A Convolutional Autoencoder with LSTM to Predict Unsteady Flows Around a Two-Dimensional Cylinder

Spencer Schwartz [1]

Abstract

A convolutional autoencoder (CAE) with long-short term memory (LSTM) is presented that predicts the velocity and pressure fields around a 2D cylinder for various Reynold numbers. The model is trained on 2500 snapshots of flow field data obtained from running simulations in the open source computational fluid dynamics (CFD) software *Basilisk*. It is shown that the CAE can successfully reconstruct images by using three convolutional layers, each with kernels that have learnable weights. Likewise, the LSTM is shown to accurately predict the next snapshot in a given sequence. However, the accuracy of this prediction notably decreases as the predicted snapshots are recursively fed back into the model.

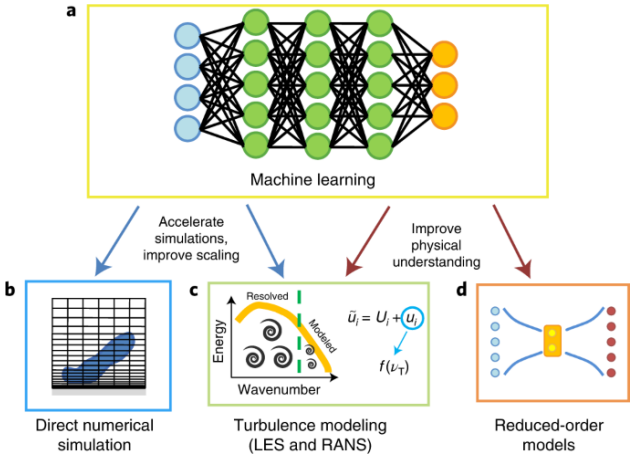

Figure 1. The different ways machine learning has been applied to CFD. Taken from (Vinuesa & Brunton, 2022).

1. Introduction

In recent years, engineers and scientists have found computational fluid dynamics (CFD) simulations an essential tool in advancing our understanding of complex fluid and gas phenomena. Though generally cheaper than experiments, the computational cost of running these simulations can be great if the investigated problem is multi-scale which requires a very fine grid resolution. Machine learning (ML) and neural networks (NN) have been key tools in reducing these computation costs through several techniques, as shown in Figure 1. For one, ML has been used to accelerate simulation time and improve scaling to capture sub-grid features (Vinuesa & Brunton, 2022). For example, (Sousa et al., 2024) and (Nastorg et al., 2022) used ML to more efficiently solve the pressure-Poisson equation, an everlasting problem of great interest for incompressible flows.

Turbulence modeling is another prominent challenge faced in the simulation community. Due to the multi-scale nature of turbulence, researchers must use either direct numerical simulation (DNS) with a very fine mesh size or a model like LES or RANS to approximately capture these effects. The DNS approach is preferred for its accuracy but scales poorly and is rarely used due to the associated cost. Papers like in (Kochkov et al., 2021; Pathak et al., 2020; Tian et al., 2022) have successfully applied ML to decrease simulation time, sometimes by 30-40x, while also representing unresolved turbulence that would otherwise be eliminated by the coarse mesh.

Lastly, reduced-order modeling (ROM) is a type of ML that learns based on reducing input dimensions to improve our physical understanding of underlying, non-linear characteristics while significantly reducing the computational cost. Thanks to this, ROMs are especially useful for predicting turbulence, real-time control systems, and identifying the effects of parameters on the flow field (Wang et al., 2023; Gupta & Jaiman, 2021; Lui & Wolf, 2019). A common type of ROM is an Autoencoder, which uses encoding layers to reduce the dimensionality of the input and decoding layers to reconstruct it back to its original form. When used with convolutional neural networks (CNNs), called convolutional autoencoders (CAE), they are even more flexible

[1]College of Engineering and Computing, University of South Carolina, United States of America. Correspondence to: Spencer Schwartz <SCS27@email.sc.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

and great for capturing non-linear relationships while being scalable (Bank et al., 2020). For these reasons, CAEs are widely used for image compression, signal processing, object recognition, text analysis, and much more.

In CFD, CAEs are commonly paired with a recurrent neural network (RNN) to process and predict sequential data. More long-term time-dependent problems require long-short term memory (LSTM), a more advanced type of RNN that remedies its vanishing/exploding gradient problem which causes the model to struggle to learn long-term dependencies. An LSTM cell has three different gates: an input gate, a forget gate, and an output gate. This makes it able to learn complex temporal patterns. For more details on how LSTMs function, the reader is referred to the seminal paper (Hochreiter & Schmidhuber, 1997). This combination of CAE w/LSTM has been applied to various unsteady problems, such as predicting flow around a cylinder (Hasegawa et al., 2020) and airfoils (Zhang, 2023).

In this paper, we attempt to replicate the results of (Hasegawa et al., 2020) and will outline our slightly modified CAE w/LSTM that predicts unsteady flow around a 2D cylinder. The paper is outlined as follows. Section 2 provides an overview of how the simulation data used for training was generated. Section 3 outlines the model architecture with its two main components: convolutional autoencoder (Section 3.1) and long-short term memory (Section 3.2). Section 4 showcases results and how each part of the model performs on reconstructing and predicting trained data and untrained data for validation. Furthermore, a brief discussion of how to improve the results and an analysis of the latent spaces is provided in Section 5. Finally, Section 6 summarizes the model's architecture, results, and future direction.

2. Simulation Setup

The case of a 2D, stationary cylinder is chosen to test the model due to its simplicity and low computational cost. The DNS is run using the open source CFD software *Basilisk*, which uses adaptive mesh refinement (AMR) to further improve speed. Furthermore, *Basilisk* solves the Navier-Stokes equations, defined as

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f} \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

using a second-order accurate scheme. The variables in Eq. (1) and (2) $\mathbf{u}, p, \nu$, and $\mathbf{f}$ represent the velocity, pressure, kinematic viscosity, and solid volume force, respectively.

Figure 2 displays the domain's geometry and boundary conditions for each edge. The Reynold's number (Re) for an infinitely long cylinder is defined as $Re = \rho U_\infty D/\mu$, where

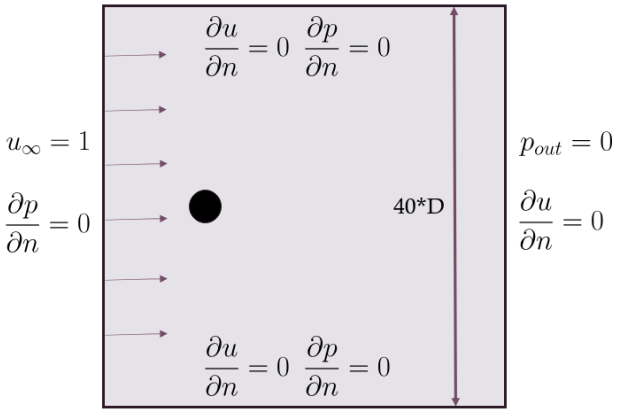

Figure 2. The square domain setup with boundary conditions (BCs) for the simulation. An inlet and outlet BC are used on the left and right walls, respectively. The top and bottom walls use a free slip or symmetric boundary condition. D here represents the diameter of the cylinder.

ρ, U_∞, D, and μ denotes the fluid's density, free-stream velocity, cylinder's diameter, and the fluid's dynamic viscosity, respectively. 10 Different Res were simulated: 20, 40, 60, 80, 100, 120, 140, 160, 180, and 200. The 10 were split in two, 5 to be used for training and the other 5 for validation.

Each simulation used a coarse cell size of $D/\Delta x = 12.8$ to speed up simulation runtime. Although coarse, we find that the overall flow field does not change drastically when using a finer mesh.

Once a steady state has been reached, around $t = 60$ for most cases, snapshots of the velocity, $\mathbf{u} = (u, v)$, and pressure, p, fields in a 10x10 region around the cylinder are saved with a spacing of $\Delta t = 0.077$ until there are 500 snapshots for each Re. The snapshots are then grouped to form a sequence with 21 snapshots, which correlates to 1 vortex shedding period at $Re = 100$. The sequences are used in the time-series forecasting of Section 3.2.

3. Neural Network Model

The model was made and trained using PyTorch. The following sub-sections will explain the details of the architecture before moving into the results section.

3.1. Convolutional Autoencoder

The purpose of the CAE is to learn how to successfully encode a high-dimensional input into a meaningful low-dimensional representation and then successfully decode that representation back into an output similar to the input.

The fully connected layers found in basic autoencoders are

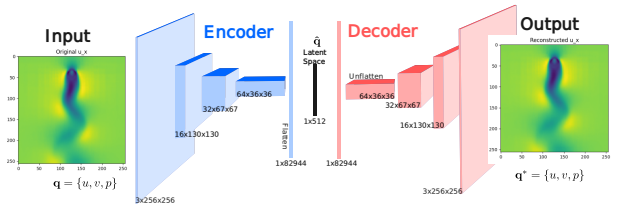

Figure 3. CAE model architecture. The 3D boxes represent the convolutional layers while the 2D rectangles represent vectors. A fully connected layer is used after flattening to shape the latent space. Likewise, a fully connected layer is used after unflattening in the decoder.

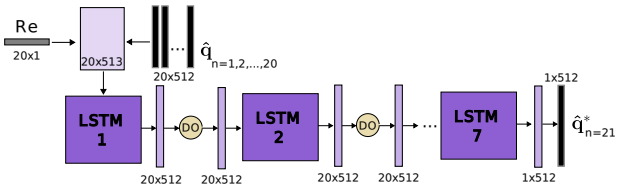

Figure 4. The LSTM's model architecture with 7 layers. Its' input is a concatenation of a Re vector, that only holds the normalized Reynolds number, with the sequence of latent vectors $\hat{\mathbf{q}}$. Its output is the latent vector of the next timestep $\hat{\mathbf{q}}^*$ DO represents the dropout layers that have a 0.2 probability of activating.

replaced with 6 convolutional layers, 3 each for the encoder and decoder. The Input into the encoder at a given time is a single snapshot of the flow field $\mathbf{q} = \{u, v, p\}$, where u and v represent the horizontal and vertical velocity components. $\mathbf{q}$ is a tensor of shape 3x256x256, with the first number representing the 3 variables and the following numbers representing the number of cells in each direction. When passed through the convolutional layers, the first number represents that number of channels.

Inside the three convolutional layers, we use a 5x5 kernel/filter with a stride of 2 which essentially halves the spacial dimensions of each layer. Due to the large number of channels used, when the encoder output is flattened, it is a large vector of size 82,944. This is too large for the LSTM to handle, so a linear layer is added to the encoder to reduce the size to 512, which represents the latent space, $\hat{\mathbf{q}}$. In our experience, the CAE can effectivity learn to reconstruct images for a wide range of tested latent spaces, although it generally does better when it is larger. However, due to efficiency concerns, the size was set to 512.

Figure 3 shows the shape of the tensors as they go through the model. Additionally, batch normalization is applied between each layer along with the Rectified Linear Unit activation function, which were all found to be crucial in minimizing loss and speeding up training.

The loss function used to train the CAE is the mean squared error (MSE) of the input and output, expressed as

$$\epsilon_{cae} = \frac{1}{N_u}\frac{1}{N_v}\frac{1}{N_p}\sum_{i=1}^{N_u}\sum_{j=1}^{N_v}\sum_{k=1}^{N_p}(q_{ijk} - q^*_{ijk})^2, \qquad (3)$$

where N represents the number of data points for each specified variable, 256 in this study. In addition, a two-fold cross-validation was used to prevent overfitting, each with 1250 epochs. In total, the CAE model was trained on 260 snapshots for each Re.

3.2. Long-Short Term Memory

All snapshots are encoded, with their sequential order preserved. Then 21 latent spaces $\hat{\mathbf{q}}$ from the encoder are arranged in a sequence. In total, there are 2,400 unique sequences, 480 each Re. The first 20 snapshots, now latent spaces, are sent to the LSTM while the 21st is set aside for loss calculation.

The LSTM has 7 layers or cells, to match that of (Hasegawa et al., 2020), each with a hidden layer of 512. The size of the hidden layer is crucial for allowing the LSTM to learn time-dependent features. From testing, we found anything below 512 gave us considerably worse results. A dropout layer with a value of 0.2 was added in between each layer to prevent overfitting.

A crucial aspect of the model is to learn how parameters affect results. In this case, the parameter is the Reynolds number, which is concatenated onto the adjusted latent space sequence before being sent to the LSTM, i.e. the input into the LSTM is of shape 20x513 where 20 represents the adjusted sequence length. In doing so, the model can learn the flow fields dependence on Re. This process is summarized in Figure 4.

When validating the LSTM, we recursively feedback the predictive latent space $\hat{\mathbf{q}}^*$ back into the sequence of initially given snapshots until a specified number of timesteps/iterations. For example, given 8 initial snapshots (n=1,2,...,8) the LSTM predicts n=9*, so the next sequence is (n=2,3,...,9*), then (n=3,4,...,9*, 10*), and so on. Figure 5 illustrates this, which is used in Section 4.2. Note to convert from the snapshot number, n, to the actual time, t, we use t=n$\Delta t + t_0$, where t_0 is the starting time.

In training, the LSTM predicts only one timestep. The loss is calculated using MSE, similar to Eq. (3) except with the values of the latent space and the predicted latent space vectors.

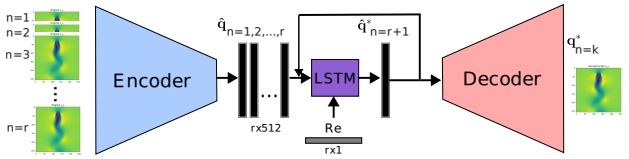

Figure 5. The process to recursively predict multiple timesteps, where r is the current last step in a sequence and k is the final step. The number of initial fields is varied and discussed in Section 4.2.

4. Results

The results are split in two to separate results from only the CAE and CAE w/LSTM. Furthermore, each section has a subsection for training results and validation results. The Res used for training are 40, 80, 120, 160, and 200, the rest (20, 60, 100, 140, 180) are saved for validation. Lastly, it should be noted that in agreement with the use of batch normalization, all plotted results (except for the error fields, which is the absolute differences between the target and output) have been normalized with their respective maximums. We will first look at the CAE results, as they are more straightforward than the ladder.

4.1. CAE Results

4.1.1. Training

Four-fold cross-validation training is used on 1400 flow field snapshots to help prevent overfitting. After 1250 epochs, the loss using Eq. (3) converges to 1x10^{-}4. Training is run using Google Colab's T-4 with High RAM. Using a batch size of 512, it uses about 11 GB of GPU memory and takes about 1.5 hours.

The training results, i.e. testing the model on data it has already seen, are shown in Figure 6 and 7. Overall, the model does a very good job of reconstructing these images. One thing to note is the discretized grid is visible in the input images, left column. This finite grid gets smoothed out when passed through the encoder thanks to the medium size 5x5 kernel size, which can be seen as an advantage since this better reflects a continuum seen in real life.

Crucially, the highest error generally occurs inside or directly adjacent from the cylinder. From basic fluid mechanics, we know that the sharpest velocity gradients occur near the cylinder's solid boundary. Consequently, the CAE struggles to accurately reconstruct this large, sharp variation across a small number of cells using a 5x5 kernel size. In future work, adjusting the size of the convolutional kernel will be useful to counteract this problem.

Additionally, forcing the data points to be null inside the air-

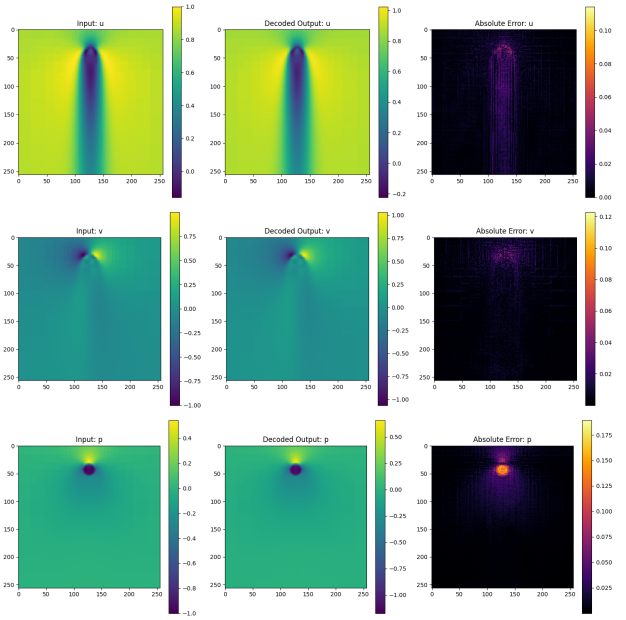

Figure 6. CAE training results for the steady regime Re = 40. MSE = 1.16x10^{-}4.

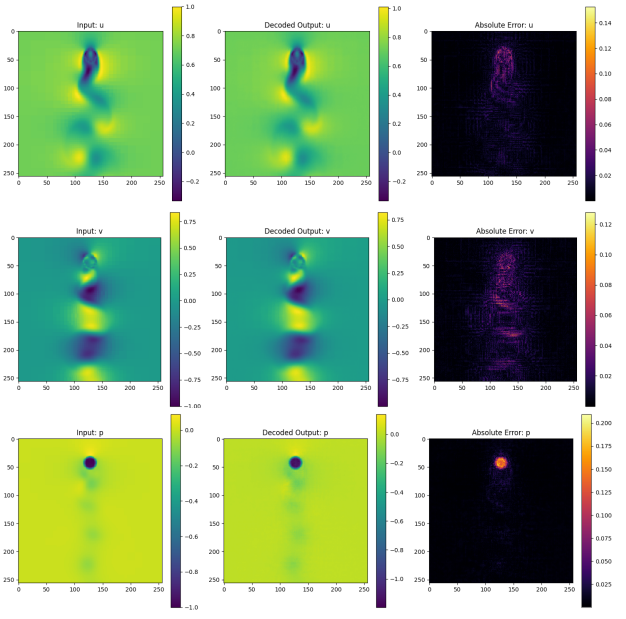

Figure 7. CAE training results for the unsteady regime Re = 200. MSE = 1.14x10^{-}4.

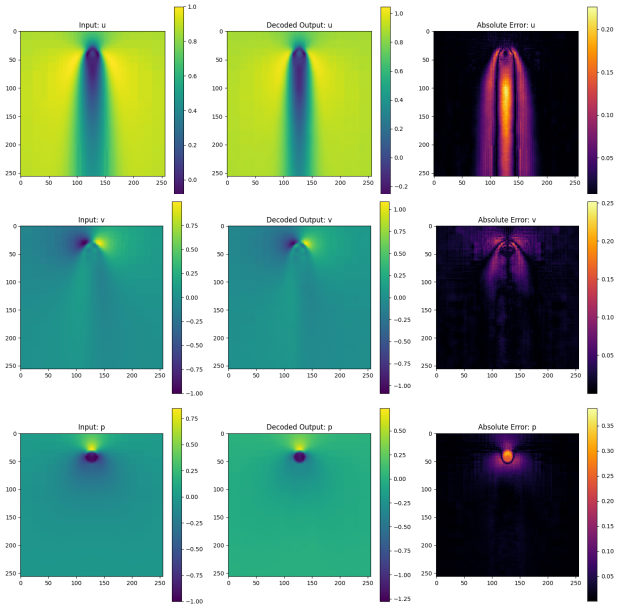

Figure 8. CAE validation results for the steady regime Re = 20. MSE = 1.32x10^{-3}.

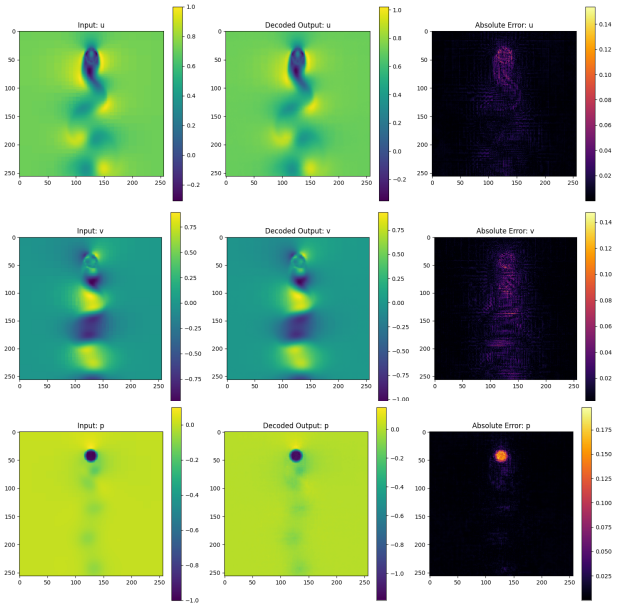

Figure 9. CAE validation results for the unsteady regime Re = 180. MSE = 1.15x10^{-4}.

foil, as done in (Zhang, 2023), would increase its accuracy and speed. The solid boundary is simulated using an immersed boundary method (IBM) similar to that of (Uhlmann, 2005). In this IBM, the flow field inside the solid object is non-zero, although it is shown in previous studies that this doesn't have a significant impact on the surrounding fluid domain.

4.1.2. Validation

Now we select data from the validation Re list to see how well the data can extrapolate and interpolate. To save space, only two Reynold's numbers will be looked at, one from each regime: steady ($Re \leq 40$) and unsteady ($Re > 40$).

From the steady regime, we choose $Re = 20$, the results of which are shown in Figure 8. Although the CAE reconstructs a sensible flow field, the MSE value and error field show that the model fails to capture the wake and field near the solid boundaries. Interestingly, the flow field predicted is almost identical to $Re = 40$. We infer this is a sign of overfitting from the model, which is supported by later results, but more investigation into the problem is required.

In the unsteady regime, a phenomenon known as vortex shedding occurs, previously shown in Figure 7. For $Re = 180$, we expect similar results to $Re = 200$, which may be why we get such a good reconstruction as shown in Figure 9 despite suffering with $Re = 20$. Nevertheless, it demonstrated that the model can accurately reconstruct snapshots in the unsteady regime, which is the focus of this paper anyway.

4.2. CAE w/LSTM Results

4.2.1. Training

The LSTM's dataset had 2400 sequences which was again split into a training and validation set. The training set had 80% of the original sequences, i.e. 1920, while the validation set had the remaining 20%, 480. The training set of training sequences was shuffled to help prevent overfitting. Unlike the CAE training, the LSTM can be trained on a relatively weak computer after computing and saving all of the latent spaces for the 2400 sequences. Figure 10 shows the loss for both sets over the full 5000 epochs. The graph is truncated to start from the 20$^{\text{th}}$ epoch for scaling purposes.

Like in the previous section, we will first look at how the full ROM (CAE w/LSTM) predicts snapshots in the steady regime. Again, we choose $Re = 40$ to represent the steady case. We use the recursive predictive process outline in Section 3.2 and Figure 5. For this case, we have an initial/input sequence length of 5 snapshots and stop the recursive predictions for evaluation after 100 steps, which correspond to $r_0 = 5$ and $k = 100$ in Figure 5. The results are displayed in Figure 11. A very good agreement is found with no deviations over the 100 steps.

Figure 12 showcases the results for the unsteady regime,

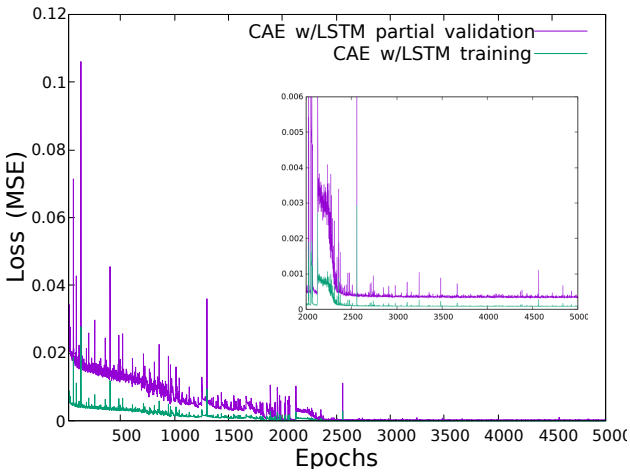

Figure 10. MSE of the training and partial validation part of the CAE w/LSTM. An inset graph is provided to help show the minor variations that take place after 2500 Epochs, although it has essentially reached a plateau. The final MSE value for the training and validation was about 1x10^{-4} and 3x10^{-4}, respectively.

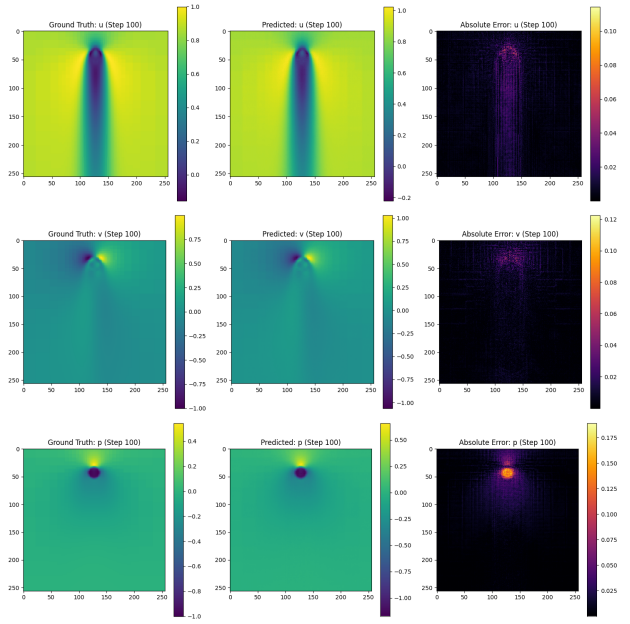

Figure 11. Flow field predictions of u after 100 iterative guesses for Re = 40 with 5 initial snapshots. MSE = 5.48x10^{-5}. Note MSE doubles if the initial snapshots increase to 20 in this case.

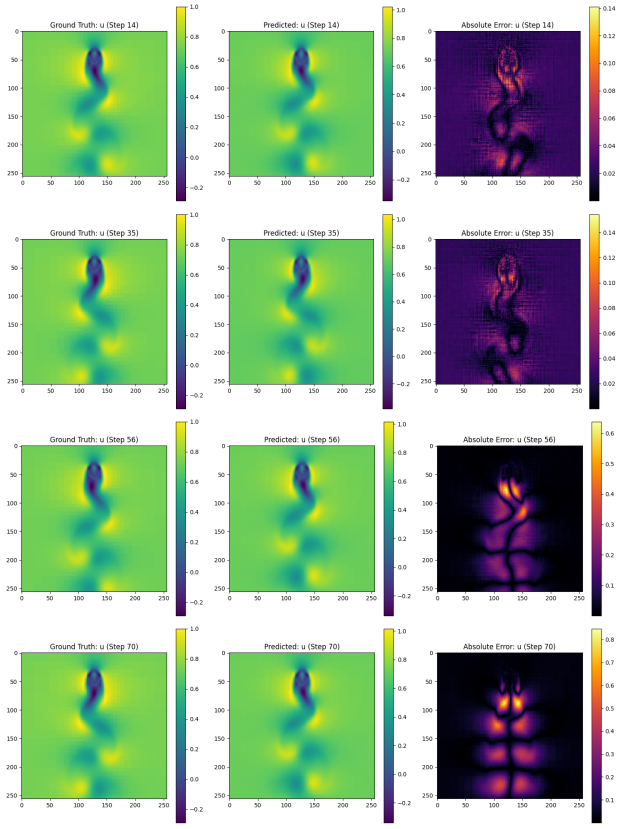

Figure 12. Flow field predictions of the u field results for Re = 160. The LSTM fails after around 50 predictions, as seen by the lack of change in the bottom 2 middle figures.

chosen to be $Re = 160$. Although there is little error during initial predictions, when trying to predict for 100 recursive iterations, the LSTM fails to predict after about 50 and outputs the same field every time. Although not shown here, we ran the same test for $Re = 120$ and got similar results, except the LSTM failed earlier after 21 iterations. This issue is discussed more in the next section.

4.2.2. Validation

$Re = 20$ and 180 were chosen for validation like for CAE. Figure 13 and 14 showcases these results. For the steady regime, we see similar results from Figure 8. The problem with the LSTM failing is not a problem for steady-state simulations, since the flow field doesn't change anyway. For this reason, we consistently see low error, albeit still incorrect due to the CAE failure to reconstruct $Re = 20$. On the other hand, for $Re = 160$, the model again fails after about 21 snapshots but achieves good agreement with the target/ground truth before that. This indicates that the LSTM can recognize and accurately predict flow fields for untrained Re, but only for a short amount of time. It should

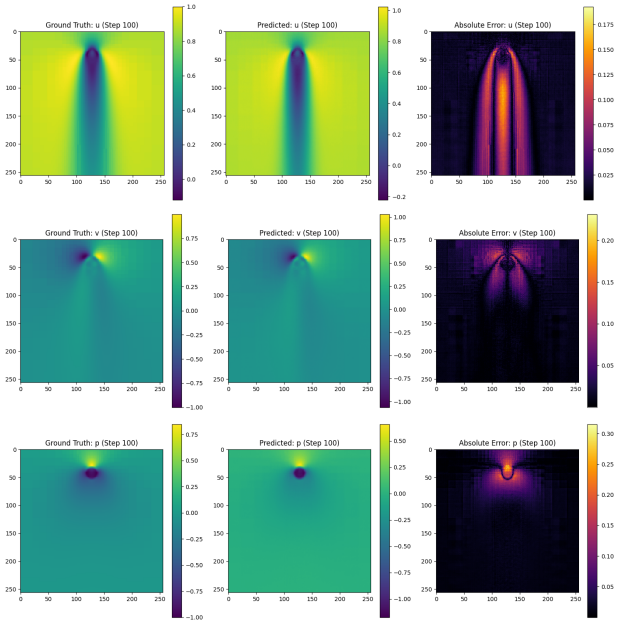

Figure 13. Flow field predictions after 100 iterative guesses for Re = 20 with 21 initial snapshots. MSE = 1x18^{-3} for all iterations.

be noted that the number of initial snapshots in both cases was 21. After testing with an increased or decreased input, the results remain essentially unchanged and the LSTM model still fails after a few predictions.

5. Potential Improvement

As discussed in Section 4.2, the LSTM fails to iteratively predict after 20-50 snapshots, depending on Re and the length of the initial sequence. This is an issue that is likely caused by the LSTM's architecture, like the size of the hidden layers, dropout layers, number of layers, etc. It can also be a result of training it for too long, illustrated in Figure 10. Additionally, the LSTM can be changed to incorporate recursive prediction inside the training loop to potentially mitigate these failures, although it was not tested here because of (Hasegawa et al., 2020) success without it.

Another key component is the latent space and how it is formed. After training a modified model, this time with a latent space size of 128, we get a worse prediction than the showcased model of 512. This may be an indicator that the latent space is losing key features from the layers in the encoder or that it must be sufficiently large to adequately capture all characteristics of a snapshot. A principal component analysis (PCA) is performed on 15 randomly selected latent space sequences, shown in Figure 15. The details of the PCA method are not discussed here, we expect to see connected contours, which indicates the latent space is

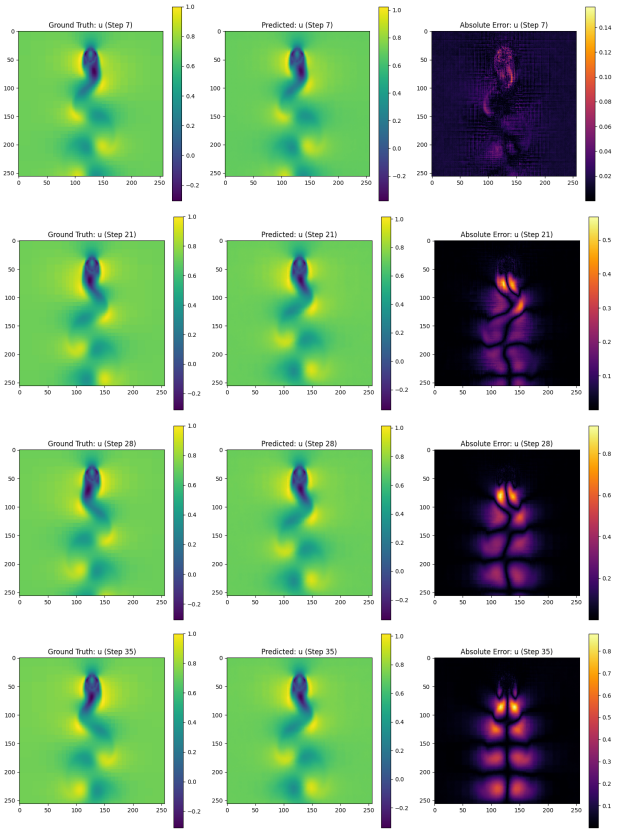

Figure 14. Flow field predictions of the u field results for Re = 180 with 21 initial snapshots. The LSTM fails after around 21 predictions, as seen by the lack of change in the bottom 2 middle figures.

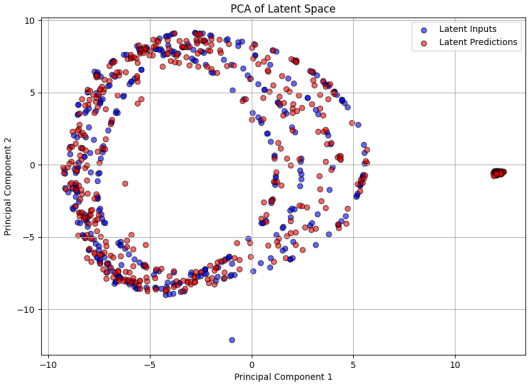

Figure 15. Principal component analysis of 15 latent spaces.

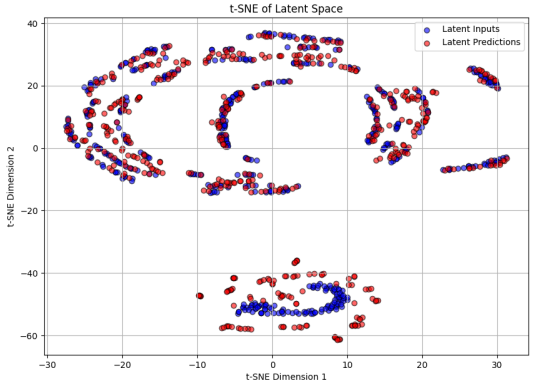

Figure 16. t-distributed stochastic neighbor embedding analysis of 15 latent spaces.

sufficiently capturing meaningful features. However, we see that the predicted data points do not match very well with the target curves. In addition, we see a congregation to a single point on the right-hand side, another indicator of a potential problem with the model.

Lastly, we perform a similar analysis called t-distributed stochastic neighbor embedding (t-SNE) on the latent spaces, see Figure 16. Again, we expect good overlap between the target and predicted latent spaces but this is not achieved.

From this analysis, it is clear that how the latent space is encoded and/or how the LSTM model is set up is causing the prediction failures. Some changes in the encoder and decoder to be tested include varying the kernel sizes between convolutional layers, changing stride length to not notably reduce dimensions between layers as done in (Zhang, 2023), increasing the latent space dimension, varying the number of channels, and incorporate a variational autoencoder into the current model to fit the data in a more normalized distribution (Bank et al., 2020). Likewise, the effect of the LSTM model's parameters like the number of layers hidden layer size, and dropout probability should be investigated and optimized.

6. Conclusion

The presented Convolutional Autoencoder (CAE) with Long-Short Term Memory (LSTM) model successfully demonstrated its ability to reconstruct and predict unsteady flow fields around a two-dimensional cylinder across varying Reynolds numbers. The CAE efficiently encoded high-dimensional flow field data into a meaningful latent space, enabling accurate reconstructions, while the LSTM captured time-dependent dynamics, albeit with limitations in recursive predictions. Despite its strengths, the model exhibited challenges in predicting long-term sequences, highlighting the need for future improvements in architecture, such as refining latent space dimensionality and optimizing LSTM parameters. These findings underline the potential of combining similar CAEs and LSTMs for fluid dynamics applications and set a foundation for further exploration into reducing computational costs and enhancing predictive accuracy for complex flows.

7. Resources

The files used for this project can be found in this GitHub repository: https://github.com/csce585-mlsystems/cae_cylinder.

References

Bank, D., Koenigstein, N., and Giryes, R. Autoencoders. 3 2020.

Gupta, R. and Jaiman, R. Three-dimensional deep learning-based reduced order model for unsteady flow dynamics with variable reynolds number. 12 2021. doi: 10.1063/5.0082741.

Hasegawa, K., Fukami, K., Murata, T., and Fukagata, K. Cnn-lstm based reduced order modeling of two-dimensional unsteady flows around a circular cylinder at different reynolds numbers. In *Fluid Dynamics Research*, volume 52. IOP Publishing Ltd, 12 2020. doi: 10.1088/1873-7005/abb91d.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997. ISSN 08997667. doi: 10.1162/neco.1997.9.8.1735.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning accelerated computational fluid dynamics. 1 2021. doi: 10.1073/pnas.2101784118.

Lui, H. F. S. and Wolf, W. R. Construction of reduced order models for fluid flows using deep feedforward neural networks. 3 2019. doi: 10.1017/jfm.2019.358.

Nastorg, M., Schoenauer, M., Charpiat, G., Faney, T., Gratien, J.-M., and Bucci, M.-A. Ds-gps : A deep statistical graph poisson solver (for faster cfd simulations). 11 2022.

Pathak, J., Mustafa, M., Kashinath, K., Motheau, E., Kurth, T., and Day, M. Using machine learning to augment coarse-grid computational fluid dynamics simulations. 9 2020.

Sousa, P., Afonso, A., and Rodrigues, C. V. Application of machine learning to model the pressure poisson equation for fluid flow on generic geometries. *Neural Computing and Applications*, 36:16581–16606, 9 2024. ISSN 14333058. doi: 10.1007/s00521-024-09935-0.

Tian, Y., Woodward, M., Stepanov, M., Fryer, C., Hyett, C., Livescu, D., and Chertkov, M. Lagrangian large eddy simulations via physics informed machine learning. 7 2022.

Uhlmann, M. An immersed boundary method with direct forcing for the simulation of particulate flows. *Journal of Computational Physics*, 209:448–476, 11 2005. ISSN 00219991. doi: 10.1016/j.jcp.2005.03.017.

Vinuesa, R. and Brunton, S. L. Enhancing computational fluid dynamics with machine learning. *Nature Computational Science*, 2:358–366, 10 2022. doi: 10.1038/s43588-022-00264-7.

Wang, Z., Liu, X., Yu, J., Wu, H., and Lyu, H. A general deep transfer learning framework for predicting the flow field of airfoils with small data. *Computers and Fluids*, 251, 1 2023. doi: 10.1016/j.compfluid.2022.105738.

Zhang, B. Airfoil-based convolutional autoencoder and long short-term memory neural network for predicting coherent structures evolution around an airfoil. *Computers and Fluids*, 258, 5 2023. ISSN 00457930. doi: 10.1016/j.compfluid.2023.105883.