Model Reduction Using Machine Learning Methods with Domain Decomposition for Fluid Flow Problems

Author: Yunzhen (Wade) Song
Supervisors: Christopher Pain, Claire Heaney
Imperial College London, South Kensington Campus London SW7 2AZ, UK

Abstract

Reduced order modelling is a powerful technique for rapidly modelling high dimensional fluid dynamics systems. Its speed could enable real-time decision making and operational modelling. Using Long Short Term Memory(LSTM) neural network to train the data in reduced space can obtain excellent prediction in a short time. GPR and domain decomposition GPR is also used to train the data in reduced space, which also give accurate result. The full process of reduced order modelling is integrated together and was implemented in both 2D and 3D problems governed by Navier-Stokes equation.

Keywords: Reduced Order Modelling, Long Short Term Memory Neural Network, Domain Decomposition

1. Introduction

1.1. Motivation

The Reduced Order Model (ROM) can exploit Domain Decomposition (DD) methods in order to increase its accuracy and further reduced the computational time for a large system [1, 2, 3], see Figure 1 & 2. ROM has an advantage in speed to make the real-time simulation of detailed fluid flows possible. Its speed could help with, for example, decision making in problems requiring the solution of large systems of equation eg weather for casting.

1.2. Problem statement

Long Short Term Memory (LSTM) neural networks and Gaussian Process Regression (GPR) have demonstrated high accuracy on time-dependent deterministic problems. They have been widely adopted in stocks price prediction, driver-less cars, chess playing, translation and speech recognition. The memory possessed in LSTMs is particularly important for model reduction and will be used for large scale problems here. For example, these methods may be able to resolve the flows within a building while simultaneously resolving the flows within an entire city. This project will develop LSTM networks and GPR machine learning by combining them with Domain Decomposition methods to produce a new Domain Decomposition Long Short Term Memory Neural Network DD-LSTM or DD-GPR. This has an increase recurency over the standard LSTM as one needs to iterate between the sub-domains

in order to dis- tribute the sub-domain solutions across the whole domain. This is important in order to satisfy the impressibility constrain in many fluid flow problems.
The project will be divided into three steps:
The first step is using an existing High Fidelity Model (HFM) based on discretizations of the governing equations, such as using the finite element method for solving the advection equation or the Navier-Stokes equations that govern fluid flows. The data comes from the snapshots taken at different time levels of the HFM. The HFM comes from IC-Ferst is the result of solving these continuity equations and momentum equations, see equations 1, 2. Continuity equation:

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

Momentum equation:

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

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 0.01 and the uniform (on the left) inlet velocity 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.
Dimensional reduction methods will first be used to compress to reduce the number of independent variables that are solved for every time step. Proper Orthogonal Decomposition (POD) reduction methods, see [4], are used here but other new methods such as Auto-Encoders (AE) could also be used.
The second step is to use LSTM to make the prediction of future time steps. In this step, I will apply LSTM to three test case of increased complexity. The first case is to use LSTM to do prediction on a simple advecting square wave. I am then using LSTM to do prediction on flow past a cylinder. In the final case, I will apply LSTM to predict the urban airflow in a small region of London. I hope to compare LSTM with Gaussian Process Regression (GPR).
The third part is to use the domain decomposition method to divide a large area into several small domains, using LSTM in each sub-domain and dealing with transporting airflow in adjacent sub-domains, see Figure 2. I hope to show an enhanced predictive ability of the resulting DD-LSTM over a more standard LSTM.

1.3. Literature review and proposed approach

1.3.1. Non-Intrusive Reduce Order Modeling (NIROM)

Intrusive Reduce Order Modeling (IROM) and NIROM are two types of ROMs. NIROM does not depend on the source code of the HFM. Moreover, it can avoid some of the instability and non-linearity efficiency issues associated with IROM [1].
NIROM has been used to solve various flow dynamic problems. Both POD and AEs have been employed by multiple NIROMs in previous studies. Coefficients of the reduced basis functions can be recovered by the decoder part of the AE network [5] or through the POD's use of Singular Value Decomposition rotation matrices. Those coefficients will be used to recover the full model.
NIROMs can be divided into two stages, *offline* and *online*. The *offline* stage is responsible for dimensionality reduction and generates a reduced basis from the data. The *online* stage handles the prediction based on the re-

duced basis on the current time step, then recovers the solution to the problem in the full dimensional space[6].

Wang et al. proposed a non-intrusive POD reduced basis method for parametrised unsteady flows [5]. Xiao et al. developed a NIROM for predicting the turbulent air flows found within an urban environment [1]. Xiao et al. developed a Domain Decomposition Non-Intrusive Reduced Order Model (DDNIROM) for turbulent flows [2]. Xiao et al. presents a DDNIROM for the Navier-Stokes equations to improve the capability of NIROM for complex flow problems over widely varying ranges of scales [3]. Xiao et al. modelled turbulent flow problem and an ocean gyre simulation [1, 6]. In this project, similar NIROMs will be applied to the three test cases.

Here we are using data generated by the HFM. The NIROM is constructed by training a neural network using LSTM with data from the HFM. The data-driven approach, as well as being applied here, has also been used in a number of disciplines. Chinesta et al. used a data-driven method on linear and nonlinear elasticity [7]. Eggersmann et al. extended the Data-Driven formulation to the elasticity problems studied by Kirchdoerfer and Ortiz. They benefited from the data-driven method by generalizing the model to handle different scenarios [8]. The data-driven approach has also been used in computational mechanics and nonlinear elasticity [9, 10].

Capuano et al. implemented the data-driven method on finite element formulations and demonstrated that the data-driven process could work with any machine learning and proved its effectiveness in reducing both the error and computational cost, see [11].

1.3.2. Long short term memory neural network

LSTM is a Recurrent Neural Network (RNN) for overcoming the vanishing of gradients, and it is formed by an explicit memory cell and four gating units. Vanishing of gradients occurs when back-propagating errors across many time steps. But for LSTM, a self-connected recurrent edge for each memory cell will ensure the gradient will not vanish after passing across many times. This structure allows LSTMs to control the sequential information in the network [12, 13].

LSTM is widely used in time series problems[14], such as stock price prediction,[15, 16] diseases propagation,[17] and climate change prediction [18]. LSTMs are also commonly used in speech recognition[19, 20, 21], natural language processing[22, 23], and classification problems[24, 25, 26].

As shown in Figure 1, the latest hidden-layer output is h_{t-1} , the latest memory activation is c_{t-1}. The latest memory activation is an input of the current LSTM cell[21].

The general LSTM use these four units: (Note: The output of an LSTM cells with l-th layer and time j is $h^l_j = o^l_j \tanh(c^l_j)$, b here is the bias term.)

(i) Memory units: store the temporal information.

The activation vector of Memory units:

$$c^l_j = f^l_j c^l_{j-1} + i^l_j \tanh(W^l_{xc} x^l_j + W^l_{hc} h^l_{j-1} + b^l_c) \quad (3)$$

(ii) Input gates: modulate the input activations into the cells.

The activation vector of input gates:

$$i^l_j = \sigma(W^l_{xi} x^l_j + W^l_{hi} h^l_{j-1} + W^l_{ci} c^l_{j-1} + b^l_i) \quad (4)$$

(iii) Output gates: modulate the output activations of the cells.

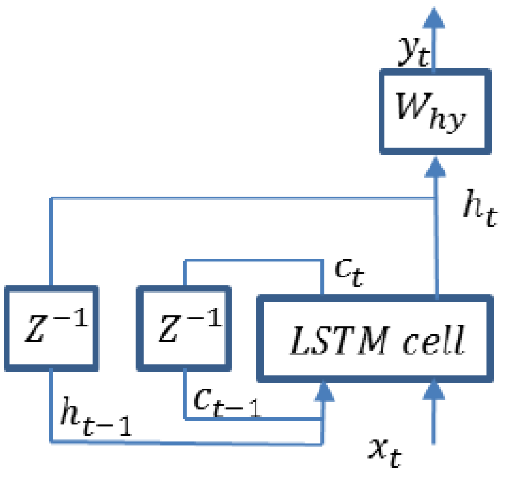


Figure 1: An LSTM Cell with One Recurrent Layer. Z^{-1} is the Time Delay Node

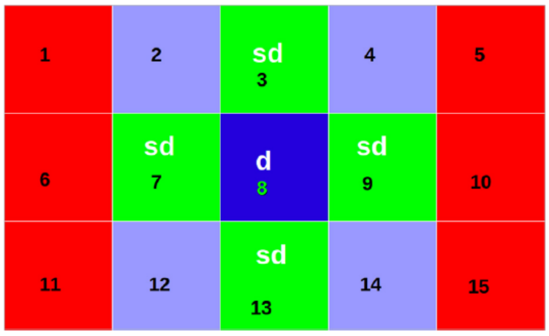


Figure 2: The figure shows a sub-domain d with its four neighbouring sub-domains [3]

The activation vector of output gates:

$$o_j^l = \sigma(W_{xo}^l x_j^l + W_{ho}^l h_{j-1}^l + W_{co}^l c_j^l + b_o^l) \quad (5)$$

(iv) Forget gates: reset the cells memory.
The activation vector of forget gate:

$$f_j^l = \sigma(W_{xf}^l x_j^l + W_{hf}^l h_{j-1}^l + W_{cf}^l c_{j-1}^l + b_f^l) \quad (6)$$

In this project, LSTM is used to predict the compressed velocity time series [27]. For that, it is trained to learn the low dimensional dynamics of the fluid system. See Francisco and Maciej who used a single loss function for losses from both AE and LSTM. In their *offline* training process, the AE first takes N dimensional data and outputs the low dimensional representation. After the AE finishes the forward pass and constructs a batch of low-dimensional representations, the LSTM is trained with those representations. In their *online* prediction process, the trained AE retrieves the predictions from the LSTM in lower dimensions. The AE then uses the decoder in a forward pass to recover the predictions in the original dimension[27]. A similar structure will be used in this project, but we will introduce domain decomposition along with LSTM for implementation onto larger systems and call this method DD-LSTM. In addition, in this project, we decouple the compression (with either POD or AE) and the DD-LSTM *offline* training.
Gaussian Process Regression (GPR) has also be used in a similar way to LSTMs in previous studies, [28, 29]. LSTMs are suitable for dealing with sudden changes of the solution variables following relatively long periods of little change in the variables. The GPR method has flexibility but is limited to the number of dimensions it can form a hypersurface in. At least more limited potentially than LSTMs and feed-forward neural networks. In wind flow forecasting both LSTM and GPR achieved higher forecasting accuracy than the conventional forecasting methods like Autoregressive Integrated Moving Average model and Back-Propagation Neural Networks [29]. When forecasting high-dimensional chaotic system, the LSTM can better capture the nonlinear dynamics compared to GPR, while GPR has the better speed to obtain the same result than LSTM (20%

percent better). As for computational complexity, the computational cost of the GPR-based approaches is significantly larger than LSTM. It was found that in short-term predictions, LSTM performs better. However, the prediction error increases faster for LSTM when increasing the level of chaos in the system [30].

In setting up LSTM neural networks the general trend is to use only a small number of hidden layers, single-layer LSTM is used in learning low-dimensional feature dynamics of fluid systems [27]. The number of hidden layers is not mentioned in ocean gyre and flow past cylinder simulation studies, see [6], however more than one layer is used in that work. Twenty hidden neurons and single hidden layer is used in the chaoticity test case of [30]. In contrast to time dependent problems, in classification problems, classification accuracy is related to the number of hidden layers [26].

1.3.3. Domain decomposition

DD is used to solve large problems in the frequency and time domains. The main feature of DD is decomposing large problems into several smaller ones. Instead of relying on iterative techniques in conventional DD, we choose to transform the original large matrix into one whose size is small enough to be manageable[31]. DD is often used with model reduction methods to conquer large problems[32].

Machine learning methods can be used to predict the geometric boundary for each domain. It reduces the number of eigenvalues before the iteration[33].

Multi-grid partitioning methods can be used to decomposed large finite element meshes into several sub-domains. Combining with automatic grid coarsening, unstructured meshes can also be decomposed. The sub-domain boundaries discourage the high crossing activity and poor conditioning but improve load balance[34]. In this project, the multi-grid partitioning method will be used to decompose the unstructured mesh.

Domain decomposition has been implemented to NIROM for modelling fluid flow and turbulent flow problems. Using Domain Decomposition methods allows one to construct local basis functions based on details of local flow solutions over each sub-domain. A Radial Basis Function (RBF) based NIROM is implemented, as several local basis functions are generated at first (by POD), RBF multi-dimensional interpolation method is then used to construct a set of hypersurfaces representing the local fluid dynamics over this sub-domain. When building the hypersurfaces for a given sub-domain, the solution in the surrounding sub-domains is taken into account by introducing their POD coefficients as inputs to the hypersurfaces of this particular sub-domain. DD-NIROM is suitable for hard problems which global Singular Value Decomposition may not decrease rapidly while its small sub-domains do[3].

The DD-LSTM method updates a sub-domain by taking the current value from itself and the future value from all its neighbours as the input, getting the output to update itself. In this case, as the input structure of every sub-domain is different because of the unstructured mesh, there will be the same number of LSTM models as the number of sub-domains, shown in Figure 2.

Xiao et al.[2] has used GPR and DD to form their NIROM. The approach developed in this project is similar, but the LSTM replaces the GPR. In Xiao et al. the Reynolds stresses are used to decided how to weight each node of the mesh. The aim then is to balance the weighted sum in each sub-domain and

minimize the communication or sum of the weights between the sub-domains. In this way, the sub-domains are equally balanced, in terms of the predictive ability, and the sub-domains form (as much as possible) dynamically isolated regions (sub-domains).
In our approach, the *offline* training process, first, divides up the domain into sub-domains, then forms the reduced order model the DD-LSTM is trained for each sub-domain separately. In the *online* process, the DD algorithm loops over all time steps, and forms the reduced order basis for each sub-domain separately. Then iterating over all the sub-domains the DD-LSTM is able to predict future time levels. Finally, project the reduced order solution is compared with the high fidelity solution.
The work flow for this project is shown in Figure 3:

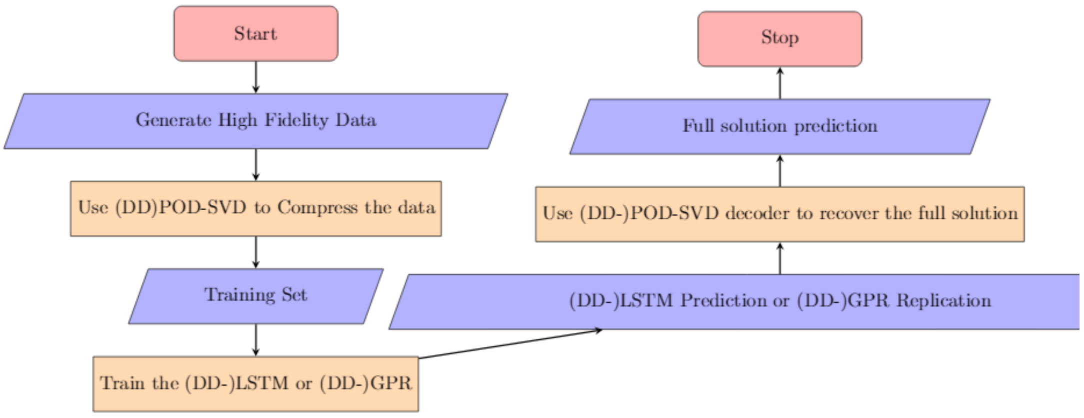
2. Software Development Life Cycle

2.1. Existing code ecosystem

The new function is developed based on IC-ferst and opal.
IC-ferst is an existing calculation tool for fluid mechanics model. IC-ferst won't be directly used in this program, but there are several modules which will be called during the process of reduced order modelling. Another essential role of IC-ferst is that generates the original data for training and testing. The input of IC-ferst is a mesh file and an mpml file. These file contents the information of the fluid mechanics problem. IC-ferst will use those file to calculate several crucial variables like velocity, pressure, viscosity, etc. In this project, velocity data will be used as the input of ROM.
Opal is a tool that allows one to interface with IC-FERST/Fluidity and can perform Reduced Order Modelling (ROM). There are some pre-exist functions for dimensionality reduction like POD and function for training low dimensional basis functions such as Gaussian Process Regression (GPR). If you want to use Opal, there's a UI software called diamond. The interface of Diamond is shown in Figure 4.
Using the Diamond, you can change the configuration of Opal. After opening the user interface of diamond, you can set the options you require. The default configuration is based on the best practice we currently have, but it's also possible to change these values. Previous to this work the method for reduced order modelling in Opal is based on POD and Gaussian Process Regression (GPR).
The data structures for input and output are already defined by vtktools from IC-ferst/Fluidity. It allows Opal to read in vtu files generated by IC-ferst/Fluidity and write the resulting predictions into vtu files.
The visualisation software of vtu files is paraview. Using paraview we can directly see the animation of the solution variables in vtu files.
A CPU efficient Fortran library was used for decomposing the domain which is used within Opal; it needs the information in the original datas vtu files to decide how to decompose the unstructured mesh. It feedbacks a list of number which is the label of domain for each node, along with a CSR representation matrix for allocating the position of nodes in unstructured mesh.

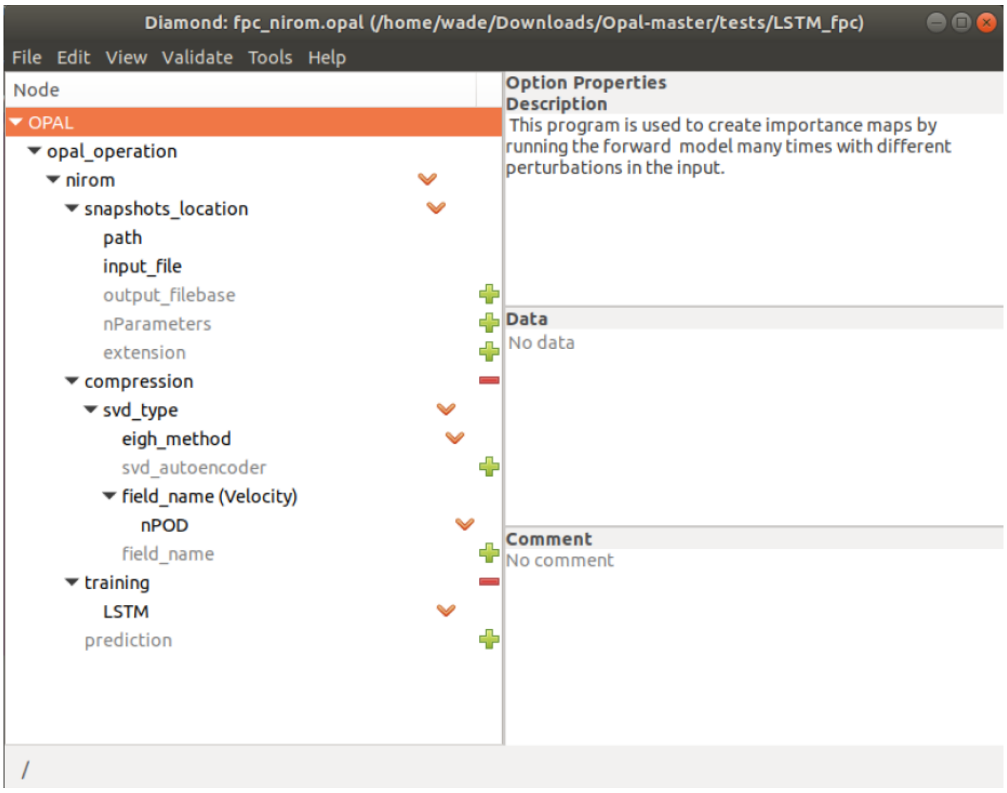
2.2. Development methodology

During the development of a new software function, Waterfall is used as the operation tools. As the requirement of software is particular, and there will be no customer complaint about the functionality of software, use



Start
Generate High Fidelity Data
Use (DD)POD-SVD to Compress the data
Training Set
Train the (DD-)LSTM or (DD-)GPR
(DD-)LSTM Prediction or (DD-)GPR Replication
Use (DD-)POD-SVD decoder to recover the full solution
Full solution prediction
Stop

Figure 3: Algorithm describing the application of the (DD-)LSTM or (DD-)GPR to an example problem.



Diamond: fpc_nirom.opal (/home/wade/Downloads/Opal-master/tests/LSTM_fpc)
File Edit View Validate Tools Help
Node

OPAL
opal_operation
nirom
snapshots_location
path
input_file
output_filebase
nParameters
extension
compression
svd_type
eigh_method
svd_autoencoder
field_name (Velocity)
nPOD
field_name
training
LSTM
prediction

Option Properties
Description
This program is used to create importance maps by running the forward model many times with different perturbations in the input.

Data
No data

Comment
No comment

/

Figure 4: Interface of the Diamond, functions on the left hand side and details on the right hand side

waterfall will be efficient. When the project plan is finished, I had already known the requirement of the software. As the new function should be integrated with Opal, I followed the rotational of how Opal was developed. So that user will be free to choose from formal Opal options or my new options. The new function which includes domain decomposition, LSTM and domain decomposition GPR were implemented one by one, each passed the test first, and then test the Opal with two and three parts together. After developing the mean functions of Opal, more choices were added to the UI which call diamond, and user will benefit from that to pick the function and value without entering the code.

2.3. Design rationale

The design of software inherited the idea of Opal, which divided the tasks into four parts. The first part is reading in the configuration and vtu files. After user defining the configuration in diamond, there will be a .opal file, which stores those settings. The settings include the path to the dataset files, the compression, training, predicting method, and the detail of them. Then, the vtu files in the path mentioned above will be read in. Here the are two class storing different functions of variable. One called fwd_options which saves all configurations, and would never be changed during the whole process. Another called nirom_options will be updated by what is obtained during the process. Here, the vtu data for velocity is stored in a variable called snapshots in nirom_options, which can be used in any part later.
The second part is compression. In this part, there are several options for users. Compression is one of the most critical components for reduced order modelling, and using POD is one of the fastest and effective ways of doing that. There are two options suiting for different situation. One called eigenvalue method is used for globally reduced the dimension. Another is called DD eigenvalue method, which is used for doing domain decomposition and dimensionality reduction. The first one, eigenvalue method, give user choose how many POD coefficients to maintain after compression. The second one provides user with number of sub-domains. To keep the DD efficient, user can only select that number which is the pow of 2. When DD eigenvalue method is used, in each subdomain, it is treated as a whole domain and POD will be used for it to obtain POD coefficients with the number set by user.
The third part is training. User should choose from LSTM or GPR if they have chosen eigenvalue method in compression, and select DD-LSTM or DD-GPR when they have chosen DD eigenvalue method. For the global one, LSTM or GPR will take the compressed pod coefficients as input and train the neural network. And for the DD method, for each sub-domain, it has an LSTM neural network or GPR neural network for only itself, the input for training comes from the compressed pod coefficients of itself and its neighbours. More details will be introduced in the Implementation Strategy part.
The last part is the prediction and output. For the global method, the input is several time steps continuous pod coefficients. For LSTM, one prediction will be made by the trained neural network, and then they'll become input for future time steps. Here the GPR is not doing the prediction, but a replication of original data, but it's using the trained neural network of GPR to do replication. As for DD method, the logic is similar, and the only difference is of each sub-domain; it takes historical data from itself and prediction data from its neighbours.

2.4. Implementation strategy

2.4.1. 1D square wave LSTM

I start with implementing an LSTM to understand a one dimension square problem. The 1D problem is used to verify that the LSTM neural network can carry out the prediction of time series 1D time series problems. This then provides the confidence needed before applying it to 2D and 3D problems.

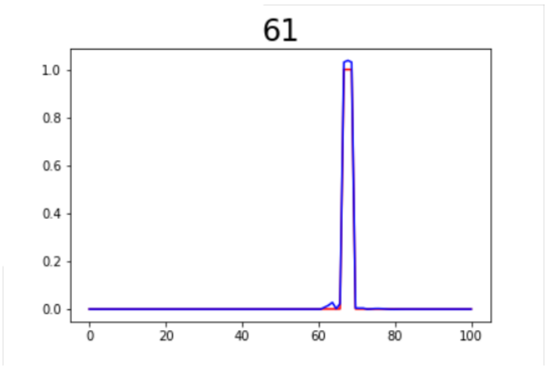


Figure 5: The animation snapshots during the training process of 1D square wave, with one layer and 50 neurons LSTM

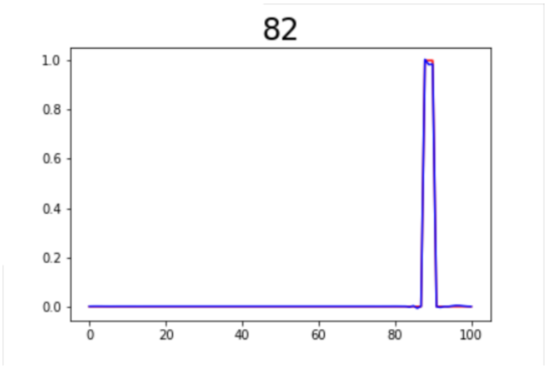


Figure 6: The animation snapshots for the prediction of 1D square wave, with one layer and 50 neurons LSTM

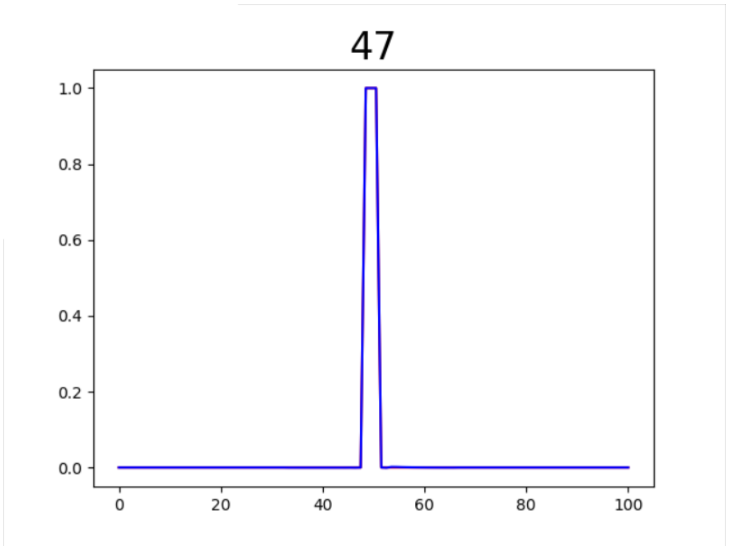


Figure 7: Train Performance of 1D DD-LSTM in 47th Step

The domain is divided into a 100 cells and each cells is given a 0 or a 1 as a representation of a square wave at a single instance in time. We then move this square wave to the right preserving its shape. The width of the square wave is 3 cells - that is 3 cells have a value of 1 associated with them and all others 0. The whole data-set of the 1D square ware is implemented using the Toeplitz matrix, [35]. By using this, its easy to change the width of the square for example.
Since this problem is relatively simple, I used an LSTM with one recurrent layer with 50 hidden neurons. As the feature for this square wave is straightforward, using only two neurons in only one hidden layer can also result in reasonable result.

2.4.2. *1D square wave DD-LSTM*

In this part, the 1D square wave is manually partitioned into several sub-domains. Each sub-domain has an LSTM within it which takes the input from both itself and its neighbouring subdomain LSTM's. Since we require preparing our method to be implicit in order to be able to apply it to in-compressible fluid flow problems where only implicit methods of solving for pressure can spread the pressure information across the domain every time step to satisfy the in-compressability constraint. That is each sub-domains' state will be affected by the sub-domains which is not its neighbour. But by iterating several times, each time each sub-domain takes the input both from the current value from itself and the prediction value from its neighbours. In this way the DD-LSTM demonstrated that it could learn the dynamics of the propagating 1D square wave. See figures 5, 6, 7.

2.4.3. 2D flow past a cylinder

In order to solve fluid flow problems we integrated the new reduced order models into the new Opal model framework. Opal is developed in Linux system. To add a function to Opal, I need to modify the class and the used interface .rng files used by the DIAMOND user interface, 4. A separate module for LSTM's neural network and training along with predicting is added and connected with Opal. So here I'm using one layer and 10 neurons for LSTM. Each time step the LSTM takes reduced variable (e.g. from POD) at 3-time previous levels. This is used to train the LSTM as well as to predict with it. See figure 8. There are 11136 nodes and 2 velocity components (velocity u and v) for the 2D flow past a cylinder problem. I ran the 2D flow past the cylinder problems on a laptop with 8 GByte of memory and with i7-8750h of CPU.
GPR takes reduced variable at single time previous level. And this is used train the GPR as well as replication with it.
For the input with 10 POD coefficients, each POD coefficient can also be visualised. Each one of them gives information to the LSTM. And the way to choose how many POD to use is also observed if there are new features in the POD with larger index. Typically the POD coefficients will contain more information. There I show six of them in figures 9, 10, 11, 12, 13, 14.
After the training of LSTM, it can be used to predict and time step's POD coefficients. The logic of predicting is the same as the 1D problem, see figure 15.
The LSTM only need 3 original data, and it uses those three data to get the first prediction in the first step. So in the second step, there are 2 original data and one prediction data and receives the second prediction. According to this process, in step 4, the input data for LSTM is only coming from prediction data, which makes it possible to predict whatever given time step.

2.4.4. 2D flow past a cylinder using DD methods

According to the benefits which have been talked about in introduction part, I implemented domain decomposition to this flow past a cylinder problem.
The first step of domain decomposition is to chop the unstructured mesh into several sub-domains. There's an existing Fortran file could do domain decomposition. So the first step is to make connection between the Fortran module and python module. This was achieved by a numpy function which transfers the .f90 file into a .so file. After that, the python module can directly import the .so file as a module, and the subroutines in Fortran would become the functions in python. The Fortran code would show which variables are inputs and which are outputs.
First, to obtain the Compressed Sparse Row matrix (CSR matrix), the Fortran module needs the coordinates for all nodes and the number of nodes, which can be obtained from

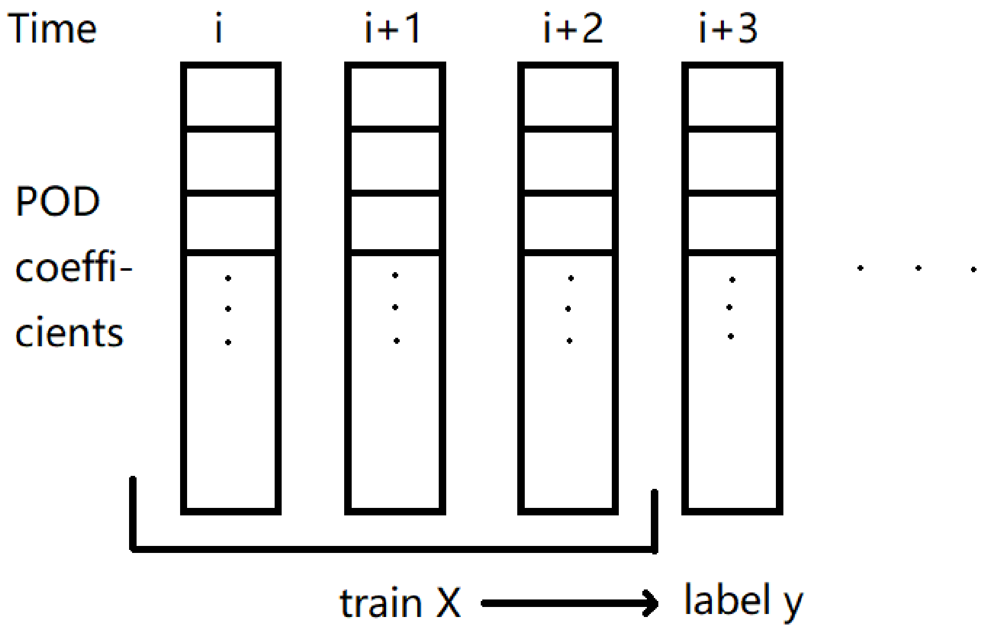


Figure 8: The method to obtain the training data X and label y for LSTM

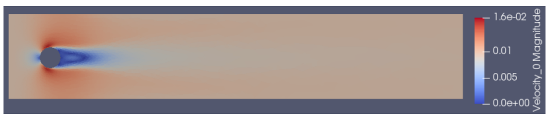


Figure 9: Recover the full model from POD coefficient 0 by basis functions, the most important feature

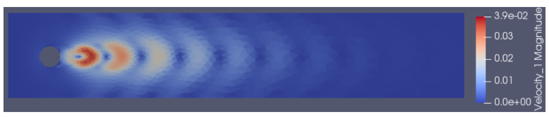


Figure 10: Recover the full model from POD coefficient 1 by basis functions, the second important feature

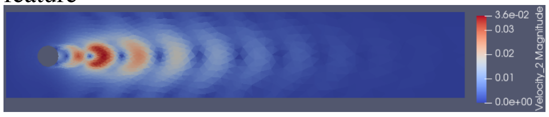


Figure 11: Recover the full model from POD coefficient 2 by basis functions

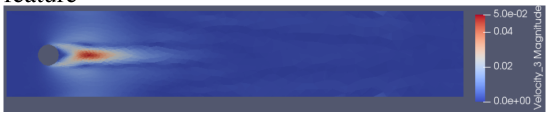


Figure 12: Recover the full model from POD coefficient 3 by basis functions

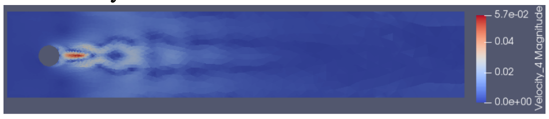


Figure 13: Recover the full model from POD coefficient 4 by basis functions

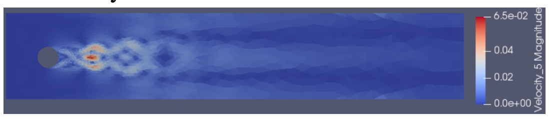


Figure 14: Recover the full model from POD coefficient 5 by basis functions

the vtu files. The CSR representation contains fina and cola. Fina is an array start with one, and store the cumulative number of data in each row. Cola stores the row position for each data [36].

After getting the expression of CSR matrix, I defined the other input such as the number of sub-domains, the weight of nodes and the

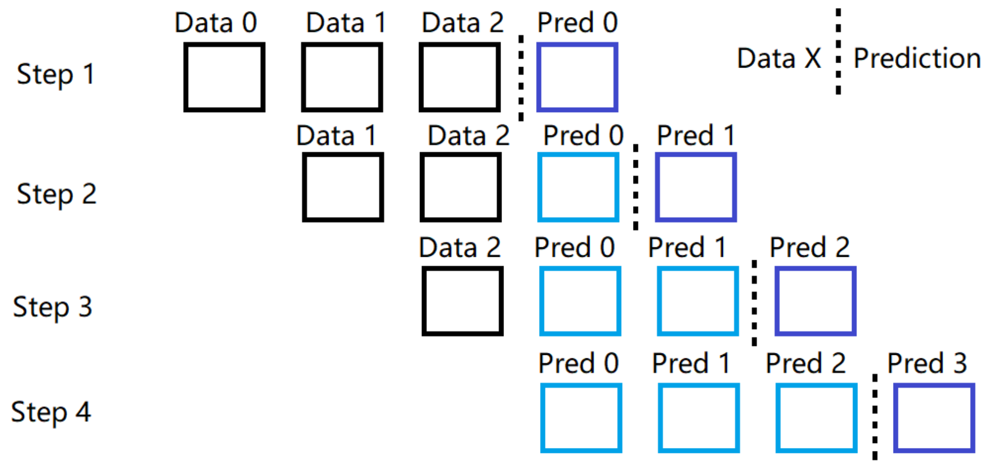


Figure 15: Prediction process of LSTM when history level = 3

rule of including or excluding a node. Using the Fortran module to get an array called 'whichd'. It's at the same length with the number of nodes. The value v of 'whichd' in position n means the node n belong to subdomain v. For efficiency, the number of subdomain will always be a power of 2. The result of 2, 4, 8, 16, 32, 64 sub-domains is shown in figures 16, 17, 18, 19, 20, 21.

Comparing with the mesh (figure 22), we know that the domain decomposition is based on the sparsity of the mesh. The area with dense nodes will be divided into smaller subdomains for the balance of all sub-domains. And the adaptive mesh is based on the activity and complexity of fluid so that after domain decomposition, each part of the sub-domains burden is lighter.

After obtaining whichd, the snapshots variable should be changed accordingly. The new dd_snapshots variable is an array with the number of sub-domains in shape 0, with the number of time steps in shape 1, and with the number of nodes in the corresponding subdomain in shape 2. The major problem here is the sequence of nodes is changed, and without recording, it'll be impossible to transfer dd_snapshots back to general snapshots. So there I use a loop with complexity O(n) to record the original position of each node for dd_snapshots.

After getting dd_snapshots, each subdomain's snapshots will be compression by POD. Here in implementation, I used same POD number in each sub-domain. The compression is same as the global method for each subdomains. Then I got a DD POD coefficient, which has the same shape as DD snapshots in shape 0 and 1. But in shape 2, DD POD coefficients' shape is the number of POD. For example, when it's chopped into 32 subdomains, and with 200-time steps, and with 10 as the number of POD, the shape of DD POD coefficient is (32, 200, 10).

Then is to use the method mentioned above to train the DD-LSTM or DD-GPR. Each subdomain's LSTM and GPR will get the data from itself and its neighbors, and iterate for several times. And the label for each steps is always the original data from one step later.

After training, the difference of LSTM and GPR is, LSTM can only take in several time steps' original data and predict the data in further time steps, while GPR does the replica-

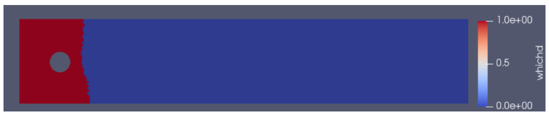


Figure 16: The domain division status after being decomposed to 2 sub-domains

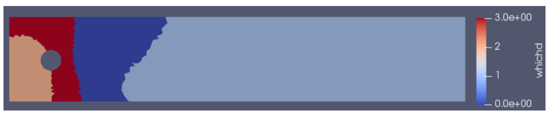


Figure 17: The domain division status after being decomposed to 4 sub-domains

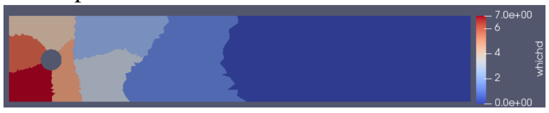


Figure 18: The domain division status after being decomposed to 8 sub-domains

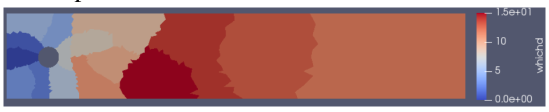


Figure 19: The domain division status after being decomposed to 16 sub-domains

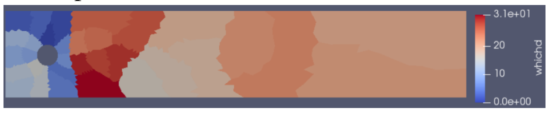


Figure 20: The domain division status after being decomposed to 32 sub-domains

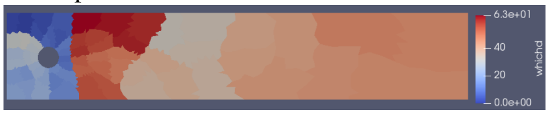


Figure 21: The domain division status after being decomposed to 64 sub-domains

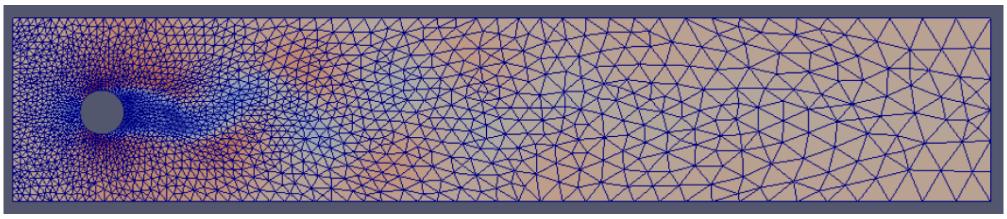


Figure 22: The finite element mesh of flow past a cylinder problem

tion of original training data. So here we can get the prediction from DD-LSTM and the replication of GPR.
The result of the replication of DD-GPR is good, while for the DD-LSTM, the result is bad. The reason will be discussed later.

2.4.5. 3D London South Bank University

In the 2D flow past a cylinder case, I've made the software flexible so that it can be directly fit for the 3D model. After changing the basic setting in diamond, we can train the LSTM and GPR to Learn the feature of this 3D model.
The major difference is this 3D problem is immense., there are 670975 points and three components (velocity u, v, w) for this 3D problem. I ran this problem on a *** workstation.
See the 3D mesh from outside and inside in figure 23, 24
After setting up the machine, I start by using the same configuration as the 2D problem to solve the 3D London South Bank University test case. We can get 10 POD coefficients for each time step by this configuration. For better visualise the feature, I cut a plane which is perpendicular to the Z-axis, with Z = 10. See figures 25,26,27,28.
Compare with the figure 29, the area with higher activity has more nodes, which is reasonable.
The training of the model is super quick as it just the same scale of input for LSTM or

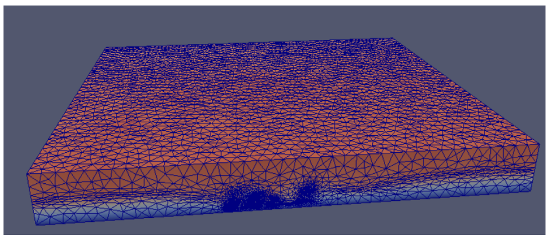


Figure 23: Watch the 3D mesh from outside of the model

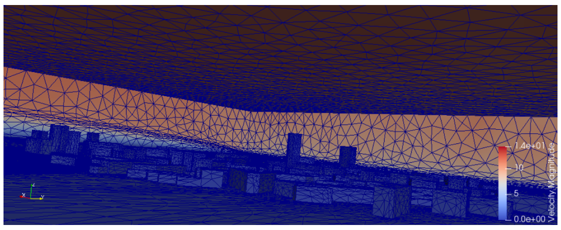


Figure 24: Watch part of the 3D mesh from inside of the model

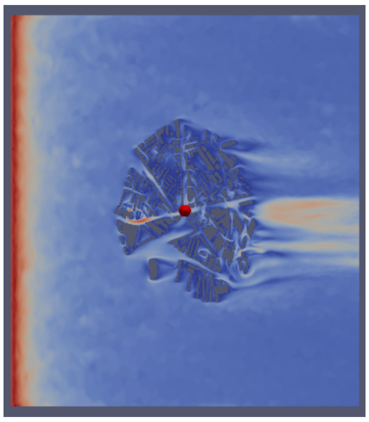


Figure 25: 3D POD coefficient 0 in a cut plane with Z = 10

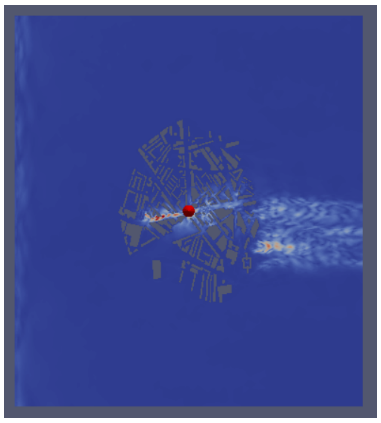


Figure 26: 3D POD coefficient 1 in a cut plane with Z = 10

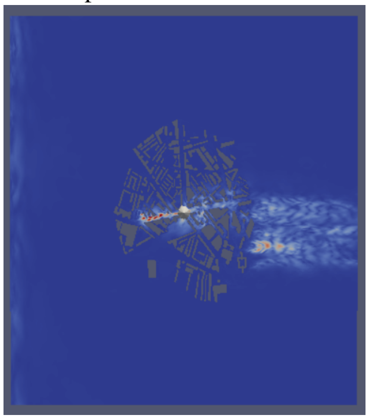


Figure 27: 3D POD coefficient 2 in a cut plane with Z = 10

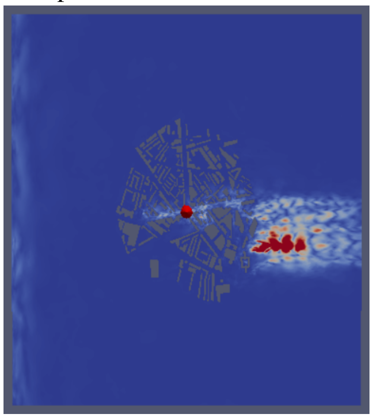


Figure 28: 3D POD coefficient 3 in a cut plane with Z = 10

GPR. While the output to file takes major time. As expected, the result of using same configuration as 2D problem is unpleasant. And to improve the performance, the most direct way is to use more POD coefficients. The way to implement this will be discussed in Implementation - section 4.

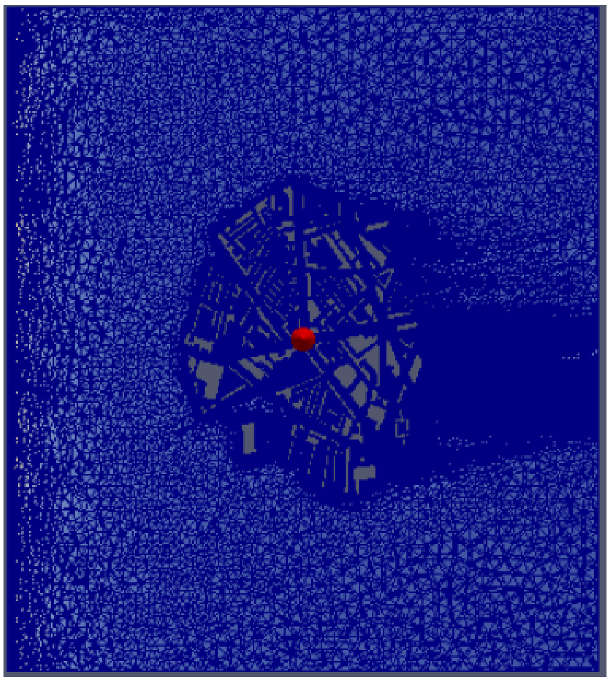


Figure 29: Mesh of 3D problem in a cut plane with Z = 10

Code metadata description	**Information**
Current code version	v3.2
Permanent link to code/repository used for this code version	$https://github.com/msc-acse/acse-9-independent-research-project-Wade003.git$
Legal Code License	MIT License
Code versioning system used	none
Software code languages, tools, and services used	python, fortran
Compilation requirements, operating environments & dependencies	Linux, python, numpy, sklearn, keras, tensorflow, pytorch, IC-ferst, Opal-spud
Developer documentation/manual	$http://mozart.github.io/documentation/$
Support email for questions	ys8718@ic.ac.uk

Table 1: Code metadata

3. Code metadata

3.1. Developing environment and setting Up

According to C7 in Table 1. In this section the commands to install these package are listed.

After setting up the Linux system, run these commands in the terminal.
pip install -U numpy
pip install sklearn
pip install keras
pip install tensorflow
pip install torch==1.2.0+cpu torchvision==0.4.0+cpu -f https://download.pytorch.org/whl/torch_stable.html
export PYTHONPATH ='/PATH TO IC-ferst/python:$PYTHONPATH'
export LD_LIBRARY_PATH =$LD_LIBRARY_PATH:/PATH TO Opal/spud
export PATH="/PATH TO Opal/spud:$PATH"
To run a 2D test case, you need to go to the directory in Opal/test/LSTM_fpc. In this directory, make sure there are a folder called snapshots, which has the original data for training. Then change the configuration if you want to. You can change you the setting in a user interface call diamond by entering:
diamond -s ../software/Opal-master/schemas/opal.rng fpc_nirom.opal
The file name should be changed to the opal file's name in another test case. As for the configuration of 2D problem, I recommend to remain the same; if you do need you change it and still want to test a 2D test case, you can change the nPOD in field_name(Velocity) in svd_type in compression. But too many or two small POD number would affect the performance.
Another setting could be changed is to change the training method from LSTM to GPR. We using GPR, it needs further setting. The scaling bounds are 0 10. The constant value is 1. The constant bounds are 1e-3 1e3. The RBF length scale is 100. The RBF length scale bounds are 1e-2 1e2. Ctrl + s to save your settings.
To run the model, you should enter the following commands in the terminal.
python2 ../software/Opal-master/opal.py fpc_nirom.opal
When it is finished, enter "paraview" in the command line to see the result. After opening the paraview, open the nirom_replication.vtu files, that would be the prediction of LSTM and the replication of GPR. Click apply, and click a dropbox showing solid colour, switch that to velocity. Then click a button to go to the second snapshots, click the re-scale button so you can see the result in reasonable colour. Then you can click play button to see the whole animation.
And if you want to run a DD test case, make sure you change both the method in compression and training to DD method. In compression, you should use dd eigh method and in training, I recommend you to use DD GPR method. You are free to choose the number of sub-domains. It will be the nth power of 2 (n is what you entered). And also don't use too large numbers which would lead to the number of nodes too small for sub-domains.
For 3D problem, the data is too large and we can hardly see the animation in paraview, which I don't recommend to use that as the test case. The total time for testing a 3D problem is more than 3 hours. And it needs a powerful computer. For 3D DD problem, it would be even more demanding.

4. Implementation

4.1. Simulation capabilities

For a different test case, the simulation capacity is different. The 1D test case is only capable of a fixed 1D problem, as the code of that is hard-wired. For 2D and 3D test case, there's no difference in code, which is flexible on any given test case. The performance will be affected by the configuration, such as the number of POD or the method.

4.2. Investigated test cases

As is discussed in Part 2 Software Development Life Cycle, there are many test cases in this project, for 1D, 2D and 3D with or without DD. I also briefly introduced the implementation of the code in that part. In this part, I'll show how did I implement the code to get the result and to analyze.

The functions used in the machine learning problems are:

Loss function: Mean square loss (Using torch.nn.MSELoss)

Learning rate: 0.02

Optimizer: Adam (Using torch.optim.Adam) [37]

4.2.1. Determining the sub-domains

When splitting the domain into sub-domains, the weight of each nodes will affect the division. The weight of node is determined by Reynolds stress[2]. The equation of calculating the weight of nodes is:

$$\lambda_i^{max} = \frac{1}{\beta} ln(\frac{\tau_i^{max}}{\alpha} + 1) \tag{7}$$

α and β are scalars, $\alpha = 0.05, \beta = 0.0019183$, λ_i^{max} is the weight of node i, τ_i^{max} is the maximum of the Reynolds stress tensor associated with node i.

Using the weights in equation **??** the subdomain partitioning balances, in some sense, the load/(complexity of the physics) associated within each sub-domain as well as minimizes the activity between the sub-domains, see [2].

4.2.2. 1D square wave test case

The result for 1D square wave predicting by LSTM is qualatively good. Using different hidden neuron sizes leads to differing accuracy. However, as long as you don't set the number of hidden neurons greater than the number of inputs - in this case the number of cells used to represent the 1D wave, then the results are also good.

For the 1D LSTM, during the train, I visualize the prediction made by the LSTM at each time step so that I can generate a real time animation that tests the accuracy of the LSTM. This is implemented using functions in matplotlib.pyplot, such as plt.ion(), plt.draw(), plt.pause(0.5) and plt.clf(). From this visualization it was observed that the training for the 1D DD-LSTM was quantitatively good, see figure 7.

4.2.3. 2D flow past a cylinder test case

The LSTM in Opal adapts its structure according to the given number of POD coefficients used. It adjusts the hidden layer size by setting the number of neurons in the only hidden layer to equal the number of POD coefficients.

The result for flow past a cylinder using LSTM is quantitatively good. Comparing the LSTM result with the original data they are visually identical. The GPR results using the same number of POD coefficients is also shown in figures 30, 31, 32.

For comparison, here I also put the result for DD-LSTM and DD-GPR in figure 33, 34.

The result from the DD-LSTM has poor accuracy using DD. The reason for this is that LSTM is a neural network base on time series data and every input for LSTM will influence the inner memory of the LSTM. As is discussed above, our LSTM for each subdomain not only takes inputs for one time but several times during the iteration. Several possible ways of solving this problem have been investigated. The first potential

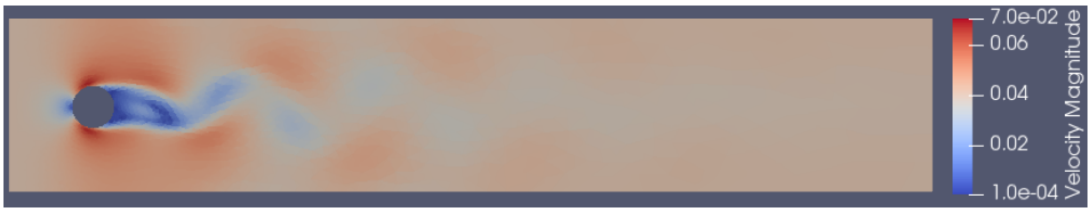


Figure 30: The velocity magnitude of HFM (Use this as the standard)

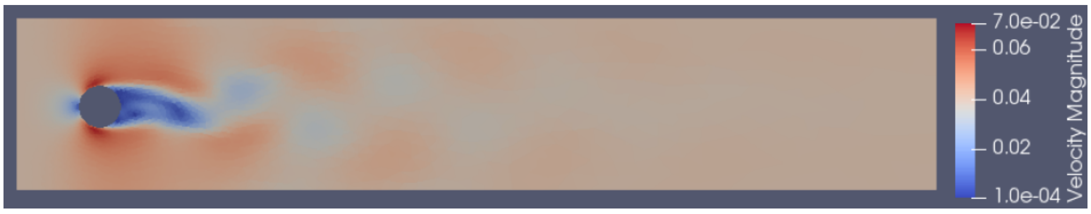


Figure 31: The velocity magnitude of the prediction by LSTM

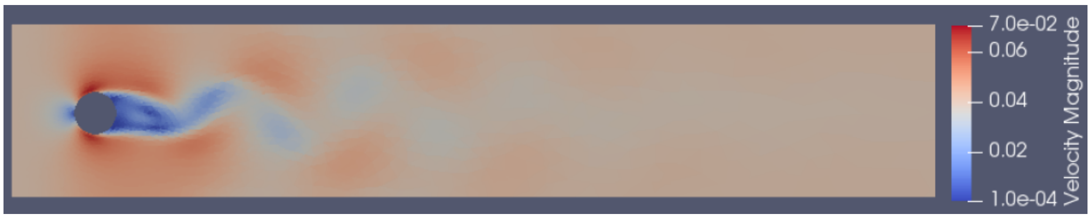


Figure 32: The velocity magnitude of the replication by GPR

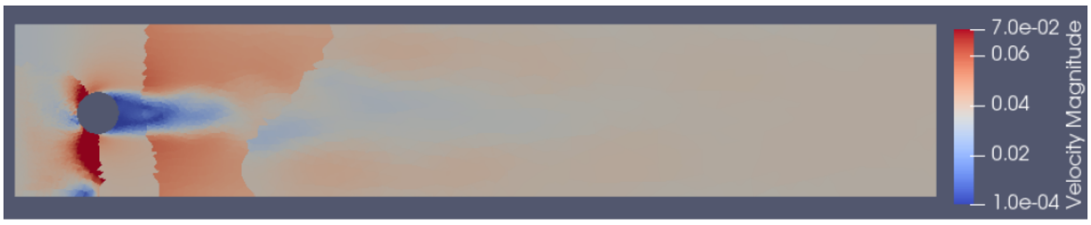


Figure 33: The velocity magnitude of the prediction by DD-LSTM

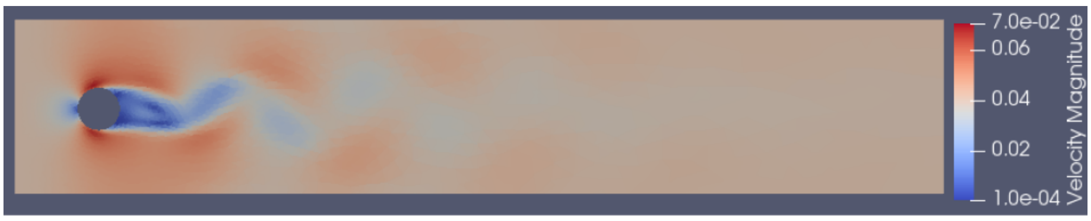


Figure 34: The velocity magnitude of the prediction by DD-GPR

solution is to store the LSTM for each subdomain temporarily at the beginning of each time step. This is used every iteration in order to find a converged through all the sub-domains solution. Once a solution has been achieved then the LSTM is allowed to have its internal memory updated. By doing this, the convergence of the predicting result for each local iteration becomes better. In the last step iteration, use the prediction made by the original LSTM as the input for the temporarily LSTM. By doing this, the inner state of LSTM is only updated by one time for one real time step, see result in figure 35.

The prediction result of the global LSTM is relatively good.

Using more POD numbers will not lead to better prediction result for LSTM. Here's the result with 10, 20, 50, 100 POD, see figures 36, 37, 38, 39, 40.

4.2.4. 3D London South Bank University test case

This section demonstrates the ability of DDNIROM using a test case in the London South Bank University (LSBU) area, shown in Figure 23 . The computational domain has a size of [0, 2041] $*$[0, 2288] $*$[0, 250] (meters). The incoming-eddy method is using the setting from [2].

In this case, the streamwise direction corresponds to a westerly wind direction. The high-fidelity solutions were obtained by running in parallel the open-source finite element fluid dynamics code called Fluidity, see [2].

To initialise the problem, the Large Eddy Simulation (LES) simulation was performed for one hour (in real time). LES is a type of turbulent flow model that filters out the small scale feature but resolves the large scale features. After the initialisation stage, the simulation is continued for 1680 s with a time-step size of 4/3 s on the fixed unstructured mesh with 670,975 nodes. During the 1680 s, 420 snapshots were taken every 4 s from the high-fidelity LES simulation results. The DDNIROM was trained with these 420 snapshots.

The result of LSTM's prediction and GPR's replication is shown in the figure 41, 42, 43. The results are shown on a plane perpendicular to the Z-axis, with Z = 10. Using the snapshots from the 100th time step - 400 seconds, see figures 41, 42 , 43.

The complexity of the dynamics of this 3D problem involving turbulent and chaotic flows means that the use of 10 POD coefficients is no enough to capture all the turbulent fluctuations.

The result with more POD number seems better but is still losing much information. So without using LSTM or GPR to train the model, I directly generate the snapshots recovering from compression. See figure 44, 45 POD lost lots of features, which will lead to less accurate results. In theory, POD can't reserve non-linear features, such as a vortex which is non-linear.

5. Discussion & Conclusions

5.1. Strengths and limitations

For reduced order modelling, the two key factors are accuracy and speed. To gain higher accuracy generally need longer time, and the objective is to use a shorter time to increase as much accuracy as possible.

The accuracy will be affected by two processes, compression and prediction/replication. Here is we fix the number of POD basis functions then we effectively fix the accuracy of the compression part of the error and we can thus focus on the prediction error. For prediction, using LSTM as a training method has fastest speed.Training is normally achieved in less than 10 seconds which is much quicker than for GPR, say. If one looks at the prediction using LSTM what one

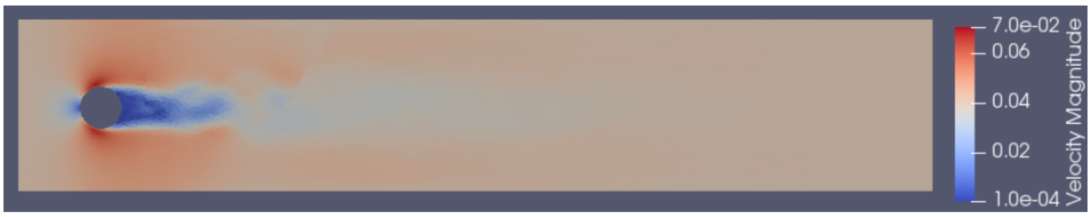


Figure 35: Storing the state of LSTMs to implement DD-LSTM

Method	Offline Calculation Time
IC-Ferst calculate	30 min
LSTM	10 s
GPR	25 min
DD-LSTM	35 s
DD-GPR	1 h 25 min

Table 2: Running time using different training method with 10 POD coefficients

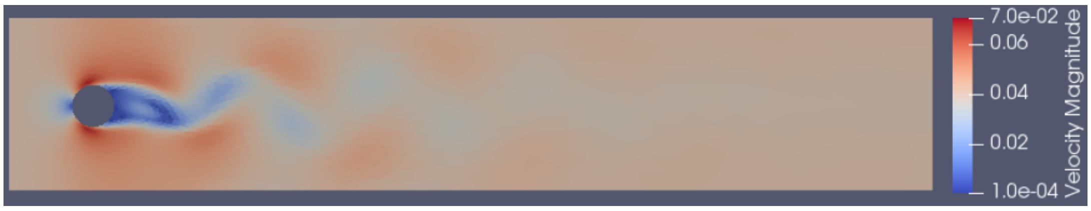


Figure 36: The velocity magnitude of HFM (Use this as the standard)

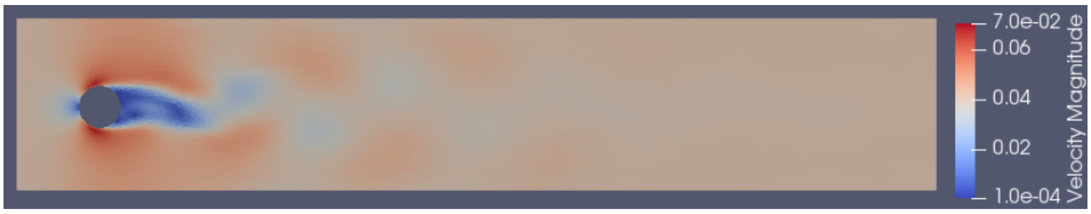


Figure 37: LSTM prediction with 10 POD basis functions and global LSTM.

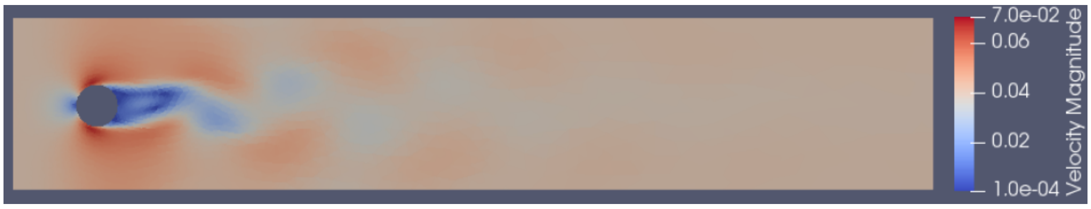


Figure 38: LSTM prediction with 20 POD basis functions and global LSTM.

observes is that after several time steps the prediction becomes very similar to that of the high fidelity model. The accuracy for GPR is generally better than LSTM in the short term,

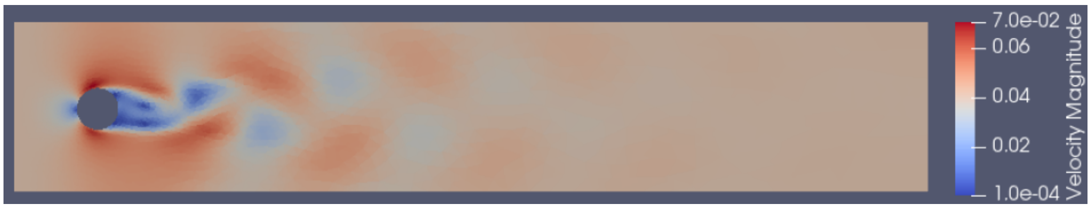


Figure 39: LSTM prediction with 50 POD basis functions and global LSTM.

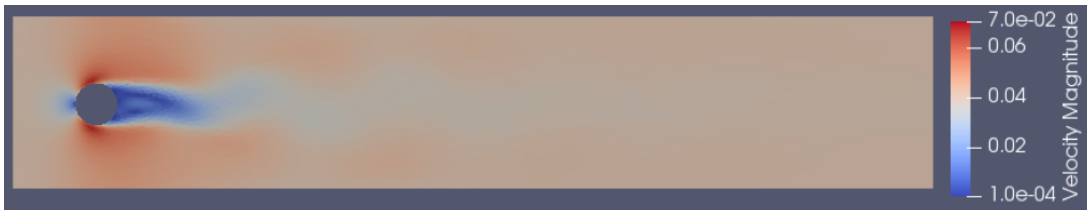


Figure 40: LSTM prediction with 100 POD basis functions and global LSTM.

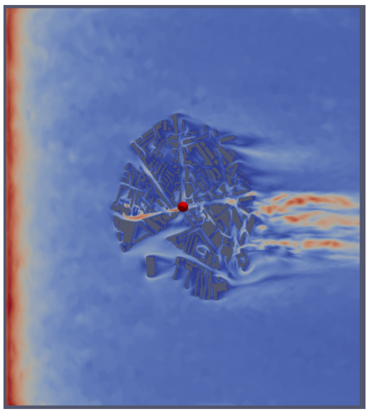


Figure 41: HFM of 3D test case

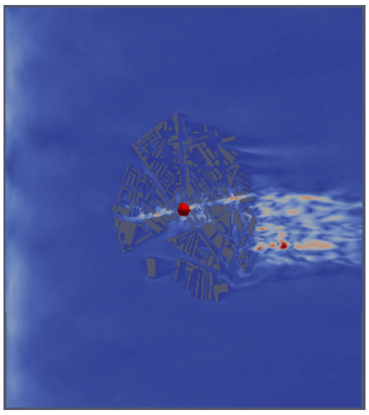


Figure 42: Prediction by LSTM for 3D test case

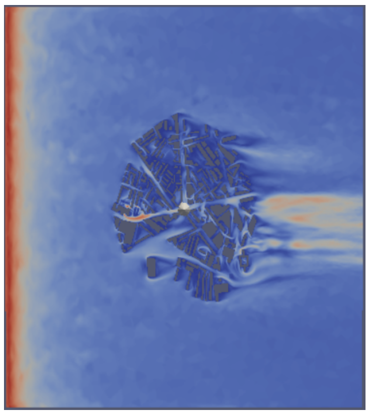


Figure 43: Replication by GPR for 3D test case

but longer simulations seem to favour LSTM and in that situation both methods are competitive. A drawback of GPR is its lengthy training times.

There are issues with the internal memory of the LSTM when using DD which makes it difficult to obtain good results. If this could be solved, it could give better accuracy than global LSTM. However, DD-GPR gives the best accuracy for short term prediction at a cost of considerable offline training times.

Using increased numbers of POD coefficients increases the accuracy of the compression of the velocity variables. However, increasing the number of POD coefficients odes not necessarily increase the accuracy of the simulations as the GPR or LSTM has then to form a hyper-surface with increased number of dimensions which is difficult to form an accurate representation for. For example, in the 2D flow past a cylinder test case, using 10 POD coefficients provided the best result, using less or more POD coefficients leads to reduced accuracy in global GPR or LSTM.

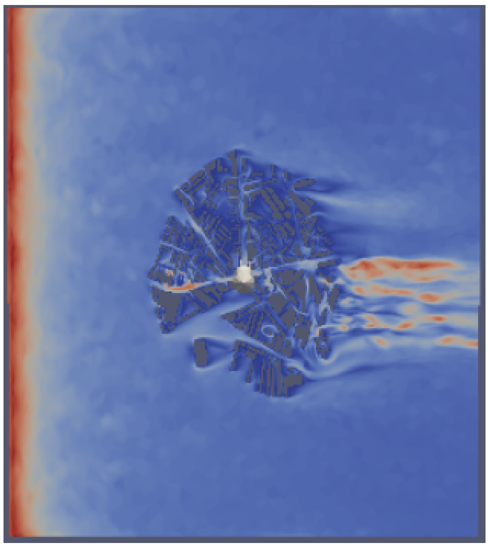


Figure 44: 3D test case original data at a cut plane

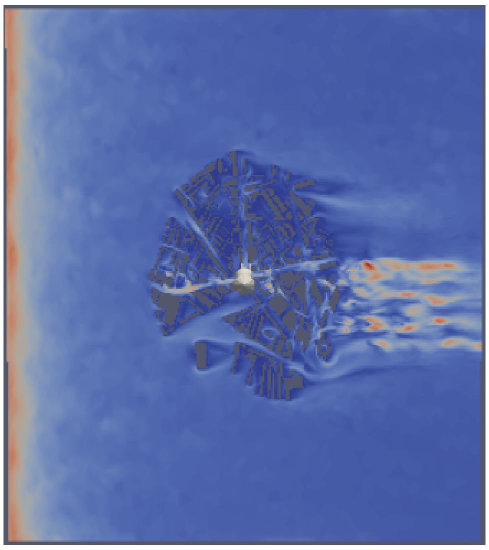


Figure 45: 3D test case recovering after being compressed by POD at a cut plane

Moreover the DD-LSTM or DD-GPR has an advantage here as the problem is broken down into a series of small (in dimension) hypersurfaces with potentially greater accuracy in the resulting models.

When using LSTM, the number of time levels, of the compressed solution variables, that are used as inputs into the LSTM in order to predict the future time level is a variable that should be explored. In the 2D flow past cylinder problem using 3 previous time levels resulted in the best performance. There is no direct way of picking the number of time levels other than trial and error. The GPR uses only 1 previous time level as its input to predict the compressed solution at the next time level.

There are three major limitation for the GPR and LSTM based NIROMs developed here: 1) First, for compression, POD is a linear compression method, so it may not be able to capture many of the non-linear flow features of the Navier-Stokes equation. Using a compressing method to retain non-linear features will be helpful e.g. an auto-encoder. 2) There are internal memory issues associated with the LSTM which make it problematic for use with DD methods. 3) The domain decomposition method with GPR is computational expensive and don't have speed advantage over global method or initial calculator.

5.2. *Conclusions:*

Using ROM is for fluid flow problems will effectively speed up the prediction process. LSTM's offline training has the advantage in speed. In low dimension problems, using more POD coefficients will not guarantee better prediction performance; using appropriate POD coefficient number will make the prediction more accuracy. Both LSTM and GPR can learn the feature of fluid flow in reduced space.

5.3. *Future work:*

1) For compression, it is suggested that the POD method may be replaced by more powerful non-linear compression methods such as auto-encoders which may be able to capture

more details of highly non-linear fluid flow problems.
2) For DD-LSTM, is may be an good idea to perform the domain decomposition iterations while avoiding updating the internal memory of LSTM. Once this iteration has converged then the memory can be updated and the time step advanced.

Acknowledgements

Appreciate the following people for their help and contribution to this project: Prof. Christopher Pain and Dr. Claire Heaney for guidance and instruction; Mr JiaYe Mao (Gary) and Mr Toby Philips for exchanging ideas.

Reference

[1] 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.

[2] 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.

[3] D. Xiao, F. Fang, C. Heaney, I. Navon, C. Pain, A domain decomposition method for the non-intrusive reduced order modelling of fluid flow, Computer Methods in Applied Mechanics and Engineering.

[4] T. S. community, numpy.linalg.eigh, `https://docs.scipy.org/doc/numpy-1.14.0/reference/generated/numpy.linalg.eigh.html` (2014).

[5] Q. Wang, J. S. Hesthaven, D. Ray, Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem, Journal of Computational Physics 384 (2019) 289–307.

[6] 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.

[7] F. Chinesta, P. Ladeveze, R. Ibanez, J. V. Aguado, E. Abisset-Chavanne, E. Cueto, Data-driven computational plasticity, Procedia engineering 207 (2017) 209–214.

[8] R. Eggersmann, T. Kirchdoerfer, S. Reese, L. Stainier, M. Ortiz, Model-free data-driven inelasticity, Computer Methods in Applied Mechanics and Engineering.

[9] L. T. K. Nguyen, M.-A. Keip, A data-driven approach to nonlinear elasticity, Computers & Structures 194 (2018) 97–115.

[10] T. Kirchdoerfer, M. Ortiz, Data-driven computational mechanics, Computer Methods in Applied Mechanics and Engineering 304 (2016) 81–101.

[11] G. Capuano, J. J. Rimoli, Smart finite elements: A novel machine learning application, Computer Methods in Applied Mechanics and Engineering 345 (2019) 363–381.

[12] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[13] Z. C. Lipton, J. Berkowitz, C. Elkan, A critical review of recurrent neural networks for sequence learning, arXiv preprint arXiv:1506.00019.

[14] Z. Shen, Y. Zhang, J. Lu, J. Xu, G. Xiao, A novel time series forecasting model with deep learning, Neurocomputing`doi:https://doi.org/10.1016/j.neucom.2018.12.084`.
URL `http://www.sciencedirect.com/science/article/pii/S0925231219304461`

[15] W. Long, Z. Lu, L. Cui, Deep learning-based feature engineering for stock price movement prediction, Knowledge-Based Systems 164 (2019) 163–173.

[16] H. Y. Kim, C. H. Won, Forecasting the volatility of stock price index: A hybrid model integrating lstm with multiple garch-type models, Expert Systems with Applications 103 (2018) 25–37.

[17] H. D. Park, Y. Han, J. H. Choi, Frequency-aware attention based lstm networks for cardiovascular disease, in: 2018 International Conference on Information and Communication Technology Convergence (ICTC), IEEE, 2018, pp. 1503–1505.

[18] Y. Yang, J. Dong, X. Sun, E. Lima, Q. Mu, X. Wang, A cfcc-lstm model for sea surface temperature prediction, IEEE Geoscience and Remote Sensing Letters 15 (2) (2017) 207–211.

[19] J. Zhao, X. Mao, L. Chen, Speech emotion recognition using deep 1d & 2d cnn lstm net-

works, Biomedical Signal Processing and Control 47 (2019) 312–323.

[20] J. Jo, S. Hwang, S. Lee, Y. Lee, Multi-mode lstm network for energy-efficient speech recognition, in: 2018 International SoC Design Conference (ISOCC), IEEE, 2018, pp. 133–134.

[21] J. Li, A. Mohamed, G. Zweig, Y. Gong, Lstm time and frequency recurrence for automatic speech recognition, in: 2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), IEEE, 2015, pp. 187–191.

[22] S. Zhang, S. Liu, M. Liu, Natural language inference using lstm model with sentence fusion, in: 2017 36th Chinese Control Conference (CCC), IEEE, 2017, pp. 11081–11085.

[23] S. A. Fahad, A. E. Yahya, Inflectional review of deep learning on natural language processing, in: 2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE), IEEE, 2018, pp. 1–4.

[24] X. Yu, L. Xu, L. Ma, Z. Chen, Y. Yan, Solar radio spectrum classification with lstm, in: 2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), IEEE, 2017, pp. 519–524.

[25] L. Wang, X. Xu, H. Dong, R. Gui, R. Yang, F. Pu, Exploring convolutional lstm for polsar image classification, in: IGARSS 2018-2018 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2018, pp. 8452–8455.

[26] A. M. Ertugrul, P. Karagoz, Movie genre classification from plot summaries using bidirectional lstm, in: 2018 IEEE 12th International Conference on Semantic Computing (ICSC), IEEE, 2018, pp. 248–251.

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

[28] Y. Zhang, H. Zhang, Z. Tian, The application of gaussian process regression in state of health prediction of lithium ion batteries, in: 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), IEEE, 2018, pp. 515–519.

[29] Y. Huang, S. Liu, L. Yang, Wind speed forecasting method using eemd and the combination forecasting method based on gpr and lstm, Sustainability 10 (10) (2018) 3693.

[30] P. R. Vlachas, W. Byeon, Z. Y. Wan, T. P. Sapsis, P. Koumoutsakos, Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 474 (2213) (2018) 20170844.

[31] R. Mittra, A novel domain decomposition technique for solving very large problems in frequency and time domains, in: The Second European Conference on Antennas and Propagation, EuCAP 2007, IET, 2007, pp. 1–4.

[32] L. Yaoyao, S. Donglin, L. Weimin, Higher order modes analysis of a hemp simulator using time-domain simulation and singular value decomposition, in: 2018 International Applied Computational Electromagnetics Society Symposium-China (ACES), IEEE, 2018, pp. 1–2.

[33] A. Heinlein, A. Klawonn, M. H. Lanser, J. Weber, Machine learning in adaptive domain decomposition methods - predicting the geometric location of constraints, Technical report, Universität zu Köln (October 2018).
URL `https://kups.ub.uni-koeln.de/8645/`

[34] C. Pain, C. E. De Oliveira, A. Goddard, A neural network graph partitioning procedure for grid-based domain decomposition, International journal for numerical methods in engineering 44 (5) (1999) 593–613.

[35] T. S. community, scipy.linalg.toeplitz, `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.linalg.toeplitz.html` (2014).

[36] T. S. community, scipy.sparse.csr_matrix, `https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.sparse.csr_matrix.html` (2014).

[37] T. Contributors, Pytorch, `https://pytorch.org/docs/stable/optim.html?highlight=adam#torch.optim.Adam` (2019).

.