Contents lists available at ScienceDirect

Computers and Fluids

journal homepage: www.elsevier.com/locate/compfluid

Invariant control strategies for active flow control using graph neural networks☆

Marius Kurz [a,1], Rohan Kaushik [b,*,1], Marcel Blind [b], Patrick Kopper [b], Anna Schwarz [b], Felix Rodach [b], Andrea Beck [b]

[a] Centrum Wiskunde & Informatica (CWI), Science Park 123, Amsterdam, 1098 XG, The Netherlands
[b] Institute of Aerodynamics and Gas Dynamics, University of Stuttgart, Wankelstraße 3, 70563, Stuttgart, Germany

ARTICLE INFO

Dataset link: GPLv3, https://github.com/flexi-framework/relexi, https://github.com/flexi-framework/flexi, https://github.com/flexi-framework/ flexi-extensions/tree/smartsim, https://github.com/m-kurz/gcnn

Keywords:
Active flow control
Machine learning
Graph neural networks
Reinforcement learning

ABSTRACT

Reinforcement learning (RL) has recently gained traction for active flow control tasks, with initial applications exploring drag mitigation via flow field augmentation around a two-dimensional cylinder. RL has since been extended to more complex turbulent flows and has shown significant potential in learning complex control strategies. However, such applications remain computationally challenging owing to its sample inefficiency and associated simulation costs. This fact is worsened by the lack of generalization capabilities of these trained policy networks, often being implicitly tied to the input configurations of their training conditions. In this work, we propose the use of graph neural networks (GNNs) to address this particular limitation, effectively increasing the range of applicability and getting more *value* out of the upfront RL training cost. GNNs can naturally process unstructured, three-dimensional flow data, preserving spatial relationships without the constraints of a Cartesian grid. Additionally, they incorporate rotational, reflectional, and permutation invariance into the learned control policies, thus improving generalization and thereby removing the shortcomings of commonly used convolutional neural networks (CNNs) or multilayer perceptron (MLP) architectures. To demonstrate the effectiveness of this approach, we revisit the well-established two-dimensional cylinder benchmark problem for active flow control. The RL training is implemented using Relexi, a high-performance RL framework, with flow simulations conducted in parallel using the high-order discontinuous Galerkin framework FLEXI. Our results show that GNN-based control policies achieve comparable performance to existing methods while benefiting from improved generalization properties. This work establishes GNNs as a promising architecture for RL-based flow control and highlights the capabilities of Relexi and FLEXI for large-scale RL applications in fluid dynamics.

1. Introduction

Reinforcement learning (RL) has gained increasing attention for the task of active flow control (AFC) [1,2]. One of the first applications of RL for AFC was reported by Rabault et al. [3], who employed RL to control the flow around a two-dimensional cylinder using a pair of inflow and suction jets at the cylinder's poles. Since then, this case has established itself as a well-known benchmark problem in the field, having been extensively studied in the literature [3–7] and serving as inspiration for a plethora of related problems (e.g. [8,9]). More recently, RL has been applied to the more complex task of controlling fully three-dimensional turbulent flow, for instance turbulent channel flow, to either reduce drag through suction and blowing at one of the walls [10,11] or control the wall cycle using streamwise traveling waves [12]. Other applications entail the control of turbulent flow around a three-dimensional cylinder [13] or the control of a separation bubble in separated flow [14].

One limiting factor for such applications is the computationally intensive nature of turbulent flow simulations, which poses significant challenges for the application of RL to these cases. This has been somewhat addressed by frameworks leveraging the efficiency of established flow solvers and distributed computing resources to accelerate training [15–17]. Unfortunately, RL is oftentimes found to be sample-inefficient in practice [18], thereby requiring a considerable amount of simulations to find a successful control policy.

☆ This article is part of a Special issue entitled: 'fusing data and physics' published in Computers and Fluids.
* Corresponding author.
E-mail address: rohan.kaushik@iag.uni-stuttgart.de (R. Kaushik).
1 M. Kurz and R. Kaushik contributed equally and cordially agree to share first authorship.

https://doi.org/10.1016/j.compfluid.2025.106854
Received 27 March 2025; Received in revised form 24 August 2025; Accepted 26 September 2025
Available online 30 September 2025

It has been increasingly acknowledged that incorporating invariance and equivariance properties into the employed neural network architectures can be crucial to improving generalizability and training efficiency [1]. These have been successfully incorporated in turbulence modeling and surrogate modeling research over the past few years. Various works utilize GNNs to completely replace the numerical solver and learn the dynamics of different physical systems. Pfaff et al. [19] use an edge-passing GNN to learn to predict the dynamics of numerous physical systems on unstructured meshes, Chen et al. [20] use a similar GNN model to learn the 2D laminar flow across arbitrary shapes and Gao and Jaiman [21] use graph nets to learn the fluid–structure coupling in flow around a elastically mounted cylinder and a hyper-elastic plate. He et al. [22] go in a tangential direction and use graph nets to reconstruct full flow states from incomplete information, while Lino et al. [23] introduce two edge-based graph nets and extend their operation to incorporate information from multiple scales, creating highly accurate and well-generalizing flow predictors in the process. He et al. [22] use GNNs to identify different flow phenomena such as vortices, while Li et al. [24] utilized GNNs to create *neural operators* - a new data-driven approach to solving PDEs. Franco et al. [25] also use graph nets to build flow-predictors, showing generalization across geometries and mesh resolutions, while Gao et al. [26] show an effective incorporation of graph net architecture into physics-informed neural networks applied to PDE systems ranging from linear elasticity to 2D Navier–Stokes equations. Similarly, there are numerous studies employing graph nets for turbulence modeling research [27–31].

Despite the great potential of these approaches, all these cited studies employ supervised learning to train their models, and only a few studies have investigated their use in the context of flow control. One example was reported by Vignon et al. [32], who demonstrated that a multi-agent reinforcement learning (MARL) approach with a translationally invariant control law improves the performance of RL-enabled control significantly for the three-dimensional Rayleigh–Bérnard problem. Similarly, Peitz et al. [33] showed that a distributed, translationally invariant control law can reduce the complexity of the control problem and improve the transferability across different problem sizes for a range of dynamical systems. While they used a standard convolutional neural network (CNN) architecture, Jeon et al. [34] employed a group-invariant CNN architecture for the Rayleigh–Bérnard problem, demonstrating improved performance and faster convergence of the RL agent.

Thus, in light of the highlighted compute cost and the limited generalization of traditional architectures, in this work we propose the use of graph neural networks (GNNs) for AFC to address these issues. The improved generalizability of GNNs at comparable compute overhead effectively allows one to extract more value out of the upfront RL training cost. GNNs can handle pointwise, unstructured data straightforwardly. This allows retaining the spatial structure of the flow field, while not restricting the probes to be distributed in a Cartesian grid — as would be required for the commonly used CNN architectures. GNNs also allow embedding rotational and reflectional equivariance into the network architecture, which enables learning control laws that are invariant under these transformations. Moreover, GNNs by design enforce permutation invariance into the control law, such that the control law becomes independent of the specific ordering of the input features. Thus, we revisit the two-dimensional cylinder flow benchmark problem [3]. The RL training is implemented using Relexi [17], which is an efficient RL framework for high-performance computing (HPC) that has already been applied to several applications in turbulence modeling [31,35,36]. Here, the training environments are provided by running multiple flow simulations with the high-order accurate FLEXI solver [37] in parallel. Within this framework, we apply GNNs to the task of learning invariant control laws for AFC. To the best of our knowledge, this is the first work employing GNNs for an application in AFC.

This paper is organized as follows. In Section 2, we introduce the methodology, outlining the simulation setup, the network architecture, and the training procedure using RL. In this work, we will learn control strategies for the described cylinder flow with both an MLP and a GNN as an agent. The MLP serves to validate our approach against existing literature, accounting for RL's stochasticity in AFC, and provides a baseline for comparison with the GNN. Notably, MLPs lack the invariances and equivariances expected in GNNs. The results of the training are reported in Section 3, where the MLP performance is first validated against results from literature, followed by a performance comparison with the GNNs. Further demonstrated are the proposed invariance properties of the GNNs. Finally, Section 4 concludes the paper with a summary and discussion of its key findings.

2. Methodology

2.1. Simulation environment

The simulation setup employed here follows the original setup proposed by Rabault et al. [3] based on the “2D-2” testcase in [38]. It describes the flow around a two-dimensional cylinder immersed in a channel with height $H = 4.1D$ and length $L = 22D$ as depicted in Fig. 1. The cylinder is offset from the channel centerline by 5% of the cylinder diameter $D = 1$ m to facilitate the onset of vortex shedding. The Reynolds number is chosen as $\mathrm{Re}_D = DU_b/\nu = 100$ with respect to D and the bulk velocity $U_b = 1$ m/s. A parabolic velocity profile is imposed at the inflow boundary via a Dirichlet boundary condition and follows the form

$$u(x=0,y) = U_b \frac{6y(H-y)}{H^2}. \tag{1}$$

The upper and lower walls of the channel are modeled as adiabatic no-slip walls and the outflow condition is prescribed via a constant pressure outlet as introduced in [39].

The immersed cylinder surface is modeled as an adiabatic wall, identical to the upper and lower channel walls. For the controlled case, two synthetic jets with angular width ω are placed at the top and bottom of the cylinder. These jets are implemented by imposing a non-zero normal velocity component at the cylinder wall $u_\perp$, while the remaining components of the state vector are computed analogously to an adiabatic no-slip wall. The velocity profile of each jet follows a cosine distribution that reaches the peak velocity at the jets' center and vanishes at its bounds at $\pm\frac{\omega}{2}$. This yields the normal velocity component at the cylinder wall as

$$u_\perp(\varphi) = \begin{cases} Q_i \frac{\pi}{2\omega D^2} \cos\left(\frac{\pi}{\omega}(\varphi-\varphi_i)\right) & \text{if } |\varphi-\varphi_i| \le \frac{\omega}{2} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where φ denotes the radial coordinate on the cylinder surface with respect to its center. Here, $\varphi_i = \pm\pi/2$ denotes the locations of the two jets with $i = 1, 2$, and Q_i is the scalar controlling the jet strength of the respective jet. Throughout this work, we enforce a net-zero mass flux by setting the sum of the jet strengths to zero, which finally yields a single scalar quantity

$$\hat{Q} = Q_1 = -Q_2, \tag{3}$$

to control both jets. The reader is referred to [3] for further details on the test case setup.

The domain is discretized using a high-order, block-structured mesh with 150 quadrilateral, curved elements using a fourth-order polynomial representation of the geometry. The simulation is then performed using a polynomial order of $N = 4$ with a Discontinuous Galerkin (DG) method on collocated Legendre–Gauss interpolation and integration points [37]. This results in a total of 3750 degrees of freedom per solution variable. In order to simulate the incompressible flow with the compressible flow solver FLEXI [37], a Mach number has to be imposed

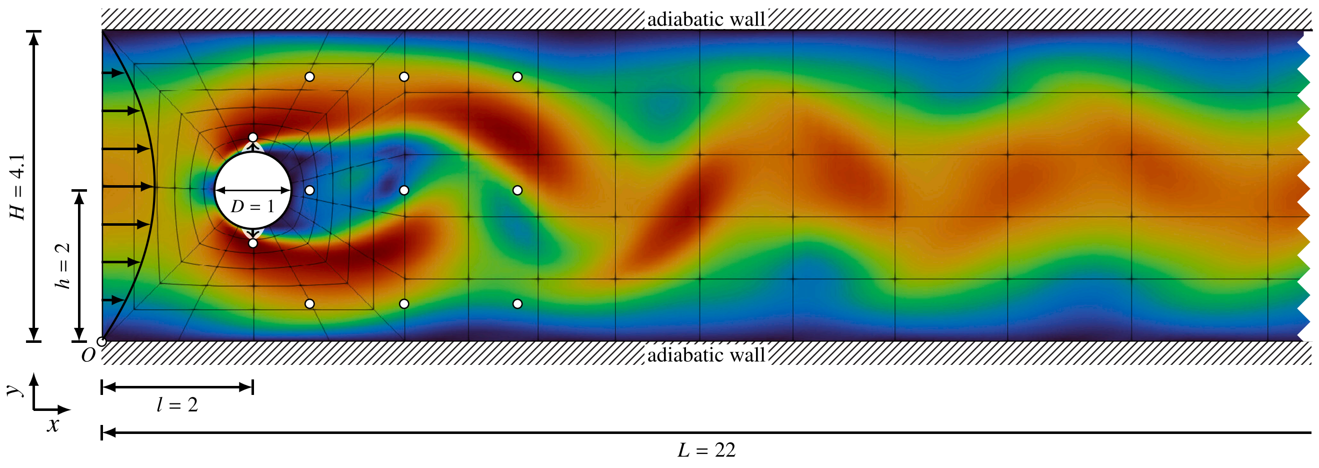

Fig. 1. Simulation setup for the flow around a two-dimensional cylinder in a channel. The positions of the 11 pressure probes are highlighted by white circles and the jets by white areas at the top and bottom of the cylinder. The field solution shows the velocity magnitude. The domain is clipped here to a length of $y = 16$ for better visual representation.

to recover the background pressure and thus a fully compressible flow state. For the present study, the Mach number is set to $\mathrm{Ma} = 0.2$. A validation of the present simulation setup against reference results from the literature with a more in-depth discussion of compressibility effects is given in Appendix A.

Common metrics of evaluation are the drag, lift, and total force coefficients, defined as

$$C_D = \frac{F_x}{U_b^2 D\rho/2}, \quad C_L = \frac{F_y}{U_b^2 D\rho/2}, \quad C_F = \frac{\sqrt{F_x^2 + F_y^2}}{U_b^2 D\rho/2}, \tag{4}$$

respectively. Here, F_x and F_y denote the forces acting on the cylinder in x- and y-direction, respectively, and the density $\rho = 1$. Similarly, we report all results in the following based on the non-dimensional time $t^* = tU_b/D$.

2.2. Reinforcement learning setup

Unlike other machine learning methods, reinforcement learning seeks to solve a class of sequential decision tasks known as *Markov Decision Processes* (MDP), by interacting with an external environment with no prior information on what a good decision policy might be. Here, an *agent* is embedded in said environment, with the environment itself residing in a state s_t at step t. The agent observes the environment through partial *observations* $o_t = o_t(s_t)$ and suggests an *action* a_t based on these observations. For each action, the environment transitions into a new state s_{t+1} and the agents receives a scalar *reward* r_{t+1} based on this transition. The reward is computed using a reward function $r_{t+1} = R(s_t, a_t, s_{t+1})$ based on how the suggested action affected the evolution of the environment's state.

Deep Reinforcement Learning (DRL) refers to the branch of RL where the agent is represented by a deep neural network. While numerous training strategies for DRL tasks exist [40–43], we employ the Proximal Policy Optimization (PPO) [40] algorithm, for two major reasons. First, PPO is mathematically and computationally simpler than other DRL methods while being robust and yielding competitive performance for a wide range of problems. Second, PPO is well established in the general RL literature and for AFC applications, and has standardized and widely tested implementations readily available for use.

The PPO method is episodic, meaning that it waits for the environment to reach a terminal state before beginning to process all the data and perform learning. This means that it only learns to optimize the policy within these episode durations. Therefore, the episode length has to be chosen sufficiently long to allow the agent to learn policies that yield stable and favorable results also for longer time scales. PPO also requires training two networks in tandem, a *policy/actor* network and a *critic/value* network. Both take as input the observations o_t. The actor network learns the control policy, and its outputs are the actions that are passed to the environment. The critic network learns to predict the *discounted sum of future rewards* for a state. This value is used in the policy loss computation, as a measure to reduce its variance and improve learning stability. For further details, the reader is referred to standard RL resources such as [40,44] since these are not the focus of the present work.

For this study, we use the simulation environment as described in Section 2.1 as the environment for the agent to interact with. For this, we define the MDP as follows.

Observations o_t

The full state of the environment s_t would correspond to the full flow field of the simulation entailing the solution at each grid point. In this work, the agent obtains only partial information of the flow field via 11 pressure probes that are dispersed in the domain, as shown in Fig. 1. The policy receives the pressure deviations from the reference pressure at these locations as input, i.e. $\Delta p_i = p(x_{\mathrm{probe},i}) - p_b$ with $i = 1, \dots, 11$. In addition, the GNN policy also receives the normalized coordinates of the probe position $x_i^* = (x_{\mathrm{probe},i} - x_{\mathrm{cylinder,center}})/D$ as input. This is a vector with two entries — the first the wall-parallel component and as second entry the wall-normal component. The reason for this is that the GNN policy requires some form of positional encoding to distinguish between the upper and lower halves of the domain. This requirement is discussed in further detail in Section 2.3. In contrast, MLP policy networks can learn this positional information from the order of the probes in its input vector. However, this characteristic is also one of the MLP's major shortcomings since it then learns to rely explicitly on the ordering of the input vector for this information, unable to function accurately when this is permuted. By formulating the position with respect to the cylinder center and the wall orientation, we ensure the policy remains invariant to the choice of the coordinate system.

Actions a_t

The action is defined identically to the reference work [3]. The agent controls the strength of the two synthetic jets at the top and bottom of the cylinder as described in Section 2.1. By imposing the net-zero mass flux condition given in Eq. (3), the agent only needs to control a single scalar quantity $\hat{Q}$. To ensure that the agent keeps the actuation within a reasonable range, the action prediction of the neural networks is limited using a sigmoid activation function and scaled linearly to the range $\hat{Q} \in [-0.067Q_{\mathrm{ref}}, 0.067Q_{\mathrm{ref}}]$, where Q_{ref} is the mean mass flow across the cylinder diameter based on the inflow condition.

Table 1
Hyperparameters and their values.

Hyperparameter	Value	Description
Ma	0.2	Flow Mach number.
Re	100	Flow Reynolds number.
t_{end}	20	Simulation end time.
Δt_{RL}	0.25	Control update time interval.
$C_{D,\text{min}}$	2.5	Minimum drag coefficient — used for reward scaling.
$\langle C_D\rangle^{\text{no control}}$	2.8	Mean drag coefficient of the uncontrolled case — used for reward scaling.
n_{env}	32	No. of episodes per update.
η^{actor}	Fig. 5	Learning rate for actor.
n^{actor}_{epochs}	40	Max. training epochs for actor per iteration.
$\mathcal{T}^{actor}_{ES}$	0.025	Early stopping threshold for actor.
η^{critic}	0.0005	Learning rate for critic.
n^{critic}_{epochs}	100	Max. training epochs for critic per iteration.
$n^{critic}_{patience}$	5	Early stopping patience value for critic.
σ^{actor}	0.02	Standard deviation of the action distributions.
Neurons	256	No. of neurons in the MLP actor and critic networks.
Layers	2	No. of layers in the MLP actor and critic networks.

Reward r_t

In this work, we modify the reward function proposed by Rabault et al. [3], which is given by

$$r_t = -\langle C_D\rangle - \alpha\left|\langle C_L\rangle\right|, \tag{5}$$

where $\langle\cdot\rangle$ denotes the time-averaged value of the respective quantity and α is a weighting factor that determines how strongly the agent is penalized for introducing a mean lift component. A major drawback of this reward function is that the drag coefficient C_D is always positive and oscillates only slightly around a constant mean value $\langle C_D\rangle \approx 2.8$ for the uncontrolled flow case. This introduces a significant bias towards strongly negative rewards giving cumulative returns per episode of approximately -230 at the start of the training. This is undesirable since it requires the critic to learn to predict such low returns during the first several training epochs. Since the advantage estimates are based on the critic's predictions, this can lead to arbitrary changes and slow convergence of the policy network.

Instead, we propose a modified reward function based on the relative drag reduction compared to the uncontrolled case, similar to the reward function proposed by Cavallazzi et al. [12]. For this, we compute the reward as

$$r_t = \frac{\langle C_D\rangle^{\text{no control}} - \langle C_D\rangle}{\langle C_D\rangle^{\text{no control}} - C_{D,\text{min}}} - \alpha|\langle C_L\rangle|, \tag{6}$$

where, the parameters $\langle C_D\rangle^{\text{no control}}$ and $C_{D,\text{min}}$ are the mean drag coefficient of the uncontrolled case and the expected minimum achievable drag coefficient, respectively. The mean drag and lift, i.e. $\langle C_D\rangle$ and $\langle C_L\rangle$, are computed using an exponential moving average during the simulation to give more weight to the more recent values.

2.3. Policy networks

Two policy network architectures are evaluated in this study, a standard multi-layer perceptron (MLP) and a Graph Convolutional Neural Network (GCNN) [45]. The policy is assumed to be distributed normally. The output of the network is used as the mean of the distribution and its standard deviation σ^{actor} is chosen to be state-independent, and thus a fixed hyperparameter. During training, the actions in each step are drawn from this parametrized Gaussian distribution for generating the trajectories. During evaluation however, the action is sampled greedily from the distribution, which corresponds to selecting the most probable value of the distribution. For a Gaussian distribution, this amounts to selecting the distribution mean. Greedy evaluation thus results in a deterministic policy.

The MLP consists of an initial input layer comprising the concatenated observations received from the environment, followed by two fully-connected hidden layers and a dense output layer as illustrated in Fig. 2. The architecture of the GNN is summarized in Table 2. In a first step, the GNN trunk processes the nodal states to learn an initial latent embedding vector for each node n_i using a fully connected layer that computes

$$\boldsymbol{h}^1_i = \phi^1\left(\underline{\boldsymbol{W}}^1 \boldsymbol{o}_i + \boldsymbol{b}^1\right), \tag{7}$$

where $\boldsymbol{o}_i$ is the input observation vector for node n_i, $\underline{\boldsymbol{W}}^1$ and $\boldsymbol{b}^1$ are the trainable weight matrix and bias vector of the layer, respectively, and ϕ^1 is the nonlinear activation function. In our case, this initial embedding has the dimension $\boldsymbol{h}^1_i \in \mathbb{R}^{16}$. This is followed by two

Table 2
Architecture of the employed GCNN policy for a graph with N nodes. The number of trainable weights and the output dimensions are given for each layer.

Layer	Node-local	Learnable Parameters	Dimension
Input Layer	Yes	–	$(N,3)$
Dense Encoder	Yes	64	$(N,16)$
Message-Passing	No	4352	$(N,256)$
Message-Passing	No	65792	$(N,256)$
Dense Decoder	Yes	4112	$(N,16)$
Graph-Average	No	–	(16)
Dense Output	–	17	(1)
Total	–	74337	–

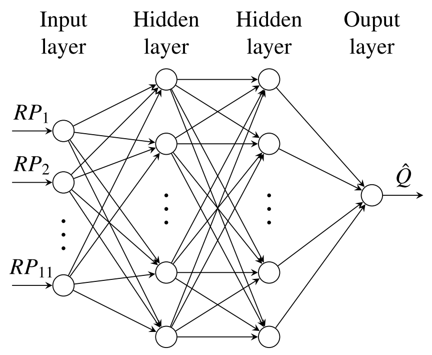

Fig. 2. MLP policy network architecture using 2 hidden layers with 256 neurons each.

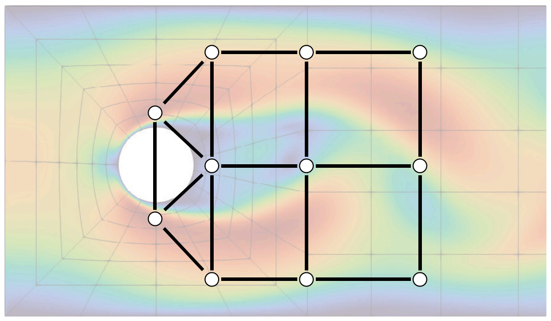

Fig. 3. Pressure probes' connectivity used to build the adjacency matrix for the graph net.

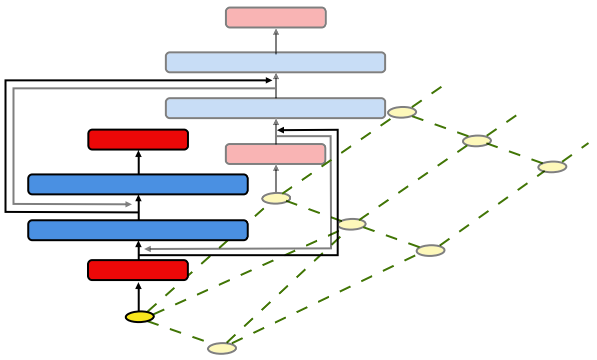
Fig. 4. GNN policy network architecture. The data exchange between a pair of neighbors is demonstrated, where the blue boxes represent the message passing layers and the red boxes are standard dense layers. By virtue of their construction, the weights are shared across the nodes, and hence this architecture can be considered to induce the MARL paradigm.

message passing layers that update the embedding vector based on embeddings of the neighboring nodes following

$$\boldsymbol{h}_i^{l+1} = \phi^l\left[\left(\boldsymbol{h}_i^l + \sum_{j\in\mathcal{N}_i} c_{ij}\boldsymbol{h}_j^l\right)\underline{\boldsymbol{W}}^l + \boldsymbol{b}^l\right]. \tag{8}$$

Here, $\boldsymbol{h}_i^l$ is the input vector at the lth message-passing layer at node n_i, $\mathcal{N}_i = \mathcal{N}(n_i)$ is the neighborhood of node n_i, and c_{ij} are the weights assigned to the connection between node n_i and a node $n_j \in \mathcal{N}_i$. Again, the matrix $\underline{\boldsymbol{W}}^l$ denotes the trainable weight matrix and $\boldsymbol{b}^l$ the bias vector of this layer, with ϕ^l as its nonlinear activation function. The adjacency matrix $\underline{\boldsymbol{A}} \in \mathbb{R}^{N\times N}$ encodes the connectivity information in a graph comprising N nodes, and is defined as

$$A_{ij} = \begin{cases} 1 & \text{if } n_j \text{ is a direct neighbor of } n_i \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

For undirected graphs as in our case, the adjacency matrix is symmetric. A GCNN computes the (non-trainable) weights c_{ij} using the adjacency matrix $\underline{\boldsymbol{A}}$ as detailed by Kipf and Welling [45]. To be clear, the adjacency matrix $\underline{\boldsymbol{A}}$ simply encodes the connectivity between the nodes of the graph, whereas the weights c_{ij} denote the *importance* assigned to these connections. We construct $\underline{\boldsymbol{A}}$ for our application to the cylinder case using the connectivity displayed in Fig. 3. At the end of the message-passing layers, similar to the encoding layer Eq. (7), a dense layer transforms each nodal state into an intermediate-size (16 neurons) representation (see Fig. 4), which is then averaged across the graph. A final dense layer then maps the graph-averaged state to the final output size. Owing to this averaging operation, the GNN is unable to distinguish between flow phenomena in the lower half of its domain and those in the upper half (as noted in Section 2.2). This is especially crucial in our case where the control action itself is directed, and depends on the policy network being able to distinguish between such cases. The MLP does not suffer from this problem because it learns implicit positional information based on the ordering of its input which is also one of its major shortcomings. To remedy this, the flow-parallel cylinder-centered normalized coordinates of the pressure probes $\boldsymbol{x}_i^*$, in addition to the pressure values, are provided to the GNN as inputs.

2.4. Critic networks

Of interest to the present study is the architecture of the critic network. A simple MLP is employed, mirroring the architecture of the MLP policy network, with the concatenated observations as its input, followed by 2 dense layers with 256 neurons and a final dense layer leading to the output of the network. This output is passed through a

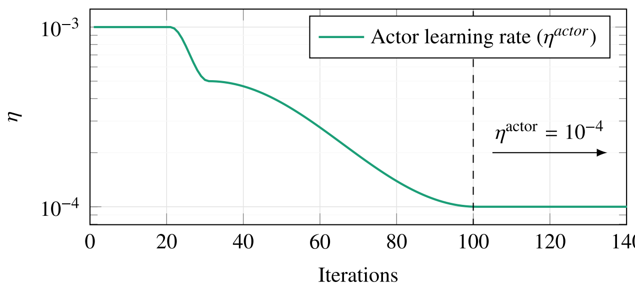

Fig. 5. Actor's learning rate decay throughout training. The learning rate becomes constant after 100 iterations with the last value (i.e. 10^{-4} in this case).

hyperbolic tangent activation to scale it to $[-1, 1]$ (the predicted future rewards can be negative), and then scaled by a single learnable weight. The same critic network architecture is employed in the training of both the MLP and GNN policy networks, to avoid giving either an unfair advantage (or disadvantage).

2.5. Implementation details

The machine learning part of this work is based on Relexi [17] which is an open-source reinforcement learning framework designed for HPC environments. It is written in Python and based on TensorFlow's [46] RL library, TF-Agents [47]. Relexi enables the use of RL for computationally intensive simulations (such as CFD), by coupling HPC simulation codes with the TF-Agents library via the SmartSim package [48] for fast HPC-scalable environments. Relexi automates the workload distribution and management of parallelized simulation instances across allocated HPC resources. The flow solver FLEXI [37] is used to run the simulations. It is a high-order accurate simulation framework designed to solve partial differential equations with a particular emphasis on computational fluid dynamics. It employs the discontinuous Galerkin spectral element method (DGSEM), which facilitates high-order accuracy and supports fully unstructured hexahedral meshes. Developed by the Numerics Research Group (NRG) at the University of Stuttgart's Institute of Aerodynamics and Gasdynamics, FLEXI has demonstrated exceptional scalability — efficiently operating on large-scale applications on over 500000 computing cores [49–51] and recently has also been adapted for GPU systems [52].

Note that unlike other studies, our sampled environments start from different points on the vortex shedding cycle, thus making the learned policy more robust to starting conditions, and providing ample variety of state–action-reward tuples for training. We provide a set of 8 start files for the environments to choose from at the beginning of each trajectory-sampling iteration.

Further, our PPO implementation implements Kullback–Leibler (KL) divergence based early stopping for the policy networks every iteration, which is standard practice in RL research [40,53–55]. The KL divergence *from* distribution P *to* distribution Q is defined as:

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x)\log\frac{p(x)}{q(x)}\,\mathrm{d}x. \tag{10}$$

Essentially, if the KL divergence between the distributions from the beginning of the update iteration to the end of an update epoch exceeds a threshold $\mathcal{T}_{ES}^{actor}$, actor-net training is halted until trajectories are resampled. This helps to limit the policy from venturing too far from the original (i.e. the one used to generate the trajectories) and avoids harming returns during updates, since clipping is known to be insufficient in this regard. Inspired by this and standard DL early stopping, we also implement simple patience-based early stopping for the critic network training, where if the value-function loss does not decrease for $n_{patience}^{critic}$ epochs, critic-net training is halted until trajectories are resampled.

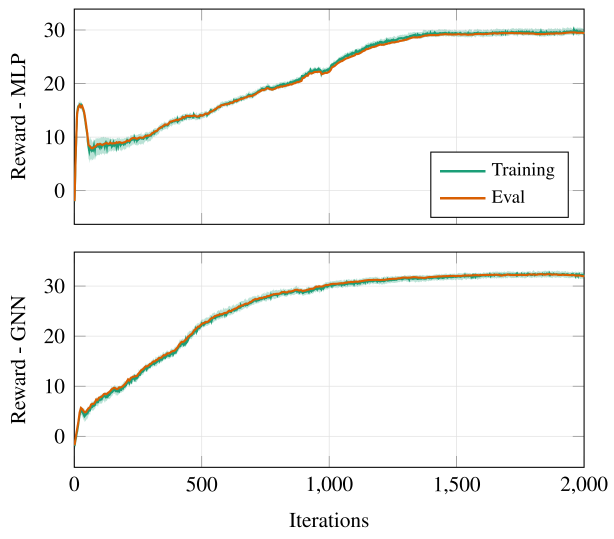

Fig. 6. Evolution of the training and evaluation return during the training for the MLP (*top*) and the GNN (*bottom*) models. As can be observed, the training is robust and converges smoothly to a stable maximum for both cases.

Both these additions help with speeding up training (as unnecessary training epochs are avoided) and preserving performance. Lastly, we use a decaying learning rate (see Fig. 5) for the actor networks to modulate performance, as noted in Engstrom et al. [56]. The specific values for all the relevant hyperparameters used in this study are summarized in Table 1.

3. Results

The following section presents the main findings of this study and is structured as follows. First, the training behavior of the MLP and GNN policies are discussed in Section 3.1 and validated against the reference study by Rabault et al. [3] in Section 3.2. Subsequently, the performance of the trained policies is analyzed in detail in Section 3.3 and the permutation invariance of the GNN policy is verified empirically in Section 3.4. Finally, we discuss the control strategies learned by the policy networks in Section 3.5.

3.1. Training behavior

To investigate the training behavior of the GNN and MLP models, the evolution of the cumulative rewards during the training is shown in Fig. 6. At the beginning of the training, both models still exhibit random initial weights and collect very similar, slightly negative rewards. Starting after this initial state, the training process appears to be very stable for both models, with the cumulative rewards evolving monotonically and converging to a stable maximum after approximately 1500 iterations. Moreover, the variation of the collected rewards across the parallel training environments appears to be negligible in comparison to the overall improvement. This robustness is also reflected by the fact that the greedy evaluation of the policy yields comparable performance to the stochastic training environments, which shows that no overfitting or mismatch between the stochastic training and the greedy evaluation runs occur. Overall, the GNN is observed to learn faster and to reach a higher final return compared to the MLP. We attribute this to the invariant nature of the GNN and its weight-sharing mechanism, which appears to improve the training efficiency per training data sample. Before investigating the behavior of the trained policies in detail, we first validate our overall setup by comparing our results to the reference study by Rabault et al. [3] in the following.

3.2. Validation with reference

First, we validate the performance of the trained MLP policy against the results reported in the original study [3]. To account for the different baseline drag coefficients due to compressibility effects (cf. Appendix A), we compare the drag reduction as follows. Besides the baseline no-control case, we compute a second reference case in which we impose a symmetry condition in the center cylinder plane to suppress the vortex shedding as detailed in Appendix B and also reported in the reference study. To distinguish it from the baseline no-control case, this case is referred to as the 'symmetric' case for the remainder of this study. Based on this, we compute an *improvement factor* r_X for a coefficient C_X as

$$r_X = \frac{\langle C_X\rangle^{\text{no control}} - \langle C_X\rangle^{\text{AFC}}}{\langle C_X\rangle^{\text{no control}} - \langle C_X\rangle^{\text{sym}}} - 1, \tag{11}$$

where $\langle C_X\rangle^{\text{no control}}$, $\langle C_X\rangle^{\text{sym}}$ and $\langle C_X\rangle^{\text{AFC}}$ are the time-averaged coefficients of the baseline case without control, the symmetric case without any vortex shedding and the case with AFC, respectively. Here, $r_X = 0$ is recovered if the AFC case achieves the same reduction as the symmetric case and $r_X > 0$ denotes that the AFC case is able to decrease the coefficient below the one reported for the symmetric case. To elucidate, the improvement factor compares the suppression of C_X through AFC w.r.t. the symmetric case without any vortex shedding — which can be seen as a good approximation to the maximum drag reduction achievable through AFC.

The time-averaged $\langle C_D\rangle$ for our MLP policy was computed by running a simulation with the trained control policy up to $t^* = 100$ and averaging over the interval $t^* \in [50, 100]$ to capture the long-term effects and to omit any influence of the initial transient. The resulting time-averaged drag coefficients of our MLP policy, the baseline case and the symmetric case, and those from the reference study [3] are summarized in Table 3. Based on these values, we also compute the improvement factors r_D for each of the two studies.

Our MLP policy achieves an improvement in drag reduction of $r_D = 9.09\%$ over the symmetric case, which indicates that the learned policy not only suppresses the vortex shedding but finds a more intricate control strategy to even undercut the results of the symmetric case. The reference study also reports a significant reduction in drag by the MLP control law in comparison to the no-control case by reducing the drag coefficient from $\langle C_D\rangle = 3.20$ to $\langle C_D\rangle = 2.99$. However, the MLP there is not able to reduce the drag down to the level of the symmetric case, which is $\langle C_D\rangle = 2.93$, resulting in a $r_D = -22.21\%$.

The MLP trained in the present work thus not only recreates, but surpasses the performance of the equivalent 11-probe case of the original reference. Even more remarkably, this policy is able to achieve this reduction using significantly fewer model parameters, with the reference study employing roughly 300000 learnable parameters while the present work uses only around 75000. Although this direct comparison is not the goal of this study, it nonetheless serves as a validation of our methodology and results. We attribute these improvements to the improved training procedure with the KL-divergence early stopping and the gradually decaying learning rate. However, the number of iterations required to reach convergence is noticeably higher than in the original study. This is likely due to their reuse of experience via a replay buffer, which is not used in this study. Based on these results, we now investigate the performance of the trained MLP and GNN policies in more detail.

3.3. Performance of the trained policies

In a next step, we investigate the performance of the trained MLP and GNN policies in more detail. For this, Fig. 7 shows the temporal evolution of the force coefficients C_L, C_D, and C_F as well as the control output of the policy networks. The statistics are also computed and summarized in Table 4 for a time period $t^* \in [10, 20]$ that lies within the

Table 3
Average drag coefficients $\langle C_D \rangle$ computed in the interval $t^* \in [50, 100]$ compared to the results of the 11 probe case reported by Rabault et al. [3].

	Present work	Reference [3]
No-Control	$\langle C_D \rangle = 2.78$	$\langle C_D \rangle = 3.20$
Symmetric	$\langle C_D \rangle = 2.67$	$\langle C_D \rangle = 2.93$
MLP	$\langle C_D \rangle = 2.66$	$\langle C_D \rangle = 2.99$
r_D	**+9.09%**	−22.21%

Table 4
Time-averaged force coefficients $\langle C_X \rangle$ as well as their amplitudes $\langle C_X{}^{\mathrm{amp}} \rangle$, computed and averaged over a time period $t^* \in [10, 20]$, which lies within the training time.

	MLP	GNN	No control	Symmetric
$\langle C_D \rangle$	2.64	2.64	2.78	2.67
$\langle C_D{}^{\mathrm{amp}} \rangle$	1.51×10^{-2}	1.38×10^{-2}	2.45×10^{-2}	0.0
r_D	+27.3%	+27.3%	––	––
$\langle C_L \rangle$	5.91×10^{-2}	7.27×10^{-3}	0.00	0.00
$\langle C_L{}^{\mathrm{amp}} \rangle$	2.52×10^{-1}	1.35×10^{-1}	7.39×10^{-1}	0.00
$\langle C_F \rangle$	2.64	2.64	2.83	2.67
$\langle C_F{}^{\mathrm{amp}} \rangle$	2.17×10^{-2}	1.74×10^{-2}	7.39×10^{-2}	0.0
r_F	+18.8%	+18.8%	––	––

Table 5
Time-averaged force coefficients $\langle C_X \rangle$ as well as their amplitudes $\langle C_X{}^{\mathrm{amp}} \rangle$, computed and averaged over a time period $t^* \in [50, 100]$ well after the training time in the quasi-steady limit of the controlled flow.

	MLP	GNN	No control	Symmetric
$\langle C_D \rangle$	2.66	2.66	2.78	2.67
$\langle C_D{}^{\mathrm{amp}} \rangle$	1.31×10^{-2}	5.02×10^{-3}	2.45×10^{-2}	0.0
r_D	+9.09%	+9.09%	––	––
$\langle C_L \rangle$	3.80×10^{-2}	7.20×10^{-2}	0.00	0.00
$\langle C_L{}^{\mathrm{amp}} \rangle$	2.07×10^{-1}	2.59×10^{-1}	7.39×10^{-1}	0.00
$\langle C_F \rangle$	2.67	2.67	2.83	2.67
$\langle C_F{}^{\mathrm{amp}} \rangle$	1.36×10^{-2}	1.19×10^{-2}	7.39×10^{-2}	0.0
r_F	0.00%	0.00%	––	––

training interval. Table 5 shows the statistics computed for an extended simulation time $t^* \in [50, 100]$ that allows for a more detailed analysis of the long-term behavior of the policy networks.

Most notably, both the MLP and the GNN policy networks achieve qualitatively and quantitatively similar results. Both models are able to reduce the drag coefficient especially well during the time period up to $t^* = 20$, which is the simulation duration used for training. Similar to the results reported in [3], the drag increases again for longer simulation times and reaches a quasi-steady state at around $t^* \approx 40$ for both models. This robustness of the trained policy is noteworthy as the quasi-steady state differs from the transient behavior observed in the training phase. This emphasizes the effectiveness of DRL in finding good decision strategies.

Besides the mean values, we also report the amplitudes of the oscillations in the lift and drag coefficients denoted by $\langle C_X{}^{\mathrm{amp}} \rangle$. For this, we compute the L^1-norm with respect to the mean value $\langle C_X \rangle$ from the oscillating curves, identify the local peaks and average their resulting values. While both networks achieve the same total reduction in the average drag, the GNN suppresses the oscillations in the drag coefficient significantly more than the MLP. Overall, the GNN also exhibits a lower vortex shedding frequency than the MLP agent. Simultaneously, the GNN tends to predict slightly lower control values, which means it has to invest less control effort to achieve the same drag reduction as the MLP policy. This general trend shifts for the lift force, where the GNN appears to have both a higher mean lift force as well as more pronounced oscillations. Interestingly, the total force acting on the cylinder C_F for both agents matches the symmetric baseline case without vortex shedding nearly perfectly. While both networks are able to reduce the drag below the symmetric case, i.e. they yield $r_D > 0$, this is at the cost of a minor contribution to the mean lift force that matches exactly the absolute drag reduction. This implies that the agent learns to balance the terms resembling the drag reduction and lift penalty in the reward function Eq. (6).

This confirms the initial assumptions that GNNs are suitable network architectures for AFC applications. In a next step, we verify the expected invariance properties of the GNN policy, and the lack thereof in the MLP network.

3.4. Verification of the permutation invariance

To verify the permutation invariance of the GNN policy and investigate the impact of the probe ordering on the MLP policy, we evaluate

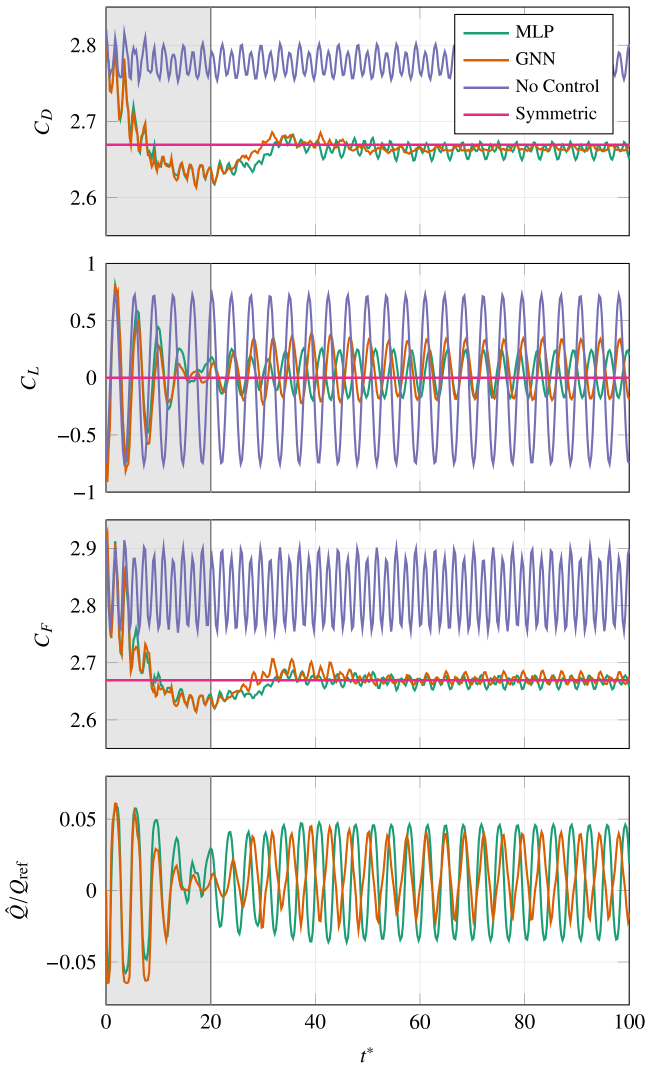

Fig. 7. Long-term evolution of the lift (C_L), drag (C_D) and total force (C_F) coefficients, and the normalized control outputs ($\hat{Q}/Q_{\mathrm{ref}}$) of the agent. The time interval up to $t^* = 20$ used for training is shaded. The no-control and the symmetric cases are shown as baselines.

both policies for the same simulation but with shuffled inputs. This means that the order in which the states of each probe are concatenated is changed in comparison to the original training data. The resulting evolution of the lift and drag coefficients as well as the control output of the MLP and GNN policies are shown in Fig. 8. As expected, the GNN policy is invariant to the ordering of the probes and yields the exact same results for the shuffled input as for the original input ordering.

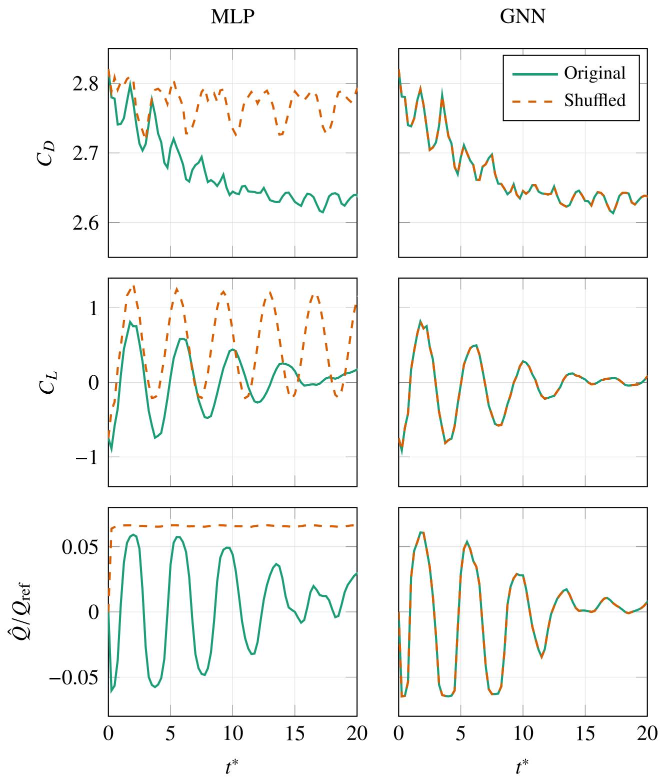

Fig. 8. Time evolution of the lift (C_L) and drag (C_D) coefficients, and the normalized control outputs ($\hat{Q}/Q_{\mathrm{ref}}$) for the MLP (*left*) and GNN (*right*) policy with either shuffled inputs or the original sorting of the probes as used during training.

This demonstrates the permutation invariance of the GNN policy. In contrast, the MLP policy fails to yield any meaningful control output for the shuffled input and just produces the maximum control value during the simulation. This again underlines the potential of GNNs for active flow control tasks, as they do not rely on an explicit ordering or flattening of the input data, but rather operate on the unstructured spatial structure of the data directly.

A key advantage of permutation invariance is that it exhibits a direct connection to more fundamental physical symmetries, such as rotational invariance. This holds because, for scalar inputs, a rotation or reflection of the domain – or of the physical phenomena within it – is inherently captured through reshuffling, which encompasses all these operations as a superset.

While a GNN thus could be applied directly to a rotated or reflected domain without any need for retraining, an MLP would require a reordering of the input data to match the original training data. The GNN policy thus promises to yield more robust and generalizable policies for more complex flow control tasks and to embed symmetries of the underlying physical phenomena directly into the policy.

3.5. Analysis of controlled flow and strategy

Both policy networks end up learning nuanced control strategies, like those reported by Rabault et al. [3],Jiang and Cao [8] and Jia and Xu [57]. And much like these studies, the long-term controlled flow approximates the hypothetical no-shedding baseline flow in terms of the forces on the cylinder. The superlative performance achieved from $t^* \approx 10$ to $t^* = 20$ (the simulation end time in the training phase) proves to be transient, and quickly asymptotes to a quasi-steady state that recovers the overall force C_F of the symmetric case. This could be due to the limited time duration of the simulations used during training and the resulting lack of future information in the latter stages of the training simulations. For the first time steps of the simulations, the *generalized advantage estimate* in the state–action-value tuple contains detailed information of the future and how actions taken at this moment affect the flow. Based on this, the DRL method learns to take optimal steps keeping the effect it has on the future in mind. Towards the end of the episode ($t^* \in [10, 20]$) the agent has limited information of this context, since the simulation is halted at $t_{end} = 20$ in every episode. The agents thus only learn to take actions that have a positive effect on their immediate future, with no regard to what happens after the simulation ends. This performance degradation of finite-horizon controllers in evaluations on longer time horizons is a well recognized issue in the field of optimal-control, and not just a RL issue [44,58].

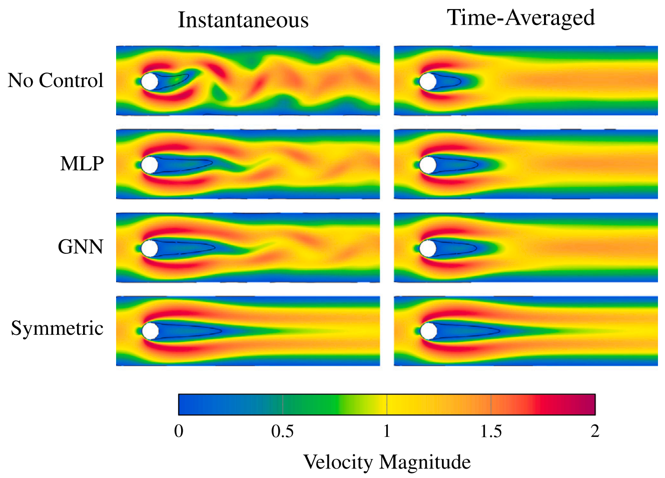

Fig. 9. Instantaneous flow field at $t^* = 20$ (*left*) and time-averaged flow field in $t^* \in [50, 100]$ (*right*) for the no-control case, the GNN policy, the MLP policy, and the symmetric case colored by velocity magnitude. The black lines show the isocontour of vanishing velocity and thus the outline of the recirculation zone.

The instantaneous and time-averaged flow field solutions for the no-control case, the GNN and MLP policies, and the symmetric case are shown in Fig. 9. Both the GNN and MLP policies are able to suppress the vortex shedding resulting in a noticeably longer recirculation bubble than in the uncontrolled case. Interestingly, this recirculation bubble is larger for the instantaneous flow field at $t^* = 20$ than for the time-averaged flow field, which directly reflects the lower drag observed during training (i.e. $t^* \leq 20$) than in the quasi-steady state as discussed prior. Also, both the GNN and MLP policies recover similar drag reduction as the symmetric case while creating only a smaller recirculation zone. This highlights again that the policies are able to find a more intricate control strategy than just suppressing the vortex shedding.

4. Conclusions

This study explored the feasibility of using GNNs for RL-based active flow control with a distinct focus on overcoming limitations associated with standard architectures such as MLPs and CNNs. To the best of our knowledge, this is the first work to employ GNNs for active flow control. The primary objectives were to develop a control strategy that efficiently processes unstructured flow data, while respecting the inherent invariance properties of the underlying physics to improve its robustness and generalization abilities. By revisiting the well-established two-dimensional cylinder benchmark problem, the effectiveness of this approach was assessed and compared against existing methods.

The training setup was validated by training an MLP architecture similar to the original study by Rabault et al. [3], which recovered similar performance as that reported in the reference. The GNN policy network was found to successfully match the performance of this MLP architecture, confirming that GNNs can achieve similar levels of accuracy while offering additional generalization capabilities. Both networks are able to reduce the drag below that observed in the symmetric case, where the vortex shedding is suppressed by imposing a symmetry condition along the centerline of the domain. This indicates that the performance of both policies stems from more than just a reduction of the vortex shedding, and points to the effectiveness of the implemented PPO approach at finding intricate, well-performing strategies. A key advantage of using GNNs over MLPs is the incorporation of permutation invariance into the control policy, possibly improving the model's ability to generalize across different conditions. This property was demonstrated successfully for the GNN, which yields identical results independent of how the input probes are ordered. In contrast, the MLP was found to outright fail if the input ordering is changed. This permutation invariance can be seen as a prerequisite for more general rotational and reflectional invariance, and is thus a distinguishing attribute in favor of the GNN architectures over the MLP. To investigate the performance of the trained policies on longer time-scales, they were evaluated on simulations lasting 5 times as long as those used for training. The learned policies were found to converge to a quasi-steady behavior that matches the reduction in overall force observed in the symmetric case. This work has demonstrated that Relexi is a suitable tool for flow control tasks and the combination with FLEXI as a flow solver positions it to be well-suited for large-scale flow-control applications in fluid dynamics.

CRediT authorship contribution statement

Marius Kurz: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Rohan Kaushik:** Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation. **Marcel Blind:** Writing – review & editing, Writing – original draft, Software, Data curation, Conceptualization. **Patrick Kopper:** Writing – review & editing, Writing – original draft, Validation, Software, Formal analysis, Data curation. **Anna Schwarz:** Writing – review & editing, Writing – original draft, Software, Project administration, Conceptualization. **Felix Rodach:** Writing – review & editing, Software, Methodology, Investigation. **Andrea Beck:** Writing – review & editing, Supervision, Project administration, Methodology, Funding acquisition, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially funded by the European Union. It has received funding from the European High Performance Computing Joint Undertaking (JU) and Sweden, Germany, Spain, Greece, and Denmark under grant agreement No 101093393. Moreover, this research presented was funded by Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy EXC 2075 – 390740016, by the DFG Rebound – 420603919, and in the framework of the research unit FOR 2895. We acknowledge the support by the Stuttgart Center for Simulation Science (SimTech). The authors also thank the Friedrich und Elisabeth Boysen-Stiftung for supporting the work under grant BOY187. Furthermore, the authors gratefully acknowledge the support and the computing time on "HAWK" provided by the HLRS through the project "hpcdg". This work was carried out during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme by M.K.

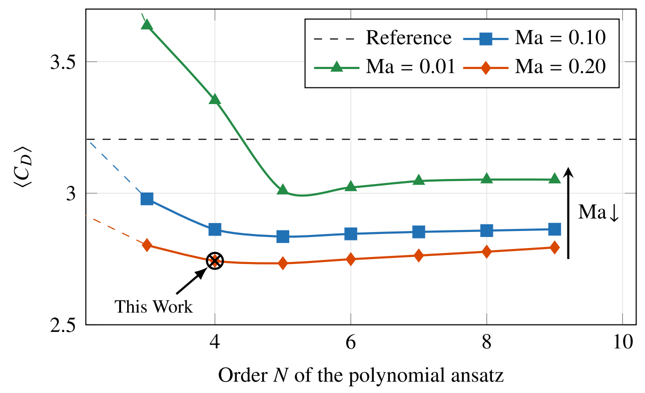

Fig. A.10. Convergence of time-averaged drag coefficient $\langle C_D \rangle$ against order of the polynomial ansatz for different Mach numbers. The results by Rabault et al. [3] serve as reference. The selected setup for this work is highlighted in black.

Table A.6
Mass-flow $\dot{m}$ and drag coefficient $\langle C_D \rangle$ differences from the target values for the $N = 9$ cases.

Mach number Ma	$\Delta\dot{m}$	$\Delta\langle C_D \rangle$
0.2	0.0943	0.1282
0.1	0.0722	0.1066
0.01	0.0366	0.0478

Appendix A. Validation of the simulation setup

This section provides a validation of the baseline simulation setup used in this study. For this, the results obtained with FLEXI for the test case of a two-dimensional cylinder immersed in a box with viscous (no-slip) walls are compared against the Ref. [3]. The reader is referred to Krais et al. [37] for details on the implementation of the discontinuous Galerkin approach and to Hindenlang et al. [59] for the validation of the convergence properties. The resulting average drag coefficient for increasing orders N of the polynomial ansatz function are show in Fig. A.10 for three different Mach numbers $\mathrm{Ma} = \{0.01, 0.1, 0.2\}$. All considered Mach numbers exhibit a marked increase in drag for very low polynomial orders, followed by a local minimum of $\langle C_D \rangle$ at approximately $N \approx 4$ and subsequent convergence. Furthermore, the setup is shown to react sensitively to the free-stream Mach number even as the results by FLEXI converge towards the incompressible reference for decreasing Mach numbers. A plausible explanation of the reduced drag is the interaction of the limited domain size with the weakly enforced Dirichlet boundary conditions. This interplay results in a mass-flow deficit directly related to the $\langle C_D \rangle$ difference, see Table A.6. From these results, the $N = 4$, $\mathrm{Ma} = 0.2$ case (highlighted in Fig. A.10) was chosen as a good compromise between computational effort and accuracy.

Appendix B. Symmetric simulation setup

We also simulate a hypothetical no-vortex-shedding case that serves as an ideal flow case with minimum drag that the AFC should be able to achieve. This allows us to compare the achieved drag reduction with the reference studies in light of the compressibility effects highlighted in Appendix A. The setup follows [3], where the cylinder is placed in the center of the channel and a symmetry condition is imposed at the centerline that forces the normal gradient and the normal component of the velocity vectors to zero. This simulation setup is advanced up to $t^* = 500$ to reach a convergened state. The results reported in Section 3 of the main text are obtained by restarting the simulation from this converged state. The mesh and flow field of the converged simulation are shown in Fig. B.11.

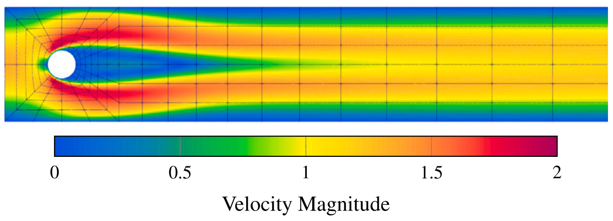

Fig. B.11. Mesh and converged flow field of the symmetric setup. Note the lack of vortex shedding in the wake of the cylinder.

Table C.7
r_D values for different Reynolds numbers. Note that the networks were trained only on the $Re = 100$ case.

Reynolds number Re	MLP	GNN
100	9.09%	9.09%
150	6.28%	−9.37%
200	8.69%	−16.09%
250	−28.95%	−27.28%
300	−32.15%	−31.43%

Appendix C. Generalization and extrapolation

Even though outside the purview of the present study's research objectives, the extrapolation capabilities of the trained networks are tested to see how well these strategum perform. First, the networks are tested on different Reynolds numbers, specifically on $Re \in \{150, 200, 250, 300\}$. This specific range was chosen primarily to see how well the networks perform when operating in a strictly laminar regime versus the so-called intermediate transitional regimes. Lastly, the GNN controller performance is tested with different sensor configurations, a task the **MLP by-design cannot** perform. As a brief refresher, r_D is defined as described in Eq. (11):

$$r_D = \frac{\langle C_D\rangle^{\text{no control}} - \langle C_D\rangle^{\text{AFC}}}{\langle C_D\rangle^{\text{no control}} - \langle C_D\rangle^{\text{sym}}} - 1,$$

where all quantities have been computed in their respective Reynolds number regimes. Note that r_D measures the excess percentage of drag reduction, over and above that achieved in the symmetric case. This implies that even negative values, as long as $r_D > -1$, signal a drag reduction compared to the un-actuated (i.e. *no control*) case, just not as strong as the symmetric case. All the values reported are for the averaged quantities from the $t^* \in [50, 100]$ interval.

C.1. Higher Reynolds numbers

From the values reported in Table C.7 it can be easily seen that both networks extrapolate reasonably well to the tested Reynolds numbers. Please note that the agents have not been re-trained on these new Reynolds numbers, but the original agents (having been trained on $Re = 100$) are now applied to these higher Reynolds number environments. This suggests, as noted in other studies [6,60], that controllers trained through DRL in lower Re regimes with lower simulation costs can be effectively applied to more complex regimes as well. The MLPs seem to extrapolate better than the GNNs in similar flow regimes, i.e. $Re \leq 200$, however both controllers appear to perform similarly well at the beginning of the intermediate transitional regime. Although this tested range is modest, it strengthens our belief in the use of GNNs for this application, and future work will add focus on bridging this observed extrapolation gap.

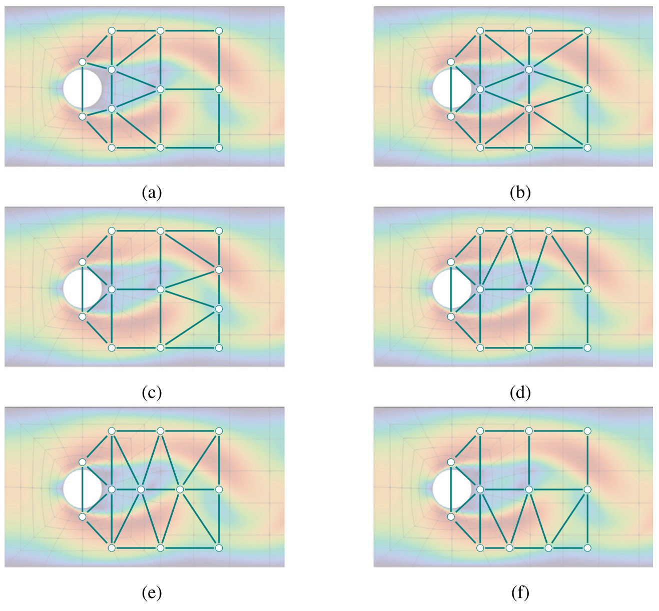

Fig. C.12. Different probe configurations, with 12 probes instead of the original 11.

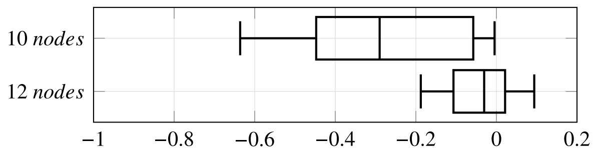

Fig. C.13. r_D spreads for the two tested cases of configurations. The box limits are defined by the first and third quartiles, the notch within is the median and the whiskers are the minimum and maximum values.

C.2. Different node configurations with the GNN

Firstly, note that the GNN has originally been trained with 11 input probes, and it is this trained network that is now being tested on different node configurations. Since the space of possible node placements is innumerable, we settle on two kinds. The first has one fewer sensor than the original configuration (i.e. 10) and the second has one more (i.e. 12). These configurations and their connectivities are shown in Fig. C.14 and Fig. C.12. The distribution of r_D observed in these cases is plotted in Fig. C.13. We can observe that despite the spread, the GNN does seem to able to operate reasonably well even on differing node configurations. Of course, the 10-node case has a larger spread, with r_D values dipping into the −0.6 range. This can be attributed to the loss of information in these specific configurations from the masking of a particular node. The 12-node case, while better than the 10-node case, still has a lower median value than the original 11-node case. This could be owing to our training process, where the node positions are kept fixed throughout training, thus building a sort of rigidity/affinity into the networks for those specific positions. As noted earlier, this was not the aim of the present study and therefore not as extensively tested, but these preliminary results highlight a key advantage of using such flexible architectures over MLPs, and motivate future work in further exploring GNN applicability in this direction.

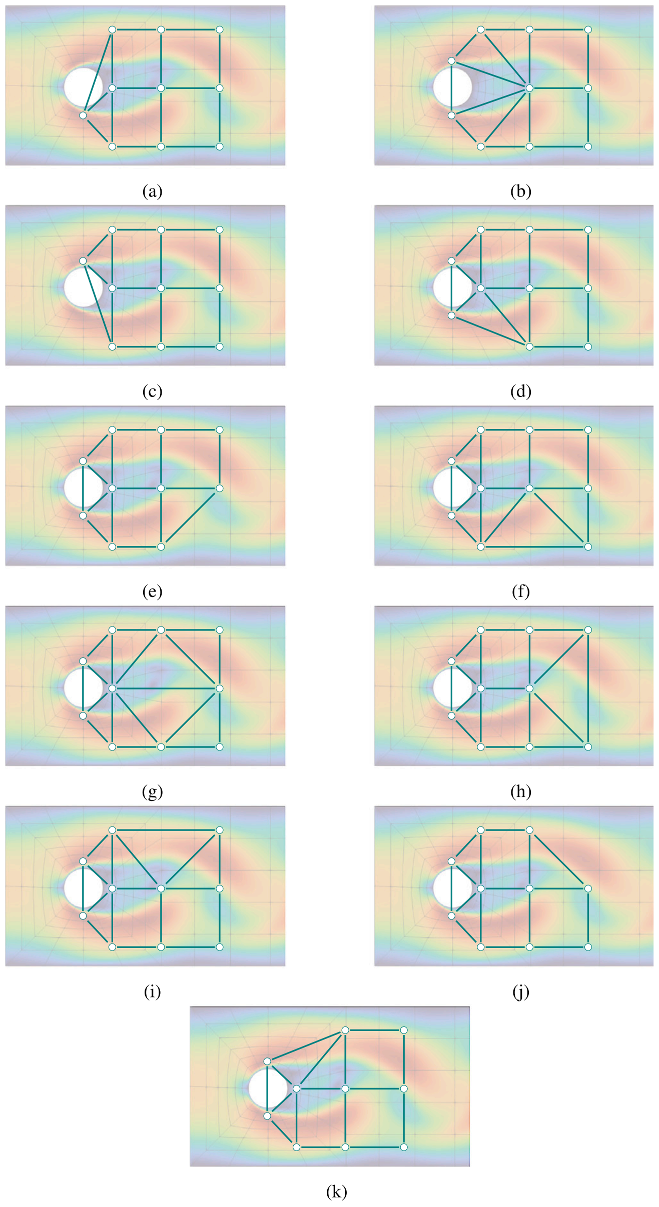

Fig. C.14. Different probe configurations, with 10 probes instead of the original 11. A single probe from the original configuration has been masked to obtain these configurations.

Data availability

The Relexi and FLEXI codes used within this work are avail- able under the GPLv3 license at:

• https://github.com/flexi-framework/relexi
• https://github.com/flexi-framework/flexi
• https://github.com/flexi-framework/ flexi-extensions/tree/smartsim

The implementation of the GNN in TensorFlow is available un- der the MIT license at:

• https://github.com/m-kurz/gcnn

The data generated in the context of this work and instruc- tions to reproduce them with these codes are made available under the CC-BY 4.0 license at:

• 10.18419/darus-4820.

References

[1] Vignon C, Rabault J, Vinuesa R. Recent advances in applying deep reinforcement learning for flow control: Perspectives and future directions. Phys Fluids 2023;35(3):031301. http://dx.doi.org/10.1063/5.0143913.
[2] Vinuesa Ricardo. Perspectives on predicting and controlling turbulent flows through deep learning. Phys Fluids 2024;36(3):031401. http://dx.doi.org/10.1063/5.0190452.
[3] Rabault Jean, Kuchta Miroslav, Jensen Atle, Reglade Ulysse, Cerardi Nicolas. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. J Fluid Mech 2019;865:281–302. http://dx.doi.org/10.1017/jfm.2019.62.
[4] Rabault Jean, Kuhnle Alexander. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. Phys Fluids 2019;31(9):094105. http://dx.doi.org/10.1063/1.5116415.
[5] Ren Feng, Rabault Jean, Tang Hui. Applying deep reinforcement learning to active flow control in turbulent conditions. Phys Fluids 2021;33(3):037121. http://dx.doi.org/10.1063/5.0037371.
[6] Varela Pau, Suárez Pol, Alcántara-Ávila Francisco, Miró Arnau, Rabault Jean, Font Bernat, García-Cuevas Luis Miguel, Lehmkuhl Oriol, Vinuesa Ricardo. Deep reinforcement learning for flow control exploits different physics for increasing Reynolds number regimes. Actuators 2022;11(12):359. http://dx.doi.org/10.3390/act11120359.
[7] Weiner Andre, Geise Janis. Model-based deep reinforcement learning for accelerated learning from flow simulations. Meccanica 2024. http://dx.doi.org/10.1007/s11012-024-01808-z.
[8] Jiang Haokui, Cao Shunxiang. Reinforcement learning-based active flow control of oscillating cylinder for drag reduction. Phys Fluids 2023;35(10):107140. http://dx.doi.org/10.1063/5.0172081.
[9] Han Bing-Zheng, Huang Wei-Xi, Xu Chun-Xiao. Deep reinforcement learning for active control of flow over a circular cylinder with rotational oscillations. Int J Heat Fluid Flow 2022;96:109008. http://dx.doi.org/10.1016/j.ijheatfluidflow.2022.109008.
[10] Guastoni Luca, Rabault Jean, Schlatter Philipp, Azizpour Hossein, Vinuesa Ricardo. Deep reinforcement learning for turbulent drag reduction in channel flows. Eur Phys J E 2023;46(4):27. http://dx.doi.org/10.1140/epje/s10189-023-00285-8.
[11] Sonoda Takahiro, Liu Zhuchen, Itoh Toshitaka, Hasegawa Yosuke. Reinforcement learning of control strategies for reducing skin friction drag in a fully developed turbulent channel flow. J Fluid Mech 2023;960:A30. http://dx.doi.org/10.1017/jfm.2023.147.
[12] Cavallazzi Giorgio Maria, Guastoni Luca, Vinuesa Ricardo, Pinelli Alfredo. Deep reinforcement learning for the management of the wall regeneration cycle in wall-bounded turbulent flows. Flow, Turbul Combust 2024. http://dx.doi.org/10.1007/s10494-024-00609-4.
[13] Suárez Pol, Alcántara-Ávila Francisco, Miró Arnau, Rabault Jean, Font Bernat, Lehmkuhl Oriol, Vinuesa Ricardo. Active flow control for drag reduction through multi-agent reinforcement learning on a turbulent cylinder at $\mathrm{Re}_D = 3900$. Flow, Turbul Combust 2025. http://dx.doi.org/10.1007/s10494-025-00642-x.
[14] Font Bernat, Alcántara-Ávila Francisco, Rabault Jean, Vinuesa Ricardo, Lehmkuhl Oriol. Deep reinforcement learning for active flow control in a turbulent separation bubble. Nat Commun 2025;16(1):1422. http://dx.doi.org/10.1038/s41467-025-56408-6.
[15] Wang Qiulei, Yan Lei, Hu Gang, Li Chao, Xiao Yiqing, Xiong Hao, Rabault Jean, Noack Bernd R. DRLinFluids: An open-source Python platform of coupling deep reinforcement learning and OpenFOAM. Phys Fluids 2022;34(8):081801. http://dx.doi.org/10.1063/5.0103113.
[16] Shams Mosayeb, Elsheikh Ahmed H. Gym-preCICE: Reinforcement learning environments for active flow control. SoftwareX 2023;23:101446. http://dx.doi.org/10.1016/j.softx.2023.101446.
[17] Kurz Marius, Offenhäuser Philipp, Viola Dominic, Resch Michael, Beck Andrea. Relexi — A scalable open source reinforcement learning framework for high-performance computing. Softw Impacts 2022;14:100422. http://dx.doi.org/10.1016/j.simpa.2022.100422.
[18] Jin Chi, Allen-Zhu Zeyuan, Bubeck Sebastien, Jordan Michael I. Is Q-learning provably efficient? In: Advances in neural information processing systems, vol. 31, Montréal, Canada; 2018.
[19] Pfaff Tobias, Fortunato Meire, Sanchez-Gonzalez Alvaro, Battaglia Peter. Learning mesh-based simulation with graph networks. In: International conference on learning representations. 2020.
[20] Chen Junfeng, Hachem Elie, Viquerat Jonathan. Graph neural networks for laminar flow prediction around random two-dimensional shapes. Phys Fluids 2021;33(12).
[21] Gao Rui, Jaiman Rajeev K. Predicting fluid–structure interaction with graph neural networks. Phys Fluids 2024;36(1):013622. http://dx.doi.org/10.1063/5.0182801, arXiv:https://pubs.aip.org/aip/pof/article-pdf/doi/10.1063/5.0182801/18930456/013622_1_5.0182801.pdf.
[22] He Xiaodong, Wang Yinan, Li Juan. Flow completion network: Inferring the fluid dynamics from incomplete flow information using graph neural networks. Phys Fluids 2022;34(8).

[23] Lino Mario, Fotiadis Stathi, Bharath Anil Anthony, Cantwell Chris D. Multi-scale rotation-equivariant graph neural networks for unsteady Eulerian fluid dynamics. Phys Fluids 2022. URL https://api.semanticscholar.org/CorpusID:250941850.

[24] Li Zong-Yi, Kovachki Nikola B, Azizzadenesheli Kamyar, Liu Burigede, Bhattacharya Kaushik, Stuart Andrew M, Anandkumar Anima. Neural operator: Graph kernel network for partial differential equations. 2020, ArXiv, arXiv:2003.03485, URL https://api.semanticscholar.org/CorpusID:211838009.

[25] Franco Nicola Rares, Fresca Stefania, Tombari Filippo, Manzoni Andrea. Deep learning-based surrogate models for parametrized PDEs: handling geometric variability through graph neural networks. Chaos 2023;33 12. URL https://api.semanticscholar.org/CorpusID:260438437.

[26] Gao Han, Zahr Matthew J, Wang Jian-Xun. Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems. Comput Methods Appl Mech Engrg 2022;390:114502.

[27] Kim Hojin, Shankar Varun, Viswanathan Venkatasubramanian, Maulik Romit. Generalizable data-driven turbulence closure modeling on unstructured grids with differentiable physics. 2023, URL https://api.semanticscholar.org/CorpusID:260154718.

[28] Dupuy Dorian, Odier Nicolas, Lapeyre Corentin, Papadogiannis Dimitrios. Modeling the wall shear stress in large-eddy simulation using graph neural networks. Data-Centric Eng 2023;4:e7.

[29] Dupuy Dorian, Odier Nicolas, Lapeyre Corentin. Using graph neural networks for wall modeling in compressible anisothermal flows. Data Centric Eng 2024;5:e10.

[30] Yang XIA, Zafar S, Wang J-X, Xiao H. Predictive large-eddy-simulation wall modeling via physics-informed neural networks. Phys Rev Fluids 2019;4:034602. http://dx.doi.org/10.1103/PhysRevFluids.4.034602, URL https://link.aps.org/doi/10.1103/PhysRevFluids.4.034602.

[31] Kurz Marius, Offenhäuser Philipp, Viola Dominic, Shcherbakov Oleksandr, Resch Michael, Beck Andrea. Deep reinforcement learning for computational fluid dynamics on HPC systems. J Comput Sci 2022;65:101884. http://dx.doi.org/10.1016/j.jocs.2022.101884.

[32] Vignon Colin, Rabault Jean, Vasanth Joel, Alcántara-Ávila Francisco, Mortensen Mikael, Vinuesa Ricardo. Effective control of two-dimensional Rayleigh–Bénard convection: Invariant multi-agent reinforcement learning is all you need. Phys Fluids 2023;35(6):065146. http://dx.doi.org/10.1063/5.0153181.

[33] Peitz Sebastian, Stenner Jan, Chidananda Vikas, Wallscheid Oliver, Brunton Steven L, Taira Kunihiko. Distributed control of partial differential equations using convolutional reinforcement learning. Phys D: Nonlinear Phenom 2024;461:134096. http://dx.doi.org/10.1016/j.physd.2024.134096.

[34] Jeon Joogoo, Rabault Jean, Vasanth Joel, Alcántara-Ávila Francisco, Baral Shilaj, Vinuesa Ricardo. Advanced deep-reinforcement-learning methods for flow control: Group-invariant and positional-encoding networks improve learning speed and quality. 2024, arXiv:2407.17822.

[35] Kurz Marius, Offenhäuser Philipp, Beck Andrea. Deep reinforcement learning for turbulence modeling in large eddy simulations. Int J Heat Fluid Flow 2023;99:109094. http://dx.doi.org/10.1016/j.ijheatfluidflow.2022.109094.

[36] Beck Andrea, Kurz Marius. Toward discretization-consistent closure schemes for large eddy simulation using reinforcement learning. Phys Fluids 2023;35(12):125122. http://dx.doi.org/10.1063/5.0176223.

[37] Krais Nico, Beck Andrea, Bolemann Thomas, Frank Hannes, Flad David, Gassner Gregor, Hindenlang Florian, Hoffmann Malte, Kuhn Thomas, Sonntag Matthias, Munz Claus-Dieter. FLEXI: A high order discontinuous Galerkin framework for hyperbolic–parabolic conservation laws. Comput Math Appl 2021;81:186–219. http://dx.doi.org/10.1016/j.camwa.2020.05.004.

[38] Schäfer M, Turek S, Durst F, Krause E, Rannacher R. Benchmark computations of laminar flow around a cylinder. In: Hirschel Ernst Heinrich, editor. Flow simulation with high-performance computers II: dFG priority research programme results 1993–1995. Wiesbaden: Vieweg+Teubner Verlag; 1996, p. 547–66. http://dx.doi.org/10.1007/978-3-322-89849-4_39.

[39] Carlson Jan-Reneé. Inflow/Outflow Boundary Conditions with Application to FUN3D. Technical Report NASA/TM–2011-217181, Langley Research Center, Hampton, VA, United States: Langley Research Center; 2011.

[40] Schulman John, Wolski Filip, Dhariwal Prafulla, Radford Alec, Klimov Oleg. Proximal policy optimization algorithms. 2017, ArXiv Preprint, arXiv:1707.06347.

[41] Lillicrap Timothy P, Hunt Jonathan J, Pritzel Alexander, Heess Nicolas, Erez Tom, Tassa Yuval, Silver David, Wierstra Daan. Continuous control with deep reinforcement learning. 2015, arXiv preprint arXiv:1509.02971.

[42] Barth-Maron Gabriel, Hoffman Matthew W, Budden David, Dabney Will, Horgan Dan, Tb Dhruva, Muldal Alistair, Heess Nicolas, Lillicrap Timothy. Distributed distributional deterministic policy gradients. 2018, arXiv preprint arXiv:1804.08617.

[43] Gu Shixiang, Lillicrap Timothy, Sutskever Ilya, Levine Sergey. Continuous deep Q-learning with model-based acceleration. In: Balcan Maria Florina, Weinberger Kilian Q, editors. Proceedings of the 33rd international conference on machine learning. Proceedings of machine learning research, 48, New York, New York, USA: PMLR; 2016, p. 2829–38, URL https://proceedings.mlr.press/v48/gu16.html.

[44] Sutton Richard S, Barto Andrew G, et al. Reinforcement learning: An introduction, vol. 1, MIT press Cambridge; 2020.

[45] Kipf Thomas N, Welling Max. Semi-supervised classification with graph convolutional networks. In: International conference on learning representations. 2017.

[46] Abadi Martín, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, Corrado Greg S, Davis Andy, Dean Jeffrey, Devin Matthieu, Ghemawat Sanjay, Goodfellow Ian, Harp Andrew, Irving Geoffrey, Isard Michael, Jia Yangqing, Jozefowicz Rafal, Kaiser Lukasz, Kudlur Manjunath, Levenberg Josh, Mané Dandelion, Monga Rajat, Moore Sherry, Murray Derek, Olah Chris, Schuster Mike, Shlens Jonathon, Steiner Benoit, Sutskever Ilya, Talwar Kunal, Tucker Paul, Vanhoucke Vincent, Vasudevan Vijay, Viégas Fernanda, Vinyals Oriol, Warden Pete, Wattenberg Martin, Wicke Martin, Yu Yuan, Zheng Xiaoqiang. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015, Software available from tensorflow.org. URL https://www.tensorflow.org/.

[47] Guadarrama Sergio, Korattikara Anoop, Ramirez Oscar, Castro Pablo, Holly Ethan, Fishman Sam, Wang Ke, Gonina Ekaterina, Wu Neal, Kokiopoulou Efi, Sbaiz Luciano, Smith Jamie, Bartók Gábor, Berent Jesse, Harris Chris, Vanhoucke Vincent, Brevdo Eugene. TF-Agents: A library for reinforcement learning in TensorFlow. 2018, Online, URL https://github.com/tensorflow/agents. [Accessed 25 June 2019].

[48] Partee Sam, Ellis Matthew, Rigazzi Alessandro, Shao Andrew E, Bachman Scott, Marques Gustavo, Robbins Benjamin. Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling. J Comput Sci 2022;62:101707. http://dx.doi.org/10.1016/j.jocs.2022.101707.

[49] Blind Marcel, Kopper Patrick, Kempf Daniel, Kurz Marius, Schwarz Anna, Munz Claus-Dieter, Beck Andrea. Performance improvements for large scale simulations using the discontinuous Galerkin framework FLEXI. In: High performance computing in science and engineering '22. Cham: Springer Nature Switzerland; 2024, p. 249–64.

[50] Blind Marcel P, Gibis Tobias, Wenzel Christoph, Beck Andrea. Wall-modeled large eddy simulation of a tandem wing configuration in transonic flow. Phys Fluids 2024;36(5):055125. http://dx.doi.org/10.1063/5.0198271.

[51] Dürrwächter Jakob, Kurz Marius, Kopper Patrick, Kempf Daniel, Munz Claus-Dieter, Beck Andrea. An efficient sliding mesh interface method for high-order discontinuous Galerkin schemes. Comput & Fluids 2021;217:104825. http://dx.doi.org/10.1016/j.compfluid.2020.104825, URL https://www.sciencedirect.com/science/article/pii/S0045793020303959.

[52] Kurz Marius, Kempf Daniel, Blind Marcel P, Kopper Patrick, Offenhäuser Philipp, Schwarz Anna, Starr Spencer, Keim Jens, Beck Andrea. GALÆXI: Solving complex compressible flows with high-order discontinuous Galerkin methods on accelerator-based systems. Comput Phys Comm 2025;306:109388. http://dx.doi.org/10.1016/j.cpc.2024.109388.

[53] Dossa Rousslan Fernand Julien, Huang Shengyi, Ontañón Santiago, Matsubara Takashi. An empirical investigation of early stopping optimizations in proximal policy optimization. IEEE Access 2021;9:117981–92. http://dx.doi.org/10.1109/ACCESS.2021.3106662.

[54] Raffin Antonin, Hill Ashley, Gleave Adam, Kanervisto Anssi, Ernestus Maximilian, Dormann Noah. Stable-Baselines3: Reliable reinforcement learning implementations. J Mach Learn Res 2021;22(268):1–8, URL http://jmlr.org/papers/v22/20-1364.html.

[55] Sun Mingfei, Kurin Vitaly, Liu Guoqing, Devlin Sam, Qin Tao, Hofmann Katja, Whiteson Shimon. You may not need ratio clipping in PPO. 2022, ArXiv Preprint, arXiv:2202.00079.

[56] Engstrom Logan, Ilyas Andrew, Santurkar Shibani, Tsipras Dimitris, Janoos Firdaus, Rudolph Larry, Madry Aleksander. Implementation matters in deep RL: A case study on PPO and TRPO. In: International conference on learning representations. 2019.

[57] Jia Wang, Xu Hang. Deep reinforcement learning-based active flow control of an elliptical cylinder: Transitioning from an elliptical cylinder to a circular cylinder and a flat plate. Phys Fluids 2024;36(7):074117. http://dx.doi.org/10.1063/5.0218408, arXiv:https://pubs.aip.org/aip/pof/article-pdf/doi/10.1063/5.0218408/20076775/074117_1_5.0218408.pdf.

[58] Kaelbling Leslie Pack, Littman Michael L, Moore Andrew W. Reinforcement learning: A survey. J Artificial Intelligence Res 1996;4:237–85.

[59] Hindenlang Florian J, Gassner Gregor J, Munz Claus-Dieter. Improving the accuracy of discontinuous Galerkin schemes at boundary layers. Internat J Numer Methods Fluids 2014;75(6):385–402. http://dx.doi.org/10.1002/fld.3898.

[60] Tang Hongwei, Rabault Jean, Kuhnle Alexander, Wang Yan, Wang Tongguang. Robust active flow control over a range of Reynolds numbers using an artificial neural network trained through deep reinforcement learning. Phys Fluids 2020;32(5).