# Data-Driven Surrogate Models for Computational Fluid Dynamics

by

Rakesh Halder

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Aerospace Engineering)
in The University of Michigan
2024

Doctoral Committee:

Professor Krzysztof Fidkowski, Co-Chair
Professor Kevin Maki, Co-Chair
Professor Matthew Collette
Professor Joaquim Martins

Rakesh Halder

rhalder@umich.edu

ORCID iD: 0000-0003-2540-9411

© Rakesh Halder 2024

# ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisors, Professor Krzysztof Fidkowski and Professor Kevin Maki. Chris and Kevin have been incredible advisors and I am grateful for their constant helpfulness and support. I appreciate the flexibility I was given to pursue my research interests during my time as a PhD student and the knowledge and skills I have gained as a result. I would also like to thank Professor Joaquim Martins and Professor Matthew Collette for agreeing to be a part of my committee and offering constructive feedback.

I'm grateful for the advice and support I have received from past and present colleagues and friends at the University of Michigan. Working with Dr. Ping He at the beginning of my PhD allowed me to learn a great deal as well as adjust to the research process and I am thankful for his knowledge and kindness. I enjoyed sharing an office with members of my research group including Miles McGruder, Alex Coppeans, Yifan Bai, and Guodong Chen. My time in Ann Arbor wouldn't have been the same without the great friends I made: Christian, Jordan, Andrew, Prit, Kim, Jack, Simon, Max, Audelia, and others. I'd also like to extend my gratitude to my mentors during my internship at Autodesk Research, Mehdi Ataei and Hesam Salehipour. I had a great time working in Toronto and was able to make some valuable contributions to this dissertation. Finally, I'd like to thank my family for their continuous support and encouragement.

# TABLE OF CONTENTS

# LIST OF FIGURES

**Figure**

# LIST OF TABLES

**Table**

# ABSTRACT

The use of computational fluid dynamics (CFD) has become essential for aerospace design optimization processes. The computational cost of high-fidelity CFD is often very large and can make design optimization prohibitively expensive if a large number of design evaluations are required. Reduced-order models (ROMs) are a method that can be used to mitigate this cost. ROMs are low-dimensional data-driven surrogate models that are trained using a set of computed high-fidelity simulation snapshots. Many ROMs utilize the proper orthogonal decomposition (POD), a linear subspace method for representing solution spaces. While ROMs are becoming increasingly popular, they do face some challenges in their practical use, which include maximizing accuracy for a given computational budget, the ability to generalize throughout a parameter space, and applicability to topologically dissimilar meshes.

In this dissertation, algorithms are introduced to improve the performance, stability, and understanding of data-driven surrogate CFD models and their applications. As ROMs tend to use a small amount of training data, their predictive performance is highly sensitive to their choice. Algorithms to improve the data selection process for POD-based ROMs are introduced using Isomap, a versatile technique for nonlinear dimensionality reduction, resulting in significantly improved predictive performance for a given computational budget when used over a traditional and widely used statistical sampling technique. Next, ROMs using artificial neural networks, specifically convolutional autoencoders (CAEs), are introduced to address the performance limits of POD for problems that are highly nonlinear or require large amounts of training

data, such as unsteady ROMs involving multiple designs. A steady ROM framework combining CAEs with Gaussian process regression (GPR) is introduced and shown to significantly outperform POD when applied to a highly nonlinear lid-driven cavity problem. Ensemble learning is also used to effectively address the issue of error propagation in unsteady ROMs, where errors made early on can accumulate and lead to large inaccuracies over long time horizons at unseen design points. Finally, field inversion and machine learning (FIML) is proposed as an an alternative to ROMs for problems that require topologically dissimilar meshes. Field inversion involves obtaining improvements to turbulence models by augmenting them with a corrective field that is obtained using gradient-based optimization. Using a machine learning model trained on local flow variables and their gradients, a data-driven turbulence model is introduced to improve the predictive capabilities of baseline turbulence models, allowing for the prediction of complex flow phenomena present in experimental results.

# CHAPTER I

# Introduction

In this chapter, motivation for the research presented in this dissertation is discussed. A brief introduction to reduced-order models and their use cases is also given to provide context for the undertaken research, although more in-depth literature reviews are provided in each technical chapter. An overview of the structure of the disseration is also provided.

## 1.1 Motivation

The use of physics-based modeling and simulation has become essential for many aerospace applications involving fluid dynamics. Physical systems exhibiting complex phenomena, extreme operating conditions, or costly physical designs can be difficult or impossible to evaluate experimentally or empirically. Computational models governed by physical equations, generally in the form of parameterized partial differential equations (PDEs), allow for obtaining highly accurate representations of systems that may otherwise be difficult to evaluate. Industrial aerospace development heavily relies upon the use of computational fluid dynamics (CFD), which has several use cases including aerodynamic design optimization, engine performance simulation, compressible flow, and renewable energy. From the first industrial uses of CFD codes in the early 1970s, their adoption has increased dramatically [1], largely in part due

to the substantial increase in computing power and its availability [2], leading to a decreased need for complex and expensive experimental instruments such as wind tunnels. Figure 1.1 shows a current plot of the computing power of the top 500 supercomputers in the world over time. Over the course of around 30 years, the performance of state-of-the-art supercomputers has increased by approximately $10^7$-fold, allowing for a substantial increase in scientific computing.

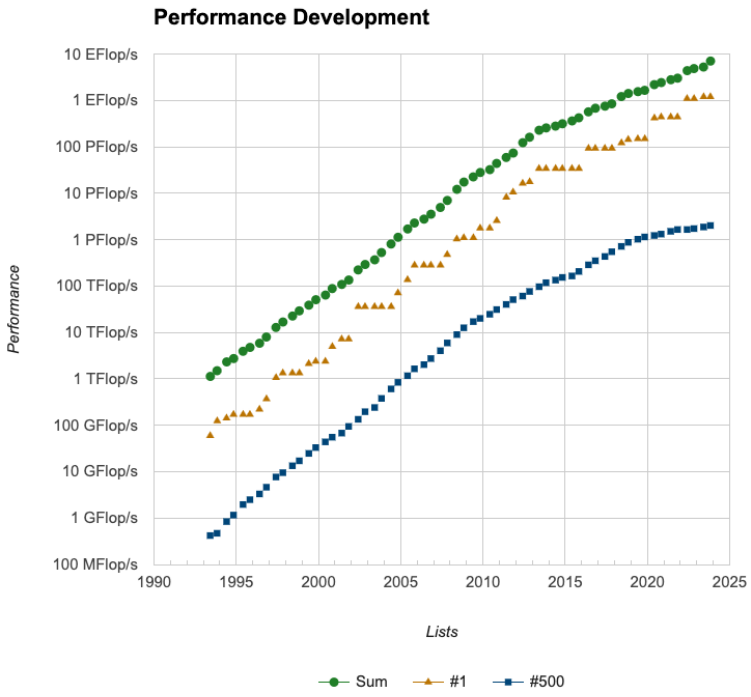



Figure 1.1: Plot showing the computing power of the top 500 supercomputers in the world over time (source: https://top500.org/).

While the use and accessibility of CFD is constantly increasing, there is still an unmet demand for computational power that can efficiently drive large-scale aerodynamic design optimization processes [3]. Design optimization involves simulating the performance and characteristics of multiple designs at different sets of design parameters $\boldsymbol{\mu}$ that control physical and geometric simulation properties. Often, the resolution, or fidelity of these simulations, is required to be high in order to accurately resolve the underlying physics [4]. As high-fidelity simulation can come at a high computational cost even when utilizing modern supercomputers, evaluating a

large number of designs can become infeasible, introducing a bottleneck in the design optimization process.

To overcome this prohibitive cost, the use of reduced-order models (ROMs) [5] is often employed. ROMs are data-driven surrogate models that can be evaluated in real-time to provide accurate approximations of the full-order model (FOM) for unseen designs. ROMs use a set of computed high-fidelity simulations as training data to build a surrogate model. ROMs use a low-dimensional embedding through a set of *expansion coefficients* that are used to provide an efficient mapping to the high-dimensional solution space. By drastically lowering the degrees of freedom of the FOM, ROMs provide a model that can be solved and evaluated rapidly to provide reliable estimates of full-order states. This enables practioners to effectively explore design spaces at a low cost to identify points of interest, where the high-fidelity model can optionally be run to obtain more accurate results.

Another widely used method for design optimization is multidisciplinary design optimization (MDO) [6], a technique that uses numerical optimization to minimize an objective function that represents the quantity of interest. MDO usually involves the use of gradient-based optimization, which itself can incur significant computational costs in computing sensitivities even when using efficient algorithms such as the adjoint method [7]. While constraints can be added to the design space for the optimization problem, it is difficult or even impossible to add certain considerations such as aesthetics, regulatory requirements, or user experience. An example of this is automotive design; while fuel efficiency is a desired aspect of vehicles and can be easily formulated into an objective function, consumers also place great importance on aesthetics and user experience when purchasing vehicles, traits that are difficult to incorporate into an optimization problem. This can result in the design space consisting of a set of pre-selected designs that are evaluated for factors such as fuel efficiency or performance to find an optimal design. While MDO is an effective method, it

cannot be used to provide fast approximations of the FOM at desired points.

When using ROMs, some important performance aspects to consider are accuracy and generalizability. ROMs must be constructed and trained in a way that makes them useful for making accurate predictions for their specific use case. While accuracy and generalizability are often inter-changeable, the latter refers to the ability of the model to offer adequate predictive performance throughout the design space rather than only at some portions.

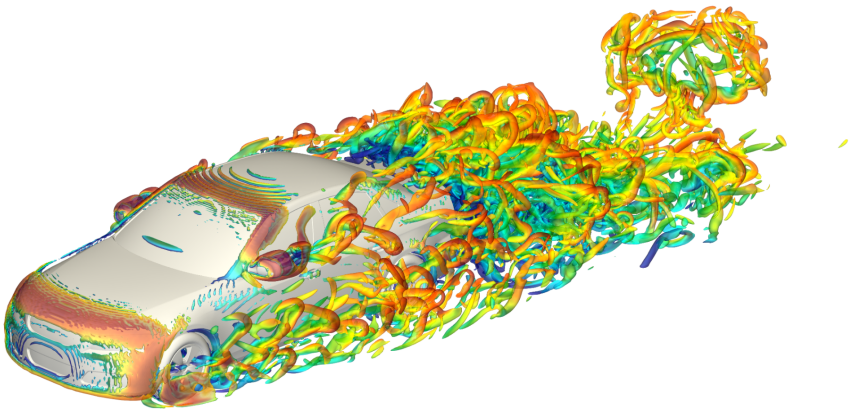


Figure 1.2: High-fidelity simulation snapshot of external flow over a vehicle containing 317 million cells (source: https://github.com/Autodesk/XLB/)

## 1.2 ROM Frameworks

ROMs require a mapping from the low-dimensional set of expansion coefficients to the high-dimensional solution space. The most popular method for this is the proper orthogonal decomposition (POD) [8, 9] method that applies the singular value decomposition (SVD) to a snapshot matrix where the columns contain different full-order states of the computed high-fidelity model at the training points. Using the training data, POD decomposes the solution space into a set of orthonormal *basis vectors*. Using the expansion coefficients, a linear combination of the basis vectors is used to approximate solutions within the parameter space. POD is widely used due to its effectiveness in dimensionality reduction, computational efficiency, linear

algebra properties, and interpretability.

ROMs consist of two stages: a computationally expensive offline stage, where the FOM is solved at chosen points within the design space to obtain training data and the surrogate model is trained, and an inexpensive online stage, where the ROM is evaluated at unseen designs. ROMs are classified as either non-intrusive or projection based. Non-intrusive ROMs (ni-ROMs) [10] incorporate a fully data-driven approach for making predictions. Predictions at unseen designs are made by using only the design parameters and expansion coefficients from the training data. For steady-state problems, this involves making a single point estimate of the expansion coefficients by using a regression model that assumes a functional dependence on the design parameters. While non-intrusive ROMs are very inexpensive to evaluate and easy to implement, they do not provide approximations that obey the governing equations. While the assumption of a functional dependence between the design parameters and expansion coefficients is largely reasonable, certain problems can exhibit a highly non-smooth relationship between the two, leading to poor regression results. As they are not intrusive into the flow solver, ni-ROMs are very easy to implement and portable between different physics codes.

Projection-based ROMs (p-ROMs) [11] incorporate the governing equations into the surrogate model by solving a low-dimensional version of the FOM. This approach leads to more physically consistent predictions that obey the underlying governing equations to some level of approximation. For this reason, the results also tend to be more accurate than those from ni-ROMs. While the offline stages for both projection-based and non-intrusive ROMs are comparable in computational cost, the online stage for p-ROMs does not have a negligible cost, although it is still typically far lower than solving the FOM. The implementation of p-ROMs is not trivial as they require access to the flow solver, often involving significant modifications. Additionally, p-ROMs are known to suffer from numerical instabilities [12] and can fail to converge for certain

problems.

The predictive performance of all data-driven models is highly dependent upon the quality and diversity of training data used to build the model. Since ROMs typically use a small amount of training data, this further increases the sensitivity of the model's performance. The physical regimes present within the design space should be well-represented, making the choice of training points both vital and non-trivial. This is particularly important for problems that exhibit highly nonlinear variations throughout the design space.

While POD is widely used and works well for many problems, the method has issues when applied to highly nonlinear problems, requiring a large number of basis vectors to provide adequate performance [13]. POD makes the assumption that the physical system being modeled behaves approximately linearly within the subspace provided by the basis vectors. This is a reasonable assumption for many problems, but can lead to a very large number of basis vectors being required for accurate reconstructions within the solution space for highly nonlinear ones. For ni-ROMs, this can lead to the compounding of individual errors in a large number of predicted expansion coefficients that results in large inaccuracies. As p-ROMs typically rely on numerically solving linear systems the size of which are in proportion to the number of basis vectors, this can lead to increased issues with numerical instabilities.

Rather than using POD to provide a low-dimensional representation of the solution space, recent advances have utilized artificial neural networks (ANNs), a deep learning [14] method, to provide a nonlinear mapping between a low-dimensional *latent space*, an abstract representation of compressed data, and the high-fidelity solution space. ANNs are computational models inspired by the human brain's neural structure and they excel at learning highly complex and abstract relationships from data. Autoencoders, a type of ANN, are useful for learning low-dimensional representations of data and have been used in ROMs where they have been shown to

outperform POD [15]. Autoencoders can very efficiently represent high-dimensional data using very few latent variables, making them ideal for highly nonlinear problems. Another case where this is useful is for problems that require large amounts of simulation data, such as unsteady ROMs trained on a large number of time snapshots from multiple designs. Although the use of ANNs can improve the performance of ROMs, their generalizability and reliability have not been firmly established, nor the type of problems for which they are more powerful than POD [16]. Additionally, deep learning requires relatively large amounts of training data, which can restrict its use for ROMs.

A major limitation of ROMs is that they require topologically similar meshes across the design space if approximations of the FOM are desired. The methods ROMs use require that the number of cells and their ordering are consistent for all full-order states. While this can be achieved using mesh deformation methods, it can remain infeasible for problems that involve large geometric changes or that require re-meshing to sufficiently resolve fine-scale physics in specific regions of the computational domain. In practice, a common baseline mesh is sometimes used to interpolate the results, but this is not trivial and can result in large amounts of information loss. This issue points towards a need for inexpensive data-driven CFD models that are mesh-agnostic.

## 1.3 Contributions and Outline

This dissertation addresses three challenges faced by ROMs. The first involves the development of computationally efficient methods to obtain training data for both non-intrusive and projection-based POD-based ROMs. Given a fixed computational budget for generating high-fidelity training data, the goal is to maximize the predictive performance of the ROM throughout the design space, aiming for high levels of accuracy and generalizability. This leads to the development of iterative

data-driven parameter selection algorithms based on the current set of training data.

The next challenge involves identification of the types of problems for which ROMs using deep learning are well-suited as well as improving their reliability for unsteady ROMs. For steady-state problems involving multiple designs, a previous work by Mrosek et al. [17] applied an autoencoder-based ni-ROM to a geometrically parameterized vehicle problem and found no improvement over using POD. The type of problem for which a ROM of this nature would be advantageous in addition to the effect of the ROM's dimensionality and neural network components is investigated. An unsteady ROM framework involving multiple designs that retains high levels of accuracy and stability over long time horizons at unseen points is also introduced.

Finally, an alternative to ROMs for CFD problems involving topologically dissimilar meshes is introduced. In particular, field inversion [18] is used to obtain data-driven corrections to turbulence models so that they better approximate high-fidelity simulations such as direct numerical simulation (DNS) and large eddy simulation (LES) or experimental data. Field inversion uses numerical optimization to solve an inverse problem for a corrective field introduced into an existing turbulence model. A machine learning model can also be used from completed field inversion runs to develop a data-driven turbulence model, a mesh-agnostic framework referred to as field inversion and machine learning (FIML).

The main contributions of the dissertation are as follows:

- Developed data-driven parameter selection algorithms for non-intrusive and projection-based ROMs using manifold learning, a machine learning technique for nonlinear dimensionality reduction.

- Introduced a non-intrusive ROM framework for steady-state problems using autoencoders augmented by Gaussian process regression to predict the latent variables.

- Constructed an unsteady non-intrusive ROM framework using autoencoders for spatial reconstruction of the full-order model and recurrent neural networks for time-series forecasting trained using ensemble learning, a machine learning technique for improving the stability of predictive models.

- Investigated the use of local state information in the form of flow variables, their gradients, and mesh wall distance for a field inversion and machine learning framework and feature importance for both wall-resolved and wall-modeled turbulent flows.

Chapter 2 introduces the proper orthogonal decomposition as well as the baseline POD-based non-intrusive and projection-based ROMs that are used. Chapter 3 presents two efficient data selection algorithms for ROMs using manifold learning to improve the predictive performance of ROMs. The two non-intrusive ROM frameworks using deep learning are introduced in Chapter 4 along with background material on artificial neural networks. Chapter 5 introduces the field inversion and machine learning framework using local state information as features to develop an enhanced data-driven turbulence model.

# CHAPTER II

# Reduced-Order Modeling

This chapter gives an overview of the baseline non-intrusive and projection-based ROMs used in this work for steady-state problems. Both ROMs utilize the proper orthogonal decomposition, which is also described. The non-intrusive ROM relies on a regression model to predict the expansion coefficients at unseen points, while the projection-based ROM solves a low-dimensional version of the full-order model using a reduced representation of the residuals. While the non-intrusive ROM is more computationally efficient and easier to implement, the projection-based ROM is often more accurate and leads to more physically consistent approximations. However, projection-based ROMs are known to suffer from stability issues that can impede convergence.

## 2.1 Full-order Model

The full-order model is considered to be the solution $\boldsymbol{x}(\boldsymbol{\mu}) \in \mathbb{R}^N$ of state variable(s) in a system governed by a set of steady-state parameterized partial differential equations discretized over a computational domain $\Omega \in \mathbb{R}^d$. We consider design parameters $\boldsymbol{\mu} \in \boldsymbol{\mathcal{D}}$ that define both the computational domain and parameters of the governing equations. Here $\boldsymbol{\mathcal{D}} \subseteq \mathbb{R}^p$ denotes the parameter space such that $\boldsymbol{x}$:

Parts of this chapter appear in or are adapted from our previously published papers [19, 20].

$\boldsymbol{\mathcal{D}} \rightarrow \mathbb{R}^N$. The set of PDEs governing the FOM is solved numerically over $\Omega$ to generate a solution $\boldsymbol{x}(\boldsymbol{\mu})$. The computational cost of numerically solving the system increases polynomially with its dimension, $N$, which is in proportion with the fineness of the mesh. Accurate or useful solutions of systems often require large values of $N$ ($\mathcal{O}(10^6 - 10^9)$), resulting in large computational costs for a single solution. In processes such as design optimization, the need to evaluate the solutions for many different designs in real-time becomes infeasible if numerous FOMs have to be solved. This large computational cost motivates the use of reduced-order models, where a small number of FOMs are solved and used to create a computationally inexpensive surrogate model that can deliver accurate approximations in real time.

### 2.1.1 Steady Navier-Stokes Equations

The majority of the test cases presented in this work are simulated by numerically solving the steady incompressible Navier-Stokes equations,

$$\int_S \overrightarrow{V} \cdot \mathrm{d}\vec{n}\, dS = 0, \tag{2.1}$$

$$\int_S \overrightarrow{V}\overrightarrow{V} \cdot \mathrm{d}\vec{n}\, dS + \int_V \nabla p \,\mathrm{d}V - \int_S (\nu + \nu_t)(\nabla \overrightarrow{V} + \nabla \overrightarrow{V}^T) \cdot \mathrm{d}\vec{n}\, dS = 0, \tag{2.2}$$

where $\overrightarrow{V} = [u, v, w]$ is the velocity vector and $u$, $v$, and $w$ are the velocity components in the $x$, $y$, and $z$ directions respectively, $S$ is the face-area vector, $\vec{n}$ is the outward-pointing normal, $V$ is the volume; $\nu$ is the kinematic viscosity, $\nu_t$ is the turbulent eddy viscosity, and $p$ is the pressure. The continuity and momentum equations are discretized over the computational domain by using the finite-volume method (FVM). Both equations are coupled through the semi-implicit method for pressure-linked equations (SIMPLE) algorithm [21] along with Rhie–Chow interpolation [22]. The SIMPLE algorithm employs a pseudo time-stepping technique through a predictor-

corrector method. In the predictor step, an intermediate velocity field is solved for, followed by the correction step where the velocity field is updated based on a pressure correction equation. For turbulent flows, the Reynolds-averaged Navier-Stokes [23] (RANS) approach is used. Turbulent flows consist of a spatially varying mean flow and random fluctuations. The RANS approach applies an averaging operation to the Navier-Stokes equation to decompose the flow into its mean and Reynolds stress terms which are required to be modelled. In this work, the Spalart–Allmaras [24] (SA) one-equation turbulence model is chosen,

$$\int_V \nabla \cdot (\overrightarrow{V}\tilde{\nu})\,\mathrm{d}V - \frac{1}{\sigma}\int_V \nabla \cdot [(\nu+\tilde{\nu})\nabla\tilde{\nu}] + c_{b2}|\nabla\tilde{\nu}|^2\,\mathrm{d}V - c_{b1}\int_V \tilde{S}\tilde{\nu}\,\mathrm{d}V + c_{w1}\int_V f_w\left(\frac{\tilde{\nu}}{d}\right)^2\mathrm{d}V = 0, \tag{2.3}$$

where $\tilde{\nu}$ is the modified viscosity, which is related to the turbulent eddy viscosity as

$$\nu_t = \tilde{\nu}\frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\tilde{\nu}}{\nu}. \tag{2.4}$$

The four terms in Eq. (2.3) represent the turbulent convection, diffusion, production, and destruction respectively. The SA model was designed for aerodynamic flows and accurately models the boundary layer. The model is also simpler than other turbulence models, such as two-equation models including $k-\omega$ [25] and $k-\epsilon$ [26]. This leads to a lower computational cost, making it desirable for large-scale problems. The original work by Spalart and Allmaras can be consulted for an in-depth overview of this model and detailed definitions of terms and closure coefficient values.

OpenFOAM [27], an open-source toolbox for multiphysics simulation, is used for numerical simulations. The OpenFOAM solver simpleFoam is used to simulate steady incompressible flow.

## 2.2 Proper Orthogonal Decomposition

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

$$\boldsymbol{S} \in \mathbb{R}^{N \times n} = [\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^n] = [\boldsymbol{x}(\boldsymbol{\mu}^1), \boldsymbol{x}(\boldsymbol{\mu}^2), \cdots, \boldsymbol{x}(\boldsymbol{\mu}^n)]. \tag{2.5}$$

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

$$\boldsymbol{x}^i \approx a_1^i \boldsymbol{\psi}^1 + a_2^i \boldsymbol{\psi}^2 \cdots + a_k^i \boldsymbol{\psi}^k. \tag{2.6}$$

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

$$\boldsymbol{S} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}. \tag{2.7}$$

Here, $\boldsymbol{U}$ contains a set of $n$ left singular vectors that form an orthonormal basis for the column space of $\boldsymbol{S}$, $\boldsymbol{V}$ contains a set of $n$ right singular vectors that form an orthonormal basis for the row space of $\boldsymbol{S}$, and $\text{diag}(\boldsymbol{\Sigma}) \in \mathbb{R}^n = [\sigma_1, \sigma_2, \cdots, \sigma_n]$ contains the singular values corresponding to the singular vectors in descending order, $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. The magnitude of the singular values represents the importance or strength of the corresponding singular vectors in $\boldsymbol{U}$. Often, the singular values

associated with the basis vectors decay very quickly and only the first $k$ singular vectors are chosen to form the POD basis, $\boldsymbol{\Psi} \in \mathbb{R}^{N\times k} = [\boldsymbol{\psi}^1, \boldsymbol{\psi}^2, \cdots, \boldsymbol{\psi}^k]$. To determine the value of $k$, the relative information content of the subspace is evaluated

$$E(k) = \frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{n} \sigma_j^2}, \tag{2.8}$$

and $k$ is chosen such that $E(k) \geq \gamma$, where $\gamma \in [0,1]$ is chosen depending on the problem, usually to a value $\gamma \geq 0.95$ [28]. This is done to preserve the most dominant features of the solution space. Using the POD basis, full-order solutions at unseen design parameters $\boldsymbol{x}(\boldsymbol{\mu}^*)$ can be approximated by

$$\boldsymbol{x}(\boldsymbol{\mu}^*) \approx \boldsymbol{\Psi}\boldsymbol{a}^* = a_1^*\boldsymbol{\psi}^1 + a_2^*\boldsymbol{\psi}^2 \cdots + a_k^*\boldsymbol{\psi}^k, \tag{2.9}$$

where $\boldsymbol{a}^*$ can be estimated through a computational model that takes $\boldsymbol{\mu}^*$ as an input.

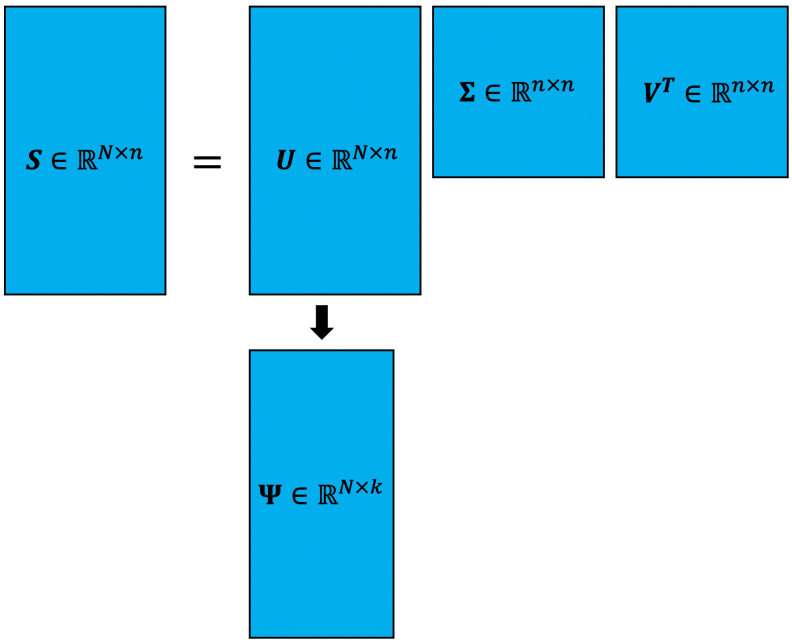



Figure 2.1: Schematic of the proper orthogonal decomposition (POD).

### 2.2.1 Projection Error

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

$$\hat{\boldsymbol{x}}^i = \boldsymbol{\Psi}\boldsymbol{\Psi}^T\boldsymbol{x}^i. \tag{2.10}$$

A measure of the relative error over all of the solution snapshots in $\boldsymbol{S}$ is the quantity

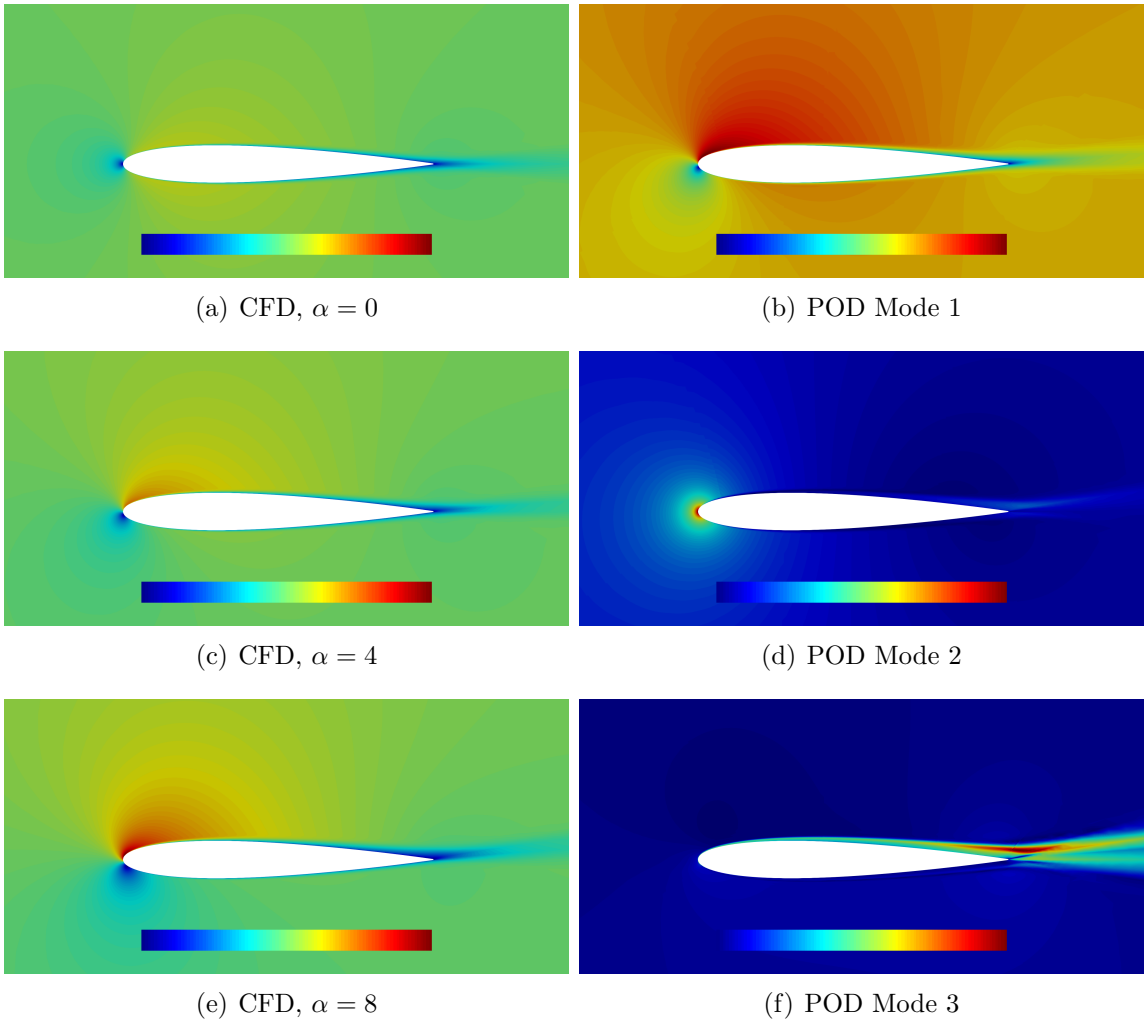
$$\epsilon_{\text{POD}} = \sum_{i=1}^{n} \frac{\left\|\boldsymbol{x}^i - \hat{\boldsymbol{x}}^i\right\|^2}{\left\|\boldsymbol{x}^i\right\|^2}. \tag{2.11}$$

The Eckart-Young theorem [29] states that the POD basis consisting of the first $k$ left singular vectors found from the SVD of $\boldsymbol{S}$ minimizes this error amongst all orthonormal bases of rank $k$. This is a desirable feature of the POD basis, as it maximizes the amount of solution space information for a given reduced dimension.

### 2.2.2 Example: NACA 0012 Airfoil

An advantage of using an orthogonal basis is that the basis vectors represent unique directions in the solution space, and as a result there is no information overlap between vectors. The basis vectors are also interpretable, and often referred to as *POD modes*. The modes can be visualized to understand important physical features. Using a NACA 0012 airfoil, steady, incompressible, turbulent flow is simulated at five different angles of attack $\alpha = [0, 2, 4, 6, 8]$. Figure 2.2 shows contours of the velocity magnitude as well as the first three POD modes. We can assume that the first mode represents an average flow field from the training data, the second represents the stagnation point at the leading edge, and the third separation caused by increasing the angle of attack. The expansion coefficients used to reconstruct or approximate designs within the solution space are also informative of the relative importance of each mode. Figure 2.3 contains a plot of the singular values for each of the five

POD modes using a log-scale. The singular values decay rapidly, and the relative information content $E$ from the first three modes is equal to 0.9994.



(a) CFD, $\alpha = 0$ (b) POD Mode 1

(c) CFD, $\alpha = 4$ (d) POD Mode 2

(e) CFD, $\alpha = 8$ (f) POD Mode 3

Figure 2.2: Velocity magnitude contours from CFD (left) and the first three POD modes (right) for a NACA 0012 case.

## 2.3 Non-intrusive ROM

This method relies on a functional dependence $\boldsymbol{a} = f(\boldsymbol{\mu})$ between the design parameters $\boldsymbol{\mu}$ and the expansion coefficients $\boldsymbol{a}$. Non-intrusive ROMs use a regression model to predict the expansion coefficients given the design parameters, leading to a fully data-driven approach that does not incorporate the governing physics of the

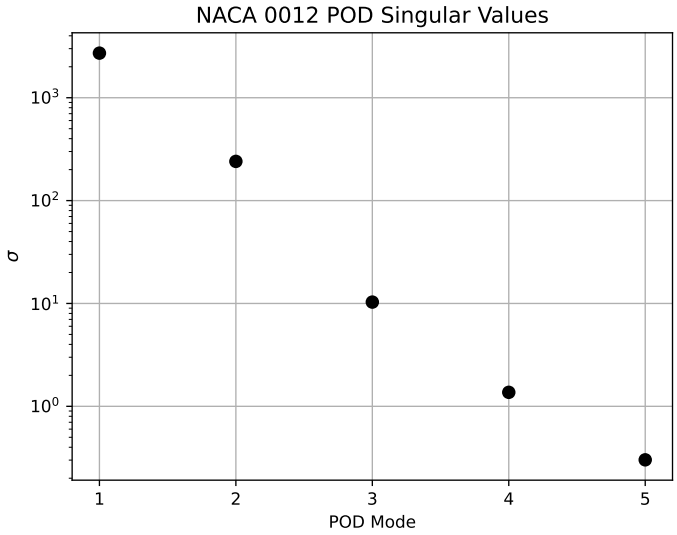



Figure 2.3: Singular values associated with the POD basis for a NACA 0012 case.

FOM. There are many popular regression models used in non-intrusive ROMs, including artificial neural networks [13], radial basis function (RBF) interpolation [30], and Gaussian process regression (GPR) [31], sometimes referred to as Kriging. While ANNs can provide superior performance to other regression methods, they require large amounts of training data to make accurate predictions at unseen points. Since ROMs are typically trained using a small amount of data, this limits the use of ANNs for predicting expansion coefficients. RBF and GPR are both similar in that they are non-parametric models that use kernel functions to model similarity between data points, leading to smooth functions. In non-intrusive ROMs, an individual regression model is usually used for each coefficient in $\boldsymbol{a}$, leading to $k$ different regression models,

$$f_i(\boldsymbol{\mu}) : \mathbb{R}^p \rightarrow \mathbb{R}, i \in [1, 2, \cdots k]. \tag{2.12}$$

GPR utilizes a probabilistic framework using gradient-based optimization to determine kernel hyperparameters, which often leads to better generalization than RBF.

### 2.3.1 Gaussian Process Regression

A brief introduction to GPR is given in this section, and a more complete overview can be found in a work by Rasmussen [32]. A Gaussian process (GP) is a set of random variables that follow a joint Gaussian distribution. In GPR, it is assumed that data are generated according to a GP with mean function $m$ and covariance function $\kappa$,

$$f(\boldsymbol{\mu}) \sim \mathrm{GP}\left(m(\boldsymbol{\mu}), \kappa(\boldsymbol{\mu}, \boldsymbol{\mu}^*)\right), \tag{2.13}$$

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

$$y = f(\boldsymbol{\mu}) + \delta, \tag{2.14}$$

where $\xi_y^2$ is a very small non-dimensional term (typically on the magnitude of $10^{-10}$) that helps keep the covariance matrix positive definite to prevent numerical instabilities. Using a finite amount of training data $\{\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{a}}\}$ and a prior joint Gaussian on the data, the prediction at a point $\boldsymbol{\mu}^*$ is given by

$$\begin{bmatrix} \bar{\boldsymbol{a}} \\ f(\boldsymbol{\mu}^*) \end{bmatrix} \sim \mathcal{N}\left( \begin{bmatrix} m(\bar{\boldsymbol{\mu}}) \\ m(\bar{\boldsymbol{\mu}}) \end{bmatrix}, \begin{bmatrix} \kappa(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\mu}}) + \xi_y^2 \boldsymbol{I} & \kappa(\bar{\boldsymbol{\mu}}, \boldsymbol{\mu}^*) \\ \kappa(\boldsymbol{\mu}^*, \bar{\boldsymbol{\mu}}) & \kappa(\boldsymbol{\mu}^*, \boldsymbol{\mu}^*) \end{bmatrix} \right), \tag{2.15}$$

where $\boldsymbol{I}$ is the identity matrix. Using the properties of conditional Gaussian distributions, the conditional expectation of $f(\boldsymbol{\mu}^*)$ is given as

$$\mathbb{E}(f(\boldsymbol{\mu}^*)|\bar{\boldsymbol{a}}) = \kappa(\boldsymbol{\mu}^*, \bar{\boldsymbol{\mu}})(\kappa(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\mu}}) + \xi_y^2 \boldsymbol{I})^{-1} \left(\bar{\boldsymbol{a}} - m(\bar{\boldsymbol{\mu}})\right). \tag{2.16}$$

In practice, the mean function $m$ is set to the mean of the training outputs,

$$m(\bar{\boldsymbol{\mu}}) = \frac{\sum_{i=1}^{n} \bar{a}^i}{n} \tag{2.17}$$

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

$$\mathcal{Z}_j^i = \frac{\mu_j^i - m(\boldsymbol{\mu}_j)}{s_j}, \tag{2.18}$$

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

$$\kappa(\boldsymbol{\mu}, \boldsymbol{\mu}^*) = \exp\left(-\frac{d(\boldsymbol{\mu}, \boldsymbol{\mu}^*)^2}{2l^2}\right), \tag{2.19}$$

where $d$ is the Euclidean distance function and $l$ represents the length scale, which is a hyperparameter. The predictive performance of the regression model is highly dependent upon the values of the hyperparameters. Gradient-based optimization is used to maximize the marginal log-likelihood of the training data to obtain an optimal set of hyperparameters, $\theta_{\text{opt}}$,

$$\theta_{\text{opt}} = \underset{\theta}{\text{argmax}} \log p(\bar{\boldsymbol{a}}|\bar{\boldsymbol{\mu}}, \theta) = -\frac{1}{2}\bar{\boldsymbol{a}}^T(\kappa(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\mu}}) + \xi_y^2\boldsymbol{I})^{-1} - \frac{1}{2}\log|\kappa(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\mu}}) + \xi_y^2\boldsymbol{I}| - \frac{n}{2}\log 2\pi. \tag{2.20}$$

### 2.3.2 POD-GPR ROM

This section describes the POD-GPR ROM, which is also sometimes referred to as the POD-Kriging ROM [31] in the literature. The offline stage first involves computing the high fidelity solutions for design parameters $\boldsymbol{\mu} \in \boldsymbol{\mathcal{U}}_{\text{train}} \in \mathbb{R}^{n \times p}$ and assembling them into $\boldsymbol{S}$. The truncated SVD of the snapshot matrix is obtained to compute $\boldsymbol{\Psi}$. When using the POD-GPR ROM, the number of basis vectors $k$ is often set to the number of training samples $n$, as the accuracy of the ROM tends to level off as $k$ approaches $n$ [28] and training the GPR models is inexpensive. After the expansion coefficients are calculated for the training data, the individual

GPR models are trained and saved for use in the online stage. Given a set of unseen design parameters $\boldsymbol{\mu}^*$, an approximation of the expansion coefficients $\tilde{\boldsymbol{a}}^*$ is computed. Matrix-vector multiplication of the POD-basis and expansion coefficients is then used to obtain an approximate full-order solution $\tilde{\boldsymbol{x}}$. The POD-GPR method is outlined in Algorithm 1.

**Algorithm 1** Offline and online stages of POD-GPR ni-ROM

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

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

## 2.4 Projection-based ROM

In contrast to non-intrusive ROMs, projection-based ROMs require access to the physics solver during the online stage. In CFD applications, the flow residuals $\boldsymbol{R} \in \mathbb{R}^N$ are evaluated to provide convergence information. Evaluating the full-order residuals for a given approximated state is expensive and leads to p-ROMs incurring significantly larger inference costs when compared to ni-ROMs. Generally, multiple residual evaluations are required during the online stage of a p-ROM. To mitigate this issue, many p-ROMs utilize hyper-reduction, which allows for approximating the full-order residuals by evaluating the residuals at a small subset of cells. A popular

hyper-reduction method for POD-based p-ROMs is the discrete empirical interpolation method (DEIM) [33]. DEIM finds an optimal small subset of evaluation cells and uses an interpolating basis of the full-order residuals to make approximations. Although DEIM does offer benefits, its implementation is non-trivial for many solvers and can lead to poor results for highly nonlinear problems [13].

The projection-based ROM that is introduced here comes from our previous work [19]. The study includes results for a p-ROM including DEIM, although the full computational gains from the method are not realized due to difficulties in implementation. Results for a *brute-force* p-ROM that explicitly computes the full-order residual are also included. This ROM is shown to offer significant speed-up over CFD simulations for two external aerodynamics problems while providing high levels of accuracy.

When applied to fluid dynamics problems, projection-based ROMs solve a low-dimensional version of the FOM through the full-order residuals $\boldsymbol{R}$. The FOM governing equations can be expressed in terms of $\boldsymbol{R}$ and are a function of the state $\boldsymbol{x}$ and design parameters $\boldsymbol{\mu}$,

$$\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{\mu}) = 0. \tag{2.21}$$

The residuals can also be approximated using the POD basis $\boldsymbol{\Psi}$ state approximation,

$$\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{\mu}) \approx \boldsymbol{R}(\boldsymbol{\Psi a}, \boldsymbol{\mu}). \tag{2.22}$$

For a new set of design parameters $\boldsymbol{\mu}^*$, the expansion coefficients $\boldsymbol{a}^*$ that satisfy Equation 2.22 are desired. A least-squares Petrov-Galerkin (LSPG) approach [34] is used for the projection-based ROM. Specifically, the expansion coefficients that minimize the $L^2$ norm of the approximate residuals are solved for,

$$\min_{\boldsymbol{a}} L^2 = \|\boldsymbol{R}(\boldsymbol{\Psi a})\|^2 . \tag{2.23}$$

Most ROMs use a regular Galerkin projection [35], which projects the FOM governing equations onto the state POD basis. The LSPG ROM formulation involves minimizing the $L^2$ norm of the residuals, which has been shown to lead to better stability and convergence for many nonlinear problems. At the minimum point, $\frac{\partial L^2}{\partial \boldsymbol{a}} = 0$, and substituting $\boldsymbol{x} = \boldsymbol{\Psi a}$ gives us an equation for a $k$-dimensional residual $\boldsymbol{r}$,

$$\boldsymbol{r} = \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\boldsymbol{\Psi}\right]^T \boldsymbol{R} = 0. \tag{2.24}$$

A Newton-Krylov approach is used to solve Equation 2.24. First, a reference full-order state $\boldsymbol{x}_{\text{ref}}$ from the training data is used to compute an initial value for $\boldsymbol{a}$. The reference state is chosen to correspond to the training point in $\boldsymbol{\mathcal{U}}_{\text{train}}$ closest in Euclidean distance to $\boldsymbol{\mu}^*$ as the initial condition should be relatively close to the solution. The low-dimensional residual is computed as well as the Jacobian $\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}}$ using a finite-difference method. A linear equation is then solved for $\Delta \boldsymbol{a}$ using the generalized minimal residual (GMRES) method implemented in PETSc [36], a C++ library for scalable scientific computing. A backtracking line search is performed to compute the updated expansion coefficients $\boldsymbol{a}^{n+1}$, with the requirement that the norm of $\boldsymbol{r}$ decreases. If this does not happen, the step size is reduced by a factor of two, until $m_{\max}$ iterations are reached, which is typically set to 5. This process is repeated until the norm of $\boldsymbol{r}$ is less than a prescribed tolerance $r_{\text{tol}}$ or the Newton-Krylov algorithm reaches $n_{\max}$ iterations. Although only the norm of $\boldsymbol{r}$ is driven to 0, the full-order residual is also expected to decrease according to the LSPG formulation. The Jacobian-matrix product from Equation 2.24 is explicitly computed for each Newton-Krylov iteration using a matrix-free method. Individual basis vectors $\boldsymbol{\psi}^i$ are extracted from $\boldsymbol{\Psi}$ and a matrix-free approach is used to compute the matrix-vector

product,

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \boldsymbol{\psi^i} \approx \frac{\boldsymbol{R}(\boldsymbol{x} + \epsilon \boldsymbol{\psi^i}) - \boldsymbol{R}(\boldsymbol{x})}{\epsilon}, \tag{2.25}$$

where $\epsilon = 10^{-6}$ is a small step size relative to the residual magnitudes tested. The full-order residuals are evaluated once for each of the $k$ columns in $\boldsymbol{\Psi}$ using DAFoam [37], an open-source adjoint derivative computation framework for OpenFOAM that uses automatic differentiation.

The offline stage of the p-ROM is similar to that of the ni-ROM in assembling the snapshot matrix and computing the POD basis. The p-ROM also uses a preconditioner (PC) matrix $\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}}\right]_{\text{PC}}$ to solve the linear equation using GMRES. The PC matrix is computed in the offline stage at a design point in the training set that is closest to the average of the training design parameters. The matrix is not updated during the online stage as the size of the linear system being solved in small and it is not difficult to solve. The entire p-ROM algorithm is outlined in Algorithm 2.

**Algorithm 2** Offline and online stages of projection-based ROM

**function** pROM_ OFFLINE($\boldsymbol{\mathcal{U}}_{\text{train}}$)
    Compute high-fidelity solutions for $\boldsymbol{\mu} \in \boldsymbol{\mathcal{U}}_{\text{train}}$ by solving FOM and assemble into $\boldsymbol{S}$
    Calculate truncated SVD of snapshot matrix to obtain POD basis $\boldsymbol{\Psi}$
    Compute the preconditioner matrix $\left[\dfrac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}}\right]_{\text{PC}}$
**end function**

**function** pROM_ ONLINE($\boldsymbol{\mu}^*, \boldsymbol{\Psi}, \boldsymbol{x}_{\text{ref}}, \left[\dfrac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}}\right]_{\text{PC}}$)
    $\boldsymbol{a}^0 = \boldsymbol{\Psi}^T \boldsymbol{x}_{\text{ref}}$ ▷ Compute the initial reduced state variable vector
    **for** $i \in \{0, 1, \ldots, i_{\max}\}$ **do** ▷ Main loop to compute $\boldsymbol{a}^*$
        $\left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\boldsymbol{\Psi}\right]^T \leftarrow \boldsymbol{a}^i, \boldsymbol{\Psi}$ ▷ Compute the Jacobian-matrix product
        $\boldsymbol{r} \leftarrow \boldsymbol{a}^i, \boldsymbol{\mu}^*, \left[\dfrac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\boldsymbol{\Psi}\right]^T$ ▷ Compute the reduced residual
        $\dfrac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}} \leftarrow \boldsymbol{a}^i, \boldsymbol{\mu}^*$ ▷ Compute the reduced Jacobian matrix
        $\Delta \boldsymbol{a}^i \leftarrow \dfrac{\partial \boldsymbol{r}}{\partial \boldsymbol{a}_{(\boldsymbol{a}^i, \boldsymbol{\mu}^*)}} \Delta \boldsymbol{a}^i = -\boldsymbol{r}_{(\boldsymbol{a}^i, \boldsymbol{\mu}^*)}$ ▷ Solve the linear equation to get $\Delta \boldsymbol{a}^i$
        **for** $m \in \{0, 1, \ldots, m_{\max}\}$ **do** ▷ Backtracking line search
            $\alpha = 1.0$
            $\boldsymbol{a}^{i+1} = \boldsymbol{a}^i + \alpha \Delta \boldsymbol{a}^i$ ▷ Update $\boldsymbol{a}^i$ with step $\alpha$
            **if** $||\boldsymbol{r}(\boldsymbol{a}^{i+1})||^2 < ||\boldsymbol{r}(\boldsymbol{a}^i)||^2$ **then** ▷ Reduced residual decreases, line search done
                **break**
            **else** ▷ Reduced residual not dropping, decrease $\alpha$
                $\alpha = 0.5\alpha$
            **end if**
        **end for**
        $\boldsymbol{a}^* = \boldsymbol{a}^{i+1}$ ▷ Assign the latest solution to $\boldsymbol{a}^*$
        **if** $||\boldsymbol{r}(\boldsymbol{a}^{i+1})||^2 < r_{\text{tol}}$ **then** ▷ Prescribed tolerance satisfied, exit the main loop
            **break**
        **end if**
    **end for**
    $\boldsymbol{x}^* \approx \boldsymbol{\Psi}\boldsymbol{a}^*$ ▷ Compute the new full-order state variable vector
    **return** $\tilde{\boldsymbol{x}}^*$
**end function**

# CHAPTER III

# Data Selection for ROMs Using Isomap

This chapter introduces data selection algorithms for POD-based ROMs using Isomap, a versatile technique for nonlinear dimensionality reduction. A popular method for generating sets of training design parameters from a multi-dimensional parameter space is Latin hypercube sampling [38] (LHS), a statistical method that aims to maximize the distance and minimize the correlation between generated samples. LHS uses a stratified sampling technique that generally leads to uniform coverage of the parameter space. While this ensures that different areas of the parameter space are equally represented, the physical regimes for a given problem may present a non-linear variation within it. This can lead to certain physical regimes being over or under-represented in the training data and result in poor predictive performance at some unseen points. Two algorithms are presented in this chapter, one that uses Isomap to produce locally selected POD bases to construct ROMs, and an adaptive sampling algorithm that iteratively selects design parameters to increase the diversity of the training data. The first algorithm shows that Isomap is effective at separating data by physical regime, while the latter outperforms LHS for both the ni-ROM and p-ROM, and results in better or similar predictive performance given a lower computational budget for generating full-order training snapshots.

Parts of this chapter appear in or are adapted from our previously published papers [39, 20].

## 3.1 Latin Hypercube Sampling

Latin hypercube sampling uses random uniform sampling to divide each dimension of a multi-dimensional parameter space into equally probable intervals. Given $n$ desired samples, $n$ intervals are constructed per dimension and their centers are chosen such that the distance between points is maximized. The center of each interval is filled exactly once per dimension in a random order to ensure that the parameter space is sufficiently covered. This is in contrast to strictly random uniform sampling, which does not construct intervals and can lead to clustering of points in certain regions of the parameter space and insufficient overall coverage. Given the generated design parameters $\boldsymbol{\mathcal{U}}_{\text{train}} \in \mathbb{R}^{n \times p}$, the cross-column Pearson correlation coefficient matrix $\boldsymbol{C} \in \mathbb{R}^{p \times p}$ is obtained. Since the random generation of the design parameters is computationally inexpensive, a brute-force iterative approach is used to minimize the correlation amongst the samples until a maximum number of iterations is reached. The minimized criteria used in this work is the maximum value of the off-diagonal values of $|\boldsymbol{C}|$, which helps keep the generated points spread apart and equally distributed. Figure 3.1 shows an example of both LHS and random uniform sampling applied to a two-dimensional parameter space with bounds give by $\mu_1, \mu_2 \in [-1, 1]$ and $n = 40$ samples. LHS leads to signifcantly better overall coverage of the parameter space. Random uniform sampling leaves a number of unfilled gaps within the parameter space in addition to many closely clustered points. LHS avoids this by using centers within each interval that are linearly spaced.

## 3.2 Isomap

Isomap [40] is a nonlinear dimensionality reduction method that estimates a low-dimensional manifold of high-dimensional data. Nonlinear dimensionality reduction methods are used to capture complex patterns present in high-dimensional data in a

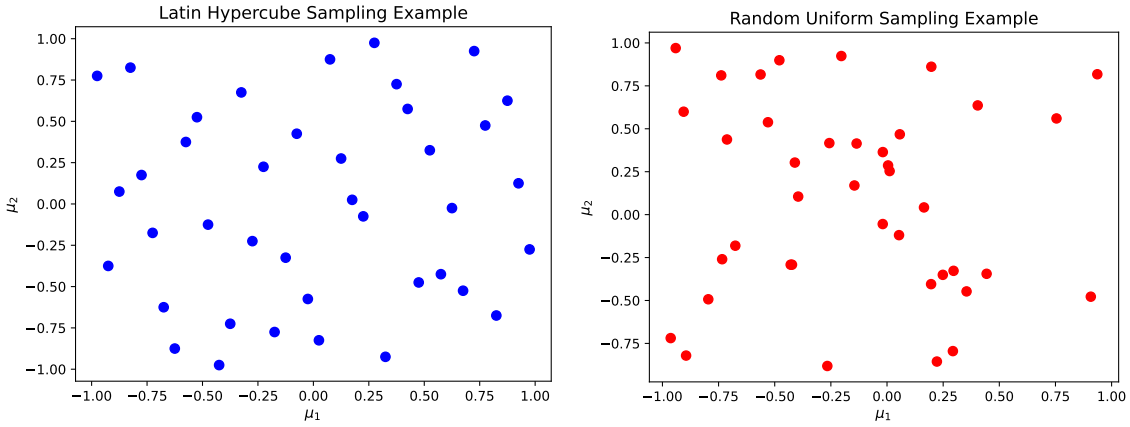



Figure 3.1: Comparison of LHS with random unform sampling.

low-dimensional space. Most real-world datasets exhibit nonlinear relationships between variables and data points. Linear methods like principal components analysis (PCA), which is closely related to the SVD, assume that there are linear relationships present within the dataset, when this is often not the case. Nonlinear dimensionality algorithms are useful for discovering the underlying intrinsic geometry, or manifold, of high-dimensional data. There are many different nonlinear dimensionality reduction algorithms, including Local Linear Embedding [41] (LLE) and t-Distributed Stochastic Neighbor Embedding [42] (t-SNE). Isomap is adept at preserving both the local and global structure of data, which makes it applicable to a wide range of problems, including gene and protein expressions [43], climate data [44], and turbulent combustion [45], where it is shown to outperform PCA in various tasks. Given a data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times N}$ with $n$ observations and dimensionality $N$, Isomap provides a latent representation matrix $\boldsymbol{W} \in \mathbb{R}^{n \times r}$, where $r \ll N$. The latent representation $\boldsymbol{w} \in \mathbb{R}^{r}$ for each sample represents its position on the low-dimensional manifold. Isomap aims to produce a manifold that preserves the geodesic distances between points in the high-dimensional space given a nearest neighbor graph $\mathcal{G}$ of the high-dimensional training data where there are connections to the $\mathcal{K}$ nearest neighbors and the edges are weighted by the distance between points. The geodesic distance between two

points is the shortest distance between them on the graph. The Euclidean distance is usually used as the metric of distance for the nearest neighbor graph, which can be computed using methods such as Dijkstra's algorithm [46]. Once the geodesic distances between points are assembled into a distance matrix $\boldsymbol{D} \in \mathbb{R}^{n \times n}$, multi-dimensional scaling (MDS) [47] is applied to obtain $\boldsymbol{W}$. MDS is an algorithm for producing low-dimensional representations that preserve high-dimensional pairwise distances. Using optimization, a stress function $\omega$ that represents the discrepancy between the pairwise distances in the low and high dimensional space is minimized. The stress function is given as

$$\omega(\boldsymbol{D}) = \sum_{i \neq j = 1, \cdots, n} \left(\boldsymbol{D}[i,j] - d(\boldsymbol{W}[i,:], \boldsymbol{W}[j,:])\right)^2, \tag{3.1}$$

where $d$ is the Euclidean distance between two latent representations. There are many ways to optimize the stress function in MDS. Isomap is implemented in this work using scikit-learn [48], an open-source machine learning library for Python. The scikit-learn implementation of Isomap uses the SMACOF (Scaling by MAjorizing a COmplicated Function) algorithm [49], which iteratively adjusts positions of points in the low-dimensional space using a majorization-minimization technique. The coordinates of the low-dimensional points are first initialized randomly, and the SMACOF algorithm is iteratively repeated until $w(\boldsymbol{D})$ is minimized. This leads to a manifold that represents the intrinsic geometry of the data. The computational aspects of the SMACOF algorithm are outside the scope of this work, and the original paper can be consulted for more detail.

The Isomap method is described in Algorithm 3. Figure 3.2 shows the manifold resulting from Isomap being applied to the swiss roll dataset, a popular 3-dimensional benchmark problem for dimensionality reduction methods. The resultant 2-dimensional manifold successfully separates points based on their positions along the roll, rather than Euclidean distances. Points which are relatively close to each

other in terms of Euclidean distance can be very far from each other along the roll. By applying Isomap to the dataset, the swiss roll is successfully *unrolled* to discover an intrinsic 2-dimensional geometric representation. When using Isomap, an important hyperparameter to consider is $\mathcal{K}$, the number of nearest neighbors used to construct $\mathcal{G}$. If $\mathcal{K}$ is too large, the local structure of the data manifold isn't represented well, while values of $\mathcal{K}$ that are too small can lead to inaccurate geodesic distances because of an outsized focus on small regions surrounding data points and neglect of the global structure of the data.

**Algorithm 3** Isomap

**Input:** $\boldsymbol{X}$ ▷ High-dimensional data matrix
**Output:** $\boldsymbol{W}$ ▷ Low-dimensional latent representation matrix
1: $\mathcal{G} \leftarrow \boldsymbol{X}$ ▷ Compute a nearest neighbor graph from each observation in $\boldsymbol{X}$ with the edges weighted by distance.
2: $\boldsymbol{D} \leftarrow \mathcal{G}$ ▷ Compute the pairwise geodesic distances between all points.
3: $\boldsymbol{W} \leftarrow \text{MDS}(\boldsymbol{D})$ ▷ Apply MDS and obtain the latent representation matrix.

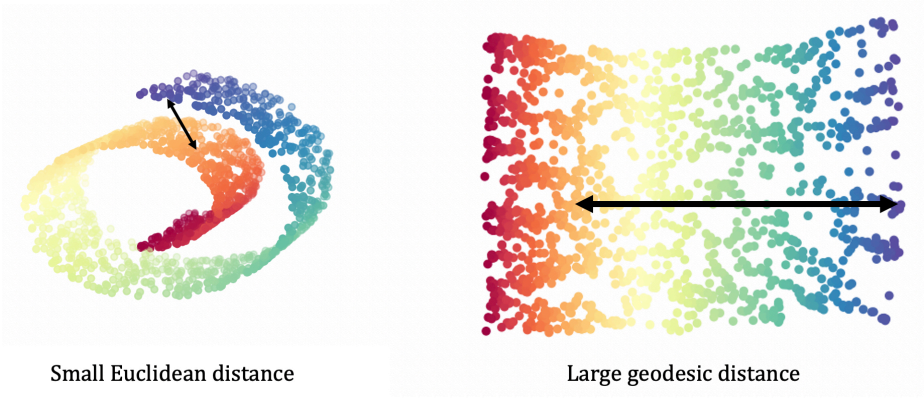



Figure 3.2: 2-dimensional manifold (right) obtained from Isomap being applied to the swiss roll dataset (left).

## 3.3 Local POD Bases

For problems exhibiting highly nonlinear behavior throughout the parameter space, POD-based ROMs can suffer performance issues. The number of basis vectors required for approximations of reasonable accuracy becomes large. For projection-based

ROMs, this can lead to numerical instabilities during the online stage due to the increased size of the linear systems being solved. While non-intrusive ROMs generally use a separate regression model for each expansion coefficient, the underlying functions can exhibit low smoothness for nonlinear problems and be difficult to model. To alleviate these issues, the use of local POD bases is often employed. This involves partioning the training data by physical regime into local snapshot matrices, from which local POD bases are obtained. An example from Amsallem et al. [50] computes local POD bases for a projection-based ROM in the offline stage by grouping the training snapshots using $k$-means clustering. During the online stage, the local basis is chosen according to the subregion of the solution space where the current high-dimensional solution lies. In a work [51] by Dupuis et al., the authors use Gaussian mixture models to pre-compute local POD bases for use in a POD-GPR non-intrusive ROM. When applied to a turbulent flow case involving both subsonic and transonic flow past an airfoil, the method is shown to offer signifcantly improved predictive performance by partioning the parameter space by flow regime by detecting the presence of shock waves. By using data that are physically similar to each other, the expansion coefficients from the local POD bases are much easier to predict even if there is a lower amount of training data available as the GPR models are much smoother.

In this section, I introduce an algorithm that uses Isomap to compute local POD bases at unseen points in the parameter space for use with the POD-GPR ni-ROM. While the previously mentioned examples pre-compute the local bases in the offline stage, this algorithm computes a tailored POD basis for each unseen point in the online stage. Although this involves computing the SVD, the cost of this is typically negligible when compared to solving the FOM.

In constructing local POD bases, the goal is to use only training samples that will be physically similar to a given prediction point. The design parameters themselves may not be informative to this end, especially when the number of parameters is large.

Applying Isomap to the transpose of the snapshot matrix, $\boldsymbol{S}^T$, a low-dimensional manifold of the training data can be estimated to obtain a latent representation $\boldsymbol{w}$ for each training sample. Using GPR, a regression model can be obtained to estimate the latent representations $\boldsymbol{w}^*$ for unseen design parameters $\boldsymbol{\mu}^*$. A separate GPR model is used to predict each latent variable, as is done with the expansion coefficients in the POD-GPR ROM. Guided by the general idea of nonlinear dimensionality reduction that points closer to each other in the lower dimensional space are more similar, a local snapshot matrix $\boldsymbol{S}_L \in \mathbb{R}^{N \times l}$ is created for an unseen point $\boldsymbol{\mu}^*$ consisting of the training samples corresponding to the $l$ closest latent representations to $\boldsymbol{w}^*$ measured by Euclidean distance. For the Euclidean distance to be a strong metric of similarity, $r$ is set to 2. When the latent variable dimension grows above 2, the Euclidean distances between points become less meaningful due to increased sparsity and the curse of dimensionality. The local POD basis $\boldsymbol{\Psi}_L \in \mathbb{R}^{N \times l}$ is found from the SVD of $\boldsymbol{S}_L$. Since ROMs using the POD-GPR method are most accurate when using all of the basis vectors, $k$ is set to $l$. Using this local snapshot matrix, the POD-GPR method can be used to make a prediction of the unknown state $\boldsymbol{x}^*$. This process is outlined in Algorithm 4.

### 3.3.1 Lid-driven Cavity

The test case used to demonstrate the performance of the local POD basis selection algorithm is a geometricaly and physically parameterized lid-driven cavity flow, a popular benchmark problem for CFD solvers. Three parameters control the computational domain $\Omega$ and one parameter controls the kinematic viscosity through the Reynolds number. A version of this problem has previously appeared in a work by Hesthaven and Ubbiali [13]. Figure 3.3 shows the boundary conditions on each edge $\Gamma_i, i \in [1, 2, 3, 4]$ of the domain; $u$, $v = 0$ on all of the edges except $\Gamma_1$, where $u = 1, v = 0$. The pressure gradient, $\nabla p$, is set to 0 on all of the edges. The reference

**Algorithm 4** POD-GPR method using local POD bases found using Isomap

**Input:** $\boldsymbol{S}, \boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{\mu}^*$ ▷ Snapshot matrix, design parameters, local POD basis dimension

**Output:** $\tilde{\boldsymbol{x}}^*$ ▷ Approximation of full-order state

$k = l$ ▷ ROM dimension is equal to local POD basis dimension

$\boldsymbol{W} \leftarrow \text{Isomap}(\boldsymbol{S}^T)$ ▷ Compute latent representations of training snapshots using Isomap

$\boldsymbol{w}^* \leftarrow \text{GPR}(\boldsymbol{W}, \boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{\mu}^*)$ ▷ Approximate latent representation of prediction point using GPR

$\boldsymbol{d} \leftarrow \text{dist}(\boldsymbol{w}^*, \boldsymbol{W})$ ▷ Compute pairwise Euclidean distances between latent representations

$\boldsymbol{i} \leftarrow \text{argsort}(\boldsymbol{d})[1:k]$ ▷ Find indices of $k$ closest training points to current prediction

$\boldsymbol{S}_L \leftarrow \boldsymbol{S}[:, \boldsymbol{i}]$ ▷ Assemble local snapshot matrix

$\boldsymbol{\Psi}_L \leftarrow \text{SVD}(\boldsymbol{S}_L)$ ▷ Compute local POD basis from SVD of local snapshot matrix

$\boldsymbol{A}_L \leftarrow \left(\boldsymbol{\Psi}_L^T \boldsymbol{S}_L\right)^T$ ▷ Compute expansion coefficients of states in local snapshot matrix

**for** $m \in \{1, 2, \ldots, k\}$ **do**

    $\tilde{\boldsymbol{a}}^*[m] \leftarrow \text{GPR}(\boldsymbol{A}_L[:, m], \boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{\mu}^*)$ ▷ Approximate each expansion coefficient using GPR

**end for**

$\tilde{\boldsymbol{x}}^* \leftarrow \boldsymbol{\Psi}_L \tilde{\boldsymbol{a}}^*$ ▷ Compute approximated full-order state

pressure is set to 0 in the bottom left corner of the domain. The parameterization of the geometry is also shown, involving three parameters which change the length of the horizontal ($\mu_1$) and slanting edges ($\mu_2$) as well as the slanting angle ($\mu_3$). The Reynolds number, $Re$ ($\mu_4$), is the fourth parameter, and is related to the kinematic viscosity $\nu$ as

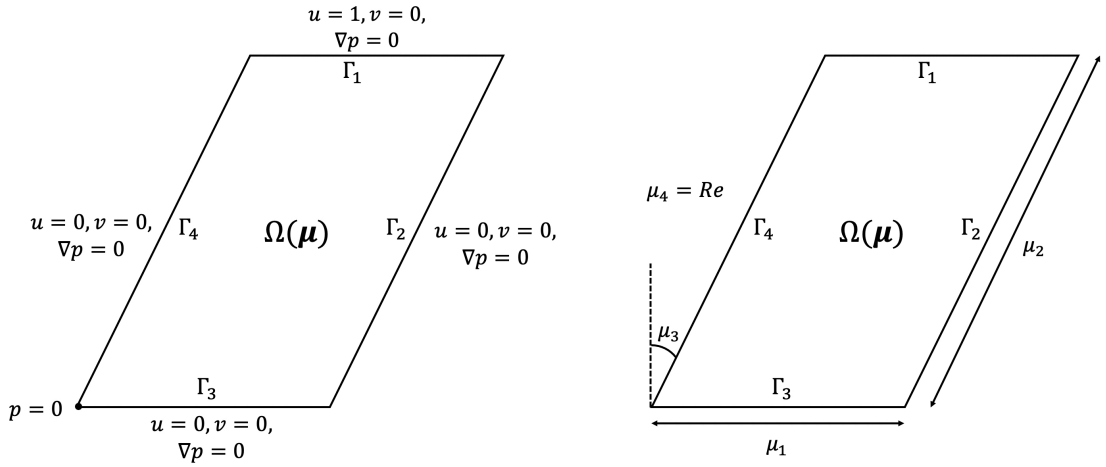
$$Re = \frac{\max(\mu_1, \mu_2)}{\nu(\boldsymbol{\mu})}. \tag{3.2}$$

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

$$\begin{aligned}
\mu_1 &\in [1, 2], \\
\mu_2 &\in [1, 2], \\
\mu_3 &\in [-\frac{\pi}{4}, \frac{\pi}{4}], \\
\mu_4 &\in [100, 600].
\end{aligned}$$

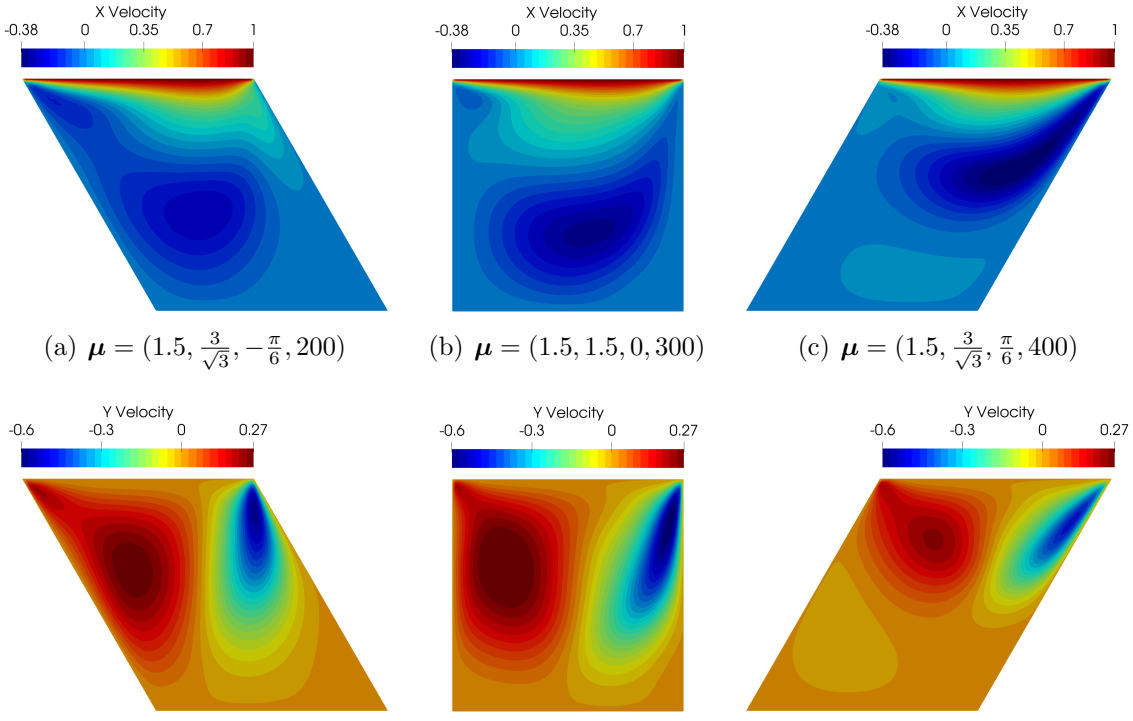
The computational mesh consists of $64 \times 64$ cells uniformly distributed in the $x$ and $y$ directions, and one cell spanning the $z$ direction, resulting in $N = 4096$. The full-order states $u$ and $v$ are used for the ROM, contours of which are shown in Figure 3.4 at three different sets of design parameters. A sharp gradient in $u$ exists at the top of the domain, and a vortex moves throughout the cavity as the design parameters change. This vortex is also shown moving throughout the cavity shown in the contours of $v$, varying in shape and size with the design parameters. The relationship between both $u$ and $v$ and $\boldsymbol{\mu}$ is shown to be highly nonlinear, making this a difficult prediction problem in the context of ROMs.

Separate ROMs are used for predicting $u$ and $v$. The Matern kernel [52] is used for GPR to predict both expansion coefficients and latent variables, and is given as




(a) Lid-driven cavity boundary conditions. (b) Lid-driven cavity design parameters.

Figure 3.3: Schematics describing the lid-driven cavity problem.




(a) $\boldsymbol{\mu} = (1.5, \frac{3}{\sqrt{3}}, -\frac{\pi}{6}, 200)$ (b) $\boldsymbol{\mu} = (1.5, 1.5, 0, 300)$ (c) $\boldsymbol{\mu} = (1.5, \frac{3}{\sqrt{3}}, \frac{\pi}{6}, 400)$

(d) $\boldsymbol{\mu} = (1.5, \frac{3}{\sqrt{3}}, -\frac{\pi}{6}, 200)$ (e) $\boldsymbol{\mu} = (1.5, 1.5, 0, 300)$ (f) $\boldsymbol{\mu} = (1.5, \frac{3}{\sqrt{3}}, \frac{\pi}{6}, 400)$

Figure 3.4: Contours of $u$ (top) and $v$ (bottom) for the lid-driven cavity problem at three different sets of design parameters.

$$\kappa(\boldsymbol{\mu}, \boldsymbol{\mu}^*) = \frac{1}{\Gamma(\nu)2^{\nu-1}} \left( \frac{\sqrt{2\nu}}{l} d(\boldsymbol{\mu}, \boldsymbol{\mu}^*) \right)^{\nu} K_\nu, \tag{3.3}$$

where $d$ is the Euclidean distance function, $\Gamma$ is the gamma function, and $K_\nu$ is the modified Bessel function of the second kind. The Matern kernel was chosen through a trial-and-error process and it was found to offer the best performance in predicting both the expansion coefficients and latent variables. The set of hyperparameters $\theta$ of the Matern kernel are $l$ and $\nu$, which control the length scale and smoothness respectively. The number of nearest neighbors $\mathcal{K}$ to construct the nearest neighbor graph is set to 5 to strike a good balance between acknowledging the local and global structure of the data.

360 sets of design parameters are generated using LHS. The ROM constructed using local POD bases found using Isomap is referred to as the *Isomap local ROM*, while the ROM that uses all of the available training data will be referred to as the *global ROM*. Additionally, results are shown for a *random local ROM* that selects POD bases from a single random subset of the training data. The metric of performance that is used is the relative $L^2$ error $\epsilon$ between the approximated state $\tilde{\boldsymbol{x}}$ and true state $\boldsymbol{x}$,

$$\epsilon = \frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2}{\|\boldsymbol{x}\|^2}. \tag{3.4}$$

To assess the performance of the ROM over the entire dataset, 5-fold cross-validation is used to create 5 different splits of the data into 288 training samples and 72 test samples. For each fold, the average relative error $\bar{\epsilon}_i, i \in [1, 2, 3, 4, 5]$ is taken over all of the test samples. The mean of these average errors is then taken over all of the folds to report a cross-validation error $\bar{\epsilon}_{\text{CV}}$

$$\bar{\epsilon}_{\text{CV}} = \frac{\sum_{i=1}^{5} \bar{\epsilon}_i}{5}. \tag{3.5}$$

Figure 3.5 shows plots of $\bar{\epsilon}_{\text{CV}}$ against the ROM dimension $k \in [50, 250]$ for the Isomap and random local ROMs. The global ROM relative cross-validation error with

$k = 288$ is also plotted for comparison. Compared to the global ROM, we can see that using Isomap to generate local POD bases offers better predictive performance in both $u$ and $v$ over a large number of ROM dimensions. When $k$ is low, the amount of data available may not be sufficient in both the quality of the POD basis and GPR model to outperform the global ROM. As $k$ grows larger, the performance offered by using local POD bases is greater than that of the global ROM, with peak performance occuring at around $k = 120$ for $u$ and $k = 150$ for $v$. Although the gains in performance are modest, signficantly less training data is being used for the GPR models, which suggests that the functions being modeled are much smoother. Although the local POD basis $\boldsymbol{\Psi}_L$ does not offer a lower projection error compared to $\boldsymbol{\Psi}$, the ROM is more accurate when working with physically similar data.

Generating random local POD bases does not exhibit this behavior; we can see that the global ROM outperforms this method over all selected values of $k$, especially when $k$ is low. The random local ROM errors approach that of the global ROM as $k$ grows large, with the decay in $\bar{\epsilon}_{\text{CV}}$ showing expected behavior. This further suggests that Isomap is highly effective in identifying training samples that are very similar to the current test sample based on their latent representations on the manifold. Figure 3.6 shows contours of the error magnitude at $\boldsymbol{\mu}^* = [1.601, 1.832, 0.3643, 543.8]$ for both the global and Isomap local ROM using a local ROM dimension of $k = 120$. The relative errors in both $u$ and $v$ decrease significantly when using the Isomap local ROM, with percent decreases in $\epsilon$ of 40.6% and 47.6% respectively compared to the global ROM.

## 3.4 Adaptive Sampling

In the previous section, a method for selecting local POD bases using Isomap was introduced and showed modest gains in predictive performance while using significantly less data when applied to a highly nonlinear lid-driven cavity case. While the

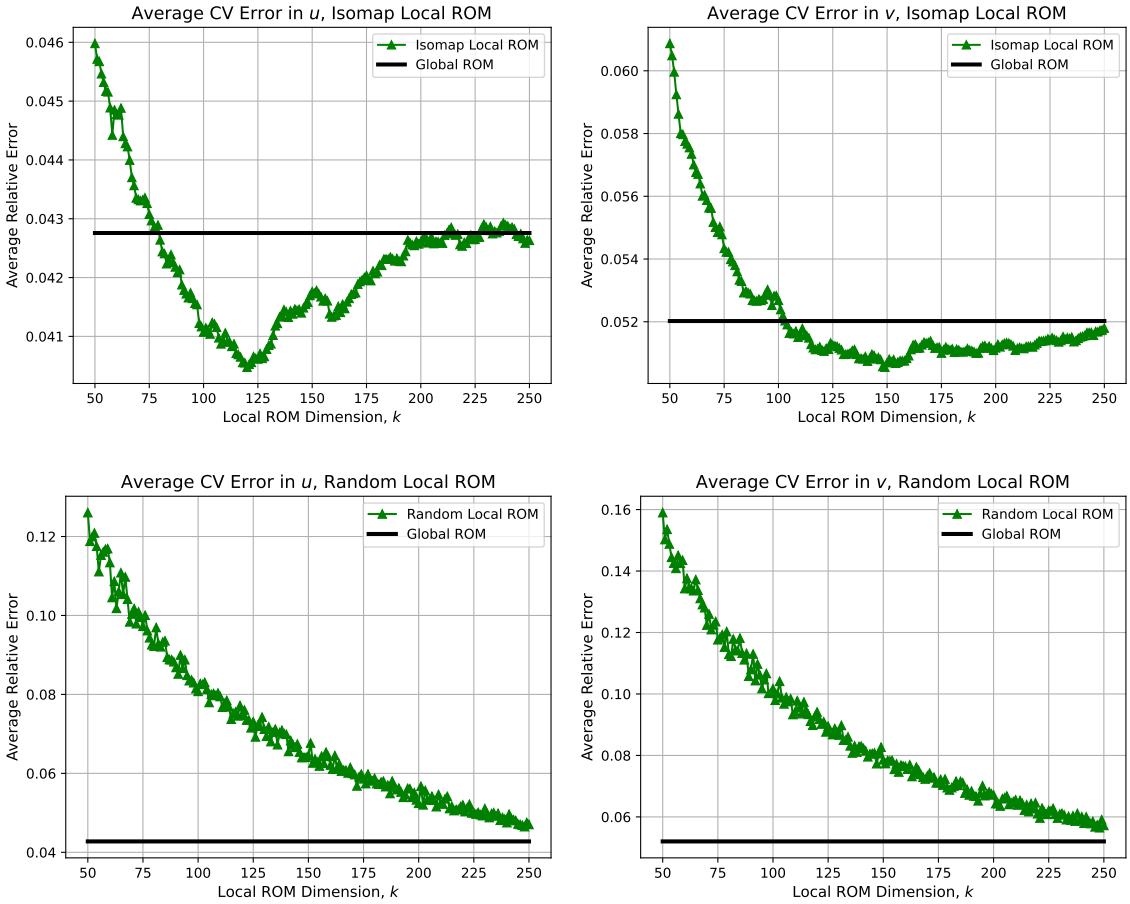
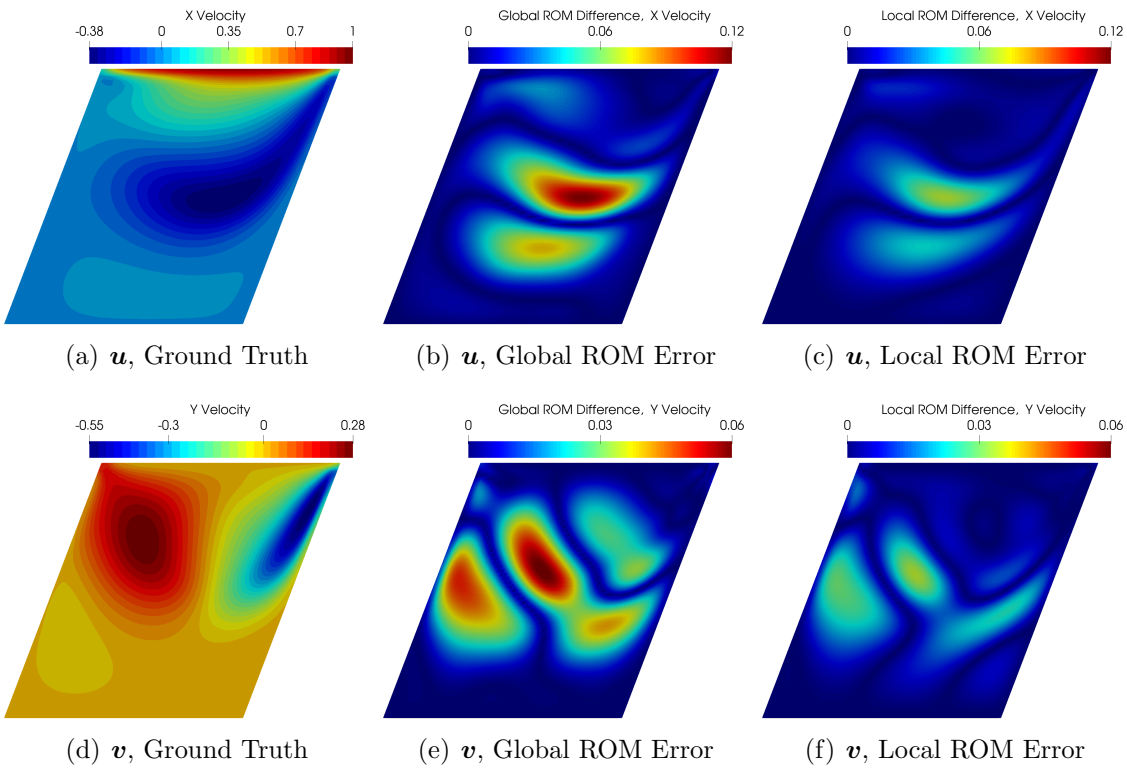



Figure 3.5: Relative cross-validation error plots for the Isomap (top) and random (bottom) local ROMs.

model does offer improved performance, the gains are slim and it is difficult to know how to choose an optimal local ROM dimension a priori. However, the results do highlight the efficacy of Isomap in separating full-order snapshots by physical regime.

For problems that do not exhibit highly nonlinear behavior throughout the parameter space, a lower amount of training data is typically needed for accurate results. However, using a small amount of data makes the ROM more sensitive to the quality of the training snapshots. Given a fixed computational budget for generating full-order snapshots, the goal is to maximize the quality of the training data. Methods that only take the design parameters into account when generating samples, like LHS, can lead to important areas of the solution space being omitted. The POD basis can




(a) $\boldsymbol{u}$, Ground Truth (b) $\boldsymbol{u}$, Global ROM Error (c) $\boldsymbol{u}$, Local ROM Error

(d) $\boldsymbol{v}$, Ground Truth (e) $\boldsymbol{v}$, Global ROM Error (f) $\boldsymbol{v}$, Local ROM Error

Figure 3.6: Comparison between global and Isomap local ROMs at $\boldsymbol{\mu}^* = [1.601, 1.832, 0.3643, 543.8]$ for $k = 120$.

lack certain physical features as a result, leading to poor predictive performance at some unseen points. In this section, I propose an adaptive sampling algorithm using Isomap and GPR to develop physically diverse training datasets for ROMs. The proposed adaptive sampling algorithm uses LHS to generate a baseline set of design parameters at which the full-order model is solved, and Isomap is then utilized to develop a greedy algorithm that iteratively selects the next design parameter from a set of candidate points to add to $\boldsymbol{\mathcal{U}}_{\text{train}}$ by filling the current low-dimensional manifold of the training data until a desired number of samples is obtained.

Previous work in developing adaptive sampling algorithms for POD-based ROMs has mainly focused on non-intrusive methods and involves computing the SVD at each adaptation iteration. Braconnier et. al [53] use an a-posteriori error estimator based on a leave-one-out approach using the POD basis. Guenot et. al [54] propose

two algorithms that select samples based on model improvement based on rotation of the POD vectors or through changes in the POD coefficients. Wang et. al [55] combine the two methods to develop a conjunction sampling strategy. My proposed algorithm does not use POD, which is more computationally expensive than Isomap when the number of observations is relatively low. When used in the algorithm, Isomap produces a manifold of a fixed low dimension, whereas the dimensionality of the POD basis grows with the number of samples. In a work [56] by Franz et al., an Isomap-based adaptive sampling algorithm is introduced and applied to both a nonlinear Isomap-based ROM [57] and POD-based ROM. The adaptive sampling algorithm in that work requires solving an optimization problem to determine the added samples, and results are limited in scope, with analysis being conducted at a single test point. Computational costs and details of the optimization process are also not provided. The criteria used to select added points are different from the one presented here, although both are dictated by distances from points within the low-dimensional space.

The proposed adaptive sampling algorithm utilizes Isomap to find a low-dimensional physical representation of the current training data to make an informed decision when selecting the next design parameter to add to the dataset. Given the current set of design parameters in the training set $\boldsymbol{\mathcal{U}}_{\text{train}} \in \mathbb{R}^{n \times p}$ and a very large set of design parameters $\boldsymbol{\mathcal{U}}_{\text{cand}} \in \mathbb{R}^{M \times p}$ generated using LHS, the current manifold of the training data is filled to diversify the represented physical states. A number $n_i$ of initial samples generated using LHS is required, and the adaptation algorithm is iteratively repeated until a total of $n_t$ samples is obtained. First, Isomap is applied to the transpose of the current snapshot matrix $\boldsymbol{S}^T$ to obtain a 2-dimensional latent representation matrix $\boldsymbol{W}_{\text{train}}$. A set of candidate design parameters $\boldsymbol{\mathcal{U}}_{\text{cand}}$ is generated using LHS. By generating enough candidate points, the parameter space $\boldsymbol{\mathcal{D}}$ is efficiently represented. GPR is then used to approximate the latent representations

$\boldsymbol{W}_{\text{cand}}$ of the candidate design parameters, with a separate regression model used for each latent variable. The pairwise distances between all of the points in $\boldsymbol{W}_{\text{train}}$ and $\boldsymbol{W}_{\text{cand}}$ are then calculated and the minimum pairwise distance for each candidate point is retained in a vector $\boldsymbol{d}_{\text{min}} \in \mathbb{R}^M$. The candidate point corresponding to the maximum value in $\boldsymbol{d}_{\text{min}}$ is chosen as the new design parameter $\boldsymbol{\mu}_{\text{new}}$ to be added to $\boldsymbol{\mathcal{U}}_{\text{train}}$. An example of this is shown in Figure 3.7. By looking at the minimum pairwise distances for each candidate point, points that are already similar to any of the current samples (i.e., the red marker in the lower left corner) are filtered out. To fill the manifold efficiently, the point with the greatest minimum distance from all current points is chosen, which is the point in the center. The FOM is then solved at $\boldsymbol{\mu}_{\text{new}}$ and the solution $\boldsymbol{x}_{\text{new}}$ is added to $\boldsymbol{S}$. The time complexity of Isomap grows mainly with $n$, which is small, instead of $N$. The overall time complexity of Isomap, including nearest neighbor graph construction, geodesic distance calculation, and MDS is given as $\mathcal{O}\left(Nn\log(n) + n^2\log(N) + n^2\right)$ when fixing the latent variable dimension and $\mathcal{K}$. The computational cost comes from the implementation of Isomap in scikit-learn. Using the same library, the time complexity of the truncated SVD is $\mathcal{O}(Nn^2)$. The adaptation algorithm is fully outlined in Algorithm 5.

### 3.4.1 Results

The two test cases used to demonstrate the performance of the adaptive sampling algorithm are a 2D NACA 0012 airfoil and a 3D Cessna 172 wing. The design parameters for both cases involve geometric deformations using a free-form deformation (FFD) method through the pyGeo [58] package. While both the non-intrusive and projection-based ROM are used for the first test case, only the non-intrusive ROM is used for the Cessna 172 case. Although the p-ROM has previously been applied to this test case [19], a very small geometric parameter range was used where there was very little variation in physics and the adaptive sampling algorithm offers no

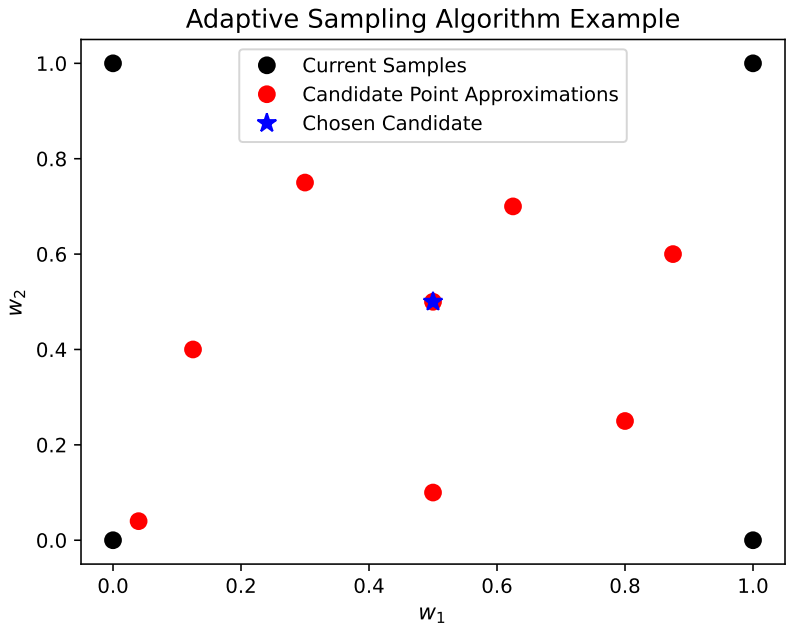



Figure 3.7: Example of the selection criteria for the adaptive sampling algorithm.

**Algorithm 5** Adaptive sampling algorithm using Isomap

**Input:** $\boldsymbol{S}, \boldsymbol{\mathcal{U}}_{\text{train}}$ ▷ Initial snapshot matrix and design parameters
**Output:** $\boldsymbol{S}, \boldsymbol{\mathcal{U}}_{\text{train}}$ ▷ Final snapshot matrix and design parameters
**for** $j \in \{n_i + 1, n_i + 2, \ldots, n_t\}$ **do** ▷ Iterate over all additional samples
  $\boldsymbol{W}_{\text{train}} \leftarrow \text{Isomap}(\boldsymbol{S}^T)$ ▷ Compute latent representations of training data
  $\boldsymbol{\mathcal{U}}_{\text{cand}} \leftarrow \text{LHS}(\boldsymbol{\mathcal{D}})$ ▷ Generate large set of candidate design parameters using LHS
  $\boldsymbol{W}_{\text{cand}} \leftarrow \text{GPR}(\boldsymbol{W}_{\text{train}}, \boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{\mathcal{U}}_{\text{cand}})$ ▷ Approximate latent representations of candidates
  $\boldsymbol{D} \leftarrow \text{dist}(\boldsymbol{W}_{\text{train}}, \boldsymbol{W}_{\text{cand}})$ ▷ Compute pairwise Euclidean distances
  $\boldsymbol{d}_{\text{min}} \leftarrow \text{sort}(\boldsymbol{D})[1, :]$ ▷ Return minimum pairwise distances for each candidate
  $\boldsymbol{\mu}_{\text{new}} \leftarrow \boldsymbol{\mathcal{U}}_{\text{cand}}[\text{argmax}(\boldsymbol{d}_{\text{min}})]$ ▷ New $\boldsymbol{\mu}$ maximizes the minimum pairwise distances
  $\boldsymbol{x}_{\text{new}} \leftarrow \text{FOM}(\boldsymbol{\mu}_{\text{new}})$ ▷ Solve full-order model at $\boldsymbol{\mu}_{\text{new}}$
  $\boldsymbol{S}[:, j] \leftarrow \boldsymbol{x}_{\text{new}}$ ▷ Add new solution to snapshot matrix
  $\boldsymbol{\mathcal{U}}_{\text{train}}[j, :] \leftarrow \boldsymbol{\mu}_{\text{new}}$ ▷ Add new design parameters to training set
**end for**

advantage over LHS. When using a larger parameter range, the p-ROM lacks stability and sometimes fails to converge, by either failing to converge to the prescribed ROM tolerance and getting stuck at a bad local minima, or having $r_{\text{tol}}$ diverge. I attempted to remedy this issue by using the ni-ROM solution as an initial condition,

using fewer basis vectors, and slightly increasing the p-ROM tolerance, but the issue persisted. This issue is common in projection-based ROMs and well-documented [12], highlighting their limitations. The number of basis vectors $k$ used for the ni-ROM is equal to the number of samples $n$. The GPR kernel used in the adaptation algorithm for predicting the latent representations is the rational quadratic kernel, which can be seen as a scale mixture of RBF kernels, given as

$$\kappa(\boldsymbol{\mu}, \boldsymbol{\mu}^*) = \left(1 + \frac{d(\boldsymbol{\mu}, \boldsymbol{\mu}^*)}{2\alpha l^2}\right)^{-\alpha}, \tag{3.6}$$

where $\alpha$ is a scale mixture parameter. The Matern kernel is chosen to predict the POD expansion coefficients. These kernels were chosen through a trial-and-error process, where the RBF kernel was also considered. The Matern and rational quadratic kernels are both accurate for predicting latent variables and POD coefficients, and the difference in performance between the two is modest. The relative information content threshold $\gamma$ from Equation 2.8 is set to 0.999 to select the number of basis vectors when using the p-ROM. I found that this strikes a good balance between the quality of the POD basis and the numerical stability of the p-ROM. The tolerance of the p-ROM is set to $r_{\text{tol}} = 100$ and the maximum number of iterations is set to $n_{\max} = 20$. The initial norm of the reduced residuals at validation points is typically on the order of $10^6$ or $10^7$, and setting $r_{\text{tol}} = 100$ requires a drop of 4-5 magnitudes. For both test cases, the number of nearest neighbors $\mathcal{K}$ used for Isomap is set to 4. Since the manifolds are computed using a relatively small number of observations, $\mathcal{K}$ must be low to provide meaningful geodesic distances as well as high enough to acknowledge global data structure, which would be ignored with extremely low or high values.

The adaptive sampling algorithm is performed using the full-order state $\boldsymbol{x} = [u, v, p]$ while both ROMs use $\boldsymbol{x} = [u, v, w, p, \nu_t, \phi]$, where $\phi$ is the cell-face flux. A subset of the flow variables is chosen for the adaptive sampling algorithm as they

are the ones that are more representative of the flow physics, while the ROMs require access to all flow variables. For both test problems, $M = 10^4$ candidate points are generated using LHS for each iteration of the adaptation algorithm and a set of $N_V = 100$ validation points generated using LHS is used to assess performance. Four different sample levels $n_t = [20, 24, 28, 32]$ are used for both test problems. Additionally, a number of trials $N_t$ at each sample level is simulated to average performance metrics over to account for the randomness associated with generating the baseline samples using LHS. The average relative $L^2$ error of a field quantity for a single trial $\epsilon_t$ over all of the validation points indexed by $i$ is given as

$$\epsilon_t = \frac{1}{N_V} \sum_{i=1}^{N_V} \frac{\|\boldsymbol{x}^i - \tilde{\boldsymbol{x}}^i\|^2}{\|\boldsymbol{x}^i\|^2}, \tag{3.7}$$

where $\tilde{\boldsymbol{x}}$ is the field approximated by the ROM. The errors are then averaged over each trial to give a single error value $\bar{\epsilon}$.

$$\bar{\epsilon} = \frac{1}{N_t} \sum_{t=1}^{N_t} \epsilon_t. \tag{3.8}$$

Relative errors in the coefficients of lift and drag $C_L$ and $C_D$ between the CFD solver and ROMs are also evaluated using the same averaging procedure. Results are presented using an initial number of LHS-generated samples $n_i = \frac{n_t}{2}$. This value is chosen as a sufficient number of initial samples is required to obtain a meaningful low-dimensional manifold from Isomap. Additionally, the final data should be composed of an adequate number of adaptively chosen samples.

#### 3.4.1.1 NACA 0012 Airfoil

A NACA 0012 airfoil, shown in Figure 3.8, is used as the first test case. The chord measures 1 m and the span is 0.1 m, with one cell in the spanwise direction. The flow is incompressible with $Re = 10^6$ with $U_\infty = 10$ m/s. A coarse mesh with

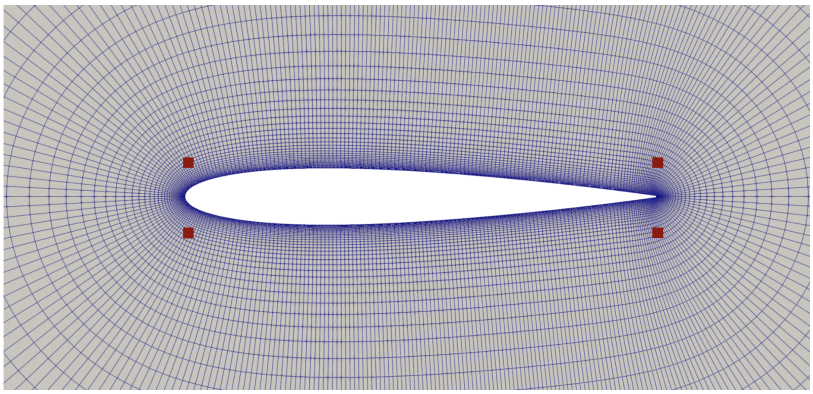


Figure 3.8: Mesh and FFD points for the NACA 0012 case.

23,182 cells is generated and the computational domain extends 30 chords from the airfoil surface. Four FFD points control the vertical displacements at the leading and trailing edges, with the bounds of the design parameters given by $\boldsymbol{\mu} \in [-0.03, 0.03]$ m, and the baseline angle of attack $\alpha$ is set to 4 degrees. The full-order model is run for 2000 iterations, after which the flow residuals fall to at least $10^{-6}$, and a total of $N_t = 20$ trials is simulated at each sample level. Table 3.1 lists computational costs associated with the ROM and adaptive sampling algorithm. It is shown that the computational cost of Isomap is signficantly lower than that of the SVD when using $n = 32$ samples and negligible compared to a single CFD simulation. This results in a very computationally efficient adaptive sampling algorithm, especially when comparing the cost to a single CFD simulation. While the wall time for the p-ROM varies with the number of iterations and residual evaluations, an average wall time of approximately 3.4 seconds was found, which leads to an average speed-up of around 11x, comparable to what was found in the original work [19] for a similar test case.

Field contours of $u, v$ and $p$ from the CFD solver, ni-ROM, and p-ROM are shown in Figure 3.9 at a validation point $\boldsymbol{\mu}^* = [0.017, 0.025, -0.0297, -0.0273]$ with $n_t = 32$ samples generated using LHS. Both ROMs are in close agreement with the FOM, although the p-ROM is more accurate, which is evident when looking at contours of $u$

| **Algorithm** | **Wall Time (s)** |
|---|---|
| CFD (average, 8 CPUs) | 38.32 |
| p-ROM (average, 8 CPUs) | 3.4 |
| SVD ($n = 32$) | 0.269 |
| Isomap ($n = 32$) | $2.73 \times 10^{-5}$ |

Table 3.1: Computational costs associated with the NACA 0012 case.

and $p$. Figure 3.10 shows the average relative field errors over the chosen sample levels for the ni-ROM (left) and p-ROM (right), while Figure 3.11 shows the average relative errors in $C_D$ and $C_L$. When comparing similar field quantities, it is shown that the p-ROM offers much better predictive performance, even when using a lower number of training samples. Using the adaptive sampling algorithm results in significantly lower errors for all field quantities when using the p-ROM, and for $v$ and $p$ when using the ni-ROM. This effect is greater when using the p-ROM; as it is a physics-based model, we can expect that a greater diversity of training samples will lead to a better gain in predictive performance compared to the ni-ROM. Errors in drag and lift exhibit a similar pattern, where both the predictive performance and reduction achieved by using the adaptive sampling algorithm are greater for the p-ROM when compared to the ni-ROM. There is also a greater relative decrease in force coefficient errors compared to field errors. Figure 3.12 shows the average number of basis vectors used for the p-ROM for both adaptation and LHS alone. For the same singular value threshold, using the adaptation algorithm results in a higher number of basis vectors at all of the sample levels, which shows that the training samples are more physically diverse. Building an ni-ROM with 28 total samples offers better predictive performance in $v, p, C_D$, and $C_L$ compared to using LHS alone with 32 samples. The same is true for building a p-ROM with 24 samples, where the relative gain in predictive performance is larger, again showing that the p-ROM benefits more from using adaptively chosen samples. For a smaller computational budget, using the adaptive sampling algorithm leads to better overall predictive performance of both

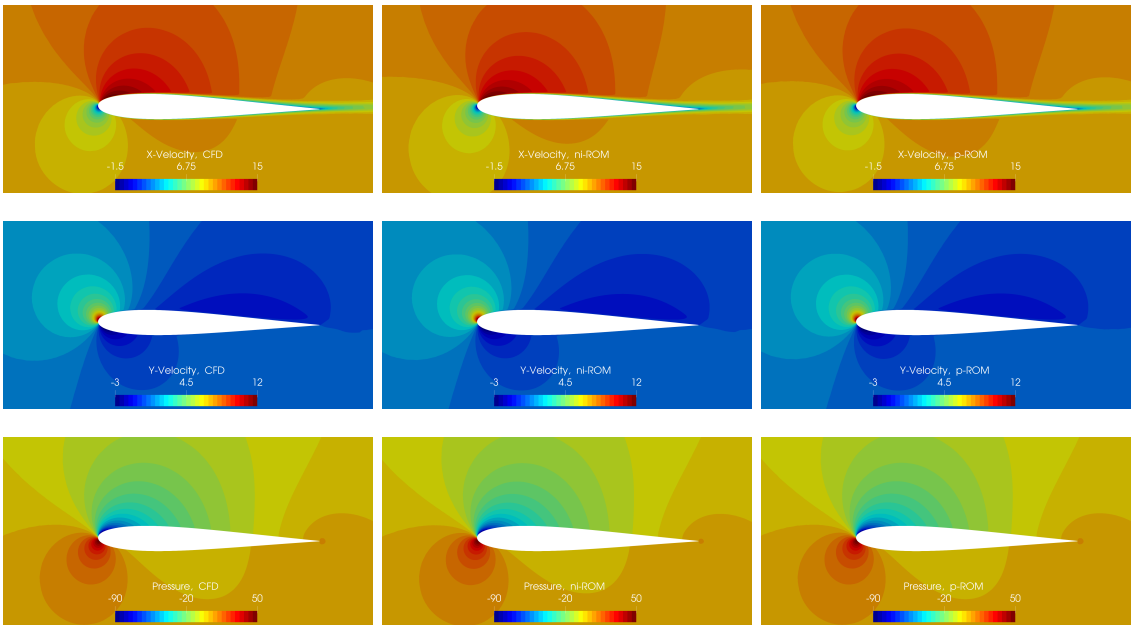



Figure 3.9: Comparison of NACA 0012 field contours with $n_t = 32$ at $\boldsymbol{\mu}^*$ =[0.017, 0.025, -0.0297, -0.0273].

field and integral quantities for both purely data-driven and physics-based ROMs.

Figures 3.13 and 3.14 show field error contours for a single trial with errors close to the reported average relative errors. The errors are averaged over all over the validation points for the ni-ROM and p-ROM respectively. The contours represent the absolute difference between the CFD solver and ROM. For all field quantities, errors are greater closer to the surface of the airfoil and negligible farther away. It should be noted that the relative $L^2$ errors reported consider the error over the entire computational domain; since the majority of the error is close to the airfoil surface, this can explain the larger relative decrease in force coefficient errors compared to field errors.

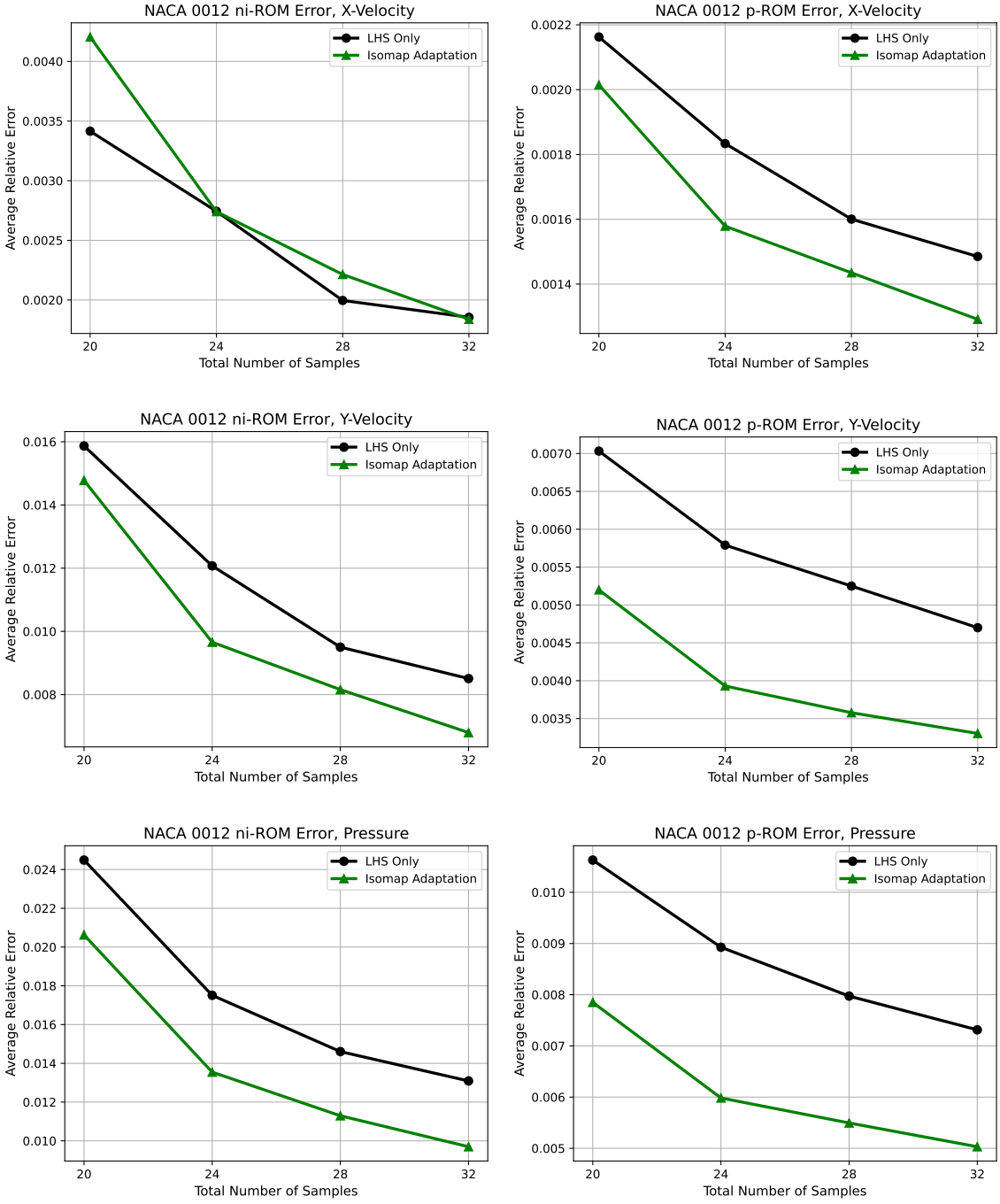



Figure 3.10: Comparison of average $L^2$ relative errors for field quantities for the ni-ROM (left) and p-ROM (right).

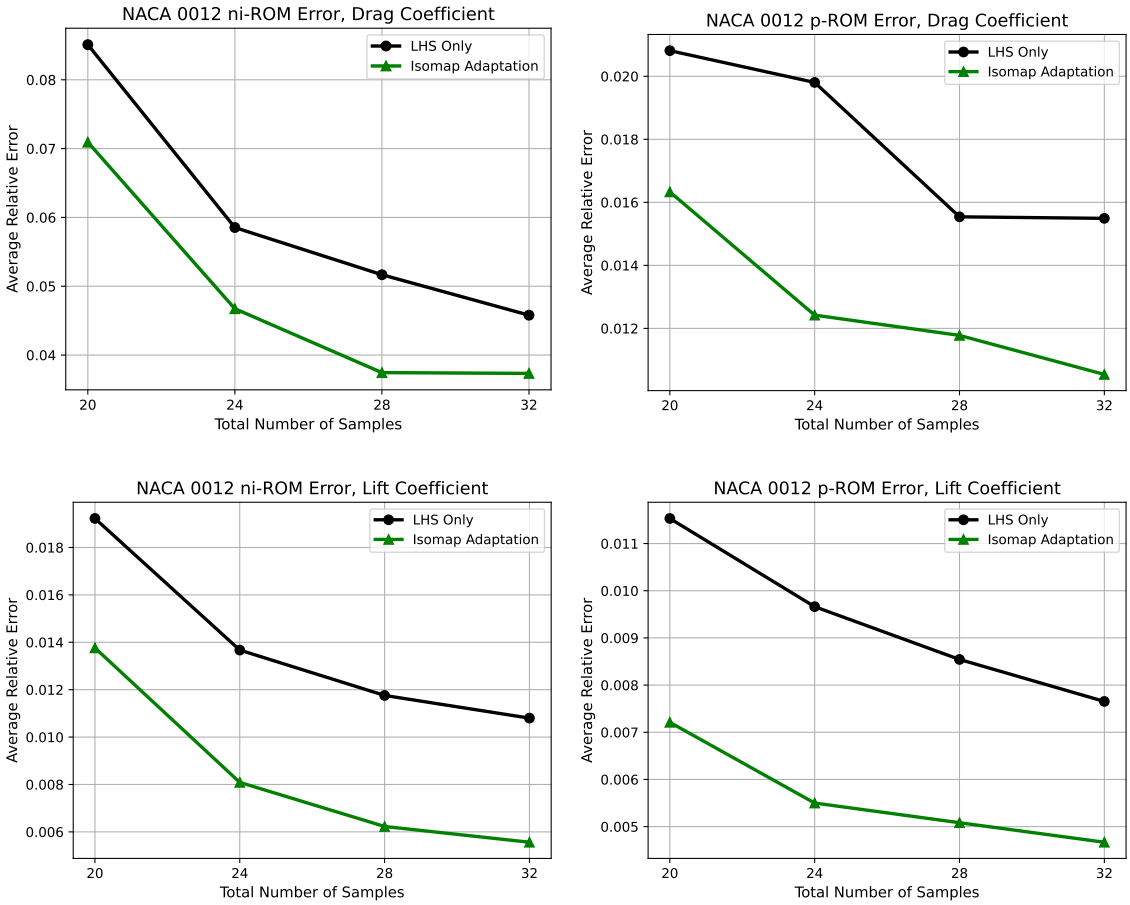



Figure 3.11: Comparison of average $L^2$ relative errors in $C_D$ and $C_L$ for the ni-ROM (left) and p-ROM (right).

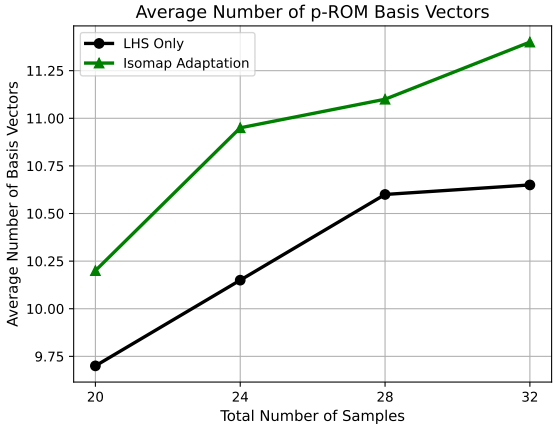



Figure 3.12: Average number of basis vectors used for the p-ROM.

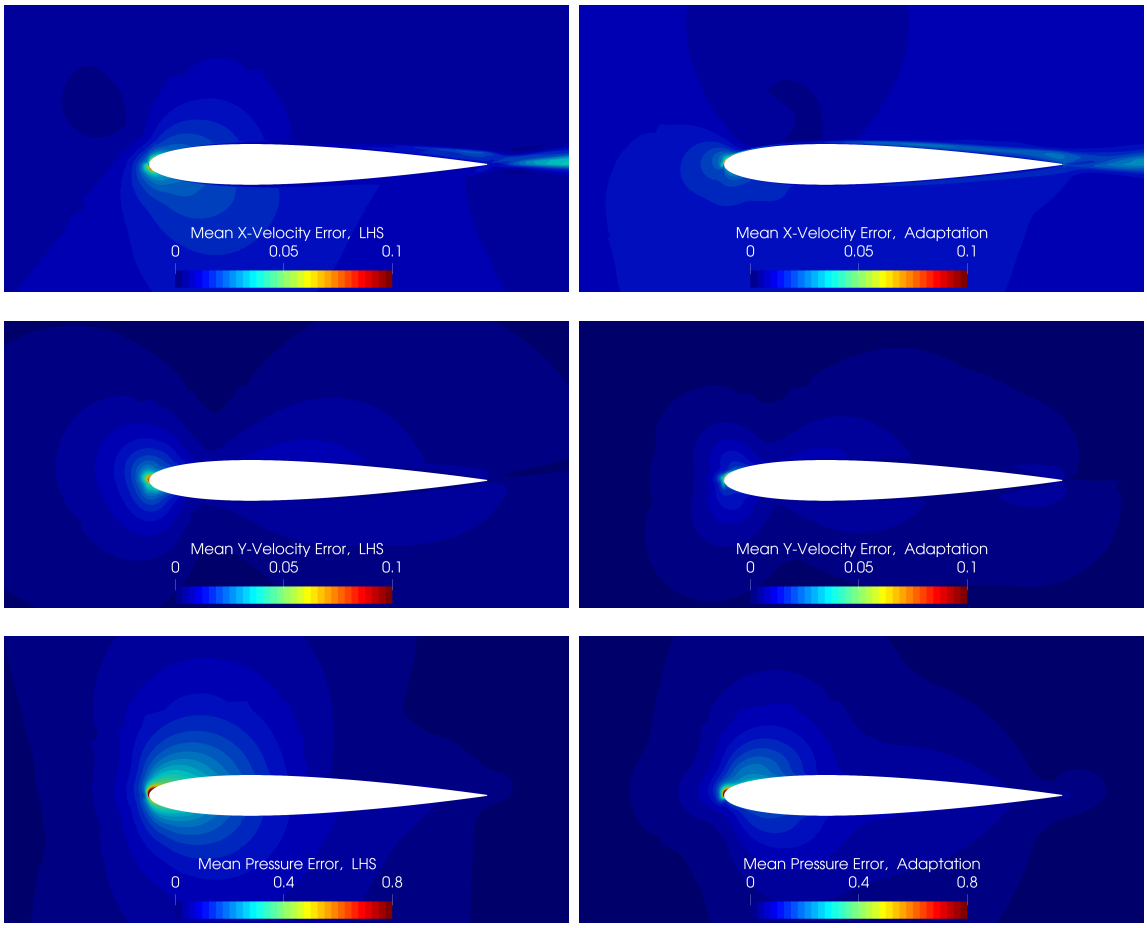



Figure 3.13: Mean ni-ROM field error contours for a single NACA 0012 trial using LHS (left) and adaptation (right) with $n_t = 32$.

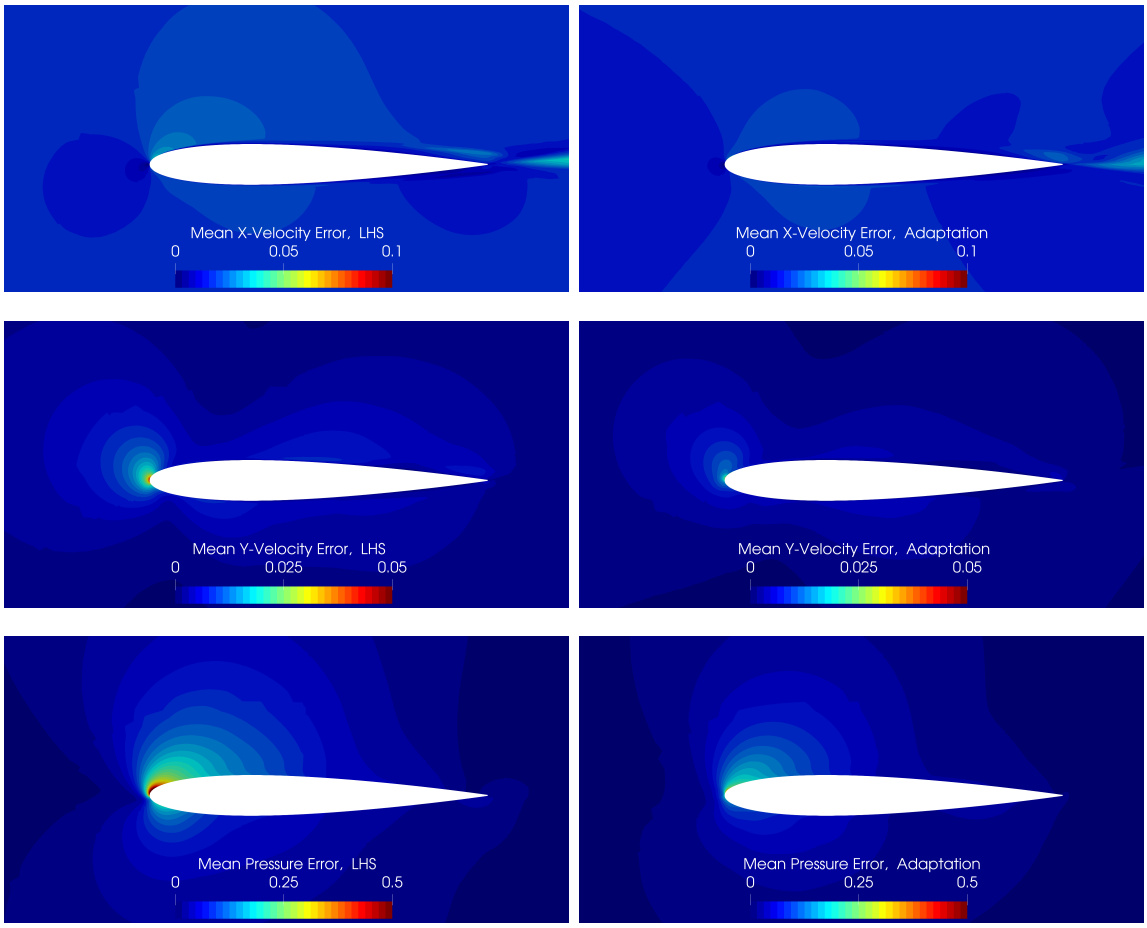



Figure 3.14: Mean p-ROM field error contours for a single NACA 0012 trial using LHS (left) and adaptation (right) with $n_t = 32$.

**Comparison to a POD-based Algorithm** The performance of the adaptive sampling algorithm is compared to a POD-based adaptive sampling algorithm found in a work [54] by Guenot et al. The original work uses a mean-centered SVD [28] to compute the POD basis, while I do not use that approach. The POD-based algorithm uses a leave-one-out (LOO) cross validation approach to determine the influence of each current sample on the POD basis. The LOO snapshot matrix $\boldsymbol{S}_{-j}$ with the $j^{\text{th}}$ snapshot left out is given as

$$\boldsymbol{S}_{-j} \in \mathbb{R}^{N \times n} = [\boldsymbol{x}^1, \cdots, \boldsymbol{x}^{j-1}, \boldsymbol{0}, \boldsymbol{x}^{j+1}, \cdots, \boldsymbol{x}^n], \tag{3.9}$$

where $\boldsymbol{0}$ is the zero vector. The LOO POD basis $\boldsymbol{\Psi}_{-j}$ comes from the SVD of $\boldsymbol{S}_{-j}$. Given the singular values $\sigma_j$ of the complete POD basis, the influence of each training sample $\boldsymbol{\mu}^i$ on the complete POD basis is given as

$$\text{infl}(\boldsymbol{\mu}^j) = \sum_{i=1}^{n} \sigma_j \left( \frac{1}{|\boldsymbol{\psi}^i \cdot \boldsymbol{\psi}^i_{-j}|} - 1 \right). \tag{3.10}$$

The dot product gives the cosine of the angle between each vector in the complete and LOO POD bases. The relative influence of a sample on the POD basis is given as

$$\text{infl}_{\text{rel}}(\boldsymbol{\mu}^j) = \frac{\text{infl}(\boldsymbol{\mu}^j)}{\sum_{i=1}^{n} \text{infl}(\boldsymbol{\mu}^i)}, \tag{3.11}$$

which has the effect of using the singular values to to assign weights to the POD basis vectors. The distances between all candidate points in $\boldsymbol{\mathcal{U}}_{\text{cand}}$ and their nearest neighbors in $\boldsymbol{\mathcal{U}}_{\text{train}}$ are calculated and stored in a vector $\boldsymbol{d}_{\text{min}}$, just as in Algorithm 5. For each candidate point, the influence of its nearest neighbor $\boldsymbol{\mu}^d$ is multiplied by the corresponding entry in $\boldsymbol{d}_{\text{min}}$ to obtain a POD basis improvement potential,

$$\text{pot}(\boldsymbol{\mu}_{\text{cand}}) = d(\boldsymbol{\mu}_{\text{cand}}, \boldsymbol{\mu}^d)\text{infl}_{\text{rel}}(\boldsymbol{\mu}^d). \tag{3.12}$$

The candidate point with the highest basis improvement potential is chosen as the next sample. To avoid the cost of computing $n$ truncated SVDs of $N \times n$ matrices at each adaptation iteration as required for the dot product in Equation 3.10, the authors of the work compute it as follows,

$$|\boldsymbol{\psi}^i \cdot \boldsymbol{\psi}^i_{-j}| = |\boldsymbol{U}_n[j,j]|, \tag{3.13}$$

where $\boldsymbol{U}_n$ is computed from the SVD of $\Sigma(I - V[j,:]^T V[j,:])$, which is an $n \times n$ matrix.

Table 3.2 shows the average relative errors between the two adaptive sampling algorithms and LHS with $n_t = 32$ and $n_i = \frac{n_t}{2}$. The POD-based algorithm offers better performance in predicting $u$, $v$, and $C_D$, while the proposed algorithm offers better performance in predicting $p$ and $C_L$. $u$ in particular is much better predicted using the POD-based algorithm. A reason for this may be that the variation in $u$ between different samples does not contribute as much to the geodesic distances between points as $v$ and $p$ do, and additional samples that vary these quantities more are selected. Table 3.3 shows that the proposed algorithm outperforms the POD-based one in predicting all quantities except $u$, which further suggests that the algorithm is better suited for projection-based ROMs. The computational costs of both algorithms as a function of the number of current samples is given in Figure 3.15. The given computed costs include all parts of the algorithms excluding the generation of candidate points using LHS. The algorithm does not display any significant trend with the number of samples, and variations can be due to the GPR algorithm or CPU background processes. The POD-based algorithm exhibits an upward trend with the number of samples, most likely due to an increased cost with computing the POD basis as more samples are added. While the difference in computational cost between the two algorithms for this case is low, we can expect that for larger problems such as the Cessna 172 test case introduced next, that the Isomap-based algorithm will be faster given the costs listed in Table 3.4, since costs apart from Isomap and POD will

not vary with $N$.

| Quantity | Isomap | POD | LHS |
|---|---|---|---|
| $u$ | 0.00184 | **0.00134** | 0.00186 |
| $v$ | 0.00680 | **0.00630** | 0.00851 |
| $p$ | **0.00970** | 0.00988 | 0.01309 |
| $C_D$ | 3.7333 | **3.5206** | 4.5794 |
| $C_L$ | **0.55636** | 0.71766 | 1.08001 |

Table 3.2: Adaptive sampling algorithms and LHS average relative errors for the NACA 0012 test case (ni-ROM, $n_t = 32$).

| Quantity | Isomap | POD | LHS |
|---|---|---|---|
| $u$ | 0.00129 | **0.00117** | 0.00147 |
| $v$ | **0.00330** | 0.00346 | 0.00463 |
| $p$ | **0.00503** | 0.00552 | 0.00722 |
| $C_D$ | **1.0539** | 1.1616 | 1.5492 |
| $C_L$ | **0.46691** | 0.55562 | 0.7555 |

Table 3.3: Adaptive sampling algorithms and LHS average relative errors for the NACA 0012 test case (p-ROM, $n_t = 32$).

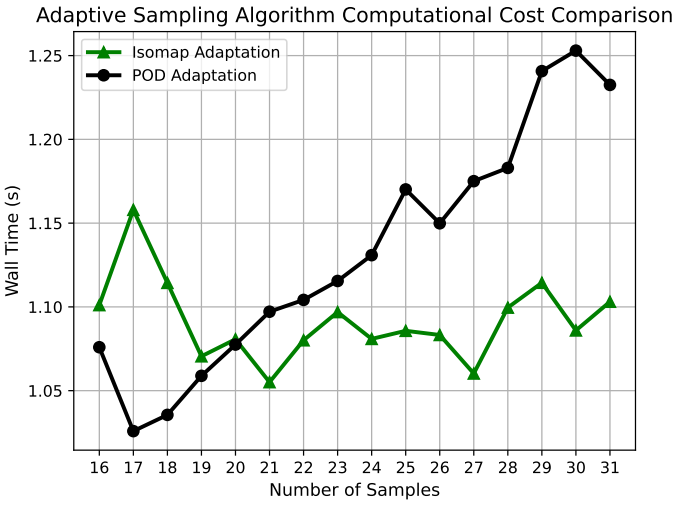



Figure 3.15: Computational costs associated with the Isomap and POD-based adaptive sampling algorithms for the NACA 0012 case.

**Effect of Initial Sample Size** The effect of the fraction of initial LHS-generated samples is investigated with $n_t = 32$ for both the ni-ROM and p-ROM. In partciular, the relative errors in the fields and force coefficients are compared to using LHS alone with $n_i = \left(\frac{n_t}{4}, \frac{3n_t}{8}, \frac{n_t}{2}, \frac{5n_t}{8}, \frac{3n_t}{4}\right)$ with $N_t = 20$ trials. Figures 3.16 and 3.17 show that there is there is no distinguishable trend between the error metrics and number of initial samples for the ni-ROM. Apart from the average relative error in $u$, which was already previously shown to be close to that of using LHS alone for the ni-ROM, the predictive performance in all other quantities is significantly better than using LHS alone. The results suggest that the effect of including adaptively chosen samples has diminishing returns as more samples are added. All of the error metrics are slightly higher for the p-ROM with $n_i = \frac{3n_t}{4}$. The p-ROM is more sensitive to the diversity of the training data, and using fewer adaptively chosen samples can lead to a relatively larger degradation in performance compared to the ni-ROM. When using the adaptation algorithm, an important consideration is the accuracy of the GPR model in predicting the latent variables at candidate points. Using too few initial samples, especially for highly non-linear problems, can lead to an inaccurate latent variable regression model and to poorly chosen samples.

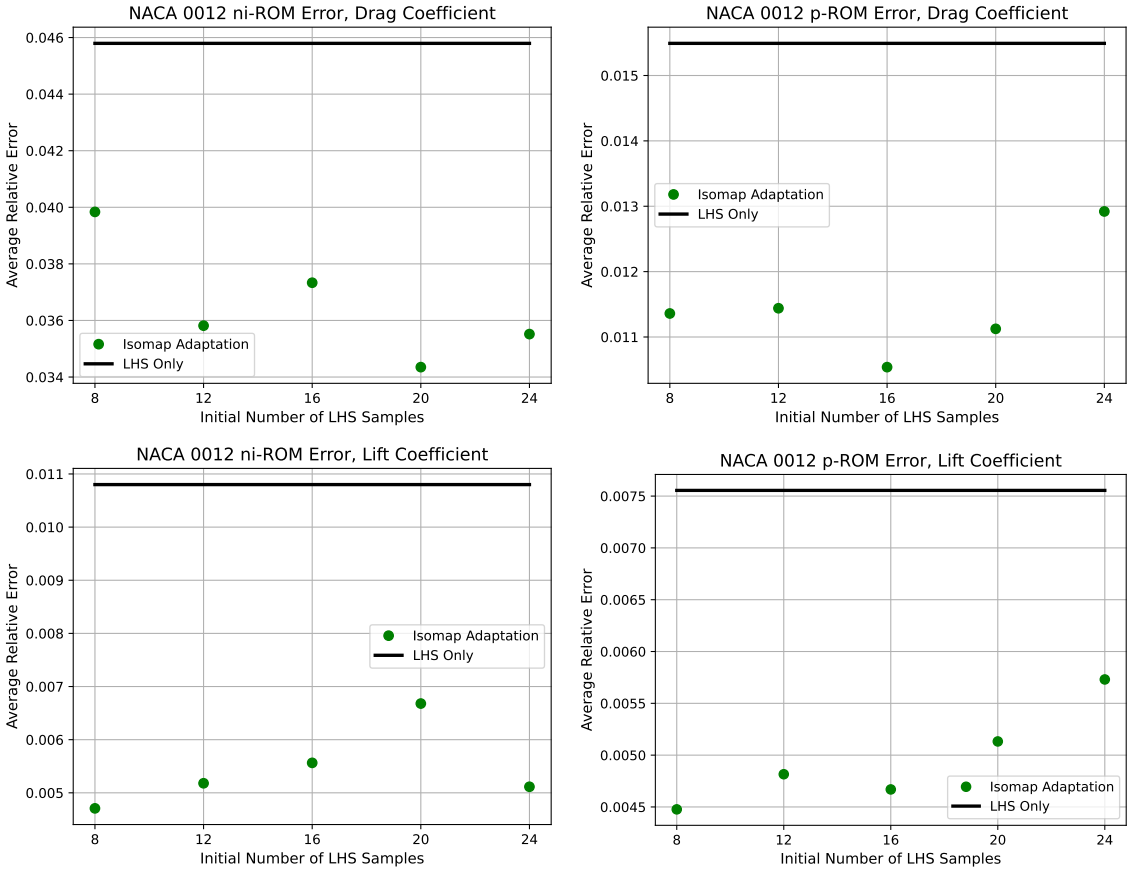



Figure 3.16: Comparison of average $L^2$ relative errors in $C_D$ and $C_L$ for the ni-ROM (left) and p-ROM (right) with $n_t = 32$.

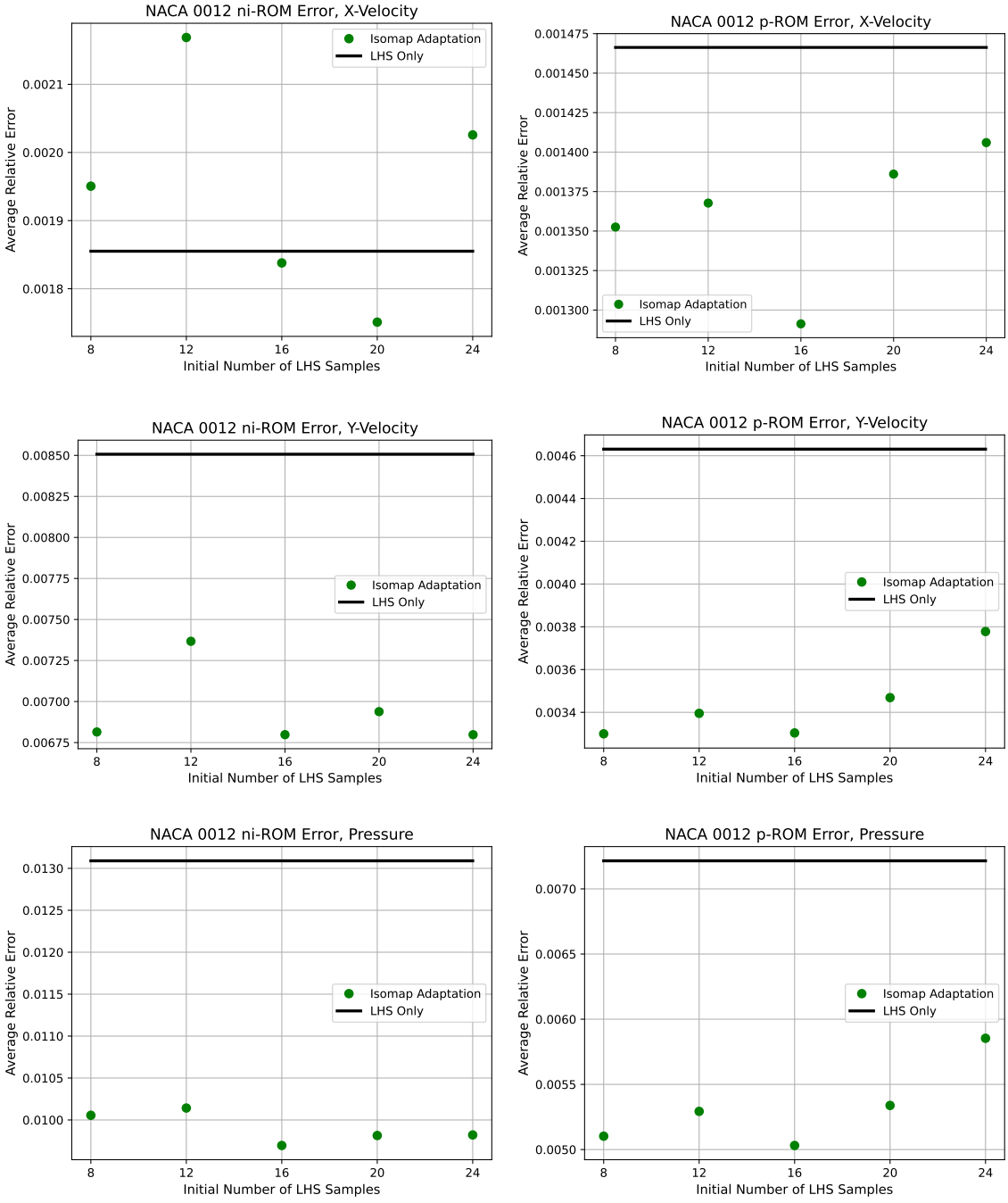



Figure 3.17: Comparison of average $L^2$ relative errors for field quantities for the ni-ROM (left) and p-ROM (right) with $n_t = 32$.

#### 3.4.1.2 Cessna 172 Wing

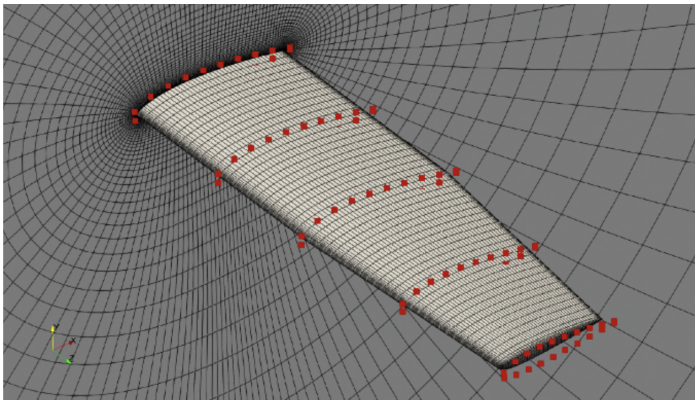


Figure 3.18: Mesh and FFD points for the Cessna 172 case.

A Cessna 172 wing, shown in Figure 3.18, is used as the next test case. The root chord of the wing measures 1.67 m, and the semi-span aspect ratio is 3.2. The Reynolds number is set to $Re = 7 \times 10^6$ with a freestream velocity of $U_\infty = 63.8$ m/s. A structured mesh with $N = 609,280$ cells is generated and the computational domain extends 30 chords from the surface. 100 FFD points are used to deform the wing surface at five spanwise locations. The design parameters are the five twist angles at these spanwise locations, which are obtained by rotating each set of FFD points. The bounds of the twist variables are $\boldsymbol{\mu} \in [-3, 3]$ degrees, and the baseline angle of attack $\alpha$ of the wing is set to 2.5 degrees. The full-order model is run for 1000 iterations, after which the flow residuals fall to at least $10^{-6}$. A total of $N_t = 5$ trials are simulated at each sample level. Computational costs for running the CFD solver as well as Isomap and the SVD with $n = 32$ are listed in Table 3.4, where it is again shown that the cost of Isomap is very low compared to the FOM and SVD, making the adaptation algorithm very efficient for this case.

Figure 3.19 shows mid-section field contours of $u$, $v$, and $p$ at a validation point $\boldsymbol{\mu}^*$ =[-2.19, -1.47, -2.79, -2.97, 2.79] from both the CFD solver and ni-ROM with $n_t = 32$ samples generated using LHS. Visually, there is very good agreement between the FOM and ROM in all three fields. The average relative errors in the fields are

| **Algorithm** | **Wall Time (s)** |
|---|---|
| CFD (16 CPUs) | 364 |
| SVD ($n = 32$) | 7.72 |
| Isomap ($n = 32$) | 0.312 |

Table 3.4: Computational costs associated with the Cessna 172 case.

shown in Figure 3.20 for the chosen sample levels. Using the Isomap adaptation algorithm leads to significantly lower errors in both $u$ and $p$ over all the sample levels and a slight error reduction in $v$. The average relative errors in $C_D$ and $C_L$ are shown in Figure 3.21, where we see that using the adaptation algorithm leads to very large reductions in the computed output errors. Building a ROM using the adaptation algorithm with 28 samples compared to 32 samples using LHS leads to comparable relative errors in $u$, $v$, and $p$ and significantly lower errors in $C_D$ and $C_L$, showing that the adaptation algorithm leads to better predictive performance given a smaller computational budget, similar to the airfoil test case. When looking at average error contours in Figure 3.22 for a single trial with relative errors close to the reported averages, it is again shown that using the adaptation algorithm leads to considerably lower errors in the computed fields closer to the surface of the wing, leading to a larger drop in error in coefficients compared to the fields, where the error metric is computed over the entire domain. Far away from the wing surface, errors in the fields are similarly low when using the adaptation algorithm and LHS alone.

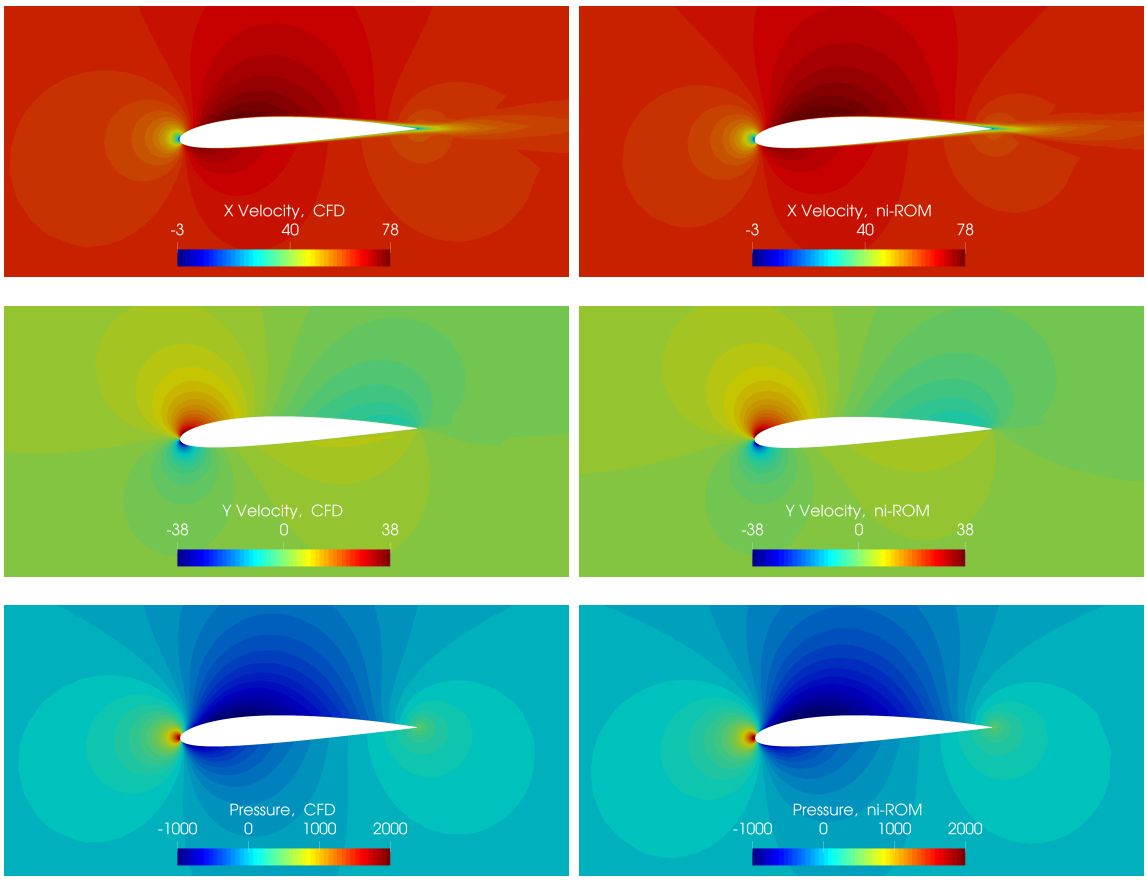



Figure 3.19: Comparison of Cessna 172 mid-section field contours with $n_t = 32$ at $\boldsymbol{\mu}^*$ =[-2.19, -1.47, -2.79, -2.97, 2.79].

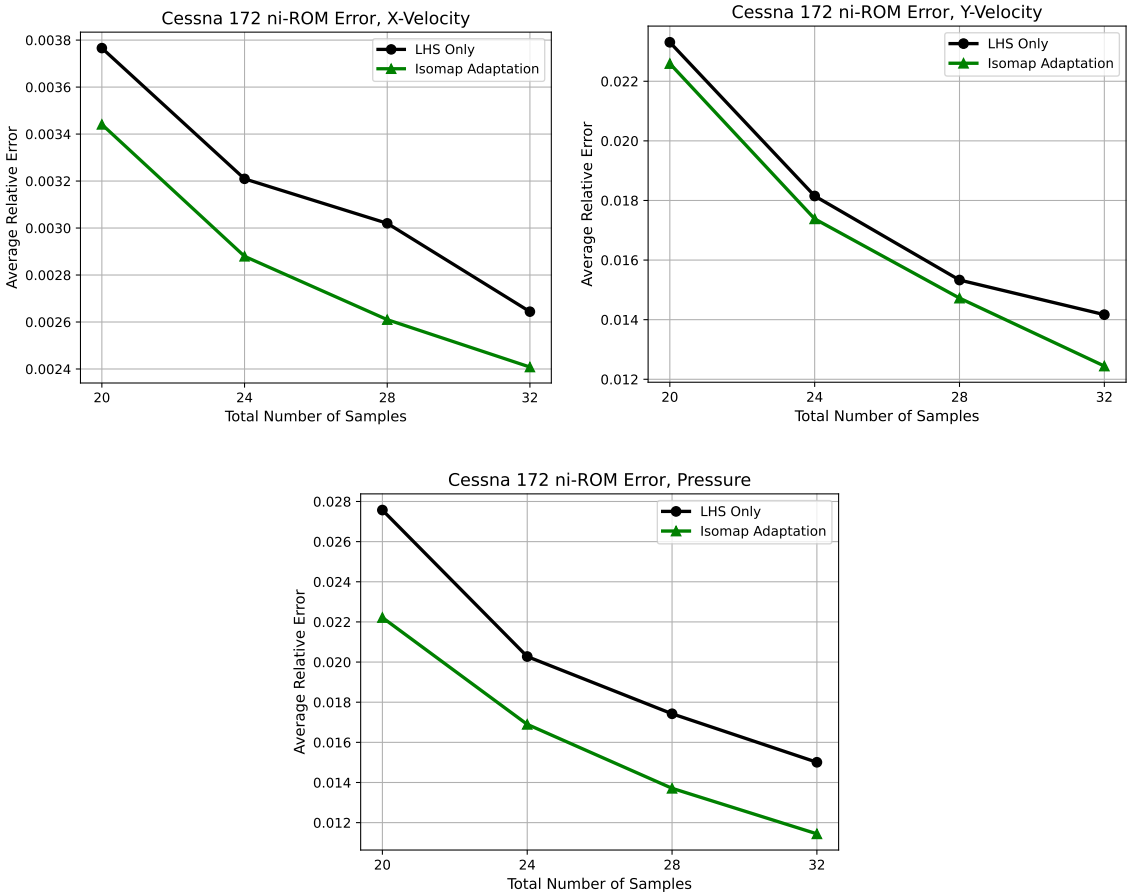



Figure 3.20: Comparison of average $L^2$ relative errors in field quantities.

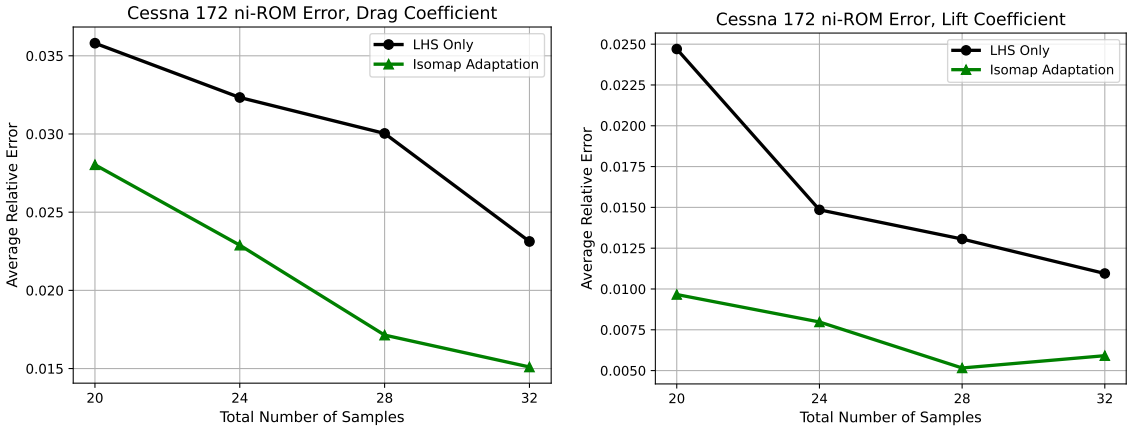



Figure 3.21: Comparison of average relative errors in $C_D$ and $C_L$.

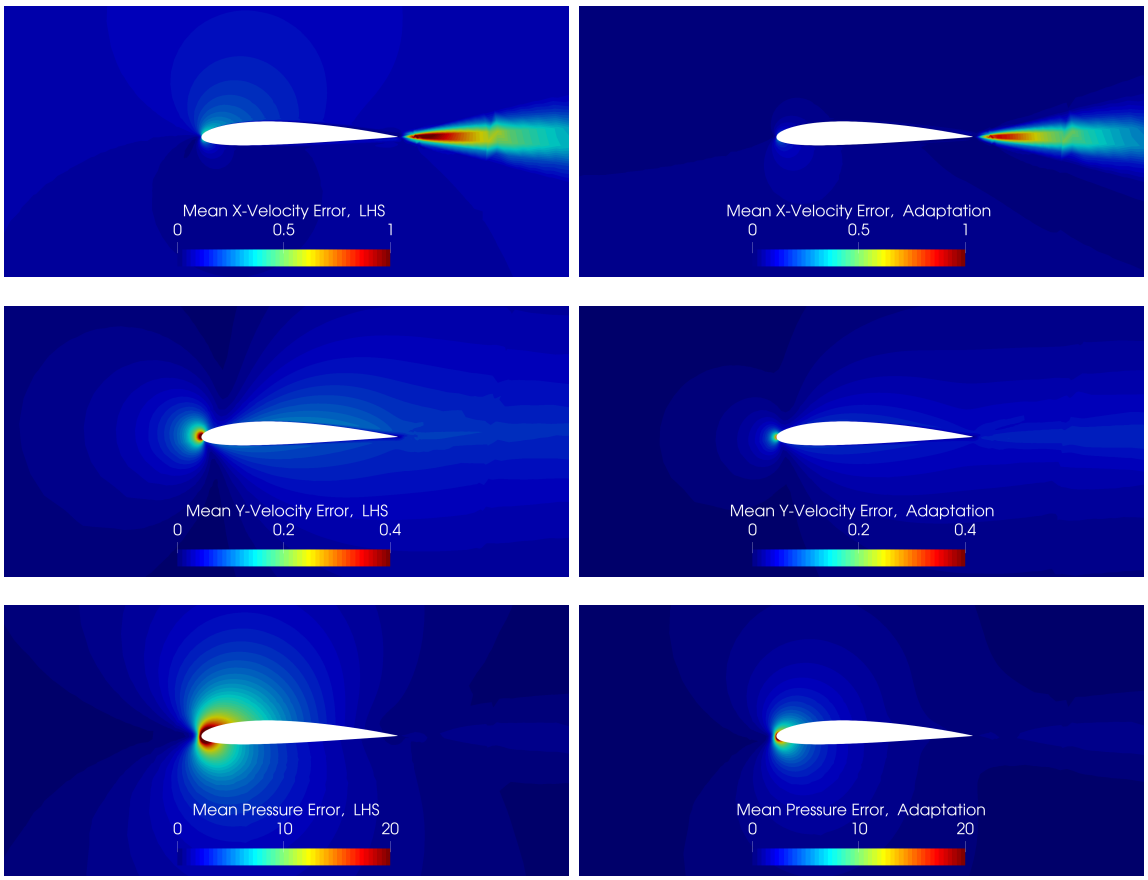



Figure 3.22: Mean mid-section field error contours for a single Cessna 172 trial using LHS (left) and adaptation (right) with $n_t = 32$.

## 3.5 Summary

In this chapter, two data selection algorithms using Isomap for ROMs are introduced. An algorithm that selects local POD bases for a POD-GPR non-intrusive ROM based on the nearest neighbors on a low-dimensional manifold is shown to offer slightly improved predictive performance while using significantly less data when applied to a highly nonlinear lid-driven cavity flow. While this data selection algorithm may not be useful in practice, it does show the efficacy of Isomap to separate data by physical regime. A computionally efficient adaptive sampling algorithm that iteratively fills the manifold of the current data set is introduced, and is shown to offer significantly improved predictive performance for both non-intrusive and projection-based ROMs when applied to two external aerodynamics problems. By filling gaps in the manifold, there are a greater diversity of physical regimes present in the training data compared to using LHS, which only accounts for the distribution of the design parameters and not the physics of the full-order model. For a given singular value threshold, using adaptively chosen samples leads to a higher number of basis vectors being used for the p-ROM, showing that the diversity of the training data increases. Given a smaller computational budget, using the adaptive sampling algorithm leads to improved predictive performance over LHS as well. Comparisons to a POD-based algorithm from another work show that the algorithm performs better when using the p-ROM, and slightly worse when using the ni-ROM.

# CHAPTER IV

# Deep Learning Based ROMs

The previous chapter introduced algorithms for selecting data in POD-based ROMs to maximize predictive performance. The performance of POD-based non-intrusive and projection-based ROMs was found to be adequate when applied to two external aerodynamics problems. However, it was also shown that for a highly nonlinear lid-driven cavity problem, the predictive performance of the non-intrusive ROM was relatively poor, in spite of using significantly more training data. As a linear method, POD has performance limits for highly nonlinear problems. POD can also be unsuitable for ROMs that are required to use a large amount of training data, such as unsteady ROMs. Since unsteady ROMs are used to predict the temporal evolution of the full-order state, multiple snapshots are required per training point and the number of basis vectors required for a given relative information content also increases. As a result, projection-based ROMs can suffer from numerical instabilities. For non-intrusive ROMs, individual errors in predicting expansion coefficients can have a compounding effect, leading to poor performance.

Recent advances have utilized nonlinear methods to create low-dimensional embeddings of the full-order solution space for use in ROMs. While POD makes a linear assumption about the solution space, using nonlinear methods can capture complex and varying patterns. Many nonlinear dimensionality reduction methods, including

Parts of this chapter appear in or are adapted from our previously published papers [59, 60].

Isomap, do not provide a direct mapping back to the high-dimensional space from the low-dimensional embedding, so this limits their use in ROMs. Deep learning [14] approaches have been utilized to develop ROMs that provide efficient nonlinear solution manifolds of physical systems. Convolutional autoencoders (CAEs), a type of artificial neural network (ANN), have been used in ROMs and have been shown to outperform POD-based methods [61] for some problems. Convolutional autoencoders are adept at learning data that are spatially distributed, including the solutions to PDEs discretized over a computational domain. Autoencoder neural networks consist of two parts: an encoder, which maps high-dimensional inputs to a low-dimensional latent space, and a decoder, which maps the low-dimensional latent space to an approximation of the high-dimensional input. Deep learning and artificial intelligence (AI) methods have been at the forefront of massive recent breakthroughs in numerous fields such as computer vision, natural language processing, and recommender systems [62, 63, 64]. However, many results in the literature restrict the analysis of CAE-based ROMs to a small number of test points [15] or find no improvement over POD for larger datasets [17], so establishing their ability to generalize is important.

In this chapter, I introduce a steady non-intrusive ROM that uses convolutional autoencoders to provide expansion coefficients that use a nonlinear mapping to the full-order state, referred to as the CAE-GPR ROM. The ROM uses GPR to predict each latent variable individually. The predictive performance of the ROM is compared to the POD-GPR method when applied to a lid-driven cavity test case over a number of ROM dimensions and entire dataset.

An unsteady ROM framework using ensemble learning that combines CAEs with LSTMs, a type of recurrent neural network (RNN), is also introduced and called the CAE-eLSTM ROM. The ROM uses LSTMs for multivariate autoregressive time-series forecasting of the CAE latent variables over long time horizons at unseen points. Bootstrap aggregating (bagging), a machine learning method for ensemble learning,

is used to train multiple LSTMs that have their productions combined by averaging. This method is shown to significantly improve the stability and accuracy of the temporal prediction of expansion coefficients for two unsteady, incompressible, laminar flow problems.

## 4.1 Artificial Neural Networks

An artificial neural network is a computational model inspired by biological neural networks existing in animal brains [65]. ANNs are widely used and versatile models for regression and classification problems and learn from a training data set $\boldsymbol{\mathcal{T}} = \{\boldsymbol{X}, \boldsymbol{Y}\}$, where $\boldsymbol{X}$ and $\boldsymbol{Y}$ refer to the inputs and outputs respectively. Feedforward neural networks are a type of ANN in which information always propagates in only one direction, creating a direct mapping between inputs and outputs. Feedforward neural networks are composed of an input layer, a number of hidden layers, and an output layer. Neurons comprising these layers are associated with weights and biases, trainable parameters which are optimized during the model training stages. Figure 4.1 shows the architecture of a simple feedforward neural network with an input layer, two hidden layers, and an output layer. There are connections between each possible pair of neurons between layers, with each connection carrying a weight term and each neuron carrying a bias term with the exception of those in the input layer. Such a network is referred to as fully connected, or a multilayer perceptron (MLP). Hidden layers in MLPs are also referred to as fully connected or dense layers. Each hidden layer state $\boldsymbol{h}_j$ is computed from the state in the previous layer, $\boldsymbol{h}_{j-1}$, along with its weights $\boldsymbol{W}_j$ and biases $\boldsymbol{b}_j$ as well as an activation function $\phi(x)$

$$\boldsymbol{h}_j = \phi\left(\boldsymbol{W}_j \boldsymbol{h}_{j-1} + \boldsymbol{b}_j\right). \tag{4.1}$$

The role of activation functions is to introduce nonlinearities into the model, allowing for complex functional relationships to arise. In addition to being able to learn the training data well, neural networks should provide reasonable accuracy for unknown data of the same class, a property referred to as generalization [65]. A commonly used activation function is the rectified linear unit (ReLU) [66], which has been shown to offer better performance and ability to generalize when compared to other common activation functions [67, 68],

$$\phi(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \tag{4.2}$$

For inputs less than 0, the ReLU activation function returns a value and gradient of zero, effectively rendering certain neurons inactive. This can be problematic for network training if a large percentage of neurons exhibit this behavior and is commonly referred to as the dying ReLU problem. The leaky ReLU [69] activation function mitigates this issue, by incorporating a small positive constant $\alpha$ for negative inputs.

$$\phi(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}. \tag{4.3}$$

### 4.1.1 Training Neural Networks

#### 4.1.1.1 Backpropagation

Feedforward networks are trained using a differentiable loss function, $\mathcal{L}(\boldsymbol{\mathcal{T}}, (\boldsymbol{W}, \boldsymbol{b}))$, which calculates a measure of error between the state found in the output layer and the correct output values from the training data. The loss function serves as an objective function in an optimization problem, where its gradients with respect to the weights and biases are calculated through backpropagation [70], an algorithm utiliz-

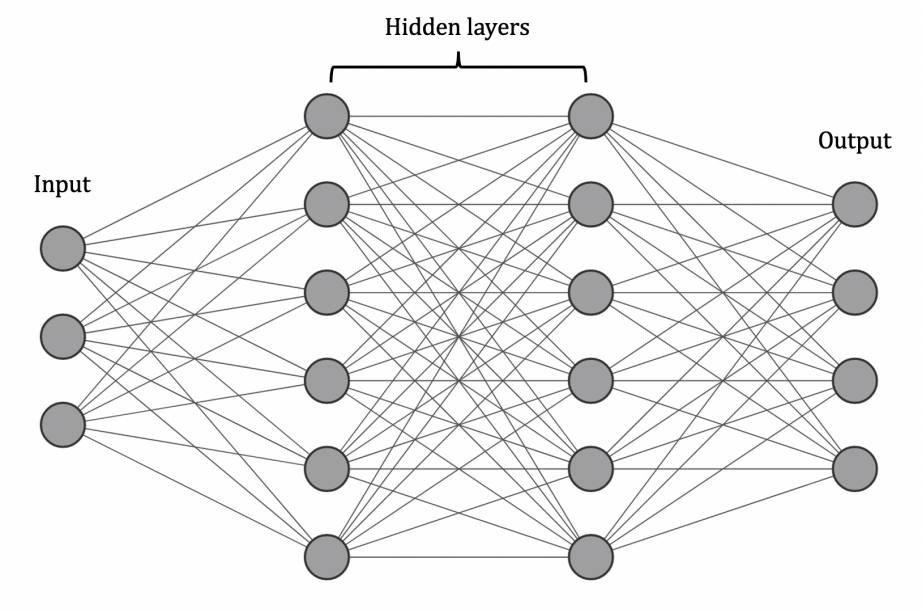



Figure 4.1: Architecture of a multilayer perceptron with a 3-dimensional input, six neurons in two fully connected layers, and four neurons in the output layer.

ing automatic differentiation. Common optimizers used in training neural networks include stochastic gradient descent (SGD) and Adam [71]. Optimizers perform a number of training epochs over $\boldsymbol{\mathcal{T}}$ in an attempt to minimize the loss function. The weights and biases update at the end of epoch $n$ according to

$$\left(\boldsymbol{W}^{n+1}, \boldsymbol{b}^{n+1}\right) = \left(\boldsymbol{W}^{n}, \boldsymbol{b}^{n}\right) - \eta \mathcal{G}\left(\frac{\partial \mathcal{L}\left(\boldsymbol{\mathcal{T}}, \left(\boldsymbol{W}^{n}, \boldsymbol{b}^{n}\right)\right)}{\partial\left(\boldsymbol{W}^{n}, \boldsymbol{b}^{n}\right)}\right), \tag{4.4}$$

where $\eta$ is the learning rate, a hyperparameter controlling the optimizer's step size and $\mathcal{G}$ is a function of the loss function's gradient dependent upon the chosen optimizer. The gradient of the loss function can be calculated using a single training sample as it is when using SGD, using the average gradient of the entire training set, or by using averages of a number of randomly selected mini-batches from $\boldsymbol{\mathcal{T}}$. Using mini-batches when training neural networks has been shown to improve the ability to generalize in addition to providing stable convergence [72]. The mini-batch size is chosen based on the size of $\boldsymbol{\mathcal{T}}$ to strike a balance between performance and computational cost.

The predictive performance of a neural network initially increases with the number of training epochs but starts to stall and then decrease as the network parameters become overly tuned towards the training data and fail to generalize, a problem referred to as overfitting. Regularization methods [73] exist to prevent overfitting. A popular and widely used regularization method is dropout [74], where during each epoch, a random fraction of neuron weights/biases in each hidden layer is deactivated by being set to 0. This leads to some neurons not contributing to the forward pass or backpropagation. The model becomes less reliant on specific neurons and neurons also avoid co-adaptation. These factors lead to better robustness and generalizability. Weight decay, a method which adds a penalty term to the loss function based on the $L^2$ norm of the model's weights/biases, is another regularization method. The penalty term prevents overfitting by lowering the likelihood that network parameters will have large numerical values. A method that uses out-of-sample data is early stopping, where the loss on a validation data set $\boldsymbol{\mathcal{V}}$ is monitored during training. If $\mathcal{L}\left(\boldsymbol{\mathcal{V}}, (\boldsymbol{w}, \boldsymbol{b})\right)$ fails to drop for a prescribed number of epochs, training is stopped.

The initial weights and biases that are used can also effect the final performance of a neural network. A commonly used weight initialization scheme for layers using ReLU activation functions is the He normal [75] initializer, which samples weights from a normal distribution centered around 0. It is a common practice to initialize the biases in each layer to 0. There is no standard and accepted approach to choosing the number of hidden layers and the number of nodes in each layer when designing multilayer perceptrons. An optimal choice depends upon a number of factors, including the number of training samples, the dimensionality of the inputs and outputs, the choice of activation functions, and the complexity of the function which is being approximated. The number of hidden layers and nodes to use is often found through a trial-and-error approach involving model validation techniques such as cross-validation. In general, the total number of trainable parameters in a network

is directly related to its capacity to learn functions. Neural networks become deeper as more hidden layers are added. However, network configurations with a large number of trainable parameters tend to overfit to the training data and fail to generalize unless regularization techniques are used. In addition, large networks are computationally expensive to train. In spite of these downfalls, deeper network architectures have become increasingly popular for complex learning tasks in multiple domains as they offer better performance [14]. Although more than two hidden layers are not required for many learning tasks, some functions are not adequately approximated by networks containing two hidden layers and using deeper networks can drastically improve performance [76, 77].

#### 4.1.1.2 Data Normalization

Similar to many other machine learning algorithms, neural networks often require that the training data be normalized in order to ensure adequate performance [78]. Data normalization allows the optimizer to learn the optimal network parameters at a much faster rate. One way to normalize the training data is to apply min-max scaling to each feature in the data matrix $\boldsymbol{X}$ containing either the inputs or outputs

$$x' = \frac{x - \min(\boldsymbol{x}_j)}{\max(\boldsymbol{x}_j) - \min(\boldsymbol{x}_j)}, \tag{4.5}$$

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

### 4.1.2 Autoencoders

Autoencoders are a type of feedforward neural network that aim to learn to reconstruct inputs in the output layer, $g : \boldsymbol{x} \rightarrow \hat{\boldsymbol{x}}$ where $\boldsymbol{x} \approx \hat{\boldsymbol{x}}$. Autoencoders use

an architecture composed of two individual feedforward neural networks. The encoder $g_{\text{enc}} : \mathbb{R}^N \rightarrow \mathbb{R}^k$ where $k \ll N$ maps a high-dimensional input $\boldsymbol{x}$ into the low-dimensional latent space $\boldsymbol{a}$. The decoder $g_{\text{dec}} : \mathbb{R}^k \rightarrow \mathbb{R}^N$ maps the latent space back to an approximation of the high-dimensional input $\hat{\boldsymbol{x}}$. The combination of the two results in

$$g : \hat{\boldsymbol{x}} = g_{\text{dec}} \circ g_{\text{enc}}(\boldsymbol{x}). \tag{4.6}$$

Autoencoders have been shown to provide robust low-dimensional representations of high dimensional data [79]. Once an autoencoder is sufficiently trained and $g(\boldsymbol{x}) \approx \boldsymbol{x}$ for all inputs over $\boldsymbol{\mathcal{T}}$, the corresponding low-dimensional codes can be passed to the decoder $g_{\text{dec}}(\boldsymbol{a})$ to obtain accurate reconstructions $\hat{\boldsymbol{x}}$ for all data in $\boldsymbol{\mathcal{T}}$. States existing outside of the training set $\boldsymbol{x}^*$ can also be well-approximated if a good approximation of the low-dimensional latent space $\boldsymbol{a}^*$ can be found. In the context of ROMs, the latent space is equivalent to the set of expansion coefficients that map from a low-dimensional representation to the high-dimensional full-order solution. Similarly, the projection of a full-order solution onto the nonlinear manifold provided by the autoencoder is given by $\hat{\boldsymbol{x}}$. Training is conducted on the combination of the encoder and decoder, while after training the encoder is often no longer useful and only the decoder is used. Figure 4.2 shows a sample architecture of a symmetric MLP autoencoder with two hidden layers between the input/output layers and latent space. Since MLP autoencoders are fully connected, the total number of trainable parameters in the network can grow very large when the dimension of the input, $N$, is high. As the number of trainable parameters increases, the amount of training data required to sufficiently train the network to make reasonably accurate predictions also grows large. This is contrary to the objective of model reduction, which aims to make predictions using a limited amount of training data.

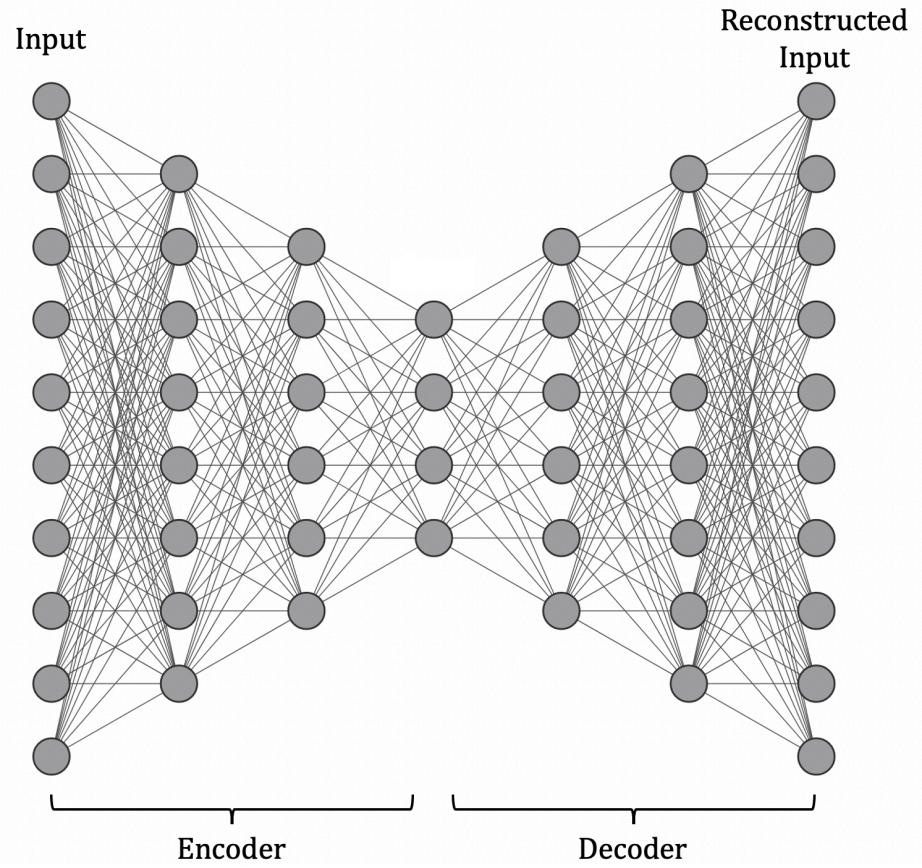



Figure 4.2: Architecture of a symmetric MLP autoencoder with two fully connected layers between the input/output and latent space.

#### 4.1.2.1 Convolutional Autoencoders

There exist neural network architectures that make use of parameter sharing, where rather than weight combinations existing for each pair of neurons between layers, multiple neurons share a single weight. Convolutional autoencoders effectively implement parameter sharing to limit the total number of trainable parameters in the network. This is done through the use of convolutional layers, which provide feature maps of input data that are spatially arranged [80]. Convolutional layers use a number of filters to convolve over spatially distributed input data, with each filter having its own set of weights. Pooling layers are also used in convolutional networks to summarize the features in input data through operations including averaging and maximization. Convolutional layers are widely used in the field of computer vision,

dealing with spatially distributed data such as images [81, 62]. CAEs can also be a useful tool for states that arise from numerically solving discretized PDEs as they tend to be spatially distributed. Data with multiple states, i.e. components of velocity or levels of red, green, and blue in images, can also be handled well by CAEs through the use of a number of input channels. More details on convolutional layers can be found in a work by Dumoulin and Visin [82]. A combination of convolutional, pooling, and fully connected layers is used to construct CAEs, as shown in a schematic of an encoder section of a CAE in Figure 4.3. Spatially distributed data arising from the solutions of discretized PDEs often vary smoothly through the computational domain. CAEs are highly adept at handling data that are naturally spatially distributed by learning spatially invariant features, allowing them to outperform other neural network architectures [83, 84]. The input and output layers of CAEs usually consist of 2-dimensional (2D) states in each channel. Training data must be reshaped before being input into the network through the use of a reshape operator

$$\boldsymbol{R} : \mathbb{R}^{N \times n_c} \rightarrow \mathbb{R}^{n_y \times n_x \times n_c}, \tag{4.7}$$

where $n_y$ refers to the number of data points in the vertical direction and $n_x$ the number of data points in the horizontal direction. The reshape operator is applied to each separate state that occupies the $n_c$ input channels. An inverse reshape operator is used to reshape state output data in each output channel into the original vector format

$$\boldsymbol{R}^{-1} : \mathbb{R}^{n_y \times n_x \times n_c} \rightarrow \mathbb{R}^{N \times n_c}. \tag{4.8}$$

### 4.1.3 Recurrent Neural Networks

Feedforward neural networks are powerful models for capturing complex functional relationships. However, they make the assumption that the inputs and outputs do not

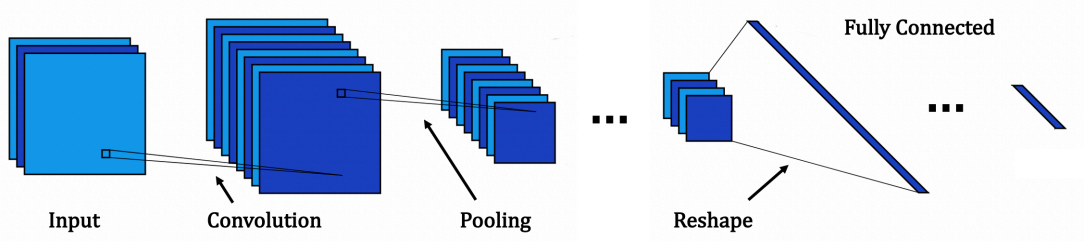



Figure 4.3: Architecture of the encoder of a convolutional autoencoder (CAE) consisting of convolutional, pooling, and fully connected layers.

exhibit any sequential or temporal dependencies. There are no mechanisms within the network to use information about the position, or order, of the inputs and outputs. This makes feedforward neural networks unsuitable for tasks such as time-series forecasting or natural language processing, where capturing the underlying order of the data is essential. Recurrent neural networks [85] (RNNs) were developed to address the limitations of feedforward neural networks in handling sequential data. RNNs utilize an architecture that processes inputs according to their sequential order through the use of a *hidden state* that retains information about previous inputs. Inputs are processed using a series of cells which contain neurons designed to update the hidden state. RNNs update the weights and biases using backpropagation through time, where the gradients are calculated with respect to the loss function for each input, or time step. A common issue encountered when training RNNs is the vanishing gradient problem [86], which occurs when gradients become very small as they are backpropagated through time. When using training data of long sequence lengths, the gradients are multiplied many times; for gradient values less than 1, this can lead to the product approaching 0. Two RNN architectures developed to address this problem are long short-term memory networks (LSTMs) [87] and gated recurrent units (GRUs) [88]. Both architectures use gating mechanisms to control information flow and use memory cells to store long-term information. LSTMs use a more involved network architecture that allows for better modeling of complex long-term dependencies, although this comes at an increased training cost due to a larger number of

network parameters.

#### 4.1.3.1 Long Short-Term Memory Neural Networks

LSTMs use a network architecture that includes a series of gates that effectively control the flow of information. An individual LSTM cell consists of a cell state $\boldsymbol{c}_t$ which acts as an internal memory, and a hidden state $\boldsymbol{h}_t$, which serves as an output. The cell state is responsible for selectively retaining or forgetting information from previous steps, while the hidden state represents the model's cumulative output. LSTM cells contain a forget gate $\boldsymbol{f}_t$, input gate $\boldsymbol{i}_t$, and output gate $\boldsymbol{o}_t$. Forget gates determine what information from previous states should be forgotten. Input gates decide what new pieces of information should be stored in the current state. Output gates use the current and previous cell states to determine what information is retained in the model. Using this architecture, the model is able to retain information that is relevant for long-range sequential dependencies and discard information that becomes irrelevant over time, making LSTMs a popular choice for sequential modeling. For a given input $\boldsymbol{a}_t \in \mathbb{R}^k$, the LSTM equations are given as

$$\begin{aligned}
\boldsymbol{f}_t &= \sigma(\mathcal{F}_f(\boldsymbol{a}_t)) \\
\boldsymbol{i}_t &= \sigma(\mathcal{F}_i(\boldsymbol{a}_t)) \\
\boldsymbol{o}_t &= \sigma(\mathcal{F}_o(\boldsymbol{a}_t)) \\
\tilde{\boldsymbol{c}}_t &= \tanh(\mathcal{F}_c(\boldsymbol{a}_t)) \\
\boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tilde{\boldsymbol{c}}_t \\
\boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t),
\end{aligned} \tag{4.9}$$

where $\sigma$ is the sigmoid function, tanh is the hyperbolic tangent function, and $\odot$ is the Hadamard product. $\mathcal{F}$ is a linear function of the weights $\boldsymbol{W}_a \in \mathbb{R}^{n_h}$ and $\boldsymbol{W}_h \in \mathbb{R}^{n_h}$ and biases $\boldsymbol{b} \in \mathbb{R}^{n_h}$, where $n_h$ is the number of neurons in the hidden layer of each LSTM cell, and is given as

$$\mathcal{F} = \boldsymbol{W}_a \boldsymbol{a}_t + \boldsymbol{W}_h \boldsymbol{h}_{t-1} + \boldsymbol{b}. \tag{4.10}$$

A unique set of weights and biases belongs to each gate or cell state. When using LSTMs for time-series prediction, an autoregressive prediction is used, often referred to as the sliding window approach. For a given input sequence containing multivariate data for $w$ timesteps, the next step in the sequence is predicted. When making the next prediction, the current prediction is incorporated into the input sequence by removing the first element and shifting the rest to the prior position. Eventually, the model inputs will consist of only previously computed predictions, which makes model performance sensitive to error propagation. The input sequence length, or window size $w$, is an important hyperparameter to consider when training LSTMs. Although LSTMs are designed to effectively learn long-range dependencies, using an input sequence length that is too long can lead to the inclusion of outdated and irrelevant information. On the other hand, using an input sequence length that is too small can ignore important long-range information.

## 4.2 Steady Non-intrusive ROM

This section describes a non-intrusive ROM for steady-state problems that uses convolutional autoencoders to provide a nonlinear solution space, referred to as the CAE-GPR ROM. Gaussian process regression is used to predict the expansion coefficients at unseen points. The offline stage involves obtaining full-order snapshots of solutions evaluated at a set of design parameters $\boldsymbol{\mathcal{U}}_{\text{train}}$ and assembling them into a snapshot matrix $\boldsymbol{S}$, similar to the POD-GPR ROM. The reshape operator $\boldsymbol{R}$ is applied to $\boldsymbol{S}$ so the data can be used to train a CAE with architecture $\boldsymbol{\mathcal{C}}$. The set of expansion coefficients $\boldsymbol{A}_{\text{train}}$ is obtained for the training set using $g_{\text{enc}}$. $k$ GPR models $\mathcal{F} = [f_1(\boldsymbol{\mu}), f_2(\boldsymbol{\mu}), \cdots f_k(\boldsymbol{\mu})]$ are trained on $\{\boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{A}_{\text{train}}\}$ and saved for use

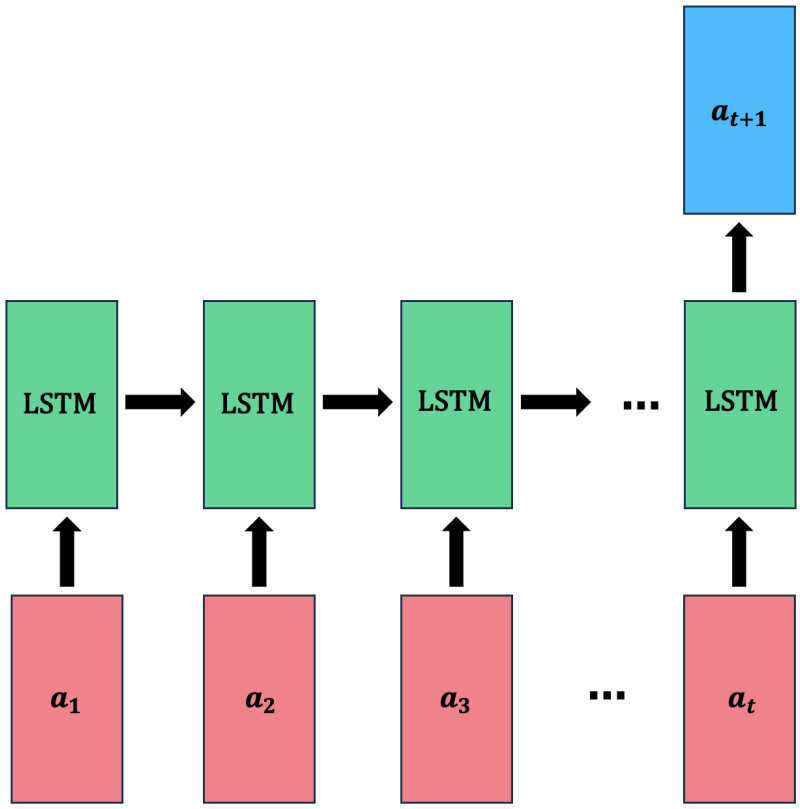



Figure 4.4: Diagram of a single-layer LSTM neural network making a prediction one timestep ahead.

in the offline stage along with the decoder $g_{\text{dec}}$. During the online stage, approximate expansion coefficients $\tilde{\boldsymbol{a}}^*$ are passed to the decoder to obtain approximate solutions $\tilde{\boldsymbol{X}}^*$. The CAE-GPR method is outlined in Algorithm 6. Training the convolutional autoencoder makes the offline stage of CAE-GPR more expensive than POD-GPR, while the online costs for both models are similar.

### 4.2.1 Lid-driven Cavity

The lid-driven cavity case from Section 3.3.1 is used to test the performance of both the CAE-GPR and POD-GPR ROMs over a number of ROM dimensions. 500

**Algorithm 6** Offline and online stages of CAE-GPR method

**function** CAEGPR_ OFFLINE($\boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{\mathcal{C}}, \boldsymbol{R}$)
    Compute high-fidelity solutions for $\boldsymbol{\mu} \in \boldsymbol{\mathcal{U}}_{\text{train}}$ by solving FOM and assemble into $\boldsymbol{S}$
    Apply reshape operator $\boldsymbol{R}$ to $\boldsymbol{S}$ to obtain $\boldsymbol{X}_{\text{train}}$
    Train convolutional autoencoder with architecture $\boldsymbol{\mathcal{C}}$ on $\{\boldsymbol{X}_{\text{train}}, \boldsymbol{X}_{\text{train}}\}$
    Calculate expansion coefficients for training data $\boldsymbol{A}_{\text{train}} = g_{\text{enc}}\left(\boldsymbol{X}_{\text{train}}\right)$
    Train $k$ GPR models $\mathcal{F} = [f_1(\boldsymbol{\mu}), f_2(\boldsymbol{\mu}), \cdots f_k(\boldsymbol{\mu})]$ for each expansion coefficient in $\{\boldsymbol{\mathcal{U}}_{\text{train}}, \boldsymbol{A}_{\text{train}}\}$
    **return** $(g_{\text{dec}}, \mathcal{F})$
**end function**

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

design parameters are generated with LHS using the same bounds,

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

Again, the computational mesh consists of $64 \times 64$ cells uniformly distributed in the $x$ and $y$ directions and one cell spanning the $z$ direction, resulting in $N = 4096$ and a reshape operator $\boldsymbol{R}$ with $n_y, n_x = 64$ and $n_c = 2$. The full-order states of $u$ and $v$ are used to compare the performance of the POD-GPR and CAE-GPR ROMs. The metric of performance used to compare the ROMs is the relative $L^2$ error $\epsilon_{\text{ROM}}$ between the FOM state $\boldsymbol{x}$ and the ROM approximated state $\tilde{\boldsymbol{x}}$

$$\epsilon_{\text{ROM}} = \frac{\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2}{\|\boldsymbol{x}\|^2}. \tag{4.11}$$

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

$$\epsilon_{\text{Proj}} = \frac{\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2}{\|\boldsymbol{x}\|^2}. \tag{4.12}$$

The CAE architecture is given in Table 4.1. The network has two input channels, one for each of the velocity components. The encoder consists of a combination of convolutional, pooling, and fully connected layers. The decoder consists of fully connected and transpose-convolutional layers. All of the layers with the exception of the output use a leaky ReLU activation function with $\alpha = 0.25$. There are a relatively small number of convolutional and pooling layers in the network; I found that adding more of them did not improve the network performance, although their absence (using a MLP) causes a large decrease in performance. Compared to images which can be very noisy, the physical states in the presented problem vary smoothly, and fewer convolutional layers are required to learn features. I found that having a fully connected layer on either side of the latent space is important for network performance, although this does signficantly increase the number of network parameters. The learning task at hand requires that reconstructions of data be highly accurate, and networks with more parameters allow for more robust functional relationships to arise. Min-max scaling is used independently on $u$ and $v$ before training and the CAE outputs are then scaled back to their original range after prediction. The output layer uses the sigmoid activation function, which scales into the range [0,1], ensuring that the outputs can be scaled back to their original range for prediction, and is given as

$$\phi(x) = \frac{1}{1 + e^{-x}}. \tag{4.13}$$

A maximum number of 7500 training epochs are used, and early stopping is enforced if the validation loss fails to decrease over 500 epochs. A mini-batch size of 8 is used for training and the mean squared error loss function is used. The Adam optimizer is used with an initial learning rate of of $\eta = 3 \times 10^{-4}$. A five-fold cross-validation approach is used to assess the performance of the ROM over the entire dataset, creating five folds of the dataset containing 400 training samples and 100 test/validation samples. These 100 samples are split evenly into 50 test and 50 validation samples, which are used to monitor the autoencoder loss during training. While every sample in the dataset is used for training, only half are used for prediction. An average cross-validation error is reported for both the ROM prediction and projection errors over all of the prediction points. The validation samples are not used for the POD-GPR ROM, which uses an individual ROM for both $u$ and $v$.

| Layer | Filters | Kernel | Stride | Activation Function | Output Size |
|---|---|---|---|---|---|
| Input | | | | | $64 \times 64 \times 2$ |
| Convolutional | 64 | $3 \times 3$ | $1 \times 1$ | Leaky ReLU | $64 \times 64 \times 64$ |
| Max-Pooling | | $2 \times 2$ | $2 \times 2$ | | $32 \times 32 \times 64$ |
| Convolutional | 32 | $3 \times 3$ | $1 \times 1$ | Leaky ReLU | $32 \times 32 \times 32$ |
| Max-Pooling | | $2 \times 2$ | $2 \times 2$ | | $16 \times 16 \times 32$ |
| Reshape | | | | | 8192 |
| Fully Connected | | | | Leaky ReLU | 128 |
| Latent Space | | | | Leaky ReLU | $k$ |
| Fully Connected | | | | Leaky ReLU | 128 |
| Fully Connected | | | | Leaky ReLU | 8192 |
| Reshape | | | | | $16 \times 16 \times 32$ |
| Convolutional Transpose | 32 | $3 \times 3$ | $2 \times 2$ | Leaky ReLU | $32 \times 32 \times 32$ |
| Convolutional Transpose | 64 | $3 \times 3$ | $2 \times 2$ | Leaky ReLU | $64 \times 64 \times 64$ |
| Convolutional Transpose | 2 | $3 \times 3$ | $1 \times 1$ | Sigmoid | $64 \times 64 \times 2$ |

Table 4.1: Detailed convolutional autoencoder architecture used for the lid-driven cavity problem.

The POD-GPR ROM is evaluated for both projection and prediction errors at ROM dimensions $k \in [1, 2, \cdots 35]$ while the CAE-GPR ROM is similarly evaluated at $k \in [2, 3, 5, 10, \cdots 35]$. Figure 4.5 shows the cross-validation relative errors for $u$ and $v$; for evaluated ROM dimensions $k > 2$, the CAE-GPR ROM exhibits higher

predictive performance over the data set. The projection error provided by the CAE is high at $k = 2$ and comparable to that of POD, and it decays rapidly until $k = 5$, after which it does not vary much. Similar results are presented in a work by Lee and Carlberg [15]. Very low values of $k$ decrease the capacity of the network to learn meaningful low-dimensional representations of the training data. As $k$ increases, the relative difference in the projection and prediction errors grows. This is an expected result since prediction errors in the individual expansion coefficients will have a cascading effect. The cross-validation projection error from POD continues to decay after $k = 35$, while the POD-GPR prediction error flattens out at around $k = 20$. The cross-validation projection error produced by the CAE for both $u$ and $v$ is lower than that of POD until around $k = 25$; even with a higher projection error, the predictive performance offered by CAE-GPR exceeds that of POD-GPR for $k \geq 25$. The CAE offers a set of expansion coefficients that are more easily predicted when using GPR compared to POD, which sees its predictive performance stall after $k$ reaches a certain value, a commonly found result for POD-GPR based ROMs [28]. In addition to giving better performance in terms of both projection and prediction for low values of $k$, the use of a nonlinear solution space for ROM construction offers a more robust relationship between the design parameters and expansion coefficients. The difference in projection and prediction errors from POD-GPR is almost 0 at low values of $k$, but rapidly increases as $k$ grows, suggesting that the individual expansion coefficients become harder to interpolate as the corresponding singular values decay.

Figure 4.6 shows the relative error plot for both ROMs at $k \in [5, 10, \cdots 35]$ for a single unseen design parameter $\boldsymbol{\mu}^* = [1.167, 1.997, -0.4665, 555.5]$, while Figure 4.7 shows the contour plots of the FOM as well as the absolute error plots for CAE-GPR at $k = 5$ and POD-GPR at $k = 35$. Results at another design parameter instance at $\boldsymbol{\mu}^* = [1.963, 1.789, 0.5890, 308.5]$ are shown in Figures 4.8 and 4.9. The generalized results from the cross-validation also hold here; for a greater projection error, CAE-

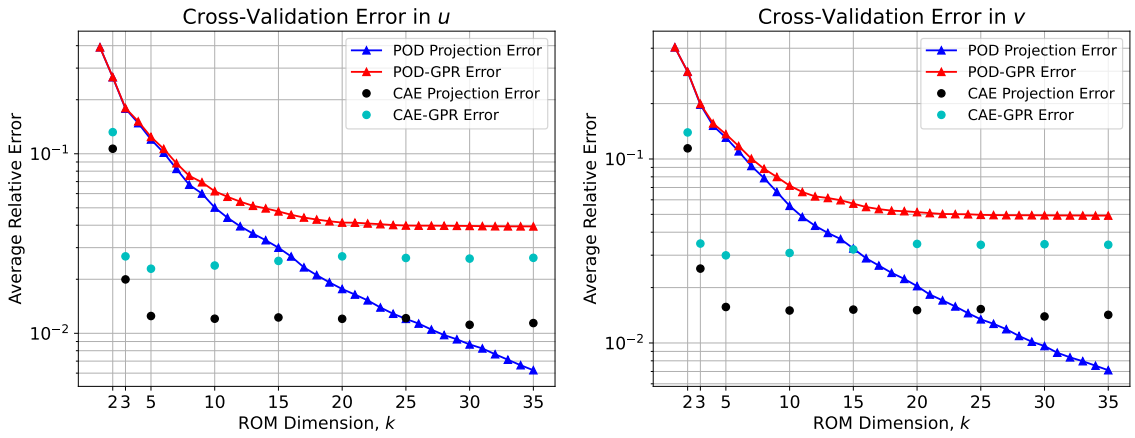



Figure 4.5: Cross-validation prediction and projection errors in $u$ and $v$ for both ROMs at different values of $k$.

GPR provides a lower prediction error. It is also shown at these design parameters that the prediction error curve of POD-GPR flattens out. There is more volatility in both the projection and prediction errors for CAE-GPR, although POD-GPR still never outperforms it in predicting $u$ and $v$. For $k = 10$ at both design parameters, the difference in the projection and prediction errors is very small, and almost 0 for $u$ at $\boldsymbol{\mu}^* = [1.167, 1.997, -0.4665, 555.5]$ and $v$ at $\boldsymbol{\mu}^* = [1.963, 1.789, 0.5890, 308.5]$. This is similar to the behavior exhibited by POD-GPR for low values of $k$, showing that CAE-GPR is also capable of producing highly accurate estimates of the expansion coefficients. The error contours at these design parameters highlight the increased predictive performance given by CAE-GPR over POD-GPR. While the error contours given by POD-GPR exhibit distinct bands of high error, the contours produced by CAE-GPR are generally more uniform and dispersed throughout the domain. At the chosen design parameters, there is a significant decrease in relative error; at $\boldsymbol{\mu}^* = [1.167, 1.997, -0.4665, 555.5]$, the percent decreases in relative error of $u$ and $v$ from POD-GPR to CAE-GPR are 42.4% and 49.8% respectively, while at $\boldsymbol{\mu}^* = [1.963, 1.789, 0.5890, 308.5]$ they are 48.3% and 75.6% respectively.

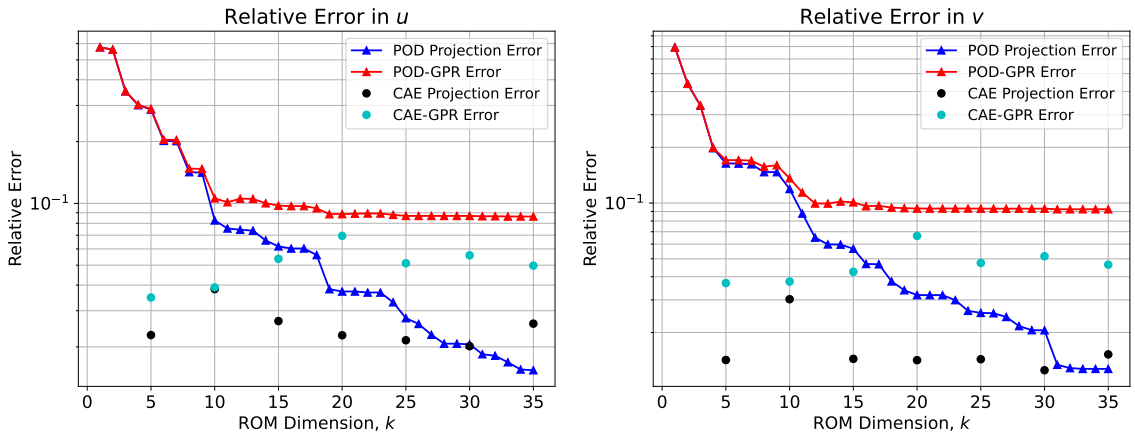
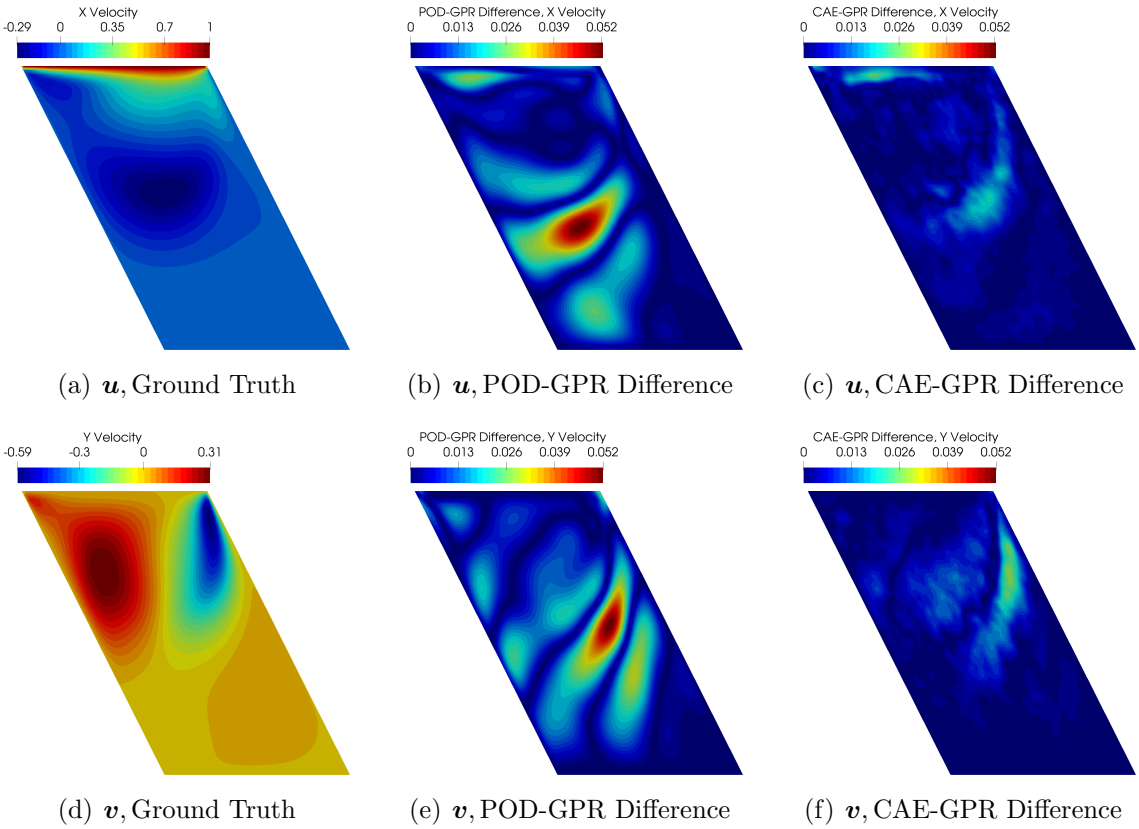



Figure 4.6: Plots of the prediction and projection errors in $u$ and $v$ for both ROMs at $\boldsymbol{\mu}^* = [1.167, 1.997, -0.4665, 555.5]$ at different values of $k$.




(a) $\boldsymbol{u}$, Ground Truth (b) $\boldsymbol{u}$, POD-GPR Difference (c) $\boldsymbol{u}$, CAE-GPR Difference

(d) $\boldsymbol{v}$, Ground Truth (e) $\boldsymbol{v}$, POD-GPR Difference (f) $\boldsymbol{v}$, CAE-GPR Difference

Figure 4.7: ROM comparison of $u$ and $v$ at $\boldsymbol{\mu}^* = [1.167, 1.997, -0.4665, 555.5]$, with $k = 5$ for CAE-GPR and $k = 35$ for POD-GPR.

**Convolutional autoencoder training** Figure 4.10 shows the training and validation losses against the number of epochs for selected folds of the training and valida-

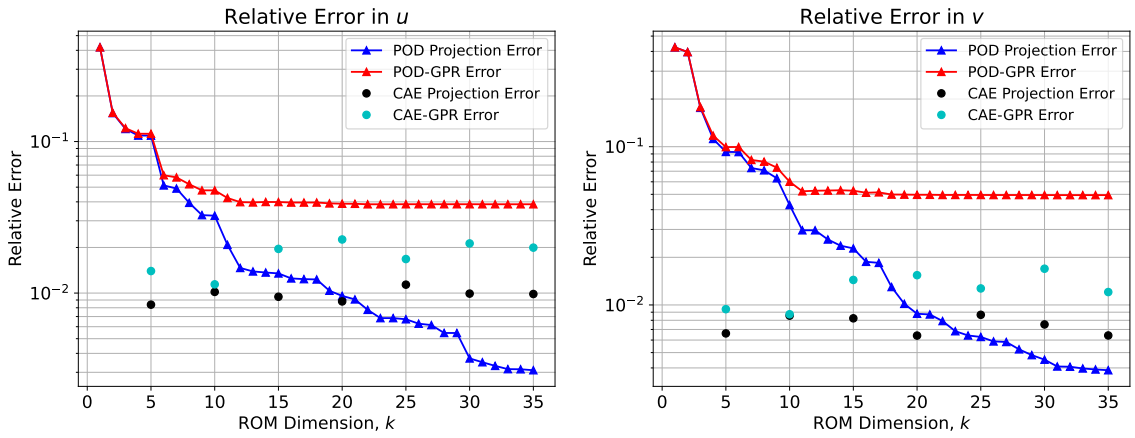
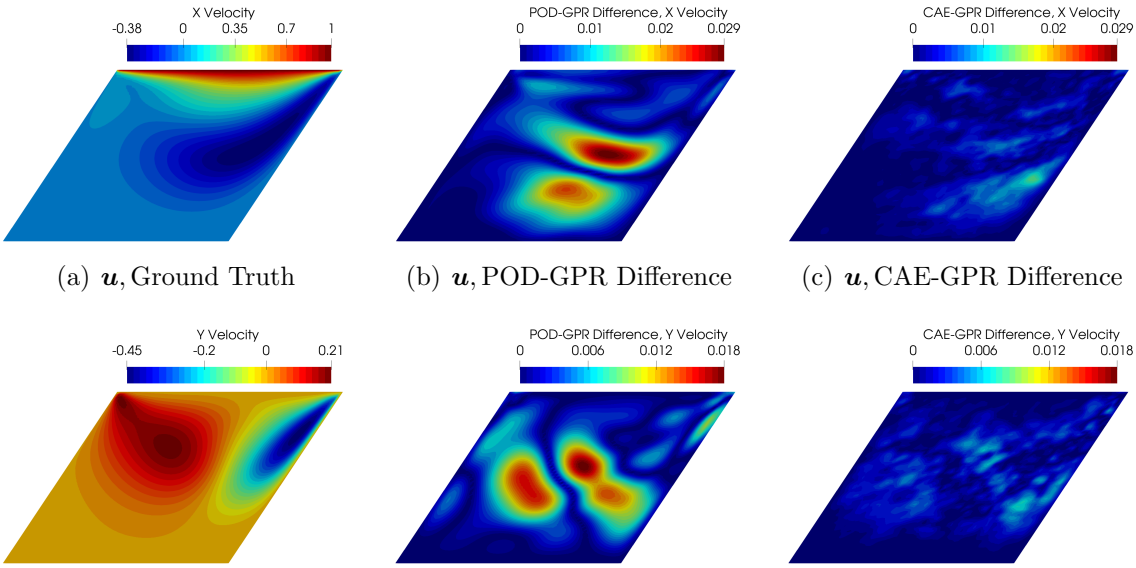



Figure 4.8: Plots of the prediction and projection errors in $u$ and $v$ for both ROMs at $\boldsymbol{\mu}^* = [1.963, 1.789, 0.5890, 308.5]$ at different values of $k$.




(a) $\boldsymbol{u}$, Ground Truth (b) $\boldsymbol{u}$, POD-GPR Difference (c) $\boldsymbol{u}$, CAE-GPR Difference

(d) $\boldsymbol{v}$, Ground Truth (e) $\boldsymbol{v}$, POD-GPR Difference (f) $\boldsymbol{v}$, CAE-GPR Difference

Figure 4.9: ROM comparison of $u$ and $v$ at $\boldsymbol{\mu}^* = [1.963, 1.789, 0.5890, 308.5]$, with $k = 5$ for CAE-GPR and $k = 35$ for POD-GPR.

tion data for $k = 5, 10, 25, 30$. Early stopping is used as a regularization method, and the validation loss fails to drop for 500 epochs well before the maximum number of 7500 epochs at $k = 5, 10, 25$. At $k = 30$, training stops after 7491 epochs. While the training loss continues to decline slowly in all of the plots, the validation loss shows asymptotic behavior. By monitoring the validation loss and using early stopping, the

network is prevented from overfitting the training data. Training is performed on an NVIDIA TITAN RTX GPU. The average wall time and number of epochs for training the CAE over all of the data folds is shown in Table 4.2; in general, increasing $k$ leads to higher computational costs. Both the number of trainable parameters and capacity of the network to learn are affected by the size of the latent space.

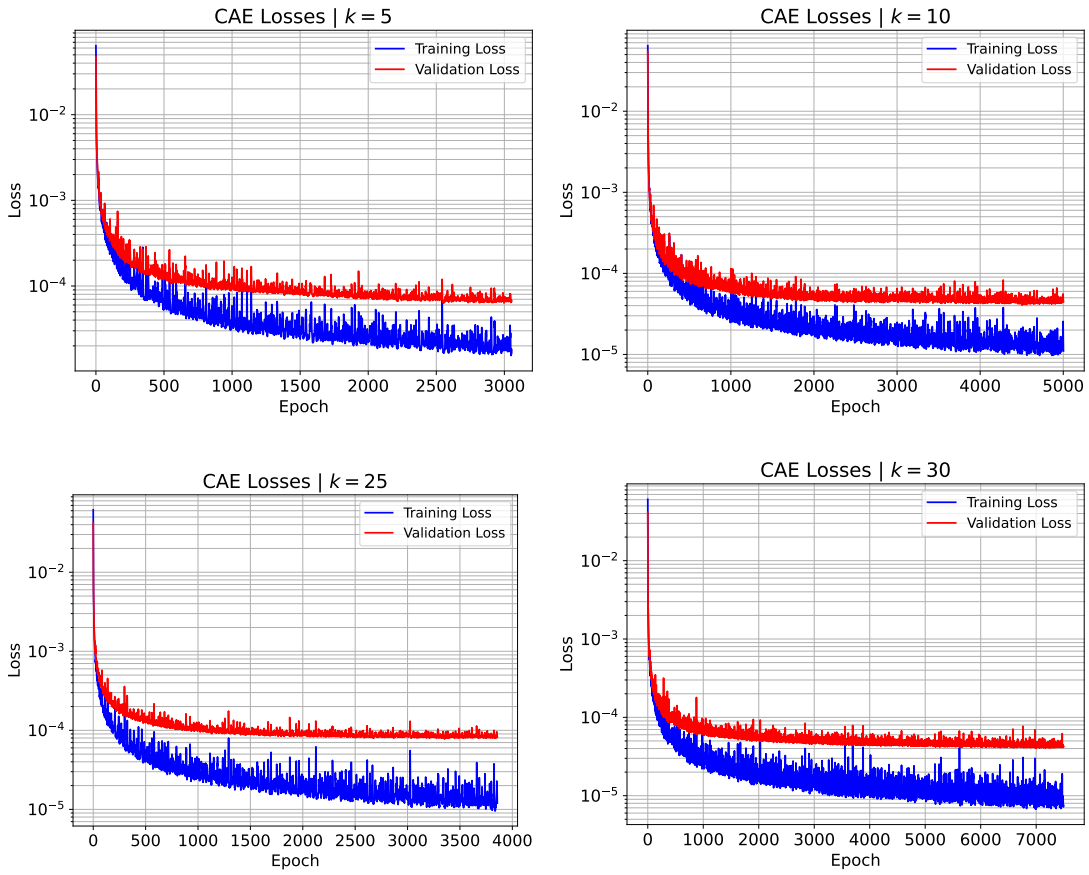



Figure 4.10: Training and validation losses at different ROM dimensions $k$ for a selected fold of the training and validation data.

**Activation function performance** In addition to the leaky ReLU activation function, results are also presented for the projection errors obtained when using the ReLU activation function as well as the tanh activation function, another popular choice for

| ROM Dimension, $k$ | Average Wall Time (s) | Average Number of Epochs |
|---|---|---|
| 5 | 884 | 3533 |
| 10 | 883 | 3564 |
| 15 | 867 | 3441 |
| 20 | 1123 | 4465 |
| 25 | 971 | 3865 |
| 30 | 1283 | 4675 |
| 35 | 1301 | 4519 |

Table 4.2: Average computational costs over all data folds for training the CAE.

neural networks which is given as

$$\phi(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4.14}$$

This is done by replacing all of the leaky ReLU activation functions ($\alpha = 0.25$) in the network in Table 4.1 with either ReLU or tanh and following the same network training procedure. Figure 4.11 shows the cross-validation projection errors obtained using networks with the three different activation functions for ROM dimensions $k \in [5, 10, \cdots 35]$ and POD. It is shown that the network using the ReLU activation function is outperformed by POD for the chosen values of $k$, and that leaky ReLU offers slightly improved performance over tanh in reconstructing $u$ and $v$. Using the leaky ReLU activation function overcomes the dying ReLU problem, which is shown to lead to networks with very poor predictive performance. The initial randomization of the network weights and biases may lead to poor performance when using the ReLU activation function; for $k = 15$, the maximum value of $\epsilon_{\text{Proj}}$ for $u$ at a single test fold is $4.228 \times 10^{-1}$, while the second highest value is $2.372 \times 10^{-2}$, which is similar to the errors provided by using leaky ReLU or tanh. The poor performance given by using ReLU is not restricted to single folds of the data either; for a specific fold over the selected $k$, $\epsilon_{\text{Proj}}$ for $u$ has a maximum of $4.233 \times 10^{-1}$ at $k = 5$ and minimum of $2.1028 \times 10^{-2}$ at $k = 25$. This implies that using the ReLU activation function leads to network performance being sensitive to the initialization of the weights and biases,

in contrast to using tanh or leaky ReLU which offer more consistent performance.

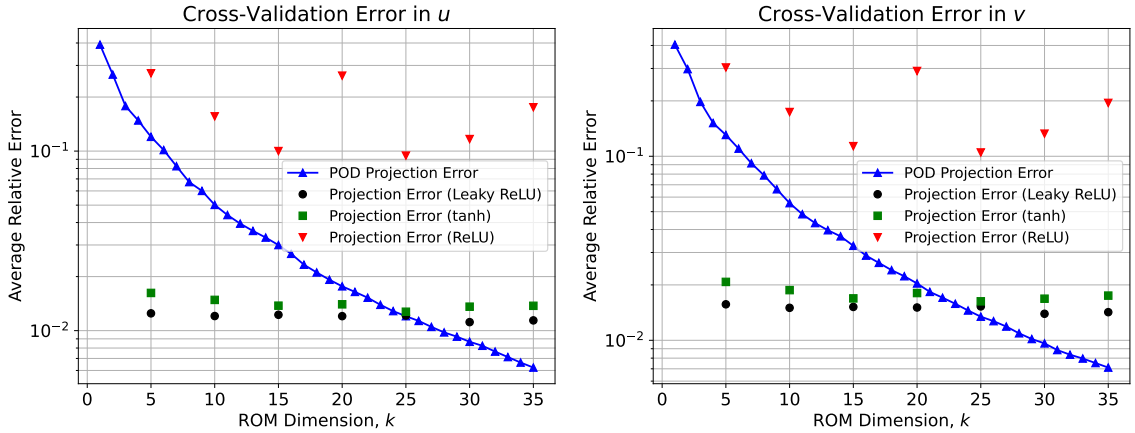



Figure 4.11: Cross-validation projection errors in $u$ and $v$ using different activation functions.

## 4.3 Unsteady Non-intrusive Ensemble ROM

An important result from the previous section is that the projection error given by CAEs does not exhibit a decrease with the ROM dimension, unlike POD. While the projection error does initially decrease for very low ROM dimensions, it stabilizes as the ROM dimension grows. CAEs can effecitively represent the solution space using a small number of expansion coefficients, which make them suitable for highly nonlinear problems. Another situation where this can be advantageous is when the number of training snapshots is high. Unsteady ROMs applied to unseen designs require multiple temporal snapshots per training point, which can become prohibitively large for POD.

Unsteady non-intrusive ROMs combine either POD or CAE for spatial reconstruction of full-order states with a model used to make time-series predictions of the expansion coefficients. Deep learning methods are popular for this as well, including LSTMs [89] and transformer neural networks [90]. Both LSTMs and transformers are powerful models for handling data that are sequential in nature. Transformers utilize an architecture that is much more complex than the one LSTMs use, which leads to

higher computational costs for both training and inference.

When making time-series predictions at unseen data sets over a long time horizon, error propagation is a common issue. Errors made in early predictions can accumulate and compound over time, leading to very large inaccuracies. Unseen data sets are particularly suspect to this, as the model may not account for shifting data patterns. Additionally, there is no feedback from previously seen data to correct errors. This phenomenon is commonly observed in data-driven CFD applications [91, 16]. As a result, most examples in the literature focus on applying unsteady non-intrusive ROMs to single-parameter problems [89, 90, 92], where the ROM is both trained on and used for a single design. I have identified two examples that combine CAEs and LSTMs for non-intrusive ROMs and apply them to unseen designs [93, 94]. The work by Maulik et al. does not mention the error propagation issue, while the work by Hasegawa et al. presents results of low accuracy. A recent paper [95] by Jeon et al. developed a hybrid AI-CFD method using flow residuals to address the issue of error propagation. This method is intrusive and requires the ROM to have access to the CFD solver for computing the residuals, which leads to a more computationally intensive online stage.

In this section, ensemble learning [96], a machine learning technique for improving the stability and lowering the variance of predictive models, is used to develop a fully data-driven framework referred to as the CAE-eLSTM ROM. Ensemble learning involves combining multiple base models, referred to as *weak learners*, to provide a composite model that offers greater accuracy. To this end, bootstrap aggregating (bagging) is used as the ensemble learning method for the temporal portion of the ROM, which involves training the weak learners on subsets of the dataset chosen randomly through sampling with replacement.

### 4.3.1 Ensemble Learning

While LSTMs are powerful models for time-series prediction, the quality of predictions over long time horizons on data existing outside of the training set is highly sensitive to the weights and biases of the model. Additionally, the total number of weights and biases is usually very large, which makes finding an optimal configuration very difficult, even when using state of the art optimization algorithms.

To mitigate the issue of high model variance, ensemble learning is a commonly used approach. By leveraging multiple base models that exhibit high variance, ensemble methods significantly reduce errors and improve robustness. Boosting and bootstrap aggregating (bagging) are the two main types of ensemble learning methods. Boosting [97] trains individual models sequentially, where each subsequent weak learner focuses on correcting prediction errors from the previous ones. While boosting is widely used and offers good performance, the base models must be trained iteratively, which incurs a large computational cost, especially in the context of neural networks.

Bagging [98] is an algorithm that consists of two stages: bootstrapping and aggregation. Bootstrapping is a resampling technique where multiple random subsets of the data set, chosen through sampling with replacement, are constructed. The subsets of the data typically have the same number of data points as the original data set. This approach leads to individual data points being present multiple times in the individual subsets. Multiple base models are trained individually on each of the bootstrapped data sets, which can be done in parallel. The aggregation stage creates an ensemble model by taking an average of the predictions given by each weak learner. Given $m$ weak learners $f_i$, the aggregate prediction $\bar{f}$ is given as

$$\bar{f} = \frac{1}{m} \sum_{i}^{m} f_i. \tag{4.15}$$

Bagging mitigates the issue of error propagation by averaging the errors of multiple weak learners, which will become increasingly small as the number of learners grows, leading to much lower variance. Bagging also offers a considerable gain in robustness and generalization by leveraging the diversity of the weak learners to increase the capability of handling varying patterns outside of the training data. Figure 4.12 shows an example of bootstrapping. The bootstrapped datasets can contain more or less instances of the original data points.

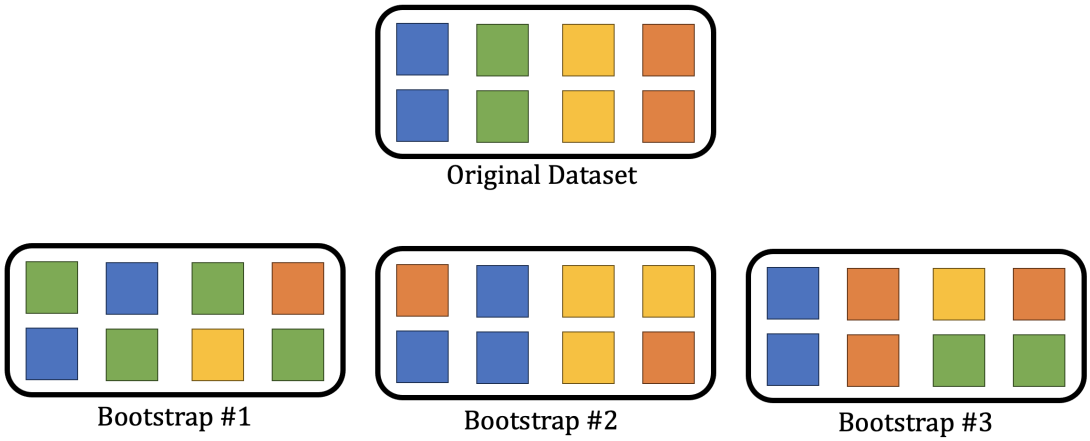



Figure 4.12: An example of bootstrapping, where random subsets of the original dataset are chosen through selection with replacement.

### 4.3.2 CAE-eLSTM ROM

The non-intrusive CAE-eLSTM ROM framework combining convolutional autoencoders for spatial reconstruction and LSTM ensembles utilizing bagging for temporal prediction is described here. Solutions to the full-order model for designs $\boldsymbol{\mu}$ in $\boldsymbol{\mathcal{U}}_{\text{train}}$ are obtained during the offline stage and are assembled into a matrix $\boldsymbol{X}$. A convolutional autoencoder with latent dimension $k$ is trained on $\boldsymbol{X}$ for a sufficient number of epochs such that inputs are accurately reconstructed. Next, the expansion coefficients for training data $\boldsymbol{A}_{\text{train}}$ are computed using the encoder $g_{\text{enc}}$. Sequences of length $w$ are then generated from $\boldsymbol{A}_{\text{train}}$ and $m$ bagged LSTMs are trained on individual data subsets chosen randomly with replacement.

The online stage involves executing the ROM to compute approximate solutions at a point $\boldsymbol{\mu}^*$, where the full-order model is initially run for $T_i$ timesteps. An initial sequence of latent variables of length $w$ is required as an input to the ROM. This means that the FOM must be run for at least $w$ timesteps, and potentially longer depending on the how useful initial simulation data is for the ROM. The latent variables for the rest of the simulation are calculated autoregressively using the bagged LSTMs by taking an average of the individual predictions. The online and offline stages are outlined in Algorithm 7. When using a ROM with a single LSTM as in previous works [93, 94], the LSTM is trained on the entire time-series dataset and is used alone for time-series prediction.

**Algorithm 7** Offline and online stages of CAE-eLSTM ROM

**function** CAE_ eLSTM_ OFFLINE($\boldsymbol{\mathcal{U}}_{\text{train}}, k, m, w$)
    Compute high-fidelity solutions for $\boldsymbol{\mu} \in \boldsymbol{\mathcal{U}}_{\text{train}}$ by solving FOM and assemble into $\boldsymbol{X}$.
    Train convolutional autoencoder with latent dimension $k$ on $\boldsymbol{X}$.
    Calculate expansion coefficients for training data $\boldsymbol{A}_{\text{train}} = g_{\text{enc}}(\boldsymbol{X})$.
    Train $m$ LSTMs $\mathcal{E} = [\text{LSTM}_1(\boldsymbol{A}), \text{LSTM}_2(\boldsymbol{A}), \cdots \text{LSTM}_m(\boldsymbol{A})]$ using a window size of $w$ on sequences selected randomly with replacement from $\boldsymbol{A}_{\text{train}}$.
    **return** $(g_{\text{enc}}, g_{\text{dec}}, \mathcal{E})$
**end function**

**function** CAEGPR_ eLSTM_ ONLINE($\boldsymbol{\mu}^*, g_{\text{enc}}, g_{\text{dec}}, \mathcal{E}$)
    Run FOM for $T_i$ timesteps at $\boldsymbol{\mu}^*$ and compute the expansion coefficients $\boldsymbol{A}^*_{T_i} = g_{\text{enc}}(\boldsymbol{X}_{T_i})$.
    **for** $t \in \{T_i, T_{i+1}, \ldots, T-1\}$ **do**
        Compute $\boldsymbol{a}^*_{t+1} = \mathcal{E}(\boldsymbol{A}^*_t)$ by using an average of the $m$ individual LSTM predictions.
        Incorporate $\boldsymbol{a}^*_{t+1}$ into the current window $\boldsymbol{A}^*_t$.
    **end for**
    Predict full-order snapshots $\tilde{\boldsymbol{X}}^* = g_{\text{dec}}(\boldsymbol{A}^*)$
    **return** $\tilde{\boldsymbol{X}}^*$
**end function**

### 4.3.3 Results

The two test cases used to demonstrate the performance of the CAE-eLSTM ROM are a lid-driven cavity and the flow over a cylinder. Both cases use two-dimensional computational domains. The lid-driven cavity case consists of three design parameters controlling the geometry of the domain, while the cylinder case consists of two design parameters controlling geometric and physical properties of the simulation. Both cases involve simulating unsteady, incompressible, laminar flow by solving the unsteady Navier-Stokes equations, which are found by adding a time derivative term for the velocity to Equation 2.2. The ROM is used to predict the vertical and horizontal components of the velocity. The training data for each velocity component are scaled to a range [0,1] using min-max scaling before being used for the CAE. Similarly, the CAE latent variables are also scaled using min-max scaling before being used for the LSTM. An NVIDIA DGX system consisting of 8 NVIDIA A100 GPUs is used to train the CAE and LSTMs, run the cylinder simulations, and for ROM inference. The lid-driven cavity simulations are run on a local workstation using a single CPU core.

The performance of the ensemble ROM is tested against a CAE-LSTM ROM that uses a single LSTM model. Using five different random initializations of weights and biases prescribed by different seeds, the time-series of latent variables are visually compared in addition to performance metrics averaged over the seeds. When training bagged LSTMs for a single seed, the same initial weights and biases are used. The seeds used to initialize the LSTM model are $s$ =[1, 2, 3, 4, 5]. For a single test point $\boldsymbol{\mu}^*$ and seed $s$, the error metric of interest is the relative $L^2$ error in a field component averaged over the prediction time horizon,

$$\epsilon_s = \frac{1}{T - Ti} \sum_{t=T_i+1}^{T} \frac{\|\boldsymbol{x}^t - \hat{\boldsymbol{x}}^t\|^2}{\|\boldsymbol{x}^t\|^2}. \tag{4.16}$$

The errors are then averaged over each seed to give a single error metric $\bar{\epsilon}$

$$\bar{\epsilon} = \frac{1}{5}\sum_{s=1}^{5}\epsilon_s. \tag{4.17}$$

To measure how sensitive the error is to the initial weights and biases of the model, the sample standard deviation $\sigma_s$ of $\epsilon_s$ is used,

$$\sigma_s = \text{std}(\epsilon_s). \tag{4.18}$$

A lower standard deviation in the error term indicates that the model performance does not vary much with the initial weights and biases.

#### 4.3.3.1 Lid-driven Cavity

The first test case is a geometrically parameterized two-dimensional lid-driven cavity flow that is similar to the one presented in Sections 3.3.1 and 4.2.1. The design parameters are the geometric parameters shown in Figure 3.3 while $Re$ is fixed to 400. The bounds for the design parameters are given as

$$\mu_1 \in [1.2, 1.8],$$
$$\mu_2 \in [1.2, 1.8],$$
$$\mu_3 \in [-\frac{\pi}{6}, \frac{\pi}{6}].$$

The computational mesh consists of $128 \times 128$ cells distributed uniformly in the $x$ and $y$ directions, resulting in $N = 16,384$. The initial condition is the solution to steady, incompressible, laminar flow at $Re = 20$. Unsteady flow is simulated using the standard OpenFOAM solver icoFoam for $T = 5$ seconds with data being saved every 0.025 seconds, leading to 200 time snapshots for a single simulation.

100 sets of design parameters are generated and randomly split into 90 training samples and 10 test points, which are given in Table 4.3. The CAE-eLSTM ROM uses $m = 64$ bagged LSTMs and a window size $w = 20$. These parameters were chosen through a trial-and-error process with the goal of maximizing predictive accuracy while trying to keep the computational cost of training the LSTM ensemble relatively low. The CAE latent dimension is set to $k = 4$; below this value, the CAE reconstruction errors were found to be worse, and increasing $k$ offered no improvement. The LSTM architecture consists of two hidden layers consisting of 50 neurons each and a dropout rate of 0.1. The output layer contains a sigmoid activation function so the outputs are constrained to a range of $[0, 1]$. The Adam optimizer is used to train both the CAE and LSTM, with an initial learning rate of $\eta = 5 \times 10^{-4}$ and weight decay of $\lambda = 1 \times 10^{-6}$. The CAE is trained for a total of 200 epochs, while an individual LSTM is trained for 250 epochs. At the test points, the full-order model is simulated for $T_i = 0.75$ seconds (15 snapshots), or 15% of the total time horizon.

| Test Case Index | $\mu_1$ | $\mu_2$ | $\mu_3$ |
|---|---|---|---|
| 1 | 1.473 | 1.701 | 0.288 |
| 2 | 1.659 | 1.653 | -0.372 |
| 3 | 1.593 | 1.611 | 0.455 |
| 4 | 1.209 | 1.443 | 0.371 |
| 5 | 1.299 | 1.689 | -0.0367 |
| 6 | 1.371 | 1.311 | -0.351 |
| 7 | 1.443 | 1.617 | -0.487 |
| 8 | 1.232 | 1.335 | -0.330 |
| 9 | 1.719 | 1.785 | -0.455 |
| 10 | 1.281 | 1.449 | -0.340 |

Table 4.3: Test case design parameters for the lid-driven cavity problem.

Figure 4.13 shows the trajectories of the latent variables computed using ensemble and single LSTMs at the test point $\boldsymbol{\mu}^* = [1.299, 1.689, -0.0367]$ using five different seeds which control the initial weights and biases of the LSTM. It is shown that for all of the latent variables, using LSTM ensembles leads to higher prediction accuracy

and significantly lower variance. When using a single LSTM, the different predictions quickly diverge from each other and the ground truth. The ensemble model is much less sensitive to the initial seed, and the predicted trajectories do not differ much. The second latent variable exhibits diverging trajectories when using the ensemble model, but the effect is much less pronounced than when using a single LSTM. Figure 4.14 shows contours of $u$ and $v$ at the test point as well as absolute ROM errors averaged over the last 10% of the simulation ($t \in [4.5,\ 5]$ seconds) for a single seed. The errors are signifcantly larger when using a single LSTM throughout the computational domain for both $u$ and $v$. Table 4.4 lists the seed-averaged relative errors $\bar{\epsilon}$ in $u$ and $v$ for the test points. The ensemble method offers better performance in predicting both fields at all of the test points, usually by wide margins. Table 4.6 lists the standard deviation of these errors; at all of the test points, the standard deviation is also smaller, showing that using the ensemble method leads to much greater stability and reliability in predictions. Table 4.5 lists the computational costs associated with CFD simulation and ROM inference. The given CFD wall time is for the portion of the simulation over the prediction time horizon; using the ROM for inference offers a speed-up of approximately 9.4x over CFD.

| Test Case Index | $\bar{\epsilon}$, $u$ (Ensemble) | $\bar{\epsilon}$, $u$ (Single) | $\bar{\epsilon}$, $v$ (Ensemble) | $\bar{\epsilon}$, $v$ (Single) |
|---|---|---|---|---|
| 1 | **0.02421** | 0.07119 | **0.02514** | 0.07204 |
| 2 | **0.01958** | 0.04199 | **0.02174** | 0.03870 |
| 3 | **0.02622** | 0.05282 | **0.02868** | 0.06296 |
| 4 | **0.03640** | 0.06575 | **0.03924** | 0.06742 |
| 5 | **0.02442** | 0.06589 | **0.01996** | 0.05131 |
| 6 | **0.04177** | 0.04939 | **0.03964** | 0.04733 |
| 7 | **0.02445** | 0.03771 | **0.02789** | 0.03915 |
| 8 | **0.04559** | 0.05304 | **0.04251** | 0.05194 |
| 9 | **0.03313** | 0.04678 | **0.03891** | 0.04900 |
| 10 | **0.02877** | 0.04291 | **0.02717** | 0.04079 |

Table 4.4: Average relative errors in $u$ and $v$ for the lid-driven cavity case

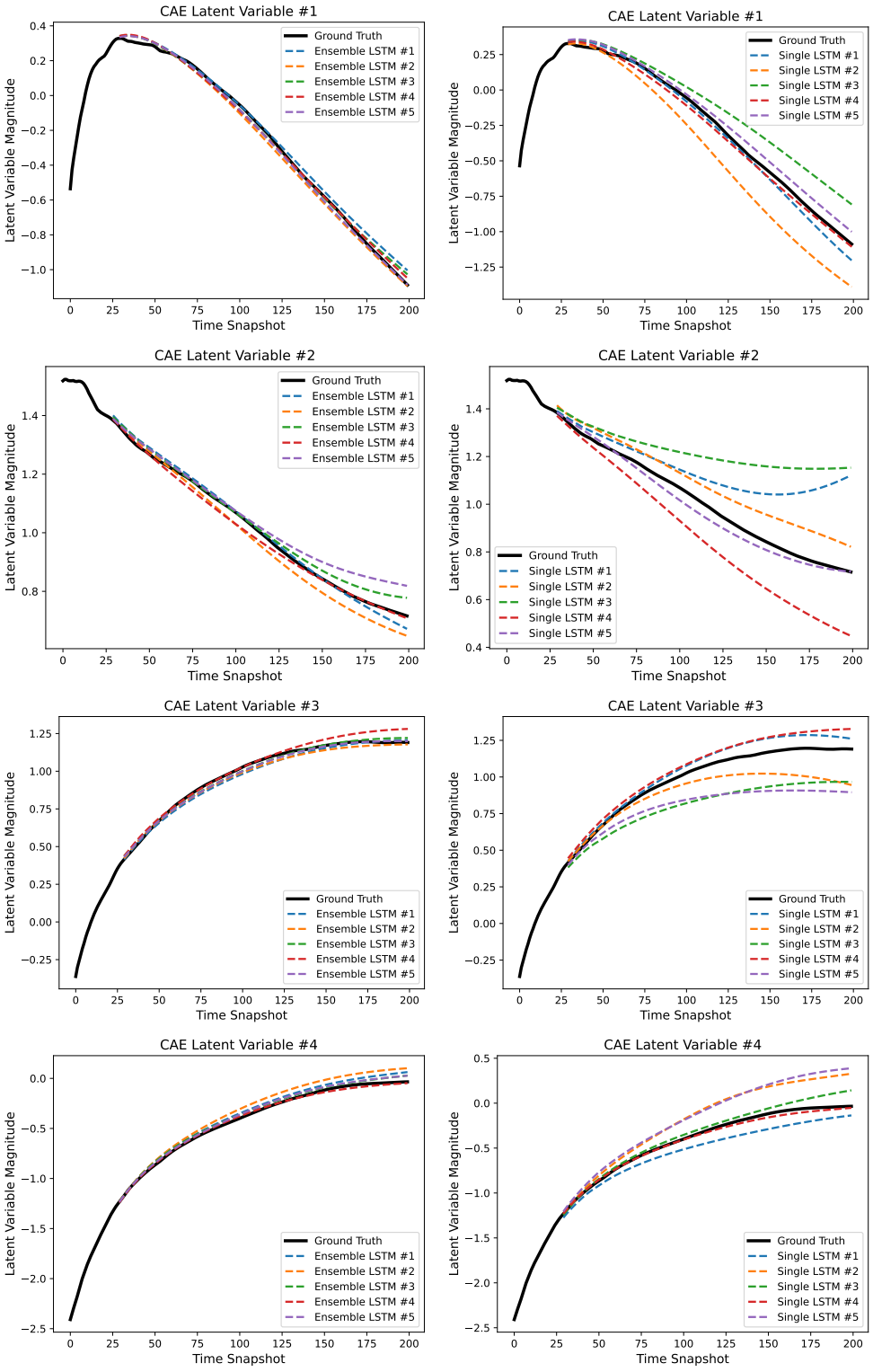



Figure 4.13: Comparison of latent variable trajectories for the lid-driven cavity case at $\boldsymbol{\mu}^* = [1.299, 1.689, -0.0367]$.

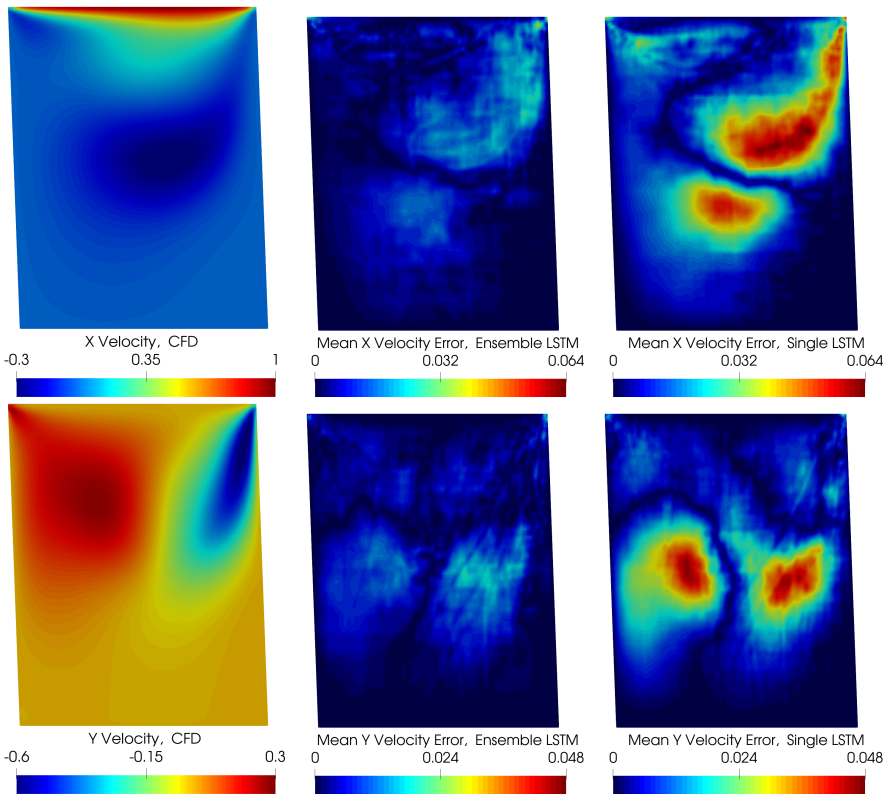



Figure 4.14: Time snapshot of $u$ (top) and $v$ (bottom) at T = 5 and errors averaged over the last 10% of the simulation.

| Task | Wall Time (s) |
|---|---|
| CFD (1 CPU) | 136 |
| ROM Inference | 14.5 |

Table 4.5: Computational costs associated with the lid-driven cavity problem.

The CAE architecture for the lid-driven cavity case is given in Table 4.7. Convolutional layers are usually followed max-pooling layers that have batch normalization [99] applied to them (referred to as pool-norm layers). Batch normalization normalizes the input of each layer over a mini-batch, reducing internal covariate shift, leading to more efficient gradient flow during backpropagation. The decoder consists of convolutional transpose layers that are used to progressively increase the number of pixels. The leaky ReLU activation function with $\alpha = 0.25$ is used for convolutional and fully connected layers.

| Test Case Index | $\sigma_s$, $u$ (Ensemble) | $\sigma_s$, $u$ (Single) | $\sigma_s$, $v$ (Ensemble) | $\sigma_s$, $v$ (Single) |
|---|---|---|---|---|
| 1 | **0.005225** | 0.02819 | **0.004470** | 0.03051 |
| 2 | **0.001637** | 0.009202 | **0.001766** | 0.01267 |
| 3 | **0.004603** | 0.01885 | **0.004040** | 0.02263 |
| 4 | **0.008210** | 0.02007 | **0.006915** | 0.01172 |
| 5 | **0.003980** | 0.01121 | **0.004222** | 0.007739 |
| 6 | **0.003244** | 0.01555 | **0.002547** | 0.01726 |
| 7 | **0.002077** | 0.007614 | **0.002466** | 0.006886 |
| 8 | **0.002393** | 0.02031 | **0.002030** | 0.02041 |
| 9 | **0.002515** | 0.007371 | **0.002550** | 0.01324 |
| 10 | **0.002467** | 0.01627 | **0.001759** | 0.01493 |

Table 4.6: Standard deviation in relative errors of $u$ and $v$ for the lid-driven cavity case.

| Layer | Filters | Kernel | Activation Function | Output Size |
|---|---|---|---|---|
| Input | | | | $128 \times 128 \times 2$ |
| Convolutional | 8 | $3 \times 3$ | Leaky ReLU | $128 \times 128 \times 8$ |
| Pool-Norm | | $2 \times 2$ | | $64 \times 64 \times 8$ |
| Convolutional | 16 | $3 \times 3$ | Leaky ReLU | $64 \times 64 \times 16$ |
| Pool-Norm | | $2 \times 2$ | | $32 \times 32 \times 16$ |
| Convolutional | 32 | $3 \times 3$ | Leaky ReLU | $32 \times 32 \times 32$ |
| Pool-Norm | | $2 \times 2$ | | $16 \times 16 \times 32$ |
| Convolutional | 64 | $3 \times 3$ | Leaky ReLU | $16 \times 16 \times 64$ |
| Pool-Norm | | $2 \times 2$ | | $8 \times 8 \times 64$ |
| Reshape | | | | 4096 |
| Fully Connected | | | Leaky ReLU | 128 |
| Batch Norm | | | | 128 |
| Fully Connected (Latent Space) | | | | 4 |
| Fully Connected | | | Leaky ReLU | 128 |
| Batch Norm | | | | 128 |
| Fully Connected | | | Leaky ReLU | 4096 |
| Batch Norm | | | | 4096 |
| Reshape | | | | $8 \times 8 \times 64$ |
| Convolutional Transpose | 32 | $3 \times 3$ | Leaky ReLU | $16 \times 16 \times 32$ |
| Batch Norm | | | | $16 \times 16 \times 32$ |
| Convolutional Transpose | 16 | $3 \times 3$ | Leaky ReLU | $32 \times 32 \times 16$ |
| Batch Norm | | | | $32 \times 32 \times 16$ |
| Convolutional Transpose | 8 | $3 \times 3$ | Leaky ReLU | $64 \times 64 \times 8$ |
| Batch Norm | | | | $64 \times 64 \times 8$ |
| Convolutional Transpose | 2 | $3 \times 3$ | Sigmoid | $128 \times 128 \times 2$ |

Table 4.7: Convolutional autoencoder architecture for the lid-driven cavity case.

#### 4.3.3.2 2D Cylinder

The next test case involves two-dimensional incompressible, unsteady, laminar flow over a cylinder. Eventually, the lid-driven cavity flow from the previous test case reaches steady-state and does not exhibit long-term transient behavior that is commonly found in fluid dynamics problems. Laminar flow over a cylinder is a well-studied problem in fluid dynamics, with both experimental and computational results present in the literature [100, 101]. Unsteady cylinder flow is characterized by the presence of vortices that separate from the surface and form in the wake. This distinctive pattern is know as the Von Kármán vortex street, where alternating vortices of a regular pattern are shed downstream of the cylinder. The unsteady Navier-Stokes equations are solved using XLB [102], a Lattice Boltzmann method [103] library utilizing the JAX framework [104] available for Python, which allows for effective scaling across multiple CPUs, GPUs, and distributed multi-GPU systems. The cited works can be referred to for an overview of the Lattice Boltzmann method and its implementation in XLB.

No-slip boundary conditions are applied to the cylinder's surface and top and bottom walls. A Poiseuille flow profile is used for the inlet velocity. Extrapolation outflow boundary conditions are used for the outlet to allow the fluid flow to exit the domain freely. The computational domain measures $1536 \times 512$ voxels that are uniformly spaced. The cylinder is centered at $x_c, y_c = [160, 256]$ (zero-based indexing is used). Simulation results are down-sampled onto a grid that measures $384 \times 128$ before being used for the ROM, resulting in $N = 49,152$, as the original domain's large size leads to a very large training cost as well as memory usage. Two design parameters are used, the diameter $d$ of the cylinder and the Reynolds number $Re$. The bounds of the design parameters are given as

$$\mu_1 = d \in [48, 68],$$

$$\mu_2 = Re \in [120, 240].$$

The diameter $d$ is set to an integer quantity by rounding to the nearest whole number. The wake structure behind the cylinder undergoes instabilities [105, 106] at a critical reynolds number of approximately $Re_c = 180$, where the vortices transition to becoming turbulent. As a result, the prescribed range given for $Re$ in the parameter space includes a variety of physical regimes. Additionally, both $Re$ and $d$ control the size and periodicity of the shed vortices, making this a difficult prediction problem for ROMs. The freestream velocity in the $x$-direction is set to $u_\infty = 0.001$. The simulation is run for $T = 750,000$ timesteps with data being saved every 1000 steps, leading to 750 snapshots for a single simulation. The initial condition for a simulation of a given diameter $d$ is the solution to steady flow at $Re = 20$.

50 sets of design parameters are generated and split into 45 training samples and 5 test points, shown in Figure 4.15. Table 4.8 also lists the test point design parameters. The ROM uses $m = 96$ bagged LSTMs and a window size of $w = 30$, which are chosen through a trial-and-error process to maximize accuracy while keeping the number of weak learners to a minimum. The CAE latent dimension is set to $k = 10$. Similar to the lid-driven cavity case, values below $k = 10$ resulted in higher reconstruction errors and increasing the latent dimension further offers no improvement. A bidirectional LSTM architecture [107] is used. Bidirectional recurrent neural networks process sequential data in both forward and backwards directions, allowing the model to learn both past and future context. The network consists of three hidden layers with 36 neurons each and a dropout rate of 0.15. The output layer again contains a sigmoid activation function, and the Adam optimizer with the same initial learning

rate $\eta = 5 \times 10^{-4}$ and weight decay of $\lambda = 1 \times 10^{-6}$ is used for both the LSTM and CAE. The CAE is trained for 200 epochs while an individual LSTM is trained for 250 epochs. At the test points, the full-order model is run for $T_i = 300,000$ timesteps (300 snapshots), or 40% of the total simulation time. This value is required to be high as the flow exhibits highly oscillatory behavior initially, leading to very noisy latent variables that cannot be used for model training. As a reult, latent variable sequences are generated starting at the 200th snapshot, and $\boldsymbol{A}_{\text{train}}$ does not contain time-series data of latent variables before this point.

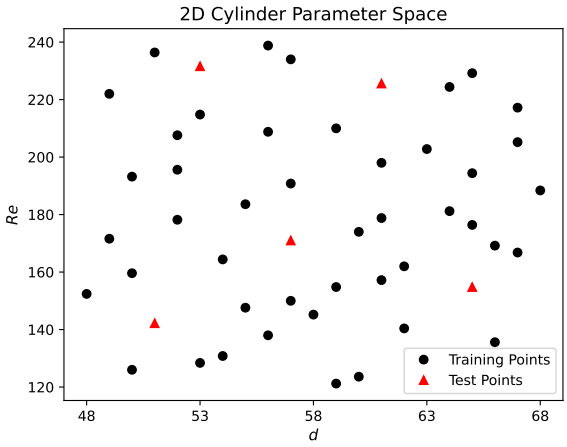



Figure 4.15: Training and test design parameters for the 2D cylinder case.

| Test Case Index | $d$ | $Re$ |
|---|---|---|
| 1 | 53 | 231.6 |
| 2 | 51 | 142.2 |
| 3 | 65 | 154.8 |
| 4 | 57 | 171.0 |
| 5 | 61 | 225.6 |

Table 4.8: Test case design parameters for the 2D cylinder case.

Figure 4.16 shows the latent variable trajectories for the first four latent variables at the test point $\boldsymbol{\mu}^* = [51, 142.2]$. For each latent variable, the predictions given by single LSTMs are initially similar, but eventually diverge in terms of both the amplitude and frequency of the latent variables, leading to large inaccuracies. Using

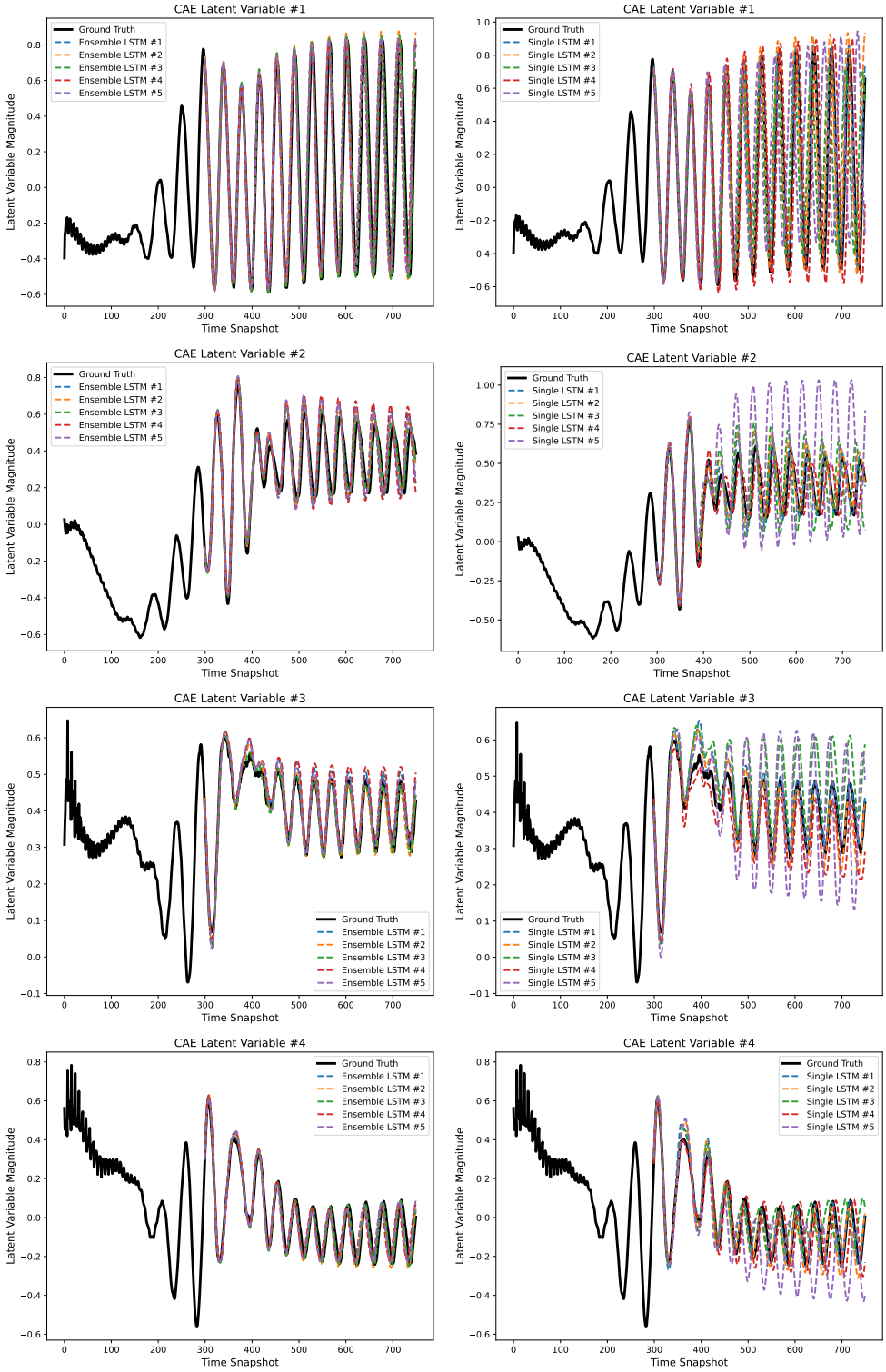



Figure 4.16: Comparison of latent variable trajectories for the 2D cylinder case at $\boldsymbol{\mu}^* = [51, 142.2]$.

LSTM ensembles greatly mitigates this effect, and the resultant latent variable trajectories follow the ground truth closely and do not differ greatly in amplitude nor frequency. The regular pattern of the Kármán vortex street is well-predicted given a small amount of initial latent variable history.

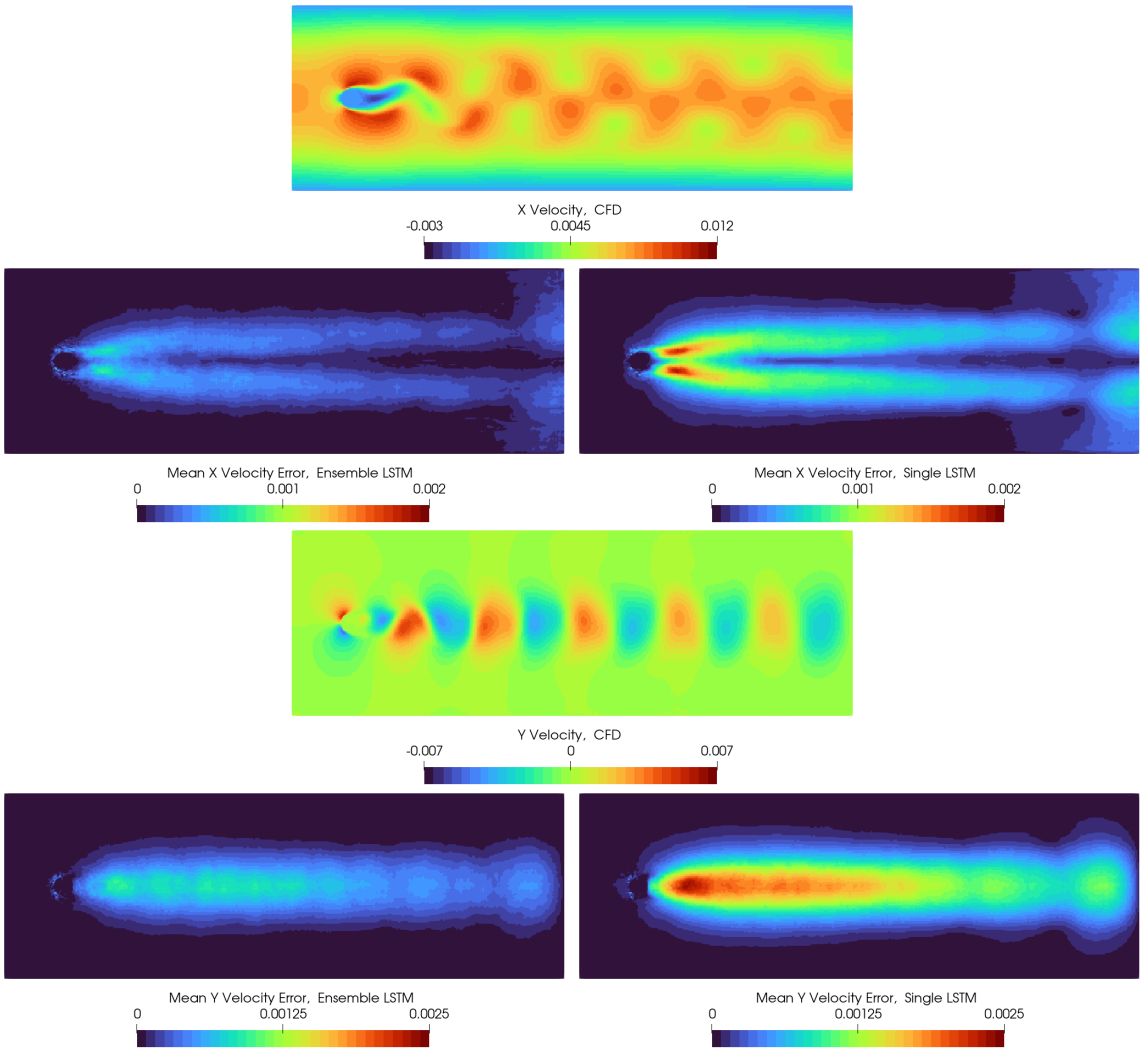



Figure 4.17: Time snapshots of $u$ and $v$ at T = 750,000 and errors averaged over the last 10% of the simulation at $\boldsymbol{\mu}^* = [51, 142.2]$.

Figure 4.17 shows snapshots of $u$ and $v$ at $T = 750,000$ as well as ROM errors averaged over the last 10% of the simulation for a single seed. The errors are significantly lower throughout the computational domain when using LSTM ensembles. These errors are greatest in the wake of the cylinder and immediately downstream

of it. Table 4.9 lists the seed-averaged relative errors in $u$ and $v$ at the test points. The ensemble ROM again offers better performance in predicting both fields at all of the test points. Table 4.10 lists the standard deviation of these errors, which are again lower at all of the test points. At the fourth test case index ($\boldsymbol{\mu}^* = [57, 171.0]$), the errors and standard deviations are similar. This test point lies in the middle of the parameter space and is well-surrounded by training samples; as a result, the advantage gained using an ensemble method may be marginal. Table 4.11 lists the computational costs associated with the problem. Using the ROM for inference offers a speed-up of approximately 27x over CFD (again, the given CFD cost is the portion of the simulation over the prediction time horizon).

| Test Case Index | $\bar{\epsilon}, u$ (Ensemble) | $\bar{\epsilon}, u$ (Single) | $\bar{\epsilon}, v$ (Ensemble) | $\bar{\epsilon}, v$ (Single) |
|---|---|---|---|---|
| 1 | **0.04324** | 0.07975 | **0.3379** | 0.6162 |
| 2 | **0.03463** | 0.05838 | **0.2971** | 0.4871 |
| 3 | **0.02380** | 0.05195 | **0.1399** | 0.3315 |
| 4 | **0.05446** | 0.06636 | **0.3995** | 0.4900 |
| 5 | **0.03655** | 0.06029 | **0.2269** | 0.4059 |

Table 4.9: Average relative errors in $u$ and $v$ for the 2D cylinder case.

| Test Case Index | $\sigma_s$, $u$ (Ensemble) | $\sigma_s$, $u$ (Single) | $\sigma_s$, $v$ (Ensemble) | $\sigma_s$, $v$ (Single) |
|---|---|---|---|---|
| 1 | **0.01321** | 0.04184 | **0.1138** | 0.3372 |
| 2 | **0.007607** | 0.03304 | **0.07005** | 0.2720 |
| 3 | **0.004610** | 0.01587 | **0.03493** | 0.1136 |
| 4 | **0.01521** | 0.02026 | **0.1186** | 0.1536 |
| 5 | **0.007528** | 0.03398 | **0.05904** | 0.2684 |

Table 4.10: Standard deviation in relative errors of $u$ and $v$ for the 2D cylinder case.

| Task | Wall Time (s) |
|---|---|
| CFD (2 GPUs) | 1305 |
| ROM Inference | 48.4 |

Table 4.11: Computational costs associated with the 2D cylinder case.

The CAE architecture for the 2D cylinder case is given in Table 4.12. It is similar to the one for the lid-driven cavity case and has a larger overall number of paramaters due to the increased input size. This also results in larger kernel sizes being used for convolutional layers close to the input and output layers to account for the broader dimensions.

| Layer | Filters | Kernel | Activation Function | Output Size |
|---|---|---|---|---|
| Input | | | | $384 \times 128 \times 2$ |
| Convolutional | 8 | $5 \times 5$ | Leaky ReLU | $192 \times 64 \times 8$ |
| Batch Norm | | | | $192 \times 64 \times 8$ |
| Convolutional | 16 | $5 \times 5$ | Leaky ReLU | $96 \times 32 \times 16$ |
| Batch Norm | | | | $96 \times 32 \times 16$ |
| Convolutional | 32 | $3 \times 3$ | Leaky ReLU | $96 \times 32 \times 32$ |
| Pool-Norm | | $2 \times 2$ | | $48 \times 16 \times 32$ |
| Convolutional | 64 | $3 \times 3$ | Leaky ReLU | $48 \times 16 \times 64$ |
| Pool-Norm | | $2 \times 2$ | | $24 \times 8 \times 64$ |
| Reshape | | | | 12288 |
| Fully Connected | | | Leaky ReLU | 128 |
| Batch Norm | | | | 128 |
| Fully Connected (Latent Space) | | | | 10 |
| Fully Connected | | | Leaky ReLU | 128 |
| Batch Norm | | | | 128 |
| Fully Connected | | | Leaky ReLU | 12288 |
| Batch Norm | | | | 12288 |
| Reshape | | | | $24 \times 8 \times 64$ |
| Convolutional Transpose | 32 | $3 \times 3$ | Leaky ReLU | $48 \times 16 \times 32$ |
| Batch Norm | | | | $48 \times 16 \times 32$ |
| Convolutional Transpose | 16 | $3 \times 3$ | Leaky ReLU | $96 \times 32 \times 16$ |
| Batch Norm | | | | $96 \times 32 \times 16$ |
| Convolutional Transpose | 8 | $5 \times 5$ | Leaky ReLU | $192 \times 64 \times 8$ |
| Batch Norm | | | | $192 \times 64 \times 8$ |
| Convolutional Transpose | 2 | $5 \times 5$ | Sigmoid | $384 \times 128 \times 2$ |

Table 4.12: Convolutional autoencoder architecture for the 2D cylinder case.

## 4.4 Summary

In this chapter, ROMs using convolutional autoencoders to provide nonlinear solution spaces are introduced for both steady and unsteady fluid dynamics problems. CAEs learn efficient low-dimensional representations of spatially distributed

data through an encoder and decoder connected by a latent space. A nonlinear relationship exists between the expansion coefficients and full-order states when using autoencoders, in contrast with the linear relationship when using POD. A ROM framework for steady problems utilizing CAEs and GPR for prediction of the expansion coefficients is shown to signficantly outperform POD when applied to a lid-driven cavity flow over a number of ROM dimensions. The projection errors provided by CAEs are significantly lower than those from POD for low ROM dimensions. Additionally, the projection error of CAEs does not decrease as the ROM dimension increases, showing that they can efficiently represent solution spaces with a low number of degrees freedom. This makes CAEs useful for unsteady ROMs, where there are typically a very large number of training snapshots. Unsteady ROMs require a time-series forecasting model for the expansion coefficients, for which LSTMs are a popular choice. A common problem encountered when making time-series predictions over long horizons at unseen data sets is error propagation. An unsteady ROM that combines CAEs with LSTM ensembles using bagging is introduced, and is shown to effectively diminish the effect of error propagation and provide accurate latent variable trajectories for a lid-driven cavity and 2D cylinder flow. A major limitation of the introduced ROMs is that they are not readily applicable to problems involving unstructured meshes due to the use of convolutional layers. While simulation data can be interpolated onto a structured mesh, this can result in information loss, especially in regions where high levels of resolution are required to accurately resolve the physics.

# CHAPTER V

# Field Inversion and Machine Learning

The previous chapters introduced ROMs using POD and CAE to provide efficient low-dimensional representations of high-dimensional solution spaces. While both methods are shown to offer good performance for different problems, they, and ROMs in general, make an assumption of topologically similar meshes for different designs. The number of cells and their ordering are required to be consistent across the design space. While this is possible to achieve using methods like FFD, it can remain infeasible for problems that involve substantial geometric changes. Additionally, CAEs cannot readily handle simulation data from unstructured meshes, greatly restricting the types of problems they can be applied to. The test cases used in Chapter IV all involved structured meshes with uniformly spaced grids.

The test cases involving steady, incompressible, turbulent flow in Chapter 2 also make the assumption that the Spalart-Allmaras turbulence model can accurately model the flow. While existing RANS turbulence models are very useful in many situations, they poorly model complex flow phenomena such as separation [108]. As a result, expensive modeling techniques such as direct numerical simulation (DNS) and large eddy simulation (LES) are required to sufficiently resolve the flow physics for some problems. A situation where LES or DNS is required and topologically similar meshes between designs cannot be achieved can lead to ROMs, and design

optimization in general, being infeasible. Attempts have been made to improve the predictive performance of turbulence models for various flow applications. A recent and popular advance is field inversion [18, 109], a method that uses reference high-fidelity or experimental data, such as field distributions or coefficients of lift/drag, to correct turbulence models. Field inversion uses gradient-based optimization to solve an inverse problem for a corrective scalar field present in each mesh cell that is introduced into an existing turbulence model by minimizing an objective function that measures the discrepancy between the reference data and RANS model outputs. Field inversion can incur a large computational cost as the number of design variables is large and the optimization process requires multiple primal solves of the full-order model.

Field inversion can be used to improve the accuracy of a turbulence model within a parameter space so that it better approximates techniques such as DNS and LES or available experimental data. Similar to ROMs, the data used from completed field inversion runs at chosen points within the parameter space can be used to construct a surrogate model to predict the corrective field for unseen designs. This paradigm is referred to as field inversion and machine learning [110] (FIML), and it constructs a machine learning model that uses local features to predict the corrective field. The machine learning model and flow solver are combined to update the corrective field. As it only relies on local cell-wise information to make predictions, FIML is agnostic to the topological similarity between meshes.

In this chapter, the use of field inversion is investigated for flows over airfoils using both wall-resolved and wall-modeled grids. The resultant corrective fields and the effects they have on flow features such as separation are detailed. Additionally, an FIML framework using neural networks that take the local states and their gradients as inputs is introduced. Feature importance is also explored for both cases, where it is shown that gradient information is significantly more important when using wall-

resolved grids.

## 5.1 Field Inversion

Field inversion involves introducing a multiplicative corrective scalar field $\boldsymbol{\beta}(\vec{\boldsymbol{r}}) \in \mathbb{R}^N$ present in each cell to an existing turbulence model, where $\vec{\boldsymbol{r}}$ represents the cell coordinates. For the Spalart-Allmaras turbulence model, the field multiplies the production term, resulting in

$$\begin{aligned} \int_V \nabla \cdot (\overrightarrow{V}\tilde{\nu})\,\mathrm{d}V - \frac{1}{\sigma}\int_V \nabla \cdot [(\nu+\tilde{\nu})\nabla\tilde{\nu}] + c_{b2}|\nabla\tilde{\nu}|^2\,\mathrm{d}V - \\ \boldsymbol{\beta} c_{b1}\int_V \tilde{S}\tilde{\nu}\,\mathrm{d}V + c_{w1}\int_V f_w\left(\frac{\tilde{\nu}}{d}\right)^2\mathrm{d}V = \frac{\partial\tilde{\nu}}{\partial t}. \end{aligned} \tag{5.1}$$

An objective function $J$ is formulated that represents the discrepancy between a high-fidelity output coefficient $C_H$ and the RANS model coefficient $C_R(\boldsymbol{\beta}, \boldsymbol{x})$,

$$J = \left(C_H - C_R(\beta, \boldsymbol{x})\right)^2 + \lambda \left\|\boldsymbol{\beta} - \boldsymbol{\beta}_1\right\|^2, \tag{5.2}$$

where $\lambda$ is a small regularization constant and $\boldsymbol{\beta}_1$ represents $\boldsymbol{\beta} = 1$ throughout the domain and gives the baseline turbulence model. The second term is included in the objective function to prevent large and unphysical values of the corrective field. Minimizing the objective function results in an altered turbulence model that better approximates high-fidelity data. While it is common to use field distributions as reference data, previous work has shown that using force coefficients results in similar flow fields for external aerodynamics applications [110]. Gradient-based optimization is used to minimize the objective function, which contains a large number of design variables. Using the chain rule, the gradient $\frac{dJ}{d\boldsymbol{\beta}}$ can be expressed as

$$\frac{dJ}{d\boldsymbol{\beta}} = \frac{\partial J}{\partial \boldsymbol{\beta}} + \frac{\partial J}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{\beta}}. \tag{5.3}$$

The partial derivatives $\frac{\partial J}{\partial \boldsymbol{\beta}}$ and $\frac{\partial J}{\partial \boldsymbol{x}}$ can be solved for explicitly and do not require re-computing the primal solution. However, computing the total derivative $\frac{d\boldsymbol{x}}{d\boldsymbol{\beta}}$ does require that the primal solution be re-computed for each entry in $\boldsymbol{\beta}$ when using methods such as finite differences, which is computationally prohibitive. To effectively calculate gradients with respect to a large number of design variables in CFD applications, the adjoint method [111] is often used. The adjoint method involves solving a set of adjoint equations along with the governing equations, which are derived from the governing equations and allow for efficient computation of gradients with respect to a large number of design variables. The governing equations can be expressed through the residuals $\boldsymbol{R}$, from which the gradients $\frac{dR}{d\boldsymbol{\beta}}$ are given as

$$\frac{d\boldsymbol{R}}{d\boldsymbol{\beta}} = \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\beta}} + \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{\beta}}. \tag{5.4}$$

The governing equations are satisfied when $\boldsymbol{R} = 0$, and the total derivative $\frac{d\boldsymbol{x}}{d\boldsymbol{\beta}}$ is found as

$$\frac{d\boldsymbol{x}}{d\boldsymbol{\beta}} = -\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\beta}}. \tag{5.5}$$

Substituting this into Equation 5.3 results in

$$\frac{dJ}{d\boldsymbol{\beta}} = \frac{\partial J}{\partial \boldsymbol{\beta}} - \frac{\partial J}{\partial \boldsymbol{x}} \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^{-1} \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\beta}} \tag{5.6}$$

where $\boldsymbol{\Phi}^T = -\frac{\partial J}{\partial \boldsymbol{x}} \left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^{-1}$ is the adjoint vector, which represents the effects of perturbing the residuals on the objective function, the expensive part of computing the gradients. Taking the transpose of the adjoint $\boldsymbol{\Phi}^T$ and moving everything to the left hand side gives the *adjoint equations*,

$$\left[\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{x}}\right]^T \boldsymbol{\Phi} + \left[\frac{\partial J}{\partial \boldsymbol{x}}\right]^T = 0. \tag{5.7}$$

The adjoint can be computed by solving Equation 5.7 and a final relation is given as

$$\frac{dJ}{d\boldsymbol{\beta}} = \frac{\partial J}{\partial \boldsymbol{\beta}} - \boldsymbol{\Phi}^T \frac{\partial \boldsymbol{R}}{\partial \boldsymbol{\beta}}. \tag{5.8}$$

In this work, DAFoam [37], an open-source adjoint derivative computation framework for OpenFOAM, is used to implement the adjoint method and compute all gradients. DAFoam uses a Jacobian-free approach to compute gradients, as detailed in a work by Kenway et al. [112] by using reverse-mode automatic differentiation. pyOptSparse [113], an object-oriented framework in Python for solving nonlinear optimization problems is used. IPOPT [114], a software package for large-scale optimization problems is available through pyOptSparse and is chosen to be used for minimizing the objective function. IPOPT employs an interior penalty method for constrained optimization problems by adding a logarithmic barrier term to the objective function. This ensures that the design variables remain within a feasible region. Although there are no hard constraints on the values of $\boldsymbol{\beta}$, setting a prescribed probable range can lead to faster convergence as the search space is reduced, allowing the optimizer to explore only physically consistent regions. If no Hessian information is provided, as is the case here, IPOPT uses a quasi-Newton method to compute the search direction using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) update algorithm.

### 5.1.1 Wall-resolved NACA 0012

A two-dimesional NACA 0012 airfoil with a chord length of $c = 1$ m is used to demonstrate the field inversion process using experimental data from a study by Gregory and O'Reilly [115]. The flow is turbulent with $Re = 6 \times 10^6$ and a freestream

velocity of $U_\infty = 51.5$ m/s and the angle of attack is set to $\alpha = 15$ degrees. The mesh, shown in Figure 5.1, contains $N = 229,376$ cells and is wall-resolved with an average $y^+$ of 0.147. The mesh is obtained from the NASA website at https://turbmodels.larc.nasa.gov/naca0012_grids.html. Wall functions are not used for the turbulence model.

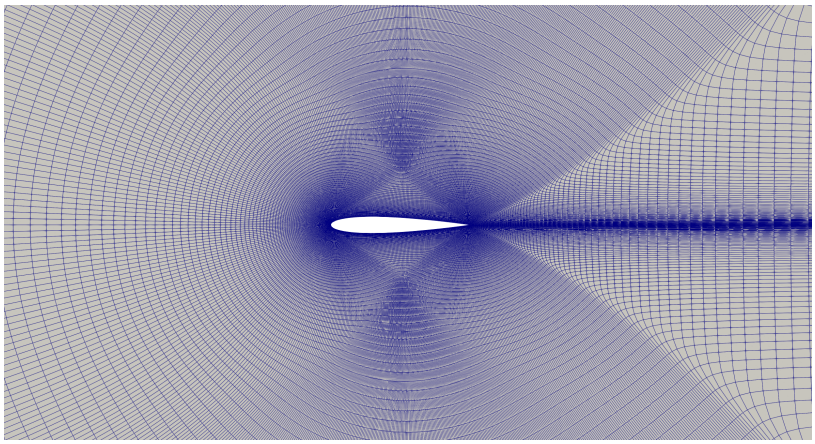


Figure 5.1: Mesh for the wall-resolved NACA 0012 case.

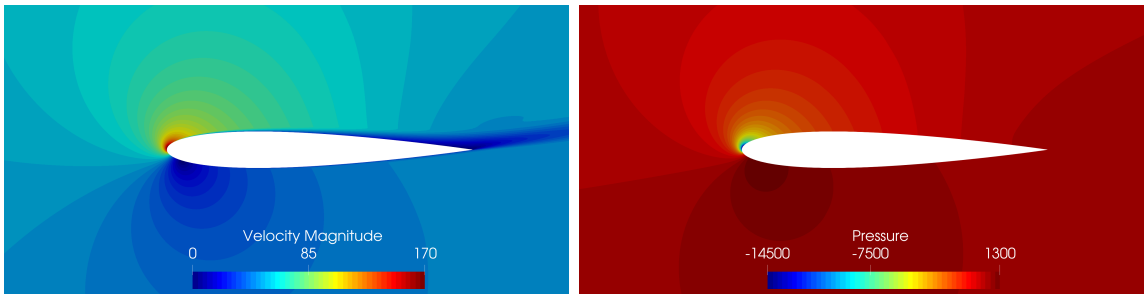



Figure 5.2: Contours of the baseline RANS velocity magnitude and pressure for the wall-resolved NACA 0012 case at $\alpha = 15$ degrees.

Contours of the velocity magnitude and pressure from the baseline RANS solution are shown in Figure 5.2. The lift coefficient given by the baseline RANS model is $C_L(\boldsymbol{\beta}_1) = 1.551$ and the experimental reference value is $C_L \approx 1.507$. Field inversion is performed by constraining the values of $\boldsymbol{\beta}$ to a range of (-10, 10) to constrain the optimizer to a likely and more physically consistent region. A regularization constant of $\lambda = 1 \times 10^{-6}$ is used, and the value of the first term in the objective function is

scaled to unity for the first iteration. An optimizer tolerance of $1 \times 10^{-6}$ is used. Figure 5.3 shows the convergence history of $C_L$; 8 total iterations are required for convergence, and each iteration required exactly one primal solve per line search. The reference experimental value of $C_L$ is matched almost exactly to 1.50699997.

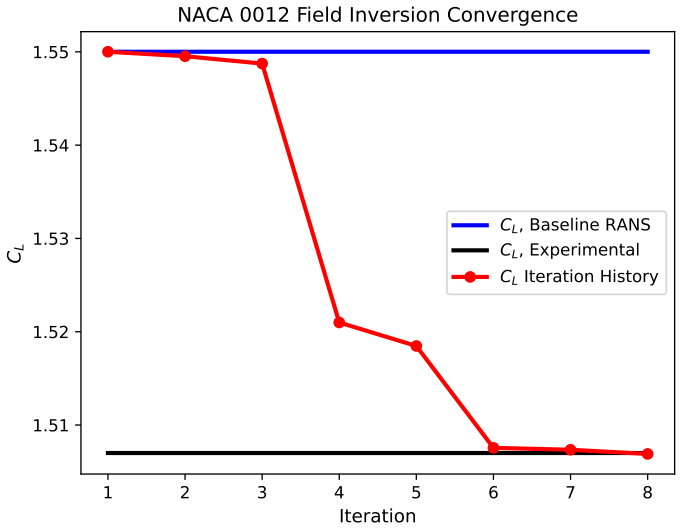



Figure 5.3: $C_L$ iterative history during field inversion for the wall-resolved NACA 0012 case.

Figures 5.4 and 5.5 show contours of $\boldsymbol{\beta}$ from the final iteration over the entire airfoil and close to the leading edge respectively. The maximum and minimum values of $\boldsymbol{\beta}$ are -9.4 and 9.6 respectively, which are close to the prescribed constraints, although they are outliers. There is a significant decrease in turbulent production just past the leading edge, followed by an increase over a larger section of the airfoil. Farther downstream, the turbulent production again decreases until slowly increasing to baseline levels at the trailing edge.

As shown in Figure 5.6, this pattern results in markedly different profiles of the turbulent eddy viscosity $\nu_t$ at different locations along the chord when plotted with the vertical distance $y$ from the airfoil surface. When $\boldsymbol{\beta} < 1$, the production of $\nu_t$ is reduced and is conversely increased when $\boldsymbol{\beta} > 1$. At the trailing edge ($x/c = 1$), where $\boldsymbol{\beta} = 1$, the profiles are similar. Increased levels of $\nu_t$ enhance turbulent mixing

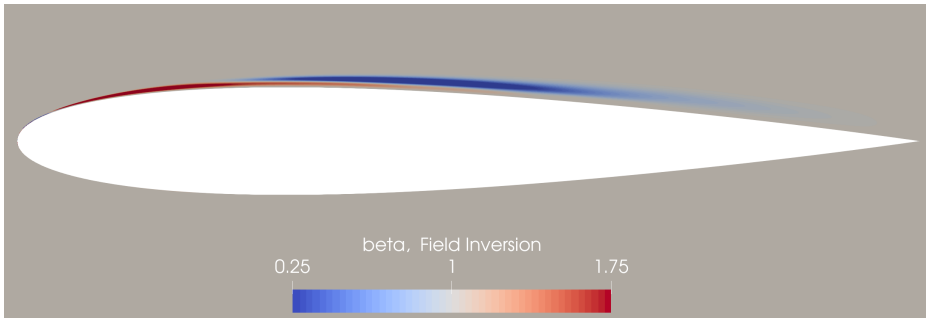



Figure 5.4: Contour of $\boldsymbol{\beta}$ obtained from field inversion for the wall-resolved NACA 0012 case.

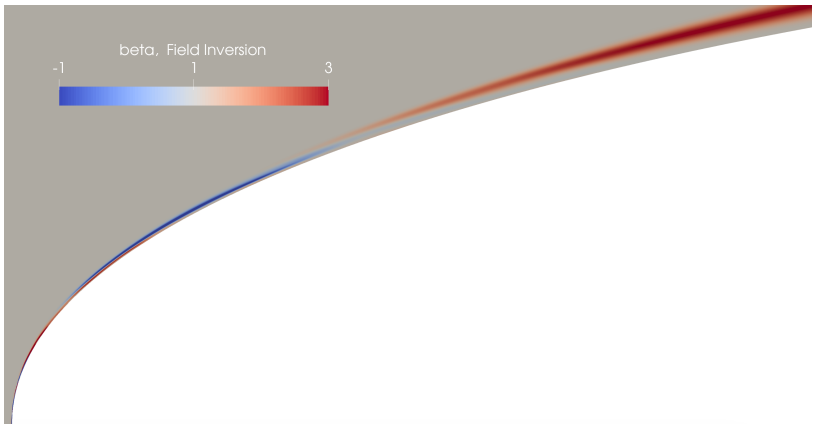



Figure 5.5: Leading edge view of $\boldsymbol{\beta}$ obtained from field inversion for the wall-resolved NACA 0012 case.

and momentum transport within the boundary layer, helping to prevent separation. Figure 5.7 shows plots on the suction side of the negative pressure coefficient $C_p$ from approximately $0.25 < x/c < 1.0$ and the skin friction coefficient $C_f$. Using field inversion results in a pressure field that matches experimental results significantly better when compared to the baseline turbulence model. The pressure is generally greater, leading to a lower lift coefficient. The skin friction coefficient distributions are also different; just past the leading edge, where the turbulent production is decreased, the wall shear stress $\tau$ drops rapidly and recovers as $\boldsymbol{\beta}$ increases moving downstream. The experimental results from Gregory and O'Reilly discuss the presence of a *laminar separation bubble* [116] past the leading edge followed by flow re-attachment for high angles of attack. Even though the flow solver is run fully turbulent, the decrease and

subsequent increase of turbulent production past the leading edge acts like a transition model [117] for the flow. While the flow does not fully separate as $\tau$ does not reach 0, the behavior of the altered turbulence model better represents the experimental results, where $C_f$ would exhibit a similar pattern of a sharp decline followed by an increase as the flow re-attaches. Field inversion achieves this by modifying the levels of $\nu_t$ through $\boldsymbol{\beta}$ at different regions of the airfoil. This also results in flow separation occuring earlier at the trailing edge.

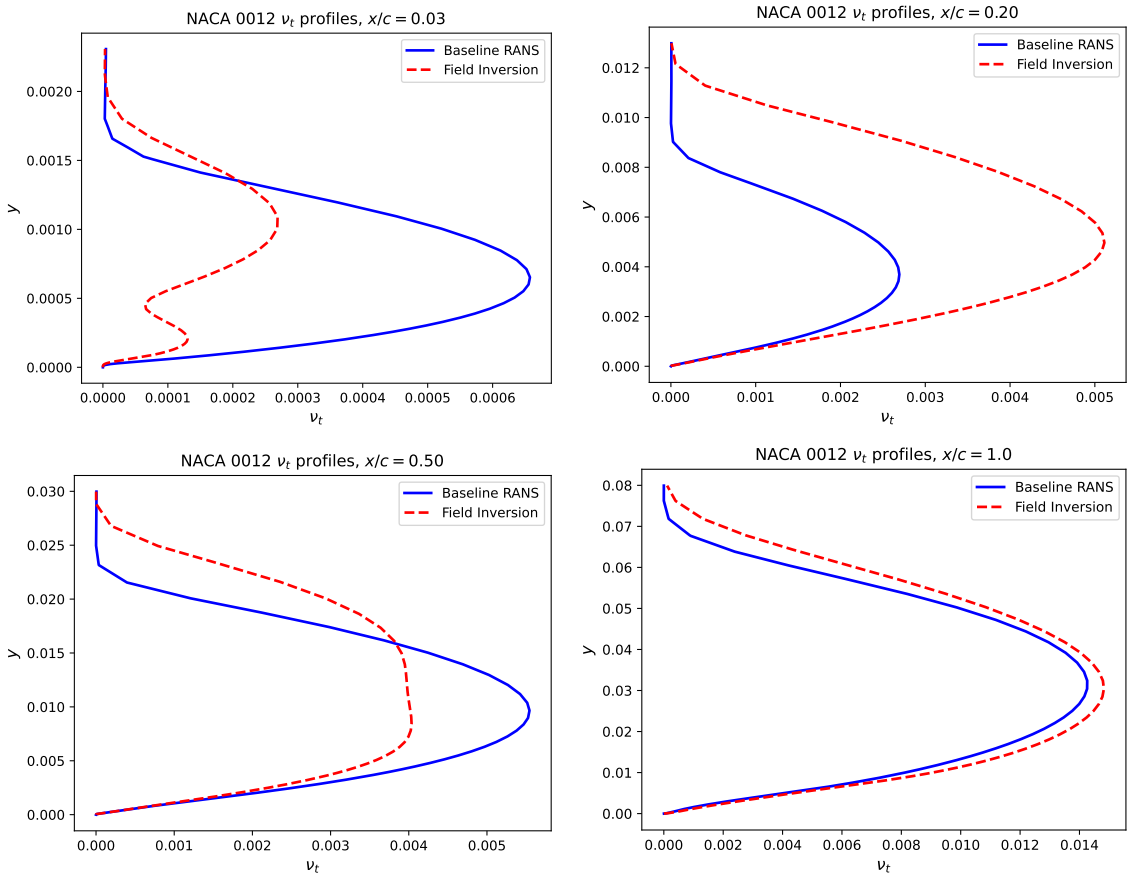



Figure 5.6: Profiles of the vertical distance $y$ against $\nu_t$ at different chord locations for the wall-resolved NACA 0012 case.

### 5.1.2 Wall-modeled NACA 0012

A NACA 0012 case using the same computational setup and experimental results as in the previous section is used. The mesh, shown in Figure 5.8, contains $N =$

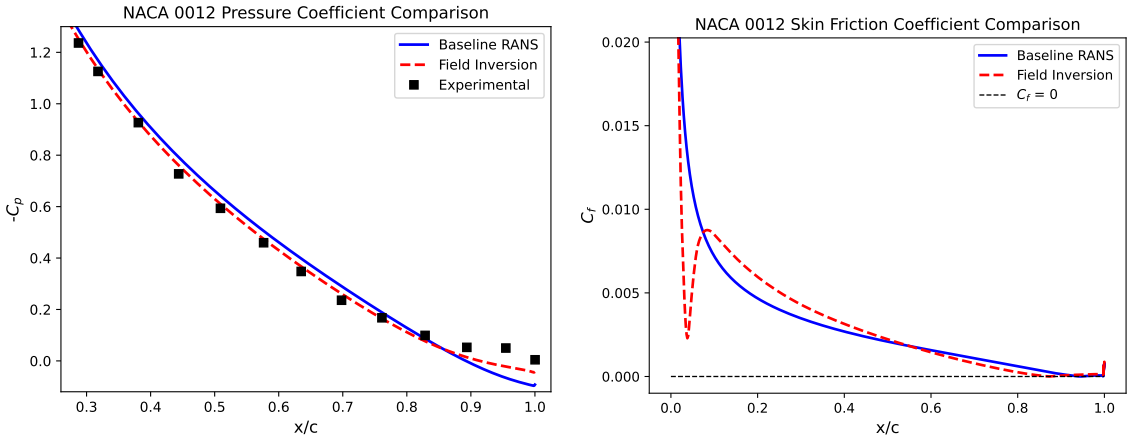



Figure 5.7: Distributions of $-C_p$ and $C_f$ against $x/c$ for the wall-resolved NACA 0012 case.

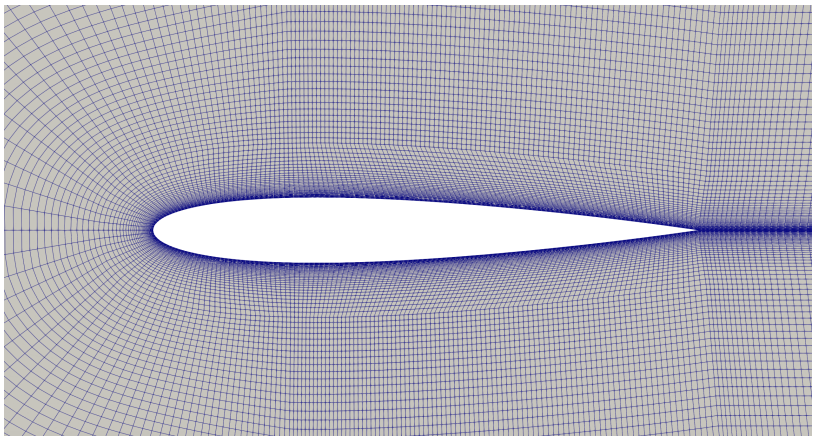


Figure 5.8: Mesh for the wall-modeled NACA 0012 case.

84,669 cells and is not wall-resolved. The average $y^+$ is 10.6 and the Spalding wall function [118] is used to model $\nu_t$. The lift coefficient given by the baseline RANS model is $C_L(\boldsymbol{\beta}_1) = 1.535$. Figure 5.9 shows the history of $C_L$ during field inversion; the reference coefficient is again matched almost exactly to 1.5069995, and 10 total primal solves are required for convergence.

Contours of the resultant corrective field are shown in Figures 5.10 and 5.11. There is a very small increase in turbulent production past the leading edge and a relatively small decrease downstream all the way until the trailing edge. Profiles of the vertical distance against $\nu_t$ are shown in Figure 5.12 at the same locations along the chord

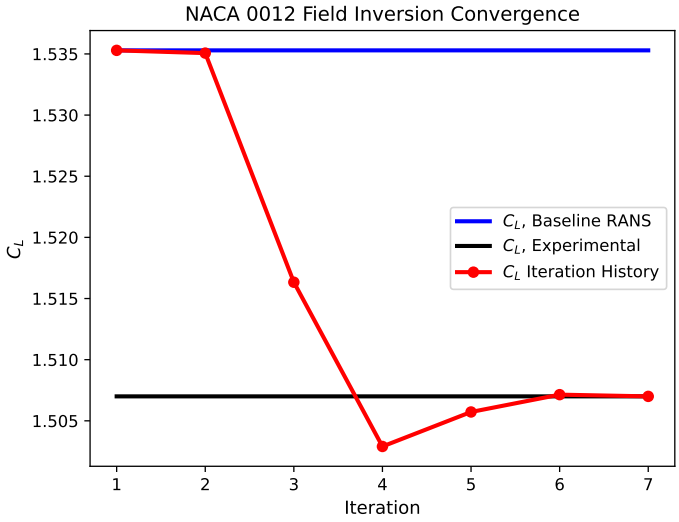



Figure 5.9: $C_L$ iterative history during field inversion for the wall-modeled NACA 0012 case.

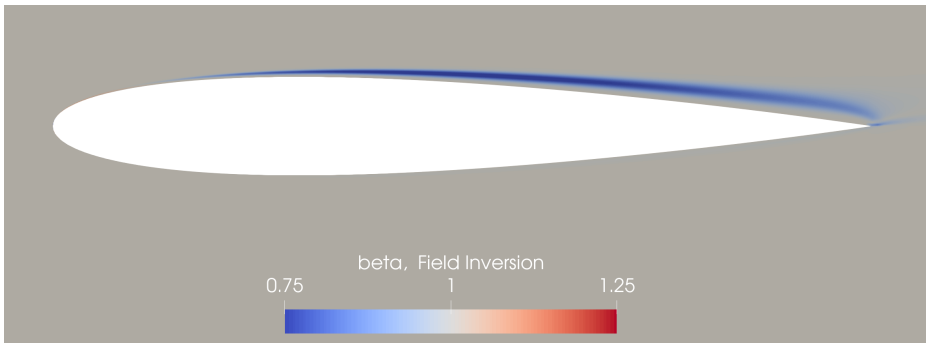



Figure 5.10: Contour of $\boldsymbol{\beta}$ obtained from field inversion for the wall-modeled NACA 0012 case.

from the previous section. While there is a noticeable difference in the profile at $x/c = 0.5$, the rest are very similar, in contrast to the differences observed for the wall-resolved case. Plots of the pressure and skin friction coefficients are shown in Figure 5.13, where it is shown that the distribution of $C_f$ does not change significantly past the leading edge. The pressure is also generally increased as shown in the plot of $C_p$, but the results do not match the experimental data as well as they did for the wall-resolved case.

The wall shear stress is slightly suppressed downstream, leading to earlier separation closer to the trailing edge. While both the wall-resolved and wall-modeled

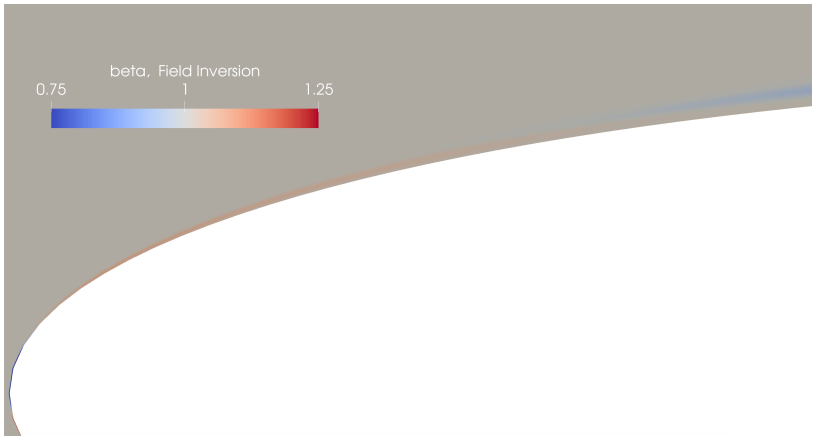



Figure 5.11: Leading edge view of $\boldsymbol{\beta}$ obtained from field inversion for the wall-modeled NACA 0012 case.

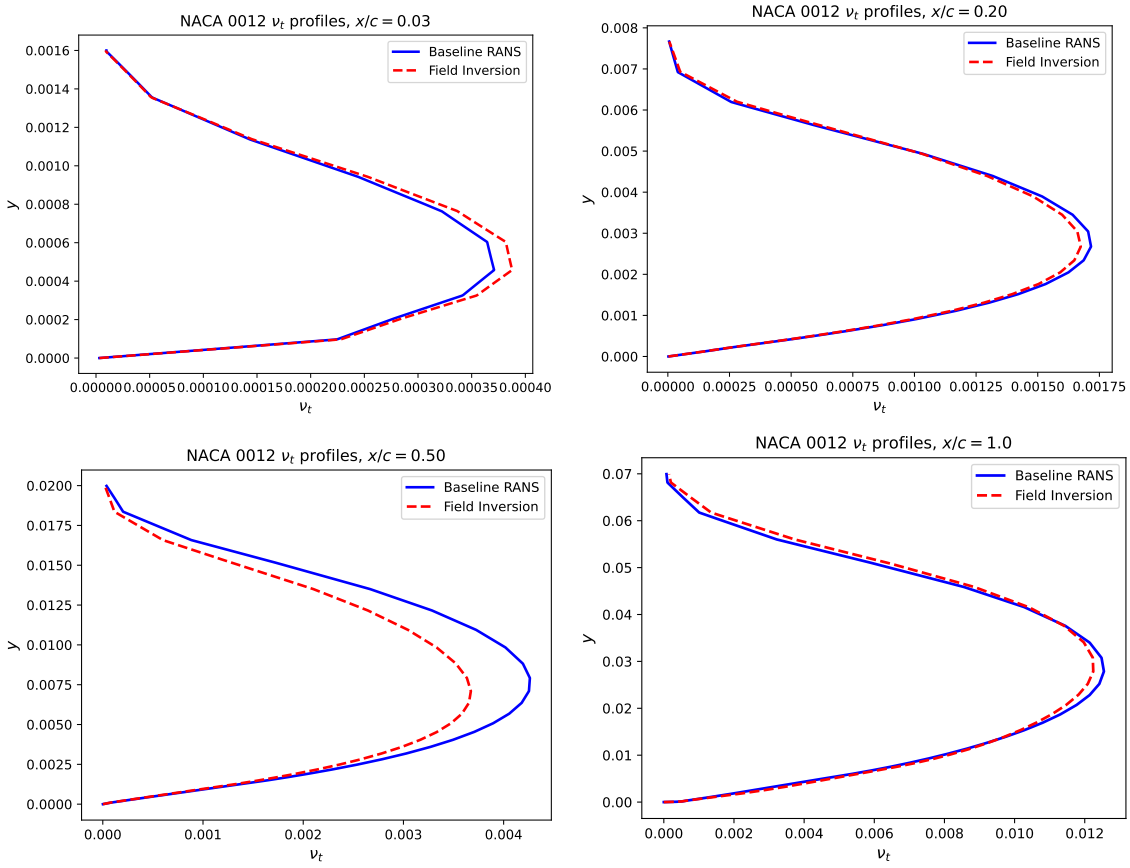



Figure 5.12: Profiles of the vertical distance $y$ against $\nu_t$ at different chord locations for the wall-modeled NACA 0012 case.

cases reach the reference lift coefficient, the resultant corrective fields and changes to the flow physics differ. Both corrective fields cause earlier separation downstream,

resulting in decreased lift. This occurs earlier when using the wall-resolved grid, at $x/c \approx 0.87$ compared to $x/c \approx 0.92$ when using the wall-modeled grid and better matches experimental results at $x/c \approx 0.8$. The corrections made when using the wall-resolved grid also better match experimental results past the leading edge, although this has a small overall impact on the lift coefficient. By providing a finer resolution of the near-wall region, the wall-resolved grid can better capture the velocity and pressure gradients, which often change rapidly and are essential for accurately computing the wall shear stress and separation. Profiles of the velocity gradient magnitudes $||\nabla u||$ and $||\nabla v||$ for both grids are shown in Figure 5.14 at $x/c \approx 0.03$, the location where $C_f$ reaches a minimum past the leading edge when using field inversion. The cell-center values for the first 40 cells are shown for the wall-resolved grid, while only the first three cells are shown for the wall-modeled grid. The fineness of the wall-resolved grid is much higher, which allows for accurately capturing the development of the gradients within the boundary layer. The wall-modeled grid's first cell is centered significantly farther from the surface, resulting in poorer resolution of the flow physics. While the Spalding wall function does compute the velocity profile in the near-wall region using an empirical logarithmic correlation, this may not accurately capture all flow features, including intricate and rapidly changing flow structures involving transition such as separation bubbles.

## 5.2 Machine Learning

Obtaining reference high-fidelity or experimental data at multiple points within a parameter space can come at a very high computational cost or be impossible due to a lack of available data. Using the data from a set of computed field inversion runs, a machine learning model can be built and combined with the CFD solver to correct the turbulence model as the simulation is run. The combined framework, field inversion and machine learning (FIML), uses local cell-wise features to predict the

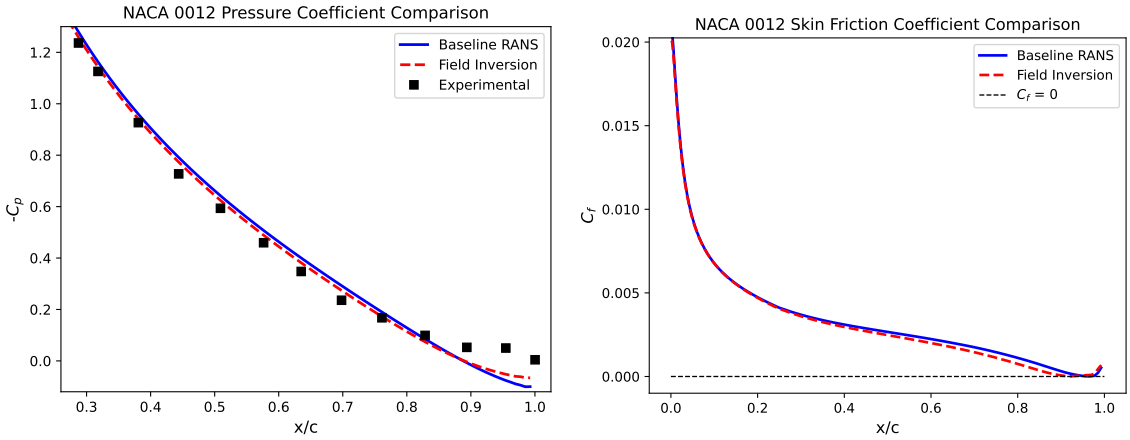



Figure 5.13: Distributions of $-C_p$ and $C_f$ against $x/c$ for the wall-modeled NACA 0012 case.

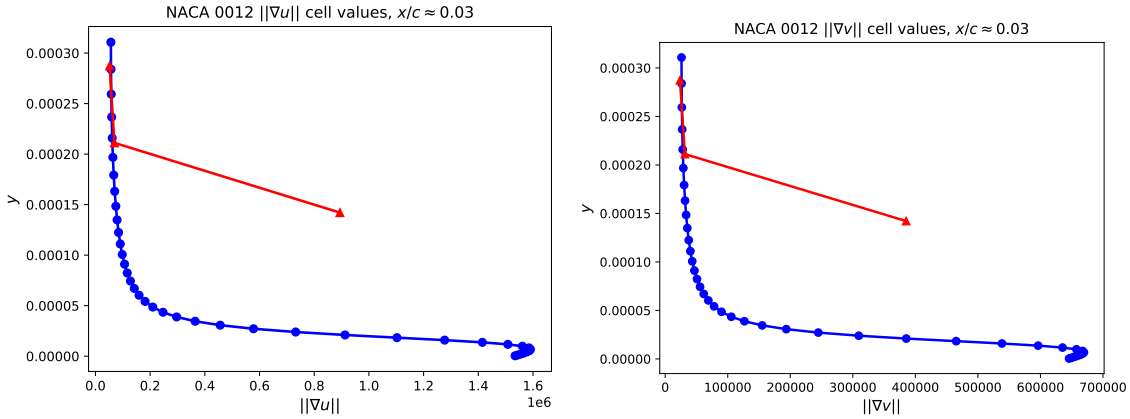



Figure 5.14: Profiles of the velocity component gradient magnitudes for the wall-resolved and wall-modeled grids at $x/c \approx 0.03$.

corrective field. Neural networks are usually used for the machine learning model, and the input features tend to include non-dimensional flow quantities such as the ratio of eddy to kinematic viscosity or vorticity to strain-rate magnitude. [110, 119]. Non-dimensional features are typically used so that the machine learning model can be applied to different flow problems (e.g. flow over an airfoil, a backward facing step, flow over a cylinder) as they tend to be universal in nature, but this approach has had little success in predicting useful corrective fields, even for cases within the training dataset [120]. As FIML lacks generalizability across different flow problems, its usefuleness in developing a general-use data-driven turbulence model has not been

realized.

Using the local state variables $\boldsymbol{x}$, their gradients $\nabla\boldsymbol{x}$, and the cell distance from the wall $d$ as features has been tested for case-specific parameter spaces in external aerodynamics with success in accuaretly predicting corrective fields [121]. However, information on the importance of the flow features has not been explored for this approach, and in general when comparing wall-resolved and wall-modeled grids. In this section, neural networks with the inputs consisting of $(\boldsymbol{x}, \nabla\boldsymbol{x}, d)$ where $\boldsymbol{x} = (u, v, p, \tilde{\nu})$ are used to predict $\boldsymbol{\beta}$ locally. $\tilde{\nu}$ is also chosen as a state variable as it is directly present in the model transport equation of the turbulence model and directly related to $\boldsymbol{\beta}$. There are two approaches to combining the machine learning model with the flow solver: an iterative method where $\boldsymbol{\beta}$ is predicted and updated at every iteration, or a successive method where the corrective field is updated at the end of complete primal solves until the objective function stops changing. While the first method is more common, more accurate results have been observed when using the latter for a single primal solve update in a study using OpenFOAM by Ho and West [119]. The authors found that the iterative method was sensitive to the initial conditions and required the use of a relaxation factor to update $\boldsymbol{\beta}$. As a result, successive method is used in this work although it does incur a larger computational cost.

A general method to determine feature importance for machine learning models is permutation feature importance. First, the model is trained on the training data using all of the features and a metric of performance such as the mean squared error is evaluated. Next, each individual feature has its values randomly shuffled within the dataset while leaving the rest unchanged to create a new dataset where one of the features is permuted. The model is then re-evaluated on this new dataset to obtain the metric of performance. After repeating this procedure for each feature, the features can be ranked by how much they impact the baseline performance metric.

### 5.2.1 NACA 0012 Angle of Attack

For both the wall-resolved and wall-modeled grids from the previous section, field inversion is performed at angles of attack $\alpha \approx [13, 14, 16]$ degrees using experimental lift coefficients and the machine learning model is used with the flow solver to make a prediction at $\alpha = 15$ degrees. The neural network contains 13 neurons in the input layer, two hidden layers with 72 neurons, and an output layer with a single neuron. A dropout rate of 0.1 is used between each of the layers to prevent overfitting. Both the inputs and outputs are scaled using min-max scaling into a range of [0,1]. The mean squared error loss function is used. ReLU activation functions are used for the hidden layers and the sigmoid activation function is used for the output. The Adam optimizer with an initial learning rate of $\eta = 1 \times 10^{-4}$ is used and the network is trained for 500 epochs with a mini-batch size of 1024.

#### 5.2.1.1 Wall-resolved grid

| $\alpha$ | $C_L$, Baseline RANS | $C_L$, Experimental | $C_L$, Field Inversion |
|---|---|---|---|
| 13.1 | 1.387 | 1.367 | 1.367 |
| 14.1 | 1.474 | 1.447 | 1.447 |
| 16.1 | 1.632 | 1.561 | 1.565 |

Table 5.1: Lift coefficients for the training data using the wall-resolved NACA 0012 grid.

The lift coefficients at the training points are listed in Table 5.1. As the angle of attack increases and the flow exhibits greater separation, the discrepancy between the baseline RANS and experimental coefficients increases. Field inversion matches the experimental coefficients almost exactly at $\alpha = [13.1, 14.1]$. At $\alpha = 16.1$, the flow field obtained from field inversion does not match the experimental lift coefficient exactly due to reaching a local minimum, but gets very close and offers a large improvement over the baseline RANS result. Contours of $\boldsymbol{\beta}$ from both field inversion and machine

learning are shown in Figures 5.15 and 5.16. The machine learning model required 14 primal solves to reach a lift coefficient of $C_L = 1.512$, close to the experimental value of 1.507, after which the lift coefficient and flow field stopped changing.

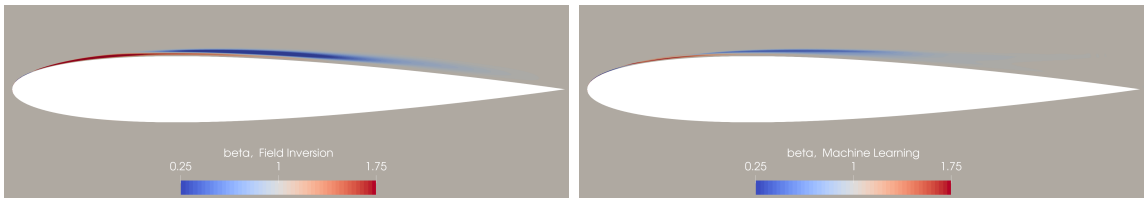



Figure 5.15: Comparison of $\boldsymbol{\beta}$ contours obtained from field inversion (left) and machine learning (right) for the wall-resolved NACA 0012.

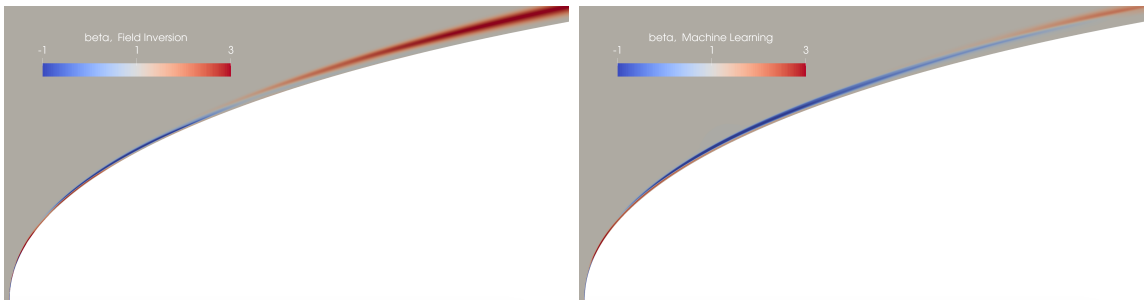



Figure 5.16: Leading edge comparisons of $\boldsymbol{\beta}$ contours obtained from field inversion (left) and machine learning (right) for the wall-resolved NACA 0012.

There is good agreement in the overall pattern between both corrective fields, where there is a decrease in turbulent production just past the leading edge, followed by an increase and then another decrease farther downstream. $\boldsymbol{\beta}$ obtained from machine learning does show a more sustained decrease in turbulent production past the leading edge. The effect of this can be seen in the plot of $C_f$ in Figure 5.17. The sudden decrease in the wall shear stress occurs farther downstream compared to field inversion and $C_f$ goes very slightly below 0, indicating that the flow does fully separate and then re-attach. Farther downstream, there is good agreement between both field inversion and machine learning, which is also shown to be true for $C_p$, where both match the experimental results well.

Figure 5.18 shows a feature importance plot for the neural network. The three features related to the pressure ($\frac{\partial p}{\partial x}, p, \frac{\partial p}{\partial y}$) are the most important features. Most of

the features are significantly important for the neural network with the exception of the gradients of $u$ and $\frac{\partial v}{\partial y}$. Since $\boldsymbol{\beta}$ only deviates from 1 close to and within the boundary layer, it is expected that quantities that show large variations throughout this region will have greater predictive power. The velocity gradients typically show large variations only very close to the surface, although $\boldsymbol{\beta}$ can deviate from 1 farther away. On the other hand, gradients of $p$ and $\tilde{\nu}$ can vary farther away from the surface in regions where $\boldsymbol{\beta}$ changes. The mesh wall distance $d$ provides topological information along with the other features, making it important for distinguishing different areas of the flow.

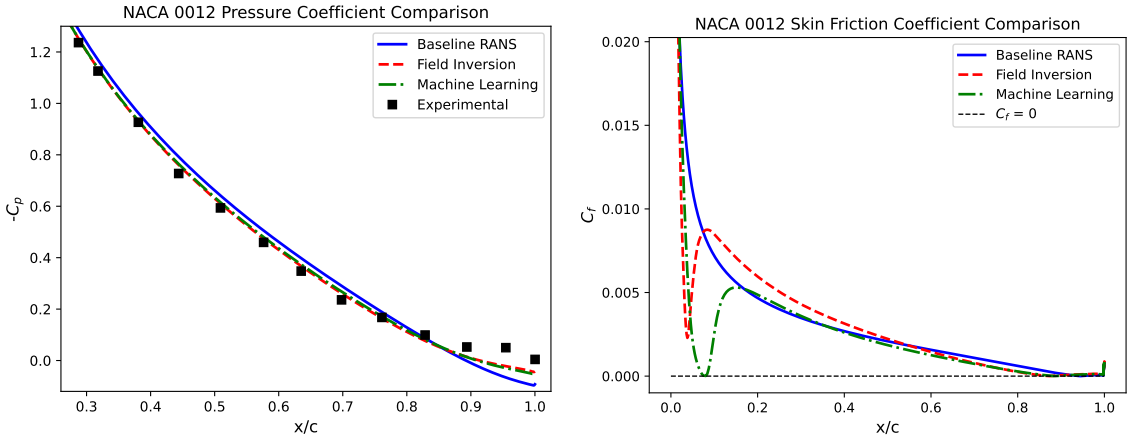



Figure 5.17: Distributions of $-C_p$ and $C_f$ from field inversion and machine learning against $x/c$ for the wall-resolved NACA 0012.

#### 5.2.1.2 Wall-modeled grid

| $\alpha$ | $C_L$, Baseline RANS | $C_L$, Experimental | $C_L$, Field Inversion |
|---|---|---|---|
| 13.1 | 1.370 | 1.367 | 1.367 |
| 14.1 | 1.458 | 1.447 | 1.447 |
| 16.1 | 1.620 | 1.561 | 1.561 |

Table 5.2: Lift coefficients for the training data using the wall-modeled NACA 0012 grid.

The lift coefficients at the training points are listed in Table 5.2. Compared to the

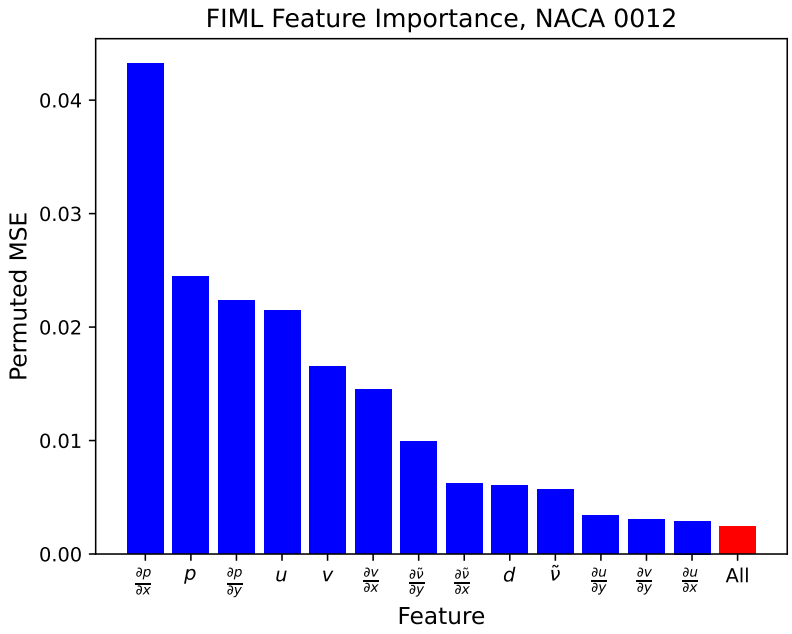



Figure 5.18: FIML feature importance for the wall-resolved NACA 0012 case.

wall-resolved case, the baseline RANS lift coefficients are closer to the experimental values, especially at $\alpha = 13.1$ where a very small discrepancy is present. The difference between the coefficients rises with the angle of attack again. Field inversion matches the experimental lift coefficients almost exactly at all of the training points. Contours of $\boldsymbol{\beta}$ from field inversion and machine learning are shown in Figures 5.19 and 5.20. The machine learning model required 3 total primal solves for convergence and resulted in a flow field with $C_L = 1.519$, higher than the value from the previous section. Again, there is a good match between the overall patterns of the corrective fields. The distributions of $C_p$ and $C_f$ are shown in Figure 5.21. The distributions from machine learning tend to be between the baseline RANS and field inversion ones, although slightly closer to field inversion.

The neural network feature importance plot is shown in Figure 5.22. The pressure and velocity components are the most important features, followed by those related to $\tilde{\nu}$. Gradients with respect to other states and $d$ follow. When using a wall-modeled grid, the gradients are not captured well and do not exhibit smooth changes, limiting

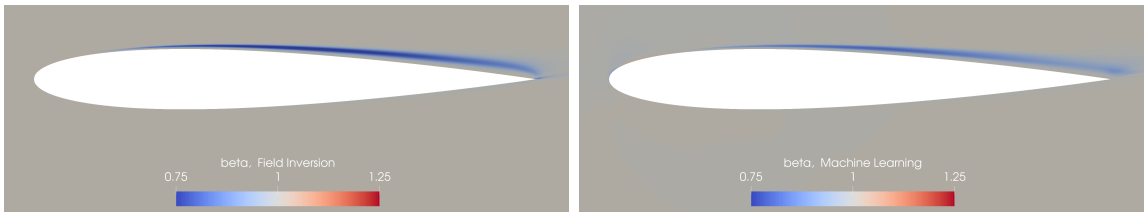



Figure 5.19: Comparison of $\boldsymbol{\beta}$ contours obtained from field inversion (left) and machine learning (right) for the wall-modeled NACA 0012.

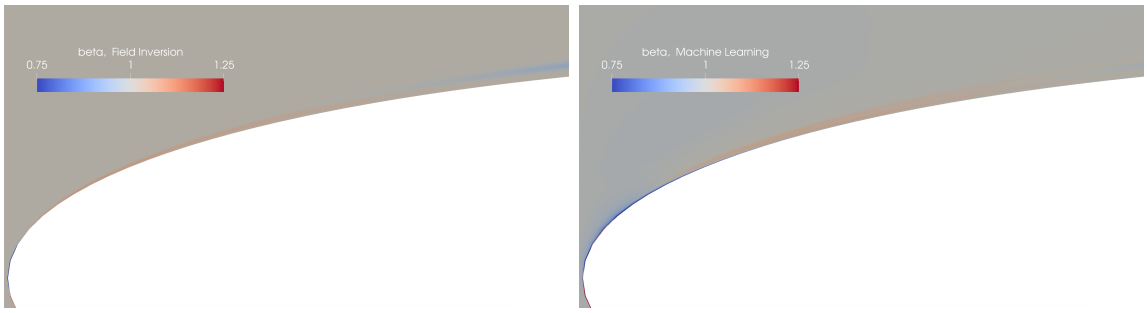



Figure 5.20: Leading edge comparisons of $\boldsymbol{\beta}$ contours obtained from field inversion (left) and machine learning (right) for the wall-modeled NACA 0012.

their usefuleness in a machine learning model. A reason the gradients of $\tilde{\nu}$ might be more important is because it can show large variations relatively far from the surface, especially close to the trailing edge in addition to being directly present in the turbulence model. Overall, the performance of FIML for this case is worse than the wall-resolved one, where there was a larger discrepancy in $C_L$ between baseline RANS and field inversion and yet a more accurate value obtained from machine learning.

### 5.2.2 Multiple NACA Airfoils

For this test case, six different NACA airfoils are used. The maximum camber, maximum camber location, and maximum thickness are all varied to achieve different geometries. The geometric changes are relatively large and as a result the meshes are not topologically similar, with the number of total cells varying between them although they are close to $N \approx 85,000$. The Reynolds number is set to $Re = 6 \times 10^6$ with a freestream velocity of $U_\infty = 51.5$ m/s and the angle of attack is set to $\alpha = 12$

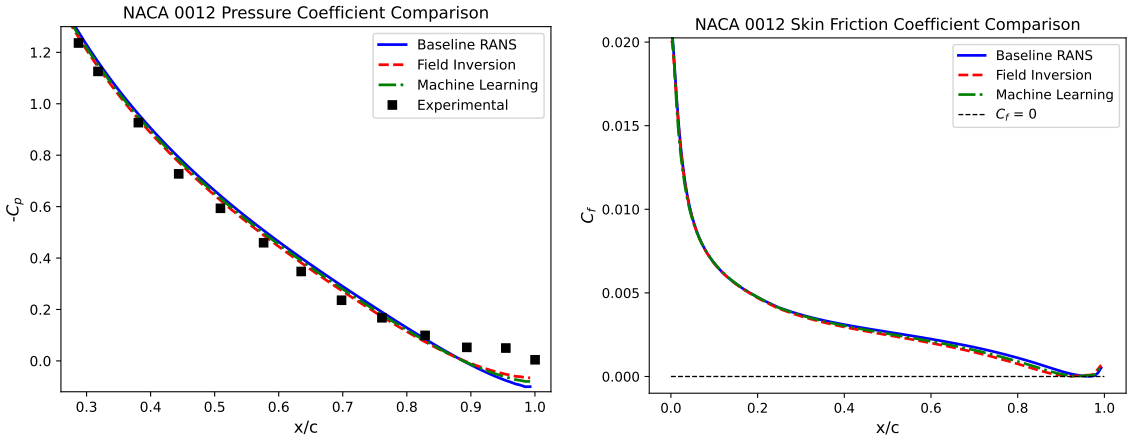



Figure 5.21: Distributions of $-C_p$ and $C_f$ from field inversion and machine learning against $x/c$ for the wall-modeled NACA 0012.

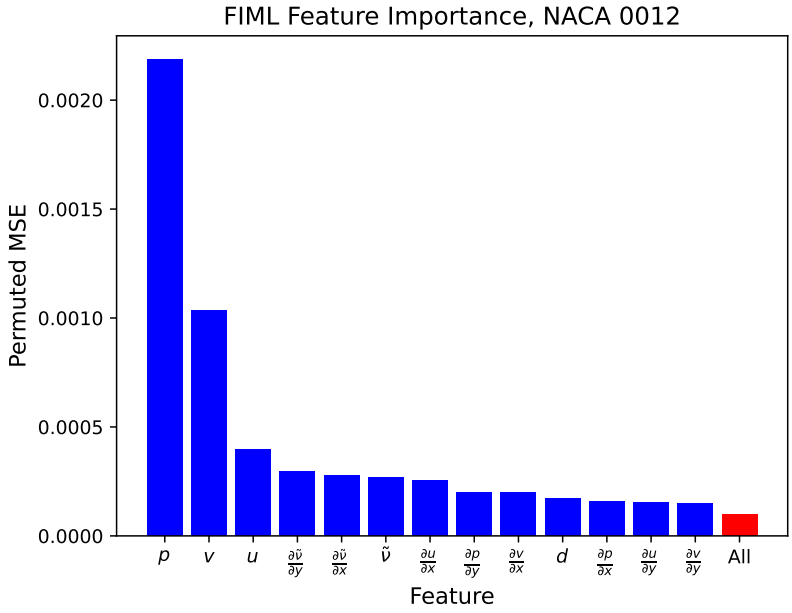



Figure 5.22: FIML feature importance for the wall-modeled NACA 0012 case.

degrees for each airfoil. The neural network used for FIML uses only $(\boldsymbol{x}, d)$ as inputs, as these are later shown to be the five most important features. The neural network again consists of two hidden layers with 72 neurons each. The hidden layers use the tanh activation function and the output layer uses the sigmoid activation function as min-max scaling is used. A dropout rate of 0.1 is used between each of the layers. The Adam optimizer with an initial learning rate of $\eta = 1 \times 10^{-4}$ is used and the

network is trained for 400 epochs with a mini-batch size of 1024.

Due to a lack of consistent experimental data for all the airfoils, mfoil [122], a MATLAB implementation of the popular airfoil flow solver XFOIL [123], is used to obtain reference lift coefficients. XFOIL uses a panel method to discretize airfoils and solves the potential flow equations to predict pressure distributions and uses empirical boundary layer models to predict the flow. The code is highly efficient and robust, making it useful for research purposes. The lift coefficients for the training data are listed in Table 5.3. For each case, the reference lift coefficient is higher and the discrepancy between baseline RANS grows smaller as the maximum camber is increased. Although field inversion does not match the mfoil lift coefficients exactly at most of the points, it gets close and offers a very large improvement over baseline RANS.

| Airfoil | $C_L$, Baseline RANS | $C_L$, mfoil | $C_L$, Field Inversion |
|---|---|---|---|
| 0012 | 1.263 | 1.319 | 1.316 |
| 0015 | 1.266 | 1.325 | 1.324 |
| 1212 | 1.363 | 1.408 | 1.412 |
| 1215 | 1.362 | 1.415 | 1.427 |
| 2412 | 1.472 | 1.510 | 1.510 |
| 2415 | 1.471 | 1.493 | 1.493 |

Table 5.3: Training data lift coefficients for multiple NACA airfoils.

A NACA 2314 airfoil is used as the test case. Contours of the velocity magnitude and pressure from baseline RANS are shown in Figure 5.23. The mfoil reference value of $C_L$ is 1.499 and the baseline RANS value is 1.466. Field inversion and machine learning result in values of 1.499 and 1.502 respectively. Field inversion requires 5 primal solves while machine learning requires 10. Contours of the corrective field from both are shown in Figure 5.24. As the lift coefficient is required to increase, the turbulent production is mainly increased rather than decreased. This delays or prevents separation, leading to lower pressure on the suction side of the airfoil

and greater lift. There is good agreement in $\boldsymbol{\beta}$ between field inversion and machine learning, although the corrective field from the machine learning model does exhibit some decrease in the turbulent production past the leading edge. Plots of $C_p$ and $C_f$ are shown in Figure 5.25. There is excellent overall agreement between field inversion and machine learning over the plotted range of the pressure coefficient, although the mfoil results better match the baseline RANS distribution closer to the trailing edge. In addition to the RANS flow being wall-modeled, mfoil makes some simplified model assumptions that lead to uncertainty in the computed distribution. There is a slight decrease in $C_f$ past the leading edge when using machine learning, likely due to the mentioned decrease in turbulent production. Farther downstream, both field inversion and machine learning are in good agreement, with higher values of the wall shear stress preventing separation as $C_f$ does not reach 0.

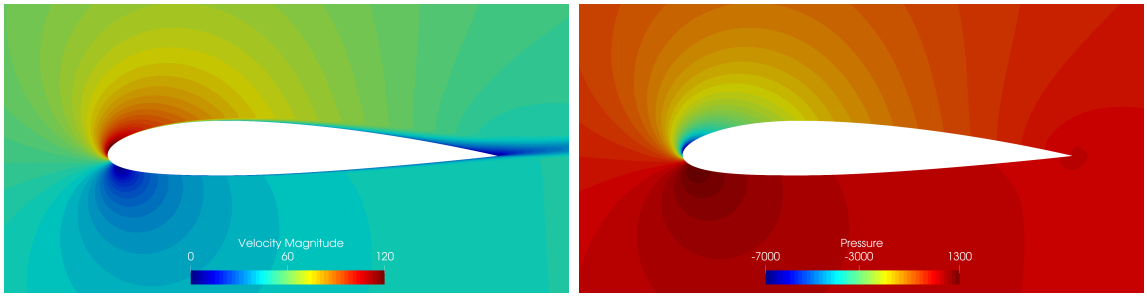



Figure 5.23: Contours of the baseline RANS velocity magnitude and pressure for the wall-modeled NACA 2314 case at $\alpha = 12$ degrees.

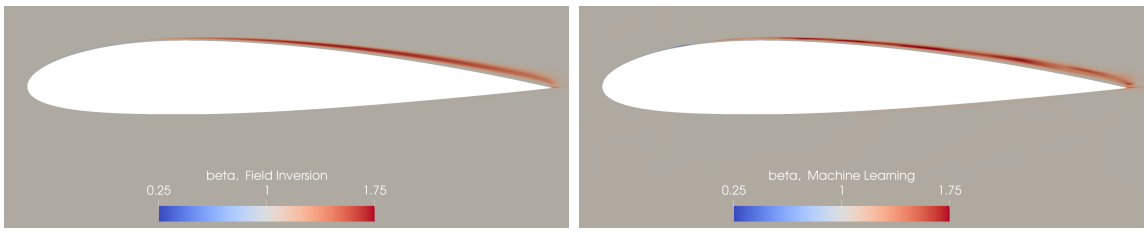



Figure 5.24: Comparison of $\boldsymbol{\beta}$ contours obtained from field inversion (left) and machine learning (right) for the NACA 2314.

A plot of the feature importance of a neural network using $(\boldsymbol{x}, \nabla\boldsymbol{x}, d)$ as inputs is shown in Figure 5.26. The most important features are shown to be the flow

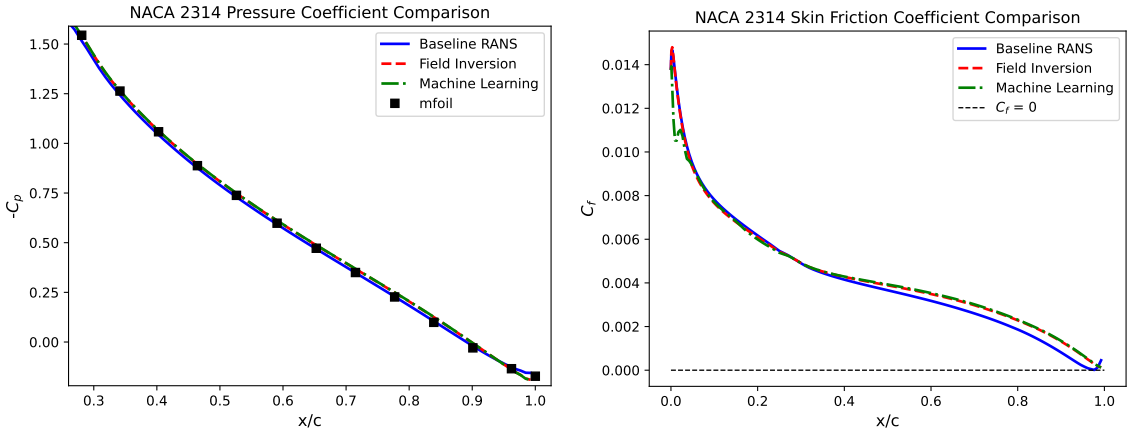



Figure 5.25: Distributions of $-C_p$ and $C_f$ from field inversion and machine learning against $x/c$ for the NACA 2314.

variables and the mesh wall distance. While the partial derivative $\frac{\partial \tilde{\nu}}{\partial y}$ is also shown to be important, randomly shuffling the rest of the gradients does not significantly degrade the predictive performance of the model. As a result, only the flow variables and mesh wall distance are chosen to be used as inputs for the machine learning model, where they alone were shown to train a model able to offer good predictive performance of the corrective field. Altering the geometry between training points can further enhance this effect due to the presence of different flow structures. This case further shows the limitations of including gradient information in the machine learning model when using wall-modeled grids.

## 5.3 Summary

In this chapter, a framework for field inversion and machine learning using local state information in the form of the flow variables, the mesh wall distance, and optionally the gradients as inputs to a neural network is introduced. When applied to two different airfoil flow problems involving changes to the boundary conditions and airfoil geometries, the proposed framework is shown to offer good predictive performance of the corrective field introduced into the Spalart-Allmaras turbulence model

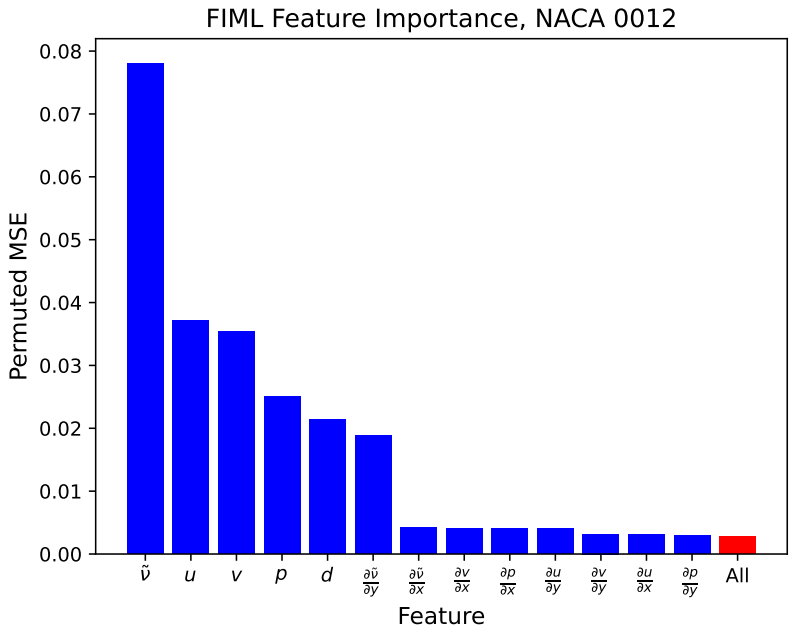



Figure 5.26: FIML feature importance when using multiple wall-modeled NACA airfoils.

as well as the reference lift coefficients used for field inversion. When using wall-resolved grids, the gradients of the flow variables are generally shown to be important to the overall predictive performance of the neural network. For wall-modeled grids, where the gradients cannot be accurately captured, this information is shown to be significantly less important. Using the flow variables and the mesh wall distance alone is sufficient to make accurate predictions of the corrective field for a case using multiple wall-modeled NACA airfoils with topologically dissimilar meshes. The introduced framework presents an alternative to ROMs for data-driven modeling of high-fidelity simulations where the reduction in computational cost comes from solving a cheaper RANS model. This also guarantees that results are physically consistent, as the governing equations are being solved directly.

# CHAPTER VI

# Conclusions and Future Work

The work presented in this disseration is motivated by the need to address common challenges concerning the utility of reduced-order models. Like all data-driven models, the performance of ROMs is determined by the quality and diversity of the training data. As ROMs are usually trained using a small amount of data, this makes it vital to choose training points judiciously. While statistical methods such as LHS are useful for partioning the parameter space, they do not account for the variation of the physics within it. Using Isomap, an algorithm for nonlinear dimensionality reduction, two algorithms for data selection are introduced. The first, applied to non-intrusive ROMs to select local POD bases, showed that Isomap is highly effective at separating data by physical regime, allowing for more accurate predictions when using significantly less data. These results are used to develop a computationally efficient adaptive sampling algorithm for both non-intrusive and projection-based ROMs using a manifold filling technique, which is shown to offer similar or better performance given a smaller computational budget for generating the training data.

While ROMs using autoencoders have been shown to outperform POD-based ROMs, their generalizability over entire parameter spaces has not been firmly established. Unsteady ROMs also suffer from error propagation issues when making temporal predictions of latent variables over long time horizons which can lead to

large inaccuracies. A non-intrusive CAE-GPR combining convolutional autoencoders with Gaussian process regression for predicting latent variables is introduced and shown to significantly outperform POD over a number of ROM dimensions when applied to a highly nonlinear lid-driven cavity problem. CAEs are much more efficient than POD at compressing data into a low-dimensional representation and can learn complex nonlinear relationships. This also makes them useful for unsteady ROMs involving multiple designs, as the number of training snapshots becomes prohibitively high when using POD. To remedy the issue of error propagation, LSTM ensembles are used for autoregressive time-series forecasting of latent variables using bagging. This is shown to effectively mitigate error propagation, allowing for stable and accurate predictions of latent variables over long time horizons.

While ROMs are useful, they do require that the design space can be represented by topologically similar meshes, a major restriction for problems involving large geometric changes. Instead of using a low-dimesnional surrogate model, data-driven RANS modeling can be used as a cheaper alternative to high-fidelity models such as LES or DNS. Field inversion, a method for altering turbulence models by solving an inverse problem for a corrective field by using numerical optimization, is an increasingly popular approach for improving the predictive capabilities of turbulence models. Using a machine learning model trained on local cell-wise features, a data-driven RANS model can be developed for use within parameter spaces. An FIML framework using neural networks trained on local state variables, their gradients, and the mesh wall distance is introduced for both wall-resolved and wall-modeled grids. While gradient information is shown to be important for wall-resolved grids, their usefuleness is limited for wall-modeled grids, where accurate corrective fields can be obtained without them. As several methods are used in this dissertation, it is important to understand their appropriate use cases. The pros and cons of each are given as follows:

### POD-Based ROMs

**Pros:**

- The modes from POD are interpretable, allowing for an understanding of dominant flow features.
- POD is easy to implement, generally computationally efficient, and deterministic.
- POD can be applied to different geometries and mesh structures.

**Cons:**

- POD suffers from performance issues when applied to highly nonlinear problems.
- Problems requiring large amounts of data, such as unsteady ROMs, can render POD infeasible.

### Deep Learning Based ROMs

**Pros:**

- Neural networks can learn complex functional relationships, making them suitable for highly nonlinear problems.
- Autoencoders are much more efficient at compressing data than POD, allowing for using a very low number of expansion coefficients even when the number of training data are high.

**Cons:**

- Deep learning requires a relatively large amount of training data.
- Convolutional autoencoders are not readily usable for unstructured meshes.
- Autoencoders are black-box models that are generally not interpretable.

**Field Inversion and Machine Learning**

**Pros:**

- FIML can be applied to topologically dissimilar meshes.
- Reference high-fidelity data can consist of distributions or output coefficients and come from experiments, which might be easy to obtain.
- FIML guarantees that results are physically consistent and obey the governing equations.

**Cons:**

- The evaluation stage of FIML is more computationally intensive, still requiring a full-order model to be solved.
- RANS models make many generalizations and can't be applied to some problems including compressible or highly unsteady flow.

The appropriate choice of data-driven surrogate model is dependent on the flow problem, desired properties of the approximate solutions, and design optimization workflow. For example, POD-based non-intrusive ROMs can be useful for obtaining approximate solutions of general problems very quickly to efficiently explore design spaces, while FIML can offer physically consistent results at a set of queried points, although at a higher cost. The algorithms introduced in this dissertation are designed with practical use cases in mind and aim for high levels of accuracy, generalizability, and stability. Future work will focus on the shortcomings and promise of the presented methods, including:

- Introduce interpretability to CAE-based ROMs while maintaining high levels of reconstruction accuracy.

- Develop methods that allow CAEs to be applied to unstructured meshes while retaining fine-scale flow information.
- Investigate the application of FIML to three-dimensional flows and optimal neural network architectures.
- Explore the utility of using FIML as a transition model within parameter spaces exhibiting laminar-turbulent behavior.

# BIBLIOGRAPHY

# BIBLIOGRAPHY


[1] Forrester T Johnson, Edward N Tinoco, and N Jong Yu. Thirty years of development and application of CFD at Boeing Commercial Airplanes, Seattle. *Computers & Fluids*, 34(10):1115–1151, 2005.

[2] Erich Strohmaier, Hans W Meuer, Jack Dongarra, and Horst D Simon. The TOP500 list and progress in high-performance computing. *Computer*, 48(11):42–49, 2015.

[3] Jeffrey P Slotnick, Abdollah Khodadoust, Juan Alonso, David Darmofal, William Gropp, Elizabeth Lurie, and Dimitri J Mavriplis. CFD vision 2030 study: a path to revolutionary computational aerosciences. Technical report, 2014.

[4] Gregory M Laskowski, James Kopriva, Vittorio Michelassi, Sriram Shankaran, Umesh Paliath, Rathakrishnan Bhaskaran, Qiqi Wang, Chaitanya Talnikar, Zhi J Wang, and Feilin Jia. Future directions of high fidelity CFD for aerothermal turbomachinery analysis and design. In *46th AIAA fluid dynamics conference*, page 3322, 2016.

[5] David J Lucia, Philip S Beran, and Walter A Silva. Reduced-order modeling: new approaches for computational physics. *Progress in aerospace sciences*, 40(1-2):51–117, 2004.

[6] Joaquim RRA Martins and Andrew B Lambe. Multidisciplinary design optimization: a survey of architectures. *AIAA journal*, 51(9):2049–2075, 2013.

[7] James Reuther, Antony Jameson, James Farmer, Luigi Martinelli, and David Saunders. Aerodynamic shape optimization of complex aircraft configurations via an adjoint formulation. In *34th aerospace sciences meeting and exhibit*, page 94, 1996.

[8] G Berkooz, PJ Holmes, and John Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25:539–575, 11 2003.

[9] Philip Holmes, John L. Lumley, Gal Berkooz, Jonathan C. Mattingly, and Ralf W. Wittenberg. Low-dimensional models of coherent structures in turbulence. *Physics Reports*, 287:337–384, 1997.

[10] Dunhui Xiao et al. *Non-intrusive reduced order models and their applications*. PhD thesis, Imperial College London, 2016.

[11] Kevin Carlberg, Charbel Bou-Mosleh, and Charbel Farhat. Efficient non-linear model reduction via a least-squares Petrov–Galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2):155–181, 2011.

[12] Sebastian Grimberg, Charbel Farhat, and Noah Youkilis. On the stability of projection-based model order reduction for convection-dominated laminar and turbulent flows. *Journal of Computational Physics*, 419:109681, 2020.

[13] J.S. Hesthaven and S. Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[15] Kookjin Lee and K. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *J. Comput. Phys.*, 404, 2020.

[16] Ricardo Vinuesa and Steven L Brunton. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2(6):358–366, 2022.

[17] Markus Mrosek, Carsten Othmer, and Rolf Radespiel. Variational autoencoders for model order reduction in vehicle aerodynamics. In *AIAA AVIATION 2021 FORUM*, page 3049, 2021.

[18] Eric J Parish and Karthik Duraisamy. A paradigm for data-driven predictive modeling using field inversion and machine learning. *Journal of computational physics*, 305:758–774, 2016.

[19] Ping He, Rakesh Halder, Krzysztof Fidkowski, Kevin Maki, and Joaquim RRA Martins. An efficient nonlinear reduced-order modeling approach for rapid aerodynamic analysis with OpenFOAM. In *AIAA Scitech 2021 Forum*, page 1476, 2021.

[20] Rakesh Halder, Krzysztof J Fidkowski, and Kevin J Maki. An adaptive sampling algorithm for reduced-order models using Isomap. *International Journal for Numerical Methods in Engineering*, page e7427, 2024.

[21] Suhas V Patankar and D Brian Spalding. A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. *International Journal of Heat and Mass Transfer*, 15(10):1787–1806, 1972.

[22] CM Rhie and W Li Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA Journal*, 21(11):1525–1532, 1983.

[23] Giancarlo Alfonsi. Reynolds-averaged Navier–Stokes equations for turbulence modeling. 2009.

[24] P. Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. In *30th Aerospace Sciences Meeting and Exhibit*, June 1992.

[25] Florian R Menter. Improved two-equation k-omega turbulence models for aerodynamic flows. Technical report, 1992.

[26] Bijan Mohammadi and Olivier Pironneau. Analysis of the k-epsilon turbulence model. 1993.

[27] Henry G Weller, Gavin Tabor, Hrvoje Jasak, and Christer Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, 12(6):620–631, 1998.

[28] Markus Mrosek, Carsten Othmer, and Rolf Radespiel. Reduced-order modeling of vehicle aerodynamics via proper orthogonal decomposition. *SAE International Journal of Passenger Cars - Mechanical Systems*, 12(3):225–236, oct 2019.

[29] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936.

[30] Dunhui Xiao, Fangxin Fang, Christopher Pain, and Guangwei Hu. Non-intrusive reduced-order modelling of the Navier–Stokes equations based on RBF interpolation. *International Journal for Numerical Methods in Fluids*, 79(11):580–595, 2015.

[31] Manyu Xiao, Piotr Breitkopf, Rajan Filomeno Coelho, Catherine Knopf-Lenoir, Maryan Sidorkiewicz, and Pierre Villon. Model reduction by CPOD and Kriging: application to the shape optimization of an intake port. *Structural and multidisciplinary optimization*, 41:555–574, 2010.

[32] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[33] Saifon Chaturantabut and Danny C Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[34] Kevin Carlberg, Matthew Barone, and Harbir Antil. Galerkin v. least-squares Petrov–Galerkin projection in nonlinear model reduction. *Journal of Computational Physics*, 330:693–734, 2017.

[35] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. iii. dynamics and scaling. *Quarterly of Applied mathematics*, 45(3):583–590, 1987.

[36] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[37] Ping He, Charles A Mader, Joaquim RRA Martins, and Kevin J Maki. An aerodynamic design optimization framework using a discrete adjoint approach with OpenFOAM. *Computers & Fluids*, 168:285–303, 2018.

[38] Ruichen Jin, Wei Chen, and Agus Sudjianto. An efficient algorithm for constructing optimal design of computer experiments. *Journal of Statistical Planning and Inference*, 134(1):268–287, September 2005.

[39] Rakesh Halder, Krzysztof Fidkowski, and Kevin Maki. Local non-intrusive reduced order modeling using isomap. In *AIAA SCITECH 2022 Forum*, page 0081, 2022.

[40] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[41] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[42] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

[43] George Lee, Carlos Rodriguez, and Anant Madabhushi. Investigating the efficacy of nonlinear dimensionality reduction schemes in classifying gene and protein expression studies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(3):368–384, 2008.

[44] Arellano J Gámez, CS Zhou, Axel Timmermann, and Jürgen Kurths. Nonlinear dimensionality reduction in climate data. *Nonlinear Processes in Geophysics*, 11(3):393–398, 2004.

[45] Gaurav Bansal, Ajith Arthur Mascarenhas, and Jacqueline H Chen. Identification of intrinsic low dimensional manifolds in turbulent combustion using an isomap based technique. Technical report, Sandia National Laboratries, Livermore, CA (United States), 2011.

[46] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[47] J Douglas Carroll and Phipps Arabie. Multidimensional scaling. *Measurement, judgment and decision making*, pages 179–250, 1998.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Rettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[49] Jan De Leeuw and Patrick Mair. Multidimensional scaling using majorization: SMACOF in R. *Journal of statistical software*, 31:1–30, 2009.

[50] David Amsallem, Matthew J. Zahr, and Charbel Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012.

[51] Romain Dupuis, Jean-Christophe Jouhaud, and Pierre Sagaut. Surrogate modeling of aerodynamic simulations for multiple operating conditions using machine learning. *Aiaa Journal*, 56(9):3622–3635, 2018.

[52] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

[53] Thierry Braconnier, Marc Ferrier, J-C Jouhaud, Marc Montagnac, and Pierre Sagaut. Towards an adaptive POD/SVD surrogate model for aeronautic design. *Computers & Fluids*, 40(1):195–209, 2011.

[54] Marc Guénot, Ingrid Lepot, Caroline Sainvitu, Jordan Goblet, and Rajan Filomeno Coelho. Adaptive sampling strategies for non-intrusive POD-based surrogates. *Engineering computations*, 30(4):521–547, 2013.

[55] Jiachen Wang, Xiaosong Du, and Joaquim R Martins. Novel adaptive sampling algorithm for POD-based non-intrusive reduced order model. In *AIAA AVIATION 2021 FORUM*, page 3051, 2021.

[56] Thomas Franz, Ralf Zimmermann, and Stefan Görtz. Adaptive sampling for nonlinear dimensionality reduction based on manifold learning. *Model Reduction of Parametrized Systems*, pages 255–269, 2017.

[57] Thomas Franz, Ralf Zimmermann, Stefan Görtz, and Niklas Karcher. Interpolation-based reduced-order modelling for steady transonic flows via manifold learning. *International Journal of Computational Fluid Dynamics*, 28(3-4):106–121, 2014.

[58] Gaetan Kenway, Graeme Kennedy, and Joaquim R. R. A. Martins. *A CAD-Free Approach to High-Fidelity Aerostructural Optimization.*

[59] Rakesh Halder, Krzysztof J Fidkowski, and Kevin J Maki. Non-intrusive reduced-order modeling using convolutional autoencoders. *International Journal for Numerical Methods in Engineering*, 123(21):5369–5390, 2022.

[60] Rakesh Halder, Mohammadmehdi Ataei, Hesam Salehipour, Krzysztof Fidkowski, and Kevin Maki. Reduced-order modeling of unsteady fluid flow using neural network ensembles. *arXiv preprint arXiv:2402.05372*, 2024.

[61] Stefania Fresca, Luca Dede, and Andrea Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized PDEs. *J. Sci. Comput.*, 87:61, 2021.

[62] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[63] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

[64] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. Deep learning based recommender system: A survey and new perspectives. *ACM Computing Surveys (CSUR)*, 52(1):1–38, 2019.

[65] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[66] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *Icml*, 2010.

[67] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8609–8613, 2013.

[68] Matthew D. Zeiler, Marc'Aurelio Ranzato, Rajat Monga, Mark Z. Mao, K. Yang, Quoc V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and Geoffrey E. Hinton. On rectified linear units for speech processing. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521, 2013.

[69] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[70] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[71] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980 Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[72] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[73] Xue Ying. An overview of overfitting and its solutions. *Journal of Physics: Conference Series*, 1168:022022, 02 2019.

[74] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[75] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[76] Matus Telgarsky. Benefits of depth in neural networks. In *Conference on learning theory*, pages 1517–1539. PMLR, 2016.

[77] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In Vitaly Feldman, Alexander Rakhlin, and Ohad Shamir, editors, *29th Annual Conference on Learning Theory*, volume 49 of *Proceedings of Machine Learning Research*, pages 907–940, Columbia University, New York, New York, USA, 23–26 Jun 2016. PMLR.

[78] J. Sola and J. Sevilla. Importance of input data normalization for the application of neural networks to complex industrial problems. *IEEE Transactions on Nuclear Science*, 44:1464–1468, 1997.

[79] G E Hinton and R R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[80] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer, 1999.

[81] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[82] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[83] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[84] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[85] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning internal representations by error propagation, 1985.

[86] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[87] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[88] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[89] Sk M Rahman, Suraj Pawar, Omer San, Adil Rasheed, and Traian Iliescu. Nonintrusive reduced order modeling framework for quasigeostrophic turbulence. *Physical Review E*, 100(5):053306, 2019.

[90] Alberto Solera-Rico, Carlos Sanmiguel Vila, M. A. Gómez, Yuning Wang, Abdulrahman Almashjary, Scott T. M. Dawson, and Ricardo Vinuesa. $\beta$-variational autoencoders and transformers for reduced-order modelling of fluid flows, 2023.

[91] Sangseung Lee and Donghyun You. Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *Journal of Fluid Mechanics*, 879:217–254, 2019.

[92] Pin Wu, Feng Qiu, Weibing Feng, Fangxing Fang, and Christopher Pain. A non-intrusive reduced order model with transformer neural network and its application. *Physics of Fluids*, 34(11), 2022.

[93] Romit Maulik, Bethany Lusch, and Prasanna Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3), 2021.

[94] Kazuto Hasegawa, Kai Fukami, Takaaki Murata, and Koji Fukagata. Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes. *Theoretical and Computational Fluid Dynamics*, 34:367–383, 2020.

[95] Joongoo Jeon, Juhyeong Lee, Ricardo Vinuesa, and Sung Joong Kim. Residual-based physics-informed transfer learning: A hybrid method for accelerating long-term CFD simulations via deep learning. *International Journal of Heat and Mass Transfer*, 220:124900, 2024.

[96] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[97] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[98] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[99] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[100] David J Tritton. Experiments on the flow past a circular cylinder at low reynolds numbers. *Journal of Fluid Mechanics*, 6(4):547–567, 1959.

[101] Eileen M Saiki and S Biringen. Numerical simulation of a cylinder in uniform flow: application of a virtual boundary method. *Journal of computational physics*, 123(2):450–465, 1996.

[102] Mohammadmehdi Ataei and Hesam Salehipour. XLB: Distributed multi-GPU lattice Boltzmann simulation framework for differentiable scientific machine learning, 2023.

[103] Shiyi Chen and Gary D Doolen. Lattice Boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364, 1998.

[104] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[105] J. H. Gerrard. Flow around circular cylinders; volume 1. fundamentals. by M. M. Zdravkovich. oxford science publications, 1997. *Journal of Fluid Mechanics*, 350:375–378, 1997.

[106] BN Rajani, A Kandasamy, and Sekhar Majumdar. Numerical simulation of laminar flow past a circular cylinder. *Applied Mathematical Modelling*, 33(3):1228–1247, 2009.

[107] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[108] Christopher L Rumsey and Susan X Ying. Prediction of high lift: review of present CFD capability. *Progress in Aerospace Sciences*, 38(2):145–180, 2002.

[109] Anand Pratap Singh and Karthik Duraisamy. Using field inversion to quantify functional errors in turbulence closures. *Physics of Fluids*, 28(4), 2016.

[110] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA journal*, 55(7):2215–2227, 2017.

[111] Mike Giles and Niles Pierce. Adjoint equations in CFD - duality, boundary conditions and solution behaviour. *AIAA Journal*, 97, 04 2000.

[112] Gaetan KW Kenway, Charles A Mader, Ping He, and Joaquim RRA Martins. Effective adjoint approaches for computational fluid dynamics. *Progress in Aerospace Sciences*, 110:100542, 2019.

[113] Neil Wu, Gaetan Kenway, Charles A. Mader, John Jasa, and Joaquim R. R. A. Martins. pyoptsparse: A Python framework for large-scale constrained nonlinear optimization of sparse systems. *Journal of Open Source Software*, 5(54):2564, 2020.

[114] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

[115] Nigel Gregory and CL O'Reilly. Low-speed aerodynamic characteristics of NACA 0012 aerofoil section, including the effects of upper-surface roughness simulating hoar frost. 1970.

[116] Michael Gaster. *The structure and behaviour of laminar separation bubbles*. Citeseer, 1967.

[117] Florian R Menter, R Langtry, and S Völker. Transition modelling for general purpose cfd codes. *Flow, turbulence and combustion*, 77:277–303, 2006.

[118] DB Spalding. A single formula for the law of the wall. *Journal of Applied Mechanics*, 28(3):455–458, 1961.

[119] Joel Ho and Alastair West. Field inversion and machine learning for turbulence modelling applied to three-dimensional separated flows. In *AIAA Aviation 2021 Forum*, page 2903, 2021.

[120] Christopher L Rumsey, Gary N Coleman, and Li Wang. In search of data-driven improvements to RANS models applied to separated flows. In *AIAA Scitech 2022 Forum*, page 0937, 2022.

[121] Krzysztof J Fidkowski. Gradient-based shape optimization for unsteady turbulent simulations using field inversion and machine learning. *Aerospace Science and Technology*, 129:107843, 2022.

[122] Krzysztof Fidkowski. An interactive airfoil analysis and design tool in Matlab. In *AIAA SCITECH 2023 Forum*, page 0551, 2023.

[123] Mark Drela. XFOIL: An analysis and design system for low reynolds number airfoils. In *Low Reynolds Number Aerodynamics: Proceedings of the Conference Notre Dame, Indiana, USA, 5–7 June 1989*, pages 1–12. Springer, 1989.