Physics of Fluids

REVIEW ARTICLE | MARCH 16 2023

Recent advances in applying deep reinforcement learning for flow control: Perspectives and future directions

C. Vignon; J. Rabault ; R. Vinuesa

Physics of Fluids 35, 031301 (2023)
https://doi.org/10.1063/5.0143913

Physics of Fluids REVIEW scitation.org/journal/phf

Recent advances in applying deep reinforcement learning for flow control: Perspectives and future directions

Cite as: Phys. Fluids **35**, 031301 (2023); doi: 10.1063/5.0143913
Submitted: 27 January 2023 · Accepted: 23 February 2023 ·
Published Online: 16 March 2023

C. Vignon,[1,2] J. Rabault,[3] and R. Vinuesa[1,a)]

AFFILIATIONS

[1]FLOW, Engineering Mechanics, KTH Royal Institute of Technology, SE-100 44 Stockholm, Sweden
[2]Mines de Paris, Université PSL, 75005 Paris, France
[3]IT Department, Norwegian Meteorological Institute, Postboks 43, 0313 Oslo, Norway

[a)]Author to whom correspondence should be addressed: rvinuesa@mech.kth.se

ABSTRACT

Deep reinforcement learning (DRL) has been applied to a variety of problems during the past decade and has provided effective control strategies in high-dimensional and non-linear situations that are challenging to traditional methods. Flourishing applications now spread out into the field of fluid dynamics and specifically active flow control (AFC). In the community of AFC, the encouraging results obtained in two-dimensional and chaotic conditions have raised the interest to study increasingly complex flows. In this review, we first provide a general overview of the reinforcement-learning and DRL frameworks, as well as their recent advances. We then focus on the application of DRL to AFC, highlighting the current limitations of the DRL algorithms in this field, and suggesting some of the potential upcoming milestones to reach, as well as open questions that are likely to attract the attention of the fluid mechanics community.

I. INTRODUCTION

Deep reinforcement learning (DRL) is a combination of deep learning (DL) and reinforcement learning (RL). Its ability to solve high-dimensional, non-linear complex problems has been proven over the past decade in multiple domains such as non-exhaustively, robotics,[1–3] language processing,[4] board games (e.g., the game of Go[5,6]) Poker,[7] video games,[8–10] and image analysis.[11] The acronym DRL is used when a deep neural network (DNN) takes place in the RL process. DNNs are well known for their ability to approximate non-linear functions on high-dimensional spaces[12] and have been introduced in RL methods for that particular purpose. Thanks to these artificial neural networks (ANNs), the algorithms of the 1990s solving low-dimensional problems[13–19] have been enhanced and now reach superhuman performances in games and can solve a broad range of non-linear, high-dimensional optimization tasks in various domains, as cited above.

Reinforcement-learning methods[20] rely on the interaction between an agent and an environment. The agent receives observations from the environment and then decides on an action according to those. Its aim is to maximize a cumulative reward that characterizes the "goodness" of the state of the environment. If the agent interacts with a model of environment instead of the environment itself, the RL algorithm is said "model-based," and otherwise, if the agent interacts directly with the environment, the RL algorithm is "model-free." In that case, the algorithm only uses the partial observations of the environment as inputs and returns the actions as an output. Such algorithms are thus considered to be "black-box" methods, because an analytical description of the environment is not needed to control it.

Problems including chaotic (turbulent) flows are usually non-linear and high-dimensional, due to the governing Navier–Stokes equations and the continuous environment space, i.e., the flow. Efficient black-box RL methods can therefore be useful and fruitful in that field, as traditional analysis methods based on local linearization usually struggle in this context. In Ref. 21, the authors review the main methods of closed-loop RL adapted to the specific case of fluid dynamics and their inherent challenges, i.e., high-dimensionality of the environment and/or action space(s), and complex non-linearity of the governing equations. The introduction of neural networks in RL algorithms has enabled one to approximate non-linear functions on high-dimensional spaces. The application of RL methods and derivatives in

fluid mechanics has thus known a flourishing growth for a few years. One may refer to recent advances with late reviews on DRL,[22,23] and more general machine-learning (ML) methods applied to fluid dynamics.[24] The use of RL in fluid mechanics has various purposes, such as (non-exhaustively) subgrid-scale,[25,26] wall,[27] and turbulence[28–30] modeling; RL-augmented computational fluid dynamics (CFD) solvers;[31,32] the shape optimization of airfoils;[33–35] the active control of the separation in a flow surrounding an aircraft;[36] the reduction of the drag past cars;[37] bluff bodies in both laminar[38] and chaotic regimes;[39–41] the study of the wake behind a cylinder;[42] or the reduction of the skin-friction drag in a turbulent channel.[43,44] Within the non-RL machine-learning methods, genetic programming[45] (GP) shows comparative results in multiple applications.[46–50] For the sake of brevity, as an entire review could be dedicated to those, GP algorithms will be briefly introduced in the present review solely to compare their main properties with the DRL ones.

In fluid mechanics, two fields are therefore mainly studied for the application of RL: shape optimization and flow control. Within the former, a body is immersed in a flow, and the RL algorithm modifies its shape in order to optimize some characteristic due to the dynamic of the fluid (e.g., the drag and/or the lift—see for instance Ref. 51). A classic example is the reduction of the drag coefficient behind a body in a laminar or turbulent flow: in Refs. 33 and 34, this idea is applied to the shape optimization of airfoils (more recent studies on that subject are given in Refs. 35 and 52, using DRL). Flow control aims at actively influencing the behavior of a flow, such as controlling separation, turbulence, or heat transfers, in order to reach a more desirable state. It is a subject of great interest from both societal and economical points of view.[35] Flow control strategies can be passive or active.[53] Conversely to the passive solutions (e.g., riblets and vortex generators[54]), the active methods need a supply in energy to perform the control sequence. As they could achieve unrivaled performances, DRL algorithms applied to active flow control (AFC) problems are largely studied for their promising engineering prospects (see, e.g., Refs. 21 and 23).

This review is aimed at presenting the current challenges in the field of active flow control (AFC), particularly when the control is based on DRL methods. To this end, a general overview of the main reinforcement-learning processes is first given in Sec. II and then follows a presentation of the recent advances in deep reinforcement learning (Sec. III). Eventually, applications of machine learning—and specifically DRL—in AFC are developed in Sec. IV. Attention will be focused on the challenges inherent to the field of fluid dynamics, and some possible upcoming milestones for the coupling of DRL and AFC will be suggested accordingly.

II. REINFORCEMENT LEARNING

A. Introduction

Reinforcement learning (RL) is a branch of data-driven methods, considered as a subsection of machine learning, which is itself a subset of artificial intelligence. The first study developing the modern form of reinforcement learning was provided by Sutton[55] at the end of the 1980s. In his time, Sutton[55] merged two distinct approaches: the learning by trials and errors, notably studied for its natural applications in the learning process of animals, and optimal control. By combining the two, the aim of modern RL is to optimize the decisions taken by an agent in a surrounding environment during a sequence of actions. In other words, the idea is to train and improve a control law (the agent) able to optimize a target system (the environment) by taking sequences of actions.

Reinforcement learning has already been applied to many fields with successful results. During the past decade, RL algorithms have been enhanced thanks to neural networks, developing DRL methods based on RL. One may refer to Refs. 1 for RL and 3 for DRL applied to robotics, to Refs. 8 and 9 for DRL applied to Atari games with superhuman performances, and to Refs. 10 for video games and 6 for Go game to name a few.

In reinforcement learning, an agent interacts with an environment in a three-step sequence:

(1) The environment gives to the agent a partial observation o of its state s. In a continuous environment space, the observation is necessarily incomplete. Thus, the observation o may not provide enough information to describe s. In order to overcome this issue, the state s_t obtained at time t is often described by the succession of actions and observations that led to it: $s_t = (s_0, a_1, o_1, a_2, \ldots, o_{t-1}, a_t)$, or similarly: $s_t = (s_0, a_1, s_1, a_2, \ldots, s_{t-1}, a_t)$.
(2) According to the observation o of state s, the agent takes an action a, following a certain policy π: $a = \pi(s)$. This action impacts the environment and modifies its state. As a consequence, its new state is now denoted s' (*a priori* different from s).
(3) Eventually, a reward r is given to the agent, characterizing the goodness of the action taken according to the old state s and the new one s'.

Through these three interactions, which take place at each time step of a global process, the RL algorithm gathers information and progressively enhances the reward. To this end, the reward defines the feature that the user wants to improve. Instead of the instantaneous reward, introduced as r above (or more precisely r_t at time t), the RL algorithm usually optimizes a cumulative reward, which is denoted by R and defined as

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

where γ is the discount factor (usually, $\gamma \gtrsim 0.99$), and T is the duration of an episode. The typical sequence of a reinforcement-learning algorithm may be described as follows:

- First, at $t = 0$, the state of the environment is (randomly) initialized (state s_0).
- At each time step t, the three-step process described above (see Fig. 1) is applied, returning a reward r_t after having changed the state of the environment from s_t to s_{t+1}, by enforcing the action a_t according to the policy π.
- When t reaches T, the process stops. The sequence of actions running during a time T is called an episode. At the end of an episode, the cumulative reward R is computed and the policy modified with the long-term objective of optimizing R. During an episode, $\lfloor \frac{T}{t} \rfloor$ actions are taken. In the cumulative reward, actions with a low instantaneous reward can be compensated by others, and no distinction is made apart from the discounted factor γ^t. This factor slightly favors short-term rather than

FIG. 1. Interactions between an agent and an environment in a generic reinforcement-learning loop.

long-term strategies, while enabling R to converge when episodes with numerous actions are considered. A full sequence $(s_0, a_1, s_1, a_2, \ldots, s_T)$ is called a trajectory and further represented by τ.

The cumulative reward is commonly not computed over the whole trajectory, but from a certain time step t_0. Equation (1) is then replaced by

$$R = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t. \tag{2}$$

Depending on the representation chosen for the environment, RL algorithms are ordinarily separated into two broad categories. If the agent interacts with an artificial modeling of environment, the algorithm is considered to be model-based; otherwise, the algorithm is model-free. In Sec. II B, examples of model-based methods are presented, but since the majority of DRL algorithms are model-free, attention will mainly be given to the model-free category in this review. Additionally, the reader can refer to the works of Jaderberg *et al.*[56] and Ha and Schmidhuber,[57] or to the recent review of Rabault and Kuhnle,[58] for examples of model-free algorithms incorporating a model-learning process. By trying to build a representation of the environment during the learning process and benefit from it, these initially model-free methods are at the cross section of model-free and model-based algorithms.

In Secs. II B–II D, we will only discuss the main concepts and features of RL algorithms, without a focus on DRL. How NNs can be used together with RL algorithms to develop DRL will be discussed in Sec. III.

B. Model-based methods

Model-based methods require a representation of the environment. Historically, linear models first spread across the field of flow control, as they were relatively simple and proved to be efficient on multiple problems, e.g., the stabilization of convectively unstable flows.[59–61] Yet, flows are also commonly characterized by their strongly nonlinear behaviors. Although linear models are justifiable in turbulent situations where the flow behaves like a linear amplifier,[62] their application to highly turbulent regimes is ambiguous and has long been debated in the community (see, e.g., Ref. 63 and the references therein). To that purpose, Brackston *et al.*[62] developed a (nonlinear) stochastic model, inspired by the work of Rigas *et al.*,[64] and governed by the nonlinear Langevin equation. A feedback controller coupled to this model successfully stabilizes the wake behind a bluff body immersed in a three-dimensional turbulent flow. The drag is efficiently reduced by suppressing the large-scale structures. In their review, Rowley and Dawson[65] analyzed the robustness of reduced-order models when applied to linear and nonlinear problems. These model-based methods aim at extracting the main dynamic processes that structure the flow, in order to simplify its modeling. The reader may also refer to the works of Queipo *et al.*[66] and Koziel *et al.*[67] for examples of model-based applications in aerodynamics.

The computational gain obtained with model-based methods results from the simplification of the environment. This gain is especially relevant in the situation of flow control, but necessarily leads to sub-optimal solutions when it comes to high-Reynolds-number regimes, as the models only partially approximate the turbulences. Hence, models can still be configured for these ranges of regime, but one is then confronted to a mediation between sub-optimal complex model-based controllers that rely on necessarily approximate models of the flow, and computationally heavy black-box model-free controllers that directly interact with a Navier–Stokes solver. Recent model-based methods, such as the probabilistic ensembles with trajectory sampling (PETS),[68] try to compete with the efficient modern model-free deep RL algorithms and thus go along the direction of developing both model-based and model-free methods.

C. Model-free methods

Model-free methods are increasingly popular in the community because of how relatively simple they are to apply. Indeed, the agent directly interacts with the environment: no assumption has to be made on the environment modeling. Thanks to regular partial observations of the environment state, the agent finds a control sequence $(a_0, \ldots, a_t, \ldots, a_T)$ that maximizes the cumulative reward.

The environment is usually considered as stochastic (i.e., when the action a is applied on the state s, the new state cannot be predicted and deduced exactly) and is commonly modeled by a Markov decision process (MDP),[20,69,70] even though it could be represented by other stochastic processes[38] or considered to be deterministic.[71]

1. Markov decision process

Markov decision processes (MDPs)[69,70] are at the core of modern reinforcement learning.[20] A MDP is a tuple $(S, A, \mathscr{P}_a, R)$ with S and A the sets of all the possible states and actions, respectively, and R the set of all the possible rewards. One can define $\mathscr{P}_a(s, s')$ as the probability that taking action a in state s will lead to state s'. Reusing the standard nomenclature proposed in Ref. 22, let us define the operator $\mathbb{P}$ as

$$\begin{aligned} &\mathbb{P} : S \times A \times S \to [0; 1], \\ &(s, a, s') \mapsto \mathbb{P}(s'|s, a) = \mathscr{P}_a(s, s'). \end{aligned} \tag{3}$$

Hence, $\mathbb{P}$ represents the probability of obtaining state s' after having taken action a in state s. The policy is defined by

$$\begin{aligned} &\pi_\theta : S \to A, \\ &s \mapsto \pi_\theta(s). \end{aligned} \tag{4}$$

Note that $\pi_\theta(s)$ returns the action probability distribution when the current state is s. The action is then chosen by the agent according to $\pi_\theta(s)$, and θ represents the set of parameters that characterizes π. The aim is to find the policy π that maximizes the cumulative reward R. Modifications of the policy are made by choosing different sets of parameters θ. Furthermore, $\mathbb{P}$ and π_θ completely describe the fundamental process of the RL algorithm: given a state s, the agent takes the

action a according to $\pi_\theta(s)$, and then, $\mathbb{P}$ returns the probability of obtaining a new state s'.

2. State value and state–action value functions

In addition, three functions are usually defined in order to describe and classify the RL algorithms. The state–action value function, also called the *Q-function*, is defined by

$$Q^\pi(s,a) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s, a_0 = a]. \tag{5}$$

Hence, $Q^\pi(s,a)$ corresponds to the expected cumulative reward when the action a is taken in the initial state s and then follows the trajectory τ according to the policy π. The state value function is defined by

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[R(\tau)|s_0 = s]. \tag{6}$$

$V^\pi(s)$ corresponds to the expected cumulative reward when the initial state is s and then follows the trajectory τ according to the policy π. Thus,

$$V^\pi(s) = \mathbb{E}_{\tau\sim\pi}[Q^\pi(s,\pi(s))]. \tag{7}$$

Eventually, the advantage function A^π is defined by

$$A^\pi = Q^\pi - V^\pi. \tag{8}$$

A^π characterizes the advantage (in terms of cumulative reward) of taking action a in state s rather than taking any other possible actions.

Among the model-free RL algorithms, two main methods are both developed for their promising performances: value-based[72] and policy-based methods.[73] During the RL process, the agent takes actions according to the state of the environment by following a policy. With policy-based methods, the aim is to directly optimize the policy π (see Sec. II C 6). Differently, value-based methods rely on learning the Q-function and then deriving the optimal policy from it (see Sec. II C 5).

3. Closed-loop and open-loop methods

Additionally to the value/policy criterion, RL methods are also commonly distinguished according to the frequency of the interactions between the agent and the environment. In closed-loop control, the agent obtains a regular feedback of the current state of the environment thanks to sensors. The RL algorithm uses these measurements to adapt its policy during the episode, in order to tend to a more desired state (see Ref. 21 for a review on closed-loop turbulence control). On the other hand, open-loop methods assume that the decision policy does not rely on the state of the environment, or at least that a steady or periodic actuation can modify the principal dynamical processes of the environment.[74] One may refer to the recent works of Shahrabi[75] and Ghraieb *et al.*[35] for the use of open-loop methods in flow control. By not considering the surrounding environment, these methods are generally less efficient than the closed-loop solutions, but easier to implement. The upcoming sections will mainly focus on closed-loop algorithms, as they are the most promising and challenging methods for the future.

4. On-policy and off-policy online and offline methods

Another distinction among RL algorithms is made between on-policy and off-policy methods. Within the former, the agent solely learns about the policy it enforces to the environment, contrarily to the latter where the agent can learn from other policies. For instance, Degris, White, and Sutton[76] execute a second (behavior) policy aimed at choosing the trajectories, while the first one decides the actions. Another example of off-policy method is the Q-learning process[77] mentioned in Sec. II C 5. Furthermore, with on-policy methods, the data collected before the last update of the (single) policy cannot be used for forthcoming updates, as these data had been generated by a policy that differs from the last version. Therefore, off-policy settings are of great interest, as they enable the algorithm to reuse previous experiences that may be stored from either earlier in the learning process or completely separate learning runs, and to learn multiple tasks in parallel thanks to the different policies.

Offline RL algorithms train and learn a policy from a fixed batch of data without exploration. The data are obtained from the distinct policy(-ies). Online strategies present the advantage to learn from the current state of the environment and maintain exploration instead of relying on past fixed situations, but require demanding computational properties,[78] not always available nor achievable. The reader may refer to the work of Degris, Pilarski, and Sutton[79] and the references therein for additional information on the differences between offline and online methods.

5. Value-based methods

Value-based methods rely on the Q-function. The optimal Q-function can be defined as $Q^* = \max_\pi Q^\pi$. In reinforcement learning, the objective of value-based methods is to have a Q-table filled with the values of $Q^*(s,a)$ for any (s,a) in $S\times A$. In other words, the aim is to be able to know the best policy to apply when starting from any tuple (s,a) in $S\times A$, in order to maximize the expected cumulative reward. The Q-table can be thought of as an array or a transition reward matrix in discrete state case or can be approximated by a NN or another function approximator in general. For simplicity, in this section, Q is considered as an array that can be filled with values, while an ANN trained by gradient descent will be considered in deep Q-learning (DQL) (see Sec. III B 1).

It is not possible to directly compute Q^*, contrarily to Q^π. Thus, the Q-table is first filled with values of Q^π, estimated by exploring the space $S\times A$. Additionally, Q^* respects the Bellman optimally condition[80]

$$\begin{aligned} Q^*(s,a) &= r(s,a) + \sum_{s'\in S} \mathbb{P}(s'|s,a)\gamma \max_{a'} Q^*(s',a') \\ &= r(s,a) + \mathbb{E}_{s'\in S}\left[\gamma \max_{a'} Q^*(s',a')\right]. \end{aligned} \tag{9}$$

Here, $r(s,a)$ represents the instantaneous reward returned when action a is taken in state s. This relationship on Q^* means that the action a is first applied to the initial state s, leading to the state s' (as it is a stochastic process, s' is not known precisely, hence the expectation on s'), and if the optimal Q-function is known at the next time step, then the best policy is to choose the next action a' that maximizes $Q^*(s',a')$.

To update the Q-table, Eq. (9) inspires the recursive update relationship given below

$$Q^{\pi}(s,a) \leftarrow \mathbb{E}_{s' \in S}\left[r(s,a) + \gamma \max_{a' \in A} Q^{\pi}(s',a')\right]. \quad (10)$$

It has been proved that updates of Q^{π} by following (10) make Q^{π} converge to Q^{*}.[80]

Eventually, when the Q-table is filled with the estimated values of Q^{*} [thanks to the recursive process given by Eq. (10)], and given a state s, one can then optimize the action that the agent will take, by choosing the action a^{*} according to the "greedy" policy

$$a^{*} = \mathrm{argmax}_{a \in A} Q^{*}(a,s). \quad (11)$$

These value-based methods are usually called Q-learning.[77] One can refer to the original work of Watkins,[72] which was already an evolution of difference learning.[55]

6. Policy-based methods

The second main category of RL methods directly relies on an estimation and an optimization of the policy function π_{θ} and does not consider an optimization of the Q-function. These policy-based methods have been developed since—at least—the work of Williams.[73] The objective function J is introduced in order to evaluate the quality of the policy. As the long-term aim still remains to maximize the expected cumulative reward, J is commonly defined by

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)]. \quad (12)$$

J can vary thanks to modifications of θ, the set of parameters characterizing the policy. Thus, the objective is to find

$$\theta^{*} = \mathrm{argmax}_{\theta} J(\theta). \quad (13)$$

To that purpose, the gradient of J is introduced, so that θ can be recursively modified by following the direction of this gradient:

$$\theta \rightarrow \theta + \lambda \nabla_{\theta} J(\theta) \quad (14)$$

with λ being a positive parameter. Two distinct options are discussed for the evaluation of $\nabla_{\theta} J$, whether the policy is considered to be stochastic or deterministic, i.e., whether $\pi(s)$ returns a probability distribution of actions a or an exact and unique value a. In the former case, the gradient is computed according to the relationship[22,73,81]

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta}}\left[\sum_{t=t_0}^{T} \nabla_{\theta} \log(\pi_{\theta}(a_t|s_t)) R(\tau)\right]. \quad (15)$$

In this stochastic approach, the ascent of J consists in first sampling the policy on different trajectories, then computing an average of the cumulative rewards obtained on these trajectories [the average thus replaces the expectation in Eq. (15)], and finally adjusting θ with Eq. (14), in order to progressively enhance the cumulative reward.

In the deterministic approach (deterministic policy-gradient algorithm or DPG algorithm[71]), the environment is still represented with a stochastic process (MDP), but the policy is considered to be deterministic. To keep in mind the difference, the deterministic policy is now represented by μ_{θ}. Replacing π_{θ} with μ_{θ}, the objective of the deterministic policy gradient algorithms is to apply the same principle as in the stochastic case. The mathematical formulation of the deterministic policy gradient is beyond the purpose of the present work. One may refer to the original work of Silver *et al.*[71] and bear in mind that with a deterministic approach, it is not needed to sample over a range of actions and states, but only over a range of states, as the actions are deterministically deduced.

D. Limitations of the classical reinforcement-learning methods

Classical RL methods, i.e., algorithms that use simple function approximators such as matrices, polynomials, or other simple functions, may encounter limitations. Historically, applications of value-based methods to real-world problems[82,83] have rapidly been overtaken by faster policy-based methods (see, e.g., the policy gradient method[84]), which quickly proved their efficiency and applicability to various real-world tasks.[14–16]

Through the idea of experience replay (see the original work of Lin[85]), value-based RL methods have known a revival (e.g., Ref. 86 for an application). The experience replay technique largely shrinks the number of interactions between the agent and the environment and thus accelerates these methods. Nevertheless, both the value-based and policy-based RL methods remain limited to problems with low state and action spaces.[9,76]

III. DEEP REINFORCEMENT LEARNING

Deep reinforcement learning (DRL) methods consist in the combination of neural networks and classical RL algorithms. Neural networks are known to be universal function approximators,[12] able to treat high-dimensional and non-linear problems, and thus to overcome the limitations set by the simple function approximators used in traditional RL. This means that, given a sufficient amount of example data representing a well-behaved (typically, continuous) function, and a large enough ANN, the ANN can represent the function with arbitrary precision; i.e., ANNs large enough are dense in the space of continuous functions. Their application in reinforcement learning led to breakthroughs in various fields as they largely enhanced the performances of the previous methods in multi-input multi-output (MIMO) problems. To cite a few, the DRL algorithm developed by Mnih *et al.*[9] obtained superhuman levels in Atari games, and the one created by Silver *et al.*[6] won against the European champion of Go five times in a row (to contextualize this achievement, no algorithm was able to beat a professional human player in this particularly complex board game before). Eventually, the proximal policy optimization (PPO) algorithm implemented by Rabault *et al.*[38] successfully led to the first control of a flow by DRL.

In this section, a short introduction to artificial neural networks is first provided. An overview of the main DRL methods—presented in light of their RL precursors—is then suggested. Within these methods, the most recent and encouraging developments are presented, as well as the challenges they face and try to overcome.

A. Artificial neural networks

Artificial neural networks (ANNs or NNs) are function approximators, based on connections between singular units, called neurons. A neuron is an entity characterized by three features: a set of weights $w = (w_1, \ldots, w_n)$, with $n \in \mathbb{N}^{*}$ corresponding to the number of inputs connected to the neuron, a bias b ($b \in \mathbb{R}$), and the activation

function σ, an hyper-parameter chosen at the conception of the neural network. Taking a vector $x = (x_1, \ldots, x_n)$ as an input, the neuron returns the scalar $\sigma(w.x + b)$. Neurons are usually organized in successive layers; we then talk about deep NNs (DNNs). Each neuron of a layer is connected to one or more neurons of the precedent and next layers. When each neuron of each layer is connected to all the neurons of the precedent and of the next layer, the NN is said fully connected. FCANNs (fully connected artificial neural networks) are commonly used in DRL. A schematic of FCANN is given in Fig. 2. This illustrative example is constituted of a layer of two output neurons (red disks) and two layers of five neurons (green disks). The first layer on the left of the figure is not constituted of neurons: the purple disks correspond to the inputs (here of size three). They are conventionally represented similarly to neurons but simply represent a scalar input (or, equivalently, a neuron with $n = 1$, $b = 0$, $w_1 = 1$, and $\sigma = \mathrm{id}_{\mathbb{R}}$). The black lines correspond to the connections between the neurons.

The training of an ANN consists in tuning all its degrees of freedom, i.e., tuning the biases and weights of all the neurons so that the ANN accurately represents a goal function known through examples. To do so, these parameters are commonly tuned step by step, by following, e.g., the back-propagating algorithm.[87]

ANNs can be used in RL algorithms for their ability to estimate complex non-linear functions.[12] The number of NNs and the functions they approximate within the RL algorithm define several different DRL methods, which we analyze and compare in the following paragraphs.

B. Value-based methods

1. Deep Q-learning

Artificial neural networks are commonly known for their ability to deal with high-dimensionality and non-linearity, following their universal approximator properties, as previously discussed. Therefore, instead of filling a Q-table with the values of Q^π—as it was done in value-based RL, ANNs may be appropriate to directly approximate Q^*.

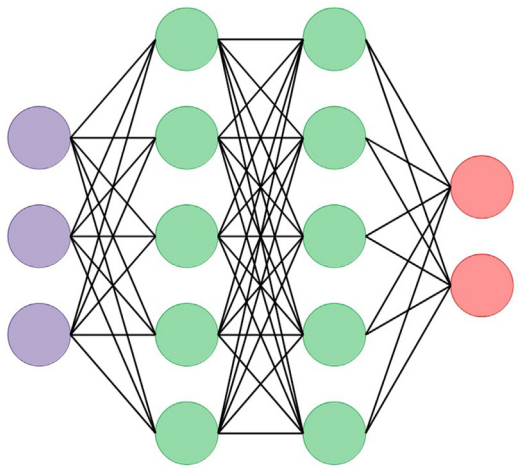


FIG. 2. Schematic of a fully connected artificial neural network. Purple denotes input, red output, and green the hidden layers.

The approximation of Q^* by the neural network is represented by Q_θ, with all the weights and biases of the neural network represented by the set θ. The objective of the training phase of the neural network is to tune θ in order to achieve the best approximation of Q^*. To do so, a loss function is defined as the mean-squared error of the Bellman equation [Eq. (9)]

$$\begin{aligned} L(\theta) &= \mathbb{E}_{s,a}\left[\frac{1}{2}\{\mathbb{E}_{s'}[r(s,a) + \gamma \max_{a'} Q_\theta(s',a')] - Q_\theta(s,a)\}^2\right] \\ &= \mathbb{E}_{s,a,s'}\left[\frac{1}{2}\{r(s,a) + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\}^2\right] \\ &\quad + \mathbb{E}_{s,a,s'}\left[\frac{1}{2}\mathbb{V}_{s'}[r(s,a) + \gamma \max_{a'} Q_\theta(s',a')]\right]. \end{aligned} \tag{16}$$

The variance of $r(s,a) + \gamma \max_{a'} Q_\theta(s',a')$ is independent of the weights θ, the second term is usually ignored,[9] and the loss function is limited to

$$L(\theta) = \mathbb{E}_{s,a,s'}\left[\frac{1}{2}\{r(s,a) + \gamma \max_{a'} Q_\theta(s',a') - Q_\theta(s,a)\}^2\right]. \tag{17}$$

More precisely, at each time step I,

$$L_i(\theta_i) = \mathbb{E}_{s,a,s'}\left[\frac{1}{2}\{r(s,a) + \gamma \max_{a'} Q_{\theta_{i-1}}(s',a') - Q_{\theta_i}(s,a)\}^2\right]. \tag{18}$$

The objective of the neural network is to minimize the loss function, thus to modify Q_{θ_i} to get it closer to the target $r(s,a) + \gamma \max_{a'} Q_{\theta_{i-1}}(s',a')$, computed with the weights θ_{i-1} of the precedent update. To this end, the update

$$\theta_{i+1} \leftarrow \theta_i - \alpha \nabla_{\theta_i} L_i(\theta_i) \tag{19}$$

is applied (with α a positive parameter) and should make Q_θ converge to Q^*.

However, a naive coupling—such as the one mentioned above—between an ANN and a RL algorithm would be compromised by two issues that may lead to divergence. First, the process of learning from sequences of actions and observations on-policy induces correlations. Indeed, the algorithm is potentially biased if it solely learns from the successions of tuples (s_t, a_t, s_{t+1}, r_t) that occur during the running episode (on-policy), because these tuples are generally not independent within an episode, which has a tendency to make the NN training fail. Furthermore, a minor modification of the value of Q may change significantly the data distribution and thus may lead to an unstable algorithm.

In order to tune these two biases, Mnih *et al.*[8] suggested the integration of a replay buffer in their algorithm. Filled with a certain amount of tuples (s_t, a_t, s_{t+1}, r_t) that occurred in past episodes, the replay buffer enables the ANN to have more uncorrelated input data and, simultaneously, a more stabilized data distribution, which is required for a correct learning process. By adapting the experience replay technique suggested in Ref. 85, Mnih *et al.*[8] thereby proposed the deep Q-learning (DQL) algorithm, which masters complex control policies in Atari games. Schaul *et al.*[88] extended the idea of replay buffer by suggesting to optimize its filling, by favoring less frequent actions instead of a uniform process, as the learning phase could benefit from these unusual events.

Even with the DQL algorithm, a major restriction remains, due to the fact that the neural network is chasing a moving target. As developed above, the ANN tries to reach the Bellman optimally condition [Eq. (9)] by minimizing the loss function L. To do so, it modifies Q_{θ_i} to get it closer to the target $r(s,a)+\gamma \max_{a'} Q_{\theta_{i-1}}(s',a')$, but the target itself changes at each time step i. This issue induces instability and divergence in many situations, notably when the estimation of Q is provided by a nonlinear function approximator,[89] such as a FCANN. This instability thus highly reduces the algorithm robustness.

2. Deep Q-networks

Developed by Mnih *et al.*,[9] deep Q-networks (DQNs) are an evolution of DQL. The DQN method mainly improves the ability of the neural network to treat high-dimensional environment spaces and suppresses the issue of divergence detailed above, by providing an innovation to the DQL algorithm.

The problem of divergence is solved by exploiting a second ANN, a "slow-reacting clone" of the original one. Let us define the loss function with Eq. (17). In DQL, the algorithm tries to equalize the target $r(s,a)+\gamma \max_{a'} Q_\theta(s',a')$ with $Q_\theta(s,a)$, by using a single neural network. In DQN, an ANN is applied for the computation of Q_θ and the "slow-reacting" copy is applied for the estimation of the target. Denoting θ' as the set of weights and biases of this second network, the loss function is then defined by

$$L(\theta)=\mathbb{E}_{s,a,s'}\left[\frac{1}{2}\{r(s,a)+\gamma \max_{a'} Q_{\theta'}(s',a')-Q_\theta(s,a)\}^2\right]. \quad (20)$$

The strategy suggested by Mnih *et al.*[9] is to regularly equalize θ and θ'. If such an equalization is made at each update of θ, then the method becomes identical to DQL. Different strategies have been developed to correlate θ and θ'. In Ref. 90, the authors develop a different method (called DDPG and detailed below), but use the same idea, and propose the iterative process $\theta' \leftarrow \tau\theta+(1-\tau)\theta$, with $\tau \ll 1$. In the original DQN algorithm,[9] θ' is actualized one time over N updates of θ ($N \in \mathbb{N}^* \gg 1$). These strategies with two ANNs thus eliminate the instability by slowing down the movements of the target.

Dueling DQN[91] is an evolution of DQN in which the neural network is split into two streams, one aimed at evaluating the advantage function, while the other one computes the state value function V^π. The Q-function is then estimated by combining the approximations of A^π and V^π (in a non-trivial manner). The advantage function depends on the actions chosen, whereas the state value function does not. By taking advantage of these two functions and their different dependencies on the actions, the dueling DQN algorithm overcomes the DQN method particularly in situations, where there are many similar-valued actions.

3. Double deep Q-learning and double deep Q-networks

Double DQL and double DQN[92,93] have been developed in order to solve the issue of overestimation happening in DQL and DQN, a problem partially unknown or ignored up to then. The overestimation in these algorithms is due to the fact that in the computation of the target, the same neural network is used both for the selection of the action and for its evaluation. Indeed, in DQL, one can define the target Y as

$$\begin{aligned} Y(s,a) &= r(s,a)+\gamma \max_{a'} Q_\theta(s',a') \\ &= r(s,a)+\gamma Q_\theta(s', \operatorname{argmax}_{a'} Q_\theta(s',a')) \end{aligned} \quad (21)$$

and likewise in DQN

$$\begin{aligned} Y(s,a) &= r(s,a)+\gamma \max_{a'} Q_{\theta'}(s',a') \\ &= r(s,a)+\gamma Q_{\theta'}(s', \operatorname{argmax}_{a'} Q_{\theta'}(s',a')). \end{aligned} \quad (22)$$

In both cases, the neural network that estimates Q^* is used to choose a' and then to apply $Q^*(s',a')$, which can lead to an overestimation of the expected cumulative reward. Not only is overestimation quite common, but it can also adversely affect the performances of the algorithm.[93] For the DQL algorithm, a new method with a second network is presented in Ref. 93, inspired by previous works.[92] In double DQL, two networks are exploited so that one focuses on the choice of the action, and the other one on the evaluation. Hence, the target becomes

$$Y(s,a)=r(s,a)+\gamma Q_{\theta^{bis}}(s', \operatorname{argmax}_{a'} Q_\theta(s',a')) \quad (23)$$

with θ^{bis} the set of weights and biases of the new neural network, used for the evaluation. The two networks regularly invert their roles, so that θ and θ^{bis} are similarly updated. The double DQN algorithm does not require any other network, a second one being already exploited within the DQN method. Indeed, van Hasselt, Guez, and Silver[93] suggested

$$Y(s,a)=r(s,a)+\gamma Q_{\theta'}(s', \operatorname{argmax}_{a'} Q_\theta(s',a')). \quad (24)$$

According to van Hasselt, Guez, and Silver,[93] the performances of the value-based DRL methods are enhanced, even the already robust performances of the DQN algorithm in Atari games.[9]

4. Challenges and openings

The DQN algorithm has first been developed to stabilize the DQL method and notably enhanced its efficiency to deal with large neural networks. Thanks to that, deep Q-networks have shown great results with high-dimensional observation spaces.[9] Yet, DQN or even double DQN algorithms are not easily able to deal with high-dimensional action spaces. Indeed, as they seek $\operatorname{argmax}_a Q_\theta(.,a)$ at each time step, increasing the dimensionality of the actions space highly reduces their performances. Some methods have been developed to overcome this issue, making continuous action spaces possible. Based on a combination of policy gradient and Q-function, they also offer larger possibilities, as they may consider either off-policy or on-policy strategies, whereas the methods mentioned above are limited to off-policy strategies,[94] due to the replay buffer. These promising policy-based methods are presented in Sec. III C.

C. Policy-gradient and actor–critic methods

1. Deep policy gradient

Amidst the current most promising DRL methods, the second main contender is the family of actor–critic methods. These are obtained by combining elements of Q-learning and policy-gradient methods, which we will discuss now.

The deep policy gradient algorithm is a policy-based method; i.e., it directly optimizes the policy (see Sec. II C 6). Explicitly, an ANN

approximates the policy π_θ, with θ representing its set of weights and biases. The loss function is defined by

$$L(\theta) = -\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=t_0}^{T} \log\left(\pi_\theta(a_t|s_t)\right)R(\tau)\right] \tag{25}$$

so that $\nabla_\theta L(\theta) = -\nabla_\theta J(\theta)$ [see Eq. (15) for $\nabla_\theta J$]. This expected value is approximated by an average over a set of trajectories. $R(\tau)$ is commonly replaced with the advantage function A^{π_θ}, as it reduces the variance of the expectation[95] and thus reduces the number of trajectories necessary to well calculate this expectation. Hence, in the deep policy gradient algorithm, the loss function

$$L(\theta) = -\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=t_0}^{T} \log\left(\pi_\theta(a_t|s_t)\right)A^{\pi_\theta}(s_t,a_t)\right] \tag{26}$$

is usually considered. Nonetheless, as the expectation is averaged over a set of trajectories at the end of an episode, the algorithm does not differentiate between the most accurate actions and the worse, but only computes the reward obtained from the whole trajectory. Actor–critic methods adapt the deep policy gradient algorithm to deal with this issue.

2. Advantage actor-critic methods

Deep policy gradient methods are limited as they do not differentiate positive from negative actions. To clear it up, actor–critic methods rely on a combination of policy-based and value-based methods, as one neural network (the actor) approximates π_θ, the policy by which the actions are chosen, and a second one (the critic) computes the Q-function $Q_{\theta'}$, in order to evaluate the goodness of the action taken by the agent. With the same prospect of variance reduction, $Q_{\theta'}$ is often replaced with $A_{\theta'}$, an estimation of the advantage function. The method is then called advantage actor–critic (A2C).[20]

One may refer to the review of Garnier *et al.*[22] for explicit implementations of the vanilla deep policy gradient and A2C algorithms. Many variations have been developed, such as the asynchronous A2C (A3C) suggested in Ref. 94. The latter was presented to have similar—if not better—performances compared to the DQN algorithm in Atari games, and additionally to be able to treat continuous action spaces, which was a reason for the development of policy-based methods. Indeed, as an $argmax_{a\in A}()$ function is not computed anymore, these methods enable one to tremendously increase the action space A, up to a continuous space in some algorithms.

3. The deep deterministic policy gradient and twin-delayed deep deterministic policy gradient methods

As explained above, a major limit of DQN and other value-based methods is their lack of robustness with high-dimensional action spaces. The deep deterministic policy gradient (DDPG) algorithm was first developed by Lillicrap *et al.*[90] to solve this specific issue. It is a model-free off-policy actor–critic method, inspired by the Deterministic Policy Gradient (DPG) algorithm presented in Ref. 71 (see Sec. II C 6). Conversely to the previously presented DRL methods, the DDPG (like the DPG) algorithm considers a deterministic policy mapping and not stochastic. Thus, the action space is explored thanks to a perturbative method

$$a_t = \mu_{\theta+N_1}(s_t) + N_2 \tag{27}$$

with N_1 and N_2 two different noisy processes, θ the set of weights and biases of the actor, and a_t the action taken at time step t in state s_t according to the deterministic policy μ. The critic computes the optimal Q-function similarly to the DQN algorithm, but replaces the $argmax_{a\in A}()$-seeking with a gradient over μ (see Ref. 90), and thus enables one to leverage high-dimensional and continuous action spaces.

Bucci *et al.*[96] succeeded in stabilizing the dynamics of a chaotic system governed by the one-dimensional Kuramoto–Sivashinsky equation thanks to a DDPG controller, while Sonoda *et al.*[43] applied this algorithm to a flow control problem (the reduction of the skin friction drag in a channel flow). Both of these examples illustrate the promising performances of the DDPG method with high-dimensional action spaces.

The twin-delayed deep deterministic policy gradient algorithm[97] (TD3) is an evolution of the DDPG algorithm. Similarly to the double DQN, a second ANN is used in order to avoid the overestimation of the Q-function, and delay between the updates of the two neural networks is introduced.[98] Additionally, the target policy is smoothed,[99] the noise processes being partially connected to the overestimation bias.[97] With these three modifications, TD3 is more robust than DDPG in terms of hyper-parameters tuning[98] and presents comparable performances to the PPO algorithm described below, if not better (see, e.g., Refs. 99 and 100 for an application in active flow control).

4. Proximal policy optimization

The proximal policy optimization (PPO) algorithm is an on-policy actor–critic method, developed by Schulman *et al.*[101] in order to deal with the lack of robustness of the DQN[9] and "vanilla" policy gradient[94] methods, and simplify the efficient-but-complex trust region policy optimization (TRPO) algorithm,[95] not discussed in the present review.

Apart from the particular DDPG algorithm, the policy gradient and actor–critic methods usually aim at optimizing the loss function defined in Eq. (26), now denoted by $L^{PG}(\theta)$. To do so, these algorithms proceed over several epochs of gradient ascent; i.e., they alternate a first phase where they fill a batch of samples (a sample being a sequence (s_t, a_t, s_{t+1}, r_t) for instance), and a second phase in which the expectation of the gradient of $L^{PG}(\theta)$ is averaged over the batch, and then, the neural network parameters θ updated following the direction of this estimation. The idea behind the TRPO and PPO algorithms is identical, but L^{PG} is replaced with other functions. For instance, the surrogate loss

$$L^{\text{Clip}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=t_0}^{T}\min\{r_t(\theta)A^{\pi_\theta}(s_t,a_t), \right.$$
$$\left. \text{clip}(r_t(\tau), 1-\varepsilon, 1+\varepsilon)A^{\pi_\theta}(s_t,a_t)\}\right] \tag{28}$$

is suggested in Ref. 101 for the PPO algorithm, with

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta^{\text{old}}}(a_t|s_t)} \tag{29}$$

being a factor that penalizes an update of θ that would induce an important modification of the policy (comparatively to the situation

before the update, with the previous set of NN parameters θ^{old}). ε is a parameter chosen between 0 and 1, and the operator clip() is defined by

$$\text{clip}(r_t(\tau),\, 1-\varepsilon, 1+\varepsilon) = \begin{cases} \mathrm{r}_t(\tau), & \text{if } r_t(\tau) \in [1-\varepsilon; 1+\varepsilon], \\ 1+\varepsilon, & \text{if } r_t(\tau) > 1+\varepsilon, \\ 1-\varepsilon, & \text{otherwise.} \end{cases} \tag{30}$$

In the TRPO algorithm, the objective function is similarly penalized according to the size of the update.[95] The PPO algorithm is a simplification of this previous work as it only considers a first-order optimization process. Another variant of loss function is proposed in Ref. 101, using a KL-penalty.[102]

After having tuned $\varepsilon = 0.2$, Schulman *et al.*[101] observed that the PPO algorithm outperforms most of the previous online policy gradient algorithms [A2C and vanilla PG algorithms,[94] a combination of A2C and trust region algorithm,[103] the TRPO method,[95] and the cross-entropy method (CEM)[104]] on almost all the continuous control environments.

Hence, the PPO algorithm is one of the most promising modern DRL methods and has already successfully been applied to a variety of problems (see, e.g., Ref. 105). It is often considered as the "state-of-the-art" for continuous control.[106] To cite an application in flow control, Rabault *et al.*[38] used a PPO algorithm for performing the first active flow control combining CFD simulations and deep reinforcement learning, in the standard example of the cylinder immersed into a 2D flow, and Ren, Rabault, and Tang[40] extended the previous work to a weakly turbulent regime.

Variations based on the original PPO algorithm now flourish, such as PPO-1.[35] PPO-1 is a degenerate[52] version of the PPO method, developed for open-loop control (the reader may also refer to Ref. 107 for an additional work on open-loop control in numerical fluid dynamics). The algorithm runs multiple simulations in parallel, shuffles the data coming from all the environments, and fills several mini-batches with these mixed data, and then, it uses these mini-batches sequentially to update the NN and eventually repeats the sequence by launching multiple parallel simulations again. Another variant is the PPO with covariance matrix adaptation (PPO-CMA),[108] an algorithm able to dynamically expand the variance of the exploration policy to speed up the seeking phase, and then to shrink it when the algorithm gets close to the global optimum. PPO is stable but sometimes prematurely reduces the exploration variance, and then may converge to local optima, hence the development of this algorithm. The reader may also refer to the maximum *a posteriori* policy optimization (MPO) algorithm[109] and extensions[110] for similar motivations.

A recent variation of PPO-CMA, called AS-PPO-CMA,[111] aims at reducing the number of actuators while limiting the loss induced in the cumulative reward. This development goes along with the prospect of increasing the complexity of the problems handled by DRL, where an effective reduction of the actions space may be critical.

For comparison, a non-comprehensive overview of RL methods is suggested in Fig. 3, while schematics representing the main different DRL architectures are gathered in Fig. 4.

D. Parallelization

Multiple processes have been developed to reduce the computational cost of DRL algorithms, in order to then consider more complex tasks. The ones gathered by the idea of parallelization are presented in this section.

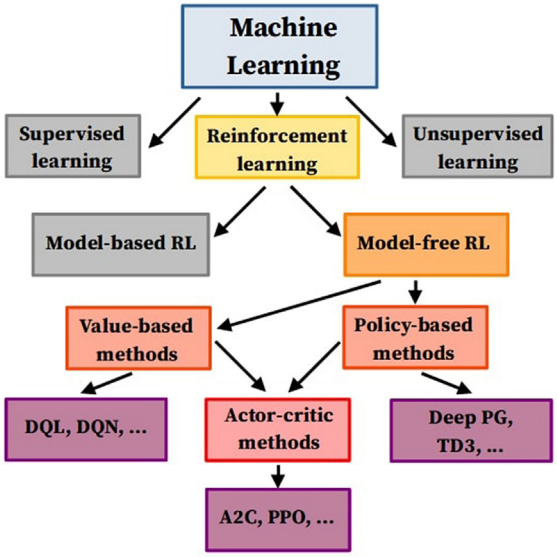


FIG. 3. Non-exhaustive classification of machine learning methods, with focus on reinforcement-learning-based techniques.

First, when it comes to fluid dynamics problems that combine CFD simulations with DRL algorithms, one may consider the parallelization of the simulations themselves. This subject has a long history,[112,113] and well-implemented parallel CFD solvers nowadays reach limitations correlated with the number of nodes,[114] hence limiting the gain of time with parallel CFD simulations.

Consequently, as the computational cost due to the simulations may remain important even with parallel CFD solvers, the parallelization of the DRL algorithms is of great interest. Speedups would enable one to study more challenging flow configurations, which are the main objective of the current research in the field. Rabault and Kuhnle[115] consider the simple bi-dimensional benchmark presented in Ref. 38 and adapt the PPO algorithm to parallelization. Their idea is to run independent episodes on several parallel environments so that the collection of data used for the training phase is parallelized. Indeed, they prove that if one runs N episodes sequentially between two updates of the neural network, running these N episodes in parallel is equivalent, and thus divides the computational time by approximately N. Results are conclusive, the computational time being divided by twenty in their most optimal implementation. This is of great interest, as problems with higher complexity usually need to increase the size of the batch and the number of episodes between updates of the NN. Increasing the complexity would therefore no longer induce an increase in the time cost, if additional parallel environments were added accordingly. The idea of PPO-1[35] is to achieve multiple simulations in parallel, then to fill several mini-batches with the shuffled data, and eventually to sequentially use the mini-batches to update the NN. The parallelization process is thus very similar to the one of Rabault and Kuhnle.[115]

Massively distributed architectures on multiple GPUs have been implemented to accelerate DRL methods, such as DQN with the

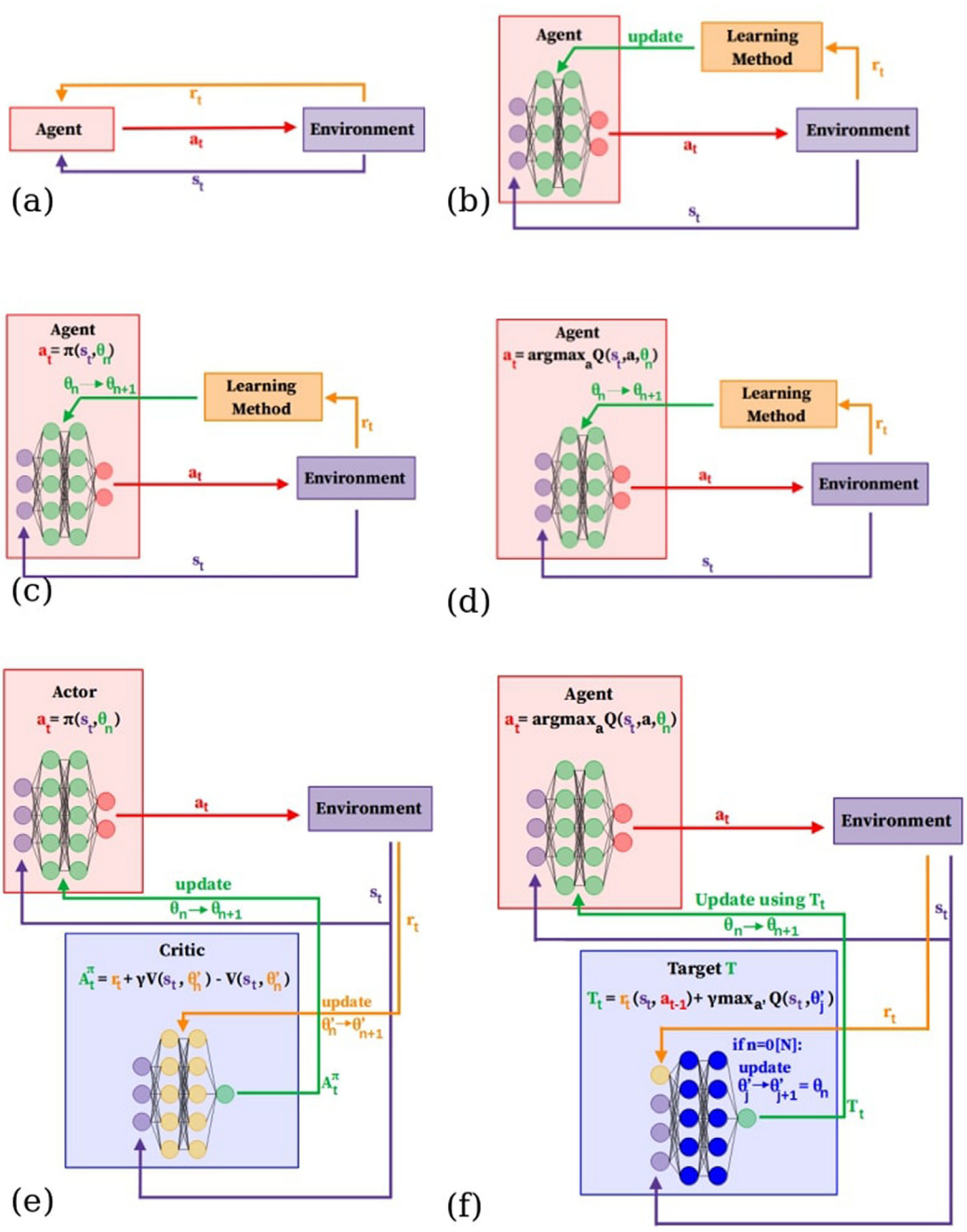


FIG. 4. Comparison of (a) RL, (b) DRL, (c) policy-based DRL, (d) value-based DRL, (e) advantage actor–critic DRL, and (f) DQN (simplified) architectures.

Gorilla framework.[116] Multiple agents act in parallel on identical copies of the environment, have their own replay buffer, and compute the loss function similarly to the DQN process [Eq. (20)]. A central actor merges the information from all the agents and then enforces the same ANN update to each of them. With 100 parallel agents, Gorilla outperforms DQN on many Atari games, with computations approximately twenty times faster. A similar process is also suggested by Ong, Chavez, and Hong,[117] whereas Mnih *et al.*[94] implemented the same principle but with multiple CPU threads on a single machine, thereby outperforming the previous methods while offering the possibility to have stable value-based and policy-based, on-policy and off-policy efficient strategies. Indeed, an important improvement brought by the parallel agents is that it stabilizes value-based methods even without any replay buffer,[94] thus allowing more efficient on-policy methods.

Eventually, Belus *et al.*[118] exploited for the first time the translational invariance and the locality of the control problem in a DRL algorithm, in order to enhance its efficiency. Their work is not a parallelization process but also enables one to reach more complex problems, and it will be more extensively discussed in Sec. IV.

IV. DEEP REINFORCEMENT LEARNING FOR FLOW CONTROL

A. Introduction passive and active control

Flow control strategies are usually separated into two categories:[53] passive and active methods. The former include, e.g., riblets and vortex generators[54] and other LEBUs (large-eddy-breakup devices).[119] They do not need a supply in energy nor a feedback from the environment. Their robustness, low manufacturing costs, and global simplicity have notably enabled them to be applied in real-world systems, contrarily to the active methods, which are often complex to implement concretely. However, the active methods present the advantage to be generally more efficient and in a wider range of operating conditions. For instance, while riblets used for drag reduction in turbulent boundary layers (TBLs) can perform a 5%–9% drag reduction,[120] modern active methods can perform reductions twice as much important, or even more (see Ref. 54 or Ref. 23 for a recent review on flow control in turbulent boundary layers).

Passive methods are intrinsically open-loop as they do not consider the evolution of the surrounding environment. However, active methods can be either open-loop or closed-loop. Open-loop active methods, by not updating the actuators with a potential feedback of the environment state, are sub-optimal. Yet, having a good understanding of the open-loop behavior is almost a prerequisite to the further development of closed-loop actuators.[54] For instance, actuators used to control flow separation have first proved to be efficient in periodic regimes[121,122] and paved the way to modern closed-loop actuators (see Ref. 38 for an example of closed-loop flow control using synthetic jets).

For historical and comprehensive reviews on active and passive methods, the reader may refer to the works of Gad-el Hak, Pollard, and Bonnet,[123] Moin and Bewley,[124] and Lumley and Blossey,[125] and to the review of Pino *et al.*[24] for the use of machine learning methods in flow control. The objective of this section is to give a general overview of the last advances in active flow control (AFC) and more particularly in the application of the DRL methods to that field. The current challenges and upcoming milestones in the combination of DRL and AFC are highlighted.

B. Active flow control

Active flow control has grown at the frontier of various traditional fields, including fluid mechanics, control theories, and machine learning.[126] Thus, it has inherently been confronted to terminology conflicts,[126–130] but has developed fruitful hybrid methods during the past two decades. Indeed, this interdisciplinary effort has been carried out with the prospect of many promising applications,[21,24] such as delaying the transition past airfoils/aircraft wings,[36,131] reducing the drag coefficient past bluff bodies (see, e.g., Ref. 106), or controlling convective heat transport,[132] to cite a few.

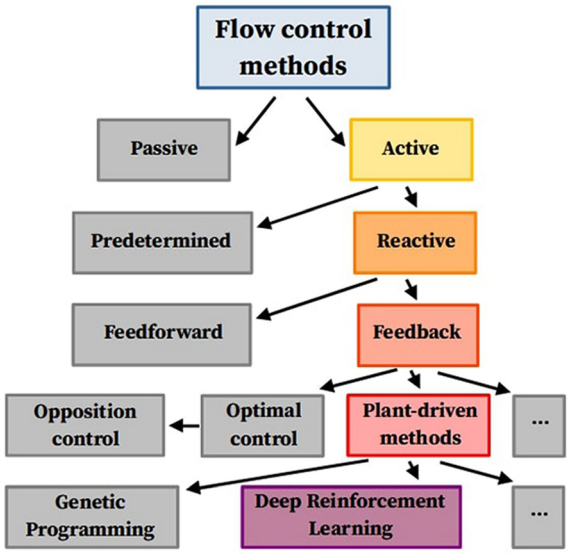
Conversely to the passive methods, the active control of a flow needs a supply in energy. In addition, the data used for the control can be provided by experimental results or CFD simulations. These data can be used during the control sequence (reactive control) or only before (predetermined, open-loop control). As feed-forward methods are not considered in the present review, active reactive feedback methods will simply be called feedback control thereafter (see Fig. 5). In terms of terminology reconciliation, open-loop and closed-loop (D)RL methods applied to the control of a flow are, respectively, assimilated to open-loop and feedback control. The closed-loop DRL methods are thus an active reactive feedback data-driven strategy (see Fig. 5).

1. Predetermined, open-loop control

The open-loop control of a flow is regardless of its instantaneous state. The control sequence is determined in advance (thus "predetermined") and then applied without any feedback. It facilitates the control, since no sensors are needed and no feedback loop has to be implemented.[43]

In a numerical approach, Jung, Mangiavacchi, and Akhavan[133] and then Quadrio and Ricco[134] enforce spanwise oscillations at the walls of a fully developed turbulent channel, and compute reductions of the turbulent drag of the order of 40%. Several studies[135–137] also consider open-loop control based on blowing and suction in turbulent wings, also leading to over 10% improvement in aerodynamic efficiency. Alongside these direct numerical simulation approaches, experimental versions of the spanwise oscillating walls have been implemented,[138–140] and important but lower values (25%–35%) of drag reduction are observed. Quadrio and Ricco[134] also report a net energy saving of 7% (taking into account the power necessary for the wall oscillations), hence highlighting that even without a feedback algorithm, sensors or complex actuators, an open-loop control method can be efficient.

Other open-loop methods have been tested in the field of flow control. One may refer to Refs. 141 and 142 for similar but streamwise blowing and suction, to Refs. 131 and 143 for uniform blowing and



suction, and to Ref. 131 for a periodic blowing on a wing flap (high-lift device). All these studies illustrate the efficiency of actuators used in a steady or cyclic manner with open-loop algorithms. Nevertheless, as they do not take into account the evolution of the dynamics in the flow, their range of efficiency remains limited. For instance, Quadrio and Ricco[134] explained that the input parameters of the oscillations depend on the Reynolds number. In complex turbulent flows, open-loop methods thus encounter limitations similar to the passive methods.

2. Feedback control

Feedback control includes a closed-loop process to take into account the instantaneous evolution of the flow. The number of actuators, their degrees of freedom, and the quantity of input data—collected by one or more sensors—can make these methods challenging. Within these methods, multiple processes can be distinguished. Opposition control methods,[144] optimal control (see, e.g., Ref. 145), genetic programming,[45] reinforcement learning,[55] and deep reinforcement learning[20] applied to AFC problems are successively discussed in Sec. IV B 2 and compared in view of the milestones to reach. If well implemented, the feedback should enhance the efficiency of the control sequence, by adapting itself to the "live" state of the flow. For instance, Ren, Wang, and Tang[49] obtained an encouraging 20% improvement of the control performances by applying a closed-loop process rather than any open-loop method in the AFC problem they consider.

Opposition control may be seen as a first-order derivative of optimal control (see the next paragraph for optimal control). Indeed, opposition control is the optimal control obtained when controlling systems with first-order dynamics and is, therefore, a recurring form of control, as it applies broadly to problems in which a linearized system around a state of equilibrium is an accurate description of the system to control. Opposition control has a wide range of applications in fluid dynamics.[21] For instance, Choi, Moin, and Kim[144] used opposition control to reduce the drag generated by the skin friction in a fully developed turbulent channel flow, by trying to annihilate the coherent structures in the boundary layer with a feedback process. These coherent structures play a preponderant role in the transport of turbulences,[146,147] and such a drag reduction would have numerous applications, since this type of wall-bounded flows is classical in engineering problems. The idea behind opposition control is to apply local blowing and suction at the wall according to a sensing detection plane, parallel to the wall, and typically located at $h^+ = 15$ wall units from the wall.[148] For instance, if a normal speed of $v(x, y, h^+)$ is detected at the position (x, y, h^+) in the detection plane, a normal blowing (or suction, according to the sign of v) of $-\alpha v(x, y, h^+)$ is applied at the point $(x, y, 0)$ of the wall (with $\alpha > 0$ the linear coefficient). By controlling either the normal (as presented above) and/or spanwise velocities in such a simple feedback control, a skin friction reduction of approximately 20%–25% is obtained.[144,149] These methods are thus simple and show relatively efficient results, but they become unusable in more complex flows. Indeed, in the standard situation presented above, the relationship between the detected speed and the actuation is considered to be linear, and the optimal coefficient (α) is found by trial and error. Yet, this trial-and-error process is quite inefficient, even in the linear approximation.[43] Additionally, the drag reduction rate obtained by opposition control in turbulent boundary layers decreases with increasing Reynolds numbers.[150] Since then, the opposition control method is not aligned with the current objectives of controlling more sophisticated and turbulent flow conditions.

Optimal control can efficiently optimize an input parameter, even with large degrees of freedom, by directly tackling the mathematical equations of the flow, and showed successful results in AFC in a weakly turbulent channel configuration.[151] In optimal control, the dynamical evolution of the system is explicitly developed until a certain time horizon and then "rewound" back to the initial state in order to solve the adjoint equations and compute a certain cost function. The optimal value of the parameter at stake is then tuned according to the cost function. As the non-negligible time horizon and the resolution of the adjoint equations require important computation time and resources, a sub-optimal theory has been developed,[152,153] carrying the strategy of setting an arbitrary small time horizon (thus suppressing the necessity to solve the adjoint equations[43]) while limiting the deterioration of the optimization. However, sub-optimal control solutions are limited. Indeed, the more complex and turbulent the flow becomes, the more longer the time horizon needs to be, in order to have a good understanding of the dynamics of the system (for a further optimization of the parameter at stake). Additionally, the time horizon cannot be extended endlessly in optimal control, the adjoint equations being unstable beyond a certain threshold.[154] Consequently, in prospect of (actively) controlling flows with high Reynolds numbers, the optimal control theory may present non-negligible shortcomings.

Machine-learning methods are promising, since they could overcome the previous flaws encountered within the opposition and optimal control methods. They take various shapes but are gathered by the same preeminence of/need for consequent data.

- The Bayesian optimization (BO) and Lipschitz global optimization (LIPO) processes are black-box global optimization techniques.[24] Compared to the upcoming genetic programming and RL methods, they present the advantage to be less complex, as they are linear and quadratic function approximators, respectively, and to require less computational time. In the classical BO method, the function to optimize (e.g., the cumulative reward) is modeled by a Gaussian process. Given a certain data set, one can thus obtain a probability distribution of the values taken by the function at stake. In the BO iterative process, a function suggesting where to sample next is also defined. The LIPO method relies on the same basis, but with a different surrogate function instead of the Gaussian process. One can refer to the work of Pino *et al.*[24] for a more extensive and complete formulation of the BO and LIPO methods applied to AFC, and to the work of Blanchard *et al.*[155] for the first application of open-loop BO in AFC.
- A genetic programming control algorithm is a nature-inspired feedback process that relies on the succession of populations.[45] A population of individuals (i.e., a set of control laws) is first generated and applied to the environment (i.e., a numerical simulation or experiment per control law is run), and then, multiple processes (replication, cross-over, mutation amidst the "individuals," i.e., the control laws) modify the previous population to produce a new one, and gradually optimize a function (e.g., the cumulative reward). Examples of successful GP applications in AFC can be found in Ref. 46 for a separation control issue, in Ref. 47 for a

drag-reduction-behind-a-car problem, in Ref. 48 for the mixing optimization of a turbulent jet, in Ref. 50 for an example of open-loop GP control, or more quantitatively in the review proposed by Noack.[156] The aforementioned applications use experimental setups to feed the closed-loop process with feedback data. Ren, Wang, and Tang[49] presented an innovative combination of CFD and GP-based AFC to suppress the vorticity induced by a vibrating cylinder. Their results outperform the overall open-loop methods by approximately 20% and thus arouse encouraging prospects.

- Eventually, the application of DRL in AFC has already shown promising results and overtook the majority of the results obtained with previous methods. With the objective of increasing the complexity of the handled flow configurations, DRL methods appear to be the most relevant contender. As a consequence, a specific section is dedicated to these methods (see Sec. IV C).

By applying PPO to laminar flow conditions ($Re = 100$), Tokarev, Palkin, and Mullyadzhanov[157] obtained performances comparable to the ones achieved with an adjoint-based method in Ref. 158. However, in the situation of a wall-bounded channel flow, and even at a low friction Reynolds number (of about 180), the DRL algorithm developed by Guastoni *et al.*[44] outperforms the opposition control method by over 20% points, with drag reductions of 46% and 20%, respectively. Furthermore, opposition and optimal control reach limitations when it comes to complex tasks such as controlling high-Reynolds turbulent flows, because the design of a relevant explicit control law becomes challenging, if not impossible. Pino *et al.*[24] additionally illustrated that GP and DDPG outperform BO and LIPO on three showcases with progressive difficulty. Nonetheless, as the generality of the function approximator increases, the number of episodes required for the training phase grows in proportion. Therefore, it may worth combining GP and DRL algorithms with "simpler" methods,[24,159,160] or associating model-free DRL algorithms with environments partially modeled thanks to the physical understanding of the dynamics of the system (see, e.g., Refs. 42 and 161 and the references therein). Compared to DRL, GP presents the advantage of performing better with fewer sensors.[162] However, DRL methods are inherently more suitable for multi-input multi-output problems than GP.[106] Section IV of this review will therefore be devoted to the application of DRL to AFC.

C. Deep reinforcement learning and active flow control: Challenges and prospects

Within the field of fluid dynamics, the deep reinforcement learning methods can be applied to a variety of problems. Historically, the first one considered consisted in the behavioral study of swimmers in shoal formations (see Refs. 163–165 and related works), where the DRL algorithm could choose the position and movements of one/multiple swimmer(s) placed in the wake of a leader. Not only did those studies enhance the scientific knowledge on complex energy-saving processes, such as the interception and optimization of vortices by the followers, but it may also have engineering repercussions and benefits in real-world applications.[166] Hence, these successful results gave a first overview of the capability and efficiency of DRL algorithms to handle complex physical problems and paved the way to multiple applications in the field of fluid dynamics.[42,167–169]

Nowadays, numerous DRL agents have been implemented in order to, e.g., optimize the position of a small static (e.g., Ref. 22) or moving (e.g., Refs. 100 and 170) body around a larger one with the prospect of reducing the recirculation bubble/the drag downstream, to directly modify the shape of the principal body,[52,171–173] or to control its movements[106,157] with the same objective of optimizing its aerodynamic/hydrodynamic properties. Other research works handle heat transport issues such as the Rayleigh–Bénard instability[132] or other convectively unstable flows.[174]

Rabault *et al.*[38] lead the first study applying a DRL algorithm (PPO) to an AFC problem. Two small synthetic jets are placed symmetrically on the top and bottom of a cylinder immersed in a 2D flow, aimed at shrinking the instabilities behind the body (i.e., the Kármán vortex street). This bi-dimensional showcase, inspired by the original setup of Schäfer *et al.*,[175] is now a generic benchmark used to test the recent developments of DRL in the field and has therefore known great and recurrent interest.[39–41,115] Other studies also consider AFC problems using actuators, to control the flow separation[36,176] or to limit the skin friction in wall-bounded channels,[43,44] to cite a few.

The community now redoubles its efforts on increasing the complexity of the AFC problems handled with DRL. The first upcoming milestone consists in proving the efficiency of the existing algorithms when it comes to highly turbulent conditions. Issues of instabilities in turbulent boundary layers[177,178] (TBLs) would for instance become reachable, then with fruitful engineering prospects (see, e.g., Ref. 179 for the control of a TBL with genetic algorithms). For this purpose, successive research works have for instance increased the Reynolds number in the originally laminar cylinder problem treated in Ref. 38 with a Reynolds number of $Re = 100$. They successfully proved the efficiency of PPO in a range $Re \in [60; 400]$,[39] then for the first time in a weakly turbulent condition ($Re = 1000$)[40] and recently reached $Re = 2000$.[41] In Ref. 41, the algorithm suggests a strategy that differs significantly from the opposition-control solution. PPO is often chosen for its relative simplicity while having comparable performances to the other DRL algorithms because flow control issues generally require continuous actuation (thus not reachable with DQN for instance). Yet, DDPG also raises promising results. In Ref. 96, the authors stabilize a chaotic system governed by the 1D Kuramoto–Sivashinsky equation with only partial observations of the environment, thereby answering some concerns of the community on the efficiency of RL methods in partial observations situations.[129] In their appendix, Beintema *et al.*[132] also confirmed the efficiency of DRL (PPO) to control chaotic systems (the Lorentz attractor). These two results are therefore encouraging with regard to the increase and control of turbulence by DRL.

For instance, a second milestone to reach is the enlargement of the situations/flows,[40] going to three-dimensional simulations. A few works have already been led in three dimensions (see, e.g., Refs. 100, 131, and 131 for model-free methods, or Ref. 62 for a model-based application). A pioneering exploitation of invariances by Belus *et al.*[118] could enable DRL algorithms to deal with an arbitrary large number of actuators while highly limiting the increase in the training cost, and thereby give access to larger AFC problems. More generally, the study of symmetries, equivariances, and invariances may enable one to consider larger and more turbulent flow conditions, but it will require a non-trivial implementation. Indeed, if the environment holds symmetries, or if the governing equations contain invariances or equivariances, the ANN will not naturally exploit them. For instance, with a

"naive" ANN, one is not sure that the agent will preserve theoretically equivariant actions in reality.[180] Nonetheless, with sufficient training data, the ANN may learn invariant and equivariant properties by setting some constraints between the weights of the network, but it will then reduce its degrees of freedom.[181,182] Hence, if one wants a constant number of degrees of freedom and take into account equivariant and invariant properties, one may require a larger ANN and thus more training data. To prevent such an increase, one has to support the ANN to take advantage of the invariances, equivariances, and symmetries when they exist, by encoding a state or state–action reduced space in which the agent can work equivalently to the real environment. Zeng and Graham[180] encode a "symmetry–reductor" applied to a 1D Kuramoto–Sivashinsky environment controlled by DDPG. They find robust strategies stabilizing the chaotic system in the state–action symmetry-reduced subspace and prove the enhanced efficiency of DDPG in this subspace in comparison with the "fully developed" space. Their work is an encouraging milestone illustrating how important it will be to consider equivariances, invariances, and symmetries when they exist in chaotic/turbulent flow control problems, and how to do so. Eventually, by defining strategies of cooperation between multiple parallel agents, multi-agent reinforcement learning (MARL) has recently proved its efficiency in fluid dynamics[27,44] and may help enhancing the complexity of the flows considered. Within the MARL framework, multiple agents act in parallel on identical copies of the environment. A central actor merges the information from all the agents and then enforces the same ANN update to each of them, similarly to the Gorilla process described in Sec. III D. The MARL method may be used to take advantage of physical invariants and then enable one to reduce the duration of the training period without increasing the number of processors.[118]

Encouraging avenues for the raise and enlargement of complexity are thus already under study. In addition, the aforementioned parallelization methods (see, e.g., Sec. III D and Ref. 115 for a direct application in AFC) are also promising, as the number of episodes needed for the training of the NN increases in proportion with the complexity of the flow. Both invariances and parallel DRL should be considered to deal with large turbulent flows. Transfer learning[183] is also of great interest. Multiple studies have already illustrated the effectiveness of DRL-learned strategies in situations different from the ones considered in the training phase (see, for instance, Refs. 39, 40, and 106), and their robustness when confronted to an input or output noise (e.g., Refs. 174 and 180). This ability the agent has to transfer its learning to a wider range of flow conditions proves – once more – the robustness of DRL algorithms and their advantageous properties compared to other AFC methods. Moreover, a recent work[111] has shown an interest in reducing the number of actuators while limiting the deterioration of the control sequence, exploiting the malleability of NNs. It could thus enable one to deal with larger problems without increasing the number of actuators. Eventually, DRL as well as machine learning in general may also enhance knowledge on physical processes, where the classical mechanics predictions encounter limitations,[156] such as the behavior of chaotic systems. Taking the example of AlphaGo,[6] not only did the algorithm impress the community for its performances, but also because it helped understanding underlying structures in the game, for instance by choosing strategies commonly thought as inefficient until then. Therefore, when applied to chaotic systems, DRL methods may highlight unknown structures and discover new physical properties. The non-opposition-control solution found by PPO in Ref. 41 goes in this direction. In other words, DRL methods may help enhance our knowledge on physical systems;[58] meanwhile, our current knowledge needs to be exploited to encode better algorithms, taking into account symmetries and invariances for instance. In Ref. 42, the performances of a reinforcement-learning algorithm are enhanced thanks to a sensitivity analysis. Inversely, the RL algorithm developed in Ref. 180 finds a new equilibrium state and successfully stabilizes the system (the Kuramoto–Sivashinsky equation). These two works go along with the direction of both gaining physical knowledge from the RL algorithm and helping the RL algorithm thanks to our current knowledge.

Hence, data-driven methods, and more particularly DRL algorithms, have already demonstrated unrivaled performances in active flow control problems. They now have to handle larger and more complex situations, as the latter would raise even more engineering applications and scientific knowledge.

As a last comment, we would emphasize the importance of open-sourcing the codes of the DRL algorithms applied to flow control problems. As it has been developed in this review, fruitful methods may emerge from combinations of existing algorithms. For a reproducibility purpose, and because technical implementation choices may have a significant influence on the results, sharing the training data alongside the code and the theoretical details seems to be essential.[58] For example of shared toolkits, the reader may refer to Refs. 184 and to 185, which, respectively, present an open-source Python platform for DRL and an open-source GP technique for fluid mechanics.

V. CONCLUDING REMARKS

In the present review, a general overview of the main RL concepts and of the DRL framework has first been given. The potential of DRL for the active control of a flow has then been highlighted—comparatively to either reactive or open-loop other methods. Not only may the DRL effectiveness be useful in multiple engineering applications, but it could also help science in general. Indeed, this exhaustive exploration tool may provide knowledge in theoretical physics fields similarly to what it gave to the Go-game-community: the discovery of previously unknown underlying structures in the studied system.

Since the formulation of the Navier–Stokes equation 200 years ago, science has focused on the application of analytical methods. This often implies local linearization, as well as stability and modal analyses. However, in many instances this is not meaningful, because there is no such thing as a clearly defined base flow that is similar enough to instantaneous snapshots to perform linearization. Therefore, full-scale applications may be far enough from a well-defined configuration that linearization is not representative. Consequently, in general, when handling non-linear high-dimensional systems, there is no choice but to study directly the full system. Thus, most analytical tools cannot be used anymore, and this is one of the reasons why methods such as DRL are attractive: they can perform very well in much more general conditions than traditional methods, at the cost of being very data intensive. Hence, DRL represents a methodological shift. It will take time before we, as a community, understand the full value and impact of these methods in AFC, and we are currently in the middle of a massive effort to investigate how far DRL can be taken for AFC applications, and what new insight can be gained from that.

ACKNOWLEDGMENTS

R.V. acknowledges financial support from ERC Grant No. "2021-CoG-101043998, DEEPCONTROL." Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

AUTHOR DECLARATIONS

Conflict of Interest

The authors have no conflicts to disclose.

Author Contributions

Colin Vignon: Writing – review & editing (lead). **Jean Rabault:** Supervision (equal); Validation (equal); Writing – review & editing (equal). **Ricardo Vinuesa:** Conceptualization (equal); Supervision (equal); Validation (equal); Writing – review & editing (equal).

DATA AVAILABILITY

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

REFERENCES

[1]J. Kober, J. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," Int. J. Rob. Res. **32**, 1238–1274 (2013).
[2]S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2017), pp. 3389–3396.
[3]OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," arXiv:1808.00177 (2018).
[4]J. Li, W. Monroe, A. Ritter, D. Jurafsky, M. Galley, and J. Gao, "Deep reinforcement learning for dialogue generation," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, Austin, TX, 2016), pp. 1192–1202.
[5]D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. baker, M. Lai, A. Bolton, Y. Chen, T. P. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," Nature **550**, 354–359 (2017).
[6]D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature **529**, 484–489 (2016).
[7]N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," Science **365**, 885–890 (2019).
[8]V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," arXiv:1312.5602 (2013).
[9]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**, 529–533 (2015).
[10]I. Szita, "Reinforcement learning in games," in *Reinforcement Learning: State-of-the-Art*, edited by M. Wiering and M. van Otterlo (Springer, Berlin, Heidelberg, 2012), pp. 539–577.
[11]A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, edited by F. Pereira, C. Burges, L. Bottou, and K. Weinberger (Curran Associates, Inc., 2012), Vol. 25.
[12]K. Hornik, M. B. Stinchcombe, and H. L. White, "Multilayer feedforward networks are universal approximators," Neural Networks **2**, 359–366 (1989).
[13]G. Tesauro, "Temporal difference learning and TD-gammon," Commun. ACM **38**, 58–68 (1995).
[14]J. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (IEEE, 2001), Cat. No.01CH37164, Vol. 2, pp. 1615–1620.
[15]A. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, and E. Berger, "Inverted autonomous helicopter flight via reinforcement learning," in Proceedings of the International Symposium on Experimental Robotics (2004).
[16]J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems* (IEEE, 2006), pp. 2219–2225.
[17]J. Peters and S. Schaal, "Learning to control in operational space," Int. J. Rob. Res. **27**, 197–212 (2008).
[18]C. Diuk, A. Cohen, and M. L. Littman, "An object-oriented representation for efficient reinforcement learning," in Proceedings of the International Conference on Machine Learning (2008).
[19]M. Riedmiller, T. Gabel, R. Hafner, and S. Lange, "Reinforcement learning for robot soccer," Auton. Rob. **27**, 55–73 (2009).
[20]R. Sutton and A. Barto, *Reinforcement Learning: An Introduction* (MIT Press, 2018).
[21]S. L. Brunton and B. R. Noack, "Closed-loop turbulence control: Progress and challenges," Appl. Mech. Rev. **67**, 050801 (2015).
[22]P. Garnier, J. Viquerat, J. Rabault, A. Larcher, A. Kuhnle, and E. Hachem, "A review on deep reinforcement learning for fluid mechanics," Comput. Fluids **225**, 104973 (2021).
[23]R. Vinuesa, O. Lehmkuhl, A. Lozano-Durán, and J. Rabault, "Flow control in wings and discovery of novel approaches via deep reinforcement learning," Fluids **7**, 62 (2022).
[24]F. Pino, L. Schena, J. Rabault, and M. A. Mendez, "Comparative analysis of machine learning methods for active flow control," arXiv:2202.11664 (2022).
[25]G. Novati, H. L. de Laroussilhe, and P. Koumoutsakos, "Automating turbulence modelling by multi-agent reinforcement learning," Nat. Mach. Intell. **3**, 87–96 (2021).
[26]M. Kurz, P. Offenhäuser, and A. Beck, "Deep reinforcement learning for turbulence modeling in large eddy simulations," Int. J. Heat Fluid Flow **99**, 109094 (2023).
[27]H. J. Bae and P. Koumoutsakos, "Scientific multi-agent reinforcement learning for wall-models of turbulent flows," Nat. Commun. **13**, 1443 (2022).
[28]A. Beck, D. Flad, and C.-D. Munz, "Deep neural networks for data-driven turbulence models," in *Proceedings of the APS Division of Fluid Dynamics Meeting Abstracts*, APS Meeting Abstracts (APS, 2019), p. G16.006.
[29]A. Beck, D. Flad, and C.-D. Munz, "Deep neural networks for data-driven LES closure models," J. Comput. Phys. **398**, 108910 (2019).
[30]A. Beck and M. Kurz, "A perspective on machine learning methods in turbulence modeling," GAMM-Mitt. **44**, e202100002 (2021).
[31]M. Kurz, P. Offenhäuser, D. Viola, O. Shcherbakov, M. Resch, and A. Beck, "Deep reinforcement learning for computational fluid dynamics on HPC systems," J. Comput. Sci. **65**, 101884 (2022).
[32]M. Kurz, P. Offenhaeuser, D. Viola, M. Resch, and A. Beck, "Relexi—A scalable open source reinforcement learning framework for high-performance computing," Software Impacts **14**, 100422 (2022).
[33]A. Lampton, A. Niksch, and J. Valasek, "Morphing airfoils with four morphing parameters," AIAA Paper No. 2008-7282, 2008.
[34]A. Lampton, A. Niksch, and J. Valasek, "Reinforcement learning of a morphing airfoil-policy and discrete learning analysis," J. Aerosp. Comput. Inf. Commun. **7**, 241–260 (2010).
[35]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," Phys. Rev. Fluids **6**, 053902 (2021).

[36]A. Batikh, L. Baldas, and S. Colin, "Application of active flow control on aircrafts-state of the art," in Proceedings of the International Workshop on Aircraft System Technologies, Hamburg, Germany (2017), Vol. 2017.
[37]B. R. Noack, Y. Li, Z. Yang, G. Y. Cornejo Maceda, F. Lusseyran, P. Oswald, and R. Semann, "Machine learning drag reduction of car and truck models with multiple actuators and sensors," in Proceedings of the Aerovehicles 4, Berlin, Germany (2021).
[38]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," J. Fluid Mech. **865**, 281–302 (2019).
[39]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," Phys. Fluids **32**, 053605 (2020).
[40]F. Ren, J. Rabault, and H. Tang, "Applying deep reinforcement learning to active flow control in weakly turbulent conditions," Phys. Fluids **33**, 037121 (2021).
[41]P. Varela, P. Suárez, F. Alcántara-Ávila, A. Miró, J. Rabault, B. Font, L. M. García-Cuevas, O. Lehmkuhl, and R. Vinuesa, "Deep reinforcement learning for flow control exploits different physics for increasing Reynolds-number regimes," Actuators **11**, 359 (2022).
[42]J. Li and M. Zhang, "Reinforcement-learning-based control of confined cylinder wakes with stability analyses," J. Fluid Mech. **932**, A44 (2021).
[43]T. Sonoda, Z. Liu, T. Itoh, and Y. Hasegawa, "Reinforcement learning of control strategies for reducing skin friction drag in a fully developed channel flow," arXiv:2206.15355 (2022).
[44]L. Guastoni, J. Rabault, P. Schlatter, H. Azizpour, and R. Vinuesa, "Deep reinforcement learning for turbulent drag reduction in channel flows," arXiv:2301.09889 (2023).
[45]J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (The MIT Press, 1992).
[46]N. Gautier, J.-L. Aider, T. Duriez, B. Noack, M. Segond, and M. Abel, "Closed-loop separation control using machine learning," J. Fluid Mech. **770**, 442–457 (2015).
[47]R. Li, B. R. Noack, L. Cordier, J. Borée, and F. Harambat, "Drag reduction of a car model by linear genetic programming control," Exp. Fluids **58**, 103 (2017).
[48]D. Fan, Y. Zhou, and B. Noack, "Artificial intelligence control of turbulence," in Proceedings of the *5th Symposium on Fluid Structure-Sound Interactions and Control (FSSIC)* (Minoa Palace-Resort, Chania, Crete Island, Greece, 2019), pp. 1–4.
[49]F. Ren, C. Wang, and H. Tang, "Active control of vortex-induced vibration of a circular cylinder using machine learning," Phys. Fluids **31**, 093601 (2019).
[50]H. Li, J. Tan, Z. Gao, and B. R. Noack, "Machine learning open-loop control of a mixing layer," Phys. Fluids **32**, 111701 (2020).
[51]H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga, and E. Hachem, "Single-step deep reinforcement learning for two- and three-dimensional optimal shape design," AIP Adv. **12**, 085108 (2022).
[52]J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem, "Direct shape optimization through deep reinforcement learning," J. Comput. Phys. **428**, 110080 (2020).
[53]M. Gad-el Hak, "Modern developments in flow control," Appl. Mech. Rev. **49**, 365–379 (1996).
[54]S. S. Collis, R. D. Joslin, A. Seifert, and V. Theofilis, "Issues in active flow control: Theory, control, simulation, and experiment," Prog. Aerosp. Sci. **40**, 237–289 (2004).
[55]R. S. Sutton, "Learning to predict by the methods of temporal differences," Mach. Learn. **3**, 9–44 (1988).
[56]M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," in Proceedings of the International Conference on Learning Representations (2017).
[57]D. Ha and J. Schmidhuber, "World models," arXiv:1803.10122 (2018).
[58]J. Rabault and A. Kuhnle, *Deep Reinforcement Learning Applied to Active Flow Control* (Cambridge University Press, 2022).
[59]A. Barbagallo, G. Dergham, D. Sipp, P. Schmid, and J.-C. Robinet, "Closed-loop control of unsteadiness over a rounded backward-facing step," J. Fluid Mech. **703**, 326–362 (2012).
[60]J. A. Dahan, A. S. Morgans, and S. Lardeau, "Feedback control for form-drag reduction on a bluff body with a blunt trailing edge," J. Fluid Mech. **704**, 360–387 (2012).
[61]N. Gautier and J. L. Aider, "Feed-forward control of a perturbed backward-facing step flow," J. Fluid Mech. **759**, 181–196 (2014).
[62]R. D. Brackston, J. M. García de la Cruz, A. Wynn, G. Rigas, and J. F. Morrison, "Stochastic modelling and feedback control of bistability in a turbulent bluff body wake," J. Fluid Mech. **802**, 726–749 (2016).
[63]J. Kim and T. R. Bewley, "A linear systems approach to flow control," Annu. Rev. Fluid Mech. **39**, 383–417 (2007).
[64]G. Rigas, A. S. Morgans, R. D. Brackston, and J. F. Morrison, "Diffusive dynamics and stochastic models of turbulent axisymmetric wakes," J. Fluid Mech. **778**, R2 (2015).
[65]C. W. Rowley and S. T. Dawson, "Model reduction for flow analysis and control," Annu. Rev. Fluid Mech. **49**, 387–417 (2017).
[66]N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker, "Surrogate-based analysis and optimization," Prog. Aerosp. Sci. **41**, 1–28 (2005).
[67]S. Koziel, Y. Tesfahunegn, A. Amrit, and L. T. Leifsson, "Rapid multi-objective aerodynamic design using co-kriging and space mapping," AIAA Paper No. 2016-0418, 2016.
[68]K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," arXiv:1805.12114 (2018).
[69]R. Bellman, "A Markovian decision process," J. Math. Mech. **6**, 679–684 (1957).
[70]R. A. Howard, *Dynamic Programming and Markov Processes* (Technology Press of Massachusetts Institute of Technology, 1960).
[71]D. Silver, G. Lever, N. M. O. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in Proceedings of the International Conference on Machine Learning (2014).
[72]C. Watkins, "Learning from delayed rewards," Ph.D. thesis (King's College, Cambridge, UK, 1989).
[73]R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Mach. Learn. **8**, 229–256 (1992).
[74]P. Meliga, D. Sipp, and J.-M. Chomaz, "Open-loop control of compressible afterbody flows using adjoint methods," in *Proceedings of the Seventh IUTAM Symposium on Laminar-Turbulent Transition*, edited by P. Schlatter and D. S. Henningson (Springer, Dordrecht, The Netherlands, 2010), pp. 283–288.
[75]A. F. Shahrabi, "The control of flow separation: Study of optimal open loop parameters," Phys. Fluids **31**, 035104 (2019).
[76]T. Degris, M. White, and R. S. Sutton, "Linear off-policy actor-critic," in Proceedings of the International Conference on Machine Learning (2012).
[77]C. Watkins and P. Dayan, "Q-learning," Mach. Learn. **8**, 279–292 (1992).
[78]R. S. Sutton and S. D. Whitehead, "Online learning with random representations," in Proceedings of the International Conference on Machine Learning (1993).
[79]T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *Proceedings of the American Control Conference (ACC)* (IEEE, Piscataway, NJ, 2012), pp. 2177–2182.
[80]R. Bellman and S. Dreyfus, *Applied Dynamic Programming* (Princeton University Press, 1962).
[81]R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, edited by S. Solla, T. Leen, and K. Müller (MIT Press, 1999), Vol. 12.
[82]G. Tesauro and T. J. Sejnowski, "A parallel network that learns to play backgammon," Artif. Intell. **39**, 357–390 (1989).
[83]R. H. Crites and A. G. Barto, "Improving elevator performance using reinforcement learning," in Proceedings of the NIPS (1995).
[84]R. S. Sutton, D. A. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in Proceedings of the Neural Information Processing Systems (2000).
[85]L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," Mach. Learn. **8**, 293–321 (1992).
[86]M. Riedmiller, M. Montemerlo, and H. Dahlkamp, "Learning to drive a real car in 20 minutes," in *Proceedings of the 2007 Frontiers in the Convergence of*

Bioscience and Information Technologies (FBIT) (IEEE Computer Society, 2007), pp. 645–650.
[87]Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature **521**, 436–444 (2015).
[88]T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv:1511.05952 (2015).
[89]J. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," IEEE Trans. Autom. Control **42**, 674–690 (1997).
[90]T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv:1509.02971 (2015).
[91]Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," arXiv:1511.06581 (2015).
[92]H. Van Hasselt, *Double Q-Learning* (2010), pp. 2613–2621.
[93]H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," arXiv:1509.06461 (2015).
[94]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:1602.01783 (2016).
[95]J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," arXiv:1502.05477 (2015).
[96]M. A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin, "Control of chaotic systems by deep reinforcement learning," Proc. R. Soc. London, Ser. A **475**, 20190351 (2019).
[97]S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," arXiv:1802.09477 (2018).
[98]J. Viquerat, P. Meliga, A. Larcher, and E. Hachem, "A review on deep reinforcement learning for fluid mechanics: An update," Phys. Fluids **34**, 111301 (2022).
[99]Y. Xie and X. Zhao, "Sloshing suppression with active controlled baffles through deep reinforcement learning–expert demonstrations–behavior cloning process," Phys. Fluids **33**, 017115 (2021).
[100]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," Proc. Natl. Acad. Sci. **117**, 26091–26098 (2020).
[101]J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv:1707.06347 (2017).
[102]S. M. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in Proceedings of the International Conference on Machine Learning (2002).
[103]Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," arXiv:1611.01224 (2016).
[104]I. Szita and A. Lörincz, "Learning tetris using the noisy cross-entropy method," Neural Comput. **18**, 2936–2941 (2006).
[105]X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "DeepMimic: Example-guided deep reinforcement learning of physics-based character skills," arXiv:1804.02717 (2018).
[106]F. Ren, C. Wang, and H. Tang, "Bluff body uses deep-reinforcement-learning trained active flow control to achieve hydrodynamic stealth," Phys. Fluids **33**, 093602 (2021).
[107]P. Lai, R. Wang, W. Zhang, and H. Xu, "Parameter optimization of open-loop control of a circular cylinder by simplified reinforcement learning," Phys. Fluids **33**, 107110 (2021).
[108]P. Hämäläinen, A. Babadi, X. Ma, and J. Lehtinen, "PPO-CMA: Proximal policy optimization with covariance matrix adaptation," arXiv:1810.02541 (2018).
[109]A. Abdolmaleki, J. T. Springenberg, Y. Tassa, R. Munos, N. Heess, and M. Riedmiller, "Maximum a posteriori policy optimisation," arXiv:1806.06920 (2018).
[110]A. Abdolmaleki, J. T. Springenberg, J. Degrave, S. Bohez, Y. Tassa, D. Belov, N. Heess, and M. Riedmiller, "Relative entropy regularized policy iteration," arXiv:1812.02256 (2018).
[111]R. Paris, S. Beneddine, and J. Dandois, "Reinforcement-learning-based actuator selection method for active flow control," J. Fluid Mech. **955**, A8 (2023).
[112]H. D. Simon, *Parallel Computation Fluid Dynamics-Implementations and Results* (National Aeronautics and Space Administration, 1992).
[113]W. D. Gropp and E. B. Smith, "Computational fluid dynamics on parallel processors," Comput. Fluids **18**, 289–304 (1990).
[114]W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith, "High-performance parallel implicit CFD," Parallel Comput. **27**, 337–362 (2001).
[115]J. Rabault and A. Kuhnle, "Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach," Phys. Fluids **31**, 094105 (2019).
[116]A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," arXiv:1507.04296 (2015).
[117]H. Y. Ong, K. Chavez, and A. Hong, "Distributed deep q-learning," arXiv:1508.04186 (2015).
[118]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 Adv. **9**, 125014 (2019).
[119]P.-H. Alfredsson and R. Örlü, "Large-eddy breakup devices—A 40 Years perspective from a Stockholm horizon," Flow Turbul. Combust. **100**, 877–888 (2018).
[120]D. W. Bechert and M. Bartenwerfer, "The viscous flow on surfaces with longitudinal ribs," J. Fluid Mech. **206**, 105–129 (1989).
[121]D. Greenblatt and I. J. Wygnanski, "The control of flow separation by periodic excitation," Prog. Aerosp. Sci. **36**, 487–545 (2000).
[122]A. Seifert, A. Darabi, and I. Wyganski, "Delay of airfoil stall by periodic excitation," J. Aircr. **33**, 691–698 (1996).
[123]M. Gad-el Hak, A. Pollard, and J. P. Bonnet, *Flow Control: Fundamentals and Practices* (Springer, 1998).
[124]P. Moin and T. R. Bewley, "Feedback control of turbulence," Appl. Mech. Rev. **47**, S3–S13 (1994).
[125]J. Lumley and P. Blossey, "Control of turbulence," Annu. Rev. Fluid Mech. **30**, 311–327 (1998).
[126]T. R. Bewley, "Flow control: New challenges for a new renaissance," Prog. Aerosp. Sci. **37**, 21–58 (2001).
[127]R. Sutton, A. Barto, and R. Williams, "Reinforcement learning is direct adaptive optimal control," IEEE Control Syst. Mag. **12**, 19–22 (1992).
[128]H. Bersini and V. Gorrini, "Three connectionist implementations of dynamic programming for optimal control: A preliminary comparative analysis," in *Proceedings of the International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing* (IEEE, Piscataway, NJ, 1996), pp. 428–437.
[129]B. Recht, "A tour of reinforcement learning: The view from continuous control," Annu. Rev. Control Rob. Auton. Syst. **2**, 253–279 (2019).
[130]R. Nian, J. Liu, and B. Huang, "A review on reinforcement learning: Introduction and applications in industrial process control," Comput. Chem. Eng. **139**, 106886 (2020).
[131]T. Kühn, V. Ciobaca, R. Rudnik, M. Bauer, B. Gölling, and W. Breitenstein, "Active flow-separation control on a high-lift wing-body configuration," J. Aircr. **50**, 56–72 (2013).
[132]G. Beintema, A. Corbetta, L. Biferale, and F. Toschi, "Controlling Rayleigh-Bénard convection via reinforcement learning," arXiv:2003.14358 (2020).
[133]W. Jung, N. Mangiavacchi, and R. Akhavan, "Suppression of turbulence in wall–bounded flows by high-frequency spanwise oscillations," Phys. Fluids A **4**, 1605 (1992).
[134]M. Quadrio and P. Ricco, "Critical assessment of turbulent drag reduction through spanwise wall oscillations," J. Fluid Mech. **521**, 251–271 (2004).
[135]M. Atzori, R. Vinuesa, G. Fahland, A. Stroh, D. Gatti, B. Frohnapfel, and P. Schlatter, "Aerodynamic effects of uniform blowing and suction on a NACA4412 airfoil," Flow Turbul. Combust. **105**, 735–759 (2020).
[136]M. Atzori, R. Vinuesa, A. Stroh, D. Gatti, B. Frohnapfel, and P. Schlatter, "Uniform blowing and suction applied to non-uniform adverse-pressure-gradient wing boundary layers," Phys. Rev. Fluids **6**, 113904 (2021).
[137]G. Fahland, A. Stroh, B. Frohnapfel, M. Atzori, R. Vinuesa, P. Schlatter, and D. Gatti, "Investigation of blowing and suction for turbulent flow control on airfoils," AIAA J. **59**, 4422–4436 (2021).

[138]F. Laadhari, L. Skandaji, and R. Morel, "Turbulence reduction in a boundary layer by a local spanwise oscillating surface," Phys. Fluids **6**, 3218 (1994).
[139]K.-S. Choi, "Near-wall structure of turbulent boundary layer with spanwise-wall oscillation," Phys. Fluids **14**, 2530 (2002).
[140]G. Karniadakis and K.-S. Choi, "Mechanisms on transverse motions in turbulent wall flows," Annu. Rev. Fluid Mech. **35**, 45–62 (2003).
[141]T. Min, S. Kang, J. Speyer, and J. Kim, "Sustained sub-laminar drag in a fully developed channel flow," J. Fluid Mech. **558**, 309–318 (2006).
[142]B. K. Lieu, R. Moarref, and M. R. Jovanović, "Controlling the onset of turbulence by streamwise travelling waves. II. Direct numerical simulation," J. Fluid Mech. **663**, 100–119 (2010).
[143]Y. Kametani and K. Fukagata, "Direct numerical simulation of spatially developing turbulent boundary layers with uniform blowing or suction," J. Fluid Mech. **681**, 154–172 (2011).
[144]H. Choi, P. Moin, and J. Kim, "Active turbulence control for drag reduction in wall-bounded flows," J. Fluid Mech. **262**, 75–110 (1994).
[145]R. Sargent, "Optimal control," J. Comput. Appl. Math. **124**, 361–371 (2000).
[146]B. J. Cantwell, "Organized motion in turbulent flow," Annu. Rev. Fluid Mech. **13**, 457–515 (1981).
[147]S. K. Robinson, "Coherent motions in the turbulent boundary layer," Annu. Rev. Fluid Mech. **23**, 601–639 (1991).
[148]E. P. Hammond, T. R. Bewley, and P. Moin, "Observed mechanisms for turbulence attenuation and enhancement in opposition-controlled wall-bounded flows," Phys. Fluids **10**, 2421–2423 (1998).
[149]Y. Chung and T. Tariq, "Effectiveness of active flow control for turbulent skin friction drag reduction," Phys. Fluids **23**, 025102 (2011).
[150]A. Stroh, B. Frohnapfel, P. Schlatter, and Y. Hasegawa, "A comparison of opposition control in turbulent boundary layer and turbulent channel flow," Phys. Fluids **27**, 075101 (2015).
[151]T. R. Bewley, P. Moin, and R. Temam, "DNS-based predictive control of turbulence: An optimal benchmark for feedback algorithms," J. Fluid Mech. **447**, 179–225 (2001).
[152]C. Lee, J. Kim, and H. Choi, "Suboptimal control of turbulent channel flow for drag reduction," J. Fluid Mech. **358**, 245–258 (1998).
[153]Y. Hasegawa and N. Kasagi, "Dissimilar control of momentum and heat transfer in a fully developed turbulent channel flow," J. Fluid Mech. **683**, 57–93 (2011).
[154]Q. Wang, R. Hu, and P. Blonigan, "Least squares shadowing sensitivity analysis of chaotic limit cycle oscillations," J. Comput. Phys. **267**, 210–224 (2014).
[155]A. B. Blanchard, G. Y. Cornejo Maceda, D. Fan, Y. Li, Y. Zhou, B. R. Noack, and T. P. Sapsis, "Bayesian optimization for active flow control," Acta Mech. Sin. **37**, 1786–1798 (2021).
[156]B. R. Noack, "Closed-loop turbulence control-from human to machine learning (and retour)," in *Proceedings of the 4th Symposium on Fluid Structure-Sound Interactions and Control (FSSIC)*, edited by Y. Zhou, M. Kimura, G. Peng, A. D. Lucey, and L. Huang (Springer, 2019), pp. 23–32.
[157]M. Tokarev, E. Palkin, and R. Mullyadzhanov, "Deep reinforcement learning control of cylinder flow using rotary oscillations at low Reynolds number," Energies **13**, 5920 (2020).
[158]T. L. B. Flinois and T. Colonius, "Optimal control of circular cylinder wakes using long control horizons," Phys. Fluids **27**, 087105 (2015).
[159]Y. Li, W. Cui, Q. Jia, Q. Li, Z. Yang, M. Morzyński, and B. R. Noack, "Explorative gradient method for active drag reduction of the fluidic pinball and slanted Ahmed body," arXiv:1905.12036 (2019).
[160]M. A. M. Mendez, A. Ianiro, B. R. Noack, and S. L. Brunton, *Data-Driven Fluid Mechanics: Combining First Principles and Machine Learning* (Cambridge University Press, 2023).
[161]S. Qin, S. Wang, J. Rabault, and G. Sun, "An application of data driven reward of deep reinforcement learning by dynamic mode decomposition in active flow control," arXiv:2106.06176 (2021).
[162]R. Castellanos, I. de La Fuente, G. Cornejo Maceda, B. Noack, A. Ianiro, and S. Discetti, "Machine learning flow control in the few sensors limit," in *Proceedings of the APS Division of Fluid Dynamics Meeting Abstracts*, APS Meeting Abstracts (APS, 2021), p. H23.008.
[163]M. Gazzola, A. Tchieu, D. Alexeev, A. de Brauer, and P. Koumoutsakos, "Learning to school in the presence of hydrodynamic interactions," J. Fluid Mech. **789**, 726–749 (2016).
[164]G. Novati, S. Verma, D. Alexeev, D. Rossinelli, W. van Rees, and P. Koumoutsakos, "Synchronisation through learning for two self-propelled swimmers," Bioinspiration Biomimetics **12**, 036001 (2017).
[165]S. Verma, G. Novati, and P. Koumoutsakos, "Efficient collective swimming by harnessing vortices through deep reinforcement learning," Proc. Natl. Acad. Sci. **115**, 5849–5854 (2018).
[166]R. W. Whittlesey, S. Liska, and J. O. Dabiri, "Fish schooling as a basis for vertical axis wind turbine farm design," Bioinspiration Biomimetics **5**, 035005 (2010).
[167]J. Rabault, F. Ren, W. Zhang, H. Tang, and H. Xu, "Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization," J. Hydrodyn. **32**, 234–246 (2020).
[168]R. Vinuesa and S. Brunton, "Enhancing computational fluid dynamics with machine learning," Nat. Comput. Sci. **2**, 358–366 (2022).
[169]S. L. Brunton, B. R. Noack, and P. Koumoutsakos, "Machine learning for fluid mechanics," Annu. Rev. Fluid Mech. **52**, 477–508 (2020).
[170]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," J. Hydrodyn. **32**, 254–258 (2020).
[171]Y. Jiao, F. Ling, S. Heydari, N. Heess, J. Merel, and E. Kanso, "Learning to swim in potential flow," Phys. Rev. Fluids **6**, 050505 (2021).
[172]X. Yan, J. Zhu, M. Kuang, and X. Wang, "Aerodynamic shape optimization using a novel optimizer based on machine learning techniques," Aerosp. Sci. Technol. **86**, 826–835 (2019).
[173]K. Yonekura and H. Hattori, "Framework for design optimization using deep reinforcement learning," Struct. Multidiscip. Optim. **60**, 1709–1713 (2019).
[174]D. Xu and M. Zhang, "Reinforcement-learning-based control of convectively unstable flows," J. Fluid Mech. **954**, A37 (2023).
[175]M. Schäfer, S. Turek, F. Durst, E. Krause, and R. Rannacher, "Benchmark computations of laminar flow around a cylinder," in *Flow Simulation with High-Performance Computers II: DFG Priority Research Programme Results 1993–1995*, edited by E. H. Hirschel (Vieweg+Teubner Verlag, 1996), pp. 547–566.
[176]S. Shimomura, S. Sekimoto, A. Oyama, K. Fujii, and H. Nishida, "Closed-loop flow separation control using the deep q network over airfoil," AIAA J. **58**, 1–11 (2020).
[177]H. Xu, J.-E. W. Lombard, and S. J. Sherwin, "Influence of localised smooth steps on the instability of a boundary layer," J. Fluid Mech. **817**, 138–170 (2017).
[178]H. Xu, S. M. Mughal, E. R. Gowree, C. J. Atkin, and S. J. Sherwin, "Destabilisation and modification of Tollmien–Schlichting disturbances by a three-dimensional surface indentation," J. Fluid Mech. **819**, 592–620 (2017).
[179]J. Yu, D. Fan, B. R. Noack, and Y. Zhou, "Genetic-algorithm-based artificial intelligence control of a turbulent boundary layer," Acta Mech. Sin. **37**, 1739–1747 (2021).
[180]K. Zeng and M. Graham, "Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics," Phys. Rev. E **104**, 014210 (2021).
[181]S. Ravanbakhsh, J. Schneider, and B. Póczos, "Equivariance through parameter-sharing," in *Proceedings of the 34th International Conference on Machine Learning*, edited by D. Precup and Y. W. Teh (PMLR, 2017), Vol. 70, pp. 2892–2901.
[182]A. Sannai, Y. Takai, and M. Cordonnier, "Universal approximations of permutation invariant/equivariant functions by deep neural networks," arXiv:1903.01939 (2019).
[183]M. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," J. Mach. Learn. Res. **10**, 1633–1685 (2009).
[184]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," Phys. Fluids **34**, 081801 (2022).
[185]G. Y. Cornejo Maceda, F. Lusseyran, and B. R. Noack, "xMLC—A toolkit for machine learning control," arXiv:2208.13172 (2022).