Assesing the Efficacy of Deep Learning Models in the Context of Active Flow Control

Department of Cognitive Robotics

Sara Boby (4645588)

Supervised by:
Dr. Ir. Cosimo Della Santina
Dr. Ir. Angeliki Laskari

Technische Universiteit Delft
February 9th , 2024

Assessing the Efficacy of Deep Learning Models in the Context of Active Flow Control

Sara Boby
TU Delft MSc Cognitive Robotics

ABSTRACT

In the field of fluid mechanics, there has been a significant shift towards the integration of machine and deep learning techniques to address challenges in reduced-order modeling, flow feature analysis, and control, especially within the realm of active flow control (AFC) for objectives such as lift optimization and drag reduction. Deep learning has taken a central role in advancing state-of-the-art AFC methods by creating data-driven models that mitigate the computational demands of conventional Computational Fluid Dynamics (CFD) simulations, enabling real-time fluid control. Despite the predominance of models trained offline and focused on simple scenarios like laminar flow around bluff bodies, the utility of sophisticated learning methods in AFC has remained largely unexplored.

This research introduces a novel benchmark in fluid dynamics—a soft robotic tentacle actuator—to evaluate the effectiveness of deep learning architectures in complex flow control situations. Through the comparison of online and offline learning frameworks for predicting system behavior, the study elucidates the strengths and limitations of deep learning networks in AFC. The findings underscore the constraints faced by deep learning architectures when dealing with aperiodic motions and demonstrate the significant benefits of adopting an online learning approach over offline training methods, thereby highlighting the advantages of adaptive learning strategies in complex AFC scenarios. The online learning framework displays more stability and increased quality of forecasts at larger time horizons.

NOMENCLATURE

τ	Threshold time constant
H	Prediction horizon
L_x, L_y	Fluid channel dimension, width and height respectively
N	Lookback horizon
N_b	Buffer size
N_x, N_y	Number of nodes in the respective directions in the Eulerian fluid domain discritization
$r(t)$	Developed metric to quantify mixing rate
T	Measurement horizon for particle tracking
$u_1(t), u_2(t)$	Control inputs
x_s, y_s	Coordinate system of the fluid domain
x_{tent}, y_{tent}	Local coordinate system of the soft robotic tentacle
$y_r(t)$	Control output

1 INTRODUCTION

Fluid mechanics, a field traditionally reliant on extensive data from experiments or simulations, is evolving through the integration of machine and deep learning frameworks. These modern approaches provide a flexible means to tackle fluid mechanics' challenges, including reduced-order modeling, enhancing experimental data resolution, and analyzing fluid dynamics [1, 2, 3]. The shift towards data-driven analysis in fluid mechanics mirrors the mid-20th century's turn to numerical methods for solving fluid dynamics equations [3]. Pollard et al.'s [4] emphasis on compiling fluid data archives underscores this evolution, highlighting the importance of accessible data for understanding complex fluid behaviors, especially turbulence [3]. As machine learning intersects with fluid mechanics, it not only advances the field but also serves as a test-bed for deep learning paradigms, pushing the boundaries of algorithmic development in complex physical systems.

The manipulation of fluids for engineering objectives is a field of research that has received increased attention in the past decade due to the advances in computational power and architectures. Thus far, the ability to control fluid behavior passively or actively has proven to be of large importance and has primarily been employed to achieve goals such as

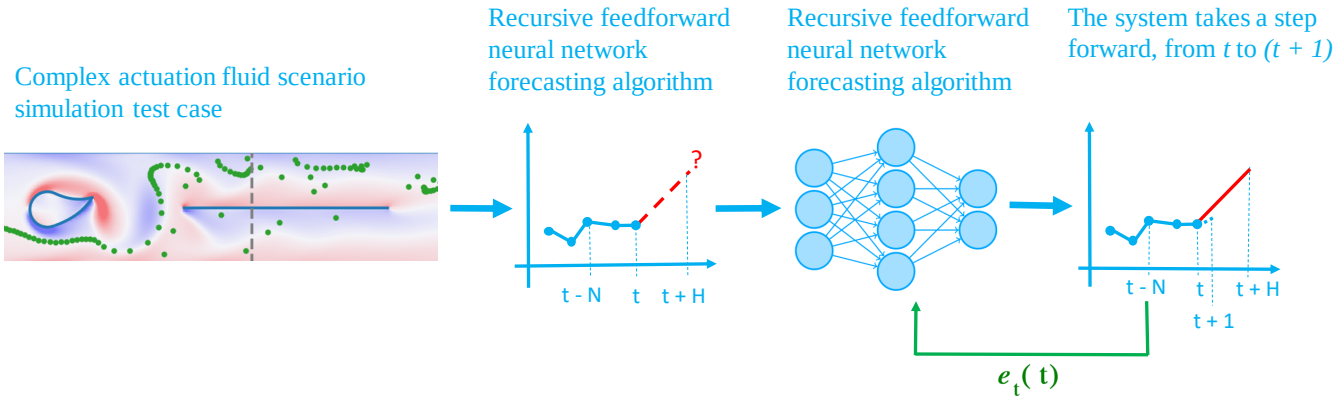


Figure 1 An overview of the main steps in the deployments phase. This is how the developed offline and online frameworks are evaluated. In the case of the online algorithm the green arrow is active, and indicated that during testing the online learning algorithm updates its network weights based on the error $e_t(t)$. In the deployment of the offline learning algorithm, this arrow is not active as the neural network is trained offline on a separate training dataset.

delay of transition, drag reduction, lift enhancement, turbulence management, separation postponement, noise suppression and more [5]. In more practical terms the applications have ranged from optimizing UAV flight [6] , the optimization of underwater robotic navigation [7] to a variety of medical applications including lab-on-chip devices [8].

The shift towards leveraging data for actionable insights underscores the complexities of fluid dynamics, marked by intricate, multi-scale phenomena and nonlinear, unsteady fluid fields. This is critical for designing energy-efficient and reliable robotic devices in complex flow environments. Deep learning in fluid mechanics focuses on three main areas: flow classification, modeling, and control. Flow classification employs techniques like Partial Orthogonal Decomposition (POD) [9] and Koopman generator methods [10] for dimensionality reduction and handling noisy data, essential for advanced flow modeling and control strategies.

There are two overarching paradigms in flow control: passive and active flow control (AFC) [11]. Passive flow control involves the physical or structural composition of the subject or surface to influence its interaction with the flow. For example riblets or surface roughness. Vortex generators on aircraft wings for turbulence and drag management are an example of passive flow control in practice. Conversely, active flow control is defined by the capability to influence fluid behavior by means of an active control system. Typical actuation in AFC are suck and blow actuators (SaOBAs, [12], Figure 3), that involve the control of the addition or removal of flow passing points of actuation. While AFC requires energy input, it offers more effective control than passive methods [11].

The concept of simplifying complex systems to their essential features is crucial for real-time feedback control in systems like fluid dynamics, where high-fidelity Computational Fluid Dynamics (CFD) simulations are too resource-intensive. For example, controlling laminar flow around a cylinder (Reynolds numbers 100-140) typically requires solving for 66,000 variables at each step. However, Peitz et al. showed that a reduced order model with just 12 degrees of freedom, instead of the full 150,000 in a CFD simulation, can still achieve optimal control, boosting controller performance significantly—up to five orders of magnitude faster, akin to the difference between running simulations in OpenFOAM versus MATLAB [13]. This highlights the ongoing research effort into developing efficient surrogate models for active fluid flow control, which has traditionally relied on offline deep learning methods. Online learning presents a valuable strategy for real-time adaptation in models, using adaptive neural networks to adjust weights or architecture based on new data. This approach aligns with continual learning, focusing on long-term adaptability through incremental updates. Despite its potential for handling systems with dynamic or uncertain behaviors, exploration of online learning in the context of active flow control remains limited.

In this work, a complex fluid dynamic benchmark case in the context of active flow control is developed to test and identify the limitations of deep learning architectures in the context of active flow control. An offline and online learning framework are developed to forecast the complex system behavior and evaluated on the complex datasets created to assess the efficacy of these deep learning architectures from two different training approaches. Figure 1 displays the deployment phase of the two approaches.

2 Related Work

State-of-the-art (SOTA) work in active flow control leverages deep learning either to improve system modeling, to derive optimal control policies, or to combine these methods within reinforcement learning (RL) and model predictive

control (MPC) frameworks. Deep RL is particularly noted for its application in active flow control issues, such as vortex shedding suppression and drag reduction in laminar flows behind bluff bodies, necessitating high-fidelity simulations or surrogate models for effective environment estimation[3, 11]. Rabault et al. were pioneers in using deep RL for controlling vortex shedding behind a cylinder, utilizing data from 151 velocity probes in CFD simulations, albeit achieving an almost constant optimal control policy, which brings into question the efficiency of learning given the rich domain information used [14].

Similarly, Mudiyanselage and Gueniat aimed to develop a RL controller based only on wall pressure sensor data, still requiring extensive CFD simulations for training, highlighting the computational demands of such approaches[15, 16].

MPC strategies, which involve solving fluid dynamics equations at every timestep, face difficulties in optimizing control for complex systems within a specific time window. Various methods, including ensembles of Koopman Generators, RNNs, and CNNs, are utilized to construct surrogate models. However, these traditionally assume complete domain knowledge, a largely impractical expectation for real-world application. There is growing effort to enable effective control based on sparse sensor data, aiming to alleviate the computational costs associated with training these models, though studies predominantly focus on simple CFD benchmark cases[17, 10, 18].

Limited research has explored the impact of an online learning approach in this field. Noriega et al. developed a control strategy leveraging neural networks' adaptability for managing unknown nonlinear dynamical systems, suggesting a method where a feed-forward neural network learns the system dynamics in real-time for more accurate and stable control by minimizing differences between goal set-points and model predictions, offering a new research direction for efficient control systems [19].

2.1 Research Gaps

Current research in the realm of fluid behavior modeling and forecasting, has predominantly relied on deep learning models that are both data-intensive and complex. These models are typically trained offline using high-fidelity data from comprehensive Computational Fluid Dynamics (CFD) simulations. However, there are significant limitations and gaps in this approach, primarily due to the simplicity of the application cases studied. The fluid dynamic cases addressed so far have been limited to benchmark scenarios with Reynolds numbers (Re) below 200, such as flow past a cylinder.

The simplicity of the application cases also means that the advantages and limitations of using such complex deep learning models remain unclear. The types of actuation employed, including suck and blow actuators or cylinder rotation, are not particularly complex (see Figure 30). Similarly, the control objectives have been limited to tasks like vortex suppression or drag reduction. These are relatively straight forward objectives,for flow past a cylinder in laminar flow, that have been seen to often give steady state solutions. Given the actuation, these objectives and the minimal latency between control input actions and the system's response imply a lack of temporal complexity in the relationship between inputs and outputs. This simplicity in both the engineering objectives and the mediums of actuation suggests that the fluid cases considered thus far may not be challenging enough to fully assess the true capabilities and limitations of these advanced deep learning models.

Milana, et. al [20], provides an example of bio inspired complex actuation for flow control. Micro-cilia are designed for fluid propulsion at low Reynolds numbers. Evident from nature, the asymmetric beating in micro-cilia can propel fluids at low Reynolds numbers (< 10). The artificial micro-cilia have two actuation points along its 'spine' that essentially dictate its curvature and motion. While in this case, the cilia have a fixed prescribed motion and do not evolve its actuation based on feedback from the flow, it is still a complex actuation that successfully actively manipulates the fluid. This provides an example of complex actuation that needs to be explored and might better illustrate the advantages or disadvantages of using deep learning based solutions for active flow control.

Therefore, there is a pressing need for research that explores more intricate fluid dynamic cases and actuation methods, to provide a clearer understanding of the potential benefits and drawbacks of using complex deep learning models in the context of active flow control.

3 Research Objectives

The work has the leading research question : **How can we assess the efficacy of deep learning models in the context of active flow control (AFC) ?** This overarching research objective is addressed by answering the following research questions:

- What benchmark cases can be defined to evaluate the efficacy of deep learning models
- Are deep learning models capable of learning relationships in complex fluid dynamic scenarios in a purely data-driven fashion ?
- What are the limitations of deep learning models and solutions being used to model or understand fluid dynamic relationships?
- How does an online learning solution compare to traditionally offline trained networks?

Note, data-driven means that information about the physical behavior of the system outside the the control parameters are not used in the solution.

3.1 Contribution

- The development of benchmark datasets for active flow control with higher complexity than addressed so far with the introduction of complex actuation.
- A verified and validated offline learning algorithm based on a feed-forward network to learn the relationship between control input and control output, with no apriori system information. The relationship is learned in a completely data-driven fashion.
- A framework for a counterpart online learning algorithm
- A comparison and analysis of the algorithms on the complex benchmark case to critically assess the efficacy of deep learning models in the context of active flow control and the identification of limitations.

3.2 Overview

The paper will be structured such that the development of the complex datasets and design choices in this process will be delved into first, including an explanation of the complex soft robotic tentacle, the fluid channel set up, and the control parameters. This will also expand on the development of the metric to quantify the behavior we want to track in this complex case. The offline learning algorithm and framework is then introduced - with the focus on the structure of the training and deployment phase, followed a the thorough verification of the architecture and framework used. The counterpart online learning methodology will then be introduced. An analysis of the neural network architecture in terms of loss functions, width, depth, and structural details will be presented to justify the architecture finally chosen to test the online and offline frameworks in the developed complex fluid case. Finally, the results will be presented, along with the conclusions derived from the work in this paper and a discussion of limitations and further work.

4 Simulation Case

A complex benchmark case in the context of active flow control was designed to assess the efficacy of the offline and online deep learning frameworks developed. There are three main components to the design of the complex benchmark case: the flow case, the actuation involved, and the metric monitored. The design choices are made with the intention of introducing an additional level of complexity to the flow cases that SOTA intelligent AFC research has addressed thus far. Introducing complex actuation is the primary way in which the developed simulation case is more complex than the typical flow past bluff bodies addressed in this field thus far.

Note: the structural model of the soft robotic tentacle was verified (subsection subsec:tentacle) during a preceding research assignment. The trade-off and selection of a sufficiently robust fluid-structure interaction (FSI) solver (subsection 4.2) was also done in the preceding research. All other steps involving design, including design of the metric, integration of the model, and algorithms developed were done in the thesis.

4.1 Flow Case Description

A 2D channel flow within the laminar flow regime is considered. In order to ensure that the complexity of the flow does not increase during manipulation by the soft robotic tentacle, a low Reynolds number of 20 is chosen. A low Reynolds number was chosen to ensure that dynamic motion in the soft robotic tentacle does not introduce flow transition or turbulence. In addition to this, by maintaining a simple flow case, the complexity introduced by the actuation can be evaluated in a controlled approach. Past research, in the context of active flow control, employed suck and blow actuators (SaOBAs, [12], Figure 3). Furthermore, maintaining a low Reynolds number is also more practical for generating data as more complex flows take significantly more computational power to resolve, especially when there is fluid-structure interaction involved.

4.2 Fluid Solver

In order to effectively simulate the influence of actuation in the soft robotic tentacle on a steady channel flow, it was required to create a one-way coupling between a structural model of the soft robotic tentacle and an FSI solver (Fluid Structure Interaction). This makes it possible to retrieve fluid domain information, such as the resulting domain velocity, and the density field in every timestep. The fluid solver had to be selected such that computations can successfully converge in cases handling dynamic moving boundaries with ease. The immersed structure in this case has a significantly complex geometry and introduces complex motion in the fluid. Taking this into consideration, through comparison with commercial solvers such as Ansys and Openfoam [21], the LaBIB-FSI (Lattice Boltzmann Immersed Boundary - Fluid Structure Interaction) solver [22] was chosen. LaBIB-FSI is the he product of a TU Delft based Masters thesis from the Aerodynamics department in Aerospace Engineering. This is an open-source fluid dynamic solver that has been validated on multiple fluid dynamics benchmark cases.

4.3 Design of the Complex Actuation : Soft Robotic Tentacle

A soft robotic tentacle is chosen as the mode of actuation for the complex fluid case. The curvature of the tentacle's central axis determines the deformation of the tentacle and can be described as:

$$c(t) = q_1(t) + q_2(t)s \tag{1}$$

Where $q_1(t)$ and $q_2(t)$ are time-dependent functions that can be chosen altered and optimized. These act as the control inputs to the soft-robotic tentacle.

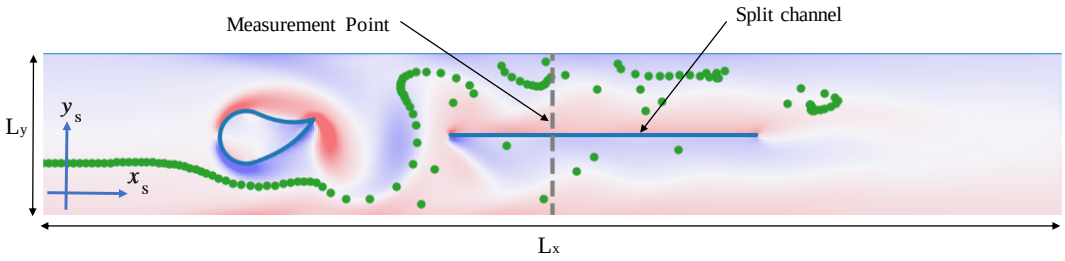


Figure 2 All simulation components captured at one timestep t. The coordinate system of the fluid domain is indicated in the bottom left corner, (x_s, y_s). The measurement point is indicated by the vertical dashed line. The simulated tracer particles can be seen in green and the particles are injected at located at $L_y/3$ of the channel inlet. The fluid flow within the channel is visualized in red and blue displays the vorticity of the flow at this time step, with high-density red being the maximum value of 10 [1/s] and high-density blue being the minimum value of -10 [1/s].

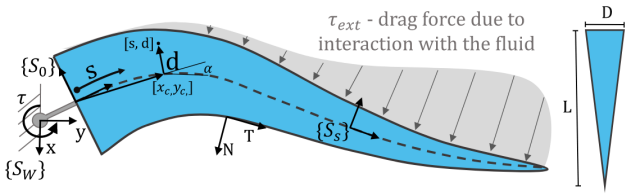


Figure 3 Coordinate system for the tentacle structural model as defined in [23]; this is the structural model used to simulate the soft robotic tentacle via the polynomial curvature model.

The structural model of the soft robotic tentacle immersed in the fluid channel needs to provide sufficient information about the tentacle motion and deformation such that the LABIB-FSI solver can resolver can converge sufficiently at each timestep, providing information about the resulting fluid motion. Namely, the structural solver must be able to provide the following outputs at each time-step. x_s, y_s define the horizontal and vertical coordinates in fluid channel. The following information needs to be returned:

- The displacement in x_s and y_s direction at each point of the discretized tentacle's perimeter relative to the initial configuration of the tentacle (completely horizontal, i.e central axis of the tentacle follows the linear line at $x_{tentacle} = 0$)
- The x_s and y_s velocity components at each point of the tentacle's perimeter

To model the soft robotic tentacle in simulation the Polynomial Curvature model is employed, limiting the curve descriptors to the first-order term. Polynomial curvature models are an effective and efficient way to represent continuous, flexible strictures through a finite approximation.

4.4 Integration

For integration with the LABIB-FSI solver, the tentacle's attachment point within the fluid channel had to be considered. The tentacle is attached flush to a cylinder at a predefined position with the fluid domain. The point of attachment between the tentacle and the cylinder is required to be tangential to ensure that disruptions due to discontinues in the structure's perimeter do not arise, risking additional complexity to be introduced to the flow. Kinematics of the distrusted perimeter of the tentacle-cylinder structure had to be derived, building on the affine curvature model.

4.5 Design of the Engineering Objective: Particle Mixing Rate Quantification

The introduction of the actuated soft-robotic tentacle impacts the mixing of the flow. As a channel flow is being considered, motion in the immersed soft-robotic tentacle can alter the natural trajectory of particles in the flow. Solid tracer particles are often used in experimental fluid dynamics to visualize the flow for velocimetry applications. The idea behind this being that, these tracer particles have densities close enough to that of the flow that it can be tracked without influencing the fluid behavior. Taking inspiration from this, massless point particles are injected into the proposed simulation set-up at a fixed position. This will allow to track the trajectory of the particles as the tentacle is actuated.

4.5.1 Tracer Particle Injection

The mass-less point particles were chosen to be injected at the fixed inlet location at $Ly/3$, where Ly is the height of the fluid channel. This was chosen because if the flow is uninterrupted there is a clear trajectory for the particles released into the channel as can be seen in Figure 4; the particles will cross and exit the channel on the bottom half. If there is actuation in the tentacle, it is possible to influence the flow such that these particles could infact cross the channel on the top half of the channel.

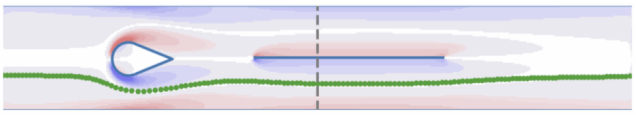


Figure 4 Steady state simulation with no actuation in the soft robotic tentacle and fixed inlet location at $Ly/3$, where Ly is the height of the fluid channel (see Figure 2).

4.5.2 Measuring Point Placement

To quantify this mixing that is possible, a split channel is introduced with a corresponding measurement point placed in the middle. These are positioned such that it takes a free-flowing particle ≈ 4 seconds to reach this point. This gives enough time and space for the tentacle to have an influence on the trajectory of the particle downstream of its actuation, introducing a time dependency to be learned between the control action and reaction. Placing the measurement point closer to the soft-robotic tentacle would result in the particle mixing being evaluated before the soft robotic tentacle influences its trajectory, and it becomes harder to distinguish that the mixing is a result of actuation in the tentacle.

4.5.3 Particle Tracker

To quantify the mixing in in the fluid channel in the x_s direction, the simulated tracer particles need to be tracked.

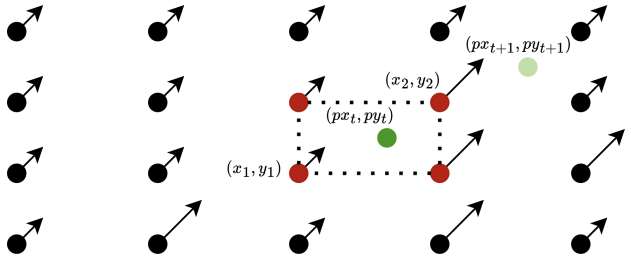


Figure 5 In the opaque green is indicated a tracer particle at time t. Nodes marked in red indicate the four closest noted within the discretized fluid domain that enclose the particle. Each of these nodes have x and y velocity components; the vector arrows indicate this at each node in the uniform Eulerian mesh. The translucent green marker indicates the position of the particle at the next timestep *(Refer to Algorithm 1)*.

The fluid domain is discretized into a uniform mesh, with N_x nodes in the y_s-direction and N_y nodes in the x_s-direction. Therefore, for every time-step in the simulation, the LABIB-FSI solver will return an array of size $(N_x + 1) \times (N_y + 1) \times 2$, containing the x_s and y_s velocity components at each node. By tracking the continuous position of the tracer particle (in accordance with the coordinate system visible in Figure 2), at each point in time, the closest nodes in the fluid domain are located and their velocity values are interpolated to determine the velocity influencing the tracer particle at that instant. Refer to algorithm 1 for further clarification on the propagation of the tracer particles over time. Algorithm 1 elaborates on how each particle is tracked, where px_t and py_t indicate the position of the particle at time t in the fluid channel coordinate system (x_s, y_s). V_x and V_y are the corresponding velocities of the particles calculated in the same coordinate system. Once a particle exits the fluid domain, it is no longer tracked, this is indicated by the condition in line 1 of Algorithm 1.

Algorithm 1 Simulate & Track Tracer Particles in a Fluid

1: **while** $(px_t, py_t) < (L_x, L_y)$ **do**
2: Identify the indices (i_x, i_y) of the four nodes that form the cell enclosing the particle
3: Calculate the bi-linearly interpolated velocity at the particle's position (V_x, V_y):
Let V_{11}, V_{21}, V_{12}, and V_{22} be the velocities at the four enclosing nodes
Let (x_1, y_1) be the coordinates of the bottom-left node, and (x_2, y_2) the coordinates of the top-right node
4: Compute the interpolation weights:
$w_{x1} = \frac{x_2 - px_t}{x_2 - x_1}$, $w_{x2} = \frac{px_t - x_1}{x_2 - x_1}$, $w_{y1} = \frac{y_2 - py_t}{y_2 - y_1}$, $w_{y2} = \frac{py_t - y_1}{y_2 - y_1}$,
5: Perform bi-linear interpolation to find V_x and V_y:
$V_x = w_{x1} \cdot (w_{y1} \cdot V_{11_x} + w_{y2} \cdot V_{12_x}) + w_{x2} \cdot (w_{y1} \cdot V_{21_x} + w_{y2} \cdot V_{22_x})$
$V_y = w_{x1} \cdot (w_{y1} \cdot V_{11_y} + w_{y2} \cdot V_{12_y}) + w_{x2} \cdot (w_{y1} \cdot V_{21_y} + w_{y2} \cdot V_{22_y})$
6: Update the particle's position using the interpolated velocity:
$px_{t+1} = px_t + V_x \cdot dt$
$py_{t+1} = py_t + V_y \cdot dt$
7: **return** (px_{t+1}, py_{t+1})
8: Set $px_t = px_{t+1}$ and $py_t = py_{t+1}$ for the next iteration
9: **end while**

4.5.4 Control Output: the Metric

A continuous metric is constructed that quantifies the ratio of particles crossing the measurement point above the split channel to those crossing the measurement point below the split channel. Monitoring fluctuations in this ratio essentially monitors the mixing rate within the fluid channel. Actuation in the soft robotic tentacle provides the possibility of controlling the concentration and by extension controlling the path of particles entering the fluid channel.

In discreet form, the metric can be defined as follows:

$$r(t) = \frac{\text{number of particles passing lower boundary}}{\text{total number of particles passing measurement point}} \quad (2)$$

However for control applications, and more so when considering the metric as a feature for neural networks, continuous signals are preferred. This is primarily to ensure smooth derivatives when learning relationships between features and/or the signal over time.

Using the discreet form of the metric, Equation 2, would result in abrupt step changes in the metric value even with just one additional particle crossing the measurement point. The metric is inversely proportional to the total number of tracer particles passing the measurement point. This can be seen in the top graph in Figure 28.

It was taken into account that there won't be particles crossing the measurement point in every time step. This needs to be considered because the simulated tracer particles are released in a single line from a fixed position. This means that for the metric to be continuous, a measurement horizon needs to be defined, T. For the simulation run in this research, $T = 1.0s$. The metric works as a sliding window moving forward in time, constructing its value based on the number of particles crossing the measurement point in the defined time window relative to that instant in time. The continuous metric constructs its value at each time step as follows, only considering particles acting within the measurement horizon T:

$$r(t) = \frac{\sum w_l}{\sum w_l + \sum w_u} \quad (3)$$

Algorithm 2 Calculate Weight for Tracer Particles

Input: δt, the time since the particle passed the measurement point.
Input: T, the defined measurement horizon.
Input: τ, the threshold time constant.
Output: Weighting value for tracer particle.
if $\delta t < \tau$ **then**
    return $0.5 \cdot (1 - \cos(\frac{\pi}{\tau} \cdot \delta t))$
else if $\delta t > T - \tau$ **then**
    return $0.5 \cdot (1 + \cos(\frac{\pi}{\tau} \cdot (\delta t - (T - \tau))))$
else
    return 1
end if

Where w_l and w_u are the weighted particle values. Each particle is weighted according to what point in the measurement horizon, τ, the particle crossed the measurement point. Particles that cross the measurement point at the start and end of the measurement horizon are weighted less to ensure continuity in the metric, while particles passing the boundary in the middle of the measurement horizon are weighted 1. This results in a smooth continuous distribution of the changes in particle concentration over time; the result of a weighting function is applied to the discreet metric. This can be seen in Figure 28. The weight assigned to each particle is determined by δt, the time since the particle passed the measurement point, T, the defined measurement horizon τ, the threshold time constant. The algorithm employed can be seen in Algorithm 2.

4.6 Control Parameters

The control parameters of the problem will be the features used to train the neural network. The control inputs describe the actuation applied to the soft robotic tentacle. The scalar metric designed to quantify the mixing of the fluid in the x_s direction becomes the control output The table below summarizes the control parameters of the developed complex benchmark fluid case:

	Simulation Parameter	Symbol
Control Input	Soft robotic tentacle actuation polynomials : $q_1(t), q_2(t)$	$u_1(t), u_2(t)$
Control Output	Custom metric to monitor fluid mixing behavior : $r(t)$	$y_r(t)$

Table 1 Summary of control parameters to be considered for forecasting simulation behavior

5 DATASETS

Two datasets, with independent experiments to act as test data, were constructed from the complex benchmark case (see Section 4). The datasets were created to test the forecasting algorithm's capacity to understand regular, periodic motion and to what extent this performance could extend to complex, less predictable behaviors. evaluating the developed frameworks on these two datasets, the neural network is exposed to a wide range of data characteristics. This will help to identify the limitations of the models.

5.1 Periodic Motion Dataset

Periodic motion is a fundamental class of behavior in dynamic systems; despite their repetitive motion, in the context of the fluid dynamic problem, due to the complexity in the mode of actuation and the system being controlled, even periodic signals produce complex control output signals. The periodic dataset will serve as a baseline to test the network's capability to capture and predict regular and predictable behavior in the complex dynamic system.

Figure 6 displays the control output signals from the periodic motion training dataset. The corresponding control input signals can be seen in Table 2.

	$u_1(t)$	$u_2(t)$
2.10	$\sin(t)$	$\sin(t)$
2.11	$2\sin(t)$	$2\sin(t)$
2.12	$0.5\sin(t)$	$0.5\sin(t)$
2.13	$\sin(2t)$	$\sin(2t)$
2.14	$\sin(4t)$	$\sin(4t)$
2.15	$\sin(5t)$	$\sin(5t)$
2.16	$\sin(6t)$	$\sin(6t)$
Test Data	$\sin(3t)$	$\sin(3t)$

Table 2 Control inputs for generating periodic motion dataset

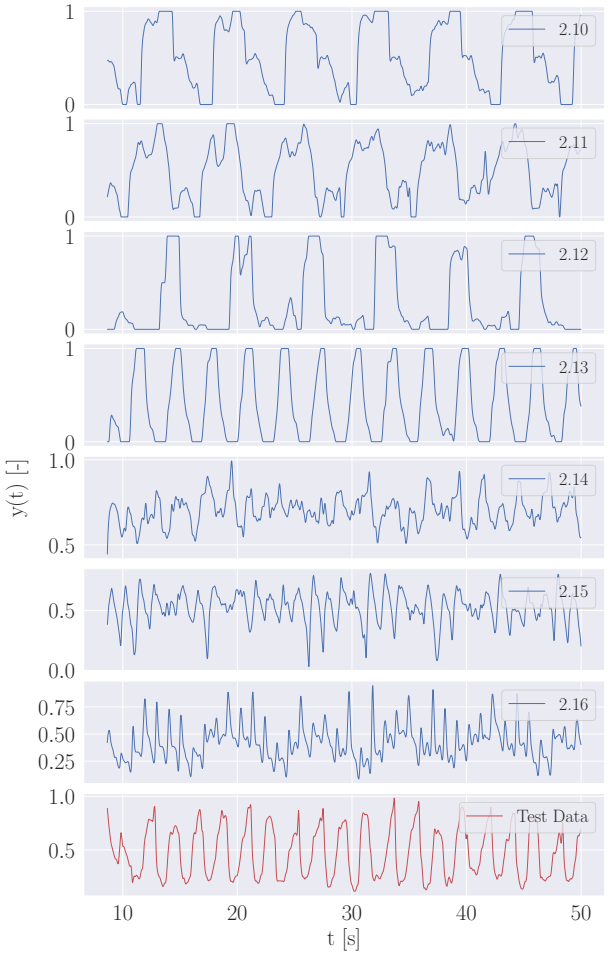


Figure 6 Periodic motion dataset: the control inputs $u_1(t)$ and $u_2(t)$ are limited to sinusoidal signals to produce a range of periodic motion in the tentacle. The training data is indicated in blue and the test data experiment is indicated in red.

5.2 Aperiodic and Asymmetric Motion Dataset

It was crucial to develop the aperiodic/asymmetric motion dataset as this will explicitly test the generalization ability of the developed algorithm and architecture. If the developed model can extend its performance to the more complex problems posed by aperiodic and asymmetric motion in the tentacle, it becomes possible to verify whether the model learns underlying system dynamics or simply memorizes data patterns. In addition to this, aperiodic and asymmetric data can represent cases where there are higher levels of noise or unexpected changes. This makes it possible to identify the edge cases of the network performance.

The control inputs used to generate the complex motion in the tentacle are listed in Table 3. These correspond to the output signals displayed in Figure 7. Experiment 2.21, in Table 3, uses a smoothing function S to ensure that the input signal remains continuous, Equation 4.

$$S(t, start, end, width) = \frac{\tanh(\frac{t-start}{width}) - \tanh(\frac{t-end}{width})}{2} \tag{4}$$

2.20	$u_1(t) = \sin(4t)$ $u_2(t) = \sin(2\pi(0.1t^{1.5}))$
2.21	$u_1(t) = \sin(t)S(t,0,10,0.1)+$ $\sin(3t)S(t,2,8,0.1)+$ $\sin(5t)S(t,4,6,0.1)$ $u_2(t) = \sin(2\pi(0.1t^{1.5}))$
2.22	$u_1(t) = \sin(t\pi(0.1+0.05\sin(0.1t)))e^{-0.1t}$ $u_2(t) = \sin(\pi t(0.2+0.1\sin(0.05t)))e^{-0.05t^2}$
2.23	$u_1(t) = \sin(2\pi t + \sin(0.5\sqrt{(t)})^2$ $u_2(t) = \sin(2\pi 0.5t)^{-0.05t} + \sin(2\pi 0.05t^2)e^{-0.05t}$
2.24	$u_1(t) = \sin(t\pi(0.15+0.1\sin(0.05t)))e^{-0.2t}$ $u_2(t) = \sin(2\pi(0.2t+0.02t\sin(0.02t)))e^{-0.2t}$
2.25	$u_1(t) = \sin(2\pi 0.2t + 2\cos(0.2t))e^{-0.05t}$ $u_2(t) = \sin(2\pi t(0.1+0.01t))^{-0.05t}$
2.26	$u_1(t) = \sin(2\pi(0.1+0.02t)t)e^{-0.1t}$ $u_2(t) = \sin(2\pi 0.2t) + \sin(2\pi 0.1t + \frac{\pi}{4}))e^{-0.1t}$
Test Data	$u_1(t) = \sin(2\pi(0.1t^{1.2}))e^{-0.1t}$ $u_2(t) = \sin(2\pi 0.1t) + \sin(2\pi 0.02t^2))e^{-0.02t}$

Table 3 Control inputs for generating aperiodic motion dataset

6 Methodology

An offline and online framework for a feed-forward neural network to forecast the behavior of the control output (the developed metric, subsection 4.5.4) is designed. To find the most suitable network for the task, performance error was evaluated when increasing network complexity (systematically increasing the width and depth of the network in the range 3-20), different loss functions and different activation functions (see Section 7). The tuned architecture is then embedded into an offline and online framework, to ultimately compare the performance gain/loss. The goal of the network (embedded in the chosen framework) is to replace the dynamic model of the complex physical system designed. The

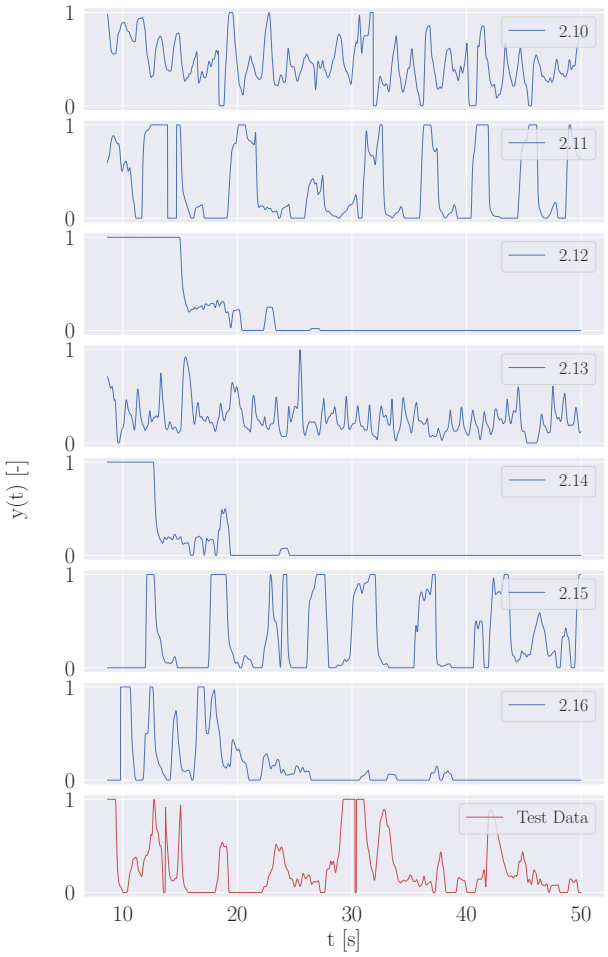


Figure 7 Aperiodic dataset: indicated in blue is the training data, composed of seven experiments. The test data is indicated in red and is a separate experiment.

algorithm must forecast the system behavior for H timesteps into the future.

This network aims to test whether N past input-output data pairs of a complex non-linear system is sufficient information to forecast the system behavior H time-steps into the future. To demonstrate the feasibility of this concept we consider the simulation and parameters described in Section 4 and Table 1. Both the online and offline frameworks take past control input-output pairs of the complex system, and use this data to forecast the system behavior in each timestep. The stark difference between the two approaches is that the offline framework trains the embedded neural network offline on approximately 30,000 data points for each case (periodic and aperiodic), whereas the online learning framework immediately deploys the network on the test data with no pre-training on other datasets. The network updates its weights as it forecasts during deployement.

This approach builds on the assumption that the system can be described as follows:

$$y_r(t+1) = f(y_r(t), y_r(t-1), \ldots, y_r(t-N), \mathbf{u}(t), \mathbf{u}(t-1), \ldots, \mathbf{u}(t-N))$$

where

$$\mathbf{u}(t) = \begin{pmatrix} u_1(t) \\ u_2(t) \end{pmatrix}$$

Where $y_r(t)$ is the control output of the designed system and $\mathbf{u}(t)$ is the control input vector that actuated the soft robotic tentacle, see Table 1.

This means assuming that future system behavior will be a function of the past control inputs and outputs [19]. The goal of the neural network is to become an approximator of this function. By using a one-step approximator recursively, it becomes possible to forecast system behavior H time-steps ahead. The frameworks use deep learning networks to approximate this function, and will be noted by $\hat{f}$.

The look-back horizon N is determined by the delay between a control input being applied and a corresponding change becoming visible in the control output. In the simulation case designed, there is approximately a $4s$ delay between actuation in the soft robotic tentacle and a control output reaction. It is also observed that in free flow, with no actuation provided to the tentacle, it takes $5.642s$ for a particle to reach the measurement point from the moment it makes contact with the leading edge of the soft robotic tentacle.

The fluid simulation uses $dt = 0.00115547s$; this is sufficiently small enough to ensure convergence in the FSI solver even with dynamic actuation in the soft robotic tentacle. A data filter is implemented to increase learning speed. The data filter allows you to specify a 'skip' parameter. This parameter was set to 15 for the experiments that follow, this means that the network sees every 15th data-point from the simulation. dt_{fs} is the dt after the filtering by the algorithm i.e 'skip' factor. In addition to reducing the amount of computation, this also helps the network learn relatively more global trends without losing too much information (the small dt allows for this). It also means that fewer data points are needed to capture the same history horizon. With a 'skip' factor defined, a minimum of 200 time-steps of past input-output pairs need to be fed to the network in order to capture the aforementioned 4s delay in the action-reaction of this fluid dynamic system.

6.1 Objectives

The goal of the offline and online learning frameworks are as follows:

1. Forecast system behavior at least 1 second into the future
2. Develop forecasts in a completely data-driven fashion, with the neural network inputs being a function of the control parameters seen in Table 1.

6.2 *Offline Learning Framework*

The offline learning framework is composed of two components: training the one-step predictor, and deployment. The one-step predictor is designed to at time t, predict $y_r(t+1)$ using the past N control input-output pairs. During deployment, this one-step predictor is used recursively, so that its predictions can be used to forecast the system output behavior H timesteps.

6.2.1 Training: the One-Step Predictor

The one-step predictor is a feed-forward neural network that aims to approximate the function that models the 'unknown' non-linear dynamic system, $\hat{f}$:

$$\begin{aligned}\hat{y}_r(t+1) = \hat{f}(&y_r(t), y_r(t-1), \ldots, y_r(t-N),\\ &u_1(t), u_1(t-1), \ldots, u_1(t-N)),\\ &u_2(t), u_2(t-1), \ldots, u_2(t-N)),\end{aligned}$$

The exploration to find the optimum architecture can be seen in Section 8. The architecture of the embedded neural network can be seen in Table 4.

The training regime uses early stopping [24], with the 'patience' parameter set to 20; this means that when the validation loss doesn't improve for 20 consecutive epochs, the training is halted as past this point the model will risk overfitting.

Each training sample is composed of $X(t)$, the input sample, and $Y(t)$ the corresponding label. In this case, each training sample is composed of the control inputs and corresponding system outputs for the past $N = 200$ timesteps, Equation 5, and the label would be the true system output at the next timestep $t = t + 1$, Equation 6.

$$\mathbf{X}(t) = \begin{bmatrix} u_1(t), u_1(t-1), ...u_1(t-N) \\ u_2(t), u_2(t-1), ...u_2(t-N) \\ y_r(t), y_r(t-1), ...y_r(t-N) \end{bmatrix} \tag{5}$$

$$Y(t) = y_r(t+1) \tag{6}$$

During training the network updates its weight such that the loss, Equation 7, is minimized. n_s is the total number of training samples available, and $\hat{y}(t+1)$ represents the output of the neural network. The loss is composed of the mean squared error in prediction with an additional penalty term η which penalizes the network for predictions of values outside the range training data. I.e as $y_r(t)$ will always assume a value between $[0, 1]$, the network will get an additional penalty when it predicts values outside this range. The first term in Equation 8 penalizes predictions greater than 1, while the second term penalizes negative predictions. The effectiveness and motivation for the custom loss function can be seen in Section 7.3.

$$e_t(t) = \frac{1}{n_s}\sum_{n=0}^{n_s}(y_r(t+1) - \hat{y}_r(t+1))^2 + \eta \tag{7}$$

$$\eta = \max(0, \hat{y}_r(t) - 1) + \max(0, -\hat{y}_r(t)) \tag{8}$$

6.2.2 Deployment: Forecasting Algorithm

The offline trained one-step predictor can then be used recursively to build a forecast horizon, predicting the system's control output's behavior for H timesteps ahead. At each time step, the control sequence $[(u_1(t+1), u_2(t+1) \ldots (u_1(t+N), u_2(t+N)]$ for the next H timesteps is assumed are available. In practice, a controller would be querying these potential control sequences before choosing the optimal control based on the performance implied by the forecasts. The complete algorithm for one timestep can be seen below,

Algorithm 3 Recursive Forecasting Algorithm

$current\_input \leftarrow \mathbf{X}(t)$
$forecast \leftarrow [\,]$
for $i = [0, H]$ **do**
  $\hat{y}_r(t+i) \leftarrow \hat{f}(current\_input)$
  $forecast.append(\hat{y}_r(t+i+1))$
    ▷ remove first entry in $current\_input$
  remove $\begin{bmatrix} u_1(t-N) \\ u_2(t-N) \\ y_r(t-N) \end{bmatrix}$
    ▷ Append next control input and network prediction to the $current\_input$
  append $\begin{bmatrix} u_1(t+i) \\ u_2(t+i) \\ \hat{y}_r(t+i) \end{bmatrix}$
end for

6.3 *Online Learning Framework*

The online learning framework is designed to employ the same feed-forward neural network architecture of the same complexity as that in the offline learning framework. The online learning framework does not have a training phase before deployment. It is not shown the training data indicated in Figures 6 and 3 as the offline learning network. Instead, the online framework has a buffer incorporated that becomes a form of memory handling. While the online learning framework employs the same architecture and forecasting algorithm outlined in Section 6.2.2, the difference is that the network's weights are updated during deployment as opposed to during a separate offline training phase.

The buffer, B, has size $N_b = 500$ with dimension dimension $(N_b, 3)$, and is initialized to zeros. It is composed

of the past N_b control system input and corresponding outputs, see Equation (9); once initialized, at the end of every timestep, the control output and corresponding control inputs of the system are appended to the buffer, and the oldest entry in the buffer is deleted. The online learning framework will not calculate system forecasts until it has been deployed long enough that the buffer is full, i.e N_b timesteps need to pass before forecasts are produced, this would happen when $t = N_b dt_{fs}$ where dt_{fs} is the filtered dt. This timestamp is denoted by t_{BA}, the time of buffer activation.

$$\mathbf{B}(t) = \begin{bmatrix} u_1(t), u_1(t-1), ...u_1(t-N_b) \\ u_2(t), u_2(t-1), ...u_2(t-N_b) \\ y(t), y(t-1), ...y(t-N_b) \end{bmatrix} \tag{9}$$

Algorithm 4 Deployment : Online Learning Framework

1: $\mathbf{B}(t) \leftarrow \text{zeros}(N_b, 3)$
2: **if** $t \leq t_b$ **then**:
3: $\quad \mathbf{B}(t) \leftarrow [u_1(t+1), u_2(t+1), y_r(t+1)]$
4: **else if** $t = t_b + dt_{fs}$ **then**
5: $\quad$ Create $\mathbf{X}(t, \cdots t - N_b), \mathbf{Y}(t, \cdots t - N_b)$
6: $\quad$ Train one-step predictor for 100 epochs on batch 1
7: $\quad$ Update and save network weights
8: $\quad$ Recursively build H step forecast (see Algorithm 3)
9: **else if** $t > t_{BA} + dt_{fs}$ **then**
10: $\quad$ Create $\mathbf{X}(t \cdots t - N_b), \mathbf{Y}(t \cdots t - N_b)$
11: $\quad$ Train one-step predictor on new batch
12: $\quad$ Update and save weights
13: $\quad$ Recursively build H step forecast (see Algorithm 3)
14: **end if**
15: Remove oldest entry in the buffer ▷ Actual system takes a step forward
16: $\mathbf{B}(t) \leftarrow [u_1(t+1), u_2(t+1), y_r(t+1)]$

$\mathbf{X(t)}$ and $Y(t)$ remain the same as described for the one-step predictor in Section 6.2.1. The online learning algorithm is outlined in Algorithm 4; this algorithm summarized the behavior of the online learning framework in each time-step, t, that the simulation is active. The following points can be clarified:

1. While $t \leq t_{BA}$, the true system outputs are added to the buffer until it is full (lines 2-1).

2. When $t \geq t_b$, a sliding window is used on the buffer of size N_b to create training samples, with each training sample $X(t)$ sample being a $(N, 3)$ vector of the past N control inputs and outputs, see Equation 5. The corresponding label, $Y(t)$, would be the $(t+1)$th control output, see Equation 6 (line 5-6, 10-11 in Algorithm 4). When the buffer is full, $n_s = (N_b - 2N - H)$, training samples with corresponding labels can be made.

3. In the first time-step where the buffer is full ($t = t_{BA} + dt_{fs}$), the one-step predictor network is trained on the samples created from the buffer for 100 epochs.

4. In the time-steps to follow, $t > t_{BA} + dt_{fs}$, the buffer is used to create training samples, and the model is trained for 10 epochs per time-step on these new sampled. The weights are updated and saved (lines 9-12, Algorithm 4).

5. The network is then used recursively to build a forecast H steps ahead

6. At the end of each time-step, the control inputs and corresponding control output are added to the buffer and the oldest entry in the buffer is removed. In this way, the buffer size is maintained at N_b (lines 15-16, Algorithm 4). Note that the online weight update still employs the custom loss function (Equation 7)

6.4 Evaluation Metrics

To evaluate the frameworks' performance the following metrics will be prioritized, in addition to the one-step prediction error, Equation 7; Equation 11 is the forecast error. This is the mean squared error for a given H step forecast built with Algorithm 3. This helps forecast quality at each time-step. Certain behavior in data is harder for the network to learn, such as unexpected turn-points and step responses; this metric will make it possible to easily observe this.

$$e(t) = y(t+1) - \hat{y}(t+1) \tag{10}$$

$$FMSE = \frac{\sum_{n=1}^{H} e(t)^2}{N} \tag{11}$$

Equation 12 is the average forecasting mean squared error; this metric quantifies the average forecasting performance of the network in a given experiment of duration t_d.

$$AFMSE = \sum_{n=0}^{t_d} \frac{\sum_{n=1}^{H} e(t)^2}{N} \tag{12}$$

7 Network Architecture & Hyperparameter Tuning

The achirecture of the feedforward neural network used when testing the online and offline methodology (6.2.2) was tuned by evaluating the influence of different baseline settings and architecture complexities on validation data derived from the periodic and aperiodic datasets (see Figure 6 and Figure 7).

7.1 Base Settings

The base setting of the network includes the activation function, the learning rate, and the use of early stopping. To evaluate the most suitable setting for the complex fluid simulation case, the forecasting error on the validation data was

evaluated. The different settings were tested on a network with 1 layer of 16 neurons each so that the performance with different activation functions and base settings.

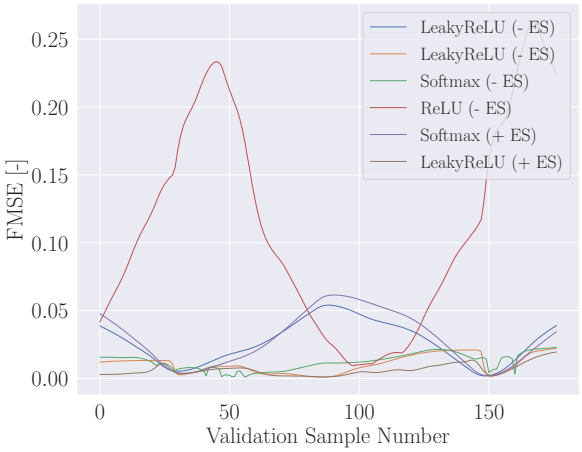


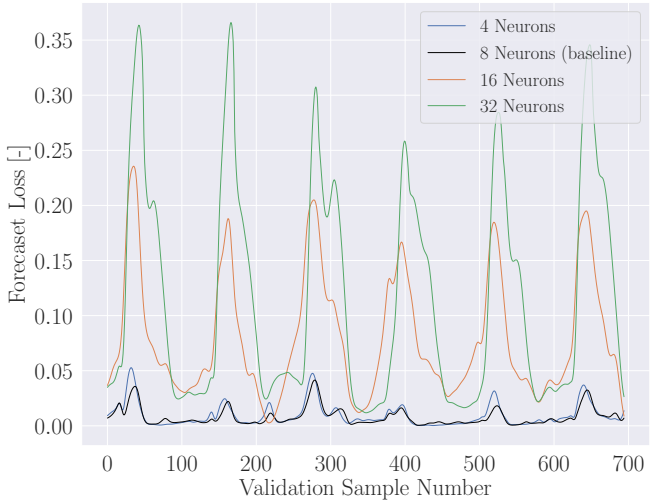
Figure 8 Baseline model performance with different activation functions and with and without early stopping applied. $(+ES)$ indicates early stopping is applied, and $(-ES)$ is where it is not applied. Best performance is achieved with LeakyReLU couples with early stopping.

This 'control' baseline network was evaluated with the different settings employed; the two activation functions that had the least average forecasting error were Softmax and LeakyReLU. Figure 8 shows the results of these activation functions when coupled with early stopping and without it, with a setting of 500 epochs of training. As is visible from the plot, the best performance is achieved when using the LeakyReLU activation function with early stopping applied.

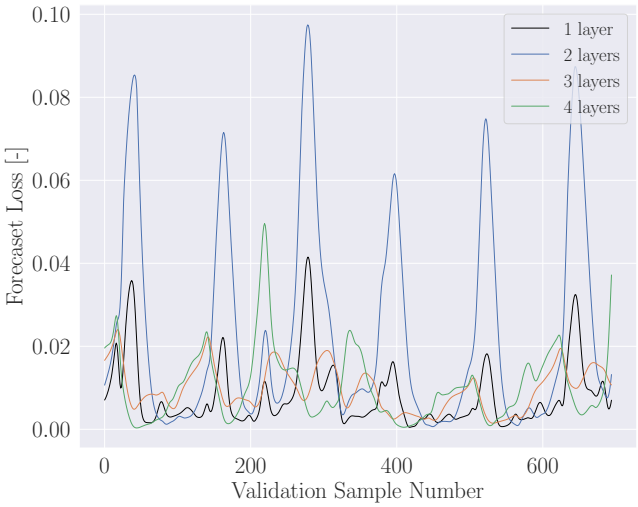
7.2 Impact of Model Complexity

To analyze the impact of network complexity on this task, the network architecture was tested in two stages: manual and automated tuning. The manual tuning was used to determine the approximate range of network depth and width to explore when using the more complex automated tuning.

Figure **??** shows the results of the manual tuning; the model performance was first tested by increasing the width per layer of the neural network. The same architecture used to tune the activation functions were used as a starting point. As can be seen from Figure 9a, an average width of 8 neurons displayed the best performance. This becomes the starting point for testing the impact of network depth on validation performance. Keeping a layer width of 8 neurons, the depth was increased i.e the number of layers were increased. As can be seen in Figure 9b, a depth of 3 layers provided the best performance. These were used as a ball mark estimate to then use Optuna to tune the learning rate, per layer network width and depth. The manual tuning was used to estimate the



(a) Manually testing the imapact of increasing the network width (i.e neurons per layer) on the network performance on the validation samples. The best performing width was at 8 neurons and is indicated by the black line in the graph. This becomes the width with which the network depth will be varied.



(b) The best result from the manual width analysis is used as the average width to vary the depth of the network and evaluate performance on the validation samples.

Figure 9 Summary of the manual evaluation of the impact of varying network complexity in terms of average number of neurons per layer and depth of the network. Plot *a* displays the impact of increasing the network width and plot *(b)* the impact on increasing network depth.

ranges within which to iterate the network width and depth combinations: the network depth was varied from 3 to 9 and the per layer width was varied in the range of 7-15. The best performing network architecture and base settings are summarized in the table below:

Network Architecture and Training Parameters	
Input shape	(N,3)
Output shape	(1,)
Early stopping	20
Optimizer	ADAM
Learning rate	0.001
Network depth	4
Width per layer	[12, 12, 9, 8]
Activation Function	Leaky ReLU
Offline learning epochs	500
Online learning epochs	100 warm start, 10 online updates

Table 4 Overview of one-step predictor network architecture, training parameters and base settings

7.3 Custom Loss Function

Section 6.2.1 proposed a custom loss function with an additional penalty added for predictions outside the range of the training data. I.e all the input data and label are in the range [0,1] (for the tentacle fluid simulation case), therefore predictions outside this range should be discouraged. Due to the nature of feed-forward neural networks, extrapolation tasks are especially difficult when given non-linear data; without the custom loss function applied, there was a significant amount of overshooting in the forecasts generated for the validation data.

This can be seen in Figure 10, displaying the absolute forecast error per validation sample from models trained with the proposed custom loss samples, versus a model trained with the default mean squared error loss. There is a significant reduction in forecast error when trained with the additional penalty term. The final architecture embedded in the the online and offline learning frameworks will use the custom loss function.

8 Verification & Benchmarking

In order to verify the network architecture's and the developed forecasting algorithm (see Algorithm 3) capability to capture system dynamics, the proposed recursive forecasting algorithm is tested on benchmark control systems where the forecasts can be compared to results from dynamic models. The benchmark cases increase in complexity, from linear time-invariant systems to seconds order systems with multiple control inputs. The cases considered are detailed in sections 8.1 to 8.3, and a summary of the results can be seen in

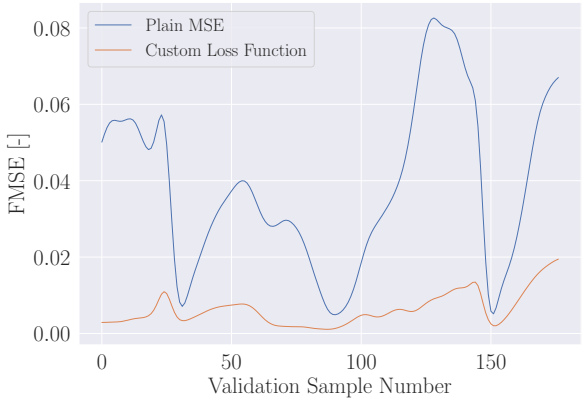


Figure 10 Performance of baseline architecture in the proposed algorithm with and without the custom loss function (see Equation 7) used for training. Absolute forecast error per validation data sample is displayed. The custom loss function reduces error significantly.

Table 5. Plots and further details of the cases considered can be seen in Appendix B.

	AFMSE [-]	H (s)	N (s)
V1	2.3868e-06	10	20
V2	1.1978e-05	10	20
V3.1	3.1487e-05	5	20
V3.2	4.4950e-05	10	40
V3.3	2.3343-04	20	80

Table 5 Forecasting performance on benchmark verification cases $V1 - V3.3$. The corresponding prediction horizon and history captured in the network input are also recorded.

8.1 V1 Linear Time Invariant Case: First Order Integrator

A linear time-invariant control system was considered. The system was a first-order integrator, and the performance of the proposed offline learning neural network architecture was tested.

The output of the system $y(t)$ is the integral of the control input $u(t)$. A constant value control input was applied to allow for manual verification of the network weight updates. For all t, a constant control input $u(t) = U$ is applied.

$$y(t) = \int u(t) = \int U dt = U \cdot t + C \text{where } C = 0 \text{ at } t = 0 \tag{13}$$

The proposed architecture is tested with a prediction horizon of 10 seconds, with network input capturing the past 20 seconds of data.

8.2 V2 First Order System: Thermal Control System

A simple thermal control system is considered: heating a room, with the control input being the heating power and the control output being the room temperature.

$$y(t) = Ku(t)(1 - e^{-t/\tau}) \tag{14}$$

Where the gain $K = 0.5$ and the time constant $\tau = 10$. The exponential term ensures the output response is gradual to mimic the room temperature response to a gradual increase in heating. The proposed architecture is tested with a prediction horizon of 10 seconds, with network input capturing the past 20 seconds of data.

8.3 V3 Second-Order System: Mass-Spring Damper System

The mass-spring-damper system is characterized by three primary components: mass (m), spring constant (k), and the damping coefficient (c). To test the developed algorithm, the second-order system is simulated with the following parameter values: $m = 1.0$, $K = 1.0$, $c = 0.1$. The system dynamics can be modeled by the second-order differential equation:

$$\ddot{x} + \frac{c}{m}\dot{x} + \frac{k}{m}x = \mathbf{u}(t) \tag{15}$$

Where x is the displacement of the mass; $\dot{x}$ and $\ddot{x}$ are the derivatives of the displacement, the velocity and acceleration of the mass and $u(t)$ is the control input to the system. The control inputs are time-varying external forces applied to the system and are composed of two signals:

$$\mathbf{u}(t) = \begin{bmatrix} u_1(t) = \sin(t) \\ u_2(t) = cos(t) \end{bmatrix} \tag{16}$$

In order to thoroughly evaluate the performance of the designed algorithm on the second order system, the performance for increasing prediction horizons was recorded: predicting a quarter period into the future (*V3.1*), a half period into the future (*V3.2*), and a full period into the future (*V3.3*). While the performance decreases as the prediction horizon is increased, it can be seen that even when the prediction horizon is a full period ahead (20 seconds of motion in this case), the proposed architecture can maintain performance with an average forecasting error in the order of 10^{-4}. The network input captures four times the prediction horizon (the 'skip' factor, 6 is used to capture more time in relatively fewer timesteps).

9 Results

The results of the performance of the neural network embedded in the online and offline framework are also compared to three developed relatively much less complex forecasting algorithms. These simple forecasting algorithms become heuristics to assess how much performance is gained by using a deep learning-based approach for this task. This will also illuminate to what extent the deep learning architecture are learning the relationship between the features compared to a numerical trend.

In addition to this, to assess the forecasting capability and by extension, the ability of the deep learning-based framework the following tests were performed:

- The performance of both online and offline learning frameworks are evaluated on periodic and aperiodic data
- The performance is evaluated at different prediction horizons (for both periodic and aperiodic datasets): $H = 0.25s, 0.5s, 1.0s$
- For each test (different value of H), the performance is compared to that of three heuristic forecasters (): a simple average predictor S1), a numerical linear extrapolator and a machine learning linear regressor
- For each time horizon, density plots are created to evaluate the ability of the deep learning architectures to comprehend global behavior of the system

9.1 Defining Heuristics

Three heuristics are developed that are numerical forecasters to discern the extent to which the deep learning architectures learn a relationship and to critically evaluate the performance gain from employing learning based methods for forecasting complex system behavior.

9.1.1 S1: Simple average predictor

The simple average predictor assumes that the next N control outputs in time can be estimated by the average of the past N control outputs. For $i = 1, 2, \cdots N$,

$$\hat{y}_r(t + i, \cdots t + N) = \frac{\sum y_r(t), \cdots y_r(t - N)}{N} \tag{17}$$

9.1.2 S2: Simple linear extrapolator

The simple linear extrapolator uses the last two control output values in the time-series data to calculate the gradient of a linear line and extrapolate it N steps into the future. For $i = 1, 2, \cdots, N$,

$$\hat{y}_r(t + i) = y_r(t) + \left(\frac{y_r(t) - y_r(t - 1)}{2}\right) y_r(t) \tag{18}$$

9.1.3 S3: Linear regressor

A linear regression model is fitted to the time series data (looking as far as the samples provided to the network). The model is then used to predict values from $i = 1, \cdots N$ that build the forecast. The linear regression model can be represented as :

$$\hat{y}_r(t+i) = \beta_0 + \beta_1 t \tag{19}$$

Where β_0 and β_1 are the coefficients estimated by the linear regression.

9.2 Points of Evaluation

The types of plots considered to aid the evaluation of the network performance include forecast density plots and plots comparing the FMSE (see Equation 11) of the heuristics. Density plots overlay all forecasts from a test experiment; if the network successfully learns a relationship, overlaying all consecutive forecasts would reconstruct the ground truth signal. The density plot allows us to evaluate accuracy as well as gain an understanding of the stability in predictions. Oscillating predictions are not desirable, as this means the network has little certainty in the learned relationship or that the training data was insufficient.

9.3 Periodic Motion

Table 6 displays a summary of the results when the network is trained and tested on the periodic motion dataset, with the AFMSE (see Equation 12) recorded for the heuristics and the neural model. The model with the best performance has its results bolded. Figures 12 to 14 display the comparison of the forecasting error per test sample of the developed heuristics against the developed neural model. Figure 11 displays the desnity plot when $H = 0.5s$; the neural model's predictions are overlayed with the forecasts of the best performing heuristic, in this case S2-the numerical linear extrapolator. Desity plots for $H = 0.25, 1, 0s$ can be seen in Appendix A.2.1.

	AFMSE [-]			
H (s)	S1	S2	S3	Offline Neural Model
0.25	0.00244	0.0027	0.0180	**0.0023**
0.50	0.01503	0.0122	0.0536	**0.0066**
1.00	0.0177	0.1069	0.0265	**0.0097**

Table 6 Summary of results on the periodic motion dataset. The best performance is bolded. The table records the AFMSE (see ref Equation 12).

9.4 Aperiodic Motion

Table 6 displays a summary of the results when the network is trained and tested on the aperiodic motion dataset, with the AFMSE (see Equation 12) recorded for the heuristics and the neural model. Figures 23a to 18 display the comparison of the forecasting error per test sample of the developed heuristics against the developed neural model. Figure 25b is the corresponding forecast density plots for $H = 0.5s$, with the most comparably performing heuristic's forecasts overlayed, which in this case was S2-the numerical linear extrapolator. Refer to Appendix A.1.2 and Appendix A.2.2 to see all the plots.

	AFMSE [-]			
H (s)	S1	S2	S3	Offline Neural Model
0.25	0.00473	0.01837	0.0038	**0.0012**
0.50	0.007557	0.09168	0.00977	**0.0068**
1.00	0.01060	0.28709	**0.01334**	0.0467

Table 7 Summary of results on the aperiodic motion dataset. The best performance is bolded. The table record the AFMSE (see ref Equation 12).

9.5 Online Learning vs Offline

Tables 8 and 9 summarize and allow for the comparison of the performance of the online and offline framework at the different values for H. Figure 22d plots the error for $H = 1.0s$ on the periodic test data in comparison to the performance of the offline learning framework for the same case and Figure 20 displays the corresponding desity plot. Figure 21 visualizes the error of the online learning and offline learning framework for the aperiodic data test case when $H = 1.0s$. Density plots for $H = 0.25$ and $H = 1.0s$ can be found in Appendix Appendix A.2.2. The density plot for aperiodic motion at $H = 1.0s$ can be seen in Appendix A.2.2.

	AFMSE [-]	
H (s)	Offline Learning	Online Learning
0.25	**0.0024**	0.0046
0.50	0.0066	**0.0042**
1.00	0.0097	**0.0052**

Table 8 Performance on periodic motion dataset

	AFMSE [-]	
H (s)	Offline Learning	Online Learning
0.25	**0.0012**	0.0089
0.50	**0.0068**	0.0100
1.00	0.0467	**0.0239**

Table 9 Performance on aperiodic motion dataset

10 EVALUATION

The results of the two proposed frameworks are evaluated with the its capacity to learn a relationship, generalize and the impact of the length of the prediction horizon as points of focus.

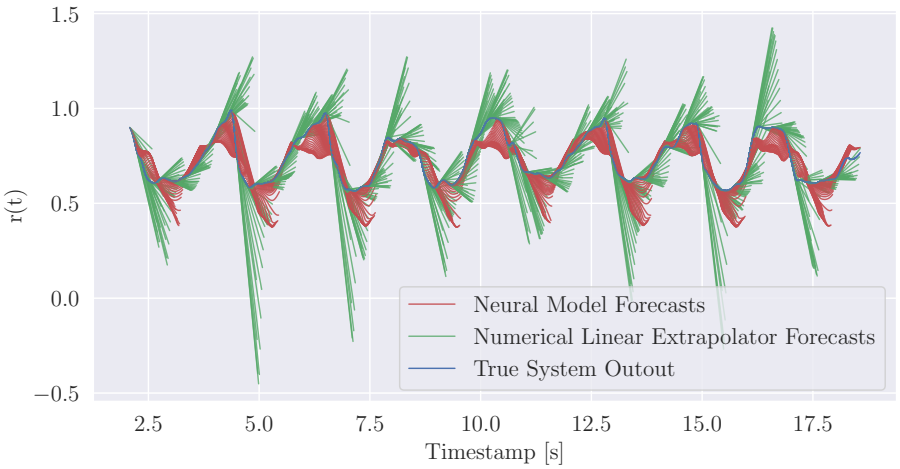


Figure 11 The density plot for qualitative evaluation of the offline learning neural model on the periodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.5s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

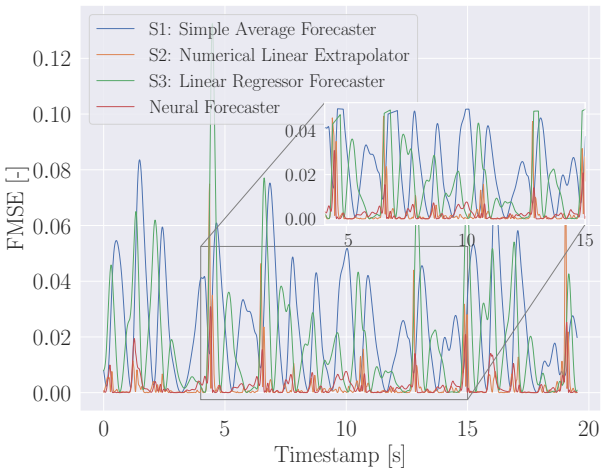


Figure 12 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 0.25s$. The neural model outperforms all heuristics by an order of magnitude.

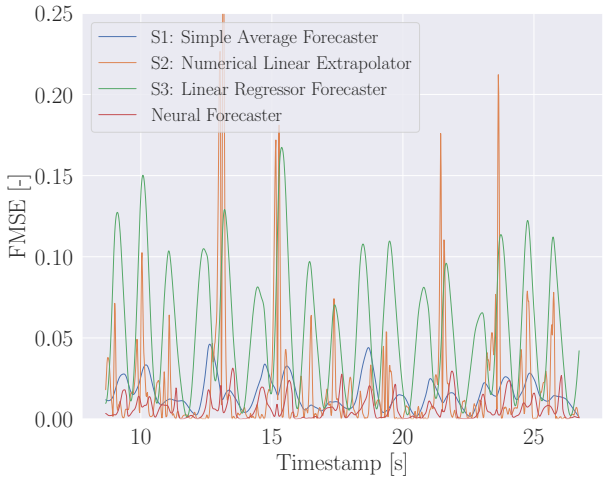


Figure 13 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 0.5s$. The neural model outperforms all heuristics by an order of magnitude.

10.1 Offline Learning Framework : Periodic Motion Dataset

Looking at Table 6, it can be seen that the neural model in the offline learning framework surpasses the heuristic models by an order of magnitude reduction in the average forecasting error for the periodic motion dataset.

The increased layering in the turn points of the signal in the density plots (Figure **??**, also indicates that these turn points are difficult for the network to gauge the location and magnitude off, however, the error at these points still does not exceed 0.02 FMSE, and this is a peak error. At prediction horizons set to 0.25s and 0.5s, Figures 12 and 13 shows the neural model having significantly better performance than the heuristics, having errors up to two orders of magnitude lower for certain test samples. The consistent improvement in performance when using the offline neural model, and its performance consistency across increasing prediction horizons, suggests that the offline neural model is indeed learning and

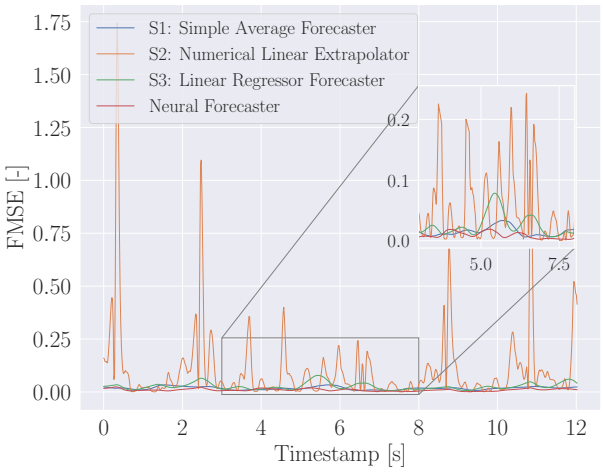


Figure 14 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 1.0s$. The neural model outperforms all heuristics by an order of magnitude.

gauged an understanding of the deeper, non-linear, relationship between the control input and output. There is a performance advantage when using the offline neural model for this forecasting task.

10.2 Offline Learning Framework: Aperiodic Motion Dataset

It can be seen that for the aperiodic motion, the offline learning neural model does not have as large a reduction in error as with the periodic motion data, and even in the case of a 1s prediction horizon, the local linear regressor model (S3) has better performance than the neural model. Looking at the density plot, Figure 25a, even with only a prediction horizon of 0.25s, it becomes clear that the network has difficulty estimating the irregular inflections. With a prediction horizon of 0.25s, despite these limitations, the neural model can be seen to still have notably less forecast error, Figure 23a. However, when the prediction horizon is increased to 0.5s, while the average forecasting error is still lower than the simple heuristic predictors, there is much more oscillation in the error, indicating significant uncertainty in successive forecasts. It struggles to reconstruct the ground truth signal.

It can be seen that the deterioration in performance with increasing prediction horizon is more significant in comparison to the periodic motion dataset. With a prediction horizon of 1s, the S3 simple heuristic forecaster even outperforms the model. With increasing prediction horizon, in the case of aperiodic motion, the network needs to capture more irregular inflections in a single forecast and the task becomes more difficult; more complexity is embedded in each forecast.

This disparity in performance of the offline learning neural model between periodic motion and aperiodic motion is largely potentially due to a lack of training data. Aproximately 300,000 training data points were used in both the periodic and aperiodic motion case. While this was sufficient to give the neural model competitive performance in the case of periodic motion data, the complexity of the aperiodic motion dataset demands for a larger training dataset to be able to draw conclusive results on the aperiodic test data.

While the periodic motion dataset deals with only sinusoidal control inputs, it can be seen in Table 3 that the aperiodic dataset had significantly more varied input signals. The complexity introduced was underestimated and not enough training data was generated to produce a comparable model for this data.

10.3 Relative Performance of the Online Learning Framework

On the periodic motion dataset, Table 8, at the shortest time horizon the offline learning framework outperforms the online learning framework. This is because at shorter time-horizons, the offline learning framework gains the advantage of having seen and been trained on a dataset of samples apriori. Online learning frameworks are designed to adapt and learn from new data incrementally. This is why in Figure 22d, the online learning framework initially has a large error that quickly drops to average less than the offline learning framework. For very short prediction horizons, the online learning model may not have adjusted adequately to the nuances of the data. In contrast, offline learning models are trained on a comprehensive dataset from the start, potentially giving them a better initial understanding of the data patterns. Furthermore, at smaller prediction horizons, the online model may be more sensitive to noise and data sparsity. Online learning continuously updates its parameters, which can be beneficial for capturing long-term trends but may lead to over-fitting or reacting too quickly to short-term noise in the data.

As the prediction horizon increases the online learning framework performs better. Figure 20 shows that the even at the largest time horizon, the online learning framework shows more stability in consecutive forecasts and a stronger correlation with the global trend of the signal than the offline learning model at relatively shorter time horizons. At $H = 1.0s$. the offline learning framework also displays a lag developed in it's forecast that can be seen in Appendix A.2.1, Figure 24c. Especially worth noting is that this lag is not apparent in the online learning framework's performance at the same time horizon.

While the performance of the online and offline learning frameworks remain within the same order of magnitude across the different time horizon's for the periodic motion dataset, the online learning framework displays less degradation in performance, with the FMSE at the largest time horizon only $\approx$13% worse than at the shortest, whereas the offline learning framework performs 3 times as worse at the largest

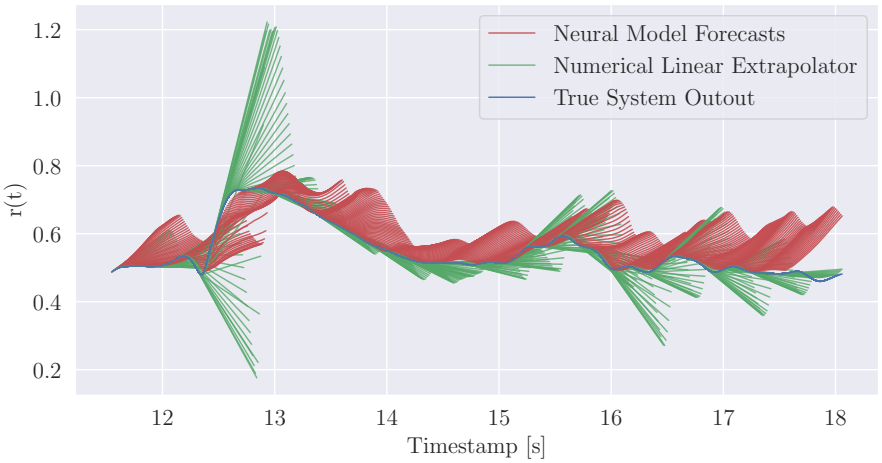


Figure 15 The density plot for qualitative evaluation of the offline learning neural model on the aperiodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.5s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

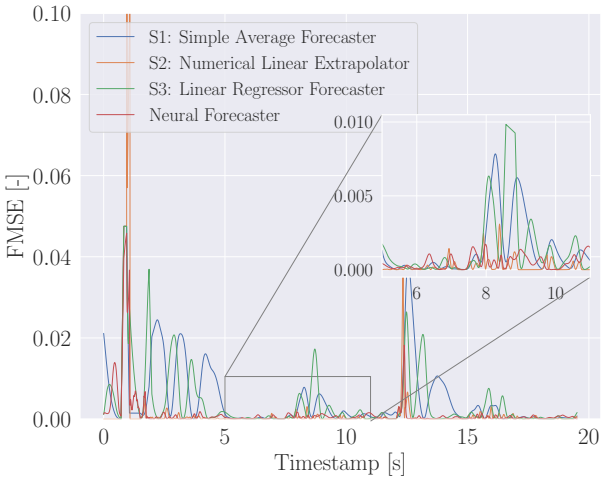


Figure 16 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 0.25s$. The neural model outperforms all heuristics by an order of magnitude.

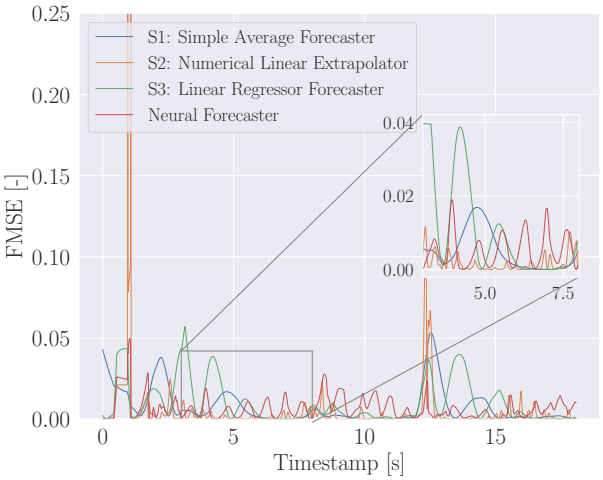


Figure 17 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 0.5s$. The average forecasting error of the neural model remains lower than that of the heuristics but oscillation is visible.

time horizon. This is clearly visible from Figure 22d

As can be seen from Table 9, on the aperiodic motion dataset, the online learning model only displays better performance at the largest time-horizon. Despite this, the density plots reveal that both frameworks struggle dealing with the irregularities of aperiodic data. While the online learning framework displays better performance at the large time horizon, it still severely oscillates around the true signal, struggling to capture the turn points and trends.

Aperiodic data lacks regular, repeating patterns, making it more challenging for both online and offline models to predict future values. The models may struggle to find a stable representation of the data that can generalize well to unseen instances. In addition to this, periodic data may contain more noise and outliers, which can mislead the learning process, especially in offline models that cannot adapt after their initial training phase. Online models can also be affected as they

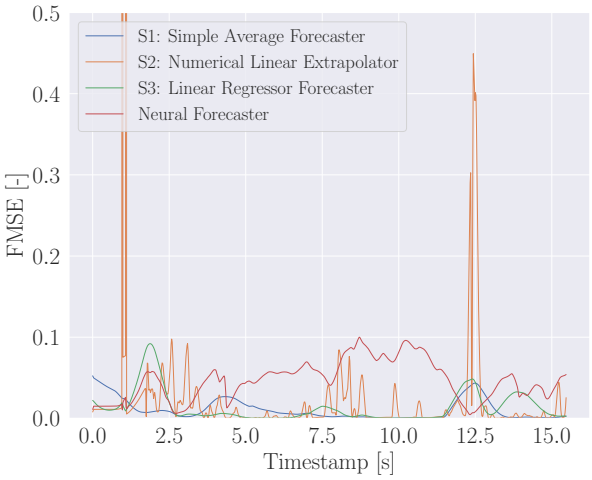


Figure 18 Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 1.0s$. Degradation in the performance of the neural model is visible with the larger prediction horizon, with S3 heuristic now outperforming the offline neural model.

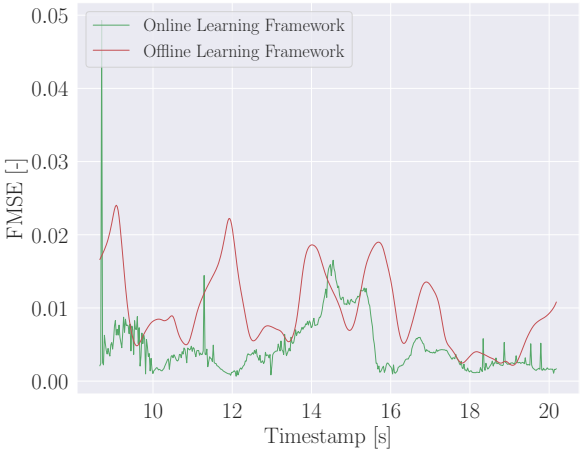


Figure 19 The error of the online learning framework and the offline learning framework for comparison on the periodic motion test set for $H = 1.0s$. At the largest time horizon, the average error of the online learning framework is seen to be lower.

might over-adjust to recent noise.

10.4 Limitations

- Performance on periodic motion dataset is not transferred to the aperiodic motion dataset. This was apparent in both online and offline learning frameworks which raises the issue that aperiodic motion might be an edge case for statistical A.I models like deep learning neural networks. The reason than these networks could capture the dynamics when tested on periodic motion data could be because periodic motion amplifies the system dyanmics.

- The results indicate that as the complexity of a scenario or system increases when dealing with offline models, a significantly larger amount of training data is required. However, it needs to be further evaluated whether increasing the training data would sufficiently address all of the model's shortcomings.

- The offline learning framework displayed significant performance degradation in both periodic and aperiodic motion datasets when the prediction horizon was increased, with a lag developing and the stability of the forecasts decreasing. While this could be because of the error accumulation native to the recursive prediction algorithm, it could also be a shortcoming of the offline learning approach in that it is not able to capture and forecast global trends in non-linear systems.

- The offline learning algorithm, being trained prior to testing offline, the model's ability to generalize is confined to the subspace it was trained in. Attempts to extrapolate beyond this trained subspace result in reduced performance, increased instability, and heightened uncertainty; these are the shortcomings in performance see when compared to the online learning framework's performance.

- The error metric FMSE[-] indicates the quality of the forecasts but should be further evaluated with correlation metrics. Because the metric is constructed by looking at the absolute error in each forecast; this metric could be aided and be more representative if the correlation between the network forecasts and the ground truth was also quantified. For example, the R^2 metric could be used - however the R^2 is designed to quantify the correlation in linear data, so would need to be developed further or other metrics explored to try and account for signal correlation in the non-linear data.

11 Conclusion

In this work, it was observed that both online and offline learning frameworks significantly outperformed traditional numerical heuristics when applied to datasets of periodic motion, maintaining a consistent magnitude of error across extended prediction horizons. Notably, the online learning framework demonstrated superior stability and lesser degradation in performance over longer prediction intervals. However, at the shortest prediction horizon, it was noted that the online algorithm's performance was not as robust as that

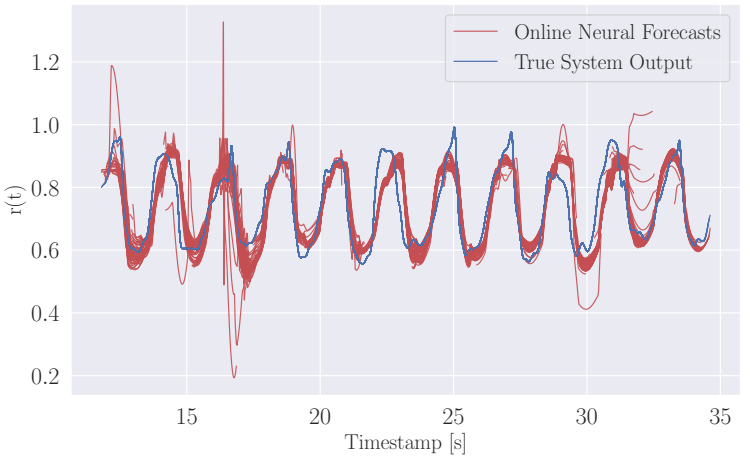


Figure 20 The density plot for qualitative evaluation of the online learning neural model on the periodic motion test set. Consecutive forecasts from the offline neural network model overlayed to create a density plot when $H = 1.0s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

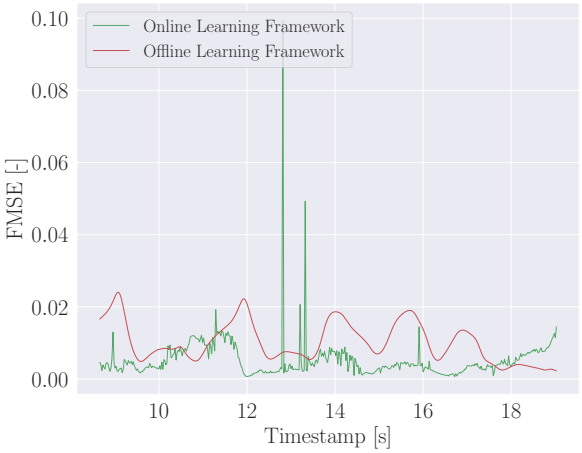


Figure 21 The error of the online learning framework and the offline learning framework for comparison on the aperiodic motion test set for $H = 1.0s$. At the largest time horizon, the average error of the online learning framework can be seen to be lower.

of the offline algorithm. This could be attributed to the online algorithm's requirement for a greater volume of data or more time within the system to fine-tune its parameters to the specifics of the dataset. As the prediction horizon decreases, the look-back horizon diminishes proportionally, potentially explaining the observed behavior. Despite these advancements, neither learning framework successfully translated their efficacy from periodic to aperiodic datasets, highlighting a significant limitation in using deep learning models for forecasting behaviors in complex systems. This limitation was evident as both frameworks struggled with stability in consecutive forecasts and accurately predicting irregular turning points in aperiodic data, despite the online model showing a relative performance advantage at the longest time horizon examined.

The findings suggest that hybrid models, which combine the strengths of both offline-trained models and the adaptability of online learning during deployment, hold promising potential, particularly in the context of active flow control. This approach could address the noted limitations by leveraging offline models' robust initial performance and enhancing it with online learning's dynamic adaptability to changing conditions, offering a comprehensive solution for forecasting in complex systems.

12 Further Work

- Enhanced Training for Offline Learning: Investigating the impact of augmenting the training dataset for the offline learning framework, particularly focusing on aperiodic motion datasets, is crucial. An increased volume of training data may improve the model's ability to capture and predict complex flow dynamics, offering insights into scalability and performance optimization in AFC.

- Online Learning with Complex Systems: Extending the application of the online learning algorithm to other complex nonlinear dynamical systems could provide valuable data on the algorithm's convergence rate. Testing across a broader spectrum of systems will help refine the algorithm's generality and applicabil-

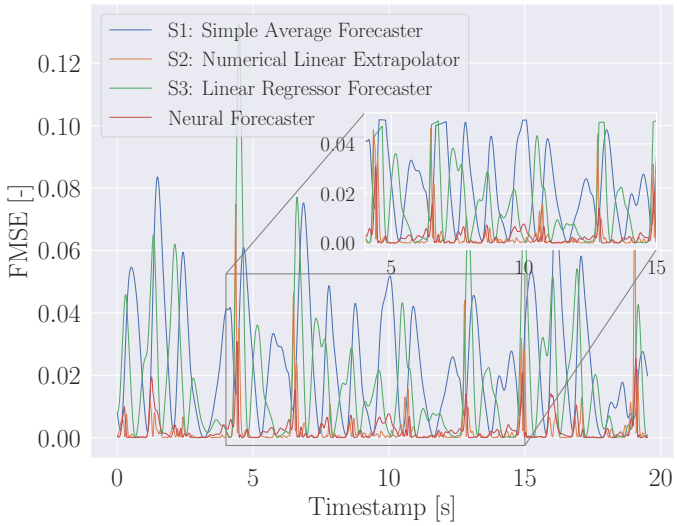
ity, paving the way for more adaptive and robust AFC solutions.

- Regularization in Online Learning: Implementing weight update regularization and establishing specific rules within the online learning algorithm are essential steps to prevent catastrophic forgetting. Such measures will ensure that the model retains previously learned information while adapting to new data, maintaining a delicate balance between stability and plasticity in learning.
- Feedback Control Loop Stability: Embedding the online learning algorithm within a feedback control loop to evaluate stability presents an exciting avenue for research. This will assess the algorithm's real-world applicability and reliability in dynamic control settings, where maintaining system stability is paramount.
- Hybrid Architectural Developments: The development of a hybrid architecture that integrates a pretrained neural model with the online learning framework during deployment offers a promising approach to enhance AFC adaptability. This strategy could leverage the strengths of both offline and online learning paradigms, ensuring rapid initial adaptation followed by continuous improvement and adjustment to new or evolving flow conditions.

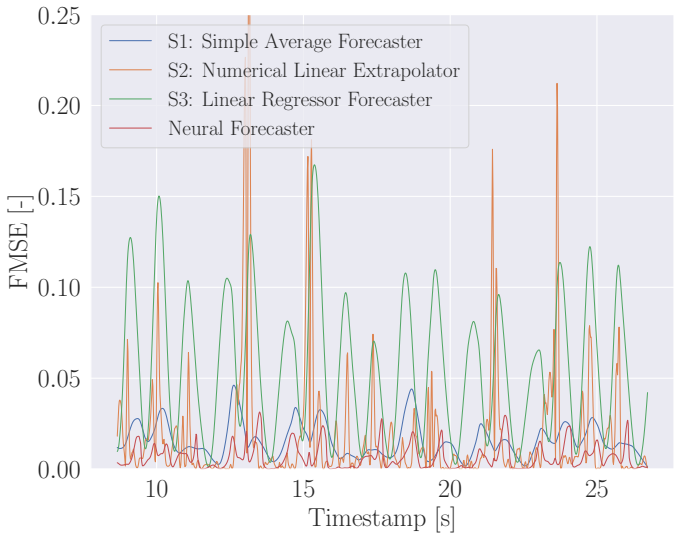
APPENDIX A ADDITIONAL VISUALS FOR RESULTS

Appendix A.1 Heuristic Error Comparison Plots

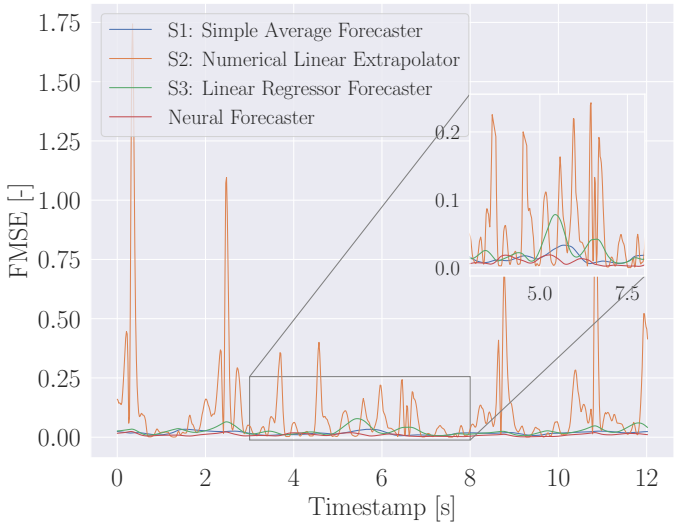
Appendix A.1.1 Periodic Motion



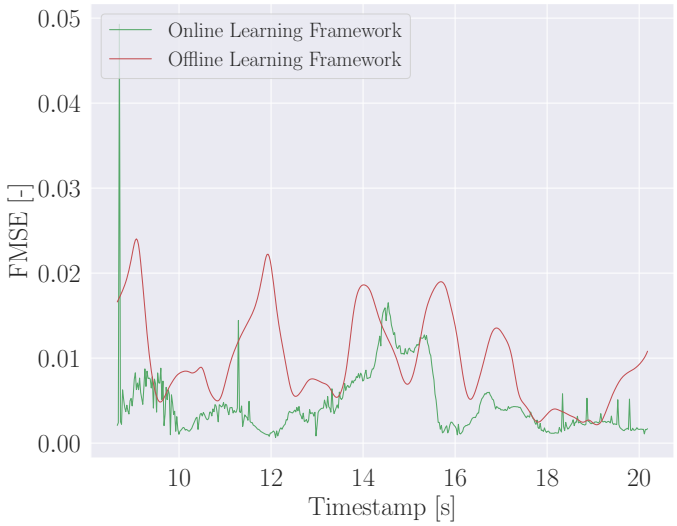
(a) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 0.25s$. The neural model outperforms all heuristics by an order of magnitude.



(b) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 0.5s$. The neural model outperforms all heuristics by an order of magnitude.

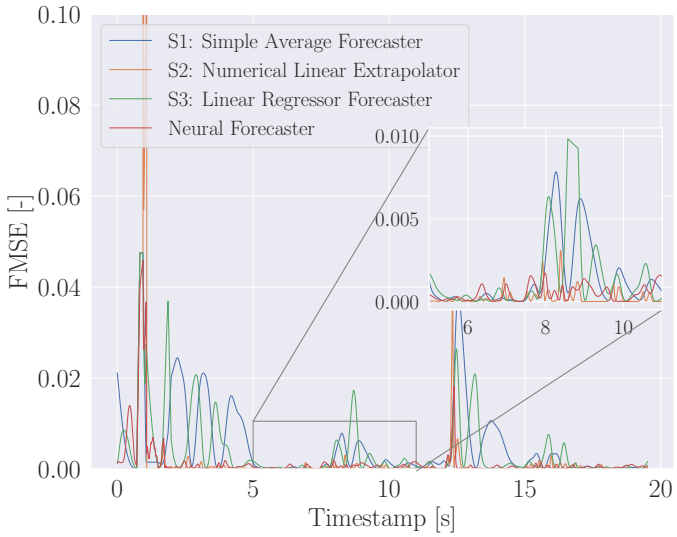


(c) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the periodic motion test case when $H = 1.0s$. The neural model outperforms all heuristics by an order of magnitude.

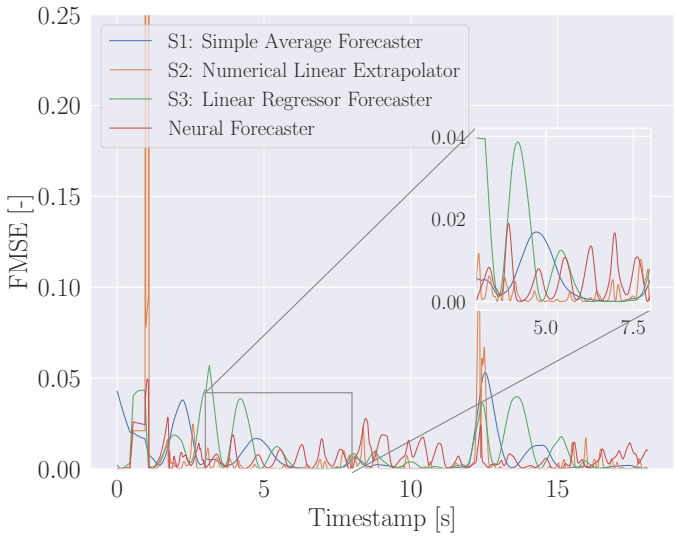


(d) The error of the online learning framework and the offline learning framework for comparison on the periodic motion test set for $H = 1.0s$. At the largest time horizon, the average error of the online learning framework is seen to be lower.

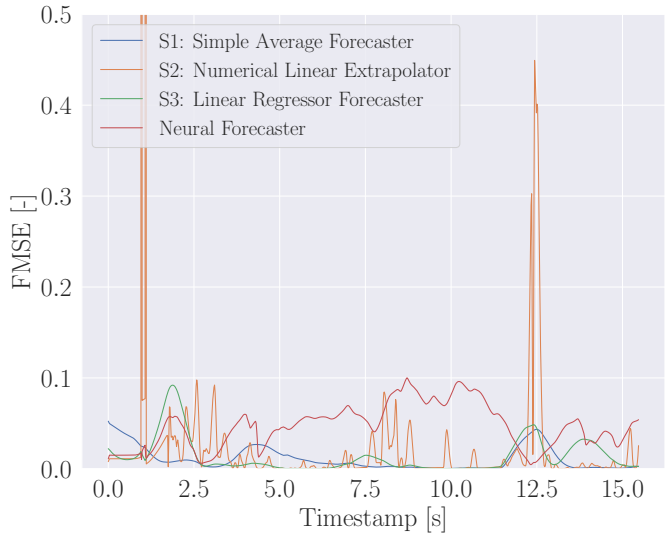
Appendix A.1.2 Aperiodic Motion



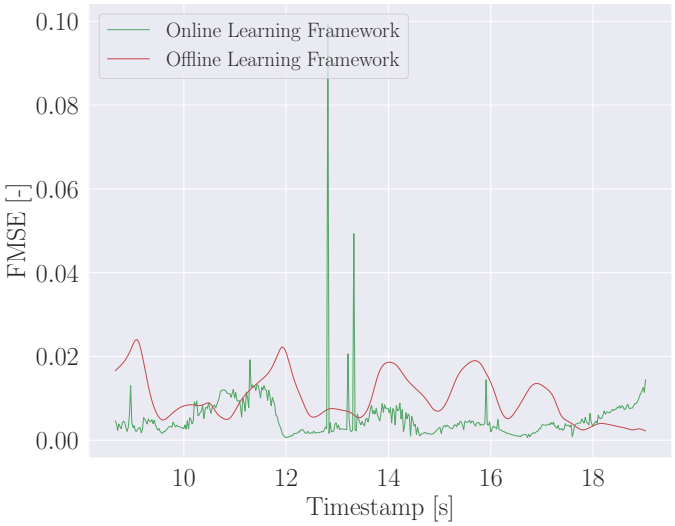
(a) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 0.25s$. The neural model outperforms all heuristics by an order of magnitude.)



(b) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 0.5s$. The neural model outperforms all heuristics by an order of magnitude.

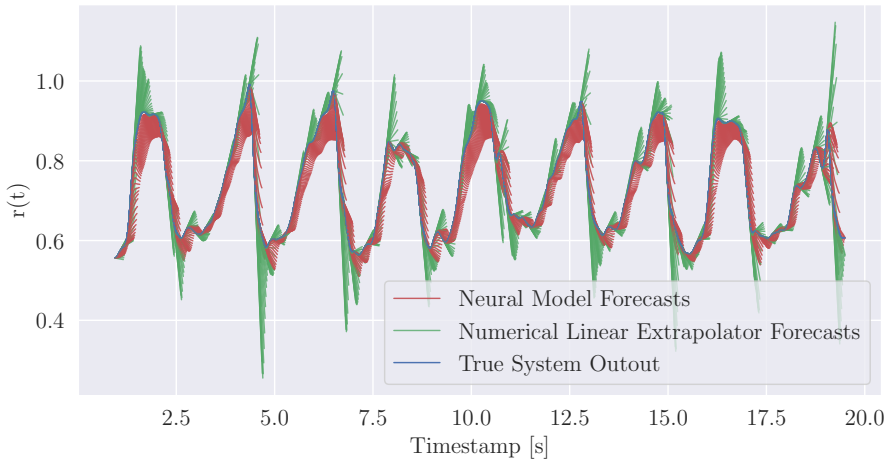


(c) Comparison of the forecast error, FMSE[-], of the heuristics and the offline neural model for the aperiodic motion test case when $H = 1.0s$. The neural model outperforms all heuristics by an order of magnitude.

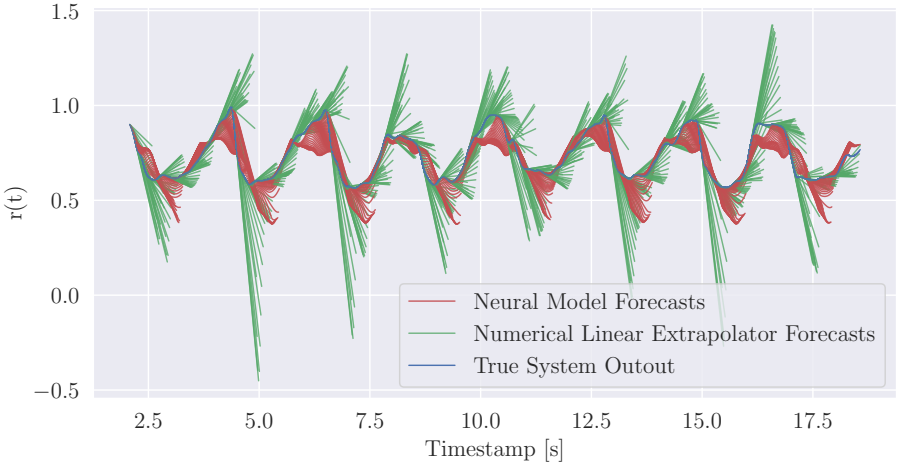


(d) The error of the online learning framework and the offline learning framework for comparison on the aperiodic motion test set for $H = 1.0s$. At the largest time horizon, the average error of the online learning framework can be seen to be lower.

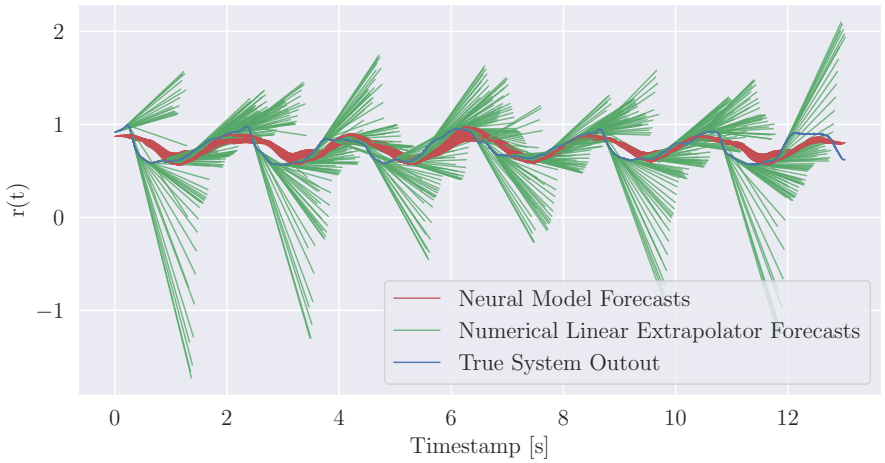
Appendix A.2.1 Periodic Motion



(a) The density plot for qualitative evaluation of the offline learning neural model on the periodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.25s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

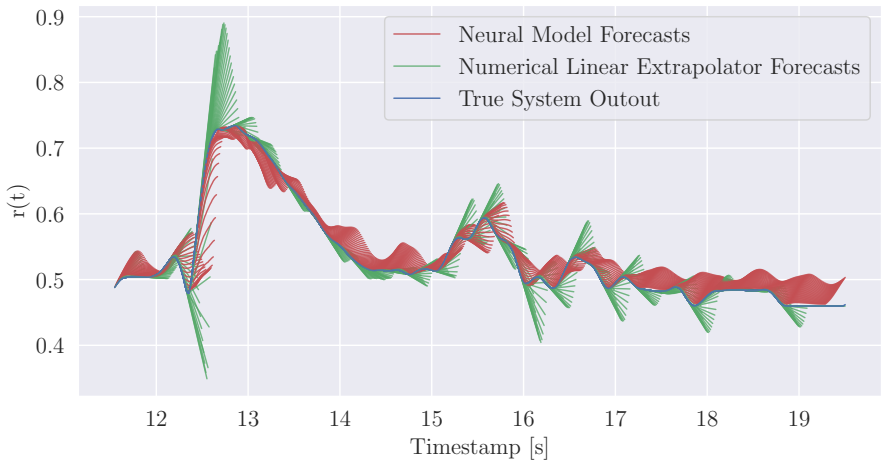


(b) The density plot for qualitative evaluation of the offline learning neural model on the periodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.5s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

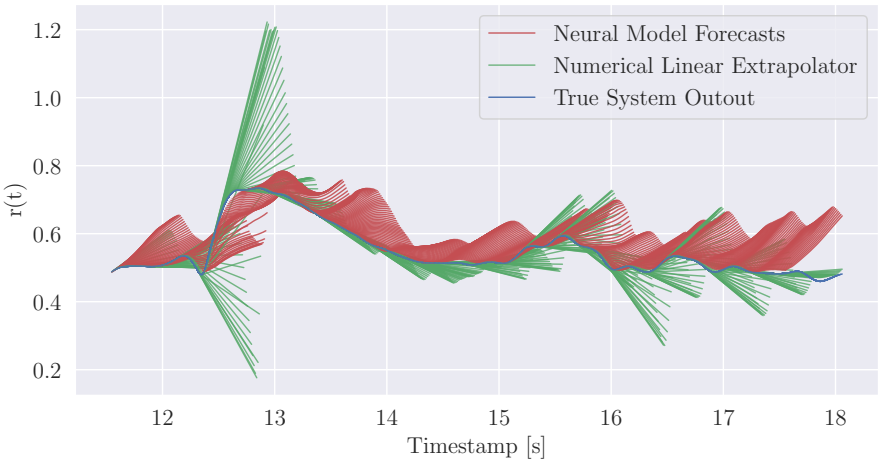


(c) The density plot for qualitative evaluation of the offline learning neural model on the periodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 1.0s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

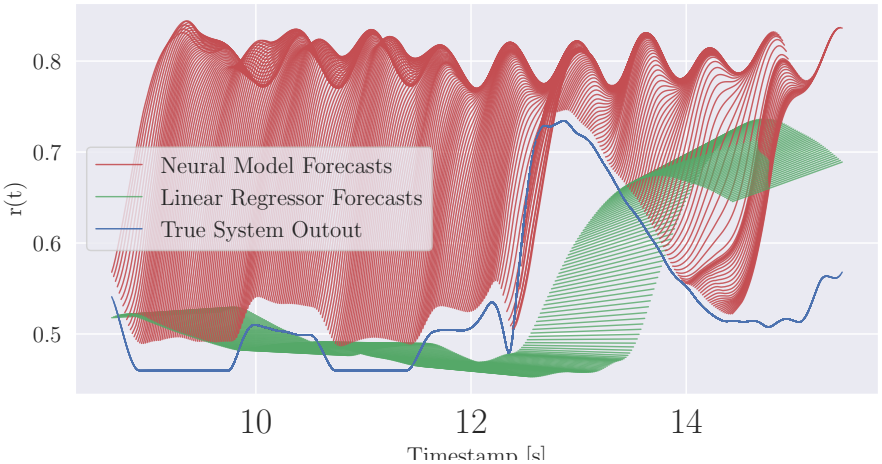
Appendix A.2.2 Aperiodic Motion



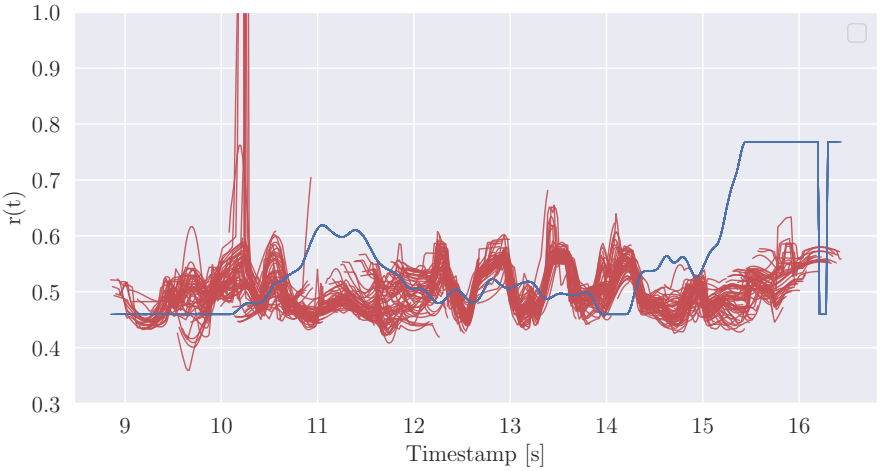
(a) The density plot for qualitative evaluation of the offline learning neural model on the aperiodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.25s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.



(b) The density plot for qualitative evaluation of the offline learning neural model on the aperiodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S2) to create a density plot when $H = 0.5s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.



(c) The density plot for qualitative evaluation of the offline learning neural model on the aperiodic motion test set. Consecutive forecasts from the offline neural network model overlayed along with the forecasts from the most comparable heuristic (S3) to create a density plot when $H = 1.0s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.



(d) The density plot for qualitative evaluation of the online learning neural model on the aperiodic motion test set. Consecutive forecasts from the online neural network model overlayed to create a density plot when $H = 1.0s$. The y-axis of the plot is the control output, $y_r(t) = r(t)$.

APPENDIX B VERIFICATION CASES AND DATA DETAILS

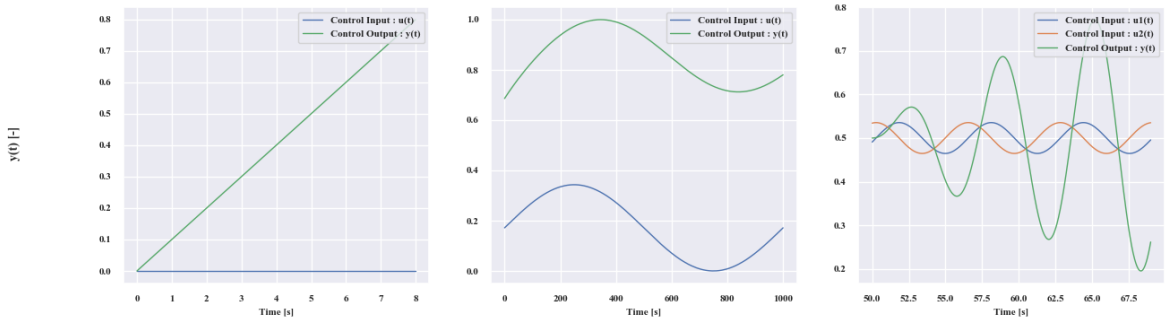


Figure 26 Verification benchmark control systems (from left to right): linear time-invariant first order integrator, first order thermal system, seconds order mass spring damper system with two degrees of freedom.

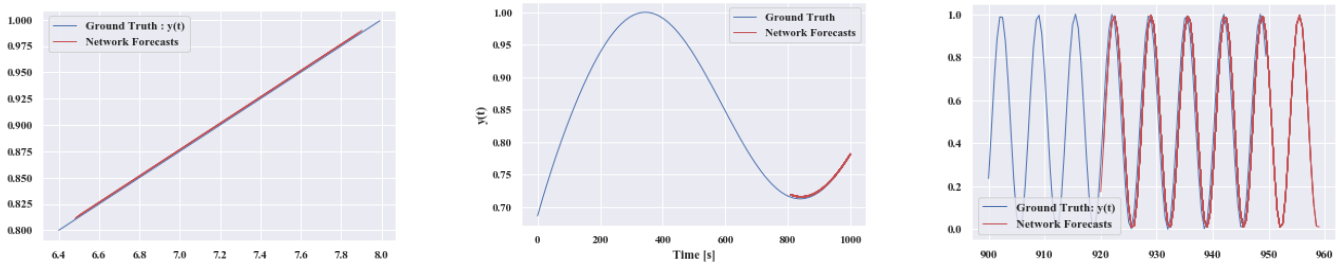


Figure 27 Density plots of the neural network forecasts of the verification cases. *From left to right:* Forecasting 10 seconds ahead at every time-step for a linear-time invariant first order integrator, forecasting 10 seconds ahead for a first-order thermal heating system , and forecasting 20 seconds ahead (a full period) for a seconds order mass spring damper system.

APPENDIX C SIMULATION CASE FURTHER DETAILS

Appendix C.1 The metric

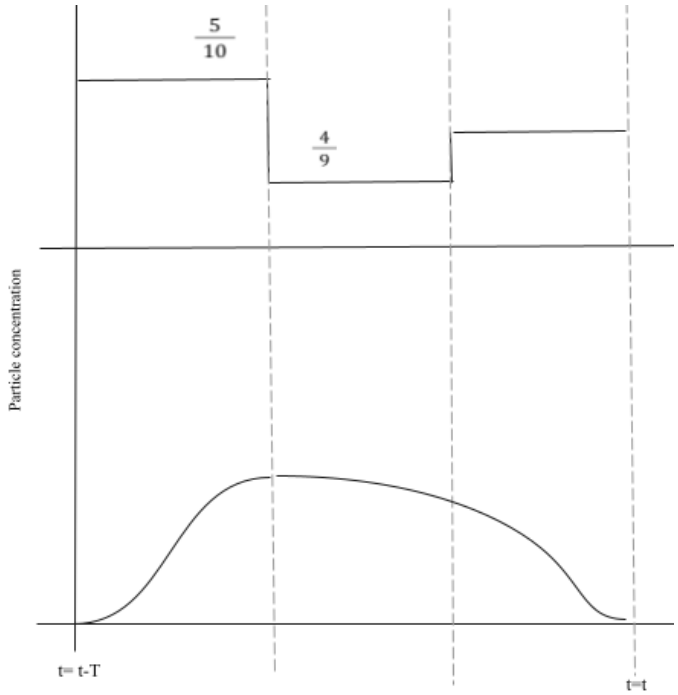


Figure 28 Top: Discreet metric over time window, Bottom: Continuous metric, a weighted version of the discreet metric across the measurement horizon T.

Appendix C.2 FSI Solver

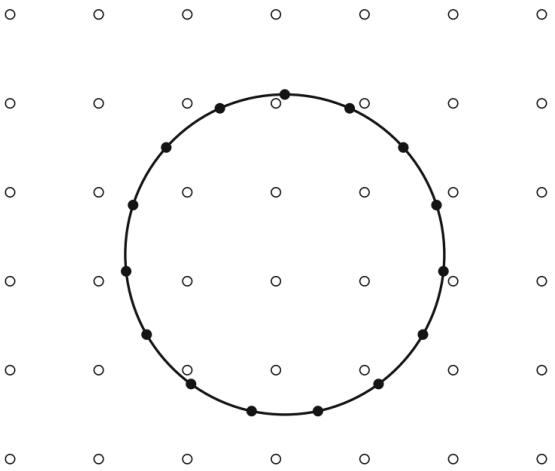


Figure 29 An example of the discritization for dynamic boundary handling with the LaBIB-FSI solver. A cylinder with aribtrary boundary markers is positioned in the regular fluid domain. A Eulerian mesh discritizing the entire fluid domain is indicated by the open circles, and he Lagrangian mesh definind the solid boundary nodes are indicated with the solid circles [25][21][22]

APPENDIX D RELATED WORK

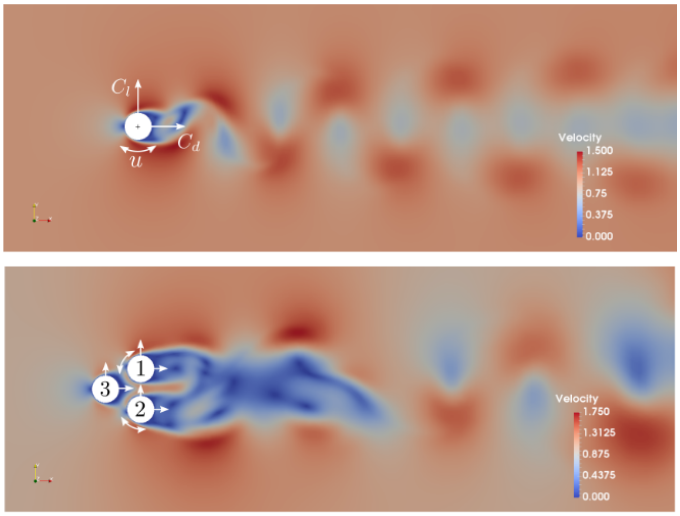


Figure 30 Example of actuation and flow cases considered in current SOTA AFC, *Top:* Single cylinder set-up. Control input is the angular velocity of the cylinder (u). *Bottom:* Fluidic pinball set-up [26]; control inputs are the angular velocities, $u1$ and $u2$, for cylinders 1 and 2 in the displayed configuration. [18]

APPENDIX E ACKNOWLEDGEMENTS

I want to thank my supervisors Cosimo and Angeliki first and foremost for giving me such a positive first experience with research. We have worked together for almost two years now, and I am so grateful for the constructive discussions we had and always constructive feedback that I received. They have become people that I truly look up to in a professional context. I am lucky to have had such a safe environment to grow and explore my potential in a research context, and I have to say that despite the challenges, I really did enjoy working on this. Regardless of the final outcome, I am proud of this work.

I would like to thank Sam; he has been my unofficial mentor for almost my entire career as a student at TU Delft. He pushed me to produce some of my best work and the discussions we have had have been invaluable to me. In the past five years, he has become somebody that I, both, look up to as well as consider a close friend. I want to especially thank him for handling my vents and spirals about my internal battles with the institution.

I could not have gotten through the last year without my best friend Mariano. Our morning coffee at the library is possibly half the reason I am motivated to go to the library every day. Quite frankly, it shocks me that we have anything left to talk about over coffee considering we've been doing this since we were 17. But I can't wait till our next coffee anyway. I want to thank Efe, who has been my rock through this last leg of my journey. He has dealt with me through the hardest period of my academic journey, and I was far from pleasant to be around. Thank you for making me laugh and dance despite all the hardship, these will be some of my core memories.

I want to thank my soulmate Cat, my closest friend since highschool. For all the sanity checks and scream cry calls and cynical spirals about society - thank you above all for being there for me after all these years. To my honourary motherfigure Karolina, it was absoloute kismat that we met. I don't know if I could have done this without you. Knowing that you carry my worry with and for me, gave me a reason to keep trying my best even when I could not anymore.

Thank you to all my girls, Eilidh, Mahima, Momo (and my bro Tristan) and Fernando. My first real friends in Delft, and the people who have become my family here.

I hope I make you proud.

Thank you to everybody who has been a part of my journey here. Let's hope I graduate after this sappy letter lol.

REFERENCES

[1] Isabel Scherl, Benjamin Strom, Jessica K Shang, Owen Williams, Brian L Polagye, and Steven L Brunton. Robust principal component analysis for modal decomposition of corrupt fluid flows. *Physical Review Fluids*, 5(5):054401, 2020.

[2] Maximilian Werhahn, You Xie, Mengyu Chu, and Nils Thuerey. A multi-pass gan for fluid flow super-resolution. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 2(2):1–21, 2019.

[3] Steven L. Brunton, Bernd R. Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. 52(1):477–508. _eprint: https://doi.org/10.1146/annurev-fluid-010719-060214.

[4] C Meneveau and I Marusic. Whither turbulence and big data in the 21st century, 2017.

[5] Mohsen Jahanmiri. Active flow control: a review. 2010.

[6] Won Tae Joe, Tim Colonius, and Douglas G MacMynowski. Feedback control of vortex shedding from an inclined flat plate. *Theoretical and Computational Fluid Dynamics*, 25:221–232, 2011.

[7] Mohamad Alsalman, Brendan Colvert, and Eva Kanso. Training bioinspired sensors to classify flows. *Bioinspiration & biomimetics*, 14(1):016009, 2018.

[8] Simona Colabrese, Kristian Gustavsson, Antonio Celani, and Luca Biferale. Flow navigation by smart microswimmers via reinforcement learning. *Physical review letters*, 118(15):158004, 2017.

[9] Kunihiko Taira, Steven L Brunton, Scott TM 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, 2017.

[10] Jeremy Morton, Antony Jameson, Mykel J Kochenderfer, and Freddie Witherden. Deep dynamical modeling and control of unsteady fluid flows. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[11] Feng Ren, Hai-bao Hu, and Hui Tang. Active flow control using machine learning: A brief review. 32(2):247–253.

[12] A Seifert, O Stalnov, D Sperber, G Arwatz, V Palei, S David, I Dayan, and I Fono. Large trucks drag reduction using active flow control. *The Aerodynamics of Heavy Vehicles II: Trucks, Buses, and Trains*, pages 115–133, 2009.

[13] Sebastian Peitz and Stefan Klus. Feedback control of nonlinear pdes using data-efficient reduced order models based on the koopman operator. *The Koopman Operator in Systems and Control: Concepts, Methodologies, and Applications*, pages 257–282, 2020.

[14] Jean Rabault, Miroslav Kuchta, Atle Jensen, Ulysse Réglade, and Nicolas Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of fluid mechanics*, 865:281–302, 2019.

[15] Jean Rabault and Alexander Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9):094105, 2019.

[16] P Mudiyanselage and F Gueniat. Deep reinforcement learning for the reduction of the drag in the flow past bluff bodies.

[17] Sebastian Peitz, Samuel E. Otto, and Clarence W. Rowley. Data-driven model predictive control using interpolated koopman generators. 19(3):2162–2193.

[18] Katharina Bieker, Sebastian Peitz, Steven L. Brunton, J. Nathan Kutz, and Michael Dellnitz. Deep model predictive control with online learning for complex physical systems.

[19] J.R. Noriega and Hong Wang. A direct adaptive neural-network control for unknown nonlinear systems and its application. 9(1):27–34. Conference Name: IEEE Transactions on Neural Networks.

[20] Edoardo Milana, Benjamin Gorissen, Sam Peerlinck, Michael De Volder, and Dominiek Reynaerts. Artificial soft cilia with asymmetric beating patterns for biomimetic low-reynolds-number fluid propulsion. 29(22):1900462. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/adfm.201900462.

[21] Sara Boby. Simulation environment for intelligent active flow control. Technical report, 2021. Unpublished manuscript.

[22] Samuel van Elsloo. Proposal and validation of immersed interface method applied to the lattice boltzmann method.

[23] Francesco Stella, Nana Obayashi, Cosimo Della Santina, and Josie Hughes. An experimental validation of the polynomial curvature model: Identification and optimal control of a soft underwater tentacle. 7(4):11410–11417. Conference Name: IEEE Robotics and Automation Letters.

[24] Keras Team. Keras documentation: EarlyStopping.

[25] Timm Krüger, Halim Kusumaatmaja, Alexandr Kuzmin, Orest Shardt, Goncalo Silva, and Erlend Magnus Viggen. The lattice boltzmann method. *Springer International Publishing*, 10(978-3):4–15, 2017.

[26] Nan Deng, Luc R Pastur, Marek Morzyński, and Bernd R Noack. Route to chaos in the fluidic pinball. In *Fluids Engineering Division Summer Meeting*, volume 51555, page V001T01A005. American Society of Mechanical Engineers, 2018.