Distributed deep reinforcement learning for simulation control

MACHINE
LEARNING
Science and Technology

OPEN ACCESS

RECEIVED
25 September 2020

REVISED
7 December 2020

ACCEPTED FOR PUBLICATION
12 January 2021

PUBLISHED
14 April 2021

PAPER

Distributed deep reinforcement learning for simulation control

Suraj Pawar[1,3] and Romit Maulik[2]

[1] School of Mechanical & Aerospace Engineering, Oklahoma State University, Stillwater, OK 74078, United States of America
[2] Argonne Leadership Computing Facility, Argonne National Laboratory, Lemont, IL 60439, United States of America
[3] This work was performed during a Research Aide appointment at the Argonne Leadership Computing Facility, Argonne National Laboratory

E-mail: rmaulik@anl.gov

Keywords: reinforcement learning, scientific machine learning, scientific computing

Abstract

Several applications in the scientific simulation of physical systems can be formulated as control/optimization problems. The computational models for such systems generally contain hyperparameters, which control solution fidelity and computational expense. The tuning of these parameters is non-trivial and the general approach is to manually 'spot-check' for good combinations. This is because optimal hyperparameter configuration search becomes intractable when the parameter space is large and when they may vary dynamically. To address this issue, we present a framework based on deep reinforcement learning (RL) to train a deep neural network agent that controls a model solve by varying parameters dynamically. First, we validate our RL framework for the problem of controlling chaos in chaotic systems by dynamically changing the parameters of the system. Subsequently, we illustrate the capabilities of our framework for accelerating the convergence of a steady-state computational fluid dynamics solver by automatically adjusting the relaxation factors of the discretized Navier–Stokes equations during run-time. The results indicate that the run-time control of the relaxation factors by the learned policy leads to a significant reduction in the number of iterations for convergence compared to the random selection of the relaxation factors. Our results point to potential benefits for learning adaptive hyperparameter learning strategies across different geometries and boundary conditions with implications for reduced computational campaign expenses[4].

1. Introduction

In recent years, there has been a proliferation in the use of machine learning (ML) for fluid mechanics problems that use the vast amount of data generated from experiments, field measurements, and large-scale high fidelity simulations [1, 2]. The vast majority of ML applications for fluid mechanics have relied on supervised learning. Examples of frameworks that utilize this paradigm include artificial neural networks (ANNs), Gaussian processes, and tree-based methods such as random forests. These methods, in particular ANNs, are popular due to their ability to approximate complex non-linear functions from large amounts of data, although this comes at an expense of reduced interpretability. Supervised learning has been adopted for a variety of tasks such as turbulence closure modeling [3], reduced order modeling [4], super-resolution and forecasting [5, 6], solving partial differential equations [7, 8], etc. However, a key limitation of supervised learning is that it relies on labeled data to learn relationships from observables to targets. Unfortunately, many practical engineering problems provide situations where labeled data is unavailable, thereby limiting the practicality of supervised ML.

In contrast, reinforcement learning (RL) is another type of ML philosophy, where an agent interacts with an environment and learns a policy that will maximize a predefined long-term reward [9]. The long-term reward is generally constructed by human-designed inductive biases and does not require any specific labeled

[4] Data and codes available at https://github.com/Romit-Maulik/PAR-RL.

data. An example of a process reward relevant to fluid mechanics can be to maximize the negative of the drag constrained to a certain lift for flow around an airfoil. The RL deployment would then attempt to maximize this reward by searching the space of airfoil configurations for specific operating conditions [10]. Multiple methods can be employed as an agent in RL, but the most successful is an ANN in conjunction with RL (commonly referred as deep RL). Deep RL is gaining widespread popularity due to its success in robotics, playing Atari games, the game of GO, and process control [11, 12]. Deep RL is also making inroads in fluid mechanics, as a means to solve fundamental problems in flow control. RL has been utilized to learn collective hydrodynamics of self-propelled swimmers to optimally achieve a specified goal [13], to train a glider to autonomously navigate in a turbulent atmosphere [14], for controlled gliding and perching of ellipsoid shaped bodies [15], and navigation of smart microswimmers in complex flows [16]. RL has also been used for closed-loop control of drag of a cylinder flow [17], flow control around bluff-bodies in experimental environments [18], control of two-dimensional convection–diffusion partial differential equation [19], designing control strategies for active flow control [10], discovering computational models from the data [20], and shape optimization [21]. RL's attractiveness stems from the ability to perform both exploitation and exploration to learn about a particular environment.

Many real-world system optimization tasks can be formulated as RL problems and offer a potential for the application of the recent advances in deep RL [22]. Several problems in scientific simulations share a similarity with system optimization problems. For example, the iterative solver in numerical simulations of fluid flows may be considered as a plant to be controlled. The iterative solver consists of different parameters such as relaxation factors, smoothing operations, type of solver, residual tolerance, etc and all these parameters could be controlled during run-time with RL to achieve faster convergence. Another example is the turbulence closure model employed in computational fluid dynamics (CFD) simulations to represent unresolved physics. The closure coefficients of the turbulence model can be controlled to provide the accurate turbulence statistics. Indeed, recently, multi-agent RL was introduced as an automated turbulence model discovery tool in the context of sub-grid scale closure modeling in large eddy simulation of homogeneous isotropic turbulence [23].

To illustrate the application of RL for the control of hyperparameters in computational models employed in scientific simulations, we consider the problem of accelerating the convergence of steady-state iterative solvers in CFD simulations by dynamically controlling relaxation factors of the discretized Navier–Stokes equations. Specifically, we train the RL agent to control relaxation factors to optimize the computational time or the number of iterations it takes for the solution to converge. This control problem differs from standard control problems, as the target state (i.e. the converged solution) is not known *a priori*. There have been some studies done for dynamically updating relaxation factors with the goal to maximize the speed of convergence. One of the methods is to use a fuzzy control algorithm based on the oscillations of the solution norm observed over a large interval prior to the current iteration [24]. Other methods based on neural network and fuzzy logic have also been proposed to automate the convergence of CFD simulations [25, 26]. Some of the limitations of the fuzzy logic methods are that they are based on certain assumptions about the system that might not always be accurate, and they do not scale well to larger or complex systems. On the other hand, the neural network agent trained using RL can discover a more accurate decision-making policy to achieve a certain objective and can be applied for complex systems. The problem of controlling relaxation factors in CFD simulations is analogous to controlling the learning rate of stochastic gradient descent (SGD) algorithm and RL has been demonstrated to achieve better convergence of SGD than human-designed learning rate [27]. In the present work, we introduce a novel method based on RL to accelerate the convergence of CFD simulations applied to turbulent flows and demonstrate the robustness of the method for the backward-facing step test case.

One of the major challenges in using RL for scientific simulations is the computational cost of simulating an environment [13, 28]. For example, the cost of CFD solver depends on several factors such as mesh refinement, complexity of the problem, level of required convergence, and can range between the order of minutes to order of several hours. RL is known to be less sample efficient and most popular RL strategies require a large number of interactions with the environment. Therefore, the RL framework should take the computational overhead of simulating an environment into account for scientific computing problems. One of the approaches to accelerate RL training is to run multiple environments concurrently on different processors and then collect their experience to train the agent. We refer to this multi-environment approach as distributed RL in this study. There are many open-source packages such as RLLib [29], Tensorforce [30], SEED RL [31], Acme [32] that can be exploited to achieve faster training with distributed RL. In addition to the distributed execution of the environment, the CFD simulations can also be parallelized to run on multiple processing elements. The list of contributions of this work can be summarized as follows:

- A distributed deep RL framework is presented for the optimization and control of computational system parameters encountered in scientific simulations.
- We apply the deep RL framework to the task of restoring chaos in transiently chaotic systems, and for accelerating the convergence of a steady-state CFD solver.
- We highlight the importance of distributed RL and how it aids in accelerated training for scientific applications.
- We also provide an open-source code and set of tutorials for deploying the introduced framework on distributed multiprocessing systems.

This paper is organized as follows. Section 2 provides background information on RL problem formulation and present proximal policy optimization (PPO) algorithm [33]. In section 3, we validate our framework using the test case of restoring chaos in chaotic systems. Then, we describe how controlling relaxation factors in CFD simulations converted to sequential decision-making problem and discuss the performance of an RL agent to generalize for different boundary conditions and different geometry setup. Finally, we conclude with the summary of our study and outline the future research directions in section 4.

2. Reinforcement learning

In this section, we discuss the formulation of the RL problem and briefly describe the PPO algorithm [33] that belongs to a class of policy-gradient methods. In RL, the agent observes the state of the environment and then based on these observations takes an action. The objective of the agent is the maximization of the expected value of the cumulative sum of the reward. This problem statement can be framed as a Markov decision process (MDP). At each time step t, the agent observes some representation of the state of the system, $s_t \in \mathcal{S}$, and based on this observation selects an action, $a_t \in \mathcal{A}$. As a consequence of the action, the agent receives the reward, $r_t \in \mathcal{R}$ and the environment enters in a new state s_{t+1}. Therefore, the interaction of an agent with the environment gives rise to a trajectory as shown below:

$$\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \dots\}. \tag{1}$$

Figure 1 displays the schematic of deep RL framework for the chaotic process control example where the Lorenz system is utilized as an environment. The advantage of the MDP framework is that it is flexible and can be applied to different problems in different ways. For example, the time steps in the MDP formulation need not refer to the fixed time step as used in CFD simulations; they can refer to arbitrary successive stages of taking the action. The goal of the RL agent is to choose an action that will maximize the expected discounted return over the trajectory τ and mathematically, it can be written as

$$R(\tau) = \sum_{t=0}^{T} \gamma^t r_t, \tag{2}$$

where γ is a parameter called a discount rate and it lies between $[0, 1]$, and T is the horizon of the trajectory. The discount rate determines how much importance to be given to the long term reward compared to immediate reward. If $\gamma \to 0$, then the agent is concerned with maximizing immediate rewards and as $\gamma \to 1$, the agent takes future reward into account more strongly. The RL tasks are also classified based on whether they terminate or not. Some tasks involve agent-environment interactions that break into a sequence of episodes, where each episode ends in a state called the terminal state. On the other hand, there are tasks, such as process control, that that goes on continually without limit. The horizon T of the trajectory for these continuing tasks goes to infinity in equation (2).

In RL, the agent's decision making procedure is characterized by a policy $\pi(s, a) \in \Pi$. The RL agent is trained to find a policy so as to optimize the expected return when starting in the state s at time step t and is called as V-value function. We can write the V-value function as

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, \pi\right]. \tag{3}$$

Similarly, the expected return starting in a state s, taking an action a, and thereafter following a policy π is called as the Q-value function and can be written as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t = s, a_t = a, \pi\right]. \tag{4}$$

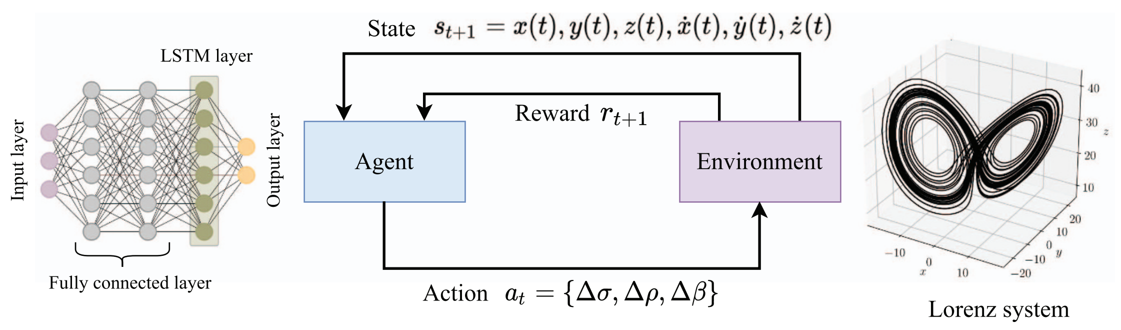
State $s_{t+1} = x(t), y(t), z(t), \dot{x}(t), \dot{y}(t), \dot{z}(t)$
LSTM layer
Input layer
Output layer
Fully connected layer
Agent
Reward r_{t+1}
Environment
Action $a_t = \{\Delta\sigma, \Delta\rho, \Delta\beta\}$
Lorenz system

Figure 1. Schematic of the deep RL framework for the chaotic process control example. The RL agent observes solution vector of the Lorenz system and its time-derivative as the state of the system and selects perturbation to the control parameters as the action. The agent either receives positive or negative reward based on the magnitude of the velocity (equations (16) and (17)). The neural network architecture is only for the purpose of representation.

We also define an advantage function that can be considered as an another version of Q-value function with lower variance by taking the V-value function as the baseline. The advantage function can be written as

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{5}$$

In deep RL, the neural network is used as an RL agent, and therefore, the weights and biases of the neural network are the parameters of the policy [34]. We use $\pi_w(\cdot)$ to denote that the policy is parameterized by $w \in \mathbb{R}^d$. The agent is trained with an objective function defined as [9]

$$J(w) \doteq V^{\pi_w}(s_0), \tag{6}$$

where an episode starts in some particular state s_0, and V^{π_w} is the value function for the policy π_w. The policy parameters w are learned by estimating the gradient of an objective function and plugging it into a gradient ascent algorithm as follows:

$$w \leftarrow w + \alpha \nabla_w J(w), \tag{7}$$

where α is the learning rate of the optimization algorithm. The gradient of an objective function can be computed using the policy gradient theorem [34] as follows:

$$\nabla_w V^{\pi_w}(s_0) = \mathbb{E}_{\pi_w}\left[\nabla_w\big(\log\, \pi_w(s,a)\big) Q^{\pi_w}(s,a)\right]. \tag{8}$$

There are two main challenges in using the above empirical expectation. The first one is the large number of samples required and the second is the difficulty of obtaining stable and steady improvement. There are different families of policy-gradient algorithms that are proposed to tackle these challenges [35–37]. The performance of policy gradient methods is highly sensitive to the learning rate α in equation (7). If the learning rate is large it can cause the training to be unstable. The PPO algorithm uses a clipped surrogate objective function to avoid excessive update in policy parameters in a simplified way [33]. The clipped objective function of the PPO algorithm is

$$J^{\mathrm{clip}}(w) = \mathbb{E}\big[\min(r_t(w)A^{\pi_w}(s,a), \mathrm{clip}(r_t(w), 1-\epsilon, 1+\epsilon)A^{\pi_w}(s,a))\big], \tag{9}$$

where $r_t(w)$ denotes the probability ratio between new and old policies as follows:

$$r_t(w) = \frac{\pi_{w+\Delta w}(s,a)}{\pi_w(s,a)}. \tag{10}$$

The ε in equation (9) is a hyperparameter that controls how much new policy is allowed to be deviated from the old. This is done using the function $\mathrm{clip}(r_t(w), 1-\varepsilon, 1+\varepsilon)$ that enforces the ratio between new and old policy $(r_t(w))$ to stay between the limit $[1-\varepsilon, 1+\varepsilon]$.

3. Numerical experiments

In this section, we demonstrate the application of deep RL for two problems where the parameters of the systems need to be controlled to achieve a certain objective. The first test case is restoring chaos is chaotic dynamical systems [38]. We utilize this test case as a validation example for our distributed deep RL framework. The second test problem is accelerating the convergence of steady-state CFD solver through deep RL and we illustrate this for turbulent flow over a backward-facing step example.

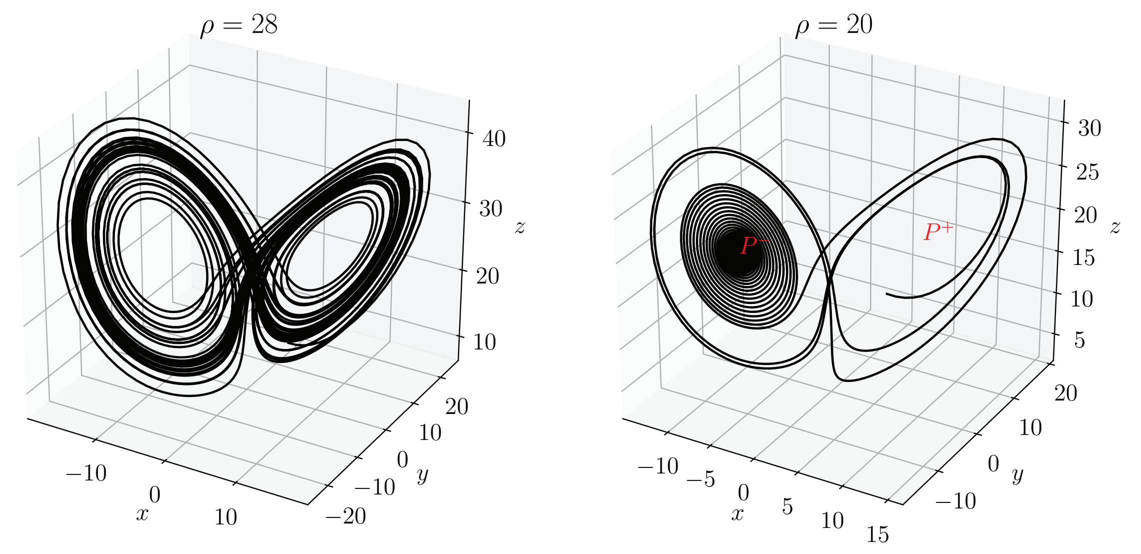

Figure 2. Evolution trajectory of the Lorenz system with $\rho = 28$ (left) and $\rho = 20$ (right).

3.1. Chaotic process control

The chaotic state is essential in many dynamical systems and the phenomenon called crisis can cause sudden destruction of chaotic attractors [39]. This can be negative in many applications where the chaos is essential such as to enhance mixing by chaotic advection [40] or to prevent the collapse of electric power systems [41]. Some of the methods to sustain chaos is to use small perturbations based on knowledge of dynamical systems and *a priori* information about the escape regions from chaos [42]. Vashishtha *et al* [38] proposed an approach using deep RL to determine small perturbations to control parameters of the Lorenz system [43] in such a way that the chaotic dynamics is sustained despite the uncontrolled dynamics being transiently chaotic. This method does not require finding escape regions, target points, and therefore attractive for systems where the complete information about the dynamical system is unknown.

The Lorenz system is described by following equations:

$$\frac{dx}{dt} = \sigma(y - x), \tag{11}$$

$$\frac{dy}{dt} = x(\rho - z) - y, \tag{12}$$

$$\frac{dz}{dt} = xy - \beta z, \tag{13}$$

where x, y, z are the state of the Lorenz system, and σ, ρ, β are the system's parameter. The Lorenz system give rise to chaotic trajectory with $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$ as shown in figure 2. However, the Lorenz system exhibits transient chaotic behavior for $\rho \in [13.93, 24.06]$ [44]. For example, if we use $\rho = 20$ for the Lorenz system, the solution converges to a specific fixed point $P^- = (-7.12, -7.12, 19)$ as illustrated in figure 2.

In order to sustain the chaos in transiently chaotic Lorenz system, we utilize the similar RL problem formulation as Vashishtha *et al* [38]. The RL agent observes the solution vector and its time-derivative as the state of the system. Therefore, we have

$$s_t = x(t),\ y(t),\ z(t),\ \dot{x}(t),\ \dot{y}(t),\ \dot{z}(t), \tag{14}$$

where the dot represent the time-derivative, i.e. velocity. Based on this observations, the agent choose an action which is the perturbation to the control parameters of the Lorenz system as given below:

$$a_t = \{\Delta\sigma, \Delta\rho, \Delta\beta\}. \tag{15}$$

The perturbation to the control parameters is restricted to lie within a certain interval which is $\pm 10\%$ of its values. Therefore, we have $\Delta\sigma \in [-\sigma/10, \sigma/10]$, $\Delta\rho \in [-\rho/10, \rho/10]$, and $\Delta\beta \in [-\beta/10, \beta/10]$. We note here that these limits remain fixed throughout the episode based on the initial values of (σ, ρ, β). For a

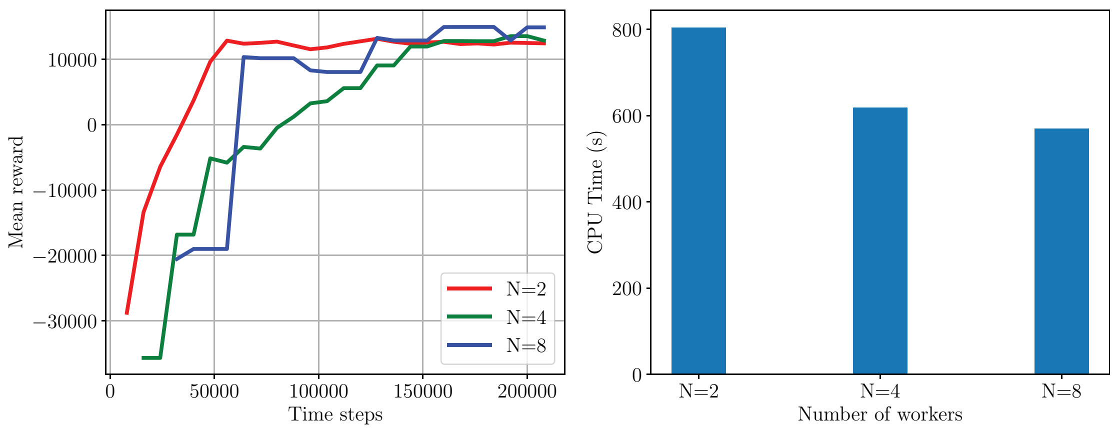

Figure 3. Evolution of the moving average mean reward for different number of workers trained with the PPO algorithm (left). The moving average window size is set at five episodes. The CPU time required for training an agent with different number of workers (right).

transiently chaotic system, the trajectory converges to a fixed point with time and therefore the magnitude of velocity will decrease consistently with time. This fact is utilized to design the reward function. When the magnitude of velocity is greater than certain threshold velocity V_{Th}, the agent is rewarded and the agent is penalized when the magnitude of velocity is less than V_{Th}. In addition to assigning reward for each time step, a terminal reward is given at the end of each episode. The reward function can be written as

$$r_t = \begin{cases} 10 & \text{for } V(t) > V_{\mathrm{Th}} \\ -10 & \text{for } V(t) \leq V_{\mathrm{Th}} \end{cases} \tag{16}$$

$$r_{\mathrm{terminal}} = \begin{cases} -100, & \text{for } \overline{r_t} \leq -2 \\ 0, & \overline{r_t} > -2 \end{cases} \tag{17}$$

where $\overline{r_t}$ is the average of stepwise reward over the last 2000 time steps of an episode. The value of the threshold velocity V_{Th} is set at 40.

For training the agent, we divide each episode into 4000 time steps with size $dt = 2 \times 10^{-2}$. We train the agent for 2×10^5 time steps which corresponds to 50 episodes. The RL agent is a fully connected neural network with two hidden layers and 64 neurons in each layer. Additionally, the output of the second hidden layer is further processed by the LSTM layer composed of 256 cells. The hyperbolic tangent (tanh) activation function is used for all hidden layers. We utilize the same architecture as the one used in the original study [38]. From the point of view of optimal agent architecture selection, there are methods like population based training that have been applied for optimizing the parameters of deep RL and hyperparameters of the agent [45]. However, considering the episode hungry nature of RL and computational expense of the environment, the hyperparameter search of the agent in deep RL can quickly become computationally intractable.

Figure 3 displays how the mean reward is progressing with training for a different number of workers. We note here that by a different number of workers, we mean that the agent-environment interactions are run concurrently on different processors. In distributed deep RL, the environment is simulated on different processors and their experience is collected. These sample batches encode one or more fragments of a trajectory and are concatenated into a larger batch that is then utilized to train the neural network. More concretely, in this particular example, we collect the experience of an agent for 8000 time steps to train the neural network. When we employ two workers, the agent-environment interaction is simulated for 4000 time steps on each worker and then this experience is concatenated to form a training batch for the neural network. Similarly, for four workers, the agent-environment interaction is simulated for 2000 time steps on each worker to collect a training batch size of 8000 time steps. Figure 3 also reports the computational time required for training. As we increase the number of workers from two to four, the computational speed is approximately reduced by 25%. However, an increase in the number of workers to eight leads to a marginal improvement in terms of CPU time. This is not surprising, as the communication frequency is increased with eight workers and hence there is no significant reduction in CPU time.

Figure 4 depicts the performance of the trained RL agent in sustaining chaos in a transiently chaotic Lorenz system. In figure 4, the black trajectory is for uncontrolled solution of the Lorenz system for parameters $(\sigma, \rho, \beta) = (10, 20, 8/3)$. When the policy learned by an RL agent is used for the control, the

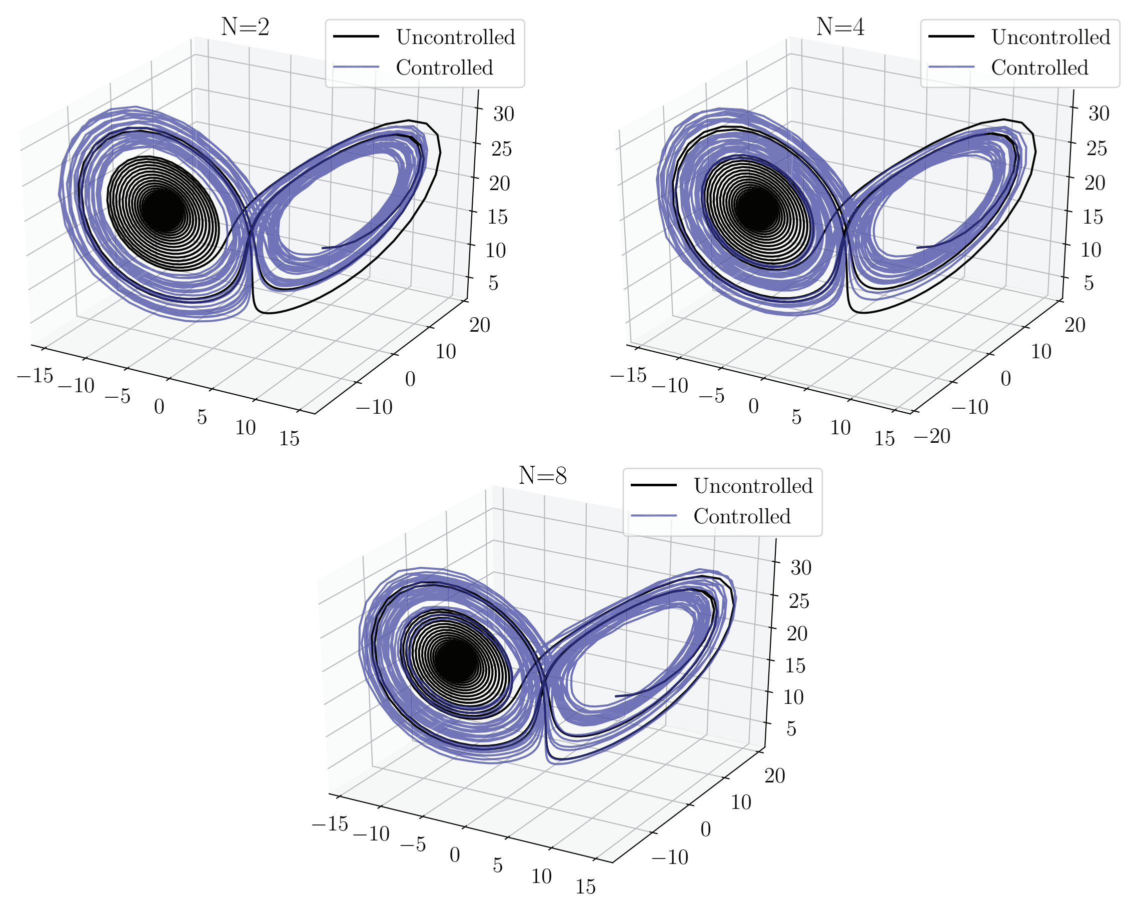

Figure 4. Comparison of the evolution of the Lorenz system without applying the control (black) and with an application of the control (blue) policy learned by an RL agent trained using different number of workers (i.e. $N = 2, 4, 8$). The parameters of the Lorenz system are $(\sigma, \beta, \rho) = (10, 8/3, 20)$ and the initial condition is $(x, y, z) = (1, 12, 9)$.

Lorenz system does not converge to a fixed point of P^-. We can also see that the RL agent trained with a different number of workers can learn the policy effectively and is able to sustain chaos over a period of temporal integration. This observation is very important for employing RL in physical systems where the simulation of a computationally intensive environment is the bottleneck and distributed deep RL can be employed to address this issue.

3.2. Solver convergence acceleration

In CFD, the Navier–Stokes equations are solved by discretizing them which leads to a system of linear equations. This system of linear equations is usually solved using iterative techniques like Gauss–Seidel methods, Krylov subspace methods, multigrid methods, etc [46]. The main logic behind iterative methods is to start with some approximate solution and then reduce the residual vector at each iteration, called relaxation steps. The relaxation steps are performed until the convergence criteria are reached. In the relaxation step, the value of the variable at the next time step is obtained by adding some fraction of the difference between the current value and predicted value to the value of the variable at the current time step. The SIMPLE algorithm [47] is widely used in CFD and it utilizes the underrelaxation method to update the solution from one iteration to another. The underrelaxation methods improve the convergence by slowing down the update to the solution field. The value of the variable at the kth iteration is obtained as the linear interpolation between the value from the previous iteration and the value obtained in the current iteration as follows:

$$x_P^{(k)} = x_P^{(k-1)} + \alpha\left(\frac{\sum a_{\mathrm{nb}} x_{\mathrm{nb}} + b}{a_P} - x_P^{(k-1)}\right), \tag{18}$$

where $0 < \alpha < 1$ is the underrelaxation factor, $x_P^{(k)}$ is the value of the variable at node P to be used for the next iteration, $x_P^{(k-1)}$ is the value of the variable at node P from the previous iteration, x_{nb} is the values of the variables at the neighboring nodes, and $a_P,\ a_{\mathrm{nb}},\ b$ are the constants obtained by discretizing Navier–Stokes equations. The underrelaxation factor α in equation (18) should be chosen in such a way that it is small

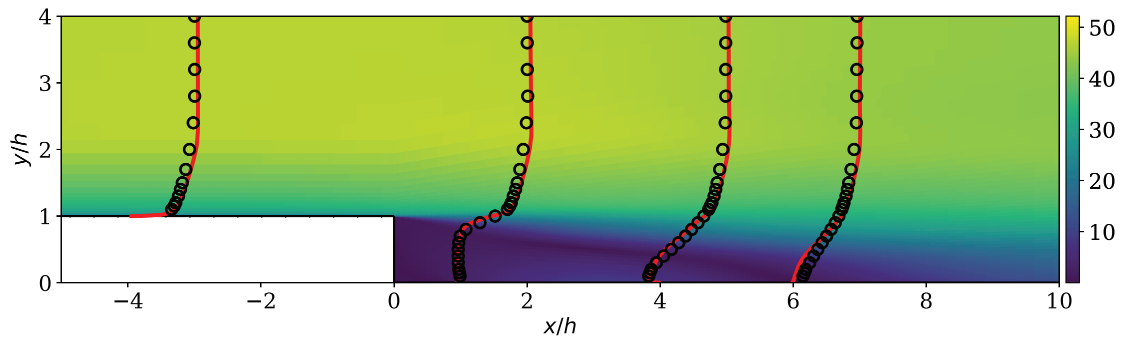

Figure 5. Validation of the CFD model with experimental measurements for normalized velocity in the x-direction at four different axial locations. The experimental data for velocity measurements are published in [48]. The contours indicate the velocity magnitude of the flow field.

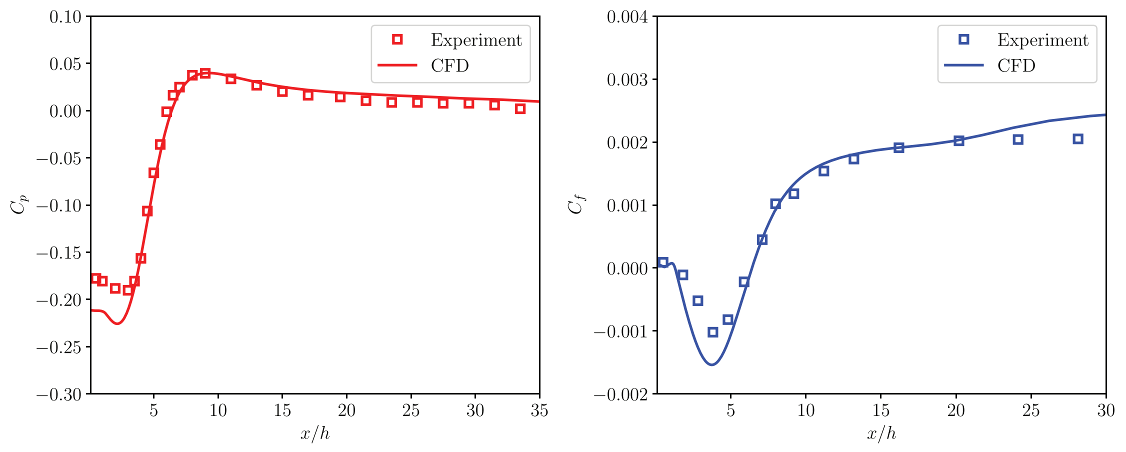

Figure 6. Comparison of the pressure coefficient (left) and skin-friction coefficient (right) at the lower wall predicted by CFD simulation and experimental measurements. The experimental data for these coefficients are published in [48].

enough to ensure stable computation and large enough to move the iterative process forward quickly. In practice, a constant value of the relaxation factor is employed throughout the computation and this value is usually decided in an ad-hoc manner. Also, the suitable value of the relaxation factor is problem-dependent and a small change in it can result in a large difference in the number of iterations needed to obtain the converged solution. In the following we attempt to obtain a dynamic adaptation of the underrelaxation factor using deep RL.

We choose a CFD model for a two-dimensional canonical backward facing step that is commonly used for solver and turbulence model validation. The height of the step is $H = 0.0127$ m and the free-stream reference velocity (flowing left to right) is $U_{\text{ref}} = 44.2$ m s^{-1}. The corresponding Reynolds number based on step height and reference free-stream velocity is approximately $\text{Re}_H = 36\,000$. The turbulent intensity at the inlet is $I_t = 6.1 \times 10^{-4}$ and the eddy-viscosity-to-molecular-viscosity ratio at the inlet is $\mu_t/\mu = 0.009$. First, we validate a CFD model corresponding to the aforementioned geometry and flow conditions using the experimental data provided in [48]. We use the computational mesh distributed by NASA [49] for our study. We utilize the steady-state simpleFoam solver in OpenFoam [50] and apply the $k - \omega$ SST turbulent model [51] with the standard values of closure coefficients to simulate the turbulent flow. Figure 5 displays the contour plot of the magnitude of the velocity along with the normalized stream-wise velocity at four different axial locations obtained experimentally and numerically. We can observe that there is a very good match between the experimental results and numerical results for normalized stream-wise velocity. As shown in figure 6, we can also see that there is a very good agreement between the experimental measurements and CFD prediction for the pressure coefficient and skin-friction coefficient on the lower wall.

Subsequently, we perform a manual analysis of the convergence characteristics of the CFD model for various underrelaxation factors. In figure 7, the number of iterations required for convergence with different values of relaxation factors for momentum and pressure equation is shown for the problem investigated in this study (i.e. the backward-facing step example). Typical residual histories for different variables that are solved in the simulation of the backward-facing step example are also displayed in figure 7. We perform a

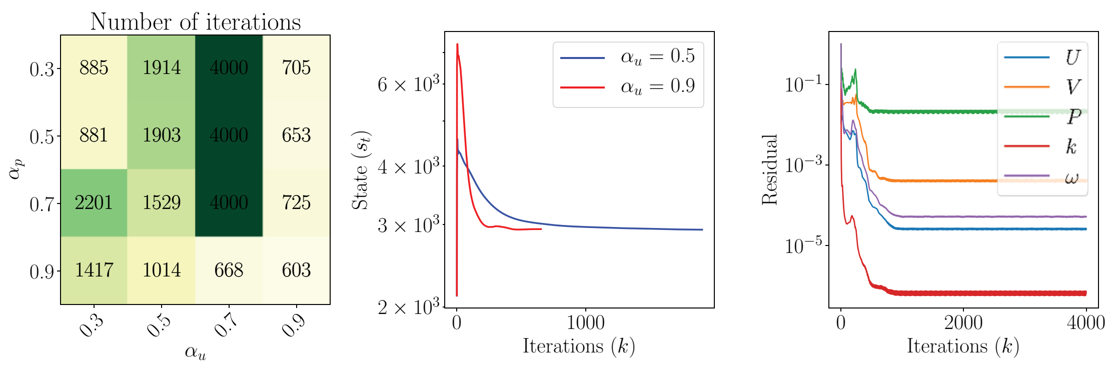

Figure 7. Number of iterations required for the convergence with different combinations of relaxation factor for momentum and pressure equation (left). The convergence of the state of an environment with iterations for two values of momentum equation relaxation factor (middle). The pressure relaxation factor is set at $\alpha_p = 0.5$ for both cases. Typical residual history for momentum equation, pressure equation, and turbulent quantities equation (right) for the backward facing step test case.

trial simulation to assess convergence by running a simulation for 4000 iterations with $\alpha_u = 0.9$ and $\alpha_p = 0.9$. Based on this residual history, we can assume that the convergence is reached once the residual for momentum equation and turbulent quantities equation falls below 5×10^{-4}, and the residual for pressure equation is reduced below 5×10^{-2}. It can be noticed that the number of iterations is less for $\alpha_u = 0.9$ (momentum equation relaxation factor), for all values of α_p (pressure equation relaxation factor). However, for $\alpha_u = 0.7$, the solution converges only for $\alpha_p = 0.9$, and for other values of α_p the solution does not converge within 4000 iterations. Even though for most of the CFD simulations, a good value for relaxation factors can be found through trial and error, the process can become intractable for large parameter space. The complexity increases even further when the relaxation factor needs to be updated on the fly during run-time for multiphysics problems. Therefore, we attempt to automate the procedure of dynamic update of the relaxation factors using deep RL to accelerate the convergence of CFD simulations.

To learn a policy that is generalizable for different inlet boundary condition, we train the RL agent for multiple values of inlet velocities. For each episode, the inlet velocity is selected randomly from the interval between [35,65] m s^{-1}. We reiterate here that, in our study, the frequency at which the relaxation factors are updated is different from the frequency of CFD iterations. In particular, we update the relaxation factor for the momentum and pressure equation after every 100 iterations of the CFD simulation. One of the advantages of supervised learning methods such as physics-informed neural network is that it allows us to embed the knowledge of physics explicitly in the learning process [7, 52]. In contrast to supervised learning, the only opportunity for physics representation in the RL paradigm is the choice of the state, where the user has to design an appropriate quantity representative of the physics that balances expressiveness and conciseness. Therefore, the state of the system should be appropriately designed based on the physics of the problem, and the maximum information is made available to the RL agent while being simple in construction. We use the average value of the square of the velocity magnitude as the characteristic quantity that represents the CFD solution. Thus, the state of the environment can be written as

$$s_t = \frac{1}{N_b}\sum_{i=1}^{N_b}[U_b(i)]^2 + \frac{1}{N_m}\sum_{i=1}^{N_m}[U_m(i)]^2, \tag{19}$$

where U_b is the velocity at the inlet boundary, U_m is the nodal value of the velocity at internal mesh, N_b is the total number of nodes at inlet boundary, and N_m is the total number of nodes in the internal mesh. At the start of an episode, the velocity in the interior of the domain is initialized with zero velocity, and hence the initial state for each episode will depend only on the value of the velocity at the inlet boundary. Figure 7 displays how the state of the environment progress toward the converged state for two different values of relaxation factor for momentum equations. As the solution starts converging, the change in the state of the system from one iteration to other will also be reduced. We note here that some other quantity based on the pressure as the characteristic quantity can also be utilized for the state of the system. Indeed, one can also use the iterative oscillations of the characteristic quantity observed over a large interval of the consecutive iterations before the current iteration as the state of the system. The information about fluctuating quantities may improve the effectiveness of the deep RL approach and we plan to consider them in our future studies.

Based on the observation of the state of the system, the agent decides the relaxation factor for the momentum and pressure equations. The action of the agent can be written as

$$a_t = \{\alpha_u, \alpha_p\}, \tag{20}$$

where α_u and α_p are the relaxation factors for momentum and pressure equation, respectively. The action space for both relaxation factors lies within the interval $[0.2, 0.95]$. Even though the action space is composed of only two relaxation factors, this problem is complicated due to its dynamic nature and simple exploratory computation will not be feasible. For more complex multiphysics problems, the action space can be easily augmented in this framework to include relaxation factors for other governing equations. The goal of the RL agent is to minimize the number of iterations required for obtaining the converged solution. We give the total number of iterations of the CFD simulation between two time steps as the reward for that time step. Accordingly, the reward at each time step can be written as

$$r_t = -(K_t - K_{t-1}), \tag{21}$$

where K is the iteration count of CFD simulation. Since each episode comes to end with the terminal state, the task of accelerating CFD simulations can be considered as an episodic task. Therefore, we set the discount factor $\gamma = 1.0$ for this test case. The stepwise reward function computed using equation (21) and with $\gamma = 1.0$ will give cumulative reward equal to the total number of iterations it took for CFD simulation to converge. Specifically, if the CFD simulation required 450 iterations to converge and the relaxation factor is updated every 100 iterations, then the trajectory for the reward will look like $\{-100, -100, -100, -100, -50\}$.

For this example, the RL agent is a fully connected neural network with 64 neurons in each hidden layer. The RL agent is trained for 2000 episodes and the learning rate of an optimizer is set to 2.5×10^{-4}. For training an RL agent, we assume that the convergence for CFD simulation is reached once the residual for momentum equation and turbulent quantities falls below 1×10^{-3} and the residual for pressure equation is reduced below 5×10^{-2}. We highlight here that during the testing phase of an agent, the CFD simulation is run till the residual for momentum equation and turbulent quantities drop below 5×10^{-4}. The use of the slightly mild criteria for the convergence of the CFD simulation during training allows us to train an agent in a computationally efficient manner. Indeed in practical CFD simulations, it is possible to ease on underrelaxation factors as we reach higher time steps to accelerate the convergence of CFD simulation.

Figure 8 shows how the moving average mean reward progress with training for a different number of workers. The moving average window size is set at 100 episodes, i.e. the average of the reward is taken over the last 100 episodes. At the beginning of the training, the agent is dominantly exploring and the mean reward at the start of training is in the range of 650–700 iterations for CFD simulation to converge. The mean reward improves over the training and the final mean reward is around 450 iterations for the convergence of CFD simulations. This corresponds to approximately 30% improvement in the number of iterations required for convergence. We highlight here that, for many CFD problems, suitable values of relaxation factors can be chosen to achieve faster convergence based on prior experience. However, for complex systems, the process of learning a rule to dynamically update relaxation factors through exploratory computations can become unmanageable. For such systems, the proposed RL framework can be attractive to achieve acceleration in the convergence. In figure 8, the CPU time for training is reported for a different number of workers. As we increase the number of workers/environments from 4 to 8, we get around 40% improvement in the CPU time for the training. The improvement in CPU speed is only marginal as the number of processor is increased to 16, which might be due to increased communication between the processors. We note here that our environment, i.e. the CFD simulation utilizes a single core as the backward-facing step is a relatively small problem and can be simulated within an order of seconds or minutes depending upon the computer architecture. For high-dimensional problems, the parallelization through domain-decomposition can also be exploited to reduce the CPU time of each CFD simulation run.

Once the agent is trained, we test the agent's performance for three different values of inlet velocity, $V = 25.0,\ 50.0,\ 75.0$. Among these three velocities, the velocity $V = 50.0$ lies within the training velocity range, i.e. [35.0, 65.0]. We run ten different samples of CFD simulation for these three inlet velocities, and utilize the RL agent to select the relaxation factor after every 100 iterations. Figure 9 shows the box plot to depict the data obtained from ten samples through their quartiles. It should be noted that the policy learned by an agent is not deterministic, and hence the number iterations required for convergence for a single inlet velocity is not constant for all samples. The mean and standard deviation of these samples are also shown in figure 9. From both these plots, we can see that the mean of the samples for all three different velocities is around 500 iterations. Therefore, we can conclude that the policy learned by the RL is generalizable to inlet boundary condition different than the training boundary condition. We plot the variation of relaxation factor for momentum equation and pressure equation in figure 10 to understand the policy learned by an RL agent. The RL agent has learned to keep the relaxation factor for momentum equation at its highest value (i.e. 0.95) throughout the CFD simulation. We do not see any specific pattern for the variation of pressure

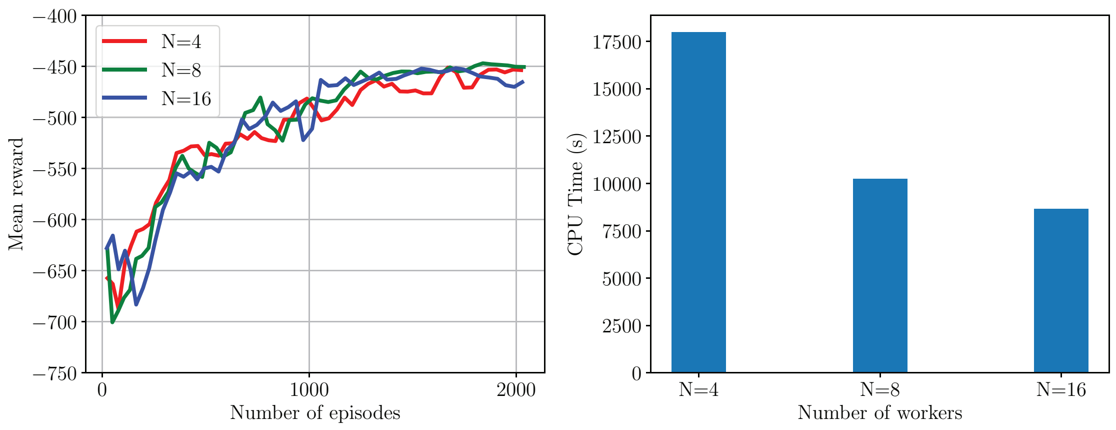

Figure 8. Evolution of the moving average mean reward for the PPO algorithm trained using different number of workers (left). The moving average window size is set at 100 episodes. The CPU wall time for agent's training for different number of workers (right).

equation relaxation factor. Since, the relaxation factor for momentum equations is constant, the difference in number of iterations to convergence for different samples is mainly due to how the pressure relaxation factor is decided by an RL agent at various stages of the CFD simulation. This behavior of the relaxation factors may be caused by the underlying nature of the Poisson pressure and momentum governing equations. However, there are multiple factors that engender this behavior aside from the governing equations such as the quality of the mesh and the discretization schemes used for the pressure Poisson solver and the momentum equations. At this point, we can not explicitly establish that causal link here despite the evidence of correlation.

Furthermore, we test the learned policy for a different backward-facing step geometry. The test geometry has the step height twice that of the train geometry. In figure 11, the flow feature for train and test geometry is shown, and it can be seen that there is a sufficient difference in terms of the flow in the recirculation region. We run a similar experiment with the test geometry for three different values of inlet velocity. Figure 12 displays the summary of the RL agent's performance in accelerating CFD simulation of the test geometry. Overall, the number of iterations required for the convergence for the test geometry is higher than the train geometry. The mean of the samples for test geometry is around 600 iterations for all three velocities. Interestingly, the variance of test geometry samples employed with the policy trained using 8 workers is less compared to the policy trained using four and 16 workers.

We emphasize here that the computational efficiency of the deep RL is one of the major bottlenecks compared to other methods such as PDE-constrained adjoint optimization. There have been several works done on applying adjoint-based methods to analyze the sensitivity of aerodynamic forces around bluff bodies at various Reynolds numbers [53]. Similarly, the adjoint-based optimization methods can be applied to obtain relaxation factor's sensitivity field. The deep RL algorithm may not be competitive when compared to adjoint methods which generally involve only two numerical simulations (one for the forward simulation, and one for the adjoint solution). Another shortcoming of deep RL algorithm is the *a priori* specification of several hyperparameters (for example the update frequency of relaxation factor) that may affect the accuracy significantly. While there are some empirical remedies for this for supervised learning applications through hyperparameter search strategies, RL and its episode hungry nature make this step intractable in the absence of extremely large compute resources. However, it is important to recognize that a trained RL policy, while costlier in terms of off-line model evaluations, can be deployed across different on-line environments provided a representative set of simulations was run during its training. Therefore RL based simulation control has possibilities with regard to computational cost amortization over the course of an extended numerical simulation campaign.

More generally, adjoint-based methods are very difficult to apply for the optimization of time-averaged quantities in turbulent flows (such as in LES applications). This is due to the chaotic nature of turbulence, and a small change in the initial condition can cause exponentially diverging adjoint solutions as soon as the length of the adjoint simulation exceeds the predictability time scale [54]. Therefore, the deep RL methods can be an alternative to adjoint-based methods in terms of their application.

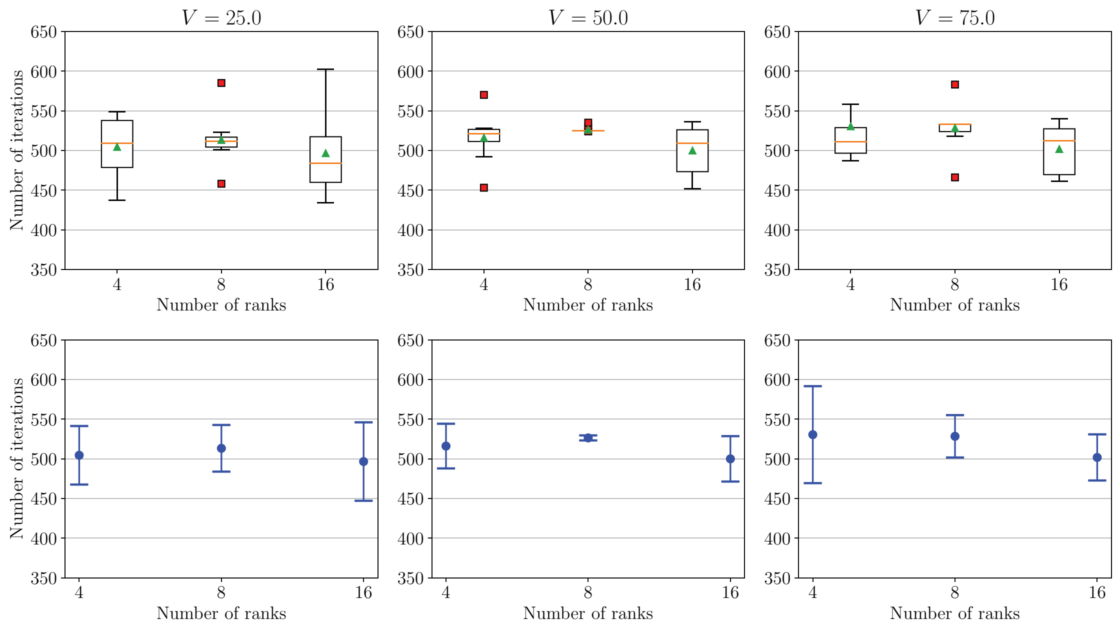

Figure 9. Evaluation of the trained agent's PPO policy for different inlet velocities—$V = 25.0$ m s^{-1} (left), $V = 50.0$ m s^{-1} (middle), $V = 75.0$ m s^{-1} (right). The top plot is the box plot in which the green marker is for the mean, red marker is for outliers, and the orange line is for the median of the data. The bottom plot shows the mean of the sample with error bars denoting the standard deviation of samples.

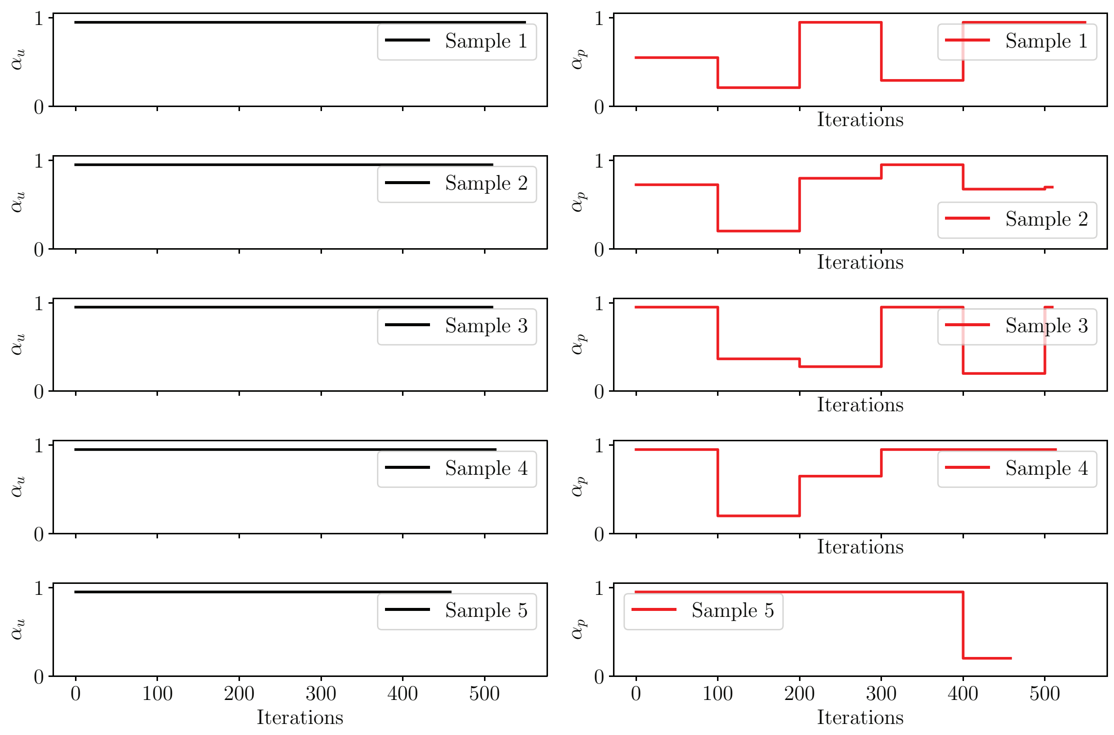

Figure 10. Variation of the relaxation factor for the momentum equation and pressure equation over iterations of the steady-state CFD solver for $V = 25.0$ m s^{-1} for five samples evaluated during testing with the learned policy.

4. Concluding remarks

In this study, we have investigated the application of distributed deep RL to automatically update the computational hyperparameters of numerical simulations, which is a common task in many scientific applications. We illustrate our framework for two problems. The first problem is tasked with sustaining chaos in chaotic systems, and the second problem is the acceleration of convergence for steady-state CFD solver. In the first example, we demonstrate that the policy learned by an RL agent trained using PPO

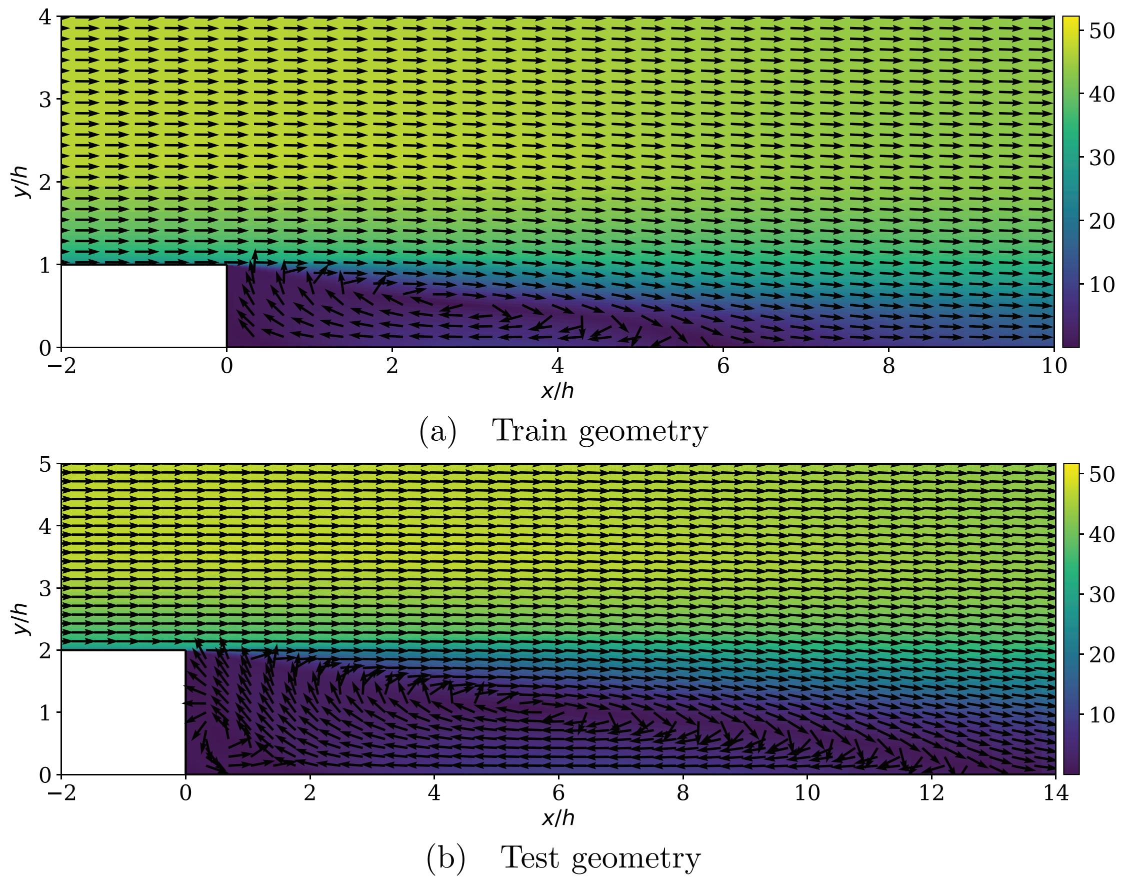

Figure 11. Flow features in the vicinity of backward facing step for train and test geometry setup. The agent is trained with a step height $h = 0.0127$ m and the test geometry has step height $h = 0.0254$ m.

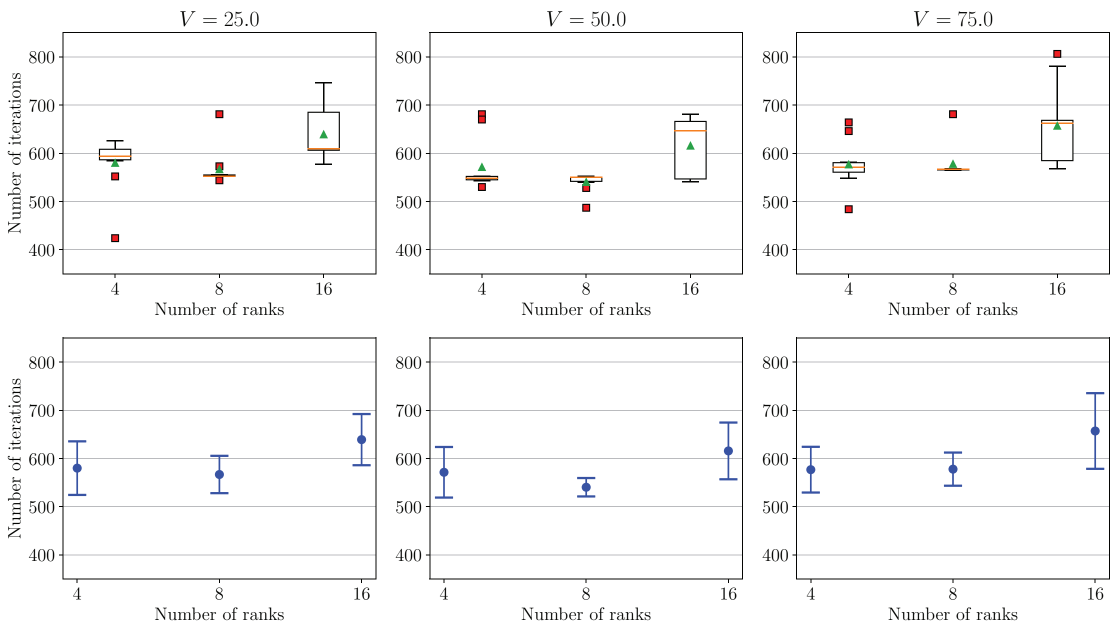

Figure 12. Evaluation of the trained agent's PPO policy for different inlet velocities in test geometry setup—$V = 25.0$ m s^{-1} (left), $V = 50.0$ m s^{-1} (middle), $V = 75.0$ m s^{-1} (right). The top plot is the box plot in which the green marker is for the mean, red marker is for outliers, and the orange line is for the median of the data. The bottom plot shows the mean of the sample with error bars denoting the standard deviation of samples.

algorithm can automatically adjust the system's parameters to maintain chaos in transiently chaotic systems. The problem of controlling relaxation factors in steady-state CFD solvers is different from the standard control problem. In a standard control problem, the target value is known in advance and this target value is used to change action variables in such a way that the state of the system is guided toward the target state. However, the problem of accelerating convergence of steady-state CFD solvers is different in a way that the target point, i.e. the converged solution is not known. Therefore, RL is a suitable algorithm for this problem to discover the decision-making strategy that will dynamically update the relaxation factor with an objective to minimize the number of iterations required for convergence. Our results of numerical experiments indicate that the learned policy leads to acceleration in the convergence of steady-state CFD solver and the learned policy is generalizable to unseen boundary conditions and different geometries.

Despite the relative simplicity in terms of the RL problem formulation and its application to backward-facing step example, this framework can be readily extended for more complex multiphysics systems. The state and the action space of an RL agent can be augmented to tackle complexities in these systems. Our future studies will revolve around efficient state construction that can optimally balance information about the environment and computational cost. This shall be crucial for large computational hyperparameter spaces in more challenging problems which will necessitate information from various flow field variables. One of the major challenges in using RL for scientific simulations is the computational time due to a slow environment, and this challenge can be addressed by utilizing distributed environment evaluation along with high-performance computing to reduce the CPU time of the solver. A thorough study of asynchronous RL techniques for large simulations in distributed environments is also important for deploying our ideas to practical CFD applications.

Acknowledgments

We acknowledge the helpful discussions with Venkat Vishwanath and the Argonne Leadership Computing Facility Datascience team. Also, we thank Omer San for his insightful comments on this manuscript. This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research was funded in part and used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. DOE or the United States Government.

Data availability statement

The data that support the findings of this study are openly available at the following URL/DOI: https://github.com/Romit-Maulik/PAR-RL.

ORCID iD

Romit Maulik https://orcid.org/0000-0001-9731-8936

References

[1] Brunton S L, Noack B R and Koumoutsakos P 2019 Machine learning for fluid mechanics *Annu. Rev. Fluid Mech.* **52** 477–508
[2] Brenner M P, Eldredge J D and Freund J B 2019 Perspective on machine learning for advancing fluid mechanics *Phys. Rev. Fluids* **4** 100501
[3] Duraisamy K, Iaccarino G and Xiao H 2019 Turbulence modeling in the age of data *Annu. Rev. Fluid Mech.* **51** 357–77
[4] Murata T, Fukami K and Fukagata K 2020 Nonlinear mode decomposition with convolutional neural networks for fluid dynamics *J. Fluid Mech.* **882** A13
[5] Jiang C M *et al* 2020 MeshfreeFlowNet: a physics-constrained deep continuous space-time super-resolution framework (arXiv: 2005.01463)
[6] Kim J and Lee C 2020 Prediction of turbulent heat transfer using convolutional neural networks *J. Fluid Mech.* **882** A18
[7] Raissi M, Perdikaris P and Karniadakis G E 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686–707
[8] Long Z, Lu Y and Dong B 2019 PDE-Net 2.0: learning PDEs from data with a numeric-symbolic hybrid deep network *J. Comput. Phys.* **399** 108925
[9] Sutton R S and Barto A G 2018 *Reinforcement Learning: An Introduction* (Cambridge, MA: MIT Press)
[10] Rabault J, Kuchta M, Jensen A, Réglade U and Cerardi N 2019 Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control *J. Fluid Mech.* **865** 281–302
[11] Silver D *et al* 2016 Mastering the game of go with deep neural networks and tree search *Nature* **529** 484–9
[12] Spielberg S P K, Gopaluni R B and Loewen P D 2017 Deep reinforcement learning approaches for process control *2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP)* (IEEE) pp 201–6

[13] Verma S, Novati G and Koumoutsakos P 2018 Efficient collective swimming by harnessing vortices through deep reinforcement learning *Proc. Natl Acad. Sci.* **115** 5849–54

[14] Reddy G, Wong-Ng J, Celani A, Sejnowski T J and Vergassola M 2018 Glider soaring via reinforcement learning in the field *Nature* **562** 236–9

[15] Novati G, Mahadevan L and Koumoutsakos P 2019 Controlled gliding and perching through deep-reinforcement-learning *Phys. Rev. Fluids* **4** 093902

[16] Colabrese S, Gustavsson K, Celani A and Biferale L 2017 Flow navigation by smart microswimmers via reinforcement learning *Phys. Rev. Lett.* **118** 158004

[17] Guéniat F, Mathelin L and Hussaini M Y 2016 A statistical learning strategy for closed-loop control of fluid flows *Theor. Comput. Fluid Dyn.* **30** 497–510

[18] Fan D, Yang L, Wang Z, Triantafyllou M S and Karniadakis G E 2020 Reinforcement learning for bluff body active flow control in experiments and simulations *Proc. Natl Acad. Sci.* **117** 26091–8

[19] Farahmand A-M, Nabi S and Nikovski D N 2017 Deep reinforcement learning for partial differential equation control *2017 American Conf. (ACC)* (IEEE) pp 3120–7

[20] Bassenne M and Lozano-Durán Aan 2019 Computational model discovery with reinforcement learning (arXiv: 2001.00008)

[21] Ghraieb H, Viquerat J, Larcher Aelien, Meliga P and Hachem E 2020 Optimization and passive flow control using single-step deep reinforcement learning (arXiv: 2006.02979)

[22] Haj-Ali A, Ahmed N K, Willke T, Gonzalez J, Asanovic K and Stoica I 2019 A view on deep reinforcement learning in system optimization (arXiv: 1908.01275)

[23] Novati G, de Laroussilhe H L and Koumoutsakos P 2020 Automating turbulence modeling by multi-agent reinforcement learning (arXiv: 2005.09023)

[24] Dragojlovic Z and Kaminski D A 2004 A fuzzy logic algorithm for acceleration of convergence in solving turbulent flow and heat transfer problems *Numer. Heat Transfer* B **46** 301–27

[25] Dragojlovic Z, Kaminski D A and Ryoo J 2001 Tuning of a fuzzy rule set for controlling convergence of a CFD solver in turbulent flow *Int. J. Heat Mass Transfer* **44** 3811–22

[26] Ryoo J, Dragojlovic Z and Kaminski D A 2005 Control of convergence in a computational fluid dynamics simulation using ANFIS *IEEE Trans. Fuzzy Syst.* **13** 42–7

[27] Xu C, Qin T, Wang G and Liu T-Y 2017 Reinforcement learning for learning rate control (arXiv: 1705.11159)

[28] Rabault J and Kuhnle A 2019 Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach *Phys. Fluids* **31** 094105

[29] Liang E, Liaw R, Nishihara R, Moritz P, Fox R, Goldberg K, Gonzalez J, Jordan M and Stoica I 2018 RLlib: abstractions for distributed reinforcement learning *Int. Conf. on Machine Learning* pp 3053–62 (http://proceedings.mlr.press/v80/liang18b.html)

[30] Schaarschmidt M, Kuhnle A and Fricke K 2017 Tensorforce: a TensorFlow library for applied reinforcement learning *Web Page* (https://github.com/tensorforce/tensorforce)

[31] Espeholt L, Marinier Rel, Stanczyk P, Wang K, and Michalski M 2019 SEED RL: scalable and efficient deep-RL with accelerated central inference (arXiv: 1910.06591)

[32] Hoffman M *et al* 2020 Acme: a research framework for distributed reinforcement learning (arXiv: 2006.00979)

[33] Schulman J, Wolski F, Dhariwal P, Radford A, and Klimov O 2017 Proximal policy optimization algorithms (arXiv: 1707.06347)

[34] Sutton R S, McAllester D A, Singh S P and Mansour Y 2000 Policy gradient methods for reinforcement learning with function approximation *Advances in Neural Information Processing Systems (Denver, CO)* pp 1057–63

[35] Konda V R and Tsitsiklis J N 2000 Actor-critic algorithms *Advances in Neural Information Processing Systems* (Cambridge, MA: MIT Press) pp 1008–14

[36] Schulman J, Levine S, Abbeel P, Jordan M and Moritz P 2015 Trust region policy optimization *Int. Conference on Machine Learning* pp 1889–97 (http://proceedings.mlr.press/v37/schulman15.html)

[37] Schulman J, Moritz P, Levine S, Jordan M, and Abbeel P 2015 High-dimensional continuous control using generalized advantage estimation (arXiv: 1506.02438)

[38] Vashishtha S and Verma S 2020 Restoring chaos using deep reinforcement learning *Chaos* **30** 031102

[39] Grebogi C, Ott E and Yorke J A 1983 Crises, sudden changes in chaotic attractors and transient chaos *Physica* D: **7** 181–200

[40] Ottino J M 1990 Mixing, chaotic advection and turbulence *Annu. Rev. Fluid Mech.* **22** 207–54

[41] Dobson I and Chiang H-D 1989 Towards a theory of voltage collapse in electric power systems *Syst. Control Lett.* **13** 253–62

[42] Yang W, Ding M, Mandell A J and Ott E 1995 Preserving chaos: control strategies to preserve complex dynamics with potential relevance to biological disorders *Phys. Rev.* E **51** 102

[43] Lorenz E N 1963 Deterministic nonperiodic flow *J. Atmos. Sci.* **20** 130–41

[44] Kaplan J L and Yorke J A 1979 Preturbulence: a regime observed in a fluid flow model of lorenz *Commun. Math. Phys.* **67** 93–108

[45] Jaderberg M *et al* 2017 Population based training of neural networks (arXiv: 1711.09846)

[46] Saad Y 2003 *Iterative Methods for Sparse Linear Systems* (Philadelphia, PA: SIAM)

[47] Patankar S 2018 *Numerical Heat Transfer and Fluid Flow* (London: Taylor & Francis)

[48] Driver D M and Lee Seegmiller H 1985 Features of a reattaching turbulent shear layer in divergent channelflow *AIAA J.* **23** 163–71

[49] Rumsey C L 2015 Recent developments on the turbulence modeling resource website *22nd AIAA Computational Fluid Conf* p 2927

[50] Jasak H *et al* OpenFOAM: a C++ library for complex physics simulations *Int. Workshop on Coupled Methods in Numerical Dynamics* 1000, 1–20 (IUC Dubrovnik Croatia) 2007.

[51] Menter F R, Kuntz M and Langtry R 2003 Ten years of industrial experience with the SST turbulence model *Turbul. Heat Mass Transfer* **4** 625–32

[52] Taghizadeh S, Witherden F D and Girimaji S S 2020 Turbulence closure modeling with data-driven techniques: physical compatibility and consistency considerations *New J. Phys.* **22** 093023

[53] Meliga P, Boujo E, Pujals G and Gallaire F 2014 Sensitivity of aerodynamic forces in laminar and turbulent flow past a square cylinder *Phys. Fluids* **26** 104101

[54] Wang Q and Gao J-H 2013 The drag-adjoint field of a circular cylinder wake at Reynolds numbers 20, 100 and 500 *J. Fluid Mech.* **730** 145–61