Autoencoder Frameworks for Low-dimensional Parametrizations of Fluid Flows

Dissertation

zur Erlangung des akademischen Grades

doctor rerum naturalium
(Dr. rer. nat.)

von **M. Sc. Yongho Kim**

geb. am **02.10.1986** in Seoul, Republik Korea

genehmigt durch die Fakultät für Mathematik

der Otto-von-Guericke-Universität Magdeburg

Gutachter: **Prof. Dr. Thomas Richter**

Dr. Jan Heiland

eingereicht am: **25.08.2025**

Verteidigung am: **13.10.2025**

PUBLICATIONS

The papers presented in this thesis have been either published or submitted for publication in either journals or conferences proceedings.

- The contents of Chapter 3 are based on the material published in

  [56]: J. Heiland and Y. Kim, Convolutional Autoencoders and Clustering for Low-dimensional Parametrization of Incompressible Flows, *Proceeding of the 25th International Symposium on MTNS*, *IFAC-PapersOnLine*, 55(30):430-435, 2022. doi:10.1016/j.ifacol.2022.11.091.

  [57]: J. Heiland and Y. Kim, Convolutional Autoencoders, Clustering, and POD for Low-dimensional Parametrization of Flow Equations, *Comput. Math. Appl.*, vol. 175, pp. 49-61, 2024. doi:10.1016/j.camwa.2024.08.032.

- Chapter 4 is based on the material introduced in

  [58]: J. Heiland and Y. Kim, Polytopic Autoencoders for Very Low-dimensional Parametrizations of Fluid Flow Models, *Proceeding of the 94th GAMM Annual Meeting*, *PAMM*, vol. 25, no. 2 e70008, 2025. doi:10.1002/pamm.70008.

  [59]: J. Heiland and Y. Kim, Polytopic Autoencoders with Smooth Clustering for Reduced-order Modeling of Flows, *J. Comput. Phys.*, vol. 521, pp. 113526, 2025. doi:10.1016/j.jcp.2024.113526.

- Section 2.4 introduces the datasets used in these papers [56, 57, 58, 59].

Note that polytopic autoencoders have been applied to a feedback control system, as presented in

[60]: J. Heiland, Y. Kim, and S.W.R. Werner, Deep polytopic autoencoders for low-dimensional linear parameter-varying approximations and nonlinear feedback design, *Arxiv*, 2025 doi:10.48550/arXiv.2403.18044. (*accepted for publication in Adv. Comput. Math.*)

However, since that paper primarily focuses on nonlinear feedback control design rather than low-dimensional parameterizations, its contents are excluded from the scope of this thesis.

The associated simulation code can be accessed through the following links:

- Convolutional Autoencoders and Clustering for Low-dimensional Parametrization of Incompressible Flows

  doi:10.5281/zenodo.6817442

- Convolutional Autoencoders, Clustering, and POD for Low-dimensional Parametrization of Flow Equations

  doi:10.5280/zenodo.8340035

- Polytopic Autoencoders for Very Low-dimensional Parametrizations of Fluid Flow Models

  doi:10.5281/zenodo.11074521

- Polytopic Autoencoders with Smooth Clustering for Reduced-order Modeling of Flows

  doi:10.5281/zenodo.10491870

- Deep Polytopic Autoencoders for Low-dimensional Linear Parameter-varying Approximations and Nonlinear Feedback Design

  doi:10.5281/zenodo.10783695

ABSTRACT

Complex dynamical systems often encounter implementation challenges such as infeasibility and high computational cost due to their large dimensionality. A common remedy is to design low-dimensional parametrizations of the system states to enable efficient modeling and computation. Among classical techniques, Proper Orthogonal Decomposition (POD) has been widely adopted across various complex systems owing to its advantageous properties including its linearity, its optimality, and the orthogonality of the POD basis for reduced-order modeling. However, its approximation capability is inherently limited by the Kolmogorov n-width, which constrains the efficiency of dimension reduction in certain cases. As an alternative to POD, nonlinear autoencoders have been extensively developed over the past decade, with their effectiveness demonstrated in a range of tasks. Concurrently, it is also known that employing local bases tailored to different regions of the state space can improve reconstruction quality.

Motivated by these insights, we propose a deep clustering architecture that integrates a nonlinear autoencoder with a clustering model. In the first proposed deep clustering model, a nonlinear encoder extracts low-dimensional latent representations that serve as meaningful feature maps. After obtaining the latent states, k-means clustering is applied to partition them into clusters, assigning a label to each latent state. For reconstruction, a suitable local mode is selected based on the corresponding cluster label, enabling accurate and efficient approximation of system states through locally adapted reconstructions. However, this framework relies on a highly nonlinear and discontinuous basis selection process, which limits its applicability to dynamical systems due to challenges in stability and differentiability.

To address this limitation, we propose the polytopic autoencoder framework which integrates a lightweight nonlinear encoder, a convex-combination decoder, and a smooth clustering network. This architecture is theoretically well-founded, ensuring that reconstructed states reside within a convex polytope defined by learned local bases. Additionally, a polytope error metric is introduced to quantitatively assess the quality of the polytope and provide insight into the reconstruction fidelity.

Another important aspect of model order reduction is the challenge of preserving essential data properties, such as sparsity, positivity, and underlying physical constraints. To alleviate this issue, we design polytopic autoencoders that reconstruct system states using

a decoding matrix derived from a CUR decomposition, which is well-suited for maintaining physical interpretability. By employing actual state samples as decoding components, the proposed approach enhances the preservation of intrinsic data properties and ensures consistency with the original physical system.

ZUSAMMENFASSUNG

Die Simulation komplexer dynamischer Systeme bedeutet in der Regel einen hohen Rechenaufwand, der auf die hohe Dimensionalität der Modellgleichung zurückzuführen ist. Eine mögliche Lösung besteht darin, niederdimensionale Parametrisierungen der Systemzustände zu entwickeln, die eine effizientere Modellierung und Berechnung zu ermöglichen. Unter den klassischen Methoden zu dieser sogenannten *Modellordnungsreduktion* hat sich die *Proper Orthogonal Decomposition* (POD) etabliert. Diese Methode ist optimal im Sinne einer linearen Parametrisierung gegebener Daten aus dem dynamischen System und zeichnet sich in der Praxis durch die Linearität und die Orthogonalität der Projektion auf die Parametrisierung aus. Dennoch ist ihre Approximationsfähigkeit grundsätzlich durch die *Kolmogorov-n-width* begrenzt, was die Effizienz der Dimensionsreduktion in bestimmten Fällen einschränkt.

Als nichtlineare Alternative zur POD wurden in den letzten zehn Jahren zunehmend Autoencoder verwendet und entwickelt, deren Funktionalität sich in einer Vielzahl von Anwendungsbereichen gezeigt hat. Gleichzeitig ist bekannt, dass die Verwendung lokal angepasster Basen, die auf unterschiedliche Bereiche des Zustandsraums zugeschnitten sind, die Rekonstruktionsqualität erheblich verbessern kann.

Aufbauend auf diesen Erkenntnissen schlagen wir eine Deep-Clustering-Architektur vor, die einen nichtlinearen Autoencoder mit einem Clustermodell kombiniert. In der ersten vorgeschlagenen Variante extrahiert ein nichtlinearer Encoder aus gegebenen Daten niederdimensionale Repräsentationen, die als bedeutungstragende Merkmalsabbildungen dienen. Nachdem die diese sogenannten latenten Zustände erfasst wurden, wird ein k-Means-Clustering angewendet, um diese in Cluster zu unterteilen und jedem latenten Zustand ein Label zuzuweisen. Für die Rekonstruktion wird anschließend ein lokaler Modus ausgewählt, der dem jeweiligen Clusterlabel entspricht. Dieses Clustering-Verfahren entspricht der Auswahl und der Verwendung einer lokalen Basis. Auf diese Weise wird eine genaue und effiziente Approximation der Systemzustände durch lokal angepasste Rekonstruktionen ermöglicht. Dieses Verfahren basiert jedoch auf einem stark nichtlinearen und diskontinuierlichen Auswahlprozess für die lokalen Basen. Daher ist seine Anwendbarkeit auf dynamische Systeme eingeschränkt, insbesondere aufgrund fehlender Stabilität und Differenzierbarkeit.

Um diese Einschränkung zu überwinden, schlagen wir das Konzept des polytopischen

Autoencoder vor. Dieses integriert einen nichtlinearen Encoder, einen Decoder auf Basis von Konvexkombinationen und ein differenzierbares Clustering über ein neuronales Netz. Die vorgeschlagene Architektur ist theoretisch fundiert und stellt sicher, dass rekonstruierte Zustände innerhalb eines konvexen Polyeders liegen, das durch gelernte lokale Basen definiert ist. Dazu wird eine Polyeder-Fehlermetrik eingeführt, um die Qualität des Polyeders quantitativ zu bewerten und Aufschluss über die Genauigkeit der Rekonstruktion zu geben.

Ein weiterer wesentlicher Aspekt der Modellordnungsreduktion im Allgemeinen ist die Erhaltung relevanter Eigenschaften der Daten wie Sparsität, Positivität oder physikalische Randbedingungen. In dieser Hinsicht entwickeln wir polytopische Autoencoder, welche die Systemzustände mithilfe einer direkt aus den Daten gezogenen Dekodierungsmatrix rekonstruieren. Diese Matrix wird aus einer CUR-Dekomposition abgeleitet und eignet sich besonders gut zur Wahrung physikalischer Interpretierbarkeit. Durch die direkte Verwendung tatsächlicher Zustandsmessungen als Dekodierungskomponenten trägt der vorgeschlagene Ansatz dazu bei, intrinsische Datenmerkmale zu erhalten und die Übereinstimmung mit dem ursprünglichen physikalischen System sicherzustellen.

CONTENTS

List of Figures xiii

List of Tables xvii

List of Algorithms xix

List of Acronyms xxi

List of Symbols xxiii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Contributions . . . 3
- 1.3 Outline . . . 4

2 Background Preliminaries 5
- 2.1 Matrix Decomposition . . . 5
  - 2.1.1 Singular Value Decomposition . . . 6
  - 2.1.2 CUR Decomposition . . . 7
  - 2.1.3 Matrix Decomposition for Linear Dimensionality Reduction . . . 11
- 2.2 Autoencoders . . . 13
  - 2.2.1 Dense Autoencoders . . . 14
  - 2.2.2 Convolutional Autoencoders . . . 16
  - 2.2.3 Autoencoder Optimization . . . 19
- 2.3 Clustering . . . 22
  - 2.3.1 k-means Clustering . . . 23
  - 2.3.2 Deep Clustering . . . 25
- 2.4 Flow Models and Data . . . 27
  - 2.4.1 Incompressible Navier-Stokes Equations . . . 28
  - 2.4.2 Viscous Burgers' Equations . . . 31
- 2.5 Reduced-order Models and LPV Approaches . . . 32

3 POD and Convolutional Autoencoders **37**
3.1 Introduction . . . 37
3.2 Low-dimensional Parametrization . . . 39
3.2.1 Proper Orthogonal Decomposition (POD) . . . 40
3.2.2 Convolutional Neural Network (CNN) . . . 40
3.2.3 Convolutional AE (CAE) . . . 42
3.2.4 POD-CAE . . . 43
3.2.5 Clustered POD (cPOD) . . . 44
3.2.6 Individual CAE (iCAE) . . . 45
3.3 Numerical Experiments . . . 46
3.3.1 Single-cylinder . . . 47
3.3.2 Double-cylinder . . . 52
3.3.3 Burgers' Flows . . . 55
3.4 Conclusion . . . 57

4 Polytopic Autoencoders **59**
4.1 Introduction . . . 59
4.2 Motivation . . . 60
4.3 Polytopic AE (PAE) . . . 63
4.3.1 Input Converter . . . 64
4.3.2 Encoder . . . 64
4.3.3 Differentiable Clustering Network . . . 66
4.3.4 Decoder . . . 67
4.3.5 Training Details for PAEs . . . 69
4.3.6 Polytope Error and Polytopic LPV Representation . . . 71
4.3.7 Application in Polytopic LPV Approximations . . . 75
4.4 Simulation Results . . . 76
4.4.1 Single-cylinder . . . 77
4.4.2 Double-cylinder . . . 85
4.5 Conclusion . . . 93

5 PCD-based Physics-preserving ROMs **95**
5.1 Introduction . . . 95
5.2 State Selection for PCD . . . 97
5.3 Physics-preserving ROMs . . . 98
5.3.1 ROM for Incompressible Fluid Flows . . . 98
5.3.2 Low-dimensional Parametrization . . . 101
5.4 Simulation Results . . . 105
5.5 Conclusion . . . 111

6 Conclusions **113**
6.1 Summary . . . 113

6.2 Outlook 114

Bibliography **117**

Statement of Scientific Cooperations **129**

LIST OF FIGURES

2.1 Approximation error: for (left) the wine dataset $X \in \mathbb{R}^{13\times 178}$ and (right) the breast cancer dataset $X \in \mathbb{R}^{30\times 569}$: tSVD refers to the rank-r truncated SVD of X based on Algorithm 2.1. CUR-RANDOM and CUR-DEIM denote the CUR decompositions of X implemented by Algorithm 2.2 and Algorithm 2.4 respectively. 11
2.2 A deep autoencoder $g \circ f$: Structure of a deep autoencoder composed of an encoder f and a decoder g, trained to reconstruct $\mathbf{x}$ via the mapping $g(f(\mathbf{x}))$ (i,e, $\mathbf{x} \approx \tilde{\mathbf{x}}$) 14
2.3 Dense connectivity vs. sparse connectivity with a kernel size of 3 17
2.4 No sharing (9 different parameters) vs. parameter sharing (a single kernel with parameters w_1, w_2, and w_3) 17
2.5 Indirectly dense connectivity formed by two convolutional layers 18
2.6 A model architecture for DAE-based deep clustering 25
2.7 Two cases of incompressible flows 30
a Single-cylinder case 30
b Double-cylinder case 30
2.8 Comparison of Burgers' flows under two distinct initial conditions 32
a Flow snapshots with the initial condition (2.21) 32
b Flow snapshots with the initial condition (2.22) 32

3.1 A CAE architecture consisting of a nonlinear convolutional encoder and a reparameterized affine linear decoder by Remark 3.2 42
3.2 Averaged relative errors as measured by the evaluation metrics (3.5), (3.6), and (3.7) for the single-cylinder case 48
3.3 Reconstruction error over time for the single-cylinder case at $r = 2, 3, 5, 8$. 49
3.4 Influence of latent space dimension and number of clusters on iCAE reconstruction error for the single-cylinder case 50
3.5 Clustering reaction in the time range $[0, 16]$ for the single-cylinder case . . 51
3.6 Averaged relative errors as measured by the evaluation metrics (3.5), (3.6), and (3.7) for the double-cylinder case 52
3.7 Reconstruction error over time for the double-cylinder case at $r = 2, 3, 5, 8$ 53

3.8 Influence of latent space dimension and number of clusters on iCAE reconstruction error for the double-cylinder case 53
3.9 Clustering reaction in the time range $[240, 760]$ for the double-cylinder case 54
3.10 Averaged relative errors as measured by the evaluation metric (3.5) for the Burgers' flows 55
3.11 Reconstruction error over time for the Burgers' flows at $r = 2, 3, 5, 8$: the solid lines represent reconstruction errors from the simulation using the initial condition (2.21), while the dashed lines indicate test reconstruction errors from the simulation with a different initial condition (2.22) 56
3.12 Clustering reaction in the time range $[0, 0.5]$ for the Burgers' flows 56

4.1 A proposed architecture consisting of a nonlinear convolutional encoder, a smooth clustering model, and a reparameterized affine linear decoder . . . 62
4.2 Inverted residual block: an efficient approach for constructing deep convolutional layers which require fewer parameters than the standard convolutions 65
4.3 When the latent states are classified into three clusters in a low-dimensional space, the clustering labels associated with the latent states within each circle, centered at c_i with radius m_i, $i = 1, 2, 3$, are selected as target labels. Unselected labels are excluded from the training of PAEs. 69
4.4 A schematic figure depicting the positions of a state $\mathbf{x}$, its reconstruction $\tilde{\mathbf{x}}$ in a polytope $\tilde{X}$, and its best approximation $\mathbf{x}^*$ 72
4.5 Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the single-cylinder case. The shaded regions represent statistical uncertainty across multiple training runs. However, this uncertainty is negligible for most methods, rendering it nearly invisible in the plots. . . 78
4.6 Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 2.0$: training session for the single-cylinder case 79
4.7 Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 14.0$: testing session for the single-cylinder case 81
4.8 Activation rates and trajectories of latent state variables for $r = 2$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case. 82
4.9 Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the single-cylinder case. The different colors represent distinct clusters. 83
4.10 Activation rates and trajectories of latent state variables for $r = 3$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case. 83

4.11 Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the single-cylinder case. The different colors represent distinct clusters. 84
4.12 Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the double-cylinder case. The shaded regions represent statistical uncertainty across multiple training runs. However, this uncertainty is negligible for most methods, rendering it nearly invisible in the plots. 86
4.13 Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 556.0$: training session for the double-cylinder case 87
4.14 Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 736.5$: testing session for the double-cylinder case 88
4.15 Activation rates and trajectories of latent state variables for $r = 2$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the double-cylinder case. 90
4.16 Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the double-cylinder case. The different colors represent distinct clusters. 91
4.17 Activation rates and trajectories of latent state variables for $r = 3$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the double-cylinder case. 92
4.18 Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the double-cylinder case. The different colors represent distinct clusters. 92

5.1 Approximation error: truncated SVD and CUR decompositions 106
5.2 Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the single-cylinder case. The shaded regions show uncertainty from multiple runs. 107
5.3 The first 5 modes out of 64 for each models when $r = 64$: the POD basis is obtained by applying truncated SVD to the centered data $\mathbf{x} - \bar{\mathbf{x}}$, the PCD-AE modes are identical to the PCD modes which are derived via CUR decomposition of the centered data $\mathbf{x} - \bar{\mathbf{x}}$, and PCD-PAE modes are actual states selected from the data $\mathbf{x}$. 108
a POD 108
b PCD/PCD-AE 108
c PCD-PAE 108
5.4 Comparison of relative trajectory reconstruction error across ROMs when $r = 64$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case. 109

5.5 Comparison between the reference produced by FOM and the snapshots generated by POD, PCD/PCD-AE, and PAE at t = 0,2,6,9,16, $r = 64$ . . . 110
a FOM . . . 110
b POD . . . 110
c PCD/PCD-AE . . . 110
d PCD-PAE . . . 110
5.6 Incompressibility: averaged L_2 error of the divergence of each mode $\mathbf{d}_i$ at $r = 64$ . . . 110
5.7 Averaged L_2 error of the divergence vs. singular value for each POD mode $\mathbf{d}_i$ . . . 111

6.1 (left) The three-dimensional reference is obtained from the FEM simulation (2.18) at $\mathsf{Re} = 500$ with the state dimension $n = 107691$, and (right) an incompressible flow is reconstructed by a PCD-based ROM with reduced dimension $r = 128$. . . . 114

LIST OF TABLES

2.1 Configuration of the FEM simulation (2.18) for two data sets; wake flows behind either a single circular cylinder or two circular cylinders respectively as shown in Figure 2.7. 29
2.2 Configuration of the FEM simulation (2.20) for the traffic flow data set as shown in Figure 2.8 31

3.1 Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, dcv: a deconvolutional layer 46

4.1 Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, c: a clustering layers ("#" refers to the number of.) 77
4.2 Computational times for $r = 3$: the offline time for POD represents the time required to obtain a POD basis on the CPU. For CAE and PAE, the offline time corresponds to the training time on the GPU. The inference time indicates the runtime for reconstructing a state on the CPU in the single-cylinder case. 80
4.3 Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, c: a clustering layers ("#" refers to the number of.) 85
4.4 Computational times for $r = 3$: the offline time for POD represents the time required to obtain a POD basis on the CPU. For CAE and PAE, the offline time corresponds to the training time on the GPU. The inference time indicates the runtime for reconstructing a state on the CPU in the double-cylinder case. 89

5.1 ROMs for $r = 64$: the interpolation error is defined as the mean reconstruction error over the time domain $[0, 10]$. The extrapolation error is the mean reconstruction error over the time domain $[10, 16]$. The divergence error is the mean of $\|\mathsf{J}\mathbf{x}\|_2$ for all $\mathbf{x}$. 109

LIST OF ALGORITHMS

2.1 Randomized SVD [52] . . . 7
2.2 CUR decomposition (random selection) [35] . . . 8
2.3 DEIM [21] . . . 10
2.4 CUR decomposition (DEIM-based selection) [21] . . . 10
2.5 Training an autoencoder $g \circ f$ . . . 20
2.6 Training k-means clustering . . . 23
2.7 Training a DAE-based deep clustering model . . . 26

3.1 cPOD . . . 44
3.2 Training an iCAE model . . . 45

4.1 Training a PAE . . . 70

5.1 CUR decomposition (genetic algorithm) . . . 97
5.2 CUR decomposition (DEIM-based selection with data pruning) . . . 98
5.3 Training a PCD-AE . . . 103
5.4 Training a PCD-PAE . . . 105

LIST OF ACRONYMS

AE	Autoencoder
CAE	Convolutional Autoencoder
cPOD	Clustered POD
CNN	Convolutional Neural Network
DAE	Deep Autoencoder
DEIM	Discrete Empirical Interpolation Method
DNN	Deep Neural Network
FEM	Finite Element Method
FOM	Full-Order Model
iCAE	Individual CAE
KL	Kullback–Leibler
L-BFGS	Limited-memory Broyden–Fletcher–Goldfarb–Shanno
LPV	Linear Parameter-Varying
MLP	Multi-Layer Perceptron
MOR	Model Order Reduction
MSE	Mean Square Error
PAE	Polytopic Autoencoder
PDE	Partial Differential Equation
PCD	Proper CUR Decomposition
POD	Proper Orthogonal Decomposition
ROM	Reduced-Order Model
SGD	Stochastic Gradient Descent
SVD	Singular Value Decomposition

LIST OF SYMBOLS

$\mathbb{N}$	the set of natural numbers (nonnegative integers)
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^{m\times n}$	the set of $m \times n$ real matrices
I_n	the $n \times n$ identity matrix
$[a, b]$	the closed interval from a to b, including endpoints a and b
(a, b)	the open interval from a to b, excluding endpoints a and b
$(a, b]$	the half-open interval from a to b, excluding a and including b
$A^\top$	the transpose of matrix A
a_{ij}, $A_{i,j}$, $A[i, j]$	the (i, j)-th entry of matrix A
$A[i, :]$	the i-th row vector of matrix A
$A[:, j]$	the j-th column vector of matrix A
$A[\mathbf{p}, :]$	the submatrix of A consisting of the row vectors indexed by the index set $\mathbf{p}$
$A[:, \mathbf{q}]$	the submatrix of A consisting of the column vectors indexed by the index set $\mathbf{q}$
A^{-1}	the inverse of nonsingular matrix A
A^+	the Moore–Penrose pseudoinverse of matrix A
$\mathrm{Tr}(A) := \sum_{i=1}^n a_{ii}$	the trace of matrix A
$\langle A, B\rangle_F := \mathrm{Tr}(A^\top B)$	the Frobenius inner product of real-valued matrices A and B
$\|A\|_F := \sqrt{\mathrm{Tr}(A^\top A)}$	the Frobenius norm of real-valued matrix A
$\|\mathbf{x}\|_p := (\sum_{i=1}^n \|x_i\|^p)^{1/p}$	the p-norm of vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]^\top$, where $1 \le p < \infty$
$\|\mathbf{x}\|_{\mathsf{M}} := \sqrt{\mathbf{x}^\top \mathsf{M}\mathbf{x}}$	the M-norm of vector $\mathbf{x}$ with a symmetric positive definite matrix M
$\|\mathbf{x}\|$	the vector of element-wise absolute values of vector $\mathbf{x}$

(X, Y)	a pair of matrices X and Y
$\mathsf{min}(m, n)$	the smaller value of real numbers m and n
$\mathsf{max}(\mathbf{x})$	the largest element of vector $\mathbf{x}$
$\mathsf{argmin}(\mathbf{x})$	the index of the minimum element in vector $\mathbf{x}$
$\mathsf{argmax}(\mathbf{x})$	the index of the maximum element in vector $\mathbf{x}$
$\mathrm{argmin}_{j=1,2,\cdots,k} d_j$	the index j at which the value d_j attains its minimum
$g \circ f$	g composed with f
$\dot{\mathbf{x}} := \frac{d\mathbf{x}}{dt}$	the time derivative of function $\mathbf{x}$
$\approx$	approximately equal to
$m \gg n$	m is much larger than n
$m \ll n$	m is much smaller than n
$\mathbf{x} \cdot \mathbf{y}$	the inner product of vectors $\mathbf{x}$ and $\mathbf{y}$
$\lfloor a \rfloor$	the integer part (floor) of value a
$\otimes$	the Kronecker product
$\mathcal{N}(\mu, \sigma^2)$	the normal distribution with mean μ and variance σ^2
$\#B$	the batch size (i.e., the number of data samples in batch B)
$*$	the convolution operator
Conv	a convolutional layer
DeConv	a deconvolutional layer
GAP	a global average pooling layer
RB	an inverted residual block
Lin	a fully connected layer
$\nabla_\theta f$	the gradient of a scalar-valued function f with respect to trainable model weights θ

CHAPTER 1
INTRODUCTION

Contents

1.1 Motivation 1
1.2 Contributions 3
1.3 Outline 4

1.1 Motivation

Large-scale dynamical systems typically require considerable computational resources and data storage capacity due to the high dimensionality of their state representations. This poses challenges for both data processing and simulation efficiency. To address this challenge, low-dimensional parametrizations of high-dimensional states have been developed as a model order reduction approach. Parametrization refers to the process of expressing the structure of a system using a finite set of parameters. These parameters typically capture variations in input features. While parametrization targets the compact representation of input or configuration spaces, model order reduction focuses on simplifying the system.

Among the available techniques, linear model order reduction techniques offer a way to decrease model complexity by approximating the original system with a reduced representation, albeit with a trade-off in accuracy. These reduced-order models can alleviate the computational load, making large-scale simulations more feasible. However, when even lower-dimensional representations are required, such as for control system design, linear methods like *proper orthogonal decomposition* (POD) reach their inherent limitations imposed by the *Kolmogorov n-width* [85].

In such cases, nonlinear techniques become necessary, as they are better suited to capture the complexity of the system states in a more compact form and provide greater flexibility in handling nonlinearities. In particular, neural-network-based methods have been widely adopted, including multilayer perceptrons (e.g., [71]), autoencoders (e.g.,

[91, 87]), convolutional autoencoders (e.g.,[37, 43, 74]), variational autoencoders (e.g., [117]), and recurrent neural networks (e.g., [39, 82]).

Despite the limitations of linear model order reduction methods in capturing nonlinear dynamics, they continue to be widely adopted across various applications; e.g., [104, 84]. This widespread use can be attributed to their inherent stability, which ensures reliable system behavior, as well as their computational efficiency, which allows for faster simulations. Also, their mathematical simplicity makes these techniques easy to implement in legacy code.

Given the distinct advantages offered by both linear and nonlinear methods, this work explores hybrid approaches that combine the strengths of each. Specifically, this study investigates nonlinear techniques for dimensionality reduction while maintaining the ability to reconstruct the states through a linear method. To this end, a novel framework is proposed that balances the flexibility offered by nonlinear reductions with the efficiency and simplicity of linear projection for state reconstruction. Additionally, it is known that using local bases for state reconstruction improves reconstruction performance; e.g., [29, 7, 8, 38, 67]. Accordingly, the investigation includes clustering reduced-dimensional states to generate local bases that more effectively capture the dynamics within specific regions of the state space, which are then used for state reconstruction.

In the field of deep learning, deep clustering refers to a framework that integrates neural networks with clustering techniques for unsupervised learning tasks; e.g., [40, 114, 77]. In particular, autoencoder-based deep clustering architectures jointly consider clustering performance and reconstruction loss during training. These models are typically trained to minimize reconstruction error in order to obtain meaningful latent states, while simultaneously predicting cluster labels based on the structure of the low-dimensional latent space. However, these models are not inherently designed to improve reconstruction quality using the clustering results. Consequently, clustering information is found to be beneficial for enhancing reconstruction performance by enabling the assignment of individual decoders according to the cluster associated with each latent state. Finally, a deep clustering network architecture, composed of a nonlinear encoder, an affine linear decoder, and a clustering network, is designed. This architecture enables the model to extract latent states via a global encoder and reconstruct them through local modes selected according to the clustering result associated with each state.

From another perspective, standard linear model order reduction methods project states and latent variables onto unbounded linear subspaces, despite the fact that these quantities are typically observed to lie within bounded regions. This mismatch implies that enforcing state reconstruction within a polytope could be beneficial. To the best of our knowledge, the concept of polytopic autoencoders has not been explored in the existing literature. We therefore introduce a novel formulation, termed a polytopic autoencoder, in which the reconstructed states are constrained to lie within a convex polytope defined by a set of support vectors. Building upon this formulation, we further demonstrate that deep clustering architectures can be naturally interpreted as polytopic autoencoders by combining this convex reconstruction mechanism with a smooth, differentiable clustering

model.

Reduced-order models often face significant challenges in preserving fundamental physical laws and critical data properties such as sparsity, positivity, and incompressibility. These limitations can result in reduced physical fidelity and degraded performance in downstream tasks. A possible approach to mitigate this issue is the use of CUR decompositions (e.g., [81, 49]), which construct low-rank approximations by selecting actual columns and rows from a given matrix. By doing so, CUR decomposition retains data properties, making it particularly well-suited for applications that demand interpretability and the retention of intrinsic data properties. As the final topic of this thesis, physics-preserving reduced-order models are developed by combining the strengths of autoencoders and CUR decompositions.

1.2 Contributions

The main contributions of this thesis are as follows:

1. *A deep clustering network architecture for low-dimensional parametrization of flow equations*: A deep clustering architecture employs nonlinear encoding to extract meaningful latent states, classifies these states in the latent space, and utilizes local bases to enhance the quality of state reconstruction. In particular, the clustering module is a differentiable and allows states to lie in transition areas among clusters. Thus, in contrast to standard clustering methods, it can be seamlessly integrated into differentiable dynamical systems.

2. *Polytopic representations for state reconstruction*: Polytopic autoencoders (PAEs) ensure that all states are reconstructed within a polytope defined by a minimal number of vertices. This formulation enables direct application to polytopic linear parameter-varying (LPV) systems without requiring additional procedures for a bounding box of states. The optimal reconstruction error is theoretically estimated by solving a convex quadratic combination problem so that PAEs reduce the uncertainty of training and evaluating models. Furthermore, it is shown that a deep clustering model composed of a PAE and a smooth clustering network can also be regarded as a polytopic autoencoder. This formulation inherits the desirable properties of PAEs while additionally benefiting from local basis representations.

3. *High accuracy of our approach in extremely low dimensions compared to POD*: The proposed models are evaluated using three flow datasets and compare them against benchmark methods such as POD. The results demonstrate that our models achieve significantly lower reconstruction errors, especially in scenarios involving extremely low-dimensional representations.

4. *A physics-preserving reduced-order modeling via proper CUR decomposition*: A column selection method is developed for CUR decompositions and used selected

columns for proper CUR Decomposition (PCD). Reduced-order models constructed using PCD can achieve simulation accuracy comparable to the POD-based model, while better preserving incompressibility in our simulation.

1.3 Outline

The rest of the thesis is organized as follows:

Chapter 2 provides background preliminaries essential for understanding the contents of this thesis. It covers matrix decompositions, autoencoders, clustering algorithms, autoencoder-based deep clustering, flow data generated by simulations of either the incompressible Navier–Stokes equations or the inviscid Burgers' equations, and a sketch of reduced-order modeling.

Chapter 3 presents a deep clustering architecture called individual convolutional autoencoders (iCAEs). It discusses their low-dimensional parameterization approach based on nonlinear encoding and the use of local bases. Moreover, it evaluates the performance of iCAEs in comparison with other benchmark models such as POD and clustered POD using diverse evaluation metrics.

Chapter 4 introduces the concept of PAEs. This chapter outlines the motivation behind PAEs, the relationship between iCAEs and PAEs, details their architecture and training process, and provides mathematical theorems related to the polytopic representation of system states. Additionally, it discusses their application to polytopic LPV approximations and presents diverse simulation results.

Chapter 5 proposes PCD and PCD-based physics-preserving reduced-order models. It describes the methodology for selecting appropriate actual states, followed by the construction of ROMs that incorporate physical constraints. Simulation results validate the proposed model order reduction approach.

Chapter 6 concludes this thesis with a summary of key findings and an outlook on future directions.

CHAPTER 2
BACKGROUND PRELIMINARIES

Contents

2.1 Matrix Decomposition . . . 5
2.1.1 Singular Value Decomposition . . . 6
2.1.2 CUR Decomposition . . . 7
2.1.3 Matrix Decomposition for Linear Dimensionality Reduction . . . 11
2.2 Autoencoders . . . 13
2.2.1 Dense Autoencoders . . . 14
2.2.2 Convolutional Autoencoders . . . 16
2.2.3 Autoencoder Optimization . . . 19
2.3 Clustering . . . 22
2.3.1 k-means Clustering . . . 23
2.3.2 Deep Clustering . . . 25
2.4 Flow Models and Data . . . 27
2.4.1 Incompressible Navier-Stokes Equations . . . 28
2.4.2 Viscous Burgers' Equations . . . 31
2.5 Reduced-order Models and LPV Approaches . . . 32

This section lays the groundwork for the following chapters by providing an overview of linear dimensionality reduction, autoencoders, clustering methods, and their integration approaches.

2.1 Matrix Decomposition

Matrix decomposition (or *factorization*) is the process of expressing a given matrix as a product of multiple matrices, such as singular value decomposition, CUR decomposition, LU decomposition, and Cholesky decomposition. It is widely used for various tasks including data compression, model order reduction, and noise reduction; e.g., [90, 61, 97]. This

section introduces singular value decomposition and CUR decomposition and explains how they can be applied to linear dimensionality reduction tasks.

2.1.1 Singular Value Decomposition

Suppose a matrix $X \in \mathbb{R}^{m \times n}$ is given. Then there exist two orthogonal matrices $V \in \mathbb{R}^{m \times m}$ and $W \in \mathbb{R}^{n \times n}$ such that

$$X = V \Sigma W^\top$$

where $\Sigma \in \mathbb{R}^{m \times n}$ is a matrix with diagonal entries $\sigma_1, \sigma_2, \ldots, \sigma_q$ called the singular values of X, satisfying $\sigma_1 \geq \sigma_2 \geq \ldots \sigma_q \geq 0$ and $q = \mathsf{min}(m, n)$. This decomposition is called the *singular value decomposition (SVD)* of X.

The SVD provides several computational advantages including

1. *Orthogonality of singular vectors in the matrices V and W*, and
2. *Optimality for low-rank approximations.*

First, due to the orthogonality of the left and right singular vectors contained in the matrices V and W respectively, these matrices facilitate the simplification of certain computations. For instance,

$$V^\top V = V V^\top = \mathsf{I}_m,\ W^\top W = W W^\top = \mathsf{I}_n,\ \text{and}\ \|V\mathbf{z}\|_2 = \|\mathbf{z}\|_2$$

for any vector $\mathbf{z}$. Furthermore, the orthogonality simplifies the computation for the inverse (or pseudoinverse) of X. Let

$$\Sigma = \left[\begin{array}{ccc|ccc} \sigma_1 & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & \sigma_r & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right] \in \mathbb{R}^{m \times n}$$

be the singular value matrix. Then, the Moore–Penrose inverse Σ^+ of Σ is given by

$$\Sigma^+ = \left[\begin{array}{ccc|ccc} \frac{1}{\sigma_1} & & & 0 & \cdots & 0 \\ & \ddots & & \vdots & & \vdots \\ & & \frac{1}{\sigma_r} & 0 & \cdots & 0 \\ \hline 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}\right] \in \mathbb{R}^{n \times m}.$$

Algorithm 2.1: Randomized SVD [52]

Input: Matrix $X \in \mathbb{R}^{m \times n}$, reduced dimension r, oversampling parameter η
Output: $V_r \in \mathbb{R}^{m \times r}$, $\Sigma_r \in \mathbb{R}^{r \times r}$, $W_r \in \mathbb{R}^{n \times r}$ $(X \approx V_r \Sigma_r W_r^\top)$

1 Generate a matrix $T \in \mathbb{R}^{n \times (r+\eta)}$ with the entries sampled from $\mathcal{N}(0, 1)$
2 Compute $Y = XT$ `// A matrix` Y `smaller than` X
3 Compute $Y = \mathsf{QR}$ `// QR decomposition of` Y
4 Compute $\mathsf{Q}^\top X = \hat{V} \Sigma W^\top$ `// SVD of` $\mathsf{Q}^\top X$
5 Compute $X \approx V_r \Sigma_r W_r^\top$ where $V_r = \mathsf{Q}\hat{V}_r$ `// truncated SVD of` X `based on top` r `singular values`
6 **return** V_r, Σ_r, W_r

Thus, the inverse (or pseudoinverse) of X can be expressed as

$$X^+ = W \Sigma^+ V^\top,$$

which allows obtaining the inverse (or pseudoinverse) of X without performing a direct matrix inversion.

Second, SVD is recognized for its optimality in constructing low-rank approximations. Specifically, it minimizes the approximation error in the Frobenius norm, as shown in Lemma 2.1. Due to this property, SVD is widely used for dimensionality reduction, which represents high-dimensional data in a lower-dimensional subspace while preserving the most significant features; see Section 2.1.3.

The SVD requires high computational costs when X is a large-scale matrix. To alleviate this computational burden, it is commonly assumed that the eigenvectors corresponding to small singular values contribute negligibly to the overall structure and approximation of X. Based on this assumption, one can employ *truncated SVD* which discards the less significant components while retaining the dominant features of the original matrix. This approach allows for substantial computational savings while maintaining a desired level of approximation accuracy. In other words, X is approximated using the truncated SVD defined as

$$X \approx V_r \Sigma_r W_r^\top,$$

where V_r is the $m \times r$ matrix containing the first r columns of V, Σ_r is the $r \times r$ diagonal matrix containing the r largest singular values, W_r is the $n \times r$ matrix consisting of the first r columns of W, and $r \ll q$. In practice, the truncated SVD of X can be implemented using a variety of efficient algorithms such as Algorithm 2.1.

2.1.2 CUR Decomposition

SVD has been widely adopted in numerous scientific and engineering applications due to its optimality and the orthogonality of its singular vector matrices. Specifically, it

Algorithm 2.2: CUR decomposition (random selection) [35]

Input: Dataset $X \in \mathbb{R}^{m \times n}$, $r \leq \mathsf{min}(m, n)$
Output: $C \in \mathbb{R}^{m \times r}$, $U \in \mathbb{R}^{r \times r}$, $R \in \mathbb{R}^{r \times n}$ ($X \approx CUR$)

Compute $s = \sum_i \sum_j X_{i,j}^2$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$
for $i = 1$ *to* m **do**
    $p_i = \sum_j X_{i,j}^2 / s$ `// m probabilities for the row selection`
for $j = 1$ *to* n **do**
    $q_j = \sum_i X_{i,j}^2 / s$ `// n probabilities for the column selection`
Set $\mathbf{p} = [p_1, p_2, \ldots, p_m]$, $\mathbf{q} = [q_1, q_2, \ldots, q_n]$
Select r columns from the multinomial distribution with $\mathbf{q}$ to construct C
Select r rows from the multinomial distribution with $\mathbf{p}$ to construct R
Compute $U = C^+ X R^+$ `// +: Moore-Penrose pseudoinverse`
return C, U, R

provides the best low-rank approximation of a matrix in terms of the Frobenius norm, and the orthogonality of the left and right singular vectors ensures numerical stability and interpretability in many contexts. These properties make SVD an appealing choice for tasks such as dimensionality reduction, signal processing, and latent semantic analysis.

However, despite its theoretical strengths, SVD also has notable limitations, particularly when applied to data analysis and interpretation. A critical drawback is that the singular vectors contained in the matrices V and W do not correspond to actual columns or rows of the original data matrix X. Instead, they are linear combinations of all columns or rows, which may result in singular vectors that lack direct interpretability. Consequently, these vectors might fail to satisfy key structural properties or constraints inherent in the data.

For example, a lot of real-world datasets exhibit sparsity, where only a small number of entries in each data vector are non-zero such as user data for recommendation systems. Other datasets may impose specific constraints such as nonnegativity, as in image intensities or probability distributions, or even integer values, as in categorical or count data. Since SVD generates dense vectors through linear combinations of actual data, it does not inherently preserve such properties. Consequently, the singular vectors produced by SVD may not align with the underlying physical or statistical characteristics of the original data.

Therefore, while SVD remains a popular matrix approximation, its limitations in preserving the intrinsic properties of the data motivate the exploration of alternative methods such as *CUR decomposition* that prioritizes interpretability and property preservation.

The CUR decomposition uses actual columns and rows obtained from a given matrix X for the decomposition

$$X \approx CUR,$$

where C is an $m \times r$ matrix containing r columns of X, R is an $r \times n$ matrix consisting of r rows of X, and U is an $r \times r$ matrix defined as $U = C^+XR^+$. The matrices C^+ and R^+ denote to the Moore-Penrose pseudoinverse of C and R respectively.

This approach provides several advantages including

1. *Sparsity preservation*,
2. *Interpretability*, and
3. *Preservation of data properties*.

When X is a large sparse matrix, both C and R are also sparse, leading to low computational costs compared to using the dense singular matrices V and W in SVD.

Furthermore, since the actual columns and rows of X are used for its low-rank approximation, the selected columns and rows directly correspond to data features (i.e., any data features are represented through the linear combinations of the selected columns or rows). As a result, the CUR decomposition is not only more interpretable than SVD from a data analysis perspective, but also the vectors in C and R satisfy the properties of X in contrast to SVD.

Lemma 2.1 (Optimal low-rank approximation [103, I.9, pp.71-75]):
Let $X \in \mathbb{R}^{m\times n}$ and let X_r be the truncated SVD of X with r modes. Then

$$\|X - X_r\|_F^2 = \min_{\substack{Z\in\mathbb{R}^{m\times n} \\ \text{rank}(Z)\leq r}} \|X - Z\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius norm and $r \leq \mathsf{min}(m, n)$. ◊

This optimality result implies that the CUR decomposition of X generally results in a higher approximation error than the truncated SVD as the lower error bound of the CUR decomposition of X,

$$\|X - CUR\|_F^2 \geq \min_Z \|X - Z\|_F^2 = \|X - X_r\|_F^2. \tag{2.1}$$

Moreover, since the approximation error in the CUR decomposition is sensitive to selected column and row distributions, the sampling method used for selecting appropriate columns and rows is a critical factor in the decomposition. In other words, the upper error bound is decided depending on sampling methods; see [81, 99, 21].

For instance, the CUR decomposition employs a random sampling method where the selection is based on a multinomial distribution with probabilities determined by the magnitudes of each column and row respectively, without replacement, as shown in Algorithm 2.2.

As an advanced sampling method, the *discrete empirical interpolation method* (DEIM) [24, 99] can be used for the CUR decomposition. In Algorithm 2.3, an approximation of the current singular vector $\mathbf{d}$ in the subspace spanned by the previously selected vectors

Algorithm 2.3: DEIM [21]

Input: Unitary matrix $D \in \mathbb{R}^{m\times r}$ $(r \leq m)$
Output: Vector of selected indices $\mathbf{p} = [p_1, p_2, \ldots, p_r]$

$\mathbf{d} = D[:, 1]$ // 1st leading singular vector
$\mathbf{p} = [\mathsf{argmax}(|\mathbf{d}|)]$ // 1st selected index
for $i = 2$ *to* r **do**
  $\mathbf{d} = D[:, i]$ // i-th leading singular vector
  $\tilde{\mathbf{d}} = D[:, 1:i-1]D[\mathbf{p}, 1:i-1]^{-1}\mathbf{d}[\mathbf{p}]$ // interpolatory projector for the approximation of $\mathbf{d}$
  $\mathbf{e} = \mathbf{d} - \tilde{\mathbf{d}}$ // error between $\mathbf{d}$ and $\tilde{\mathbf{d}}$
  $\mathbf{p} = [\mathbf{p}, \mathsf{argmax}(|\mathbf{e}|)]$ // storage of the i-th selected index
return $\mathbf{p}$

Algorithm 2.4: CUR decomposition (DEIM-based selection) [21]

Input: Dataset $X \in \mathbb{R}^{m\times n}$, $r \leq \mathsf{min}(m, n)$
Output: $C \in \mathbb{R}^{m\times r}$, $U \in \mathbb{R}^{r\times r}$, $R \in \mathbb{R}^{r\times n}$ $(X \approx CUR)$

Compute $X \approx V\Sigma W^\top$ // rank-r truncated SVD of X
Compute $\mathbf{a} = \mathsf{DEIM}(W)$ // select r column indices for C
Compute $\mathbf{b} = \mathsf{DEIM}(V)$ // select r row indices for R
Define $C = X[:, \mathbf{a}]$ and $R = X[\mathbf{b}, :]$
Compute $U = C^+XR^+$ // $+$: Moore-Penrose pseudoinverse
return C, U, R

is computed as $\tilde{\mathbf{d}}$. Next, p_i, regarded as an interpolation point not explained by the previously selected points $p_1, \ldots, p_{i-1}$, is identified via $\mathsf{max}(|\mathbf{e}|)$. This avoids the redundant selection of interpolation points and can achieve high accuracy compared to Algorithm 2.2. As shown in Algorithm 2.4, the DEIM-based selection method is utilized to obtain two vectors $\mathbf{a}$ and $\mathbf{b}$, corresponding to the chosen column and row indices respectively.

To validate the proposed methods, two empirical datasets, Wine [1] and Breast Cancer [113], are used. The wine dataset comprises 178 chemical analysis results of wines with 13 chemical constituents. The breast cancer dataset consists of 569 samples with 30 real-valued features extracted from digitized images of fine needle aspirates of breast masses. Since they exhibit distinct characteristics in terms of dimensionality, feature distribution, and class structure, they are suitable for a robust comparison of low-rank approximation methods.

Figure 2.1 presents the approximation errors of three dimensionality reduction methods applied to the datasets.

Across both datasets, tSVD consistently yields the lowest approximation error, as expected from inequality (2.1). Among the CUR-based methods, CUR-DEIM outper-

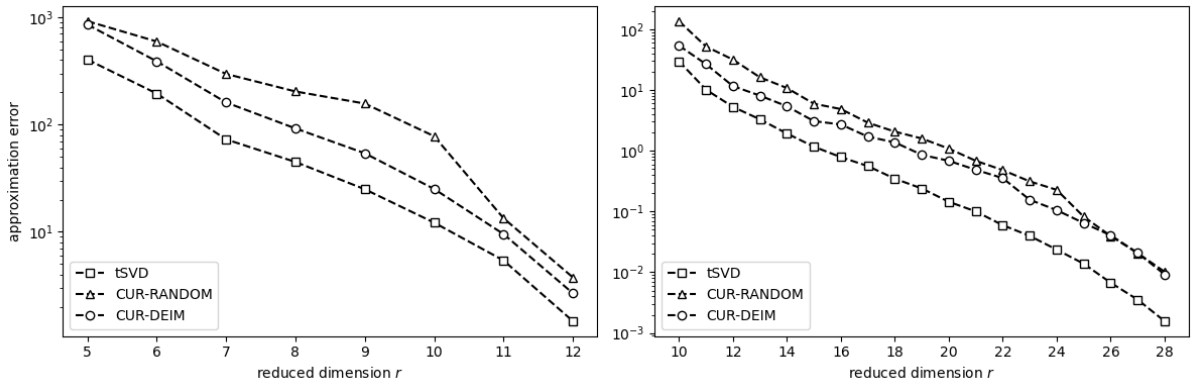


Figure 2.1: Approximation error: for (left) the wine dataset $X \in \mathbb{R}^{13\times178}$ and (right) the breast cancer dataset $X \in \mathbb{R}^{30\times569}$: tSVD refers to the rank-r truncated SVD of X based on Algorithm 2.1. CUR-RANDOM and CUR-DEIM denote the CUR decompositions of X implemented by Algorithm 2.2 and Algorithm 2.4 respectively.

forms CUR-RANDOM underscoring the effectiveness of DEIM in identifying informative columns and rows for CUR approximation. As the reduced dimension r increases, all methods exhibit a clear downward trend in approximation error, reflecting enhanced reconstruction performance. These results demonstrate that, while tSVD offers the highest accuracy, CUR-DEIM serves as a competitive and more interpretable alternative, achieving improved approximation quality without compromising the structural interpretability inherent in CUR decompositions. CUR-DEIM is a deterministic algorithm, whereas CUR-RANDOM is stochastic. For CUR-RANDOM, the minimum error from 15 iterations is selected as the representative error.

In Section 5.3, other sampling methods for CUR decompositions of flow data are discussed.

2.1.3 Matrix Decomposition for Linear Dimensionality Reduction

Linear dimensionality reduction refers to the technique of transforming high-dimensional data onto a low-dimensional space and projecting the reduced-dimensional data (*latent states* or *latent variables*) back to the original space using linear projection methods that involve the reconstruction error between the original data and its approximations. This process typically involves two linear operators, *encoder* and *decoder*: one for projecting the data into the reduced space, and the other for reconstructing it back to the original space, such that the reconstruction error is minimized. Mathematically, this is expressed

as

$$\boldsymbol{\rho} = F\mathbf{x},$$
$$\tilde{\mathbf{x}} = G\boldsymbol{\rho},$$

where F and G are $r \times m$ and $m \times r$ matrices respectively and $\mathbf{x}$, $\tilde{\mathbf{x}}$ and $\boldsymbol{\rho}$ are an $m \times 1$ vector, the reconstruction of $\mathbf{x}$ (i.e., $\tilde{\mathbf{x}} \approx \mathbf{x}$), and an $r \times 1$ latent variable respectively ($r \ll m$). To obtain appropriate F and G, for example, either the truncated SVD or the CUR decomposition of a given $m \times n$ dataset

$$X = \begin{bmatrix} | & | & & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{n-1} & \mathbf{x}_n \\ | & | & & | & | \end{bmatrix}$$

is employed such that $X \approx V_r \Sigma_r W_r^\top$ or $X \approx CUR$ as discussed in Section 2.1.1 and Section 2.1.2. Based on the truncated SVD, $F = V_r^\top$ and $G = V_r$ such that the linear dimensionality reduction method is defined as

$$\boldsymbol{\rho} = V_r^\top \mathbf{x},$$
$$\tilde{\mathbf{x}} = V_r \boldsymbol{\rho}.$$

This approach is known as *proper orthogonal decomposition* (POD), and the columns of V_r are referred to as *POD modes*. POD has been widely used in a variety of fields, including fluid dynamics and machine learning, due to its linearity and its optimality in the sense of the Frobenius norm.

Alternatively, using the CUR decomposition, C^+ and C are used to set F and G respectively then the linear dimensionality reduction method called *proper CUR decomposition* (PCD) is defined as

$$\boldsymbol{\rho} = C^+ \mathbf{x},$$
$$\tilde{\mathbf{x}} = C\boldsymbol{\rho}.$$

The matrix C^+ refers to the Moore-Penrose pseudoinverse of C. The PCD approach preserves the physical interpretability of the basis vectors, since the columns of C correspond to actual data rather than abstract orthogonal directions as in SVD.

Lemma 2.2 (Projection error bound for CUR decomposition):

$$\|X - CC^+X\|_F^2 \leq \|X - CUR\|_F^2,$$

where $X \approx CUR$ is a CUR decomposition of X and C^+ is Moore–Penrose inverse of C.

Proof. Let $Y = CUR$. Then

$$\begin{aligned} \|X - CUR\|_F^2 &= \|X - Y\|_F^2 \\ &= \|(X - CC^+X) + (CC^+X - Y)\|_F^2 \\ &= \|X - CC^+X\|_F^2 + \|CC^+X - Y\|_F^2 + 2\langle X - CC^+X, CC^+X - Y\rangle_F \end{aligned}$$

where $\langle \cdot, \cdot \rangle_F$ is Frobenius inner product.

$$
\begin{aligned}
&\langle X - CC^+X, CC^+X - Y \rangle_F \\
&= \mathrm{Tr}((X - CC^+X)^\top (CC^+X - Y)) \\
&= \mathrm{Tr}((X^\top - X^\top CC^+)(CC^+X - Y)) \, (\because (CC^+)^\top = CC^+) \\
&= \mathrm{Tr}(X^\top CC^+X - X^\top Y - X^\top CC^+CC^+X + X^\top CC^+Y) \\
&= \mathrm{Tr}(X^\top CC^+X - X^\top Y - X^\top CC^+X + X^\top CC^+Y) \, (\because CC^+C = C) \\
&= \mathrm{Tr}(-X^\top Y + X^\top CC^+Y) \\
&= \mathrm{Tr}(-X^\top CC^+XR^+R + X^\top CC^+CC^+XR^+R) \, (\because Y = CUR, U = C^+XR^+) \\
&= \mathrm{Tr}(-X^\top CC^+XR^+R + X^\top CC^+XR^+R) \, (\because CC^+C = C) \\
&= \mathrm{Tr}(0) \\
&= 0
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\|X - CUR\|_F^2 &= \|X - Y\|_F^2 \\
&= \|(X - CC^+X) + (CC^+X - Y)\|_F^2 \\
&= \|X - CC^+X\|_F^2 + \|CC^+X - Y\|_F^2 \\
&\geq \|X - CC^+X\|_F^2
\end{aligned}
$$

□

By Lemma 2.2, the projection (or reconstruction) error $\|X - CC^+X\|_F^2$ is less than or equal to the approximation error $\|X - CUR\|_F^2$. Similarly, it can be shown that the projection error $\|X - V_r V_r^\top X\|_F^2$ is less than or equal to the approximation error $\|X - V_r \Sigma_r W_r^\top\|_F^2$.

2.2 Autoencoders

An *autoencoder* is a neural network designed to extract features of input data and reconstruct the input data from the features; e.g., [13],[48, Ch. 14]. The fundamental architecture consists of two components: an *encoder* that extracts features referred to as latent variables from inputs and a *decoder* that reconstructs the inputs from the latent variables. It can be represented as

$$
\begin{aligned}
\boldsymbol{\rho} &= f(\mathbf{x}), \\
\tilde{\mathbf{x}} &= g(\boldsymbol{\rho}),
\end{aligned}
$$

where f and g denote the encoder and the decoder respectively and where $\mathbf{x}$, $\boldsymbol{\rho}$, and $\tilde{\mathbf{x}}$ denote an input, a latent variable, and a reconstructed input respectively. Accordingly, $\tilde{\mathbf{x}}$ refers to an approximation of $\mathbf{x}$ (i.e., $\tilde{\mathbf{x}} \approx \mathbf{x}$).

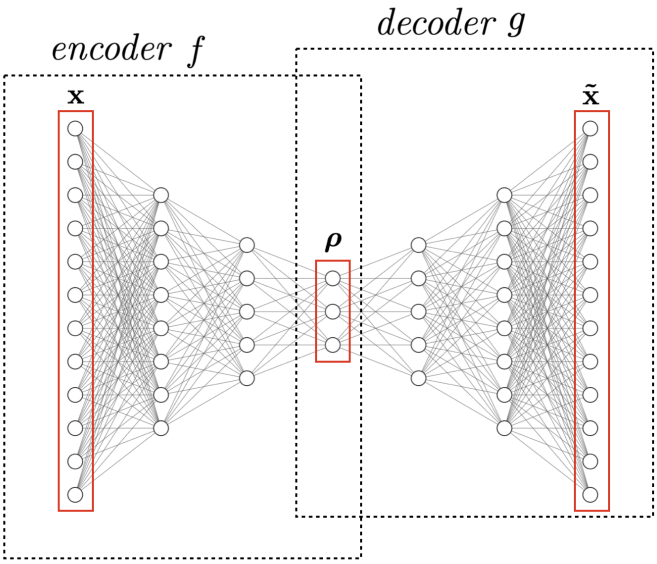


Figure 2.2: A deep autoencoder $g \circ f$: Structure of a deep autoencoder composed of an encoder f and a decoder g, trained to reconstruct $\mathbf{x}$ via the mapping $g(f(\mathbf{x}))$ (i,e, $\mathbf{x} \approx \tilde{\mathbf{x}}$)

The mechanisms of encoding and decoding are not limited to a specific field. Rather, they constitute general principles that are broadly applicable in areas such as information theory, neuroscience, and machine learning. In practice, different forms of autoencoders have been utilized for a wide range of tasks including data generation (e.g., [86, 41, 9, 63]) and dimensionality reduction (e.g., [111, 40, 65, 102]). In data generation, autoencoders are designed to create new data or remove noise from inputs. In contrast, for dimensionality reduction, it is important to reconstruct exact inputs from low-dimensional latent variables in $\mathbb{R}^r$ and the dimension of $\boldsymbol{\rho}$ typically tends to be extremely low compared to the original high dimension n of inputs (e.g., $n = 20,000$ and $r = 50$). However, reconstruction performance diminishes as r decreases, so the key question is how to design dimensionality reduction methods that can effectively use low reduced dimensions while achieving satisfactory reconstruction. To achieve this goal, one possible strategy is to design deep and nonlinear autoencoders.

2.2.1 Dense Autoencoders

The design of basic autoencoders $g \circ f$ typically begins with neural networks comprising *dense layers* as shown in Figure 2.2, and these are commonly referred to as *dense autoencoders*. A dense layer refers to a neural network layer where the connection between two consecutive layers without nonlinear applications such as nonlinear activation functions

is affine linear as follows:

$$\begin{aligned}\mathbf{z}_1 &= W_1\mathbf{x} + \mathbf{b}_1,\\ \mathbf{z}_2 &= W_2\mathbf{z}_1 + \mathbf{b}_2,\\ &\vdots\\ \tilde{\mathbf{x}} &= W_{L-1}\mathbf{z}_{L-2} + \mathbf{b}_{L-1},\end{aligned}$$

where $W_1, \ldots, W_{L-1}$ are learnable weights, $\mathbf{b}_1, \ldots, \mathbf{b}_{L-1}$ are learnable biases, and L is the number of layers. Furthermore, the composite of the affine linear transformations preserves the affine linearity as shown in Remark 2.3.

Remark 2.3:
The composite of the affine linear transformations is affine linear.

Proof.

$$\begin{aligned}\tilde{\mathbf{x}} &= W_{L-1}\mathbf{z}_{L-2} + \mathbf{b}_{L-1},\\ &= W_{L-1}(W_{L-2}\mathbf{z}_{L-3} + \mathbf{b}_{L-2}) + \mathbf{b}_{L-1},\\ &= W_{L-1}(W_{L-2}(W_{L-3}\mathbf{z}_{L-4} + \mathbf{b}_{L-3}) + \mathbf{b}_{L-2}) + \mathbf{b}_{L-1},\\ &\vdots\\ &= W\mathbf{x} + \mathbf{b},\end{aligned}$$

where $W = W_{L-1}W_{L-2}\ldots W_1$ and $\mathbf{b} = \mathbf{b}_{L-1} + W_{L-1}\mathbf{b}_{L-2} + W_{L-2}W_{L-1}\mathbf{b}_{L-3} + \ldots + W_2\ldots W_{L-1}\mathbf{b}_1$. □

This linear nature may lead to high reconstruction error, even with deep autoencoders. Hence, to introduce nonlinearity in deep autoencoders, a nonlinear activation function such as tanh function can be applied in each layer as follows:

$$\begin{aligned}\mathbf{z}_1 &= a_1(W_1\mathbf{x} + \mathbf{b}_1),\\ \mathbf{z}_2 &= a_2(W_2\mathbf{z}_1 + \mathbf{b}_2),\\ &\vdots\\ \tilde{\mathbf{x}} &= a_{L-1}(W_{L-1}\mathbf{z}_{L-2} + \mathbf{b}_{L-1}),\end{aligned}$$

where $a_1, \ldots a_{L-1}$ are activation functions (some of them could be linear). These activation functions are applied elementwise to the output of each linear layer.

In a dense layer, each node is connected to every node in the previous layer; see Figure 2.3. This connection, formed by an affine linear combination, requires expensive computations and large memory resources, particularly when handling large datasets or a large number of nodes. Moreover, dense layers may be inadequate for capturing spatial patterns in datasets such as images or time series, as they lack sparse connectivity. Sparse

connectivity refers to a network structure in which each node is connected to only a subset of other units, resulting in a weight matrix with many zero entries. To overcome these issues, dense layers along with other types of layers such as convolutional, pooling, and recurrent layers have been utilized to build deep neural networks; e.g., [73, 26].

2.2.2 Convolutional Autoencoders

A *convolutional layer* is a neural network layer where the connection between two consecutive layers is defined by the *convolution operation* ($*$) described as

$$(I*K)(x_1,x_2,\ldots,x_D)=\sum_{i_1=0}^{k_1-1}\sum_{i_2=0}^{k_2-1}\cdots\sum_{i_D=0}^{k_D-1}I(x_1-i_1,x_2-i_2,\ldots,x_D-i_D)K(i_1,i_2,\ldots,i_D),$$

where $I(x_1,x_2,\ldots,x_D)$ is the input value at pixel position $(x_1,x_2,\ldots,x_D)\in\mathbb{N}^D$ and $K(i_1,i_2,\ldots,i_D)$ is the parameter of a $k_1\times k_2\times\cdots k_D$ *kernel* K at position $(i_1,i_2,\ldots,i_D)$. The convolution $I*K$ at $(x_1,x_2,\ldots,x_D)$ yields an output value by summing of the products of input values and kernel parameters corresponding to them in a local region centered at the coordinate $(x_1,x_2,\ldots,x_D)$. A convolutional layer includes node values obtained by convolutions applied to the input data using a kernel. The kernel slides across the input data with a step size known as the *stride* and the resulting set of node values is referred to as a *feature map*.

For instance, when I is an image with 32×32 pixels and K is a 3×3 kernel, the convolution is defined as

$$(I*K)(x,y)=\sum_{i=0}^{2}\sum_{j=0}^{2}I(x-i,y-j)K(i,j),$$

where $(x,y)\in\{2,3,\ldots,31\}\times\{2,3,\ldots,31\}$ denotes a pixel point in I. When the stride is set to 1, the kernel moves one pixel at a time and the computations result in a 30×30 feature map I_F in the convolutional layer. Moreover, the convolution operation can be reformulated based on various options such as padding and dilation. To simplify the notation of the convolution, the series of convolution operations as described above to obtain a feature map from an input I is denoted as $I_F=\mathsf{Conv}(I)$.

By the definition of Conv, Conv itself offers several advantages including

1. *Linearity*,
2. *Sparse connectivity*,
3. *Parameter sharing*,
4. *Indirectly dense connectivity*, and
5. *Translation invariance*.

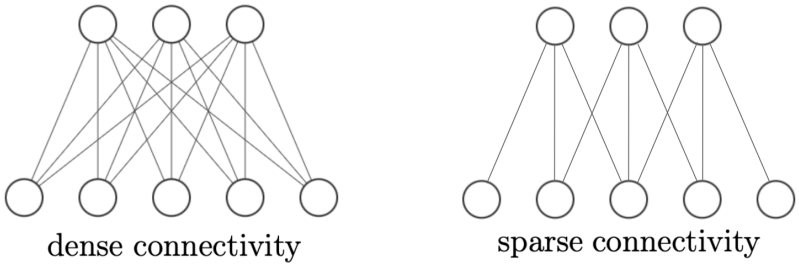


Figure 2.3: Dense connectivity vs. sparse connectivity with a kernel size of 3

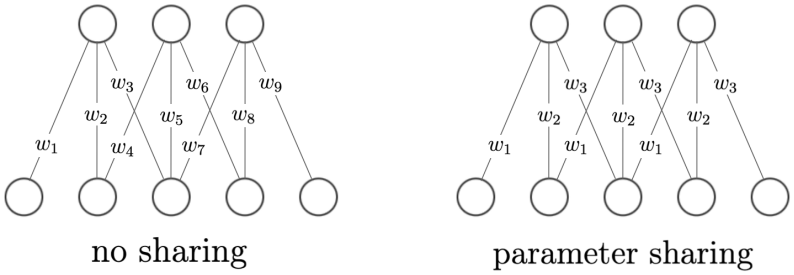


Figure 2.4: No sharing (9 different parameters) vs. parameter sharing (a single kernel with parameters w_1, w_2, and w_3)

Firstly, Conv is linear, and as a result, the composition of multiple convolutional layers applied in sequence also constitutes a linear transformation. In the absence of nonlinear activation functions between these layers, a sequence of convolutional layers can therefore be structurally reparameterized into a single convolutional layer without any loss in representational capacity. In this context, reparameterization means that the combined effect of multiple convolutions can be expressed as one equivalent convolution by appropriately merging their respective kernels; see Remark 3.2. This property not only allows for a reduction in the computational cost and inference time of deep neural networks by eliminating unnecessary operations, but also enables the intentional preservation of linearity in the model architecture; e.g., [34].

Secondly, in Conv each output node is connected only to a limited number of nodes in the previous layer, a characteristic known as sparse connectivity, as illustrated in Figure 2.3. Thus, Conv can use fewer parameters (referred to as weights) and be designed to handle deep hidden layers more efficiently than dense layers. Additionally, it enables neural networks to extract feature maps from local information.

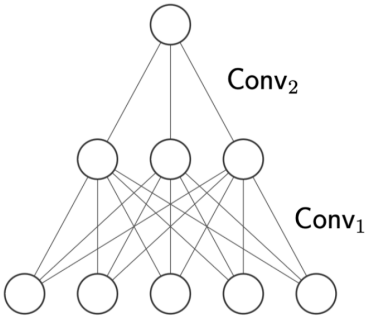


Figure 2.5: Indirectly dense connectivity formed by two convolutional layers

Thirdly, Conv utilizes a single kernel to extract a feature map, meaning that kernel parameters remain unchanged regardless of its position in the input. As depicted in Figure 2.4, parameter sharing indicates the use of the same parameters in each computational area. This property, along with the sparse connectivity, helps reduce the number of parameters.

Fourthly, *receptive field* (e.g., [79]) refers to the size of input values that affect a single node in the output layer. It tends to expand with an increasing number of layers; thus, dense connectivity between input and output nodes in a neural network can be achieved through deep layers, even though any two consecutive layers are sparsely connected. As shown in Figure 2.5, each node in the output of Conv_1 is not connected to all nodes in the input layer. However, the node in the last layer depends on all nodes in the input layer. In this case, the receptive field size equals the number of nodes in the input layer. This property is referred to as indirectly dense connectivity.

Fifthly, Conv exhibits a degree of translation invariance owing to sparse connectivity and parameter sharing. Translation invariance implies that the outcome remains consistent despite translations. For example, although an object is translated in an image, Conv can extract similar features as those from the original image, as its weights remain invariant regardless of the object's position. Consequently, Conv facilitates the classification of the object into the same predicted class as in the original image.

A *convolutional autoencoder* (CAE) is an autoencoder containing one or more convolutional layers. The output tends to be smaller than the feature map in the previous layer. For example, when I is a 2D image with $h \times w$ pixels, K is a $k_1 \times k_2$ kernel, and the stride is set to s, the feature map size $H \times W$ is calculated as follows:

$$H = \frac{h - k_1}{s} + 1,$$
$$W = \frac{w - k_2}{s} + 1.$$

As a downsampling method, Conv is structurally apt for extracting low-dimensional features compared to inputs. However, it is unsuitable for extending dimensions in a

decoder.

To address this issue while possessing properties such as linearity, sparse connectivity, and parameter sharing, *deconvolution* (e.g., [116]) has been commonly employed for decoding latent variables. The goal of deconvolution is to reverse the convolution process. However, it is usually referred to as *transposed convolution*, as the actual inverse of convolution does not exist.

For example, when I is a 2D image with $h \times w$ pixels, K is a $k_1 \times k_2$ kernel, and a stride is set to s, the feature map size $H \times W$ is calculated as follows:

$$H = s(h-1) + k_1,$$
$$W = s(w-1) + k_2,$$

and each value $O(x, y)$ at a pixel point (x, y) in an output feature map I_F is obtained by

$$O(x,y) = \sum_{i=0}^{k_1-1} \sum_{j=0}^{k_2-1} I_v(i,j) K(i,j),$$

where

$$I_v(i,j) = \begin{cases} 0, & \text{if } \lfloor \frac{x-i}{s} \rfloor < 0 \text{ or } \lfloor \frac{y-j}{s} \rfloor < 0 \\ I(\lfloor \frac{x-i}{s} \rfloor, \lfloor \frac{y-j}{s} \rfloor), & \text{otherwise.} \end{cases}$$

To simplify the notation of the deconvolution, the series of deconvolution processes used to obtain a feature map I_F from an input I is denoted as $I_F = \mathsf{DeConv}(I)$. Finally, a standard deep CAE $g \circ f$ is designed using dense, convolutional, and deconvolutional layers with nonlinear activation functions.

2.2.3 Autoencoder Optimization

Autoencoders are designed to reconstruct inputs, so the objective function for model optimization measures the error between inputs and outputs without labelled data. Thus, the training strategy is unsupervised learning which refers to training neural networks using unlabeled data as shown in Algorithm 2.5.

Remark 2.4 (Supervised learning vs. unsupervised learning):
Let τ be a neural network that defines a mapping from an input space to an output space, expressed as $\tilde{\mathbf{y}} = \tau(\mathbf{x})$, where $\mathbf{x}$ is the input and $\tilde{\mathbf{y}}$ denotes the output. Supervised learning refers to a training paradigm in which the model learns from labeled data, meaning that each input $\mathbf{x}$ is paired with a corresponding ground-truth output $\mathbf{y}$. The goal of supervised learning is to optimize the model parameters such that the predicted output $\tilde{\mathbf{y}}$ closely matches the true output $\mathbf{y}$, typically by minimizing a loss function that quantifies prediction error. In contrast, unsupervised learning deals with unlabeled data where no explicit target output $\mathbf{y}$ is provided.

Algorithm 2.5: Training an autoencoder $g \circ f$

Input: Training dataset $X = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, learning rate η, number of epochs e
Output: Trained encoding and decoding parameters; θ_f, θ_g

Initialize model parameters θ_f, θ_g
for *epoch* $= 1$ *to* e **do**
  for $B \subset X$ *(B: random mini-batch)* **do**
    Compute $Z = f(B; \theta_f)$ `// Encoding` B `to latent variables` Z
    Compute $\tilde{X} = g(Z; \theta_g)$ `// Decoding` Z `to reconstruction` $\tilde{X}$
    Compute $\mathcal{L} = \frac{1}{\#B} \sum_{\mathbf{x}^{(j)} \in B, \tilde{\mathbf{x}}^{(j)} \in \tilde{X}} \|\mathbf{x}^{(j)} - \tilde{\mathbf{x}}^{(j)}\|_2^2$ `// Reconstruction loss`
    Update parameters $\theta_f \leftarrow \theta_f - \eta \nabla_{\theta_f} \mathcal{L}$, $\theta_g \leftarrow \theta_g - \eta \nabla_{\theta_g} \mathcal{L}$
return θ_f, θ_g

Instead, the objective is to uncover hidden structures, patterns, or distributions inherent in the input data $\mathbf{x}$. Common approaches include clustering and dimensionality reduction. In this setting, the model learns to represent or organize the data in a meaningful way without direct supervision. ◊

The reconstruction loss function serves as a critical component in training models for dimensionality reduction and representation learning. It quantifies the discrepancy between the original data and its reconstructed approximation. Depending on the nature of the data and the specific goals of the learning task, various loss functions can be employed. Common choices include the mean squared error (MSE), which penalizes large deviations in a Euclidean sense, and the Kullback–Leibler (KL) divergence, which is often used when a probabilistic interpretation of the data distribution is desired. For instance, MSE is adopted in Algorithm 2.5 to measure reconstruction fidelity. To ensure tractable and stable optimization, it is advantageous to formulate the loss function as a convex function, as this guarantees the existence of a unique global minimum and facilitates efficient convergence of the optimization algorithm. However, in practice, the loss function with respect to the model parameters is typically non-convex due to the inherent nonlinearity of neural networks, making the optimization process particularly challenging.

In terms of reconstruction performance, a decrease in the number of output nodes (a reduced dimension of r) from the encoder typically leads to a decline in reconstruction quality, similar to the behavior observed in linear dimensionality reduction methods.

Remark 2.5 (Gradient descent):
Descent methods are optimization algorithms that aim to find a local minimum (and preferably a global minimum) of a function $\mathcal{L}$ with respect to a variable vector θ by moving in the direction of the negative gradient of $\mathcal{L}$. The general form of the update can be written as

$$\theta \leftarrow \theta + \eta \Delta\theta, \; \nabla_\theta \mathcal{L}(\theta)^\top \Delta\theta < 0,$$

where $\Delta\theta$ is the descent direction and η denotes the *learning rate* which adjusts the step size. If the descent direction is chosen as $\Delta\theta = -\nabla_\theta \mathcal{L}(\theta)$, then the condition is satisfied:

$$\nabla\mathcal{L}(\theta)^\top \Delta\theta = -\nabla_\theta \mathcal{L}(\theta)^\top \nabla\mathcal{L}(\theta) < 0.$$

In this case, the update rule becomes

$$\theta \leftarrow \theta - \eta\nabla_\theta \mathcal{L}(\theta),$$

which defines the well-known *gradient descent* method.

Gradient descent is an iterative optimization method widely used in machine learning to minimize an objective function $\mathcal{L}$, typically referred to as a loss function with respect to model parameters θ. At each iteration, the model parameters θ are updated by moving in the direction of the negative gradient of the loss function. However, gradient descent requires computation of the gradient over the entire dataset at every iteration, which can be computationally expensive and memory-intensive for large-scale data. To address this limitation, *stochastic gradient descent* (SGD) is commonly used, where the gradient is approximated using a randomly sampled subset (mini-batch) of the data at each iteration. ◊

For the training of model parameters, advanced variants of SGD are widely employed due to their scalability and practical effectiveness in navigating high-dimensional parameter spaces. SGD updates the model parameters by computing gradients based on a small sampled subset of the training dataset, referred to as a mini-batch. This approach is applied iteratively, and each update step contributes to gradually optimizing the model toward minimizing the loss function as explained in Remark 2.5. Thus, SGD significantly reduces the memory requirement per iteration, which is particularly advantageous when dealing with large-scale datasets, as the number of batches tends to be large. However, despite its computational efficiency, SGD is inherently stochastic and may suffer from convergence issues. In particular, it has the potential to become trapped in local minima or flat saddle points, especially in non-convex optimization landscapes, which are common in deep learning models.

Enhancements to the standard SGD, such as momentum-based methods, which help accelerate convergence and mitigate oscillations, and adaptive learning rate algorithms like Adam, have become the default choices for many machine learning applications (e.g., [105, 69]). These methods adapt the step sizes dynamically for each parameter, thereby improving convergence in problems with sparse gradients or noisy loss landscapes.

Furthermore, learning rate scheduling methods (e.g., [78, 54]) can be employed to further enhance the performance of SGD-based optimizers. These methods adjust the learning rate dynamically during training and are typically designed to reduce the learning rate over time to balance the trade-off between rapid convergence in the early stages and fine-tuning near the optimum. In addition, learning rate schedules help avoid issues such as overshooting or getting stuck in sharp local minima by following predefined rules for how to adjust the learning rate.

In scenarios where higher precision or more nuanced curvature information is required, second-order optimization techniques can be employed. One notable example is the limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) algorithm [20], which approximates the Hessian matrix and provides faster convergence in some settings. Although computationally more intensive, these methods are suitable for smaller-scale problems or as a fine-tuning step following SGD-based pretraining.

Overfitting occurs when a model learns to fit the training data too closely, thereby failing to generalize well to unseen data. This results in high accuracy on the training set but poor performance on validation or test sets. It indicates that the model has learned specific details rather than learned the underlying structure of the data. To address these challenges, regularization methods (e.g., [100, 106]), batch sampling, and early stopping are employed to constrain the model complexity and prevent it from fitting the noise in the data.

The development and deployment of such optimization strategies have been greatly simplified by modern deep learning frameworks such as Pytorch and TensorFlow. These libraries provide automatic differentiation capabilities that eliminate the need for manual gradient computation, thus enabling researchers and practitioners to focus primarily on high-level model design and hyperparameter tuning.

2.3 Clustering

Classification and *clustering* algorithms typically divide a dataset X into disjoint subsets, $C_i, i = 1, 2, \ldots k$ such that $X = \cup_{i=1,2,\ldots k} C_i$. In classification, each C_i is termed a *class*, while in clustering, it is referred to as a *cluster*. They have been extensively applied to diverse domains including spam detection, object detection, and recommendation systems; e.g., [89, 118, 3].

Classification is a supervised learning method that requires labeled data during model training. While the use of actual labels achieves high accuracy compared to clustering models, labeling is time-consuming and carries the risk of mislabeling.

In contrast, clustering is an unsupervised learning method that does not require labels for training. Whereas clustering typically lags behind classification models in terms of accuracy, it remains feasible unlike classification models, even when a large amount of unlabeled data is acquired or when data cannot be labeled.

In this thesis, fluid flows are treated as snapshot datasets. Labeling such data for classification tasks is particularly challenging for several reasons. First, fluid flows often exhibit high-dimensional, nonlinear, and multiscale features, and the transitions between different flow regimes are typically continuous rather than discrete. As a result, it is difficult to define clear class boundaries. Furthermore, fluid flows frequently exhibit complex features that are not easily discernible through visual inspection, especially in transitional or turbulent regimes. This lack of clear visual separation complicates the labeling process, and generating accurate labels requires expert knowledge. Given these difficulties,

Algorithm 2.6: Training k-means clustering

Input: Training dataset $X = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, number of clusters k
Output: Optimal centroids $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k$

```
Initialize centroids c_1, c_2, ..., c_k
while convergence criterion has not been satisfied do
    for i = 1 to N do
        for j = 1 to k do
            Compute the distance d_j = ||x^(i) − c_j||_2
        Assign each x^(i) to the l-th cluster where l = argmin_{j=1,2,...,k} d_j
    Update each centroid to the mean of data in each cluster
    Check a predefined convergence criterion
return c_1, c_2, ..., c_k
```

clustering provides a more suitable framework for analyzing fluid flows in an unsupervised manner. By grouping snapshots into clusters based on inherent similarities, clustering can identify characteristic flow patterns without relying on predefined labels. Motivated by these advantages, clustering algorithms are employed to identify distinct flow patterns.

Since k-means clustering requires low computational cost and offers straightforward implementation compared to other clustering algorithms, it has been chosen for initial exploration and benchmarking. On the other hand, deep clustering more effectively captures complex features in the data by learning low-dimensional representations and performing clustering in the latent space rather than in the original high-dimensional space. For these reasons, k-means clustering and deep clustering are considered appropriate methods for clustering flow snapshot data.

2.3.1 k-means Clustering

k-means clustering is one of the most popular clustering algorithms for grouping a dataset into a predefined number of clusters; e.g., [80, 12, 96]. The training procedure begins by initializing a set of k centroids, each representing the center of a potential cluster. During each iteration, the algorithm assigns each data point to the cluster whose centroid is closest in terms of a chosen distance metric such as the L_2 distance. This assignment phase ensures that data points within the same cluster exhibit high similarity, while those in different clusters are relatively dissimilar. Once the assignments are completed, the centroids are updated by computing the mean of all data points currently assigned to each cluster. These assignment and update steps are repeated until a convergence criterion is satisfied. For instance, a commonly used convergence criterion is the minimization of the total within-cluster sum of squared distances (WCSS) between each data point and

the centroid of its assigned cluster, which is defined as

$$\text{WCSS} = \sum_{j=1}^{k} \sum_{\mathbf{x} \in C_j} \|\mathbf{x} - \mathbf{c}_j\|_2^2,$$

where $\mathbf{x}^{(i)}$ denotes i-th data point, $\mathbf{c}_j$ is the centroid of cluster C_j, and k is the number of clusters. Alternatively, a maximum number of iterations can also be used as a stopping condition. During inference, the algorithm computes the distance between a given input and all centroids, then assigns the input to the cluster whose centroid yields the minimum distance. The overall procedure for training k-means clustering is summarized in Algorithm 2.6.

k-means clustering is widely used due to several properties including

1. *Intuitiveness*,
2. *Computational efficiency*, and
3. *Flexibility in choosing the number of clusters.*

Firstly, its concept and implementation are highly comprehensible making it a common baseline clustering model for comparison with other clustering methods and easy to integrate with other machine learning techniques.

Secondly, it computes (Euclidean) distances to assign given data points to clusters, which results in efficient computation without any additional procedures for clustering.

Thirdly, the number of clusters can be selected based on datasets and analysis purposes. If the number of clusters is not predefined, methods such as the elbow method (e.g., [95]) can be used to determine an appropriate number.

However, k-means clustering can face several challenges in high-dimensional spaces, known as the *curse of dimensionality*; e.g., [2, 19]. In high-dimensional spaces, the effectiveness of distance metrics diminishes, which results in suboptimal centroids. Furthermore, the computational cost increases as the number of features (the dimension of $\mathbf{x}^{(i)}$ in Algorithm 2.6) grows.

To alleviate these problems, the dimensionality of a given dataset can be reduced using *feature selection* and *feature extraction* before clustering; e.g., [64]. Feature selection involves choosing certain features from the given features while retaining important information. Feature extraction refers to methods of mapping high-dimensional data onto a low-dimensional space, preserving essential information. For instance, latent variables in a relatively low-dimensional space are obtained from a neural network (as a feature extraction), and then k-means clustering groups these variables into clusters instead of directly using the original data for clustering.

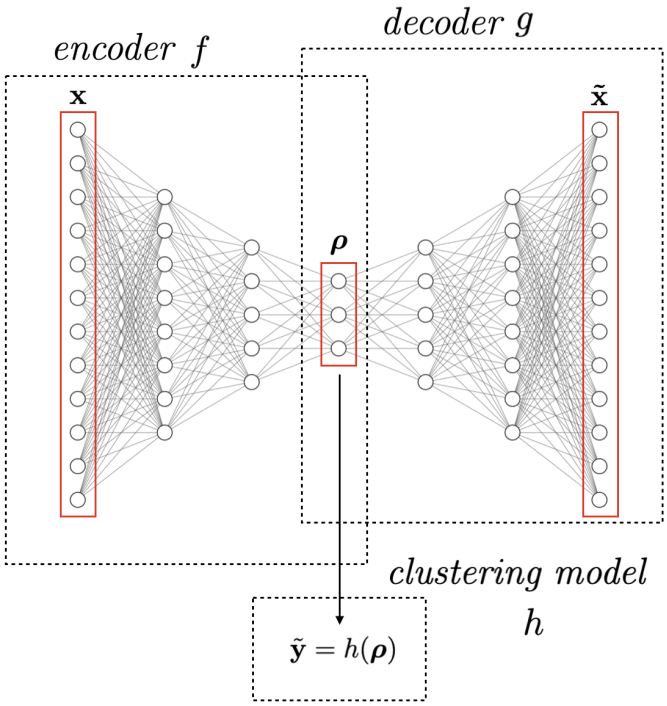


Figure 2.6: A model architecture for DAE-based deep clustering

2.3.2 Deep Clustering

Deep clustering is a technique that combines deep neural networks with clustering algorithms. This approach has been implemented through a variety of model architectures, including deep autoencoder (DAE)-based methods (e.g., [51, 40, 114]) and deep neural network (DNN)-based methods (e.g., [23, 25, 92]). The fundamental idea is to use deep neural networks to project input data into a latent space where the low-dimensional representations serve as meaningful features for clustering. Typically, a deep clustering model consists of two main components: a feature extraction module and a clustering part. In the feature extraction stage, a deep neural network or an autoencoder is employed to learn informative low-dimensional embeddings of the input data. After this step, a clustering algorithm, such as k-means, is applied to classify these latent representations into distinct clusters.

Among the various architectural options for deep clustering, DAE-based models have demonstrated capabilities in simultaneously reconstructing input data and learning latent representations; e.g., [51, 40, 114]. The structure of DAEs naturally supports both dimensionality reduction and input reconstruction, which makes them especially suitable for model order reduction tasks. Due to these properties, DAE-based deep clustering models are well-suited for applications that require both efficient low-dimensional parameteriza-

Algorithm 2.7: Training a DAE-based deep clustering model

Input: Training dataset $(X, Y) = \{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^{N}$, learning rate η, number of epochs e, number of clusters k, clustering loss weight λ

Output: Trained model parameters; θ_f, θ_g, and θ_h

Initialize model parameters θ_f and θ_g

for *epoch* $= 1$ *to* e **do**

  for $(B_X, B_Y) \subset (X, Y)$ *((B_X, B_Y): random mini-batch)* **do**

    Compute $Z = f(B_X; \theta_f)$ `// Encoding` B `to latent variables` Z

    Compute $\tilde{X} = g(Z; \theta_g)$ `// Decoding` Z `to reconstruction` $\tilde{X}$

    Compute $\tilde{Y} = h(Z; \theta_h)$ `// Classification`

    Compute $\mathcal{L}_r = \frac{1}{\#B} \sum_{\substack{\mathbf{x}^{(j)} \in B_X \\ \tilde{\mathbf{x}}^{(j)} \in \tilde{X}}} \|\mathbf{x}^{(j)} - \tilde{\mathbf{x}}^{(j)}\|_2^2$ `// Reconstruction loss`

    Compute $\mathcal{L}_c = \frac{1}{\#B} \sum_{\substack{\mathbf{y}^{(i)} \in B_Y \\ \tilde{\mathbf{y}}^{(i)} \in \tilde{Y}}} \mathbf{y}^{(i)} \cdot \log \frac{\mathbf{y}^{(i)}}{\tilde{\mathbf{y}}^{(i)}}$ `// Clustering loss`

    Compute $\mathcal{L} = \mathcal{L}_{rec} + \lambda \mathcal{L}_{clt}$ `// Joint loss`

    Update parameters $\theta_f, \theta_g, \theta_h$ by a SGD method

return θ_f, θ_g, θ_h

tions and meaningful clustering of system states. DAE-based deep clustering models are especially appealing because they perform three essential tasks as follows:

1. *Nonlinear dimensionality reduction (encoding)*
2. *Data reconstruction (decoding)*
3. *Clustering (i.e., data classification without labeling)*

As shown in Figure 2.6, a typical DAE-based deep clustering model consists of three neural network components: an encoder, a decoder, and a clustering network. These can be described as

$$\begin{aligned} \boldsymbol{\rho} &= f(\mathbf{x}), \\ \tilde{\mathbf{x}} &= g(\boldsymbol{\rho}), \\ \tilde{\mathbf{y}} &= h(\boldsymbol{\rho}), \end{aligned}$$

where f, g, and h are an encoder, a decoder, and a neural network for clustering respectively and where $\mathbf{x}$, $\boldsymbol{\rho}$, $\tilde{\mathbf{x}}$, and $\tilde{\mathbf{y}}$ denote an input, a latent variable, a reconstructed input ($\tilde{\mathbf{x}} \approx \mathbf{x}$), and a label assigned to $\mathbf{x}$ respectively.

The encoder f maps the input data to a lower-dimensional latent space, effectively compressing the information while hopefully preserving its essential structure. The decoder g reconstructs the original input from the latent code, encouraging the latent space to capture the underlying manifold of the data distribution. The clustering model h, typically

implemented as a fully connected neural network, assigns each latent state to a cluster in an unsupervised fashion. By jointly optimizing the reconstruction loss and clustering objective, the model learns a latent space that is not only informative and compact but also discriminative for clustering purposes.

For the model optimization, Algorithm 2.5 can be employed when h is a well-defined classical clustering algorithm such as k-means clustering, as an unsupervised learning approach. In contrast, when h is a semi-supervised learning-based neural network that uses target labels $\mathbf{y}$ for optimizing the clustering model h, the objective function can be defined as a joint loss function $\mathcal{L}$ comprising both the reconstruction loss $\mathcal{L}_r$ and clustering loss $\mathcal{L}_c$ as shown in Algorithm 2.7. For instance, the MSE and KL-divergence can be chosen for the reconstruction loss and clustering loss respectively. However, the target labels $\mathbf{y}$ are not provided, so they must be collected in advance of training the model. To obtain $Y = \{\mathbf{y}^{(i)}\}_{i=1}^{N}$ as a set of pseudo labels, one possible way is to use classical clustering algorithms.

A key advantage of DAE-based deep clustering models lies in their ability to extract compact latent representations that preserve semantically meaningful features, while simultaneously facilitating accurate and interpretable clustering in an unsupervised setting. These capabilities have led to widespread applications across a variety of domains such as computer vision (e.g., [114, 50]), bioinformatics (e.g., [30]), and natural language processing (e.g., [4]).

2.4 Flow Models and Data

Contents

2.4.1 Incompressible Navier-Stokes Equations 28
2.4.2 Viscous Burgers' Equations 31

A wide range of physical phenomena can be modeled as flow dynamical systems, including aerodynamics, weather forecasting, and biological flows. However, achieving high fidelity in simulating these systems often requires substantial computational resources. Therefore, developing efficient and accurate flow models, such as surrogate models and reduced-order models, becomes essential. In this context, flow snapshot data provide essential information for building reduced-order and surrogate models that can approximate the original fluid flows while retaining physical properties. A snapshot refers to the state of a system at a specific time in the context of time-dependent problems, such as fluid flow or heat diffusion. Such snapshots are commonly used for post-simulation analysis and for training data-driven models. For this purpose, flow snapshots generated from numerical simulations governed by either the incompressible Navier-Stokes equations or viscous Burgers' equations are used to train and evaluate the proposed methods presented in Chapters 3 to 5. This section introduces these equations and explains how

their corresponding discrete algebraic equations are derived.

2.4.1 Incompressible Navier-Stokes Equations

The behavior of incompressible fluids is governed by the incompressible *Navier–Stokes equations*, which embody the fundamental principles of mass and the momentum conservation; see [66]. Incompressible flow problems are not only of intrinsic theoretical importance, but also form the basis of more sophisticated models used to describe a wide range of physical phenomena and engineering processes, such as hemodynamics and aerospace applications; e.g., [110, 32]. By accounting for both inertial and viscous effects, the incompressible Navier–Stokes framework enables the simulation of diverse flow regimes, ranging from steady laminar motion to fully developed turbulence. However, solving the Navier-Stokes equations analytically poses significant challenges due to their complexity. As a result, numerical methods are commonly utilized to obtain the approximation of the solutions for the analysis and prediction of fluid behavior.

The incompressible Navier-Stokes equations can be described as

$$\frac{\partial}{\partial t}\mathrm{v} + (\mathrm{v}\cdot\nabla)\mathrm{v} + \frac{1}{\mathsf{Re}}\Delta\mathrm{v} - \nabla\mathrm{p} = \mathrm{f}, \tag{2.15a}$$

$$\nabla\cdot\mathrm{v} = 0, \tag{2.15b}$$

where v, p, f, and Re are the velocity, pressure, forcing term, and Reynolds number for time $t > 0$ respectively. As the ratio of inertial to viscous forces, the Reynolds number Re characterizes flow motions. For instance, low Reynolds numbers generally result in laminar, stable flows, whereas high Reynolds numbers can lead to turbulence such as vortices and eddies. However, even at relatively low Reynolds numbers (e.g., $\mathsf{Re} = 60$), chaotic or periodic flow patterns can arise depending on simulation settings such as the spatial domain.

By the spatial discretization of Equation (2.15) based on the *finite element method* (FEM), a semi-discrete model is obtained in the form of the following system:

$$\mathsf{M}\dot{\mathbf{x}}(t) + \mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) - \mathsf{J}^\top\mathbf{p}(t) - \mathbf{f}(t) = \mathbf{0}, \tag{2.16a}$$

$$\mathsf{J}\mathbf{x}(t) = \mathbf{0}, \tag{2.16b}$$

where $\mathbf{p}(t) \in \mathbb{R}^p$ and $\mathbf{x}(t) \in \mathbb{R}^n$ denote the states of the velocity and the pressure at time t respectively, and where $\mathsf{M} \in \mathbb{R}^{n\times n}$, $\mathsf{N}(\cdot) \in \mathbb{R}^{n\times n}$, $\mathsf{A} \in \mathbb{R}^{n\times n}$, $\mathsf{J} \in \mathbb{R}^{p\times n}$, and $\mathbf{f}(t) \in \mathbb{R}^n$ are the mass, linear state-dependent coefficient convection, diffusion, discrete divergence matrices, and the forcing term at time t respectively; see e.g., [15] for implementation details.

In the form of an *ordinary differential equation* (ODE), a formulation that excludes the pressure term $\mathbf{p}(t)$ from Equation (2.16) is employed; cp. [6]. It is assumed that M and $\mathsf{J}\mathsf{M}^{-1}\mathsf{J}^\top$ are invertible, which is always the case for stable FEM discretizations, and it is sure that $\mathsf{J}\mathbf{x}(t) = \mathbf{0}$ leads to $\mathsf{J}\dot{\mathbf{x}}(t) = \mathbf{0}$.

	Single-cylinder	**Double-cylinder**
Spatial domain	$(0,5) \times (0,1) \subset \mathbb{R}^2$	$(-20,50) \times (-20,20) \subset \mathbb{R}^2$
Time interval	$[0, 16]$	$[240, 760]$
State dimension (n)	42764	46014
Reynolds number (Re)	40	60
Number of snapshots (m)	800	1152

Table 2.1: Configuration of the FEM simulation (2.18) for two data sets; wake flows behind either a single circular cylinder or two circular cylinders respectively as shown in Figure 2.7.

By these properties, Equation (2.16) can be reformulated as follows:

$$\mathsf{J}^\top \mathbf{p}(t) = \mathsf{M}\dot{\mathbf{x}}(t) + \mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) - \mathbf{f}(t). \tag{2.17}$$

Multiplying both sides of Equation (2.17) by M^{-1} yields

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

Then, by multiplying J on both sides,

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

Since $\mathsf{J}\dot{\mathbf{x}}(t) = \mathbf{0}$, this simplifies to

$$\mathsf{J}\mathsf{M}^{-1}\mathsf{J}^\top \mathbf{p}(t) = \mathsf{J}\mathsf{M}^{-1}(\mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) - \mathbf{f}(t)).$$

Let $\mathsf{S} = \mathsf{J}\mathsf{M}^{-1}\mathsf{J}^\top$. Since S is invertible,

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

As a result, by replacing $\mathbf{p}(t)$ with $\mathsf{S}^{-1}\mathsf{J}\mathsf{M}^{-1}(\mathsf{N}(\mathbf{x}(t))\mathbf{x}(t)+\mathsf{A}\mathbf{x}(t)-\mathbf{f}(t))$ in Equation (2.16), the ODE formulation is described as

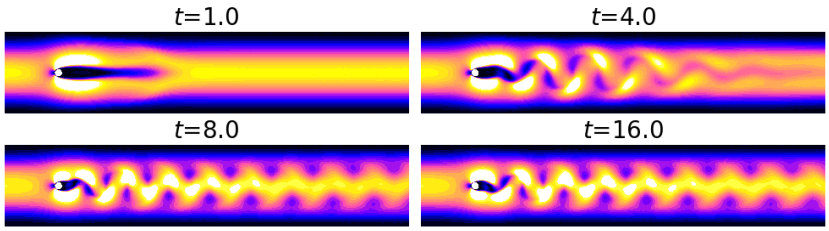
$$\mathsf{M}\dot{\mathbf{x}}(t) = \mathsf{\Pi}^\top(\mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) - \mathbf{f}(t)), \tag{2.18}$$

where $\mathsf{\Pi} = \mathsf{M}^{-1}\mathsf{J}^\top\mathsf{S}^{-1}\mathsf{J} - \mathsf{I}_n$.

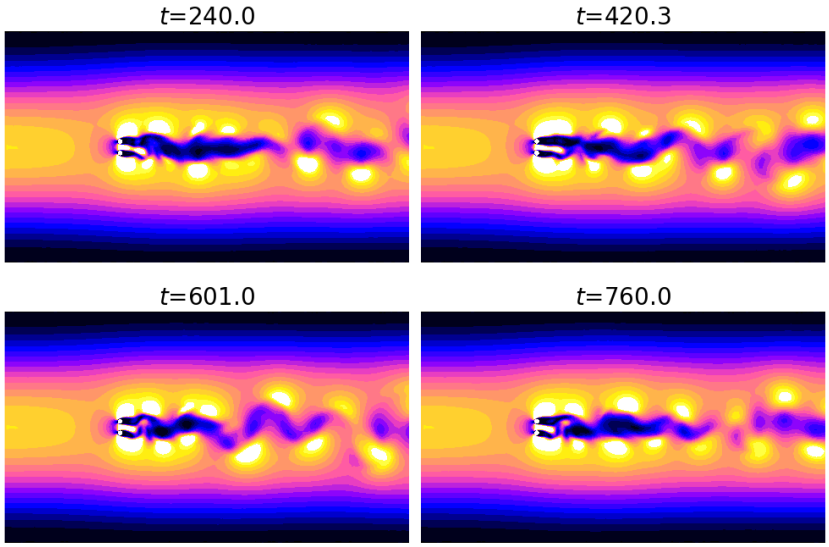
All snapshots are generated by the implicit-explicit Euler scheme with respect to time t in Equation (2.18) and the velocity data are stored as a matrix form

$$X = \begin{bmatrix} | & | & & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{m-1} & \mathbf{x}_m \\ | & | & & | & | \end{bmatrix},$$

where $\mathbf{x}_1 = \mathbf{x}(0)$, $\mathbf{x}_2 = \mathbf{x}(\Delta t)$, $\mathbf{x}_3 = \mathbf{x}(2\Delta t)$, $\dots$, $\mathbf{x}_m = \mathbf{x}((m-1)\Delta t)$ for $m \in \mathbb{N}$. Δt is a time step for saving equidistant snapshots; i.e., Δt is not a time step used during the numerical simulation of Equation (2.18).



(a) Single-cylinder case



(b) Double-cylinder case

Figure 2.7: Two cases of incompressible flows

In two-dimensional spatial domains, each snapshot contains $x-$ and $y-$directional velocity values, u_i and w_i, on a predefined mesh topology with the indices of $\frac{n}{2}$ mesh points (i.e., $i \in \{1, 2, \ldots, \frac{n}{2}\}$).

The cylinder wake is a well-established benchmark example known for generating diverse flow patterns based on simulation configurations and the Reynolds number.High Reynolds numbers often lead to flow instabilities such as eddies and vortices. However, relatively low Reynolds numbers (e.g., $\mathsf{Re} \in [40, 100]$) can also result in unsteady flows including periodic or chaotic behavior, depending on the simulation configuration. Consequently, low Reynolds numbers are frequently employed to study flow patterns, model order reduction, and control systems; e.g., [28, 107, 33, 17].

Based on the configurations in Table 2.1, periodic flows behind a single cylinder and chaotic flows behind two cylinders are generated, as illustrated in Figure 2.7.

In the single-cylinder case, the first 500 snapshots (i.e., $t \in [0, 10]$) are used to train low-dimensional parametrization models, while the remaining 300 snapshots in the time

	Burgers' flows
Spatial domain	$(-1,1) \times (-1,1) \subset \mathbb{R}^2$
Time interval	$[0, 0.5]$
State dimension (n)	20402
Kinematic viscosity (ν)	10^{-4}
Number of snapshots (m)	100

Table 2.2: Configuration of the FEM simulation (2.20) for the traffic flow data set as shown in Figure 2.8

interval $[10, 16]$ are used to evaluate the extrapolation performance of the pretrained models.

In the double-cylinder case, 720 snapshots from the time interval $[240, 480]$ are used for training, as the flow exhibits relatively less chaotic behavior in the earlier interval $[0, 240]$. For evaluation, 432 snapshots from the time domain $[480, 760]$ are employed.

The procedures for training and evaluating the proposed methods using these datasets, denoted by X_{single} and X_{double} respectively, are discussed in Chapters 3 to 5.

2.4.2 Viscous Burgers' Equations

As another benchmark example, the viscous *Burgers' equations* are considered, given by

$$\frac{\partial}{\partial t}\mathrm{v} + (\mathrm{v} \cdot \nabla)\mathrm{v} - \nu\Delta\mathrm{v} = 0, \tag{2.19}$$

where v and ν are the velocity and the kinematic viscosity for time $t > 0$ respectively. The equations are often used to simulate diverse flow phenomena including shock waves; e.g., [46].

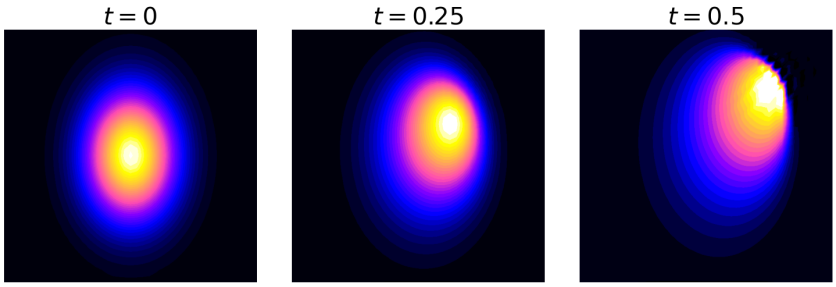
Similar to the spatial discretization in Section 2.4.1, spatially discretized versions of Equation (2.19) can be expressed as

$$\mathsf{M}\dot{\mathbf{x}}(t) + \mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) = \mathbf{0}, \tag{2.20}$$

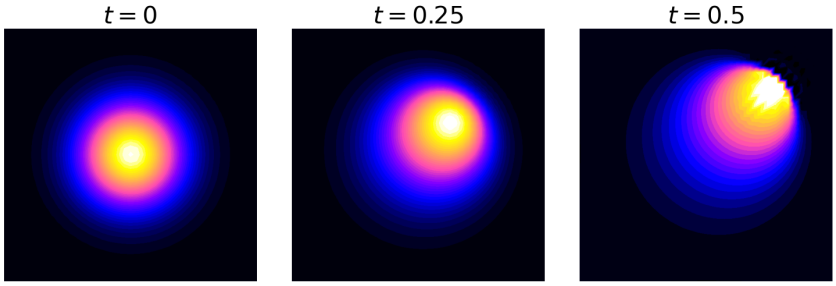
where $\mathbf{x}(t) \in \mathbb{R}^n$ denotes the states of the velocity at time $t > 0$, and $\mathsf{M} \in \mathbb{R}^{n\times n}$, $\mathsf{N}(\cdot) \in \mathbb{R}^{n\times n}$, $\mathsf{A} \in \mathbb{R}^{n\times n}$ are the mass, the state-dependent convection, and the diffusion matrices at time t respectively.

Based on the configuration shown in Table 2.2, snapshots are generated using the implicit Euler method for time t in Equation (2.20). Since this model results in a blow-up at the final time (at least with the spatial discretization used here), The performance of the reduced models in extrapolation is assessed by applying the trained approaches to states that evolve within the same time interval but originate from a different initial condition. Thus, equidistant snapshots are collected from the time evolution of two distinct initial conditions,

$$\mathbf{x}(0) = [\{e^{-4x^2-2y^2} | (x, y) \in (-1, 1) \times (-1, 1)\}]^\top \tag{2.21}$$



(a) Flow snapshots with the initial condition (2.21)



(b) Flow snapshots with the initial condition (2.22)

Figure 2.8: Comparison of Burgers' flows under two distinct initial conditions

and

$$\mathbf{x}(0) = [\{e^{-3x^2-3y^2} | (x, y) \in (-1, 1) \times (-1, 1)\}]^\top, \tag{2.22}$$

respectively where each coordinate (x, y) corresponds to a mesh point in the spatial area $(-1, 1) \times (-1, 1)$. The dataset X_{bg_1} with the initial condition (2.21) is used for training models, while the dataset X_{bg_2} with the initial condition (2.22) is used to evaluate the reconstruction performance of the pretrained models.

In Chapter 3, the proposed methods are trained using X_{bg_1} and evaluated on X_{bg_2} to investigate state reconstruction performance under conditions where noise is present in the states.

2.5 Reduced-order Models and LPV Approaches

Dynamical simulations, especially those involving high-dimensional systems such as fluid flows or structural dynamics, frequently entail significant computational expense, which can make their use impractical in real-time applications or on resource-constrained hardware platforms. To overcome these challenges, *reduced-order models* (ROMs) have been extensively developed and employed as efficient surrogates for the full order models. ROMs aim to capture the dominant dynamics of the system while significantly reducing the computational burden. For reduced-order modeling, high-dimensional state trajectories are typically collected over time and a low-dimensional parametrization or surrogate

model is then constructed using data-driven techniques such as projection-based methods, operator learning, and nonlinear embeddings; e.g., [14, 16, 94].

Reduced-order modeling can be categorized into *intrusive* and *non-intrusive* approaches, depending on the level of access to the full-order model (FOM). Intrusive methods assume full access to the governing equations of the FOM and construct a ROM by directly modifying the original equations through low-dimensional parametrizations; e.g., [14, 101]. In contrast, non-intrusive methods treat the FOM as a black box and build surrogate models based on data collected from simulations or experiments. These approaches are particularly appealing when the underlying equations are unavailable, difficult to access, or impractical to modify; e.g., [16, 43, 67].

Regardless of the modeling approaches, reduced-order models are designed to alleviate the computational costs of full-order models while ensuring that the approximation error remains within acceptable bounds and that the dynamical features are well-preserved.

For instance, the methods introduced in Sections 2.1 and 2.2 form the backbone of many ROM frameworks by extracting low-dimensional structures from high-dimensional data, thereby enabling fast and accurate approximations of complex dynamical behaviors; e.g., [104, 82].

Given a high-dimensional system

$$\dot{\mathbf{x}}(t) = \eta(\mathbf{x}(t)), \tag{2.23}$$

where $\eta : \mathbb{R}^n \to \mathbb{R}^n$ is a function and $\mathbf{x}(t)$ is an $n \times 1$ state of the system at time t (e.g., $n = 10^6$).

Define a dataset

$$X = \begin{bmatrix} | & | & & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_{m-1} & \mathbf{x}_m \\ | & | & & | & | \end{bmatrix},$$

where $\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_m$ are $n \times 1$ state vectors selected from states within the predefined time interval.

Using POD with a small reduced dimension of r ($r \ll n$), a POD basis $V_r \in \mathbb{R}^{n \times r}$ is obtained such that $\mathbf{x}(t)$ is approximated by

$$\mathbf{x}(t) \approx V_r \boldsymbol{\rho}(t) = V_r(V_r^\top \mathbf{x}(t)),$$

where $\boldsymbol{\rho}(t) \in \mathbb{R}^r$ is a latent state for any time t, as mentioned in Sections 2.1 and 2.2.

Assuming that POD guarantees a certain level of approximation error, the state $\mathbf{x}(t)$ in Equation (2.23) is replaced by $V_r\boldsymbol{\rho}(t)$, leading to

$$V_r\dot{\boldsymbol{\rho}}(t) = \eta(V_r\boldsymbol{\rho}(t)).$$

Since $V_r^\top V_r = \mathsf{I}_r$, a reduced order model of Equation (2.23) can be defined as

$$\dot{\boldsymbol{\rho}}(t) = V_r^\top \eta(V_r\boldsymbol{\rho}(t)).$$

Another approach for reduced-order modeling is the *linear parameter-varying (LPV) approximation*, which provides a systematic framework for reducing the order of nonlinear and high-dimensional dynamical control systems. The core concept of LPV approximation involves expressing the nonlinear components of a system in terms of affine linear functions that depend on a set of time-varying scheduling parameters. For instance, the approximation can be described as

$$A(\mathbf{x}(t)) \approx A_0 + \sum_{i=1}^{r} \rho_i(t) A_i,$$

where $\mathbf{x}(t)$ is a state at time t, $A(\cdot)$ is a state dependent matrix, $\rho_1(t), \rho_2(t), \ldots, \rho_r(t)$ are scheduling parameters at time t, and $A_0, A_1, \ldots, A_r$ are constant matrices. This affine approximation is typically constructed by identifying these constant matrices through local linearization or empirical interpolation of the nonlinear function $A(x(t))$, and representing the original matrix as a linear combination of these bases weighted by time-varying scheduling parameters. The scheduling parameters are introduced to parameterize the most significant nonlinear dependencies in the system, thereby enabling an efficient and structured approximation of the original dynamics. In practical applications, the scheduling parameters are often extracted from low-dimensional coordinates obtained via projection (e.g., POD coefficients) or learned representations (e.g., encoder outputs in AEs). Their design directly influences the quality of the LPV approximation, and care must be taken to ensure that they adequately capture the nonlinear behaviors of the system.

By embedding the nonlinearity into parameter dependence, the original system can be approximated by an affine or linear time-varying system, where the system matrices vary continuously concerning the scheduling parameters. This representation enables the use of linear system theory for analysis, control synthesis, and model reduction, while still retaining the ability to capture essential nonlinear effects. As a result, LPV approximation serves as a powerful tool for constructing computationally efficient and theoretically grounded reduced-order models for complex dynamical systems.

This approach has garnered significant interest in domains such as control engineering, aerospace, and fluid dynamics, where real-time performance, reduced computational burden, and model interpretability are of paramount importance. The practical advantages and effectiveness of LPV-based reduced-order modeling have been validated in numerous studies, including those presented in [71, 109, 91, 61, 60].

For instance, to apply the LPV approximation to the dynamical systems in Section 2.4, an encoder $\mu : \mathbb{R}^n \to \mathbb{R}^r$ $(n \gg r)$ is considered such that $\boldsymbol{\rho}(t) = \mu(\mathbf{x}(t))$ at time t. The state $\mathbf{x}(t)$ is then reconstructed by linearly projecting the latent state $\boldsymbol{\rho}(t)$ onto n-dimensional space using a basis $\{\mathbf{d}_i | i = 1, 2, \ldots, r\}$ yielding $\mathbf{x}(t) \approx \tilde{\mathbf{x}}(t)$. Consequently, for the convection terms in Equations (2.16) and (2.20), the LPV approximation of $\mathsf{N}(\mathbf{x}(t))\mathbf{x}(t)$ becomes

$$\mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) \approx \mathsf{N}(\tilde{\mathbf{x}}(t))\mathbf{x}(t) = \mathsf{N}(\sum_{i=1}^{r} \rho_i(t)\mathbf{d}_i)\mathbf{x}(t) = (\sum_{i=1}^{r} \rho_i(t)\mathsf{N}_i)\mathbf{x}(t),$$

where $\boldsymbol{\rho}(t) = [\rho_1(t), \rho_2(t), \ldots, \rho_r(t)]^\top$ and the matrices $\mathsf{N}_i = \mathsf{N}(\mathbf{d}_i)$ are fixed. As a result, the state-dependent coefficient matrix $\mathsf{N}(\cdot)$ can be efficiently computed by handling r-dimensional latent states $\boldsymbol{\rho}(t)$ instead of much larger dimensional states $\mathbf{x}(t)$.

Furthermore, polytopic LPV approximations have a possibility to guarantee the stability of control systems described as

$$\begin{aligned}\dot{\mathbf{x}}(t) &= A(\boldsymbol{\rho}(t))\mathbf{x}(t) + B(\boldsymbol{\rho}(t))\mathbf{u}(t),\\ \mathbf{y}(t) &= C(\boldsymbol{\rho}(t))\mathbf{x}(t) + D(\boldsymbol{\rho}(t))\mathbf{u}(t),\end{aligned}$$

where $\mathbf{x}(t) \in \mathbb{R}^n$, $\mathbf{u}(t) \in \mathbb{R}^m$, and $\mathbf{y}(t) \in \mathbb{R}^k$ are a state, system input, and output at time t respectively, $A(\cdot) \in \mathbb{R}^{n\times n}, B(\cdot) \in \mathbb{R}^{n\times m}, C(\cdot) \in \mathbb{R}^{k\times n}$, and $D(\cdot) \in \mathbb{R}^{k\times m}$ are state-space matrices with respect to $\boldsymbol{\rho}(t) = [\rho_1(t), \rho_2(t), \ldots, \rho_r(t)]^\top$ satisfying $\rho_i(t) \geq 0$ and $\sum_{i=1}^r \rho_i(t) = 1$ for time $t > 0$; see, e.g., [11].

For instance, state-dependent Riccati equations can be used in feedback control design to verify the stability of LPV approximations by solving a family of parameter-dependent Riccati equations; see, e.g., [5, 60].

Despite their advantages, ROMs face several challenges that limit their applicability and robustness, particularly in systems characterized by strong nonlinearities or when the system states deviate significantly from the training regime.

Nonlinear dynamical systems often exhibit intricate interactions and behaviors that are not easily captured through linear subspace projections. To overcome these limitations, nonlinear embedding techniques have been introduced to learn more expressive low-dimensional representations. However, these methods cause new challenges related to model training, stability, and interpretability.

Another major challenge lies in the generalization capabilities of data-driven ROMs. While such models can achieve high accuracy within the training regime, they often struggle to extrapolate reliably to unseen parameter regimes or extended temporal horizons. Ensuring robustness under uncertainty remains a critical and largely unresolved issue in the development of trustworthy ROM frameworks.

CHAPTER 3

POD AND CONVOLUTIONAL AUTOENCODERS

Contents

3.1 Introduction . . . 37
3.2 Low-dimensional Parametrization . . . 39
3.2.1 Proper Orthogonal Decomposition (POD) . . . 40
3.2.2 Convolutional Neural Network (CNN) . . . 40
3.2.3 Convolutional AE (CAE) . . . 42
3.2.4 POD-CAE . . . 43
3.2.5 Clustered POD (cPOD) . . . 44
3.2.6 Individual CAE (iCAE) . . . 45
3.3 Numerical Experiments . . . 46
3.3.1 Single-cylinder . . . 47
3.3.2 Double-cylinder . . . 52
3.3.3 Burgers' Flows . . . 55
3.4 Conclusion . . . 57

This chapter introduces CAEs with clustering for low-dimensional parametrization of fluid flows presented in [56, 57].

3.1 Introduction

Accurate FEM discretizations of fluid flow often result in dynamical systems with millions of degrees of freedom, creating significant computational challenges for simulations. These challenges are further compounded by the nonlinearities involved, particularly when designing controllers using these models.

The core idea and promise of model order reduction techniques lie in identifying and utilizing lower-dimensional coordinates that effectively represent or approximate the dynamics of inherently high-dimensional systems. For instance, projection methods, such as the widely used POD, rely on linear projections onto subspaces designed to capture the

system's most relevant dynamics. These methods efficiently reduce the state-space dimension of dynamical systems, resulting in significant savings in memory and computational resources during simulations. However, linear methods are inherently limited in accuracy for a given number of degrees of freedom, as described by the *Kolmogorov n-width*; e.g., [85]. In general, achieving a highly accurate yet very low-dimensional parametrization often necessitates the use of nonlinear approaches; e.g., [55, 74, 91].

This chapter focuses on the development of very low-dimensional state representations using nonlinear encoders and decoders. Such representations serve as a foundation for downstream applications such as feedback control and state estimation, although the modeling of reduced dynamics is not addressed in the present discussion. Parametrization refers to representing the states of a given system in an alternative coordinate system, while approximation implies designing this representation to prioritize the minimal dimensionality of the parametrizing coordinates, often at the expense of some accuracy. The term encoding denotes the process of computing a reduced and parametrized representation of a state, whereas decoding or reconstruction refers to computing the state in its original coordinates from the parametrized representation. The mechanisms or algorithms enabling these processes are referred to as the encoder and decoder respectively.

POD can be interpreted within an AE framework, where both encoding and decoding are performed through linear projection. Extending this idea, AEs (see Section 2.2) allow for nonlinear transformations that are trained directly on data. In particular, CAEs (see Section 2.2.2) leverage the local structure of spatially distributed data making them well-suited for encoding fluid flow fields. By employing convolutional layers with sparse connectivity and weight sharing, CAEs offer compact representations and efficient training, especially when combined with architectural enhancements such as structural reparameterization.

An AE typically refers to a neural network designed to efficiently encode data and trained solely on the data itself. The underlying network architecture can be tailored to the problem and may include multiple hidden layers. Nonlinearity in encoding and decoding often arises from the use of activation functions. Building on these principles, various types of AEs have been developed, including sparse AEs, denoising AEs, and contractive AEs; e.g., [13]. The advancement of deep-layer architectures has enabled modern neural networks to achieve remarkable success in computer vision; e.g., [98, 54, 108].

However, in industrial applications, hardware constraints and the need for fast inference often restrict the use of large networks. To address this limitation while preserving the performance of deep architectures, one potential solution is to employ structural re-parameterization techniques for model size reduction, as discussed in [34, 76]. For example, the composition of multiple convolution operations can be redefined as a single operation, and parallel convolution operations can be consolidated into a single equivalent operation. With the increasing availability of computational resources and neural network design and training toolboxes, AEs have been explored as alternatives in model order reduction for well over a decade; e.g, [71, 29, 74, 75].

In the context of PDE-based models, variants of AEs leveraging CNNs [73] appear particularly effective, as evidenced by several recent studies employing CNNs to solve PDEs; e.g., [44, 68, 112].

The realization of the linear propagation between layers as a convolution operation significantly reduces the network's parameters by enforcing sparse connectivity. This means that each variable in a layer is connected to only a limited subset of variables in the adjacent layers. Additionally, the operation of pooling, which aggregates several neighboring variables into one, reduces the dimensionality of variables in each layer. In FEM approaches, sparse connectivity is a key performance criterion, typically achieved by employing basis functions with local support. Similarly, the concept of pooling can be related to multiscale methods.

Once a very low-dimensional encoding of a state is obtained, the decoder's role is to reconstruct the high-dimensional state from this code. Intuitively, the complexity of the decoder design increases as the variations in the output space that it needs to reproduce become more significant. Clustering in the reduced-order coordinates offers an effective strategy for partitioning the state space. This enables individual decoders to specialize in reconstructing specific subsets of the overall variation, thereby simplifying both their design and implementation.

Clustering algorithms are used to identify patterns in data and divide a given dataset into subsets without the need for labels, i.e., without prior knowledge of the characteristics of each subset. Furthermore, clustering algorithms integrate well with neural networks, especially as clustering becomes easier in lower-dimensional data, such as in low-dimensional feature spaces; see Section 2.3.

While the use of local bases is well-established in transport-dominated problems (see [29] for an example application), clustering has only recently been explored in the context of dynamical systems. This delay may be attributed to the challenges of identifying clusters in high-dimensional spaces and the nonsmooth nature of clustering.

Nevertheless, it has been observed that dimensionality reduction techniques combined with clustering can enhance retrieval performance and reduce computational costs by processing subsets of large-scale datasets individually, as seen in applications like text retrieval systems; e.g., [45].

In this chapter, the focus lies on identifying very low-dimensional parametrizations of states rather than constructing reduced dynamical models. As a result, the nonsmooth nature of clustering does not pose a significant concern, and clustering can be applied directly within the numerical experiments.

3.2 Low-dimensional Parametrization

This section investigates several potentially nonlinear methods for encoding and decoding, with particular emphasis on achieving very low-dimensional parametrizations. The standard POD method is employed as a benchmark for comparison.

As described in Section 2.4, the datasets X_{single}, X_{double}, X_{bg_1}, and X_{bg_2} are considered. Each dataset consists of states containing velocity components in the x- and y-directions, represented in $\mathbb{R}^n$ for each experiment.

For the purpose of low-dimensional parametrization using r-dimensional latent and reconstructed states, the following notation is adopted:

- $\mathbf{x}(t) \in \mathbb{R}^n$: a state at time t
- $\boldsymbol{\rho}(t) \in \mathbb{R}^r$: a latent state at time t
- $\tilde{\mathbf{x}}(t) \in \mathbb{R}^n$: a reconstructed state at time t (i.e., $\mathbf{x}(t) \approx \tilde{\mathbf{x}}(t)$)

These notations will be consistently used throughout the discussion.

3.2.1 Proper Orthogonal Decomposition (POD)

The method of POD can be interpreted as an AE with the following components:

(1) a *linear* encoder defined as

$$\boldsymbol{\rho}(t) = V^\top \mathbf{x}(t)$$

(2) and a *linear* decoder

$$\tilde{\mathbf{x}}(t) = V\boldsymbol{\rho}(t),$$

where $V \in \mathbb{R}^{n\times r}$ is a POD basis obtained by the method described in Section 2.1.1.

3.2.2 Convolutional Neural Network (CNN)

To apply standard convolutional approaches, the input data must be available on a tensorized grid, meaning a grid of rectangles with no hanging nodes, aligned with the coordinate axes; for an illustration of this requirement, see [56, Fig. 3 and 4]. However, FEM simulations typically compute numerical solutions on unstructured mesh structures which are often composed of elements such as triangles in two dimensions or tetrahedra in three dimensions, rather than on grid-aligned domains. These meshes are specifically designed to conform to complex geometries and boundary conditions, making them fundamentally different from the structured grids required for standard convolutional operations. To achieve this, a suitable (sparse) interpolation matrix I_c is employed so that the CNN input $\mathbf{x}_{\mathsf{CNN}}(t)$ can be generated by the linear transformation

$$\mathbf{x}_{\mathsf{CNN}}(t) = I_c\mathbf{x}(t), \tag{3.1}$$

where the interpolation matrix I_c is defined as described in Remark 3.1.

The CNN architecture introduced in [55] is considered, which utilizes POD modes for the state reconstruction, described as

(1) a *nonlinear convolutional* encoder

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

(2) and a *linear* decoder

$$\tilde{\mathbf{x}} = \varphi(\boldsymbol{\rho}(t)) = V(W\boldsymbol{\rho}(t)),$$

where $\mathbf{x}_{\mathsf{CNN}}(t) \in \mathbb{R}^{2\times w\times h}$ is a CNN input (i.e., x- and y-direction velocity snapshots on the $w \times h$ pixel grid), $V \in \mathbb{R}^{n\times q}$ is a POD basis with q modes, and $W \in \mathbb{R}^{q\times r}$ is a trainable matrix.

Remark 3.1 (Sparse interpolation matrix I_c):
In FEM simulations, a state vector $\mathbf{x}(t) \in \mathbb{R}^n$ is associated with a function $\tilde{v}(t)$ of a spatial coordinate $x \in \Omega \subset \mathbb{R}^2$ by the relationship

$$\tilde{v}(t; x) = \sum_{i=1}^{n_v} v_i(t)\xi_i(x),$$

where the ξ_i are the FEM basis functions and where the $v_i(t)$ are the elements of $\mathbf{x}(t)$, and Ω is the considered spatial domain of the simulation.

Given a rectangular grid of pixels $\Omega_r := \{x_{ij} | i = 1, \ldots, r_1,\, j = 1, \ldots, r_2\} \subset \Omega$, the interpolation of $\mathbf{x}(t)$ on Ω_r is readily computed through

$$\tilde{v}(t; x_{ij}) = \sum_{\ell=1}^{n} v_\ell(t)\xi_\ell(x_{ij}) = \begin{bmatrix}\xi_1(x_{ij}) & \xi_2(x_{ij}) & \ldots & \xi_n(x_{ij})\end{bmatrix} \begin{bmatrix} v_1(t) \\ v_2(t) \\ \vdots \\ v_n(t)\end{bmatrix}. \tag{3.2}$$

If the interpolant $\tilde{v}\big|_{\Omega_r}(t)$ is vectorized, a vector $\tilde{\mathbf{x}}(t)\big|_{\Omega_r} \in \mathbb{R}^{r_1\cdot r_2}$ is obtained through

$$\tilde{\mathbf{x}}(t)\big|_{\Omega_r} = I_{\Omega_r}\mathbf{x}(t)$$

with the interpolation matrix

$$I_{\Omega_r} = \begin{bmatrix} \xi_1(x_{1\,1}) & \xi_2(x_{1\,1}) & \ldots & \xi_n(x_{1\,1}) \\ \xi_1(x_{1\,2}) & \xi_2(x_{1\,2}) & \ldots & \xi_n(x_{1\,2}) \\ \vdots & \vdots & & \vdots \\ \xi_1(x_{1\,r_2}) & \xi_2(x_{1\,r_2}) & \ldots & \xi_n(x_{1\,r_2}) \\ \vdots & \vdots & & \vdots \\ \xi_1(x_{r_1\,1}) & \xi_2(x_{r_1\,1}) & \ldots & \xi_n(x_{r_1\,1}) \\ \xi_1(x_{r_1\,2}) & \xi_2(x_{r_1\,2}) & \ldots & \xi_n(x_{r_1\,2}) \\ \vdots & \vdots & & \vdots \\ \xi_1(x_{r_1\,r_2}) & \xi_2(x_{r_1\,r_2}) & \ldots & \xi_n(x_{r_1\,r_2}) \end{bmatrix}. \tag{3.3}$$

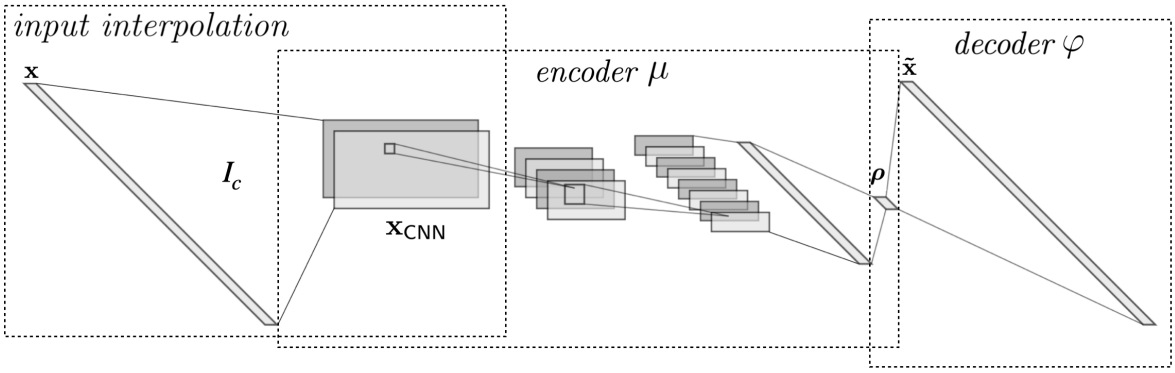


Figure 3.1: A CAE architecture consisting of a nonlinear convolutional encoder and a reparameterized affine linear decoder by Remark 3.2

In our case, since x- and y-direction velocity values in each point x_{ij} are considered, the interpolation matrix I_c is defined as a $2 \cdot r_1 \cdot r_2 \times n$ matrix, and the resulting interpolated values are reshaped into a $2 \times r_1 \times r_2$ tensor which serves as the CNN input.

Additionally, the basis functions have local support (meaning that $\xi_i(x) = 0$ except in a small portion of the domain) so that I_c is a sparse matrix. In fact, the sparsity of I_c is more than 99% leading to efficient computation for dimensionality reduction. ◊

For training the model, Algorithm 2.5 is utilized in conjunction with the ADAM optimizer [69] to optimize the model parameters efficiently. It is important to note that the parameters in I_c are fixed as stated in Remark 3.1. Hence, they do not require any updates.

3.2.3 Convolutional AE (CAE)

Instead of relying solely on POD modes for reconstruction, one can employ transposed convolutional layers without nonlinear activation, combined with an trainable matrix I_p that maps values from the tensorized grid back to the FEM grid. This approach enables exploration of the reconstruction capabilities of CNNs while maintaining the process as affine-linear, as it is composed of affine-linear operators. The resulting CAE, therefore, comprises

(1) a *nonlinear convolutional* encoder

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

and

(2) an *affine linear deconvolutional* decoder

$$\tilde{\mathbf{x}} = \varphi(\boldsymbol{\rho}(t)) = D\boldsymbol{\rho}(t) + \mathbf{b}$$

where $\mathbf{D} \in \mathbb{R}^{n \times r}$ is a matrix, and $\mathbf{b} \in \mathbb{R}^n$ is a vector.

Remark 3.2 (Structural reparameterization of (de-)convolutional layers): The standard (or transposed) convolution can be represented as an affine-linear transformation $\mathbf{x}_O = T\mathbf{x}_I + \mathbf{c}$ where $\mathbf{c}$ is a bias vector, T is a matrix, and where $\mathbf{x}_O$ and $\mathbf{x}_I$ are vectors as vectorized CNN feature maps respectively. In other words, in the absence of any nonlinear activation functions, the entire convolutional process is simply a composition of affine-linear transformations.

Let $L_k(\boldsymbol{\rho}) = T_k\boldsymbol{\rho} + \mathbf{c}_k$, $k = 1, 2, \ldots, m$ be transposed convolutions where k refers to k-th deconvolutional layer and $\boldsymbol{\rho}$ is a latent state. Thus, a deep decoder network φ is defined as $\varphi(\boldsymbol{\rho}) = I_p((L_n \circ L_{n-1} \circ \cdots \circ L_2 \circ L_1)(\boldsymbol{\rho}))$. Then

$$\begin{aligned}
(L_2 \circ L_1)(\boldsymbol{\rho}) &= T_2T_1\boldsymbol{\rho} + T_2\mathbf{c}_1 + \mathbf{c}_2, \\
(L_3 \circ L_2 \circ L_1)(\boldsymbol{\rho}) &= T_3T_2T_1\boldsymbol{\rho} + T_3T_2\mathbf{c}_1 + T_3\mathbf{c}_2 + \mathbf{c}_3, \\
&\vdots \\
(L_n \circ \cdots \circ L_1)(\boldsymbol{\rho}) &= T_n \cdots T_1\boldsymbol{\rho} + T_n \cdots T_2\mathbf{c}_1 + T_n \cdots T_3\mathbf{c}_2 + \cdots + T_n\mathbf{c}_{n-1} + \mathbf{c}_n, \\
&= T\boldsymbol{\rho} + \mathbf{c},
\end{aligned}$$

where $T = T_n \cdots T_1$ and $\mathbf{c} = T_n \cdots T_2\mathbf{c}_1 + T_n \cdots T_3\mathbf{c}_2 + \cdots + T_n\mathbf{c}_{n-1} + \mathbf{c}_n$.

Since the composition of affine transformations is an affine transformation, the decoder φ is structurally re-parameterized as

$$\tilde{\mathbf{x}} = \varphi(\boldsymbol{\rho}(t)) = D\boldsymbol{\rho}(t) + \mathbf{b},$$

where $D = I_pT$ and $\mathbf{b} = I_p\mathbf{c}$ for $t > 0$. ◊

The CAE model can be trained, while maintaining its deep linear deconvolutional architecture, by the same training procedure as that for the CNN model.

The decoder model can be defined as a deep neural network. However, if nonlinear activation functions are excluded, the decoder can be structurally re-parameterized post-training as a single affine-linear layer $D\boldsymbol{\rho} + \mathbf{b}$ as shown in Section 3.2.3. This results in a model size comparable to that of the POD basis during inference, while still providing deep flow features useful for analytical purposes (e.g., [115]). Similarly, nonlinear decoders can also be structurally reparameterized by leveraging the linearity of the convolution operation; e.g., [34].

3.2.4 POD-CAE

The basic concept of the *POD-DL* model introduced in [43] is to utilize a relatively extensive POD basis for an initial reduction followed by the application of a CAE for further reduction within the POD coordinates. To examine the impact of the interpolation matrix I_c and the POD basis on reconstruction performance, a POD-CAE is designed as a network architecture comprising

Algorithm 3.1: cPOD

Input: Dataset $X = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, number of clusters k
Output: a POD basis V for X and k decoder matrices, $W_1, W_2, \ldots, W_k$, for each cluster

1 Obtain a POD basis for X
2 Divide X into k clusters, $X_1, X_2, \ldots X_k$, using k-means clustering in a latent space
3 Generate k POD bases, $V_1, V_2, \ldots, V_k$, for each cluster
4 Obtain k decoder matrices $W_i = V_i(V_i^\top V)$, $i = 1, 2, \ldots, k$
5 **return** $V, W_1, W_2, \ldots, W_k$

(1) a *nonlinear convolutional* encoder

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

and

(2) an *affine linear deconvolutional* decoder

$$\tilde{\mathbf{x}}(t) = \phi(\boldsymbol{\rho}(t)) = D\boldsymbol{\rho}(t) + \mathbf{b},$$

where $\mathbf{D} \in \mathbb{R}^{n\times r}$ is a matrix, and $\mathbf{b} \in \mathbb{R}^n$ is a vector.

In contrast to the CAE in Section 3.2.3, this model employs a POD basis $V^\top \in \mathbb{R}^{q\times n}$ with a sufficiently large q ($q > r$) in the first encoding layer instead of the interpolation matrix I_c. (i.e., $\mathbf{x}_{\mathsf{CNN}}(t) = V^\top \mathbf{x}(t)$). However, the decoder architecture is identical to that in Section 3.2.3 as well as the training strategy follows the same approach as that used for the CAE model.

3.2.5 Clustered POD (cPOD)

As highlighted in the introduction, reconstruction performance can be significantly enhanced by utilizing multiple decoders tailored to specific clusters. Before proceeding with clustering using general AEs, a *clustered POD (cPOD)* is defined as a model consisting of

(1) a *linear* encoder

$$\boldsymbol{\rho}(t) = V^\top \mathbf{x}(t) \text{ and}$$

(2a) a cluster selection algorithm

$$c\colon \boldsymbol{\rho}(t) \mapsto l \in \{1, 2, \ldots, k\}, \tag{3.4}$$

(2b) and k *linear* decoders

$$\tilde{\mathbf{x}}(t) = W_l \boldsymbol{\rho}(t),$$

Algorithm 3.2: Training an iCAE model

Input: Training dataset $X = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, learning rate η, number of epochs e, number of clusters k

Output: Trained decoding parameters for k decoders; $\theta_{\varphi_1}, \theta_{\varphi_2}, \ldots, \theta_{\varphi_k}$

```
Initialize model parameters θ_{φ1}, θ_{φ2}, ..., θ_{φk}
Set a pretrained encoder μ (immutable encoding parameters)
Divide X into k subsets, X_1, X_2, ..., X_k, using k-means clustering in a latent space
for i = 1 to k do
    for epoch = 1 to e do
        for B ⊂ X_i (B: random mini-batch) do
            Compute Z = μ(B)                    // Encoding B to latent variables Z
            Compute X̃ = φ_i(Z; θ_{φi})          // Decoding Z to reconstruction X̃
            Compute L = 1/#B Σ_{x^(j)∈B, x̃^(j)∈X̃} ||x^(j) − x̃^(j)||_2^2   // Reconstruction loss
            Update parameters θ_{φi} ← θ_{φi} − η∇_{θ_{φi}} L
return θ_{φ1}, θ_{φ2}, ..., θ_{φk}
```

where $V \in \mathbb{R}^{n\times r}$ is a POD basis obtained from a given dataset and $V_l \in \mathbb{R}^{n\times r}$ is a POD basis associated with the l-th cluster, $l = 1, 2, \ldots, k$ and the number of clusters k such that

$$W_l = V_l(V_l^\top V).$$

To generate V_l, k-means clustering is implemented in a latent space, classify the actual data $\mathbf{x}(t)$ based on the labels assigned to their latent states $\boldsymbol{\rho}(t)$ and then compute POD bases for each cluster separately as shown in Algorithm 3.1.

Thus, a label $l \in \{1, \ldots, k\}$ is assigned to each latent state $\boldsymbol{\rho}(t)$ and a local POD basis is selected for the given label. It is important to note that clustering is a nonlinear and discontinuous process, which results in the decoding also becoming nonlinear and discontinuous.

3.2.6 Individual CAE (iCAE)

An *individual CAE (iCAE)* method is adapted from the model introduced in [56] for application to general CAEs. An iCAE model consists of

(1) a *nonlinear convolutional* encoder

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

and

Table 3.1: Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, dcv: a deconvolutional layer

	POD	CNN	CAE	POD-CAE
#encoding layers	pod	I_c + 3cv + fc	I_c + 3cv + fc	pod + 3cv + fc
#encoding parameters ($r = 8$)	342,112	**39,582**	**41,244**	8,212,784
#decoding layers	pod	fc + pod	fc + 3dcv + fc	fc + 3dcv + fc
#decoding parameters ($r = 8$)	342,112	342,112*	342,112*	342,112*

* the decoder parts can be structurally reparametrized to a linear transformation $\tau : \mathbb{R}^{42764} \to \mathbb{R}^8$.

(2) *k affine linear deconvolutional* decoders

$$\tilde{\mathbf{x}} = \varphi_l(\boldsymbol{\rho}(t)) = D_l\boldsymbol{\rho}(t) + \mathbf{b}_l,$$

where φ_l is the l-th affine linear deconvolutional decoder trained by the l-th cluster, $l = 1, 2, \ldots, k$. The encoder is continuous and differentiable, whereas the decoding process is nonlinear and discontinuous. It is well-established that data compression can enhance clustering accuracy. Thus, the pretrained CAE encoder can be utilized without additional training allowing latent states $\boldsymbol{\rho}(t)$ to be used for k-means clustering.

Consequently, only k decoders corresponding to the k clusters are trained, while the encoder parameters remain fixed, as outlined in Algorithm 3.2. During inference, the encoder extracts a latent state, which is simultaneously paired with a cluster label. Based on the label, the appropriate decoder is selected for state reconstruction.

In both the cPOD and iCAE approaches, the encoding of the states into latent representations is independent of the clustering process. Clustering is involved solely in the decoding phase. Consequently, the states are effectively represented in the chosen low-dimensional space.

3.3 Numerical Experiments

To investigate very low-dimensional parametrization of incompressible flows, small reduced dimensions $r = 2, 3, 5, 8$ are considered. All the data are introduced in Section 2.4.

The evaluation of all experiments is based on the following three metrics:

- the averaged relative error of state reconstruction

$$\frac{1}{T}\sum_{i=1}^{T} \|\mathbf{x}_i - \tilde{\mathbf{x}}_i\|_{\mathsf{M}} / \|\mathbf{x}_i\|_{\mathsf{M}} \tag{3.5}$$

- the averaged relative error of the reconstruction in the convection operator $\mathsf{N}(\mathbf{x})$

$$\frac{1}{T}\sum_{i=1}^{T}\|\mathsf{N}(\mathbf{x}_i)\mathbf{x}_i - \mathsf{N}(\tilde{\mathbf{x}}_i)\mathbf{x}_i\|_{\mathsf{M}^{-1}}/\|\mathsf{N}(\mathbf{x}_i)\mathbf{x}_i\|_{\mathsf{M}^{-1}}, \tag{3.6}$$

- and the averaged relative residual

$$\frac{1}{T}\sum_{i=1}^{T}\|(\mathsf{N}(\tilde{\mathbf{x}}_i)\tilde{\mathbf{x}}_i + \mathsf{A}\tilde{\mathbf{x}}_i) - (\mathsf{N}(\mathbf{x}_i)\mathbf{x}_i + \mathsf{A}\mathbf{x}_i)\|_{\mathsf{M}^{-1}}/\|\mathsf{N}(\mathbf{x}_i)\mathbf{x}_i + \mathsf{A}\mathbf{x}_i\|_{\mathsf{M}^{-1}}, \tag{3.7}$$

where $\|\cdot\|_{\mathsf{M}}$ and $\|\cdot\|_{\mathsf{M}^{-1}}$ are weighted norms (e.g., $\|\mathbf{z}\|_Q = \sqrt{\mathbf{z}^\top Q\mathbf{z}}$), where $\mathbf{x}_i$ is the i-th reference and $\tilde{\mathbf{x}}_i$ is the i-th reconstructed velocity snapshot, $i = 1, 2, \ldots, T$, and where T is the number of snapshots.

Relative errors like these metrics have been commonly used since it allows to compare quantities of different scales. The reconstruction error (3.5) indicates the reconstruction quality in the discretized spatial domain using the mass matrix M, which contains the physical relevance of each degree of freedom in the finite element discretization. In metrics (3.6) and (3.7), the M^{-1}-norm can be used to quantify the influence of the nonlinear force on acceleration. This choice is physically consistent, as it aligns with the structure of Equation (2.18), where the acceleration $\dot{\mathbf{x}}(t)$ is obtained by applying M^{-1} to the right-hand side of Equation (2.18).

3.3.1 Single-cylinder

The values in each velocity state $\mathbf{x}$ are defined at 42,764 nodal points (i.e., $\mathbf{x} \in \mathbb{R}^{42764}$) on an irregular mesh. To input $\mathbf{x}$ into a convolutional encoder μ in CNNs, CAEs, and iCAEs, $\mathbf{x}(t)$ is transformed into $\mathbf{x}_{\mathsf{CNN}}(t) \in \mathbb{R}^{2\times 63\times 47}$, that represents two 63×47 feature maps; x-directional velocity feature map and y-directional velocity feature map on the 63×47 pixel grid at time t. The transformation is performed using Equation (3.1) followed by reshaping a 5922×1 interpolation vector into the three-dimensional tensors (i.e., $5922 = 2 \times 63 \times 47$).

Table 3.1 outlines the model specifications. When $r = 8$, a POD basis $V \in \mathbb{R}^{42764\times 8}$ consists of 342,112 parameters. Despite being deep neural networks, CNN and CAE encoders have fewer parameters than POD due to the high sparsity of I_c. In the POD-CAE, 192 POD modes are applied to the first encoding layer, which results in a larger number of encoding parameters, instead of I_c. The encoder for all models consists of an initial dimensional reduction layer implemented using either a POD mode or an interpolation matrix followed by three convolutional layers and a fully connected layer. In each hidden layer, the nonlinear activation function ELU [27] is used.

As discussed in Section 3.2, decoder architectures are designed without any nonlinear activation functions to preserve the linearity of the decoding process. In POD, latent

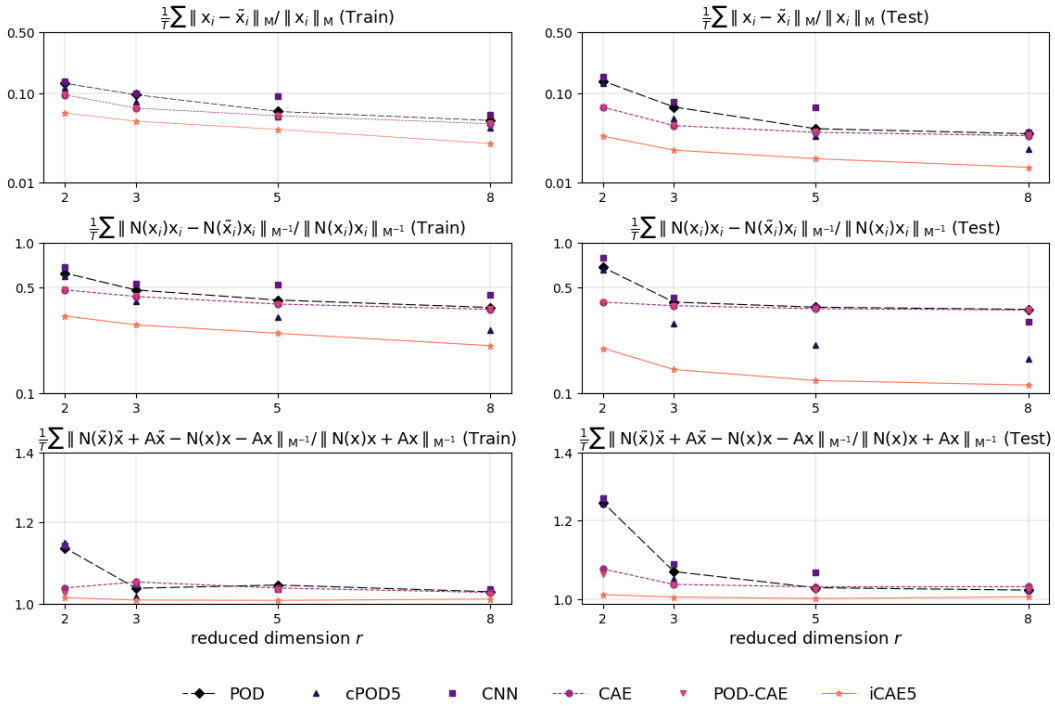


Figure 3.2: Averaged relative errors as measured by the evaluation metrics (3.5), (3.6), and (3.7) for the single-cylinder case

states are directly projected onto the original space using a POD basis. In the CNN model, latent states are transformed into reconstructed states through two consecutive linear transformations: a linear fully connected layer and a 42764×15 POD basis. The CAE and POD-CAE models use the same decoder architecture that includes two fully connected layers and three deconvolutional layers.

According to Remark 3.2, each decoder can be represented as a linear transformation $\tau : \mathbb{R}^8 \to \mathbb{R}^{42764}$ such that the number of decoding parameters is the same as that in POD during inference.

For the clustering-based models, the POD and cPOD models utilize the same encoder architecture while the CAE and iCAE models also share an identical encoder. In the cPOD and iCAE models, each individual decoder is (affine) linear and a decoder is selected from k decoders for state reconstruction corresponding to each latent state. The clustering-based models are denoted as cPOD5 and iCAE5, respectively, when the number of clusters k is set to 5.

As illustrated in Figure 3.2, each model demonstrates the expected dependency of reconstruction errors on the reduced dimension r. Additionally, it is reconfirmed that nonlinear approaches outperform POD at lower values of r. Notably, the POD-CAE achieves nearly identical performance to the CAE during both the training and testing phases. This observation suggests that a sequence of sparsely connected layers (such as I_c

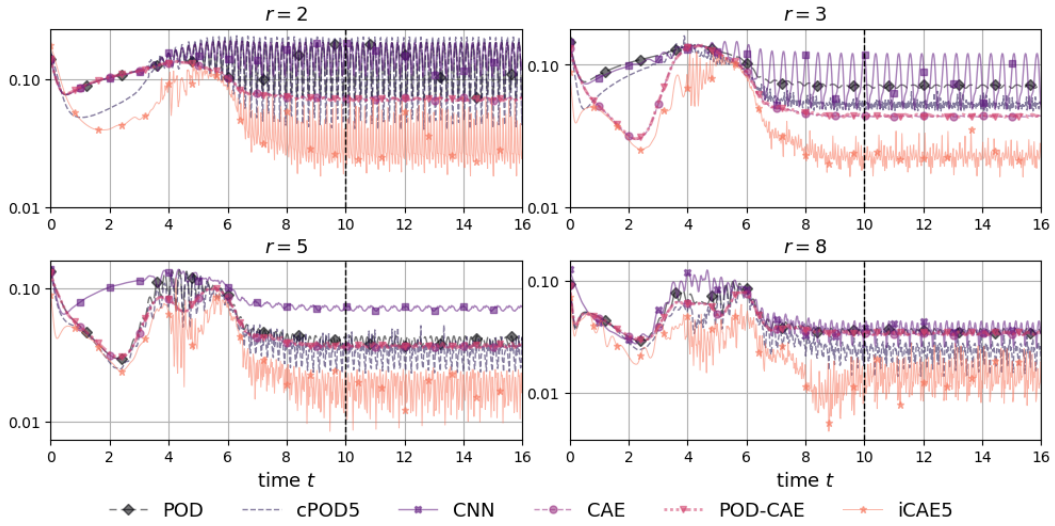


Figure 3.3: Reconstruction error over time for the single-cylinder case at $r = 2, 3, 5, 8$

and convolutional layers) is effective in extracting meaningful features for low-dimensional parametrizations comparable to the CAE integrated with POD. Furthermore, the plots highlight how the clustering enhancements incorporated in cPOD5 and iCAE5 can significantly improve the reconstruction performance compared to the standard POD and CAE models, respectively.

In Figure 3.3, the extrapolation results exhibit performance levels comparable to those observed during the training phase. This is likely attributable to the data's inherent periodicity, particularly pronounced around $t = 7$ which the models effectively capture. Moreover, the plots demonstrate that the clustering enhancements incorporated into cPOD5 and iCAE5 significantly improve the reconstruction performance of the POD and CAE models respectively.

Figure 3.5 reveals that the periodicity inherent in the developed flow phase is clearly discernible in the clusters derived from the low-order coordinates. Furthermore, this figure demonstrates that the initial phase ($0 < t < 4$) is distinctly separated from the developed flow regimes in the cluster analysis, provided a sufficient number of clusters ($k \geq 4$) are employed. At $k = 4$, this separation requires retaining a sufficient amount of information in the latent variables ($r \geq 5$). Interestingly, with $k = 5$, the initial phase is effectively separated with only minor mislabeling observed for $t \in [4, 5)$. Complete separation is achieved with $k = 5$ and $r = 8$.

Figure 3.4 clearly illustrates a trend of improved reconstruction performance with increasing numbers of clusters and latent variables.

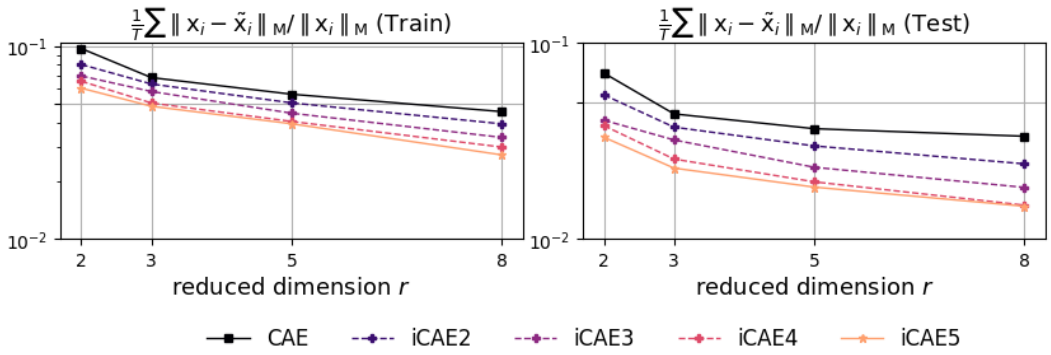


Figure 3.4: Influence of latent space dimension and number of clusters on iCAE reconstruction error for the single-cylinder case

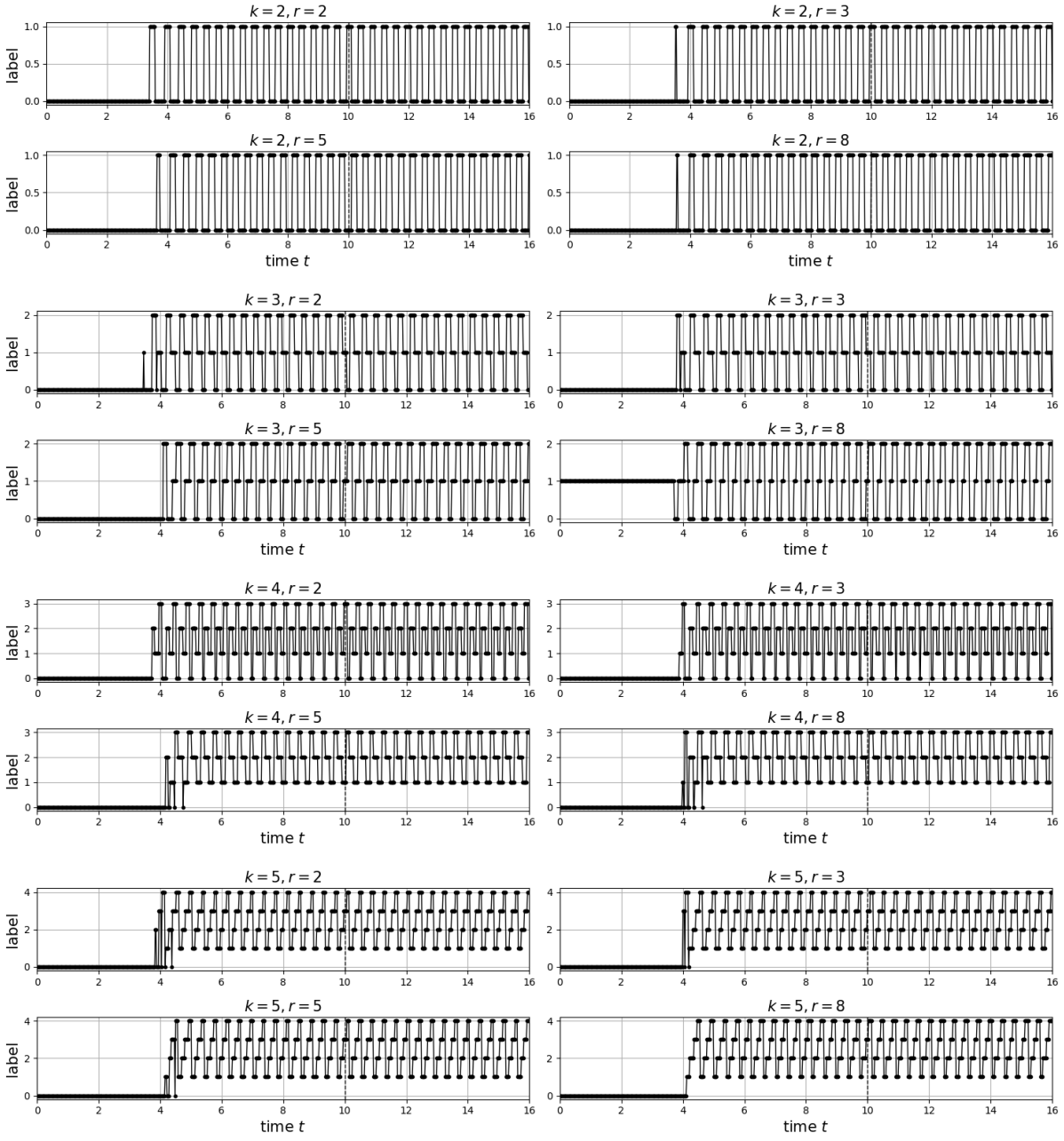


Figure 3.5: Clustering reaction in the time range $[0, 16]$ for the single-cylinder case

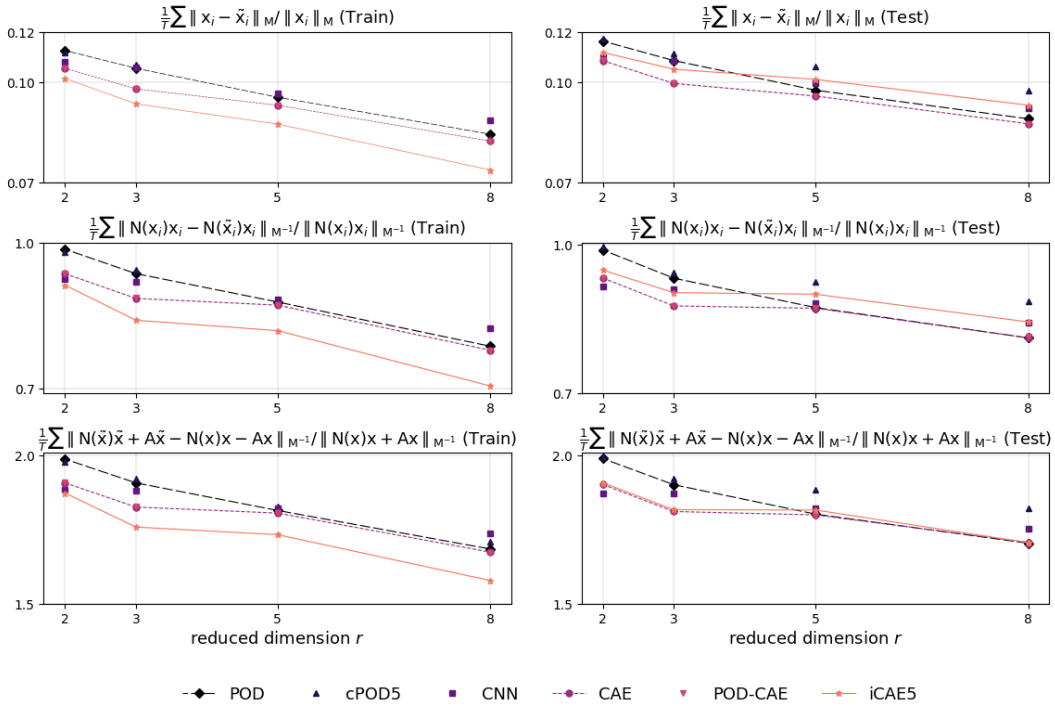


Figure 3.6: Averaged relative errors as measured by the evaluation metrics (3.5), (3.6), and (3.7) for the double-cylinder case

3.3.2 Double-cylinder

As a second example with more complex dynamics, a dataset containing double-cylinder wake flow snapshots at a Reynolds number of 60, with 46,014 nodal points is considered, as presented in Section 2.4.1.

In this case, an interpolation matrix $I_c \in \mathbb{R}^{46014\times 5922}$ is defined as stated in Remark 3.1 to generate the CNN input data $\mathbf{x}_{\mathsf{CNN}}(t) \in \mathbb{R}^{2\times 63\times 47}$. The remainder of the experimental setup is identical to that of the single-cylinder case, with the same models and hyperparameters applied to the double-cylinder case.

In Figure 3.6, the averaged errors support the findings of the previous example, confirming that iCAE5 achieves the best results in the training phase. In contrast, cPOD5 does not reduce the error compared to POD. Both CAE and POD-CAE outperform POD in all cases, with POD-CAE achieving reconstruction errors comparable to those of CAE. When $n_\rho = 2, 3$, convolution-based models, including CNN, outperform POD-based models.

Moreover, the errors of cPOD5 and iCAE5 have noticeably increased compared to POD and CAE during the evaluation phase primarily indicating that the clusters are not well identified in the chaotic flow regime. One possible reason for this shortcoming is the presence of multiple distinct regimes. This complexity is reflected in the clustering results, which do not exhibit clear patterns when compared across different dimensions

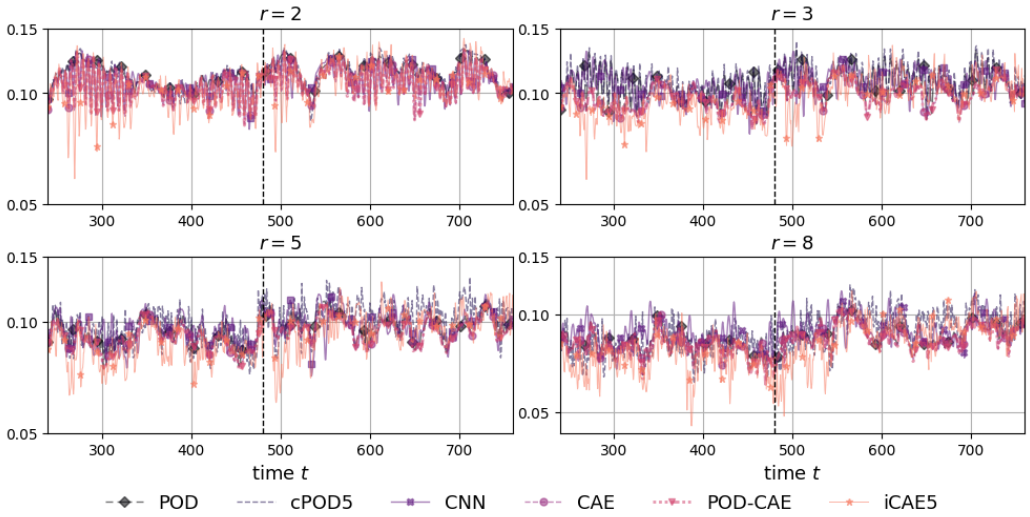


Figure 3.7: Reconstruction error over time for the double-cylinder case at $r = 2, 3, 5, 8$

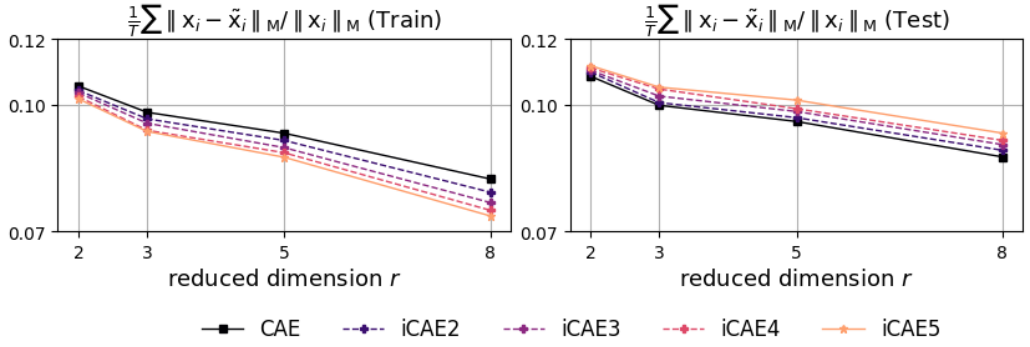


Figure 3.8: Influence of latent space dimension and number of clusters on iCAE reconstruction error for the double-cylinder case

of $\boldsymbol{\rho}$. It is also observed through the clustering results as shown in Figure 3.9.

As illustrated in Figure 3.8, the dependency of the reconstruction error is reversed in the extrapolation regime within the time domain $[480, 760]$, meaning that reconstruction improves as the number of clusters decreases. This further indicates a general failure in classification, which is likely to be mitigated by incorporating more data that covers a broader range of regimes during training. Nevertheless, Figure 3.7 shows that all models maintain the same error levels as observed in the training and evaluation results.

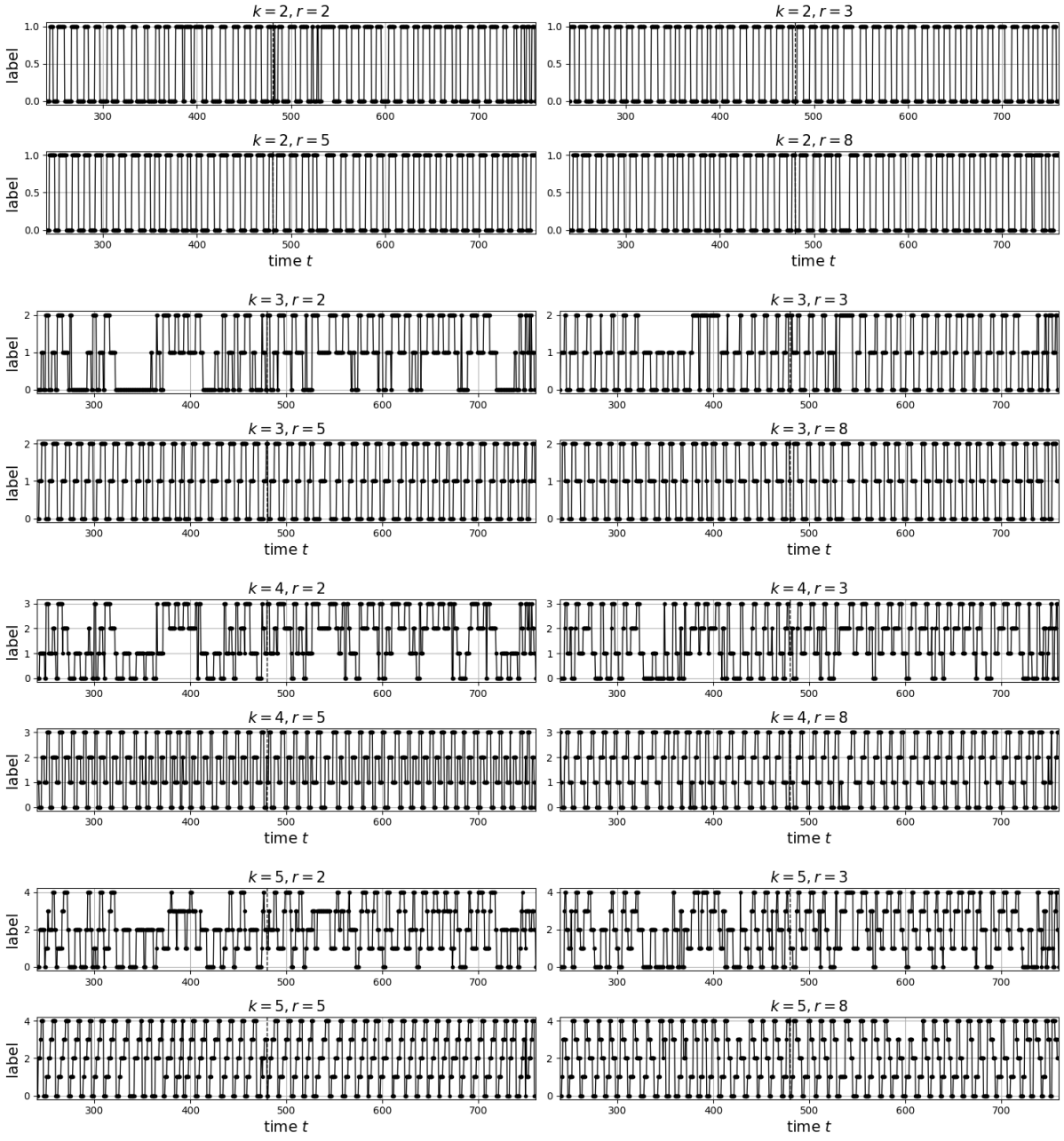


Figure 3.9: Clustering reaction in the time range $[240, 760]$ for the double-cylinder case

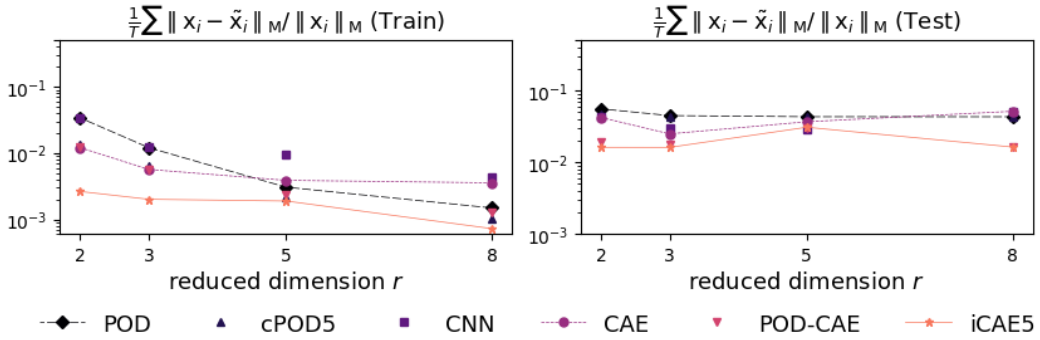


Figure 3.10: Averaged relative errors as measured by the evaluation metric (3.5) for the Burgers' flows

3.3.3 Burgers' Flows

As mentioned in Section 2.4.2, two different datasets of flows evolved from distinct initial conditions, (2.21) and (2.22) are used. In this simulation setup, reconstruction performance is evaluated under conditions different from the cylinder cases to assess robustness to noisy or perturbed inputs.

To serve as input to the CNNs, each state vector is interpolated onto a $2 \times 60 \times 60$ image which contains the $x-$ and $y-$directional velocity values on the tensorized grid.

Figure 3.10 and In Figure 3.11 show that the error level in the reconstruction is about two orders of magnitude smaller than the errors of Navier-Stokes flows shown in Figure 3.2 and Figure 3.6. This low error level, combined with the dependence on data and the stochastic nature of algorithms that may locally converge, may explain why neural network-based approaches do not show a consistent improvement as r increases.

The POD-CAE utilizes 98 POD modes instead of I_c and outperforms the POD and CAE models. Based on the reconstruction performance, iCAE5 encoders are defined as the pretrained POD-CAE encoders. As a result, the iCAE5 improves the reconstruction performance of the POD-CAE during training. When extrapolating to a different initial condition (2.22), the iCAE5 maintains error levels comparable to those of the POD-CAE. Similarly, while iPOD5 enhances the reconstruction performance of the POD model during training, it achieves a comparable error level to POD.

Regarding the clustering shown in Figure 3.12, the clusters generally appear to accurately capture the transition through the states. This suggests that the clusters in the latent variables, where the clustering occurs, effectively represent clusters in the physical space.

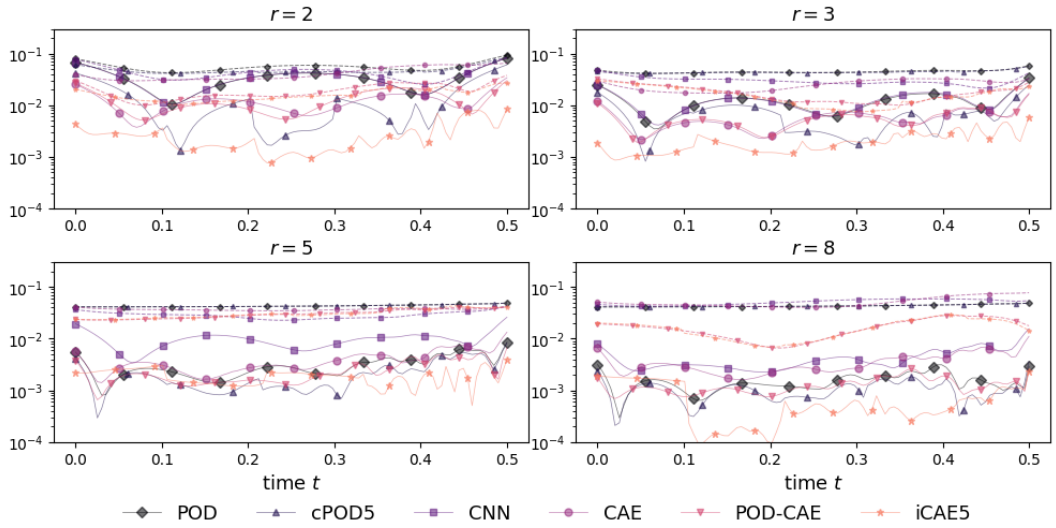


Figure 3.11: Reconstruction error over time for the Burgers' flows at $r = 2, 3, 5, 8$: the solid lines represent reconstruction errors from the simulation using the initial condition (2.21), while the dashed lines indicate test reconstruction errors from the simulation with a different initial condition (2.22)

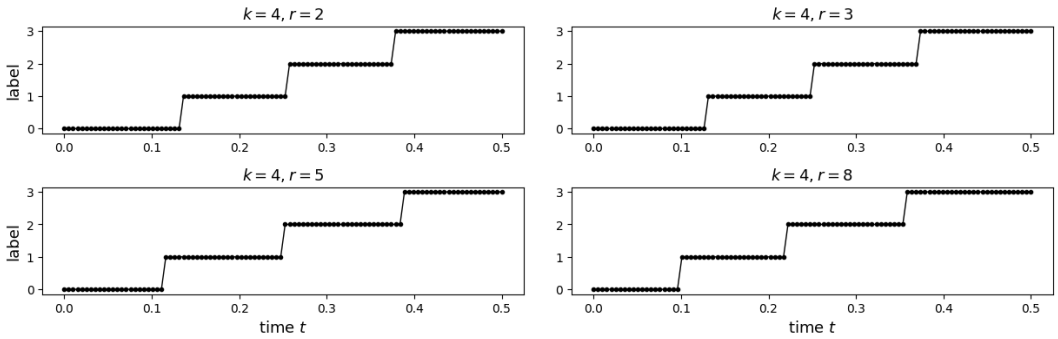


Figure 3.12: Clustering reaction in the time range $[0, 0.5]$ for the Burgers' flows

3.4 Conclusion

This work explored various combinations of POD, CAEs, and clustering techniques for constructing very low-dimensional parametrizations of fluid flows. Consistent with existing theoretical and numerical studies, the results demonstrated that nonlinear approaches can outperform linear methods such as POD, particularly in regimes requiring very low-dimensional representations. Furthermore, neural networks were shown to provide a flexible and effective framework for learning compact state representations, even in high-dimensional settings, especially when exploiting the sparsity structures naturally encoded by convolutional architectures.

To examine the effectiveness of local bases in improving reconstruction performance, it has been observed that models incorporating k-means clustering on the reduced coordinates enable reconstructions tailored to specific flow regimes. While this approach, particularly the iCAEs, achieves significantly lower error levels, especially at low dimensions, its practical application remains uncertain due to the nonsmooth operations involved in decoding. Moreover, clustering methods demonstrated certain limitations when applied to the extrapolation of chaotic flows.

CHAPTER 4
POLYTOPIC AUTOENCODERS

Contents

4.1 Introduction . . . 59
4.2 Motivation . . . 60
4.3 Polytopic AE (PAE) . . . 63
4.3.1 Input Converter . . . 64
4.3.2 Encoder . . . 64
4.3.3 Differentiable Clustering Network . . . 66
4.3.4 Decoder . . . 67
4.3.5 Training Details for PAEs . . . 69
4.3.6 Polytope Error and Polytopic LPV Representation . . . 71
4.3.7 Application in Polytopic LPV Approximations . . . 75
4.4 Simulation Results . . . 76
4.4.1 Single-cylinder . . . 77
4.4.2 Double-cylinder . . . 85
4.5 Conclusion . . . 93

This chapter presents polytopic AEs with clustering for low-dimensional parametrization of fluid flows as discussed in [59, 58].

4.1 Introduction

Linear model reduction methods enable efficient simulation, optimization, and control design due to their linearly optimal projection properties (see, e.g., [61]). Among such methods, POD has been widely adopted owing to its well-defined linear projection structure:

$$\boldsymbol{\rho} = \mathbf{V}\mathbf{x},$$
$$\tilde{\mathbf{x}} = \mathbf{V}^\top \boldsymbol{\rho},$$

where $\mathbf{x} \in \mathbb{R}^n$ is the high-dimensional state, $\tilde{\mathbf{x}} \in \mathbb{R}^n$ is its reconstruction, $\boldsymbol{\rho} \in \mathbb{R}^r$ is the reduced latent variable, and $\mathbf{V} \in \mathbb{R}^{r \times n}$ is a projection matrix composed of the first r POD modes.

Although the latent variables and reconstructed states are typically bounded in practice, linear reduced-order models such as POD parametrize them in unbounded linear spaces. This discrepancy motivates the exploration of alternative representations in bounded sets, particularly within convex polytopes. Since any convex combination of a fixed set of vectors lies within the convex hull defined by those vectors, polytopic representations offer a natural and interpretable framework for bounding the reconstructed states.

One approach to realizing such polytopic parametrizations is through AEs, which utilize nonlinear activation functions to capture complex data structures and often outperform linear methods in terms of expressivity. Furthermore, AEs can be combined with various techniques, including clustering algorithms, classical machine learning models, and numerical schemes, making them well-suited for flexible reduced-order modeling.

In the proposed architecture, this polytopic structure is incorporated by defining the reconstruction as a convex combination of support vectors associated with smoothly assigned clusters. By leveraging standard neural network training methods and introducing *pseudo-labels*, it is possible to regulate the number of active support vectors, enabling the construction of low-dimensional local bases.

This convex combination-based decoding naturally induces an affine parameterization within a polytope, which is particularly advantageous for nonlinear controller design within the framework of LPV systems. For instance, the resulting polytopic LPV representations can be exploited for robust control design; see Section 2.5 and e.g., [47, 53, 109, 91].

This chapter presents the motivation and foundational concepts behind the integration of AEs, convex polytopes, smooth clustering, and polytope-based error analysis. The proposed *polytopic* AE (PAE) framework is then introduced, and its performance is evaluated in comparison with POD.

4.2 Motivation

High-dimensional dynamical systems of the form

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)), \quad \text{with } \mathbf{x}(t) \in \mathbb{R}^n, \quad \text{for time } t > 0. \tag{4.2}$$

In the context of AEs, the reduced-order coordinates are referred to as latent states, while in the LPV context, $\boldsymbol{\rho}$ is treated as a parametrization of the states in dynamical systems described in (4.2).

General nonlinear AEs introduced in Section 2.2 have been effectively used to parametrize dynamical systems such as (4.2) on a very low-dimensional manifold; e.g., [74, 87, 22].

However, the low-dimensional approximation of (4.2) using

$$\dot{\mathbf{x}}(t) \approx \dot{\mathbf{x}}(t) = \frac{d\varphi}{d\boldsymbol{\rho}}\dot{\boldsymbol{\rho}}(t) = f(\varphi(\boldsymbol{\rho}(t))), \tag{4.3}$$

aside from the nonlinear reconstruction $\dot{\mathbf{x}}(t) = \varphi(\boldsymbol{\rho}(t))$, necessitates repeated evaluations of the Jacobian $\frac{d\varphi}{d\boldsymbol{\rho}}$ which can be computationally expensive. Furthermore, inferring an unstructured nonlinear mapping from the low-dimensional space to the high-dimensional state space is inherently ill-posed. Due to these limitations, a clear performance advantage of nonlinear model order reduction over linear projections has yet to be conclusively demonstrated.

To alleviate the problems, a possible way is to use local linear bases, which offers a general solution by imposing a structured approach that reduces the computational effort for reconstruction and improves the well-posedness of approximation tasks in designing the decoder φ. Chapter 3 demonstrated that employing individual affine linear decoders for clusters identified beforehand in the latent space enables superior reconstruction while requiring only the relatively inexpensive operation of determining the current value of $\boldsymbol{\rho}$ within the appropriate cluster.

However, using clustering or any other selection algorithm for local bases often comes at the expense of a discontinuous decoding map φ. Therefore, this work aims to achieve the reconstruction performance of local bases while ensuring a smooth selection algorithm (Definition 4.2) making the reconstruction $\tilde{\mathbf{x}}(t)$ differentiable.

Definition 4.1:
Let $\boldsymbol{\rho}$ be a vector and Y be a set of k classes $C = \{1, 2, \cdots, k\}$. For any $j \in C$, the class probability $P(Y = j|\boldsymbol{\rho})$ is defined as the conditional probability that $\boldsymbol{\rho}$ belongs to the j-th cluster. A vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_k]^\top \in \mathbb{R}^k$ is said to be a *smooth label* if $\alpha_j = P(Y = j|\boldsymbol{\rho})$ satisfying these conditions;

$$\alpha_j \geq 0, \sum_{i=1}^{k} \alpha_i = 1. \qquad \lozenge$$

Definition 4.2:
A function $c : \mathbb{R}^p \to \mathbb{R}^k$ is said to be a *smooth clustering model* if every output of c is a smooth label as defined in Definition 4.1 for each input vector in $\mathbb{R}^p$. $\lozenge$

To accomplish this, the decoding process can be formulated based on the Kronecker product $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ where $\boldsymbol{\rho}$ is the latent variable and $\boldsymbol{\alpha} = c(\boldsymbol{\rho})$ is a smoothly varying clustering variable. The Kronecker product can be used to explain smooth clustering in a low-dimensional space, but also can be efficiently computed using a variety of well-optimized numerical libraries. Moreover, this representation is interpretable and provide useful properties grounded in linear algebra. The reconstruction is then obtained as a linear combination using $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ as coefficients; see Figure 4.1 for an illustration.

Using standard neural network training techniques under the constraints

$$\sum_{i=1}^{r} \rho_i = 1 \text{ and } \rho_i \geq 0 \text{ for } \boldsymbol{\rho} = [\rho_1, \rho_2, \cdots, \rho_r]^\top \in \mathbb{R}^r,$$

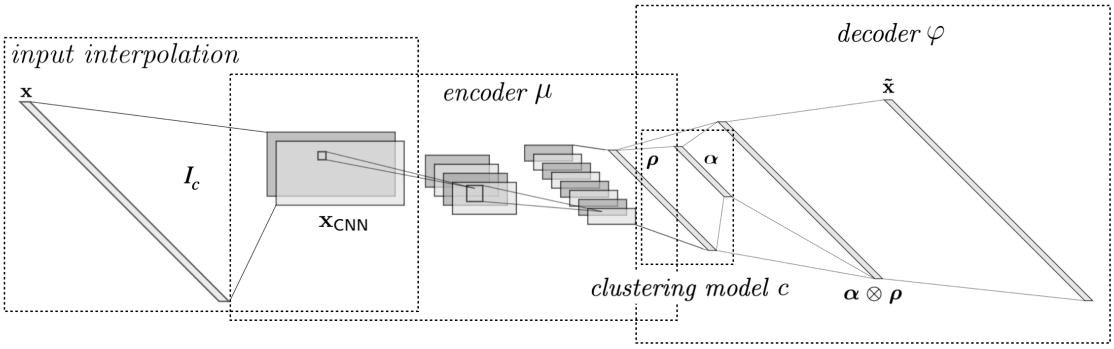


Figure 4.1: A proposed architecture consisting of a nonlinear convolutional encoder, a smooth clustering model, and a reparameterized affine linear decoder

the number of nonzero entries in $\boldsymbol{\alpha}$ can be regulated to ensure that the reconstruction relies on a limited set of local basis vectors. Furthermore, constraints can be imposed such that the entries of $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ remain nonnegative and sum to one. Under these conditions, the reconstruction admits an interpretation as a convex combination within a polytope, as formalized in Lemma 4.3.

Lemma 4.3:
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 nonnegative and sum up to one. ◊

Proof. Since $\rho_i \geq 0$, $\alpha_j \geq 0$, we have that $\rho_i \alpha_j$ is nonnegative for every i and j. Moreover, for the sum over the elements of $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$, we obtain

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

The proposed model provides several key advantages in manifold-based representations. First, the nonlinearity in decoding is confined to the mapping $\boldsymbol{\rho} \to \boldsymbol{\alpha}$ between low-dimensional sets. Second, the reconstruction occurs within a polytope defined by a limited number of vertices. Third, since $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ serves as the coefficient of a convex combination with only a few nonzero entries, the reconstruction remains within a bounded set, enhancing stability by reducing summations.

Moreover, the mapping

$$\boldsymbol{\alpha} \otimes \boldsymbol{\rho} \to \tilde{\mathbf{x}}$$

is linear, offering an approximate polytopic expansion of the state $\mathbf{x}$, which is particularly beneficial for the numerical treatment of LPV systems.

Specifically, the proposed AE (see Figure 4.1) consists of four main components:

1. A linear input converter interpolates spatially distributed data onto a rectangular grid, ensuring compatibility with conventional convolutional layers in a neural network; see Remark 3.1.

2. A nonlinear encoder maps high-dimensional input states to low-dimensional latent variables, with the final layer structured to enforce nonnegativity and a unit-sum constraint, allowing the latent variables to serve as coefficients in a convex combination of supporting vectors.

3. A nonlinear smooth clustering model identifies the position of latent variables within one or more of k clusters.

4. A linear decoder combines the latent variables and clustering results to reconstruct a high-dimensional state within a polytope.

In terms of mathematical models, the mapping from the input $\mathbf{x}$ to its reconstruction $\tilde{\mathbf{x}}$ is expressed as

$$\begin{aligned} \mathbf{x}_{\mathsf{CNN}} &= I_C \mathbf{x}, \\ \boldsymbol{\rho} &= \mu(\mathbf{x}_{\mathsf{CNN}}), \\ \boldsymbol{\alpha} &= c(\boldsymbol{\rho}), \\ \tilde{\mathbf{x}} &= \varphi(\boldsymbol{\rho}) = U(\boldsymbol{\alpha} \otimes \boldsymbol{\rho}), \end{aligned}$$

where $\mathbf{x}$ represents the data, $\mathbf{x}_{\mathsf{CNN}}$ is the preprocessed input for a CNN, I_C is an interpolation matrix, U denotes the matrix containing all supporting vectors, $\boldsymbol{\rho}$ is the latent state, and $\tilde{\mathbf{x}}$ is the reconstruction designed to closely approximate $\mathbf{x}$.

For applications in dynamical equations, it is observed that all mappings involved are differentiable. As a result, a time-differentiable input ensures a differentiable output, given by

$$\frac{d\tilde{\mathbf{x}}}{dt} = \frac{d\varphi}{d\boldsymbol{\rho}}\dot{\boldsymbol{\rho}} = J_D \dot{\boldsymbol{\rho}},$$

where J_D is the Jacobian matrix of the decoder. In the proposed framework, the Jacobian matrix is given by

$$J_D = U[\frac{dc(\boldsymbol{\rho})}{d\boldsymbol{\rho}} \otimes \boldsymbol{\rho} + c(\boldsymbol{\rho}) \otimes \mathsf{I}_r], \tag{4.5}$$

where I_r denotes the $r \times r$ identity matrix. Since U is constant, the computation of J_D only requires updating the small Jacobian matrix of $c(\boldsymbol{\rho})$ with $\boldsymbol{\rho}$ changes.

4.3 Polytopic AE (PAE)

In linear reduced-order models, while reconstructed states and latent variables are generally bounded, they are parametrized within an unbounded linear space. This discrepancy

motivates the consideration of state reconstruction within a polytope. It is well established that any convex combination of a fixed set of vectors lies within a polytope whose vertices are these vectors. Furthermore, states can be parametrized within a polytope instead of an unbounded linear space.

Definition 4.4:
An AE $g \circ f$ consisting of an encoder $f : \mathbb{R}^n \to \mathbb{R}^m$ and a decoder $g : \mathbb{R}^m \to \mathbb{R}^n$, is called a *polytopic AE* if

1. the image of f lies in a normalized polytope, i.e., for $\boldsymbol{\rho} = f(\mathbf{x})$, it holds that $\sum_{i=1}^{m} \rho_i = 1$, $\rho_i \geq 0$ for $\boldsymbol{\rho} = [\rho_1, \rho_2, \cdots, \rho_m]^\top \in \mathbb{R}^m$ and
2. g is linear, i.e., $g(\boldsymbol{\rho}) = \sum_{i=1}^{m} \rho_i \mathbf{v}_i$ for some vectors $\mathbf{v}_i \in \mathbb{R}^n$. ◊

To realize polytopic parametrizations, general *Polytopic Autoencoders* (PAEs) are introduced as defined in Definition 4.4. According to Definition 4.4, the proposed autoencoder architecture, illustrated in Figure 4.1, satisfies the criteria for a PAE.

This section describes the design of a lightweight convolutional encoder architecture that employs depthwise and pointwise convolutions, thereby achieving a substantial reduction in the number of model parameters compared to fully connected layers and POD. In addition, methods are presented for clustering latent states in a low-dimensional space, reconstructing states within a polytope, training PAEs, and quantifying approximation errors within the associated polytopes.

4.3.1 Input Converter

As discussed in Section 2.4, the data are obtained from FEM simulations conducted on potentially nonuniform meshes, which are not directly compatible with CNNs. Instead of applying mesh-specific techniques such as graph neural networks (e.g., [88]), the data are interpolated onto a tensorized grid, as described in Remark 3.1. This interpolation is implemented using a highly sparse matrix I_C, which contains only 0.03% nonzero entries.

4.3.2 Encoder

When handling high-dimensional data, the number of parameters in neural networks becomes a critical concern, not only for training efficiency but also for memory usage and computational cost during forward evaluations. Also, in a standard convolution operation, feature maps are extracted using a $K \times K \times C_O \times C_I$ kernel which leads to high memory consumption where K is the kernel size, C_O is the number of output channels, and C_I is the number of input channels. To alleviate the memory demands, the standard convolution can be decomposed into a *depthwise convolution* and a *pointwise convolution*.

A depthwise convolution applies a $K \times K$ convolution operation to each input channel independently, resulting in a total of $K \times K \times 1 \times C_I$ parameters. This operation captures

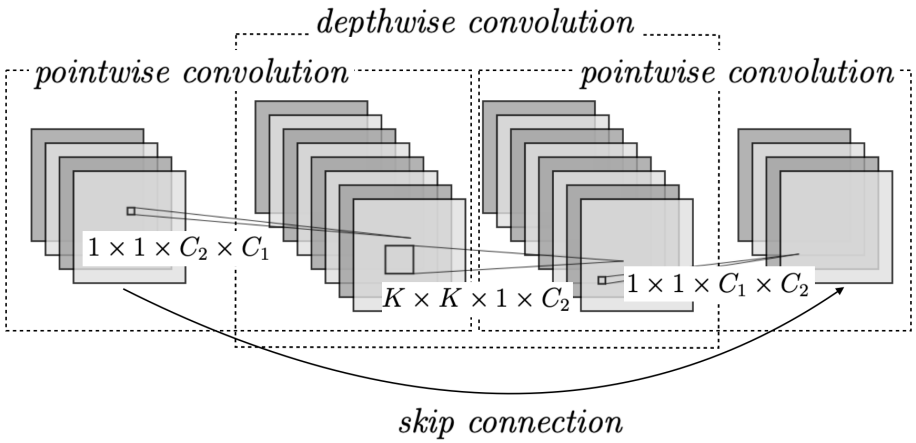


Figure 4.2: Inverted residual block: an efficient approach for constructing deep convolutional layers which require fewer parameters than the standard convolutions

spatial information within each input channel without mixing information across channels. A pointwise convolution, on the other hand, uses a $1 \times 1 \times C_O \times C_I$ kernel to combine the outputs of the depthwise convolution, effectively integrating information across channels to generate the final feature map. As a result, the total number of kernel weights is $K \times K \times C_I + C_O \times C_I$, achieving a weight 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 instance, 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.

To incorporate these specialized convolutions into the encoder, inverted residual blocks are employed. These blocks consist of a depthwise convolution followed by two pointwise convolutions, in contrast to the standard convolutional layers described in Section 3.2.2.

As shown in Figure 4.2, an inverted residual block, as proposed in [93], is designed to produce an output feature map of the same size as the input, enabling the computation of a skip connection between them. Consequently, stacking multiple inverted residual blocks enhances the encoder's receptive field while simultaneously reducing the number of trainable parameters.

Finally, μ is implemented as the deep convolutional encoder

$$\begin{aligned}
h_1 &= a(\mathsf{Conv}_{3\times3}(\mathbf{x}_{\mathsf{CNN}})),\\
h_l &= \mathsf{RB}(h_{l-1}), l = 2, \cdots, L-1,\\
h'_{L-1} &= \mathsf{GAP}(h_{L-1}),\\
\boldsymbol{\rho} &= \sigma(\mathsf{Lin}(h'_{L-1})),
\end{aligned}$$

where $a(\cdot)$, $\mathsf{Conv}_{3\times3}$, RB, GAP, Lin, and σ denote a nonlinear activation function, a 3×3 standard convolution, an inverted residual block, a global average pooling, a linear layer,

and a modified softmax function respectively. RB generates feature maps of the same size as the inputs while excluding bias terms to reduce the number of model parameters, except in layers where downsampling is required. When downsampling is applied, RB omits the residual connection and reduces the spatial dimensions of the 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 smaller than the input (height $\times$ width) while the channel dimension C_{final} is larger than the input channel size.

To ensure that the output vector $\boldsymbol{\rho}(t) = \mu(\mathbf{x}_{\mathsf{CNN}}(t))$ satisfies the constraints

$$\rho_i(t) \geq 0, \quad \text{for } i = 1, \ldots, r \quad \text{and} \quad \sum_{i=1}^{r} \rho_i(t) = 1, \tag{4.7}$$

where the reduced dimension r is significantly smaller than n, The modified softmax function

$$\sigma(\mathbf{x})_i = \frac{x_i \cdot \tanh(cx_i)}{\sum_{j=1}^{r} x_j \cdot \tanh(cx_j)}, \quad c > 0, \tag{4.8}$$

is employed in place of the conventional softmax function

$$\operatorname{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{r} e^{x_j}}. \tag{4.9}$$

While the standard softmax function is widely used in machine learning to produce probability distributions, it is unsuitable for selecting polytope vertices in our application. Specifically, the softmax function never attains exact zero values and requires significantly large input magnitudes to produce outputs close to zero. This limitation motivated the modification from the standard softmax (4.9) to the proposed formulation in 4.8.

Since the signs of x and $\tanh(x)$ always coincide, the differentiable function $x \to x \cdot \tanh(cx)$ produces nonnegative values, ensuring that σ, as defined in 4.8, satisfies the desired property (4.7). This allows the output to serve as coefficients in a convex combination. The parameter c plays a critical role: as $c \to \infty$, the function $x \cdot \tanh(cx)$ approaches $|x|$, indicating that c governs the trade-off between smooth differentiability and approximation of the absolute value function. The absolute value function is often employed as an activation function due to its immunity to the vanishing gradient problem. Based on preliminary experiments, the choice $c = 10$ was found to provide favorable convergence properties during training and yielded improved approximation accuracy.

4.3.3 Differentiable Clustering Network

Most conventional clustering methods, such as k-means clustering [96], rely on discontinuous functions like argmin and argmax which are non-differentiable. To address this issue while preserving the benefits of clustering for reconstruction, a differentiable clustering

network is employed. This network operates in the r-dimensional latent space and is defined as

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

where c is a multi-layer perceptron (MLP) that incorporates the modified softmax function (4.8) in its output layer. This approach ensures differentiability while enabling effective clustering within the latent space.

For reconstruction within a polytope with a limited number of vertices, The design aims to ensure that α contains only a small number of nonzero values. Ideally, a single nonzero entry of 1 would correspond to a distinct cluster selection, while a few nonzero entries would represent a smooth transition between clusters.

Although the modified softmax function theoretically maps to the closed interval $[0, 1]$, in practice, its values are rarely exactly 0 or 1, except in special cases. To enforce decisive cluster selections, pseudo-labeling is introduced during the training of the clustering network c, ensuring that the learned representations maintain clear and interpretable assignments, as described in Section 4.3.5.

4.3.4 Decoder

In Section 3.2.6, iCAEs utilize a single nonlinear encoder

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

while reconstructing states through k individual (affine) linear decoders

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

where $U_l \in \mathbb{R}^{n\times r}$ represents a matrix of local basis vectors, and $\mathbf{b}_l \in \mathbb{R}^n$ is a bias term associated with the reconstruction in the l-th cluster.

Neglecting the bias terms, the reconstructed states $\tilde{\mathbf{x}}$ can be expressed as a discontinuous decoder:

$$\tilde{\mathbf{x}} = \sum_{i=1}^{k} \beta_i U_i \boldsymbol{\rho} = \sum_{j=1}^{r}\sum_{i=1}^{k} \beta_i \rho_j \mathbf{u}_{i;j},$$

where the vector $\boldsymbol{\beta} = [\beta_1, \beta_2, \cdots, \beta_k]$ satisfies the condition

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

which is determined via k-means clustering. Here, $\mathbf{u}_{i;j}$ is the j-th column vector of U_i, and this implies that all elements of β are zero except for a single nonzero entry equal to one, which is called *one-hot vector*. In other words, the decoder selects an individual

decoder based on the assigned cluster, which involves a nonsmooth operation when the states transition from one cluster to another.

To address this issue, the selection vector $\boldsymbol{\beta}$ is replaced with the output $\boldsymbol{\alpha}$ of a smooth clustering network c, which allows multiple nonzero entries and thereby facilitates smooth transitions between clusters:

$$\begin{aligned} \tilde{\mathbf{x}} &= \sum_{j=1}^{r} \sum_{i=1}^{k} \alpha_i \rho_j \mathbf{u}_{i;j} \\ &= U(c(\boldsymbol{\rho}) \otimes \boldsymbol{\rho}), \end{aligned} \tag{4.11}$$

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

$$U = \begin{bmatrix} | & | & & | & & | & & | \\ \mathbf{u}_{1;1} & \mathbf{u}_{1;2} & \cdots & \mathbf{u}_{1;r} & \cdots & \mathbf{u}_{k;1} & \cdots & \mathbf{u}_{k;r} \\ | & | & & | & & | & & | \end{bmatrix} \tag{4.12}$$

is the matrix containing all support vectors used for reconstruction.

Due to the modified softmax function in the last layer of μ and c, the architecture conforms to Lemma 4.3. As a result, the decoder output $\varphi(\boldsymbol{\rho}(t))$ is expressed as a convex combination of the column vectors of U; i.e., all reconstructed states lie within the polytope defined by the vertices $\mathbf{u}_{1;1}, \ldots, \mathbf{u}_{k;r}$.

Remark 4.5 (Kronecker product and convolutions):
By design and according to the definition of the Kronecker product,

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

the reconstruction in (4.11) is based on the multiplication of the coordinates vector $\boldsymbol{\rho}$ with weights α_i referring to the i-th cluster. Instead of this basic scalar weighting, one might alternatively consider (discrete) convolutions:

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

where the convolution kernels g_i are suitably chosen and may depend on $\boldsymbol{\rho}$, similar to the clustering coefficients $\boldsymbol{\alpha}$. In fact, it can be shown that if all kernel coefficients are positive and sum to one, the result of the convolutions satisfies the conditions in (4.7) required for the polytopic reconstruction.

Additionally, due to the multiplicative nature of convolutions, formulas for the Jacobian, similar to (4.5) can be derived. However, for larger kernels g_i the mapping $\boldsymbol{\rho} \to \mathbf{g}$ (and its Jacobian) will become more complex. ◊

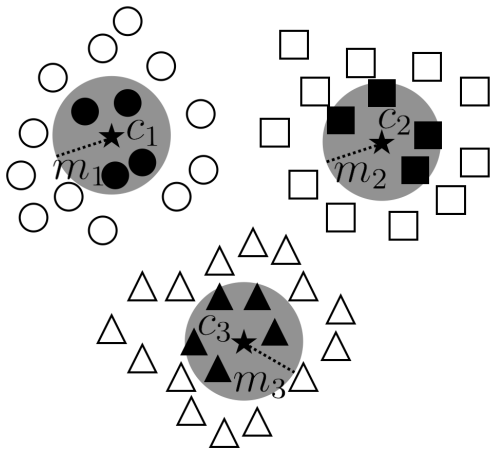


Figure 4.3: When the latent states are classified into three clusters in a low-dimensional space, the clustering labels associated with the latent states within each circle, centered at c_i with radius m_i, $i = 1, 2, 3$, are selected as target labels. Unselected labels are excluded from the training of PAEs.

4.3.5 Training Details for PAEs

The proposed encoder, decoder, and clustering network are fully differentiable with respect to the input variables and all model nodes, allowing for the joint optimization of all model parameters using a gradient-based optimizer; e.g., Adam [69]. However, to enhance training efficiency and because the smooth clustering network is trained using pseudo-labels derived from k-means clustering, the overall training strategy consists of two preparatory steps. In the final step, all components are fine-tuned through joint optimization.

As outlined in Algorithm 4.1, the training process consists of three steps:

Step 1 (initialization and identification of the latent space): A PAE is initially trained consisting of a nonlinear encoder μ and a polytopic decoder $\bar{\varphi}$, without incorporating clustering.

Step 2 (k-means clustering for pseudo-label generation and individual decoder initialization): Latent variable coordinates are extracted from the pretrained encoder using the training dataset. Subsequently, k-means clustering is applied in the latent space to generate pseudo-labels and to initialize individual decoders corresponding to each identified cluster.

To mitigate potential overfitting of the k-means clustering results and to shift the emphasis from precise centroids to their surrounding regions, pseudo-labels are defined to represent local neighborhoods around each cluster center. The clustering network is then trained to approximate these pseudo-labels in a distributional sense.

Specifically, for each cluster i, the data points $j(i)$ are selected such that their latent

Algorithm 4.1: Training a PAE

Input: Hyperparameters; e.g., r, k, and the number of epochs N
Output: Pretrained encoding, decoding, and clustering model parameters; $\theta_\mu, \theta_\varphi, \theta_c$

Note that, e.g., $\boldsymbol{\rho} = \mu(\theta_\mu; \mathbf{x})$ represents the evaluation using the current values of the parameters
Step 1. Train a CAE consisting of an encoder μ and a decoder $\bar{\varphi}$
Implement Algorithm 2.5 to obtain pretrained model parameters θ_μ and $\theta_{\bar{\varphi}}$
Step 2. Train individual decoders; $\varphi_1, \cdots, \varphi_k$
Freeze the weights θ_μ of μ
Select a batch of data based on partial k-means clustering and store their labels $\mathbf{l}$
Implement Algorithm 3.2 to obtain pretrained model parameters $\varphi_1, \cdots, \varphi_k$
Step 3. Train the clustering net c and fine-tuning the PAE
Define a matrix $U = [\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}]$ $(\theta_\varphi = \{\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}\})$
for $e = 1, \cdots, N$ **do**
  for *random batch:* $i \in B$ **do**
    Compute $\boldsymbol{\rho} = \mu(\theta_\mu; \mathbf{x}_{\mathsf{CNN}})$
    Compute $\tilde{\mathbf{x}} = \varphi(\theta_\varphi; \boldsymbol{\rho}) = U(c(\theta_c; \boldsymbol{\rho}) \otimes \boldsymbol{\rho})$
    Compute $\mathcal{L}_{\mathsf{rec}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}) = \frac{1}{\#B} \sum_{i \in B} \| \tilde{\mathbf{x}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}^{(i)}) - \mathbf{x}^{(i)} \|_{\mathsf{M}}$
    Compute $\mathcal{L}_{\mathsf{clt}}(\theta_\mu, \theta_c; \boldsymbol{\rho}) = -\frac{1}{\#P} \sum_{j \in P \subset B} \mathbf{l}^{(j)} \cdot \log(c(\theta_\mu, \theta_c; \boldsymbol{\rho}^{(j)}))$
    Compute $\mathcal{L} = \mathcal{L}_{\mathsf{rec}}(\theta_\mu, \theta_\varphi, \theta_c; \boldsymbol{\rho}) + 10^{-4} \mathcal{L}_{\mathsf{clt}}(\theta_\mu, \theta_c; \boldsymbol{\rho})$
    Optimize model parameters θ_μ, θ_φ, and θ_c
return $\theta_\mu, \theta_\varphi, \theta_c$

coordinates satisfy

$$|\boldsymbol{\rho}^{(j(i))} - \mathbf{c}_i| < m_i, \quad i = 1, 2, \ldots, k, \tag{4.13}$$

where $\mathbf{c}_i$ denotes the centroid of the i-th cluster, and m_i is the mean distance between the latent variables and the centroid $\mathbf{c}_i$ within the same cluster. Each selected data point is then assigned the i-th unit vector as its pseudo-label.

By applying this procedure to all clusters, a set of data-label pairs is constructed, denoted as

$$D_{cl} := \{(\mathbf{x}^{(1)}, \mathbf{l}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{l}^{(2)}), \ldots, (\mathbf{x}^{(N_l)}, \mathbf{l}^{(N_l)})\}, \tag{4.14}$$

where N_l is the total number of data points selected based on criterion (4.13), and the labels $\mathbf{l}^{(i)}$ are unit vectors in $\mathbb{R}^r$ representing the corresponding cluster.

In the next preparatory step, the clustered and labeled dataset D_{cl} is used to train individual polytopic decoders $\varphi_1, \cdots, \varphi_k$ in each cluster. Then, the matrix

$$U = [\theta_{\varphi_1}, \cdots, \theta_{\varphi_k}],$$

which gathers all the supporting vectors of the individual decoders, is used to initialize the weights of a global, smooth, and clustering-based decoder.

Step 3 (training the clustering net and fine-tuning the PAE): A PAE is trained by fine-tuning μ and U while concurrently optimizing the clustering network c. For the model optimization, a reconstruction loss is defined as

$$\mathcal{L}_{\mathsf{rec}} = \frac{1}{\#B} \sum_{i \in B} \| \tilde{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)} \|_{\mathsf{M}},$$

where B refers to the index set of a data batch drawn from the training data. The loss function is the standard *mean squared error* loss but it uses the M-norm which is aligned with the PDE framework.

Additionally, a clustering loss is defined as the cross entropy loss

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

where j is drawn from the subset $P \subset B$ that contains only those indices corresponding to data that are part of the clustered and labelled data set D_{cl} as described in (4.14). The cross-entropy loss function measures the distance between two probability distributions and is widely used as a loss function for training multi-class classification machine learning models. Note that both the labels $\mathbf{l}^{(j)}$ which are unit vectors representing a sharp unimodal distribution, and the clustering output $\boldsymbol{\alpha}^{(j)} = c(\boldsymbol{\rho}^{(j)})$ (due to the modified softmax function σ in the final layer) represent probability distributions.

Finally, a joint loss function is defined as

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

where the weight 10^{-4} was found to be an effective balance between accuracy and overfitting.

4.3.6 Polytope Error and Polytopic LPV Representation

PAEs guarantee that every state is reconstructed within a polytope. However, they do not ensure that the polytope is well constructed in terms of the state reconstruction.

Definition 4.6 (Polytope error and best approximation):
Let $\tilde{X} \subset \mathbb{R}^n$ be a convex polytope. For a given data point $\mathbf{x} \in \mathbb{R}^n$, let $\| \cdot \|_{\mathsf{M}}$ denote the norm induced by the M-weighted inner product. Define the *polytope error* as

$$\mathrm{dist}_{\mathsf{M}}(\mathbf{x}, \tilde{X}) := \min_{\mathbf{w} \in \tilde{X}} \|\mathbf{x} - \mathbf{w}\|_{\mathsf{M}}$$

and let $\mathbf{x}^* \in \tilde{X}$ be the *best approximation* that achieves the minimum. Figure 4.4 shows the conceptual representation of the polytope error. ◊

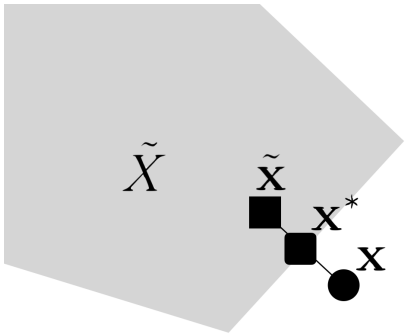


Figure 4.4: A schematic figure depicting the positions of a state $\mathbf{x}$, its reconstruction $\tilde{\mathbf{x}}$ in a polytope $\tilde{X}$, and its best approximation $\mathbf{x}^*$

Here, the polytope is denoted by $\tilde{X}$, and its quality is evaluated according to Definition 4.6. The polytope error with respect to the best approximation is well defined, as established by the following three lemmas:

Lemma 4.7:
Let A be a positive definite matrix. Then

$$\mathbf{a}^\top A\mathbf{b} \leq \sqrt{\mathbf{a}^\top A\mathbf{a}}\sqrt{\mathbf{b}^\top A\mathbf{b}}$$

for any nonzero vectors $\mathbf{a}$ and $\mathbf{b}$. ◊

Proof. Since A is positive definite, the quadratic form satisfies the inequality

$$\begin{aligned}(\mathbf{a} - c\mathbf{b})^\top A(\mathbf{a} - c\mathbf{b}) &\geq 0\\ \rightarrow \mathbf{a}^\top A\mathbf{a} - 2c\mathbf{a}^\top A\mathbf{b} + c^2\mathbf{b}^\top A\mathbf{b} &\geq 0\end{aligned}$$

where c is a scalar. Let

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

Then $p(c) \geq 0$ is obtained. Hence, the discriminant of $p(c)$ is less than 0 as follows:

$$\begin{aligned}(\mathbf{a}^\top A\mathbf{b})^2 - (\mathbf{a}^\top A\mathbf{a})(\mathbf{b}^\top A\mathbf{b}) &\leq 0\\ \rightarrow (\mathbf{a}^\top A\mathbf{b})^2 &\leq (\mathbf{a}^\top A\mathbf{a})(\mathbf{b}^\top A\mathbf{b})\\ \rightarrow -\sqrt{\mathbf{a}^\top A\mathbf{a}}\sqrt{\mathbf{b}^\top A\mathbf{b}} \leq \mathbf{a}^\top A\mathbf{b} &\leq \sqrt{\mathbf{a}^\top A\mathbf{a}}\sqrt{\mathbf{b}^\top A\mathbf{b}}\end{aligned}$$

Thus,

$$\mathbf{a}^\top A\mathbf{b} \leq \sqrt{\mathbf{a}^\top A\mathbf{a}}\sqrt{\mathbf{b}^\top A\mathbf{b}}. \tag{4.17}$$

Moreover, we note that the strict inequality in (4.17) holds if and only if $\mathbf{a}$ and $\mathbf{b}$ are linearly independent. □

Lemma 4.8:
Let $\tilde{X} \subset \mathbb{R}^n$ be a convex polytope. For any $\mathbf{x} \in \mathbb{R}^n$, there exists a unique $\mathbf{x}^* \in \tilde{X}$ such that

$$\|\mathbf{x} - \mathbf{x}^*\|_{\mathsf{M}} = \min\{\|\mathbf{x} - \tilde{\mathbf{x}}\|_{\mathsf{M}} \colon \tilde{\mathbf{x}} \in \tilde{X}\}. \qquad \lozenge$$

Proof. Without loss of generality, we can assume that $\mathbf{x} \notin \tilde{X}$. Otherwise, we have that $\mathbf{x}^* = \mathbf{x}$ is the unique minimizer. Define

$$f_{\mathbf{x}}(\mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_{\mathsf{M}}, \quad \text{for } \mathbf{x}, \mathbf{y} \in \tilde{X}.$$

Consider first the existence of a minimum of $f_{\mathbf{x}}$. Since $\tilde{X}$ is compact and $f_{\mathbf{x}}$ is continuous, there exists a minimizer $\mathbf{x}^* \in \tilde{X}$ such that

$$f_{\mathbf{x}}(\mathbf{x}^*) \leq f_{\mathbf{x}}(\mathbf{y}), \quad \forall \mathbf{y} \in \tilde{X}. \tag{4.18}$$

Thus, $f_{\mathbf{x}}(\mathbf{x}^*)$ is a minimum of $f_{\mathbf{x}}$.

Next, let $\mathbf{y}, \mathbf{z} \in \tilde{X}$. We have

$$\begin{aligned} f_{\mathbf{x}}(\lambda\mathbf{z} + (1-\lambda)\mathbf{y}) &= \|\mathbf{x} - (\lambda\mathbf{z} + (1-\lambda)\mathbf{y})\|_{\mathsf{M}} \\ &= \|\mathbf{x} - \lambda\mathbf{x} + \lambda\mathbf{x} - (\lambda\mathbf{z} + (1-\lambda)\mathbf{y})\|_{\mathsf{M}} \\ &= \|\lambda(\mathbf{x} - \mathbf{z}) + (1-\lambda)(\mathbf{x} - \mathbf{y})\|_{\mathsf{M}}. \end{aligned}$$

for any $\lambda \in \mathbb{R}$, $0 \leq \lambda \leq 1$.

Let $\mathbf{u} = \mathbf{x} - \mathbf{z}$ and $\mathbf{v} = \mathbf{x} - \mathbf{y}$. Then

$$\begin{aligned} &\|\lambda\mathbf{u} + (1-\lambda)\mathbf{v}\|_{\mathsf{M}} \\ &\qquad = \sqrt{(\lambda^2\|\mathbf{u}\|_{\mathsf{M}}^2 + 2\lambda(1-\lambda)\mathbf{u}^\top\mathsf{M}\mathbf{v} + (1-\lambda)^2\|\mathbf{v}\|_{\mathsf{M}}^2} \\ &\qquad \leqslant \sqrt{(\lambda^2\|\mathbf{u}\|_{\mathsf{M}}^2 + 2\lambda(1-\lambda)\|\mathbf{u}\|_{\mathsf{M}}\|\mathbf{v}\|_{\mathsf{M}} + (1-\lambda)^2\|\mathbf{v}\|_{\mathsf{M}}^2} \quad (\because \text{Lemma 4.7}) \\ &\qquad = \sqrt{(\lambda f_{\mathbf{x}}(\mathbf{z}) + (1-\lambda)f_{\mathbf{x}}(\mathbf{y}))^2} \\ &\qquad = \lambda f_{\mathbf{x}}(\mathbf{z}) + (1-\lambda)f_{\mathbf{x}}(\mathbf{y}), \end{aligned}$$

for any λ, $0 \leq \lambda \leq 1$.

$$\therefore f_{\mathbf{x}}(\lambda\mathbf{z} + (1-\lambda)\mathbf{y}) \leq \lambda f_{\mathbf{x}}(\mathbf{z}) + (1-\lambda)f_{\mathbf{x}}(\mathbf{y}). \tag{4.21}$$

To prove the uniqueness of $\mathbf{x}^*$, assume that there exists $\mathbf{w} \in \tilde{X}$ such that

$$f_{\mathbf{x}}(\mathbf{x}^*) = f_{\mathbf{x}}(\mathbf{w}).$$

Because of the convexity of $\tilde{X}$, it follows that for any λ, $0 \leq \lambda \leq 1$,

$$\mathbf{x}_\lambda = \lambda\mathbf{x}^* + (1-\lambda)\mathbf{w} \in \tilde{X}.$$

Then

$$\begin{aligned} f_{\mathbf{x}}(\mathbf{x}_\lambda) \leq & \lambda f_{\mathbf{x}}(\mathbf{x}^*) + (1-\lambda) f_{\mathbf{x}}(\mathbf{w}) \quad \text{(by (4.21))} \\ = & \lambda f_{\mathbf{x}}(\mathbf{x}^*) + (1-\lambda) f_{\mathbf{x}}(\mathbf{x}^*) \quad (\because f_{\mathbf{x}}(\mathbf{x}^*) = f_{\mathbf{x}}(\mathbf{w})) \\ = & f_{\mathbf{x}}(\mathbf{x}^*). \end{aligned}$$

By the minimum property (4.18), this can only be the case if

$$f_{\mathbf{x}}(\mathbf{x}_\lambda) = f_{\mathbf{x}}(\mathbf{x}^*) = f_{\mathbf{x}}(\mathbf{w}),$$

and since the equality in Lemma 4.7 only holds for linear dependent vectors, we have

$$\mathbf{x} - \mathbf{x}^* = \beta(\mathbf{x} - \mathbf{w}) \quad \text{for some } \beta \in \mathbb{R}.$$

Then we obtain

$$f_{\mathbf{x}}(\mathbf{x}^*) = \|\mathbf{x} - \mathbf{x}^*\|_{\mathsf{M}} = \|\beta(\mathbf{x} - \mathbf{w})\|_{\mathsf{M}} = |\beta| \|\mathbf{x} - \mathbf{w}\|_{\mathsf{M}} = |\beta| f_{\mathbf{x}}(\mathbf{w})$$

Since $f_{\mathbf{x}}(\mathbf{x}^*) = f_{\mathbf{x}}(\mathbf{w})$, we conclude $\beta = \pm 1$.

For $\beta = -1$, we have $\mathbf{x} - \mathbf{x}^* = -\mathbf{x} + \mathbf{w}$ and thus, because of convexity of $\tilde{X}$ that

$$\therefore \mathbf{x} = \frac{\mathbf{x}^* + \mathbf{w}}{2} \in \tilde{X}.$$

However, this contradicts $\mathbf{x} \notin \tilde{X}$. Thus,

$$\beta = 1,$$

and, therefore, we conclude $\mathbf{x}^* = \mathbf{w}$, i.e., $f(\mathbf{x}^*)$ is the unique minimum. □

Given available data, the polytope error is evaluated for the identified polytope used in the encoding and reconstruction by a PAE. This analysis provides best-case estimates for:

1. the extent to which the identified polytope can represent the data and
2. the proximity of the reconstruction to this theoretical lower bound.

Computing the best approximation is a nontrivial task. To determine the polytope error for a given $\mathbf{x}$, the following optimization problem is solved:

$$\begin{aligned} \min_{\rho \in \mathbb{R}^r} \|U\boldsymbol{\rho} - \mathbf{x}\|_{\mathsf{M}}^2, \\ \text{subject to} \quad \boldsymbol{\rho} \geq 0, \\ \mathbb{1}_r \boldsymbol{\rho} = 1, \end{aligned}$$

for the coordinates $\boldsymbol{\rho}^*$ of the best approximation $\mathbf{x}^* = U\boldsymbol{\rho}$ where U is the matrix of vertices (4.12), and $\mathbb{1}_r = [1, 1, \cdots, 1]$ is the row vector of ones with r entries.

This convex quadratic programming problem, subject to linear constraints, does not have a closed-form solution. However, numerical methods such as active-set and interior-point methods can efficiently compute solutions. In the experiments, a quadratic programming solver from the Python library `cvxopt` is utilized, which implements interior-point methods [10].

4.3.7 Application in Polytopic LPV Approximations

The intended application of approximating general nonlinear functions

$$f\colon \mathbb{R}^n \to \mathbb{R}^n$$

using LPV approximations with preferably low parameter dimension is briefly discussed. Such approximations serve as a promising component in nonlinear controller design; e.g., [31] for the foundational theory and proof-of-concept studies.

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

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

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

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

Given the utility of affine linear polytopic LPV representations in controller design, the following demonstrates how a polytopic reconstruction based on $\boldsymbol{\alpha}\otimes\boldsymbol{\rho}$ can be reformulated as a polytopic LPV approximation, where $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ functions as the scheduling parameter set.

Lemma 4.9:
Let $A(\cdot)\colon \mathbb{R}^n \to \mathbb{R}^{n\times n}$ be a linear map and let $\tilde{\mathbf{x}}(t)$ be a convex combination of n vertices of a polytope. Then $A(\tilde{\mathbf{x}}(t))$ is a convex linear combination of n matrices representing a polytope in $\mathbb{R}^{n\times n}$.

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

$$\tilde{\mathbf{x}}(t) = \sum_{i=1}^{n} \eta_i \mathbf{u}_i,$$

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

$$A(\tilde{\mathbf{x}}(t)) = A(\sum_{i=1}^{n} \eta_i \mathbf{u}_i).$$

Since A is linear in its argument, it follows that

$$A(\tilde{\mathbf{x}}(t)) = A(\sum_{i=1}^{n} \eta_i \mathbf{u}_i) = \sum_{i=1}^{n} \eta_i A_i,$$

where $A_i := A(\mathbf{u}_i)$. Therefore, $A(\tilde{\mathbf{x}}(t))$ is a convex combination of $A_1, A_2, \cdots, A_n$. □

The standard alternative approach to obtaining an affine LPV representation with parameter dimension r within a polytope involves determining the r-dimensional bounding box; e.g., [72]. In this case, the polytope has 2^r vertices, leading to an exponential increase in complexity as r grows. If a bounding polytope with fewer vertices can be identified, the challenge shifts to computing the corresponding coordinates, for which no established method exists for dimensions beyond $r = 3$; see the discussion in [31].

In contrast, PAEs offer a promising alternative based on Lemma 4.9. Firstly, the relevant parametrization $\boldsymbol{\alpha} \otimes \boldsymbol{\rho}$ inherently defines the required bounding polytope whose size $r \cdot k$ scales linearly with the number of clusters k and the parameter dimension r. Notably, this linear growth becomes advantageous compared to the exponential growth of 2^r for moderate and large values of r;

$$\text{e.g., } r \cdot k < 2^r \text{ for } r \geq 5 \text{ and } k \leq r.$$

Second, the decoder directly provides the coordinates within the constructed polytope. This implies that no additional procedure is needed to construct a bounding box; see the discussion in e.g., [60].

4.4 Simulation Results

This section investigates PAEs using the two datasets of incompressible flows introduced introduced in Section 2.4.1 such as the trajectories of reconstructed states and latent variables, and clustering results. For each reduced dimension r, the reconstruction performance of PAEs is evaluated in comparison to other methods by computing the averaged relative error defined in (3.5). In addition, based on Definition 4.6, the averaged relative polytope error ε_p is defined as

$$\varepsilon_p := \frac{1}{T} \sum_{i=1}^{T} \frac{\|\mathbf{x}_i^* - \mathbf{x}_i\|_{\mathsf{M}}}{\|\mathbf{x}_i\|_{\mathsf{M}}},$$

where T denotes the total number of snapshots, and $\mathbf{x}_i$ and $\mathbf{x}_i^*$ represent the original state and its best approximation, respectively.

For memory efficiency analysis, the number of encoding parameters, decoding parameters, and vertices of the polytopic LPV representations is reported. The number of encoding parameters in both CAE and PAE is significantly lower than that in POD. This reduction arises from the use of the fixed and highly sparse interpolation matrix I_C, along with the implementation of depthwise separable convolutions. Regarding the number of decoding parameters, both POD and CAE employ linear decoders, leading to a size of nr where $n \times r$ represents the matrix dimensions. In contrast, the decoding size for PAE is $nrk + m$ where m denotes the number of parameters in the clustering model.

As an additional performance metric, the number R of vertices in the polytope containing the reconstruction values is reported, as this quantity is relevant for LPV approximations; cp. Section 4.3.7. In the case of POD approximations, the empirically defined

Table 4.1: Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, c: a clustering layers ("#" refers to the number of.)

	POD	CAE	PAE
#encoding layers	pod	$I_c + 13\mathsf{cv} + \mathsf{fc}$	$I_c + 13\mathsf{cv} + \mathsf{fc}$
#decoding layers	pod	fc	fc
#encoding parameters ($r = 2$)	85,528	**36,692**	**36,692**
#decoding parameters ($r = 2$)	**85,528**	**85,528**	256,584
#vertices of a polytope ($r = 2$)	4	**2**	6
#encoding parameters ($r = 3$)	128,292	**36,725**	**36,725**
#decoding parameters ($r = 3$)	**128,292**	**128,292**	384,876
#vertices of a polytope ($r = 3$)	8	**3**	9
#encoding parameters ($r = 5$)	213,820	**36,791**	**36,791**
#decoding parameters ($r = 5$)	**213,820**	**213,820**	641,460
#vertices of a polytope ($r = 5$)	32	**5**	15
#encoding parameters ($r = 8$)	342,112	**36,890**	**36,890**
#decoding parameters ($r = 8$)	**342,112**	**342,112**	1,026,336
#vertices of a polytope ($r = 8$)	256	**8**	24

bounding box for $\boldsymbol{\rho}$ is used as a surrogate polytope, resulting in

$$R = 2^r,$$

where r is the dimension of $\boldsymbol{\rho}$ which is the standard approach in the absence of an algorithm for computing a polytopic expansion within a general polytope. However, for CAE and PAE, this polytopic expansion is naturally provided within a polytope of

$$R = r \text{ or } R = kr$$

vertices where k is the number of clusters.

In practice, selecting appropriate values for the latent dimension r and the number of clusters k involves balancing accuracy and computational complexity. For PAEs, the numerical results indicate that in controller design, it is generally preferable to increase r while setting $k = 1$. For projection-based model reduction (cp. Equation (4.3)), choosing $k > 1$ can lead to improved reconstruction accuracy with only a minor increase in computational cost; see, e.g., Figure 4.5.

4.4.1 Single-cylinder

In this simulation, both the PAE and CAE employ a deep convolutional encoder comprising 14 convolutional layers followed by a fully connected layer. The convolutional layers

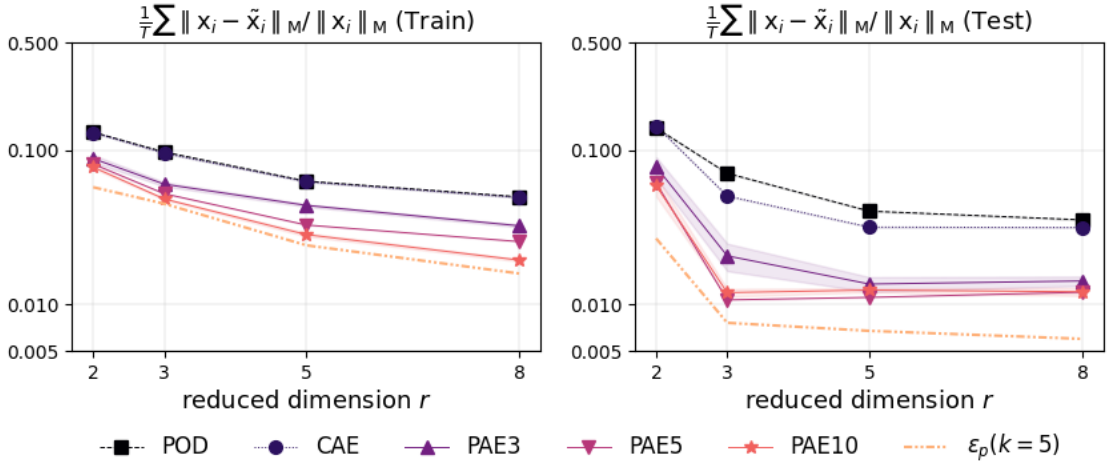


Figure 4.5: Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the single-cylinder case. The shaded regions represent statistical uncertainty across multiple training runs. However, this uncertainty is negligible for most methods, rendering it nearly invisible in the plots.

utilize the ELU activation function while the final layer applies the modified softmax function (4.8). To reduce the number of nodes in the last layer, global average pooling is performed before applying the fully connected layer.

The decoder of POD is formulated as a linear combination of r basis vectors, whereas the CAE decoder is a convex combination of r vertices. The PAE decoder exhibits partial linearity as discussed in Section 4.3.4. The CAE consists of an encoding part

$$\begin{aligned} \mathbf{x}_{\mathsf{CNN}}(t) &= I_C \mathbf{x}(t), \\ \boldsymbol{\rho}(t) &= \mu(\mathbf{x}_{\mathsf{CNN}}(t)), \end{aligned}$$

and a decoding part

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

without clustering. This implies that the CAE is regarded as a PAE with 1 cluster (i.e., $k = 1$).

Table 4.1 compares the number of encoding and decoding parameters. Both the CAE and PAE maintain a relatively constant number of encoding parameters across varying reduced dimensions in contrast with POD. In practice, for $r = 2, 3, 5, 8$, the encoding size of the CAE and PAE is only $42.6\%, 28.6\%, 17.2\%$, and 10.8%, respectively, of the encoding size of the POD, based on the number of encoding parameters.

The decoding size of POD and CAE is determined 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 $\boldsymbol{\rho}$.

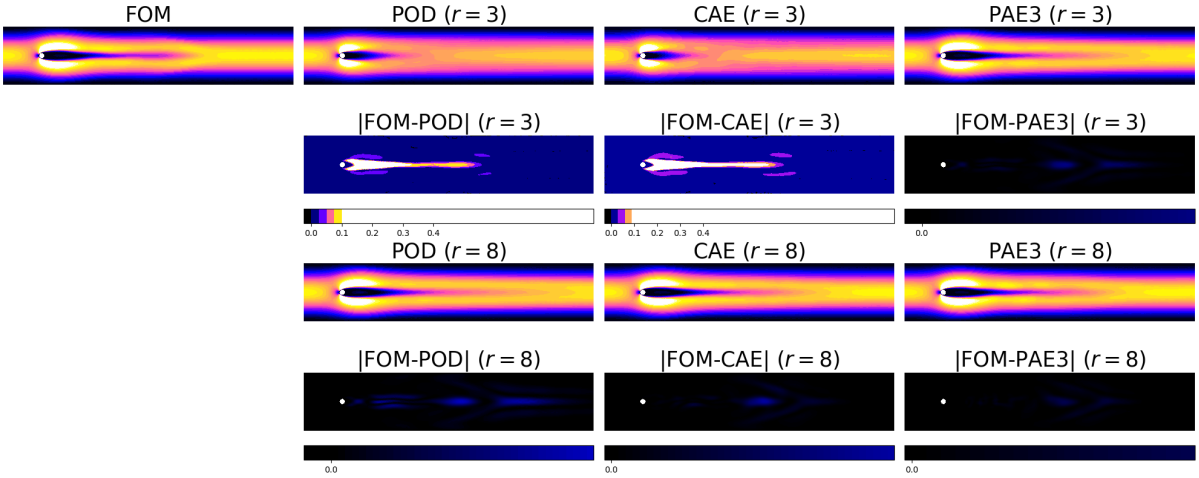


Figure 4.6: Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 2.0$: training session for the single-cylinder case

As discussed in Section 4.3.2, it is observed that depthwise separable convolutions require 10.6% fewer parameters than standard convolutions, although the PAE has 10 more encoding layers based on Table 3.1 and Table 4.1.

Next, the reconstruction of periodic flows using PAEs and the organization of their latent variables in very low-dimensional spaces are examined. Figure 4.5 displays the reconstruction errors of POD, CAE, and PAE with respect to the reduced dimension r. These errors are computed by averaging the errors from 5 training trials, which result 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 surpasses POD in test reconstruction errors except for the error at $r = 2$. Regarding the reconstruction error over time, the CAE exhibits higher errors than POD during the transition period, when the flows shift from relatively stable to periodic states (approximately within the time range $[4, 6]$). However, the CAE reconstructs periodic flows more accurately than POD.

This phenomenon is hypothesized to result from the model's inherent learning bias toward periodic flow structures. The disproportionately large number of periodic flow snapshots skews the distribution of the training data, leading to a common machine learning issue known as data imbalance.

Overall, the reconstruction performance of CAEs is comparable to that of POD. However, CAEs achieve this with fewer model parameters and identify a smaller R for designing polytopic LPV systems. The PAE(k) models, which cluster latent variables into k clusters (e.g., PAE3, PAE5, PAE10), outperform both CAEs and POD. Moreover, the reconstruction errors of PAEs tend to decrease as k increases. However, the difference in reconstruction errors between PAE5 and PAE10 is not significant.

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

Table 4.2: Computational times for $r = 3$: the offline time for POD represents the time required to obtain a POD basis on the CPU. For CAE and PAE, the offline time corresponds to the training time on the GPU. The inference time indicates the runtime for reconstructing a state on the CPU in the single-cylinder case.

As a result, the polytope errors for $k = 5$ remain below 2.7% for the periodic flows in the testing range $[10, 16]$. Specifically, the reconstruction errors for the reduced dimensions $r = 2, 3, 5, 8$ are $2.7\%, 0.8\%, 0.7\%$, and 0.6% respectively. This suggests that the polytopes defined by PAE5 are well-constructed, even when working with very low-dimensional latent states.

Figure 4.6 and Figure 4.7 present a comparison of the developed snapshots from FOM, POD, CAE, and PAE3 at times $t = 2.0, 14.0$ respectively for $r = 3, 8$. The absolute error plots demonstrate that PAE3 outperforms the other models in terms of state reconstruction, even with very low-dimensional parametrizations.

Table 4.2 presents the computational timings[1] for training the AEs and evaluating their performance compared to the POD. The computation of the POD coefficients is approximately 340 times faster than training the neural networks of the CAE and PAEs. When it comes to state reconstruction from latent variables, POD outperforms CAE and PAE by a factor of about 3.

Figure 4.8 illustrates the activation rates of polytope vertices and the trajectories of latent state variables for $r = 2$ with $k = 3$ (i.e., PAE3). The results indicate that the latent variables for each PAE remain in the range $[0, 1]$ as the coefficients of a convex combination.

The activation rate quantifies the relative contribution of each latent variable to the state reconstruction. The activation rate of the i-th latent variable is defined as

$$\frac{\sum_{q=1}^{N} \rho_{i,q}}{\sum_{p=1}^{r} \sum_{q=1}^{N} \rho_{p,q}},$$

where N is the number of snapshots. Since

$$\begin{aligned}\frac{\sum_{q=1}^{N}(\alpha_1\rho_{i,q} + \cdots \alpha_k\rho_{i,q})}{\sum_{p=1}^{r}\sum_{q=1}^{N}(\alpha_1\rho_{p,q} + \cdots \alpha_k\rho_{p,q})} &= \frac{\sum_{q=1}^{N}(\alpha_1 + \cdots \alpha_k)\rho_{i,q}}{\sum_{p=1}^{r}\sum_{q=1}^{N}(\alpha_1 + \cdots \alpha_k)\rho_{p,q}} \\ &= \frac{\sum_{q=1}^{N} \rho_{i,q}}{\sum_{p=1}^{r}\sum_{q=1}^{N} \rho_{p,q}},\end{aligned}$$

[1] System specifications: Intel i5-12600K CPU@3.70GHz, 32GB RAM, NVIDIA RTX A4000 GPU

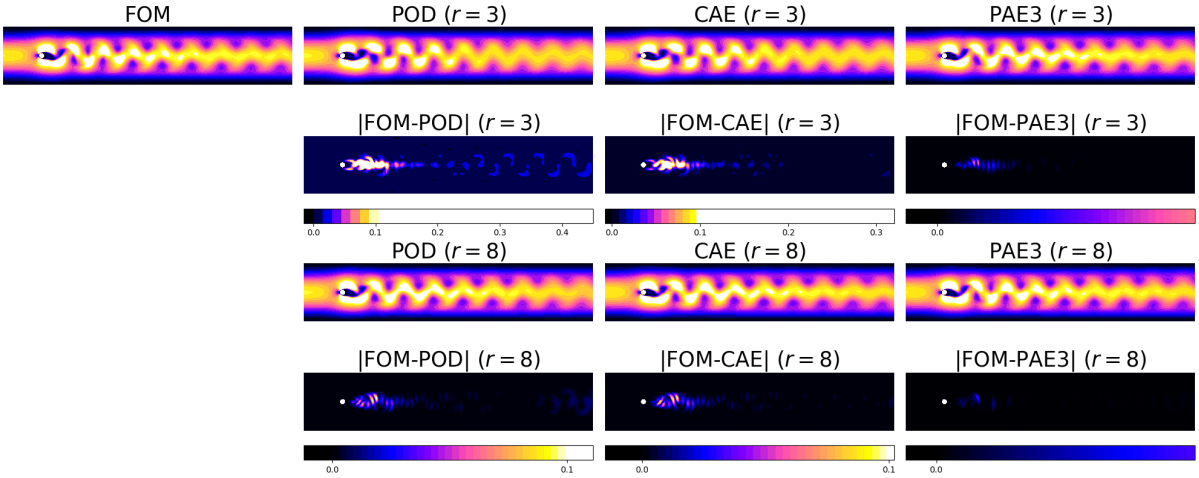


Figure 4.7: Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 14.0$: testing session for the single-cylinder case

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

As illustrated in the bar chart, when $r = 2$, the state reconstruction primarily relies on the first latent variable which contributes 77.9%. Consequently, the three vertices weighted by ρ_1 of the polytope play a dominant role in the state reconstruction, with an influence rate of 77.9%.

Figure 4.9 verifies that the latent variables adhere to the convex combination constraints,

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

and that the clustering network c categorizes latent states in a manner similar to k-means clustering. In other words, all latent variables align along the line

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

when $r = 2$.

Figure 4.10 depicts the activation rates of the polytope vertices along with the trajectories of latent state variables for $r = 3$ and $k = 3$ (i.e., PAE3). For any r, the encoder guarantees that all latent variables remain nonnegative with each $\boldsymbol{\rho}(t)$ satisfying the constraint that its elements sum to 1. As a result, the trajectories of the latent variables remain confined within the expected range of $[0, 1]$.

In the bar chart, the activation rates of each latent state are 17.4%, 58.9%, and 23.7% respectively. It indicates that the three vertices weighted by $\rho_2(t)$ of the polytope contribute more significantly to the state reconstruction than the other vertices.

Another observation is that relatively stable states (approximately in the range $[0, 4]$) are primarily reconstructed by a significant contribution from ρ_2 placing them in a tran-

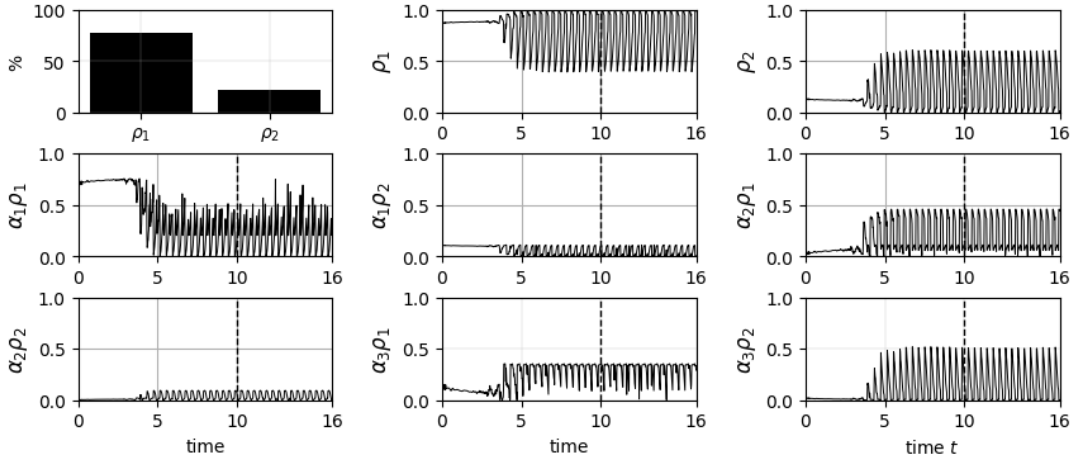


Figure 4.8: Activation rates and trajectories of latent state variables for $r = 2$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case.

sition area between the first and third clusters. This is evident in the graphs of $\alpha_1\rho_2$ and $\alpha_3\rho_2$ in the time range $[0, 4]$. In this way, it is evident that ρ_2 has a significant influence on the reconstruction of periodic flows.

Figure 4.11 shows the distribution of latent variables in a three-dimensional space comparing the labels obtained from the clustering net with those from k-means clustering. Since the latent state variables represent the coefficients of a polytope, they are nonnegative and lie on a plane.

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

The clustering network assigns labels to latent variables in a way that closely resembles k-means clustering, as it uses pseudo-labels derived from k-means clustering. However, the clustering network is less constrained by centroid distances due to its training with the joint loss function described in Section 4.3.5.

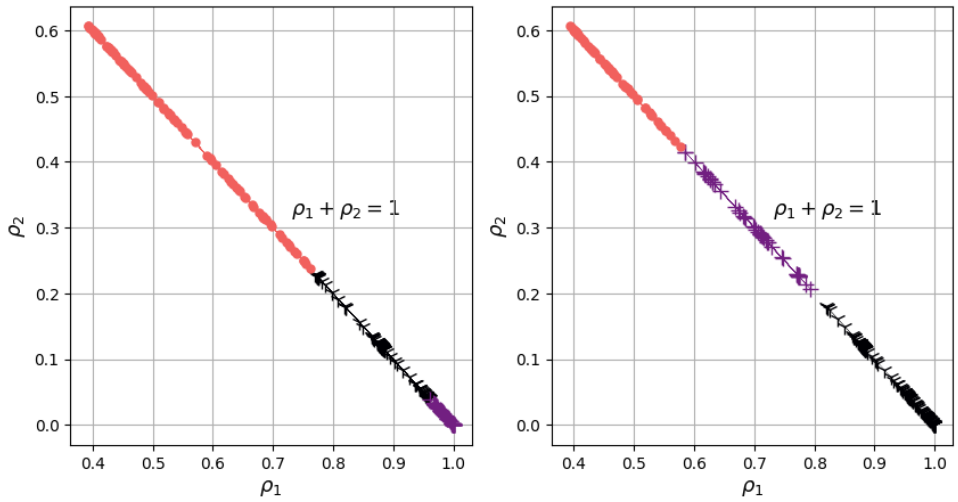


Figure 4.9: Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the single-cylinder case. The different colors represent distinct clusters.

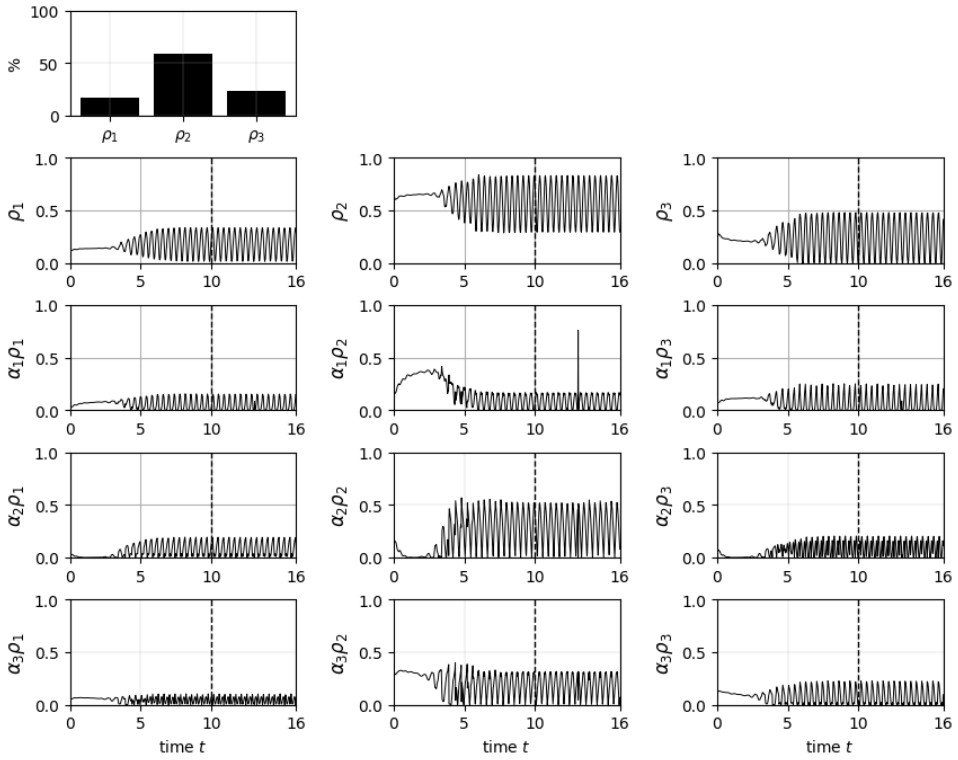


Figure 4.10: Activation rates and trajectories of latent state variables for $r = 3$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case.

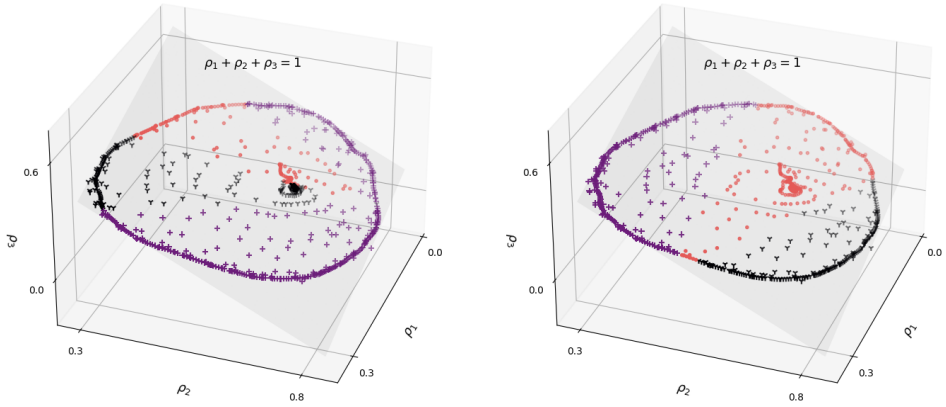


Figure 4.11: Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the single-cylinder case. The different colors represent distinct clusters.

Table 4.3: Model specification: fc: a fully connected layer, pod: a POD basis, I_c: an interpolation matrix, cv: a convolutional layer, c: a clustering layers ("#" refers to the number of.)

	POD	CAE	PAE
#encoding layers	pod	I_c + 25cv + fc	I_c + 25cv + fc
#decoding layers	pod	fc	fc
#encoding parameters ($r = 2$)	92,028	**91,410**	**91,410**
#decoding parameters ($r = 2$)	**92,028**	**92,028**	276,084
#vertices of a polytope ($r = 2$)	4	**2**	6
#encoding parameters ($r = 3$)	138,042	**91,443**	**91,443**
#decoding parameters ($r = 3$)	**138,042**	**138,042**	414,126
#vertices of a polytope ($r = 3$)	8	**3**	9
#encoding parameters ($r = 5$)	230,070	**91,509**	**91,509**
#decoding parameters ($r = 5$)	**230,070**	**230,070**	690,210
#vertices of a polytope ($r = 5$)	32	**5**	15
#encoding parameters ($r = 8$)	368,112	**91,608**	**91,608**
#decoding parameters ($r = 8$)	**368,112**	**368,112**	1,104,336
#vertices of a polytope ($r = 8$)	256	**8**	24

4.4.2 Double-cylinder

In this simulation, each PAE and CAE features a deep convolutional encoder comprising 26 convolutional layers, which include Inverted residual blocks, and a fully connected layer. The ELU activation function is applied to the convolutional layers, and the modified softmax function (4.8) is used in the last layer. To decrease the number of nodes in the final layer, global average pooling is applied prior to the fully connected computation, following the same architecture as the model used for the single-cylinder case in Section 4.4.1.

Table 4.3 provides a summary of the parameters used in the different models. 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.

In Figure 4.12, all the models exhibit a general trend of decreasing reconstruction error as the reduced dimension r increases during the training phase. This trend highlights the expected improvement in approximation accuracy when preserving more latent information. Notably, the results demonstrate that PAE consistently outperforms both POD and CAE in terms of reconstruction performance across different values of r.

The superior performance of PAEs suggests that incorporating polytopic structures allows for a more expressive and flexible representation of the data manifold, leading to reduced reconstruction errors. Furthermore, as both the number of clusters k and the reduced dimension r increase, the reconstruction errors of PAEs decline significantly,

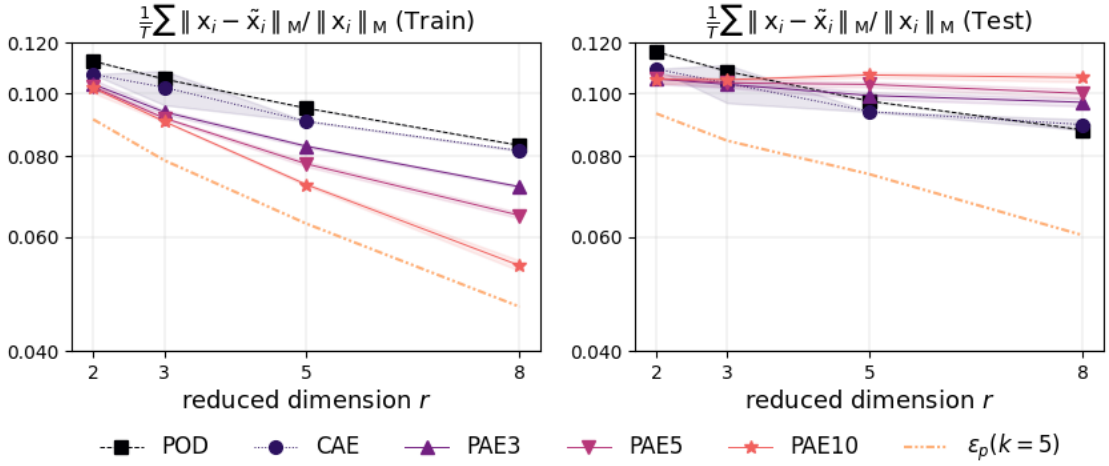


Figure 4.12: Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the double-cylinder case. The shaded regions represent statistical uncertainty across multiple training runs. However, this uncertainty is negligible for most methods, rendering it nearly invisible in the plots.

indicating the benefits of using a higher number of convex components in capturing the underlying data distribution.

Despite these promising results, a noticeable discrepancy between training and test errors is observed, with test errors consistently exceeding their training counterparts. This suggests that while PAEs achieve excellent reconstruction within the training set, their ability to generalize to unseen data remains a challenge.

The results indicate that PAEs fit the training data exceptionally well, potentially to the extent of overfitting. This issue persists despite implementing several regularization strategies aimed at enhancing model generalization, including $L1$ and L_2 regularization, *Dropout* [100], *label smoothing* [106], and reducing the number of trainable parameters. The persistence of generalization issues suggests that while PAEs leverage polytopic clustering effectively, further refinements in the training process or regularization strategies may be necessary to improve their robustness on unseen data.

Nevertheless, an important observation is that the polytope errors ε_p with $k = 5$ reach a relatively stable and consistent level in both training and test phases. This consistency implies that each polytope defined by the PAEs is well-constructed, capturing meaningful structures within the dataset. The stability of ε_p suggests that while overfitting remains a concern, the underlying polytope representation maintains coherence across different data distributions. Given these findings, there is potential to further enhance model performance by refining the encoding and clustering mechanisms involved in determining the convex combination coefficients. Improvements in these components could lead to better generalization by enforcing a more structured latent space, thereby balancing expressiveness with robustness.

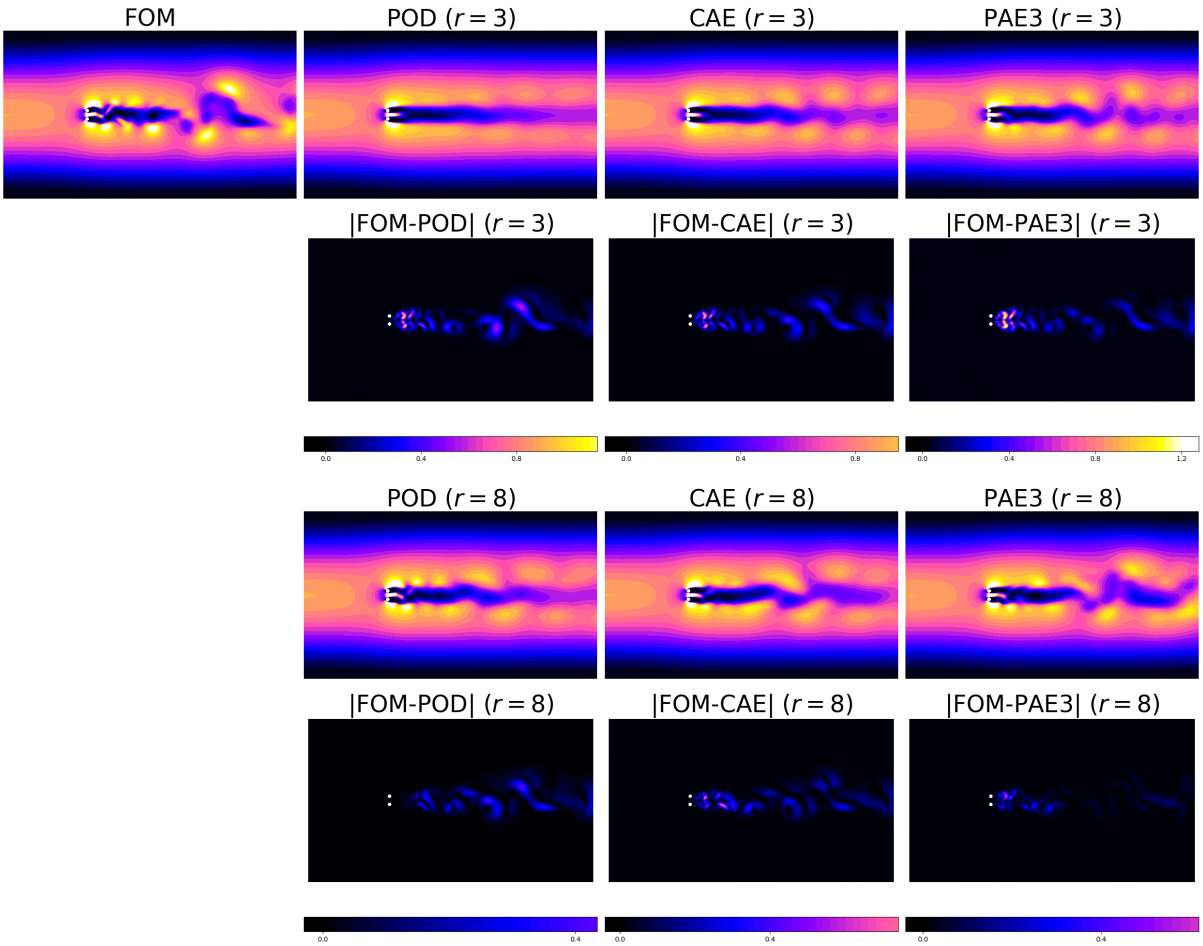


Figure 4.13: Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 556.0$: training session for the double-cylinder case

Figure 4.13 and Figure 4.14 illustrate a comparative analysis of the reconstructed snapshots generated by FOM, POD, CAE, and PAE3 at two distinct time instances, namely $t = 556.0, 736.5$. The reconstructions are presented for two different reduced dimensions, $r = 3, 8$, to evaluate the performance of the dimensionality reduction techniques across different levels of compression.

At $t = 736.5$, noticeable differences are observed in the reconstructed snapshots, especially for $r = 3$. The POD-based reconstruction exhibits a visibly smoother flow in the cylinder wake region compared to both CAE and PAE3. This suggests that POD may introduce excessive smoothing at lower dimensions, which could lead to the loss of small-scale turbulent structures. On the other hand, CAE and PAE3 retain more localized flow features but exhibit slightly higher reconstruction errors in certain regions. When the

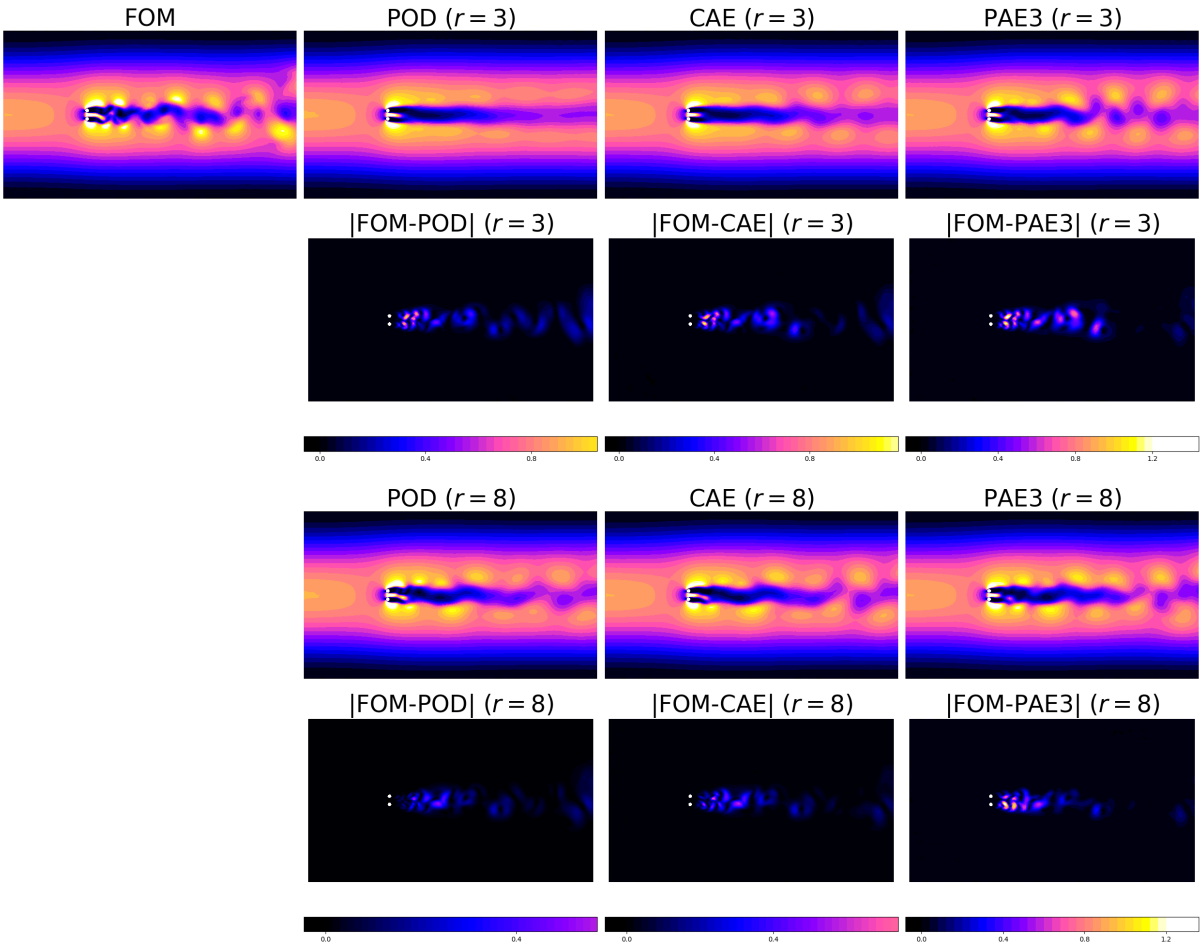


Figure 4.14: Comparison between the reference produced by the full-order model (FOM) and the snapshots generated by POD, CAE, and PAE at $t = 736.5$: testing session for the double-cylinder case

reduced dimension is increased to $r = 8$, the differences among the methods become less pronounced, as all models demonstrate an improved ability to capture the underlying flow structures with greater accuracy. The error distributions also indicate that increasing the latent dimension reduces the discrepancies between the reconstructions and the FOM, reinforcing the importance of selecting an appropriate reduced dimension based on the desired trade-off between accuracy and computational efficiency.

Overall, these results highlight the effectiveness of PAE3 in preserving critical flow features at lower dimensions while demonstrating its advantages over conventional methods such as POD and CAE. However, the increased complexity of PAEs necessitates further investigation into regularization techniques and hyperparameter tuning to ensure robustness and generalizability across different flow conditions.

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

Table 4.4: Computational times for $r = 3$: the offline time for POD represents the time required to obtain a POD basis on the CPU. For CAE and PAE, the offline time corresponds to the training time on the GPU. The inference time indicates the runtime for reconstructing a state on the CPU in the double-cylinder case.

Remark 4.10:
Generally, the parameter r denotes the dimension of the latent states. For POD this is equivalent to the size of $\rho(t)$. However, for polytopic decoding, due to the summation condition $\sum_{i=1}^{r} \rho_i(t) = 1$, one dimension is redundant. Therefore, the actual latent dimension is theoretically $r-1$. In the conducted experiments, this straightforward method for further reducing the parametrization was not employed. Nonetheless, within the context of controller design, the elimination of one degree of freedom could potentially lead to additional performance improvements. ◊

Proof. Since $\rho_1(t) + \rho_2(t) + \cdots + \rho_r(t) = 1$, polytopic decoders can be reformulated as follows:

$$\begin{aligned}
\tilde{\mathbf{x}}(t) = D\boldsymbol{\rho}(t) &= \sum_{i=1}^{r} \rho_i(t)\mathbf{d}_i \\
&= \sum_{i=1}^{r-1} \rho_i(t)\mathbf{d}_i + (1 - \rho_1(t) - \rho_2(t) - \cdots - \rho_{r-1}(t))\mathbf{d}_r \\
&= \mathbf{d}_r + \sum_{i=1}^{r-1} \rho_i(t)(\mathbf{d}_i - \mathbf{d}_r) \\
&= \mathbf{u}_0 + \sum_{i=1}^{r-1} \rho_i(t)\mathbf{u}_i
\end{aligned}$$

where D is a decoding matrix, $\mathbf{d}_i$ denotes the i-th column vector of D, $\mathbf{u}_0 = \mathbf{d}_r$, and $\mathbf{u}_i = \mathbf{d}_i - \mathbf{d}_r$ for $i = 1, 2, \cdots, r-1$. □

In terms of computational effort, as shown in Table 4.4, training the autoencoders requires approximately 500 times more time than computing the POD bases. However, the reconstruction time differs by a factor of only about 5.5. The relatively small increase observed for the PAE suggests that the incorporation of the polytopic structure does not introduce significant computational overhead. POD is the most computationally efficient, making it ideal for fast reconstructions. However, CAE and PAE, despite their higher

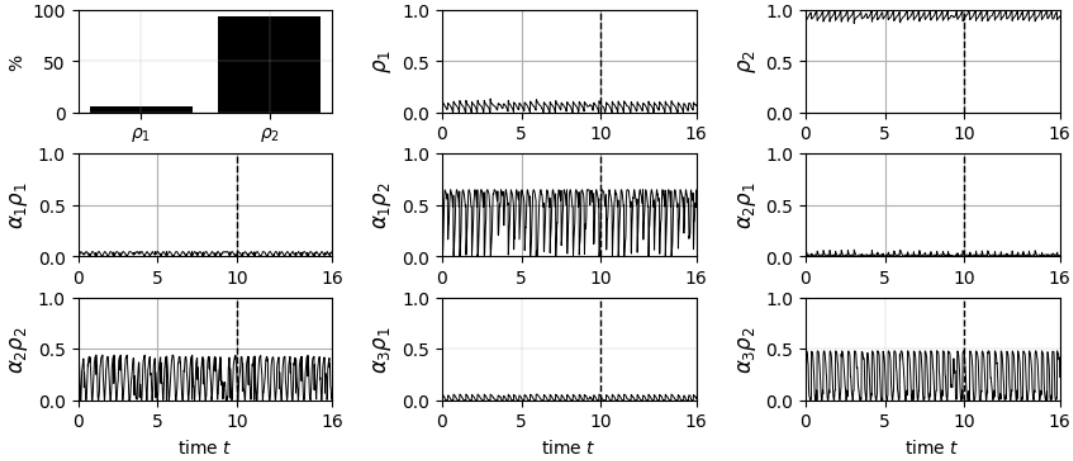


Figure 4.15: Activation rates and trajectories of latent state variables for $r = 2$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the double-cylinder case.

offline costs, offer improved expressive power for complex, nonlinear dynamics. Overall, the slight increase in computational cost for PAE suggests its potential as a promising alternative to standard AEs, and there is potential to increase the inference speed of PAE by optimizing the number of layers, reducing it to fewer than 27.

Figure 4.15 unequivocally demonstrates that $\rho_2(t)$ exerts a profoundly greater impact on state reconstruction achieving an impressive activation rate of 92.8%, which strongly suggests that the temporal evolution of $\rho_2(t)$ is the primary determinant in accurately reconstructing the state.

Figure 4.16 visually represents latent variables that satisfy convex combination constraints,

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

Furthermore, the clustering model c classifies latent variables in a manner similar to k-means clustering. However, in contrast to k-means clustering, the clustering net assigns only two labels to the latent states. In the clustering net, this suggests that two-dimensional latent states of chaotic flows may not exhibit a well-defined separation into three distinct groups, leading the model to merge data into fewer clusters. In other words, if two clusters are significantly more distinct than the third, the model may incorporate the less distinct cluster into one of the existing two, rather than treating it as an independent group.

In Figure 4.17, the activation rates are 17.6%, 11.8%, and 70.6% respectively. Consequently, the three vertices weighted by $\rho_3(t)$ in the polytope exert the most significant influence on state reconstruction, surpassing the contributions of the others. In contrast, both $\rho_1(t)$ and $\rho_2(t)$ display oscillatory behavior with low amplitudes. Furthermore, state reconstruction is predominantly governed by $\rho_3(t)$ regardless of the clustering results.

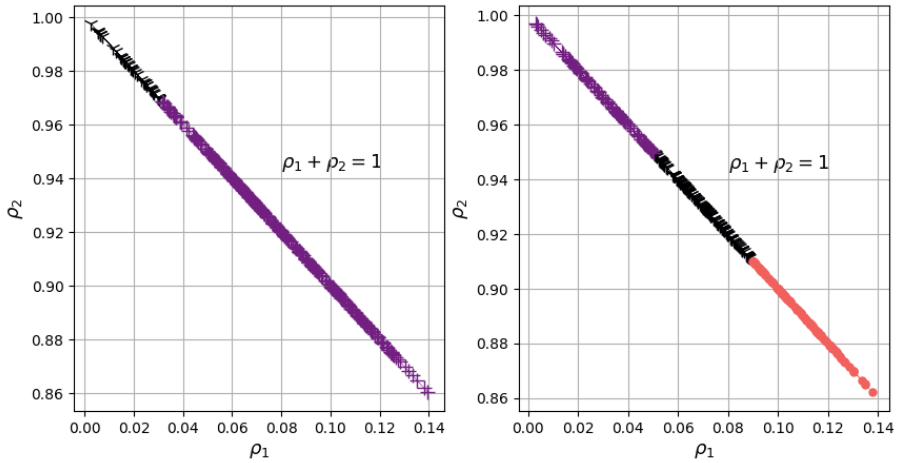


Figure 4.16: Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the two-dimensional space for the double-cylinder case. The different colors represent distinct clusters.

This suggests that all the considered chaotic flows are highly correlated with $\rho_3(t)$.

As shown in Figure 4.18, the trajectory of the latent states on the plane

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

Overall, the clustering net tends to classify latent variables similarly to k-means clustering. However, in some cases, it assigns labels differently from distance-based methods.

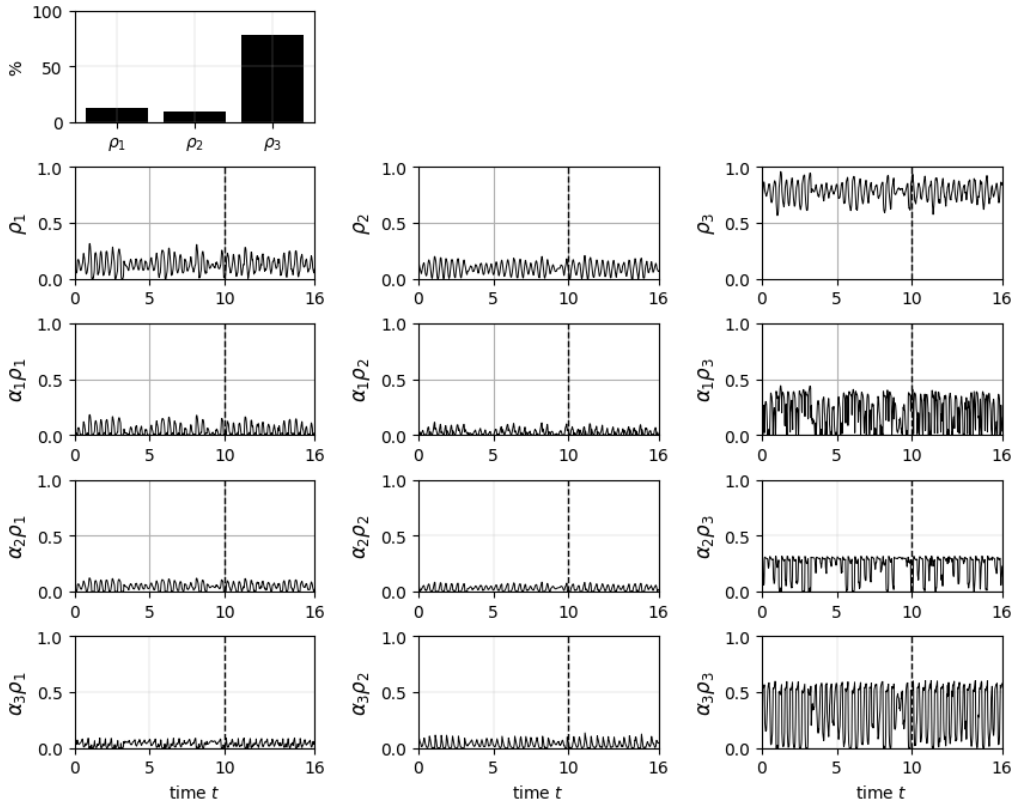


Figure 4.17: Activation rates and trajectories of latent state variables for $r = 3$ with $k = 3$: the dashed line distinguishes between the training phase and the extrapolation phase for the double-cylinder case.

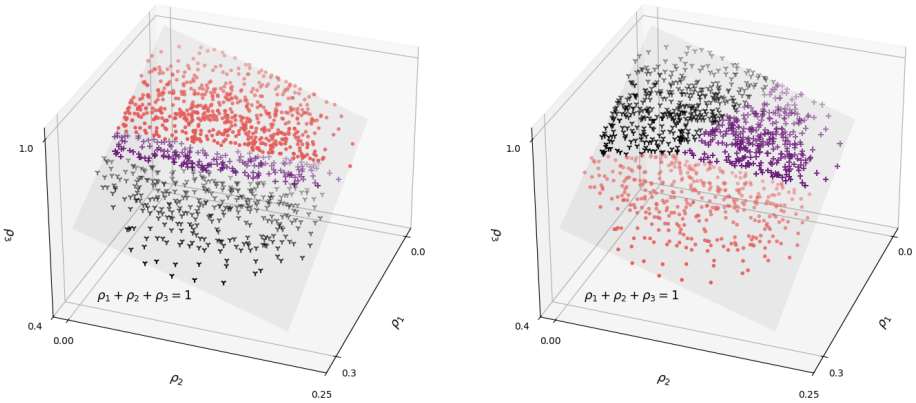


Figure 4.18: Comparison of the outcomes of (left) the smooth clustering $c(\boldsymbol{\rho})$ and (right) k-means clustering with 3 clusters in the three-dimensional space for the double-cylinder case. The different colors represent distinct clusters.

4.5 Conclusion

This section introduced a PAE architecture consisting of a lightweight nonlinear encoder, a convex combination decoder, and a differentiable clustering network. The inclusion of the differentiable clustering component within the decoder was shown to enhance reconstruction accuracy. Crucially, the model guarantees that all reconstructed states remain within a polytope, with the corresponding latent variables directly interpreted as convex combination coefficients.

To evaluate the effectiveness of the constructed polytopes, polytope errors were computed. In addition, the influence of individual polytope vertices on the reconstruction process was investigated by analyzing the activation rates of the latent variables.

In the single-cylinder case, the proposed model achieved substantially higher reconstruction accuracy compared to POD. For the more complex double-cylinder scenario, although low training errors and minimal model mismatch were observed—as indicated by the approximation error within the polytope—the reconstruction quality declined in the extrapolation regime. Nonetheless, the polytope structure was found to remain well-formed across all evaluated regimes.

This suggests a promising avenue for future research by refining the model architecture or training strategy to better leverage its potential. Possible improvements include incorporating residuals into the loss function or extending the networks to handle previously unseen clusters more effectively.

CHAPTER 5

PCD-BASED PHYSICS-PRESERVING ROMS

Contents

5.1 Introduction 95
5.2 State Selection for PCD 97
5.3 Physics-preserving ROMs 98
5.3.1 ROM for Incompressible Fluid Flows 98
5.3.2 Low-dimensional Parametrization 101
5.4 Simulation Results 105
5.5 Conclusion 111

5.1 Introduction

MOR is a fundamental approach used to simplify high-dimensional dynamical systems by representing their states with a low-dimensional parametrization. This reduction significantly enhances computational efficiency and reduces memory usage making it feasible to implement complex systems.

However, this simplification often comes at the expense of reduced simulation accuracy and potential difficulties in preserving data properties such as sparsity, positivity, and adherence to fundamental physical laws. As a result, choosing an appropriate MOR technique and determining an optimal latent dimension are critical steps in achieving a well-balanced trade-off between computational efficiency and the accuracy of the ROM. The objective is to develop a method that preserves essential physical characteristics while significantly reducing computational complexity.

This study introduces PCD-based AEs for MOR, in which the decoder reconstructs states using a decoding matrix composed of actual states selected from a given dataset. Particular attention is given to linear ROM methods to enable efficient computational performance.

As discussed in Section 2.1.2, POD possesses several advantageous properties, such as the linearity and orthogonality of POD modes, as well as the fact that these modes are expressed as linear combinations of actual system states, allowing certain physical properties to be retained. Due to these characteristics, POD has been widely used to construct ROMs (ROMs); e.g., [62, 104]. However, despite its benefits, POD has inherent limitations in preserving crucial data properties, including positivity and sparsity, which can be essential for accurately capturing the underlying dynamics of complex systems.

In contrast, PCD ensures that the reduced representation retains a meaningful relationship with the original high-dimensional states, ensuring better interpretability and property preservation. Due to these advantages, PCD has been applied to a wide range of tasks across various domains; e.g., [81, 49]. However, despite its strengths in maintaining essential system properties, PCD typically results in higher approximation errors compared to POD, which can limit its effectiveness in scenarios where reconstruction error is a primary concern. To mitigate this issue, various methods have been developed for selecting columns and rows from a given matrix including the random selection and DEIM; see Section 2.1.2.

One possible improvement is to use genetic selection algorithms; e.g., [42]. Unlike other methods, particularly gradient-based algorithms, they are less likely to get stuck in local minima because they explore a large search space through crossovers and mutations across multiple generations. Additionally, they can be easily applied to a variety of optimization problems. However, their performance is sensitive to hyperparameters including the mutation rate and crossover probability, and they do not guarantee optimality. Moreover, computations with large populations and generations can be computationally expensive.

As an alternative approach, a method based on DEIM is proposed, incorporating data clustering techniques. In the context of snapshot data, especially from numerical simulations with very small time steps, temporally adjacent snapshots often exhibit similar characteristics and magnitudes. This inherent redundancy poses challenges for commonly used methods such as random selection and DEIM, which typically select columns and rows based on norm-based leverage scores. These methods tend to favor similar data entries, resulting in redundant selections that fail to capture the diversity of the dataset, ultimately degrading approximation quality.

To mitigate this issue, the proposed strategy assumes that excluding certain data points from the selection process can help prevent the choice of highly similar columns and rows. This work applies a clustering-based DEIM method for selecting PCD modes, which are then used to construct PCD-based ROMs.

In addition, AE architectures consisting of a nonlinear encoder and a PCD-based decoder are explored, where the decoding matrix is formed from selected states drawn directly from the dataset. To assess the effectiveness of these approaches, ROMs are constructed and applied to simulate the wake flow past a single cylinder governed by the incompressible Navier–Stokes equations given in (2.16). The evaluation emphasizes simulation error and the degree to which the reduced models maintain adherence to physical constraints, such as incompressibility.

Algorithm 5.1: CUR decomposition (genetic algorithm)

Input: Dataset $X \in \mathbb{R}^{n \times m}$, $k \leq min(n, m)$, population size z, generations N
Output: $C \in \mathbb{R}^{n \times k}$, $U \in \mathbb{R}^{k \times k}$, $R \in \mathbb{R}^{k \times m}$ ($X \approx CUR$, $U = C^{+}AR^{+}$)

```
Generate z pairs of (C, U, R) at random                    // Initial population
for i = 1 to N do
    Compute the approximation error of each (C, U, R)      // Fitness score
    Produce offspring from pairs based on the evaluation   // Crossover
    Implement random alteration with a low probability     // Mutation
    Update the pairs of (C, U, R)                          // New generation
return C, U, R
```

5.2 State Selection for PCD

In Section 2.1.2, the CUR decomposition was introduced alongside two selection algorithms presented in Algorithm 2.2 and Algorithm 2.4. The comparison results underscored the significance of the selection strategy in minimizing the approximation error associated with CUR decompositions. Moreover, the results suggest that additional error reduction may be achievable through more refined selection techniques. This section evaluates two advanced selection methods and compares their performance with the standard approaches discussed in Section 2.1.2.

The experiments utilize a genetic algorithm implemented in the Python library `deap`. Algorithm 5.1 outlines a genetic algorithm-based CUR decomposition of a given dataset X. The algorithm begins by generating an initial population of z random candidate solutions. For each generation, the approximation error of each triplet (C, U, R) is evaluated to determine its fitness. Offspring are then produced through crossover operations based on this evaluation. A mutation step is applied with low probability to maintain diversity in the population. The updated solutions form the new generation, and the process is repeated for N generations. Finally, the best-performing triplet is returned as the approximate CUR decomposition.

To address the tendency of DEIM to select highly similar states, particularly in temporally correlated datasets, we propose a modified selection strategy that combines DEIM with a clustering-based data pruning step. The dataset is first partitioned using k-means, and then a subset of representative states is retained from each cluster before applying DEIM. This approach increases the diversity of the selected columns and leads to improved approximation quality in the resulting CUR decomposition. As a result, a DEIM-based selection method incorporating data pruning is also employed, as outlined in Algorithm 5.2. The procedure begins by applying k-means clustering to divide the dataset X into k clusters. The method involves two hyperparameters: the number of clusters for k-means clustering, and a pruning ratio that controls the fraction of data points removed from each cluster. Both parameters are selected empirically to balance

Algorithm 5.2: CUR decomposition (DEIM-based selection with data pruning)

Input: Dataset $X \in \mathbb{R}^{m\times n}$, $r \leq \mathsf{min}(m,n)$
Output: $C \in \mathbb{R}^{m\times r}$, $U \in \mathbb{R}^{r\times r}$, $R \in \mathbb{R}^{r\times n}$ ($X \approx CUR$)

1 Implement Algorithm 2.6 `// k-means clustering`
2 Define $\hat{X} \subset X$ by eliminating certain data in each cluster and gathering the remaining data
3 Compute $\hat{X} \approx V\Sigma W^\top$ `// rank-r truncated SVD of X`
4 Compute $\mathbf{a} = \mathsf{DEIM}(W)$ `// select r column indices for C`
5 Compute $\mathbf{b} = \mathsf{DEIM}(V)$ `// select r row indices for R`
6 Define $C = \hat{X}[:, \mathbf{a}]$ and $R = \hat{X}[\mathbf{b}, :]$
7 Compute $U = C^+ X R^+$ `// +: Moore-Penrose pseudoinverse`
8 **return** C, U, R

representativeness and diversity in the selected subset. Then a pruned subset $\hat{X} \subset X$ is constructed by discarding a portion of the data within each cluster, thereby promoting diversity and reducing similarities among the selected samples. A truncated SVD is subsequently performed on $\hat{X}$ and then the DEIM-based selection method (Algorithm 2.3) selects r rows and r columns.

For PCD-based ROMs, only r columns are required to construct the PCD modes that serve as the basis for state reconstruction; i.e., selection of r rows is not necessary.

5.3 Physics-preserving ROMs

This section presents the derivation process for physics-preserving ROMs of the incompressible Navier-Stokes equations, constructed using POD and PCD-based AEs.

5.3.1 ROM for Incompressible Fluid Flows

For physics-preserving ROMs, consider the semi-discrete model (2.16) of incompressible Navier-Stokes equations,

$$\mathsf{M}\dot{\mathbf{x}}(t) + \mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + \mathsf{A}\mathbf{x}(t) - \mathsf{J}^\top \mathbf{p}(t) - \mathbf{f}(t) = \mathbf{0}, \tag{5.1a}$$

$$\mathsf{J}\mathbf{x}(t) = \mathbf{0}, \tag{5.1b}$$

where $\mathbf{p}(t) \in \mathbb{R}^p$ and $\mathbf{x}(t) \in \mathbb{R}^n$ denote the states of the velocity and the pressure at time t respectively, and where $\mathsf{M} \in \mathbb{R}^{n\times n}$, $\mathsf{N}(\cdot) \in \mathbb{R}^{n\times n}$, $\mathsf{A} \in \mathbb{R}^{n\times n}$, $\mathsf{J} \in \mathbb{R}^{p\times n}$, and $\mathbf{f}(t) \in \mathbb{R}^n$ are the mass, linear state-dependent coefficient convection, diffusion, discrete divergence matrices, and the forcing term at time t respectively.

For physics-preserving models, it is assumed that there exists a matrix

$$D = \begin{bmatrix} | & | & & | & | \\ \mathbf{d}_1 & \mathbf{d}_2 & \cdots & \mathbf{d}_{r-1} & \mathbf{d}_r \\ | & | & & | & | \end{bmatrix} \in \mathbb{R}^{n \times r},$$

such that $\mathsf{J}D = O$. Hence, the pressure term is eliminated by multiplying $D^\top$ on both sides of (5.1a) ($\because D^\top \mathsf{J}^\top = O$) so that a simplified system of (5.1) is obtained as follows:

$$D^\top \mathsf{M}\dot{\mathbf{x}}(t) + D^\top \mathsf{N}(\mathbf{x}(t))\mathbf{x}(t) + D^\top \mathsf{A}\mathbf{x}(t) - D^\top \mathbf{f}(t) = \mathbf{0}, \tag{5.2a}$$

$$\mathsf{J}\mathbf{x}(t) = \mathbf{0}. \tag{5.2b}$$

To address the high computational costs for the system (5.2), a MOR approach (e.g., POD) is defined a model consisting of an encoder

$$\mu : \mathbb{R}^n \to \mathbb{R}^r,\ \mathbf{x} \mapsto \mathbf{z} \coloneqq [z_1\ z_2\ \cdots\ z_r]^\top$$

and a linear decoder

$$\varphi : \mathbb{R}^r \to \mathbb{R}^n,\ \mathbf{z} \mapsto \tilde{\mathbf{x}}$$

represented by the matrix D. Then the predefined relationship

$$\mathbf{x}(t) \approx \tilde{\mathbf{x}}(t) = D\mathbf{z}(t)$$

is used to build a ROM where $\mathbf{x}(t) \in \mathbb{R}^n$, $\tilde{\mathbf{x}}(t) \in \mathbb{R}^n$, and $\mathbf{z}(t) = \mu(\mathbf{x}(t)) \in \mathbb{R}^r$ are the state, reconstructed state, and latent state at time t respectively.

Suppose that there exists a sufficiently small $\varepsilon > 0$ such that $\|\mathbf{x}(t) - \tilde{\mathbf{x}}(t)\|_{\mathsf{M}} < \varepsilon$ for $t > 0$ where $\|\mathbf{a}\|_{\mathsf{M}} = \sqrt{\mathbf{a}^\top \mathsf{M}\mathbf{a}}$ for all $\mathbf{a} \in \mathbb{R}^n$.

Then, by replacing $\mathbf{x}(t)$ with $D\mathbf{z}(t)$, the system is reformulated as

$$D^\top \mathsf{M}D\dot{\mathbf{z}}(t) + D^\top \mathsf{N}(D\mathbf{z}(t))D\mathbf{z}(t) + D^\top \mathsf{A}D\mathbf{z}(t) - D^\top \mathbf{f}(t) = \mathbf{0}, \tag{5.3a}$$

$$\mathsf{J}D\mathbf{z}(t) = \mathbf{0}. \tag{5.3b}$$

Since $\mathsf{J}D = O$, the incompressibility condition from (5.3) is automatically satisfied and can therefore be omitted. As a result, the reduced system simplifies to

$$D^\top \mathsf{M}D\dot{\mathbf{z}}(t) + D^\top \mathsf{N}(D\mathbf{z}(t))D\mathbf{z}(t) + D^\top \mathsf{A}D\mathbf{z}(t) - D^\top \mathbf{f}(t) = \mathbf{0},$$

That is, the reduced-order model of (5.2) becomes a pressure-free system in which incompressibility is no longer explicitly enforced.

Due to the linearity of $\mathsf{N}(\cdot)$,

$$\begin{aligned} D^\top \mathsf{N}(D\mathbf{z}(t))D &= D^\top \mathsf{N}(\sum_{i=1}^{r} z(t)_i \mathbf{d}_i)D \\ &= D^\top \sum_{i=1}^{r} z(t)_i \mathsf{N}(\mathbf{d}_i)D \\ &= \sum_{i=1}^{r} z(t)_i D^\top \mathsf{N}(\mathbf{d}_i)D \\ &= \sum_{i=1}^{r} z(t)_i \mathsf{N}_i \end{aligned}$$

where $\mathsf{N}_i = D^\top \mathsf{N}(\mathbf{d}_i)D$, $z(t)_i$ is the i-th element of $\mathbf{z}(t)$, and $\mathbf{d}_i$ is the i-th column vector of D, $i = 1, 2, \cdots r$ $(r \ll n)$.

Finally, a ROM of the full-order model (FOM) (5.2) is constructed as

$$\dot{\mathbf{z}}(t) + \hat{\mathsf{N}}(\mathbf{z}(t))\mathbf{z}(t) + \hat{\mathsf{A}}\mathbf{z}(t) - \hat{\mathbf{f}}(t) = \mathbf{0}, \tag{5.5}$$

where $\hat{\mathsf{M}} = D^\top \mathsf{M} D \in \mathbb{R}^{r\times r}$, $\hat{\mathsf{N}}(\mathbf{z}(t)) = \sum_{i=1}^{r} z(t)_i \hat{\mathsf{M}}^{-1}\mathsf{N}_i \in \mathbb{R}^{r\times r}$, $\hat{\mathsf{A}} = \hat{\mathsf{M}}^{-1}D^\top \mathsf{A} D \in \mathbb{R}^{r\times r}$, and $\hat{\mathbf{f}}(t) = \hat{\mathsf{M}}^{-1}D^\top \mathbf{f}(t) \in \mathbb{R}^r$.

To employ the time evolution of Eq. (5.5), it is transformed into the form of an ODE as

$$\dot{\mathbf{z}}(t) = -(\hat{\mathsf{N}}(\mathbf{z}(t)) + \hat{\mathsf{A}})\mathbf{z}(t) + \hat{\mathbf{f}}(t). \tag{5.6}$$

In case that the decoder φ is affine linear defined as $\mathbf{x}(t) \approx \tilde{\mathbf{x}}(t) = D\mathbf{z}(t) + \mathbf{d}_0$ where $\mathbf{d}_0$ is a $n \times 1$ vector that satisfies $\mathsf{J}\mathbf{d}_0 = \mathbf{0}$, a ROM can also be derived in the form of the ODE formulation

$$\dot{\mathbf{z}}(t) = -(\hat{\mathsf{N}}(\mathbf{z}(t)) + \hat{\mathsf{A}})\mathbf{z}(t) + \hat{\mathbf{f}}(t), \tag{5.7}$$

where $\hat{\mathsf{M}} = D^\top \mathsf{M} D \in \mathbb{R}^{r\times r}$, $\hat{\mathsf{N}}(\mathbf{z}(t)) = \sum_{i=1}^{r} z(t)_i \hat{\mathsf{M}}^{-1}\mathsf{N}_i \in \mathbb{R}^{r\times r}$, $\hat{\mathsf{A}} = \hat{\mathsf{M}}^{-1}D^\top(\mathsf{A}D + \mathsf{N}(\mathbf{d}_0)D + \hat{\mathsf{N}}_0) \in \mathbb{R}^{r\times r}$, $\hat{\mathbf{f}}(t) = \hat{\mathsf{M}}^{-1}D^\top(\mathbf{f}(t) - \mathsf{A}\mathbf{d}_0 - \mathsf{N}(\mathbf{d}_0)\mathbf{d}_0) \in \mathbb{R}^r$, and

$$\hat{\mathsf{N}}_0 = \begin{bmatrix} | & | & & | \\ \mathsf{N}(\mathbf{d}_1)\mathbf{d}_0 & \mathsf{N}(\mathbf{d}_2)\mathbf{d}_0 & \cdots & \mathsf{N}(\mathbf{d}_r)\mathbf{d}_0 \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n\times r}.$$

Consequently, the computational cost of the simulation can be reduced by implementing the proposed ROMs. Specifically, as a metric for computational efficiency, the number of floating point operations (FLOPs) can be quantified by the function r;

$$\mathrm{FLOPs}(r) = 2r^3 + 2r^2.$$

5.3.2 Low-dimensional Parametrization

Remark 5.1 states that the proposed ROMs guarantee the incompressibility (divergence free) for any latent states.

Remark 5.1 (Incompressibility in ROMs):
Since $\mathsf{J}D = O$ and $\mathsf{J}\mathbf{d}_0 = \mathbf{0}$, it follows that

$$\mathsf{J}\mathbf{x}(t) \approx \mathsf{J}(D\mathbf{z}(t) + \mathbf{d}_0) = (\mathsf{J}D)\mathbf{z}(t) + \mathsf{J}\mathbf{d}_0 = \mathbf{0}. \qquad \Diamond$$

This result provides theoretical evidence for the use of the developed ROMs governed by the incompressible Navier-Stokes equations.

To define the matrix D and bias $\mathbf{d}_0$, consider first the SVD of X, $X = V\Sigma W^\top$ introduced in Section 2.1.1 where

$$X = \begin{bmatrix} | & | & & | & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} & \mathbf{x}_m \\ | & | & & | & | \end{bmatrix} \in \mathbb{R}^{n\times m},$$

$$V = \begin{bmatrix} | & | & & | & | \\ \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_{m-1} & \mathbf{v}_m \\ | & | & & | & | \end{bmatrix} \in \mathbb{R}^{n\times n},$$

$$W = \begin{bmatrix} | & | & & | & | \\ \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_{n-1} & \mathbf{w}_n \\ | & | & & | & | \end{bmatrix} \in \mathbb{R}^{m\times m},$$

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_m \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{n\times m},$$

and $n > m$. Since $W^\top W = I$,

$$XW = V\Sigma.$$

Then

$$\sum_{j=1}^{m} (\mathbf{w}_i)_j \mathbf{x}_j = \sigma_i \mathbf{v}_i, i \in \{1, 2, \cdots, n\}, j \in \{1, 2, \cdots, m\}.$$

As a result, every POD mode $\mathbf{v}_i$ can be presented as a linear combination of $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$,

$$\mathbf{v}_i = \sum_{j=1}^{m} c_{ij}\mathbf{x}_j,$$

where $c_{ij} = \frac{1}{\sigma_i}(\mathbf{w}_i)_j$, $i = 1, 2, \cdots, n$, and $j = 1, 2, \cdots, m$. Since $\mathsf{J}\mathbf{x}_j = \mathbf{0}$ for every j,

$$\mathsf{J}\mathbf{v}_i = \mathsf{J}(\sum_{j=1}^{m} c_{ij}\mathbf{x}_j) = \sum_{j=1}^{m} c_{ij}\mathsf{J}\mathbf{x}_j = \mathbf{0}.$$

Thus,

$$\mathsf{J}V = O.$$

In CUR decompositions, every column of C is selected from the actual states $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ such that the columns themselves satisfy the incompressibility condition (5.1b). Consequently, it leads to the condition

$$\mathsf{J}C = O.$$

Therefore, both matrices V and C respectively are used as a decoder matrix D to implement ROMs (5.7). For $\mathbf{d}_0$, an actual state or a linear combination of $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$ can be used such as the initial velocity state and the mean of $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$.

In the POD-based ROM, the decoding matrix D and bias $\mathbf{d}_0$ are defined as

$$D = V \text{ and } \mathbf{d}_0 = \bar{\mathbf{x}},$$

where V is a $n \times r$ POD basis obtained by

$$\text{a truncated SVD of } \bar{X} = \{\mathbf{x}_1 - \bar{\mathbf{x}}, \mathbf{x}_2 - \bar{\mathbf{x}}, \cdots, \mathbf{x}_m - \bar{\mathbf{x}}\}$$

and

$$\bar{\mathbf{x}} = \frac{\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_m}{m}.$$

In other words, the latent state $\mathbf{z}$ is defined as

$$\mathbf{z}(t) = V^\top(\mathbf{x}(t) - \bar{\mathbf{x}})$$

and the reconstructed state $\tilde{\mathbf{x}}$ is obtained by

$$\tilde{\mathbf{x}}(t) = V\mathbf{z}(t) + \bar{\mathbf{x}} \text{ for time } t.$$

As the standard PCD-based ROM, the CUR-DEIM-DP of $\bar{X}$ is employed so that the latent state $\mathbf{z}$ is defined as

$$\mathbf{z}(t) = C^+(\mathbf{x}(t) - \bar{\mathbf{x}})$$

and the reconstructed state $\tilde{\mathbf{x}}$ is obtained by

$$\tilde{\mathbf{x}}(t) = C\mathbf{z}(t) + \bar{\mathbf{x}} \text{ for time } t,$$

Algorithm 5.3: Training a PCD-AE

Input: Dataset X, hyperparameters; e.g., r, the number of epochs N
Output: Pretrained encoding parameters; θ_μ

Generate a $n \times r$ decoding matrix C using a CUR decomposition of the dataset $\bar{X}$
Solve a least square problem $\min_{\mathbf{z}} \|\mathbf{x} - \bar{\mathbf{x}} - C\mathbf{z}\|_{\mathsf{M}}$ to collect the optimal latent states $\mathbf{z}^*$
Define a training dataset $\{(\mathbf{x}_1, \mathbf{z}_1^*), (\mathbf{x}_2, \mathbf{z}_2^*), \cdots, (\mathbf{x}_m, \mathbf{z}_m^*)\}$
for $e = 1, \cdots, N$ **do**
    for *random batch:* $i \in B$ **do**
        Compute $\mathcal{L}_{\mathsf{rec}} = \frac{1}{\#B} \sum_{i \in B} \|\mathbf{z}_i^* - \mu(\mathbf{x}_i - \bar{\mathbf{x}})\|_2$
        Optimize encoding parameters θ_μ
return θ_μ

where C is a $n \times r$ matrix containing r vectors selected from $\bar{X}$. Then, D and $\mathbf{d}_0$ are defined as C and $\bar{\mathbf{x}}$ respectively.

It is known that the approximation error of PCD is generally higher than that of POD. To improve approximation accuracy, two nonlinear encoders are developed and their influence on simulation error in ROMs is investigated. The decoder remains affine, as defined by the PCD framework, in order to preserve the linearity of the decoding process and ensure compatibility with the ROM formulation presented in Section 5.3.

The nonlinear convolutional encoder μ, introduced in Section 3.2.3, is modified to the form

$$\mathbf{z}(t) = \mu(\mathbf{x}_{\mathsf{CNN}}(t) - \bar{\mathbf{x}}).$$

where $\mathbf{z}(t)$ is the $r \times 1$ latent state at time t. In this architecture, no activation function is applied in the last layer so the codomain of μ is unbounded. As a result, D and $\mathbf{d}_0$ are defined as C and $\bar{\mathbf{x}}$ respectively. For convenience, this model is referred to as PCD-AE.

For training PCD-AE models, the CUR-DEIM-DP of $\bar{X}$ is implemented to obtain C, and C is used for the decoding matrix. Here, the decoder does not need to be included in the training procedure since C is immutable. Consequently, the training time can be reduced by excluding the decoding part from the training iterations.

In Algorithm 5.3, the optimal latent states $\mathbf{z}_1^*, \mathbf{z}_2^*, \cdots, \mathbf{z}_m^*$ are obtained by solving a least square problem of the matrix equation $\mathbf{x}_i - \bar{\mathbf{x}} = C\mathbf{z}_i$ for each state $\mathbf{x}_i, i = 1, 2, \cdots, m$. The loss function is then defined as

$$\mathcal{L}_{\mathsf{rec}} = \frac{1}{\#B} \sum_{i \in B} \|\mathbf{z}_i^* - \mu(\mathbf{x}_i - \bar{\mathbf{x}})\|_2,$$

where B represents a batch of randomly selected indices. During iterative training, The Adam optimizer updates the model parameters θ_μ based on the gradient information on $\mathcal{L}_{\mathsf{rec}}$.

Second, to design a PAE, the nonlinear convolutional encoder is built as follows:

$$\mathbf{z}^{+1}(t) = \mu(\mathbf{x}_{\mathsf{CNN}}(t))$$

with the modified softmax function (4.8) in the last layer ensuring that

$$\mathbf{z}^{+1}(t) = [z_1(t), z_2(t), \cdots, z_{r+1}(t)]^\top$$

serves as the coefficients of convex combinations for state reconstruction. As mentioned above, $(r+1) \times 1$ latent states are considered to leverage the advantage of the polytopic representation mentioned in Remark 4.10, and eliminate one degree of freedom as follows:

$$\begin{aligned}
\tilde{\mathbf{x}}(t) = C\mathbf{z}^{+1}(t) &= \sum_{i=1}^{r+1} z_i(t)\mathbf{c}_i \\
&= \sum_{i=1}^{r} z_i(t)\mathbf{c}_i + (1 - z_1(t) - z_2(t) - \cdots - z_r(t))\mathbf{c}_{r+1} \\
&= \mathbf{c}_{r+1} + \sum_{i=1}^{r} z_i(t)(\mathbf{c}_i - \mathbf{c}_{r+1}) \\
&= \mathbf{c}_{r+1} + \bar{C}\mathbf{z}
\end{aligned}$$

where $\mathbf{z}(t) = [z_1(t), z_2(t), \cdots, z_r(t)]^\top$, $\mathbf{c}_i$ denotes the i-th column vector of C, and

$$\bar{C} = \begin{bmatrix} | & | & & | \\ \mathbf{c}_1 - \mathbf{c}_{r+1} & \mathbf{c}_2 - \mathbf{c}_{r+1} & \cdots & \mathbf{c}_r - \mathbf{c}_{r+1} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{m \times r}.$$

Accordingly, the decoder is defined using the selected components as $D = \bar{C}$ and $\mathbf{d}_0 = \mathbf{c}_{r+1}$ for constructing ROMs. This model is referred to as PCD-PAE.

Since DEIM-based selection methods could not be suitable to handle the convex combination constraints

$$z_i \geq 0 \text{ and } \sum_{i=1}^{r+1} z_i = 1, \tag{5.9}$$

CUR-RANDOM is employed 100 times iteratively after the data pruning (line 1,2 in Algorithm 5.2). During the iterations, the best matrix C is chosen based on the relative polytope error

$$\sum_{i=1}^{m'} \frac{\|\mathbf{x}_i - C\mathbf{z}_i^{+1*}\|_{\mathsf{M}}}{\|\mathbf{x}_i\|_{\mathsf{M}}},$$

where m' is the number of data selected by the data pruning method, and $\mathbf{z}_i^{+1*}$ is the optimal solution of the quadratic convex optimization problem of $\mathbf{x}_i = C\mathbf{z}_i^{+1*}$ with the constraints (5.9).

Algorithm 5.4: Training a PCD-PAE

Input: Dataset X, hyperparameters; e.g., r, the number of epochs N
Output: Pretrained encoding parameters; θ_μ

Generate an $n \times (r+1)$ decoding matrix C using a CUR decomposition of X
Solve a convex quadratic optimization problem $\min_{\mathbf{z}} \|\mathbf{x} - C\mathbf{z}^{+1}\|_{\mathsf{M}}$ to collect the optimal latent states $\mathbf{z}^*$
Define a training dataset $\{(\mathbf{x}_1, \mathbf{z}_1^{+1*}), (\mathbf{x}_2, \mathbf{z}_2^{+1*}), \cdots, (\mathbf{x}_m, \mathbf{z}_m^{+1*})\}$
for $e = 1, \cdots, N$ **do**
    for *random batch:* $i \in B$ **do**
        Compute $\mathcal{L}_{\mathsf{rec}} = \frac{1}{\#B} \sum_{i \in B} \|\mathbf{z}_i^{+1*} - \mu(\mathbf{x}_i)\|_2$
        Optimize encoding parameters θ_μ
return θ_μ

In Algorithm 5.4, similar to training PCD-AE models, the decoding part is not included in the training process for PCD-PAE. To obtain the optimal latent states $\mathbf{z}_1^{+1*}, \mathbf{z}_2^{+1*}, \cdots, \mathbf{z}_m^{+1*}$, a convex quadratic optimization problem is solved for the matrix equation

$$\mathbf{x} = C\mathbf{z}^{+1}$$

subject to the constraints (5.9) for every $\mathbf{x}$. Then the loss function is defined as

$$\mathcal{L}_{\mathsf{rec}} = \frac{1}{|B|} \sum_{i \in B} \|\mathbf{z}_i^{+1*} - \mu(\mathbf{x}_i)\|_2.$$

Then the Adam optimizer updates encoding parameters θ_μ. In practical implementation, the optimal latent states are obtained using Python's least squares solver `lstsq` and the quadratic programming solver `cvxopt`.

5.4 Simulation Results

The single-cylinder case (Section 2.4.1) is considered and the approximation errors of CUR decompositions and truncated SVDs are evaluated. Furthermore, reconstruction errors are examined for four low-dimensional parametrization methods, POD, PCD, PCD-AE, and PCD-PAE, and the simulation errors of their corresponding ROM) are assessed.

Figure 5.1 compares the relative approximation error

$$\frac{\|X - \tilde{X}\|_{\mathsf{F}}}{\|X\|_{\mathsf{F}}}$$

for five different dimensionality reduction methods across various reduced dimensions, $r = 2, 4, 8, 12, 16, 32, 64$. As established in Lemma 2.1, the truncated SVD outperforms the CUR decompositions.

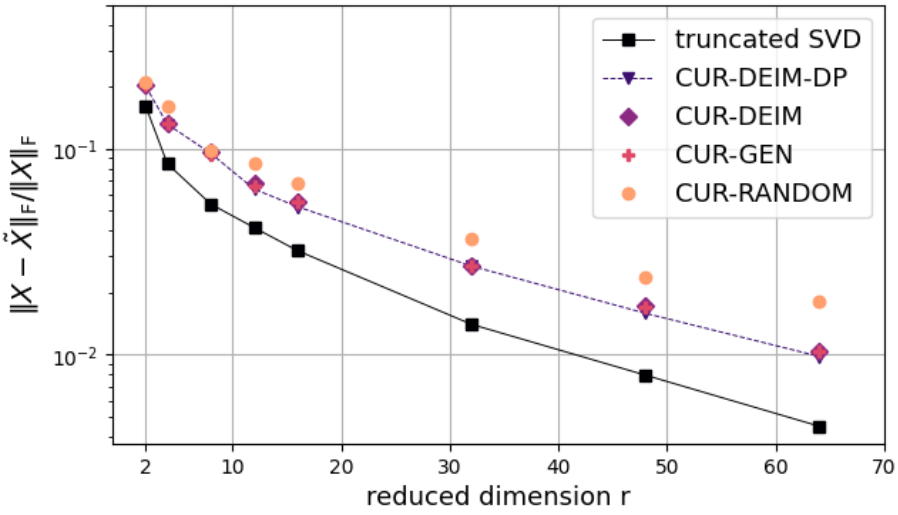


Figure 5.1: Approximation error: truncated SVD and CUR decompositions

CUR-DEIM yields lower approximation errors than CUR-RANDOM like the results observed for the two real-world datasets presented in Figure 2.1.

For CUR-DEIM-DP, the number of clusters is set to 50, and a pruning ratio of 0.3 is applied. While CUR-DEIM-DP exhibits a higher approximation error than the truncated SVD, it outperforms the other CUR-based methods. Specifically, CUR-DEIM-DP improves the approximation errors by $0.4\% \sim 6.7\%$ compared to CUR-DEIM.

For the implementation of CUR-GEN, the population size and the number of generations are set to 20 and 30 respectively with a crossover probability of 0.3 and a mutation rate of 0.5. Under this configuration, CUR-GEN achieves a performance improvement of $0\% \sim 3.77\%$ over CUR-DEIM. In constrast, CUR-DEIM-DP outperforms CUR-GEN by $0.4\% \sim 6.7\%$ at most reduced dimensions, except at $r = 8$, where CUR-DEIM-DP results in a 1.6% higher error compared to CUR-GEN. However, compared to the DEIM-based methods, CUR-GEN not only incurs higher computational costs but also depends on several critical hyperparameters that require careful tuning.

Figure 5.2 shows a comparison of the reconstruction errors of four low-dimensional parametrization methods for varying reduced dimensions, $r = 2, 4, 8, 12, 16, 32, 48, 64$. Regarding the training errors, the performance of all methods generally improves as the reduced dimension r increases. Among them, POD consistently achieves the lowest reconstruction error in both the training and test ranges, $[0, 10]$ and $[10, 16]$.

PCD-AE yields reconstruction errors comparable to those of PCD for $r \leq 32$. However, at higher dimensions, $r = 48, 64$, PCD-AE appears to be insufficiently trained, despite using the same decoder as PCD.

In contrast to the results observed in Figure 4.5, PCD-PAE exhibits the highest reconstruction errors. This discrepancy suggests that PCD modes depend on negative coefficients to accurately reconstruct states, a property that is inherently restricted in

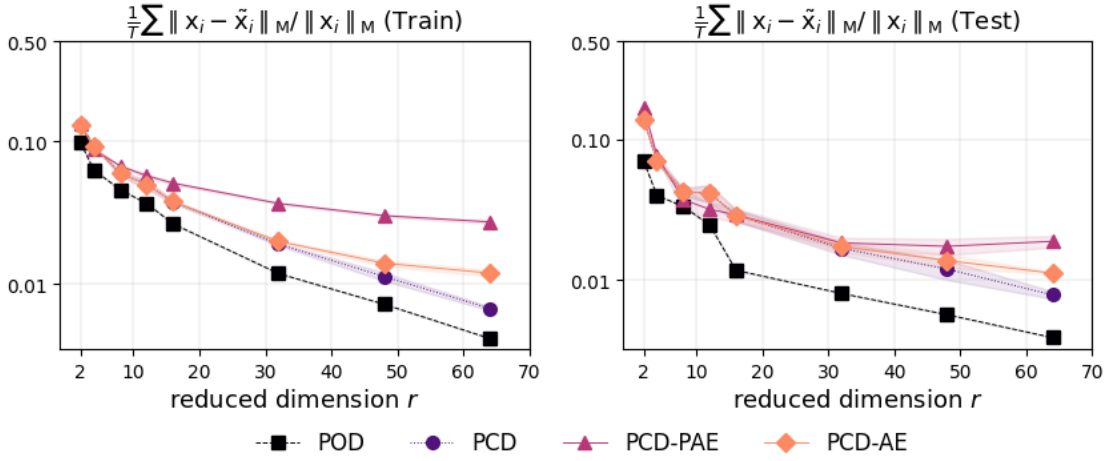


Figure 5.2: Reconstruction error as a function of the reduced dimension r, averaged over 5 runs for the single-cylinder case. The shaded regions show uncertainty from multiple runs.

PAEs due to the convex combination constraints (5.9).

The small standard deviations indicate that the variability in training results is low. For the reconstruction of periodic flows in the test range, all the models achieve error levels comparable to their respective training errors.

A pretrained model is selected for each ROM based on the lowest training reconstruction error specific to each model reported in Figure 5.2. In our simulations, PCD and PCD-AE share the same decoding matrix C, and therefore their ROMs produce identical results regardless of their encoding architectures.

Figure 5.3 presents the modes obtained from each method. Since the centered data $\mathbf{x}-\bar{\mathbf{x}}$ are used to obtain POD, PCD, and PCD-AE modes, the magnitudes of these modes are smaller than those of the PCD-PAE modes.

To satisfy the initial condition of the full order model (2.18) in PCD-based models, the first column vector $\mathbf{c}_1$ of C is defined as a vector corresponding to the initial state $\mathbf{x}_1$ as illustrated in the first mode for PCD-PAE in Figure 5.3. In PCD-PAE, the first latent state $\mathbf{z}_1$ is the first unit vector $[1, 0, \cdots, 0]$; i.e.,

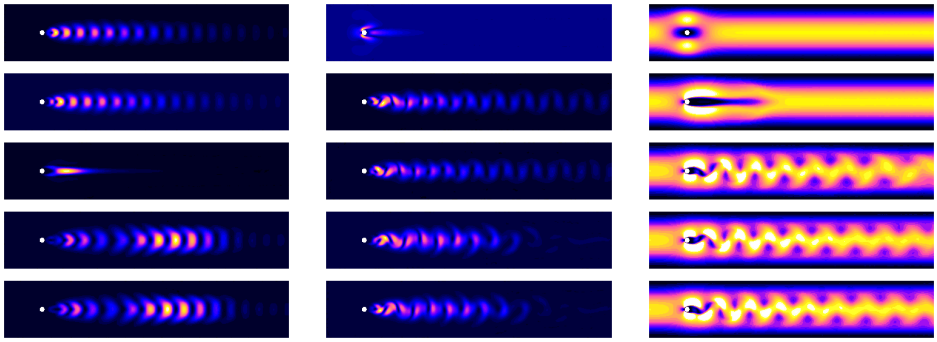
$$\mathbf{x}_1 = \mathbf{c}_1 = C\mathbf{z}_1.$$

In PCD and PCD-AE,

$$\mathbf{x}_1 - \bar{\mathbf{x}} = \mathbf{c}_1 = C\mathbf{z}_1,$$

where $\mathbf{z}_1$ is the first unit vector $[1, 0, \cdots, 0]$.

Remark 5.2:
Let $\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_r$ be PCD modes obtained by a CUR decomposition of the centered data $\mathbf{x} - \bar{\mathbf{x}}$. Then $\mathbf{c}_i + \bar{\mathbf{x}} \in X, i - 1, 2, \cdots, r$. ◊



(a) POD (b) PCD/PCD-AE (c) PCD-PAE

Figure 5.3: The first 5 modes out of 64 for each models when $r = 64$: the POD basis is obtained by applying truncated SVD to the centered data $\mathbf{x} - \bar{\mathbf{x}}$, the PCD-AE modes are identical to the PCD modes which are derived via CUR decomposition of the centered data $\mathbf{x} - \bar{\mathbf{x}}$, and PCD-PAE modes are actual states selected from the data $\mathbf{x}$.

Figure 5.4 illustrates the relative trajectory reconstruction error defined as

$$\frac{\|\mathbf{x} - \mathbf{x}_{ROM}\|_{\mathsf{M}}}{\|\mathbf{x}\|_{\mathsf{M}}},$$

plotted over time for four ROMs: POD, PCD, PCD-AE, and PCD-PAE. $\mathbf{x}_{ROM}$ represents the reconstructed state obtained by linearly projecting the latent state $\mathbf{z}$ during the numerical solution of the reduced dimensional system (5.7). During the transient phase from relatively stable flows to periodic flows at around $t = 5$, all ROMs exhibit an increase in reconstruction error. After then, the POD trajectory stabilizes around a higher error plateau of approximately 10^{-1} indicating its limitations in capturing nonlinear dynamics. In contrast, the PCD-based models demonstrate superior performance in the range of periodic flows. All the models continue to plateau at suboptimal performance indicating the robustness of the extrapolation in the time range of $[10, 16]$.

Compared to Figure 5.2, these results demonstrate that PCD-based modes lead to better reconstruction in the ROM simulation, indicating that the approximation errors of the dimensionality reduction methods are not necessarily proportional to the resulting simulation errors.

Table 5.1 presents a quantitative comparison of the ROMs using the interpolation error, extrapolation error, and divergence error at $r = 64$. The interpolation error, measured over the time domain $[0, 10]$, reflects how well the ROMs reconstruct trajectories within the time span of the training data. Among the models, PCD/PCD-AE achieves the lowest interpolation error at 0.0908. Additionally, the PCD-based models exhibit significantly lower errors compared to POD, highlighting their generalization performance beyond the training time domain. In terms of the divergence error, FOM, as expected, exhibits almost

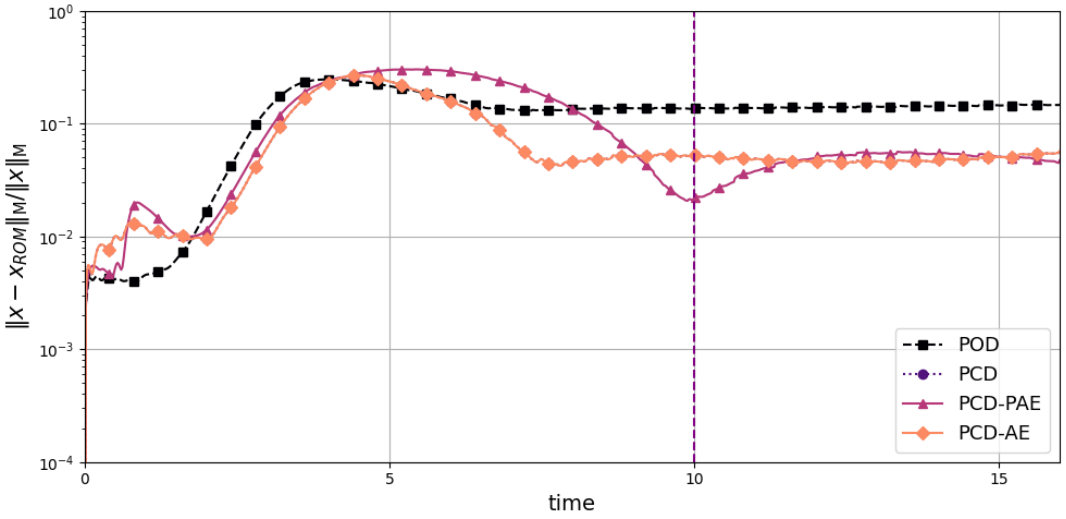


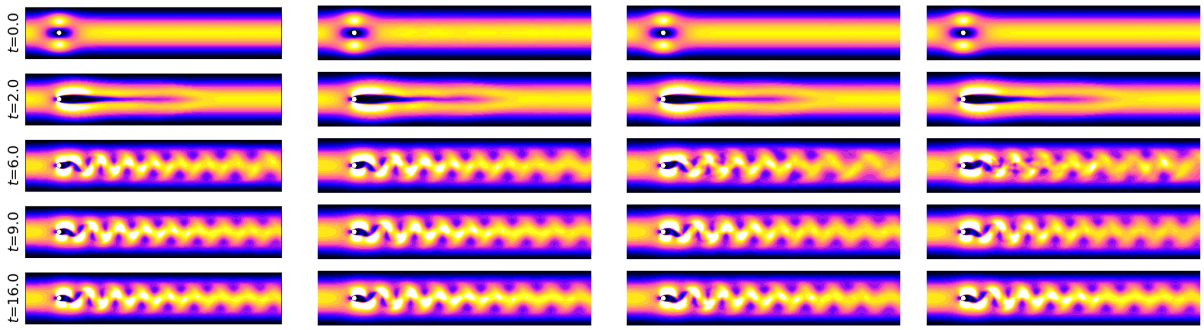
Figure 5.4: Comparison of relative trajectory reconstruction error across ROMs when $r = 64$: the dashed line distinguishes between the training phase and the extrapolation phase for the single-cylinder case.

Metric	FOM	POD	PCD/PCD-AE	PCD-PAE
Interpolation error	-	0.1257	**0.0908**	0.1365
Extrapolation error	-	0.1415	0.0485	**0.0472**
Divergence error	5.51×10^{-15}	8.40×10^{-7}	$\mathbf{1.31 \times 10^{-7}}$	2.90×10^{-7}

Table 5.1: ROMs for $r = 64$: the interpolation error is defined as the mean reconstruction error over the time domain $[0, 10]$. The extrapolation error is the mean reconstruction error over the time domain $[10, 16]$. The divergence error is the mean of $\|\mathbf{J}\mathbf{x}\|_2$ for all $\mathbf{x}$.

divergence free behavior. Compared to POD, the divergence error of PCD/PCD-AE is approximately 6.41 times smaller, and that of PCD-PAE is about 2.90 times smaller. Overall, PCD/PCD-AE demonstrates superior performance in both state reconstruction accuracy and adherence to the divergence free condition.

Figure 5.5 presents a qualitative comparison between FOM and the reduced-order approximations obtained from POD, PCD/PCD-AE, and PCD-PAE at selected times $t = 0, 2, 6, 9, 16$ with a reduced dimension $r = 64$. The FOM results serve as the reference snapshots capturing the detailed spatiotemporal evolution of the flow field. The POD-based reconstruction reproduces the large-scale velocity snapshots but exhibits visible smoothing and loss of finer-scale structures, particularly in the wake region at later times. In contrast, the PCD-based models improve the reconstruction of vortex dynamics. These results highlight the superior representational capability of PCD-based models in capturing periodic vortex behaviors within reduced-order frameworks.



(a) FOM (b) POD (c) PCD/PCD-AE (d) PCD-PAE

Figure 5.5: Comparison between the reference produced by FOM and the snapshots generated by POD, PCD/PCD-AE, and PAE at t = 0,2,6,9,16, $r = 64$

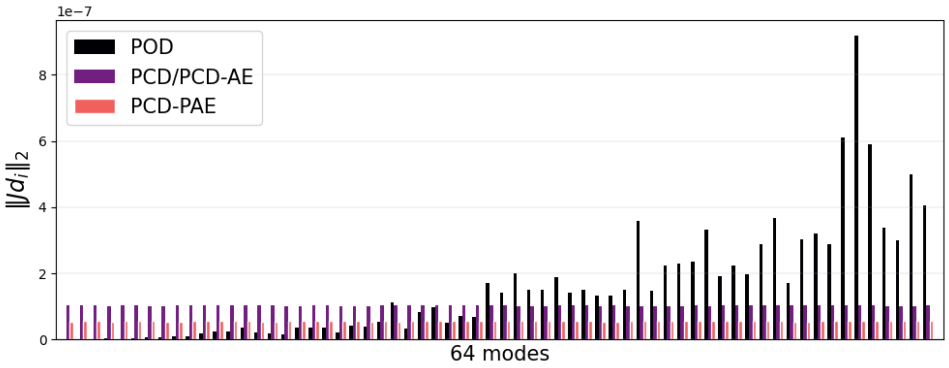


Figure 5.6: Incompressibility: averaged L_2 error of the divergence of each mode $\mathbf{d}_i$ at $r = 64$

In Figure 5.6, the averaged divergence error of each mode $\mathbf{d}_i$, $i = 1, 2, \cdots, 64$ is measured to assess the incompressibility preservation in the modes when $r = 64$. Approximately the first half of the POD modes exhibit very low divergence errors; however, the divergence increases significantly for higher-index modes. This behavior indicates that the low-energy POD modes struggle to satisfy the incompressibility constraint, particularly in regions associated with finer-scale flow structures. In contrast, the PCD-based methods maintain consistently lower divergence errors across all modes. Among them, the PCD-PAE approach achieves the lowest error magnitudes effectively preserving the divergence-free condition throughout the entire reduced basis.

Figure 5.7 shows the relationship between the divergence error and the corresponding singular values of the POD modes. The results indicate a clear trend in which modes associated with smaller singular values tend to exhibit significantly larger divergence errors. This inverse correlation highlights a key limitation of standard POD-based

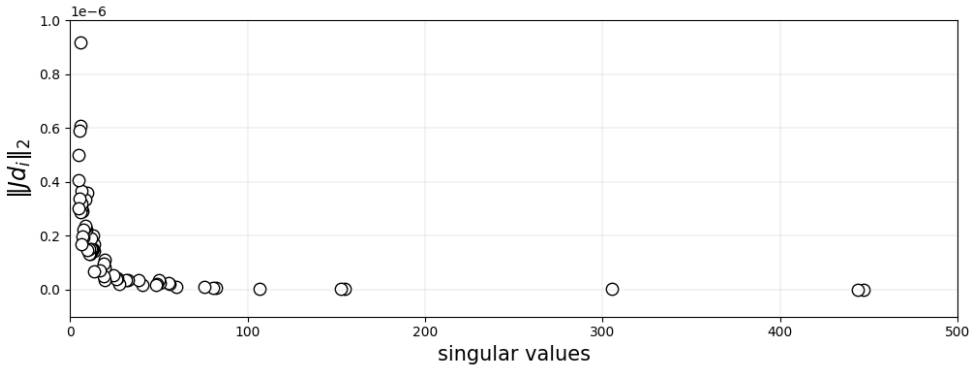


Figure 5.7: Averaged L_2 error of the divergence vs. singular value for each POD mode $\mathbf{d}_i$

methods. While higher-energy modes are more physically consistent, lower-energy modes may compromise the physical fidelity of the reduced-order representation. These findings emphasize the importance of incorporating physics-based constraints, particularly when retaining lower-energy modes.

5.5 Conclusion

This section presents the development of physics-preserving ROMs based on the POD and PCD frameworks. The results indicate that PCD-based models achieve reconstruction errors comparable to those of POD in reduced-order simulations. Notably, PCD-based models exhibit superior performance in capturing periodic vortex-dominated flows.

Although all methods theoretically satisfy the divergence-free condition, numerical divergence errors are inevitably introduced during reconstruction. Among the considered approaches, the PCD-based models exhibit lower average divergence errors compared to POD. In particular, the higher-index POD modes tend to produce larger divergence errors, while the PCD modes consistently maintain lower and more uniform divergence levels across the basis.

Our simulations revealed that the accuracy of CUR decompositions is highly sensitive to the selection strategies used for determining the matrices C and R. To address this issue, a DEIM-based selection method incorporating data pruning was proposed, resulting in improved approximation accuracy by increasing the diversity and representativeness of the selected samples. Despite this improvement, the development of effective and robust selection strategies remains a challenging direction for future research.

CHAPTER 6

CONCLUSIONS

Contents

6.1 Summary 113
6.2 Outlook 114

This chapter provides the concluding remarks of this thesis, summarizing the main contributions and insights gained through the research. In addition, it outlines several potential directions for further investigation and development that build upon the findings presented in this study.

6.1 Summary

This thesis explored advanced autoencoder-based methods for constructing low-dimensional parametrizations of fluid flows. The investigation began with the development of deep clustering architectures called iCAEs. They integrate deep learning with local bases for low-dimensional parametrization and state reconstruction. These models demonstrated that nonlinear approaches, especially when complemented by clustering in the latent space, can outperform POD, particularly in scenarios requiring extremely low-dimensional representations.

To realize linear projection onto a polytope which is bounded, the concept of PAEs was introduced. PAEs enforce reconstructions within a convex polytope, allowing for geometrically and physically meaningful state representations. Furthermore, a smooth clustering network was incorporated into the decoder, enabling the construction of a differentiable decoder that selects local bases via soft assignments. This structure allows the model to generate latent states that serve as convex coefficients for polytope vertices, thereby bridging nonlinear dimensionality reduction with the linear framework of polytopic LPV systems. As demonstrated in [60], PAEs with smooth clustering enhance LPV approximations in the context of nonlinear feedback control design.

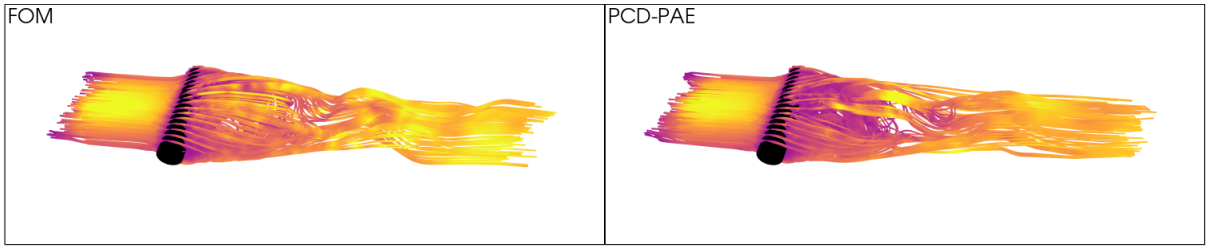


Figure 6.1: (left) The three-dimensional reference is obtained from the FEM simulation (2.18) at $\mathsf{Re} = 500$ with the state dimension $n = 107691$, and (right) an incompressible flow is reconstructed by a PCD-based ROM with reduced dimension $r = 128$.

As another task for reduced-order modeling, we proposed physics-preserving reduced-order models using PCD. These models leverage the interpretability and data property preservation offered by CUR approximations, while achieving simulation accuracy comparable to POD. In particular, we demonstrated that the proposed PCD-based ROMs maintain incompressibility, a critical physical constraint in fluid dynamics, and exhibit improved extrapolation capabilities compared to POD.

6.2 Outlook

Smooth clustering models enable soft transitions between clusters, which helps mitigate the mislabeling of flow regimes, particularly in highly chaotic flows. This soft assignment mechanism improves robustness in partitioning the latent space. However, it also introduces a risk of overfitting, as the model may over-adapt to irregular patterns in turbulent or highly nonlinear flow data. Future work could focus on refining existing smooth clustering techniques (e.g., [83]) or proposing deep clustering architectures that balance flexibility and generalizability.

In terms of network architectures, we employed affine linear decoders to maintain the linearity of the decoding process, facilitating the efficient application of our models to LPV representations and physics-preserving reduced-order models. While this design ensures compatibility with existing linear control and modeling frameworks, it inherently limits the expressiveness and accuracy of state reconstruction, particularly in highly nonlinear regimes. To overcome this limitation, future work could explore the design of nonlinear decoders that preserve essential properties, such as convex reconstruction, interpretability, and physical fidelity, while enhancing reconstruction accuracy.

Another promising research direction lies in strengthening the robustness of low-dimensional parametrizations obtained from the PCD and PAE frameworks. The proposed methods not only exhibit strong performance in two-dimensional fluid flows but also possess conceptual components, such as sparse interpolation and convolutional structures, that can be naturally extended to three-dimensional simulations. In practice, preliminary tests

(e.g., Figure 6.1) suggest that such extensions are feasible. Nevertheless, their effectiveness in three-dimensional settings remains to be rigorously validated. In particular, applying these approaches to three-dimensional flows introduces additional challenges due to the increased complexity, which may impact both the reliability and the accuracy of the resulting reduced-order models; see, e.g., [70, 18, 36]. Exploring their applicability in such settings is essential, particularly for real-world scenarios like aerospace design and climate modeling, where robust and scalable reduced-order models for high-dimensional systems are critical.

BIBLIOGRAPHY

[1] S. Aeberhard and M. Forina. *Wine*. UCI Machine Learning Repository, 1991. doi:10.24432/C5PC7J. 10

[2] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th ICDT*, pages 420–434, 2001. doi:10.1007/3-540-44503-X_27. 24

[3] S. Airen and J. Agrawal. Movie recommender system using parameter tuning of user and movie neighbourhood via co-clustering. *Procedia Comput. Sci.*, 218:1176–1183, 2023. doi:10.1016/j.procs.2023.01.096. 22

[4] M. W. Akram, M. Salman, M. F. Bashir, S. M. S. Salman, T. R. Gadekallu, and A. R. Javed. A novel deep auto-encoder based linguistics clustering model for social text. *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, 2022. doi:10.1145/3527838. 27

[5] A. Alla, D. Kalise, and V. Simoncini. State-dependent riccati equation feedback stabilization for nonlinear pdes. *Adv. Comput. Math.*, 49(1):9, 2023. doi:10.1007/s10444-022-09998-4. 35

[6] R. Altmann and J. Heiland. Finite element decomposition and minimal extension for flow equations. *ESAIM Math. Model. Numer. Anal.*, 49(5):1489–1509, September 2015. doi:10.1051/m2an/2015029. 28

[7] D. Amsallem and B. Haasdonk. PEBL-ROM: Projection-error based local reduced-order models. *Advanced Modeling and Simulation in Engineering Sciences*, 3(1), 2016. doi:10.1186/s40323-016-0059-7. 2

[8] D. Amsallem, M. J. Zahr, and C. Farhat. Nonlinear model order reduction based on local reduced-order bases. *Int. J. Numer. Methods Eng.*, 92(10):891–916, 2012. doi:10.1002/nme.4371. 2

[9] S. An and J.-J. Jeon. Distributional learning of variational autoencoder: Application to synthetic data generation. *NeurIPS 2023*, 2023. URL: https://openreview.net/forum?id=GxL6PrmEUw. 14

[10] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe. Interior-point methods for large-scale cone programming. *Optimization for Machine Learning*, 2011. doi:10.7551/mitpress/8996.003.0005. 74

[11] P. Apkarian, P. Gahinet, and G. Becker. Self-scheduled H_∞ control of linear parameter-varying systems: a design example. *Autom.*, 31(9):1251–1261, 1995. doi:10.1016/0005-1098(95)00038-X. 35

[12] D. Arthur and S. Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007. URL: https://dl.acm.org/doi/10.5555/1283383.1283494. 23

[13] D. Bank, N. Koenigstein, and R. Giryes. Autoencoders. *arXiv*, 2020. doi:10.48550/arXiv.2003.05991. 13, 38

[14] M. F. Barone, I. Kalashnikova, D. J. Segalman, and H. K. Thornquist. Stable Galerkin reduced order models for linearized compressible flow. *J. Comput. Phys.*, 228(6):1932–1946, 2009. doi:10.1016/j.jcp.2008.11.015. 33

[15] M. Behr, P. Benner, and J. Heiland. Example setups of Navier-Stokes equations with control and observation: Spatial discretization and representation via linear-quadratic matrix coefficients. *ArXiv*, 2017. doi:10.48550/arXiv.1707.08711. 28

[16] P. Benner, P. Goyal, B. Kramer, B. Peherstorfer, and K. Willcox. Operator inference for non-intrusive model reduction of systems with non-polynomial nonlinear terms. *Comput. Methods Appl. Mech. Eng.*, 372:113433, 2020. doi:10.1016/j.cma.2020.113433. 33

[17] P. Benner, J. Heiland, and S. W. R. Werner. Robust output-feedback stabilization for incompressible flows using low-dimensional $\mathcal{H}_\infty$-controllers. *Comput. Optim. Appl.*, 82(1):225–249, 2022. doi:10.1007/s10589-022-00359-x. 30

[18] M. Benosman, J. Borggaard, O. San, and B. Kramer. Learning-based robust stabilization for reduced-order models of 2D and 3D Boussinesq equations. *Appl. Math. Model.*, 49:162–181, 2017. doi:10.1016/j.apm.2017.04.032. 115

[19] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Database Theory — ICDT'99*, pages 217–235, 1999. doi:10.1007/3-540-49257-7_15. 24

[20] R. Bollapragada, D. Mudigere, J. Nocedal, H. M. Shi, and P. T. P. Tang. A progressive batching L-BFGS method for machine learning. In *Proc. of the 35th ICML 2018*, volume 80, pages 619–628, 2018. URL: http://proceedings.mlr.press/v80/bollapragada18a.html. 22

[21] C. Boutsidis and D. P. Woodruff. Optimal CUR matrix decompositions. *SIAM J. Sci. Comput.*, 46(2):543–589, 2017. doi:10.1137/140977898. xix, 9, 10

[22] P. Buchfink, S. Glas, and B. Haasdonk. Symplectic model reduction of Hamiltonian systems on nonlinear manifolds and approximation with weakly symplectic autoencoder. *SIAM J. Sci. Comput.*, 45(2):A289–A311, 2023. doi:10.1137/21M1466657. 60

[23] M. Caron, P. Bojanowski, A. Joulin, and M. Douze. Deep clustering for unsupervised learning of visual features. In *Computer Vision – ECCV 2018*, pages 139–156, Cham, 2018. doi:10.1007/978-3-030-01264-9_9. 25

[24] S. Chaturantabut and D. C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM J. Sci. Comput.*, 32(5):2737–2764, 2010. doi:10.1137/090766498. 9

[25] M. Cho, K. Alizadeh-Vahid, S. Adya, and M. Rastegari. DKM: differentiable k-means clustering layer for neural network compression. *ICLR 2022*, 2022. URL: https://openreview.net/forum?id=J_F_qqCE3Z5. 25

[26] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NIPS 2014 Workshop on Deep Learning*, 2014. doi:10.48550/arXiv.1412.3555. 16

[27] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *ICLR 2016 (Poster)*, 2016. URL: https://arxiv.org/pdf/1511.07289.pdf. 47

[28] P. Conti, G. Gobat, S. Fresca, A. Manzoni, and A. Frangi. Reduced order modeling of parametrized systems through autoencoders and SINDy approach: continuation of periodic solutions. *Comput. Methods Appl. Mech. Eng.*, 411:116072, 2023. doi:10.1016/j.cma.2023.116072. 30

[29] M. Cracco, G. Stabile, A. Lario, M. Larcher, F. Casadei, G. Valsamos, and G. Rozza. Deep learning-based reduced-order methods for fast transient dynamics. *CoRR*, abs/2212.07737, 2022. doi:10.48550/arXiv.2212.07737. 2, 38, 39

[30] X. Cui, R. Wu, Y. Liu, P. Chen, Q. Chang, P. Liang, and C. He. scsmd: a deep learning method for accurate clustering of single cells based on auto-encoder. *BMC Bioinformatics*, 26(1):33, 2025. doi:10.1186/s12859-025-06047-x. 27

[31] A. Das and J. Heiland. Low-order linear parameter varying approximations for nonlinear controller design for flows. *ECC2024*, pages 2065–2070, 2024. doi:10.23919/ECC64448.2024.10591292. 75, 76

[32] M. D'Elia, M. Perego, and A. Veneziani. A variational data assimilation procedure for the incompressible Navier-Stokes equations in hemodynamics. *J. Sci. Comput.*, 52(2):340–359, 2012. doi:10.1007/s10915-011-9547-6. 28

[33] N. Deng, B. Noack, M. Morzyński, and L. Pastur. Low-order model for successive bifurcations of the fluidic pinball. *J. Fluid Mech.*, 884:A37, 2020. doi:10.1017/jfm.2019.959. 30

[34] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun. RepVGG: Making VGG-style ConvNets great again. In *IEEE Conference on CVPR 2021, virtual*, pages 13733–13742, 2021. doi:10.1109/CVPR46437.2021.01352. 17, 38, 43

[35] P. Drineas, R. Kannan, and M. W. Mahoney. Fast Monte Carlo algorithms for matrices i: Approximating matrix multiplication. *SIAM J. Comput.*, 36(1):132–157, 2006. doi:10.1137/S0097539704442684. xix, 8

[36] J. Du, F. Fang, C. Pain, I. Navon, J. Zhu, and D. Ham. POD reduced-order unstructured mesh modeling applied to 2D and 3D fluid flow. *Comput. Math. Appl.*, 65(3):362–379, 2013. doi:10.1016/j.camwa.2012.06.009. 115

[37] J. Duan and J. S. Hesthaven. Non-intrusive data-driven reduced-order modeling for time-dependent parametrized problems. *J. Comput. Phys.*, 497:112621, 2024. doi:10.1016/j.jcp.2023.112621. 2

[38] R. Dupuis, J.-C. Jouhaud, and P. Sagaut. Surrogate modeling of aerodynamic simulations for multiple operating conditions using machine learning. *AIAA Journal*, 56(9):3622–3635, 2018. doi:10.2514/1.J056405. 2

[39] H. Eivazi, H. Veisi, M. H. Naderi, and V. Esfahanian. Deep neural networks for nonlinear model order reduction of unsteady flows. *Phys. Fluids*, 32(10):105104, 2020. doi:10.1063/5.0020526. 2

[40] M. M. Fard, T. Thonet, and É. Gaussier. Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognit. Lett.*, 138:185–192, 2020. doi:10.1016/j.patrec.2020.07.028. 2, 14, 25

[41] X. Feng, Q. Jonathan Wu, Y. Yang, and L. Cao. An autuencoder-based data augmentation strategy for generalization improvement of DCNNs. *Neurocomputing*, 402:283–297, 2020. doi:10.1016/j.neucom.2020.03.062. 14

[42] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13(70):2171–2175, 2012. URL: http://jmlr.org/papers/v13/fortin12a.html. 96

[43] S. Fresca and A. Manzoni. POD-DL-ROM: Enhancing deep learning-based reduced order models for nonlinear parametrized PDEs by proper orthogonal decomposition. *Comput. Methods Appl. Mech. Engrg.*, 388:114181, 2022. doi:10.1016/j.cma.2021.114181. 2, 33, 43

[44] H. Gao, L. Sun, and J. Wang. PhyGeoNet: Physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *J. Comput. Phys.*, 428:110079, 2021. doi:10.1016/j.jcp.2020.110079. 39

[45] J. Gao and J. Zhang. Clustered SVD strategies in latent semantic indexing. *Inf. Process. Manag.*, 41(5), 2005. doi:10.1016/j.ipm.2004.10.005. 39

[46] Q. Gao and M. Zou. An analytical solution for two and three dimensional nonlinear Burgers' equation. *Appl. Math. Model.*, 45:255–270, 2017. doi:10.1016/j.apm.2016.12.018. 31

[47] J. C. Geromel and P. Colaneri. Robust stability of time varying polytopic systems. *Syst. Control. Lett.*, 55(1):81–85, 2006. doi:10.1016/J.SYSCONLE.2004.11.016. 60

[48] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org. 13

[49] Y. L. Guennec, J.-P. Brunet, F.-Z. Daim, M. Chau, and Y. Tourbier. A parametric and non-intrusive reduced order model of car crash simulation. *Comput. Methods Appl. Mech. Eng.*, 338:186–207, 2018. doi:10.1016/j.cma.2018.03.005. 3, 96

[50] X. Guo, L. Gao, X. Liu, and J. Yin. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th IJCAI*, IJCAI'17, pages 1753–1759. AAAI Press, 2017. doi:10.5555/3172077.3172131. 27

[51] X. Guo, X. Liu, E. Zhu, and J. Yin. Deep clustering with convolutional autoencoders. In *Neural Information Processing*, pages 373–382, Cham, 2017. doi:10.1007/978-3-319-70096-0_39. 25

[52] N. Halko, P. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, 2011. doi:10.1137/090771806. xix, 7

[53] S. M. Hashemi and H. Werner. LPV modelling and control of Burgers' equation. *IFAC Proceedings Volumes*, 44(1):5430–5435, 2011. doi:10.3182/20110828-6-IT-1002.03318. 60

[54] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on CVPR*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90. 21, 38

[55] J. Heiland, P. Benner, and R. Bahmani. Convolutional neural networks for very low-dimensional LPV approximations of incompressible Navier-Stokes equations. *Frontiers Appl. Math. Stat.*, 8:879140, 2022. doi:10.3389/fams.2022.879140. 38, 40

[56] J. Heiland and Y. Kim. Convolutional autoencoders and clustering for low-dimensional parametrization of incompressible flows. *IFAC-PapersOnLine*, 55(30):430–435, 2022. Proc. of the 25th International Symposium on MTNS. doi:10.1016/j.ifacol.2022.11.091. iii, 37, 40, 45, 129

[57] J. Heiland and Y. Kim. Convolutional autoencoders, clustering, and POD for low-dimensional parametrization of flow equations. *Comput. Math. Appl.*, 175:49–61, 2024. doi:10.1016/j.camwa.2024.08.032. iii, 37, 129

[58] J. Heiland and Y. Kim. Polytopic autoencoders for very low-dimensional parametrizations of fluid flow models. *PAMM*, 25(2):e70008, 2025. Proc. of the 94th Annual GAMM Meeting. doi:10.1002/pamm.70008. iii, 59, 129

[59] J. Heiland and Y. Kim. Polytopic autoencoders with smooth clustering for reduced-order modeling of flows. *J. Comput. Phys.*, 521:113526, 2025. doi:10.1016/j.jcp.2024.113526. iii, 59, 129

[60] J. Heiland, Y. Kim, and S. W. R. Werner. Deep polytopic autoencoders for low-dimensional linearparameter-varying approximations and nonlinear feedback design. *Arxiv*, 2025. doi:10.48550/arXiv.2403.18044. iii, 34, 35, 76, 113

[61] J. Heiland and S. W. R. Werner. Low-complexity linear parameter-varying approximations of incompressible Navier-Stokes equations for truncated state-dependent riccati feedback. *IEEE Control Syst. Lett.*, 7:3012–3017, 2023. doi:10.1109/LCSYS.2023.3291231. 5, 34, 59

[62] S. Hijazi, G. Stabile, A. Mola, and G. Rozza. Data-driven POD-Galerkin reduced order model for turbulent flows. *J. Comput. Phys.*, 416:109513, 2020. doi:10.1016/j.jcp.2020.109513. 96

[63] T. Hu, F. Chen, H. Wang, J. Li, W. Wang, J. Sun, and Z. Li. Complexity matters: Rethinking the latent space for generative modeling. *NeurIPS 2023*, 2023. URL: https://openreview.net/forum?id=00EKYYu3fD. 14

[64] W. Jia, M. Sun, J. Lian, and S. Hou. Feature dimensionality reduction: a review. *Complex Intell. Syst.*, 8(3):2663–2693, 2022. doi:10.1007/s40747-021-00637-x. 24

[65] L. Jing, J. Zbontar, and Y. LeCun. Implicit rank-minimizing autoencoder. *NeurIPS 2020*, 2020. doi:doi.org/10.48550/arXiv.2010.00679. 14

[66] V. John. *Finite element methods for incompressible flow problems*, volume 51 of *Springer Ser. Comput. Math.* Springer, 2016. doi:10.1007/978-3-319-45750-5. 28

[67] Y.-E. Kang, S. Shon, and K. Yee. Local non-intrusive reduced order modeling based on soft clustering and classification algorithm. *Int. J. Numer. Methods Eng.*, 123(10):2237–2261, 2022. doi:10.1002/nme.6934. 2, 33

[68] Y. Kim and Y. Choi. Learning finite difference methods for reaction-diffusion type equations with FCNN. *Comput. Math. Appl.*, 123:115–122, 2022. doi:10.1016/j.camwa.2022.08.006. 39

[69] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR 2015, Conference Track Proceedings*, 2015. doi:10.48550/arXiv.1412.6980. 21, 42, 69

[70] F. Koehler, S. Niedermayr, R. Westermann, and N. Thuerey. APEBench: A benchmark for autoregressive neural emulators of PDEs. *NeurIPS 2024*, 2024. doi:10.48550/arXiv.2411.00180. 115

[71] P. J. W. Koelewijn and R. Tóth. Scheduling dimension reduction of LPV models - A deep neural network approach. *Proceedings of the IEEE*, pages 1111–1117, 2020. doi:10.23919/ACC45564.2020.9147310. 1, 34, 38

[72] A. Kwiatkowski and H. Werner. PCA-based parameter set mappings for LPV models with fewer parameters and less overbounding. *IEEE Trans. Control. Syst. Technol.*, 16(4):781–788, 2008. doi:10.1109/TCST.2007.903094. 76

[73] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, 1998. doi:10.1109/5.726791. 16, 39

[74] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J. Comput. Phys.*, 404, 2020. doi:10.1016/J.JCP.2019.108973. 2, 38, 60

[75] S. Lee, K. Jang, S. Lee, H. Cho, and S. Shin. Parametric model order reduction by machine learning for fluid–structure interaction analysis. *Engineering with Computers*, 2023. doi:10.1007/s00366-023-01782-2. 38

[76] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, and X. Wei. YOLOv6: A single-stage object detection framework for industrial applications. *arXiv*, 2022. doi:10.48550/arXiv.2209.02976. 38

[77] D. Liu, S. Xie, Y. Li, D. Zhao, and E. M. El-Alfy, editors. *Deep Clustering with Convolutional Autoencoders*, volume 10635 of *Lecture Notes in Comput. Sci.* Springer, 2017. doi:10.1007/978-3-319-70096-0_39. 2

[78] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with warm restarts. *5th ICLR 2017 Conference Track Proceedings*, 2017. doi:10.48550/arXiv.1608.03983. 21

[79] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. *Advances in NIPS 29*, pages 4898–4906, 2016. doi:10.48550/arXiv.1701.04128. 18

[80] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Berkeley Symp. on Math. Statist. and Prob.*, 1:281–297, 1967. URL: http://projecteuclid.org/euclid.bsmsp/1200512992. 23

[81] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA*, 106(3):697–702, 2009. doi:10.1073/PNAS.0803205106. 3, 9, 96

[82] R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Phys. Fluids*, 33(3):037106, 2021. doi:10.1063/5.0039986. 2, 33

[83] C. Mavridis and K. H. Johansson. Hybrid learning for model predictive control approximation. *ECC2025*, 2025. 114

[84] V. B. Nguyen, S. B. Q. Tran, S. A. Khan, J. Rong, and J. Lou. POD-DEIM model order reduction technique for model predictive control in continuous chemical processing. *Comput. Chem. Eng.*, 133:106638, 2020. doi:10.1016/j.compchemeng.2019.106638. 2

[85] M. Ohlberger and S. Rave. Reduced basis methods: Success, limitations and future challenges. *Proceedings of the Conference Algoritmy*, pages 1–12, 2016. doi:10.48550/arXiv.1511.02021. 1, 38

[86] H. Ohno. Auto-encoder-based generative models for data augmentation on regression problems. *Soft Comput.*, 24(11):7999–8009, 2020. doi:10.1007/s00500-019-04094-0. 14

[87] T. R. F. Phillips, C. E. Heaney, P. N. Smith, and C. C. Pain. An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion. *Int. J. Numer. Methods Eng.*, 122(15):3780–3811, 2021. doi:10.1002/nme.6681. 2, 60

[88] F. Pichi, B. Moya, and J. S. Hesthaven. A graph convolutional autoencoder approach to model order reduction for parametrized PDEs, 2023. URL: http://arxiv.org/abs/2305.08573. 64

[89] R. Prabhakar and M. Basavaraju. A novel method of spam mail detection using text based clustering approach. *Int. J. of Comput. Appl.*, 5(4):15–25, 2010. doi:10.5120/906-1283. 22

[90] S. Rendle. Factorization machines. In *The 10th IEEE ICDM 2010*, pages 995–1000. IEEE Computer Society, 2010. doi:10.1109/ICDM.2010.127. 5

[91] S. Z. Rizvi, F. Abbasi, and J. M. Velni. Model reduction in linear parameter-varying models using autoencoder neural networks. *IEEE*, pages 6415–6420, 2018. doi:10.23919/ACC.2018.8431912. 2, 34, 38, 60

[92] M. Ronen, S. E. Finder, and O. Freifeld. Deepdpm: Deep clustering with an unknown number of clusters. In *IEEE/CVF Conference on CVPR 2022*, pages 9851–9860. IEEE, 2022. doi:10.1109/CVPR52688.2022.00963. 25

[93] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on CVPR*, pages 4510–4520, 2018. doi:10.1109/CVPR.2018.00474. 65

[94] P. J. SCHMID. Dynamic mode decomposition of numerical and experimental data. *J. Fluid Mech.*, 656:5–28, 2010. doi:doi:10.1017/S0022112010001217. 33

[95] E. Schubert. Stop using the elbow criterion for k-means and how to choose the number of clusters instead. *ACM SIGKDD Explorations Newsletter*, 25(1):36–42, 2023. doi:10.1145/3606274.3606278. 24

[96] D. Sculley. Web-scale k-means clustering. *WWW'10*, pages 1177–1178, 2010. doi:10.1145/1772690.1772862. 23, 66

[97] K. Shin, J. Hammond, and P. White. Iterative SVD method for noise reduction of low-dimensional chaotic time series. *Mech. Syst. Signal Process.*, 13(1):115–124, 1999. doi:10.1006/mssp.1998.9999. 5

[98] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *3rd ICLR, Conference Track Proceedings*, 2015. doi:10.48550/arXiv.1409.1556. 38

[99] D. C. Sorensen and M. Embree. A DEIM induced CUR factorization. *SIAM J. Sci. Comput.*, 38(3), 2016. doi:10.1137/140978430. 9

[100] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014. doi:10.5555/2627435.2670313. 22, 86

[101] G. Stabile and G. Rozza. Finite volume POD-Galerkin stabilised reduced order methods for the parametrised incompressible Navier–Stokes equations. *Comput. Fluids*, 173:273–284, 2018. doi:10.1016/j.compfluid.2018.01.035. 33

[102] H. Steck. Autoencoders that don't overfit towards the identity. *NeurIPS 2020*, 2020. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/e33d974aae13e4d877477d51d8bafdc4-Paper.pdf. 14

[103] G. Strang. *Linear Algebra and Learning from Data*. Wellesley-Cambridge Press, 2019. doi:10.1137/1.9780692196380. 9

[104] C. C. Sullivan, H. Yamashita, and H. Sugiyama. Reduced order modeling of deformable tire-soil interaction with proper orthogonal decomposition. *J. Comput. Nonlinear Dyn.*, 17(5):051009, 2022. doi:10.1115/1.4053592. 2, 33, 96

[105] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th ICML*, pages 1139–1147. PMLR, 2013. URL: https://proceedings.mlr.press/v28/sutskever13.html. 21

[106] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *2016 IEEE Conference on CVPR*, pages 2818–2826. IEEE Computer Society, 2016. doi:10.1109/CVPR.2016.308. 22, 86

[107] K. Taira, M. S. Hemati, S. L. Brunton, Y. Sun, K. Duraisamy, S. Bagheri, S. T. M. Dawson, and C.-A. Yeh. Modal analysis of fluid flows: Applications and outlook. *AIAA Journal*, 58:3:998–1022, 2020. doi:10.2514/1.J058462. 30

[108] M. Tan and Q. V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. URL: http://proceedings.mlr.press/v97/tan19a.html. 38

[109] M. Trudgen, S. Z. Rizvi, and J. Mohammadpour. Linear parameter-varying approach for modeling rapid thermal processes. In *2016 ACC*, pages 3243–3248, 2016. doi:10.1109/ACC.2016.7525417. 34, 60

[110] J. Vos, A. Rizzi, D. Darracq, and E. Hirschel. Navier–Stokes solvers in European aircraft design. *Prog. Aerosp. Sci.*, 38(8):601–697, 2002. doi:10.1016/S0376-0421(02)00050-7. 28

[111] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016. doi:10.1016/j.neucom.2015.08.104. 14

[112] N. Winovich, K. Ramani, and G. Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *J. Comput. Phys.*, 394:263–279, 2019. doi:10.1016/j.jcp.2019.05.026. 39

[113] W. Wolberg, O. Mangasarian, N. Street, and W. Street. *Breast Cancer Wisconsin (Diagnostic)*. UCI Machine Learning Repository, 1995. doi:10.24432/C5DW2B. 10

[114] J. Xie, R. B. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, volume 48, pages 478–487, 2016. URL: http://proceedings.mlr.press/v48/xieb16.html. 2, 25, 27

[115] J. Yosinski, J. Clune, A. M. Nguyen, T. J. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *CoRR*, 2015. URL: http://arxiv.org/abs/1506.06579. 43

[116] M. D. Zeiler, D. Krishnan, G. W. Taylor, and R. Fergus. Deconvolutional networks. In *2010 IEEE Conference on CVPR*, pages 2528–2535, 2010. doi:10.1109/CVPR.2010.5539957. 19

[117] X. Zhang and L. Jiang. Conditional variational autoencoder with Gaussian process regression recognition for parametric models. *J. Comput. Appl. Math.*, 438:115532, 2024. doi:10.1016/j.cam.2023.115532. 2

[118] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye. Object detection in 20 years: A survey. *Proc. of the IEEE*, 111(3):257–276, 2023. doi:10.1109/JPROC.2023.3238524. 22

STATEMENT OF SCIENTIFIC COOPERATIONS

This work is based on articles and reports (published and unpublished) that have been obtained in cooperation with various coauthors. To guarantee a fair assessment of this thesis, this statement clarifies the contributions that each individual coauthor has made. The following people contributed to the content of this work:

- Prof. Dr. Peter Benner (PB), Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany
- Dr. Jan Heiland (JH), Max Planck Institute for Dynamics of Complex Technical Systems, Magdeburg, Germany

Section 2.4

JH provided his own code for data collection and suggested that I generate the snapshot data with minor modifications.

Chapter 3

JH proofread and suggested several improvements for the manuscripts [56, 57]. He proposed the interpolation matrix described in Remark 3.1, for which I developed an efficient Python implementation. I derived structural reparameterization of (de-)convolutional layers in Remark 3.2. The concepts of iCAEs and cPOD were proposed by me while JH contributed to improving the decoding architecture of cPOD. I conducted all numerical experiments including data generation, model training, and performance evaluation.

Chapter 4

JH proofread and suggested several improvement for the manuscripts [58, 59]. He proposed the concept of a polytopic representation of states. Building on this idea, I discovered that iCAEs can be reformulated as a PAE by incorporating a smooth clustering model through several modifications. Guided by this insight, I proposed the definition of PAEs and developed a lightweight PAE architecture. Additionally, I introduced the concept of the activation rate, which quantifies the relative contribution of each latent

variable to the state reconstruction. JH suggested me to present an advantage of the polytopic decoders in Remark 4.10. All numerical experiments including data generation, model training, and evaluation were carried out by me.

Chapter 5

PB and JH provided insights on physics-preserving reduced-order modeling including CUR decomposition and DEIM-based selection methods. I proposed a DEIM-based selection method with data pruning and developed several CUR decomposition algorithms. Additionally, I derived a reduced-order model system for the incompressible Navier–Stokes equations. I conducted all numerical experiments including data generation, model training, and performance evaluation.