanding. Moreover, the task of inferring an unstructured nonlinear map from the low-dimensional range of $\boldsymbol{\rho}$ to the high-dimensional state space is ill-posed. Because of these shortcomings, a decisive performance advantage of nonlinear model order reduction over linear projections is yet to be established.

The use of local linear bases to express $\tilde{\mathbf{v}}$, provides a general remedy, as they enforce a certain structure that pays off both in less computational effort for the reconstruction and in better posed approximations tasks for the design of the decoder φ. In our previous work [26], we have confirmed that with individual affine linear decoders for clusters identified a priori in the latent space, a superior reconstruction can be achieved with the comparatively cheap operation of locating the current value of $\boldsymbol{\rho}$ in the correct cluster.

The use of clustering as well as any other selection algorithm for local bases, however, comes at the cost of a noncontinuous decoding map φ. Therefore, the presented work aims at providing the reconstruction performance of local bases but with a smooth selection algorithm so that $\frac{d\varphi}{d\rho}$ and, thus, $\tilde{\mathbf{v}}$ is differentiable. For this, we base the decoding on the Kronecker product $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ of the latent variable $\boldsymbol{\rho}$ and a smooth clustering variable $\boldsymbol{\alpha} = c(\boldsymbol{\rho})$ and obtain the reconstruction as a linear combination with $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ as coefficients; see Fig. 1 for an illustration.

By standard techniques from the training of neural networks, we will control the number of nonzero values in $\boldsymbol{\alpha}$ so that the reconstruction will happen with a limited number of local basis vectors. Even more, we can guarantee that the entries of $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ are nonnegative and sum up to one so that the reconstruction has the interpretation of a convex combination within a polytope.

The intended manifold advantages are as follows:

- The nonlinearity of the decoding is reduced to the map $\boldsymbol{\rho} \to \boldsymbol{\alpha}$ between small dimensional sets.
- The decoding happens in a polytope defined by a small number of vertices.
- With $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ being the coefficients of a convex combination with a small number of nonzero values, the reconstruction happens within a bounded set with improved stability due to the reduction of summations.
- The map $\boldsymbol{\alpha} \otimes \boldsymbol{\rho} \to \tilde{\mathbf{v}}$ is linear and provides an approximative polytopic expansion of the state $\mathbf{v}$, which is useful in the numerical treatment of LPV systems.

Concretely, the proposed autoencoder comprises four key components:

1. A linear input converter that interpolates the spatially distributed data to a rectangular grid as it is needed for conventional convolutional layers in a neural network.
2. A nonlinear encoder which maps the high dimensional input states onto low dimensional latent variables with the final layer designed such that the latent variables are all positive and sum up to one (to later serve as coefficients for a convex combination (cp. Definition 3.1)) of supporting vectors.
3. A nonlinear smooth clustering network, responsible for locating the latent variables within one or more of k clusters.
4. A linear decoder, which turns the latent variable and the clustering result into a reconstruction of a high-dimensional state within a polytope.

In terms of equations, the map of an input $\mathbf{v}$ to the reconstruction $\tilde{\mathbf{v}}$ will read

$$\mathbf{v}_{\text{CNN}} = \mathbf{I}_C \mathbf{v}$$

$$\boldsymbol{\rho} = \mu(\mathbf{v}_{\mathsf{CNN}})$$

$$\boldsymbol{\alpha} = c(\boldsymbol{\rho})$$

$$\tilde{\mathbf{v}} = \varphi(\boldsymbol{\rho}) = \mathbf{U}(\boldsymbol{\alpha} \otimes \boldsymbol{\rho})$$

where $\mathbf{v}$ is the data, $\mathbf{v}_{\mathsf{CNN}}$ is the input for a CNN, $\mathbf{I}_C$ is an interpolation matrix, $\mathbf{U}$ is the matrix of all supporting vectors, $\boldsymbol{\rho}$ is the latent state, and $\tilde{\mathbf{v}}$ is the reconstruction designed to well approximate $\mathbf{v}$. In view of applications to dynamical equations, we note that all involved mappings are differentiable so that a differentiable (in time t) input results in a differentiable output by virtue of

$$\frac{d\tilde{\mathbf{v}}}{dt} = \frac{d\varphi}{d\boldsymbol{\rho}}\dot{\boldsymbol{\rho}} = \mathbf{J}_D\dot{\boldsymbol{\rho}}$$

where $\mathbf{J}_D$ is the Jacobian matrix of the decoder. In the proposed setup, we obtain the Jacobian matrix

$$\mathbf{J}_D = \mathbf{U}[\frac{dc(\boldsymbol{\rho})}{d\boldsymbol{\rho}} \otimes \boldsymbol{\rho} + c(\boldsymbol{\rho}) \otimes \mathbf{I}_r] \tag{5}$$

where $\mathbf{I}_r$ is the $r \times r$ identity matrix. Consequently, as $\mathbf{U}$ is constant, the computation of $\mathbf{J}_D$ only requires the update of the small Jacobian matrix of $c(\boldsymbol{\rho})$ with a change in $\boldsymbol{\rho}$.

3. Preliminaries and notation

We introduce basic notations and definitions concerning convex combinations and polytopes and lay out details of the various components of the proposed architecture.

Definition 3.1. Let $\mathcal{U} := \{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n\} \subset \mathcal{V}$ be a finite subset of a real vector space $\mathcal{V}$.

1. Let $\lambda_1, \lambda_2, \cdots, \lambda_n$ be nonnegative real values satisfying $\sum_{i=1}^n \lambda_i = 1$. Then $\sum_{i=1}^n \lambda_i \mathbf{u}_i$ is said to be a *convex combination* of the vectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n$ and
2. the *convex hull* of $\mathcal{U}$ is defined and denoted as

$$\mathrm{Co}(\mathcal{U}) = \{\mathbf{z} \in \mathcal{V} \mid \mathbf{z} \text{ is a convex combination of the vectors of } \mathcal{U}\}.$$

3. Moreover, if for any subset $\bar{\mathcal{U}} \subsetneq \mathcal{U}$ it holds that $\mathrm{Co}(\bar{\mathcal{U}}) \subsetneq \mathrm{Co}(\mathcal{U})$, then we call $\mathrm{Co}(\mathcal{U})$ a *convex polytope*.

Remark 1. By construction, the convex hull $\mathrm{Co}(\mathcal{U})$ is convex. This means that for any $z_1,\ z_2 \in \mathrm{Co}(\mathcal{U})$ and $\lambda \in [0,1]$, we have $z = \lambda z_1 + (1-\lambda) z_2 \in \mathrm{Co}(\mathcal{U})$. In what follows, we generally assume that $\mathrm{Co}(\mathcal{U})$ is a convex polytope which basically means that the vectors in $\mathcal{U}$ are linearly independent.

Lemma 3.1. *Let* $\boldsymbol{\rho} = [\rho_1, \rho_2, \cdots, \rho_r]^\top$ *and* $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_k]^\top$ *with* $\rho_i \geq 0,\ i \in \{1, 2, \cdots, r\},\ \sum_{i=1}^r \rho_i = 1,\ \alpha_j \geq 0,\ j \in \{1, 2, \cdots, k\}$ *and* $\sum_{j=1}^k \alpha_j = 1$. *Then the entries* $\alpha_{ij} = \alpha_i \rho_j$ *of* $\boldsymbol{\alpha} \otimes \boldsymbol{\rho} \in \mathbb{R}^{kr}$ *are positive and sum up to one.*

Proof. To begin, we note that by $\rho_i \geq 0,\ \alpha_j \geq 0$, we have that $\rho_i \alpha_j$ is nonnegative for all i and j. Moreover, for the sum over the elements of $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$, we have that

$$\sum_{i=1}^k \sum_{j=1}^r \alpha_i \rho_j = \sum_{i=1}^k \alpha_i \sum_{j=1}^r \rho_j = (\alpha_1 + \alpha_2 + \cdots + \alpha_k)(\rho_1 + \rho_2 + \cdots + \rho_r) = 1. \quad \square$$

Throughout the paper, we adopt the following notation. Since we consider data from dynamical systems, the involved variables $\mathbf{v}$, $\mathbf{v}_{\mathsf{CNN}}$, $\boldsymbol{\rho}$, $\boldsymbol{\alpha}$ can be seen as functions of time t. Where appropriate, we will drop these time dependencies and write, e.g., $\boldsymbol{\rho} = \mu(\mathbf{v}_{\mathsf{CNN}})$. For general data points we will write, e.g., $\mathbf{v}(t)$. Further down the text, where we consider data that was sampled at time instances $t^{(k)}$, we write, e.g., $\mathbf{v}^{(:)} = \mathbf{v}(t^{(k)})$. A subscript like in $\rho_i(t)$ will denote the i-th component of a vector-valued quantity or an enumerated set of items (like columns of a matrix).

4. State reconstruction within a polytope

In this section, we provide detailed information about *Polytopic Autoencoders (PAEs)* as illustrated in Fig. 1. Specifically, we demonstrate the process of feeding data generated by the Finite Element Method (FEM) to Convolutional Neural Networks (CNNs). We detail the design of a lightweight convolutional encoder architecture using depthwise and pointwise convolutions, which significantly reduces the number of model parameters compared to general full linear layers and also to POD. We also explain the methods for clustering reduced states in a low-dimensional space, reconstructing states within a polytope, training PAEs, and evaluating approximation errors in the polytopes that are defined by the PAEs.

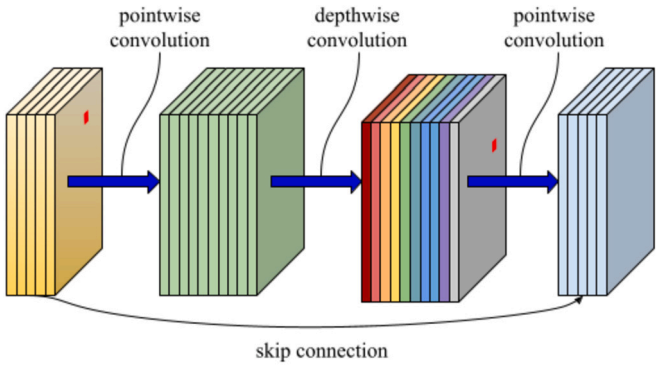

Fig. 2. Inverted residual block: an efficient approach for designing deep convolutional layers with fewer parameters compared to standard convolutions. (Section 4.2).

4.1. Input converter

We consider data that comes from FEM simulations on possibly nonuniform meshes which are not readily suited as inputs for convolutional neural networks. Rather than resorting to particular techniques like graph-convolutional neural networks in this context (e.g., [36]), we interpolate the data on a tensorized grid (cp. [37]) which is efficiently realized through a very sparse (only 0.03% nonzero entries) interpolation matrix $\mathbf{I}_C$. As the data can be multivariate, we consider the interpolation result $\mathbf{v}_{\mathsf{CNN}}(t) = \mathbf{I}_C \mathbf{v}(t)$ of a data point $\mathbf{v}(t)$ as a three-dimensional tensor in $\mathbb{R}^{C\times H\times W}$), where C stands for the number of input channels (i.e., the dimension of the data) and H and W are the number of data points (i.e., the number of pixels) in the two spatial dimensions.

4.2. Encoder

As the encoder

$$\mu : \mathbb{R}^{C\times H\times W} \to \mathbb{R}^r : \mathbf{v}_{\mathsf{CNN}}(t) \to \boldsymbol{\rho}(t),$$

we set up a convolutional neural network with the final layer employing a *softmax* function that ensures for the output vector $\boldsymbol{\rho}(t) = \mu(\mathbf{v}_{\mathsf{CNN}}(t))$ that all components are positive and sum up to one, i.e.,

$$\rho_i(t) \geq 0, \quad \text{for } i = 1, \dots, r \quad \text{and} \quad \sum_{i=1}^{r} \rho_i(t) = 1, \tag{6}$$

and with, importantly, the reduced dimension r being significantly smaller than n.

The *softmax function*

$$\mathbf{x} \to \left[\frac{e^{x_i}}{\sum_{j=1}^{r} e^{x_j}} \right]_{i=1,\dots,r}$$

that is commonly used in machine learning to produce outputs in the form of a probability distribution appeared unsuited for the intended selection of vertices of a polytope later. In fact, the standard *softmax* never attains zero exactly and requires rather large (in magnitude) input values for outputs close to zero. That is why, we utilized a modified *softmax* function

$$\mathrm{softmax}(\mathbf{x})_i = \frac{x_i \cdot \tanh(a x_i)}{\sum_{j=1}^{r} x_j \cdot \tanh(a x_j)} \tag{7}$$

where $a > 0$. As the signs of x and $\tanh(x)$ always coincide, the (differentiable) function $x \to x \cdot \tanh(ax)$ produces nonnegative values so that the softmax as defined in (7) ensures the desired property (6) for its output to potentially serve as coefficients of a convex combination. The parameter a serves the following purpose: For $a \to \infty$, we have $x \cdot \tanh(ax) \to |x|$ so that a defines a compromise between a smoothly differentiable function and the absolute value, which is a commonly used activation function because it does not suffer from the *vanishing gradient* phenomenon. In preliminary experiments, we identified a value of $a = 10$ as superior for both convergence in the training and accuracy in the approximation.

Remark 2. In view of treating high-dimensional data, the number of parameters in the neural networks is a pressing issue in terms of training efficiency but also memory requirements and computational efforts for forward evaluations. To utilize a larger receptive field in the encoder with fewer parameters, we construct deep neural networks using convolution blocks that include a depthwise convolution and two pointwise convolutions as illustrated Fig. 2. A standard convolution operation extracts feature maps using a $K \times K \times C_O \times C_I$ kernel where K represents the kernel size, C_O is the number of output channels, and C_I is the number of input channels. To soften possibly large memory requirements, one possibly may decompose the standard convolution into a depthwise convolution and a pointwise convolution, creating what is called *depthwise separable convolution* [38]. We explain in Appendix B how depthwise separable convolution utilizes fewer model parameters than the standard convolution.

4.3. Differentiable clustering network

Naturally most popular clustering methods like the k-means clustering (e.g., [39]) classify data using discontinuous functions such as min and max functions which are non-differentiable. To alleviate this problem while maintaining the advantages of clustering for reconstruction, we resort to a differentiable clustering network, that is implemented on the r-dimensional latent space and defined as

$$c : \mathbb{R}^r \to \mathbb{R}^k : \boldsymbol{\rho}(t) \to \boldsymbol{\alpha}(t),$$

where c is a multi-layer *perceptron* with the modified softmax function $\mathrm{softmax}(\mathbf{x})$, cp. (7), in the last layer.

In view of reconstruction within a polytope of a limited number of vertices, we intend to ensure that $\boldsymbol{\alpha}$ only has a small number of nonzero values. Basically, a single nonzero entry of 1 will correspond to a cluster selection whereas a few nonzero entries will describe a smooth transition between the clusters.

Although for the modified softmax functions, in theory, the range covers the closed interval $[0, 1]$ the values will not be exactly zero or one in practice (except from a few rare cases). Thus, we enforce decisive selections of clusters by including the selection of pseudo labels for training the clustering network c; see Section 4.5.

4.4. Decoder

We recall the approach of *individual convolutional autoencoders (iCAEs)* [26] that, for a single nonlinear encoder

$$\boldsymbol{\rho} = \mu(\mathbf{v}_{\mathsf{CNN}}),$$

bases the reconstruction on k individual (affine) linear decoders

$$\tilde{\mathbf{v}} = \mathbf{U}_l \boldsymbol{\rho} + \mathbf{b}_l,\ l = 1, 2, \cdots, k$$

on k clusters. Here, $\mathbf{U}_l \in \mathbb{R}^{n \times r}$ and $\mathbf{b}_l \in \mathbb{R}^n$ are a matrix of local basis vectors and a bias for the reconstruction of states in the l-th cluster.

If one leaves aside the bias terms, then the reconstructed states $\tilde{\mathbf{v}}$ can be described as a discontinuous decoder

$$\tilde{\mathbf{v}} = \sum_{i=1}^{k} \beta_i \mathbf{U}_i \boldsymbol{\rho} = \sum_{j=1}^{r} \sum_{i=1}^{k} \beta_i \rho_j \mathbf{u}_{i;j}$$

where β_i is i-th element of a vector

$$\boldsymbol{\beta} = \begin{cases} 1 & \text{if } i = l, \\ 0 & \text{if } i \neq l \end{cases}$$

generated by k-means clustering (i.e., all its elements are 0 except for a single element which has a value of 1), and $\mathbf{u}_{i;j}$ is the j-th column vector of $\mathbf{U}_i$. This decoder is to select an individual decoder depending on the cluster which is a nonsmooth operation where the states leave one cluster for another.

Here, we replace the selection vector $\boldsymbol{\beta}$ by the output α of a smooth clustering network c that allows for several nonzero entries and, thus, enables smooth transitions between clusters:

$$\begin{aligned} \tilde{\mathbf{v}} &= \sum_{j=1}^{r} \sum_{i=1}^{k} \alpha_i \rho_j \mathbf{u}_{i;j} \\ &= \mathbf{U}(c(\boldsymbol{\rho}) \otimes \boldsymbol{\rho}) \end{aligned} \tag{9}$$

where α_i is i-th element of the smooth clustering output $\boldsymbol{\alpha} = c(\boldsymbol{\rho})$, where $\otimes$ is the Kronecker product, and where

$$\mathbf{U} = \begin{bmatrix} | & | & & | & & | & & | \\ \mathbf{u}_{1;1} & \mathbf{u}_{1;2} & \cdots & \mathbf{u}_{1;r} & \cdots & \mathbf{u}_{k;1} & \cdots & \mathbf{u}_{k;r} \\ | & | & & | & & | & & | \end{bmatrix}$$

is the matrix of all supporting vectors for the reconstruction.

Remark 3. Due to the modified softmax function in the last layer of μ and c, it holds that $\boldsymbol{\rho}(t) = \mu(\mathbf{v}_{\mathsf{CNN}}(t))$ and $\boldsymbol{\alpha}(t) = c(\boldsymbol{\rho}(t))$ both satisfy property (6), so that by Lemma 3.1, the decoder output $\varphi(\boldsymbol{\rho}(t))$ is a convex combination of the column vectors of $\mathbf{U}$. Consequently, all reconstructed states are generated within a polytope $\tilde{\mathcal{V}}$ defined by the vertices $\mathbf{u}_{1;1}, \ldots, \mathbf{u}_{k;r}$.

Remark 4. By design and by the definition of the Kronecker product

$$\boldsymbol{\alpha} \otimes \boldsymbol{\rho} = \begin{bmatrix} \alpha_1 \boldsymbol{\rho} & \alpha_2 \boldsymbol{\rho} & \ldots & \alpha_k \boldsymbol{\rho} \end{bmatrix},$$

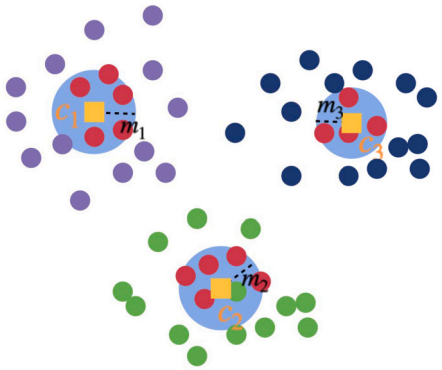

Fig. 3. When latent variables are divided into three clusters in a low-dimensional space, the clustering labels corresponding to the latent variables (red circles) within each circle with radius m_i, $i = 1, 2, 3$, are chosen as target labels. Unselected labels are not used for training PAEs. (Section 4.5).

the reconstruction in (9) bases on a multiplication of the coordinates vector $\boldsymbol{\rho}$ with weights a_i referring to the i-th cluster. Rather than this basic weighting by scalars, one may similarly consider (discrete) convolutions

$$\mathbf{g} * \boldsymbol{\rho} = \begin{bmatrix} g_1 * \boldsymbol{\rho} & g_2 * \boldsymbol{\rho} & \dots & g_k * \boldsymbol{\rho} \end{bmatrix},$$

with suitably chosen convolution kernels g_i that may depend on $\boldsymbol{\rho}$ like the clustering coefficients $\boldsymbol{\alpha}$. Indeed, one can show that if all coefficients of the kernels are positive and sum up to one, then the result of the convolutions satisfies the conditions (6) as needed for the polytopic reconstruction. Furthermore, because of the multiplicative nature of a convolution, formulas for the Jacobian similar to (5) can be derived. However, for larger kernels g_i the mapping $\boldsymbol{\rho} \to \mathbf{g}$ (and it's Jacobian) will become more involved.

4.5. Training strategy for PAEs

The proposed encoder, decoder, and clustering network are fully differentiable with respect to input variables and all model nodes, enabling a joint optimization of all model parameters using gradient-based techniques; e.g., Adam [40]. Nonetheless, in view of training efficiency and since the smooth clustering network will be trained by means of pseudo-labels obtained from k-means clustering, the overall training strategy includes two preparatory steps. In a final step, all components are then fine-tuned in a joint optimization.

We comment on these three training steps with details deferred to the technical description in Appendix C.

Step 1 (initialization and identification of the latent space): We train a CAE consisting of a nonlinear encoder μ and a polytopic decoder $\bar{\varphi}$.

Step 2 (k-means clustering for generating pseudo-labels and initialization of individual decoders): From the pretrained encoder, we obtain the latent variable coordinates for the training data which are then used for k-means clustering in the latent space. The clustering results are then used both for generating target labels for a supervised training of the smooth clustering network c and for training individual decoders in the clusters.

In order to avoid overfitting of the k-means results and to shift the focus away from the cluster centers to the region around them, we assign the pseudo labels to represent areas around the centroids and train the clustering network to best match the pseudo-labels in a distributional sense.

Concretely, for each cluster i, we select those $j(i)$ data points for which the latent coordinates satisfy

$$\|\boldsymbol{\rho}^{(j(i))} - \mathbf{c}_i\| < m_i,\ i = 1, 2, \cdots, k \tag{10}$$

where $\mathbf{c}_i$ is the centroid of the i-th cluster and where m_i is the mean of the distances between the latent variables and $\mathbf{c}_i$ in this cluster. To these selected data points we assign the i-th unit vector as the pseudo-label.

With this procedure repeated for all clusters, we collect a set of data/labels pairs

$$\mathcal{X}_{\text{cl}} := \{(\mathbf{v}^{(1)}, \mathbf{l}^{(1)}), (\mathbf{v}^{(2)}, \mathbf{l}^{(2)}), \dots, (\mathbf{v}^{(N_l)}, \mathbf{l}^{(N_l)})\} \tag{11}$$

where N_l is the overall number of data points selected by the criterion (10), where $\mathbf{v}^{(i)}$ is the velocity data so that $\boldsymbol{\rho}^{(i)} = \mu(\mathbf{v}^{(i)})$, and where the labels $\mathbf{l}^{(i)}$ are unit vectors in $\mathbb{R}^r$ representing the corresponding cluster.

In a next preparatory step, the clustered and labeled data $\mathcal{X}_{\text{cl}}$ is used to train individual convex combination-based decoders $\varphi_1, \cdots, \varphi_k$ in each cluster.

Then, the matrix $\mathbf{U} = [\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}]$ that collects all supporting vectors of the individual decoders is used to initialize the weights of a global, smooth and clustering-based decoder.

Step 3 (training of the clustering net and fine-tuning of the PAE): we train a PAE by fine-tuning μ and $\mathbf{U}$ while simultaneously optimizing the clustering network c. For the model optimization, we define a reconstruction loss as

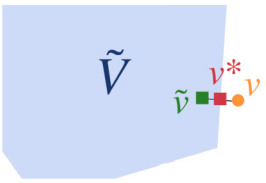

Fig. 4. Conceptual figure depicting the positions of a state $\mathbf{v}$, it's reconstruction $\tilde{\mathbf{v}}$ in a polytope, and it's best approximation $\mathbf{v}^*$. (Section 4.6).

$$\mathcal{L}_{\text{rec}} = \frac{1}{|B|} \sum_{i \in B} \| \tilde{\mathbf{v}}^{(i)} - \mathbf{v}^{(i)} \|_{\mathbf{M}}$$

where B is the index set of a data batch drawn from the training data. This loss function is the standard *mean squared error* loss but in the $\mathbf{M}$-norm that reflects the PDE setup. Also, we define a clustering loss as the cross entropy loss

$$\mathcal{L}_{\text{clt}} = -\frac{1}{|P|} \sum_{j \in P \subset B} \mathbf{l}^{(j)} \cdot \log(c(\boldsymbol{\rho}^{(j)})).$$

where j comes from that subset $P \subset B$ that contains only those indices that address data that is part of the clustered and labeled data set $\mathcal{X}_{\text{cl}}$ too; cp. (11). The cross-entropy loss function describes the distance between two probability distributions and is commonly used as a loss function for training multi-class classification machine learning models. Note that both the labels $\mathbf{l}^{(j)}$ (as unit vectors representing a sharp uni-modal distribution) and the clustering output $\boldsymbol{\alpha}^{(j)} = c(\boldsymbol{\rho}^{(j)})$ (by virtue of the softmax in the final layer; cp. Section 4.3) represent probability distributions.

Finally, we consider the joint loss function

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + 10^{-4} \mathcal{L}_{\text{clt}},$$

where the weight 10^{-4} turned out to be a good compromise between accuracy and overfitting.

4.6. Polytope error and polytopic LPV representation

By design, the reconstruction $\tilde{\mathbf{v}}$ is generated inside a polytope, cp. Remark 3. We denote this polytope by $\mathcal{V}$.

Definition 4.1 *(Polytope error and best approximation).* Let $\mathcal{V} \subset \mathbb{R}^n$ be a convex polytope. For a given data point $\mathbf{v} \in \mathbb{R}^n$, let $\| \cdot \|_{\mathbf{M}}$ be the norm that is induced by the $\mathbf{M}$-weighted inner product and let

$$\text{dist}_{\mathbf{M}}(\mathbf{v}, \mathcal{V}) := \min_{\mathbf{w} \in \mathcal{V}} \|\mathbf{v} - \mathbf{w}\|_{\mathbf{M}}$$

be the *polytope error* and let $\mathbf{v}^* \in \mathcal{V}$ be the *best approximation* that realizes the minimum.

We note that as shown in the appendix in Lemma 6.3, the *polytope errors* and the *best approximation* is well defined; see also Fig. 4 that illustrates the conceptual representation of the polytope error.

In the numerical experiments, for the data at hand, we will evaluate the polytope error for the polytope that is identified and used for the encoding and the reconstruction by the PAE. This provides best-case estimates for

1. how well the identified polytope can represent the data and
2. how close the reconstruction gets to this theoretical lower bound.

The computation of the best approximation is a nontrivial task. To compute the *polytope error* for a given $\mathbf{v}$, we solve the optimization problem

$$\min_{\rho \in \mathbb{R}^r} \|\mathbf{U}\boldsymbol{\rho} - \mathbf{v}\|_{\mathbf{M}}^2$$

$$\text{subject to} \quad \boldsymbol{\rho} \geq 0,$$

$$\mathbb{1}_r \boldsymbol{\rho} = 1,$$

for the coordinates $\boldsymbol{\rho}^*$ of the best approximation $v^* = \mathbf{U}\boldsymbol{\rho}$, where $\mathbf{U}$ is the matrix of the vertices (cp. Remark 3), and where $\mathbb{1}_r = [1, 1, \cdots, 1]$ is the row vector of ones with r entries.

This convex quadratic programming problem involving linear constraints does not have closed-form solution but numerical methods including active-set methods and interior-point methods can efficiently find solutions. In the experiments, we use a quadratic programming solver provided by the Python library `cvxopt` which implements interior-point methods [41].

4.7. Application in polytopic LPV approximations

We briefly comment on the intended application in approximating general nonlinear functions $f : \mathbb{R}^n \to \mathbb{R}^n$ by so-called linear-parameter varying approximations of preferably low parameter dimension. Such approximations are a promising ingredient for nonlinear controller design; see [42] for the basic theory and proofs of concepts.

An intermediate step is the representation of f in state-dependent coefficient form

$$f(\mathbf{v}) = A(\mathbf{v})\,\mathbf{v}$$

which always exists under mild regularity conditions and in the case that $f(0) = 0$. If then $A(\mathbf{v}) \approx A(\tilde{\mathbf{v}})$ is approximated by the autoencoded state $\tilde{\mathbf{v}} = \varphi(\boldsymbol{\rho}(\mathbf{v}))$, an LPV approximation with $\boldsymbol{\rho} = \boldsymbol{\rho}(\mathbf{v})$ as the parameter is obtained by means of

$$f(\mathbf{v}) \approx A(\tilde{\mathbf{v}})\,\mathbf{v} = A(\varphi(\boldsymbol{\rho}(\mathbf{v})))\,\mathbf{v} =: \tilde{A}(\boldsymbol{\rho})\,\mathbf{v}.$$

As so-called affine linear polytopic LPV representations are of particular use for controller design, we show how a polytopic reconstruction with $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ transfers to a polytopic LPV approximation with now $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ as the parameters.

Lemma 4.1. *Let $\mathbf{A}(\cdot) : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ be a linear map and let $\tilde{\mathbf{v}}(t)$ be a (convex) combination of n vertices of a polytope. Then $\mathbf{A}(\tilde{\mathbf{v}}(t))$ is a (convex) combination of n matrices representing a polytope in $\mathbb{R}^{n\times n}$.*

Proof. Since $\tilde{\mathbf{v}}(t)$ is a (convex) linear combination of n vertices, $\tilde{\mathbf{v}}(t)$ can be expressed as

$$\tilde{\mathbf{v}}(t) = \sum_{i=1}^{n} \zeta_i \mathbf{u}_i$$

where $\zeta_i \in \mathbb{R}$ ($\zeta_i \geq 0$, $\sum_{i=1}^{n} \zeta_i = 1$) and $\mathbf{u}_i$ is i-th vertex of a polytope, $i \in \{1, 2, \cdots, n\}$. Then

$$\mathbf{A}(\tilde{\mathbf{v}}(t)) = \mathbf{A}(\sum_{i=1}^{n} \zeta_i \mathbf{u}_i)$$

Since $\mathbf{A}$ is linear in its argument, we have that

$$\mathbf{A}(\tilde{\mathbf{v}}(t)) = \mathbf{A}(\sum_{i=1}^{n} \zeta_i \mathbf{u}_i) = \sum_{i=1}^{n} \zeta_i \mathbf{A}_i$$

where $\mathbf{A}_i := \mathbf{A}(\mathbf{u}_i)$. Therefore, $\mathbf{A}(\tilde{\mathbf{v}}(t))$ is a (convex) linear combination of $\mathbf{A}_1, \mathbf{A}_2, \cdots, \mathbf{A}_n$. □

The alternative and standard way of locating an affine LPV representation of parameter dimension r in a polytope is to determine the r-dimensional bounding box; e.g., [43]. Here, the number of the vertices of the polytope is 2^r, and thus, increases exponentially with the reduced dimension r. If one succeeds to identify a bounding polytope of less vertices, one is confronted with computing the coordinates within these coordinates for which there is no established method for dimensions beyond $r = 3$; cp. the discussion in [42].

In both respects, PAEs offer a promising alternative. Firstly, the relevant parametrization $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ already defines the needed bounding polytope with a size of $r \cdot k$ which grows linearly in the number of clusters k and the dimension of the parametrization. Note that the difference to 2^r becomes advantageous for moderate and large r, meaning that, e.g., $r \cdot k < 2^r$ for $r \geq 5$ and $k \leq r$. Secondly, the decoder readily provides the coordinates in the considered polytope.

5. Simulation results

As an example application, we consider flow simulations with the *incompressible Navier-Stokes equations*

$$\frac{\partial}{\partial t}\mathrm{v} + (\mathrm{v}\cdot\nabla)\mathrm{v} + \frac{1}{\mathsf{Re}}\Delta\mathrm{v} - \nabla\mathrm{p} = \mathrm{f} \tag{13a}$$

$$\nabla\cdot\mathrm{v} = \mathbf{0}, \tag{13b}$$

where v, p, and f are the velocity, pressure, and forcing term respectively and where Re is the Reynolds number. After an FEM discretization of Equation (13) a semi-discrete model is obtained as

$$\mathbf{M}\dot{\mathbf{v}}(t) + \mathbf{N}(\mathbf{v}(t))\mathbf{v}(t) + \mathbf{A}\mathbf{v}(t) - \mathbf{J}^\top\mathbf{p}(t) - \mathbf{f}(t) = \mathbf{0} \tag{14a}$$

$$\mathbf{J}\mathbf{v}(t) = \mathbf{0}, \tag{14b}$$

where $\mathbf{v}(t) \in \mathbb{R}^n$ and $\mathbf{p}(t) \in \mathbb{R}^p$ are the states of the velocity and the pressure at time t respectively, and where $\mathbf{M} \in \mathbb{R}^{n\times n}$, $\mathbf{N}(\cdot) \in \mathbb{R}^{n\times n}$, $\mathbf{A} \in \mathbb{R}^{n\times n}$, $\mathbf{J} \in \mathbb{R}^{p\times n}$, and $\mathbf{f}(t) \in \mathbb{R}^n$ are the mass, convection, diffusion, discrete divergence matrices, and the forcing term at time t respectively; see the details for realizing the convection as a state-dependent coefficient in [44].

For the presentation that follows, we employ the so-called ODE formulation of the DAE (14) leaving aside all technical challenges associated with handling the pressure $\mathbf{p}(t)$ in numerical schemes; cp. [45]. The ODE formulation of (14) is derived under the reasonable assumption that $\mathbf{M}$ and $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\top$ are invertible and the observation that $\mathbf{J}\mathbf{v}(t) = \mathbf{0}$ implies

$$\mathbf{J}\dot{\mathbf{v}}(t) = \mathbf{0}.$$

Then, by multiplying $\mathbf{M}^{-1}$ on both sides of Equation (14), we obtain

$$\dot{\mathbf{v}}(t) + \mathbf{M}^{-1}(\mathbf{N}(\mathbf{v}(t))\mathbf{v}(t) + \mathbf{A}\mathbf{v}(t) - \mathbf{J}^\top\mathbf{p}(t) - \mathbf{f}(t)) = \mathbf{0}$$

Then, by multiplying $\mathbf{J}$ on both sides,

$$\mathbf{J}\mathbf{M}^{-1}(\mathbf{N}(\mathbf{v}(t))\mathbf{v}(t) + \mathbf{A}\mathbf{v}(t) - \mathbf{J}^\top\mathbf{p}(t) - \mathbf{f}(t)) = \mathbf{0}\,(\because \mathbf{J}\dot{\mathbf{v}}(t) = \mathbf{0})$$

Finally, $\mathbf{p}(t)$ can be described with respect to $\mathbf{v}(t)$ and $\mathbf{f}(t)$ as follows:

$$\mathbf{p}(t) = \mathbf{S}^{-1}\mathbf{J}\mathbf{M}^{-1}(\mathbf{N}(\mathbf{v}(t))\mathbf{v}(t) + \mathbf{A}\mathbf{v}(t) - \mathbf{f}(t))$$

where $\mathbf{S} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^\top$. Thus, we can eliminate the pressure $\mathbf{p}(t)$ from (14) and the equation can be presented as

$$\mathbf{M}\dot{\mathbf{v}}(t) = \mathbf{\Pi}^\top(\mathbf{N}(\mathbf{v}(t))\mathbf{v}(t) + \mathbf{A}\mathbf{v}(t) - \mathbf{f}(t)) \tag{15}$$

where $\mathbf{\Pi} = \mathbf{M}^{-1}\mathbf{J}^\top\mathbf{S}^{-1}\mathbf{J} - \mathbf{I}$. All data are generated by Equation (15).

5.1. Data acquisition and performance measures

For different setups we perform simulations of the FEM discretized incompressible flow equations (14) over time t starting from the associated *Stokes* steady state and collect the data for training the autoencoders. The data points are then given as the *snapshots* of the (discrete) velocity variable $\mathbf{v}^{(k)} = \mathbf{v}(t^{(k)})$ at time instances $t^{(k)}$.

Using this data, we compute POD approximations and optimize the neural networks according to the following performance criteria.

In each reduced dimension of r, we evaluate the reconstruction performance of PAEs compared to other methods by measuring the averaged relative error

$$\frac{1}{T}\sum_{i=1}^{T}\frac{\|\,\tilde{\mathbf{v}}^{(i)} - \mathbf{v}^{(i)}\,\|_{\mathbf{M}}}{\|\,\mathbf{v}^{(i)}\,\|_{\mathbf{M}}},$$

and the averaged relative polytope error ε_p (cp. Definition 4.1)

$$\frac{1}{T}\sum_{i=1}^{T}\frac{\|\,\mathbf{v}^{(i)*} - \mathbf{v}^{(i)}\,\|_{\mathbf{M}}}{\|\,\mathbf{v}^{(i)}\,\|_{\mathbf{M}}}$$

where T is the number of snapshots.

Additionally, we investigate the trajectories of reconstructed states and latent variables, as well as the polytopes used for the reconstruction.

In view of memory efficiency, we report the number of encoding parameters, decoding parameters, and vertices of polytopic LPV representations. The number of encoding parameters for CAE and PAE is much less than that of POD. This reduction in size is attributed to the immutable and sparse interpolation matrix $\mathbf{I}_C$ and depthwise separable convolutions. Regarding the number of decoding parameters, for POD and CAE, their decoders are linear, resulting in sizes of nr, denoting the number of elements for an $n \times r$ matrix. The decoding size of PAE is $nrk + m$ where m is the number of parameters in the clustering net.

As an additional performance characteristic, we report the number R of vertices of a polytope that contains the reconstruction values as it would be used for LPV approximations; cp. Section 4.7. For the POD approximations, we consider the bounding box with $R = 2^r$, where r is the dimension of $\boldsymbol{\rho}$ which is the standard approach in absence of, say, an algorithm that would compute a polytopic expansion in a general polytope. For the CAE or the PAE, however, this polytopic expansion is readily given in a polytope of $R = r$ or $R = kr$ vertices, where k is the number of clusters.

In practice, the right choice of r (number of latent variables) and k (number of clusters) will be a tradeoff between accuracy and complexity. In the case of PAEs, regarding the interplay of k and r, the numerical results suggest that for controller design (cp. Section 4.7) one should rather increase the r and set $k = 1$ whereas for projection based model reduction (cp. Equation (3)), one may well consider $k > 1$ for a better reconstruction error with only a little computational overhead; cp., e.g., Fig. 5 where the CAE for $r = 5$ shows a similar error level as the PAE3 (at $r = 2$ but with $k = 3$ clusters).

Data availability

The source code of the implementations used to compute the presented results is available from doi:10.5281/zenodo.10491870 under the Creative Commons Attribution 4.0 international license and is authored by Yongho Kim.

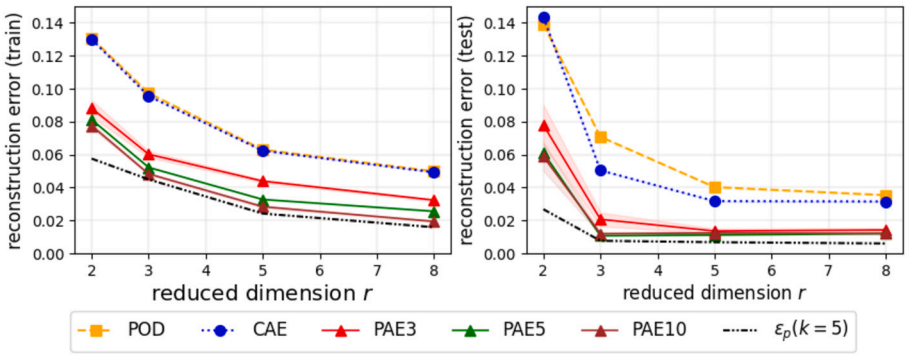

Fig. 5. Reconstruction error across the reduced dimension r averaged for 5 runs for the single cylinder case (Section 5.3). The shaded regions mark the statistical uncertainty measured through several training runs and appears to be insignificant and, thus, invisible in the plots for most methods.

5.2. Dataset: single cylinder

Our data are generated in the time domain $[0, 16]$. We use a Reynolds number of 40 for the single cylinder case. Each snapshot vector $\mathbf{v}(t)$ has 42,764 states (i.e., $n = 42764$) associated with nodes of the FEM mesh in the spatial domain $(0, 5) \times (0, 1) \subset \mathbb{R}^2$.

The dataset is divided into a training set containing 500 snapshots in the time interval $[0, 10]$ and a test set including 300 snapshots in $[10, 16]$. When convolutional encoders are employed, the interpolation matrix $\mathbf{I}_c \in \mathbb{R}^{42764 \times 5922}$ maps $\mathbf{v}(t) \in \mathbb{R}^{42764}$ into $\mathbf{v}_{\text{CNN}}(t) \in \mathbb{R}^{2 \times 63 \times 47}$ consisting of the x-directional velocity and the y-directional velocity values at each mesh point on a rectangular grid of size 63×47.

5.3. PAEs: single cylinder

In this simulation, each PAE and CAE has a deep convolutional encoder consisting of 14 convolutional layers and a fully connected layer. We use the ELU activation function [46] in the convolutional layers and the modified softmax function (7) in the last layer. To reduce the number of nodes in the last layer, the global average pooling is used before performing the fully connected computation. The decoder of POD is a linear combination of r vectors and the decoder of CAE is a convex combination of r vertices. The PAE decoder is partially linear as mentioned in Section 4.4. In other words, CAE consists of an encoding part

$$\mathbf{v}_{\text{CNN}}(t) = \mathbf{I}_C \mathbf{v}(t)$$
$$\boldsymbol{\rho}(t) = \mu(\mathbf{v}_{\text{CNN}}(t))$$

and a decoding part

$$\tilde{\mathbf{v}}(t) = \varphi(\boldsymbol{\rho}(t))$$

without clustering. Consequently, CAE is regarded as PAE with 1 cluster (i.e., $k = 1$).

Table 1 presents a comparison of the number of encoding and decoding parameters and the CPU time for each model. CAE and PAE maintain a relatively consistent number of encoding parameters regardless of reduced dimensions, in contrast with POD. In practice, when $r = 2, 3, 5, 8$, the encoding size of CAE and PAE is only $42.6\%, 28.6\%, 17.2\%$, and 10.8% of the encoding size of POD respectively in terms of the number of encoding parameters. The decoding size of POD and CAE is decided by a $n \times r$ decoding matrix. In contrast, the decoding size of PAE is larger than them due to the Kronecker product of $\boldsymbol{\alpha}$ and $\boldsymbol{\rho}$. For training CAEs and PAEs, the Adam optimizer is used with a learning rate η of 10^{-4}, a batch size of 64; see the details in Appendix C.

5.4. Results: single cylinder

In this section, we investigate how PAEs reconstruct periodic flows and handle their latent variables in very low-dimensional spaces. Fig. 5 shows the reconstruction errors of POD, CAE, and PAE against the reduced dimension r. These errors are calculated using the averaged errors obtained from 5 training trials, resulting in very small standard deviations. The CAE achieves similar averaged errors to POD on the training data over the time range $[0, 10]$. However, it outperforms POD in the test reconstruction errors except for the error at $r = 2$. In terms of the reconstruction error at each time, it is observed that the errors of the CAE exceed those of POD during the transition period where the flows evolve from relatively stable flows to periodic flows (approximately within the time range $[4, 6]$) and the CAE reconstructs periodic flows better than POD. We speculate that this phenomenon is due to the model learning being biased toward periodic flows, as the relatively large number of snapshots of periodic flows disproportionately influences the distribution of the training data, a common issue referred to as data imbalance in machine learning.

Overall, the reconstruction performance of the CAEs is comparable to POD, although the CAEs utilize fewer model parameters for the reconstruction and determine a smaller R for the design of polytopic LPV systems. The PAE(k)s that cluster latent variables into

Table 1
Model information: the number of encoding (L_e) and decoding (L_d) layers, the number of encoding (P_e) and decoding (P_d) parameters, and the number R which counts the vertices of a bounding box in $\mathbb{R}^r$ (for POD) or of the polytopes used for the reconstruction for CAE and PAE for the single cylinder case (Section 5.3); see how to calculate P_e and P_d in Section 5.1.

Model	r	L_e	P_e	L_d	P_d	R
POD	2	1	85,528	1	85,528	4
CAE	2	15	36,692	1	85,528	2
PAE(k = 3)	2	15	36,692	3	256,584	6
POD	3	1	128,292	1	**128,292**	8
CAE	3	15	**36,725**	1	**128,292**	**3**
PAE(k = 3)	3	15	**36,725**	3	384,876	9
POD	5	1	213,820	1	**213,820**	32
CAE	5	15	**36,791**	1	**213,820**	**5**
PAE(k = 3)	5	15	**36,791**	3	641,460	15
POD	8	1	342,112	1	**342,112**	256
CAE	8	15	**36,890**	1	**342,112**	**8**
PAE(k = 3)	8	15	**36,890**	3	1,026,336	24

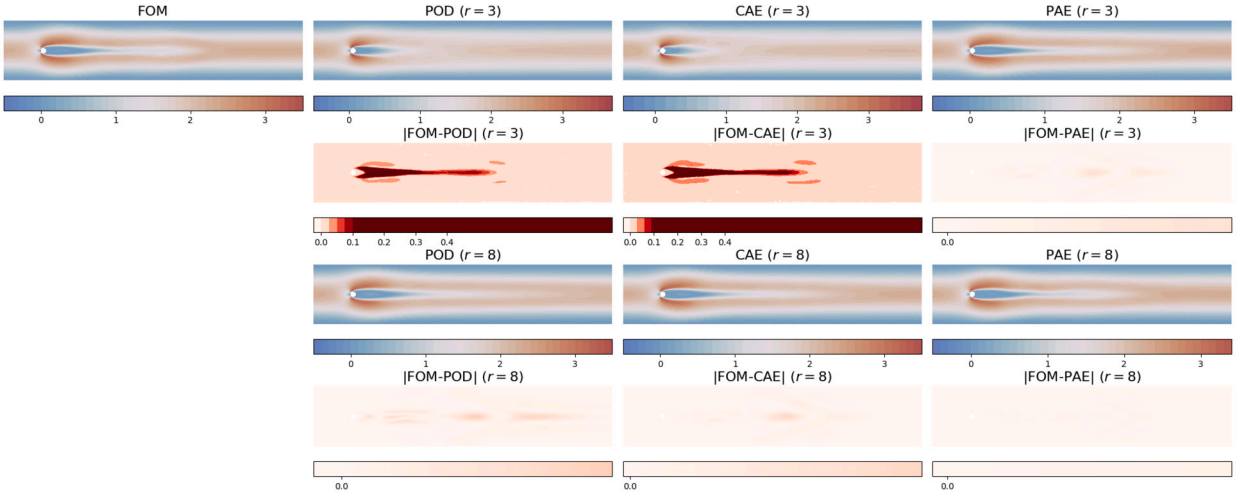

Fig. 6. Comparison of the reference generated by the full order model (FOM) and the developed snapshots of POD, CAE, and PAE at $t = 2.0$: training session for the single cylinder case (Section 5.3).

k clusters (e.g., PAE3, PAE5, PAE10) outperform the CAEs and POD. The reconstruction errors of the PAEs tend to be reduced as k gets larger. However, there is no significant gap between the errors of PAE5 and PAE10.

As a result of the polytope errors with $k = 5$, they are less than 2.7% for the periodic flows in the testing range $[10, 16]$. Specifically, these errors depending on the reduced dimensions $r = 2, 3, 5, 8$ achieve $2.7\%, 0.8\%, 0.7\%$, and 0.6% respectively. It indicates that the polytopes defined by the PAE5 are well-constructed, even when dealing with very low-dimensional latent variables.

Remark 5. Generally, the parameter r denotes the dimension of the latent state. For POD this equals the size of $\rho(t)$. For the polytopic decoding, due to the summation condition that $\sum_{i=1}^{r} \rho_i(t) = 1$, one dimension is redundant so that, theoretically, the actual latent dimension is $r - 1$. In our experiments, we haven't made use of this straight-forward way to reduce the parametrization even further. However, in the intended application in controller design, the elimination of one degree of freedom may well lead to an additional gain in performance.

Fig. 6 and Fig. 7 show a comparison of the developed snapshots from FOM, POD, CAE, and PAE3 at time $t = 2.0, 14.0$ respectively when $r = 3, 8$. The figures of their absolute errors show that PAE3 outperforms other models in terms of the state reconstruction with very low-dimensional parametrizations.

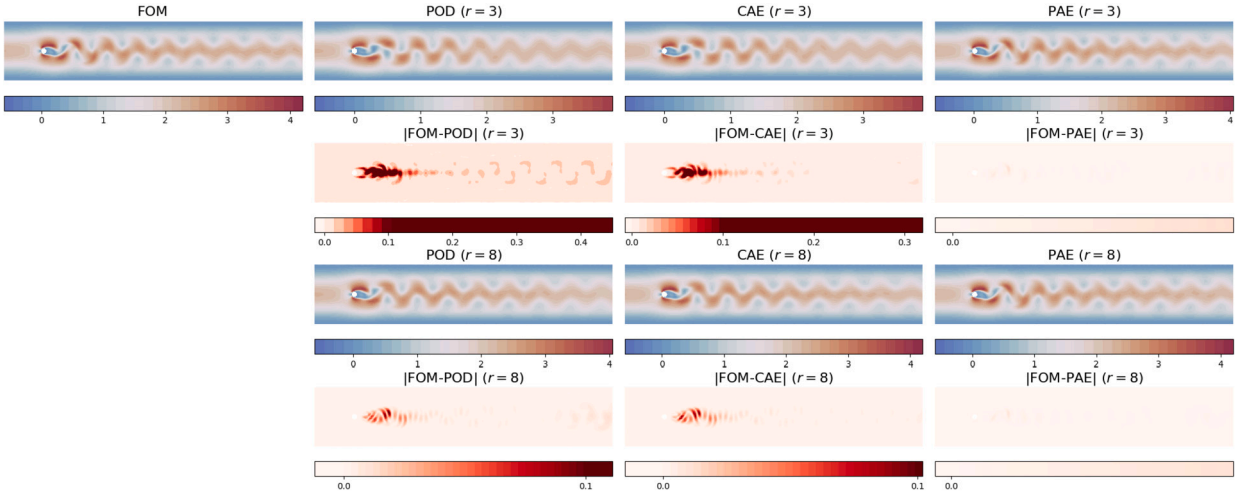

Fig. 7. Comparison of the reference generated by the full order model (FOM) and the developed snapshots of POD, CAE, and PAE at $t = 14.0$: evaluation session for the single cylinder case (Section 5.3).

Table 2
Computational times for $r = 3$: the offline time for POD denotes the runtime required to obtain a POD basis on CPU. For CAE and PAE, the offline time is the training time on GPU. The inference time refers to the runtime for reconstructing a state on CPU in the single cylinder case (Section 5.3).

Model	#epochs	offline time [s]	inference time [s]
POD	-	0.19	0.000236
CAE	800	63.19 (GPU)	0.000657
PAE(k = 3)	800	65.63 (GPU)	0.000756

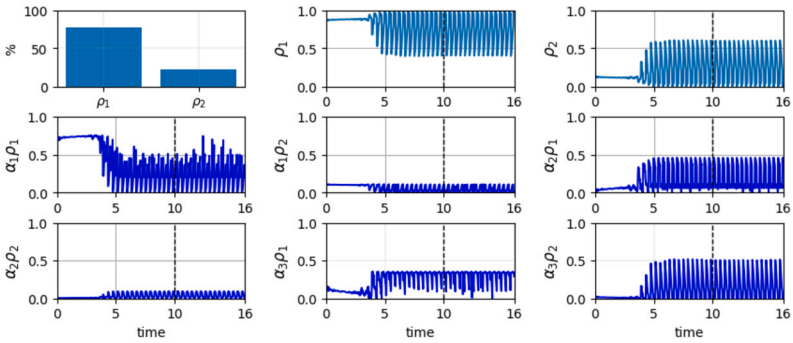

Fig. 8. Activation rates and trajectories of latent state variables when $r = 2$: the dashed line separates the training and the extrapolation phases for the single cylinder case (Section 5.3).

In Table 2, we report some computational timings[1] for the training autoencoders and their evaluation in comparison to the POD. The computation of the POD coefficients is about 340 times faster than the training of the neural networks that make up the CAE and PAEs. For reconstructing a state from its latent variables, the POD outperforms the other autoencoders by a factor of about 3.

Fig. 8 displays the activation rates of the vertices for a polytope and the trajectories of latent state variables when $r = 2$ with $k = 3$ (i.e., PAE3). It is shown that the latent variables for each PAE are within the range $[0, 1]$ as the coefficients of a convex combination.

The activation rate is a metric indicating the relative extent to which each latent variable influences the state reconstruction. The activation rate of the i-th latent variable is defined as

$$\frac{\sum_{j=1}^{N} \rho_{i,j}}{\sum_{l=1}^{r} \sum_{j=1}^{N} \rho_{l,j}}$$

[1] System specifications for the simulations: Intel i5-12600K CPU @ 3.70 GHz, 32 GB RAM and an NVIDIA RTX A4000 GPU.

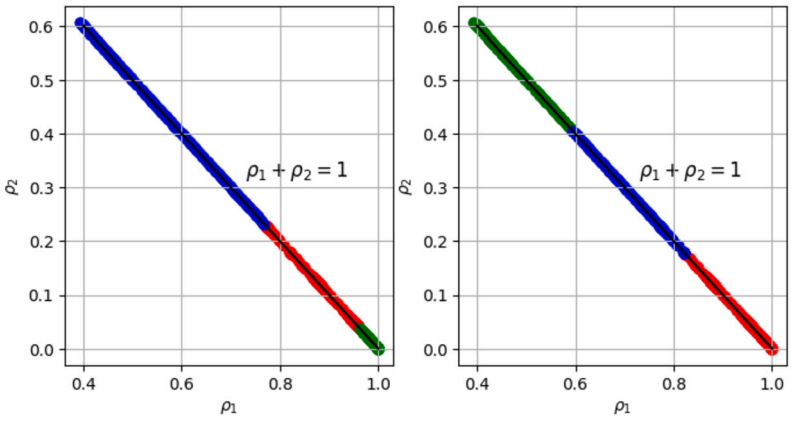

Fig. 9. Comparison of the results of (left) the smooth clustering $c(\rho)$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the single cylinder case (Section 5.3). The different colors denote different clusters.

where N is the number of snapshots. Since

$$\frac{\sum_{j=1}^{N}(\alpha_1\rho_{i,j} + \cdots \alpha_k\rho_{i,j})}{\sum_{l=1}^{r}\sum_{j=1}^{N}(\alpha_1\rho_{l,j} + \cdots \alpha_k\rho_{l,j})} = \frac{\sum_{j=1}^{N}(\alpha_1 + \cdots \alpha_k)\rho_{i,j}}{\sum_{l=1}^{r}\sum_{j=1}^{N}(\alpha_1 + \cdots \alpha_k)\rho_{l,j}} = \frac{\sum_{j=1}^{N}\rho_{i,j}}{\sum_{l=1}^{r}\sum_{j=1}^{N}\rho_{l,j}},$$

the activation rate of the polytope coefficients related to the i-th latent variable is identical to the activation rate of the i-th latent variable.

As shown in the bar chart, when $r = 2$, the state reconstruction overwhelmingly depends on the first latent variable accounting for 77.9%. Consequently, the three vertices weighted by ρ_1 of the polytope significantly influence the state reconstruction with a rate of 77.9%.

Fig. 9 confirms that the latent variables satisfy the convex combination constraints,

$$\rho_1(t) + \rho_2(t) = 1 \text{ and } \rho_1(t), \rho_2(t) \geq 0$$

and the clustering net c classifies latent variables similarly to k-means clustering. In other words, any latent variables lie on the line

$$\rho_1(t) + \rho_2(t) = 1, \forall t > 0.$$

when $r = 2$.

Fig. 10 shows the activation rates of the vertices for a polytope and the trajectories of latent state variables when $r = 3$ with $k = 3$ (i.e., PAE3). For any r, the encoder ensures that all latent variables are nonnegative, and the summation of the elements for each $\boldsymbol{\rho}(t)$ is 1. Thus, the trajectories of latent variables are within the expected range of $[0, 1]$. In the bar chart, the activation rates of each latent variable are 17.4%, 58.9%, and 23.7%. Consequently, the three vertices weighted by $\rho_2(t)$ of the polytope have a large impact on the state reconstruction compared to the others.

Fig. 11 displays the distribution of latent variables in a three-dimensional space, comparing labels obtained by the clustering net with those from k-means clustering. As latent variables represent the coefficients of a polytope, they are nonnegative and lie on the plane

$$\rho_1(t) + \rho_2(t) + \rho_3(t) = 1, \forall t > 0.$$

The clustering net tends to assign labels to latent variables in a manner similar to k-means clustering, as it utilizes pseudo-labels obtained from k-means clustering. However, the clustering net is less constrained by distances from centroids due to its training with the joint loss function outlined in Section 4.5.

5.5. Dataset: double cylinder

The double cylinder setups features rich and chaotic dynamics even for relatively low Reynolds numbers $\mathsf{Re} \in [40, 100]$. For our experiments we consider the flow at $\mathsf{Re} = 60$ and generate training data as laid out above. Now, each snapshot vector $\mathbf{v}(t)$ encompasses 46,014 states (i.e., $n = 46014$), corresponding to nodes in the FEM mesh within the spatial domain $(-20, 50) \times (-20, 20) \subset \mathbb{R}^2$.

The states in the time domain $[0, 240]$ As the initial value is rather unphysical and appeared to dominate the data in an unfavorable fashion, we allowed the flow to evolve first and only considered data points after the time $t = 240$. Again, the dataset is partitioned into a training set, comprising 720 snapshots in the time interval $[240, 480]$, and a test set, encompassing 432 snapshots within $[480, 760]$.

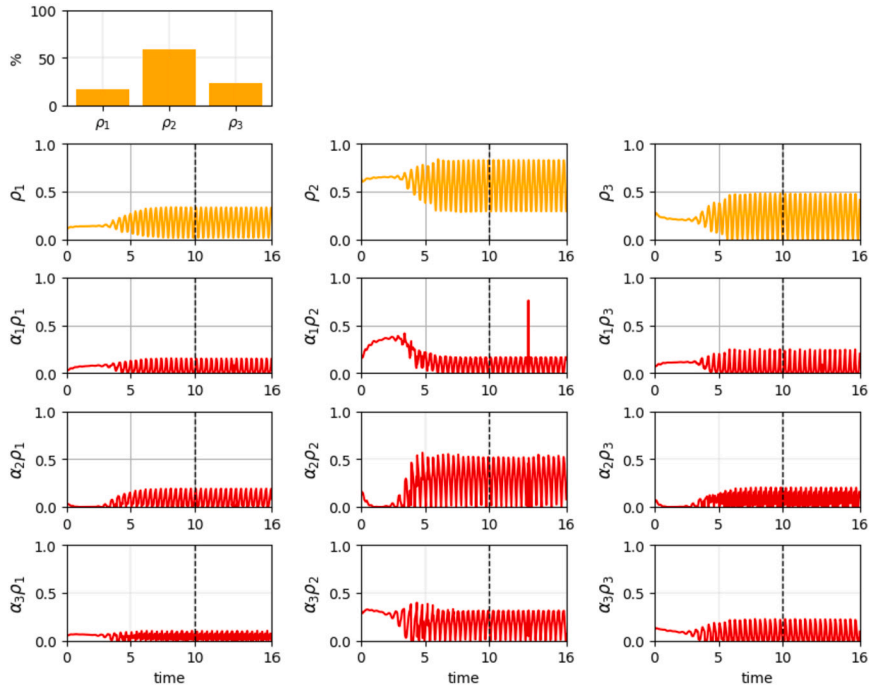

Fig. 10. Activation rates and trajectories of latent state variables when $r = 3$: the dashed line separates the training and the extrapolation phases for the single cylinder case (Section 5.3).

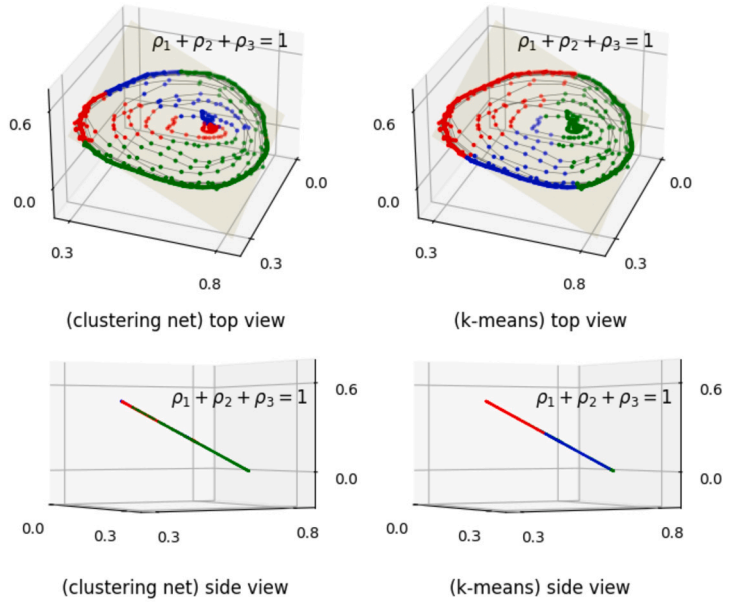

Fig. 11. Comparison of the results of (left) the smooth clustering $c(\rho)$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the single cylinder case as described in Section 5.3. The different colors denote different clusters.

When employing convolutional encoders, the interpolation matrix $\mathbf{I}_C \in \mathbb{R}^{46014\times 7505}$ transforms $\mathbf{v}(t) \in \mathbb{R}^{42764}$ into $\mathbf{v}_{\mathsf{CNN}}(t) \in \mathbb{R}^{2\times 95\times 79}$, representing the x-directional velocity and y-directional velocity at each mesh point on the rectangular grid of size 95×79.

5.6. PAEs: double cylinder

In this simulation, each PAE and CAE has a deep convolutional encoder consisting of 26 convolutional layers and a fully connected layer. The ELU activation function is applied to the convolutional layers and the modified softmax function (7) in the last layer. To reduce the number of nodes in the last layer, the global average pooling is used before performing the fully connected computation.

Table 3
Model information: the number of encoding (L_e) and decoding (L_d) layers, the number of encoding (P_e) and decoding (P_d) parameters, and the number R which counts the vertices of a bounding box in $\mathbb{R}^r$ (for POD) or of the polytopes used for the reconstruction for CAE and PAE for the double cylinder case (Section 5.6); see how to calculate P_e and P_d in Section 5.1.

Model	r	L_e	P_e	L_d	P_d	R
POD	2	1	92,028	1	92,028	4
CAE	2	27	91,410	1	92,028	2
PAE($k=3$)	2	27	91,410	3	460,140	6
POD	3	1	138,042	1	**138,042**	8
CAE	3	27	**91,443**	1	**138,042**	**3**
PAE($k=3$)	3	27	**91,443**	3	690,266	9
POD	5	1	230,070	1	**230,070**	32
CAE	5	27	**91,509**	1	**230,070**	**5**
PAE($k=3$)	5	27	**91,509**	3	690,210	15
POD	8	1	368,112	1	**368,112**	256
CAE	8	27	**91,608**	1	**368,112**	**8**
PAE($k=3$)	8	27	**91,608**	3	1,104,336	24

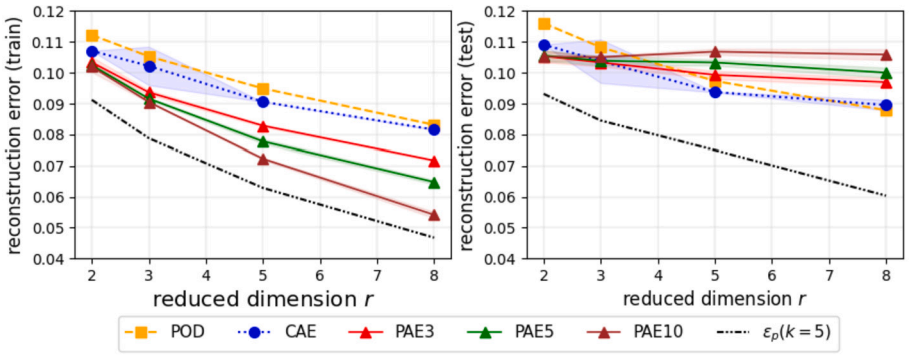

Fig. 12. Reconstruction error across the reduced dimension r averaged for 5 runs for the double cylinder case (Section 5.6). The shaded regions mark the statistical uncertainty measured through several training runs and appears to be insignificant and, thus, invisible in the plots for most methods.

In Table 3 we tabulate the different model parameters. Apparently, CAE and PAE maintain a relatively consistent number of encoding parameters regardless of reduced dimensions, in contrast with POD. In practice, when $r = 2, 3, 5, 8$, the encoding size of CAE and PAE is only $99.3\%, 66.2\%, 39.8\%$, and 24.9% of the encoding size of POD respectively.

According to their decoding sizes, for POD and CAE, their decoders are linear, resulting in sizes of nr, denoting the number of elements for a $n \times r$ matrix. the decoding size of PAE is $nrk + m$ where m is the number of parameters in the clustering net. In comparison with POD, CAE and PAE allow for the definition of a small number of R vertices for polytopic LPV representations. For training CAEs and PAEs, the Adam optimizer is used with a learning rate η of 10^{-4}, a batch size of 64; see Appendix C.

5.7. Results: double cylinder

In Fig. 12, PAE demonstrates superiority over POD and CAE across reduced dimensions in terms of the reconstruction performance for the training data. The reconstruction errors of PAE tend to significantly decrease as k gets larger and r increases.

However, PAEs fit the training data so well that the models lack generalization. Despite efforts to address overfitting through various regularization techniques including L_1 *regularization*, L_2 *regularization*, *Dropout* [47], *label smoothing* [48], and the application of fewer model parameters, the issue persists.

Nevertheless, the polytope errors ε_p with $k = 5$ reach a certain level in both the training and test phases. It indicates that each polytope defined by PAEs is well-constructed. Therefore, there is a potential to improve the model by tuning the encoding and the clustering parts involved in convex combination coefficients.

Fig. 13 and Fig. 14 display a comparison of the developed snapshots from FOM, POD, CAE, and PAE3 at time $t = 556.0, 736.5$ respectively when $r = 3, 8$. At $t = 556.0$, PAE3 reconstructs the state more clearly than the others. In the reconstructed snapshot obtained from POD with $r = 3$ at time $t = 736.5$, the cylinder wake exhibits a notably smoother flow compared to CAE and PAE3. When $r = 8$, there is no discernible difference among the applied methods.

In terms of computational efforts, we note that the training of the autoencoders takes about 500 times longer than the computation of the POD bases whereas the time needed for reconstruction differs by a factor of about 5.5, see Table 4.

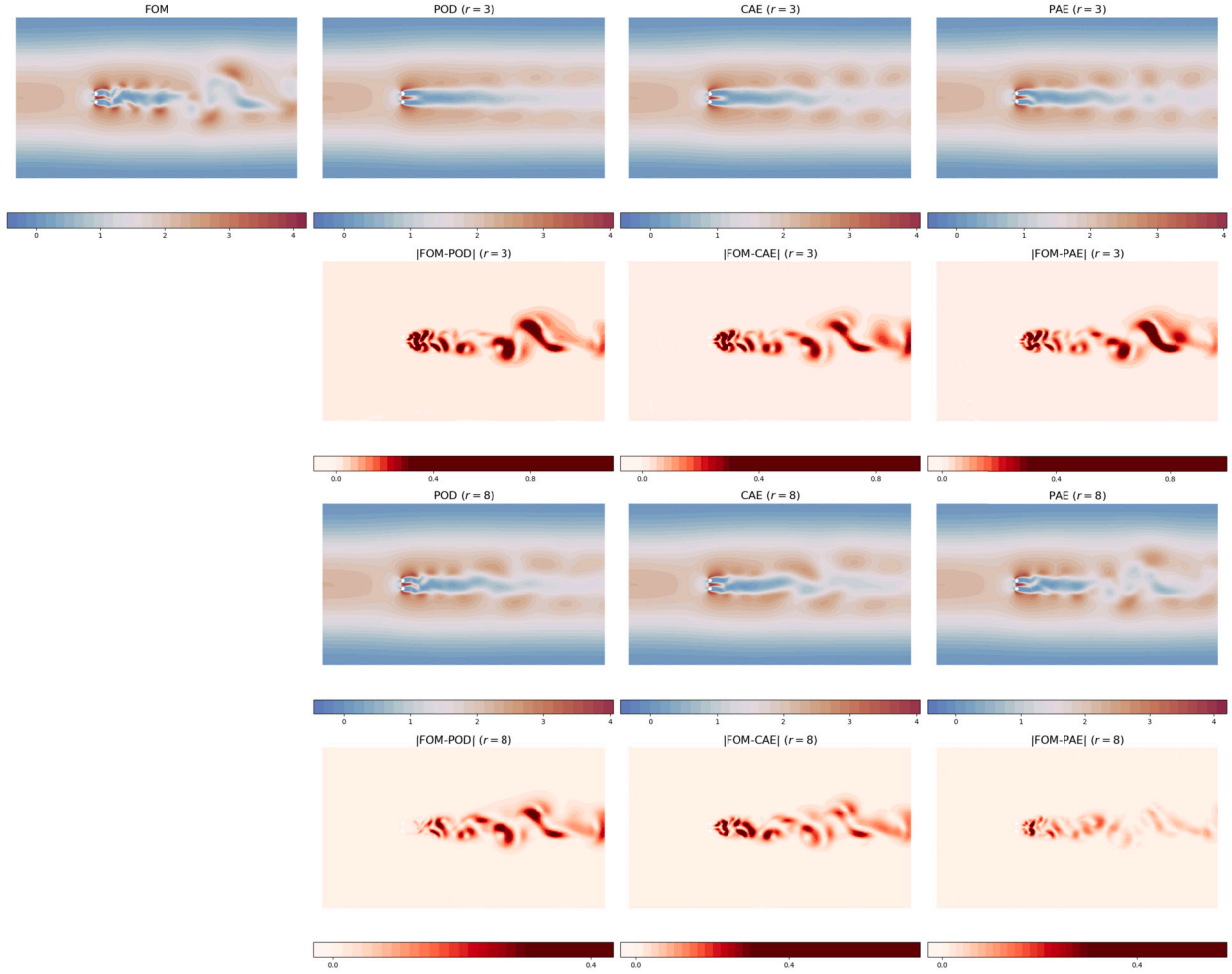

Fig. 13. Comparison of the reference generated by the full order model (FOM) and the developed snapshots of POD, CAE, and PAE at $t = 556.0$: training session for the double cylinder case (Section 5.6).

Table 4
Computational times for $r = 3$: the offline time for POD denotes the runtime required to obtain a POD basis on a CPU. For CAE and PAE, the offline time is the training time on a GPU. The inference time refers to the runtime for reconstructing a state on a CPU in the double cylinder case (Section 5.6).

Model	#epochs	offline time [s]	inference time [s]
POD	-	0.30	0.00023
CAE	1000	144.53 (GPU)	0.00125
PAE(k = 3)	1000	154.38 (GPU)	0.00135

In Fig. 15, it is evident that $\rho_2(t)$ exerts a predominant influence on state reconstruction, with an activation rate of 92.8%. Fig. 16 visually represents the latent variables satisfying convex combination constraints. Moreover, the clustering net c classifies latent variables in a manner reminiscent of k-means clustering. However, the clustering net assigns only two labels to the latent variables, in contrast to k-means clustering.

In Fig. 17, the activation rates are 17.6%, 11.8%, and 70.6% respectively. As a result, the substantial influence on state reconstruction comes from the three vertices that are weighted by $\rho_3(t)$ in the polytope surpassing the impact of others.

Fig. 18 shows the trajectory of the latent variables on the plane

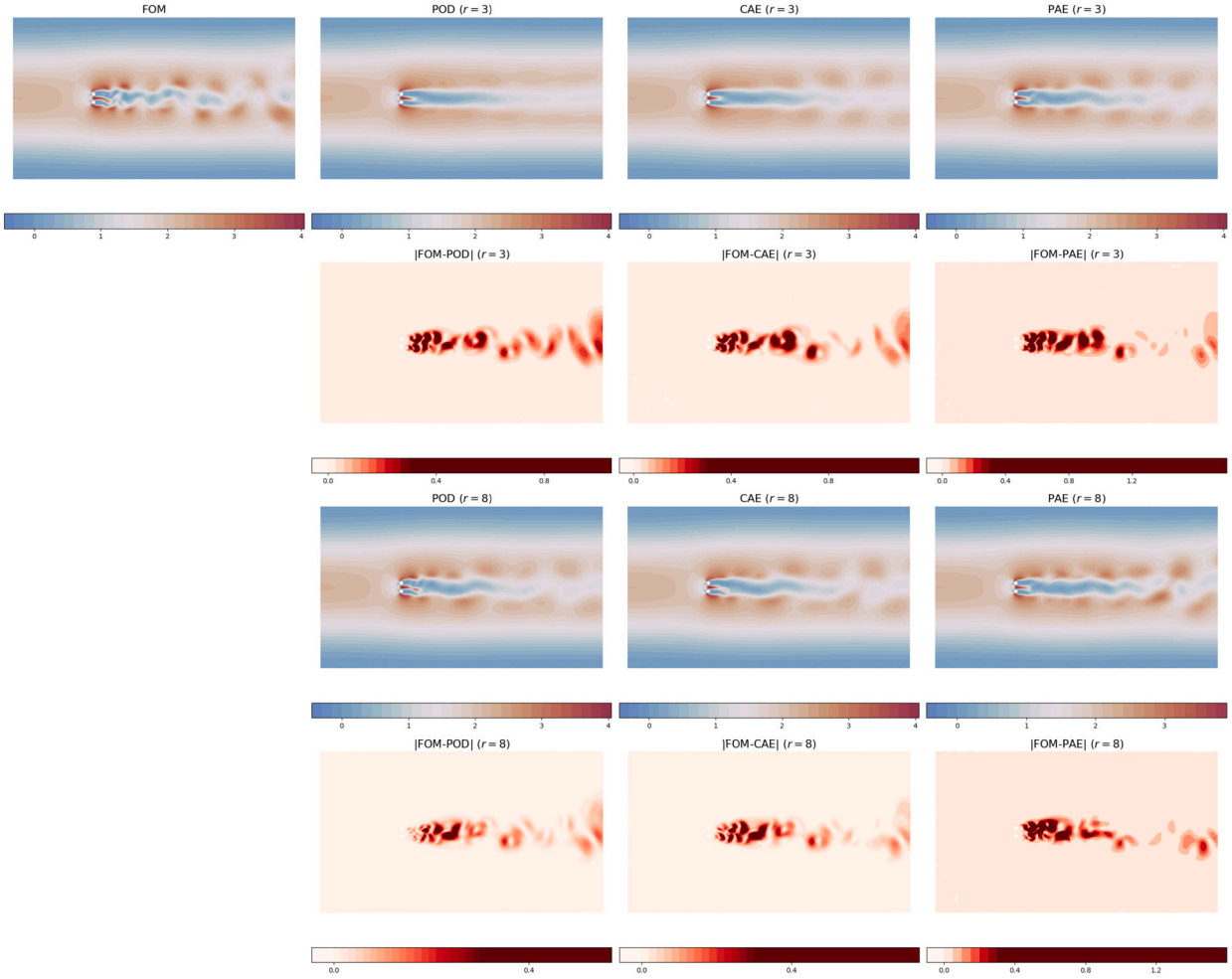

Fig. 14. Comparison of the reference generated by the full order model (FOM) and the developed snapshots of POD, CAE, and PAE at $t = 736.5$: evaluation session for the double cylinder case (Section 5.6).

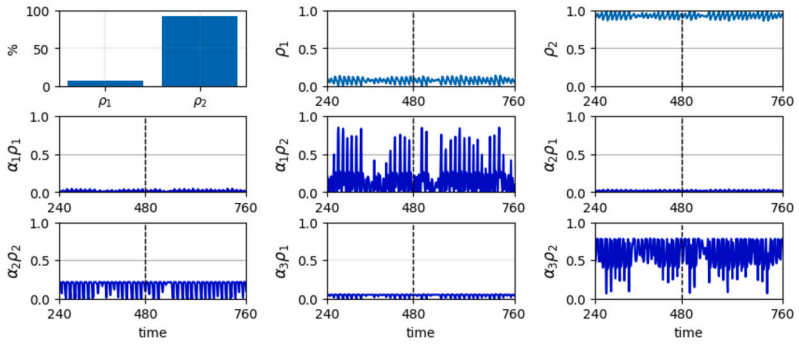

Fig. 15. Activation rates and trajectories of latent state variables when $r = 2$: the dashed line separates the training and the extrapolation phases for the double cylinder case (Section 5.6).

$$\rho_1(t) + \rho_2(t) + \rho_3(t) = 1, \forall t > 0.$$

Overall, The clustering net tends to classify latent variables in a manner similar to k-means clustering. Nevertheless, in certain instances, the clustering net assigns a label to them in defiance of distance-based methods.

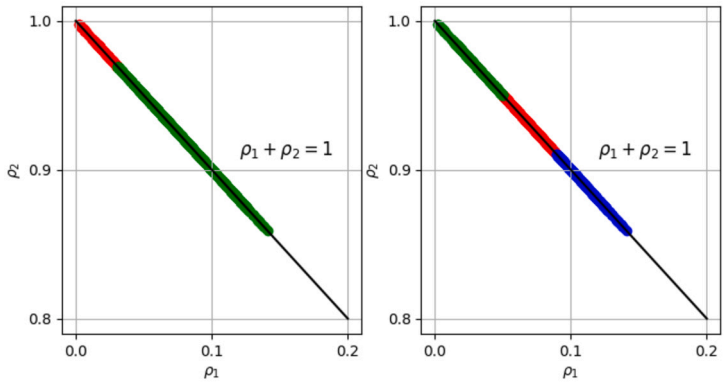

Fig. 16. Comparison of the results of (left) the smooth clustering $c(\rho)$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the double cylinder case (Section 5.6). The different colors denote different clusters.

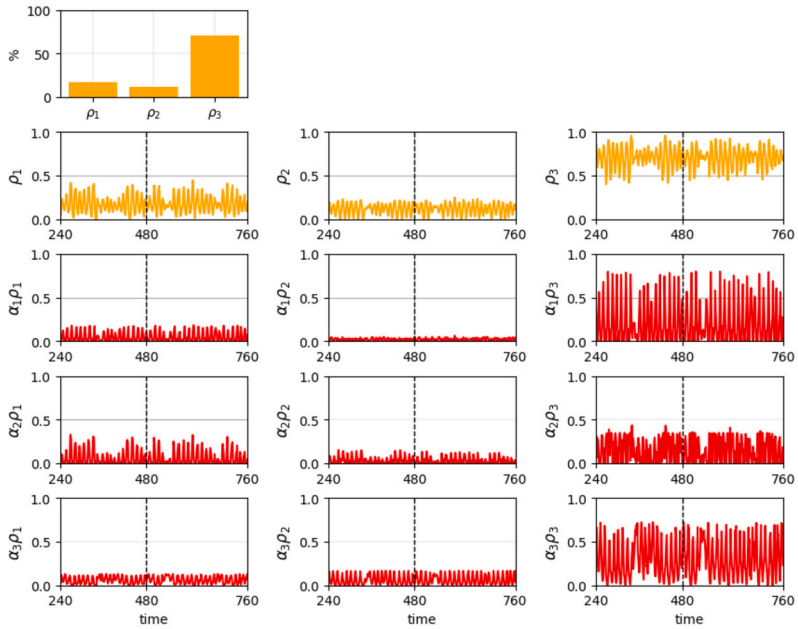

Fig. 17. Activation rates and trajectories of latent state variables when $r = 3$: the dashed line separates the training and the extrapolation phases for the double cylinder case (Section 5.6).

6. Conclusion

In this article, we proposed a polytopic autoencoder architecture consisting of a lightweight nonlinear encoder, a convex combination decoder, and a differentiable clustering network. We also showed how a differentiable clustering network embedded in the decoder improves state reconstruction errors. Notably, the model ensures that all reconstructed states reside within a polytope, with their latent variables serving directly as the convex combination coefficients.

To estimate the optimal performance of polytopes obtained by the proposed model, we measured polytope errors. From another perspective, we investigated the dominance of vertices in state reconstruction within a polytope by utilizing the activation rates of latent variables.

The results of the single cylinder case indicated that our model well outperforms POD in terms of reconstruction. For the more challenging dynamics of the double cylinder, despite a rather low error in the training regime and a rather low potential model mismatch as estimated by the approximation error in the polytope, the reconstruction deteriorated in the extrapolation regime. Nevertheless, we confirmed that the polytope is well-constructed in all the regimes. Thus, a potential avenue for further research is to enable this potential by revising the architecture or training strategy e.g., by including residuals in the loss functions or by extending the networks to better handle unseen clusters.

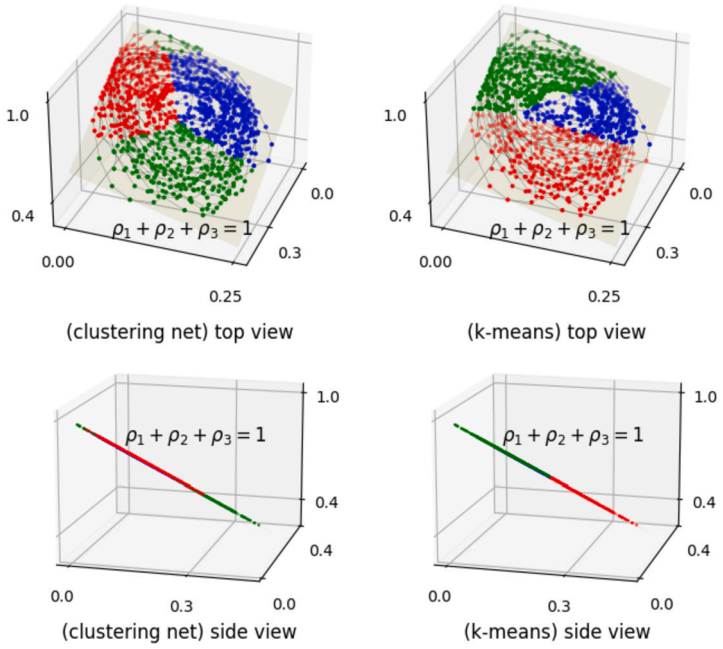

Fig. 18. Comparison of the results of (left) the smooth clustering $c(\rho)$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the double cylinder case (Section 5.6). The different colors denote different clusters.

CRediT authorship contribution statement

Jan Heiland: Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Yongho Kim:** Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

We acknowledge funding by the German Research Foundation (DFG) through the research training group 2297 “MathCoRe”, Magdeburg (project number 314838170).

Appendix A

The well-posedness of the polytope error (Definition 4.1) can be derived from the following three lemmas:

Lemma 6.1. *Let* $\mathbf{A}$ *be a positive definite matrix. Then*

$$\mathbf{x}^\top \mathbf{A}\mathbf{y} \le \sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}\sqrt{\mathbf{y}^\top \mathbf{A}\mathbf{y}}$$

for any nonzero vectors $\mathbf{x}$ *and* $\mathbf{y}$.

Proof. Since $\mathbf{A}$ is positive definite, we obtain the inequality

$$(\mathbf{x} - a\mathbf{y})^\top \mathbf{A}(\mathbf{x} - a\mathbf{y}) \ge 0$$
$$\to \mathbf{x}^\top \mathbf{A}\mathbf{x} - 2a\mathbf{x}^\top \mathbf{A}\mathbf{y} + a^2 \mathbf{y}^\top \mathbf{A}\mathbf{y} \ge 0$$

where a is a scalar. Let

$$p(a) = (\mathbf{y}^\top \mathbf{A}\mathbf{y})a^2 - 2(\mathbf{x}^\top \mathbf{A}\mathbf{y})a + (\mathbf{x}^\top \mathbf{A}\mathbf{x}).$$

Then we get $p(a) \ge 0$. Hence, the discriminant of $p(a)$ is less than 0 as follows:

$$(\mathbf{x}^\top \mathbf{A}\mathbf{y})^2 - (\mathbf{x}^\top \mathbf{A}\mathbf{x})(\mathbf{y}^\top \mathbf{A}\mathbf{y}) \leq 0$$
$$\rightarrow (\mathbf{x}^\top \mathbf{A}\mathbf{y})^2 \leq (\mathbf{x}^\top \mathbf{A}\mathbf{x})(\mathbf{y}^\top \mathbf{A}\mathbf{y})$$
$$\rightarrow -\sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}\sqrt{\mathbf{y}^\top \mathbf{A}\mathbf{y}} \leq \mathbf{x}^\top \mathbf{A}\mathbf{y} \leq \sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}\sqrt{\mathbf{y}^\top \mathbf{A}\mathbf{y}}$$

Thus,

$$\mathbf{x}^\top \mathbf{A}\mathbf{y} \leq \sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}\sqrt{\mathbf{y}^\top \mathbf{A}\mathbf{y}}.$$

When $\mathbf{y} \neq k\mathbf{x}$ with a scalar k,

$$\mathbf{x}^\top \mathbf{A}\mathbf{y} < \sqrt{\mathbf{x}^\top \mathbf{A}\mathbf{x}}\sqrt{\mathbf{y}^\top \mathbf{A}\mathbf{y}}. \quad \square$$

Lemma 6.2. *Let* $\mathbf{x}$ *and* $\mathbf{y}$ *be vectors in a convex polytope* $\tilde{\mathcal{V}}$. *If* $\mathbf{x} \neq \mathbf{y}$ *then* $\mathbf{y} \neq k\mathbf{x}$ *with a scalar* k.

Proof. We deal with the contrapositive of the above statement. Suppose that $\mathbf{y} = k\mathbf{x}$. Then $\exists\ c_i \geq 0$, $\forall i$, and $\sum_i c_i = 1$ such that

$$\mathbf{x} = \sum_i c_i \mathbf{u}_i \in \tilde{\mathcal{V}},\ \mathbf{y} = k \sum_i c_i \mathbf{u}_i$$

where $\mathbf{u}_i$ is the i-th vertex of $\tilde{\mathcal{V}}$. Since $\mathbf{y} \in \tilde{\mathcal{V}}$,

$$k \sum_i c_i = 1 \text{ (i.e., } k = 1).$$

Thus, $\mathbf{x} = \mathbf{y}$. $\square$

Lemma 6.3. *Let* $\tilde{\mathcal{V}} \subset \mathbb{R}^n$ *be a convex polytope. Then for any* $\mathbf{v} \in \mathbb{R}^n \setminus \tilde{\mathcal{V}}$ *there exists a unique* $\mathbf{v}^* \in \tilde{\mathcal{V}}$ *such that*

$$\|\mathbf{v} - \mathbf{v}^*\|_{\mathbf{M}} = \min\{\|\mathbf{v} - \tilde{\mathbf{v}}\|_{\mathbf{M}} :\ \tilde{\mathbf{v}} \in \tilde{\mathcal{V}}\}.$$

Proof. Define

$$f(\mathbf{x}) = \|\mathbf{x}\|_{\mathbf{M}}, \mathbf{x} \in \tilde{\mathcal{V}}.$$

We first consider the existence of $\mathbf{v}^*$. Since $\tilde{\mathcal{V}}$ are compact and f is continuous, $\exists \mathbf{v}^* \in \tilde{\mathcal{V}}$ such that

$$f(\mathbf{v}^*) \leq f(\mathbf{x}), \forall \mathbf{x} \in \tilde{\mathcal{V}}.$$

To show the uniqueness of $\mathbf{v}^*$, we first prove that f is strictly convex.

$$\begin{aligned} f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) &= \sqrt{(\lambda\mathbf{x} + (1-\lambda)\mathbf{y})^\top \mathbf{M}(\lambda\mathbf{x} + (1-\lambda)\mathbf{y})} \\ &= \sqrt{(\lambda^2\{f(\mathbf{x})\}^2 + 2\lambda(1-\lambda)\mathbf{x}^\top \mathbf{M}\mathbf{y} + (1-\lambda)^2 f(\mathbf{y})\}^2} \\ &< \sqrt{(\lambda^2\{f(\mathbf{x})\}^2 + 2\lambda(1-\lambda)f(\mathbf{x})f(\mathbf{y}) + (1-\lambda)^2\{f(\mathbf{y})\}^2} \ (\because \text{Lemmas 6.1, 6.2}) \\ &= \sqrt{(\lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}))^2} \\ &= \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y}) \end{aligned}$$

where $0 < \lambda < 1$, $\mathbf{x} \neq \mathbf{y}$, and $\forall \mathbf{x}, \mathbf{y} \in \tilde{\mathcal{V}}$.
Hence, we obtain the inequality

$$f(\lambda\mathbf{x} + (1-\lambda)\mathbf{y}) < \lambda f(\mathbf{x}) + (1-\lambda)f(\mathbf{y})$$

Therefore, f is strictly convex.
To show the uniqueness of $\mathbf{v}^*$ by contradiction, suppose $\exists \mathbf{w} \in \tilde{\mathcal{V}}, \mathbf{v}^* \neq \mathbf{w}$ such that

$$f(\mathbf{v}^*) = f(\mathbf{w})$$

Since f is strictly convex and $\mathbf{v}^* \neq \mathbf{w}$,

$$f(\lambda\mathbf{v}^* + (1-\lambda)\mathbf{w}) < \lambda f(\mathbf{v}^*) + (1-\lambda)f(\mathbf{w}), 0 < \lambda < 1.$$

Since $f(\mathbf{v}^*) = f(\mathbf{w})$,

Algorithm 1 Three-step training strategy for PAEs.

Step 0. Prepare the Model
Set hyperparameters like r, k, and the number of epochs N_1, N_2, N_3
Initialize model parameters θ for the corresponding models (e.g., θ_μ, θ_φ, θ_c). Note that, e.g., $\boldsymbol{\rho} = \mu(\theta_\mu; \mathbf{v})$ denotes the evaluation with the current values of the parameters
Generate CNN inputs $\mathbf{v}_{\mathsf{CNN}} = \mathbf{I}_C \mathbf{v}$
Step 1. Train a CAE with encoder μ and decoder $\bar{\varphi}$
for $e = 1, \cdots, N_1$ **do**
  for batch iterations: $i \in B_1$ **do**
    Compute $\boldsymbol{\rho} = \mu(\theta_\mu; \mathbf{v}_{\mathsf{CNN}})$
    Compute $\tilde{\mathbf{v}} = \bar{\varphi}(\theta_\varphi; \boldsymbol{\rho})$
    Compute $\mathcal{L}_{\mathsf{rec}}(\theta_\mu, \theta_{\bar{\varphi}}; \mathbf{v}_{\mathsf{CNN}}) = \frac{1}{|B_1|} \sum_{i \in B_1} \| \tilde{\mathbf{v}}(\theta_\mu, \theta_{\bar{\varphi}}; \mathbf{v}_{\mathsf{CNN}}{}^{(i)}) - \mathbf{v}^{(i)} \|_{\mathbf{M}}$
    Update model parameters θ_μ and $\theta_{\bar{\varphi}}$
  end for
end for
Step 2. Train individual decoders, $\varphi_1, \cdots, \varphi_k$
Freeze the weights θ_μ of μ
for $l = 1, \cdots, k$ **do**
  Select small batch data based on partial k-means clustering and save their labels $\mathbf{l}$
  for $e = 1, \cdots, N_2$ **do**
    for batch iterations: $i \in B_2$ **do**
      Compute $\boldsymbol{\rho} = \mu(\mathbf{v}_{\mathsf{CNN}})$
      Compute $\tilde{\mathbf{v}} = \varphi^{(l)}(\theta_{\varphi^{(l)}}; \boldsymbol{\rho})$
      Compute $\mathcal{L}_{\mathsf{rec}}(\theta_{\varphi^{(l)}}; \boldsymbol{\rho}) = \frac{1}{|B_2|} \sum_{i \in B_2} \|\tilde{\mathbf{v}}(\theta_{\varphi^{(l)}}; \boldsymbol{\rho}^{(i)}) - \mathbf{v}^{(i)}\|_{\mathbf{M}}$
      Update model parameters θ_{φ_l}
    end for
  end for
end for
Step 3. Train a PAE containing μ, φ, and c
Define a matrix $\mathbf{U} = [\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}]$ ($\theta_\varphi = \{\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}\}$)
for $e = 1, \cdots, N_3$ **do**
  for batch iterations: $i \in B_3$ **do**
    Compute $\boldsymbol{\rho} = \mu(\theta_\mu; \mathbf{v}_{\mathsf{CNN}})$
    Compute $\tilde{\mathbf{v}} = \varphi(\theta_\varphi; \boldsymbol{\rho}) = \mathbf{U}(c(\theta_c; \boldsymbol{\rho}) \otimes \boldsymbol{\rho})$
    Compute $\mathcal{L}_{\mathsf{rec}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}) = \frac{1}{|B_3|} \sum_{i \in B_3} \| \tilde{\mathbf{v}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}^{(i)}) - \mathbf{v}^{(i)} \|_{\mathbf{M}}$
    Compute $\mathcal{L}_{\mathsf{clt}}(\theta_\mu, \theta_c; \boldsymbol{\rho}) = -\frac{1}{|P|} \sum_{j \in P \subset B_3} \mathbf{l}^{(j)} \cdot \log(c(\theta_\mu, \theta_c; \boldsymbol{\rho}^{(j)}))$
    Compute $\mathcal{L} = \mathcal{L}_{\mathsf{rec}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}) + 10^{-4} \mathcal{L}_{\mathsf{clt}}(\theta_\mu, \theta_c; \boldsymbol{\rho})$
    Update model parameters θ_μ, θ_φ, and θ_c
  end for
end for

$$
\begin{aligned}
f(\lambda \mathbf{v}^* + (1-\lambda)\mathbf{w}) &< \lambda f(\mathbf{v}^*) + (1-\lambda) f(\mathbf{w}) \\
&= \lambda f(\mathbf{v}^*) + (1-\lambda) f(\mathbf{v}^*) \\
&= f(\mathbf{v}^*)
\end{aligned}
$$

$$\therefore f(\mathbf{v}^*) > f(\lambda \mathbf{v}^* + (1-\lambda)\mathbf{w}).$$

Then it contradicts $f(\mathbf{v}^*)$ is a minimum of f. Thus, $\mathbf{v}^* = \mathbf{w}$. (i.e., $\mathbf{v}^*$ is unique.)
Since $\|\mathbf{x}\|_{\mathbf{M}}$ has a unique minimum, a parallel translation of the norm, $\|\mathbf{v} - \mathbf{x}\|_{\mathbf{M}}$, also has a unique minimum. □

Appendix B

A depthwise convolution involves applying a $K \times K$ convolution operation to each input channel independently, resulting in a total number of parameters equal to $K \times K \times 1 \times C_I$. The convolution captures spatial information within each input channel. Pointwise convolution is a convolution with a kernel size of $1 \times 1 \times C_O \times C_I$. It combines the output from the depthwise convolution to create the final feature map. Thus, the total number of parameters is $K \times K \times C_I + C_O \times C_I$, resulting in a parameter reduction rate denoted as

$$\frac{K \times K \times C_I + C_O \times C_I}{K \times K \times C_O \times C_I} = \frac{1}{C_O} + \frac{1}{K^2}.$$

For example, when $K = 3$ and $C_O = 8$, the convolution parameters are only 23% of those in the $3 \times 3 \times 8 \times C_I$ standard convolution. Finally, an encoder μ contains deep inverted residual blocks:

$$h_1 = a(\mathrm{CV}_{3\times3}(\mathbf{v}_{\mathsf{CNN}}))$$

$$h_l = \mathrm{RB}(h_{l-1}), l = 2, \cdots, L-1$$

$$h'_{L-1} = \mathrm{GAP}(h_{L-1})$$
$$h_L = \sigma(\mathrm{LN}(h'_{L-1}))$$

where $a(\cdot)$, $CV_{3\times3}$, RB, GAP, and LN are a nonlinear activation function, a 3×3 standard convolution, an inverted residual block, a global average pooling, and a linear layer respectively. RB generates feature maps that match the size of the inputs without including any bias terms to minimize the number of model parameters, except for downsampling in specific layers. When downsampling occurs, RB omits computing a residual connection, reducing the size of feature maps by adjusting the stride from 1 to 2. Therefore, the final $C_{final} \times H_{final} \times W_{final}$ feature map h_{L-1} is obtained where $(H_{final} \times W_{final})$ is a smaller size, but a channel dimension C_{final} is larger than the input size $C \times H \times W$.

Appendix C

We train PAEs through the following three steps (Algorithm 1):

- **Step 1**: Initialization and identification of the latent space
- **Step 2**: Construction of polytopes for state reconstruction in each cluster
- **Step 3**: Construction of a polytope for state reconstruction

See Section 4.5 for explanations and definitions of the involved quantities.

Appendix D. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcp.2024.113526.

Data availability

The paper includes a link to access the source code.

References

[1] C.C. Sullivan, H. Yamashita, H. Sugiyama, Reduced order modeling of deformable tire-soil interaction with proper orthogonal decomposition, J. Comput. Nonlinear Dyn. 17 (5) (2022) 051009, https://doi.org/10.1115/1.4053592.
[2] N. Lauzeral, D. Borzacchiello, M. Kugler, D. George, Y. Rémond, A. Hostettler, F. Chinesta, A model order reduction approach to create patient-specific mechanical models of human liver in computational medicine applications, Comput. Methods Programs Biomed. 170 (2019) 95–106, https://doi.org/10.1016/j.cmpb.2019.01.003.
[3] M. Calka, P. Perrier, J. Ohayon, C. Grivot-Boichon, M. Rochette, Y. Payan, Machine-learning based model order reduction of a biomechanical model of the human tongue, Comput. Methods Programs Biomed. 198 (2021) 105786, https://doi.org/10.1016/j.cmpb.2020.105786.
[4] V.B. Nguyen, S.B.Q. Tran, S.A. Khan, J. Rong, J. Lou, POD-DEIM model order reduction technique for model predictive control in continuous chemical processing, Comput. Chem. Eng. 133 (2020) 106638, https://doi.org/10.1016/j.compchemeng.2019.106638.
[5] F. Matter, I. Iroz, P. Eberhard, Methods of model order reduction for coupled systems applied to a brake disc-wheel composite, PAMM 22 (1) (2023) e202200323, https://doi.org/10.1002/pamm.202200323.
[6] G. Berkooz, P. Holmes, J.L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annu. Rev. Fluid Mech. 25 (1) (1993) 539–575, https://doi.org/10.1146/annurev.fl.25.010193.002543.
[7] N. Halko, P. Martinsson, J.A. Tropp, Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288, https://doi.org/10.1137/090771806.
[8] M. Ohlberger, S. Rave, Reduced basis methods: success, limitations and future challenges, in: Proceedings of the Conference Algoritmy, 2016, pp. 1–12, https://arxiv.org/abs/1511.02021.
[9] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, arXiv, https://arxiv.org/abs/2003.05991, 2020.
[10] I.J. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, Cambridge, MA, USA, 2016, http://www.deeplearningbook.org.
[11] D.P. Kingma, M. Welling, Auto-encoding variational bayes, in: 2nd ICLR 2014, Conference Track Proceedings, 2014, http://arxiv.org/abs/1312.6114.
[12] P.J.W. Koelewijn, R. Tóth, Scheduling dimension reduction of LPV models - a deep neural network approach, in: Proceedings of the IEEE, 2020, pp. 1111–1117.
[13] H. Eivazi, H. Veisi, M.H. Naderi, V. Esfahanian, Deep neural networks for nonlinear model order reduction of unsteady flows, Phys. Fluids 32 (10) (2020) 105104, https://doi.org/10.1063/5.0020526.
[14] T.R.F. Phillips, C.E. Heaney, P.N. Smith, C.C. Pain, An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion, Int. J. Numer. Methods Eng. 122 (15) (2021) 3780–3811, https://doi.org/10.1002/nme.6681.
[15] R. Maulik, B. Lusch, P. Balaprakash, Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders, Phys. Fluids 33 (3) (2021) 037106, https://doi.org/10.1063/5.0039986.
[16] S. Fresca, A. Manzoni, POD-DL-ROM: enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition, Comput. Methods Appl. Mech. Eng. 388 (2022) 114181, https://doi.org/10.1016/j.cma.2021.114181.
[17] P. Buchfink, S. Glas, B. Haasdonk, Symplectic model reduction of hamiltonian systems on nonlinear manifolds and approximation with weakly symplectic autoencoder, SIAM J. Sci. Comput. 45 (2) (2023) A289–A311, https://doi.org/10.1137/21M1466657.
[18] X. Zhang, L. Jiang, Conditional variational autoencoder with Gaussian process regression recognition for parametric models, J. Comput. Appl. Math. 438 (2024) 115532, https://doi.org/10.1016/j.cam.2023.115532.
[19] J. Duan, J.S. Hesthaven, Non-intrusive data-driven reduced-order modeling for time-dependent parametrized problems, J. Comput. Phys. 497 (2024) 112621, https://doi.org/10.1016/j.jcp.2023.112621.
[20] D. Amsallem, B. Haasdonk, PEBL-ROM: projection-error based local reduced-order models, Adv. Model. Simul. Eng. Sci. 3 (1) (2016), https://doi.org/10.1186/s40323-016-0059-7.

[21] D. Amsallem, M.J. Zahr, C. Farhat, Nonlinear model order reduction based on local reduced-order bases, Int. J. Numer. Methods Eng. 92 (10) (2012) 891–916, https://doi.org/10.1002/nme.4371.
[22] R. Dupuis, J.-C. Jouhaud, P. Sagaut, Surrogate modeling of aerodynamic simulations for multiple operating conditions using machine learning, AIAA J. 56 (9) (2018) 3622–3635, https://doi.org/10.2514/1.J056405.
[23] Y.-E. Kang, S. Shon, K. Yee, Local non-intrusive reduced order modeling based on soft clustering and classification algorithm, Int. J. Numer. Methods Eng. 123 (10) (2022) 2237–2261, https://doi.org/10.1002/nme.6934.
[24] D. Arthur, S. Vassilvitskii, K-means++: the advantages of careful seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, Society for Industrial and Applied Mathematics, USA, 2007, pp. 1027–1035.
[25] J.C. Dunn, A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters, J. Cybern. 3 (3) (1973) 32–57, https://doi.org/10.1080/01969727308546046.
[26] J. Heiland, Y. Kim, Convolutional autoencoders and clustering for low-dimensional parametrization of incompressible flows, IFAC-PapersOnLine 55 (30) (2022) 430–435, https://doi.org/10.1016/j.ifacol.2022.11.091, proceedings of the 25th International Symposium on MTNS.
[27] M. Cho, K. Alizadeh-Vahid, S. Adya, M. Rastegari, DKM: differentiable k-means clustering layer for neural network compression, in: The 10th ICLR 2022, OpenReview.net, 2022, https://openreview.net/forum?id=J_F_qqCE3Z5.
[28] A. Genevay, G. Dulac-Arnold, J. Vert, Differentiable deep clustering with cluster size constraints, arXiv, http://arxiv.org/abs/1910.09036, 2019.
[29] M.M. Fard, T. Thonet, É. Gaussier, Deep k-means: jointly clustering with k-means and learning representations, Pattern Recognit. Lett. 138 (2020) 185–192, https://doi.org/10.1016/J.PATREC.2020.07.028.
[30] P. Apkarian, P. Gahinet, G. Becker, Self-scheduled H_∞ control of linear parameter-varying systems: a design example, Automatica 31 (9) (1995) 1251–1261, https://doi.org/10.1016/0005-1098(95)00038-X.
[31] J.C. Geromel, P. Colaneri, Robust stability of time varying polytopic systems, Syst. Control Lett. 55 (1) (2006) 81–85, https://doi.org/10.1016/J.SYSCONLE.2004.11.016.
[32] S.M. Hashemi, H. Werner, LPV modelling and control of Burgers' equation, IFAC Proc. Vol. 44 (1) (2011) 5430–5435, https://doi.org/10.3182/20110828-6-IT-1002.03318.
[33] M. Trudgen, S.Z. Rizvi, J. Mohammadpour, Linear parameter-varying approach for modeling rapid thermal processes, in: 2016 ACC, 2016, pp. 3243–3248.
[34] S.Z. Rizvi, F. Abbasi, J.M. Velni, Model reduction in linear parameter-varying models using autoencoder neural networks, in: 2018 Annual American Control Conference, 2018, pp. 6415–6420.
[35] K. Lee, K.T. Carlberg, Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders, J. Comput. Phys. 404 (2020), https://doi.org/10.1016/J.JCP.2019.108973.
[36] F. Pichi, B. Moya, J.S. Hesthaven, A graph convolutional autoencoder approach to model order reduction for parametrized PDEs, http://arxiv.org/abs/2305.08573, 2023.
[37] J. Heiland, P. Benner, R. Bahmani, Convolutional neural networks for very low-dimensional LPV approximations of incompressible Navier-Stokes equations, Front. Appl. Math. Stat. 8 (2022) 879140, https://doi.org/10.3389/fams.2022.879140.
[38] M. Sandler, A.G. Howard, M. Zhu, A. Zhmoginov, L. Chen, MobileNetV2: inverted residuals and linear bottlenecks, in: 2018 IEEE Conference on CVPR 2018, 2018, pp. 4510–4520.
[39] D. Sculley, Web-scale k-means clustering, in: WWW'10, 2010, pp. 1177–1178.
[40] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, in: ICLR 2015, Conference Track Proceedings, 2015, http://arxiv.org/abs/1412.6980.
[41] M. Andersen, J. Dahl, Z. Liu, L. Vandenberghe, Interior-point methods for large-scale cone programming, in: Optimization for Machine Learning, 2011.
[42] A. Das, J. Heiland, Low-order linear parameter varying approximations for nonlinear controller design for flows, Tech. rep., 2023, submitted to ECC2024, https://arxiv.org/abs/2311.05305.
[43] A. Kwiatkowski, H. Werner, PCA-based parameter set mappings for LPV models with fewer parameters and less overbounding, IEEE Trans. Control Syst. Technol. 16 (4) (2008) 781–788, https://doi.org/10.1109/TCST.2007.903094.
[44] M. Behr, P. Benner, J. Heiland, Example setups of Navier-Stokes equations with control and observation: spatial discretization and representation via linear-quadratic matrix coefficients, arXiv, https://arxiv.org/abs/1707.08711, 2017.
[45] R. Altmann, J. Heiland, Finite element decomposition and minimal extension for flow equations, ESAIM Math. Model. Numer. Anal. 49 (5) (2015) 1489–1509, https://doi.org/10.1051/m2an/2015029.
[46] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), ICLR 2016 (Poster), https://arxiv.org/pdf/1511.07289.pdf, 2016.
[47] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958, https://doi.org/10.5555/2627435.2670313.
[48] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: 2016 IEEE Conference on CVPR 2016, IEEE Computer Society, 2016, pp. 2818–2826.