A review on deep reinforcement learning for fluid mechanics: an update

J Viquerat, P Meliga, A Larcher, E Hachem

Accepted to Phys. Fluids 10.1063/5.0128446

A review on deep reinforcement learning for fluid mechanics: an update

A Preprint

J. Viquerat[*]
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
jonathan.viquerat@mines-paristech.fr

P. Meliga
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
philippe.meliga@mines-paristech.fr

A. Larcher
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
aurelien.larcher@mines-paristech.fr

E. Hachem
MINES Paristech, CEMEF
PSL - Research University
06904 Sophia Antipolis, France
elie.hachem@mines-paristech.fr

November 1, 2022

Abstract

In the past couple of years, the interest of the fluid mechanics community for deep reinforcement learning techniques has increased at fast pace, leading to a growing bibliography on the topic. Due to its ability to solve complex decision-making problems, deep reinforcement learning has especially emerged as a valuable tool to perform flow control, but recent publications also advertise great potential for other applications, such as shape optimization or micro-fluidics. The present work proposes an exhaustive review of the existing literature, and is a follow-up to our previous review on the topic. The contributions are regrouped by domain of application, and are compared together regarding algorithmic and technical choices, such as state selection, reward design, time granularity, and more. Based on these comparisons, general conclusions are drawn regarding the current state-of-the-art, and perspectives for future improvements are sketched.

***K*eywords** Deep reinforcement learning · Fluid mechanics

1 Introduction

During the past decade, machine learning methods, and more specifically deep neural network, have achieved great successes in a wide variety of domains. State-of-the-art neural network architectures have reached astonishing performance levels in image classification tasks [1, 2], speech recognition [3] or generative tasks [4]. With a generalized access to GPU computational resources through cheaper hardware or cloud computing, such advances have been paving the way for a general evolution of the reference methods in these domains at both academic and industrial levels. Machine learning has been making especially rapid inroads in fluid mechanics, as a flexible modeling framework that can be fit to address many challenges, including reduced-order modeling, experimental data processing, shape optimization, turbulence closure modeling, and control [5].

[*] Corresponding author

The rapid expansion of neural networks to multiple domains has also yielded important progress in the domain of decision-making techniques, by the coupling of deep neural networks with reinforcement learning algorithms (called deep reinforcement learning, or DRL). Several major obstacles that had been hindering classical reinforcement learning have been lifted using the feature extraction capabilities of deep neural networks and their ability to handle high-dimensional state spaces. Unprecedented efficiency has been achieved in many domains such as robotics [6], language processing [7], or games [8,9], but DRL has also proven useful in many industrial applications, such as autonomous cars [10,11], or data center cooling [12].

Different incentives trigger the interest of DRL for fluid mechanics applications, that constitutes the core subject of this review:

- unsteady flow fields exhibiting complex, multiscale phenomena require algorithms capable of handling nonlinearities and multiple spatiotemporal scales. Those are thus particularly amenable to DRL and its representation capabilities, all the more so in the context of flow control, where molding a flow into a more desired state may change the system dynamics and make predictions based on data of uncontrolled systems obsolete,
- in the context of resource expensive numerical environments, DRL offers a way to learn policies by encoding system goals into the reward function and leveraging exploration during training, which is especially beneficial to flow control and optimization problems,
- DRL is a framework that can account for long-term dependencies in decision-making, which is especially interesting to tackle sensitivity to the initial condition or memory effects in turbulence,
- a deeper understanding and exploitation of fluid mechanics is expected to become instrumental in complementing engineering intuition and practical experience, as fluid dynamics has wide applicability, from the way gases circulate around planets, to the way fuel combusts in engines, to the modelling of blood flow and the design of medical implants.

Despite these motivations, the efforts for applying DRL to fluid mechanics are ongoing but still at an early stage, with only a handful of pioneering studies providing insight into the performance improvements to be delivered in the field. Nonetheless, from a few liminal contributions in 2016 [13] and early 2018 [14], the domain has undergone an increasing inflow of contributions, that pinnacled in 2020, with no less than 16 pre-prints and articles, and a clear focus on drag reduction problems, as shown in figure 1. This enthusiasm can be explained by two main factors: the increasing number of open-source initiatives [15–17], that has led to an accelerated diffusion of the methods in the community, and the sustained commitment from the machine learning community, that has allowed concurrently expanding the scope from computationally inexpensive, low-dimensional reductions of the underlying fluid dynamics to complex Navier–Stokes systems, all the way to experimental set-ups.

The present review proposes a six-year perspective of deep reinforcement learning applied to fluid flow problems, in the context of both numerical and experimental environments. It is intended as a follow up to our first review released as a pre-print in 2019 [18], that was followed in 2020 by a short review from another group of authors, focused on drag reduction and shape optimization problems [19]. To the best of the authors knowledge, those are the only other similar initiatives preceding this one, that are also featured in figure 1 for the sake of completeness. in practice, only contributions focusing on the application of deep reinforcement learning techniques (not machine learning in a broader sense) to fluid dynamical systems, either experimental or numerical as governed by Navier–Stokes equations (or a reduced model thereof), have been considered. The objective is two-fold: first to analyze the main trends, achievements and research directions currently pursued in the community. Second, to identify the main needs to inform future developments towards practical deployment.

The organization is as follows: a reminder on the main DRL algorithms that have been used in a fluid dynamics context is provided in section 2. Section 3 lists of relevant issues to consider when evaluating the progress of DRL for practically fluid flow problems. An extensive bibliographical review is then conducted in sections 4 and 5, that covers a total of 43 papers, most of them subsequent to the previous review articles. Contributions are grouped and compared by domain of applications. Finally, section 6 draws general conclusions on the state-of-the-art and proposes a transversal study including suggestions for future work in the field.

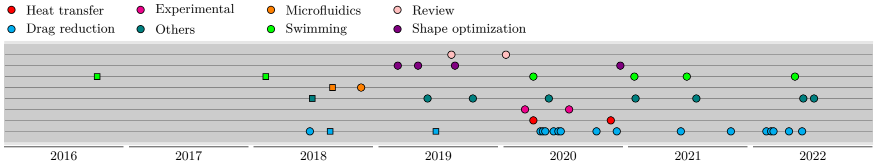


Figure 1: **Timeline of recent publications on deep reinforcement learning for fluid dynamics.** Colors indicate different fields of application. Please note that we retain here the date of the first pre-print publication, and not that of final publication in peer-review journals. Indeed, the fast-paced evolution of the deep reinforcement learning community brings particular importance to pre-prints, sometimes supported by code release. The square symbols denote references included in our previous review [18].

2 Deep reinforcement learning

This section covers the basic concepts of deep reinforcement learning, and briefly describes the methods most represented in the selected contributions. First, the basic concepts of reinforcement learning are introduced, whereafter value-based and policy-based methods are distinguished. Then, specificities of DRL are detailed, and a curated list of algorithms is proposed, including deep Q-networks (DQN), advantage actor-critic (A2C), proximal policy optimization (PPO), trust-region policy optimization (TRPO), deep deterministic policy gradients (DDPG), twin-delayed deep deterministic policy gradients (TD3), soft actor-critic (SAC) and policy-based optimization (PBO). For a more sophisticated introduction to DRL, the reader is referred to [20]. The notations used in the remaining of this review can be found in table 1.

2.1 Reinforcement learning

Reinforcement learning (RL) is a class of methods designed for decision-making problems, in which an agent learns to interact with an environment by (i) observing it, (ii) taking actions based on these observations, and (iii) receiving rewards from it, as a measure of the quality of the action taken. RL is based on Markov decision processes, for which a typical execution goes as follows (see also figure 2):

- Assume the environment is in state $s_t \in \mathcal{S}$ at iteration t, where $\mathcal{S}$ is a set of states;
- The agent uses w_t, an observation of the current environment state (and possibly a partial subset of s_t) to take action $a_t \in \mathcal{A}$, where $\mathcal{A}$ is a set of actions;
- The environment reacts to the action by transitionning from s_t to state $s_{t+1} \in \mathcal{S}$;
- The agent is fed with a reward $r_t \in \mathcal{R}$, where $\mathcal{R}$ is a set of rewards, and a new observation w_{t+1}.

The steps described above repeat until a termination state is reached, and the succession of states and actions then define a finite trajectory $\tau = \left(s_0, a_0, s_1, a_1, ...\right)$. In any given state, the objective of the agent is to determine the adequate action to maximize its cumulative reward over an episode, *i.e.* over one trajectory. Most often, the quantity of interest is the discounted cumulative reward along a trajectory, defined as:

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t \,, \tag{1}$$

where T is the horizon of the trajectory (*i.e.* the terminal time station), and $\gamma \in [0, 1]$ is a discount factor that weights the relative importance of present and future rewards. Within the zoology of DRL methods, we distinguish two categories, namely *model-based* and *model-free* algorithms. Model-based method incorporates a model of the environment they interact with, and will not be considered in this paper (the reader is referred to [20] and references therein for details about model-based methods). On

Table 1: **List of notations and acronysms.**

γ	discount factor
λ	learning rate
α, β	step-size
ϵ	probability of random action
s,s'	states
$\mathcal{S}^+$	set of all states
$\mathcal{S}$	set of non-termination states
a	action
$\mathcal{A}$	set of all actions
r	reward
$\mathcal{R}$	set of all rewards
$\mathcal{D}$	set of collected transitions
t	time station
T	final time station
a_t	action at time t
s_t	state at time t
r_t	reward at time t
$r(s,a)$	reward received for taking action a in state s
$R(\tau)$	discounted cumulative reward following trajectory τ
G_t	discounted cumulative reward starting from time t
π	policy
θ, θ'	parameterization vector of a policy
π_θ	policy parameterized by θ
$\pi(s)$	action probability distribution in state s following π
$\pi(a\|s)$	probability of taking action a in state s following π
$V^\pi(s)$	value of state s under policy π
$V^*(s)$	value of state s under the optimal policy
$Q^\pi(s,a)$	value of taking action a in state s under policy π
$Q^*(s,a)$	value of taking action a in state s under the optimal policy
$Q_\theta(s,a)$	estimated value of taking action a in state s with parameterization θ
A2C	Advantage actor-critic
BO	Bayesian optimization
CFD	Computational fluid dynamics
CMA-ES	Covariance matrix adaptation evolution strategy
DDPG	Deep deterministic policy gradient
DQN	Deep Q-networks
DDQN	Double deep Q-networks
DRL	Deep reinforcement learning
GP	Genetic programming
LIPO	Lipschitz global optimization
LSTM	Long-short term memory
MFEC	Model-free episodic control
PBO	Policy-based optimization
PPO	Proximal policy optimization
PPO-1	Single-step proximal policy optimization
RL	Reinforcement learning
SAC	Soft actor-critic
TD3	Twin-delayed deep deterministic policy gradient
TRPO	Trust-region policy optimization

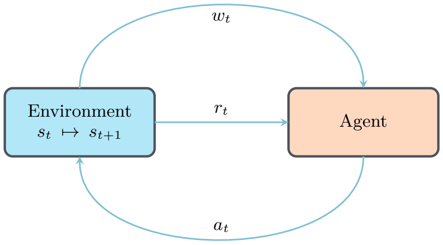


Figure 2: **Reinforcement learning agent and its interactions with the environment.**

the contrary, model-free algorithms directly interact with their environment, and are currently the most commonly used within the DRL community, mainly for their ease of application and implementation. Model-free methods are further distinguished between *value-based* methods and *policy-based* methods [20]. Although both approaches aim at maximizing their expected return, policy-based methods do so by directly optimizing the parameterized policy, while value-based methods learn to estimate the expected value of a state-action pair optimally, which in turn determines the best action to take in each state.

2.1.1 Value-based methods

In value-based methods, the agent learns to optimally estimate a *value function*, which in turn dictates the policy of the agent by selecting the action of the highest value. One usually defines the *state value function*:

$$V^\pi(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}\big[R(\tau)|s\big],$$

denoting the expected discounted cumulative reward starting in state s, then following trajectory τ according to policy π, and the *state-action value function*, or Q-function:

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}\big[R(\tau)|s,a\big],$$

denoting the same expected discounted cumulative reward starting in state s and taking action a. Both values are quite obviously such that:

$$V^\pi(s) = \mathop{\mathbb{E}}_{a\sim\pi}\big[Q^\pi(s,a)\big],$$

meaning that in practice, $V^\pi(s)$ is the weighted average of $Q^\pi(s,a)$ over all possible actions by the probability of each action. One of the main value-based methods in use is called Q-learning, as it relies on the learning of the Q-function to find an optimal policy. In classical Q-learning, the Q-function is stored in a Q-table, which is a simple array representing the estimated value of the optimal Q-function $Q^*(s,a)$ for each pair $(s,a) \in \mathcal{S}\times\mathcal{A}$. The Q-table is initialized randomly, and its values are progressively updated as the agent explores the environment, until the Bellman optimality condition [21] is reached:

$$Q^*(s,a) = r(s,a) + \gamma \max_{a'} Q^*(s',a'), \tag{2}$$

at which point the Q-table estimate of the Q-value has converged, and taking the action with the highest Q-value systematically leads to the optimal policy.

2.1.2 Policy-based methods

Policy methods maximize the expected discounted cumulative reward of a policy $\pi(a|s)$ mapping states to actions, and resort not to a value function, but to a probability distribution over actions given states. Compared to value-based methods, policy-based methods offer three main advantages:

- they have better convergence properties, although they tend to get trapped in local minima;
- they naturally handle high dimensional action spaces;
- they can learn stochastic policies.

Most reinforcement learning algorithms applied to fluid mechanics problems are policy gradient methods, in which gradient ascent is used to optimize a parameterized policy $\pi_\theta(a|s)$ with respect to some measure of the expected return. In practice, one defines an objective function based on the expected discounted cumulative reward:

$$J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\big[R(\tau)\big],$$

and seeks the optimal parameterization θ^* that maximizes $J(\theta)$:

$$\theta^* = \arg\max_{\theta} \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\big[R(\tau)\big],$$

which can be done on paper by plugging an estimator of the policy gradient $\nabla_\theta J(\theta)$ into a gradient ascent algorithm. In practice, this is no small task, as one is looking for the gradient with respect to the policy parameters θ, in a context where the effects of policy changes on the state distribution are unknown (since modifying the policy will most likely modify the set of visited states, which will in turn affect performance in some indefinite manner). The standard derivation relies on the log-probability trick [22], and allows expressing $\nabla_\theta J(\theta)$ as an evaluable expected value:

$$\nabla_\theta J(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log\left(\pi_\theta(a_t|s_t)\right) R(\tau)\right], \tag{3}$$

after which the gradient is used to update the policy parameters:

$$\theta \leftarrow \theta + \lambda \nabla_\theta J(\theta). \tag{4}$$

It must be noted that expression (3) represents the simplest form of the policy gradient, and that, in practice, more elaborate expressions are chosen instead of $R(\tau)$ that decrease the variance of the associated estimator without introducing bias, such as the reward-to-go or the advantage function, among others (see section 2.3.3).

2.2 Deep reinforcement learning

Deep reinforcement learning (or DRL) is the result of applying reinforcement learning with deep neural networks to output either value functions (value-based RL methods), or action distributions given input states (policy-based RL methods)[2]. A neural network is a collection of artificial neurons, *i.e.* connected computational units with universal approximation capabilities [23, 24], that can be trained to arbitrarily well approximate the mapping function between input and output spaces. Each connection provides the output of a neuron as an input to another neuron. Each neuron performs a weighted sum of its inputs, to assign significance to the inputs with regard to the task the algorithm is trying to learn. It then adds a bias to better represent the part of the output that is actually independent of the input. Finally, it feeds a non-linear activation function that determines whether

[2] An alternative presented above is to use tables to store the values for every state or state-action pair, but such a strategy generally does not scale with the size of state-action spaces, and is thus limited to discrete spaces.

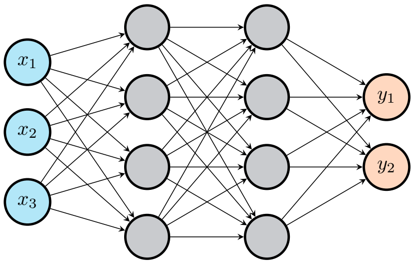


Figure 3: **Fully connected neural network with two hidden layers**.

and to what extent the computed value should affect the ultimate outcome. As sketched in figure 3, a fully connected network is generally organized into layers, with the neurons of one layer being connected solely to those of the immediately preceding and following layers. The layer that receives the external data is the input layer, the layer that produces the outcome is the output layer, and in between them are zero or more hidden layers.

The learning process in neural networks consists in adjusting all the biases and weights of the network in order to reduce the value of a well-chosen loss function that represents the quality of the network prediction. This update is usually performed by a stochastic gradient method, in which the gradients of the loss function with respect to the weights and biases (*i.e.* the parameterization θ of the neural network) are obtained using a back-propagation algorithm. The abundant literature available on this topic (see [25] and the references therein) points out that a relevant network architecture (*e.g.* type of network, depth, width of each layer), finely tuned hyper parameters (*i.e.* parameters whose value cannot be estimated from data, *e.g.* , optimizer, learning rate, batch size) and a sufficiently large amount of data to learn from are key ingredients for a successful network training.

2.3 Deep reinforcement learning algorithms

This section briefly reviews some of the most popular DRL methods encountered in the field of DRL for fluid mechanics. Only their main features are reviewed here, the reader interested in further details (or in the numerous custom variations introduced in the RL literature) is referred to the various references given in this section.

2.3.1 Deep Q-networks (DQN)

In Q-learning methods, obtaining a converged Q-table for large state and actions spaces can be particularly expensive in terms of environment interactions. To overcome this issue, the map $\mathcal{S}^+ \times \mathcal{A} \longrightarrow \mathbb{R}$ is represented by a neural network, called deep Q-network [26], tasked with providing an estimate of the Q-value for each possible action given an input state. To do so, the Q-network is trained on state-action-reward transitions obtained by interacting with the environment. The loss used for the training is classically obtained from the Bellman equation (2):

$$L(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\frac{1}{2} \left(\left[r(s,a) + \gamma \max_{a'} Q_\theta(s',a') \right] - Q_\theta(s,a) \right)^2 \right], \tag{5}$$

where $Q_\theta(s,a)$ is the Q-value *estimate* provided by the DQN for action s and state a under network parameterization θ, and $\mathcal{D}$ represents the set of transitions collected from the environment. The quantity $r(s,a) + \gamma \max_{a'} Q_\theta(s',a')$, denoted *target*, also appears in the Bellman equation (2), as the estimate $Q_\theta(s,a)$ is equal to the target (and $L(\theta)$ is thus zero) when the optimal set of parameters θ^* is reached.

In order to balance the trade-off between exploration and exploitation, the DQN algorithm classically implements a stochastic exploration strategy called ϵ-greedy: before each action, a random parameter p is drawn in $[0,1]$ and compared to a user-defined value $\epsilon \in [0,1]$, and the action prescribed by $\max_a Q_\theta(s,a)$ is taken only if $p > \epsilon$ (otherwise a random action is taken). The value of ϵ usually decreases during the learning process, thereby progressively reducing exploration in favour of exploitation. Nevertheless, the performance of vanilla DQN remains limited. This has led to multiple developments aimed at stabilizing learning and at improving performance, a handful of which have become standard practice *e.g.*, replay [27], target networks [26], or double deep Q-networks [28]. For the sake of clarity, we shall not go into the specifics of these evolutions, for which the interested reader can instead refer to [20] and the references therein.

2.3.2 Vanilla deep policy gradient

In policy methods, a stochastic gradient algorithm is used to perform network updates from the policy loss:

$$L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \log\left(\pi_\theta(a_t|s_t)\right) R(\tau)\right]. \tag{6}$$

whose gradient is equal to the policy gradient (3). The latter is computed with the back-propagation algorithm with respect to each weight and bias by the chain rule, one layer at the time from the output to the input layer. Such a method is also known as Monte Carlo policy gradient, as the loss (6) takes the form of an expected value, that can be numerically calculated using an empirical average over a set of full trajectories. However, if some low-quality actions are taken along the trajectory, their negative impact will be averaged by the high-quality actions and will remain undetected. This problem can be overcome using actor-critic methods, in which a Q-function evaluation is used in conjunction with a policy optimization.

2.3.3 Advantage actor-critic (A2C)

Different strategies are available to alleviate the high variance of training the agent from (6), for which it has become customary to replace the discounted cumulative reward by the *advantage function*:

$$A(s,a) = Q(s,a) - V(s),$$

that represents the improvement in the expected cumulative reward when taking action a in state s, compared to the average of all possible actions taken in state s. As a result, the loss function reads:

$$L(\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \log\left(\pi_\theta(a_t|s_t)\right) A^{\pi_\theta}(s_t,a_t)\right].$$

In practice, the classical policy network (called *actor*) is used concurrently with a second network (called *critic*), that learns to predict the state-value function $V(s)$. The advantage function is then approximated as

$$A(s_t,a_t) \sim r(s_t,a_t) + \gamma V(s_{t+1}) - V(s_t).$$

to avoid having a third network learn to predict the state-action value $Q(s,a)$. In contrast to the Monte Carlo-style update of vanilla policy gradient methods, the actor-critic algorithm allows training the policy network in a temporal-difference manner, meaning that updates can be performed several times during an episode, thanks to the critic state-value estimate [29].

2.3.4 Trust-region and proximal policy optimization (TRPO and PPO)

The performance of policy gradient methods is hurt by the high sensitivity to the learning rate, *i.e.*, the size of the step to be taken in the gradient direction. Indeed, small learning rates are detrimental to learning, but large learning rates can lead to a performance collapse if the agent falls off the cliff and restarts from a poorly performing state with a locally bad policy (an issue magnified by the fact that the learning rate cannot be tuned locally). Trust region policy optimization (TRPO [30])) ensures continuous improvement by leveraging second-order natural gradient optimization to update the policy parameters within a trust-region of fixed maximum Kullback-Leibler divergence between previous and current policies. Proximal policy optimization (PPO [31]) uses a simpler yet effective heuristic to similarly avoid destructive updates. Namely, it relies on the clipped surrogate loss:

$$L(\theta) = \mathop{\mathbb{E}}_{(s,a)\sim\pi_{\theta_{\text{old}}}} \left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}, g\left(\epsilon, A^{\pi_{\theta_{\text{old}}}}(s,a)\right) \right) A^{\pi_{\theta_{\text{old}}}}(s,a) \right],$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & \text{if } A \geq 0, \\ (1-\epsilon)A & \text{if } A < 0, \end{cases}$$

and ϵ is the clipping range, a small, user-defined parameter defining how far away the new policy is allowed to go from the old one. The general picture is that a positive (resp. negative) advantage increases (resp. decreases) the probability of taking action a in state s, but always by a proportion smaller than ϵ, otherwise the min kicks in (2.3.4) and its argument hits a ceiling of $1+\epsilon$ (resp. a floor of $1-\epsilon$). This prevents stepping too far away from the current policy, and ensures that the new policy will behave similarly.

Due to its improved learning stability and its relatively robust behaviour with respect to hyper-parameters, the PPO algorithm has received considerable attention in the DRL community. As shown in table 4, it is by far the most common DRL algorithm exploited in the context of DRL-based control for fluid dynamics.

2.3.5 Deep deterministic policy gradient (DDPG)

Deep deterministic policy gradient (DDPG) can be thought as a Deep Q-network algorithm for continuous actions spaces, that combines the learning of a Q-network $Q_\theta(s,a)$ (as in the DQN algorithm) and a deterministic policy network $\mu_\phi(s)$. As in DQN, the replay buffer and target network tricks are used, the latter being a key ingredient of the method. Looking back at the DQN loss (5), it is obvious that the $\max_{a'} Q_\theta(s',a')$ term does not make sense in the context of a continuous action space. In DDPG, the latter is thus approximated using the target network, yielding the modified loss:

$$L(\theta) = \mathop{\mathbb{E}}_{(s,a,r,s')\sim\mathcal{D}} \left[\frac{1}{2} \left(\left[r(s,a) + \gamma\, Q_{\theta_{\text{targ}}}(s', \mu_{\phi_{\text{targ}}}(s')) \right] - Q_\theta(s,a) \right)^2 \right]. \tag{7}$$

Hence, the policy $\mu_\phi(s)$ is expected to produce actions corresponding to a maximum value predicted by the Q-network, and therefore its loss is obtained straightforwardly as:

$$L(\phi) = \mathop{\mathbb{E}}_{s\sim\mathcal{D}} \left[Q_\theta\left(s, \mu_\phi(s)\right) \right]. \tag{8}$$

Finally, a gaussian noise is usually applied to the predicted actions in order to achieve an efficient balance between exploration and exploitation.

2.3.6 Twin-delayed deep deterministic policy gradient (TD3)

The twin-delayed deep deterministic policy gradient (TD3) algorithm is a refinement of the DDPG method that improves its learning stability and robustness against hyper-parameters [32]. For the sake of brevity, only the differences between the two methods are pointed out here, namely:

- the use of a second Q-network to avoid the common problem of overestimation of the Q-value, as is done in DDQN [28];
- additional delays in the policy and target network updates;
- additional noises in the target actions.

Compared to standard DDPG, these three modifications largely improve the stability and performance of the method. Yet, as shown in table 4, these two methods have received little attention in the field of DRL-based control for fluid dynamics.

2.3.7 Soft actor-critic (SAC)

The soft actor-critic (SAC) algorithm shares similar traits with the TD3 algorithm, but presents two major differences with the latter:

- ◇ it exploits a stochastic policy and not a deterministic one;
- ◇ it maximizes a trade-off between the expected return and the policy entropy, thus efficiently balancing exploration and exploitation.

In the present review, a single article exploits this technique, as shown in table 4.

2.4 Single-step deep reinforcement learning

In several contributions assessed in this review, the optimal policy to be learnt by the neural network is state-independent, as is notably the case in optimization and open-loop control problems. We group here under the ”single-step DRL” label the class of algorithms dedicated to this class of problems under the premise that it may be enough for the neural network to get only one attempt per episode at finding the optimal. In essence, the proposed methods inherit from deep policy gradient algorithms in the sense that relevant probability density function parameters are obtained from neural networks trained using a policy gradient-like loss. Yet, they also fall heir of evolutionary strategies (ES), as their successive steps follow a generation/individual nomenclature, exploiting information from previous generations in order to update the parameters of a probability density function. The seminal PPO-1 algorithm proceeds from the standard PPO algorithm (section 2.3.4) and samples actions isotropically from scalar covariance matrices [17, 33, 34]. The follow-up policy-based optimization (PBO) algorithm relies on a variant of the vanilla policy gradient method and delivers several major improvements by adopting key heuristics from the covariance matrix adaptation evolution strategy (CMA-ES [35]), including the use of a valid, full covariance matrix generated from neural network outputs [36]. It is noted here that adjective ”deep” is used more as an extension of the original method name than as a real description of the networks depth. Indeed, due to the use of a fixed input state, these techniques exploit extremely small networks, with only a few tens of parameters in total.

3 Open challenges

Before delving into the specifics of the compiled papers, it is important to define a consistent list of challenges to serve as a common thread to measure the progress of DRL in the context of fluid mechanics applications (and also to examine the willingness of the community to take on these challenges). For those challenges left mostly unanswered, section 6 proposes a series of possible mitigation strategies that have received consideration in the literature, albeit in a different context. The retained challenges are computational cost (more generally, sampling-efficiency), turbulence, robustness, partial observability, delays, and any combination of them. Nonetheless, there are several other challenges that should be considered on the second level to help bridge the gap between DRL capabilities and the requirements of practical deployment, for instance multi-agent DRL (leveraging experience from multiple agents learning concurrently) or multi-objective reward (training an agent in reasoning about several weighted objectives), see, *e.g.* [37–39] for comprehensive domain-agnostic surveys.

∘ *Computational cost/sampling efficiency:* the environment of computational fluid dynamics (CFD) problems is resource expensive, as it routinely involves numerical simulations with tens or hundreds of millions of degrees of freedom (unless an appropriate low-dimensional reduction is achieved, which in itself often proves very challenging). This is all the more problematic since classical RL methods have low sample efficiency, *i.e.* many trials are required for the agent to learn a purposive behavior.

∘ *Stochasticity (turbulence):* most natural and engineered flows are turbulent and carry energy distributed over a wide range of scales with varying degrees of spatial and temporal coherence. Their dynamics therefore inherently includes some degree of stochasticity, which might lead to high variance gradient estimates that hamper learning.

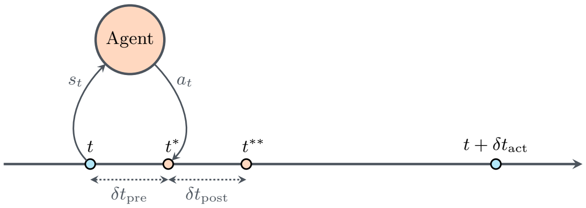

Agent
s_t
a_t
t
t^*
t^{**}
$t + \delta t_{\mathrm{act}}$
δt_{pre}
δt_{post}

Figure 4: **Different type of delays encountered in deep reinforcement learning environments.** *Pre-action delay* corresponds to the time delay existing between the moment of state collection and the moment when action is applied to the environment (this type of delay does not exist in the case of numerical environments). *Post-action delay* is defined as the time delay existing between the moment the action is applied to the environment, and the moment where it becomes fully effective in the dynamics of the system.

○ *Robustness:* optimizing robust policies is a key issue for fluid flow applications, with multiple sources of uncertainty relating to the occurrence of irregular transient dynamics and the high sensitivity to initial conditions and system parameters variations (all non-normal amplification mechanisms associated with the asymmetry of the Navier–Stokes convection operator), not to mention the difficulty to consistently ascertain the accuracy of the computed numerical solutions.

○ *Partial observability:* the traditional states space of fluid flow problems are easily prohibitively large for policy learning. The agent must therefore operate under partially observable environments, in which case the performance becomes highly dependent on the quality and relevance of the data available for observation. This issue is strongly related to data-driven model reduction techniques for large scale dynamical systems, which usually require using measures of observability as an information quality metric.

○ *Pre-action delays:* in numerical environments, states are collected in the environment, provided to the agent, and actions are returned instantaneously, which amounts to artificially interrupting the lapse of time every time the agent must draw new actions. In real-world environments, a certain delay is inevitable due to data processing, data transition, and physical constraints of sensors and actuators, during which the environment keeps evolving, meaning that the agent actually takes actions based on out-dated states.

○ *Post-action delays:* an environment has an intrinsic response time that depends on the interplay between transient amplification of the action-induced initial energy and non-linear saturation (the former all the more important in fluid mechanics where non-normal systems are common occurrence). This entails *post-action delays*, defined as the time interval between the moment an action is applied, and the moment it efficiently reaches the current state, that can undermine the accuracy of the reward estimation and even prevent learning if they exceed the Lyapunov time (the characteristic time scale on which a dynamical system is chaotic). The two types of delays defined above are illustrated in figure 4, the general picture being that post-action delays affect both numerical or experimental environment, while pre-action delays affect only experimental environments (unless they are purposely included in a numerical model).

4 Deep reinforcement learning for computational fluid dynamics

Of the 43 papers compiled in the present review, 40 consider applying DRL to computational fluid dynamic (CFD) systems. Those are classified and presented here in one of the categories listed in table 2, to put similar papers in perspective with respect to one another and to point out their specificities.

4.1 Drag reduction

Drag reduction is by far the most represented application domain in the literature, with 16 different papers implementing various control strategies using zero-mass-flow-rate jets [15, 40, 41, 44, 47–55], ro-

Table 2: **Classification of the reviewed papers by domain of application.** The most represented domain of application is drag reduction, with no less than 18 papers in total.

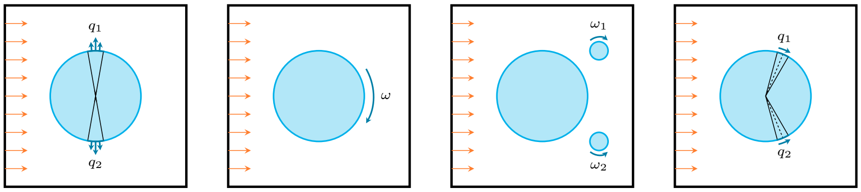
Category	Domain	Reference
Numerical	Drag reduction	[15, 33, 40–55]
	Heat transfer	[34, 56]
	Microfluidics	[57, 58]
	Swimming	[13, 14, 59–62]
	Shape optimization	[17, 63–65]
	Other	[16, 66–72]
Experimental	Drag reduction	[73]
	Flow separation	[74]
	Microfluidics	[58]
Review	-	[18, 19]

tating cylinders [33, 42, 43, 46], plasma actuators [45], or passive devices [33], as illustrated in figure 5. Almost all studies focus on prototypal, two-dimensional (2-D) incompressible flows past span-wise infinite cylinders (generally different sections of a single cylinder) subjected to a uniform and/or parabolic velocity profile, at the exception of [54], which considers cylinders with variable cross-sections, and [52], which considers the case of a NACA airfoil, again in a parabolic flow. As seen from the comparison between studies provided in table 3, most DRL algorithms belong to the actor-critic category, with a clear preference for ready-to-use PPO implementations (see section 2.3.4), either from Tensorforce [75], OpenAI baselines [76], or Stable Baselines [77]. Regarding the CFD solvers, FeniCS [78] is well represented, mostly because the open-source diffusion of the seminal work from Rabault *et al.* [15] has been heavily re-used in follow-up works [41, 43–46, 48, 51, 53]. Regarding the numerical implementation, since performing a relevant network update requires evaluating a sufficient number of actions drawn from the current policy (which in turn requires computing the same amount of rewards from resource-expensive numerical simulations), most studies have the agent acquire experience at a faster pace by interacting with multiple environments simultaneously. This has become a customary procedure after the methodological paper by Rabault and Kuhnle [41] on the accelerated gathering of state-action-reward transitions, which highlighted an almost perfect speedup up to 20 parallel environments, and a decent performance improvement up to 60.

Regarding the flow regimes, almost all contributions assume laminar conditions with Reynolds numbers in a range of one hundred to a few hundred. The only exceptions are [49, 52], where weakly turbulent flows at intermediate Reynolds numbers ($Re = 1000$ and $Re = 3000$ respectively) are explicitly targeted, and [33], where moderately large Reynolds number in the range of a few hundreds to a few ten thousands are tackled in the frame of Reynolds averaged Navier–Stokes (RANS) (see figure 6 for an illustration pertaining to the fluidic pinball, an equilateral triangle arrangement of rotating cylinders immersed in a turbulent stream). As stressed in [49], even weakly turbulent conditions make it significantly harder to achieve successful drag reduction, as evidenced by the increased number of episodes needed to learn an efficient policy at higher Reynolds numbers. Moreover, the use of transfer learning from strategies learned at $Re = 100$ to flow control at $Re = 1000$ is shown to be ineffective in this configuration, due to too different flow dynamics. Nonetheless, it is shown possible in [44] to achieve robust flow control over a range of Reynolds numbers by training simultaneously a single agent at four different Reynolds numbers distributed between 100 and 400. After training, the agent succeeds in efficiently reducing drag for Reynolds numbers in the range from 60 to 400, although the performance for each value of Re is slightly lower than that achieved training an agent specifically at this Reynolds number. Also, the onset of weak turbulence in the shear layer begins to affect the state space observed in the wake and triggers a high level of irregular drag and actuation fluctuations. In [47], the authors also underline the interest of using non-dimensionalized quantities as input states, which can increase the robustness of control strategies even learned at a single Reynolds number. Yet, as stated above

Table 3: **Summary of the main features of deep reinforcement learning drag reduction applications.** Only explicitly stated informations were retained from the contributions. Missing informations are noted by a question mark ”?”, while non-applicable data are noted with a double-dash ”–”. In the case where multiple studies were conducted in the considered paper with different parameter values, the most significant one was retained. n_{probes} corresponds to the number of sensors placed in the domain for observation collection, while n_{act} represents the action dimensionality. The information in the actor column refers to fully-connected network architecture used (in most contributions, the critic architecture is missing). Finally, the information in the last two columns pertain respectively to the ratio $\delta t_{\text{act}}/\delta t$ of the action to the simulation time-steps, and to the ratio $\delta t_{\text{phy}}/\delta t_{\text{act}}$ of the physical time scale (here equal to the vortex shedding period) to the action time-step. The acronyms for the DRL packages are the following: TFce = TensorForce, OAIb = OpenAI Baselines, StB = Stable Baselines.

Control	**Reference**	**Strategy**	*Re*	**DRL**	**CFD**	$n_{\textbf{probes}}$	$n_{\textbf{act}}$	**Actor**	$\delta t_{\textbf{act}}/\delta t$	$\delta t_{\textbf{phy}}/\delta t_{\textbf{act}}$
Active/Closed-loop	[15]	Jets	100	PPO (TFce)	FeniCS	151 (v)	1	$[512, 512]$	50	12
	[41]	≫	100	PPO (TFce)	FeniCS	151 (v)	1	$[512, 512]$	50	12
	[40]	≫	100	DDPG (in-house)	UPACS	1 (v)	1	$[400, 300]$	100	60
	[48]	≫	100	PPO (TFce)	FeniCS	63 (p)	1	$[512, 512]$	50	12
	[47]	≫	120	PPO (in-house)	FastS	3–16 (p)	1	$[512, 512]$	50	22
	[44]	≫	100–400	PPO (TFce)	FeniCS	236 (?)	2	$[512, 512]$	200	30
	[49]	≫	1000	PPO (in-house)	LBM	151 (v)	1	?	300	–
	[50]	≫	100	PPO (TFce)	Nek5000	86 (v)	1	$[512, 512]$	40	12–16
	[52]	≫	3000	PPO (TFce)	OpFm	? (p, v)	3	$[512, 512]$	?	?
	[53]	≫	400	DDPG (in-house)	FeniCS	5 (p)	2	$[128, 128]$	100	33
	[51]	≫	100	PPO (TFce)	FeniCS	5–11 (v)	1	$[512, 512]$	50	12
	[55]	≫	400	PPO (TFce)	FeniCS	236 (v)	3	$[512, 512]$	100	29
	[42]	Rotation	100	PPO (OAIb)	T-Flows	12 (p)	1	$[64, 64]$	30	20
	[46]	≫	100–200	PPO (TFce)	FeniCS	476 (p)	3	$[512, 512]$	85–350	10–20
	[43]	≫	240	PPO (TFce)	FeniCS	99 (v)	2	?	?	?
ActiveOpen-loop	[45]	Plasma	100	A2C (in-house)	FeniCS	10 (p)	1	$[128, 64]$	?	1
	[33]	Rotation	2200	PPO-1 (StB)	Cimlib	–	2–3	$[4, 4]$	–	–
Passive	[33]	Device	100-22000	PPO-1 (StB)	Cimlib	–	2–3	$[4, 4]$	–	–



(a) Lateral zero-mass-flow-rate jets.

(b) Main cylinder rotating.

(c) Downstream rotating control cylinders.

(d) Symmetric plasma actuators.

Figure 5: **Different drag reduction methods represented in the deep reinforcement learning literature, in the context of moderate Reynolds flows around a 2D circular cylinder.** (5a) Zero-mass-flow-rate jets are used to blow or suck fluid on the lateral sides of the obstacle. There can be two or four, possibly tilted. (5b) An angular velocity is applied to the obstacle, in order to alter the downstream flow and reduce drag. (5c) Two small control cylinders, placed downstream of the obstacle, are given angular velocities in order to stabilize the shedding of the main cylinder. (5d) Two symmetric plasma actuators are controlled to alter the fluid flow near the flow-separation point, thus reducing the overall drag on the obstacle.

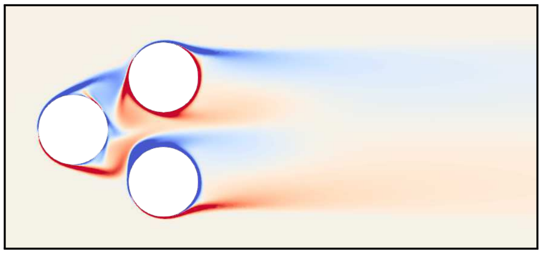


Figure 6: **Vorticity field of the fluidic pinball case considered in [33].** The three cylinders are free to rotate at different angular velocities, leading to complex flow features in the near wake region.

about the work of [49], this approach is limited to cases with similar flow patterns, and does not carry over to turbulent (not even weakly turbulent) flows. The case at Re=400 has been recently revisited by [55] using an original combination of Markov decision processes with time delays, to manage the time elapsed between the actuation and the flow response, by taking into account previous actuation informations in the agent's current decision, and autoregressive policy models, to handle the difficult exploration of the agent due to the presence of weak turbulence. In the case of Gaussian policy distribution, the action can be represented as a sum of deterministic parameterized mean and a scaled white noise. It is the white noise that can undermine the exploration behavior, as the mean usually varies smoothly between subsequent states. Autoregressive policy models replace the white noise component with a stationary autoregressive Gaussian process that has stationary standard normal distribution and exhibits temporal coherence between subsequent observations. Such an approach shows promise for turbulent flows as it is reduces the magnitude of drag and lift fluctuations by approximately 90% while achieving a similar level of drag reduction.

About the numerical reward, most contributions rely on the design proposed in [15]:

$$r_t = -\langle C_D \rangle - \beta \left| \langle C_L \rangle \right|, \tag{9}$$

where the operator $\langle \cdot \rangle$ indicates the sliding average over one vortex shedding period T [15,45,47,51,53] or over one action time-step δt_{act} [42,44,46,49,52]. The parameter β varies in a range from 0.2 to 1 in the papers reviewed in this section, and prevents the network to achieve efficient drag reduction by

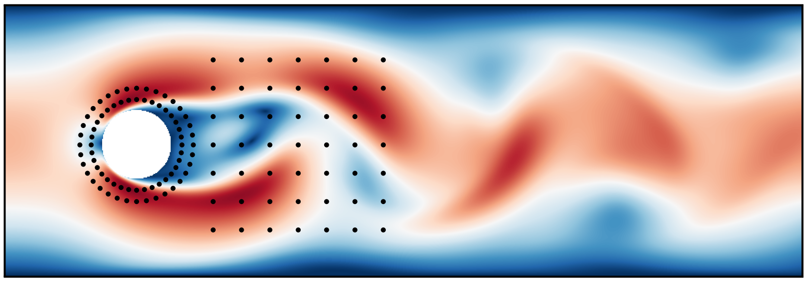


Figure 7: **Typical probe array for observation collection** in the context of drag reduction application on a 2D cylinder at moderate Reynolds numbers. Although the amount and position of probes vary, many contributions position probes in the vicinity of the obstacle and downstream of it. Insights on the impact of this choice regarding the performance of the agent can be found in [15] and [47].

relying on a large induced lift, as it is damageable in many practical applications. Whenever a different reward is used, penalization terms associated with the cost of the control are rarely considered (with reference [33] being an exception), as it has been found customary to explicitly bound the actuation amplitude for the control to remain small compared to the system relevant physical quantities. Of particular interest is the recent approach of [48] exploiting dynamic mode decomposition to design a reward function based on mode amplitudes, that has led to efficient control strategies, although the proposed approach supposes additional reward tuning compared to (9).

While the considered number of free action parameters n_{act} remains limited to 4 at most, the number of probes used to collect observations varies considerably from one contribution to another, even for similar setups. The baseline configuration consists of a certain number of velocity or pressure probes, uniformly distributed in the vicinity of the cylinder as well as in its wake region, as illustrated in figure 7. The sensitivity of the learned control strategy to the probes distribution has been briefly studied in [15]. A subsequent, more complete analysis has been performed in [47], where the authors evidence a critical impact on the control performance, the information provided by sensors positioned in the near wake being reportedly more relevant for learning than that of sensors positioned further downstream. This can be attributed to the fact that, by observing the flow closely downstream of the cylinder, the agent is able to observe the consequences of its actions right after they were taken, while more distant sensors provide a delayed feedback that can be more difficult to interpret. An extended sensitivity analyses proposed in [50] supports these results, showing that more efficient control laws are obtained when data is collected in the areas of high sensitivity. Yet, it most cases (at high Re values, for example), performing a sensitivity analysis of the problem may not be a possible option. To circumvent this issue, the authors in [47] introduce a specific method, called sparse PPO-CMA, in which an optimal set of sensors is automatically selected during the learning process. Another notable approach is that of [40], where only one probe is used downstream of the cylinder, collecting observations at a higher rate than the action time-step δt_{act}, and stacking them into a single observation vector when feeding them to the agent. In all relevant contributions, pressure or velocity are used indifferently as observations.

As stated previously, the frequency at which the agent is allowed to provide new actions to the environment is defined by an action time-step δt_{act}, that must be larger than the numerical simulation time-step δt (for the agent to be able to observe the effects of its actions on the environment), but smaller than the characteristic time scale δt_{phy} of the physical process to be controlled (for the actions taken to be able to significantly alter the flow dynamics). Considerations about numerical stability and accuracy of the numerical flow solution call for $\delta t \ll \delta t_{\text{phy}}$, meaning that the user has considerable leeway to adjust the action time step within these two bounds. In the reviewed contributions, the ratio of the action time-step to the physical time scale $(\delta t_{\text{act}}/\delta t_{\text{phy}})$ is in a range of a few tens (*i.e.* , a few tens of actions are taken per vortex shedding period). Meanwhile, the ratio of the action time-step to the simulation time-step $(\delta t_{\text{act}}/\delta t)$ varies considerably with the Reynolds number, as it is set to a few tens at $Re = 100$ [15, 42, 47], but increases up to a few hundreds at higher Reynolds values [44, 46, 49]. Between each interaction with the environment, an interpolation scheme is usually exploited to avoid

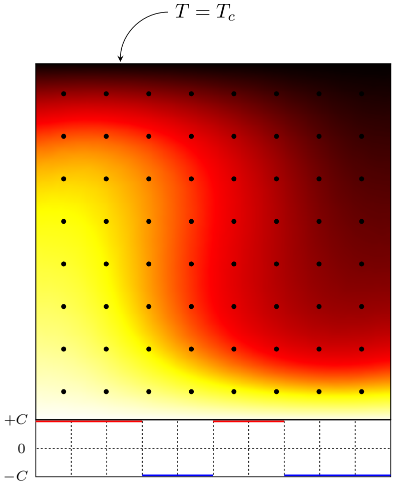


Figure 8: **Illustration of the Rayleigh-Bénard convection control setup** as presented in [56]. The top wall temperature is set to a constant temperature $T = T_c$, while the bottom temperature profile is cut in 10 segments on which the temperature can take values equal to $T_h + C$ or $T_h - C$. On the left and right walls, adiabatic conditions are imposed. Finally, the temperature and velocity fields are collected on a grid of probes equispaced in the computational domain.

abrupt control changes in the environment, which could cause numerical instabilities. The considered papers either use a simple linear interpolation (as in [44]), or an exponential decay, as in [15].

Finally, the agent architecture networks are also consistent between the different studies, with two fully-connected layers used in all cases. While seven contributions appear to use pretty large layer sizes [15, 40, 44, 46–48, 50, 52], smaller networks are successfully used for problems of similar dimensionalities [42, 45]. To the best knowledge of the authors, no large-scale study of the impact of the network architecture on the final agent performance was performed, and this choice remains most often empirical. Of particular interest is the use of a tailored single-step method in [33], that allows performing *passive* control with extremely small networks (see section 2.4), which is because the agent is not required to learn a complex state-action relation, but only a transformation from a constant input state to a given action.

4.2 Conjugate heat transfer

Although conjugate heat transfer systems governed by the coupled Navier–Stokes and heat equations seem natural candidates to extend the scope the DRL methodology and to increase the complexity of the targeted applications, the field has received little initial attention from the community. However, it may be starting gaining ground with two studies of DRL-based thermal control over the past two years.

In [56], the authors consider the closed-loop control of natural convection in a 2-D Rayleigh-Bénard convection cell simulated with an in-house lattice-Boltzmann code at Rayleigh numbers (based on the time-averaged temperature difference between the upper and lower plates) ranging from $Ra = 10^3$ (just before the onset of convection) to 10^7 (mild turbulence). The set-up, synthesized in figure 8, is as follows: the upper plate and the time-averaged lower plate temperature distributions are assumed constant. A discrete PPO agent whose implementation relies on OpenAI's stable-baselines [77] is then used to provide (after a normalization step) a zero-mean, piecewise-constant lower temperature fluctuation with the intent to reduce the convective effects. The actor is a fully-connected neural

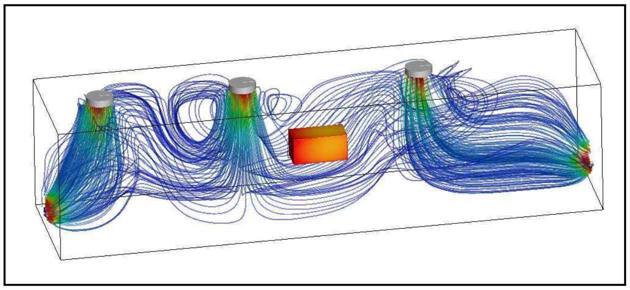


Figure 9: **Passive control of 3D forced convection in the context of workpiece cooling [34].** The positions of the three injectors are optimised in order to minimize the local temperature gradients in the workpiece during the cooling process. *Reproduced from E. Hachem, H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga; Deep reinforcement learning for the control of conjugate heat transfer with application to workpiece cooling; arXiv:2011.15035, 2020; licensed under a Creative Commons Attribution (CC BY) license.*

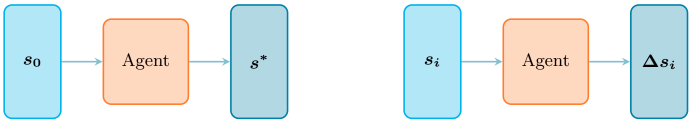
network with two hidden layers of width 64, and the instantaneous reward is defined as the opposite of the instantaneous Nusselt number Nu, which spurs the agent to minimize the convective effects at play. As in drag reduction applications, an array of probes is uniformy distributed over the computational domain to collect observations, under the form of both the temperature and velocity fields.

A particularity of this implementation is the systematic use of the four most recent observations in the state buffer passed to the agent, an approach similar to that of [40], although the authors do not provide insights about the impact of this choice on the agent performance. The agent is able to entirely stabilize the convective flow up to $Ra = 10^5$, and consistently outperform state-of-the-art linear approaches (proportional and proportional-derivative controllers) up to $Ra = 10^7$. Finally, the authors also illustrate the controllability limits of the system using the simplified Lorenz attractor system. By introducing a tunable artificial delay in the control, they show that exceeding half the Lyapunov time in delay results in a highly degraded performance of the learned control.

Passive control of a similar Rayleigh-Bénard natural convection problem is performed in [34] with the single-step approach presented in section 2.4. Compared to [56], the authors report excellent control efficiency using much smaller networks (two hidden layers of width 2 vs. 64) and less parallel environments (8 vs. 512) at $R_a = 10^4$, a value for which the optimal control determined in [56] ends up being actually time-independent (unlike at higher Rayleigh numbers). The authors then use the same approach and network architecture to minimize open-loop the inhomogeneity of temperature gradients across the surface of two and three-dimensional hot workpieces under impingement cooling in a closed cavity, identifying either optimal positions for cold air injectors relative to a fixed workpiece position, or optimal workpiece position relative to a fixed injector distribution (see illustration in figure 9).

4.3 Shape optimization

Shape optimization is another field fundamentally interrelated with flow control, that can seem as a natural domain application for the DRL techniques covered above. Nonetheless, it is worth noticing that shape optimization generally consists in determining a fixed shape meeting a set of required criteria (*e.g.* high lift-to-drag ratio, low pressure loss). This is not *per se* the original purpose of DRL, that aims at identifying optimal state-to-action relations (by means of neural network training) and is thus best suited to dynamically manipulate a deformable shape. Two approaches exist in the literature in the context of DRL-based shape optimization, a first one that directly optimizes state-independent shape parameters (hence, *direct shape optimization* [17]) and a second one that incrementally modifies an initial shape into an optimal one (hence, *incremental shape optimization* [63–65]). The conceptual differences between these two approaches are illustrated in figure 10, and their implementations are detailed in the following paragraphs.[3]



(a) Direct shape optimization.

(b) Incremental shape optimization.

Figure 10: **Direct and incremental DRL-based shape optimization techniques** present in the literature. In direct shape optimization (10a), the agent is used as a proxy to optimize a direct mapping from a constant, initial state vector $\boldsymbol{s_0}$ to the optimal state $\boldsymbol{s}^*$, using a degenerate, single-step DRL algorithm. In incremental shape optimization (10b), the agent learns the adequate mapping from the current state vector $\boldsymbol{s_i}$ to an incremental modification to apply to the latter, hence determining a path of incremental deformations to apply from $\boldsymbol{s_0}$ to $\boldsymbol{s}^*$. In both cases, multiple episodes are required for the agent to converge.

In direct shape optimization, the agent is used as a proxy to optimize a direct mapping from a constant, initial state vector $\boldsymbol{s_0}$ to the optimal state $\boldsymbol{s}^*$. This approach is implemented in [17] using single-step DRL (the degenerate class of DRL algorithms intended to optimize state-independent agent behavior) to design 2-D aerodynamic profiles without any *a priori* knowledge, feeding systematically an initial circle as input to the agent in single-step episodes (hence the adjective *stateless*). In practice, the shapes are described by a set of Bézier curves connecting the same number of control points, each with 3 free parameters (2 coordinates, plus a local curvature radius). As shown in figure 11, the agent is able to design airfoil-like shapes maximizing the lift-to-drag ratio at Reynolds numbers of about a few hundred, which takes between one and three thousand CFD evaluations (*i.e.* single-step episodes) for problem dimensionality ranging from 3 to 12, respectively.

The literature proposes three other DRL-based shape optimization contributions conversely relying on incremental shape transformations, with the incremental modifications in [63, 65] taking the current geometric parameters as input (of dimension 8 and 10, respectively), while the input states in [64] consist in a distribution of wall Mach number (of dimension 4). In all three contributions, a pre-trained surrogate or a simplified model is used, either for full agent training, or to perform an initial learning phase before re-training on a CFD environment using transfer learning. A key difference lies in the fact that the authors in [63, 65] always use the same input state and consequently produce a single optimized shape per training, while [64] relies on a set of input states randomly selected at the beginning of each episode, meaning that the trained agent can be successfully re-used in production on out-of-training input shapes.

From an algorithmic point of view, the choices are in line with those reported in the previous sections. All algorithms are actor-critic, either PPO [64] or DDPG [63, 65], using fully-connected networks with 2 or 3 layers of width from 200 to 512 neurons per layer, except for [17]. As it represents an arbitrary design choice in this specific application, the number of steps per episode is low, ranging from 5 to 20. A simple reward signal based on the lift-to-drag ratio is used in [63] and [17], but more complex designs were used in the other two contributions. In [64], the instantaneous reward is based on the difference of drag between the current and the previous generation, while [65] exploits a complex reward expression based on the results of a principal component analysis. Overall, it is extremely difficult to draw conclusions from these different approaches. Moving forward, a careful performance comparison between direct and incremental approaches constitutes a topic of outmost importance, but a more specific study focused on reward design could also be of great practical interest.

[3] Although not directly included in the scope of the current review, it is worth mentioning the work from Lampton *et al.* [79], who considered the use of standard Q-learning method for shape optimization in 2008. In this contribution, optimization of airfoil geometries with four free parameters is considered, and the optimal policy is obtained by updating a Q-table in a temporal-difference fashion.

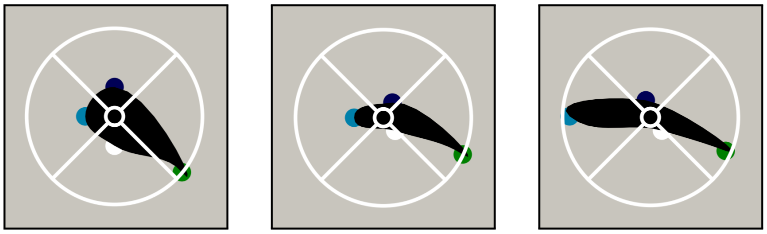


Figure 11: **Shapes of optimal lift-to-drag ratio obtained with direct shape optimization** with 3 free parameters (left), 9 free parameters (center) and 12 free parameters (right) [17]. The shape parameterization relies on Bézier curves joining control points, the agent controlling their position and local curvature radius. These shapes were obtained by learning an optimal mapping from a simple cylinder. *Reproduced from J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, E. Hachem; Direct shape optimization through deep reinforcement learning; arXiv:1908.09885, 2019; licensed under a Creative Commons Attribution (CC BY) license.*

4.4 Swimming

The control of swimmers has been a pioneering field for applying deep reinforcement learning to fluid mechanics problems, with a couple of contributions [13, 14] building on early seminal studies focusing RL for schooling [80, 81]. In [13], the kinematics of two swimmers in a leader-follower configuration are analyzed based on 2-D simulations of viscous incompressible flows. The first fish (leader) swims with a steady gait and the second fish (follower) uses DRL to adapt its behaviour dynamically to account for the effects of the wake encountered. The retained algorithm is DQN (see section 2.3.1), with input states made up of the lateral displacements and orientation of the follower compared to leader, as well as the two most recent actions, and the tail-beat status. An ϵ-greedy strategy is used to perform exploration, with randomness decaying from 0.5 to 0.1 over the course of learning. The reward design is straightforward, and increasingly penalizes the follower when it strays too far away from the leader path:

$$r_t = 1 - 2\frac{|\Delta y|}{L}, \tag{10}$$

where Δy is the aforementioned deviation, and L is the length of the swimmer. It takes roughly 100 000 transitions to learn the optimal behavior, and the results indicate that swimming in synchronized tandem (with the follower seeking to maintain its position in the center of the leader's wake, and its head synchronized with the vortices shed by the leader) can yield up to about 30% reduction in energy expenditure for the follower.

Reference [14] is a follow-up of [13] extended to 3-D schooling configurations, as illustrated in figure 12. A key contribution of this study is the use of a recurrent neural network, as the authors advertise (and demonstrate by providing performance comparisons with standard feedforward neural network) a greatly accelerated learning process using long-short term memory (LSTM) cells to encode the unsteadiness of the value function, which in turn is found to enable far more robust smart-swimmers. The retained recurrent network is composed of three layers of fully-connected LSTM units. The DQN algorithm with Adam optimizer is used to perform training in a temporal-difference manner, using again an ϵ-greedy exploration, with randomness decaying from 1 to 0.1. The training procedure requires 46 000 transitions (a reduction by roughly 50% with respect to the LSTM-less 2-D case). The results support the conjecture that swimming in formation is energetically advantageous, with the trained fishes showing collective energy-savings behaviors by appropriately placing themselves in appropriate locations in the wake of other swimmers and interacting judiciously with their shed vortices. An almost identical set-up (*i.e.* DQN algorithm exploiting an LSTM-based agent) is used in [60], to tackle a series of different swimming problems, namely (i) point-to-point travel in quiescent flow, with reward based

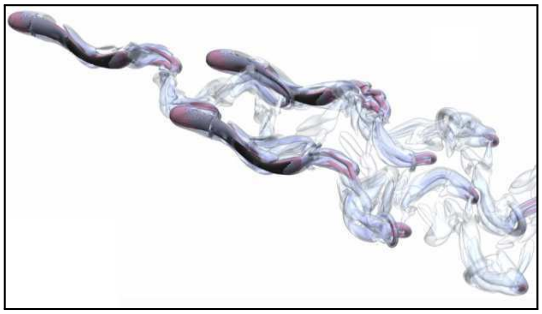


Figure 12: **Coordinated schooling of three swimmers [14].** The two followers interact with both rows of the wake shedding to increase their swimming efficiency. *Reproduced from S. Verma, G. Novati, P. Koumoutsakos; Efficient collective swimming by harnessing vortices through deep reinforcement learning; arXiv:1802.02674, 2018; licensed under a Creative Commons Attribution (CC BY) license.*

on normalized distance to target, (ii) holding a steady position in a rotating fluid flow, with reward based on averaged translation velocity of the fish center of mass, and (iii) holding a steady position in a Karman vortex street. The authors also emphasize the necessity to provide richer information to the agent to reduce variability over multiple episodes. The retained approach consists in feeding the agent with informations about the fish dynamics over the last four periods (*e.g.* , depending on the case, distance to objective, orientation of the swimmer, mean swimming velocities) and to add the actions taken over the same history steps, which indeed is found to yield stable learning and efficient swimming strategies. A similar setup (DQN with LSTM units) is employed in [62], coupled to an LBM solver. In this latter paper, a swimmer learns to reach a given destination located upstream of its position in a vortical flow. Two other contributions are to be noticed [59, 61], in which the authors consider the coupling of an actor-critic agent with a strongly-coupled fluid-solid interaction solver, based on the arbitrary lagrangian-eulerian method. In this context, the swimmer learns to perform several tasks, such as swimming along pre-defined curvilinear trajectories, or learning to avoid obstacles while reaching a target position.

4.5 Microfluidics

Micro-fluidics is one of the first fluid dynamics problems tackled with deep reinforcement learning techniques, but the related literature has since stalled to a single contribution from 2019 [57] (along with an additional experimental study [58] reviewed in section 5). In [57], the authors consider the inverse design problem of flow sculpting, in which a relevant sequence of micro-pillars is designed to controllably deform an initial flow field into a desired one. A double-DQN agent (section 2.3.1) is used that implements a convolutional policy, the full flow map being passed as input state [28]. The agent network is composed of three convolutional/max-pooling layers followed by three batch-norm/fully-connected layers. The DDQN is supplemented with an experience replay method [82]. The implemented reward is based on a pixel match rate (PMR) that measures the similarity of the current flow with the target flow. This contribution also contains an interesting analysis comparing DDQN performance with that of canonical methods, *e.g.* , genetic algorithms and brute force approaches.

4.6 Other applications

This section connects to other contributions of the literature applying DRL to more restricted subdomains of fluid mechanics, *e.g.* , turbulence model generation [69], sloshing suppression [68] or instability mitigation in fluids [16], among others. It is worth insisting that the scarcity of publications on these topics does not reflect a lack of interest or priority, but rather the suddenness with which DRL has opened up new opportunities for a wide range of applications, as was already clear from the previous sections. Note also, the literature features a few other publications pertaining to more peripheral domains of applications, *e.g.* energy efficiency [83, 84] or wave energy converters [85, 86].

These generally use low-dimensional models basically unrelated to the equations of fluid dynamics, and are thus not formally considered to keep the scope of the review well-defined.

4.6.1 Flow control

In an early contribution by Ma *et al.* in 2018 [66], a TRPO (section 2.3.4) agent learns to play different games (from rigid body balancing to complex music-playing games) based on the control of rigid body by steerable fluid jets. Regarding the environment, the Navier–Stokes equations are marched in time using a grid-based fluid-solid solver with adaptive refinement. A convolutional auto-encoder trained on-the-fly is used to efficiently extract fluid flow features from the environment. After their dimensionality has been reduced to an acceptable range, those are combined with rigid body features and serve as input for the agent, which is shown to significantly improve the learning speed compared to using rigid body features only. The state vector, whose size is lower than 100 elements, is fed to a standard fully-connected network of size $[128, 64, 64, 32]$, which yields typical training times in a range from 2 to 20 hours, depending of the game played.

Another under-represented type of application is the control of sloshing in tanks, despite obvious practical interest for engineering applications, such as liquid carriers in ground, marine, or air transport vehicles, as well as in earthquake excited water supply towers. Reference [68] is the only contribution in the field, that considers suppressing sloshing in a tank initially submitted to a sinusoidal excitation using two active controlled horizontal baffles. The comparison of two policy-gradient algorithms, namely PPO (section 2.3.4) and TD3 (section 2.3.6) is a key contribution of this study. The state information consists of the positions of the baffles, as well as the elevation and vertical velocities of two additional probes in the tank. Given such inputs, the agent provides in return the horizontal velocities to be applied to the baffles. For both algorithms, the actor is composed of a fully-connected network with two layers of width 64. Actions are taken by the agent every 30 numerical time-steps, one episode consisting in 200 actions, linearly interpolated from one time-step to the following. The reward is equal to the time-averaged sloshing height, plus a penalization term to limit the displacements of the baffles. Good convergence is reported both for PPO and TD3, although learning proves to be more stable using TD3, as shown in figure 13. With direct learning, the authors notice a lack of robustness when applying the learned strategy beyond the largest time used during training, which they show can be overcome using behavior cloning to pre-train the agent.

Another noteworthy contribution is that of Belus *et al.* [16], that introduces a technique based on invariants intended for problems with large dimensional (up to 20) actions spaces. In this study, a PPO agent (section 2.3.4) is used to mitigate the natural instabilities developping in a 1-D falling liquid film using small jets blowing orthogonally to the flow direction. The number of jets and their positions can vary, leading to different levels in control complexity. A three-layer, fully-connected network of size $[128, 64, 64]$ is used, with actions provided every 50 numerical time-steps to a variable number of jets, based on local inputs recorded in the vicinity of each jet. Finally, the reward function steers the agent to alleviate the waves arising from the instability of the flow. Three training methods are compared, that differ by their ability to handle a large number of control jets: (i) local states are concatenated and flattened before being fed to the actor, its output dimensionality being equal to the number of jets; (ii) a similar approach is used, but instead of being flattened, the input states are fed as is to a convolutional network; (iii) the vicinity of each jet is considered a local environment and used to provide some states and a reward to a unique agent. This latter approach relies on the translational invariance of the physical problem. It significantly enhances the amount of experience collected by the agent during an episode, which the authors show allows tackling large dimensional action spaces without increasing the amount of simulation time, as shown in figure 14.

Vibration suppression was also considered in the context of an academic test-case [70]. In this article, the authors consider the reduction of the vibration of a cylinder in a $Re = 100$ flow, its motion being constrained by a damping device and a spring. Similarly to the drag reduction cases, the cylinder is equipped with synthetic jets, which mass flow rate is controlled by the agent. The amplitude of the vibrations is successfully reduced by more than 80% using a soft actor-critic (SAC) agent, and these results are compared with those of an active learning approach.

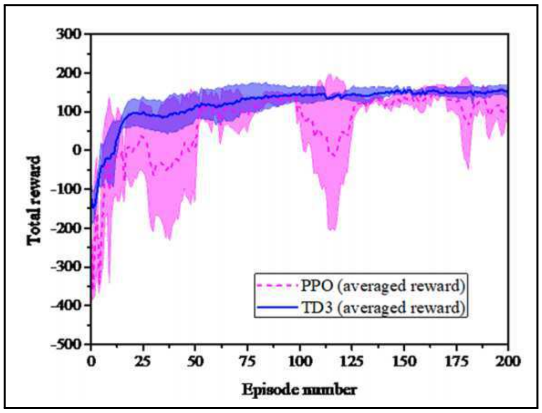
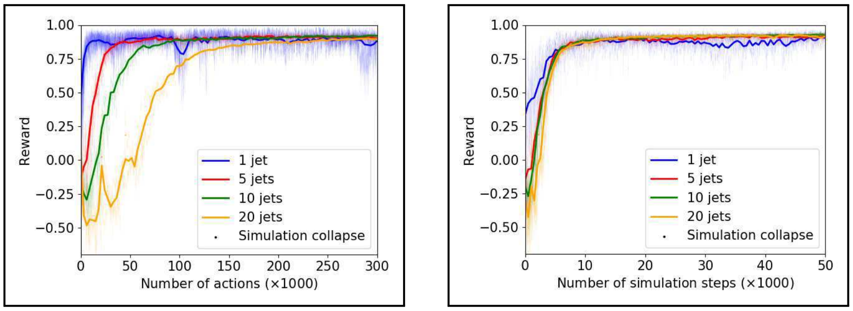


Figure 13: **Performance comparison of proximal policy optimization (PPO) and twin-delayed deep deterministic policy gradient (TD3) for sloshing suppression task** [68]**.** Although similar performance levels are obtained, the learning process proves to be more stable for TD3. *Reproduced from Y. Xie, X. Zhao; Sloshing suppression with active controlled baffles through deep reinforcement learning–expert demonstrations–behavior cloning process; Physics of Fluids, vol. 33, 2021; with the permission of AIP Publishing.*



(a) Naïve training method

(b) Training approach based on invariance

Figure 14: **Learning curves obtained with the naïve learning technique (**14a**) and with the invariance-based approach (**14b**)** [16]. On the left figure, it can be seen that increasing the number of jets significantly increases learning time, here counted in number of actions. On the right figure, similar training times (counted in simulation steps) are required, whatever the number of jets (and therefore the action space dimension) used to control the instability. *Reproduced from V. Belus, J. Rabault, J. Viquerat, Zhizhao Che, E. Hachem; Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film; AIP Advances 9, 125014 (2019), with the permission of AIP Publishing.*

4.6.2 Turbulence modeling

An approach somehow similar to that in [16] is used in another study by Novati *et al.* [69] to adjust the coefficients of an eddy viscosity closure model in the attempt to reproduce the energy spectrum of DNS computations. To this end, multiple agents are dispatched in the computational domain, with each agent controlling locally the dissipation coefficient of the Smagorinsky SGS model. The provided states are a blend of local (invariants of the gradient and Hessian of the velocity field) and global quantities (modes of the energy spectrum, rate of viscous dissipation, total dissipation). Two types of reward are proposed, based either on the Germano identity, or on a distance to a pre-computed DNS spectrum. The agent uses the remember-and-forget experience replay method, in which networks update are performed within a trust region, using a buffer holding the most recent transitions collected by the policy (which supposedly greatly improves the sample efficiency by enabling data to be reused multiple times for training) while dismissing those actions too unlikely under the current policy. The network parameters are shared between agents, and their aggregated experiences are collected in a shared dataset used for training. This approach was extended by [72] in order to build SGS models for wall-bounded turbulence, by introducing physical constraints in the training of a TD3 agent.

4.6.3 Differential equation resolution

An original contribution from Wei *et al.* [67] is concerned with the direct resolution of ordinary and partial differential equations by exploiting a modified actor-critic framework, and especially the resolution of the Navier-Stokes equations. In this paper, the authors rely on the actor to propose candidate solutions to the considered equation, while the ”critic” is in fact returning the residual of the candidate solution when injected in the governing equations of the problem. This approach shares similar traits both with standard actor-critic techniques, due to the way candidate solutions are generated, and with unsupervised physics-informed methods, due to the substitution of the critic with a computation of the residual of the candidate solutions. Excellent agreement is observed on several equations, such as the Schrödinger equation or the Navier-Stokes equations, among others.

5 Deep reinforcement learning and experimental fluid dynamics

The coupling of DRL and experimental fluid mechanics remains insufficiently explored, with only 3 out of the 32 papers compiled in this review applying DRL for experimental flow control purposes. Besides the possibly limited access to experimental devices for DRL practitionners, this is likely because several challenges such as controllability (the ability to efficiently reach a given state), observability (the ability to reliably measure changes in the state), sensitivity (to noise and system uncertainty) and system delays (see section 3) become increasingly important in experimental setups, even though they have received little attention in the context of idealized numerical environments.

5.1 Drag reduction

In [73], a drag reduction problem similar to that in figure 5c is considered, where an agent is given control over the angular velocity of two rotating cylinders located in the wake of a fixed principal cylinder. The Reynolds number is about $Re = 10^4$, and the agent is allowed to interact with the environment every 0.1 s. An entire episode last 40 s, plus additional time consumption for initialisation (4s, the time needed to wash out the transient before collecting any data) and for the reset procedure (2 mn, the time needed for the entire system to come back to rest). Overall, an experimental episode lasts between 3 and 4 mn. A TD3 agent (section 2.3.6) based on Tensorflow is used, the updates being performed only between episodes with a reward function similar to (9). The states provided to the agent are the drag and lift coefficients measured on the main cylinder and the two control cylinders (an approach noticeably different from that described in section 4.1). A key outcome of this study is the necessity to high pass filter the experimental states before they are fed to the agent, as a comparison of the performance with and without providing beforehand the experimental states as input to a Kalman filter shows that the agent is essentially unable to learn an efficient strategy without the filtering stage. Additional experiments are also performed to account for the power loss due to the friction of the control cylinders

5.2 Flow separation

In [74], a DQN agent (section 2.3.1) learns to perform flow reattachment behind a NACA 0015 airfoil by controlling the burst frequency of a plasma actuator at $Re = 6.3 \times 10^4$. Two different angles of attack are considered, namely 12° and 15°. The states provided to the agent consist of the unfiltered time-series data of the pressure at the surface of the airfoil, recorded through a set of 29 holes with high-frequency sensors, eventually downsampled to a total of 80 values. The actions are selected among a set of pre-defined burst frequencies, that includes four different values as well as an ”off” choice. The reward is zero if the flow is not attached, and one if it is attached, as determined from the pressure coefficient at the trailing edge of the airfoil. The DQN agent achieves a satisfactory learning at the first angle of attack of 12°, with efficient strategies available after as little as 200 episodes, although not more efficient that a naive open-loop control with adequately selected burst frequency. Conversely, the agent significantly outperforms the naive open-loop design at the second angle of 15°, but learning is then much more challenging and takes about up to 800 episodes.

5.3 Microfluidics

The problem considered in [58] relates to the performance of microfluidics experiment platforms when operated on extended periods of time. To overcome degraded flow stability beyond a certain timescale, the authors introduce a DRL agent to adjust the flow conditions and maintain the experiment operability *in an experimental device*. Two low-Reynolds applications are considered, namely the positioning of an interface between two miscible flows, and the dynamic control of the size of water-in-oil droplets within a segmented flow. On both applications, the performances of a DQN [26] agent and a model-free episodic control (MFEC) [87] are compared, although it must be noted that the algorithm run with different interaction frequencies (250 actions per episode for DQN, vs. 150 for MFEC) due to equipment limitations. Observations are obtained from a high-speed camera and processed into an 84×84 pixels frame. In the first experiment, the reward is obtained calculating the distance between the current observed interface and its target position, while in the second experiment it is computed from the estimated radii of the generated droplets. The authors find that DQN requires a considerable amount of frames (approximately 145 000 in the first experiment) to surpass human-level performance, albeit with large-scale fluctuations, while MFEC requires a reasonable number of frames to improve and reach a stable level of performance (approximately 11 000 frames in the first experiment), but does not reach the peak performance of DQN in the first case.

6 Transversal remarks

The contents of previous sections, although presented per application, helps identify trends regarding several technical aspects of state-of-the-art contributions in DRL for fluid flow problems. The present section underlines some of the latter, and raises open questions regarding possible future improvements in the field.

6.1 Taking on the challenges

Based on this review, we deem there is a good understanding of the key issues relevant to fluid flow problems. Many of the compiled references are primarily aimed at proving either feasibility in such or such sub-domain, or beyond state-of-the-art performance of such or such algorithm, but several milestone contributions assess the ability of novel developments to increase the complexity of the problems presented to the DRL agent. Among the challenges listed in section 3, computational efficiency [41], stochasticity [33, 49, 56] and partial observability [47] have received the most attention, but robustness and delays remain largely ignored (save for the unique combination of stochasticity and post-action time delays examined in [56]), even though real-world environments likely feature all mechanisms in strong interaction one with another. In particular, [55] demonstrates the necessity of including both a physics-informed delay and regressive models in the Markov decision process (not just one or the other) to achieve a robust and temporal-coherent control under weak turbulent conditions.

A lot has been achieved in a short period of time, but many related issues remain to be addressed for which the RL literature provides a number of a off-the-shelf methods already proved fruitful in different context (mostly robotics), that could help reach even higher levels of performance and robustness.

Table 4: **Usage frequency of different deep reinforcement learning algorithms in the articles considered in the present review.** Proximal policy optimization is obviously the most spread method, most probably due to several open-source releases.

Deep Q-networks (DQN)	5
Double deep Q-networks (DDQN)	2
Advantage actor-critic (A2C)	4
Proximal policy optimization (PPO)	15
Trust-region policy optimization (TRPO)	1
Deep deterministic policy gradient (DDPG)	4
Twin-delayed deep deterministic policy gradient (TD3)	3
Soft actor-critic (SAC)	1
Single-step PPO (PPO-1)/Policy-based optimization (PBO)	3
Others	3

Typical examples include learning a model of the environment in such a way that errors in the model do not degrade the asymptotic performance [88, 89], or wrapping redundant states into equivalent classes of canonical spaces [90] to increase the data efficiency; using data augmentation and randomization techniques to train over a wide distribution of states [91, 92] or partitioning the initial state distribution and training different policies later to be merged [93] to alleviate stochasticity; optimizing for worst case expected return objectives [94] or pursuing soft-robustness [95] to improve robustness; adding incentives to increase the policy entropy to provide more choices in solving a problem when situations are changed from the training, and thus to ease transfer learning [96]; using the frameworks of partially observable Markov decision process [97] and delay-aware Markov Decision Process [98] to account for partial observability and delayed dynamics.

6.2 Providing guidelines for the selection of the deep reinforcement learning algorithm

An obvious preference for policy gradient techniques appears from the review, with PPO the clear-cut go-to algorithm; see table 4. This is noteworthy because PPO is an on-policy algorithm, that updates the policy used to generate the training data (in contrast to off-policy algorithms, that also learn from data generated with other policies). PPO is generally acknowledged to improve the sample efficiency of regular actor-critic techniques, but there could be a fad component to this rise to prominence (partly attributable to the early open-source code release of several projects relying on this technique [15, 41, 44]), given that off-policy methods are expected to have even higher sample efficiency, and that most authors fail to explain the rationale for choosing a particular algorithm over another. Given the high CPU requirements of CFD solvers (that remains an important limitation regarding the application of DRL to 3-D flows of engineering importance), this calls for more careful, consistent and systematic testing of state-of-the-art on- and off-policy techniques in a fluid mechanics context. At the time of writing, only two such comparison studies are available in the literature, namely PPO vs. TD3 in [68], and DQN vs. MFEC in [58].

6.3 Fighting the reproducibility crisis

DRL a very fast-moving field, and as the number of contributions is growing, it becomes harder and harder to make a proper comparison between DRL algorithms, all the more so as a bevy of algorithms have been developed, to be used from dedicated libraries (*e.g.* Tensorforce [75], Stable Baselines [77], OpenAI Baselines [76]) or implemented in-house (which relates to 10 out of the 32 reviewed contributions). Compounding the matter are the high amount of time needed to train DRL agents, that creates a high barrier for reevaluation of previous work,; the general lack of complete information regarding the network architecture (*e.g.* size and depth of the hidden layers, activation functions, normalization, initialization) and training procedure (*e.g.* optimizer, batch size, number of epoch per update, update frequency, learning rate); and (for numerical environments) the additional variance in the numerical solutions themselves.

Encouraging the open sourcing of appropriate code on public git repositories is thus a critical step to ensure the reproducibility and durability of the developments, to maximize their impact, and to ultimately help establish DRL as a mature and stable technique for the analysis and design of complex flow systems. In this respect, it is disappointing to note that only 9 out of the 32 studies compiled in this review have come with such open-source releases [15–17, 41, 42, 44–46, 69]. Creating and providing exhaustive benchmark datasets and metrics is another alternative that would certainly add value to the community, and lay the ground for solid further developments in the field. The authors take this opportunity to underline the work by Wang *et al.* [71], that proposes a specific, ready-to-use platform for the coupling of DRL with CFD applications. This platform uses the Tensorforce library for the control side, and OpenFoam for the numerical solver.

6.4 Other research gaps

The present review has also allowed us to identify several other important gaps to consider when evaluating the progress of DRL for practically meaningful fluid mechanics.

∘ *Network architecture:* almost all provided references use fully-connected networks, with two or three hidden layers, each holding a number of neurons in the range from a few tens to a few hundreds. Nonetheless, our review did not reveal any large-scale study of the impact of the network architecture on the agent performance, and the choice remains most often empirical. The single-step method used in [33] is especially interesting in this regards, as it succeeds in learning optimal state-independent policies from extremely small networks. It should also be noted that the successful use of long-short term memory cells instead of regular fully-connected networks was advertised in swimming applications [14, 60], and that additional comparative experiments on different problems could lead to a more systematic use of such architectures.

∘ *State space dimensionality:* in some cases, state selection seems arbitrary, which can lead to either (i) incomplete observations or (ii) a too large inputs to the actor, which can be detrimental to learning. Specific methods have been proposed to tackle this issue, either by adding an intelligent state selection mechanism [47], or by exploiting state compression [66]. Shall they be pursued further, such efforts could lead to systematic techniques for state input from CFD environments.

∘ *Action space dimensionality:* in most contributions, the dimension of the action space remained limited, usually between 1 and 3. In this context, Belus *et al.* showed that exploiting the physical invariants of the problem was a particularly efficient way to tackle action spaces of larger dimensions (up to 20) [16].

∘ *Time granularity:* the frequency at which the agent interacts with its environment is usually set based on physical considerations, but the ratio of the typical physical time scale to the action time-step remains highly variable from one contribution to another (even for very similar cases; see table 3). Since this hyper-parameter can dramatically affect the attainable performance of the agent and the difficulty of the learning task (too large intervals lead to inefficient actions, while too small intervals hinder the learning process), the development of systematic selection criteria is another aspect that could benefit the community and help close the gap with real-world testing.

∘ *Scale effects:* due to the scarce literature that considers the application of DRL to experimental flow systems, the existence of potential scale effects when pivoting from idealized numerical systems to real physical models has never been assessed at the time of writing. Such effects arising from imperfect numerical modeling could lead to considerable deviation when model control laws are extrapolated to prototype values, which can easily, *e.g.* , impact wave dynamics or optimal propeller design.

∘ *Comparison with other control methods:* it is generally acknowledged that the main advantage for using DRL over more traditional control or optimization algorithms lies in its ability to reveal complex and unanticipated solutions or parameter relations, where most control strategies used in published works about active flow control rely on relatively simple harmonic or constant control input. That being said, the general literature considering the comparison of DRL control approaches with other control methods is extremely scarce. In the context of the coupling with fluid dynamics, only two references could be found that include comparisons of DRL with other approaches. For control, Pino *et al.* [53] propose a comparison of genetic programming (GP), Bayesian optimization (BO) and Lipschitz global optimization (LIPO) against DPPG on different cases, including the viscous Burgers equation and the 2D Turek problem. It was shown that GP and DDPG performed better than BO and LIPO, although DDPG had a better sample efficiency as well as a lower learning variance than GP. For optimization, the introductory PBO paper [36] compares the latter with the standard CMA-ES algorithm, showing that

PBO performs at least as well as CMA-ES. The pros and cons of DRL and canonical adjoint methods in the context of optimization problems are also discussed in [33]. Obviously, additional efforts from the community should be provided in the direction of broader and more systematic comparison with existing control techniques.

7 Conclusion

In the present review, the contributions of the last six years in the field of deep reinforcement learning applied to fluid mechanics problems were presented. The type of application, its complexity, the choice of control methods as well as their associated technical choices were analyzed and compared across the different contributions. Several trends and general rules of thumb currently in use in the domain were pointed out, while unusual choices and techniques were highlighted. This systematic work aims at providing a general frame of the existing usages and techniques to the researchers working in the domain, but also to help newcomers identify standard approaches and state-of-the-art performance level in the field of deep reinforcement learning-based control for fluid dynamics.

Overall, impressive performances were observed in multiple complex control tasks. Yet, a large amount of technical questions remain unanswered, and serious efforts remain to be provided by the community in order to efficiently tackle cases of industrial-level complexity within reasonable time. In the pursue of this goal, the access to efficient computational fluid dynamics solvers and to large computational resources remains an issue to many teams. In this perspective, the ability to successfully transfer agents from numerical to experimental environments remains to be explored more thoroughly, as the literature dealing with the coupling of deep reinforcement learning with experimental configurations remains, to this day, extremely scarce. It makes no doubt that the upcoming years will see the mastering of these obstacles, supported by the constant progress made in the deep reinforcement learning field and driven by the numerous industrial challenges that could benefit from it.

Acknowledgements

This work is supported by the Carnot M.I.N.E.S. Institute through the M.I.N.D.S. project.

References

[1] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: a comprehensive review. Neural Computation, 29:2352–2449, 2017.

[2] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi. A survey of the recent architectures of deep convolutional neural networks. Artificial Intelligence Review, pages 2352–2449, 2020.

[3] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural networks: a systematic review. IEEE Access, 7:19143–19165, 2019.

[4] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye. A review on generative adversarial networks: algorithms, theory, and applications. arXiv preprint arXiv:2001.06937, 2020.

[5] Steven L Brunton, Bernd R Noack, and Petros Koumoutsakos. Machine learning for fluid mechanics. Annual review of fluid mechanics, 52:477–508, 2020.

[6] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. arXiv preprint arXiv:1710.06542, 2017.

[7] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. arXiv preprint arXiv:1607.07086, 2016.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. Nature, 550, 2017.

[10] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. arXiv preprint arXiv:1807.00412, 2018.

[11] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall. Learning to drive from simulation without real world labels. arXiv preprint arXiv:1812.03823, 2018.

[12] W. Knight. Google just gave control over data center cooling to an AI. http://www.technologyreview.com/s/611902/google-just-gave-control-over-data-center-cooling-to-an-ai/, 2018.

[13] G. Novati, S. Verma, D. Alexeev, D. Rossinelli, W. M. van Rees, and P. Koumoutsakos. Synchronisation through learning for two self-propelled swimmers. Bioinspiration & Biomimetics, 12(3):036001, 2017.

[14] S. Verma, G. Novati, and P. Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. arXiv preprint arXiv:1802.02674, 2018.

[15] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. Journal of Fluid Mechanics, 865:281–302, 2019.

[16] V. Belus, J. Rabault, J. Viquerat, Z. Che, E. Hachem, and U. Reglade. Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film. AIP Advances, 9(12):125014, 2019.

[17] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem. Direct shape optimization through deep reinforcement learning. arXiv preprint arXiv:1908.09885, 2019.

[18] P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem. A review on deep reinforcement learning for fluid mechanics. Computers & Fluids, 225:104973, 2021.

[19] W. Zhang J. Rabault, F. Ren. Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization. Journal of Hydrodynamics, 32:234–246, 2020.

[20] R. S. Sutton and A. G. Barto. Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 2018.

[21] R. Bellman and S. E. Dreyfus. Applied dynamic programming. Princeton University Press Princeton, N.J, 1962.

[22] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256, 1992.

[23] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. Neural Networks, 2(5):359–366, 1989.

[24] H. T. Siegelmann and E. D. Sontag. On the computational power of neural nets. Journal of Computer and System Sciences, 50(1):132–150, 1995.

[25] I. Goodfellow, Y. Bengio, and A. Courville. The Deep Learning Book. MIT Press, 2017.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. Nature, 518, 2015.

[27] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2016.

[28] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. arXiv preprint arXiv:1509.06461, 2015.

[29] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783, 2016.

[30] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. arXiv preprint arXiv:1502.05477, 2015.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[32] S.Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477, 2018.

[33] H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga, and E. Hachem. Single-step deep reinforcement learning for open-loop control of laminar and turbulent flows. arXiv preprint arXiv:2006.02979, 2020.

[34] E. Hachem, H. Ghraieb, J. Viquerat, A. Larcher, and P. Meliga. Deep reinforcement learning for the control of conjugate heat transfer with application to workpiece cooling. arXiv preprint arXiv:2011.15035, 2020.

[35] N. Hansen. The cma evolution strategy: a rtutorial. arXiv preprint arXiv:1604.00772, 2016.

[36] J. Viquerat, R. Duvigneau, P. Meliga, A. Kuhnle, and E. Hachem. Policy-based optimization: single-step policy gradient method seen as an evolution strategy. arXiv preprint arXiv:2104.06175, 2021.

[37] G. Dulac-Arnold, D. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning. arXiv preprint arXiv:1904.12901, 2019.

[38] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning, pages 1–50, 2021.

[39] J. J. Garau-Luis, E. Crawley, and B. Cameron. Evaluating the progress of deep reinforcement learning in the real world: aligning domain-agnostic and domain-specific research. arXiv preprint arXiv:2107.03015, 2021.

[40] H. Koizumi, S. Tsutsumi, and E. Shima. Feedback control of karman vortex shedding from a cylinder using deep reinforcement learning. 2018 Flow Control Conference, 2018.

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

[42] M. Tokarev, E. Palkin, and R. Mullyadzhanov. Deep reinforcement learning control of cylinder flow using rotary oscillations at low reynolds number. Energies, 13(22), 2020.

[43] H. Xu, W. Zhang, J. Deng, and J. Rabault. Active flow control with rotating cylinders by an artificial neural network trained by deep reinforcement learning. Journal of Hydrodynamics, 32:254–258, 2020.

[44] H. Tang, J. Rabault, A. Kuhnle, Y. Wang, and T. Wang. Robust active flow control over a range of reynolds numbers using an artificial neural network trained through deep reinforcement learning. Physics of Fluids, 32(5):053605, 2020.

[45] M. A. Elhawary. Deep reinforcement learning for active flow control around a circular cylinder using unsteady-mode plasma actuators. arXiv preprint arXiv:2012.10165, 2020.

[46] M. Holm. Using deep reinforcement learning for active flow control. Master's thesis, University of Oslo, 2020.

[47] R. Paris, S. Beneddine, and J. Dandois. Robust flow control and optimal sensor placement using deep reinforcement learning. arXiv preprint arXiv:2006.11005, 2020.

[48] S. Qin, S. Wang, and G. Sun. An application of data driven reward of deep reinforcement learning by dynamic mode decomposition in active flow control. arXiv preprint arXiv:2106.06176, 2021.

[49] F. Ren, J. Rabault, and H. Tang. Applying deep reinforcement learning to active flow control in weakly turbulent conditions. Physics of Fluids, 33(3):037121, 2021.

[50] J. Li and M. Zhang. Reinforcement-learning-based control of confined cylinder wakes with stability analyses. arXiv preprint arXiv:2111.07498, 2021.

[51] R. Castellanos, G. Y. Cornejo Maceda, I. de la Fuente, B. R. Noack, A. Ianiro, and S. Discetti. Machine learning flow control with few sensors feedback and measurement noise. arXiv preprint arXiv:2202.12685, 2022.

[52] Y.-Z. Wang, Y.-F. Mei, and N. Aubry. Deep reinforcement learning based synthetic jet control on disturbed flow over airfoil. Physics of Fluids, 34:033606, 2022.

[53] F. Pino, L. Schena, J. Rabault, A. Kuhnle, and M. A. Mendez. Comparative analysis of machine learning methods for active flow control. arXiv preprint arXiv:2202.11664, 2022.

[54] Y.-F. Mei, C. Zheng, Y. Hua, Q. Zhao, P. Wu, and W.-T. Wu. Active control for the flow around various geometries through deep reinforcement learning. Fluids Dynamics Research, 54:015510, 2022.

[55] Y. Mao and S. Zhong and H. Yin. Active flow control using deep reinforcement learning with time delays in Markov decision process and autoregressive policy. Physics of Fluids, 34(5):053602, 2022.

[56] G. Beintema, A. Corbetta, L. Biferale, and F. Toschi. Controlling rayleigh–bénard convection via reinforcement learning. Journal of Turbulence, 21(9-10):585–605, 2020.

[57] X. Y. Lee, A. Balu, D. Stoecklein, B. Ganapathysubramanian, and S. Sarkar. A case study of deep reinforcement learning for engineering design: application to microfluidic devices for flow sculpting. Journal of Mechanical Design, 141(11), 2019.

[58] O. J. Dressler, P. D. Howes, J. Choo, and A. J. deMello. Reinforcement learning for dynamic microfluidic control. ACS Omega, 3(8):10084–10091, 2018.

[59] L. Yan and X. Chang and R. Tian and N. Wang and L. Zhang and W. Liu. A numerical simulation method for bionic fish self-propelled swimming under control based on deep reinforcement learning. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 234(17):3397–3415, 2020.

[60] Y. Zhu, F.-B. Tian, J. Young, J. C. Liao, and J. C. S. Lai. A numerical study of fish adaption behaviors in complex environments with a deep reinforcement learning and immersed boundary–lattice boltzmann method. Nature Scientific Reports, 11(1691), 2021.

[61] L. Yan and X. Chang and N. Wang and R. Tian and L. Zhang and W. Liu. Learning how to avoid obstacles: A numerical investigation for maneuvering of self-propelled fish based on deep reinforcement learning. International Journal for Numerical Methods in Fluids, 93:3073–3091, 2021.

[62] Y. Zhu, J.-H. Pang, and F.-B. Tian. Point-to-point navigation of a fish-like swimmer in a vortical flow with deep reinforcement learning. Frontiers in Physics, 10:870273, 2022.

[63] X. Yan, J. Zhu, M. Kuang, and X. Wang. Aerodynamic shape optimization using a novel optimizer based on machine learning techniques. Aerospace Science and Technology, 86:826–835, 2019.

[64] R. Li, Y. Zhang, and H. Chen. Learning the aerodynamic design of supercritical airfoils through deep reinforcement learning. arXiv preprint arXiv:2010.03651, 2020.

[65] S. Qin, S. Wang, L. Wang, C. Wang, G. Sun, and Y. Zhong. Multi-objective optimization of cascade blade profile based on reinforcement learning. Applied Sciences, 11(1), 2021.

[66] P. Ma, Y. Tian, Z. Pan, B. Ren, and D. Manocha. Fluid directed rigid body control using deep reinforcement learning. ACM Transactions on Graphics, 37(4), 2018.

[67] S. Wei and X. Jin and H. Li. General solutions for nonlinear differential equations: a rule-based self-learning approach using deep reinforcement learning. arXiv preprint arXiv:1805.07297, 2019.

[68] Y. Xie and X. Zhao. Sloshing suppression with active controlled baffles through deep reinforcement learning–expert demonstrations–behavior cloning process. Physics of Fluids, 33(1):017115, 2021.

[69] G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos. Automating turbulence modelling by multi-agent reinforcement learning. Nature Machine Intelligence, 3:87–96, 2021.

[70] C. Zheng, T. Ji, F. Xie, X. Zhang, H. Zheng, and Y. Zheng. From active learning to deep reinforcement learning: intelligent active flow control in suppressing vortex-induced vibration. Physics of Fluids, 33:063607, 2021.

[71] Q. Wang, L. Yan, G. Hu, C. Li, Y. Xiao, H. Xiong, J. Rabault, and B. R. Noack. Drlinfluids - an open-source python platform of coupling deep reinforcement learning and openfoam. arXiv preprint arXiv:2205.12699, 2022.

[72] J. Kim, H. Kim, J. Kim, and C. Lee. Deep reinforcement learning for large-eddy simulation modeling in wall-bounded turbulence. arXiv preprint arXiv:2201.09505, 2022.

[73] D. Fan, L. Yang, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. Proceedings of the National Academy of Sciences, 117(42):26091–26098, 2020.

[74] S. Shimomura, S. Sekimoto, A. Oyama, K. Fujii, and H. Nishida. Closed-loop flow separation control using the deep q-network over airfoil. AIAA Journal, 58(10):4260–4270, 2020.

[75] A. Kuhnle, M. Schaarschmidt, and K. Fricke. Tensorforce: a tensorflow library for applied reinforcement learning. https://github.com/tensorforce/tensorforce, 2017.

[76] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

[77] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[78] M. S. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The fenics project version 1.5. Archive of Numerical Software, 3(100), 2015.

[79] A. Lampton, A. Niksch, and J. Valasek. Morphing airfoils with four morphing parameters. In AIAA Guidance, Navigation and Control Conference and Exhibit, 2008.

[80] M. Gazzola, B. Hejazialhosseini, and P. Koumoutsakos. Reinforcement learning and wavelet adapted vortex methods for simulations of self-propelled swimmers. SIAM Journal of Scientific Computing, 36:622–639, 2014.

[81] M. Gazzola, A. A. Tchieu, D. Alexeev, A. de Brauer, and P. Koumoutsakos. Learning to school in the presence of hydrodynamic interactions. Journal of Fluid Mechanics, 789:726–749, 2016.

[82] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. arXiv preprint arXiv:1707.01495, 2018.

[83] H. Kazmi, F. Mehmood, S. Lodeweyckx, and J. Driesen. Gigawatt-hour scale savings on a budget of zero: Deep reinforcement learning based optimal control of hot water systems. Energy, 144:159–168, 2018.

[84] T. Zhang, J. Luo, P. Chen, and J. Liu. Flow rate control in smart district heating systems using deep reinforcement learning. arXiv preprint arXiv:1912.05313, 2019.

[85] E. Anderlini and D.I.M. Forehand and E. Bannon and Q. Xiao and M. Abusara. Reactive control of a two-body point absorber using reinforcement learning. Ocean Engineering, 148:650–658, 2018.

[86] L. Bruzzone and P. Fanghella and G. Berselli. Reinforcement learning control of an onshore oscillating arm wave energy converter. Ocean Engineering, 206:107346, 2020.

[87] C. Blundell, B. Uria, A. Pritzel, Y. Li, A. Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. arXiv preprint arXiv:1606.04460, 2016.

[88] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. arXiv preprint arXiv:1805.12114, 2018.

[89] J. Buckman, D; Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. arXiv preprint arXiv:1807.01675, 2018.

[90] C. Wu, A. Kreidieh, E. Vinitsky, and A. M. Bayen. Emergent behaviors in mixed-autonomy traffic. In Conference on Robot Learning, pages 398–407, 2017.

[91] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), pages 23–30, 2017.

[92] K. Lee, K. Lee, J. Shin, and H. Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. arXiv preprint arXiv:1910.05396, 2019.

[93] D. Ghosh, A. Singh, A. Rajeswaran, V. Kumar, and S. Levine. Divide-and-conquer reinforcement learning. arXiv preprint arXiv:1711.09874, 2017.

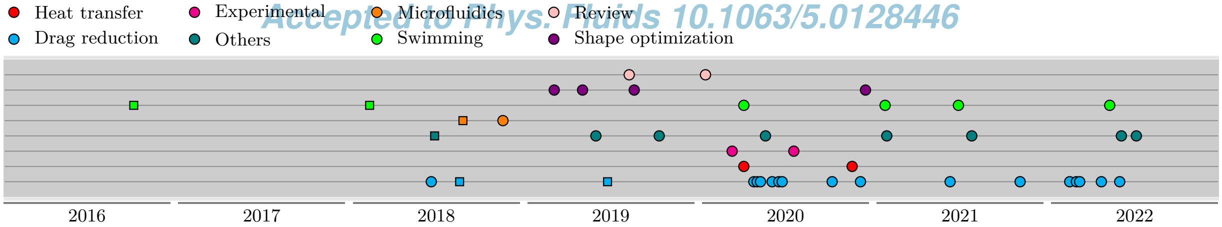
[94] D. J. Mankowitz, N. Levine, R. Jeong, Y. Shi, J. Kay, A. Abdolmaleki, J. T. Springenberg, T. Mann, T. Hester, and M. Riedmiller. Robust reinforcement learning for continuous control with model misspecification. arXiv preprint arXiv:1906.07516, 2019.

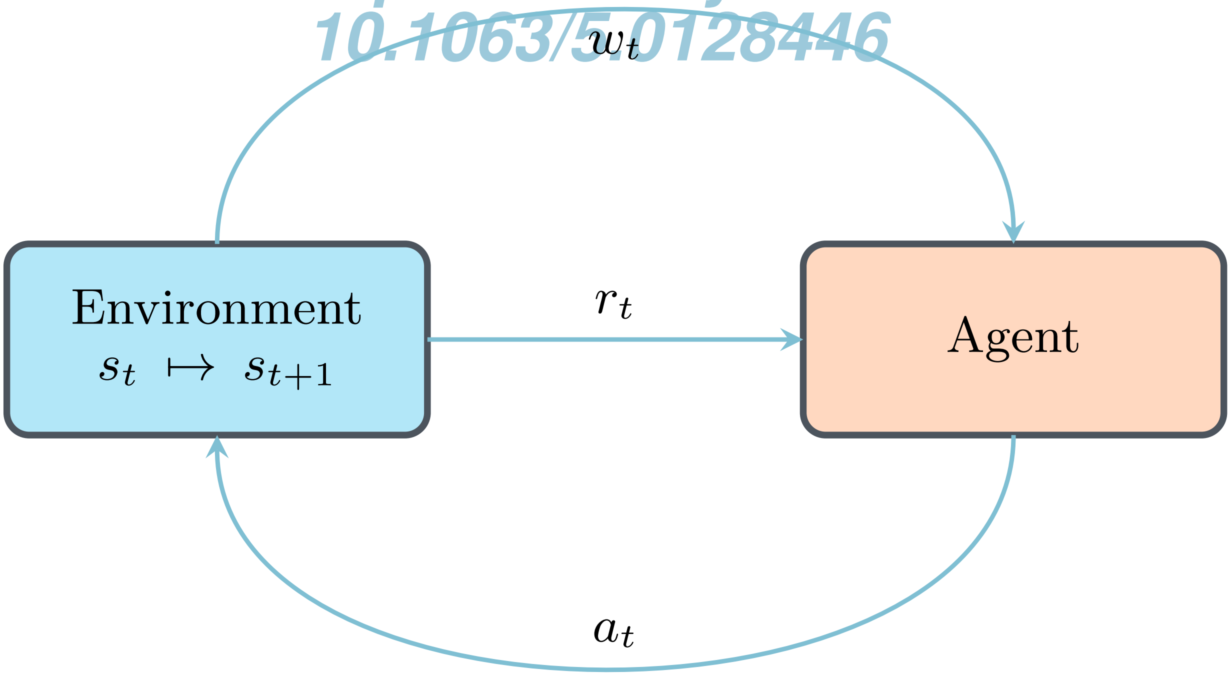
[95] E. Derman, D. J. Mankowitz, T. A. Mann, and S. Mannor. Soft-robust actor-critic policy-gradient. arXiv preprint arXiv:1803.04848, 2018.

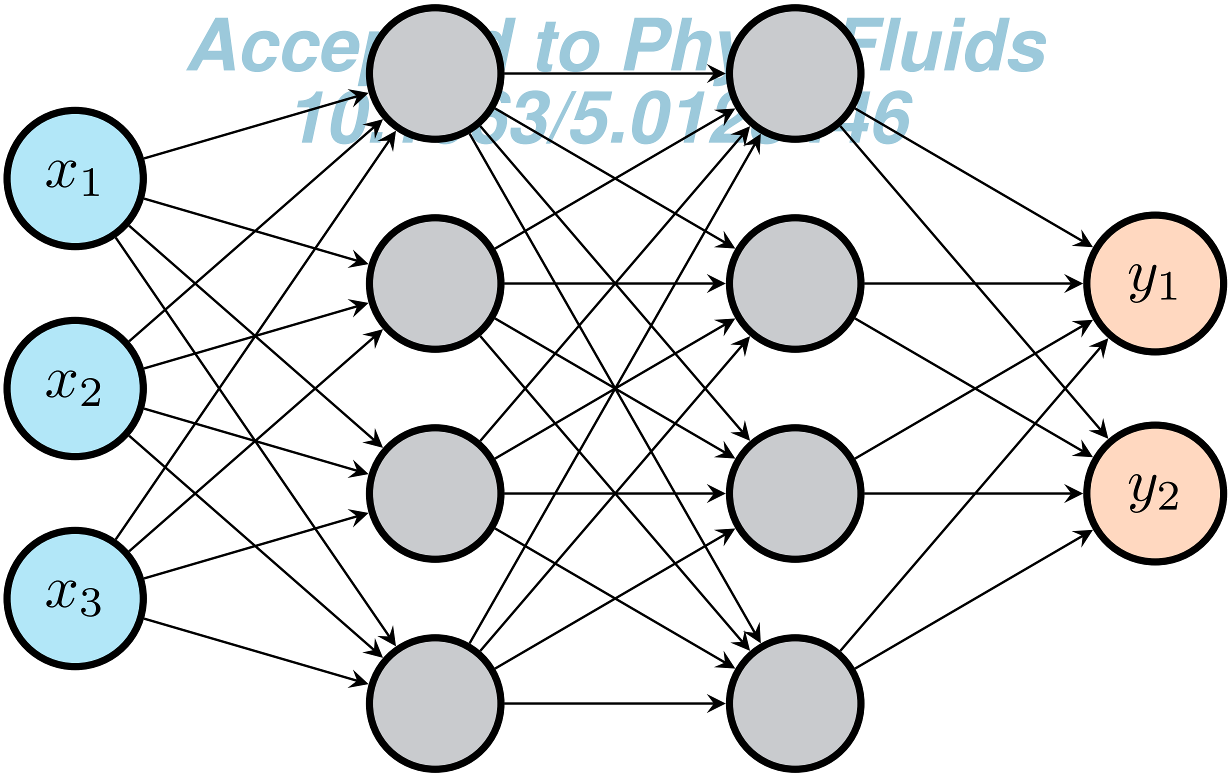
[96] Benjamin Eysenbach and Sergey Levine. Maximum entropy rl (provably) solves some robust rl problems. arXiv preprint arXiv:2103.06257, 2021.

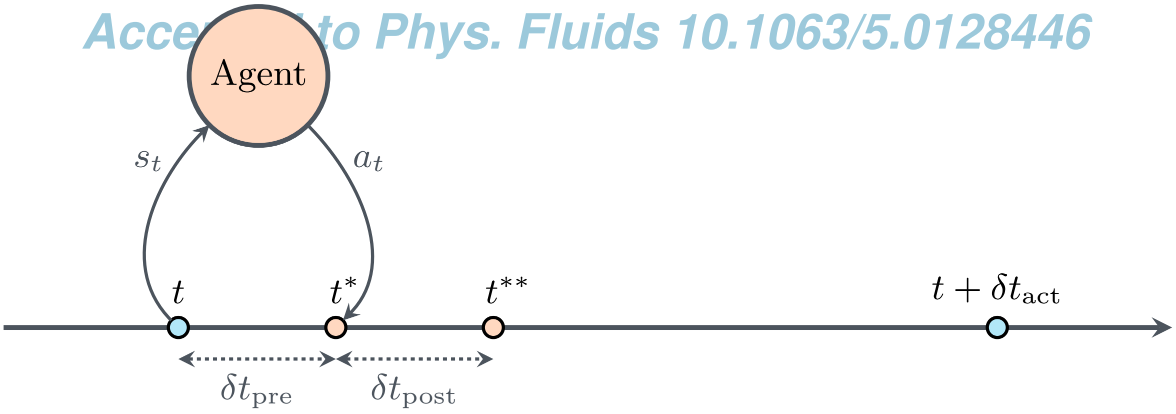
[97] A. R. Cassandra, L. P. Kaelbling, and M. L. Littman. Acting optimally in partially observable stochastic domains. In AAAI, volume 94, pages 1023–1028, 1994.

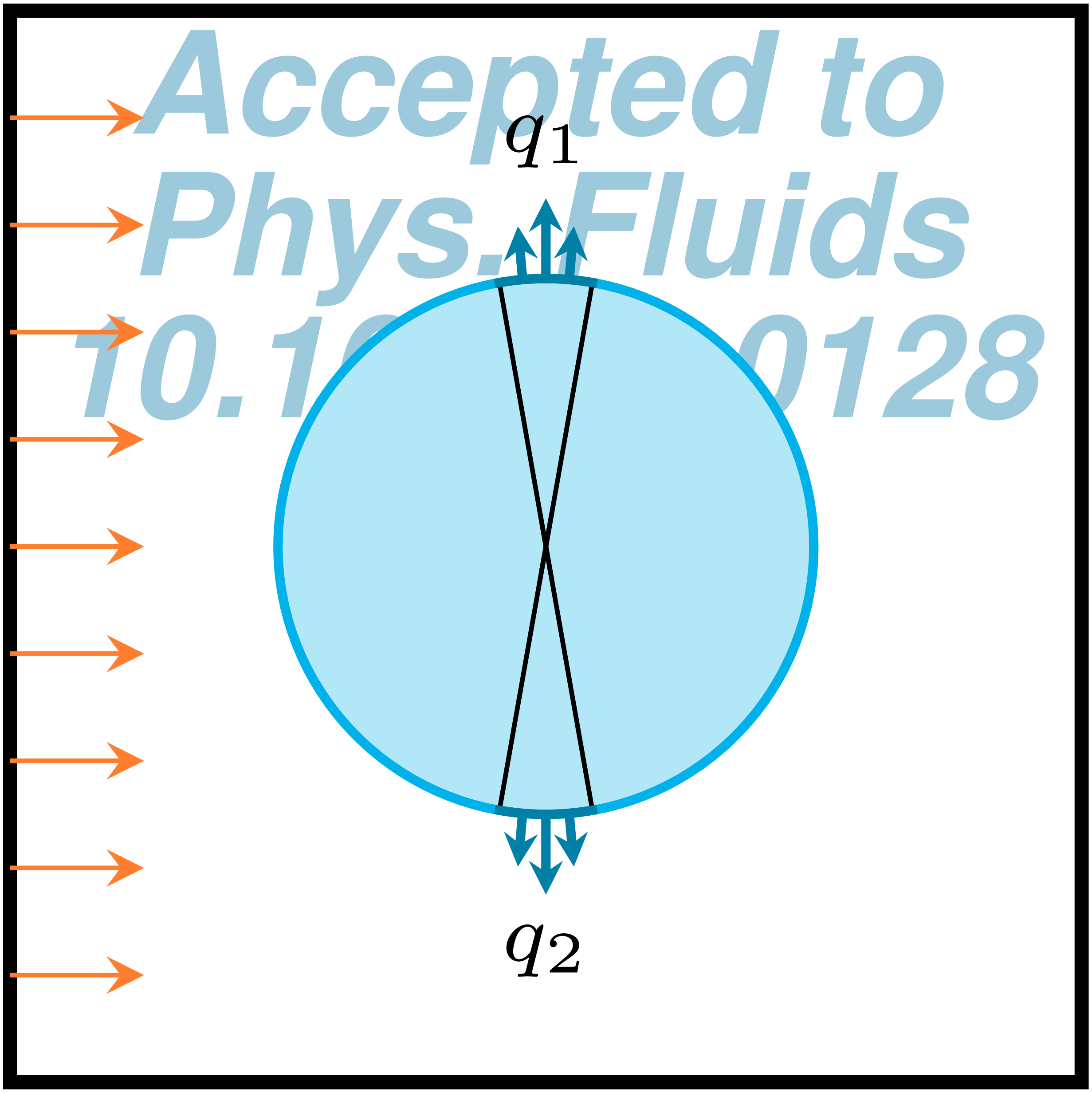
[98] B. Chen, M. Xu, L. Li, and D. Zhao. Delay-aware model-based reinforcement learning for continuous control. Neurocomputing, 450:119–128, 2021.

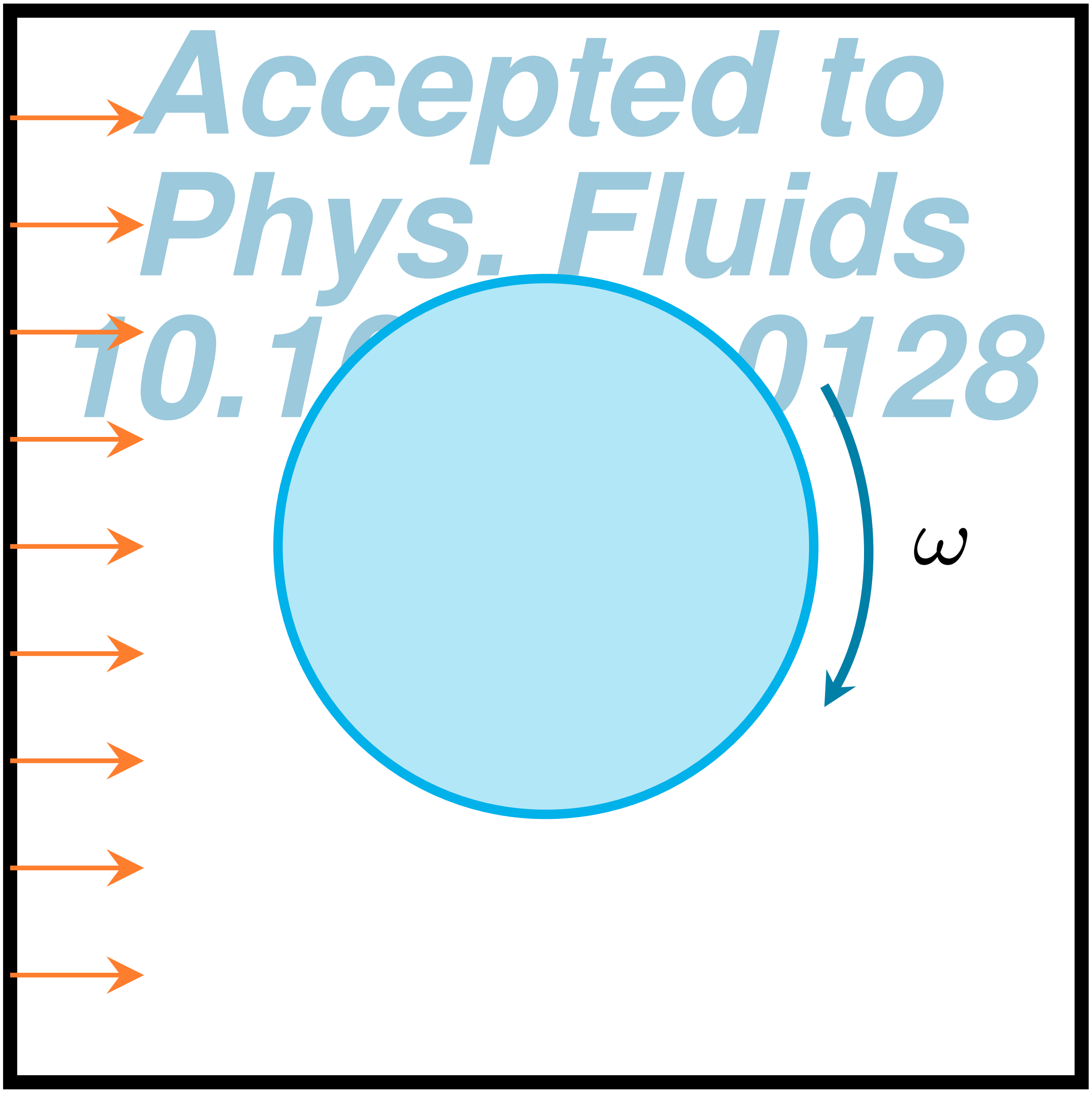

● Heat transfer
● Drag reduction
● Experimental
● Others
● Microfluidics
● Swimming
○ Review
● Shape optimization
2016
2017
2018
2019
2020
2021
2022

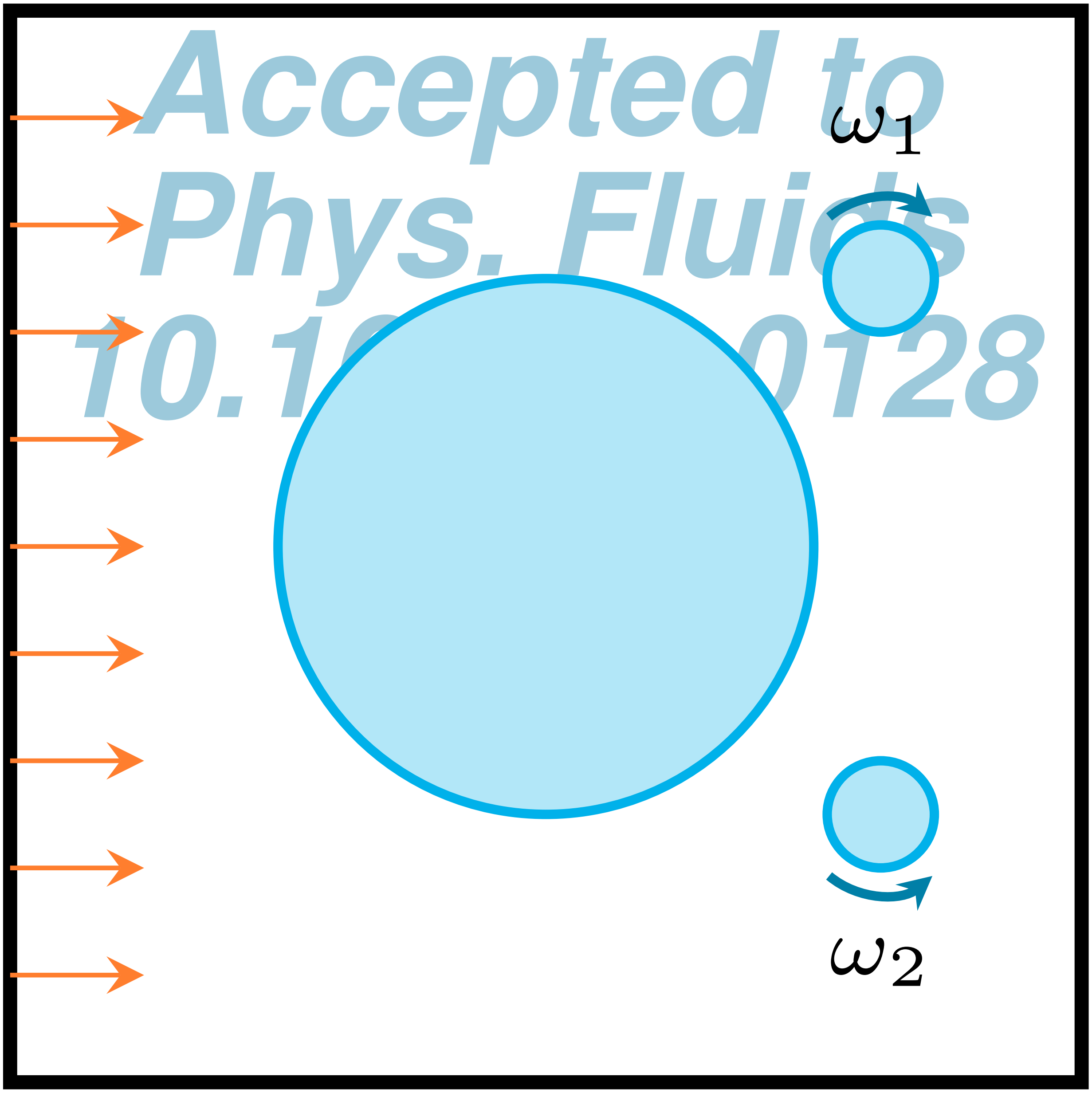

w_t
Environment
$s_t \mapsto s_{t+1}$
r_t
Agent
a_t

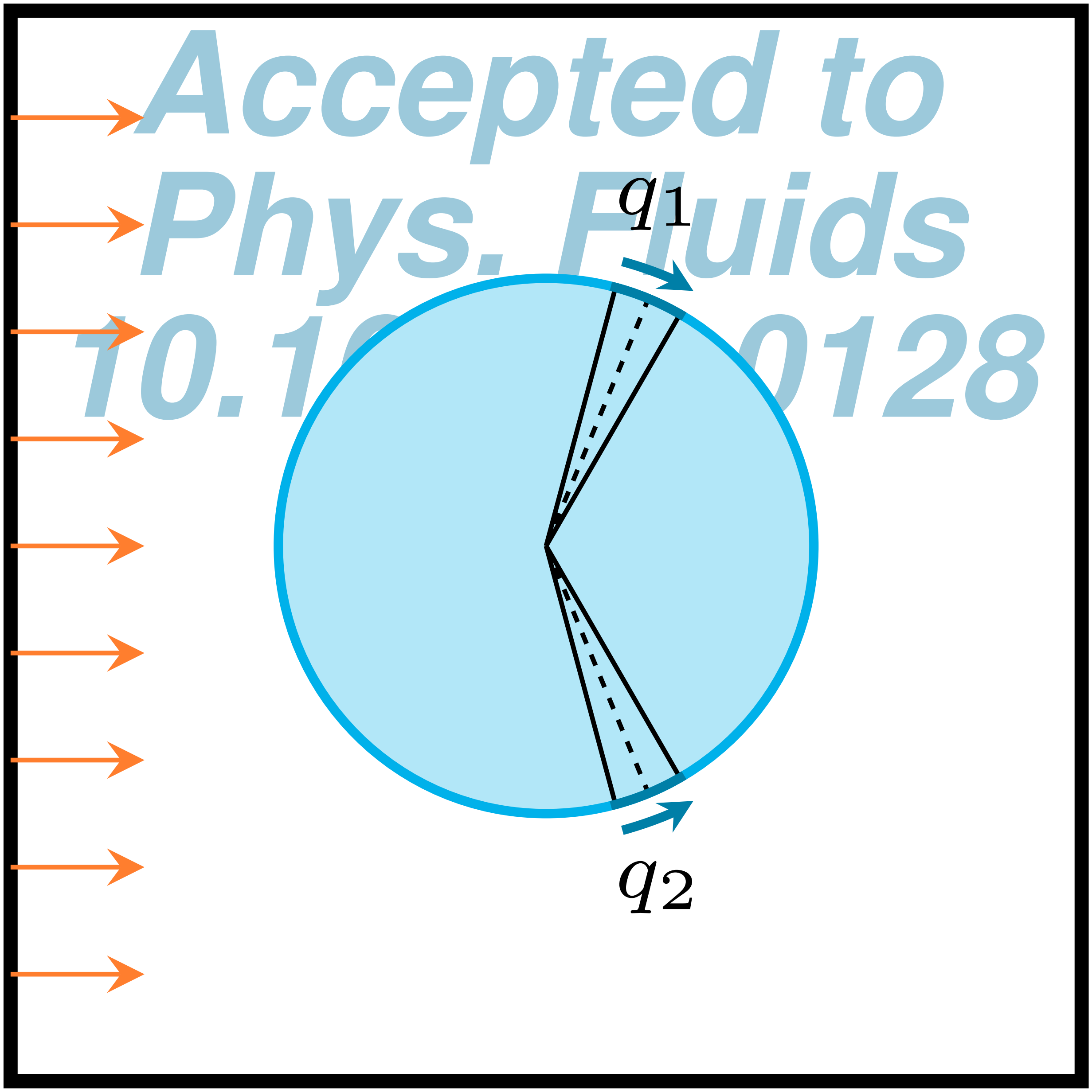

x_1
x_2
x_3
y_1
y_2

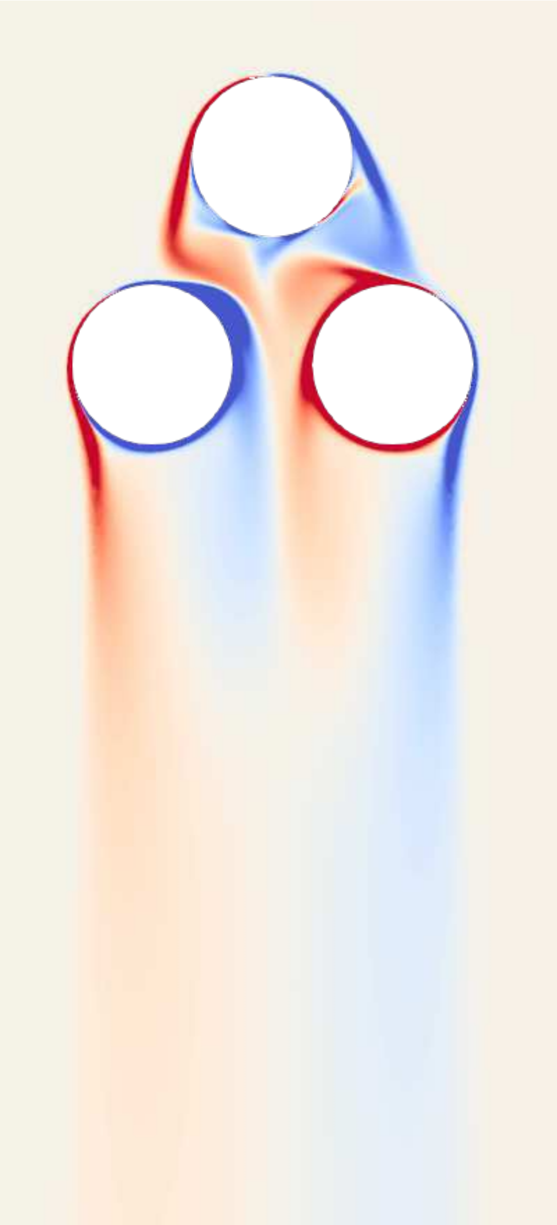

Agent
s_t
a_t
t
t^*
t^{**}
$t + \delta t_{\mathrm{act}}$
δt_{pre}
δt_{post}

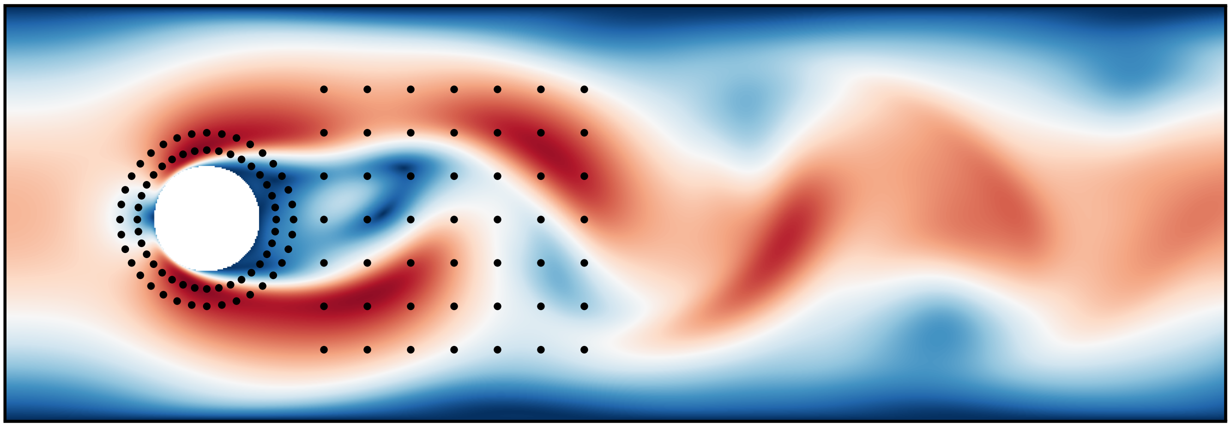

q_1
q_2

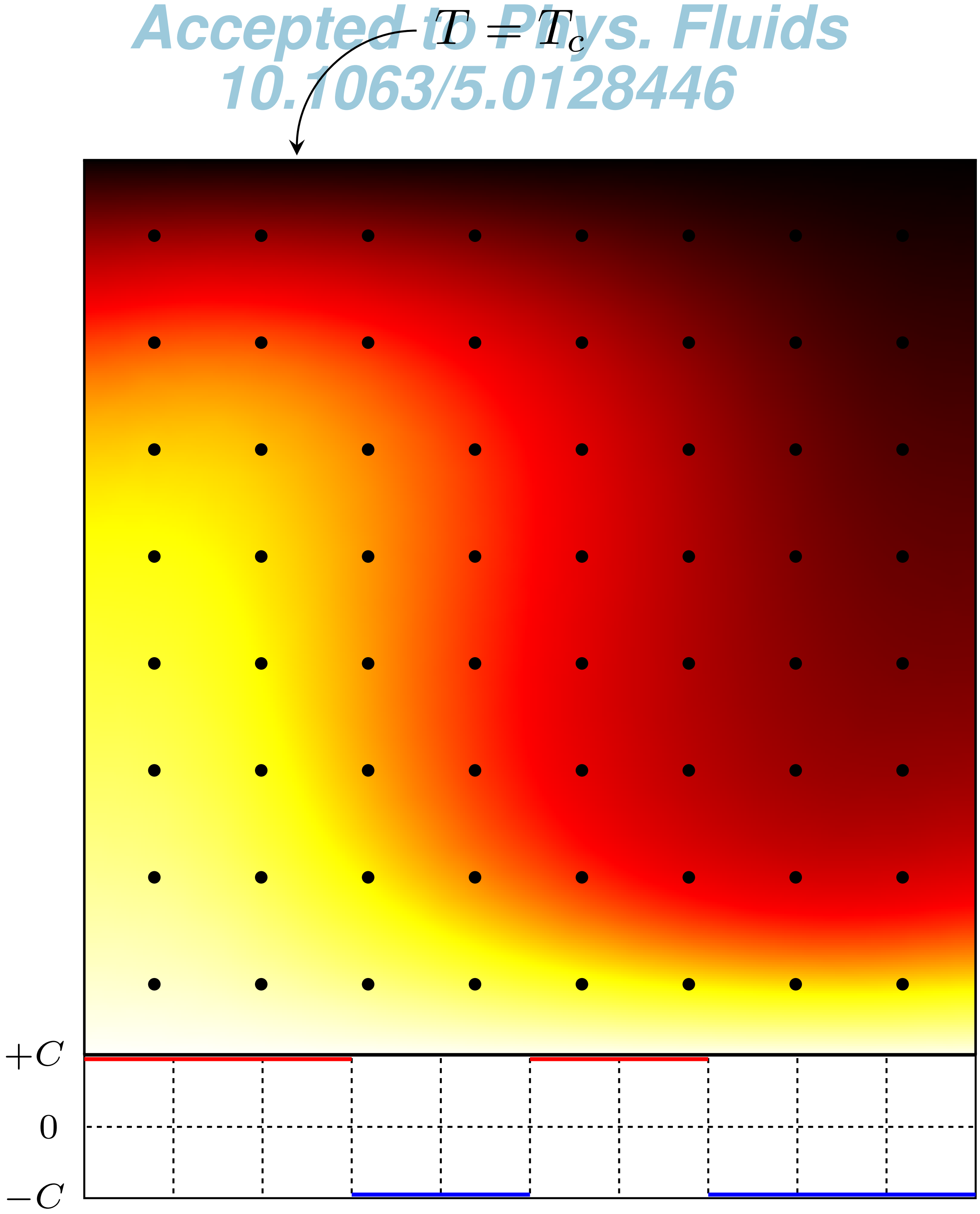

ω

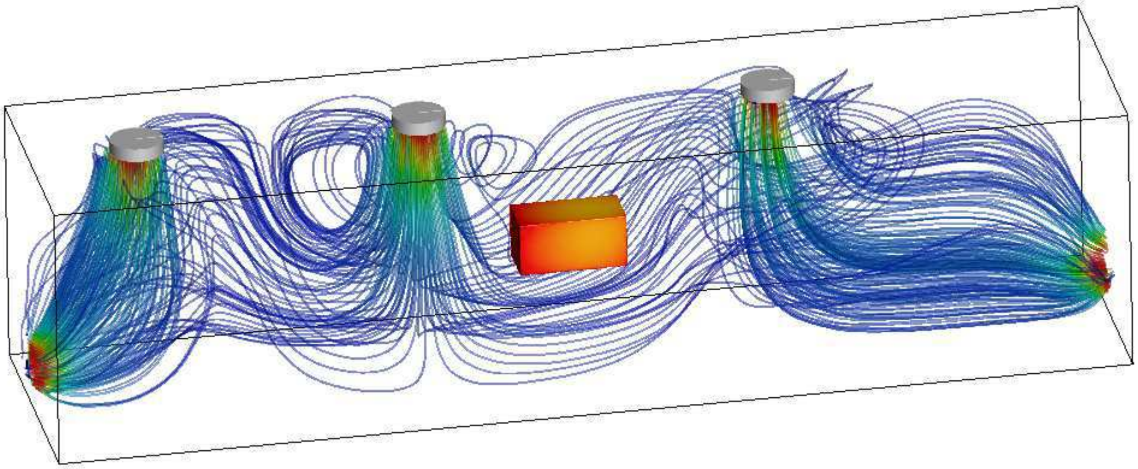

ω_1
ω_2

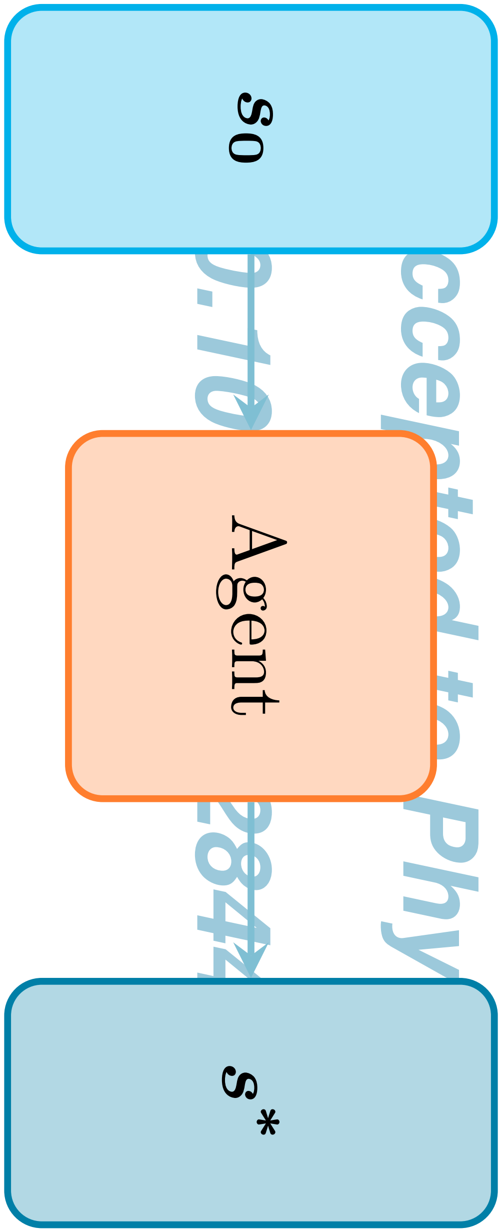

q_1
q_2

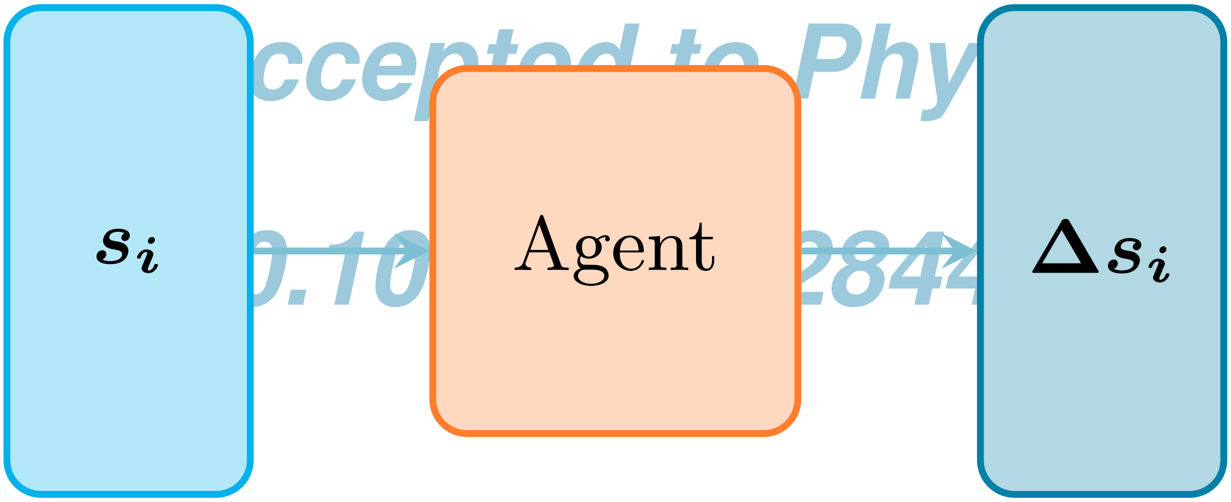


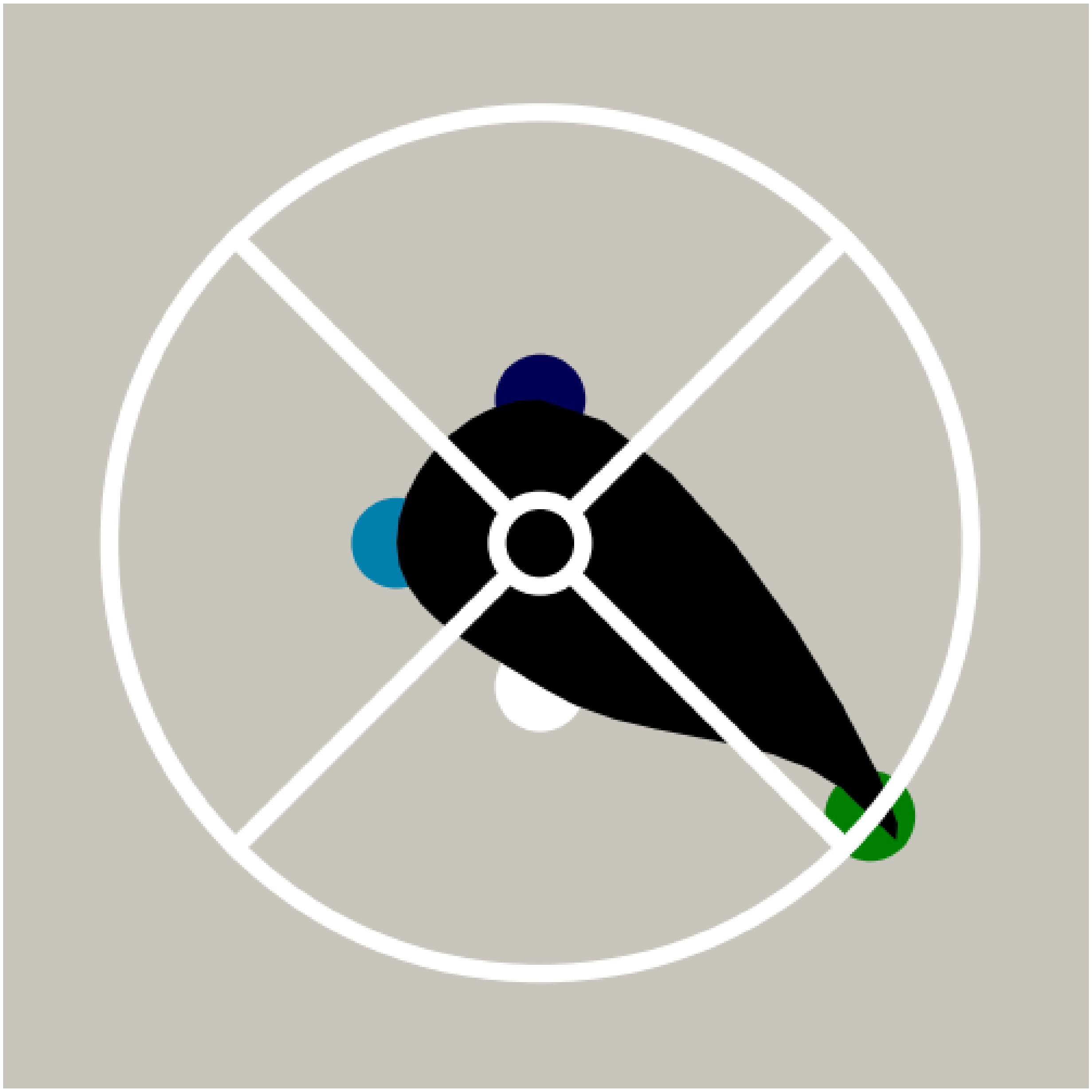


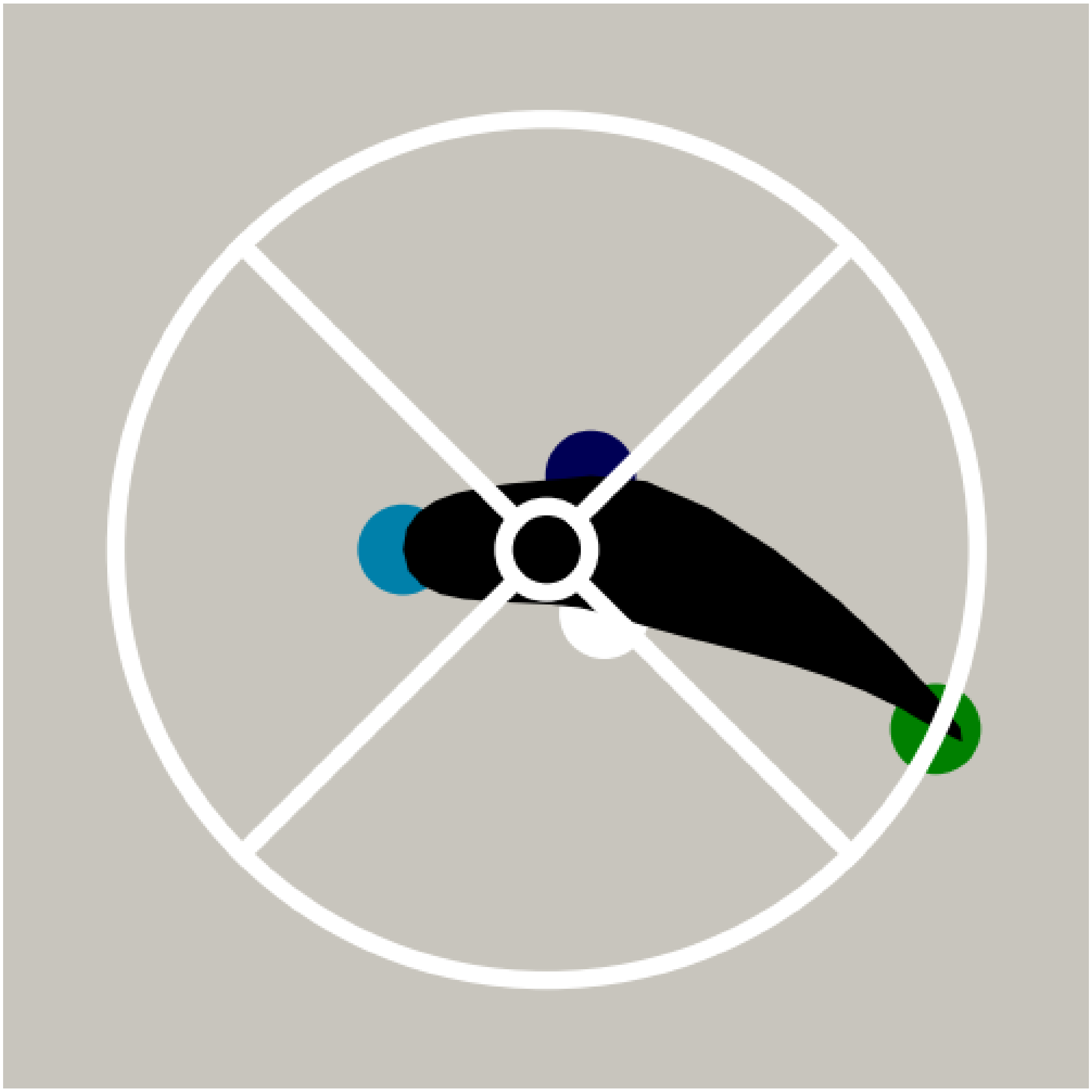

$T = T_c$
$+C$
0
$-C$

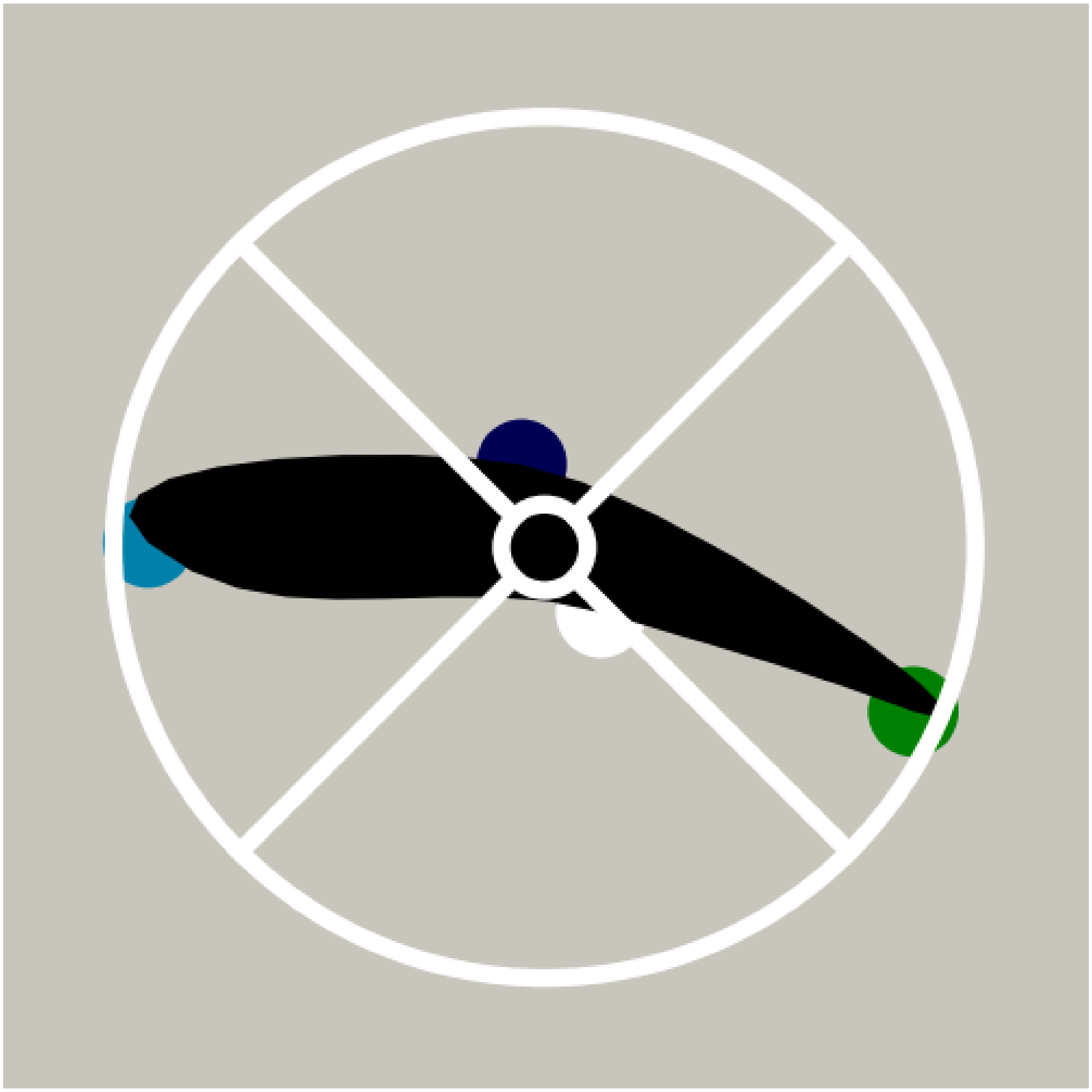


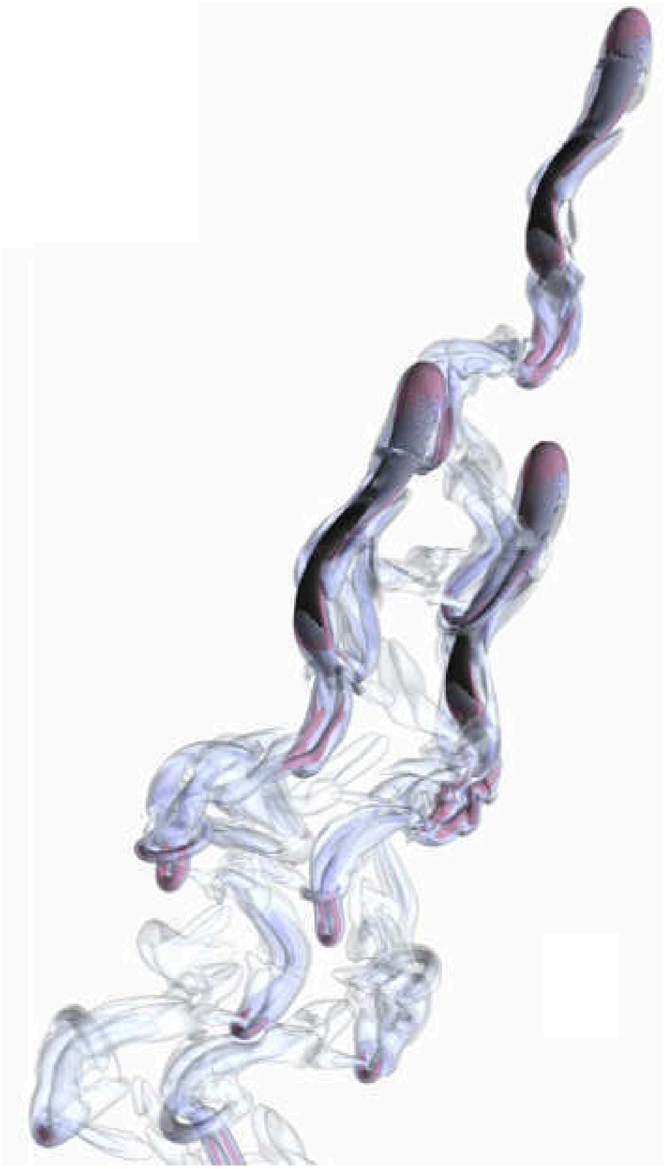

$\boldsymbol{s}_0$
Agent
$\boldsymbol{s}^*$

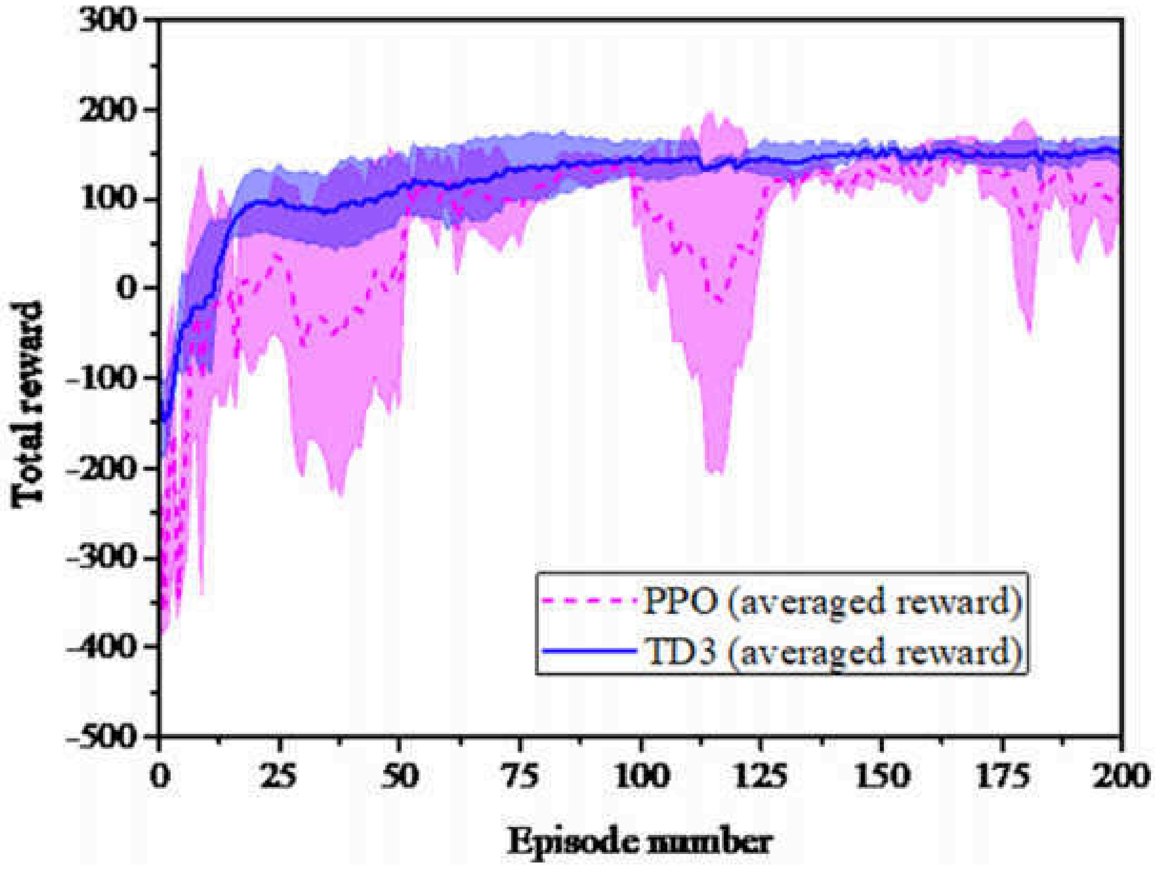

s_i
Agent
Δs_i

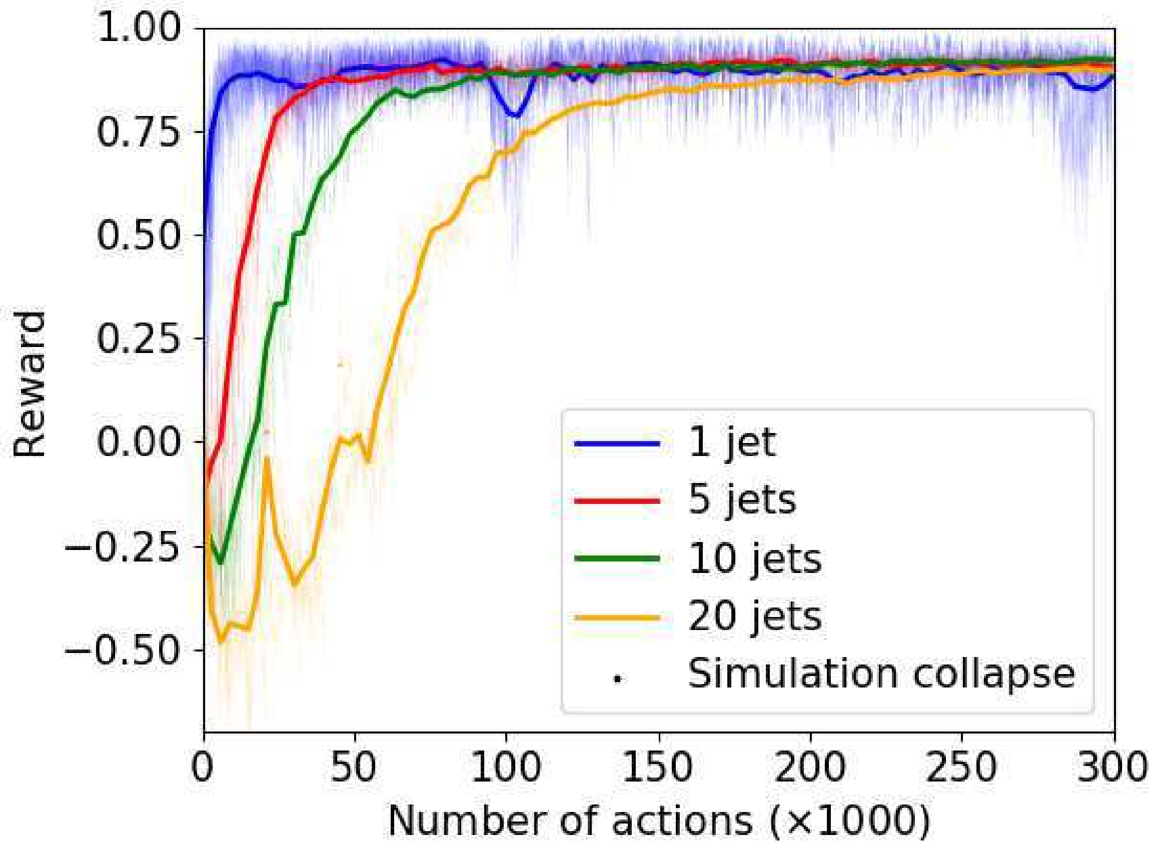


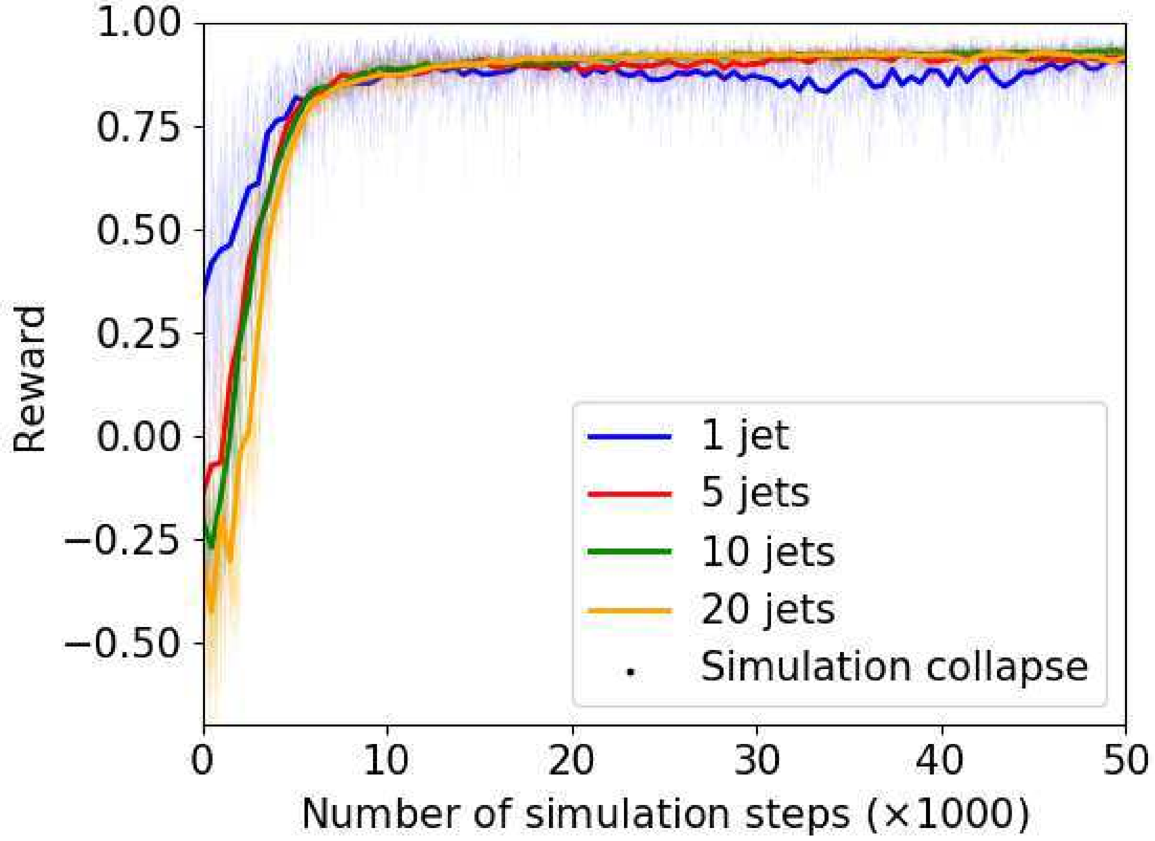







Total reward
300
200
100
0
-100
-200
-300
-400
-500
0
25
50
75
100
125
150
175
200
Episode number
PPO (averaged reward)
TD3 (averaged reward)


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Reward
0
50
100
150
200
250
300
Number of actions (×1000)
1 jet
5 jets
10 jets
20 jets
Simulation collapse


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
Reward
0
10
20
30
40
50
Number of simulation steps (×1000)
1 jet
5 jets
10 jets
20 jets
Simulation collapse