A convolutional auto-encoder for reduced order models on unstructured meshes

Author : Jiaye Mao
Supervisors : Christopher Pain, Claire Heaney

Department of Earth Science and Engineering, Imperial College London

Abstract

In this project, I develop a new convolutional auto-encoder for compressing the number of fluid flow variables solved for on unstructured finite element or control-volume meshes. This is developed within the OPAL toolbox. OPAL is an open-source software written in Python, used for Reduced Order Modeling of fluid flows and geothermal applications. It provides a set of compression and predicting methods which are needed in order to construct Reduced Order Models (ROMs). This work implements into OPAL (for the first time) a non-linear machine learning method for compressing fluid flow solution variables. In some cases, the convolutional auto-encoder in Reduced Order Modelling shows better performance than traditional methods such as POD. The ultimate aim is to reduce the number of solution variables down from many millions, that are solved for each time step, to perhaps a few hundred. Furthermore to perform this reduction in variables on an unstructured mesh. We use interpolation methods to interpolate unstructured mesh solution variables onto structured meshes so that it is possible to apply standard convolution methods designed for compressing images. In addition, we apply domain decomposition methods in which we split the solution domain into several regions or subdomains with similar resolution within each subdomain. This enables the auto-encoder to be applied at a number of different length scales. We demonstrate our model on some illustrative examples, each, highlighting the models performance in the prediction of flows.

Keywords:
Reduced order modelling (ROM), Unstructured mesh, Auto-encoder, Domain decomposition

1. Introduction and objective

Reduced order models capitalise on dimensionality reduction which can lead to a computational acceleration of many orders of magnitude resulting in a model that can be efficient in multi-query problems (such as optimisation) and real-time calculations (such as predictive control). Proper orthogonal decomposition (POD) is one way to reduce the dimensionality, which has been broadly applied in engineering fields, including image processing, signal analysis [1]. However, this method cannot easily be used to construct a non-linear manifold [2]. Deep learning technology has flourished in recent years because of the advancement in computational power, and it is capable of finding more hidden correlation within the data than traditional methods [3]. A further advantage of auto-encoders (one such deep learning method) is their ability to produce non-linear representations that are position invariant.

The first step is using an existing High Fidelity Model (HFM) based on discretizations of the governing equations to generate snapshots that taken at different time levels of the HFM. This process includes solving the advection equation or the

Navier-Stokes equations for fluid flows using the finite element method:

Continuity Equation:-

$$\nabla \cdot \mathbf{v} = 0 \tag{1}$$

Momentum Equation:-

$$\rho\Big(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v} \cdot \nabla)\mathbf{v}\Big) = -\rho \mathbf{g} h - \nabla p + \nabla \cdot \overline{\overline{\tau}} \tag{2}$$

in which $\mathbf{v} = (u, v, w)^T$ is the velocity with 3D velocity components and in 2D $\mathbf{v} = (u, v)^T$. The viscosity was set to $0.01Pa \cdot s$ and the uniform inlet velocity (on the left) was set to unity which results in laminar flow with a Reynolds number of 100. In the above $\nabla \cdot \overline{\overline{\tau}}$ contains the stress terms with the above viscosity.

There are two main challenges associated with this project. The first challenge is how to apply current convolutional neural networks, that are based on structured representations used in image analysis, to unstructured meshes. Unstructured meshes are irregular tessellations (often of simplices) used in finite element analysis. Current convolutional networks cannot be applied straightforwardly to unstructured meshes because of their irregular connectivity. The simplest approach is to interpolate the variables from an unstructured mesh to a structured mesh. This results in a fast algorithm for training that can exploit the advanced codes such as tensorflow that are designed for regular mesh image analysis. Image processing is also a mature field in machine learning.

The second difficulty is how to solve the computational issue associated with the uneven density of resolution or nodes/cells in some unstructured meshes. For many problems ROM is applied to, there will be areas with different densities of nodes. If we interpolate the unstructured mesh solution variables onto a uniform structured mesh for the convolutional auto-encoder, then some details in the finer parts of the unstructured mesh may be lost. If we interpolate the solution variables on the unstructured mesh onto a dense uniform structured mesh, then the largely over-sampled uniform mesh will result in large CPU requirements. To solve this problem, a domain-decomposition method is adopted in which each subdomain spans different regions with different resolutions. If we can interpolate these unstructured areas into structured meshes with different densities, then this method will become more flexible and achieve better compression performance in terms of CPU speed of compression and accuracy.

This report describes the architecture and implementation of Opal-AE, a Python-based, convolutional, and domain-decomposition-capable, auto-encoder part of Opal. This part is implemented to learn an optimal low-dimensional representation. The low dimensional representation is in terms of coordinates on the expressive nonlinear data-supporting manifold within an unstructured mesh finite element fluid model. Auto-encoders implemented in this project include a global convolutional auto-encoder and a domain-decomposition convolutional auto-encoder. We also implemented two different ways to do the structured transformation. We tested them on flow past a cylinder, making comparison with traditional approaches such as POD and singular value decomposition.

Because this project is mainly working on the auto-encoder part of Opal, we only present the details of Opal using auto-encoder (Opal-AE), and compare this with other methods in Opal.

2. Background

2.1. NIROM

Due to large dimension size and complexity, it is challenging to solve real-world problems which require large scale dynamic simulations on today's computers. Reduced Order Models (ROMs) offer the potential to increase substantially computational efficiency while maintaining reasonable accuracy [4]. This potential is achieved by reducing the model's associated state space dimension or degrees of freedom.

Reduced order models can be divided into two types: projection-based ROMs and non-intrusive ROMs (NIROMs). Projection-based ROMs project the governing equations onto a subspace spanned by basis functions obtained from the compression of a dataset of solution snapshots [2]. However,

for specific applications, this method has been reported to suffer from instability and non-linearity efficiency issues [3]. NIROMs have become popular recently because, as well as addressing these issue, they avoid cumbersome and troublesome modification [5] of the source code for complex dynamical systems [3].

Wang et al. for the first time used LSTM to construct a non-intrusive reduced order model. The computation time outcome of ocean gyre simulation and flow past a cylinder simulation shows that the CPU time of ROM is reduced by three orders of magnitude compared to the full model [3].

Instead of LSTM method, Xiao et al. presented a ROM based on POD and the GPR method, which is six orders of magnitude faster than the high-fidelity model [5] Xiao et al. also combined domain decomposition and NIROM, improving the capability of the NIROM for large-scale problems [6].

2.2. Auto-encoder

POD, also referred to principal component analysis (PCA)[7], is a method of compressing the snapshots matrix into a small number of basis functions which keep the principle behaviour of the system [8]. It first applies the SVD to the snapshots matrix $S = U\Sigma V^T$and the POD basis functions are the first N columns of U. If some singular values are relatively small compared to a given tolerance, the size of basis functions can be further reduced.

The main limitation of POD is its linear nature [9] since many data sets include essential nonlinear structures that are invisible to POD [10]. The use of machine learning methods has increased dramatically in the last 15 years due to the increased capability of computers. One method, the auto-encoder, has shown great promise for image compression. For example, using efficient convolutional architecture and other techniques, auto-encoder has achieved better performance than JPEG 2000 [11].

Recently, the auto-encoder has been incorporated into a ROM framework as depicted in Figure 1. An auto-encoder is an unsupervised learning algorithm setting the target values to be equal to the inputs that can be trained by applying back-propagation. The network is able to discover correlations that exist inside the input features. It learns a compressed representation of the input and tries to reconstruct the input [12]. The auto-encoder includes adaptive encoder network to transform the high-dimensional data into a low-dimensional representation and a similar decoder network to recover the data from the low-dimensional representation [13]. Auto-encoders have been used in multiple applications, such as dimensionality reduction, image denoising [14] and image retrieval [15]

An auto-encoder can also be described as a nonlinear generalisation of POD [13], and in fact, POD can be constructed as two-layer auto-encoder, with linear activation functions for its linearly weighted input [16].

Carlberg et al. combine local compression using auto-encoders followed by global compression using POD. It is supposed that global compression is necessary because even if the number of degrees of freedom in each element is reduced, the number of elements in the mesh may still be large. The result showed that the auto-encoder reduced the dimensionality of the element-local state from 320 to 24 without incurring significant errors. [17]

DAgostino et al. use five hidden layers composed of 160-50-N-50-160 neurons and exponential linear unit (ELU) activation functions for each hidden layer, except for the output layer where a linear activation function is used. For the original (M = 27) design space, a reconstruction error achieved by this deep convolutional auto-encoder is smaller than 5% with 12 variables.[18]

2.3. Convolutional Neural Network

Fully-connected auto-encoders have two drawbacks: (1) The input data must be assembled into a 1D array, and the local spatial relations between values are thus eliminated[2] (2) It becomes computationally intractable for input data of sizes larger than 10^6 since the parameters for the neuron network will be extremely large.

Thus, we will investigate an alternative to fully-connected auto-encoders in which data is struc-

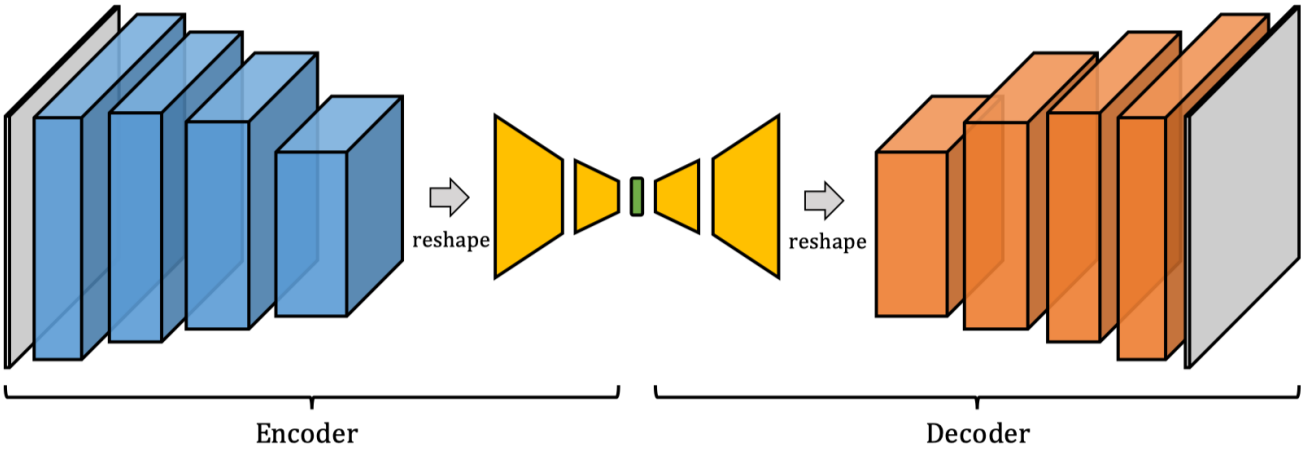


Figure 1: Global convolutional auto-encoder structure used in Opal-AE, taken from Gonzalez et al. 2018. It includes 4 convolutional layers and 2 dense layers each in encoder and decoder

tured as multiple arrays. First introduced by LeCun et at. in 1989[19], convolutional auto-encoders are able to address the drawbacks mentioned above, and they have achieved excellent performance in fields such as hand-writing classification or image detection.[20]

Feeding in two input signals (one is flipped), convolution is the operation that multiplies and adds together the instantaneous values of the overlapping samples. This procedure will be regarded as 2D convolution if the convolution is performed on two signals that have two perpendicular dimensions [21].

CNNs are neural networks in which the activation functions are expressed in terms of convolutional kernels or filters[22]. These filters are just small patches that represent visual feature; these are the weights and biases of a CNN. We take each filter and convolve it over the input volume to get a single activation map. In convolutional neural networks, a layer consists of feature maps, which is calculated by filters in the previous layer.

The filter is moved across the image left to right, top to bottom, with a one-pixel column change on the horizontal movements, then a one-pixel row change on the vertical movements. The convolution for a 3×3 filter is depicted graphically in Figure 2, The three graphs show the effect of padding and stride.

The amount of movement between applications of the filter to the input image is referred to as the stride, and it is almost always symmetrical in height and width dimensions. The stride can be changed, which has an effect both on how the filter is applied to the image and, in turn, the size of the resulting feature map.

For example, the stride can be changed to (2,2) in the third graph of figure 2. This has the effect of moving the filter two pixels right for each horizontal movement of the filter and two pixels down for each vertical movement of the filter when creating the feature map.

Padding is a margin of zero values which are placed around the image. The benefit for adding padding is that each pixel in the image is given an opportunity to be at the center of the filter as depicted as graph 2.

For the 2D output X, a convolutional layer generates feature maps Y by applying 2D convolutions [2]:

$Y^f_{i,j} = \sum_{k=0}^{a-1} \sum_{l=0}^{b-1} K^f_{a-k,b-l} X_{1+s(i-1)-k,1+s(j-1)-l}$

where s is the stride.

The two critical properties of convolutional neural networks are: [23] 1) Local connections: it computes local low-level features in a small subset [24]. 2) Shared weights: the shared nature of each filter bank allows the identification of similar features in the overall input, which leads to a reduction in the trainable parameters.

Similar to detecting an instance of an object

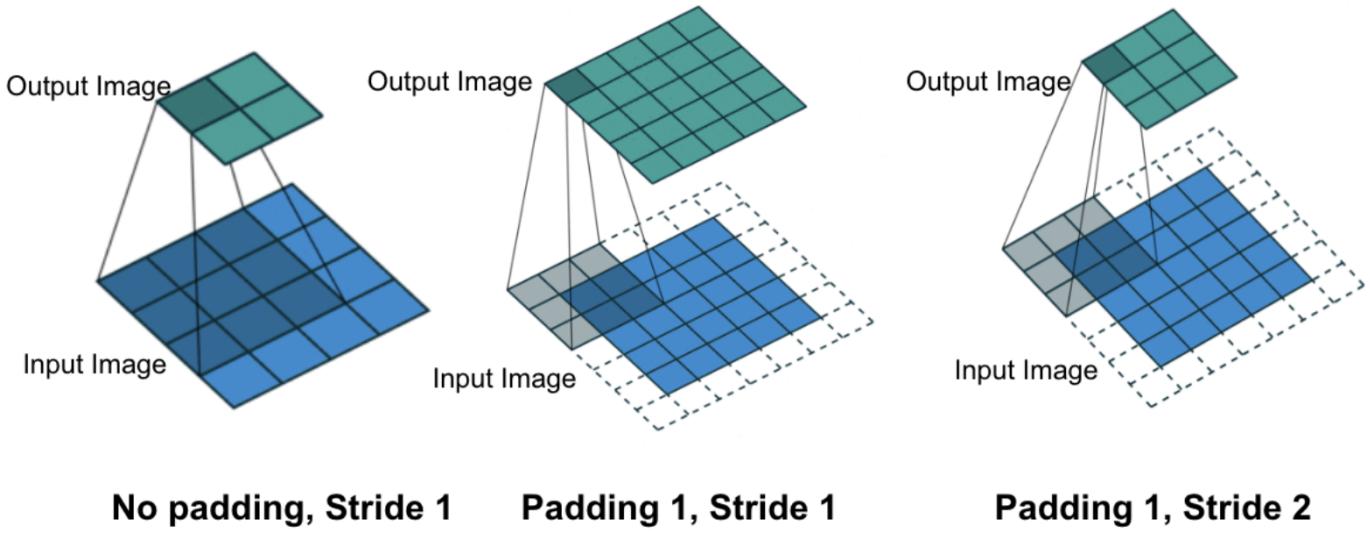


Figure 2: 2D convolution procedure, taken from Murphy 2016. Last two graph shows the effect of adding padding and stride,

anywhere in an image, a convolutional layer in an auto-encoder should be able to capture the large parameter variations implicitly defined in the initial condition. This feature is called location invariance, and the two properties of CNNs mentioned above work to detect this feature.

Gonzalez et al. [2] presented a parameter-varying flow in a periodic box to indicate the location-invariance capabilities. The result showed that compared to POD, the recurrent convolutional auto-encoder (with 8 latent vectors) performs well in predicting new solutions using fewer features than for POD, illustrating the effectiveness of deep auto-encoder-based approaches to nonlinear model reduction.

As a result of location invariance, the components in the low-dimensional representation have a direct meaning, for example, x or y position. This feature enables the compressed output to be linearly computed. In this project, we hope to reduce the dimension of building matrix to a rank-4 feature map, including two-dimensional size, location and rotation.

Kashima examined the projection error on the reduction of a singularly perturbed system for an auto-encoder and POD [25]. The error for POD is 200 times greater than for the auto-encoder. This result shows when approximating the noise response data, the accuracy of a two-dimensional nonlinear manifold cannot be realised by any two-dimensional linear manifold.

In image processing, a few filters are set and used to perform a few tasks. For example, if a 3×3 filter has the same weight on each element, then it can be used to blur images. Each pixel's new value is set to a weighted average of that pixel's neighbourhood. Other filters, such as Sobel filters, can give us an activation map that indicates where the edges are in the input.

3. Program architecture

3.1. Opal

Written in Python and partly Fortran, Opal is an extensible Application Programming Interface (API) that can manipulate the input and output files of some AMCG software packages (IC-Ferst and Fluidity). To date it has been used to generate sensitivities and to build non-intrusive reduced order models. In the latter, the available method for compression is POD, and for prediction, the method available is Gaussian Process Regression (GPR). In this project, compression using an auto-encoder is implemented in Opal.

Code metadata description	Please fill in this column
Current code version	v0.1
Permanent link to code/repository used for this code version	`https://github.com/msc-acse/acse-9-independent-research-proj`
Legal Code License	MIT License
Code versioning system used	git
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Linux, python, sklearn [26], keras [27], tensorflow [28], ICferst [29], Opal
Support email for questions	jm2518@ic.ac.uk

Table 1: Code metadata

3.1.1. Settings of Opal

The Opal settings includes variable parameters in the process of compression and prediction. In auto-encoder compression, we aim to design a flexible system that can be applied in various scenarios. Below is a summarized list of the options available for Opal-AE:

- nsplt: number of domains in domain decomposition. This value is the logarithm of domain number to the base 2.
- dim: number of dimensions. it could be 2 or 3
- nLatent: number of latent variables to which OPAL compresses
- epochs: number of iteration in auto-encoder training
- generate_directory: the name of the directory which contains the output files, including interpolated matrix and the prediction vtu files.
- file_prefix: the file prefix for snapshots. It enables Opal-AE reading in snapshots conveniently. e.g. if one of the file of snapshots is "Flowpast_0.vtu", then the file_prefix should be "Flowpast_".
- batch_size: batch size in auto-encoder training

These settings can be set in Diamond, a user-friendly GUI for XML files, as depicted as figure 3. Diamond can read the rng file from Opal and has a number of default settings. The rng file is configurable to adjust the default settings.

3.1.2. Input files and core files in Opal

Opal reads in several files to set up a problem:

- Snapshots for the variable of interest (e.g. velocity): snapshots are generated from the high-fidelity model (HFM). For the examples studied in this project we will use a finite element code with the ability to solve on unstructured meshes (Fluidity and ICFERST) [5]. The columns of the matrix are snapshots of the solution to the HFM taken at certain times.
- Settings for snapshot generation: an XML file that is used to generate snapshots in ICFERST. Opal reads this file in order to get important information about the snapshots, such as file names or number of snapshots.
- Opal settings file: an XML file which can be modified by Diamond. Its filename extension is usually .opal. See figure 3.

Introduction of the core files:

- dd_autoencoder.py: The helper for auto-encoder, including the global and domain decomposition auto-encoders. The domain class, autoencoder_interpolation class and all the im-

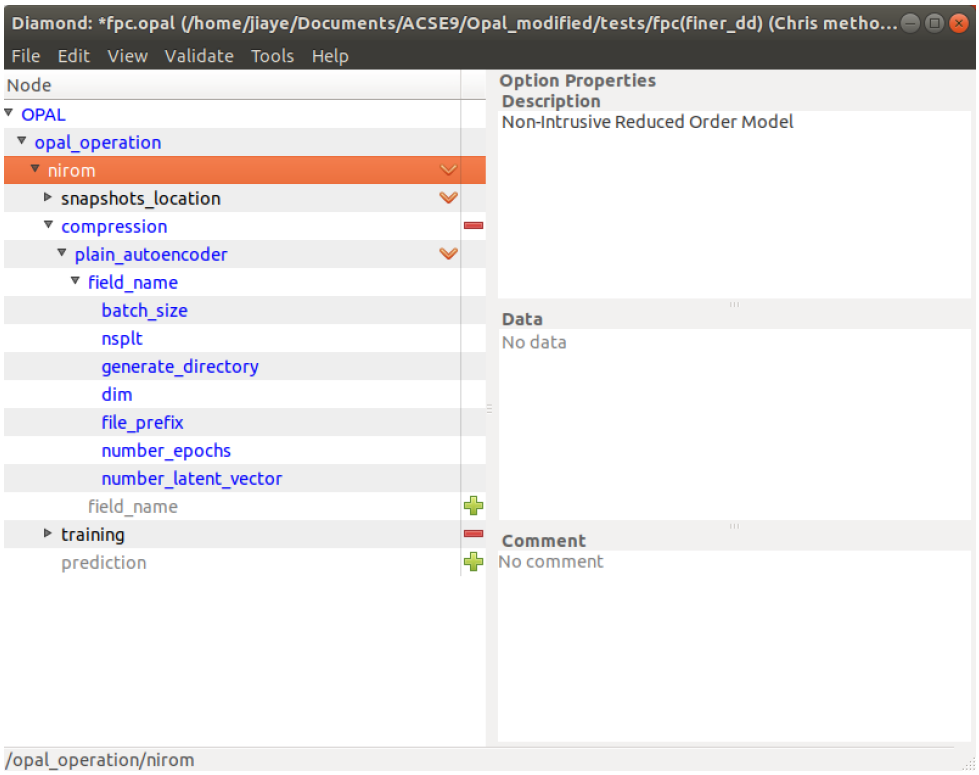


Figure 3: Diamond setting for Opal-AE. The rich settings provide a powerful capability that can be applied in various scenarios.

portant functions for the compression stage are here.

- opal.py: The entrance of Opal. An general function chooser for several capabilities. We use NIROM in this project.
- nirom.py: The entrance of NIROM function. The skeleton of the whole nirom file.
- nirom_tools.py: The helper for NIROM. This file includes GPR and SVD method. The auto-encoder skeleton is also in this file.

3.1.3. Procedure of Opal-AE

After certain steps, we save result for reuse. When re-running a test case, the program can then skip these steps which enhances the program's efficiency.

As shown in figure 4, Opal-AE begins from reading in NIROM settings and snapshots, provided by the user. It then transforms the NIROM settings into a Python class, and transforms the snapshots into variables, such as coordinates or values.

The 3rd step is determining domains of the whole mesh. In this project, we implement a method that determines the size and position of suitable subdomains depending on some factors. The method has two aims, firstly to try to balance the weight sum of each sub-domain, to make them same or as similar as possible. It also minimizes the sum of the weights of nodes on the boundary between sub-domains. In other words, it reduces the communication between subdomains.

The 4th step of NIROM is to do the interpolation. It is hard to perform a convolution on an unstructured mesh since its connectivity is irregular, so we interpolate to transform the data from an unstructured to a structured mesh. If the interpolated matrices are already saved in the target directory, which means this process has been done before, then the program will just read the files and skip the interpolation part.

The 5th step is training the auto-encoder. We implement a general code for the auto-encoder that could be used in a global approach and also in the domain decomposition approach. The param-

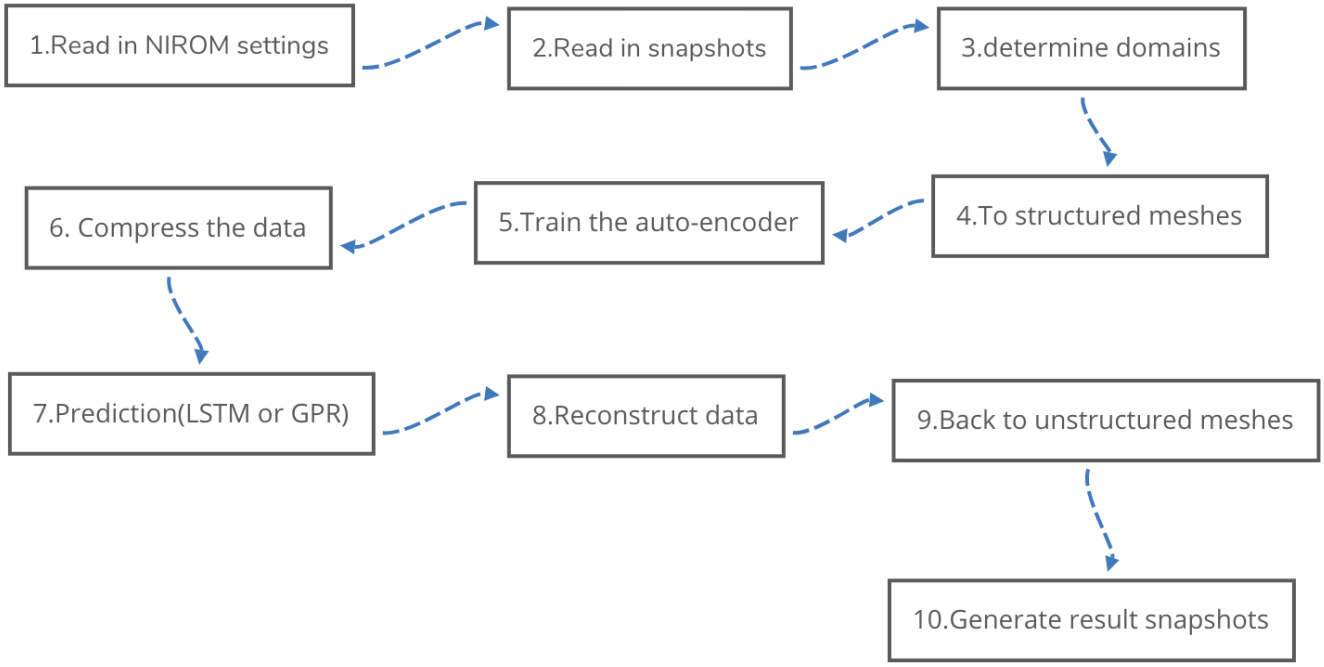


Figure 4: Overall process of Opal-AE

eter called nsplt determines the number of subdomains when using domain decomposition. Users can adjust it to decide using global or domain decomposition approach (If nsplt is 0 then the global approach is used). If the auto-encoder models already exist, then the program skips the training and reads the model from directory.

The 6th step is compressing the data. A low-dimensional representation is constructed by the encoder network.

The 7th step is to train a neural network to generate hypersurfaces of the model that should have similar predictive power as the high-fidelity model. The neural network is able to map the set of latent vectors from one-time level to the next time level, and so, predict the variables. There are several methods to generate hypersurfaces, for example, Gaussian Process Regression (GPR) and Long Short-Term Memory (LSTM) [5]. If the results matrix exists in the directory, it is read in by Opan and this part is omitted.

After a certain number of steps, the decoder network allows the user to reconstruct the full-dimensional state.

The last step is generating the snapshots. The program will determine which subdomain that each node belongs to, will interpolate the matrix on the target domain and will get the values at the nodes of the unstructured mesh.

4. Software Development Life Cycle

4.1. Pre-tests

To test the auto-encoder function in terms of the compression performance, we first apply an 1D problem to find the capability of fully-connected auto-encoder. We achieve a loss of 10^{-2}, which is not as low as we would like. This poor performance is due in part to the fact that the fully-connected auto-encoder uses many parameters. In order to reduce the cost and enhance the performance, we decide to use a convolutional auto-encoder and test it on a 2D dataset–CIFAR-10. The reason why we use this dataset is its size. Being 2^5 (CIFAR-10 is 32×32) it is larger than MNIST (28×28), it gives the flexibility to test strides larger than one. Another reason to use this dataset is that it is coloured, which means

it has more than one channel, similar to the 3 velocity components in a 3D flow problem.

The best loss we have obtained for the convolutional auto-encoder on CIFAR-10 compression is 6×10^{-3}, and the output result we have got is reasonable.

4.2. Determine the subdomain partitions

The aim of domain decomposition is to group together the nodes with same density so that the density of the interpolated matrix can adapt to the density of different parts of mesh. The density of the interpolated matrix is related to the domain size, position, and interpolation interval. The weight for a node is the smallest volume divided by the biggest element volume surrounding a node, i say. This weight becomes low in regions where between areas of relative high and low density of nodes or resolution. The equation for the weight w_i of a node i is:

$$w_i = \frac{V_{mini}}{V_{maxi}}, \tag{3}$$

in which V_{mini}, V_{maxi} are the minimum and the maximum element volumes that are connected to node i respectively. The aim of the domain partitioning method is then to balance the weight sum of all the nodes in each subdomain as well as minimize the sum of the weights in the halo between subdomains. The partitioning encourages the boundaries of the subdomains to occur through areas between dense and high resolution areas. It is hoped this approach then isolates in subdomains differing resolutions.

Figure 5 shows an outline of how to form the structured meshes necessary for the convolutional auto-encoder. After the weight of each node is found, Opal-AE will find $\mathcal{D}_i$ (the subdomain number that node i belongs too) which is thus an array with length equal to the number of nodes in the unstructured mesh. Depending on the situation, there are three ways to obtain $\mathcal{D}_i$. If the code has already been executed, and the $\mathcal{D}_i$ has been saved in the directory, then read the $\mathcal{D}_i$ from the file. If using the global method, then set $\mathcal{D}_i$ to 1 for all i (as there is only one domain). If the number of dubdomains exceeds 1, then call the get_whichd method (get $\mathcal{D}_i$). This method uses a K-way recurrent mean-field neural network [30] to separate the subdomains on the whole mesh. It balances the weight sum of each subdomain, and minimizes the sum of the weights of nodes on the boundary between subdomains.

After the $\mathcal{D}_i$ is found, then calculate the minimum and maximum coordinates of each domain. These values indicate the position and size of the domains. As depicted as figure 9, the structured meshes are the minimum rectangles that can include all the nodes in the domain. This method is simple but useful. One disadvantage of this method is that if the boundary is tilted (for example, 45 degrees), the overlap between subdomains will be very large, and undoubtedly this will cause a waste of resources since the auto-encoder will take into account this overlap area more than once. In the last section of this report, a better way to overcome this problem is discussed.

The next step is finding the structured mesh size that determines the density of each domain after the transformation from unstructured meshes. Two methods are available for Opal-AE, one is to balance the area size, another is to balance the area density. The first method set number of nodes on the shortest edge to 32 and find the that on other edge according to the proportion of number of nodes of this edge to that of the shortest edge. In the second method we suppose every domain contains a similar number of nodes. If we balance the total number of nodes in the structured meshes, then the the result could be more accurate since the density in each structured area is follow the one of the unstructured area. It performs the same job initially as the first method, but for every edge, a proportion of product of numbers of nodes on every edge on maximum domain to that of current domain is taken into consideration as a coefficient. In other words, Opal-AE make the product of numbers of nodes on every edges as similar as possible.

In the auto-encoder structure, it contains four convolution layers in each domain, and the size of feature will be reduced by half in each layer. We choose 32 (2^5) as the starting matrix length and

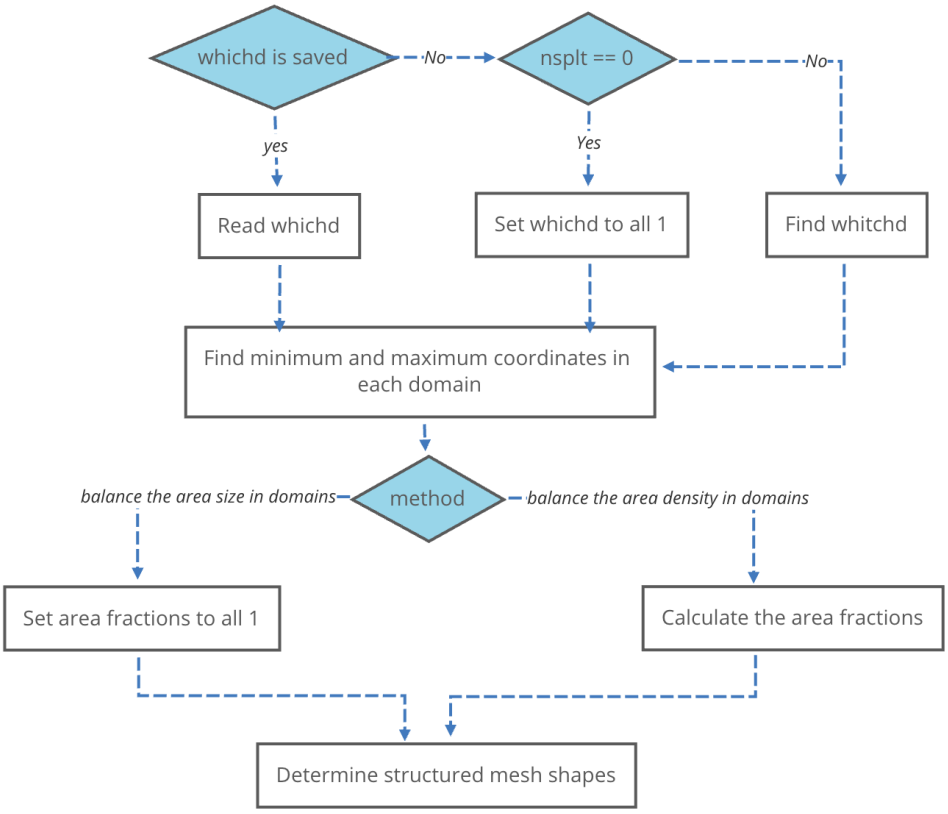


Figure 5: Process of how to find mesh structure in Opal-AE

Figure 6: Density domain decomposition example when ndomain=2, 4, 8

16 (2^4) as step (etc. 32, 48, 64) because when it has been reduced for four times, the matrix length will start from 2 and the step is 1. This is the smallest value for matrix length that we can take. Although the convolutional encoder will still work fine if the start length is not 2^5 but another value,

Figure 7: Structured matrices example on domains when ndomain=2

the problem lies in here is when decoding¿ In this case the reconstructed matrix size will not be consistent with the input size. This structure cannot even be called an auto-encoder due to the difference between input and output size.

4.3. Interpolation

In this part, if the structured mesh data has been obtained before, then use the previous values. If not, then generate all the subdomains. The domain class contains all the information for the subdomains, including the minimum and the maximum coordinate, the shape of structured mesh, the index of the matrix, the coordinates of every nodes inside each subdomain. Since all other information has been obtained in the former step, in this step, we only need to find out the last parameter. This coordinates can be generated by the formula:

$$C_{result} = C_{minimum} + index \times interval. \quad (4)$$

where C_{result} represent the coordinate, $C_{minimum}$ represent the up left conner of the matrix, the index represent the row index and col index of the matrix, the interval is the x and y component of distance between each nodes. After obtained the domain, we generate a matrix overlapping the unstructured mesh and interpolate every element on this matrix [31]. This procedure is included in the library called VTU tools [32]. VTU tools use linear interpolation. The equations for 2D linear interpolation are as follows:

$$w_y = 1 - \frac{y - y_{1,1}}{\Delta y} \quad (5)$$

$$C_{left} = w_y C_{1,1} + (1 - w_y) C_{2,1} \quad (6)$$

$$C_{right} = w_y C_{1,2} + (1 - w_y) C_{2,2} \quad (7)$$

$$w_x = 1 - \frac{x - x_{1,1}}{\Delta x} \quad (8)$$

$$C_{interpolated} = w_x C_{left} + (1 - w_x) C_{right} \quad (9)$$

For example, in flow past a cylinder, the number of vertices is 7908 and this unstructured mesh is transformed into a 64×64 regular matrix. The value of each point of the matrix is generated by interpolating the original unstructured mesh on the coordinates of points on the regular matrix.

4.4. Global auto-encoder structure

Our goal is to develop a neural network model to compress and reconstruct datasets. We tested various convolutional architectures characterized by different numbers of parameters. Using the fully-connected architectures we are able to obtain a Mean Square Error (MSE) loss of 0.006. Given the high number of parameters, these networks suffer from severe overfitting. In order to improve on this, we turned to deeper neural network structures and convolutional layers.

Throughout this project we used the ELU activation function instead of ReLU as it decreases the bias shift by pushing the mean activation towards zero [33]. Though several specific architectures and variations were tested the following architecture was the best one we found.

The main challenge is the small number of snapshots available, hence the overfitting problem in the training. Since most of trainable parameters are in the dense layer, there are several methods we adopted to decrease the number of parameters and therefore solve the problem of overfitting:

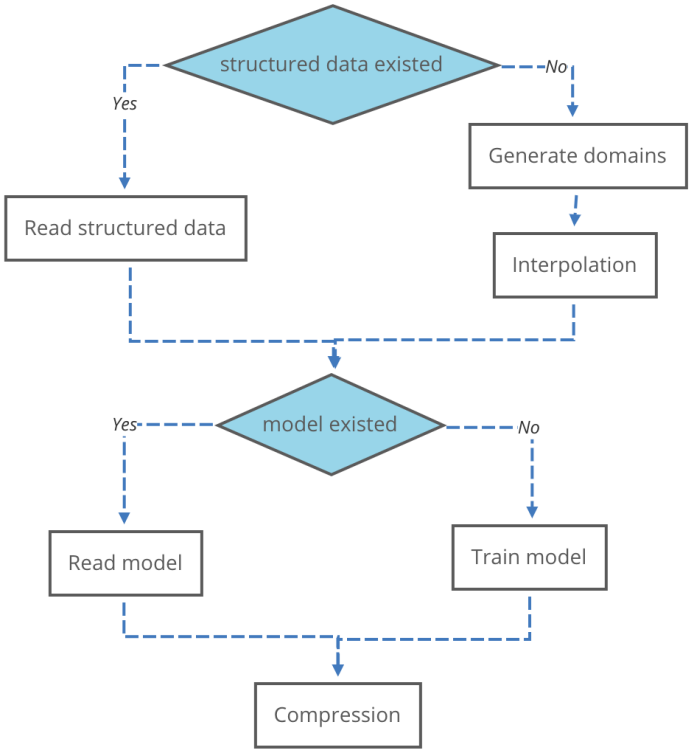


Figure 8: Process of interpolation part of Opal-AE

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(n, 64, 64, 2)	0
conv2d_1 (Conv2D)	(n, 32, 32, 4)	204
conv2d_2 (Conv2D)	(n, 16, 16, 8)	808
conv2d_3 (Conv2D)	(n, 8, 8, 8)	584
conv2d_4 (Conv2D)	(n, 4, 4, 8)	584
flatten_1 (Flatten)	(n, 128)	0
dense_1 (Dense)	(n, 64)	8256
dense_2 (Dense)	(n, 32)	2080
dense_3 (Dense)	(n, 64)	2112
dense_4 (Dense)	(n, 128)	8320
reshape_1 (Reshape)	(n, 4, 4, 8)	0
conv2d_transpose_1	(n, 8, 8, 8)	584
conv2d_transpose_2	(n, 16, 16, 8)	584
conv2d_transpose_3	(n, 32, 32, 4)	804
conv2d_transpose_4	(n, 64, 64, 2)	202

Table 2: Example of Auto-encoder structure for global Opal-AE

1. Increase the number of convolutional layers: We applied stride=2 in every convolutional layer, which means the size of the matrix will be reduced after the convolution. If the number of convolutional layers is increased, the input size of matrix in the dense layer will be relatively small.

2. Control the size of feature map: To retain most features in the input, the feature map size is doubled in the first two convolutional layers. To decrease the input size of dense layers, the feature map size remains the same in the third and fourth convolutional layers.

3. Choose the best filter size: The filter size for the first two layers is 5×5, because a larger filter size can extract more features from feature maps. The filter size for the last two layers is 3×3, because the size of the feature map in these layers is small enough that 5×5 filter is overqualified. A smaller filter size can also reduce the number of parameters.

Other parameters such as the number of epochs and batch size will affect the performance of the auto-encoder. The number of epochs should be as large as possible. Considering the computa-

tional efficiency, 1000 iterations were used in this project. Small batch sizes can be very slow to train, while large batch sizes will lead to low performance in the dimensionality reduction. On flow past a cylinder, using a global approach, the loss achieved for a batch size of 1000 is 8 times larger than for a batch size of 128. Although we have not tested it on a large range of batch sizes, the result of batch size of 128 is the best number we can have in this project that balances the performance and speed in training.

As depicted in table 2, many variations on the gradient descent method are available to optimize the loss function. We tested three different optimizers, SGD (Stochastic Gradient Descent), RMSprop and Adam. We usually obtain a faster convergence with Adam, though SGD was the common choice by others due to stability in the validation accuracy.

4.5. General domain decomposition auto-encoder design

In domain decomposition, using the same structure as the global auto-encoder is appropriate, since the inputs for this auto-encoder are multiple domains. Instead we applied the structure depicted in figure 9. This structure has several sets of convolutional encoder layers, and the output of these convolutional layers is concatenated to feed into the dense layers. The latent vectors are the total compressed representation for all domains. The decoder structure is just the inverse of the encoder structure.

We also considered applying a separate auto-encoder for each domain (and not combining the dense layers), but the performance was not as good as we had hoped. It takes more parameters and training time to achieve the same performance as the combined auto-encoder.

We devised a general method to build the auto-encoder structure that can be applied to different numbers of subdomains and different numbers of layers. Since the general method needs to adapt itself to whatever number of domains, the code for convolutional layers should be general for each domain, which means the convolutional structures for each domain are exactly the same.

The problem we faced here is when the number of domains increases, the inputs to the dense layers will also increase. The original method we used, as depicted in figure 10, concatenates every output from the convolutional layers and feeds this into the fully-connected layers. In each level, there is only one dense layer. The number of parameters in dense layers is relatively large since we did not split it. If the number of input becomes large, the number of parameters for training will be large as well, and thus the overfitting problem will be an issue once again.

We adopted nested fully-connected layers to overcome this problem. The nested fully-connected layers concatenate each two outputs of the previous layer as a pair. This method will reduce the parameters used in training.

4.6. Recovering the unstructured mesh solution variables (VTU files) from a regular mesh convolutional autoencoder

The post-processing that transforms the regular mesh data back to the unstructured mesh is not included in the VTU library. Linear interpolation is the easiest way to solve this problem. The values of points in the grid can be calculated from the values of four corners of this cell. If this is a 2D problem, the procedure is the same as the linear interpolation explained in section 4.3.

For a 3D problem, the procedure is similar, the only difference is in the first step to find the 2D interpolation value for the projection of the point on the left and right side of the grid.

5. Results

5.1. Global method

We generated 100 snapshots for flow past a cylinder in 2D and predicted for one second using a GPR. The snapshots are generated from ICFERST. The computational domain has length 2.2m and width 0.41m. Figure 12 shows the original plot of flow past a cylinder at 0.4 seconds, and figure 13 shows the results for prediction at this time. Despite some loss of detail in the flow (especially near the position x=0.35m, y=0.22m),

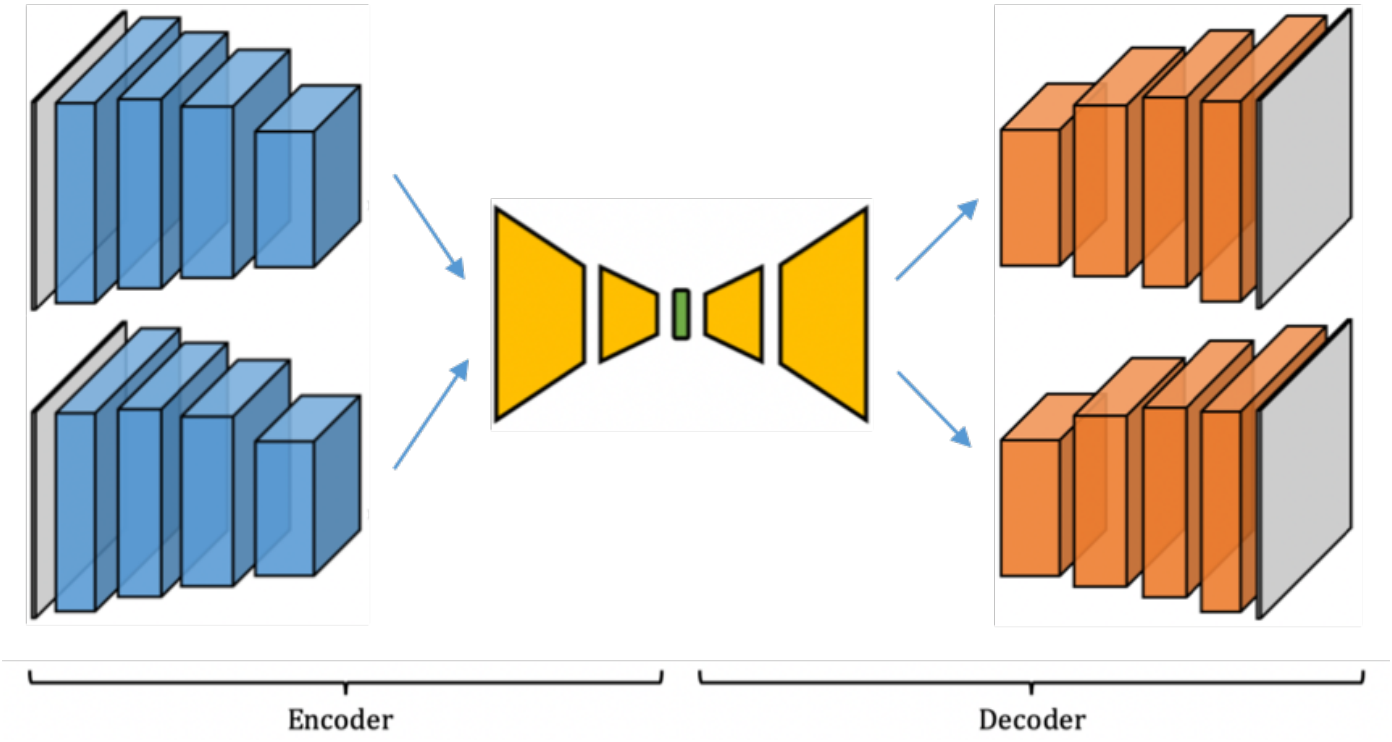
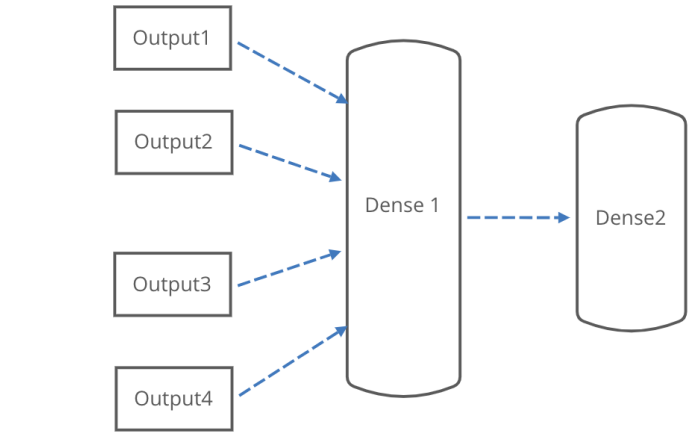
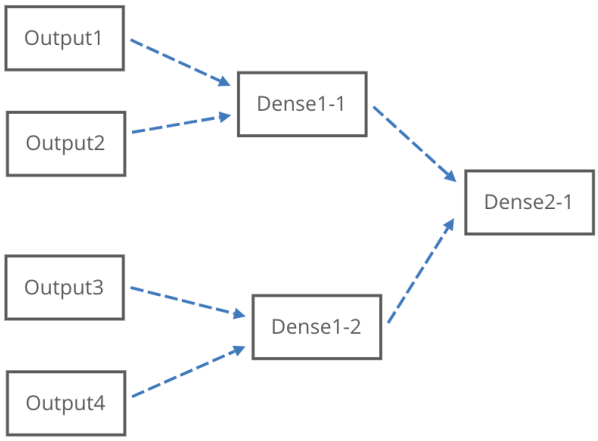


Figure 9: Auto-encoder structure for 2 domains



/2/2

Figure 10: Original fully-connected layers



/2/2

Figure 11: Nested fully-connected layers

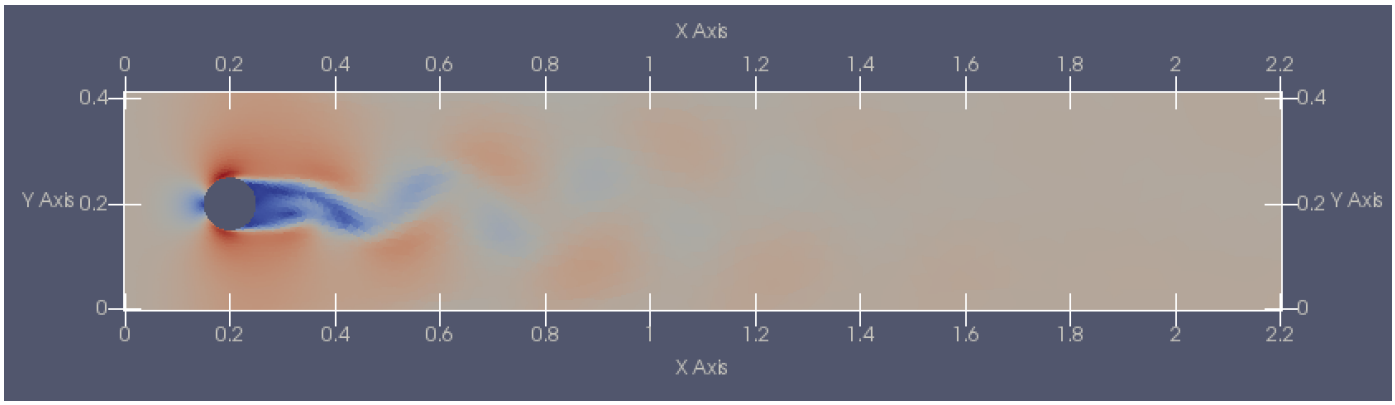


Figure 12: Original snapshot at 0.4s

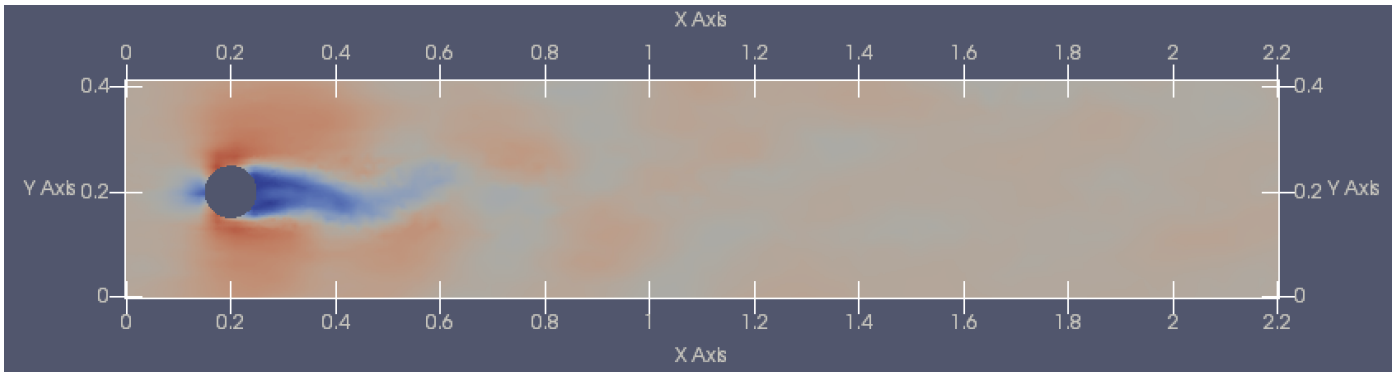


Figure 13: Result snapshot for global auto-encoder at 0.4s

the NIROM approximation retains most of information in the original snapshot.

Figures 14 and 15 show the L_2 norms and L_∞ norms. The equation for this two norm is following :

$$x_2 = \sqrt[2]{\sum_i^n (a_i)^2} \tag{10}$$

$$x_\infty = \sqrt[\infty]{\sum_i^n (a_i)^\infty} \tag{11}$$

The norm values reflect the difference between the original snapshots and the predictions by NIROM. The higher the norm is, the larger the difference will be. These two figures shows the norm values remain in a reasonable range over the whole time period of interest.

Figures 16 and 17 show the velocity time histories at a point (the midpoint of the y axis and at x=0.3. The upper plot is velocity from the original snapshots, and lower plot is velocity obtained by the auto-encoder by feeding the full velocity through the auto-encoder. The similarity between them reflects the capability of auto-encoders to maintain accuracy while massively compressing the velocity solution variables.

5.2. Simple domain decomposition

We generated 100 snapshots using domain decomposition for the test case of 2D flow past a cylinder. We equally decompose it into 2 domains, and, because the left side has more unstructured nodes, we transform the grid into a 48×48 left structured mesh and a 32×32 right one. Figures 18 and 19 show the results for prediction at 0.4s. The NIROM approximation of the snapshots retains most of the information in the original snapshot, including the details in the flow that were not clear when using global compression.

Figures 20 and 21 show the L_2 and L_∞ norms of the error. These two figures show the error

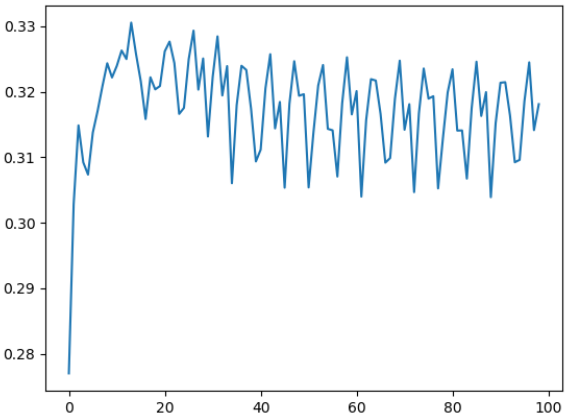


Figure 14: Norm for global auto-encoder, order=2. Norm represent the difference between the original data and result data.

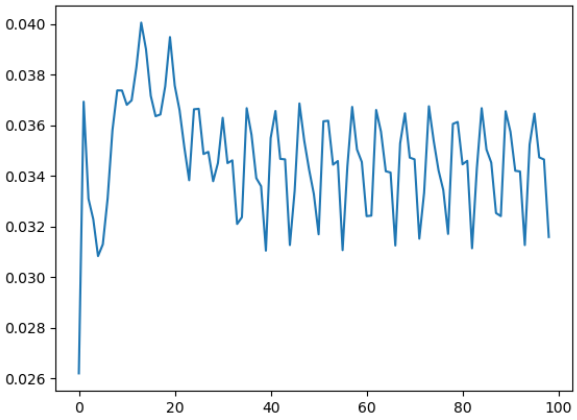


Figure 15: Norm for global auto-encoder, order=infinity. Norm represent the difference between the original data and result data.

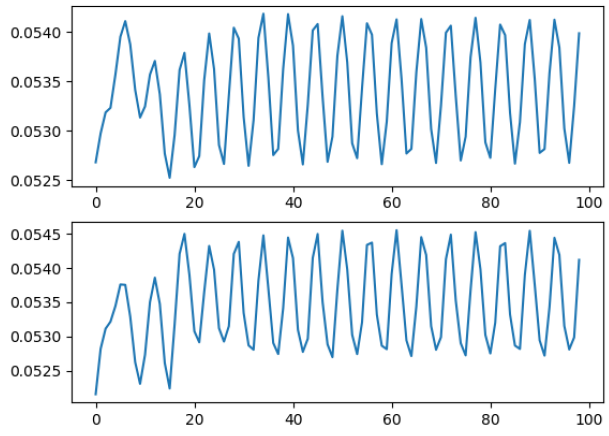


Figure 16: The x component of velocity plotted against time at $x = 0.3$, $y = 0.205$. The top figure is the high fidelity model and bottom figure is the result from the auto-encoder.

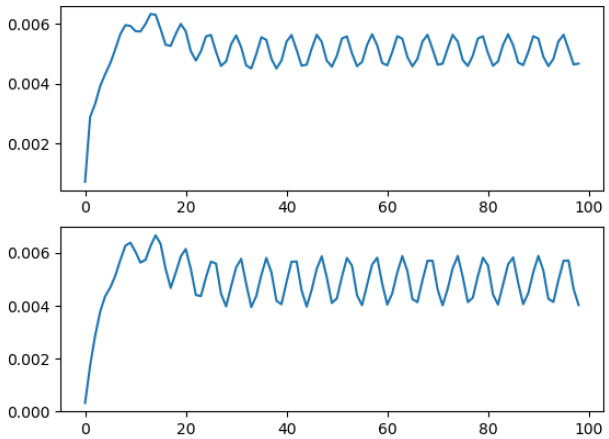


Figure 17: The y component of velocity against time at $x = 0.3$, $y = 0.205$. The top figure is the high fidelity model and bottom figure is the result from the auto-encoder.

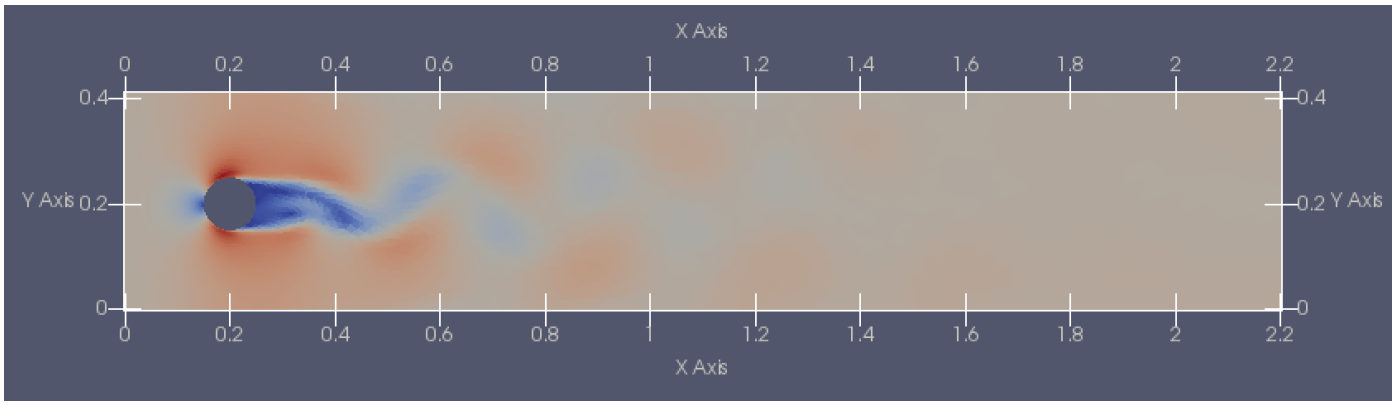


Figure 18: Original snapshot at 0.4s for domain decomposition

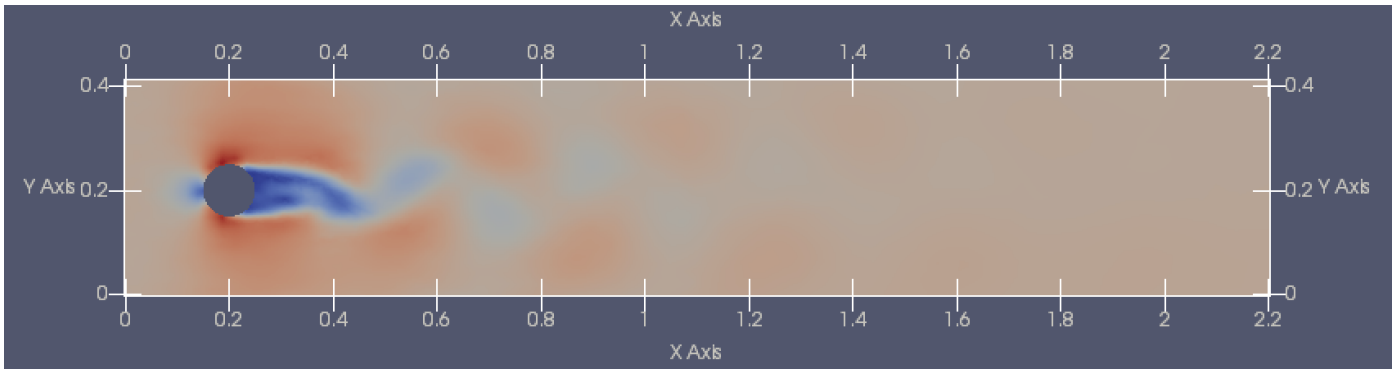


Figure 19: Auto-encoder output snapshot at 0.4s for domain decomposition

norm values remain in a reasonable range over the entire simulation.

The reason for the large norm values at beginning is because the flow is not obvious at the beginning, and the dense part cannot detect those features in the flow as it do in the later time series, which cause some inaccuracy in before the flow stable.

Figures 22 and 23 show velocity plotting at a point (the midpoint of the y axis and at $x = 0.3$). The upper plot is velocity in original snapshots, and the lower plot is velocity as approximated by NIROM. The similarity between them reflects the capability for the auto-encoder to maintain accuracy.

5.3. Density domain decomposition (balance the number of nodes in each subdomain)

As shown in figure 9, the domains for that balance the number of nodes do not have equal areas. The method to determine the size of the subdomains attempts to balance the number pf nodes in each subdomain. The order 2 norm depicted in figure 24 shows that the norm is stabilized around 0.25, better than the result of global method and that of simple domain decomposition (splitting the domain according to the x-coordinates of the nodes). The fluctuation is much more obvious in this graph than in the norm plot of simple domain decomposition. It indicates when the time reaches the peak of the repeated movement of the curve, the loss will become large.

The results when using 4 subdomains are shown in figure 26 which shows that error norm for this method is further reduced (the value of the norm is around 0.23).

5.4. Density domain decomposition (balance the area density)

Furthermore, we supposed every subdomain has a similar number of nodes inside but tries to form paritions that cut across areas with changing mesh resolution. This uses the weight function for the domain decomposition defined by equation 3.

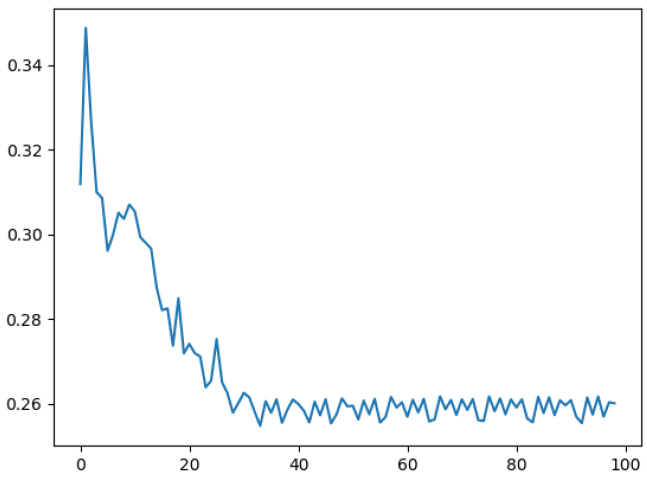


Figure 20: Error norm for 2 domains simple domain decomposition (splitting the domain according to the x-coordinates of the nodes). The L_2 norm represents the difference between the original data and the NIROM approximation.

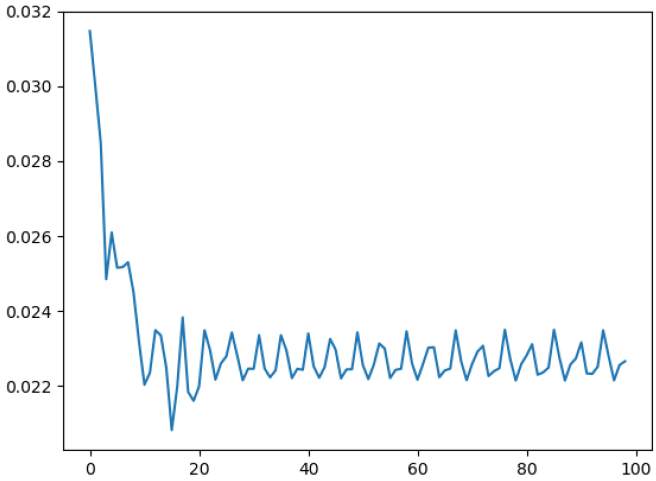


Figure 21: Norm for 2 domains simple domain decomposition (splitting the domain according to the x-coordinates of the nodes). The L_∞ norm represents the difference between the original data and the NIROM approximation.

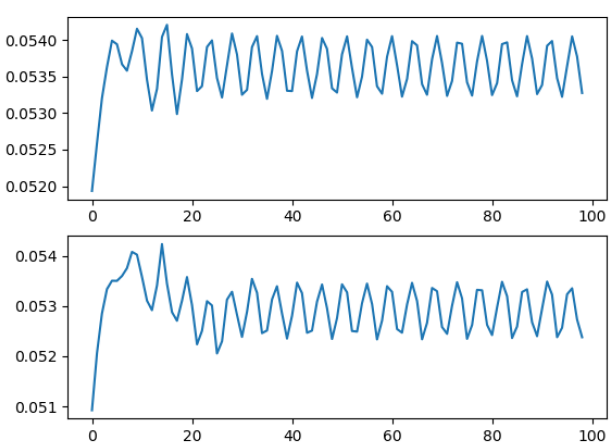


Figure 22: The x component of velocity at (0.3, 0.205) using simple domain decomposition (splitting the domain according to the x-coordinates of the nodes). The top is the high fidelity model and the bottom is the auto-encoder solution.

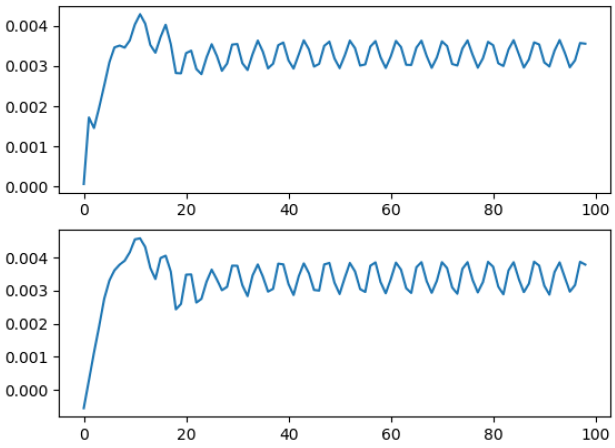


Figure 23: The y component of velocity at (0.3, 0.205) using simple domain decomposition. The top is the high fidelity model and the bottom is the auto-encoder solution.

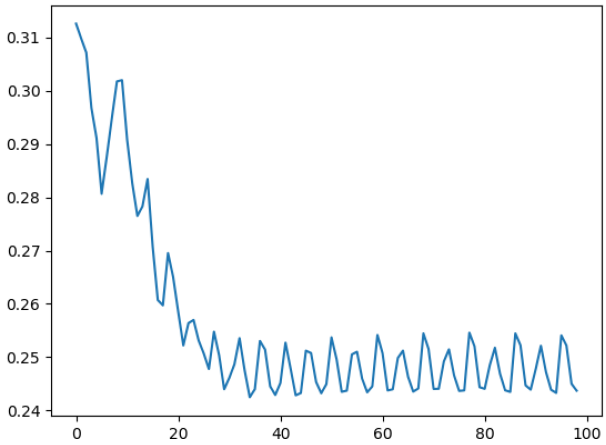


Figure 24: L_2 norm of the difference between the original snapshots and NIROM approximation for 2 domains using subdomains that have approximately the same number of nodes.

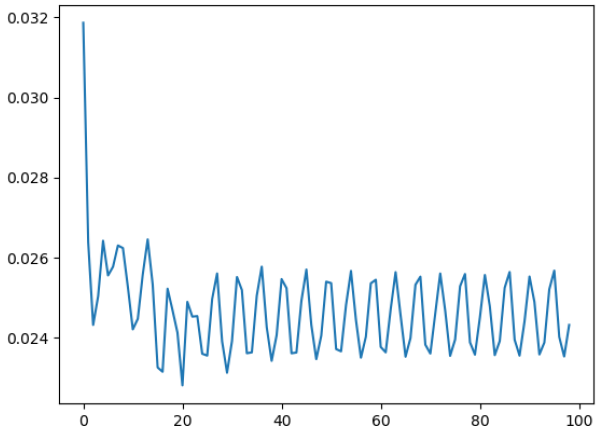


Figure 25: L_∞ norm of the difference between the original snapshots and NIROM approximation for 2 domains using subdomains that have approximately the same number of nodes.

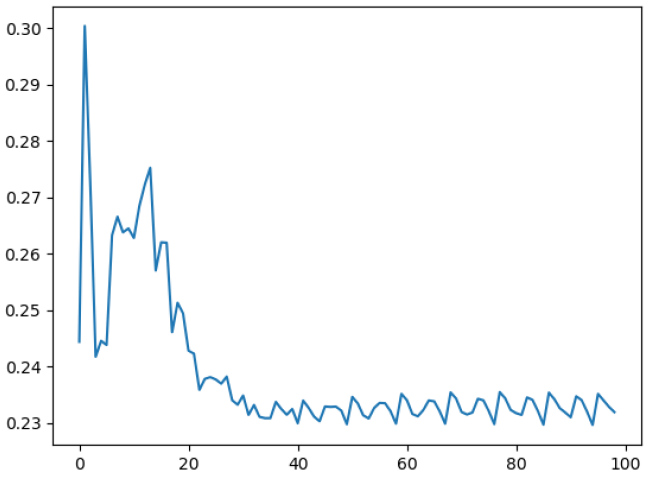


Figure 26: L_2 norm of the difference between the original snapshots and NIROM approximation for 4 domains using subdomains that have approximately the same number of nodes.

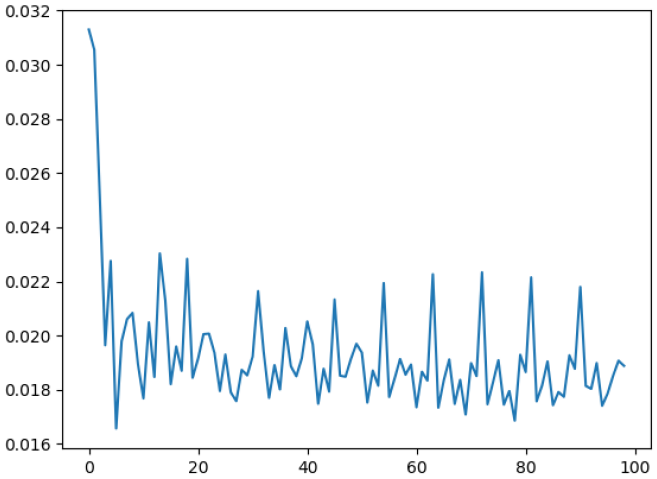


Figure 27: L_∞ norm of the difference between the original snapshots and NIROM approximation for 2 domains using subdomains that have approximately the same number of nodes.

Figure 28 presents the L_2 norm for 2 domains. The result shown that the norm is around 0.23, better than the former method for 2 domains.

Figure 28 presents the L_2 norm for 2 domains that balances the area density. The result shows that the norm is around 0.21, better than all the other results.

6. Discussion

In section 4.2, we introduce the limitation of transforming from unstructured meshes to structured meshes, that is, when the boundary is tilted, the overlap between domains can become very large and this can impact auto-encoder efficiency. There are two way we can fix it. The first one is relatively simple. Make the boundary vertical or horizontal, then the overlap between the domains will disappear, but the waste on the boundary of domain still exists. Our current algorithm is not enough to be used on this method.

Another method is to deform the structured meshes to fit the shape of domain. Because the convolution operation is not related to the distance between nodes, the coordinates of structured meshes do not necessary to be physically same as those of unstructured meshes. If the structured mesh is deformed to adapt the shape of domain, there will be no waste in the auto-encoder training.

For now, a 3D example is not included in Opal-AE. 3D modelling requires a huge number of parameters to be calculated when training the auto-encoder. The main difficulty will be how to overcome the overfitting since the input snapshots are limited. One potential improvement includes adding a data augmentation feature for the training. I believe these can be done in the near future.

The most difficulty exists in the training of auto-encoder. There seems to be a serious bug in the Linux version of Keras. Initially, every model training was tested in the MacOS, but when we put all the code in the Linux system, the training loss became high and difficult to explain. We used the exact same script to run on MacOS and Linux. After a certain number of epochs, the loss on Linux OS would explode, yet the result on MacOS kept reducing. Since Opal is only running on Linux, this bug was so serious that, every time running Opal, we have to copy the model we trained on MacOS into the Opal directory to be able to train the auto-encoders.

We also discussed another method to adapt the convolution on the unstructured meshes. Initially, we tried to change the way that the convolution works. We manage to choose some certain points—the points belongs to the maximum independent set of the mesh—on the mesh as convolution centres, and do the convolution of the values of their neighbouring nodes and a 1D filter to get the result as one time of convolution. This method seems promising, but actually there will be some question that remains hard to solve. The main problem is, we do not know the position of neighbours, and the result of the convolution strongly depends on them. For a 3×3 filter applied on a 2D regular convolution, the 9 weights in filter represent the influence of different position. For example, a vertical filter set the upper row to 1, the bottom row to -1 and middle row to 0. If the position information is ambiguous in the unstructured convolution, and for every nodes the neighbour position is different, the result cannot be ideal.

7. Conclusions

In this paper, we have presented the new Opal-AE, Python-based, flexible, and powerful NIROM compression method that provides a suite of settings for compression of flow and other solution variables.

We interpolate the solution variables from an unstructured finite element mesh to a structured mesh in order to use a convolutional auto-encoder designed for imaging. We also implemented a domain decomposition version of the convolutional auto-encoder so that it may be efficiently applied to unstructured meshes with vastly different mesh resolutions.

We have proved the effectiveness of the auto-encoder with flow past a cylinder test cases. The results show that the domain decomposition convolutional autoencoder method is a promising ap-

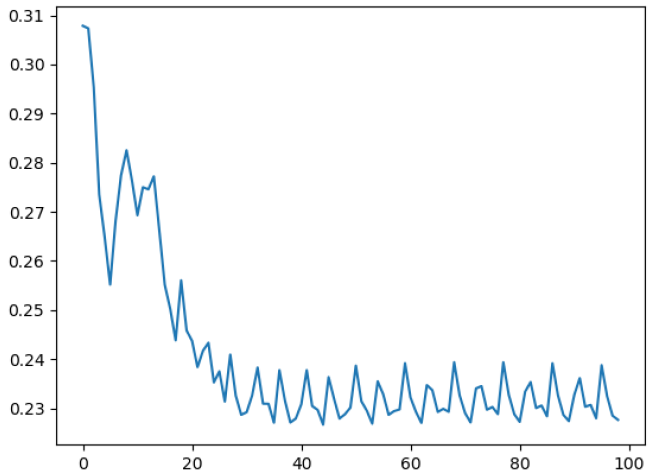


Figure 28: L_2 for 2 domains density domain decomposition, balancing the area density

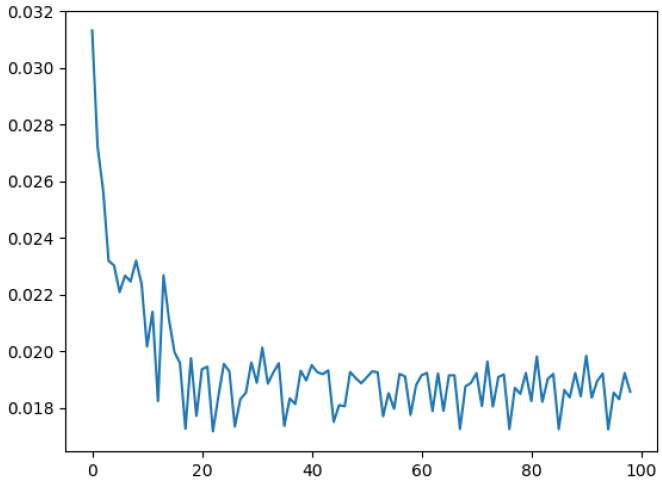


Figure 29: L_∞ for 2 domains density domain decomposition, balancing the area density

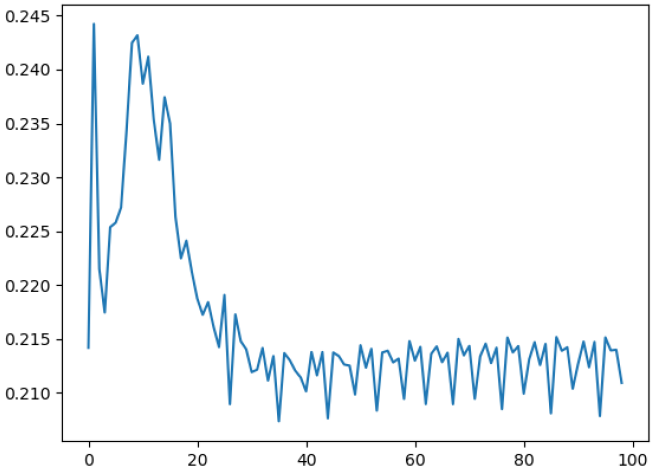


Figure 30: L_2 norm for 4 domains using using subdomains that have approximately the same number of nodes, and balancing the area density

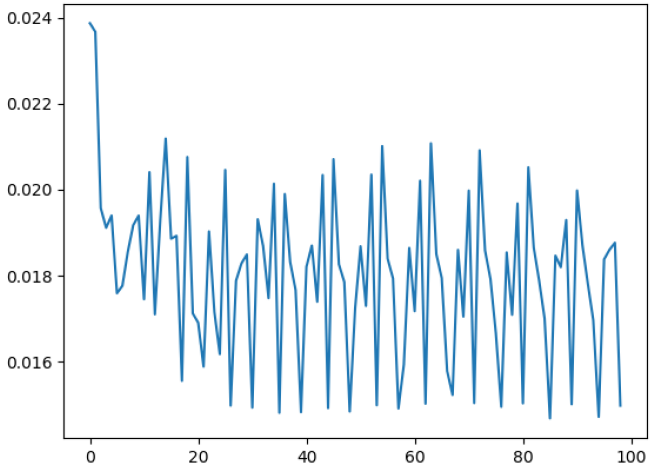


Figure 31: L_∞ norm for 4 domains using subdomains that have approximately the same number of nodes and subdomains that balance the area density

proach for compression on unstructured meshes. As the number of subdomains increases the results become more accurate.

[1] Y. Liang, H. Lee, S. Lim, W. Lin, K. Lee, C. Wu, Proper orthogonal decomposition and its applications-part i: Theory, Journal of Sound and Vibration 252 (3) (2002) 527–544.

[2] F. Gonzalez, M. Balajewicz, Deep convolutional recurrent autoencoders for learning lowdimensional feature dynamics of fluid systems, arXiv preprint arXiv:1808.01346.

[3] Z. Wang, D. Xiao, F. Fang, R. Govindan, C. C. Pain, Y. Guo, Model identification of reduced order fluid dynamics systems using deep learning, International Journal for Numerical Methods in Fluids 86 (4) (2018) 255–268.

[4] T. Lassila, A. Manzoni, A. Quarteroni, G. Rozza, Model order reduction in fluid dynamics: challenges and perspectives, in: Reduced Order Methods for modeling and computational reduction, Springer, 2014, pp. 235–273.

[5] D. Xiao, C. Heaney, L. Mottet, F. Fang, W. Lin, I. Navon, Y. Guo, O. Matar, A. Robins, C. Pain, A reduced order model for turbulent flows in the urban environment using machine learning, Building and Environment 148 (2019) 323–337.

[6] D. Xiao, C. Heaney, F. Fang, L. Mottet, R. Hu, D. Bistrian, E. Aristodemou, I. Navon, C. Pain, A domain decomposition non-intrusive reduced order model for turbulent flows, Computers & Fluids 182 (2019) 15–27.

[7] R. Pinnau, Model reduction via proper orthogonal decomposition, in: Model order reduction: theory, research aspects and applications, Springer, 2008, pp. 95–109.

[8] K. Willcox, J. Peraire, Balanced model reduction via the proper orthogonal decomposition, AIAA Journal 40 (11) (2002) 2323–2330.

[9] G. Berkooz, P. Holmes, J. L. Lumley, The proper orthogonal decomposition in the analysis of turbulent flows, Annual Review of Fluid Mechanics 25 (1) (1993) 539–575.

[10] J. B. Tenenbaum, V. De Silva, J. C. Langford, A global geometric framework for nonlinear dimensionality reduction, Science 290 (2000) 2319–2323.

[11] L. Theis, W. Shi, A. Cunningham, F. Huszár, Lossy image compression with compressive autoencoders, arXiv preprint arXiv:1703.00395.

[12] A. Ng, et al., Sparse autoencoder, CS294A Lecture notes 72 (2011) (2011) 1–19.

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

[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, Journal of Machine Learning Research 11 (Dec) (2010) 3371–3408.

[15] A. Krizhevsky, G. E. Hinton, Using very deep autoencoders for content-based image retrieval., in: ESANN, 2011.

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

[17] K. T. Carlberg, A. Jameson, M. J. Kochenderfer, J. Morton, L. Peng, F. D. Witherden, Recovering missing cfd data for high-order discretizations using deep neural networks and dynamics learning, Journal of Computational Physics.

[18] D. D'Agostino, A. Serani, E. F. Campana, M. Diez, Deep autoencoder for off-line design-space dimensionality reduction in shape optimization, in: 2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, 2018, p. 1648.

[19] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural computation 1 (4) (1989) 541–551.

[20] M. D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European conference on computer vision, Springer, 2014, pp. 818–833.

[21] C. S. Burrus, T. Parks, Convolution Algorithms, Citeseer, 1985.

[22] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Tech. rep., Citeseer (2009).

[23] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436.

[24] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 52–59.

[25] K. Kashima, Nonlinear model reduction by deep autoencoder of noise response data, in: 2016 IEEE 55th Conference on Decision and Control (CDC), IEEE, 2016, pp. 5750–5755.

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

[27] F. Chollet, et al., Keras, `https://github.com/fchollet/keras` (2015).

[28] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Van-

houcke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL http://tensorflow.org/

[29] M. Jackson, J. Percival, P. Mostaghimi, B. Tollit, D. Pavlidis, C. Pain, J. Gomes, A. H. Elsheikh, P. Salinas, A. Muggeridge, et al., Reservoir modeling for flow simulation by use of surfaces, adaptive unstructured meshes, and an overlapping-control-volume finite-element method, SPE Reservoir Evaluation & Engineering 18 (02) (2015) 115–132.

[30] C. C. Pain, C. De Oliveira, A. J. Goddard, A. Umpleby, K-way neural network graph partitioning with separator vertices, Biological cybernetics 80 (4) (1999) 227–234.

[31] P. Vanek, J. Mandel, M. Brezina, Algebraic multigrid on unstructured meshes, University of Colorado at Denver, UCD= CCM Report (34).

[32] W. J. Schroeder, K. M. Martin, W. E. Lorensen, The design and implementation of an object-oriented toolkit for 3d graphics and visualization, in: Proceedings of Seventh Annual IEEE Visualization'96, IEEE, 1996, pp. 93–100.

[33] D. Pedamonti, Comparison of non-linear activation functions for deep neural networks on mnist classification task, CoRR abs/1804.02763.