Evaluating the Effectiveness of the Convolutional LSTM Neural Network for Simulations in Computational Fluid Dynamics

Augustin Castillo

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2024

Committee:

Erika Parsons

Program Authorized to Offer Degree:

Computer Science and Software Engineering

University of Washington

Abstract

Evaluating the Effectiveness of the Convolutional LSTM Neural Network for Simulations in Computational Fluid Dynamics

Augustin Castillo

Chair of the Supervisory Committee:
Erika Parsons
School of Science, Technology, Engineering & Mathematics

Computational Fluid Dynamics (CFD) is an important part of engineering design, with applications in diverse areas. Although its practical application is widespread, the computational cost hinders its utilization. This research evaluates the effectiveness of the Convolutional LSTM (ConvLSTM) neural network for Computational Fluid Dynamics simulations when creating Reduce Order Models (ROM) and simulating turbulent fluid flows interacting with an obstacle. We propose a novel end-to-end Artificial Neural Network (ANN) model architecture based entirely on ConvLSTM that can successfully predict the spatiotemporal evolution of a fluid flow. This data-driven approach achieves similar results to a classical CFD method with direct numerical simulation with a Mean Squared Error of 1.107×10^{-05} in a quarter of its execution time. This model could be used to accelerate CFD simulations, leading to a faster engineering development process. By providing rapid preliminary results for prototype testing, engineers can explore more design ideas without waiting days or weeks for simulation results.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my gratitude to Prof. Erika Parsons for all the valuable guidance and help during this work and to Mathew Fuentes for helping me define the research topic. I would like to sincerely thank Prof. Afra Mashhadi and Prof. Munehiro Fukuda for accepting my request to be on the committee for this thesis. I would like to extend my thanks to Prof. Kool for helping me proofread my thesis paper. My gratitude also goes to my family and friends for their encouragement and support through my studies and difficult times.

Chapter 1

INTRODUCTION

1.1 Background

Fluid mechanics plays a vital role across various major industries due to the diverse range of technologies and devices that require interaction with fluids for their operation. These industries encompass multiple applications, ranging from aircraft, maritime vessels, and architectural structures to essential medical equipment like ventilators and dialysis machines. For example, in the aviation sector, airplanes navigate through the fluid medium of air externally while internally relying on various fluids such as aviation fuel and hydraulic fluids for proper functioning. Similarly, in sectors like structural engineering, structures like buildings and bridges must be designed to withstand various wind conditions, including hurricanes and tornados. There are several healthcare industrial fluid mechanics applications in the medical realm, from designing medical equipment to disease prevention. Examples of these applications involve modeling blood flow through the human body to identify conditions that may cause health issues and controlling air quality to prevent airborne dispersion and transmission of viruses such as COVID-19. All these examples use fluid mechanics to design a solution.

Modern fluid mechanics solutions use Computational Fluid Dynamics (CFD) simulations to assess engineering challenges. For example, CFD simulations are used in developing innovative airfoil designs for airplanes to enhance efficiency, minimize fuel consumption, and mitigate pollution emissions. To tackle this, it is crucial to solve fluid mechanics models involving large complex systems of equations over large volumes of data. However, there are some challenges in using those simulation techniques. Calculating such fluid dynamics interactions over an aerodynamic system can quickly become a computationally heavy task,

especially with the growing complexity of new designs. Significant computation times resulting from complex calculations can drastically increase the development time. As a result, it causes project delays across the various stages of development because it is impossible to perform a fast iterative design process. For example, while creating an airfoil shape, an aeronautical engineer will have to perform several CFD simulations to test the design's performance, implement the necessary modifications to improve the design and test it again, and wait several days for the simulation to run, to continue the work. Having reliable and fast tools to analyze fluid interactions to support an iterative design process strategy is essential to the engineering process. Finding new ways to improve the speed of these simulations can significantly impact the engineering process development time, making it the primary motivation of this work.

Performing CFD simulations can pose a challenge due to the need for substantial computational resources, including processing power and memory. Simulations that are large in scale or have higher grid resolutions may take a long time to complete, leading to a delay in obtaining results. This limitation can hinder the usage of CFD simulations for specific applications or restrict the exploration of design spaces. As a result, it is crucial to develop solutions that can enhance the simulations' execution time. High Performance Computing (HPC) and parallelization techniques are currently used to improve the simulations' execution times and analysis. However, implementing and optimizing these acceleration techniques can be even more challenging than conducting the CFD simulation or the design task for which the simulation is required, consuming research time and computing resources. Although modern high-performance computers provide increased computing power, allowing designers to use CFD simulations instead of real-life experiments, this process is costly or impractical to conduct in many cases. This is especially problematic at the initial design stages of airplanes, boats, bridges, buildings, cars, and other machines and structures interacting with fluids like air or water.

In recent years, Artificial Intelligence (AI) and Machine Learning (ML) have experienced great success thanks to the improvement of computational capability available to researchers,

particularly in the area of Deep Learning (DL) with the application of Artificial Neural Networks (ANN) [21]. It has been used in diverse applications in many areas of science to solve complex problems, for instance, techniques including time series prediction, generative models, dimensionality reduction and principal component analysis, and super-resolution of images and videos. ML and DL techniques have the potential to enchase CFD simulations [31]. Recently, there have been emerging research trends of ML applications for CFDs [30] in direct numerical simulation, turbulent modeling, and reduced order models.

This project explores a potential improvement of Computational Fluid Dynamics simulations using Artificial Intelligence to increase the simulation speed while maintaining a low error compared to classical CFD methods.

1.2 Research Objective and Solution

This research aims to employ Deep Learning (DL) methods to create Computational Fluid Dynamics (CFD) simulations involving the interaction of fluid flow with an obstacle in a 2-dimensional environment. The objective is to decrease the simulation's execution time compared to conventional simulation techniques. To achieve this goal, a novel neural network architecture incorporating approaches from existing research in DL for CFD and new ideas for this field are proposed. The DL solution proposed is an end-to-end data-driven solution. This means it is a unified process that can use data to learn the complexities of a fluid flow spatial structure's evolution over time. It can also quickly adapt to changing environments represented by different datasets by directly learning from raw data that represents the intrinsic patterns of fluid mechanics. This solution combines an autoencoder and a generator implemented with a ConvLSTM neural network.

Compared to related solutions in DL for CFD that use different datasets (like Homogeneous Isotropic Turbulence data) or focus on the Dimensionality Reduction techniques of the input data, this research's emphasis is on generating a fluid flow simulation interacting with an obstacle and how well it generalize between distinct shapes of obstacles in various positions and sizes. Additionally, this research studies the effectiveness of the Convolutional

LSTM neural network by implementing the model using only this type of architecture.

Two primary metrics are used to evaluate the success of this solution: execution time and accuracy. The execution time of the simulation using the DL model is compared to a traditional CFD simulation. Its accuracy, when compared to the conventional method, is measured using the Mean Squared Error (MSE) (See Equation 1.1), also known as the Mean Squared Deviation (MSD). The goal is to reduce the execution time while maintaining a good enough accuracy to preserve the pattern structure of the fluid in the generated sequence flow.

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1.1}$$

1.3 Scope

The proposed solution focuses on the simple case of a fluid flow interacting with a stationary shape in a 2-dimensional environment and replicates the fluid's behavior using a DL model as accurately as possible while improving the execution time compared to a traditional simulation.

Because the Navier-Stokes equations used in fluid dynamics are so complex and chaotic (See Section 2), finding an analytical solution for some problems is impossible. This is why numerical techniques are used to approximate the solutions. Research and industry rely on approximated results to perform their experiments and designs. This means that even when the model solution results are not so precise but provide a fast approximation of the data, it still has value since it is a tool to quickly iterate initial designs that can later be validated with more accurate but slow and resource-demanding methods.

1.4 Paper overview

This paper is organized as follows: Chapter 2 explains the main concepts for this work related to Computational Fluid Dynamics and Deep Learning. Chapter 3 presents related work and the current state of DL research for CFD and discusses previous related research relevant

to this study. Chapter 4 explains all the methods involved in developing this research and the solution, including the data collection, the DL model architecture and training, and the evaluation techniques. Chapter 5 shows the results with its analysis and discussion. Finally, Chapter 6 presents the conclusions of this research, its limitations, and future work based on the results obtained.

Chapter 2

THEORETICAL BACKGROUND

This chapter includes the theoretical background for this work discussing related concepts in Computational Fluid Dynamics and Deep Learning.

2.1 Fluid dynamics and Navier-Stokes equations

Fluid dynamics is the branch of physics that studies the motion of fluids, both liquids and gases and their interactions with solid boundaries and other fluids. It encompasses a wide range of phenomena, from the flow of water in rivers and the atmosphere's motion to blood circulation in organisms and the behavior of fluids in engineering systems.

The Navier-Stokes equations (See Equations 2.1) are fundamental partial differential equations governing the motion of viscous fluids. They describe how fluid velocity, pressure, density, and viscosity evolve over time in response to external forces. The equations are derived from Newton's second law of motion, conservation of mass, and conservation of momentum principles.

The equations consist of two main components: 1) the continuity equation, which represents the conservation of mass, and 2) the conservation of momentum equation, derived from Newton's second law, which describes how the velocity of the fluid changes in response to external forces and internal forces (pressure and viscosity). This is defined in Equation 2.1 with t time, ρ density, u velocity, p pressure, μ viscosity, and F external forces.

$$\begin{aligned} \nabla \cdot u &= 0 \\ \rho(\frac{\partial u}{\partial t} + (u \cdot \nabla)u) &= -\nabla p + \mu \nabla^2 u + \rho F \end{aligned} \tag{2.1}$$

The Navier-Stokes equations are nonlinear, leading to complex and sometimes chaotic

behavior, such as turbulence. Despite their simplicity, solving these equations analytically for many practical problems is often impossible, leading to the widespread use of numerical methods.

The Navier-Stokes equations have vast applications in various fields, including engineering, meteorology, oceanography. They support the design of aircraft, ships, and vehicles, studying weather patterns, and understanding fluid flow in pipes and channels. However, many fundamental aspects of these equations, including turbulence, remain unsolved problems in mathematics and physics.

2.2 Turbulent flow and Reynolds number

Turbulent flow is characterized by chaotic and unpredictable fluid motion, with irregular fluctuations in velocity, pressure, and flow patterns. It occurs when inertial forces dominate over viscous forces, leading to mixing and eddy formation. The Reynolds number (Re) is a dimensionless parameter that characterizes the flow regime of a fluid. When Re is high, it is a turbulent flow, indicated by the visual appearance of swirling patterns (or eddies), while low Re values indicate a laminar flow type, recognized by a "smooth" flow of the fluid. Thus, turbulent flow is directly correlated to Reynolds numbers, with higher Re values corresponding to more turbulent behavior and lower Re values indicating laminar flow. Equation 2.2 defines Re, with ρ fluid density, L length scale, U velocity, and μ viscosity.

$$Re = \frac{\rho L U}{\mu} \tag{2.2}$$

2.3 Direct Numerical Simulations, RANS and LES

In computational fluid dynamics (CFD), Direct Numerical Simulation (DNS), Reynolds-Averaged Navier-Stokes (RANS), and Large Eddy Simulation (LES) are three common approaches for simulating fluid flows, each with its own set of advantages and limitations.

DNS directly solves the Navier-Stokes equations without any modeling assumptions, pro-

viding detailed information on all scales of motion in the flow. However, DNS requires high computational resources and is typically only feasible for relatively low Reynolds number flows due to its high computational cost scaling with the cube of the Reynolds number.

RANS averages the Navier-Stokes equations over time to obtain mean flow quantities and then models the effects of turbulent fluctuations using empirical turbulence models. RANS is computationally less expensive than DNS and is suitable for a wide range of engineering applications. However, RANS relies on turbulence models that introduce modeling errors and uncertainties, particularly for complex flows.

LES resolves large-scale turbulent structures explicitly while modeling the effects of smaller-scale turbulence. It strikes a balance between the accuracy of DNS and the computational cost of RANS, making it suitable for simulating moderately high Reynolds number flows. LES captures the essential features of turbulence while reducing modeling errors compared to RANS.

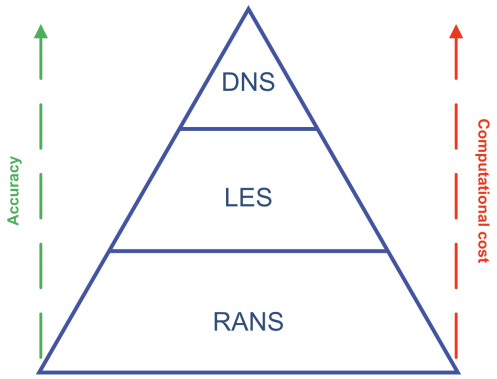


Figure 2.1: Comparison between DNS, LES, and RANS modeling

The choice between DNS, RANS, and LES depends on the flow's specific characteristics, the desired detail level, and the available computational resources. Figure 2.1 shows a comparison between the different modeling approaches. DNS provides the most accurate results but is computationally expensive, while RANS and LES offer compromises between accuracy

and computational cost, making them more practical for many engineering applications.

2.4 Time series

Time Series is a type of data that presents a temporal ordering. It is used in many real-world applications, such as signal processing, finance, weather forecasting, control engineering, communication, human activity recognition, cyber-security, or earthquake prediction. Time series can be described as univariate or multidimensional. Univariate or 1-dimensional time series is an ordered set of real values X of length equal to the number of real values T, where $X = [x_1, x_2, ..., x_T]$. While a multidimensional or M-dimensional time series consists of M different univariate time series, where $X = [X^1, X^2, ..., X^M]$ with $X^i \in \mathbb{R}^T$. A Time Series Dataset is defined as $D = \{(X_1, Y_1), (X_2, Y_2), ..., (X_N, Y_N)\}$ as a collection of pairs (X_i, Y_i) where X_i could be a 1-dimensional or M-dimensional time series and an output Y_i.

2.5 LSTM

Recurrent Neural Networks (RNN) are a type of neural network that can keep information about what happened before, they do this with loop connections to their neurons. These neural networks are widely used in speech recognition, language modeling, translation, etc. The main problem with this simple architecture is with "long-term dependencies" when the relevant information happened too long ago. Long Short Term Memory (LSTM) networks are designed to learn long-term dependencies to overcome this issue. The main idea behind the LSTM structure is a memory cell that can accumulate information that can be written and cleared by structures called gates. Fully Connected LSTM (FC-LSTM) is a multivariate version of LSTM, meaning that the input, output, and state are all 1-dimensional vectors.

LSTM neural networks are widely used in Natural Language Processing (NLP), where a long sequence of words in a text needs to be analyzed and used for prediction and classification.

2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of neural network primarily designed to process and analyze data in a grid-like organization, especially visual data such as images. Inspired by the organization of the animal visual cortex in the brain, CNNs have been successful in computer vision applications. The key operation in CNNs is the convolution mathematical operation, a specialized kind of linear operation. These networks have a structure called filters, which are applied to the input data to extract features, allowing the neural network to learn hierarchical representations.

CNNs have become the cornerstone of various computer vision tasks, including image classification, object detection, facial recognition, and medical image analysis. Their ability to automatically learn relevant features from raw data makes them particularly effective in tasks where traditional algorithms struggle, such as image understanding and pattern recognition. Additionally, CNNs offer advantages such as parameter sharing, which reduces the number of parameters and enhances model efficiency and translational invariance, enabling robust performance even with variations in object position within an image. Overall, CNNs have revolutionized the field of computer vision and continue to drive advancements in artificial intelligence and image processing applications.

2.7 ConvLSTM

LSTM neural networks are good for long-time series prediction because they are designed to maintain a context memory of important information that happened long ago as well as recent information. In applications with many dimensions, like spatial data, using an LSTM is inefficient as it contains too much redundancy in the connections between the input.

ConvLSTM extends the Long Short-Term Memory (LSTM) architecture, incorporating convolutional operations within the LSTM units. It is specifically designed to handle spatiotemporal data, such as video sequences or spatial-temporal patterns in data. ConvLSTM preserves the sequential memory capabilities of LSTM while exploiting the spatial informa-

tion in data through convolutions. This allows it to capture both temporal dependencies and spatial correlations simultaneously, making it ideal for tasks like video prediction, precipitation nowcasting, and motion tracking. Compared to traditional LSTMs, ConvLSTMs excel in modeling spatial dependencies within sequences, enabling more accurate predictions and better handling of spatially structured data.

2.8 Autoencoders

Autoencoders are an artificial neural network used for unsupervised learning tasks, particularly in dimensionality reduction and data compression. Comprising an encoder and a decoder, they aim to reconstruct input data while learning efficient representations. The encoder compresses the input into a lower-dimensional latent space, while the decoder reconstructs the original data from this representation. Advantages include their ability to learn meaningful representations from unlabeled data, aiding in feature extraction and data denoising. They are widely used in anomaly detection, image denoising, and data generation tasks. Additionally, autoencoders can serve as the foundation for generative models, such as variational autoencoders (VAEs) and generative adversarial networks (GANs), enabling the synthesis of new data samples. Their versatility, simplicity, and capability to learn compact representations make them valuable tools in various domains, including computer vision and natural language processing.

Chapter 3
RELATED WORK

This chapter summarizes current research on Deep Learning (DL) that can be applied in Computational Fluid Dynamics (CFD) to support this study. It describes the current state of DL for CFD and discusses previous work done in this area that differs from this research. Additionally, it presents research that is outside of other areas that can be applied to CFDs.

3.1 Potential areas of impact of DL in CFD

The Computational Fluid Dynamics field deals with numerical simulations of fluid flows. Navier-Stokes equations are Partial Differential Equations (PDE) describing fluid flows. Solving those PDEs for chaotic (turbulent) flows requires computationally intensive numerical methods because of the necessary space and time scales. These turbulent flows can be simulated with different accuracy and resource costs. Data-driven modeling methods are revolutionizing scientific discoveries enabled by advances in scientific computing [11]. Furthermore, Machine Learning and Deep Learning models have the potential to enhance those CFD simulations [31] [12], and recently, there has been an emerging trend of research applications of ML and DL for CFD [30]. ML applications in CFD can be categorized into three main areas: Direct Numerical Simulation, Turbulent Modeling, and Reduce Order Models.

Direct Numerical Simulation

Direct Numerical Simulations (DNS) provide high accuracy for solving Navier-Stokes equations for fluid flows. Typically, there is a trade-off between the solution's accuracy and feasibility because solving this equation at scale is limited by the computational cost. Some proposed solutions focus on accelerating the numerical simulation by using Deep Learning

to approximate parts of the computations involved. For example, [20] developed a method to accurately correct errors caused in low-resolution simulations by learning parameters affected by the coarse grid. They got stable results comparable to high-resolution numerical simulations and reduced the computational cost with a low-resolution discretization grid. Other research [9] [34] focuses on using DL to solve the Poisson equation for corrections in incompressible fluids. This is done by exploiting data from previous examples to map deviations between uncorrected velocity and the resulting pressure field.

Those methods only apply ML techniques in part of the solution but still rely on classical numerical methods to calculate the fluid flow. The solution proposed in this research is an end-to-end deep learning model that will apply DL to the entire simulation pipeline with a unified process. This simpler approach has the benefit of eliminating intermediate tasks in the simulation pipeline and thus reducing its complexity.

Turbulent Modeling

Resolving direct numerical simulations is computationally expensive, so industry CFD applications use Reynolds-Averaged Navier-Stokes (RANS) and Large-Eddy Simulations (LES) methods. Various ML methods used to improve RANS turbulence modeling are explored in [13] to improve its accuracy. Particularly the use of Physics-Informed Neural Networks (PINNs) [32] [15]. These PINNS models combine the data-driven modeling approach with using physics equations. This combined approach has the advantage of giving the model consistency with known physics laws; however, it makes the solution more complex to implement than a purely data-driven approach.

In comparison, the model developed in this research is a data-driven approach that can learn from the given data to directly produce predictive results. This approach has the benefit of adapting to the particularities of each use case's data characteristics.

Reduce Order Models

A useful application of ML in CFD is to develop Reduce Order Models (ROMs) to reduce the data complexity. This can be done because fluid flows contain principal structures that provide essential information about the flow. These ROMs represent the characteristics of the flow in a low-dimension representation that describes the evolution of those principal structures in the fluid, providing a substitute model for faster model predictions. A ROMs technique learns a low-dimensional coordinate system using Proper Orthogonal Decomposition (POD) [28] [24]. This technique is related to standard statistics and data-driven modeling techniques for dimensionality reduction: Principal Component Analysis (PCA) and Singular Value Decomposition (SVD) [11]. Deep Learning can be used to perform those methods in dimensionality reduction by learning the subspace representation from the data [22]. Autoencoders are neural networks where the input and output have the same dimensions, but in the middle, there is a bottleneck that reduces the dimensionality of the data. This type of model has two components: the first will take the input and compress it down to a smaller representation with fewer dimensions than the input, and the second one reconstruct the input from the representation. This low-dimensional representation subspace is called latent space and can simplify the original data by compressing it into a smaller representation. Autoencoders can be used to improve the performance of models that rely on classical ROM techniques. Because of the proven advantage of doing dimensionality reduction using DL Autoencoders, this model includes this type of component in its architecture.

3.2 Related solutions of DL for CFD

Recent works have validated the idea of using DL models for CFD modeling using different types of architectures and techniques. Here, we explore research examples that are the most relevant related work to this research, with similar goals but different solutions. It is important to note that although there are some similarities between them and this research, all of them have significant differences, mainly in the dataset used and the proposed architectures.

The use of an extended LSTM network with a Convolutional Neural Network (CNN) and Autoencoders was explored in [23] and [16]. Both used an Encoder-Decoder implemented with CNN to compress the data and an LSTM network enhanced with Convolutional filters that use the compressed data to generate the next part of the sequence. [23] develops a data-driven approach for Homogeneous Isotropic Turbulence data in a three-dimensional space. This solution is a sequence-to-sequence model, meaning it gets an initial sequence as input and produces the following sequence as output. The researchers proved that the model achieves good accuracy with long-term stability of the cyclic predictions while being very computationally efficient. [16] use a similar model with the same architecture for two-dimensional fluid flows around an obstacle. Their model shows similar results compared with flow fields calculated by a computational fluid dynamic solver.

A similar method was proposed by [17] to develop ROMs for two-dimensional unsteady flows around a circular cylinder. They also used a CNN for the Autoencoder, with the goal of learning the temporal evolution of the flow in the latent space, they used a much simpler LSTM network. Additionally, they examined the accuracy dependence of this model with different Reynolds numbers (between 20 and 160). Their model was able to reconstruct flows with Reynolds numbers that were not used during the training of the model.

A DL approach to solving Partial Differential Equation systems was proposed in [18]. They developed a sequence-to-sequence model where the Autoencoder was implemented with a Convolutional LSTM. This model was tested with the moving MNIST dataset and the 2-D viscous Burgers equation. The researchers conclude that this model is an excellent network to use when predicting the time series of a dynamical system. Another solution for solving time-dependent PDEs was done in [27] where they present a machine-learning approach based on a fully convolutional LSTM network to enhance finite-difference and finite-volume methods (FDM/FVM) common to solve PDEs. This method reduced the error by a factor of 2 or 3 compared to baseline algorithms.

A more advanced architecture of DL for CFD was recently proposed by [33]. The researchers explore the idea of using a combination of β-variational autoencoders (β-VAE)

[14] to learn a compact representation of the flow velocity and transformers to predict the temporal-dynamics. They use a dataset of a flow around a square cylinder obtained using an open-source numerical simulation with a Reynolds number of 500. They also perform a study to find the optimal values for the β-VAE, and the effect on model's performance of hyper-parameters such as architecture complexity, regularization β, and latent vector size. Their model achieves excellent performance, with a 97.8% of reconstruction accuracy and 96.5% in temporal-dynamics predictions, demonstrating that this combination has the potential for developing ROMs in complex flows.

3.3 Research from other domains

We also explored other research ideas in the deep learning domain that are not related to CFD applications but can be extrapolated to be applied in this area. These ideas come from video analytics and weather forecasting. We explored how deep learning research in those areas can also be applied to CFD simulations.

Video representation and prediction

The data for a fluid flow is a sequence of frames representing the state of the fluid at each time; in the same way, a video is a sequence of images in a timeline. Deep learning solutions that can learn features from videos for representation and future prediction have to deal with the same challenges as applications in CFD. In both cases, data has a spatio-temporal structure that requires the model to understand the space and time dimensions to predict future data. In [26], they use an LSTM neural network to learn representations of video sequences with an autoencoder and then extrapolate the learned video representations into the past and future. They also show that those representations help improve classification accuracy. The architecture is a composite model trained to perform two tasks simultaneously; using the encoded state, the model can predict the next frames as well as input reconstruction. They say that when done separately, the future predictor tends to only store information about the most recent frames, but when the model also has to reconstruct all of the input sequences,

it cannot pay attention to just the last frames. This model performance was evaluated using the moving MNIST dataset and 300 hours of YouTube data. It was able to correctly predict the future frames, even when the objects superimpose or bounce while moving.

ConvLSTM for precipitation nowcasting

Another area where analyzing spatiotemporal data to predict its future evolution is in weather. More specifically, the prediction of future rainfall intensity in a region over a short period of time, also known as precipitation nowcasting. In [25], they propose a new type of neural network called ConvLSTM by extending the fully connected LSTM by adding convolutional structures. This end-to-end solution is able to outperform LSTM and state-of-the-art methods for the precipitation nowcasting problem. The main reason for this new neural network is that a fully connected LSTM has too much redundancy for spatial data. They mention that this network is not specific for precipitation nowcasting but can also be used for other prediction problems involving spatiotemporal sequence data.

Chapter 4

METHODS

This chapter covers the procedures involved in this research, from data collection to the proposed DL model's architecture and its training, how it is evaluated, and all the tools used during this study. It provides an explanation of how the solution for this study works and its justification.

4.1 Data Collection

The dataset for this study consists of fluid flows interacting with an obstacle. Its purpose is to train and test the neural network solution proposed in this research. The fluid flows are generated using a numerical method, as is typically done in computational fluid dynamics simulations. The sequences represent the evolution of the fluid flow in space and time; therefore, this dataset could be considered two-dimensional Time Series data (See Section 2.4).

Each fluid flow is a sequence of 400 frames representing the state at a given time. The frames are a two-dimensional grid discretization of the simulated space, with a resolution of 200-width by 100-height cells. The values in the cells represent either the fluid's velocity at the position or -1 if it belongs to the obstacle.

All the fluid flows in the dataset have a Reynolds number of 220 and flow from left to right, interacting with an obstacle with several dimensions and positions. These obstacles are either circumferences or ellipses, as shown in Figure 4.1. Circumferences are simple shapes to model and are usually used to demonstrate fluid flow simulations. In contrast, ellipses are an easy simplification of an aircraft wing cross-section with different dimensions and inclinations (known as the "angle of attack").

In aeronautics, several pre-defined airfoils shapes (NACA airfoils) have been studied for

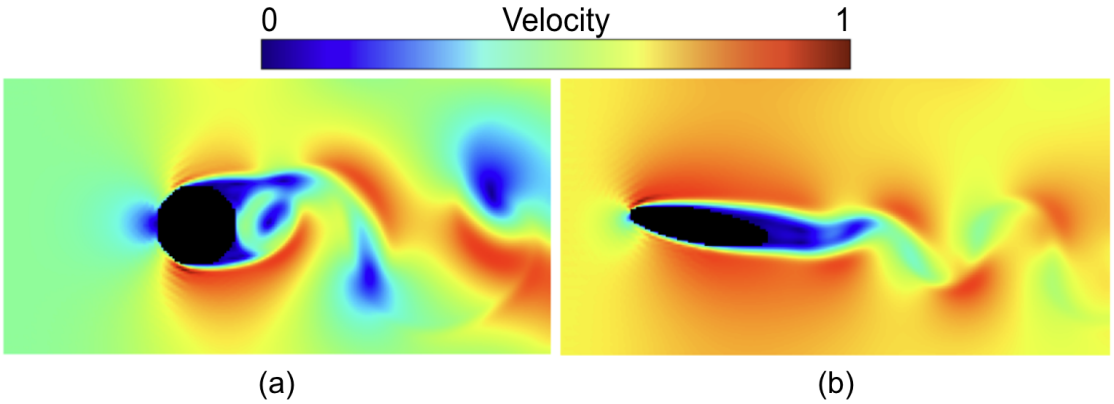


Figure 4.1: Two CFD frames from different sequences showing the (a)circumference and the (b)ellipse obstacles

the design of aircraft wings [8]; in particular, the NACA 2412 is used in the popular Cessna 172 Skyhawk. Figure 4.2 shows examples of those airfoils. However, these shapes are very complex to calculate. As an alternative for this work, an ellipse shape was chosen as an approximation to the NACA airfoils. The ellipse geometry has a smooth, curved shape that can resemble the streamlined profile of the airfoil. It can provide a good approximation for the leading and trailing edges, capturing the essential aerodynamic characteristics while simplifying the complex geometry of the airfoil. This approximation is particularly useful for preliminary design and analysis, where exact precision is less critical. It allows for easier mathematical manipulation and analysis compared to the exact airfoil shape.

4.1.1 Definitions

The fluid flow sequence data can be represented in the following mathematical way. Data is taken from velocity observations in a spatial region on a $W \times H$ grid, which consists of W columns and H rows. Each grid cell has a velocity measurement that varies over time T.

Definition 1. Let T be a finite period of time and $t_i \in T$, where $i \in \{0, 1, ..., T\}$, is a time instance.

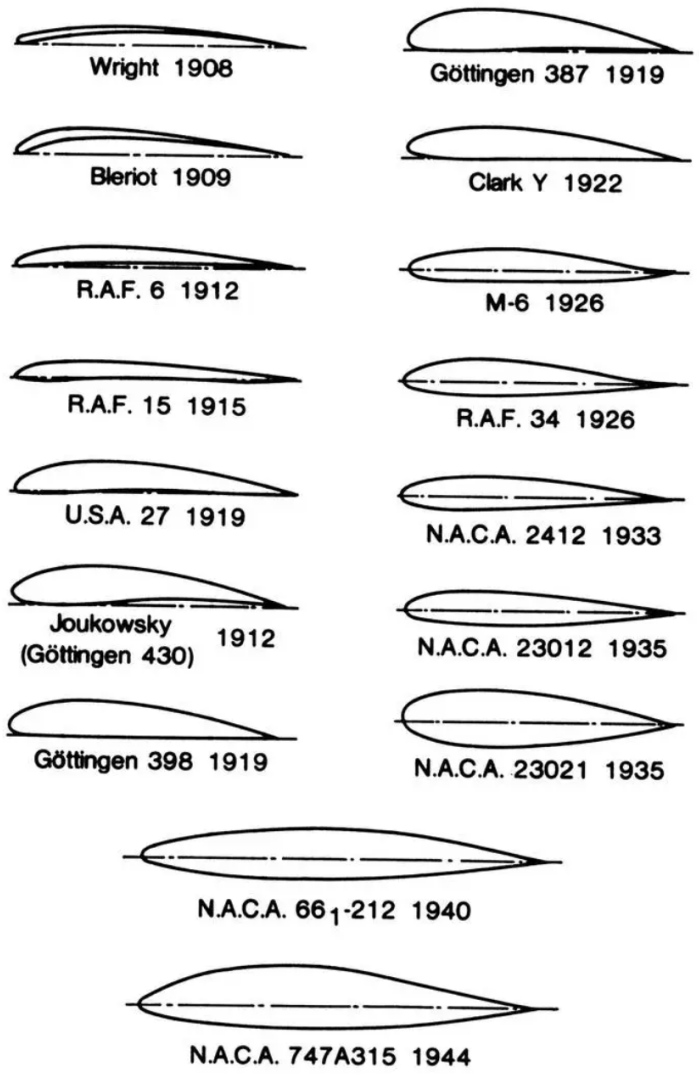


Figure 4.2: The historical evolution of airfoil sections from 1908-1944. Credit: NACA-NASA

We can then mathematically represent a fluid flow sequence as follows:

Definition 2. A fluid flow sequence is represented by a tensor $X \in \mathbb{R}^{T \times W \times H}$, where $x \in X$ and $x \in \mathbb{R}^{W \times H}$ is a velocity observation or fluid state at any given time $t \in T$.

Definition 3. Let $\mathcal{W}$ be a window of time $\subseteq T$, where $\mathcal{W}$ is of size m and $1 < m < T$.

When observations of $x \in X$ are recorded periodically over a time period T, we can think

of them as a sequence of frames $x_0, x_1, x_2, ..., x_i, ..., x_T$. For this research, the spatiotemporal sequence generation problem is to generate the next most likely frame x_i observation, given a window of previous observations $x_{i-m}, ..., x_{i-2}, x_{i-1}$. The problem can be formalized by equation 4.1, where g represents the *generate* function performed by the model.

$$x_i = g(x_{i-m}, ..., x_{i-2}, x_{i-1}) \tag{4.1}$$

4.2 Data Preparation

Before the data is used to train and evaluate the model, the following preprocessing steps are applied to transform the data: 1) data normalization, 2) data slicing, and 3) dividing the dataset into a training and testing set.

1. **Data normalization:** First, the data is normalized by scaling the velocity values between 0 and 1. This data normalization improves the gradient descent optimization during the neural network's training, which is a common requirement for deep learning methods. This scaling is done according to Equation 4.2 below.

$$v_{scaled} = \frac{v - v_{min}}{v_{max} - v_{min}} \tag{4.2}$$

   This scaling technique provides robustness to very small standard deviations in the dataset's velocity values. During this process, the obstacle cells are left with a value of -1 to distinguish them from the fluid while maintaining the velocity values in the 0 to 1 range.

2. **Data slicing:** This step focuses on creating simulated sub-sequences to train the model. The input and output datasets are generated using the original dataset, which consists of the simulated [fluid] sequences. The model takes as input a specific window $\mathcal{W}$ consisting of m frames from the simulated sequence of size T; it then uses this window to generate the next frame. Since $m < T$, the length of the input dataset

elements must be reduced to match the this window. To do this, each original sequence is segmented into sub-sequences of length m (size of $\mathcal{W}$). After segmenting the original sequence of size T, it results in more than one sub-sequence of size m in the input dataset.

Figure 4.3 illustrates this process, where the input dataset to the model (the set of fluid sub-sequences X of size m over a time window $\mathcal{W}$ using definition 3), will be used to generate the output set (the next set of frames). During the training step, the model will learn how to infer x_t from the previous m frames (see Equation 4.1).

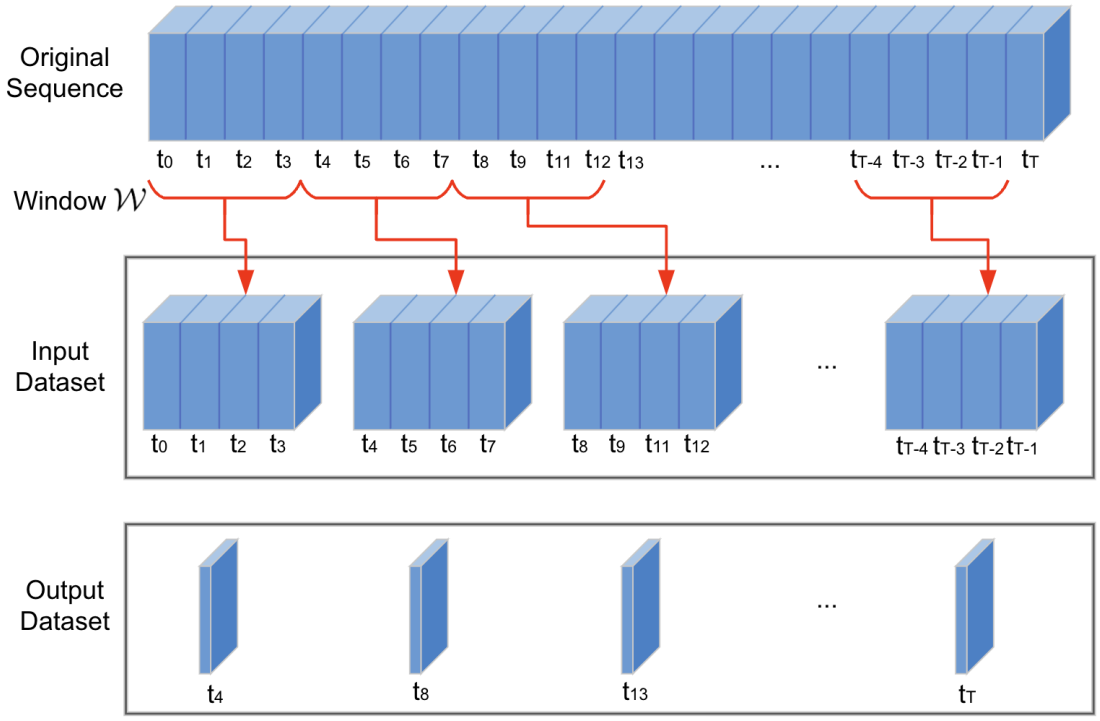


Figure 4.3: Slicing of sequence into $\mathcal{W}$ sub-sequences used to train the model.

3. **Training, Validation, and Testing sets:** Finally, the dataset is shuffled and then randomly divided into training, validation, and testing sets with an 8:1:1 ratio, respectively. This is done to validate and test the model using the cross-validation method

with samples not used during training. This aims to provide an unbiased evaluation of the model's efficacy, which – ideally – should appropriately generalize (for new data) without over-fitting (training data).

4.3 Model Architecture

The DL solution proposed in this research is an end-to-end model, meaning it will perform all the tasks from data input to generating the fluid flow simulation output in one unified process. In contrast, other related research uses machine-learning or DL techniques for only a specific part of the simulation, leaving the rest to a classical numerical method and making the process more complex. A benefit of our simple approach with only one task is eliminating the need for complex pipelines between separate parts in the simulation process, thus reducing development time and potential sources of error. Additionally, end-to-end models can better adapt to diverse datasets and changing environments since they learn directly from raw data, capturing intricate patterns and relationships that might be missed in traditional approaches, like DNS.

The goal of this model is to generate predictions of a fluid flow's evolution. To accomplish this, the model looks at past states in the flow and generates the following future state. The future state can then be used as an input to continue generating new states in the simulation. This step can be repeated as many times as necessary as a feedback loop shown in Figure 4.4 below. As a result of this feedback loop, the model can produce a long fluid flow sequence. In Figure 4.4.a we can see how the resulting frame is put at the end of the input sequence to generate a new frame (see Section 4.3.2). Figure 4.4.b shows how the model looks at a certain window of the frame and moves forward in time to generate the rest of the sequence. At the beginning of a simulation, the model uses an initial condition or "ground truth" represented by a number of frames equal to the sliding window ($\mathcal{W}$). As $\mathcal{W}$ slides, new frames are generated, which are in turn used to generate more subsequent frames. Eventually, an entire sequence can be generated, knowing only the first frames from the initial fluid's condition, and the rest are completely generated by the model.

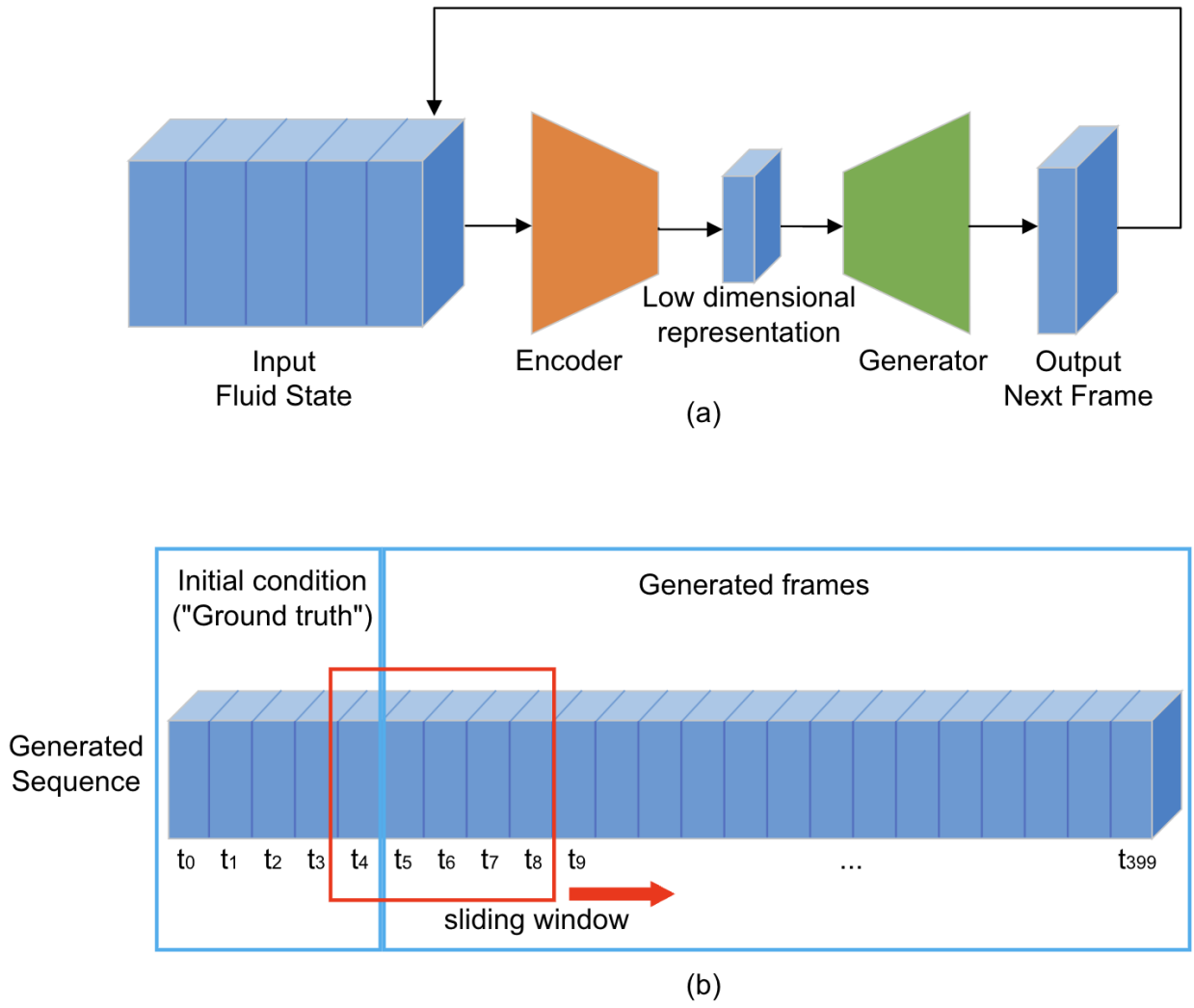


Figure 4.4: Diagram showing a) the flow between the model's main components and b) sliding window approach for prediction.

Because of the data's spatiotemporal characteristics, the neural network has to be capable of analyzing and learning the evolution of fluid flows in two dimensions: space and time. A CNN (See Section 2.6) can help understand the spatial structure of the fluid to extract the flow's features and patterns in space. Additionally, the model needs to "remember" what happened in the past to produce the next state, so it needs a memory mechanism that can be provided by a recurrent neural network such as an LSTM (See Section 2.5), commonly used in Natural Language Processing tasks. As mentioned before, these two types of neural networks have been combined to create the ConvLSTM (See Section 2.7) as an extension of the LSTM network that can also "look" for features in space and time. For this reason, the ConvLSTM network was chosen to implement the neural network architecture proposed in this study.

Because this data has many dimensions and complexities, a dimensionality reduction is applied to capture the principal components of the flow before generating the next frame. This ensures that the model will rely on a minimal representation of the fluid flow that accurately describes its behavior. For this model architecture, the dimensionality reduction is implemented by an Autoencoder (See Section 2.8) neural network that can produce it as part of the same model.

In summary, the model has two main parts, as shown in Figure 4.5: 1) an Encoder that reduces the data's dimensionality and 2) a Generator that gets a representation of the past frames and creates the next one.

4.3.1 Encoder

The Autoencoder is a neural network architecture composed of an Encoder and a Decoder, and it is used for dimensionality reduction (See section 2.8). It takes data with many dimensions and creates a representation of this data in a lower-dimensional space. It is used similarly to classical statistical methods, such as singular value decomposition or principal component analysis. As explained in the related works chapter 3, autoencoders were previously used for fluid flow analysis, such as identifying the main components in a fluid flow

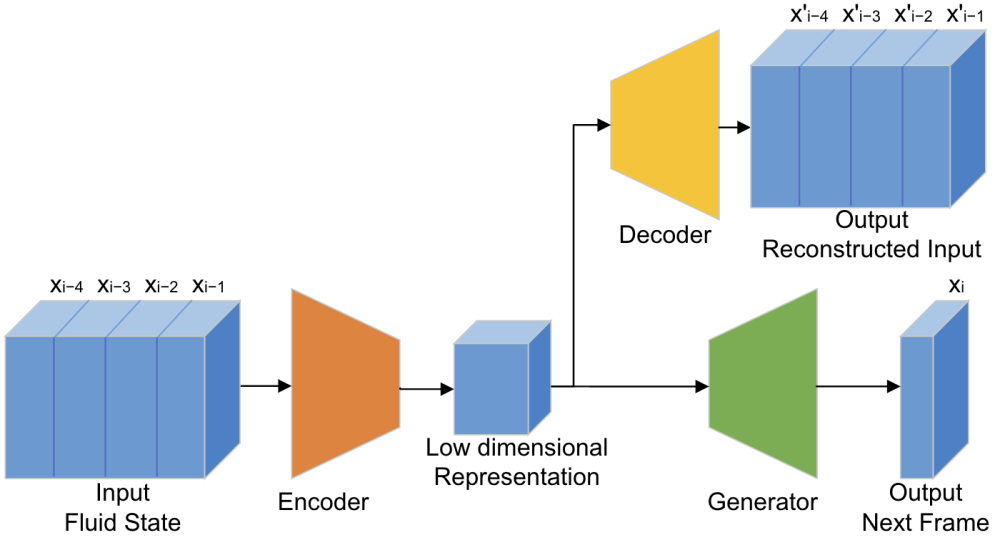


Figure 4.5: Schematic diagram of the model architecture

and identifying eddies in the flows. It has also been used for dimensionality reduction for other methods that are not deep learning models.

4.3.2 Generator

The generator takes the lower-dimensional representation of the fluid flow state and generates a new frame in the sequence. Using a lower representation of the data instead of all the original dimensions makes this work easier because the generator will get as input the main components that can describe the flow.

4.4 Model training

The Encoder is trained in conjunction with the Decoder component, which takes the lower dimensionality representation and tries to reconstruct the original input. If the decoder can reconstruct the original data using the reduced representation from the encoder, this means that the representation captures the main elements of the sequence, and the encoder works correctly. This decoder is auxiliary and discarded once the model's training is completed.

Cross-validation [10] is used to ensure that the model performance generalizes well to unseen data during training. This method, commonly used in machine learning, involves splitting the dataset into three parts: a training set, a validation set, and a testing set. The training set is used to train the model, and simultaneously, the validation set is used to evaluate the model's performance during training. Once the model is trained and optimized, it is tested on a separate and unseen test dataset to assess its generalization performance. This technique also helps prevent overfitting by providing an independent dataset for model evaluation. It ensures that the model's performance estimates are more reliable and indicate its performance on new data.

The neural network training job was distributed across the two GPUs available on the server. This was done using the Mirrored Strategy, a synchronous data parallelism algorithm for neural network training. This algorithm replicates the model on both GPUs and splits the dataset between devices. The CPU prepares and sends the data batches to the GPUs. During training, each GPU performs a forward pass over the model on different input data to compute the loss function; subsequently, gradients are calculated on each device based on that loss. Both sets of gradients are then combined with an all-reduce operation by averaging them and re-distributing them across devices to update the model parameters on each replica, synchronizing them. This process is repeated for each data batch for every training epoch.

4.5 Hyperparameter Optimization

The model architecture and training have specific characteristics or hyperparameters that can take various possible values. These hyperparameters impact the model's performance, i.e., different combinations of such values may result in different model efficacy; consequently, it is important to find a suitable hyperparameter combination to optimize the model efficacy.

Hyperparameter optimization with Random Grid Search [10] in ML involves systematically exploring a predefined hyperparameter space by randomly sampling combinations of hyperparameters, rather than exhaustively searching through all possible such combina-

tions. This approach helps efficiently find an optimal set of hyperparameters for the model by balancing computational resources and exploring the parameter space. Random grid search randomly selects hyperparameter values from specified distributions and evaluates each combination using cross-validation to determine the set that yields the best performance metric. This technique can effectively search a large hyperparameter space, improving model performance without exhaustive computational costs.

The hyperparameters considered for this model are:

- Learning rate: it determines the update step size of the weight in the neural network at each iteration during the optimization process, i.e., the loss function moving towards a minimum. A high learning rate can lead to rapid convergence at the risk of overshooting the optimal solution, causing the model to diverge. On the other hand, a low learning rate ensures steady convergence at the sake of a slower training process which can get stuck in local minima. This parameter significantly impacts the efficiency and effectiveness of the model training. The best training results were achieved with a learning rate of 0.001.

- Number of layers: the number of neural network layers affects the *capacity* of the model to learn complex representations. In the context of this research, more layers can enable the model to capture more hierarchical features and intricate patterns in the fluid-flows. This can improve the model's performance. However, having more layers makes the model more complex and more susceptible to overfitting the training data. The model achieved the best results using 3 ConvLSTM layers on each Encoder and Decoder component and 4 ConvLSTM layers in the Generator component.

- Number of filters or kernels on each layer: convolutional filters detect spatial features in the input data. These filters enable the model to capture both spatial and temporal dependencies in the fluid flow. The number of filters impacts the model's ability to extract relevant features. Having more filters can capture more complex data patterns

but at the cost of increasing computational cost. This parameter affects the accuracy and efficiency of the model. In the ConvLSTM layers of the Autoencoder, this model architecture got the best results using 64 filters in the first and last layers and 32 in the other ones. In the Generator component, the first layer uses 32 filters and 64 in the rest.

- Size of the filter: this refers to the convolutional filters' dimension to capture features of the input data. The filter size determines the scope of the local spatial region examined by the model. Larger filters can capture broader spatial patterns but may miss fine-grained details, while smaller filters focus on more localized features, potentially overlooking broader context. This impacts the model's ability to detect relevant features. The following filter sizes in the ConvLSTM layers of the architecture were used to get the best results, in order from the first to last layer of each component: the Encoder has filter sizes of 4 by 4, 3 by 3, and 2 by 2; the Decoder has filters of 2 by 2, 3 by 3, and 4 by 4; and the Generator filter sizes are 3 by 3 for all of its layers.

In order to define a search space of hyperparameter combinations, a range of values is set for each of these hyperparameters. Potential combinations are selected by random sampling, then a model is trained using those combinations, and the one with the best performance is selected. Once the best combination is found, the model is trained during more epochs to get the final version of the model.

When choosing the m size of the window $\mathcal{W}$, several aspects were considered, e.g., the amount of "historical" data used to generate a prediction. Similar to the Exponential Moving Average (EMA) heuristic [19] [29], which can be used in time-series forecasting scenarios and LSTM models to weight the amount and relevance of historic measurements vs. predicted ones. In our case, we can think of m as the amount of historical information that the model will receive to make a future prediction. In this sense, using a small window will not give enough information to the model. On the other hand, having a bigger window size provides more information and is better for the model, but if m is too big, it will need too much

historical information, making the model pointless. Additionally, a bigger window expands the input dimensions, so more computations would be required to compress the data to create the lower dimensional representation. This increase in computations makes the model slower to execute and train, which given our limited computational resources, could render our model impractical. In addition, since the sequences have 400 frames each, the window size m has to be a divisor of 400 to split them into sub-sequences for data preparation, as explained in 4.2. Overall, since a window of size 2 would be too small to provide the model with enough information, a window size of 4 was chosen since it is the smallest m that allows for the even division of sub-sequences while keeping the model practical.

4.6 Proposed Architecture Details

The final model architecture has two components: the **Autoencoder** and the **Generator**. The Autoencoder is divided into the Encoder and Decoder. The window $\mathcal{W}$ of $m = 4$ frames is first input into the Decoder, which creates a low-resolution representation of the simulation. Next, this low-resolution representation is input into the Generator to create the next frame state of the fluid flow. All these components are implemented in the neural network by a set of ConvLSTM layers followed by either MaxPooling or UpSampling layers. ConvLSTM layers extract features in the data by using convolutional operators called filters or kernels. MaxPooling and UpSampling layers compress or uncompress the intermediate data outputs between ConvLSTM layers. MaxPooling downsamples the input by taking the maximum value in the kernel window along the spatial dimension. UpSampling layers resize the input by interpolating its values. Figure 4.6 shows a detailed diagram of this architecture where we can see the order of these layers. In the Encoder, successive layers of ConvLSTM and MaxPooling layers downsample the data, resulting in a representation that is half the input size. This representation is then reconstructed in the Decoder by upsampling it using UpSampling and ConvLSTM layers. Each of these layers has a different amount of filters or pool windows of different sizes, respectively. Figure 4.6 also shows the output dimension of each layer.

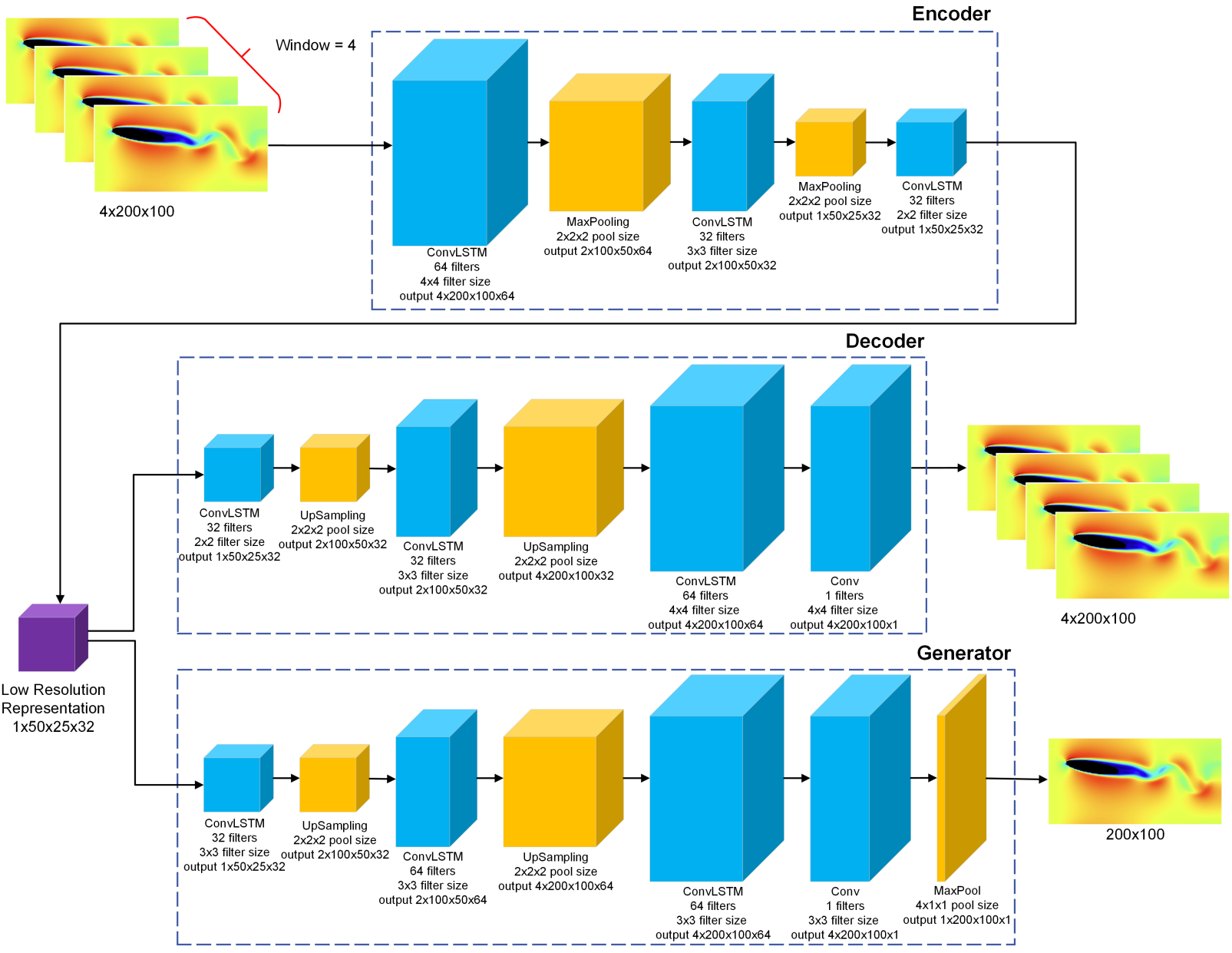


Figure 4.6: Detail diagram of the model architecture

4.7 Evaluation Methods

The model evaluation is done using three strategies:

1. Calculating the Mean Square Error between the original and generated simulations.

2. Visually inspecting the simulation result by rendering an image with the generated data and comparing it with the expected data (the ground truth).

3. Creating a histogram of frame values for the original and the generated data, these will be compared "side-by-side" to analyze similarities and identify possible drastic differences.

4.8 Software and Tools

For the implementation of all the methods described in the previous section, the Python [5] scripting language with the following libraries: TensorFlow [7] and Keras [1], for the neural network implementations and training; Numpy [3] and scikit-learn [6], for data manipulation and preprocessing; and Matplotlib [2], to generate the plots and visualizations.

VS Code was used as an IDE for code implementation, and data analysis of the results was done using Jupyter Notebook [4].

The server used for training and evaluation of the neural network model has the hardware specifications described in Table 4.1 below.

Table 4.1: Server Hardware specifications

CPU	Intel(R) Xeon(R) Gold 5118 CPU 2.30GHz of frequency 12 cores
RAM	192 GB
GPU	2x Nvidia Tesla V100 with 16 GB or RAM each Nvidia Volta Microarchitecture Compute Capability 7.0

Chapter 5

RESULTS

This chapter covers the experiment design and setup, describing in detail results obtained during different phases and tests. This chapter is organized as follows: first, we discuss the results of the training and its validation in Section 5.1, then in Section 5.2 and Section 5.3 we explore the DL model performance from the perspective of each component individually using the methods defined in Section 4.7. Lastly, in Section 5.4, we analyze the performance of the DL model based on the two main evaluation metrics defined in Section 1.2 (MSE and execution time).

Results from experiments in this chapter are based on the dataset described in Section 4.1. The experiments performed can be categorized into two main types based on the obstacle shape used:

1. Circular obstacle: the obstacle is randomly positioned in the simulation space, and its size is given by radius $\mathbf{r} = qH$, where $q \in [\frac{1}{9}, \frac{1}{5}]$ and H is the height of the simulated region.

2. Elliptical obstacle: the obstacle is also randomly positioned in the space, and its size is given by semi-major axis $\mathbf{a} = qH$, where $q \in [\frac{1}{5}, \frac{1}{3}]$ and H is the simulated region height, and the semi-minor axis $\mathbf{b} = p\mathbf{a}$, where $p \in [\frac{1}{5}, \frac{1}{4}]$. The ellipse is also tilted with an angle $\boldsymbol{\alpha} \in [-30°, 30°]$ with respect to $\mathbf{a}$ and the Cartesian x-axis.

The ranges of the obstacle dimensions and positions were chosen to fully fit the shape inside the simulated space without touching its perimeter. The dimensions of the Ellipse object were chosen to maintain a shape similar to that of an airfoil as described in Section 4.1,

and its inclination range to represent common wing "angle of attack" including the critical or stalling "angle of attack" (typically between 18° and 25° degrees)[8].

5.1 Training Results

The neural network was trained for 500 epochs, across which, the Mean Squared Error (MSE) loss functions results were collected for both the Autoencoder and the Generator. The training error is captured in Both plots in Figure 5.1 show that the is generally lower than the validation error; this is to be expected given the proposed model was evaluated with sequences used during the training process.

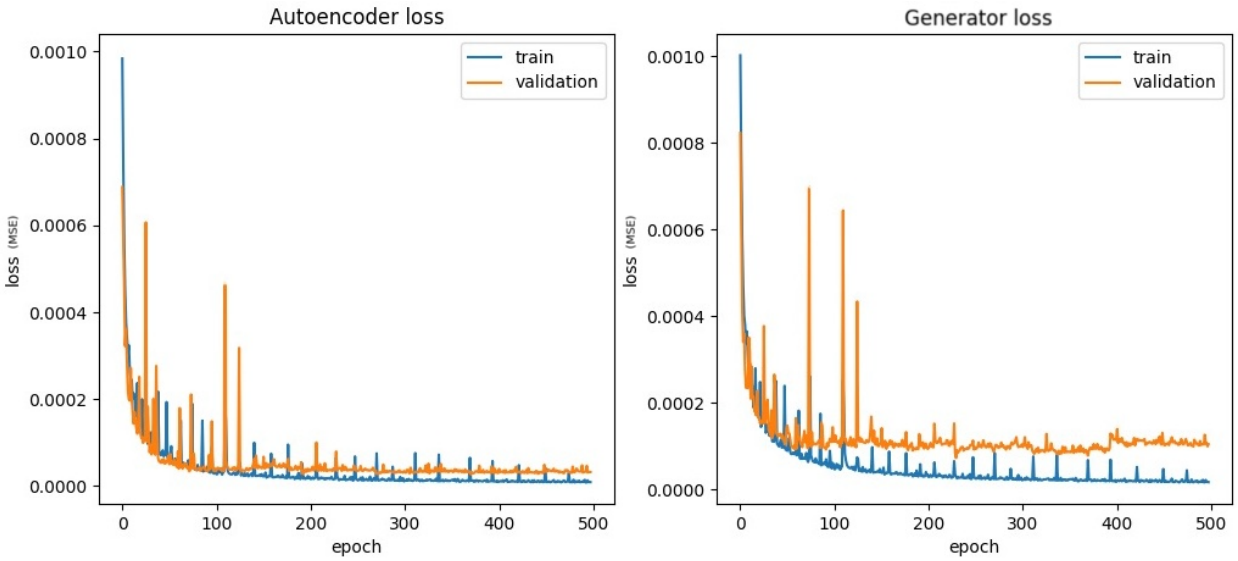


Figure 5.1: Autoencoder and Generator loss evolution during training

Figure 5.1 demonstrates the stability of the MSE loss function's evolution during training. The loss trend quickly descends, and after 200 epochs, the error stabilizes and converges to its final value. In addition, note that the slight difference between the training and validation errors signifies that the model does not excessively over-fit the training data. Regular error spikes appear during training, representing instances where the model encountered local

maximums before finding a local minimum. This happens when, during parameter optimization, a new combination of the network's weights is worse than the previous one, but then it improves again. This further underscores the model's reliability. These spikes decrease as the training advances, indicating a stable and reliable training process. Another critical aspect to note is that those spikes happen almost on the same epoch number and with a similar intensity for both of the model's components (Autoencoder and Generator), meaning that these components are working in collaboration to find the best result. This supports the election of the model architecture.

Table 5.1: Training and Testing errors (MSE)

	Autoencoder	Generator
Training	7.7727×10^{-06}	1.5429×10^{-05}
Validation	3.2173×10^{-05}	2.0414×10^{-05}

A comparison between training and validation errors is displayed in Table 5.1 for both the Autoencoder and Generator components of the model. The low error rate in all cases indicates the model's good performance. It is important to remember that validation samples were not used during training, which is noticeable in how lower the training errors are compared to the validation errors. Since the model has already "seen" the training samples to optimize its weights, it learns how to approximate the output based on those values.

In the following sections, we analyze the model's performance more deeply, examining each component's performance individually.

5.2 Autoencoder Results

To evaluate the Autoencoder, we need to verify that the Decoder can reconstruct the original information using the low-resolution representation of the data created by the Encoder. Because the input to the model is a set of frames according to the window described in

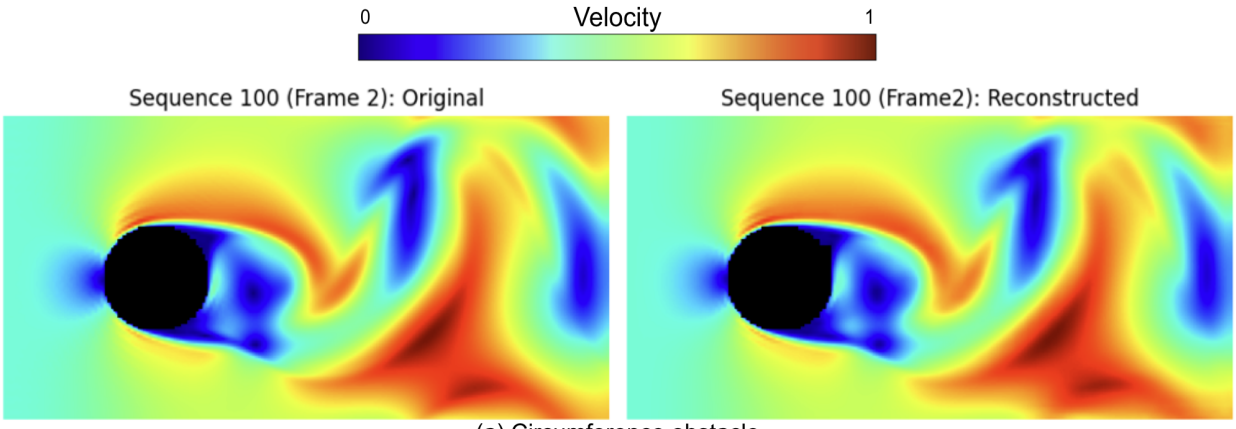
Chapter 4, the Decoder output will also have those dimensions. This means that to evaluate the Decoder's output, we must compare all the frames in the set.

We did this evaluation with two methods:

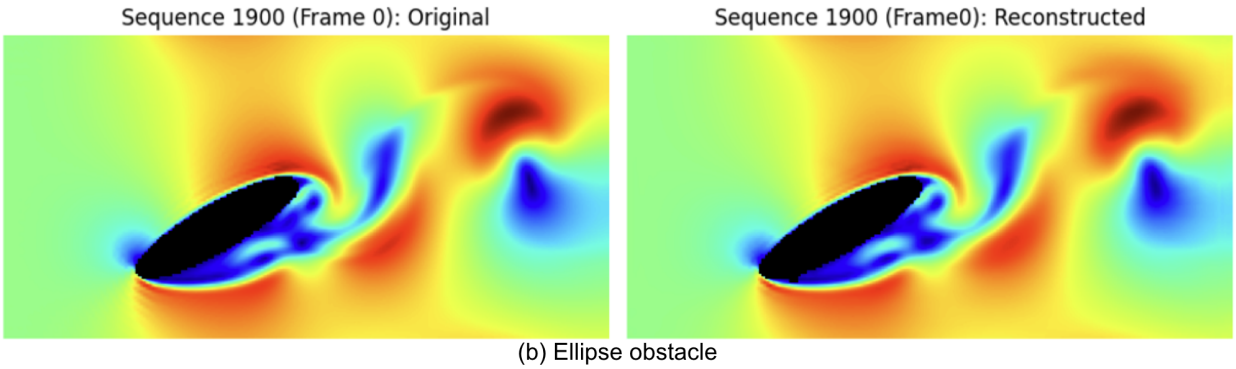
1. Comparing the reconstructed frames with the originals.
2. Comparing the histogram of velocity values.

Using the Decoder's output sequence representing the fluid state, a *heatmap* of fluid velocity values was rendered to compare the results visually. Figure 5.2 shows examples of frames with each type of obstacle. In each case, we show the Original frame to the left and the Reconstructed frame output by the Decoder to the right. We can see that although there are some minor differences, both frames are almost identical. Similar results were obtained for the rest of the sequences. The differences found can be seen in small changes of intensity of the *heatmap* colors, which may indicate that the velocity values approximate the original ones but are either lower or higher than expected. Although there are some minor differences, the similarities between the Original and Reconstructed frames give us a clue that the Autoencoder is working as intended.

For the following evaluation method, we compare the velocity values between the Original and Reconstructed frames to verify their similarity. This is done by creating a histogram of velocities, i.e., a frequency count of velocity values on each grid cell in the frame. Figure 5.3 shows examples of those histograms for Origianl and Reconstructed frames containing each type of obstacle. The frame examples are the same as Figure 5.2 used in the previous evaluation method. On the x-axis, we have the range of all the velocity values in the frame. These values are between 0 and 1 because the data was previously normalized, as explained in Section 4.2. On the y-axis, the frequency or occurrence of each velocity value is represented. To compare all the velocity histograms, we calculated the Jensen-Shannon distance between each frequency distribution. The resulting average distance between the original and reconstructed frames was 0.021, with a standard deviation of 0.014.



(a) Circumference obstacle



(b) Ellipse obstacle

Figure 5.2: Original vs Reconstructed frames

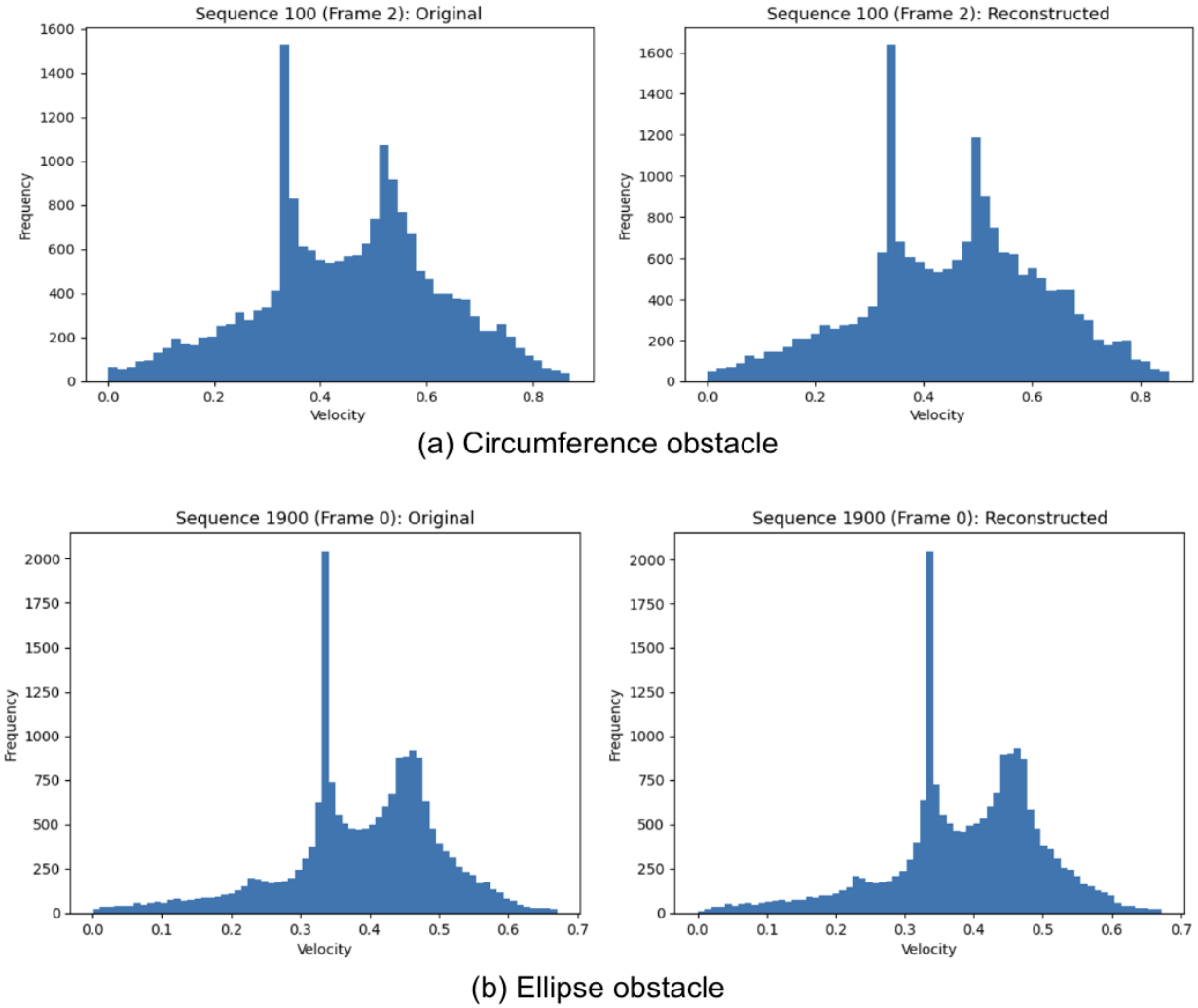


Figure 5.3: Original vs Reconstructed frames velocity histograms

Both evaluations tell us that the Autoencoder successfully reduces the data's dimensionality so that the original data can be reconstructed using that low-resolution representation. This means that the training of this model's component was successful. Although there are some minor errors, the fluid flow structure remains correct, and the approximations of the velocities are very close.

The Autoencoder is a vital component of the model because it guarantees that the model

successfully extracts enough information to represent the original data. This process of reducing the amount of information with a lower representation is important to support the next phase, which is the generation of the next fluid state.

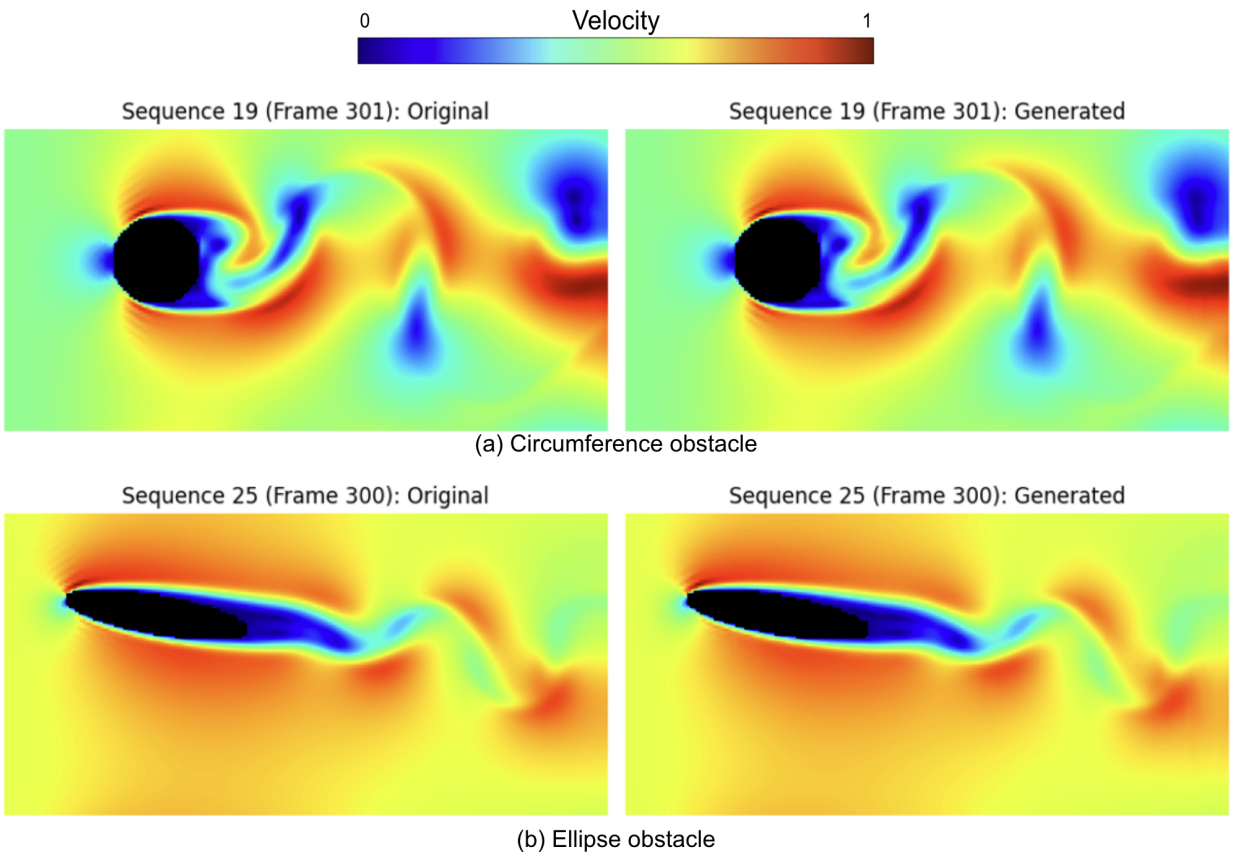
5.3 Generator Results

The Generator's goal is to generate the next fluid flow state using as an input the low-resolution representation created by the Encoder. To evaluate this component, the resulting frame is compared against the expected frame taken from the dataset created by the numerical simulation. For this evaluation we used similar methods than the Autoencoder evaluation.

Figure 5.4 shows a comparison between a generated frame velocity *heatmap* at the right, and the expected frame at the left. The images show a result example for each of the possible obstacle types. Similar to the Autoencoder results, very small differences appear in the *heatmap* colors, however, both images are almost similar.

We created the velocity histogram for each generated frame and compared it to the original frame. Figure 5.5 compares 2 examples of the velocity histograms. Then, we calculated the Jensen-Shannon distance between the velocity distributions of the frames in all the sequences. This results in an average distance between the expected and generated frames of 0.043 with a standard deviation of 0.010. The low average value of the distance metric between both distributions indicates that the original and generated frames are very similar.

These evaluations show that the model can successfully approximate the next state in the fluid flow sequence. The same level of accuracy in both evaluation methods was observed across all the cases in the testing dataset. Although there are some differences between the expected and generated frames, the model can replicate the evolution of the fluid flow structure across the sequence simulation time.



(a) Circumference obstacle

(b) Ellipse obstacle

Figure 5.4: Original vs Generated frames

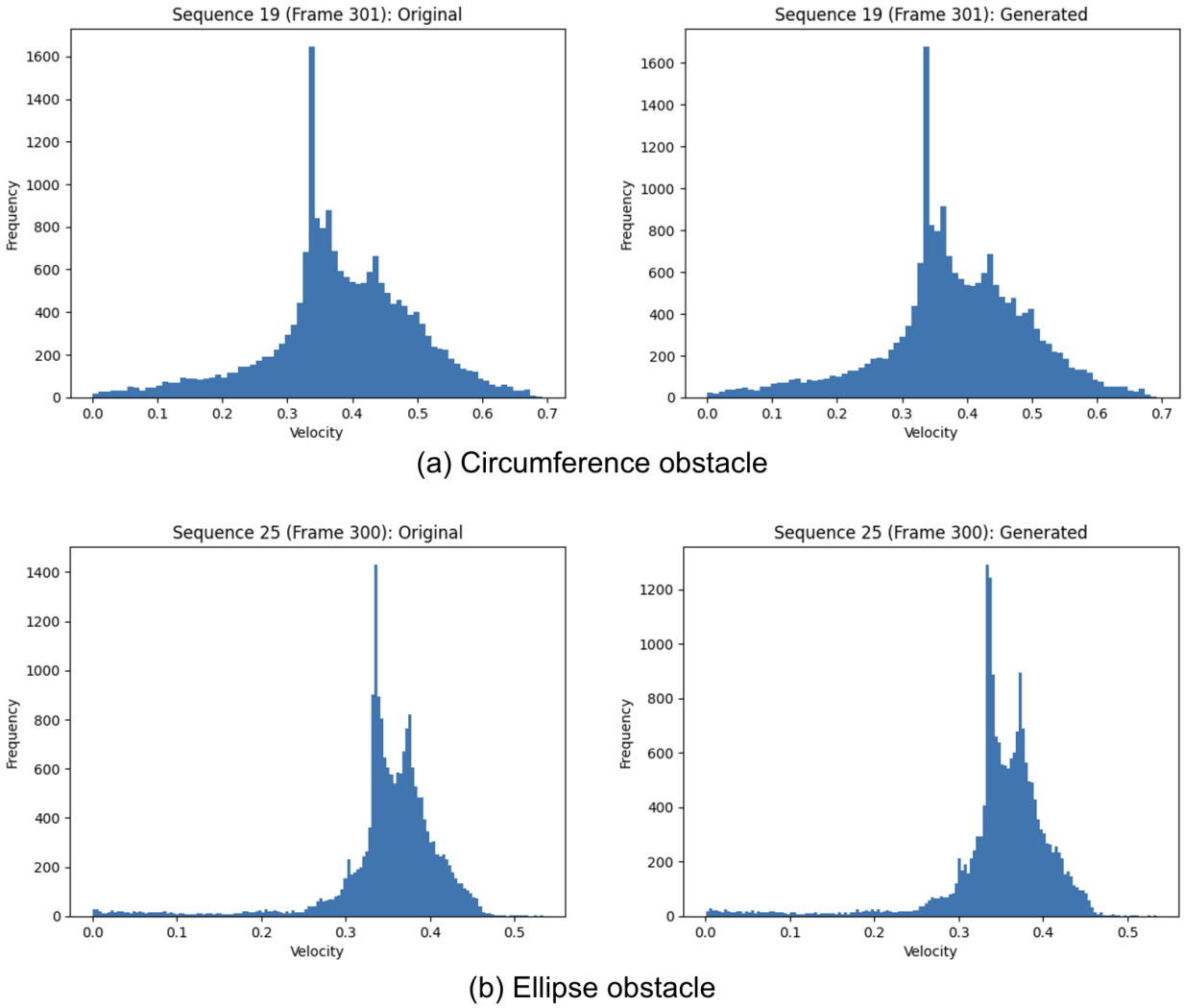


Figure 5.5: Original vs Generated frames velocity histograms

5.4 Model Performance

In this section we explore the results of the two main metrics chosen to evaluate this models performance as explained in Section 1.2. These metrics are: the Error measured with MSE, and the execution time of the simulation measured in seconds.

5.4.1 Error Measurements

Figure 5.6 shows the results of the MSE metric. The results are divided into three groups, one for all the shapes in the dataset together, one with only Circumferences obstacles, and another for Ellipses obstacles. The minimum, maximum, and average error values are plotted for each group. It is important to mention that the dataset is balanced, meaning that the amount of examples with each obstacle type is the same, which is essential to ensure fairness in the results.

The following analysis can be done by looking at the MSE plots in Figure 5.6. We can see that the minimum and maximum errors across the entire dataset are both in simulations with an ellipse obstacle. Additionally, the difference between the minimum and maximum error is lower for the circumference than the ellipse obstacle. This could be caused by circumference obstacles presenting less variability in their shapes, with only a change in the radius, while ellipses obstacles have more diversity in their shapes. This variability in the obstacle shapes makes it more challenging for the model to learn how to simulate the ellipse objects. However, because the average errors are similar between the two types of obstacles, we can conclude that no specific obstacle shape is significantly more difficult for the model to simulate.

5.4.2 Execution time

As explained in Section 1.2, the goal of this model is to reduce the execution time of the simulation while maintaining a low error to preserve the pattern structure of the fluid flow in the generated sequence. Table 5.2 compares execution time between the simulation and the

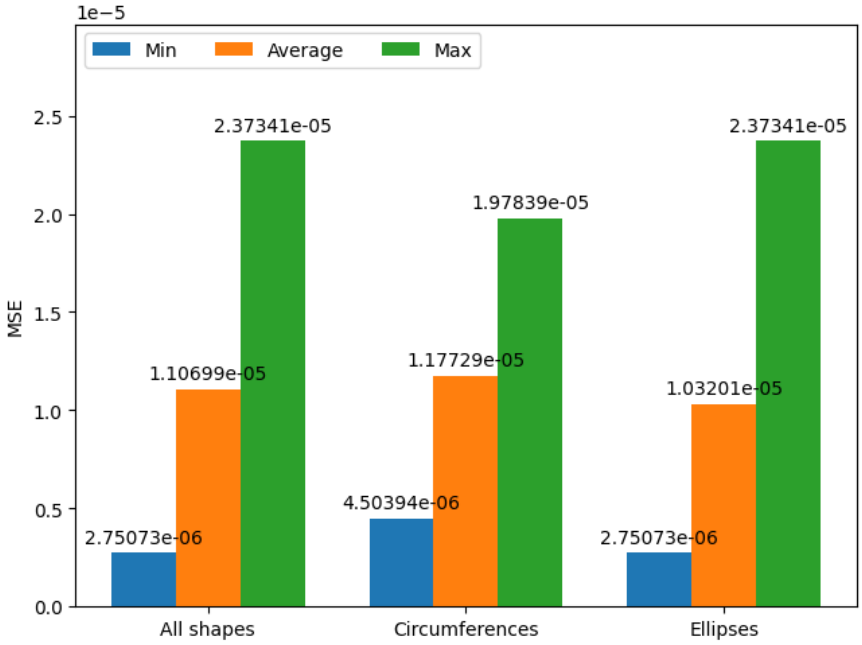


Figure 5.6: Model MSE error metric

DL Model. The simulation took, on average, 191 seconds (3.2 minutes), while the DL model took, on average, only 42 seconds. This represents a 4.5 times improvement in execution speed over the numerical simulation. This result shows that using this DL model improves the simulation's performance, reducing the total execution time while successfully simulating the evolution of the fluid flow.

Table 5.2: CFD simulation vs DL Model execution time

	Average Execution Time
CFD Simulation	191
DL Model	42

Chapter 6

CONCLUSION

6.1 Conclusion

In this research, we analyze the use of ConvLSTM to implement a deep-learning model for turbulent fluid flow simulations. We propose a neural network architecture to create a Reduced Order Model (ROM) and generate a fluid flow by predicting its spatiotemporal dynamics. This architecture is implemented completely using the ConvLSTM network for all its components. The model was trained and tested using a dataset of fluid flows interacting with circumference and ellipses obstacles of diverse sizes and positions in a two-dimensional space. The data was generated using a Direct Numerical Simulation (DNS). The performance was evaluated using the MSE error metric, achieving results comparable to a DNS simulation. When it was evaluated using the training and testing data, the model achieved similar error metrics, meaning that it was able to generalize well to unknown data. It is also worth mentioning that the error was balanced across the two types of obstacles. Additionally, the simulation execution time with the DL model was four times faster than the DNS. These results demonstrate that a neural network model proposed, combining an autoencoder and a generator implemented with a ConvLSTM, is suitable for predicting turbulent fluid flows and can accelerate Computational Fluid Dynamics simulations.

6.2 Contributions

6.2.1 Data preparation and training methods

The dataset we created for this research builds upon others used in CFD research by including a greater diversity of obstacles with different shapes and positions in the simulated space. Additionally, it takes into account a common and important shape in fluid dynamics, which

is the airfoil shape, by using a simple approximation of it.

The methods defined to preprocess the raw data of simulated fluid flows are essential to prepare the dataset to be used for the model's training process. This greatly influences the final model performance. The methods we describe in this study can be easily extended and adapted to other applied cases, like longer simulated sequences or different window $\mathcal{W}$ sizes. Additionally, this could be applied to any neural network training problem that relies on a multidimensional time-series dataset.

6.2.2 Model arquitecture

This research demonstrates the effectiveness of the ConvLSTM type of neural network in CFD by presenting a model architecture that is totally implemented with ConvLSTM and can achieve similar results to a DNS of fluid dynamics. This architecture was successful in other domains involving multivariate time series, like weather prediction, and we show how a solution in CFD can completely rely on this neural network.

The DL model presented further demonstrates how a data-driven approach with an end-to-end model can be used to generate fluid flow simulations in CFD applications. Additionally, the model architecture shows how useful is to include the creation of ROMs with an Autoencoder as a component of the DL model. The proposed model could represent the input data with half of the initial dimension and still reconstruct the original data. By including the dimensionality reduction as part of a unified model, the entire CFD process is simplified because it reduces the steps in the process to only one. Additionally, this research showed how much faster is to execute the neural network model compared with the DNS. By obtaining an improvement in the execution time of about 4 times faster, the research proves that this model can accelerate scientific computing simulations, including CFD.

6.3 Limitations

Some limitations exist in this work that are worth mentioning:

- Available hardware resources: the available hardware had 2 GPUs with 16GB of RAM each. This limits the size of the model and the dataset that could be loaded in memory. Because the model and the dataset have to share the GPU memory simultaneously, it limits the outcome of the model training. While training, the process used about 95% of the available memory on each GPU.

- Dataset: The dataset only has a single obstacle of two possible types with a simple shape. Furthermore, the shapes are static during the simulation. Additionally, all the fluid flows considered have the same Reynolds number.

- Comparing the results to other solutions: because the dataset was different than equivalent methods, there was no way to compare the results without implementing all the other solutions again.

6.4 Future work

This work opens plenty of opportunities to extend this research. These opportunities are related to scope limitations that exist in this work and possible improvements to this research. Future work could focus on extending the dataset to consider a wider range of shape types and complexity. Additionally, moving obstacles could be considered for the dataset. Including fluid flow sequences with multiple obstacles or complex obstacles that use simple ones as building blocks could also be interesting to test. Additional work could consider more complex turbulent flows with different Reynolds numbers and evaluate how well the model results generalize to those cases. In future research, it might be worthwhile to investigate how the window size m affects the accuracy and performance of the model; this could involve determining the point of diminishing returns when increasing the size of the window while giving the model a small amount of input information.

Computational Fluid Dynamics is already a mature field. It utilizes principles from fluid dynamics that are at least 300 years old and relies on computer simulation algorithms and

methods that are about 60 years old. However, recent research like this one shows that progress can still be made with the introduction of Deep Learning to the field.

BIBLIOGRAPHY

[1] Keras. `https://keras.io/`.

[2] Matplotlib. `https://matplotlib.org/`.

[3] Numpy. `https://numpy.org/`.

[4] Project jupyter. `https://jupyter.org`.

[5] Python. `https://www.python.org/`.

[6] scikit-learn. `https://scikit-learn.org/stable/`.

[7] Tensorflow. `https://www.tensorflow.org/`.

[8] Abbott, Ira H, Von Doenhoff, Albert E, and Stivers, Louis, Jr. *Summary of Airfoil Data*. National Advisory Committee for Aeronautics (NACA), 1945.

[9] Ekhi Ajuria-Illaramendi, Antonio Alguacil, Michaël Bauerheim, Antony Misdariis, Bénédicte Cuenot, and Emmanuel Benazera. Towards a hybrid computational strategy based on Deep Learning for incompressible flows. In *AIAA AVIATION FORUM*, pages 1–17, Virtual Event, United States, June 2020.

[10] Yoshua Bengio. Practical Recommendations for Gradient-Based Training of Deep Architectures. In Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade: Second Edition*, pages 437–478. Springer, Berlin, Heidelberg, 2012.

[11] Steven L. Brunton and J. Nathan Kutz. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control, May 2022. ISBN: 9781009089517 Publisher: Cambridge University Press.

[12] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine Learning for Fluid Mechanics. *Annual Review of Fluid Mechanics*, 52(Volume 52, 2020):477–508, January 2020. Publisher: Annual Reviews.

[13] Karthik Duraisamy, Gianluca Iaccarino, and Heng Xiao. Turbulence Modeling in the Age of Data. *Annual Review of Fluid Mechanics*, 51(Volume 51, 2019):357–377, January 2019. Publisher: Annual Reviews.

[14] Hamidreza Eivazi, Soledad Le Clainche, Sergio Hoyas, and Ricardo Vinuesa. Towards extraction of orthogonal and parsimonious non-linear modes from turbulent flows. *Expert Systems with Applications*, 202:117038, September 2022.

[15] Hamidreza Eivazi, Mojtaba Tahani, Philipp Schlatter, and Ricardo Vinuesa. Physics-informed neural networks for solving Reynolds-averaged Navier–Stokes equations. *Physics of Fluids*, 34(7):075117, July 2022.

[16] Renkun Han, Yixing Wang, Yang Zhang, and Gang Chen. A new prediction method of unsteady wake flow by the hybrid deep neural network. *Physics of Fluids*, 31(12):127101, December 2019. arXiv:1908.00294 [physics].

[17] Kazuto Hasegawa, Kai Fukami, Takaaki Murata, and Koji Fukagata. CNN-LSTM based reduced order modeling of two-dimensional unsteady flows around a circular cylinder at different Reynolds numbers. *Fluid Dyn. Res.*, 52(6):065501, November 2020. Publisher: IOP Publishing.

[18] Priyesh Rajesh Kakka. Sequence to sequence AE-ConvLSTM network for modelling the dynamics of PDE systems, August 2022. arXiv:2208.07315 [physics].

[19] Prajakta S. Kalekar. Time series Forecasting using Holt-Winters Exponential Smoothing. 2004.

[20] Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning-accelerated computational fluid dynamics. *Proc Natl Acad Sci U S A*, 118(21):e2101784118, May 2021.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015. Number: 7553 Publisher: Nature Publishing Group.

[22] Bethany Lusch, J. Nathan Kutz, and Steven L. Brunton. Deep learning for universal linear embeddings of nonlinear dynamics. *Nat Commun*, 9(1):4950, November 2018.

[23] Arvind Mohan, Don Daniel, Michael Chertkov, and Daniel Livescu. Compressed Convolutional LSTM: An Efficient Deep Learning framework to Model High Fidelity 3D Turbulence, March 2019. arXiv:1903.00033 [nlin, physics:physics].

[24] Clarence W. Rowley and Scott T.M. Dawson. Model Reduction for Flow Analysis and Control. *Annu. Rev. Fluid Mech.*, 49(1):387–417, January 2017.

[25] Xingjian SHI, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun WOO. Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[26] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov. Unsupervised learning of video representations using LSTMs. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 843–852, Lille, France, July 2015. JMLR.org.

[27] Ben Stevens and Tim Colonius. FiniteNet: A Fully Convolutional LSTM Network Architecture for Time-Dependent Partial Differential Equations, February 2020.

[28] Kunihiko Taira, Steven L. Brunton, Scott T. M. Dawson, Clarence W. Rowley, Tim Colonius, Beverley J. McKeon, Oliver T. Schmidt, Stanislav Gordeyev, Vassilios Theofilis, and Lawrence S. Ukeiley. Modal Analysis of Fluid Flows: An Overview. *AIAA*

Journal, 55(12):4013–4041, December 2017. Publisher: American Institute of Aeronautics and Astronautics.

[29] Dilshod Bazarov Ravshan Ugli, Jingyeom Kim, Alaelddin F. Y. Mohammed, and Joohyung Lee. Cognitive Video Surveillance Management in Hierarchical Edge Computing System with Long Short-Term Memory Model. *Sensors (Basel)*, 23(5):2869, March 2023.

[30] Ricardo Vinuesa and Steven L. Brunton. Emerging Trends in Machine Learning for Computational Fluid Dynamics. *Computing in Science & Engineering*, 24(5):33–41, September 2022. Conference Name: Computing in Science & Engineering.

[31] Ricardo Vinuesa and Steven L. Brunton. Enhancing computational fluid dynamics with machine learning. *Nat Comput Sci*, 2(6):358–366, June 2022. Publisher: Nature Publishing Group.

[32] Rui Wang and Rose Yu. Physics-Guided Deep Learning for Dynamical Systems: A Survey, February 2023. arXiv:2107.01272 [cs].

[33] Yuning Wang, Alberto Solera-Rico, Carlos Sanmiguel Vila, and Ricardo Vinuesa. Towards optimal β-variational autoencoders combined with transformers for reduced-order modelling of turbulent flows. *International Journal of Heat and Fluid Flow*, 105:109254, February 2024.

[34] Ali Girayhan Özbay, Arash Hamzehloo, Sylvain Laizet, Panagiotis Tzirakis, Georgios Rizos, and Björn Schuller. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering*, 2:e6, January 2021.