Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project
ACSE Final Report

A comparison of dimensionality reduction methods for fluid flow problems focusing on hierarchical autoencoders

by

Fan Yang

fan.yang20@imperial.ac.uk
GitHub login: acse-fy120

Supervisors:

Dr. Claire Heaney
Prof. Christopher Pain

August 2021

GitHub repository: https://github.com/acse-fy120/acse-9-independent-research-project-fy120

Abstract

In recent years, a massive amount of high-dimensional data have been accumulated as big fluid data due to high-performance computing and advances in experimental measurement capabilities. To extract meaningful information from data that ease understanding of essential fluid phenomena, the dimensionality reduction method is introduced to transform the high-dimensional data into low-dimensional data. To investigate the application of the dimensionality reduction methods in fluid dynamics, this project implements and compares many dimensionality reduction methods, including proper orthogonal decomposition(POD), fully-connected autoencoder (FC-AE), convolutional autoencoder (CAE), space-filling curve - convolutional autoencoder (SFC-CAE), hierarchical autoencoder (HAE) and sequential autoencoder (SAE). Moreover, the novel space-filling curve-hierarchical autoencoder (SFC-HAE) is proposed, extracting principal components from the fluid data on unstructured mesh in order. The mentioned methods are assessed with two fluid solutions, namely: (1) burgers equation, (2) flow past cylinder, in terms of mean square error (MSE) and computation time. The results of the experiments reveal that although the capability to reduce the dimensionality of each method is different, the mentioned methods all capture the basic features of fluid flow very well using the appropriate number of latent variables. Based on these results, the advantages and drawbacks of each method are discussed in this report.

Keywords: Dimension Reduction, Hierarchical Autoencoder, Autoencoder, Proper Orthogonal Decomposition, Space-Filling Curve

1 Introduction

High-dimensional data is common throughout computational science due to progress made in high-performance computing and high-resolution numerical methods (Van Der Maaten et al. 2009). In fluid dynamics, because of the high-resolution data in both space and time, a massive amount of high-dimensional data can be accumulated (Hasegawa et al. 2020*a*). To remove redundant, irrelevant data and mitigate the curse of dimensionality, the data's dimensionality can be reduced (Jimenez & Landgrebe 1998). Dimensionality reduction is the transformation of high-dimensional data into low-dimensional data so that the low-dimensional data retains the important information and features of the original data to ease its use in consequent tasks (Van Der Maaten et al. 2009, Kramer 1991).

Numerous dimensionality reduction methods have been proposed for decades, including proper orthogonal decomposition (POD)/principal component analysis(PCA) and autoencoder. The POD contributes the optimal information preserving transformation among linear methods (Fukunaga & Koontz 1970). The application of the POD to the fluid mechanics' community was pioneered by Lumley (1967), by combining with the method called snapshots (Sirovich 1987), successfully extracting the information from turbulent flow data. The POD is a suitable, general-purpose method to perform dimensionality reduction. However, because of the linear property, the POD may oversimplify the complex relationship among the original data, leading to an unsatisfactorily high loss of important information (Kramer 1991). In addition, for nonlinear systems, the troublesome problem is that the reproduction of the flow fields requires a large number of modes since nonlinear phenomena should be approximated by a linear superposition of orthogonal modes (Hasegawa et al. 2020*b*). For example, Alfonsi & Primavera (2007) found that a flow field in the plane channel needs at least 7260 modes to reconstruct the solutions to a reasonable degree of accuracy. To overcome these limitations, the concept of nonlinear PCA has been proposed and results in the development and application of various methods. For example, Irie & Kawato (1991) adopted a sandglass-type multi-layered perceptron(MLP) and Hastie & Stuetzle (1989) adopted principal curves to perform the nonlinear

PCA. Unfortunately, these methods cannot determine the ratio of contribution of each principal component. In addition, Schölkopf et al. (1998) proposed a method called kernel principal component analysis, which performs PCA after performing nonlinear mapping of the data. However, it is hard to determine the mapping function due to the unknown nonlinear mapping relationship. Therefore, these methods are limited in application to solve real-world problems. Because the neural network has the property of good nonlinear processing ability and strong learning ability, the autoencoder, a kind of unsupervised artificial neural network with nonlinear activation functions, seems to be a better choice.

The autoencoder was originally by Kramer (1991), who constructed five layers neural network to extract nonlinear principal components successfully. The application of the autoencoder to dimensionality reduction was pioneered by Milano & Koumoutsakos (2002), who compared the reconstruction and prediction capabilities of the POD and autoencoder for the near-wall velocity fields. Finally, they concluded that the autoencoder provided better dimensionality reduction capabilities than the POD. Brunton et al. (2020), Phillips et al. (2020), Baldi & Hornik (1989) attributed the superior dimensionality reduction capability of the autoencoder to the presence of a nonlinear activation function, which facilitates to process complex nonlinear data. However, the application of the autoencoder may need more considerable computational time because of the iterative training process. Based on the theories of autoencoder and SVD, Phillips et al. (2020) proposed an SVD-autoencoder (SVD-AE) method to reduce the computational cost by decreasing the length of the input vector of the autoencoder. In addition, since the number of learnable weights in convolutional neural network is independent of the input feature size, compared with the fully-connected autoencoder (FC-AE), the convolutional autoencoder (CAE) is suitable to deal with high-dimensional data (Tencer & Potter 2020). Hasegawa et al. (2020*a*) has adopted the CAE to reduce the dimensionality of data successfully before learning the dynamics of the system with a long short-term memory (LSTM) network. However, because all the principal components are extracted by a single network, compared with the POD, the encoded latent variables of the autoencoder cannot be ranked in the order of energy contribution (Fukami et al. 2020). To solve the problem, Kramer (1991) proposed sequential autoencoder (SAE), which extracts the principal component by using multiple separate neural networks, and the residual of the current network is the input of the succeeding network. In addition, Saegusa et al. (2004) proposed a similar method called hierarchical autoencoder (HAE), replacing the residual with the principal component. Many scholars have adopted the theories of the hierarchical autoencoder and sequential autoencoder to solve problems. For example, Fukami et al. (2020) used the hierarchical convolutional autoencoder (HCAE) to extract the features of turbulent channel flow at $\mathrm{Re} = 180$ successfully. Finally, they concluded that compared with CAE, the HCAE provides more efficient compression capabilities. Recent work has shown that CAE is restricted to structured meshes (Xu et al. 2021, Jiang et al. 2018, 2019, Casas et al. 2020), which means they perform poorly for data held on unstructured meshes. However, for many computational fluid dynamics (CFD) problems, an unstructured mesh is necessary to solve complex geometry and interfaces (Walton et al. 2017, Xie et al. 2016). To solve the limitations, Heaney et al. (2020) used the space-filling curve (SFC) to transform the high-dimensional data on unstructured meshes to one-dimensionality data and then used 1D CAE to perform the dimensionality reduction. However, they only applied the traditional CAE and did not extend this method to other types of autoencoder, such as hierarchical autoencoder and sequential autoencoder.

The existing research focusing on the hierarchical autoencoder mainly use the convolutional autoencoder (Fukami et al. 2020). However, it is hardly to see that combining the space-filling curve and hierarchical autoencoder to deal with the data on unstructured mesh. Moreover, motivated by the lack of a systematic comparison of dimensionality reduction methods applied to fluid flow problems, the project is to investigate the application of the dimensionality reduction methods in fluid dynamics more specifically. In this project, the effectiveness of several dimensionality reduction methods will be compared and the relative merits of each method as applied to different types and amounts of data

will be assessed. The detailed information of the dimensionality reduction methods and fluid flow data used in this project are displayed in appendices. Distinct from previous research, the methods of this project is not limited to SFC-CAE and hierarchical autoencoder. Both space-filling curve hierarchical autoencoder (SFC-HAE) and space-filling curve sequential autoencoder (SFC-SAE) have been implemented.

The detailed description of the methodology used in this project will be presented in section 2. Code implementation and evaluation of results will be demonstrated in section 3. Finally, section 4 will evaluate the methods and deliver the conclusion.

2 Methodology

This section summarizes the dimensionality reduction methods used in this project, which were coded based on paper (Fukami et al. 2020) and code on Github. The specific information about the Github code is provided in section 2.5. All dimensionality reduction methods were implemented in Python. The reason is the PyTorch, a polished open-source machine learning library (Paszke et al. 2019). For the training process of the network, the jupyter notebook running on Colab is chosen since the Colab provides the GPU to strongly accelerate the computation and reduce the training time. The code about flow past cylinder data post-processing is preferable to be run on the local computer because the file of vtu could be opened with Paraview (Ahrens et al. 2005) on the local computer.

2.1 Dataset

2.1.1 Burgers Equation

The Burgers equation is a fundamental partial differential equation, whose general form is present as following,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \tag{1}$$

where $u(x,t)$ is the speed of the fluid at spatial and temporal coordinates, x is the spatial coordinate, $t \in [0,T]$ is temporal coordinates, ν represents the viscosity of the fluid and in code. The initial condition is set as:

$$u(x,0) \;=\; \begin{cases} e^{-0.01x^2}, & x \in (-L/2, L/2) \\ 0, & otherwise \end{cases} \tag{2}$$

In this problem, $L = 200$, $T = 300$, $\nu = 0.01$. The spatial domain is discretised to 200 nodes with $\Delta x = 1$. The time domain is discretised to 300 time steps with $\Delta t = 1$. The holdout cross-validation is adopted, and the dataset is randomly divided into three parts according to the ratio of 8:1:1 for training, validation and testing. Based on the considerably trial, SVD-autoencoder has been found that is more sensitive to normalization. Therefore, the normalization is only adopted to SVD-autoencoder, which transforms u into the range from 0 to 1 after employing SVD.

2.1.2 Flow Past Cylinder

The conservation form of the equations of motion of an incompressible fluid may be written as

$$\nabla \cdot u \;=\; 0, \tag{3}$$

$$\frac{\partial}{\partial t}(\rho u) + \nabla \cdot (\rho u \otimes u) = -\nabla p + \nabla \cdot \tau + \rho g, \tag{4}$$

where t indicates the time, $u \;=\; [u\; v]^T$ indicates the velocity with u and v in the x and y directions, p is the pressure, ∇ is the divergence, $\otimes$ is the output product, τ is the deviatoric stress matrix, g is the body accelerations acting on the continuum and is set as 0 in this problem. In addition, the Reynolds number Re is set as 3900.

The dataset, with the type of vtu type generated from the solutions of the above equations, consists of 2000 snapshots. Each snapshot has N = 20550 nodes. Although the vtu file also includes some extra information like pressure, the velocity $u = [u\, v]^T$ is the focus in this project. Compared with the u, the y-direction velocity v is very small. Consequently, data normalization is an indispensable pre-processing step that concerns transforming different features into a common scale. Hence, the greater numeric feature values u cannot dominate the smaller numeric features values v (Singh & Singh 2020). After the data normalization, both u and v range from -1 to 1. The holdout cross-validation is adopted in this problem, and the dataset division ratio is still 8:1:1 for training, validation and testing.

2.2 Dimensionality Reduction Methods

The concepts will be explained based on the actual application in the project. From the limitation of the number of words, only a brief introduction is given here.

2.2.1 Proper Orthogonal Decomposition

The S is a snapshots matrix($S \in \boldsymbol{R}^{\mathrm{N}\times\mathrm{M}}$), where S_{ij} is the value of ith variable in jth snapshot. To apply POD, the solution is usually arranged in a column vector, which means for the 2D case, the number of degrees of freedom N should be twice the number of nodes. For example, the flow past cylinder solution with size (2000,20550,2) is arranged in a snapshots matrix with size (2000,41100). The POD basis functions can be calculated via a truncated singular value decomposition(SVD) of the data matrix given by the snapshots. The SVD is a factorization of the form

$$S = U\Sigma V^T, \tag{5}$$

where U and V are square orthogonal matrices, Σ is a diagonal matrix containing the singular values. The POD basis functions consist of the columns of U. If the number of the basis functions P is confirmed, P basis functions are stored in the matrix $R \in \boldsymbol{R}^{N\times P}$. The reduced variable $\alpha(\alpha \in \boldsymbol{R}^{P\times 1})$ of a snapshot u ($u \in \boldsymbol{R}^{\mathrm{N}\times 1}$) can be calculated by

$$\alpha = R^T u, \tag{6}$$

and the reconstructed variable can be calculated by

$$u = R\alpha, \tag{7}$$

In this project, the mean square error(MSE) is the optimisation criterion, defined as following in POD. Note that the lower MSE means the higher fidelity of the reconstruction.

$$\frac{1}{N}\sum_{p=1}^{N}(u^{t_p} - RR^T u^{t_p}) \cdot (u^{t_p} - RR^T u^{t_p}), \tag{8}$$

where N is the number of the snapshots. The architecture of the POD is presented in figure 1(a).

2.2.2 Autoencoder

The autoencoder consists of encoder and decoder, defined as extraction function ϕ and reconstruction function ψ. Supposing the original input x ($x \in \boldsymbol{R}^n$), the reduced variable $y(y \in \boldsymbol{R}^m)$ after compressing by encoder can be derived using ϕ, which is defined as

$$y = \phi(x, \omega_{ENC}), \tag{9}$$

The ψ that is used to reconstruct from the y is defined as

$$\widehat{x} = \psi(y, \omega_{DEC}), \tag{10}$$

where ω_{ENC} and ω_{DEC} are the weights in encoder and decoder respectively, $\widehat{x}$ ($\widehat{x} \in \boldsymbol{R}^n$) is the output of autoencoder, whose dimension is identical to the original input x'. Combining the ϕ and ψ, the autoencoder function is defined as

$$\widehat{x} = F(x, \omega) = \psi(\phi(x, \omega_{ENC}), \omega_{DEC}), \tag{11}$$

where ω is the weight in the autoencoder. The loss is minimised by adjusting weights ω in the training process, using the backpropagation algorithm. The optimisation criterion, MSE in autoencoder, is defined as following

$$MSE \ = \ E[\| \ x - \widehat{x} \ \|^2], \tag{12}$$

Where E represents the expected value. The architecture of fully-connected autoencoder is presented in figure 1(b). As for the convolutional autoencoder, the encoder has several convolutional layers generally followed by some fully-connected layers, and the decoder is the reverse of this.

2.2.3 SVD Autoencoder

This method is the combination of the SVD and autoencoder, whose architecture is shown in figure 1(c). First, the SVD is adopted to reduce the input vector from size N to size $P(P << N)$, see equation (6). Then the autoencoder will be trained to further reduce the dimensionality, whose input is the reduced vector processed by SVD. Based on trial observation, it is not recommended to directly use SVD to reduce the data dimension to a low value because this will lead to poor dimensionality reduction effects of subsequent autoencoders. Therefore, when using SVD-autoencoder to perform dimensionality reduction, 150 basis functions and 1600 basis functions have been selected respectively in dealing with burgers equation and flow past cylinder solution. For more details on the SVD-autoencoder, see Phillips et al. (2020)'s paper.

2.2.4 Sace-Filling Curve Autoencoder

The space-filling curve (SFC) is a method that maps the multi-dimensional space into the one-dimensional space representation, which acts like a line that traverses each node or cell in the multi-dimensional space and that visits each node or cell exactly once. Many SFCs including Peano, Z-order and Moore have been widely adopted as mapping schemes (Moore 1900, Sagan 2012). The main difference of the SFC attributes to the way of mapping to the one-dimensional space. It is generally accepted that the Hilbert SFC achieves the optimal clustering (Abel & Mark 1990, Moon et al. 2001). Hence, only the Hilbert SFC is adopted in this project. The mapping that transforms multi-dimensional data into 1D data through SFC contributes a pre-processing step to multi-dimensional data on unstructured meshes. Thus the conventional convolutional autoencoder could be applied to post-processing data directly. The architecture of the convolutional autoencoder with two SFCs is shown in figure 1(d). For more details on the SFC-autoencoder, see Heaney et al. (2020)'s paper.

2.2.5 Hierarchical Autoencoder

The hierarchical autoencoder consisted of several independent subnetworks is capable of extracting principal components $y_1, y_2, \ \cdots \ y_m$ in an ordered manner. Generally, the number of the subnetworks coincides with the number of principal components to be extracted, which means the number of each subnetwork's latent variable is one. The architecture of the hierarchical autoencoder is presented in figure 1(e). The extraction function ϕ_i: $\boldsymbol{R}^n \rightarrow \boldsymbol{R}^1$ is

$$y_i = \phi_i(x, \omega_{ENC_i}), \quad i \ = 1, \cdots m, \tag{13}$$

The reconstruction function ψ_i : $\boldsymbol{R}^i \rightarrow \boldsymbol{R}^n$ is

$$\widehat{x}_i = \psi_i(y_1, \cdots y_i, \omega_{DEC_i}), \quad i \ = \ 1, \cdots, m, \tag{14}$$

where ω_{ENC_i} and ω_{DEC_i} are the weights of encoder and decoder respectively in the i_{th} subnetwork. Combining the ϕ_i and ψ_i, the i_{th} subnetwork function is defined as

$$\widehat{x}_i = F_i\left(x,\ \{y_1, \cdots y_{i-1}\},\ \omega_i\right), \quad i\ = 1, \cdots m, \tag{15}$$

where ω_i is the weight of the i_{th} subnetwork. The input vector x and the principal components $y_1,\ y_2, \cdots y_{i-1}$ from all lower-order subnetwork are the input of the i_{th} subnetwork. Each subnetwork is trained independently, which means the weight ω_i of the i_{th} subnetwork is only modified in the i_{th} subnetwork. The weight of each subnetwork is adjusted to minimise the optimisation criterion, MSE, defined in equation (12).

A similar method was proposed by Kramer (1991), where the residuals instead of the principal components of the lower-order subnetworks are transferred to the higher-order subnetwork. Although this architecture also belongs to a hierarchical autoencoder, it is called sequential autoencoder in this report to distinguish from the above-mentioned hierarchical autoencoder. The architecture of the sequential autoencoder is shown in figure 1(f).

As stated in the above theory of the hierarchical autoencoder and sequential autoencoder, the number of principal components to be extracted corresponds to the number of subnetworks. However, in this project, because of the complexity of the fluid flow, numerous principal components are required to represent it. It is hard to construct numerous different subnetworks to perform dimensionality reduction. Consequently, a concept of ordered autoencoder mode family is adopted, where each subnetwork can extract multiple principal components.

2.3 Development methodology

This project uses different dimensionality reduction methods to process different dimensional data, where each dimensionality reduction method does not depend on the implementations of other methods. In addition, it is common to train multiple autoencoders with different architectures simultaneously. Therefore, agile development (Mall 2018) is appropriate for this project.

2.4 Architectural Design

For flow past cylinder data, pre-processing is employed to convert the file from vtu type to csv type to prepare for loading the csv data into arrays afterwards. After the training, post-processing is used to convert the file from csv to vtu to visualise the data directly using Paraview. For the burgers equation, the pre-processing and post-processing are unnecessary because the dataset is the type of array generated using the function. The architectural design of the code is presented in figure 2. The detailed code logic is discussed in section 3.

2.5 Code metadata

The detailed information of the operating system, hardware requirements, platform, languages and libraries are presented in table 1.

Because the GitHub repository has a maximum limit of 1 GB and the flow past cylinder data exceeds the repository limit, the data using too much memory with different types is stored in google drive and the link is provided on the Github repository. In the Github repository, all the source code and ipynb files are uploaded. The results and implementation of the methods applied burgers equation and flow past cylinder can be found in $burgers\_equation\_methods$ and $fpc\_methods$ directory. The results of visualisation are placed in pics folder. The user should deploy files on Colab according to the directory tree in README files.

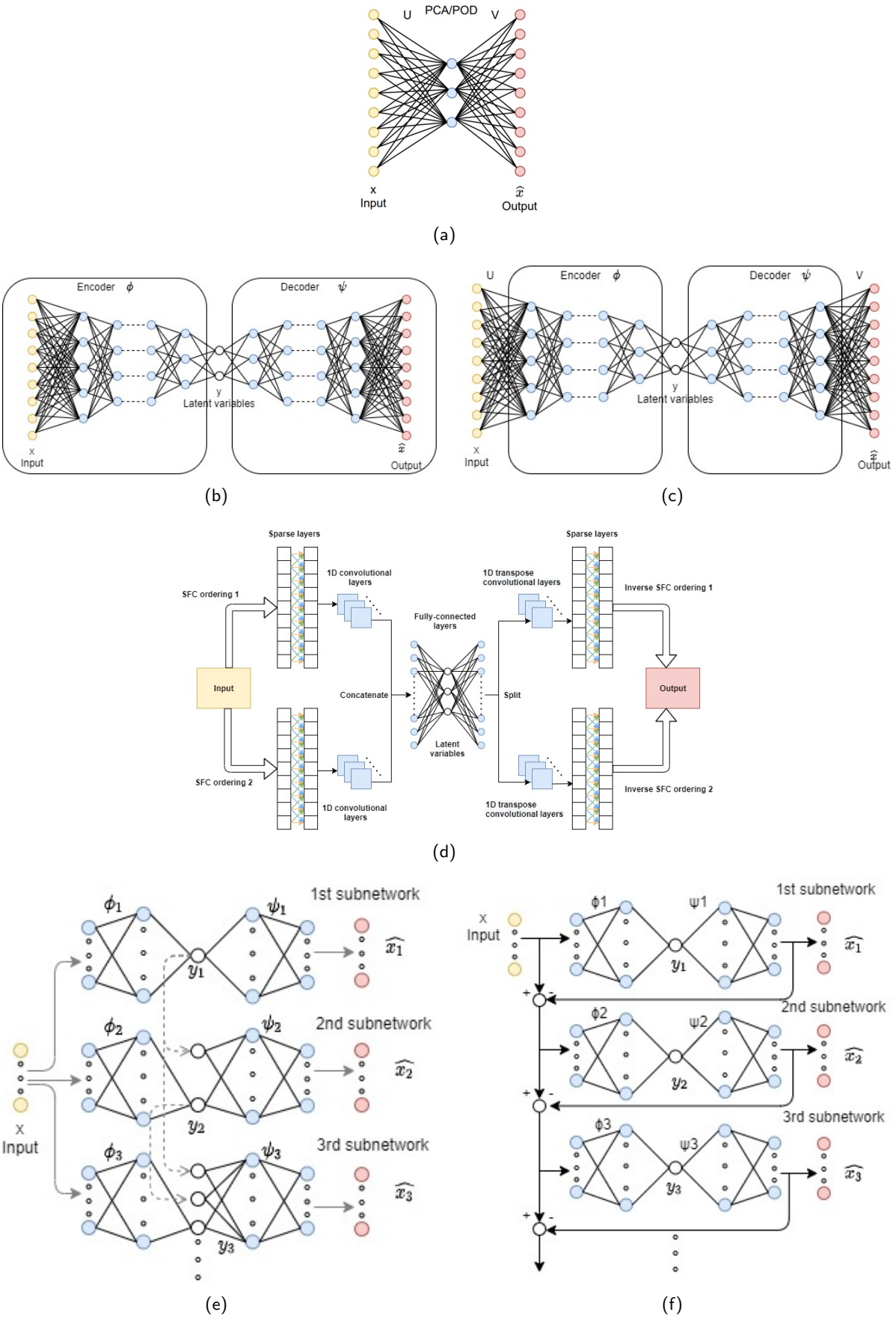


Figure 1: The architecture of mentioned dimensionality reduction methods. (a) POD/PCA; (b) Fully-connected autoencoder; (c) SVD-autoencoder; (d) SFC-convolutional autoencoder; (e) Hierarchical autoencoder; (f) Sequential autoencoder.

Operating system		
Windows 10		
Hardware		
A laptop with intel i5, 16G RAM and 512G ROM		
Platform		
Visual studio Code		Used for converting from csv file to vtu file
Colab		Used for training the autoencoder
Language		
Python		Most code are writted in Python
Fortran		The $space\_filling\_decomp\_new$ is written in Fortran
Libraries		
Name	Type	Description
space_filling_decomp_new	Proprietary	Provided by Professor Christopher Pain and Dr Claire Heaney. It contains the functions that nest dissection of the domain into subdomains in order to form a space-filling curve
vtktools	Proprietary	Available from Fluidity's github page.The URL is https://github.com/FluidityProject/fluidity/tree/main/python. It contains the functions to deal with VTK unstructured grids
SFC-CAE	Proprietary	Provided by Dr Claire Heaney. The URL is https://github.com/ImperialCollegeLondon/SFC-CAE. It contains the files that are used to convert the file type between vtu and csv
ffmpeg	Open source	Combined with cv2
cv2	Open source	Combined with ffmpeg, it is used to generate video from all figures in a folder
numpy	Open source	It is used to store and process data
pandas	Open source	The same to numpy. In addition, it contains the function that read CSV data to an array
scipy	Open source	It provides functions for mathematics
torch	Open source	It provides a wide range of algorithms, functions and models for machine learning
matplotlib	Open source	It is used for visualization
math	Open source	It is used to do math calculation
time	Open source	It is used to record the running time for the program
os	Open source	It is used to process the path of the current working directory

Table 1: The Code metadata

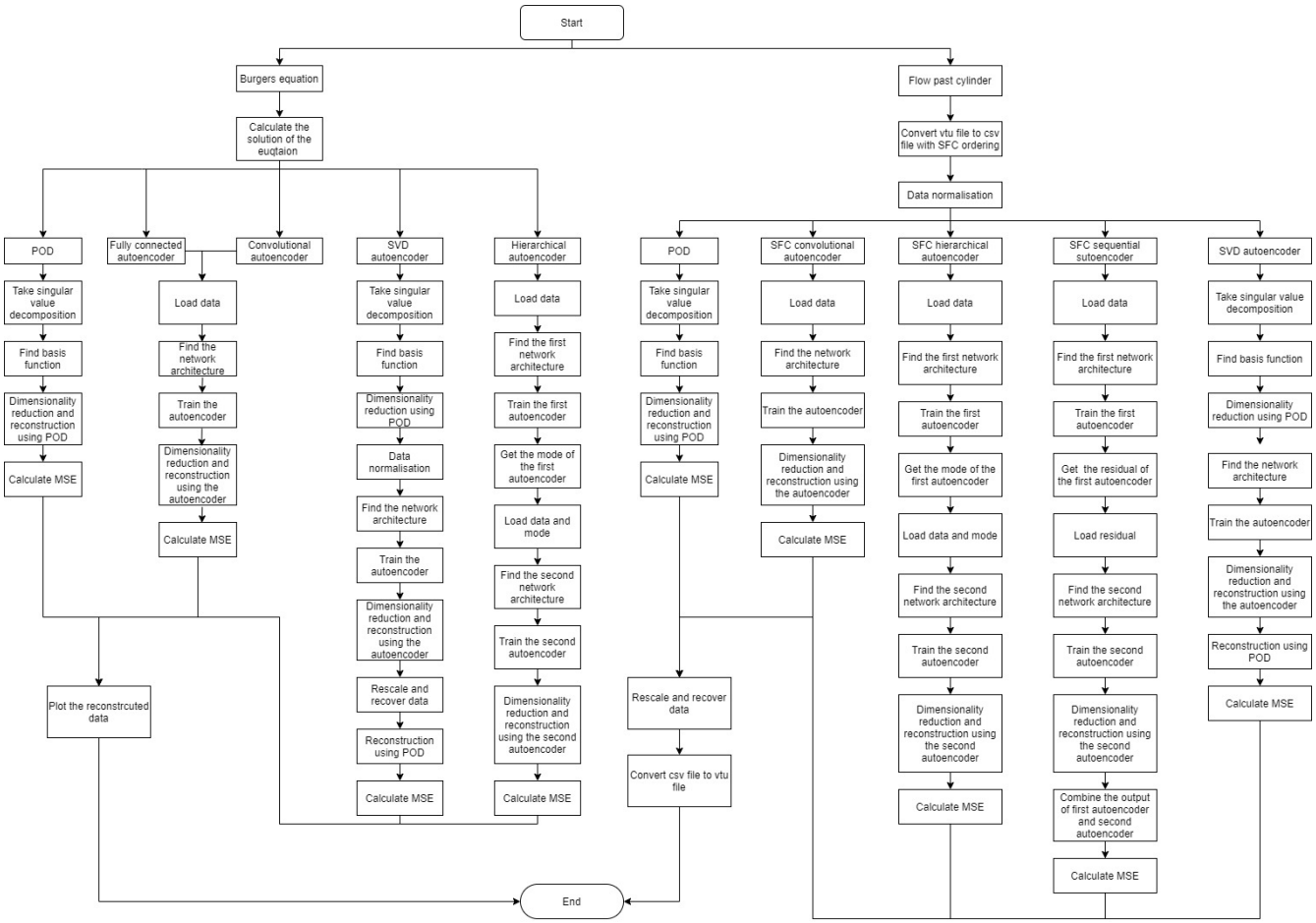


Figure 2: Architectural design of the project

3 Implementation and Results

In this section, the specific implementation of present dimensionality reduction methods applied to the different datasets will be described and the results will be presented. The focus of project was to appraise the effectiveness of the present dimensionality reduction methods, and each method is independent of the other. The unit test is adopted for each method and the success of the test for each method depends on whether the error between the reconstructed data and the original data is acceptable.

3.1 Burgers equation

3.1.1 Implementation Dimensionality Reduction Methods

The implementation of POD is fixed. Through using equation (5), (6), (7) to find basis function, reduce dimensionality and reconstruct the data separately. Unlike the POD, the input data to the autoencoder is usually arranged in a row vector. Based on trial observations, the performance of the autoencoder is sensitive to hyperparameters, including learning rate, number of epochs and batch size. The hyperparameter optimisation is performed manually and the results are shown in appendix. In addition, because the Adam algorithm (Kingma & Ba 2014) is straightforward to implement, computationally efficient and robust, the Adam is selected as optimiser in all models. For the weights and biases initialisation, the Xavier initialisation (Glorot & Bengio 2010) is adopted for the linear layer and Kaiming initialisation (He et al. 2015) is adopted for the convolutional and transposed convolution layer. Because the training process of the autoencoder is stochastic, and the dimensionality reduction

capability of convolutional autoencoder is sensitive to weights and biases initialisation in this problem. For every selected dimensionality, each method was run 10 times and the smallest MSE was reported in table 2.

3.1.2 Visualization

Because only the SVD-autoencoder adopts the data normalization, the reconstructed data obtained from SVD-autoencoder should be recovered to the original range. The function picTovideo is provided to generate a video that presents the change in curve over time. The video is saved in a specified path.

3.1.3 Simulation Results

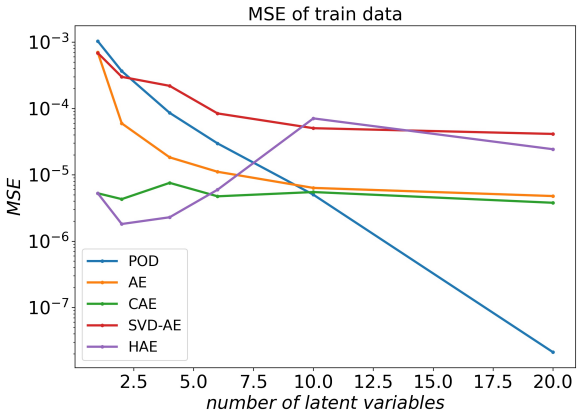
In this subsection, the effect of the latent variables on the performance of different dimensionality reduction methods on the burgers equation dataset is investigated. Multiple dimensionality reduction methods are adopted respectively to reduce the dimensionality from 200 to 1, 2, 4, 6, 10, 20. The results of the MSE are recorded in table 2. The curves of the table 2 data are plotted in figure 3.

From figure 3, it demonstrates that although the MSEs of train data, validation data and test data for each method are different, the decreasing trend of MSE in different data parts is similar for each method as the number of latent variables increases. Compared with the autoencoders, the MSE of POD decreases rapidly and linearly with the increasing number of latent variables. When the number of the latent variables is less than 6, the reconstructed data by autoencoder has high fidelity. The advantage of the autoencoder over POD attribute to the high representation ability of multiple hidden layers and nonlinear mapping functions, resulting in each latent variable is able to capture more information than POD basis function. According to the universal approximation theorem introduced by Hornik et al. (1989), when the large number of training data that far exceed the parameters of the network are available, an adequately large neural network is capable of representing a discretionarily complicated function. However, in this problem, the number of the training data is limited. This implies that with a larger number of latent variables, the dimensionality reduction capability of the autoencoder is restricted, leading to higher MSE compared with POD.

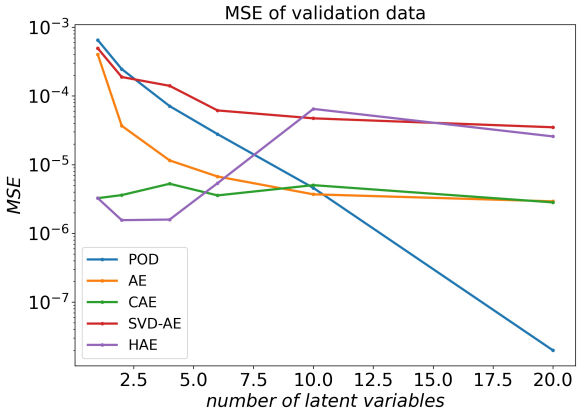
From the point of view of autoencoder, the network with additional hidden layers is able to learn more complex patterns and represent complex data effectively. Thus, the convolutional autoencoder consisting of convolutional layers and fully-connected layers can achieve more effective dimensionality reduction, resulting in lower MSE compared with the fully-connected autoencoder in this problem. The performance of the hierarchical autoencoder is excellent when the number of the latent variables is less than 6. As the number of variables increases further, the performance of the hierarchical autoencoder deteriorates. The reason is that the subnetworks cannot compress the variable to 5 and 10 latent variables effectively, resulting in poor performance of the whole network. This problem can probably be solved by changing the sub-network structure. Compared with other methods, the MSE of SVD-autoencoder finally stays at a large value, since the dimensionality reduction ability of SVD-autoencoder is limited by both POD and autoencoder simultaneously, which prevents the MSE of SVD-autoencoder from decreasing further.

Next, the reconstructed data obtained from the methods are compared with the original data in figure 4. Although the capability to capture features of each method is different, they all capture the basic features of burgers equation solution very well.

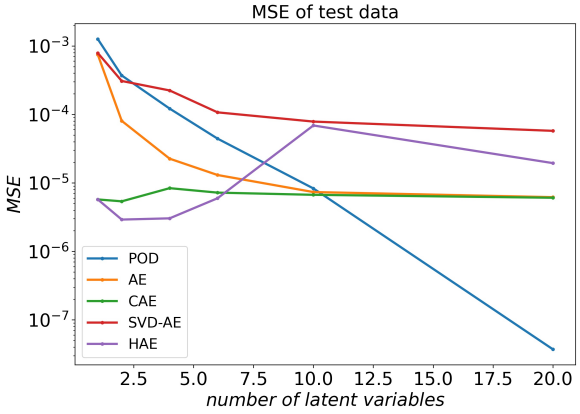
The comparison of the computational cost of mentioned dimensionality reduction methods is summarized in table 3(a). The reconstruction time is similar for POD and autoencoders and is a fraction of a second in all cases. Therefore, the computation time is not an essential metric to evaluate the dimensionality reduction methods in this problem.



(a)



(b)



(c)

Figure 3: The MSE of present dimensionality reduction methods in reconstructing the training, validation and test samples of burgers equation. The x-axis represents the number of latent variables in the reconstruction, and the y-axis represents the MSE of corresponding data (a) The MSE of present dimensionality reduction methods in training data; (b) The MSE of present dimensionality reduction methods in validation data; (c) The MSE of present dimensionality reduction methods in test data.

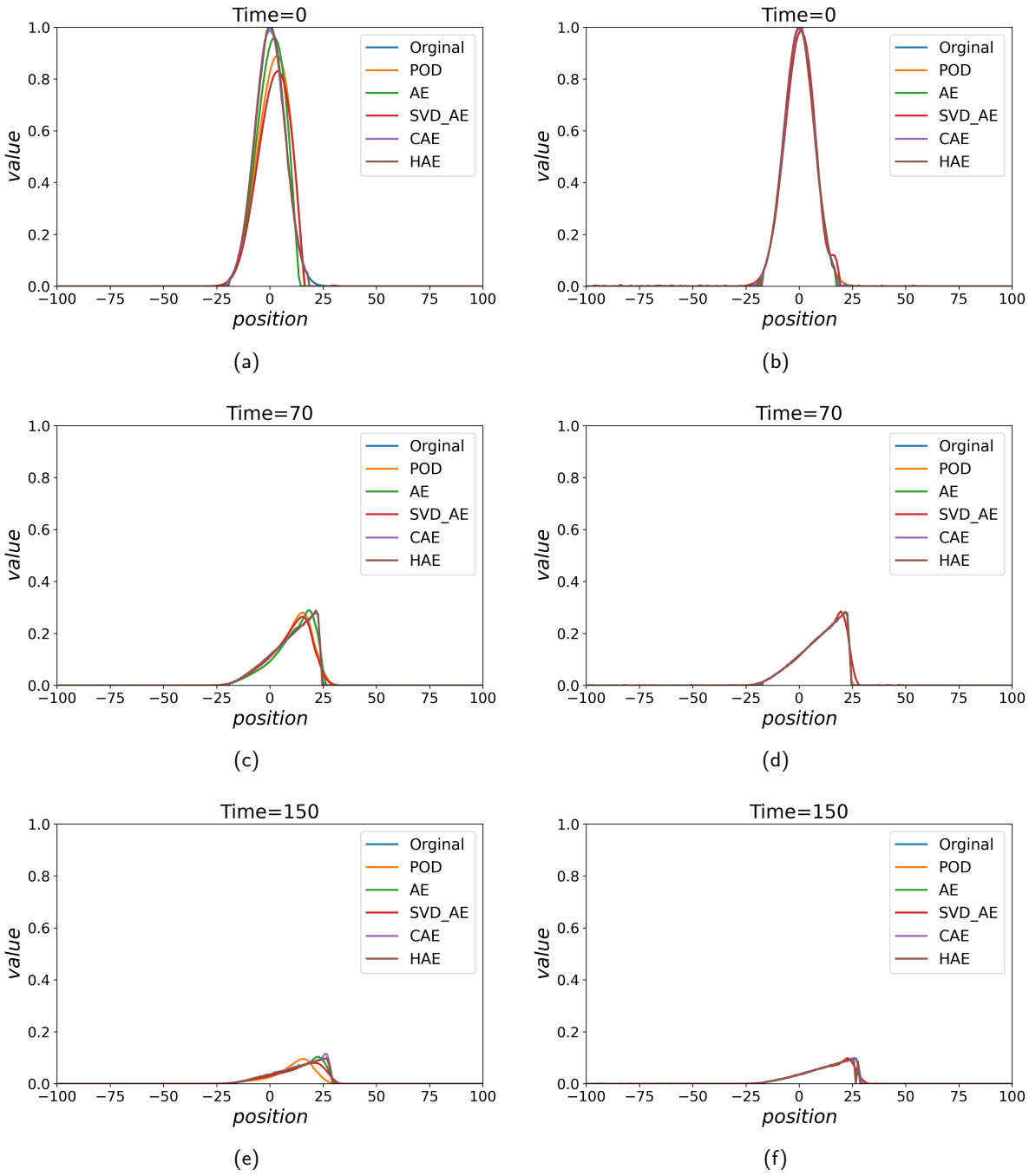


Figure 4: The comparison between the reconstructed data obtained from multiple dimensionality reduction methods and original data. (a) one sample (t = 0) in training data with 2 latent variables; (b) one sample (t = 0) in training data with 20 latent variables; (c) one sample (t = 70) in validation data with 2 latent variables; (d) one sample (t = 70) in validation data with 20 latent variables; (e) one sample (t = 150) in test data with 2 latent variables; (f) one sample (t = 150) in test data with 20 latent variables.

	POD	AE	CAE	SVD-AE	HAE	Compression
Training data	1.033×10^{-3}	6.968×10^{-4}	5.261×10^{-6}	6.852×10^{-4}		
Validation data	6.509×10^{-4}	4.010×10^{-4}	3.257×10^{-6}	4.934×10^{-4}		1
Test data	1.261×10^{-3}	7.488×10^{-4}	5.735×10^{-6}	7.890×10^{-4}		
Training data	3.655×10^{-4}	5.958×10^{-5}	4.297×10^{-6}	2.995×10^{-4}	1.812×10^{-6}	
Validation data	2.460×10^{-4}	3.680×10^{-5}	3.612×10^{-6}	1.887×10^{-4}	1.559×10^{-6}	2
Test data	3.705×10^{-4}	8.054×10^{-5}	5.379×10^{-6}	3.079×10^{-4}	2.914×10^{-6}	
Training data	8.577×10^{-5}	1.831×10^{-5}	7.554×10^{-6}	2.190×10^{-4}	2.289×10^{-6}	
Validation data	7.121×10^{-5}	1.154×10^{-5}	5.278×10^{-6}	1.405×10^{-4}	1.591×10^{-6}	4
Test data	1.217×10^{-4}	2.258×10^{-5}	8.378×10^{-6}	2.239×10^{-4}	3.037×10^{-6}	
Training data	2.971×10^{-5}	1.110×10^{-5}	4.737×10^{-6}	8.412×10^{-5}	5.955×10^{-6}	
Validation data	2.783×10^{-5}	6.730×10^{-6}	3.574×10^{-6}	6.159×10^{-5}	5.383×10^{-6}	6
Test data	4.432×10^{-5}	1.304×10^{-5}	7.210×10^{-6}	1.072×10^{-4}	5.970×10^{-6}	
Training data	5.013×10^{-6}	6.339×10^{-6}	5.492×10^{-6}	5.037×10^{-5}	7.079×10^{-5}	
Validation data	4.594×10^{-6}	3.706×10^{-6}	5.039×10^{-6}	4.726×10^{-5}	6.480×10^{-5}	10
Test data	8.305×10^{-6}	7.352×10^{-6}	6.699×10^{-6}	7.862×10^{-5}	6.898×10^{-5}	
Training data	2.122×10^{-8}	4.779×10^{-6}	3.785×10^{-6}	4.127×10^{-5}	2.428×10^{-5}	
Validation data	1.992×10^{-8}	2.929×10^{-6}	2.828×10^{-6}	3.503×10^{-5}	2.578×10^{-5}	20
Test data	3.715×10^{-8}	6.188×10^{-6}	6.068×10^{-6}	5.758×10^{-5}	1.945×10^{-5}	

Table 2: The MSE of present dimensionality reduction methods, trained to compress to the following number of variables: 1, 2, 4, 6, 10, 20. The dataset is the burgers equation. (Note that the lowest MSE values in test data are highlighted in grey)

(a)

	POD	AE	CAE	SVD-AE	HAE
Training[min]	4.737×10^{-4}	1.04	2.81	6.5	5.75
Reconstruction[s]	2.15×10^{-3}	1.75×10^{-3}	8.14×10^{-3}	1.99×10^{-3}	1.04×10^{-2}

(b)

	POD	SFC-CAE	SFC-HAE	SFC-SAE
Training[min]	4.1×10^{-1}	4.05×10^{2}	8.20×10^{2}	8.76×10^{2}
Reconstruction[s]	8.72×10^{-1}	5.312	5.55	7.27
	SVD-AE	2SFC-CAE	2SFC-HAE	2SFC-SAE
Training[min]	5.05	7.25×10^{2}	1.444×10^{3}	1.438×10^{3}
Reconstruction[s]	3.58	9.97	8.53	10.09

Table 3: Computation time for training the autoencoder, calculating the POD basis functions and reconstructing the data (a) Burgers equation; (b) Flow past cylinder (2SFC-HAE means the two SFC are generated simultaneously and each subnetwork will use two SFCs. This name is to distinguish it from the hierarchical autoencoder that generates two SFCs separately later).

3.2 Flow Past Cylinder

3.2.1 Implementation Dimensionality Reduction Methods

For POD, the procedure is the same as described in 3.1.1, except that the two-dimensional data is reorganised into one-dimensional data. As stated in section 2.2.3, for SVD-autoencoder, in this problem, the dimension of the data is reduced to 1600 from 20550 using SVD, and then the total data is arranged in a matrix with size(2000,3200), which is the input data of the subsequent autoencoder. For every selected dimensionality, SVD-autoencoder was run 10 times and the smallest MSE was reported in table 4.

As for hierarchical autoencoder, according to the Fukami et al. (2020)'s suggestion, the first subnetwork is built as a standard convolutional autoencoder. The architecture of subsequent subnetworks is obtained by changing the channels, stride, padding, activation function and number of layers. In the case of hierarchical autoencoder and sequential autoencoder, 10 networks runs were carried out, and the network with the best performance was selected as one of the subnetworks of the whole network. The activation function and hyperparameters shown in appendix is from Heaney et al. (2020), who have optimised the neural networks by tuning these hyperparameters. By simulating the same networks of SFC-CAE, SFC-HAE, SFC-SAE several times, it is found that the MSE of each simulation is similar. Therefore, for every selected dimensionality, SFC-autoencoders was only run 1 time and the MSE was reported in table 4.

(Note that the hyperparameters and architectures of mentioned methods are presented in 5.2 and 5.3 in appendix.)

3.2.2 Visualization

Because each dimensionality reduction method adopts the data normalization, the reconstructed data obtained from all methods should be recovered to the original range. The video or figure can be seen directly through using Paraview to open the files with type vtu.

3.2.3 Simulation Results

Figure 5 shows the MSE of mentioned dimensionality reduction methods using the different number of latent variables. From figure 5(a), 5(b), when the number of the latent variables increases, the MSE of SFC-HAE and SFC-SAE on training data and validation data is smaller than SFC-CAE. The MSE of the SFC-HAE and SFC-SAE on test data is also superior to that of the SFC-CAE, indicating that they do not over-train. These results demonstrate that both SFC-HAE and SFC-SAE can reconstruct data from the reduced variables in an effective manner. From figure 3, the MSE starts to level off when the number of the latent variable equals 10. However, figure 5 shows the number of latent variables leading to the same phenomena for MSE of SFC-CAE, SFC-HAE, SFC-SAE and SVD-AE is about 128. This happens because the flow past cylinder dataset is more complex, requiring more latent variables to represent the data. Moreover, from table 4 and figure 5, compared with the SFC-CAE, both the SFC-HAE and SFC-SAE perform poorly when the latent variables are equalling 2. The reason is that the fluid flow of flow past cylinder is complicated. The two subnetworks compress the variable from 20550 to 1 respectively, where the latent variables capture the most important, but potentially similar information in the original data, hence redundancy. With the increasing latent variable, the information captured by the two different subnetworks is different. The combination of the latent variable obtained from the two subnetworks carries much information from the original data, resulting in the lower MSE. As the number of latent variables increases, SVD-autoencoder seems to be overfitted which means the network is learning the training data well, but it does not generalize to new data. This phenomenon is because the number of iterations of weight learning is overlarge, leading to fit the unrepresentative features and noise in the training data. Regularization can be adopted to solve this problem.

Table 5(a) shows the losses of the two-SFCs based hierarchical autoencoders and one-SFC based hierarchical autoencoders. The two SFCs are generated respectively. Judging from the losses, the two-SFC based hierarchical autoencoder is not always better than the one-SFC based hierarchical autoencoder. The reason is that the dimensionality reduction capability depends on the quality of the single generating SFC. Although the second SFC is still Hilbert SFC, there is no guarantee that the quality is better than the first one.

From table 5(b), it is found that the MSE of hierarchical autoencoders consist of three subnetworks is slightly bigger than the MSE of hierarchical autoencoders consist of two subnetworks, indicating that the MSE cannot always be reduced by adding subnetworks. The number of subnetworks should be selected cautiously to achieve a reasonable compression since the suboptimality of one network will affect the performance of the following networks, leading to poor overall accuracy. That may also be why the MSE of SFC-SAE consists of 2 subnetworks is larger than the MSE of SFC-CAE when compressing the variables to 8. In addition, the two SFCs generated simultaneously could capture features in two different directions at a given node in the mesh, leading to the two-SFCs based networks perform better than the one-SFC based networks.

The dimensionality reduction effect of each method can be seen in figure 6 and 7. In figure 6, compared with the POD and SVD-autoencoder, although part of the fluid reconstructed by the SFC-autoencoders is blurry, they still retain the basic features of the FPC, which is acceptable. Moreover, the fluid reconstructed by the SFC-HAE and SFC-SAE is clearer than SFC-CAE's. A more obvious comparison can be seen in figure 7. For the velocity of a fixed point on the vtu file, the difference between the velocity of the reconstructed data and the original velocity is ranked from largest to smallest as SFC-CAE, SFC-HAE, SFC-SAE, SVD-AE and POD, which conforms to the MSE results in the table 4.

The computation time for different methods applied to flow past cylinder is shown in table 3(b). Because the SFC-HAE and SFC-SAE need to train 2 subnetworks respectively, the time spent is approximately twice SFC-CAE's. Although 2SFC based network could further improve the accuracy, it comes at the cost of a long training time. Compared with POD and SVD-autoencoder, other methods are time-consuming. When the number of the latent variable is more than 32, both the POD and SVD-autoencoder show better performance in dimensionality reduction and execution time than the other methods, making POD and SVD-autoencoder better candidates. Moreover, due to the accuracy, the SFC-autoencoders are preferred to perform dimensionality reduction when the number of latent variables is smaller than 32.

3.3 Documentation and Testing

Documentation files are generated with $Sphinx$ and upload to the doc directory on the Github repository.

The unit testing for the dimensionality reduction methods is difficult. The reason is that the important prerequisite for unit testing is repeatability and stability. However, the output of these methods is unstable and random. Therefore, the results of each method are generated to judge whether this method is passed the test through observing the reconstructed data.

	SFC-CAE	SFC-HAE	SFC-SAE	SVD-AE	POD	compression
Training data	2.571×10^{-3}			1.055×10^{-2}	2.107×10^{-2}	
Validation data	3.263×10^{-3}			9.996×10^{-3}	2.082×10^{-2}	1
Test data	2.767×10^{-3}			1.170×10^{-2}	2.130×10^{-2}	
Training data	1.715×10^{-3}	1.878×10^{-3}	1.680×10^{-3}	3.336×10^{-3}	1.154×10^{-2}	
Validation data	1.627×10^{-3}	1.847×10^{-3}	2.383×10^{-3}	3.293×10^{-3}	1.087×10^{-2}	2
Test data	1.519×10^{-3}	1.768×10^{-3}	2.043×10^{-3}	3.089×10^{-3}	1.239×10^{-2}	
Training data	1.800×10^{-3}	7.673×10^{-4}	9.525×10^{-4}	2.651×10^{-3}	2.778×10^{-3}	
Validation data	1.679×10^{-3}	7.232×10^{-4}	1.036×10^{-3}	2.628×10^{-3}	2.770×10^{-3}	4
Test data	1.606×10^{-3}	7.137×10^{-4}	9.900×10^{-4}	2.501×10^{-3}	2.590×10^{-3}	
Training data	7.751×10^{-4}	6.950×10^{-4}	9.130×10^{-4}	1.507×10^{-3}	1.631×10^{-3}	
Validation data	7.296×10^{-4}	6.616×10^{-4}	9.490×10^{-4}	1.465×10^{-3}	1.568×10^{-3}	8
Test data	7.196×10^{-4}	6.488×10^{-4}	8.842×10^{-4}	1.421×10^{-3}	1.423×10^{-3}	
Training data	5.865×10^{-4}	4.614×10^{-4}	4.487×10^{-4}	6.536×10^{-4}	7.155×10^{-4}	
Validation data	5.631×10^{-4}	4.432×10^{-4}	4.488×10^{-4}	6.951×10^{-4}	6.605×10^{-4}	16
Test data	5.540×10^{-4}	4.356×10^{-4}	4.299×10^{-4}	6.727×10^{-4}	6.523×10^{-4}	
Training data	4.398×10^{-4}	3.605×10^{-4}	3.023×10^{-4}	2.455×10^{-4}	2.688×10^{-4}	
Validation data	4.301×10^{-4}	3.528×10^{-4}	2.984×10^{-4}	3.297×10^{-4}	2.628×10^{-4}	32
Test data	4.220×10^{-4}	3.444×10^{-4}	2.899×10^{-4}	3.184×10^{-4}	2.591×10^{-4}	
Training data	3.380×10^{-4}	2.709×10^{-4}	2.080×10^{-4}	7.811×10^{-5}	7.061×10^{-5}	
Validation data	3.331×10^{-4}	2.651×10^{-4}	2.101×10^{-4}	1.867×10^{-4}	8.063×10^{-5}	64
Test data	3.248×10^{-4}	2.596×10^{-4}	2.017×10^{-4}	1.611×10^{-4}	7.631×10^{-5}	
Training data	3.040×10^{-4}	2.332×10^{-4}	1.461×10^{-4}	3.236×10^{-5}	1.209×10^{-5}	
Validation data	2.970×10^{-4}	2.268×10^{-4}	1.467×10^{-4}	1.103×10^{-4}	1.710×10^{-5}	128
Test data	2.926×10^{-4}	2.218×10^{-4}	1.410×10^{-4}	1.066×10^{-4}	1.641×10^{-5}	
Training data	2.989×10^{-4}	2.258×10^{-4}	1.151×10^{-4}	2.463×10^{-5}	1.513×10^{-6}	
Validation data	2.910×10^{-4}	2.203×10^{-4}	1.145×10^{-4}	1.022×10^{-4}	3.390×10^{-6}	256
Test data	2.878×10^{-4}	2.135×10^{-4}	1.108×10^{-4}	1.001×10^{-4}	3.227×10^{-6}	

Table 4: The MSE of present dimensionality reduction methods, trained to compress to the following number of variables: 1, 2, 4, 8, 16, 32, 64, 128, 256. The dataset is the flow past cylinder. (Note that the lowest MSE values in test data are highlighted in grey)

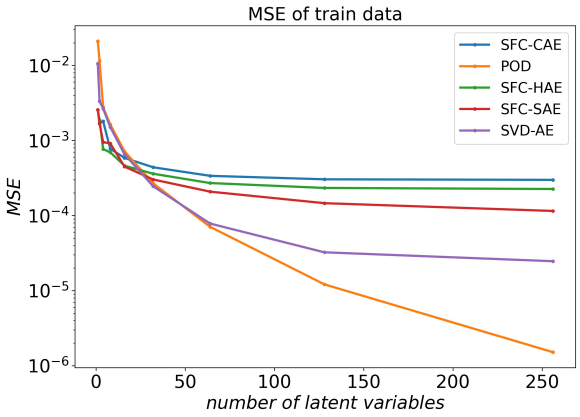
(a)

		2	4	8	16
SFC-HAE	Training data	1.878×10^{-3}	7.673×10^{-4}	6.950×10^{-4}	4.614×10^{-4}
	Validation data	1.847×10^{-3}	7.232×10^{-4}	6.616×10^{-4}	4.432×10^{-4}
	Test data	1.768×10^{-3}	7.137×10^{-4}	6.488×10^{-4}	4.356×10^{-4}
2SFC-2HAE	Training data	1.795×10^{-3}	7.957×10^{-4}	7.781×10^{-4}	4.841×10^{-4}
	Validation data	2.524×10^{-3}	7.494×10^{-4}	7.394×10^{-4}	4.632×10^{-4}
	Test data	2.042×10^{-3}	7.627×10^{-4}	7.328×10^{-4}	4.557×10^{-4}
		32	64	128	256
SFC-HAE	Training data	3.605×10^{-4}	2.709×10^{-4}	2.332×10^{-4}	2.258×10^{-4}
	Validation data	3.528×10^{-4}	2.651×10^{-4}	2.268×10^{-4}	2.203×10^{-4}
	Test data	3.444×10^{-4}	2.596×10^{-4}	2.218×10^{-4}	2.135×10^{-4}
2SFC-2HAE	Training data	3.442×10^{-4}	2.713×10^{-4}	2.332×10^{-4}	2.169×10^{-4}
	Validation data	3.360×10^{-4}	2.655×10^{-4}	2.267×10^{-4}	2.106×10^{-4}
	Test data	3.261×10^{-4}	2.611×10^{-4}	2.228×10^{-4}	2.059×10^{-4}

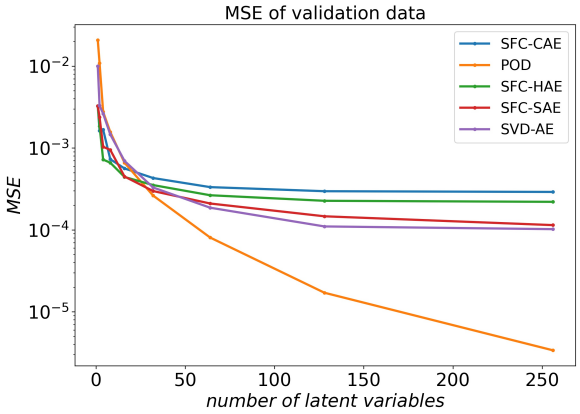
(b)

	SFC-CAE	2SFC-CAE	SFC-HAE	SFC-SAE
Training data	3.013×10^{-4}	1.819×10^{-4}	2.258×10^{-4}	1.151×10^{-4}
Validation data	2.930×10^{-4}	1.775×10^{-4}	2.203×10^{-4}	1.145×10^{-4}
Test data	2.894×10^{-4}	1.747×10^{-4}	2.135×10^{-4}	1.108×10^{-4}
	SFC-3HAE	2SFC-2HAE	2SFC-HAE	2SFC-SAE
Training data	2.365×10^{-4}	2.169×10^{-4}	1.365×10^{-4}	7.437×10^{-5}
Validation data	2.309×10^{-4}	2.106×10^{-4}	1.348×10^{-4}	7.578×10^{-5}
Test data	2.263×10^{-4}	2.059×10^{-4}	1.317×10^{-4}	7.259×10^{-5}

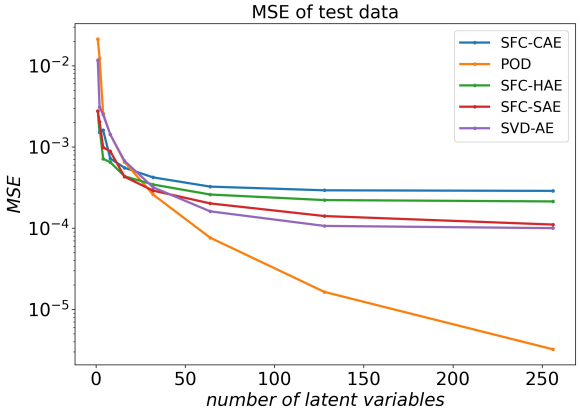
Table 5: (a) The comparison between SFC-HAE and 2SFC-2HAE in terms of MSE. Both are trained to compress to the following numbers of variables: 1, 2, 4, 8, 16, 32, 64, 128, 256. The dataset is the flow past cylinder. The two SFCs are generated respectively and each subnetwork will only use one SFC. (Note that the lowest MSE values in test data are highlighted in grey); (b) The comparison of different SFC autoencoders in terms of MSE. The methods in the table are trained to compress to 256. 2SFC-2HAE means the hierarchical autoencoder consists of two subnetworks that use two different SFC. 2SFC-HAE means the hierarchical autoencoder consists of two subnetworks and each subnetwork use two SFCs.



(a)



(b)



(c)

Figure 5: The MSE of present dimensionality reduction methods in reconstructing the training, validation and test samples of flow past cylinder. The x-axis represents the number of latent variables in the reconstruction, and the y-axis represents the MSE of corresponding data; (a) The MSE of present dimensionality reduction methods in training data; (b) The MSE of present dimensionality reduction methods in validation data; (c) The MSE of present dimensionality reduction methods in test data.

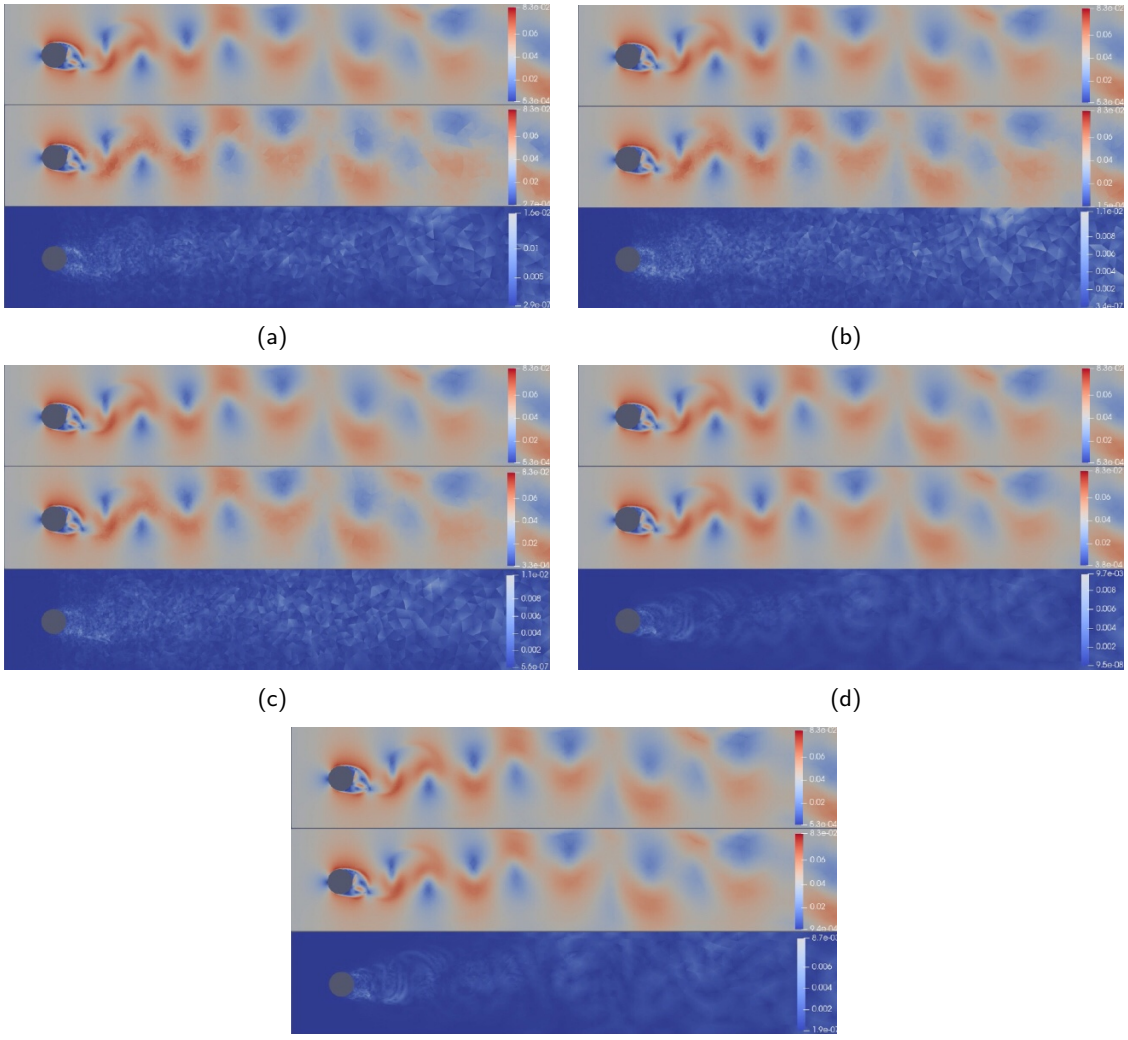


Figure 6: Comparison of results of present methods to compress to 64 latent variables. The example (t=1908) is from test data. Each set of plots is divided into: velocity of the original data (top); velocity of the reconstructed data by corresponding dimensionality reduction method (middle); pointwise error in velocity (bottom). (a) SFC-CAE; (b) SFC-HAE; (c) SFC-SAE; (d) SVD-AE; (e) POD.

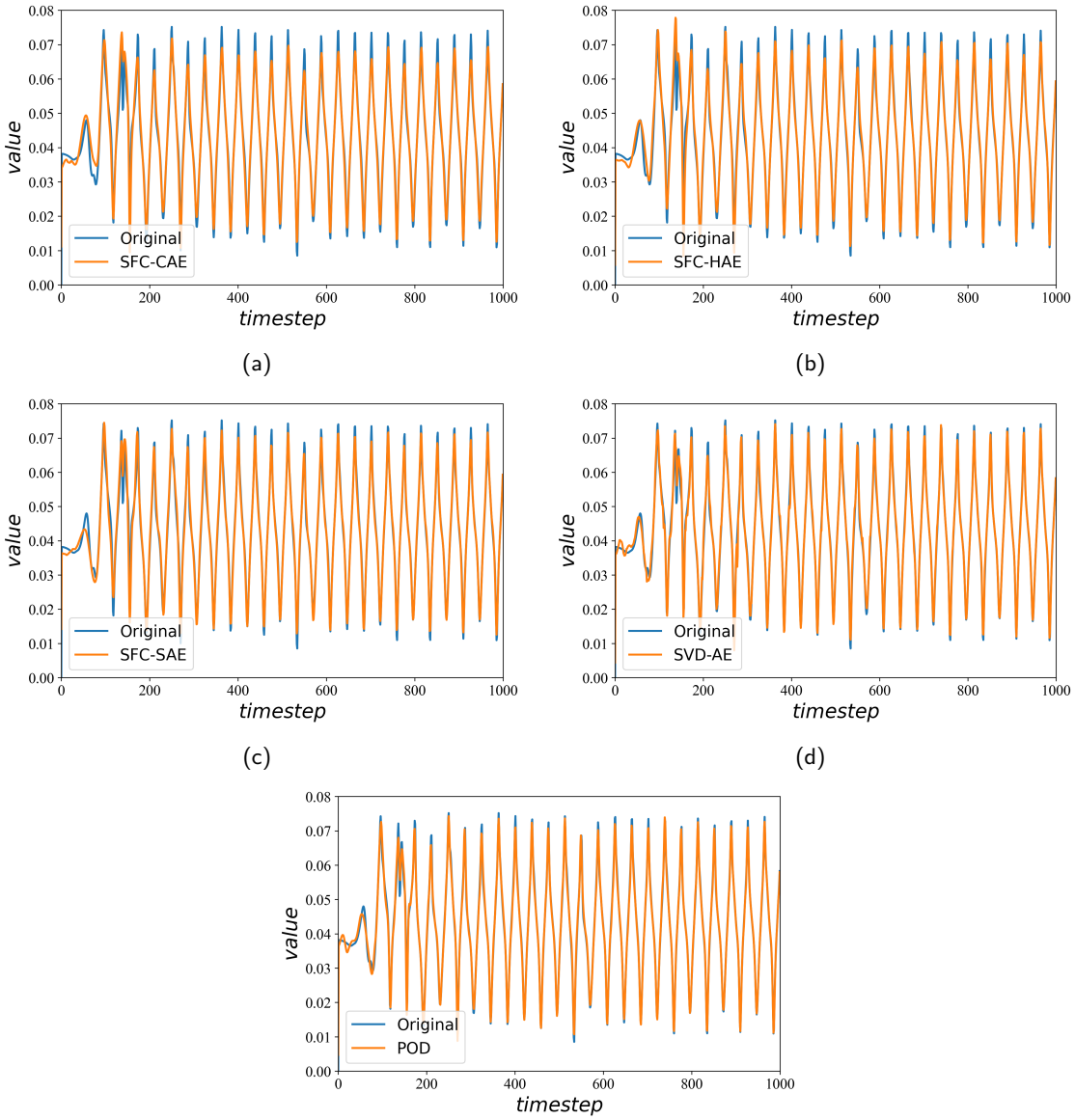


Figure 7: The velocity of flow past cylinder at (0.4,0.15) is reconstructed by present methods. The data was compressed to 64 latent variables. The first 1000 time steps are chosen. (a) SFC-CAE; (b) SFC-HAE; (c) SFC-SAE; (d) SVD-AE; (e) POD.

4 Discussion and Conclusion

4.1 Discussion

Overall, multiple dimensionality reduction methods have been applied to Burgers' equation solution and flow past cylinder successfully, and the result of the comparison based on dimensionality reduction capabilities have been reported in tables and figures. Based on the above results, for both burgers equation and flow past cylinder solutions, the autoencoder performs better than POD when using small latent variables 2. The MSE obtained from autoencoders is at least one order of magnitude lower than POD's. However, compared with the POD, the advantages of autoencoder gradually diminish as the number of latent variables increase. POD demonstrates good performance in compressing to a large number of latent variables. SVD-autoencoder seems to be a compromise for POD and autoencoder. In these two test cases, when using a small number of latent variables, SVD-autoencoder performs worse than autoencoder and when using a large number of latent variables, it performs worse than POD. However, compared with the autoencoder, SVD-autoencoder is more time-efficient when dealing with data of large spatial degrees of freedom. The SFC-CAE perform well to deal with unstructured data. However, before training the autoencoder, the number of the principal component need to be determined in advance. Due to the hierarchy manner, for both hierarchical autoencoder and sequential autoencoder, the user can determine the number of principal components after the adjustment of the architecture of the autoencoder. Additionally, When the new principal component is added, the re-training of the whole network is not needed, instead of training the additional subnetworks. Moreover, although both the SFC-HAE and SFC-SAE seems to be better compared with SFC-CAE, there is no substantial evidence to prove that the hierarchical or sequential autoencoder consist of multiple subnetworks should always be better than the single network. As stated in section 3.2.3, the number of the subnetworks should be selected cautiously to achieve a reasonable compression because the suboptimality of one network leads to poor overall accuracy. In brief, for a specific problem that compress the data to a fixed number of principal components, there is no applicable method for determining the number of the subnetwork or guidance to design the architecture of the subnetwork. Both the number of the subnetworks and the architecture of the subnetwork should be explored by the user.

The most challenging task is to find the appropriate architecture of the subnetwork in hierarchical autoencoder and sequential autoencoder. Network training is insipid because different networks need to be trained in an ordered manner. In addition, due to the limitation of the GPU, the output of the 2SFCs autoencoders need to be generated respectively, then calculate the MSE separately. The solution may be not elegant, but fortunately, the final result is not affected.

4.2 Conclusion

This paper provides a comparative study of different dimensionality reduction methods applied to the fluid data of burgers equation and flow past cylinder, and results are presented and discussed. Through generating specified fluid data, normalising data, adopting different dimensionality reduction methods, reconstructing data from reduced variables, the dimensionality reduction capability of corresponding methods is assessed by the MSE. The dynamic display of reconstructed data by different dimensionality reduction methods are provided in the GitHub repository.

With more time, the multiple dimensionality reduction methods running on the high-performance computing system(HPC) can be used to process a massive amount of high-dimensional data arising from high-resolution 3D solutions to fluid flow problems. In addition, when dealing with data on unstructured meshes, only the SFC that maps the multi-dimensional space into the one-dimensional space representation is adopted. In the following study, the dimensionality reduction capability of convolutional autoencoder, hierarchical autoencoder and sequential autoencoder in combination with various methods for processing unstructured data such as interpolation (Löhner 1995), graph neural

networks (Wu et al. 2020) is also worth investigating. In addition to the methods dealing with unstructured data, the subnetworks of hierarchical autoencoder and sequential autoencoder can be optimised as well. In this project, both consist of the convolutional autoencoder. In the following study, whether the hierarchical autoencoder and sequential autoencoder consisting of different types of autoencoder such as fully-connected, variational, recurrent further improve the dimensionality reduction capability is an exciting topic to discuss and research. Therefore, this study proposes a preliminary comparison of specifying dimensionality reduction methods applying to fluid data of burgers equation and flow past cylinder, whose the type of dimensionality reduction methods and the type of fluid data could be optionally expanded in the future depending on interest.

Acknowledgements

First of all, I would like to express my gratitude and appreciation for my supervisors: Dr Claire Heaney and Prof. Christopher Pain. Thanks to their consistent support and patient guidance during the running of this project, I was able to complete the tasks of the independent research project successfully. Furthermore, I would like to thank Dr Marijan Beg and Dr Adriana Paluszny, who give me careful assistance in writing the project plan and project report. Then, I also wish to thank my friends in the autoencoders group, who gave me many coding suggestions. Finally, I would like to thank my family for supporting me during the compilation of this project.

References

Abel, D. J. & Mark, D. M. (1990), 'A comparative analysis of some two-dimensional orderings', *International Journal of Geographical Information Systems* **4**(1), 21–31.

Ahrens, J., Geveci, B. & Law, C. (2005), 'Paraview: An end-user tool for large data visualization', *The visualization handbook* **717**(8).

Alfonsi, G. & Primavera, L. (2007), 'The structure of turbulent boundary layers in the wall region of plane channel flow', *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **463**(2078), 593–612.

Baldi, P. & Hornik, K. (1989), 'Neural networks and principal component analysis: Learning from examples without local minima', *Neural networks* **2**(1), 53–58.

Brunton, S. L., Noack, B. R. & Koumoutsakos, P. (2020), 'Machine learning for fluid mechanics', *Annual Review of Fluid Mechanics* **52**, 477–508.

Casas, C. Q., Arcucci, R. & Guo, Y. (2020), Urban air pollution forecasts generated from latent space representation, *in* 'ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations'.

Fukami, K., Nakamura, T. & Fukagata, K. (2020), 'Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data', *Physics of Fluids* **32**(9), 095110.

Fukunaga, K. & Koontz, W. L. (1970), 'Application of the karhunen-loeve expansion to feature selection and ordering', *IEEE Transactions on computers* **100**(4), 311–318.

Glorot, X. & Bengio, Y. (2010), Understanding the difficulty of training deep feedforward neural networks, *in* 'Proceedings of the thirteenth international conference on artificial intelligence and statistics', JMLR Workshop and Conference Proceedings, pp. 249–256.

Hasegawa, K., Fukami, K., Murata, T. & Fukagata, K. (2020*a*), 'Cnn-lstm based reduced order modeling of two-dimensional unsteady flows around a circular cylinder at different reynolds numbers', *Fluid Dynamics Research* **52**(6), 065501.

Hasegawa, K., Fukami, K., Murata, T. & Fukagata, K. (2020*b*), 'Machine-learning-based reduced-order modeling for unsteady flows around bluff bodies of various shapes', *Theoretical and Computational Fluid Dynamics* **34**(4), 367–383.

Hastie, T. & Stuetzle, W. (1989), 'Principal curves', *Journal of the American Statistical Association* **84**(406), 502–516.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *in* 'Proceedings of the IEEE international conference on computer vision', pp. 1026–1034.

Heaney, C., Li, Y., Matar, O. & Pain, C. (2020), 'Applying convolutional neural networks to data on unstructured meshes with space-filling curves', *arXiv preprint arXiv:2011.14820* .

Hornik, K., Stinchcombe, M. & White, H. (1989), 'Multilayer feedforward networks are universal approximators', *Neural networks* **2**(5), 359–366.

Irie, B. & Kawato, M. (1991), 'Acquisition of internal representation by multilayered perceptrons', *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* **74**(11), 112–118.

Jiang, C., Huang, J., Kashinath, K., Marcus, P., Niessner, M. et al. (2019), 'Spherical cnns on unstructured grids', *arXiv preprint arXiv:1901.02039* .

Jiang, C. M., Kashinath, K., Mudigonda, M., Mahesh, A., O'Brien, T. A., Marcus, P. S. & Prabhat, M. (2018), Deep learning on the sphere: Convolutional neural network on unstructured mesh, *in* 'AGU Fall Meeting Abstracts', Vol. 2018, pp. IN33A–01.

Jimenez, L. O. & Landgrebe, D. A. (1998), 'Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* **28**(1), 39–54.

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization', *arXiv preprint arXiv:1412.6980* .

Kramer, M. A. (1991), 'Nonlinear principal component analysis using autoassociative neural networks', *AIChE journal* **37**(2), 233–243.

Löhner, R. (1995), 'Robust, vectorized search algorithms for interpolation on unstructured grids', *Journal of computational Physics* **118**(2), 380–387.

Lumley, J. L. (1967), 'The structure of inhomogeneous turbulent flows', *Atmospheric turbulence and radio wave propagation* .

Mall, R. (2018), *Fundamentals of software engineering*, PHI Learning Pvt. Ltd.

Milano, M. & Koumoutsakos, P. (2002), 'Neural network modeling for near wall turbulent flow', *Journal of Computational Physics* **182**(1), 1–26.

Moon, B., Jagadish, H. V., Faloutsos, C. & Saltz, J. H. (2001), 'Analysis of the clustering properties of the hilbert space-filling curve', *IEEE Transactions on knowledge and data engineering* **13**(1), 124–141.

Moore, E. H. (1900), 'On certain crinkly curves', *Transactions of the American Mathematical Society* **1**(1), 72–90.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L. et al. (2019), 'Pytorch: An imperative style, high-performance deep learning library', *Advances in neural information processing systems* **32**, 8026–8037.

Phillips, T., Heaney, C. E., Smith, P. N. & Pain, C. C. (2020), 'An autoencoder-based reduced-order model for eigenvalue problems with application to neutron diffusion', *arXiv preprint arXiv:2008.10532* .

Saegusa, R., Sakano, H. & Hashimoto, S. (2004), 'Nonlinear principal component analysis to preserve the order of principal components', *Neurocomputing* **61**, 57–70.

Sagan, H. (2012), *Space-filling curves*, Springer Science & Business Media.

Schölkopf, B., Smola, A. & Müller, K.-R. (1998), 'Nonlinear component analysis as a kernel eigenvalue problem', *Neural computation* **10**(5), 1299–1319.

Singh, D. & Singh, B. (2020), 'Investigating the impact of data normalization on classification performance', *Applied Soft Computing* **97**, 105524.

Sirovich, L. (1987), 'Turbulence and the dynamics of coherent structures. i. coherent structures', *Quarterly of applied mathematics* **45**(3), 561–571.

Tencer, J. & Potter, K. (2020), 'A tailored convolutional neural network for nonlinear manifold learning of computational physics data using unstructured spatial discretizations', *arXiv preprint arXiv:2006.06154* .

Van Der Maaten, L., Postma, E. & Van den Herik, J. (2009), 'Dimensionality reduction: a comparative review', *J Mach Learn Res* **10**(66-71), 13.

Walton, S., Hassan, O. & Morgan, K. (2017), 'Advances in co-volume mesh generation and mesh optimisation techniques', *Computers & Structures* **181**, 70–88.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C. & Philip, S. Y. (2020), 'A comprehensive survey on graph neural networks', *IEEE transactions on neural networks and learning systems* **32**(1), 4–24.

Xie, Z., Pavlidis, D., Salinas, P., Percival, J. R., Pain, C. C. & Matar, O. K. (2016), 'A balanced-force control volume finite element method for interfacial flows with surface tension using adaptive anisotropic unstructured meshes', *Computers & Fluids* **138**, 38–50.

Xu, M., Song, S., Sun, X. & Zhang, W. (2021), 'Ucnn: A convolutional strategy on unstructured mesh', *arXiv preprint arXiv:2101.05207* .

5 Appendices

5.1 The information about dimensionality reduction methods and fluid data

Table 6 shows the detailed information of dimensionality reduction methods and fluid data used in this project.

5.2 Hyper parameters used in present models

Table 7, 8 shows the hyperparameters of models used in this project.

5.3 Architectures of the models

5.3.1 Architectures of the models in burgers equation

Table 9, 10, 11 and 12 are the architectures of the autoencoder used in burgers equation.

	Burgers equation(1D)	Flow past cylinder(2D)
POD/PCA	y	y
FC-AE	y	/
SVD-AE	y	y
CAE	y	/
SFC-CAE	n	y
Hierarchical AE	y	y
Sequential AE	n	y

Table 6: The information about dimensionality reduction methods and fluid data

Parameter	Value	Parameter	Value
optimiser	Adam	Loss	MSE
β_1 of Adam	0.9	β_2 of Adam	0.999
Fully-connected autoencoder			
Batch size	32	Learning rate	1e-4
Epochs	5000	activation function	ReLU
Convolutional autoecnoder			
Batch size	16	Learning rate	1e-4
Epochs	3000	activation function	ReLU
SVD-autoencoder			
Batch size	32	Learning rate	1e-4
Epochs	30000	activation function	Sigmoid
Hierarchical autoencoder			
Batch size	16	Learning rate	1e-4
Epochs	3000	activation function	ReLU

Table 7: Hyperparameters used in burgers equation

Parameter	Value	Parameter	Value
optimiser	Adam	Loss	MSE
β_1 of Adam	0.9	β_2 of Adam	0.999
SFC-convolutional autoecnoder			
Batch size	16	Learning rate	1e-4
Epochs	2000	activation function	tanh
Hierarchical autoencoder			
Batch size	16	Learning rate	1e-4
Epochs	2000	activation function	tanh
Sequential autoencoder			
Batch size	16	Learning rate	1e-4
Epochs	2000	activation function	tanh
SVD-autoencoder			
Batch size	16	Learning rate	1e-3
Epochs	2000	activation function	LeakyReLU

Table 8: Hyperparameters used in flow past cylinder

layers	input size	output size	activation
1-FC	200	latent variable	ReLU
2-FC	latent variable	200	ReLU

Table 9: The architecture of fully-connected autoencoder in burgers equation. The number of the latent variable can be adjusted from 1 to any proper value.

layers	input size	output size	activation
Take matrix R^T of SVD to compress the nodes from 200 to 150			
1-FC	150	latent variable	Sigmoid
2-FC	latent variable	150	Sigmoid
Take matrix R of SVD to reconstruct the nodes from 150 to 200			

Table 10: The architecture of SVD-autoencoder in burgers equation. The number of the latent variable can be adjusted from 1 to any proper value.

layers	input size	kernel size	channels	stride	padding	output size	activation
Encoder							
1- Conv1d	(1,200)	3	8	3	15	(8,76)	ReLU
2-Conv1d	(8,76)	3	16	3	15	(16,35)	ReLU
3-Conv1d	(16,35)	3	16	3	15	(16,21)	ReLU
Flatten the data to 1D, then use fully-connected layers to compress further							
4-FC	336					Latent variable	ReLU
Decoder							
5-FC	Latent variable					336	ReLU
Reshape 336 to (16,21)							
6-ConvTranspose1d	(16,21)	3	16	3	14	(16,35)	ReLU
7-ConvTranspose1d	(16,35)	4	8	3	15	(8,76)	ReLU
8-ConvTranspose1d	(8,76)	3	1	3	14	(1,200)	ReLU

Table 11: The architecture of convolutional autoencoder in burgers equation. This is also the first subnetwork of hierarchical autoencoder. The number of the latent variable can be adjusted from 1 to any proper value.

layers	input size	kernel size	channels	stride	padding	output size	activation
Encoder							
1- Conv1d	(1,200)	3	16	2	5	(16,104)	ReLU
2-Conv1d	(16,104)	3	16	2	5	(16,56)	ReLU
3-Conv1d	(16,56)	3	16	2	5	(16,32)	ReLU
Flatten the data to 1D, then use fully-connected layers to compress further							
4-FC	512					Latent variable	ReLU
Decoder							
5-FC	Latent variable					512	ReLU
Reshape 512 to (16,32)							
6-ConvTranspose1d	(16,32)	4	16	2	5	(16,56)	ReLU
7-ConvTranspose1d	(16,56)	4	16	2	5	(16,104)	ReLU
8-ConvTranspose1d	(16,104)	4	1	2	5	(1,200)	ReLU

Table 12: The second subnetwork of hierarchical autoencoder in burgers equation. The number of the latent variable can be adjusted from 1 to any proper value.

layers	input size	output size	activation
Take matrix R^T of SVD to compress the nodes from 41100 to 1600			
1-FC	1600	latent variable	LeakyReLU
2-FC	latent variable	1600	LeakyReLU
Take matrix R of SVD to reconstruct the nodes from 1600 to 41100			

Table 13: The architecture of SVD-autoencoder in flow past cylinder. The number of the latent variable can be adjusted from 1 to any proper value.

layers	input size	kernel size	channels	stride	padding	output size	activation
1-FEM	(2, 20550, FEM)	1 Identity	2	1	0	(2, 20550, SFC1)	Identity
Apply SFC ordering, split (2, 20550, SFC1) into two:(1, 20550, SFC1u), (1, 20550, SFC1v)							
2-SFC1u	(1, 20550, SFC1u)	3 Variable (3 × 20550)	2	1	0	(2, 20550, SFC1u)	tanh
2-SFC1v	(1, 20550, SFC1v)	3 Variable (3 × 20550)	2	1	0	(2, 20550, SFC1v)	tanh
concatenate:((2, 20550, SFC1u), (2, 20550, SFC1v)) forming (4, 20550, SFC1)							
Encoder							
3-Conv1d-SFC1	(4, 20550, SFC1)	16	8	4	9	(8, 5139, SFC1)	tanh
4-Conv1d-SFC1	(8, 5139, SFC1)	16	8	4	9	(8, 1286, SFC1)	tanh
5-Conv1d-SFC1	(8, 1286, SFC1)	16	16	4	9	(16, 323, SFC1)	tanh
6-Conv1d-SFC1	(16, 323, SFC1)	16	16	4	9	(16, 82, SFC1)	tanh
Flatten the output data of layer 6 to 1D							
7-FC	1312					Latent variable	tanh
Decoder							
8-FC	Latent variable					1312	tanh
Reshape (1312,1,SFC1) to (16, 82, SFC1)							
9-ConvTranspose1d-SFC1	(16, 82, SFC1)	17	16	4	9	(16, 323, SFC1)	tanh
10-ConvTranspose1d-SFC1	(16, 323, SFC1)	16	8	4	9	(8, 1286, SFC1)	tanh
11-ConvTranspose1d-SFC1	(8, 1286, SFC1)	17	16	8	9	(8, 5139, SFC1)	tanh
12-ConvTranspose1d-SFC1	(8, 5139, SFC1)	16	16	4	9	(4, 20550, SFC1)	tanh
Apply inverse SFC ordering, split (4, 20550, SFC1) into two:(2, 20550, SFC1u), (2, 20550, SFC1v)							
13-SFC1u	(2, 20550, SFC1u)	3 Variable (3 × 20550)	2	1	0	(1, 20550, SFC1u)	Identity
13-SFC1v	(2, 20550, SFC1v)	3 Variable (3 × 20550)	2	1	0	(1 20550, SFC1v)	Identity
14-SFC1u	(1, 20550, SFC1u)	1 Identity	1	1	0	(1, 20550, FEM1u)	Identity
14-SFC1v	(1, 20550, SFC1v)	1 Identity	1	1	0	(1, 20550, FEM1v)	Identity
15-FEM	(1 × 2, 20550, FEM)	1 Identity concat(FEM1u,FEM1v)	2	1	0	(2, 20550, FEM)	tanh

Table 14: The architecture of SFC-CAE with 1 SFC orderings in flow past cylinder. '3 Variable' would give the nearest-neighbour smoothing. This is also the first subnetwork of SFC-HAE and SFC-SAE. The number of the latent variable can be adjusted from 1 to any proper value.

5.3.2 Architectures of the models in flow past cylinder

Table 13, 14, 15 are the architectures of the autoencoder used in flow past cylinder.

layers	input size	kernel size	channels	stride	padding	output size	activation
1-FEM	(2, 20550, FEM)	1 Identity	2	1	0	(2, 20550, SFC1)	Identity
Apply SFC ordering, split (2, 20550, SFC1) into two:(1, 20550, SFC1u), (1, 20550, SFC1v)							
2-SFC1u	(1, 20550, SFC1u)	3 Variable (3×20550)	2	1	0	(2, 20550, SFC1u)	tanh
2-SFC1v	(1, 20550, SFC1v)	3 Variable (3×20550)	2	1	0	(2, 20550, SFC1v)	tanh
concatenate:((2, 20550, SFC1u), (2, 20550, SFC1v)) forming (4, 20550, SFC1)							
Encoder							
3-Conv1d-SFC1	(4, 20550, SFC1)	32	16	4	16	(16, 5138, SFC1)	tanh
4-Conv1d-SFC1	(16, 5138, SFC1)	32	16	4	16	(16, 1285, SFC1)	tanh
5-Conv1d-SFC1	(16, 1285, SFC1)	32	16	4	16	(16, 322, SFC1)	tanh
6-Conv1d-SFC1	(16, 322, SFC1)	32	16	4	16	(16, 81, SFC1)	tanh
Flatten the output data of layer 6 to 1D							
7-FC	1296					Latent variable	tanh
Decoder							
8-FC	Latent variable					1296	tanh
Reshape (1296,1,SFC1) to (16, 81, SFC1)							
9-ConvTranspose1d-SFC1	(16, 81, SFC1)	32	16	4	15	(16, 322, SFC1)	tanh
10-ConvTranspose1d-SFC1	(16, 322, SFC1)	32	16	4	15	(16, 1286, SFC1)	tanh
11-ConvTranspose1d-SFC1	(16, 1286, SFC1)	32	16	4	16	(16, 5140, SFC1)	tanh
12-ConvTranspose1d-SFC1	(16, 5140, SFC1)	32	16	4	19	(4, 20550, SFC1)	tanh
Apply inverse SFC ordering, split (4, 20550, SFC1) into two:(2, 20550, SFC1u), (2, 20550, SFC1v)							
13-SFC1u	(2, 20550, SFC1u)	3 Variable (3×20550)	2	1	0	(1, 20550, SFC1u)	Identity
13-SFC1v	(2, 20550, SFC1v)	3 Variable (3×20550)	2	1	0	(1 20550, SFC1v)	Identity
14-SFC1u	(1, 20550, SFC1u)	1 Identity	1	1	0	(1, 20550, FEM1u)	Identity
14-SFC1v	(1, 20550, SFC1v)	1 Identity	1	1	0	(1, 20550, FEM1v)	Identity
15-FEM	(1×2, 20550, FEM)	1 Identity concat(FEM1u,FEM1v)	2	1	0	(2, 20550, FEM)	tanh

Table 15: The architecture of the second subnetwork of SFC-HAE and SFC-SAE in flow past cylinder. '3 Variable' would give the nearest-neighbour smoothing. The number of the latent variable can be adjusted from 1 to any proper value.