An efficient intrusive deep reinforcement learning framework for OpenFOAM

Meccanica
https://doi.org/10.1007/s11012-024-01830-1

RESEARCH

An efficient intrusive deep reinforcement learning framework for OpenFOAM

Saeed Salehi

Received: 29 February 2024 / Accepted: 27 May 2024

Abstract Recent advancements in artificial intelligence and deep learning offer tremendous opportunities to tackle high-dimensional and challenging problems. Particularly, deep reinforcement learning (DRL) has been shown to be able to address optimal decision-making problems and control complex dynamical systems. DRL has received increased attention in the realm of computational fluid dynamics (CFD) due to its demonstrated ability to optimize complex flow control strategies. However, DRL algorithms often suffer from low sampling efficiency and require numerous interactions between the agent and the environment, necessitating frequent data exchanges. One significant bottleneck in coupled DRL–CFD algorithms is the extensive data communication between DRL and CFD codes. Non-intrusive algorithms where the DRL agent treats the CFD environment as a black box may come with the deficiency of increased computational cost due to overhead associated with the information exchange between the two DRL and CFD modules. In this article, a TensorFlow-based intrusive DRL–CFD framework is introduced where the agent model is integrated within the open-source CFD solver OpenFOAM. The integration eliminates the need for any external information exchange during DRL episodes. The framework is parallelized using the message passing interface to manage parallel environments for computationally intensive CFD cases through distributed computing. The performance and effectiveness of the framework are verified by controlling the vortex shedding behind two and three-dimensional cylinders, achieved as a result of minimizing drag and lift forces through an active flow control mechanism. The simulation results indicate that the trained controller can stabilize the flow and effectively mitigate the vortex shedding.

Keywords Deep reinforcement learning · Computational fluid dynamics · OpenFOAM · Intrusive · Active flow control

S. Salehi
Division of Fluid Dynamics, Department of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, Sweden

S. Salehi (✉)
Chalmers Industriteknik, Gothenburg, Sweden
e-mail: saeed.salehi@chalmers.se

1 Introduction

Reinforcement learning (RL) is a category of machine learning methods designed for solving decision-making and control problems. An RL algorithm involves an agent that learns to make decisions by interacting with an environment to achieve higher reward values. The agent's learning process usually relies on a trial-and-error procedure.

RL is a concept based on the Markov decision process (MDP) that describes a state-action-reward process. As shown in Fig. 1, the agent observes the *state*

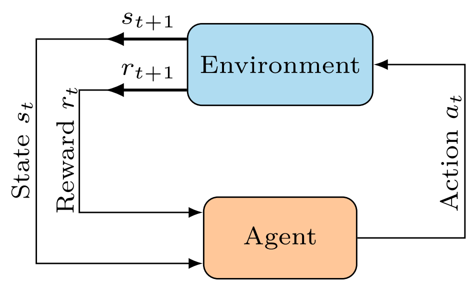


Fig. 1 Illustration of state, action, and reward sequence in the interactions between the agent and environment within the reinforcement learning algorithm

of the environment at time t (s_t), makes a decision, and performs an *action* (a_t). Based on the performed action, the environment transitions to a new state (s_{t+1}) and provides a *reward* (r_{t+1}) signal to the agent that determines the quality of the taken action. The agent aims to learn a policy that maps the states of the environment to actions such that it can make optimal decisions to maximize the expected return which is defined as the cumulative reward, often subject to discounting.

Deep reinforcement learning (DRL) is a combination of deep learning and RL. In DRL, a deep neural network (DNN) works as the agent and is trained to make optimal decisions. Recent advancements in artificial intelligence and deep learning enabled tackling high-dimensional control and decision-making problems through DRL. It has been shown that DRL can perform immensely complicated cognitive tasks at a superhuman level, such as playing classical Atari games [1] or the game of Go [2]. Although RL has been an active research area for years, these recent breakthroughs made the DRL field much more striving and progress much faster.

DRL has been utilized in fluid mechanics for different purposes, for instance, training an autonomous glider [3], exploring swimming strategies of fish [4], controlling a fluid-directed rigid body [5], proposing shape optimization [6, 7], or active flow control (AFC) [8]. It is repeatedly shown that DNNs are able to learn complicated control strategies through RL, to reduce the drag and mitigate the vortex shedding effects behind a 2D cylinder using two synthetic jets (such as Refs. [8, 9]). Other researchers utilized DNNs to find optimal strategies for active flow control through DRL. There have been successful attempts to stabilize the vortex shedding effects behind a 2D cylinder by using DRL through rotational oscillations in numerical simulations [10, 11].

Beintema et al. [12] sought a control mechanism for a 2D Rayleigh–Bénard convection, and showed that DRL-based controls remarkably outperform linear approaches. Later, the methodology was applied to control flow over a 2D NACA airfoil under weak turbulent conditions, and significant drag reduction and lift stabilization were achieved [13]. The application of DRL in fluid mechanics is not restricted to numerical simulations, and the feasibility of discovering effective active flow control strategies in experimental fluid mechanics has also been demonstrated [14].

The RL methods typically exhibit a low sampling efficiency, meaning that a large number of interactions between the environment and the agent are required to train a model. Often, thousands of such interactions are needed for the agent to learn to make near-optimal decisions. However, when applying RL to computational fluid dynamics (CFD) environments, a significant challenge arises due to the substantial amount of data exchange. Thereby, one of the main bottlenecks of DRL–CFD frameworks remains the efficient data communications.

The DRL (agent) and CFD (environment) parts of such coupled algorithms are often implemented in separate programs, necessitating a communication interface for agent-environment interaction. A rather straightforward method to manage such communication is the “non-intrusive" approach, where the DRL code treats the CFD program as a black box, interacting with it without any direct integration.

One example of such non-intrusive DRL–CFD couplings is the DRLinFluids [15], a framework designed to couple OpenFOAM with DRL programs. However, a significant drawback of the framework is that it heavily relies on Input/Output (I/O) operations. Each agent-environment interaction requires interrupting and restarting the environment simulation. The CFD simulation is stopped to read the newly produced action by the agent and restarts to continue to the next control step. The extensive I/O operations severely reduce the efficiency, making it impractical for realistic CFD applications with considerable computational costs.

The communication burden between the agent and the environment has been addressed more efficiently through various methods [16–18].

Relexi [16] integrates TF-Agents with a high-order spectral CFD code, FLEXI. The SmartSim [19] package was utilized to facilitate information exchange between the DRL (TF-Agents) and CFD (Flexi) components of the framework. It was shown that Relexi could retain the scaling capabilities of the CFD code for high-performance computing.

Gym-preCICE [17] integrates OpenFOAM with DRL using the external package preCICE [20], which is an open-source coupling library for partitioned multi-physics simulations, to handle information exchange.

The non-intrusive strategy can even provide the possibility of coupling with commercial CFD software where the code is not accessible. DRLFluent [21] exemplifies such a framework by coupling Ansys Fluent with a Python DRL program through the omniORB package.

The non-intrusive frameworks may occasionally be preferred in the realm of CFD, due to their simplicity of implementation and lack of necessity to have access to and modify the CFD code. However, it's important to recognize that such algorithms rely on a communication interface between the DRL agent and the CFD solver through a third-party package. While reliance on an external package can simplify implementation, it introduces challenges such as dependency management and lack of control. Additionally, external communication interfaces may add computational overhead, leading to higher costs that are not easily measured.

On the contrary, an intrusive framework directly integrates the DRL agent within the CFD environment, eliminating the need for external information exchange. As an example of such a framework, drlFoam [22] couples a PyTorch-based DRL framework with OpenFOAM intrusively. The agent's model is directly loaded inside OpenFOAM and thereby no external communication is needed during one episode.

The current article concerns introducing yet another intrusive DRL–CFD coupling framework for OpenFOAM. Similar to dlrFoam, the agent's model is directly integrated within the CFD environment, eliminating the need for external information exchange. The agent's action is generated within OpenFOAM while having access to the state (current flow field solution). A full DRL episode can be conducted seamlessly and without any need to wait for data exchange. However, unlike drlFoam, the framework relies on the TensorFlow-based DRL package Tensorforce [23]. Additionally, the parallelization of the current framework is done through Python's implementation of Message Passing Interface (MPI), i.e., mpi4py [24] enabling the harnessing of distributed computing across multiple nodes of clusters, particularly advantageous for resource-intensive CFD cases.

A comprehensive explanation of the framework implementation and parallelization strategy is provided in Sect. 2. Section 3 tests and verifies the performance of the framework on two test cases, namely, active flow control of vortex shedding behind two and three-dimensional cylinders. Finally, the conclusions are provided in Sect. 4.

2 Developed framework

The fundamental theories of the MDP and DRL are not explained for brevity, and the readers are referred to the literature for more information [25]. Therefore, the main focus of the current section is on the practical aspects of the implementation of the developed intrusive DRL–CFD coupling framework.

2.1 Intrusive DRL–CFD coupling

The DRL computations are performed using the open-source library Tensorforce [23] which is a TensorFlow-based library for applied RL. Particularly, the latest version of this library, namely Tensorforce 0.6.5, is utilized. As explained in the previous section, all the episodes' computations and information exchange between the agent and the environment are carried out within OpenFOAM. Here, the term “intrusive" refers to the necessity of accessing and modifying the CFD program.

The DRL code initializes an agent model, which is then loaded into OpenFOAM to execute complete episodes. After a certain number of episodes, the state-action-reward sequence is sent back to the DRL code to update and calculate a new agent, based on the recorded experience, and the loop is continued.

2.2 OpenFOAM implementation

The integration of the DRL agent within the OpenFOAM can be carried out through implementation into various OpenFOAM library components such as physical or numerical models. In this article, one such implementation is explored that incorporates a DRL agent as an OpenFOAM boundary condition, as a proof of concept. The boundary condition supplies a fixed value inlet velocity, acting as a jet controller for active flow control within a fluid flow domain. At each control time step, the jet velocity magnitude is determined by the agent, which is a DNN. As explained previously, to minimize unnecessary information exchange between the Python and C++ codes, the DNN is loaded directly into the implemented boundary condition. Here, the TensorFlow agent policy is loaded in the OpenFOAM boundary condition using the CppFlow library [26] that utilizes the TensorFlow C Application Programming Interface (API) to run the models.

To perform an action, the current state of the environment should be evaluated. The state could be considered as any fundamental or derived properties of the flow field at any location. In the current implementation, users have the capability to specify both the monitored field and the corresponding locations.

Since the agent policy model does not usually alter throughout an entire DRL episode, it is loaded into the boundary condition as a member function only once inside the constructor of the class, as illustrated in Listing 1. The input of the `cppflow::model` constructor is the address of the policy model stored in the user-defined variable `policyDirName_`.

The agent action is computed by providing the appropriate inputs to the DNN model and performing a forward pass. The characteristics of the required inputs and the generated output of the saved model can be examined using TensorFlow's command-line interface tool. In Listing 2, the forward pass of the model is depicted to generate an action. The `stateTensor` contains the current state of the environment, whereas the `detTensor` is a boolean-type tensor specifying whether the model should act deterministically (for the evaluation) or stochastically (for the training), corresponding to true and false cases, respectively. The DRL agent model produces a Probability Distribution Function (PDF) of action values. In the deterministic mode, the mean of the PDF is utilized, while in the stochastic mode, the action is randomly sampled from the PDF.

Listing 1: Loading the agent policy model

```
model_
(
    cppflow::model
    (
        fileName
        (
            db().time().globalPath()/policyDirName_
        )
    )
)
```

Listing 2: Forward pass of the policy model to create an action

```
auto action = model_
(
    {
        // Model inputs
        {"serving_default_args_0:0", stateTensor},
        {"serving_default_deterministic:0", detTensor}
    },
    {
        // Model output
        "StatefulPartitionedCall:0"
    }
);
```

In OpenFOAM, boundary conditions are typically updated whenever the discretized equation matrices are constructed, which occurs at every corrector loop of the pressure–velocity coupling solver, within each time step. This approach works well for most OpenFOAM boundary conditions, where values remain constant within each time step. However, during the training phase of DRL, the agent acts non-deterministically to explore the action space effectively. Each time the code runs the forward pass of the policy model, it produces a different value. This means that the controller jet velocity varies even within corrector loops of a single time step. To address this issue, and have a fixed boundary value during a single CFD time step, the jet controller velocity is updated only once the time index is updated, which happens at the beginning of each time step. This ensures consistency in the DRL training process within the OpenFOAM framework.

Furthermore, OpenFOAM employs domain decomposition to facilitate parallel processing, which means that the boundary surface where the agent policy is applied can potentially decompose into different domains, with computations handled by different processors. However, because of the inherent randomness of the agent policy model, each processor may end up with different boundary values. To maintain consistency in boundary values across different processors, the agent's computations are performed only by the master processor, and the new action is broadcast to other processors without modification using the OpenFOAM's inter-processor communications stream (see Listing 3).

Listing 3: Master processor computation of agent's actions and broadcasting across processors

```
if (Pstream::master())
{
    actionOld_ = actionNew_;
    actionNew_ = agentAction(state);
    writeStateAction(state, actionNew_);
    Info<< "New action = " << actionNew_
        << ", Old action = " << actionOld_ << endl;
}
// Broadcast the same action value on all processors
Pstream::scatter(actionNew_);
Pstream::scatter(actionOld_);
```

The action value can vary significantly between subsequent control time steps, which may cause numerical instability. To mitigate this issue and ensure numerical convergence, the previous action value is linearly ramped up or down to the new value. The jet velocity at CFD simulation time step i is computed using

$$U_i = a_{j-1} + \alpha_i(a_j - a_{j-1}), \tag{1}$$

where index j indicates the index of the action time step, while a_{j-1} and a_j represent the previous (old) and current (new) action values. α_i is the ramping function linearly varies from 0 to 1 during the ramping period. The ramping period is a fraction of the action period and can be chosen by the user.

Lastly, Listing 4 shows an example of the required inputs for the implemented boundary condition on the user side.

It should be noted that the OpenFOAM's implementation of the current framework is not limited to the Tensorforce model and can operate with any TensorFlow-based DRL agent, such as TF-Agents.

2.3 Parallelization

The current framework was specifically designed to handle large-scale CFD applications on a cluster. Tensorforce package already offers parallel computations of the simultaneous environments through Python's multiprocessing package. However, a key limitation of this implementation is that it does not inherently support distributed computing across multiple machines or nodes.

As a remedy, the parallelization procedure was reimplemented via Python's implementation of MPI, mpi4py [24]. This allows the DRL training program to be executed on multiple nodes of a cluster via MPI, for instance:

Listing 4: An example the implemented boundary condition inputs

```
jet
{
    type            agentJet;
    deterministic   false;
    controlPeriod   0.4;
    rampUpPeriod    0.2;
    policyDir       "model";
    stateField      p;
    stateProbesNo   99;
    interpolationScheme cellPoint;
    stateProbeLocations nonuniform List<vector>
    99
    (
        ( 0.550000 -1.250000 0 )
        ( 0.550000 -0.937500 0 )
        ( 0.550000 -0.625000 0 )
        // the list continues
    )
}
```

```
mpirun -np 5 python3 training_mpi.py &> log.training
```

In this exemplary command, mpi4py launches five instances of the training program (i.e., processor0 through processor4). Each instance independently executes a sequence of CFD simulations by spawning parallel OpenFOAM cases on multiple cores. While all processors are responsible for generating their own state-action-rewards sequences, only the master processor handles the DRL calculations. Therefore, all other processors send their corresponding state-action-rewards sequences to the master processor through MPI communication. The master processor updates the agent based on DRL calculations, and then sends the updated agent to the rest of the processors. This process continues iteratively until the training reaches the predefined maximum number of epochs.

The schematic diagram, shown in Fig. 2, provides an overview of the framework's workflow. It visually illustrates the process described above, detailing how the training program operates across multiple processors, the role of the master processor in aggregating and updating data, and the communication between processors via MPI.

3 Case study

The performance of the developed DRL–CFD framework is assessed using different numerical case studies. The first test case focuses on controlling vortex shedding by reducing the drag of a two-dimensional cylinder. The second test case is more computationally intensive and involves drag reduction of a three-dimensional confined cylinder.

It is worth mentioning that the current test cases were studied using OpenFOAM-v2112 and Tensorforce 0.6.5.

3.1 Vortex shedding behind a 2D cylinder

Active flow control of the laminar vortex shedding behind a two-dimensional circular cylinder has been frequently used in the literature for examining the performance of DRL algorithms within CFD [9, 27]. The goal of the DRL would be to optimize a control mechanism to minimize the drag and lift forces and consequently eliminate the von Kármán vortex street phenomenon.

The configuration of the under-investigation test case is illustrated in Fig. 3. As seen in Fig. 3a, two synthetic jets are considered on the top and bottom of the cylinder as the actuators of the controller, whose flow rates are decided by the DRL algorithm. The cylinder is assumed unconfined, and placed at the center of the computational domain (shown in Fig. 3b). The domain is considered large enough to minimize the effect of boundary conditions. The Reynolds number of the flow field, based on the inlet velocity and the cylinder diameter, is $\mathrm{Re} = U_\infty D/\nu = 100$.

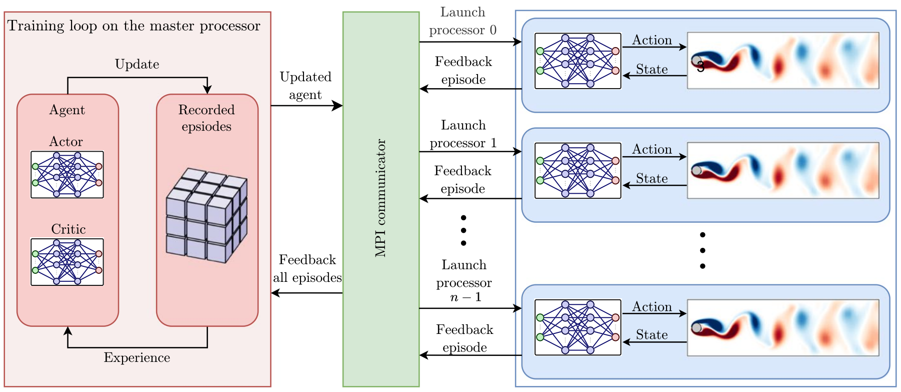
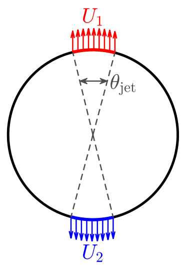
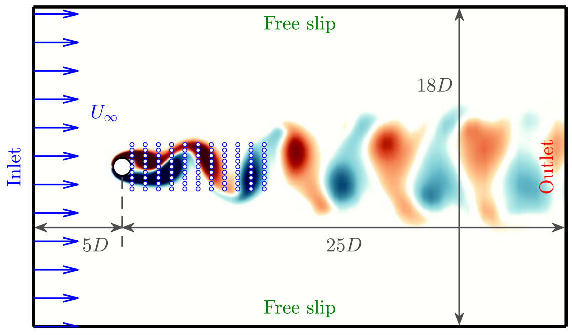


Fig. 2 High-level architecture of the implemented framework



(a) Close view of the cylinder and the jet actuators



(b) Full computational domain and boundaries

Fig. 3 Configuration of the 2D environment flow field, boundaries, the agent actuators, and the locations of the probes (blue circles) to sense the environment state

3.1.1 CFD framework and verification

The transient incompressible Navier–Stokes equations are solved on a collocated mesh using OpenFOAM. Temporal derivatives are discretized employing the implicit second-order backward scheme [28]. The transient flow is simulated using a non-dimensional time step of $\Delta t = 10^{-2}$. All time units are normalized via the characteristic timescale D/U_∞. The convective terms are discretized using the second-order linear scheme to minimize the numerical diffusion.

The pressure is linked to the velocity field through the PIMPLE pressure correction algorithm, combining SIMPLE [29] and PISO [30] algorithms. A maximum of eight outer correction and two inner correction loops are performed at each time step to ensure proper convergence. The residual of 10^{-4} is employed as a stopping criterion for the outer correction loop.

A spatially uniform fixed velocity boundary condition is applied on the inlet of the domain, while the outlet boundary condition is set to constant static pressure. The top and bottom boundaries are considered free-slip walls, while a no-slip condition is imposed on the cylinder.

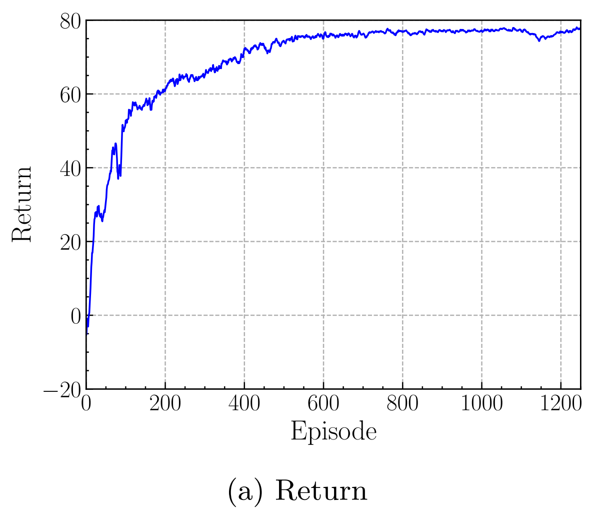
The dependence of the numerical results on mesh resolution is assessed here through a mesh study, considering five computational meshes with varying levels of density. For each mesh, a baseline (uncontrolled) vortex shedding simulation is conducted. Drag coefficient, lift coefficient, and shedding Strouhal number, defined by

$$C_D = \frac{F_D}{\frac{1}{2}\rho U_\infty^2 D}, \qquad C_L = \frac{F_L}{\frac{1}{2}\rho U_\infty^2 D}, \qquad \mathrm{St} = \frac{fD}{U_\infty} \tag{2}$$

are examined and compared to the reported values in the literature [31] for verification and validation. Here, F_D, F_L, and f are the drag force, lift force, and fundamental frequency of the shedding phenomenon, respectively. As presented in Table 1, the results exhibit a converging trend with mesh refinement. Considering the relative errors (calculated with respect to the reference values), mesh Level 4 is chosen for the DRL computations.

Table 1 Mesh independence study and validation of the 2D vortex shedding case

Level	N_{cells}	Drag (C_D)		Lift (C_L)		Strouhal number (St)	
		Mean	Error (%)	RMS	Error (%)	Value	Error (%)
1	7.18×10^3	1.401	0.05	0.189	21.33	0.1733	1.96
2	1.48×10^4	1.417	1.23	0.209	13.08	0.1733	1.96
3	2.41×10^4	1.424	1.69	0.216	10.01	0.1733	1.96
4	3.69×10^4	1.427	1.91	0.219	8.83	0.1733	1.96
5	6.46×10^4	1.428	2.02	0.218	9.27	0.1733	1.96
Ref. [31]	–	1.40	–	0.240	–	0.170	–



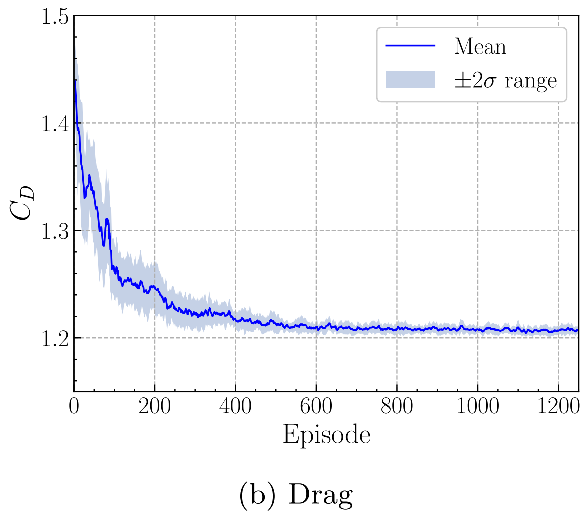


Fig. 4 History of **a** episodes' return (undiscounted sum of rewards) and **b** drag coefficient variation throughout the training simulation

3.1.2 Configuration of the DRL training

The controller actuators are considered as two synthetic jets with the angle of $\theta_{\mathrm{jet}} = 10°$ (see Fig. 3a). These jets are capable of injecting or suctioning a uniform flow in the normal direction. The sum of velocities on the synthetic jets is set to zero, $U_1 + U_2 = 0$. This implies that the drag reduction is achieved through indirect active flow control, rather than by injecting momentum in the axial direction to alleviate the separation zone. The maximum jet flow rate is considered to be 10% of the flow rate encountering the cylinder, i.e., $Q_{\mathrm{j,max}} = 0.1 U_\infty D$.

The state of the environment is characterized by 99 probes arranged uniformly within the vortex shedding region, in an array of 11×9 spanning the range of $0.55 \leq x/D \leq 8$ and $-1.25 \leq y/D \leq 1.25$. The location of the probes is illustrated by blue circles in Fig. 3b. The pressure field is sensed by these probes and the resulting vector of 99 pressure values is fed back to the DRL agent as the current state of the environment.

The DRL algorithm strives to find a control law that maximizes the expected cumulative reward. Hence, to minimize drag and lift forces, the reward function is formulated as

$$r = \langle C_D \rangle_{\mathrm{baseline}} - \langle C_D \rangle - \beta \langle |C_L| \rangle. \tag{3}$$

Here, $\langle \cdot \rangle$ denotes the moving-averaged value over a complete shedding period. $\langle C_D \rangle_{\mathrm{baseline}}$ is a constant and represents the time-averaged drag coefficient of

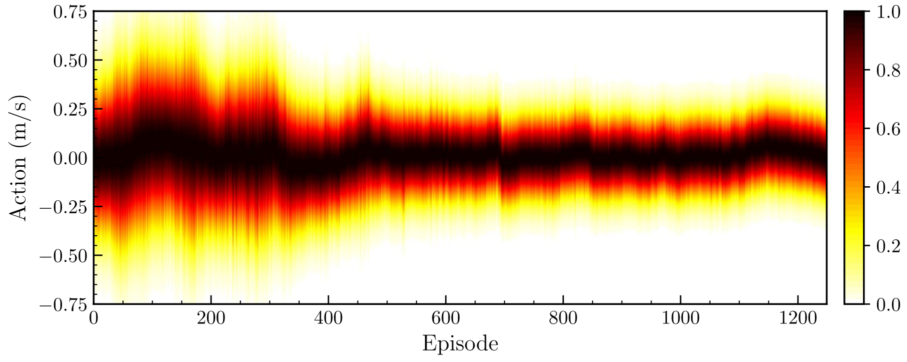


Fig. 5 History of normalized 2D PDF of action values during the training simulation

the baseline (uncontrolled) case and is calculated at 1.427 (Table 1). Including this constant in the reward function facilitates relative comparison with the baseline case. A successful drag reduction results in a positive reward, while failure to do so yields a negative reward for the algorithm. Incorporating the lift coefficient into the reward function not only accelerates the convergence of training but also mitigates the risk of encountering extreme solutions with significant lift oscillations. The β specifies the importance of the lift coefficient in the reward function and is chosen $\beta = 0.2$ in this study.

The non-dimensional action time step, i.e., the time between two consecutive actions, is considered $\Delta T_{\text{action}} = 0.4$ to ensure the execution of approximately 15 actions at each shedding period. As explained in Sect. 2, in order to avoid numerical instability from abrupt changes in the agent's action, the jet value is linearly varied from the previous to new values. The ramping period is considered half of the action time, $\Delta T_{\text{ramp}} = 0.2$. Each episode consists of 400 actions, resulting in an episode period of $T_{\text{episode}} = 160$. It's worth noting that all episodes initiate from a unique saturated shedding flow condition.

The agent is trained by the Proximal Policy Optimization (PPO) [32] algorithm to maximize the expected return. The policy (actor) and value (critic) models are represented by DNNs with similar architectures. Each DNN is a fully connected network comprised of input, two dense hidden, and output layers. Specifically, each hidden layer contains 64 neurons activated by the hyperbolic tangent (tanh) activation function.

In training mode, the policy network generates a Gaussian distribution from which the utilized action value of the actuator is sampled, allowing for probabilistic decision-making that facilitates exploration. Conversely, in the evaluation mode, the network produces a deterministic action which is the mean of the distribution, to maximize the exploitation of the learned policy. Additionally, the value network provides the state value which is necessary for calculating advantage estimates during the training.

The DRL framework is designed to execute multiple episodes in parallel. In this setup, a batch size of five episodes is utilized, meaning that five CFD simulations run simultaneously during each iteration of the DRL algorithm. These parallel episodes share the same policy model. Upon completion of the batch, the recorded data, including states, actions, and rewards, is collected via MPI communications and stored for subsequent training.

It was observed that incorporating data from previous batches in the training phase leads to more stable convergence. Consequently, the policy and value models are trained using data from the most recent 25 episodes.

The training is performed through the stochastic gradient descent method, utilizing the Adam optimizer algorithm. The learning rates of the policy and value networks are considered 0.0005 and 0.001, respectively. The rewards are discounted using the factor of 0.99 while the policy ratio is clipped with a ratio of 0.2. More information about the details of the studied test case is presented in Appendix A.

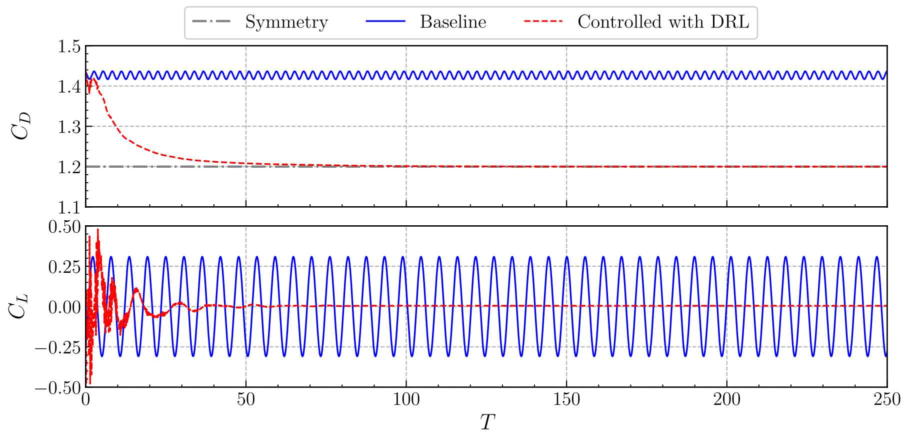


Fig. 6 Variation of drag and lift coefficients for the uncontrolled (baseline) and DRL controlled cases. The controlling mechanism starts at $T = 0$

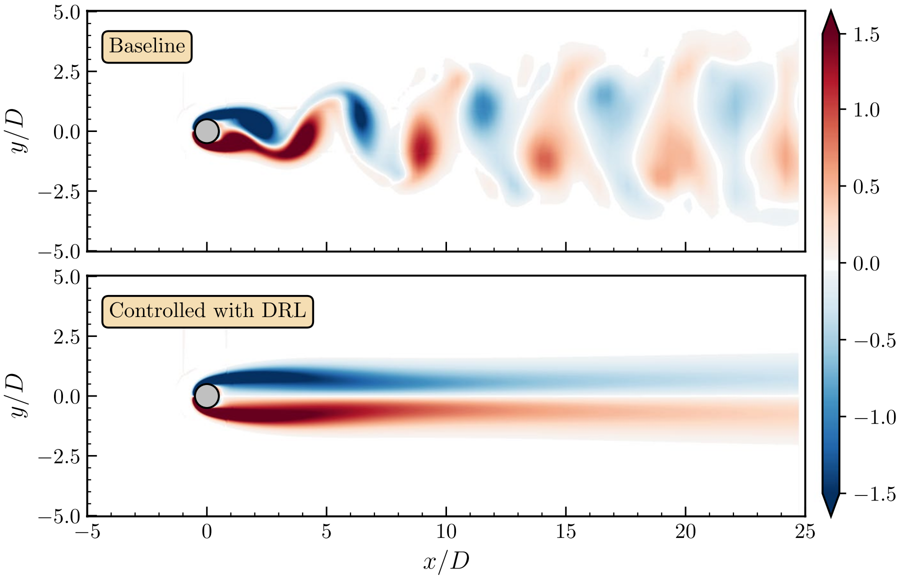


Fig. 7 Comparison of vorticity contours (ω_z) in the baseline (uncontrolled) and controlled cases at $T = 200$

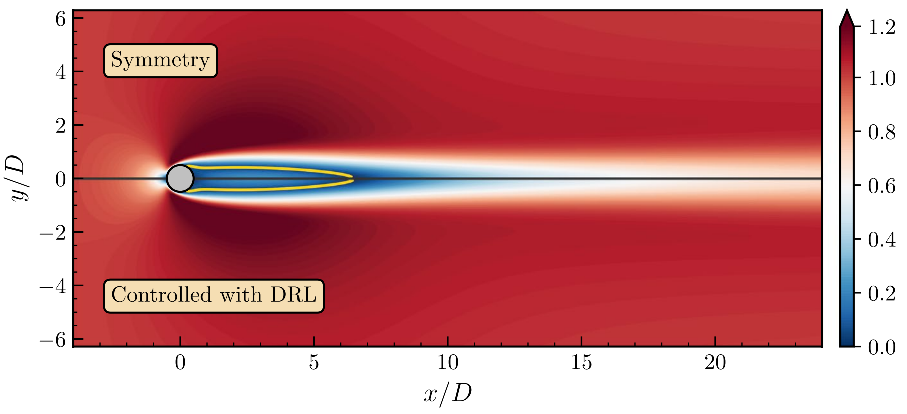


Fig. 8 Comparison of velocity magnitude contours in the steady symmetry (top) and unsteady controlled (bottom) cases. The solid gold line represents the $U_x = 0$ iso-surface illustrating the recirculation region behind the cylinder

3.1.3 Training results

The simulation is performed using 250 iterations of the DRL algorithm (250 epochs). Considering the parallel CFD batches, the DRL agent is trained based on data collected from a total of 1250 CFD simulation episodes.

The history of the episodes' return (undiscounted sum of rewards in an episode) is plotted against the episode number in Fig. 4a. The values of the parallel environments are concatenated sequentially, providing a comprehensive view of the returns across all episodes. The variation indicates a smooth convergence throughout the whole simulation. The controller reaches convergence after around 800 episodes, and return values do not seem to change considerably with further iterations.

The variation of the drag coefficient, averaged over the second half of each episode, along with its corresponding standard deviation range, is illustrated in Fig. 4b. Initially, there is a notable variation in the drag coefficient due to the agent's high exploration, which gradually decreases as the simulation converges and the policy becomes less stochastic.

Figure 5 displays the evolution of the normalized 2D probability distribution function (PDF/ max(PDF)) of all action values over the entire episode throughout the training process. In the initial stages of training, when the policy is less mature, it explores a wide range of actions, resulting in a broad variation of drag coefficients. However, as the agent

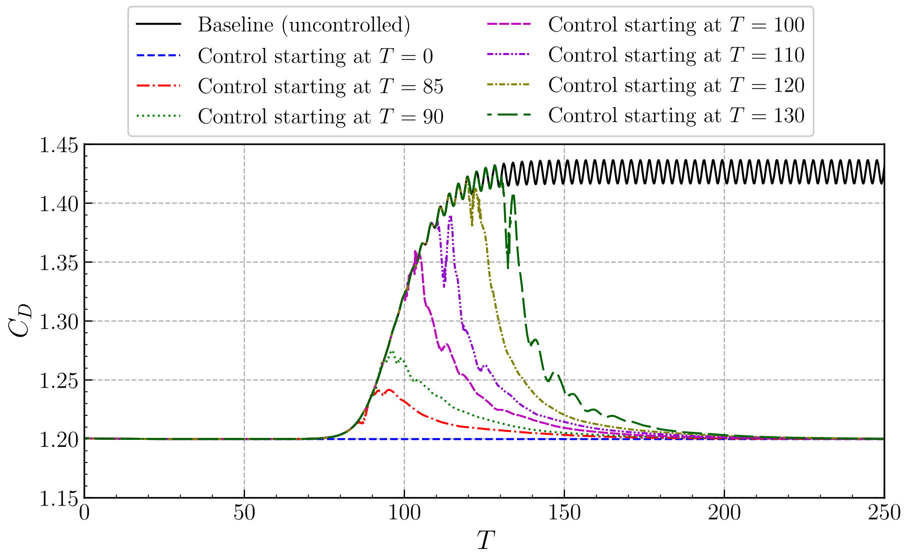
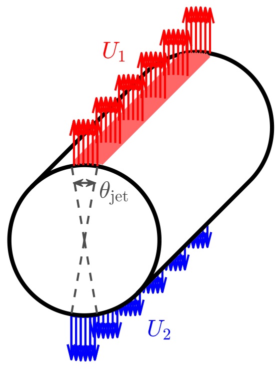


Fig. 9 Assessing robustness of the trained controller by exposing it to different flow conditions that are not experienced in the training phase. The figure illustrates the variation of the drag coefficient when the controller activates at non-saturated shedding conditions

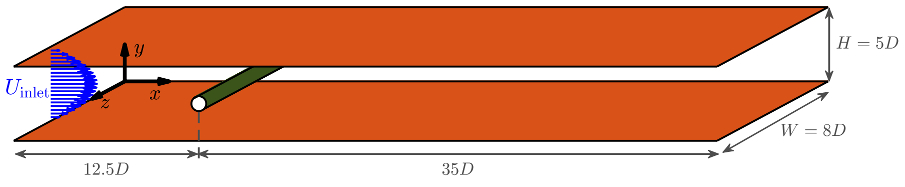
becomes more experienced, it refines the policy in favor of higher rewards, which leads to reduced exploration in the action values. As depicted, the normalized 2D PDF of action values progressively narrows over the course of training. This shift towards a more deterministic policy reflects the agent's improved understanding of the environment and its ability to consistently select actions that lead to favorable outcomes. It is noteworthy that although the PDF range decreases substantially, a certain degree of exploration is maintained throughout the whole training process.

After the convergence of the training process, the trained policy is utilized in deterministic mode to perform a single-episode simulation, evaluating the performance of the learned model.

In order to provide a reference value for drag reduction, a steady simulation considering the top half of the domain using a symmetry boundary condition was performed. As explained in the literature [33], the mean drag consists of contributions from



(a) Close view of the cylinder and the jet actuators



(b) Full computational domain and boundaries

Fig. 10 Configuration of the 3D environment flow field, boundaries, and the agent actuators

Table 2 Mesh independence study and validation of the 3D vortex shedding case

Level	N_{cells}	Drag (C_D)		Lift (C_L)		Strouhal number (St)	
		Mean	Error (%)	RMS	Error (%)	Value	Error (%)
1	1.58×10^5	1.1593	6.422	0.1620	40.25	0.1667	9.91
2	3.20×10^5	1.1841	4.424	0.1892	30.17	0.1833	0.90
3	6.33×10^5	1.1976	3.332	0.2089	22.93	0.1833	0.90
4	1.32×10^6	1.2090	2.414	0.2234	17.56	0.1833	0.90
5	2.56×10^6	1.2143	1.989	0.2309	14.81	0.1833	0.90
6	5.64×10^6	1.2188	1.619	0.2378	12.23	0.1833	0.90
7	1.13×10^7	1.2207	1.469	0.2402	11.37	0.1833	0.90
Ref. [35]	6.77×10^6	1.2389	–	0.271	–	0.1850	–

two components: the steady symmetrical flow (base flow) and the oscillations of the vortex shedding [33]. The drag contribution from the base flow cannot be reduced at a fixed Reynolds number, and a controller can only decrease the contribution from the vortex shedding [34]. Therefore, the base symmetry flow can serve as a reference value for evaluating the controller's performance. In this particular test case, the base flow drag coefficient is calculated $C_{D,\text{sym}} = 1.20$.

Figure 6 compares the variation of drag and lift coefficients for both the baseline (uncontrolled) and controlled cases. It is evident that both forces reduce significantly as a result of the designed controller. The drag coefficient reduces by around 16% and reaches the value observed in the symmetry simulation (base flow), indicating that the DRL algorithm has reached the global minimum and completely removed the contribution of the vortex shedding to the drag while maintaining a close to zero RMS of the lift coefficient.

Figure 7 compares the contours of vorticity (ω_z) behind the cylinder for the uncontrolled (baseline) and controlled cases after reaching a quasi-stationary condition (at $T = 200$). The controller has effectively eliminated the vortex shedding. Additionally, Fig. 8 compares the velocity magnitude contours of the controlled case with the symmetry base flow. The size and the shape of the recirculation region (shown with

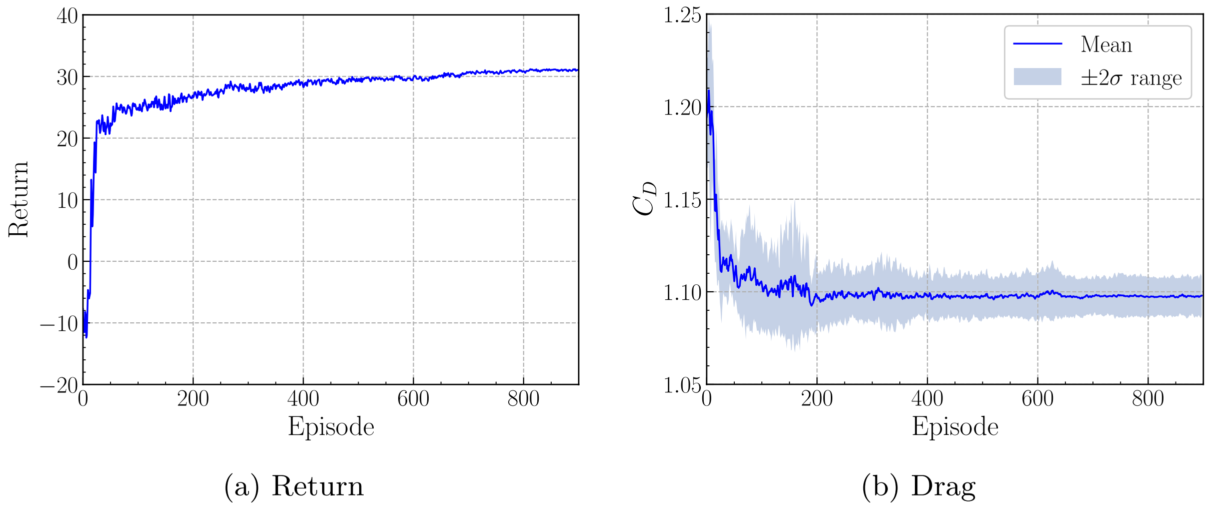


Fig. 11 History of **a** episodes' return (undiscounted sum of rewards) and **b** drag coefficient variation throughout the training simulation

a gold contour line) suggest that the controller has been able to provide a similar regime to the ideally stable flow field. This similarity implies achieving the maximum possible drag reduction.

The robustness of the trained controller agent is evaluated by assessing its performance across various flow conditions. In this study, an uncontrolled flow field is initialized at $T = 0$, beginning with an ideally stable state derived from the steady symmetry condition. Subsequently, the controller is activated at different time instances, i.e., $T = 0$, 85, 90, 100, 110, 120, and 130. Notably, all of these activation times precede the attainment of a saturated quasi-stationary state in vortex shedding, which seems to occur around $T = 140$. It is important to note that during the training phase, the agent was exposed to episodes initiated from the fully saturated shedding. Thus, the initial flow conditions encountered by the controller in this robustness analysis were not included during training. Nevertheless, as depicted in Fig. 9, the controller effectively stabilizes these conditions, ultimately reaching fully stable states for all starting times, underscoring its robustness.

3.2 Vortex shedding behind a confined 3D cylinder

The second investigated test case involves a circular cylinder symmetrically placed within a planar channel. The configuration of the geometry is presented in Fig. 10. The cylinder blockage ratio is $D/H = 1/5$, where H is the height of the channel and D is the diameter of the cylinder. The channel's spanwise depth is $W = 8D$. The channel inlet is $12.5D$ upstream of the cylinder, whereas the outlet is at $35D$ downstream. The flow configuration is chosen based on the previously published studies [35, 36].

The inlet velocity distribution follows the fully developed laminar channel flow (Posiseuille flow) which has the parabolic profile of

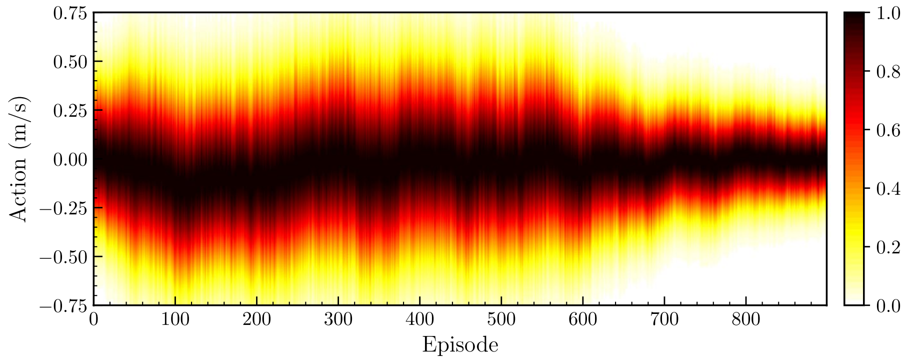


Fig. 12 History of normalized 2D PDF of action values during the training simulation

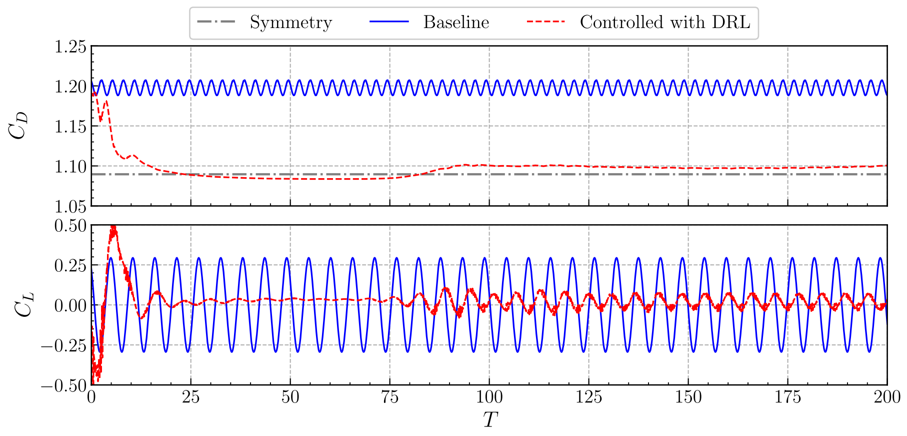


Fig. 13 Variation of drag and lift coefficients for the uncontrolled (baseline) and DRL controlled cases. The controlling mechanism starts at $T = 0$

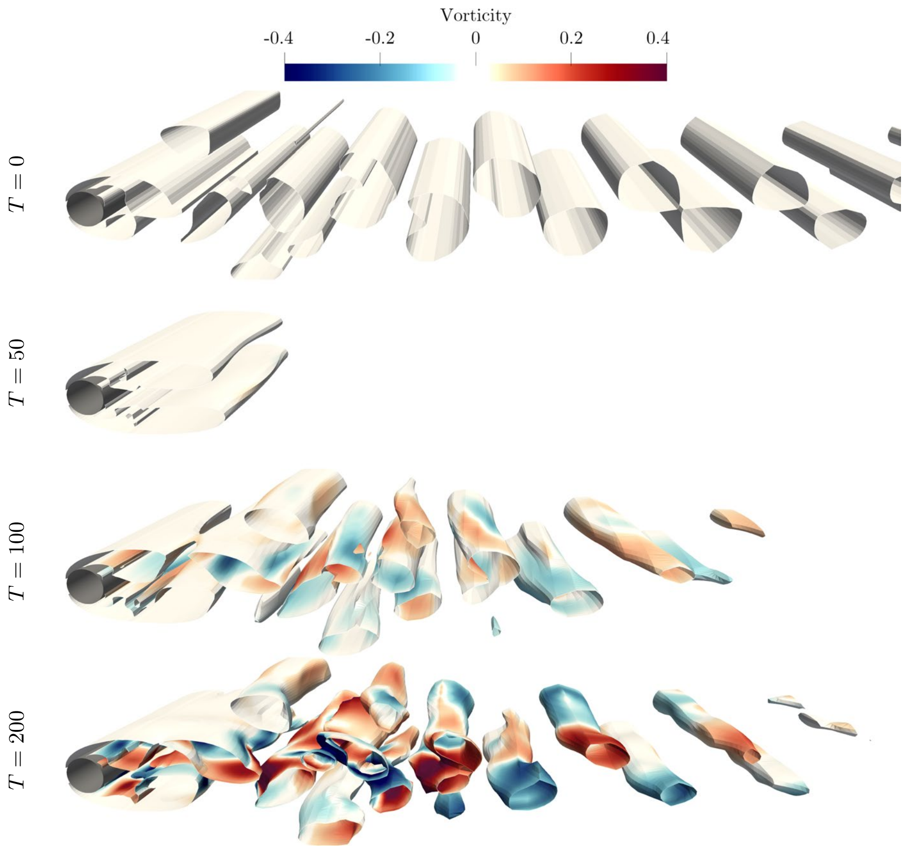


Fig. 14 Illustration of vertical structures of the DRL-controlled case through λ_2 iso-surfaces at different times. The iso-surfaces are colored by the streamwise vorticity (ω_x), indicating three-dimensional effects

$$U_{\text{inlet}} = U_c \left[1 - \left(\frac{y}{H/2} \right)^2 \right], \tag{4}$$

where U_c is the centerline inlet velocity. The Reynolds number, based on centerline velocity and cylinder diameter, is $\mathrm{Re} = U_c D/\nu = 150$. The flow field in this regime is known to remain two-dimensional, meaning no gradient exists in the spanwise direction. The transition to three-dimensional flow is reported to occur within the interval of $180 < \mathrm{Re} < 210$ [35–37].

The outlet boundary condition is considered as fixed pressure and velocity is computed using a zero-gradient condition. Following Kanaris et al. [35], a zero-gradient condition was also applied to the side boundaries. They argued that the Neumann condition captures the details of the instabilities more accurately than the periodic condition. For additional information about the test case under consideration, readers are referred to Ref. [35].

3.2.1 *CFD framework and verification*

The numerical aspects of the adopted CFD framework are similar to the previous test case presented in Sect. 3.1.1 and won't be repeated here.

The mesh dependency of the results is studied using seven different mesh resolutions, and the results are validated against the reference data of Kanaris et al. [35]. Data presented in Table 2 indicate a satisfactory convergence with mesh refinement. The prediction of the drag coefficient and Strouhal number is notably accurate, whereas, similar to the 2D case, the lift coefficient exhibits a higher level of error. Despite the finest mesh yielding the most precise results, mesh Level 3, comprising 6.33×10^5 cells, is chosen for the DRL analysis. The decision is made as a compromise between accuracy and efficiency, particularly given the extensive cost of DRL computations requiring hundreds of transient CFD simulations.

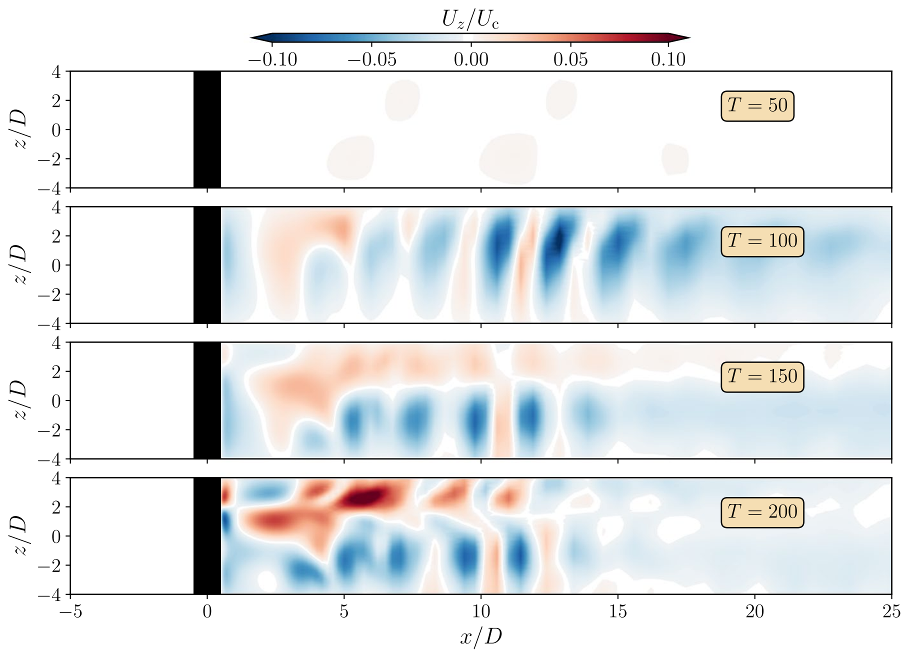


Fig. 15 Time evolution of the spanwise component of velocity, Uz, on the mid-plane between the channel walls

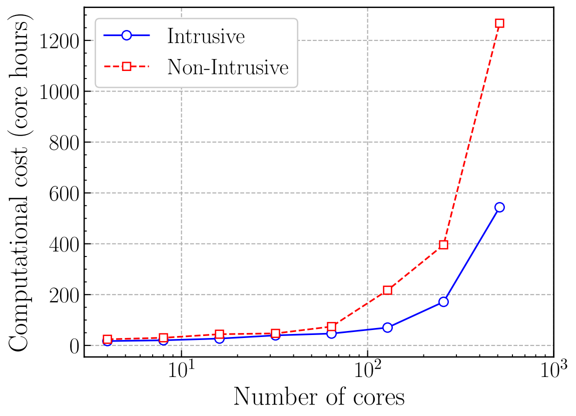


Fig. 16 Computational costs of the current intrusive framework and an I/O-based non-intrusive alternative with the number of cores. The y-axis represents the cost of a single-episode simulation of the 3D confined cylinder in core hours

3.2.2 Configuration of the DRL training

The core elements of the DRL framework remain similar to those in the 2D scenario presented in Sect. 3.1.2, and thus only the differences are elaborated here. As mentioned above, despite the three-dimensional flow configuration, the flow regime is two-dimensional. Thereby, the controller actuators, shown in Fig. 10a, are considered as two synthetic jet slots with uniform velocity in the spanwise direction. The opening angle of the jets is $\theta_{\text{jet}} = 10°$. Similar to the previous test case, the maximum jet flow rate is considered to be 10% of the flow rate encountering the cylinder.

315 pressure probes are utilized to sense the environment state. They are arranged uniformly downstream the cylinder, in an array of $9 \times 5 \times 7$, spanning the range of $0.55 \leq x/D \leq 8$, $-1.25 \leq y/D \leq 1.25$, and $-4 \leq z/D \leq 4$. The probes are not shown in Fig. 10b for simplicity.

The β factor in the reward function was chosen 0.1, demonstrating smoother convergence characteristics. The non-dimensional action time is considered $\Delta T_{\text{action}} = 0.375$ to ensure 15 actions at each shedding period, while the ramping period is $\Delta T_{\text{ramp}} = 0.15$. The time units are normalized by the timescale D/U_c.

3.2.3 Training results

The agent is trained using 180 epochs, with each epoch comprising five episodes, resulting in a total of 900 unsteady CFD simulations.

The history of the undiscounted return, in Fig. 11a, reveals a pronounced increase within the initial 50 episodes (corresponding to the first 10 epochs). Nevertheless, the rate of convergence decreases rapidly, indicating that achieving further enhancements in controller performance necessitates a significantly greater number of episodes. The training process appears to have reached convergence within 800 episodes, with the return values plateauing thereafter.

Similarly, the drag history (Fig. 11b), averaged over the second half of the CFD simulation, shows a sudden drop in the first 50 episodes, followed by a phase of slow convergence. It seems that the drag coefficient reaches a plateau after 400 episodes, with no further decrease. However, the return graph continues to show a gradual increase beyond this point, indicating that the controller primarily struggles with minimizing fluctuations in the lift coefficient (not depicted here for brevity). The normalized 2D PDF of action values, depicted in Fig. 12, shows that the policy maintains a high level of exploration until 600 episodes, after which exploration decreases, shifting towards exploitation to reach convergence. This contrasts with the 2D case (Fig. 5), where action exploration decreases much earlier, highlighting the more challenging nature of the 3D case.

The trained controller is then evaluated in the deterministic mode. The comparison of the drag and lift coefficients between the uncontrolled baseline and controlled scenarios is illustrated in Fig. 13. Both coefficients experience an initial notable reduction due to the implemented controller. However, unlike the previous 2D case study, the drag coefficient begins to rise after $T = 75$, accompanied by the gradual increase in the fluctuations of the lift coefficient, although still maintaining levels lower than those observed in the uncontrolled case.

In order to further assess the imperfect drag reduction, the vortical structures of the controlled flow field are studied. Figure 14 illustrates the λ_2 criterion iso-surfaces colored by the streamwise component of vorticity (ω_x) to signify the streamwise rotation of the flow structures. At $T = 0$, the uncontrolled flow field shows strong yet two-dimensional vortex shedding with near-zero streamwise rotations.

The effectiveness of the controller actions is highest at $T = 50$, at which time the minimum drag and lift coefficients were also observed previously. The vortex shedding is significantly diminished at this time, and the flow field is entirely two-dimensional. However, as time progresses, the flow becomes increasingly unstable and transitions to a three-dimensional state characterized by significant streamwise rotation at $T = 200$. The three-dimensionality of the flow is the direct effect of the implemented actuators, without which the flow would have remained two-dimensional. Nevertheless, it's worth noting that the jet slots, providing uniform injection or suction velocity in the spanwise direction, are not anticipated to effectively control or diminish any three-dimensional effects. Potentially, diminishing the three-dimensionality of the flow field could be achieved through a more sophisticated actuator capable of providing non-uniform jet velocities in the spanwise direction. Exploring such possibilities can be regarded as a subject for future investigation.

The initial indication of transitioning to three-dimensionality is usually evident in the amplification of the spanwise velocity (U_z) [35]. This is investigated in Fig. 15 which displays the contours of the spanwise velocity for the controlled case on the mid-plane between the channel walls. The spanwise velocity remains negligible at $T = 50$, after which significant growth is visible in the U_z contours, indicating increasing three-dimensionality of the flow over time. By $T = 200$, the three-dimensional instability becomes apparent right after the cylinder, triggered by the jet actuation.

3.3 Computational efficiency

The computational efficiency of the developed intrusive algorithm is assessed in this section. To facilitate comparison, an I/O-based non-intrusive DRL–CFD framework was developed, similar to DRLInFluids [15], and its computational cost for one complete DRL episode, with 400 actions, was compared to the intrusive alternative. A series of simulations was performed using parallel processing with different numbers of cores.

As expected, Fig. 16 indicates that the computational cost (core hours) of an episode using the intrusive framework is significantly lower compared to I/O-based non-intrusive algorithms. The cost of the non-intrusive option appears to grow exponentially with the number of cores due to the extensive increase in of I/O operations.

As mentioned in the introduction, the non-intrusive paradigm is not limited to I/O operations, and more efficient non-intrusive alternatives are also proposed in the literature such as direct MPI

Table 3 The main hyperparameters and numerical details of the PPO algorithm and test cases

Parameter	Value	Comment
PPO hyperparameters		
Actor architecture	(64×64)	2 fully connected layers
Critic architecture	(64×64)	2 fully connected layers
Activation function	tanh	Both actor and critic
Discount factor	0.99	–
Optimizer	Adam	Both actor and critic
Actor learning rate	5×10^{-4}	–
Critic learning rate	1×10^{-3}	–
Clipping parameter	0.2	–
Batch size	full	–
Cylinder 2D		
Reynolds number	$\mathrm{Re} = U_\infty D/\nu = 100$	Based on inlet velocity
No. epochs	250	–
Parallelized environments	5	–
Total No. episodes	1250	–
CFD time step	0.01	Normalized with D/U_∞
Action time step	0.4	Normalized with D/U_∞
Action ramping period	0.2	Normalized with D/U_∞
No. actions per episode	400	–
No. CPUs per episode	10	–
Total cost (core hours)	~ 6.2 k	–
Cylinder 3D		
Reynolds number	$\mathrm{Re} = UcD/\nu = 150$	Based on centerline velocity
No. epochs	180	–
Parallelized environments	5	–
Total No. episodes	900	–
CFD time step	0.015	Normalized with D/U_c
Action time step	0.375	Normalized with D/U_c
Action ramping period	0.15	Normalized with D/U_c
No. actions per episode	400	–
No. CPUs per episode	64	–
Total cost (core hours)	~ 50 k	–

communications (e.g., [16, 17]). Evaluating and benchmarking the efficiency of different non-intrusive and intrusive coupling DRL–CFD methods would yield valuable insights. However, this task is beyond the scope of the current article and is considered for future research.

4 Conclusion

An efficient intrusive TensorFlow-based DRL–CFD framework was introduced in this study. The idea was to integrate the DRL agent within the CFD solver, rather than having an external DRL module that needs to communicate with the CFD environment. The CFD computations were performed using the open-source CFD solver OpenFOAM. The framework was parallelized using Python's MPI implementation, mpi4py, to handle the simultaneous calculation of computationally intensive environments through distributed computing on the clusters. As a proof of concept, the DRL agent was integrated within an OpenFOAM boundary condition that acts as a jet actuator for performing active flow control within a fluid flow domain. The inherent randomness of the agent's policy in the training mode poses challenges in the pressure–velocity coupling algorithm and parallel processing, which need to be carefully addressed.

The performance of the developed framework was examined by controlling vortex shedding through drag reduction of two and three-dimensional flow configuration. In both cases, the agent was trained through the PPO algorithm considering two DNNs for the policy and value models.

The 2D scenario demonstrated smooth training convergence, with the trained controller achieving optimal drag reduction by completely eliminating the vortex shedding contribution to the drag. However, the 3D vortex shedding presented a more complex challenge. Although the geometry of the flow configuration was 3D, the uncontrolled flow field remained two-dimensional with no gradients in the spanwise direction. Nevertheless, the implementation of jet actuation induced three-dimensional instabilities. Consequently, the controller, which was designed based on the two-dimensional flow assumption, proved suboptimal for controlling the induced three-dimensional flow dynamics. The efficiency of the developed intrusive framework was compared to an I/O-based non-intrusive alternative that revealed remarkable efficiency improvement. Evaluating and benchmarking the efficiency of other non-intrusive alternatives is regarded as a future work.

Acknowledgements The research presented was carried out as a part of the "Swedish Centre for Sustainable Hydropower - SVC". SVC has been established by the Swedish Energy Agency, Energiforsk and Svenska kraftnät together with Luleå University of Technology, Uppsala University, KTH Royal Institute of Technology, Chalmers University of Technology, Karlstad University, Umeå University and Lund University, svc.energiforsk.se.

The computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at NSC and C3SE partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

Finally, the supports from the organizing committee of the 18th OpenFOAM Workshop is greatly appreciated.

Funding Open access funding provided by Chalmers University of Technology. The funding was provided by the "Swedish Centre for Sustainable Hydropower - SVC". SVC has been established by the Swedish Energy Agency, Energiforsk and Svenska kraftnät together with Luleå University of Technology, Uppsala University, KTH Royal Institute of Technology, Chalmers University of Technology, Karlstad University, Umeå University and Lund University,svc.energiforsk.se.

Availability of data and materials The developed framework and the presented case studies are found open-source at the GitHub repository: https://github.com/salehisaeed/TensorforceFoam.

Declarations

Ethical approval Not applicable.

Appendix: Simulations details

The main hyperparameters of the employed PPO algorithm as well as the numerical parameters of both verification test cases are presented in Table 3.

References

1. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G, Petersen S, Beattie C, Sadik A, Antonoglou I, King H, Kumaran D, Wierstra D, Legg S, Hassabis D (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533. https://doi.org/10.1038/nature14236
2. Silver D, Schrittwieser J, Simonyan K, Antonoglou Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y, Lillicrap T, Hui F, Sifre L, Driessche G, Graepel T, Hassabis D (2017) Mastering the game of Go without human knowledge, vol 550. Nature Publishing Group. https://doi.org/10.1038/nature24270
3. Reddy G, Celani A, Sejnowski TJ, Vergassola M (2016) Learning to soar in turbulent environments. Proc Natl

Acad Sci USA 113(33):4877–4884. https://doi.org/10.1073/pnas.1606075113

4. Verma S, Novati G, Koumoutsakos P (2018) Efficient collective swimming by harnessing vortices through deep reinforcement learning. Proc Natl Acad Sci USA 115(23):5849–5854. https://doi.org/10.1073/pnas.1800923115
5. Ma P, Tian Y, Pan Z, Ren B, Manocha D (2018) Fluid directed rigid body control using deep reinforcement learning. ACM Trans Graph 37(4):1–11. https://doi.org/10.1145/3197517.3201334
6. Lee XY, Balu A, Stoecklein D, Ganapathysubramanian B, Sarkar S (2018) Flow shape design for microfluidic devices using deep reinforcement learning. CoRR arXiv:1811.12444
7. Viquerat J, Rabault J, Kuhnle A, Ghraieb H, Larcher A, Hachem E (2021) Direct shape optimization through deep reinforcement learning. J Comput Phys. https://doi.org/10.1016/j.jcp.2020.110080
8. Rabault J, Kuchta M, Jensen A, Réglade U, Cerardi N (2019) Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. J Fluid Mech 865:281–302. https://doi.org/10.1017/jfm.2019.62
9. Li J, Zhang M (2022) Reinforcement-learning-based control of confined cylinder wakes with stability analyses. J Fluid Mech 932:44. https://doi.org/10.1017/jfm.2021.1045
10. Xu H, Zhang W, Deng J, Rabault J (2020) Active flow control with rotating cylinders by an artificial neural network trained by deep reinforcement learning. J Hydrodyn 32(2):254–258. https://doi.org/10.1007/s42241-020-0027-z
11. Tokarev M, Palkin E, Mullyadzhanov R (2020) Deep reinforcement learning control of cylinder flow using rotary oscillations at low Reynolds number. Energies 13(22):1–11. https://doi.org/10.3390/en13225920
12. Beintema G, Corbetta A, Biferale L, Toschi F (2020) Controlling Rayleigh–Bénard convection via reinforcement learning. J Turbul 21(9–10):585–605. https://doi.org/10.1080/14685248.2020.1797059
13. Wang Y-Z, Mei Y-F, Aubry N, Chen Z, Wu P, Wu W-T (2022) Deep reinforcement learning based synthetic jet control on disturbed flow over airfoil. Phys Fluids 34(3):033606. https://doi.org/10.1063/5.0080922
14. Fan D, Yang L, Wang Z, Triantafyllou MS, Karniadakis GE (2020) Reinforcement learning for bluff body active flow control in experiments and simulations. Proc Natl Acad Sci USA 117(42):26091–26098. https://doi.org/10.1073/pnas.2004939117
15. Wang Q, Yan L, Hu G, Li C, Xiao Y, Xiong H, Rabault J, Noack BR (2022) Drlinfluids: an open-source python platform of coupling deep reinforcement learning and openfoam. Phys Fluids 34(8):081801. https://doi.org/10.1063/5.0103113
16. Kurz M, Offenhäuser P, Viola D, Resch M, Beck A (2022) Relexi—a scalable open source reinforcement learning framework for high-performance computing. Soft Impacts 14:100422. https://doi.org/10.1016/j.simpa.2022.100422
17. Shams M, Elsheikh AH (2023) Gym-precice: reinforcement learning environments for active flow control. SoftwareX 23:101446. https://doi.org/10.1016/j.softx.2023.101446
18. Guastoni L, Rabault J, Schlatter P, Azizpour H, Vinuesa R (2023) Deep reinforcement learning for turbulent drag reduction in channel flows. Eur Phys J E 46(4):27. https://doi.org/10.1140/epje/s10189-023-00285-8
19. Partee S, Ellis M, Rigazzi A, Shao AE, Bachman S, Marques G, Robbins B (2022) Using machine learning at scale in numerical simulations with SmartSim: an application to ocean climate modeling. J Comput Sci 62:101707. https://doi.org/10.1016/j.jocs.2022.101707
20. Chourdakis G, Davis K, Rodenberg B, Schulte M, Simonis F, Uekermann B, Abrams G, Bungartz H, Cheung Yau L, Desai I, Eder K, Hertrich R, Lindner F, Rusch A, Sashko D, Schneider D, Totounferoush A, Volland D, Vollmer P, Koseomur O (2022) preCICE v2: a sustainable and user-friendly coupling library [version 2; peer review: 2 approved]. Open Research Europe. https://doi.org/10.12688/openreseurope.14445.2
21. Mao Y, Zhong S, Yin H (2023) Drlfluent: a distributed co-simulation framework coupling deep reinforcement learning with ansys-fluent on high-performance computing systems. J Comput Sci 74:102171. https://doi.org/10.1016/j.jocs.2023.102171
22. Weiner A (2024) drlFoam. GitHub repository. https://github.com/OFDataCommittee/drlfoam
23. Kuhnle A, Schaarschmidt M, Fricke K (2017) Tensorforce: a TensorFlow library for applied reinforcement learning. GitHub repository. https://github.com/tensorforce/tensorforce
24. Dalcin L, Fang Y-LL (2021) mpi4py: Status update after 12 years of development. Comput Sci Eng 23(4):47–54. https://doi.org/10.1109/MCSE.2021.3083216
25. Sutton RS, Barto AG (2018) Reinforcement learning: an introduction. MIT Press, Cambridge
26. Izquierdo S (2019) CppFlow: Run TensorFlow models in C++ without installation and without Bazel. GitHub repository. https://doi.org/10.5281/zenodo.7107618
27. Rabault J, Kuhnle A (2019) Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. Phys Fluids 10(1063/1):5116415
28. Jasak H (1996) Error analysis and estimation for the finite volume method with applications to fluid flows. PhD thesis, Imperial College London
29. Patankar SV, Spalding DB (1972) A calculation procedure for heat, mass and momentum transfer in three-dimensional parabolic flows. Int J Heat Mass Transf 15(10):1787–1806. https://doi.org/10.1016/0017-9310(72)90054-3
30. Issa RI (1986) Solution of the implicitly discretised fluid flow equations by operator-splitting. J Comput Phys 62(1):40–65. https://doi.org/10.1016/0021-9991(86)90099-9
31. Muddada S, Patnaik BSV (2010) An active flow control strategy for the suppression of vortex structures behind a circular cylinder. Eur J Mech B Fluids 29(2):93–104. https://doi.org/10.1016/j.euromechflu.2009.11.002
32. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347

33. Protas B, Wesfreid JE (2002) Drag force in the open-loop control of the cylinder wake in the laminar regime. Phys Fluids 14(2):810–826. https://doi.org/10.1063/1.1432695
34. Bergmann M, Cordier L, Brancher J-P (2005) Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model. Phys Fluids 17(9):097101. https://doi.org/10.1063/1.2033624
35. Kanaris N, Grigoriadis D, Kassinos S (2011) Three dimensional flow around a circular cylinder confined in a plane channel. Phys Fluids 23(6):064106. https://doi.org/10.1063/1.3599703
36. Camarri S, Giannetti F (2010) Effect of confinement on three-dimensional stability in the wake of a circular cylinder. J Fluid Mech 642:477–487. https://doi.org/10.1017/S0022112009992345
37. Barkley D, Henderson RD (1996) Three-dimensional floquet stability analysis of the wake of a circular cylinder. J Fluid Mech 322:215–241. https://doi.org/10.1017/S0022112096002777