



MOX-Report No. 75/2026

HypeMARL: Multi-Agent Reinforcement Learning For High-Dimensional, Parametric, and Distributed Systems

Botteghi, N; Tomasetto, M; Fasel, U; Braghin, F; Manzoni, A

MOX, Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

mox-dmat@polimi.it

<https://mox.polimi.it>

HypeMARL: Multi-Agent Reinforcement Learning For High-Dimensional, Parametric, and Distributed Systems

Nicolò Botteghi^a, Matteo Tomasetto^b, Urban Fasel^c, Francesco Braghin^b, Andrea Manzoni^a

^a*MOX – Department of Mathematics, Politecnico di Milano, Piazza Leonardo da Vinci 32, I-20133, Milano, Italy*

^b*Department of Mechanical Engineering, Politecnico di Milano, Milano Italy*

^c*Department of Aeronautics, Imperial College London, London, United Kingdom*

Abstract

Deep reinforcement learning has recently emerged as a promising feedback control strategy for complex dynamical systems governed by partial differential equations (PDEs). When dealing with distributed, high-dimensional problems in state and control variables, multi-agent reinforcement learning (MARL) has been proposed as a scalable approach for breaking the curse of dimensionality. In particular, through decentralized training and execution, multiple agents cooperate to steer the system towards a target configuration, relying solely on local state and reward information. However, the principle of locality may become a limiting factor whenever a collective, nonlocal behavior of the agents is crucial to maximize the reward function, as typically happens in PDE-constrained optimal control problems. In this work, we propose *HypeMARL*: a decentralized MARL algorithm tailored to the control of high-dimensional, parametric, and distributed systems. HypeMARL employs hypernetworks to effectively parametrize the agents’ policies and value functions with respect to the system parameters and the agents’ relative positions, encoded by sinusoidal positional encoding. Through the application on challenging control problems, such as density and flow control, we show that HypeMARL *(i)* can effectively control systems through a collective behavior of the agents, outperforming state-of-the-art decentralized MARL, *(ii)* can efficiently deal with parametric dependencies, *(iii)* requires minimal hyperparameter tuning and *(iv)* can reduce the amount of expensive environment interactions thanks to its model-based extension, *MB-HypeMARL*, which relies on computationally efficient deep learning-based surrogate models approximating the dynamics locally, with minimal deterioration of the policy performance.

1. Introduction

Many modern and large-scale engineering systems are composed of several smaller subsystems, such as in robotics, sensor networks, transportation systems, smart grids, and biological networks. We refer to these systems as *distributed systems*, consisting of the interconnection of two or more subsystems that work together to perform complex tasks [1]. The control of distributed systems relies on two or more controllers, each receiving an observation stream from a local sensor and providing a control input to the local subsystem. Unlike *centralized* systems, where a single controller controls the behavior of the entire system, distributed systems rely on *decentralized* decision-making. Each controller typically operates autonomously, often with only partial knowledge of the global state, and interacts with its neighbors. The central challenge lies in designing local yet coordinated control strategies that leverage only limited and local information, ensuring optimal behavior across the entire system. The a priori design of an effective control strategy may be very difficult or even impossible due to the complexity inherent in distributed systems. Indeed, the controllers might need to reach agreement on a shared quantity, maintain a collective formation, synchronize behaviors, or optimize a common cost function. On top of that, the controllers may be heterogeneous, have different dynamics, sensing capabilities, or objectives. Additionally, the controller’s behavior often needs to be updated online in order to reflect, for instance, changes in the environment over time, while gradually improving the performance of the whole distributed system [2].

In this paper we propose a new framework exploiting *reinforcement learning* (RL) [3] to address the problem of controlling high-dimensional, nonlinear, time-dependent, and parametrized dynamical systems, describing, e.g., the evolution of incompressible fluid flows in multiple scenarios. The dynamics of these

systems, arising in our case from the discretization of partial differential equations (PDEs) through suitable space and time discretization schemes, can be written as

$$\mathbf{y}_{t+1} = F(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\mu}), \quad t = 1, \dots, N_t, \quad (1)$$

where $\mathbf{y}_t$ and $\mathbf{y}_{t+1} \in \mathbb{R}^{N_y}$ represent, respectively, the system state at time step t and $t+1$, with N_y indicating the dimension of the state and $t = 1, \dots, N_t$ the time steps, and $\mathbf{u}_t \in \mathbb{R}^{N_u}$ denotes the N_u -dimensional control input at time step t , $F : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \times \mathbb{R}^{N_\mu} \rightarrow \mathbb{R}^{N_y}$ is the state transition map, and $\boldsymbol{\mu} \in \mathbb{R}^{N_\mu}$ is a N_μ -dimensional parameter vector belonging to a suitable parameter space $\mathcal{P} \subseteq \mathbb{R}^{N_\mu}$. The dynamical system may be high-dimensional in state and control dimensions, i.e., $N_y \gg 1$ and $N_u \gg 1$. In these settings, we seek for an effective control strategy that can either deal with the high dimensionality of the problem, while being adaptive to different instances of the system parameters. The latter makes traditional optimal control approaches based on the Hamilton–Jacobi–Bellman equation or the Pontryagin maximum principle intractable, as different optimal control problems must be solved separately for each parameter instance $\boldsymbol{\mu}$.

1.1. Reinforcement Learning and Multi-agent Reinforcement Learning: existing approaches

RL is a branch of machine learning concerned with how *agents*, namely the controllers of the distributed system, should take *actions*, namely select control inputs, in an environment in order to maximize their cumulative reward [3]. Since the agent does not initially know which actions will yield the highest reward, it must explore different strategies to improve its behavior, balancing the trade-off between exploiting known high-reward actions and exploring new, potentially even higher-reward actions. In recent years, advances in function approximation, particularly through deep neural networks (NNs), have significantly enhanced RL, enabling it to solve high-dimensional and complex problems that were previously intractable [4, 5]. Deep RL have achieved success across a large variety of domains, spanning from game playing [6, 7, 8, 9, 10, 11] to robotic control [12, 13, 14, 15, 16, 17, 18], autonomous driving [19, 20, 20], resource management [21, 22], and PDEs [23, 24, 25, 26, 27], to mention a few.

Multi-agent reinforcement learning (MARL) is the generalization of RL to multi-agent systems where optimal policies have to be determined for a set of agents. An important distinction among MARL algorithms lies in the assumptions made when training and deploying the agents. In particular, we can identify three classes of methods, namely (i) *centralized training and execution*, (ii) *decentralized training and execution*, and (iii) *centralized training and decentralized execution*. In the centralized training and execution approach, all agents share all the state observations with each other during the learning and deployment phase. Conversely, decentralized training and execution approaches assume no shared knowledge among the agents, with every agent relying solely on local information. Eventually, centralized training and decentralized execution methods combine the two aforementioned strategies, with observations shared among the agents during training, while only local state information is taken into account during execution. Typical application domains of MARL are distributed control [2], multi-robot systems [28, 29], resource management [30], games and competitive environments [31, 32], autonomous driving and traffic control [33, 34], and only recently active flow control [35, 36, 37, 38, 39]. Active flow control and, more in general, the control of distributed systems described in terms of PDEs, poses an extreme challenge to centralized RL algorithms due to the high dimension of state and action spaces. Especially the curse of dimensionality of the action space has been the key motivation for the employment and the development of (decentralized) MARL approaches in these settings [40, 35]. However,

- the (full) decentralization, while enabling effective scaling of RL to high-dimensional problems, does not allow the learning of cooperative policies among the different agents, which is essential to discover effective control strategies, and
- the adaptation capabilities of MARL approaches has not been demonstrated in control problems governed by parametrized PDEs, where the parameters of the system dynamics vary.

Recent research in MARL has focused on centralized training and decentralized execution approaches. Examples in this direction are multi-agent deep deterministic policy gradient (MADDPG) [41], multi-agent proximal policy optimization (PPO) [42], multi-agent trust region policy optimization [43], and multi-agent attention actor-critic [44]. However, when dealing with a large number of agents, learning a joint value

function is extremely difficult [44], making these approaches not suitable for the class of control problems of interest. To mitigate this issue, it is possible to consider value factorization methods, such as value decomposition network [45], QMIX [46], FacMADDPG [47], QTRAN [48], and mean-field MARL [49], which learn a factorization of the joint value function. However, these methods suffer from relative overgeneralization, where policies converge to suboptimal joint actions [50, 51]. Independent Q-learning has been first proposed in [52], decentralized approaches – such as, e.g., independent PPO [51] and independent actor-critic COMA [53] – have recently regained popularity as they reduce the complexity of the algorithms and, often, the size of NNs employed. It is important to notice that, while independent learning improves algorithm complexity, it often does not work well when the task requires coordination among the agents [54, 55], which is critical for effective and distributed control of a dynamical system. Many other factors can be used to classify MARL algorithms, such as the number of agents, the assumptions on the rewards, or the type of communication among the agents: for the sake of conciseness, we refer the reader to [56] for additional details.

1.2. Proposed Approach

In this research, we propose *HypeMARL*, a novel decentralized MARL algorithm tailored to the control of high-dimensional, parametrized, and distributed systems described in terms of (possibly nonlinear) time-dependent PDEs. A summary of the HypeMARL strategy is provided in Figure 1.

Unlike other decentralized MARL approaches for PDEs control recently proposed in the literature [35, 39, 36], we focus on developing an approach capable at the same time to (i) control systems with parametric dependencies, e.g., in the context of flow control, through different inflow angles of attack, and (ii) perform tasks that require coordination among a large number of agents. Specifically, we inform the local policies with respect to the system parameters μ and the relative position p_i of each agent with respect to the others to enhance generalization capabilities and coordination among agents, respectively. The relative position

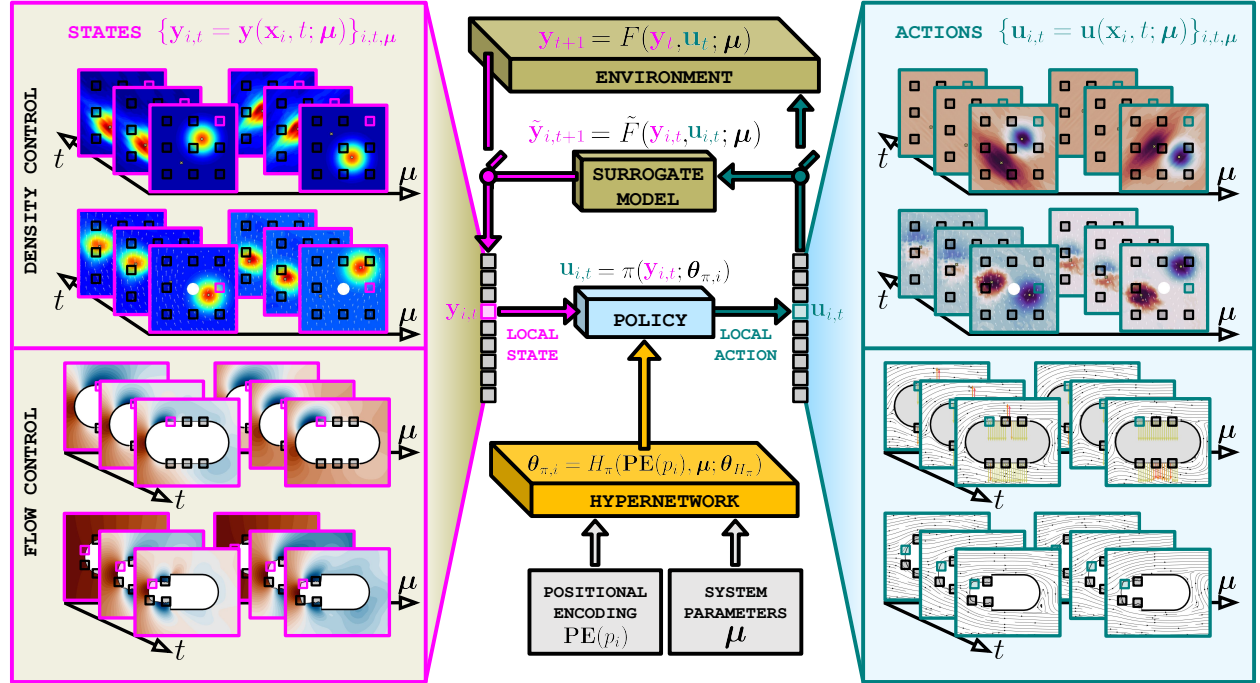


Figure 1. HypeMARL for control of high-dimensional, parametric and distributed systems. Each agent relies on local rewards $r_{i,t}$ to learn the optimal local policy. After training, the local policy provides the optimal action for every agent over time $\mathbf{u}_{i,t} = \pi(\mathbf{y}_{i,t}; \theta_{\pi,i})$, starting from the local state $\mathbf{y}_{i,t}$. To specialize the policy with respect to different agents, the policy parameters $\theta_{\pi,i}$ are determined through a hypernetwork with respect to positional encoding of the agent’s relative position $\mathbf{PE}(p_i)$ and the system parameters μ , that is $\theta_{\pi,i} = H_{\pi}(\mathbf{PE}(p_i), \mu; \theta_H)$. While model-free HypeMARL interacts with the real environment $\mathbf{y}_{t+1} = F(\mathbf{y}_t, \mathbf{u}_t; \mu)$, where $\mathbf{y}_t$ and $\mathbf{u}_t$ collect the local state and control values, respectively, the model-based version MB-HypeMARL learns a surrogate model approximating the local dynamics $\tilde{\mathbf{y}}_{i,t+1} = \tilde{F}(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}; \mu)$ in order to enhance sample efficiency and speed up training.

of each agent is encoded via *sinusoidal position encoding* [57], as taken into account in natural language processing to embed the position of each word in a sentence, that is

$$\mathbf{PE}(p_i) = \left[\sin\left(\frac{p_i}{\omega_1}\right), \cos\left(\frac{p_i}{\omega_1}\right), \sin\left(\frac{p_i}{\omega_2}\right), \dots, \sin\left(\frac{p_i}{\omega_d}\right), \cos\left(\frac{p_i}{\omega_d}\right) \right],$$

where $\omega_j = n^{2j/d}$ for $1 \leq j \leq d$, with d being the dimension of the positional embedding vector and n a constant value. Rather than relying on standard neural networks to improve expressiveness and generalization capabilities of the policies, we employ an *hypernetwork* [58] to learn weights and biases of all local policies π_i that are dependent on physical system information $\boldsymbol{\mu}$ and the inter-agents relations $\mathbf{PE}(p_i)$. The hypernetwork $H_\pi(\cdot; \boldsymbol{\theta}_{H_\pi})$, with learnable parameters $\boldsymbol{\theta}_{H_\pi}$, learns the parameters of each agent's policy $\boldsymbol{\theta}_{\pi,i}$:

$$\begin{aligned} \boldsymbol{\theta}_{\pi,i} &= H_\pi(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi}) \\ \mathbf{u}_{i,t} &= \pi(\mathbf{y}_{i,t}; \boldsymbol{\theta}_{\pi,i}), \quad i = 1, \dots, N \end{aligned}$$

where $\mathbf{y}_{i,t}$ and $\mathbf{u}_{i,t}$ indicate state and control of agent i , and N denotes the number of agents.

Hypernetworks are a class of NNs that provide the parameters, i.e., the weights and biases, of other NNs, often referred to as main or primary networks. Hypernetworks improve the expressivity and performance of NN-based architectures [59], with promising results in a wide variety of deep learning problems, including continual learning [60], weight pruning [61], uncertainty quantification [62], and recently DRL [63]. Hypernetworks in RL were first exploited in [63] to learn the parameters of the NNs modeling the value function or the policy. In the context of meta RL, zero-shot RL, and continual RL hypernetworks have been employed by [64, 65, 66]. In the context of control of parametric PDEs, instead, our previous work [67] showed that hypernetworks enable strong generalization capabilities across different instances of the systems parameters, outperforming standard NN-based parametrizations. However, this hypernetwork-based approach has only been tested in single-agent and centralized RL settings in presence of low-dimensional states and controls. In the context of decentralized MARL, instead, hypernetworks were shown to be an effective way to achieve parameter sharing, which is an important strategy for improving the agents' performance and scaling to large numbers of agents by significantly reducing the number of learnable parameters and consequently training times [56, 52, 68, 69].

Besides the model-free HypeMARL strategy, we also propose a model-based extension, named *model-based HypeMARL* (*MB-HypeMARL*), where we learn a computationally efficient deep learning-based surrogate model approximating the *local* environment dynamics, resulting in faster training procedures with minimal deterioration of the policy performance

$$\tilde{\mathbf{y}}_{i,t+1} = \tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\mu}),$$

where we indicate with $\tilde{\mathbf{y}}_{i,t+1}$ and $\tilde{\mathbf{y}}_{i,t}$ the states predicted by the surrogate model $\tilde{F}$ to distinguish them from the real-environment measurable states $\mathbf{y}_{i,t+1}$ and $\mathbf{y}_{i,t}$. It is worth mentioning that $\tilde{F}(\cdot)$ solely predicts the evolution of the measured state $\tilde{\mathbf{y}}_{i,t}$ when action $\mathbf{u}_{i,t}$ is applied by agent i and not the evolution of the full environment state, which may be extremely challenging and computational demanding due to its high dimensionality. Specifically, we approximate $\tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\mu}) = \tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}, \boldsymbol{\mu}; \boldsymbol{\theta}_{\tilde{F}})$ with a shallow neural network of parameters $\boldsymbol{\theta}_{\tilde{F}}$. Using the data collected from the real-environment interaction, the parameters of the local forward model $\boldsymbol{\theta}_{\tilde{F}}$ are optimized by minimizing the prediction error between the samples from the environment $\mathbf{y}_{t+1}$ and the surrogate model predictions $\tilde{\mathbf{y}}_{t+1}$. To assess the performance of the proposed algorithms, we consider different high-dimensional, parametric, and distributed control problems, namely *density control* in a vacuum and in a fluid, and *flow control* with different actuation strategies, such as jets and shape-morphing obstacles.

The paper is organized as follows: in Section 2, we introduce RL, MARL, and the building blocks of our framework. In Section 3, we present in detail our proposed strategy, HypeMARL, while in Section 4 we show a series of numerical results related with (i) density control in a vacuum, (ii) density control in a fluid, and (iii) flow control with either vertical jets or shape morphing. Eventually, Section 5 concludes the paper and discusses our main findings.

2. Background

2.1. Reinforcement Learning

RL lays its mathematical foundation on the Markov decision process (MDP) framework [3]. Formally, a MDP is defined by a tuple $(\mathcal{Y}, \mathcal{U}, F, R)$, where $\mathcal{Y} \subset \mathbb{R}^{N_y}$ and $\mathcal{U} \subset \mathbb{R}^{N_u}$ denote the state and action spaces with dimensions N_y and N_u , respectively, $F(\mathbf{y}, \mathbf{y}', \mathbf{u}) : \mathcal{Y} \times \mathcal{Y} \times \mathcal{U} \rightarrow [0, 1]$ indicates the transition probability for any state $\mathbf{y}, \mathbf{y}' \in \mathcal{Y}$ and control $\mathbf{u} \in \mathcal{U}$, while $R(\mathbf{y}, \mathbf{u}) : \mathcal{Y} \times \mathcal{U} \rightarrow \mathbb{R}$ indicates the reward function that determines the instantaneous reward the agent receives for each action taken in a given state. The agent's behavior is defined by its *policy* π , namely the set of rules that the agent exploits to select a specific action in a given state $\mathbf{y}$. The policy may be either stochastic $\pi : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \rightarrow [0, 1]$ or deterministic $\pi : \mathbb{R}^{N_y} \rightarrow \mathbb{R}^{N_u}$. The agent's goal is to find the optimal policy, i.e., the policy that maximizes the total expected cumulative reward from every state:

$$J(\mathbf{y}) = \mathbb{E}_\pi \left[\sum_{k=0}^{N_t-1} \gamma^k r_k | Y_0 = \mathbf{y} \right], \quad (2)$$

where $\mathbb{E}_\pi$ denotes the expected value under the policy π , $\gamma \in [0, 1)$ the discount factor, and N_t the horizon over which actions are taken that may be either finite or infinite. However, maximizing the long-term behavior while only receiving immediate reward is extremely challenging.

RL algorithms can be classified according to the strategy used to find the optimal policy: (i) *value-based methods*, (ii) *policy-based methods*, and (iii) *actor-critic methods*. Value-based methods look for an estimate of the state-action value function $Q : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \rightarrow \mathbb{R}$ associated with a policy π . The state-action value function gives the expected return obtained when following the policy π , starting from a given state. The optimal policy is derived by taking the greedy action with respect to the value function estimate. Examples of value-based methods are Q-learning [70] and SARSA [71]. Policy-based methods, instead, consider a parametrization of the policy π_θ with respect to a set of parameters θ – which may be, for instance, NN weights and biases in the context of deep reinforcement learning (DRL) [4, 5, 72] – and aim at optimizing those parameters through the policy gradient [73, 3]. Examples of policy-based methods are REINFORCE [74] and deterministic policy gradient [75]. Eventually, actor-critic strategies combine the estimation of the value function with the direct optimization of the policy parameters. Examples are PPO [76] and DDPG [77]. Another important distinction regarding RL algorithms is that between (i) *model-free* and (ii) *model-based* algorithms. Model-free algorithms utilize data collected from the interaction with the environment to directly learn the optimal policy and value function. Conversely, model-based RL exploits the data stream to build a *surrogate* model of the environment dynamics that serves as a computationally efficient proxy of the real environment to optimize the control strategy. Model-based RL addresses the problem of the sample inefficiency of model-free RL, but this approach is often subjected to model biases that degrade the policy performance when deployed on the real system.

2.1.1. Multi-agent reinforcement learning

When multiple decision-making agents interact with the environment to achieve certain goals, we can talk about a multi-agent system [56]. Multi-agent systems find applications in a large variety of domains such as robotics, distributed control, collaborative decision-support systems. MARL algorithms learn optimal policies for a set of agents in a multi-agent system. MARL is formally modeled using the multi-agent extension of the MDP, namely the stochastic game [2].

Definition 2.1. A stochastic game is defined by a tuple $(\mathcal{Y}, \mathcal{U}_1, \dots, \mathcal{U}_N, F, R_1, \dots, R_N)$, where $\mathcal{Y} \subset \mathbb{R}^{N_y}$ denotes the N_y -dimensional state space, $\mathcal{U}_i \subset \mathbb{R}^{N_{u_i}}$ with $i = 1, \dots, N$ are the N_{u_i} -dimensional actions spaces of the individual agents, N is the number of agents, $\mathcal{U} = \mathcal{U}_1 \times \mathcal{U}_2 \times \dots \times \mathcal{U}_N$ is the joint action space, $F : \mathcal{Y} \times \mathcal{Y} \times \mathcal{U} \mapsto [0, 1]$ indicates the transition probability, and $R_i : \mathcal{Y} \times \mathcal{U} \mapsto \mathbb{R}$ with $i = 1, \dots, N$ indicates the reward functions of the agents.

We can identify three classes of methods, namely (i) centralized training and execution, (ii) decentralized training and execution, and (iii) centralized training and decentralized execution. In the first case, the value function is defined using the state and joint actions space, i.e., $Q : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \rightarrow \mathbb{R}$, while the (stochastic) policy of the i^{th} agent is $\pi_i : \mathbb{R}^{N_y} \times \mathbb{R}^{N_{u_i}} \rightarrow [0, 1]$. Conversely, decentralized training and execution result in local value functions and policies based only on local state information, i.e., $Q_i : \mathbb{R}^{N_{y_i}} \times \mathbb{R}^{N_{u_i}} \rightarrow \mathbb{R}$ and

$\pi_i : \mathbb{R}^{N_{y_i}} \times \mathbb{R}^{N_{u_i}} \rightarrow [0, 1]$, where N_{y_i} denotes the dimension of the local state space. Eventually, centralized training and decentralized execution is a hybrid case of the first two, in which a global value function is estimated, namely $Q : \mathbb{R}^{N_y} \times \mathbb{R}^{N_u} \rightarrow \mathbb{R}$, and a local policy is optimized, namely $\pi_i : \mathbb{R}^{N_{y_i}} \times \mathbb{R}^{N_{u_i}} \rightarrow [0, 1]$.

2.2. Twin-delayed deep deterministic policy gradient

In our numerical experiments, we utilize an actor-critic approach, namely twin-delayed deep deterministic policy gradient (TD3) [78]. TD3 can handle continuous state and action spaces, making it a suitable candidate for controlling parametric PDEs using smooth control strategies. TD3 learns a deterministic policy $\pi(\cdot)$, i.e., the actor, and two action-value function $Q_1(\cdot)$ and $Q_2(\cdot)$, i.e., the critics. The actor and the critics are parametrized by means of NNs of parameters θ_π , θ_{Q_1} and θ_{Q_2} , respectively. We indicate the parametrized policy with $\pi(\mathbf{y}_t; \theta_\pi)$ and the action-value functions with $Q_1(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_1})$ and $Q_2(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_2})$.

The learnable parameters of policy and value functions are optimized using the interaction data $(\mathbf{y}_t, \mathbf{u}_t, r_t, \mathbf{y}_{t+1})$ for all time steps t stored in a memory buffer $\mathcal{D}$. Given a randomly-sampled batch of interaction tuples, we can update the parameters θ_{Q_j} of the j^{th} action-value function $Q_j(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_j})$ as:

$$\mathcal{L}(\theta_{Q_j}) = \mathbb{E}_{\mathbf{y}_t, \mathbf{u}_t, \sim \mathcal{D}} \left[L \left(Q_{\text{target}} - Q_j(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_j}) \right) \right], \quad j = 1, 2, \quad (3)$$

where L is a arbitrary chosen loss function, e.g. mean-squared error, and $\mathbf{y}_t, \mathbf{u}_t, \sim \mathcal{D}$ denotes the random sampling of the interaction data from the memory buffer $\mathcal{D}$. Similarly to [79], we choose L to be the Huber loss. The Huber loss behaves quadratically for small errors and linearly for large errors and reduces the influence of outliers in the target values and further improves the stability of critic learning [79]:

$$\text{Huber}(x) = \begin{cases} \frac{1}{2}(x)^2, & \text{if } |x| < 1 \\ |x| - \frac{1}{2}, & \text{otherwise.} \end{cases} \quad (4)$$

Although Equation (3) might look like a standard regression problem, in RL the regression target Q_{target} is not known a-priori and has to be generated using the same Q-function that is learned from data. This procedure is commonly known as *bootstrapping* [3]. To improve training stability and prevent the overestimation of the regression target, Q_{target} is computed using target networks, which are copies of the original networks with frozen parameters that are not updated in the backpropagation step. In TD3, the policy and each value function have associated target networks that we indicate with $\bar{\pi}(\mathbf{y}_t; \theta_{\bar{\pi}})$, $\bar{Q}_1(\mathbf{y}_t, \mathbf{u}_t; \theta_{\bar{Q}_1})$, and $\bar{Q}_2(\mathbf{y}_t, \mathbf{u}_t; \theta_{\bar{Q}_2})$. Using the target networks, the regression target can be computed as from:

$$\begin{aligned} q_{t+1} &= \min \left(\bar{Q}_1(\mathbf{y}_{t+1}, \bar{\pi}(\mathbf{y}_{t+1}; \theta_{\bar{\pi}}); \theta_{\bar{Q}_1}), \bar{Q}_2(\mathbf{y}_{t+1}, \bar{\pi}(\mathbf{y}_{t+1}; \theta_{\bar{\pi}}); \theta_{\bar{Q}_2}) \right), \\ Q_{\text{target}} &= r_t + \gamma q_{t+1}, \end{aligned} \quad (5)$$

with $r_t, \mathbf{y}_{t+1} \sim \mathcal{D}$.

At the same time, the action-value functions $Q_1(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_1})$ and $Q_2(\mathbf{y}_t, \mathbf{u}_t; \theta_{Q_2})$ are used to update the parameters of the policy $\pi(\mathbf{y}_t; \theta_\pi)$ according to the deterministic policy gradient theorem [75]. In particular, the gradient of the critic guides the improvements of the actor and the policy parameters are updated to ascend the action-value function:

$$\mathcal{L}(\theta_\pi) = \mathbb{E}_{\mathbf{y}_t \sim \mathcal{D}} \left[-\frac{1}{2} Q_1(\mathbf{y}_t, \pi(\mathbf{y}_t; \theta_\pi); \theta_{Q_1}) - \frac{1}{2} Q_2(\mathbf{y}_t, \pi(\mathbf{y}_t; \theta_\pi); \theta_{Q_2}) \right]. \quad (6)$$

Eventually, the target networks, parametrized by $\theta_{\bar{Q}_1}$, $\theta_{\bar{Q}_2}$, and $\theta_{\bar{\pi}}$, are updated with a slower frequency than the actor and the critic according to using a soft-update rule:

$$\begin{aligned} \theta_{\bar{Q}_1} &= \rho \theta_{Q_1} + (1 - \rho) \theta_{\bar{Q}_1}, \\ \theta_{\bar{Q}_2} &= \rho \theta_{Q_2} + (1 - \rho) \theta_{\bar{Q}_2}, \\ \theta_{\bar{\pi}} &= \rho \theta_\pi + (1 - \rho) \theta_{\bar{\pi}}, \end{aligned} \quad (7)$$

where ρ is a constant factor determining the speed of the updates of the target parameters.

2.3. HypeRL: Hypernetwork-based Reinforcement Learning

Our previous work [67] already proposed a way to enhance the TD3 algorithm by hypernetworks in the context of single-agent RL for control of parametrized, low-dimensional dynamical systems. HypeRL estimates two action-value functions $Q_1(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\theta}_{Q_1})$ and $Q_2(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\theta}_{Q_2})$ and a policy $\pi(\mathbf{y}_t; \boldsymbol{\theta}_\pi)$ using hypernetworks [58].

Hypernetworks are a family of neural network architectures in which the weights and biases of a main network are generated by a separate neural network, referred to as the hypernetwork [58]. Let $g(\cdot; \boldsymbol{\theta}_g)$ denote a main neural network, whose parameters $\boldsymbol{\theta}_g$ are generated by a hypernetwork $H(\cdot; \boldsymbol{\theta}_H)$ with parameters $\boldsymbol{\theta}_H$. Given a conditioning variable $\mathbf{z}$, the parameters of the target network are obtained as

$$\boldsymbol{\theta}_g = H(\mathbf{z}; \boldsymbol{\theta}_H),$$

where $\mathbf{z}$ may encode task-specific information, or physical parameters in the context of parametrized dynamical systems. For a given input $\mathbf{x}$, the output of the target network is then computed as

$$\hat{\mathbf{y}} = g(\mathbf{x}; \boldsymbol{\theta}_g).$$

Analogously to traditional neural networks, we can use gradient-based optimization to update $\boldsymbol{\theta}_g, \boldsymbol{\theta}_H$, propagating gradients through both the main network and the hypernetwork.

Using hypernetworks, the parameters of the main networks $\boldsymbol{\theta}_{Q_1}, \boldsymbol{\theta}_{Q_2}, \boldsymbol{\theta}_\pi$ are now learned using three hypernetworks $H_{Q_1}(\mathbf{z}_t; \boldsymbol{\theta}_{H_{Q_1}})$, $H_{Q_2}(\mathbf{z}_t; \boldsymbol{\theta}_{H_{Q_2}})$, and $H_\pi(\mathbf{z}_t; \boldsymbol{\theta}_{H_\pi})$:

$$\begin{aligned}\boldsymbol{\theta}_{Q_1} &= H_{Q_1}(\mathbf{z}_t; \boldsymbol{\theta}_{H_{Q_1}}), \\ \boldsymbol{\theta}_{Q_2} &= H_{Q_2}(\mathbf{z}_t; \boldsymbol{\theta}_{H_{Q_2}}), \\ \boldsymbol{\theta}_\pi &= H_\pi(\mathbf{z}_t; \boldsymbol{\theta}_{H_\pi}),\end{aligned}\tag{8}$$

where $\mathbf{z}_t$ indicates the hypernetwork input, encoding physical or task-dependent information of the dynamical systems. Using this parametrization, HypeRL replaces the standard NN-based policy with:

$$\begin{aligned}\boldsymbol{\theta}_\pi &= H_\pi(\mathbf{z}_t; \boldsymbol{\theta}_{H_\pi}), \\ \mathbf{u}_t &= \pi(\mathbf{y}_t; \boldsymbol{\theta}_\pi),\end{aligned}\tag{9}$$

and analogously, the value functions with:

$$\begin{aligned}\boldsymbol{\theta}_{Q_j} &= H_{Q_j}(\mathbf{z}_t; \boldsymbol{\theta}_{H_{Q_j}}), \\ q_{j,t} &= Q_j(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\theta}_{Q_j}), \quad j = 1, 2,\end{aligned}\tag{10}$$

where $q_{j,t}$ is the predicted Q-value by the action-value function $Q_j(\cdot, \cdot)$. The hypernetwork parameters are jointly optimized with the main network parameters by simply allowing the gradient of the TD3 training objectives in Equations (3) and (6) with the respective target networks that are updated using Equation (7), leading to:

$$\mathcal{L}(\boldsymbol{\theta}_{Q_j}, \boldsymbol{\theta}_{H_{Q_j}}) = \mathbb{E}_{\mathbf{y}_t, \mathbf{u}_t, \sim \mathcal{D}} \left[\text{Huber} \left(Q_{\text{target}} - Q_j(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\theta}_{Q_j}) \right) \right], \quad j = 1, 2,\tag{11}$$

and

$$\mathcal{L}(\boldsymbol{\theta}_\pi, \boldsymbol{\theta}_{H_\pi}) = \mathbb{E}_{\mathbf{y}_t \sim \mathcal{D}} \left[-\frac{1}{2} Q_1(\mathbf{y}_t, \pi(\mathbf{y}_t; \boldsymbol{\theta}_\pi); \boldsymbol{\theta}_{Q_1}) - \frac{1}{2} Q_2(\mathbf{y}_t, \pi(\mathbf{y}_t; \boldsymbol{\theta}_\pi); \boldsymbol{\theta}_{Q_2}) \right].\tag{12}$$

3. Hypernetwork-based positional encoding for multi-agent reinforcement learning

Many real-world control applications are described by dynamical systems with high-dimensional state and control spaces, e.g., active flow control. Let us now consider the case of a high-dimensional, nonlinear, time-dependent, and parametric dynamical system, e.g., obtained through suitable time and space discretization schemes of a PDE:

$$\mathbf{y}_{t+1} = F(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\mu}), \quad t = 1, \dots, N_t.$$

From an RL perspective, the large number of state variables and control inputs, i.e. $N_y \gg 1$ and $N_u \gg 1$, leads to prohibitively large state-action spaces, making the direct use of single-agent RL methods computationally inefficient or even impractical, e.g., in the density and flow control problem presented in this work (see Figure A.12 in Appendix A). Therefore, motivated by the recent works from the literature, e.g., [39], we cast the control problem of such a system as a fully decentralized MARL problem, in which each agent relies solely on local state measurements and on the knowledge of the system parameters μ to determine its actions. However, we argue that while full decentralization is required in order to scale up to large numbers of agents, it is not enough to achieve global coordinated behaviors that are needed for certain control tasks, e.g., in density and flow control (see Section 4). In the following, we introduce a hypernetwork-based positional encoding algorithm, that we named HypeMARL.

3.1. Exploiting Agents' Relative Positions and System Parameters

Instead of learning a global policy $\mathbf{u}_t = \pi(\mathbf{y}_t; \mu)$ operating from high-dimensional states and controls, we learn N local policies $\mathbf{u}_{i,t} = \pi(\mathbf{y}_{i,t}; \theta_{\pi,i})$, obtained as specializations of the global policy π with respect to the local state measurements $\mathbf{y}_{i,t}$. To enable coordination among the agents, we exploit the knowledge of the relative agent position p_i , that is simply an agent's ID and, in our case, corresponds to the index assigned by the finite element numerical solver. While many positional encoding strategies have been proposed in the literature [80], in this work we focus on sinusoidal positional encoding [57] to embed the relative position of an agent p_i into a higher-dimensional vector $\mathbf{PE}(p_i)$ using sine and cosine functions at different frequencies

$$\mathbf{PE}(p_i) = \left[\sin\left(\frac{p_i}{\omega_1}\right), \cos\left(\frac{p_i}{\omega_1}\right), \sin\left(\frac{p_i}{\omega_2}\right), \dots, \sin\left(\frac{p_i}{\omega_d}\right), \cos\left(\frac{p_i}{\omega_d}\right) \right],$$

where $\omega_j = n^{2j/d}$ for $1 \leq j \leq d$, with d being the dimension of the positional embedding vector and n a constant value. Differently from traditional decentralized MARL strategies, where the same local state values intrinsically result in the same local actions regardless of the agent's position, the knowledge of the relative position is crucial to specialize the independent agents even when observing the same state values and to achieve coordinated strategies even though relying solely on local information. For instance, in the density control test cases (detailed in the Section 4), different parametric position-specific actions are required for the same observed state. In addition to the relative positions of the agents, we exploit the system parameters μ , which may be known a priori, estimated from data, or related to task-dependent information specified by the user, to learn adaptive policies that can generalize across different scenarios.

3.2. Hypernetwork-based Encoding and Parameter Sharing

While the positional encoding vector $\mathbf{PE}(p_i)$ and the system parameters μ may be concatenated to the local state $\mathbf{y}_{i,t}$ and provided as input to the agent's policy and value function, we argue that concatenation may not be the best choice. As shown in Figure B.13 in Appendix B, this naive solution does not yield acceptable results due to the remarkably high dimension of the resulting state, making the learning of the local policies and value functions extremely challenging. Therefore, following upon our previously proposed approach [67], we propose to use hypernetworks to encode positional-encoding and parameter dependencies into the weights and biases of the local policies and value functions. Building upon the TD3 algorithm (see Section 2.2) and our previous work HypeRL (see Section 2.3), we rely on three hypernetworks, namely H_π , H_{Q_1} , and H_{Q_2} , which parametrize the weights and biases of the policy and value functions with respect to positional encoding $\mathbf{PE}(p_i)$ and system parameters μ . Specifically, the hypernetworks $H_\pi : \mathbb{R}^{d+N\mu} \rightarrow \mathbb{R}^{N\theta_\pi}$, $H_{Q_1} : \mathbb{R}^{d+N\mu} \rightarrow \mathbb{R}^{N\theta_{Q_1}}$, and $H_{Q_2} : \mathbb{R}^{d+N\mu} \rightarrow \mathbb{R}^{N\theta_{Q_2}}$ learn the parameters of the NNs that model, respectively, all the local policies and value functions of the N agents, that is

$$\theta_{\pi,i} = H_\pi(\mathbf{PE}(p_i), \mu; \theta_{H_\pi}), \quad \theta_{Q_1,i} = H_{Q_1}(\mathbf{PE}(p_i), \mu; \theta_{H_{Q_1}}), \quad \theta_{Q_2,i} = H_{Q_2}(\mathbf{PE}(p_i), \mu; \theta_{H_{Q_2}}), \quad (13)$$

resulting into the following policies and value functions

$$\pi(\mathbf{y}_{i,t}; \theta_{\pi,i}), \quad Q_1(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}; \theta_{Q_1,i}), \quad Q_2(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}; \theta_{Q_2,i}), \quad i = 1, \dots, N. \quad (14)$$

The architecture of the hypernetworks is discussed in Appendix C. In the context of MARL, hypernetworks can be viewed as an effective form of parameter sharing. Indeed, the three hypernetworks in Equation (13)

are exploited to generate the weights and biases of the policy and value function for all the agents as function of $\mathbf{PE}(p_i)$ and $\boldsymbol{\mu}$. Each local policy and value function is optimized according to the TD3 objectives (see Equations (3) and (6)) with the gradients of the loss functions updating the three different hypernetworks:

$$\mathcal{L}(\boldsymbol{\theta}_{\pi,i}, \boldsymbol{\theta}_{H_{\pi}}) = \mathbb{E}_{\mathbf{y}_t \sim \mathcal{D}} \left[-\frac{1}{2} Q_1(\mathbf{y}_t, \pi(\mathbf{y}_t; \boldsymbol{\theta}_{\pi,i}); \boldsymbol{\theta}_{Q_1,i}) - \frac{1}{2} Q_2(\mathbf{y}_t, \pi(\mathbf{y}_t; \boldsymbol{\theta}_{\pi,i}); \boldsymbol{\theta}_{Q_2,i}) \right], \quad (15)$$

and

$$\mathcal{L}(\boldsymbol{\theta}_{Q_j,i}, \boldsymbol{\theta}_{H_{Q_j}}) = \mathbb{E}_{\mathbf{y}_t, \mathbf{u}_t \sim \mathcal{D}} \left[\text{Huber} \left(Q_{\text{target}} - Q_j(\mathbf{y}_t, \mathbf{u}_t; \boldsymbol{\theta}_{Q_j,i}) \right) \right], \quad j = 1, 2. \quad (16)$$

A graphical visualization of the training pipeline of HypeMARL is depicted in Figure 2. The data collected during the interaction with the environment are stored in the memory buffer and subsequently sampled to train local policies and value functions. While in this work we rely on a TD3 algorithm for each agent, it is

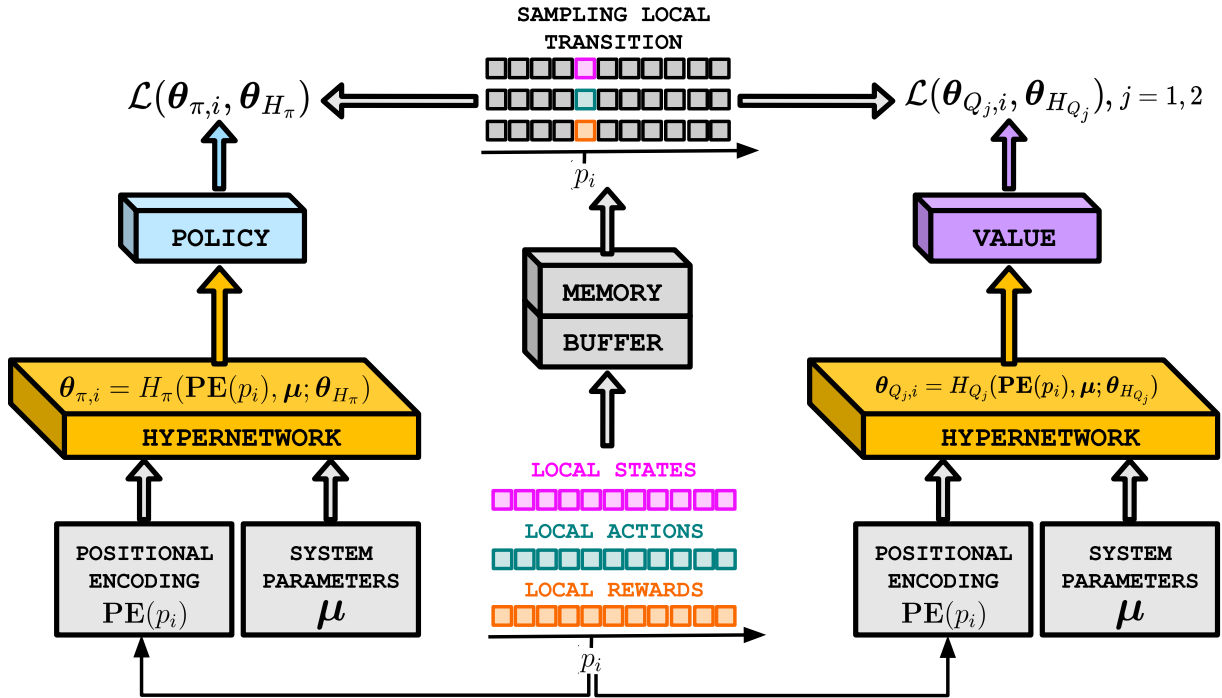


Figure 2. Schematic diagram of the training pipeline of HypeMARL.

worth mentioning that the hypernetwork-based parametrization of policy and value function can be used in combination with any other (MA)RL algorithm, without changing its core working principles.

3.3. Model-based HypeMARL

In general, model-free (MA)RL can be quite sample inefficient. Sample efficiency may be a limiting factor for very complex systems where only limited data can be collected and running simulations is computationally expensive, such as in the case of fluid dynamics. To mitigate this issue, we propose model-based HypeMARL (MB-HypeMARL) to drastically reduce the number of interactions with the real environment (see Section 4). Differently from standard model-based RL algorithms that learn a global surrogate model of the environment’s dynamics, which can be extremely challenging for high-dimensional systems, MB-HypeMARL learns a surrogate model of the local dynamics $\tilde{F}$ that can be autoregressively queried to predict the local evolution of the system – namely the evolution of the local measured state $\mathbf{y}_{i,t}$ when the local action $\mathbf{u}_{i,t}$ is applied by agent i for a given parameter $\boldsymbol{\mu}$:

$$\tilde{\mathbf{y}}_{i,t+1} = \tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\mu}),$$

where we indicate with $\tilde{\mathbf{y}}_{i,t+1}$ and $\tilde{\mathbf{y}}_{i,t}$ the states predicted by the surrogate model $\tilde{F}$ to distinguish them from the real-environment states $\mathbf{y}_{i,t+1}$ and $\mathbf{y}_{i,t}$. Importantly, local transition functions allow us to avoid the learning of global high-dimensional mappings, which (i) would require (possibly delayed or faulty) communication to gather the local data in realistic settings, and (ii) may be computationally expensive to train and evaluate, especially when model-order reduction strategies are not taken into account. In this work, we consider a shallow neural network of parameters $\boldsymbol{\theta}_{\tilde{F}}$ to approximate the local dynamics, that is $\tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\mu}) = \tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}, \boldsymbol{\mu}; \boldsymbol{\theta}_{\tilde{F}})$. Using the data collected from the real-environment interaction, the parameters of the local forward model $\boldsymbol{\theta}_{\tilde{F}}$ are optimized by minimizing the prediction error between the samples from the environment $\mathbf{y}_{t+1}$ and the surrogate model predictions $\tilde{\mathbf{y}}_{t+1}$, that is the loss function

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}_{\tilde{F}}) &= \mathbb{E}_{\mathbf{y}_t, \mathbf{u}_t, \mathbf{y}_{t+1}, \boldsymbol{\mu}} \left[\sum_{i=1}^N \|\mathbf{y}_{i,t+1} - \tilde{\mathbf{y}}_{i,t+1}\|_2^2 \right] \\ &= \mathbb{E}_{\mathbf{y}_t, \mathbf{u}_t, \mathbf{y}_{t+1}, \boldsymbol{\mu}} \left[\sum_{i=1}^N \|\mathbf{y}_{i,t+1} - \tilde{F}(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}, \boldsymbol{\mu}; \boldsymbol{\theta}_{\tilde{F}})\|_2^2 \right]. \end{aligned}$$

The pseudo-code of the two variants of HypeMARL, namely model-free HypeMARL and MB-HypeMARL, are summarized in Algorithm 1 in Appendix D.

4. Numerical Results

In this section, we present the numerical results obtained when applying HypeMARL to four high-dimensional, parametric, and distributed control problems: two dealing with *density control* and two considering *flow control* problems with different distributed actuation strategies. It is worth highlighting that, due to the high-dimensionality of these problems in terms of states and, especially, controls, a single-agent RL algorithm is not a viable option, as shown in the Appendix. In recent years, decentralized MARL has shown promising results in controlling distributed systems governed by partial differential equations [35, 37, 36], and it will be used as a baseline in the following sections to compare with HypeMARL and MB-HypeMARL. Unlike HypeMARL, decentralized MARL does not take into account positional encoding and hypernetworks, with local policies that predict local control actions relying solely on local state values and system parameters. It is important to note that HypeMARL performs well in all the test cases presented, even though we take into account the same hyperparameter values, as described in the Appendix E, and the same training setting.

4.1. Density control

This section presents two challenging test cases in which the aim is to move an arbitrary distribution from a random initial mean position to a desired target position by optimally steering the source of a diffusion-advection PDE. In particular, inspired by [81], we consider two settings: density control in a vacuum and in a parametric fluid. While the former case does not consider an underlying transport effect, the latter deals with a fluid moving the density upwards. Density control may be of interest in several scientific fields. Indeed, density functions serve as macroscopic models of different quantities, such as, e.g., drone or wildlife swarms, traffic, and concentrations of pollutants or chemicals, to mention a few.

4.1.1. Density control in a vacuum

The first test case considers density control in a vacuum, i.e., without taking into account any underlying transport effects, such as wind or water currents, that move the density. Let $y(\mathbf{x}, t) \in \mathbb{R}$ be the state variable representing a density function in the space-time domain $\Omega \times (0, T]$, with final time $T > 0$. The state dynamics is described by the Fokker-Planck equation

$$\begin{cases} \frac{\partial y}{\partial t} - \kappa \Delta y = u & \text{in } \Omega \times (0, T] \\ \nabla y \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \times (0, T] \\ y(\mathbf{x}, 0) = y_0(\mathbf{x}; \boldsymbol{\mu}_0) & \text{in } \Omega, \end{cases} \quad (17)$$

where $\Omega = (-1, 1)^2$ is the spatial domain, $\kappa = 0.001$ is the diffusion coefficient, $\mathbf{n}$ is the outward normal unity vector to the boundary $\partial\Omega$, and $u(\mathbf{x}, t) \in [-5, 5] \subset \mathbb{R}$ is the external source term. Starting from an initial distribution in the form of a Gaussian density with fixed variance equal to 0.05 and random mean $\boldsymbol{\mu}_0 = (\mu_{0,1}, \mu_{0,2})$, that is

$$y_0(\mathbf{x}; \boldsymbol{\mu}_0) = \frac{10}{\pi} \exp(-10(x_1 - \mu_{0,1})^2 - 10(x_2 - \mu_{0,2})^2), \quad \mathbf{x} = (x_1, x_2) \in \Omega, \quad (18)$$

the density slightly diffuses over time. The goal in this setting is to move the density from a random initial position $\boldsymbol{\mu}_0 \in [-0.75, 0] \times [-0.75, 0.75]$ to a desired final destination $\boldsymbol{\mu}_T = (\mu_{T,1}, \mu_{T,2}) \in [0, 0.75] \times [-0.75, 0.75]$. To do so, we control the term u in Equation (17) over the whole domain Ω , which can be interpreted as a source or a sink effect. Notice that only the target destination selected by the controller is regarded as a system parameter, i.e. $\boldsymbol{\mu} = \boldsymbol{\mu}_T$, while the initial position $\boldsymbol{\mu}_0$ is implicitly captured by the local states observed in the algorithm. Notice also that, after discretizing the spatial domain with a regular mesh, the dimensionality of the state and the control is equal to $N_y = N_u = 1089$, thus resulting in a high-dimensional and distributed control problem. In addition, to solve the Fokker-Planck equation with the finite element method, we utilize a uniform time discretization of the interval $[0, T]$ with time step $\Delta t = 0.1s$ and final time $T = 1s$.

We employ MARL, HypeMARL and MB-HypeMARL to learn a parametric policy capable of controlling this challenging high-dimensional, parametric, and distributed setting. The local rewards $r_{i,t}$ maximized while training the local agents are defined as

$$r_{i,t} = r(y_{i,t}, u_{i,t}; \boldsymbol{\mu}) = -(y_{i,t} - y_{i,T}(\boldsymbol{\mu}))^2, \quad (19)$$

where $y_{i,T}(\boldsymbol{\mu}) = y_T(\mathbf{x}; \boldsymbol{\mu})$ corresponds to the local values of the target Gaussian density

$$y_T(\mathbf{x}; \boldsymbol{\mu}_T) = \frac{10}{\pi} \exp(-10(x_1 - \mu_{T,1})^2 - 10(x_2 - \mu_{T,2})^2),$$

centered at the final target destination. Figure 3 reports the average local rewards during training and evaluation of the different MARL algorithms with respect to the number of training episodes. In particular, we display the median and interquartile range of 5 independent runs with 5 different seeds for a fair and robust comparison. We train the algorithms for a total of 500 episodes, exploring the action space with additive Gaussian noise, which linearly decays after the first 25 warm-up interactions, on the actions predicted by the policy. Regarding the evaluations, we test the policy during training every 50 episodes, and we evaluate the optimal policy after training on 5 different combinations of initial positions $\boldsymbol{\mu}_0$ and final destination $\boldsymbol{\mu} = \boldsymbol{\mu}_T$. When considering MB-HypeMARL, to highlight the data efficiency with respect to model-free algorithms, we only report the training rewards collected when interacting with the environment, leaving out the interactions with the surrogate environment. In particular, after the first 25 episodes of warm-up, MB-HypeMARL performs 10 episodes of interaction with the surrogate environment for every episode of interaction with the real environment. Both HypeMARL and MB-HypeMARL outperform MARL in terms of expected cumulative reward during training and evaluation. In addition, while 500 environment interactions are required to optimally steer the system with HypeMARL, MB-HypeMARL is capable of learning effective policies with only 73 environment interactions.

Figure 4, instead, displays three examples of controlled solutions at two different time instants, namely $t = 0.0s$ and $t = 0.5s$, obtained after training HypeMARL, MB-HypeMARL, and MARL when dealing with three different random initial positions $\boldsymbol{\mu}_0$ and target destinations $\boldsymbol{\mu} = \boldsymbol{\mu}_T$. Although all algorithms are capable of suppressing the initial density around $\boldsymbol{\mu}_0$, only HypeMARL and MB-HypeMARL are able to make it move around $\boldsymbol{\mu}_T$. The difficulties of MARL are intrinsic to the fully decentralized nature of the algorithm. Indeed, MARL policies have no way of differentiating states with similar values but located in different positions. As soon as the density is suppressed, all the local states show similar values (≈ 0) and, without any knowledge of their own position, the agents are not capable of making the distribution appear in the target location. Thanks to positional encoding, the local agents of HypeMARL are aware of their (relative) position with respect to the others and can effectively solve this challenging task. These results show the importance of developing coordination among agents even in decentralized settings.

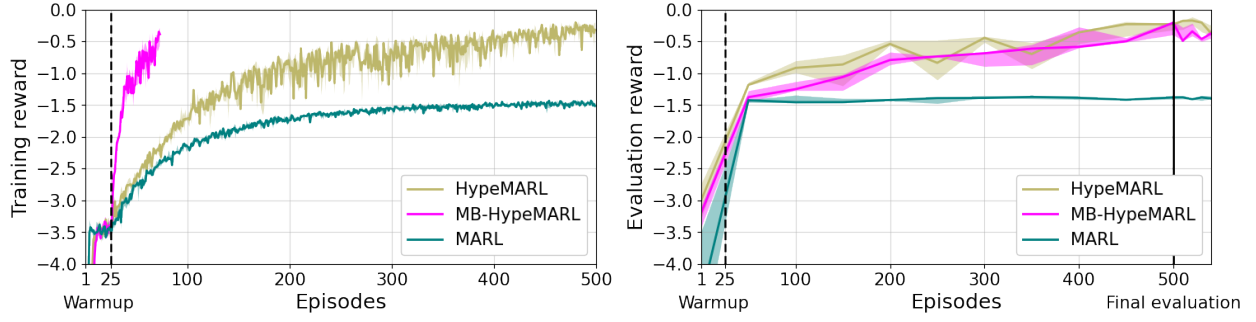


Figure 3. *Density control in a vacuum.* Median and interquartile range of HypeMARL, MB-HypeMARL and MARL training and evaluation rewards.

4.1.2. Density control in a fluid

In this second test case, differently from the setting presented in the previous section, we consider a circular obstacle with radius 0.15 in the middle of the square $(-1, 1)^2$, that is $\Omega = (-1, 1)^2 \setminus \mathbb{B}_{0.15}(0, 0)$. Moreover, we include a parametric transport term in Equation (17), that is, we deal with the following system

$$\begin{cases} \frac{\partial y}{\partial t} + \nabla \cdot (-\kappa \nabla y + \mathbf{v}(\alpha)y) = u & \text{in } \Omega \times (0, T] \\ (-\kappa \nabla y + \mathbf{v}(\alpha)y) \cdot \mathbf{n} = 0 & \text{on } \partial\Omega \times (0, T] \\ y(\mathbf{x}, 0) = y_0(\mathbf{x}; \boldsymbol{\mu}_0) & \text{in } \Omega \end{cases} \quad (20)$$

where $\kappa = 0.001$ is the diffusion coefficient, $\mathbf{n}$ is the outward normal vector to the domain boundary, and $u(\mathbf{x}, t) \in [-5, 5] \subset \mathbb{R}$ is the distributed control. The advection $\mathbf{v}(\mathbf{x}, t; \alpha)$ models an underlying fluid flow representing, for instance, a wind or a water current. The fluid enters the domain from the bottom boundary $\Gamma_{\text{in}} = \partial\Omega \cap \{x_2 = -1\}$, and moves the density upwards according to the steady Navier-Stokes equations

$$\begin{cases} -\nu \Delta \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p = \mathbf{0} & \text{in } \Omega \\ \nabla \cdot \mathbf{v} = 0 & \text{in } \Omega \\ \mathbf{v} = \mathbf{0} & \text{on } \Gamma_{\text{obs}} \\ \mathbf{v} = \mathbf{v}_{\text{in}}(\alpha) & \text{on } \Gamma_{\text{in}} \\ \mathbf{v} \cdot \mathbf{n} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} \cdot \mathbf{t} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} = \mathbf{0} & \text{on } \Gamma_{\text{out}}, \end{cases} \quad (21)$$

where $\nu = 0.1$ is the kinematic viscosity, $p(\mathbf{x}, t) \in \mathbb{R}$ is the pressure field, $\mathbf{t}$ is the tangential vector to the boundary $\partial\Omega$, $\Gamma_{\text{out}} = \partial\Omega \cap \{x_2 = 1\}$, $\Gamma_{\text{wall}} = \partial\Omega \cap \{x_1 = \pm 1\}$, and Γ_{obs} denotes the obstacle boundary. The inflow Dirichlet boundary datum is defined as

$$\mathbf{v}_{\text{in}} = [(x_1 + 1)(x_1 - 1) \sin(\alpha), \cos(\alpha)]^\top,$$

where the angle of attack $\alpha \in [-1, 1]\text{rad}$ is regarded as an additional system parameter, that is $\boldsymbol{\mu} = [\mu_{T,1}, \mu_{T,2}, \alpha]$, while the parabolic profile $(x_1 + 1)(1 - x_1)$ avoids singularities at the domain corners.

To solve the full-order model of Equations (20) and (21) through the finite element method, we introduce a spatial discretization of the domain, resulting in state and control dimensions $N_y = N_u = 1638$, and a uniform time discretization of the interval $[0, T]$ with $\Delta t = 0.1\text{s}$ and final time $T = 1\text{s}$. Similarly to the test case detailed in the previous section, the goal is to move the density from a random initial position $\boldsymbol{\mu}_0 = [\mu_{0,1}, \mu_{0,2}] \in [-0.75, -0.25] \times [-0.75, 0.75]$ towards the parametric final destination $\boldsymbol{\mu}_T = [\mu_{T,1}, \mu_{T,2}] \in [0.25, 0.75] \times [-0.75, 0.75]$, maximizing the local rewards defined in Equation (19).

Figure 5 shows the median and interquartile range of training and evaluation rewards obtained with 5 independent runs with 5 different seeds. Similarly to the previous test case, HypeMARL and MB-HypeMARL

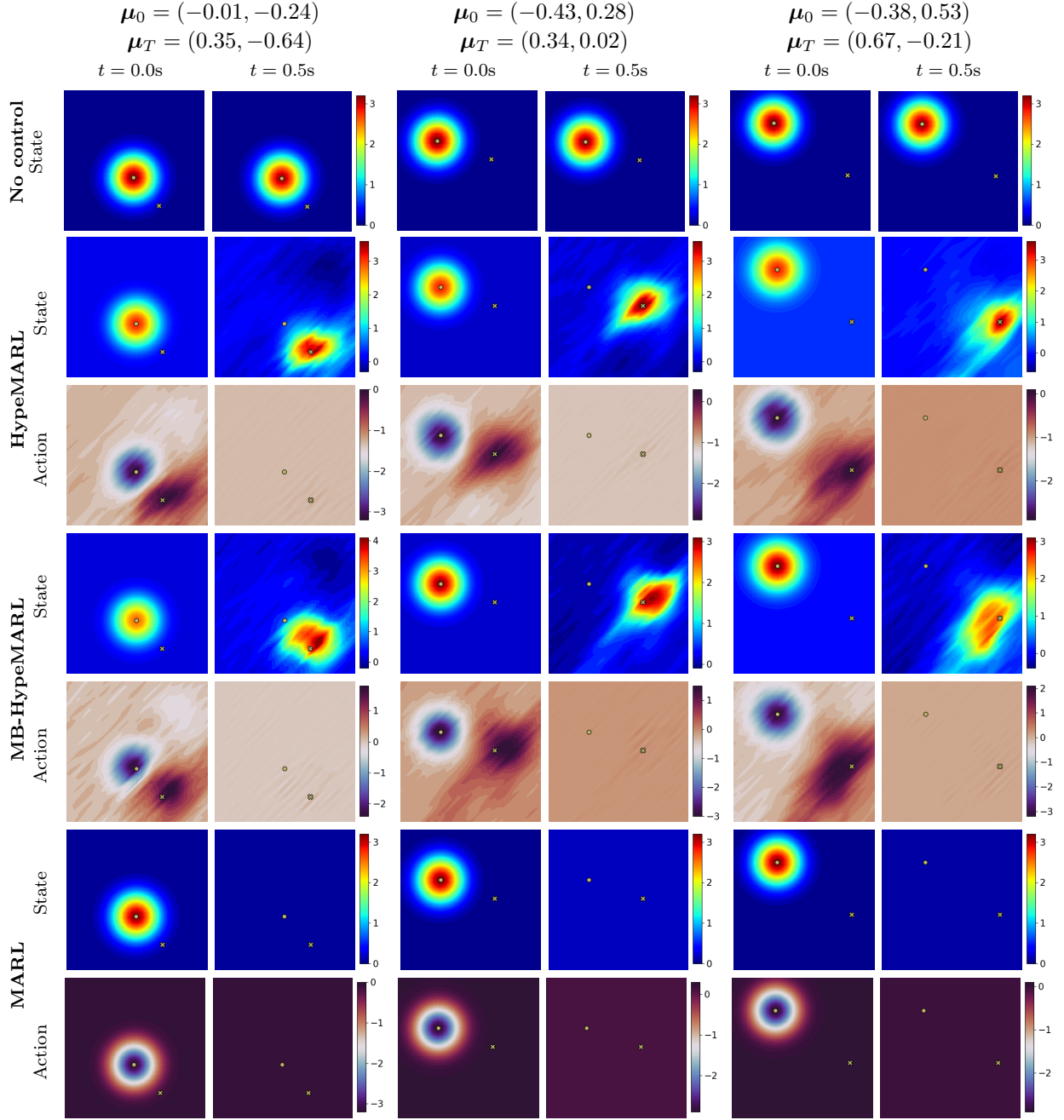


Figure 4. *Density control in a vacuum.* Uncontrolled states (first row), controlled states and optimal actions obtained with HypeMARL (second and third row), MB-HypeMARL (fourth and fifth row) and MARL (sixth and seventh row) over time in three different random parametric settings.

outperform MARL with remarkably higher rewards. Indeed, as visible in Figure 6 showing three examples of controlled solutions over time, all control methods are able to suppress the initial density. However, only HypeMARL and MB-HypeMARL are capable of controlling the source term to make the density appear at the final desired position. Notice that, due to the presence of the underlying fluid, after moving the density to the target location, the agents have to maintain the density in place, counterbalancing the parametric transport effect. Thanks to an accurate deep learning-based surrogate model approximating the local dynamics, which represents a very limited additional complexity to the algorithm, it is possible to limit the real environment interactions with the model-based algorithm MB-HypeMARL, thus resulting in a faster training procedure, with minimal deterioration of the control performance.

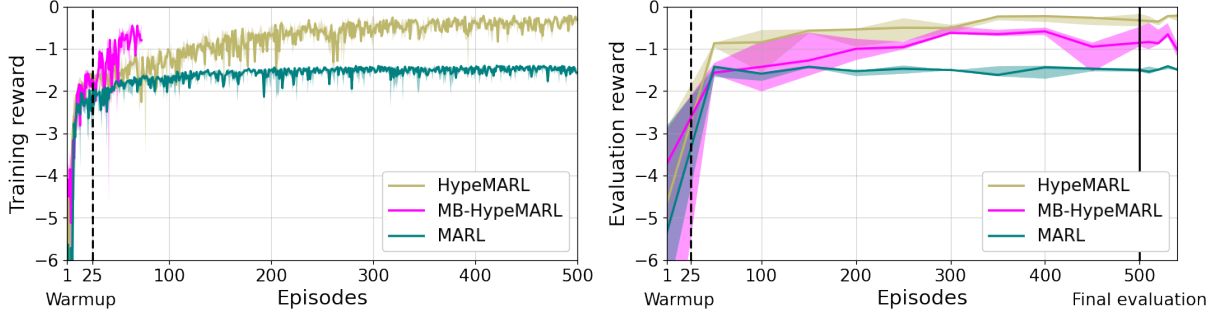


Figure 5. *Density control in a fluid.* Median and interquartile range of HypeMARL, MB-HypeMARL and MARL training and evaluation rewards.

4.2. Flow control

The second class of applications considered deals with the control of a flow around an object exploiting different actuation strategies. In particular, the first setting considers vertical jets, which blow and inject fluid on the top and bottom sides of the object to maximize (minimize) the pressure below (above) the obstacle. In the second scenario, the controller actively changes the obstacle's shape to minimize the pressure at the front of the obstacle.

4.2.1. Flow control with vertical jets

In this section we test the MARL algorithms to steer an incompressible fluid flow around an object. We consider a rectangular channel $\Omega = (0, 10) \times (0, 2) \setminus \Omega_{\text{obs}}$ with an inner obstacle consisting of the union of a rectangle and two semicircles, that is, $\Omega_{\text{obs}} = \Omega_l \cup \Omega_c \cup \Omega_r$ where $\Omega_l = \mathbb{B}_{0.25}(0.75, 1.00)$, $\Omega_c = [0.75, 1.25] \times [0.75, 1.25]$ and $\Omega_r = \mathbb{B}_{0.25}(1.25, 1.00)$. The fluid dynamics in terms of velocity $\mathbf{v}(\mathbf{x}, t) \in \mathbb{R}^2$ and pressure $p(\mathbf{x}, t) \in \mathbb{R}$ is described by the unsteady Navier-Stokes equations

$$\begin{cases} \frac{\partial \mathbf{v}}{\partial t} - \nu \Delta \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p = \mathbf{0} & \text{in } \Omega \times (0, T] \\ \nabla \cdot \mathbf{v} = 0 & \text{in } \Omega \times (0, T] \\ \mathbf{v} = \mathbf{0} & \text{on } \Gamma_l \cup \Gamma_r \\ \mathbf{v} = u \mathbf{n} & \text{on } \Gamma_c \\ \mathbf{v} = \mathbf{v}_{\text{in}}(\alpha) & \text{on } \Gamma_{\text{in}} \\ \mathbf{v} \cdot \mathbf{n} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} \cdot \mathbf{t} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} = \mathbf{0} & \text{on } \Gamma_{\text{out}} \\ \mathbf{v}(\mathbf{x}, 0) = \mathbf{0} & \text{in } \Omega, \end{cases} \quad (22)$$

where $\nu = 0.01$ is the kinematic viscosity, $\mathbf{n}$ and $\mathbf{t}$ are the normal and tangential versors to the domain boundary, $\Gamma_{\text{in}} = \partial\Omega \cap \{x_1 = 0\}$ is the inflow boundary, $\Gamma_{\text{out}} = \partial\Omega \cap \{x_1 = 10\}$ is the outflow boundary, $\Gamma_{\text{wall}} = \partial\Omega \cap (\{x_2 = 0\} \cup \{x_2 = 2\})$ is the wall, $\Gamma_c = \partial\Omega_{\text{obs}} \cap \{0.75 < x_1 < 1.25\}$ is the control boundary,

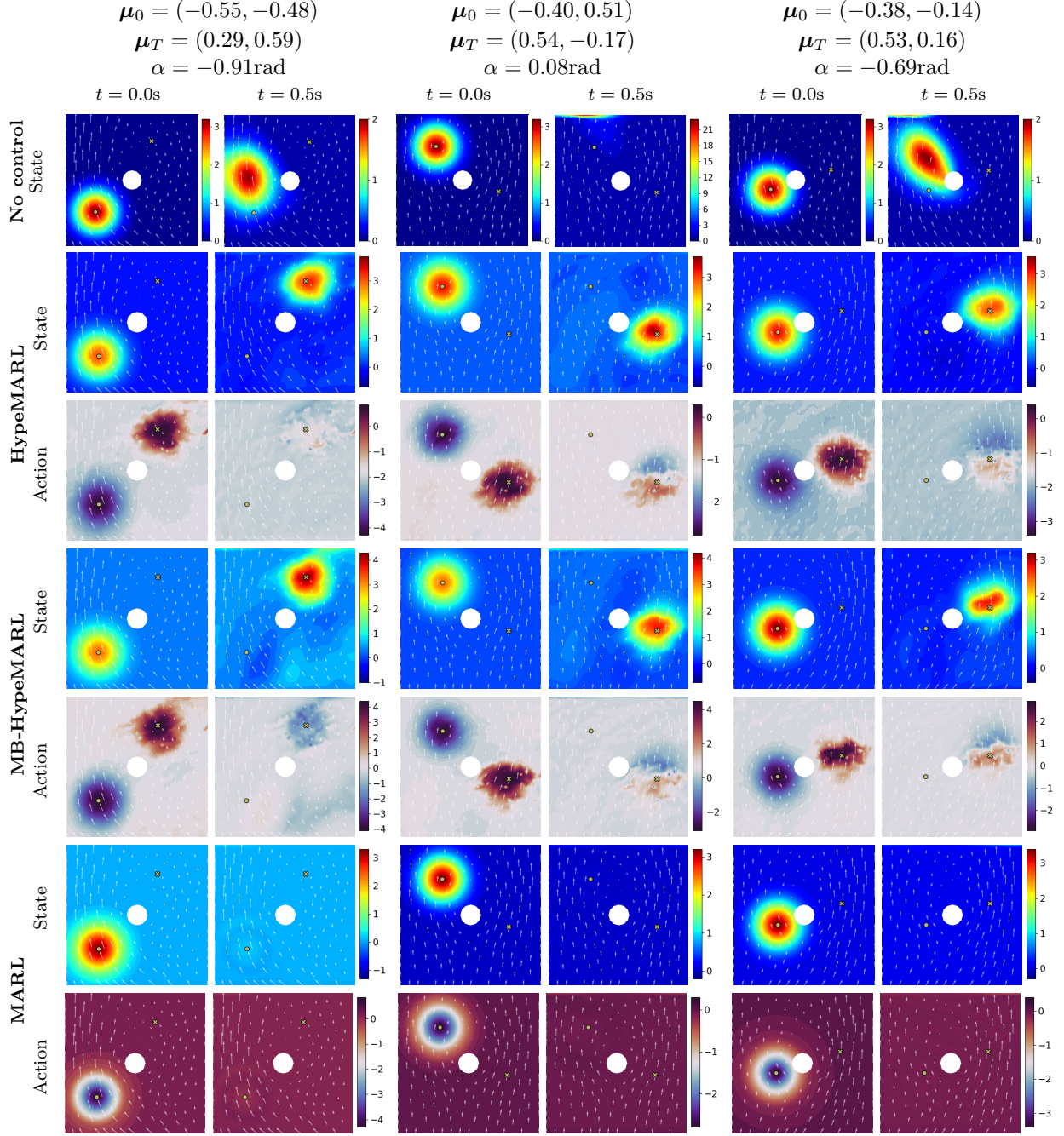


Figure 6. *Density control in a fluid.* Uncontrolled states (first row), controlled states and optimal actions obtained with HypeMARL (second and third row), MB-HypeMARL (fourth and fifth row) and MARL (sixth and seventh row) over time in three different random parametric settings.

while $\Gamma_l = \partial\Omega_{\text{obs}} \cap \{x_1 \leq 0.75\}$ and $\Gamma_r = \partial\Omega_{\text{obs}} \cap \{x_1 \geq 1.25\}$ are the rounded obstacle boundaries. To steer the fluid flow, we consider vertical jets on Γ_c with intensities $u(\mathbf{x}, t) \in [-1, 1] \subset \mathbb{R}$. Moreover, we take into account an inflow velocity defined as

$$\mathbf{v}_{\text{in}}(\mathbf{x}, t; \alpha) = [10x_2(2 - x_2) \cos(\alpha), 10 \sin(\alpha)]^\top,$$

where the angle of attack $\alpha \in [-1, 1]\text{rad}$ is regarded as a system parameter, i.e. $\mu = \alpha$. The Reynolds number for this test case is equal to 1000. To solve Equation (22) with the finite element method, we discretize the spatial domain, resulting in a velocity and pressure dimension equal to, respectively, $N_v = 100600$ and $N_p = 12712$. Moreover, we consider a uniform time grid in the interval $[0, T]$ with time step $\Delta t = 0.1\text{s}$ and final time $T = 5.5\text{s}$.

We now consider MARL, HypeMARL and MB-HypeMARL to control this challenging parametric and high-dimensional problem. Differently from the density control test cases where the state and the actions are observed and predicted over the whole domain, the vertical jets are defined on the obstacle boundary Γ_c . Therefore, starting from the $N_y = 74$ local pressure values $y_{i,t} = p_{i,t} = p(\mathbf{x}_i, t)$, with $\mathbf{x}_i = (x_{i,1}, x_{i,2}) \in \Gamma_c$, MARL algorithms have to predict the optimal intensities of the corresponding $N_u = 74$ jets $u_{i,t}$ in the same locations in order to maximize the local rewards

$$r_{i,t} = r(p_{i,t}, u_{i,t}) = \begin{cases} p_{i,t} & \text{if } x_{i,2} = 0.75 \\ -p_{i,t} & \text{if } x_{i,2} = 1.25 \end{cases}.$$

Specifically, we aim at maximizing the pressure values at the bottom boundary of the obstacle, while minimizing the local pressure on top. Notice that the boundary control is applied after letting the uncontrolled system evolve for 0.5s, and it is updated by the control methods every 0.5s.

The first row of Figure 7 reports the average local rewards in training and evaluation of the different competing algorithms with respect to the number of training episodes. The results show the median and interquartile range of 5 independent runs with 5 different seeds for a fair and robust comparison. We train the algorithms for a total of 250 episodes, exploring the action space with additive Gaussian random noise, which linearly decays after the first 25 episodes of warm-up, on the actions predicted by the policy. To highlight the data efficiency of MB-HypeMARL, we only report the training rewards collected when interacting with the environment, while we leave out the interactions with the surrogate one. After the first 25 warm-up interactions with the real environment, MB-HypeMARL performs 10 episodes of interaction with the surrogate environment for every episode of interaction with the real one. The agents are evaluated once every 25 training episodes. HypeMARL and MB-HypeMARL outperform classical decentralized MARL with remarkably higher rewards. Moreover, the model-based version is extremely data efficient and is capable of achieving effective control strategies with only 48 real environment interactions. Besides the evaluation rewards, the left panel in the second row of Figure 7 provides the evaluation lift of the different control strategies. It is important to note that, due to the high correlation of the selected reward with the lift force, HypeMARL and MB-HypeMARL provide higher lift values compared to the classical decentralized MARL. Moreover, the right panel in the second row of Figure 7 compares the reward of the different competing methods across different angles of attack in the range $[-1, 1]\text{rad}$ with respect to the uncontrolled setting. In particular, it is possible to further assess the effectiveness of the proposed algorithms in all the parametric scenarios considered and the unsatisfactory results of classical decentralized MARL.

Figure 8 displays, instead, three examples of controlled solutions obtained with the different approaches for as many random angles of attack $\mu = \alpha$. Specifically, each subfigure shows the vertical jets with the velocity streamlines around the obstacle (top left), the pressure around the obstacle (top right), and the flow velocity (bottom).

4.2.2. Flow control with shape morphing

This last test case aims to control the flow around an object by morphing the front shape of the object itself. Similarly to the previous test case, we consider a fluid flow in the domain $\Omega(\mathbf{d}) = (0, 10) \times (0, 2) \setminus \Omega_{\text{obs}}(\mathbf{d})$. Starting from the shape introduced in the previous section, the obstacle $\Omega_{\text{obs}}(\mathbf{d}) = \Omega_l(\mathbf{d}) \cup \Omega_c \cup \Omega_r$ can now modify the round left boundary $\Gamma_l(\mathbf{d}) = \partial\Omega_{\text{obs}}(\mathbf{d}) \cap \{x_1 \leq 0.75\}$ according to a deformation field $\mathbf{d}(\mathbf{x}, t) = d(\mathbf{x}, t)\mathbf{n}_{\text{ref}}$, with intensity $d(\mathbf{x}, t) \in [-0.1, 0.1]$ and direction $\mathbf{n}_{\text{ref}}$, that is the normal vector to

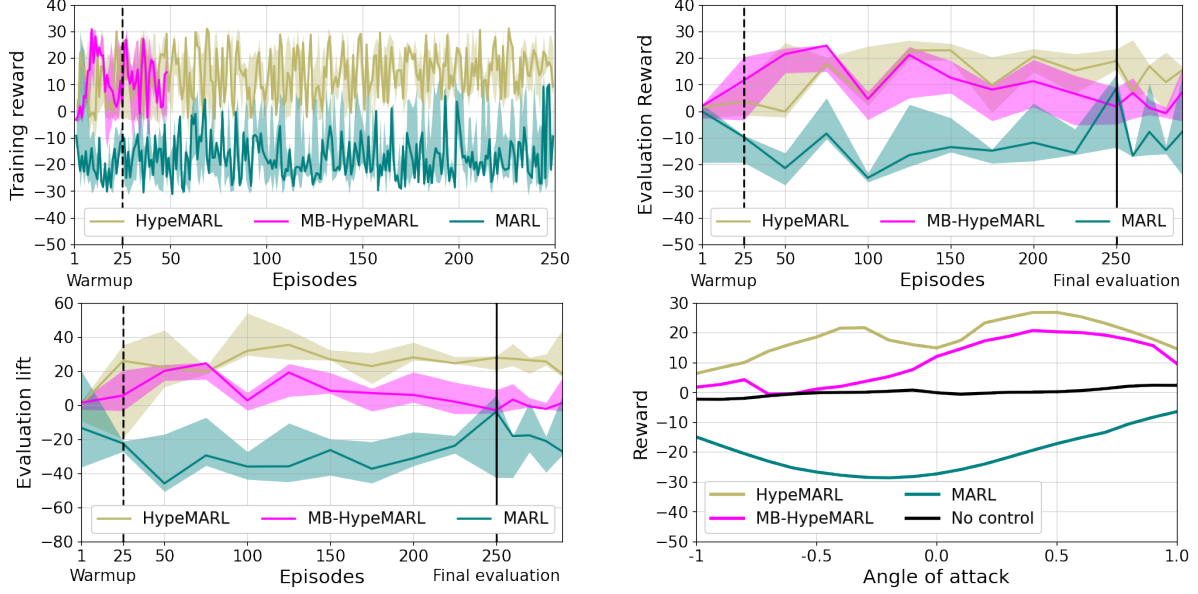


Figure 7. *Flow control with vertical jets.* First row: median and interquartile range of HypeMARL, MB-HypeMARL and MARL training end evaluation rewards. Second row: median and interquartile range of HypeMARL, MB-HypeMARL and MARL evaluation lifts (left) and reward obtained with HypeMARL, MB-HypeMARL, MARL and no control for different angles of attack (right).

the initial boundary with no deformation. The flow dynamics is modeled by the unsteady Navier-Stokes equations

$$\begin{cases} \frac{\partial \mathbf{v}}{\partial t} - \nu \Delta \mathbf{v} + (\mathbf{v} \cdot \nabla) \mathbf{v} + \nabla p = \mathbf{0} & \text{in } \Omega(\mathbf{d}) \times (0, T] \\ \nabla \cdot \mathbf{v} = 0 & \text{in } \Omega(\mathbf{d}) \times (0, T] \\ \mathbf{v} = \mathbf{0} & \text{on } \Gamma_c \cup \Gamma_r \\ \mathbf{v} = u \mathbf{n}_{\text{ref}} & \text{on } \Gamma_l(\mathbf{d}) \\ \mathbf{v} = \mathbf{v}_{\text{in}}(\alpha) & \text{on } \Gamma_{\text{in}} \\ \mathbf{v} \cdot \mathbf{n} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} \cdot \mathbf{t} = 0 & \text{on } \Gamma_{\text{walls}} \\ (\nu \nabla \mathbf{v} - p) \mathbf{n} = \mathbf{0} & \text{on } \Gamma_{\text{out}} \\ \mathbf{v}(\mathbf{x}, 0) = \mathbf{0} & \text{in } \Omega(\mathbf{d}), \end{cases} \quad (23)$$

with $\nu = 0.01$ the kinematic viscosity, and $u(\mathbf{x}, t) \in [-0.01, 0.01]$ the incremental deformation. Notice that, by discretizing the time interval $[0, T]$ with a uniform grid $t = 1, \dots, T_{\text{max}}$, we have $d(\mathbf{x}, \tau) = \sum_{t=1}^{\tau} u(\mathbf{x}, t)$ for every $\tau = 1, \dots, T_{\text{max}}$. After observing the local pressure values $y_{i,t} = p_{i,t}$ at $N_y = 37$ points on $\Gamma_l(\mathbf{d})$, the controllers are allowed to deform the front part of the obstacle every 0.5s by applying a deformation $u_{i,t}$ in as many boundary points ($N_u = 37$). In this setting, the goal is to minimize the local pressure in front of the obstacle, that is

$$r_{i,t} = r(p_{i,t}, u_{i,t}) = -p_{i,t}.$$

See the previous section for further details on domain boundaries, space-time discretization and inflow velocity. Similarly to the previous test case, the angle of attack $\alpha \in [-1, 1]$ rad is regarded as a system parameter, i.e. $\mu = \alpha$, and the Reynolds number ranges from 900 to 1100, according to the obstacle deformation.

The first row of Figure 9 presents the median and interquartile range of training and evaluation rewards obtained with 5 independent runs with 5 different seeds. HypeMARL and MB-HypeMARL agents outperform MARL with higher training and evaluation rewards, and thus more effective morphing strategies. Moreover, as visible in the left panel in the second row of Figure 9, despite the simplicity of the reward function,

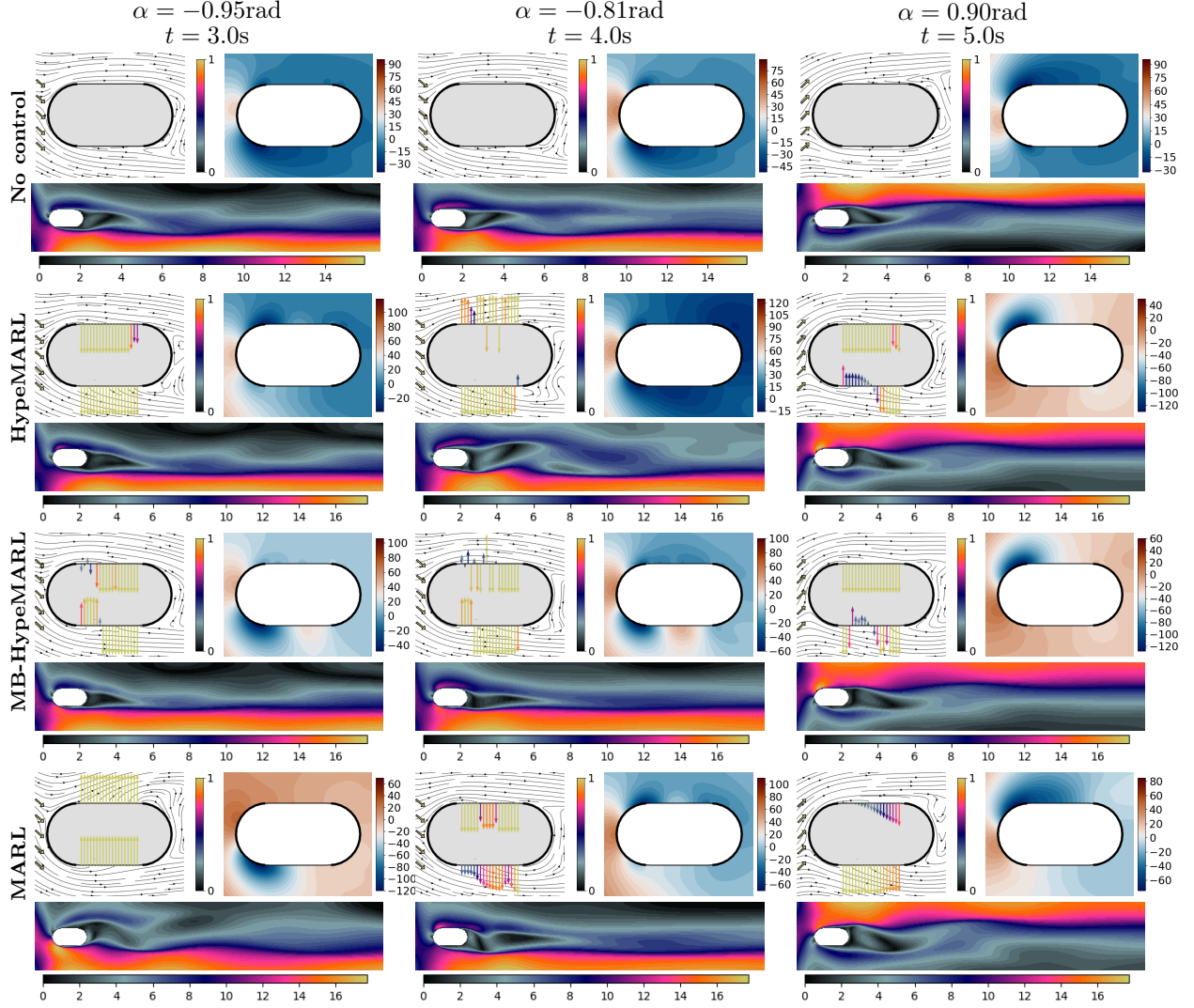


Figure 8. Flow control with vertical jets. Uncontrolled dynamics (first row), controlled dynamics obtained with HypeMARL (second row), MB-HypeMARL (third row) and MARL (fourth row) in three different random parametric settings. Every panel presents the vertical jets applied with the velocity streamlines around the obstacle (top left), the pressure around the obstacle (top right), and the flow velocity norm (bottom).

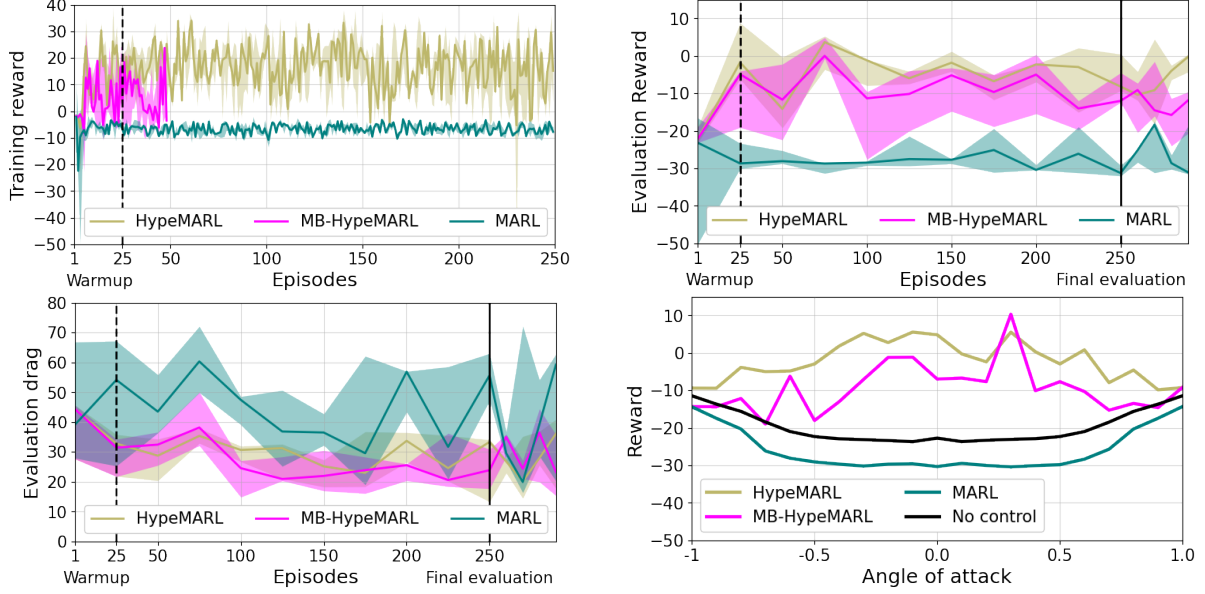


Figure 9. *Flow control with shape morphing.* First row: median and interquartile range of HypeMARL, MB-HypeMARL and MARL training end evaluation rewards. Second row: median and interquartile range of HypeMARL, MB-HypeMARL and MARL evaluation drags (left) and reward obtained with HypeMARL, MB-HypeMARL, MARL and no control for different angles of attack (right).

HypeMARL and MB-HypeMARL indirectly allow for smaller drag values across different parametric settings with respect to classical decentralized MARL. The right panel in the second row of Figure 9 shows the rewards of the different competing models for different angles of attack in the range $[-1, 1]$ rad, along with the ones in the uncontrolled setting.

Figure 10 shows three examples of shape morphing when facing as many values of the angle of attack, i.e. $\alpha = \{-0.5, 0.0, 0.5\}$ rad. Specifically, it is possible to assess that HypeMARL and MB-HypeMARL are able to learn non-trivial narrow front shapes, allowing for local pressure minimization. Moreover, the resulting shapes clearly exploit the parametric knowledge to adapt the front orientation with respect to the different angles of attack α . Conversely, MARL is only capable of finding a trivial shape corresponding to the maximum displacement allowed, without any adaptation corresponding to the different parameter values. Similarly to the previous test cases, MB-HypeMARL is again capable of reducing the number of real environment interactions, with a minimal performance deterioration with respect to HypeMARL.

While the results obtained in this very challenging example by HypeMARL and MB-HypeMARL are interesting and promising, as they show a collective and coordinated behavior of the agents, it is worth highlighting that the shapes found are not the optimal shapes usually obtained in the context of shape optimization in fluid flows. Improving the shape smoothness requires higher degrees of freedom in the boundary displacements. In our experiments, we only consider radial displacements of the front part of the object, limiting the shapes that can be obtained. Moreover, feasibility constraints should be taken into account to guarantee physical and geometrical admissibility of the resulting shape. In addition, the obstacle aerodynamics could be enhanced by maximizing global reward metrics, such as promoting drag reduction or penalizing non-smooth shapes. However, the maximization of global reward functions in decentralized settings is extremely challenging, and it will be the subject of future works, as highlighted in the Discussion section.

5. Discussion and conclusion

In this work, we propose a novel decentralized MARL approach, named HypeMARL, that allows for solving challenging high-dimensional, parametric, and distributed control problems. HypeMARL relies on (i) sinusoidal positional encoding of the agents' relative position to enhance spatial awareness and coordination

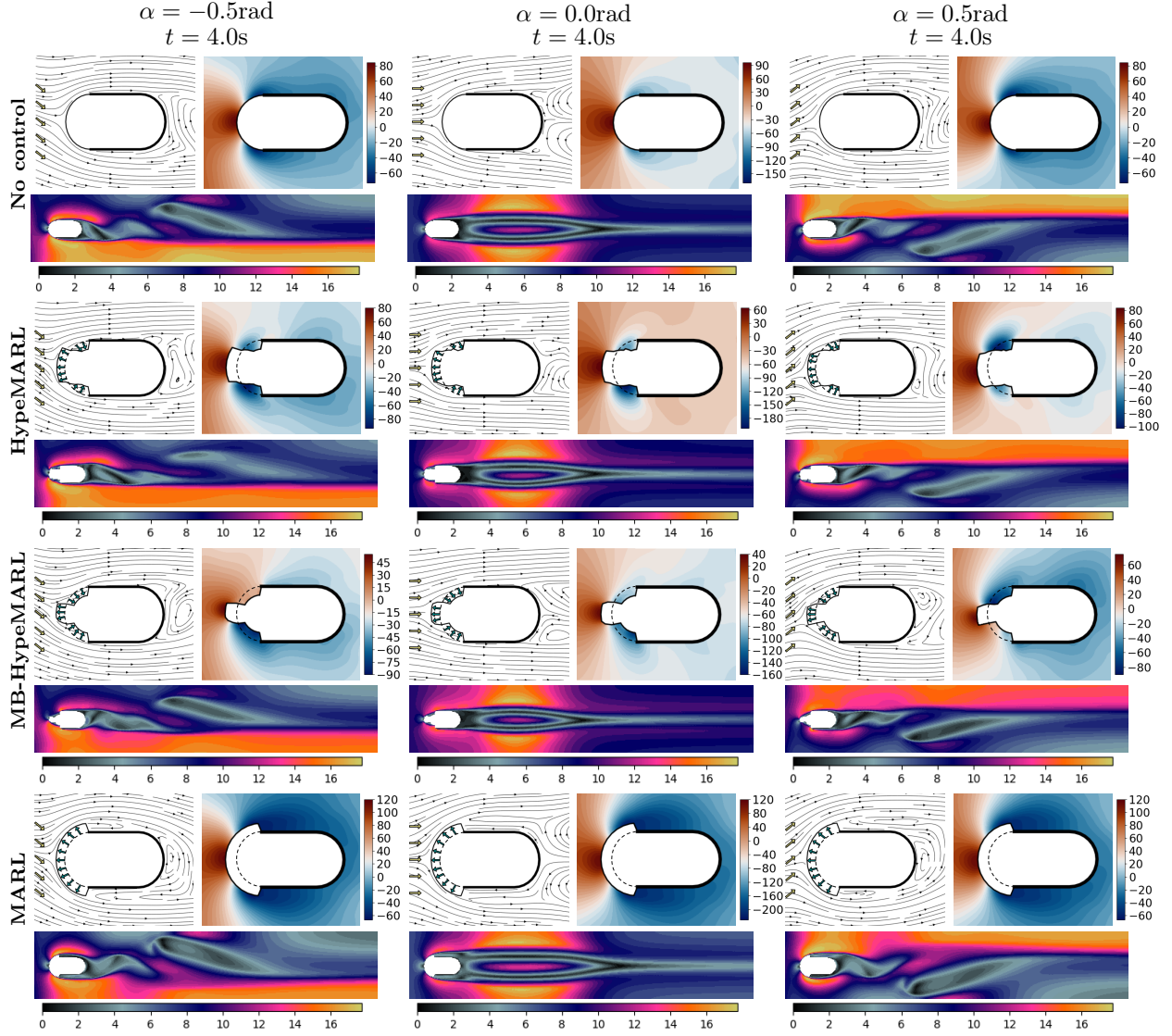


Figure 10. Flow control with shape morphing. Uncontrolled dynamics (first row), controlled dynamics obtained with HypeMARL (second row), MB-HypeMARL (third row) and MARL (fourth row) in three different parametric settings. Every panel presents the deformed obstacle shape with the velocity streamlines around the obstacle (top left), the pressure around the obstacle (top right), and the flow velocity norm (bottom).

among the agents, and (ii) hypernetworks to effectively parametrize the control policies as functions of system parameters and positional information. Moreover, we propose a model-based version of the algorithm, named MB-HypeMARL, allowing for a significant reduction of interactions with the real environment (by a factor ~ 10) thanks to a cheap deep learning-based surrogate model approximating local dynamics.

By means of several numerical experiments dealing with two challenging classes of high-dimensional control problems, namely density and flow control, we show that both HypeMARL and MB-HypeMARL can effectively solve problems that the state-of-the-art decentralized MARL approach cannot. Specifically, our results demonstrate that hypernetworks combined with the sinusoidal positional embedding of the agents' (relative) position allow for strategic coordination strategies among the agents by only relying on local state and reward information. The model-based algorithm MB-HypeMARL is especially appealing for applications where only a limited amount of data is available, as the agents rely on surrogate models approximating the local dynamics to reduce the need for real systems interactions. These local surrogate models are modeled through shallow neural networks with a small number of learnable parameters and are learned online from the interaction data collected by the policies. Despite their simplicity, these models show impressive accuracy

even when trained with limited data, and opening new doors towards scaling MARL to complex real-world engineering systems. Future research will further investigate the architectural design of these models, their predictive capabilities over long time horizons, and their ability to generalize to real-world scenarios.

In particular, the control of density functions is extremely relevant in several scientific fields. For example, a density function can represent a robot swarm and its optimal control can have impact in security, survey, monitoring, and surveillance applications [82]. Density functions can also model wildlife, or traffic flows. In all these cases, developing (decentralized) algorithms that can achieve coordination among the agents and collective behavior while scaling to large number of agents, such as the ones learned by HypeMARL and MB-HypeMARL, is essential for solving complex problems. While the high-dimensionality of the states is not a limiting factor for state-of-the-art deep reinforcement learning algorithms, dealing with large actions spaces still remains an open challenge.

The ability to manipulate a flow field actively is one of the most important technological challenges of the present and past decades. Flow control involves transition delay, separation postponement, lift enhancement, turbulence augmentation, noise suppression, and drag reduction [83]. Drag reduction is especially critical for reducing fuel consumption of land, air, and sea vehicles and emissions. However, actuation strategies such as jets, or morphing structures typically require coordination among a large number of distributed actuators and, consequently, control algorithms capable of dealing with high-dimensional control problems. In addition, collecting data from real-world flow experiments, for example in a wind tunnel, is extremely costly and tedious. At the same time, full-order models capable of simulating fluid flows or fluid-structure interactions are computationally expensive. Data-efficient approaches as HypeMARL and MB-HypeMARL can lay the foundations towards the next generation of energy-efficient vehicles through active flow control.

The underlying assumption of decentralized MARL approaches is that the maximization of the cumulative local rewards corresponds to the maximization of a global task-specific metric. Although it is feasible to design such reward functions for various problems, it is not always straightforward. Indeed, local rewards and global task-specific metrics may be conflicting incentives due to the effect of the agents’ interactions. Moreover, the credit assignment problem may arise whenever local rewards are uninformative about global performance. While HypeMARL and MB-HypeMARL prioritize collective behavior of the agents over immediate local payoff, the direct maximization of global reward functions will be the subject of future works. Another interesting future direction is the learning of local reward functions corresponding to global reward metrics from given expert demonstrations, in the direction of Inverse RL [84] and imitation learning [85].

Code availability

The implementation of the proposed methodology and the code necessary to replicate the test cases detailed in the Results section are publicly available in the HypeMARL repository at <https://github.com/nicob15/HypeMARL>.

Acknowledgements

NB and AM acknowledge the Project “Reduced Order Modeling and Deep Learning for the real-time approximation of PDEs (DREAM)” (Starting Grant No. FIS00003154), funded by the Italian Science Fund (FIS) - Ministero dell’Università e della Ricerca. AM also acknowledges the project “Dipartimento di Eccellenza” 2023-2027 funded by MUR.

References

- [1] Jan H van Schuppen, Olivier Boutin, Pia L Kempker, Jan Komenda, Tomáš Masopust, Nicola Pambakian, and André CM Ran. Control of distributed systems: Tutorial and overview. *European Journal of Control*, 17(5-6):579–602, 2011.
- [2] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. *Innovations in multi-agent systems and applications-1*, pages 183–221, 2010.
- [3] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

- [4] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.
- [5] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.
- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- [7] Deheng Ye, Zhao Liu, Mingfei Sun, Bei Shi, Peilin Zhao, Hao Wu, Hongsheng Yu, Shaojie Yang, Xipeng Wu, Qingwei Guo, et al. Mastering complex control in moba games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6672–6679, 2020.
- [8] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.
- [9] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.
- [10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
- [11] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.
- [12] Long-Ji Lin. *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.
- [13] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.
- [14] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.
- [15] Fangyi Zhang, Jürgen Leitner, Michael Milford, Ben Upcroft, and Peter Corke. Towards vision-based deep reinforcement learning for robotic motion control. In *Australasian Conference on Robotics and Automation 2015*. Australian Robotics and Automation Association (ARAA), 2015.
- [16] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.
- [17] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*, pages 737–744. IEEE, 2020.
- [18] Nicolò Botteghi, Khaled Alaa, Mannes Poel, Beril Sirmacek, Christoph Brune, Abeje Mersha, and Stefano Stramigioli. Low dimensional state representation learning with robotics priors in continuous action spaces. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 190–197. IEEE, 2021.
- [19] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926, 2021.
- [20] Sen Wang, Daoyuan Jia, and Xinshuo Weng. Deep reinforcement learning for autonomous driving. *arXiv preprint arXiv:1811.11329*, 2018.
- [21] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56, 2016.

- [22] Johanna Andrea Hurtado Sánchez, Katherine Casilimas, and Oscar Mauricio Caicedo Rendon. Deep reinforcement learning for resource management on network slicing: A survey. *Sensors*, 22(8):3031, 2022.
- [23] Michele Alessandro Bucci, Onofrio Semeraro, Alexandre Allauzen, Guillaume Wisniewski, Laurent Cordier, and Lionel Mathelin. Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A*, 475(2231):20190351, 2019.
- [24] Jean Rabault and Alexander Kuhnle. Accelerating deep reinforcement learning strategies of flow control through a multi-environment approach. *Physics of Fluids*, 31(9), 2019.
- [25] Jean Rabault, Feng Ren, Wei Zhang, Hui Tang, and Hui Xu. Deep reinforcement learning in fluid mechanics: A promising method for both active flow control and shape optimization. *Journal of Hydrodynamics*, 32:234–246, 2020.
- [26] Chengwei Xia, Junjie Zhang, Eric C Kerrigan, and Georgios Rigas. Active flow control for bluff body drag reduction using reinforcement learning with partial measurements. *Journal of Fluid Mechanics*, 981:A17, 2024.
- [27] Nicolò Botteghi and Urban Fasel. Parametric pde control with deep reinforcement learning and l0 sparse polynomial policies. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*, pages 6108–6115. IEEE, 2024.
- [28] James Orr and Ayan Dutta. Multi-agent deep reinforcement learning for multi-robot applications: A survey. *Sensors*, 23(7):3625, 2023.
- [29] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Deep reinforcement learning for swarm systems. *Journal of Machine Learning Research*, 20(54):1–31, 2019.
- [30] Haixia Peng and Xuemin Shen. Multi-agent reinforcement learning based resource management in mec- and uav-assisted vehicular networks. *IEEE Journal on Selected Areas in Communications*, 39(1):131–141, 2020.
- [31] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.
- [32] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.
- [33] Ana LC Bazzan. Opportunities for multiagent systems and multiagent reinforcement learning in traffic control. *Autonomous Agents and Multi-Agent Systems*, 18:342–375, 2009.
- [34] Marco A Wiering et al. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pages 1151–1158, 2000.
- [35] Colin Vignon, Jean Rabault, Joel Vasanth, Francisco Alcántara-Ávila, Mikael Mortensen, and Ricardo Vinuesa. Effective control of two-dimensional rayleigh–bénard convection: Invariant multi-agent reinforcement learning is all you need. *Physics of Fluids*, 35(6), 2023.
- [36] Sebastian Peitz, Jan Stenner, Vikas Chidananda, Oliver Wallscheid, Steven L Brunton, and Kunihiro Taira. Distributed control of partial differential equations using convolutional reinforcement learning. *arXiv preprint arXiv:2301.10737*, 2023.
- [37] Pol Suárez, Francisco Alcántara-Ávila, Arnau Miró, Jean Rabault, Bernat Font, Oriol Lehmkuhl, and Ricardo Vinuesa. Active flow control for drag reduction through multi-agent reinforcement learning on a turbulent cylinder at $Re = 3900$. *Flow, Turbulence and Combustion*, pages 1–25, 2025.

- [38] Pol Suárez, Francisco Alcántara-Ávila, Jean Rabault, Arnau Miró, Bernat Font, Oriol Lehmkuhl, and Ricardo Vinuesa. Flow control of three-dimensional cylinders transitioning to turbulence via multi-agent reinforcement learning. *Communications Engineering*, 4(1):113, 6 2025.
- [39] Joel Vasanth, Jean Rabault, Francisco Alcántara-Ávila, · Mikael Mortensen, and Ricardo Vinuesa. Multi-agent Reinforcement Learning for the Control of Three-Dimensional Rayleigh-Bénard Convection Multi-agent reinforcement learning · Machine learning. 115:1319–1355, 2025.
- [40] Vincent Belus, Jean Rabault, Jonathan Viquerat, Zhizhao Che, Elie Hachem, and Ulysse Reglade. Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film. *AIP Advances*, 9(12), 2019.
- [41] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.
- [42] Chao Yu, Akash Velu, Eugene Vinitzky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.
- [43] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.
- [44] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pages 2961–2970. PMLR, 2019.
- [45] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.
- [46] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.
- [47] Christian Schroeder de Witt, Bei Peng, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Deep multi-agent reinforcement learning for decentralized continuous cooperative control. *arXiv preprint arXiv:2003.06709*, 19, 2020.
- [48] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.
- [49] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In *International conference on machine learning*, pages 5571–5580. PMLR, 2018.
- [50] Ermo Wei and Sean Luke. Lenient learning in independent-learner stochastic cooperative games. *Journal of Machine Learning Research*, 17(84):1–42, 2016.
- [51] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makovychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.
- [52] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.
- [53] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

- [54] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.
- [55] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.
- [56] Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.
- [57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [58] David Ha, Andrew Dai, and Quoc V. Le. Hypernetworks, 2016.
- [59] Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *arXiv preprint arXiv:2306.06955*, 2023.
- [60] Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.
- [61] Yawei Li, Shuhang Gu, Kai Zhang, Luc Van Gool, and Radu Timofte. Dhp: Differentiable meta pruning via hypernetworks. In *European conference on computer vision*, pages 608–624. Springer, 2020.
- [62] Florian Wenzel, Jasper Snoek, Dustin Tran, and Rodolphe Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in neural information processing systems*, 33:6514–6527, 2020.
- [63] Elad Sarafian, Shai Keynan, and Sarit Kraus. Recomposing the reinforcement learning building blocks with hypernetworks. In *International Conference on Machine Learning*, pages 9301–9312. PMLR, 2021.
- [64] Jacob Beck, Matthew Thomas Jackson, Risto Vuorio, and Shimon Whiteson. Hypernetworks in meta-reinforcement learning. In *Conference on Robot Learning*, pages 1478–1487. PMLR, 2023.
- [65] Sahand Rezaei-Shoshtari, Charlotte Morissette, Francois R Hogan, Gregory Dudek, and David Meger. Hypernetworks for zero-shot transfer in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 9579–9587, 2023.
- [66] Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 799–805. IEEE, 2021.
- [67] Nicolò Botteghi, Stefania Fresca, Mengwu Guo, and Andrea Manzoni. Hyperl: Hypernetwork-based reinforcement learning for control of parametrized dynamical systems, 2026.
- [68] Filippos Christianos, Georgios Papoudakis, Muhammad A Rahman, and Stefano V Albrecht. Scaling multi-agent reinforcement learning with selective parameter sharing. In *International Conference on Machine Learning*, pages 1989–1998. PMLR, 2021.
- [69] Justin K Terry, Nathaniel Grammel, Sanghyun Son, Benjamin Black, and Aakriti Agrawal. Revisiting parameter sharing in multi-agent deep reinforcement learning. *arXiv preprint arXiv:2005.13625*, 2020.
- [70] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.
- [71] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38:287–308, 2000.

- [72] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, Joelle Pineau, et al. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018.
- [73] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- [74] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.
- [75] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.
- [76] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [77] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [78] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.
- [79] Scott Fujimoto, Wei-Di Chang, Edward Smith, Shixiang Shane Gu, Doina Precup, and David Meger. For sale: State-action representation learning for deep reinforcement learning. *Advances in neural information processing systems*, 36:61573–61624, 2023.
- [80] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.
- [81] Matteo Tomasetto, Francesco Braghin, and Andrea Manzoni. Latent feedback control of distributed systems in multiple scenarios through deep learning-based reduced order models. *Computer Methods in Applied Mechanics and Engineering*, 442:118030, 2025.
- [82] Anam Tahir, Jari Böling, Mohammad-Hashem Haghbayan, Hannu T Toivonen, and Juha Plosila. Swarms of unmanned aerial vehicles—a survey. *Journal of Industrial Information Integration*, 16:100106, 2019.
- [83] Mohamed Gad-el Hak. Flow control: The future. *Journal of aircraft*, 38(3):402–418, 2001.
- [84] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.
- [85] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.
- [86] Gidi Littwin and Lior Wolf. Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1824–1833, 2019.

Appendix A. Comparison with single-agent RL

In this section, we compare HypeMARL, MB-HypeMARL, and conventional decentralized MARL with single-agent RL. Specifically, single-agent RL focuses on learning a single global policy mapping high-dimensional states to distributed control actions. Similarly to MARL-based methods, we use the TD3 algorithm for single-agent RL. Due to the prohibitive high-dimensionality of density control problems with more than 1000 sensors and actuators, single-agent RL cannot learn effective policies, resulting in extremely

low rewards both in training and in evaluation, as visible in the first two rows of Figure A.12. The performance of single-agent RL improves when considering flow control problems, as visible in the last two rows of Figure A.12, due to a moderate dimensionality of state and action spaces, i.e., < 100 . In particular, in the case of flow control with vertical jets, the concatenation of local states and actions in single-agent RL is enough to coordinate agents and allows for discovering effective control strategies, outperforming conventional decentralized MARL. However, the hypernetwork-based positional encoding in HypeMARL allows us to learn more specialized cooperation among agents, resulting in higher reward values during both training and evaluation. Importantly, the advantage of taking into account HypeMARL is even more evident when dealing with improved control strength, such as vertical jets continuously updated over time, as visible in Figure A.11. Agents' coordination is even more crucial in the case of flow control with shape morphing. While HypeMARL and MB-HypeMARL can effectively deform the obstacle shape over time to minimize the front pressure, the non-locality in single-agent RL is not enough in this challenging context.

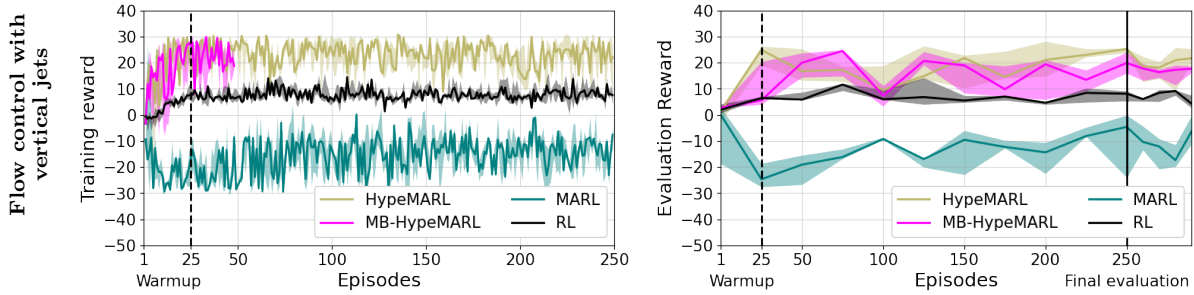


Figure A.11. Comparison with single-agent RL. Median and interquartile range of HypeMARL, MB-HypeMARL, MARL and single-agent RL training end evaluation rewards. The following test case is considered: flow control with vertical jets continuously updated over time.

Appendix B. Comparison with naive encoding strategies

In this section, we compare HypeMARL, MB-HypeMARL, and conventional decentralized MARL with the naive encoding strategy PE-MARL, where system parameters μ and positional encoding vector $\mathbf{PE}(p_i)$ are concatenated to the local state $\mathbf{y}_{i,t}$, and provided as additional input to the agent's policy and value function. Figure B.13 shows the training and evaluation rewards of the competing methods in the four test cases presented in the Results section. Despite the same pieces of information considered in input, hypernetworks outperform the naive baseline, especially when dealing with high-dimensional and distributed control problems where coordination among agents is fundamental to achieve satisfactory control strategies. The naive encoding strategy is not effective since local policies struggle to predict optimal local actions starting from high-dimensional inputs, with the local state value that loses significance when concatenated with system parameters and positional encoding vector.

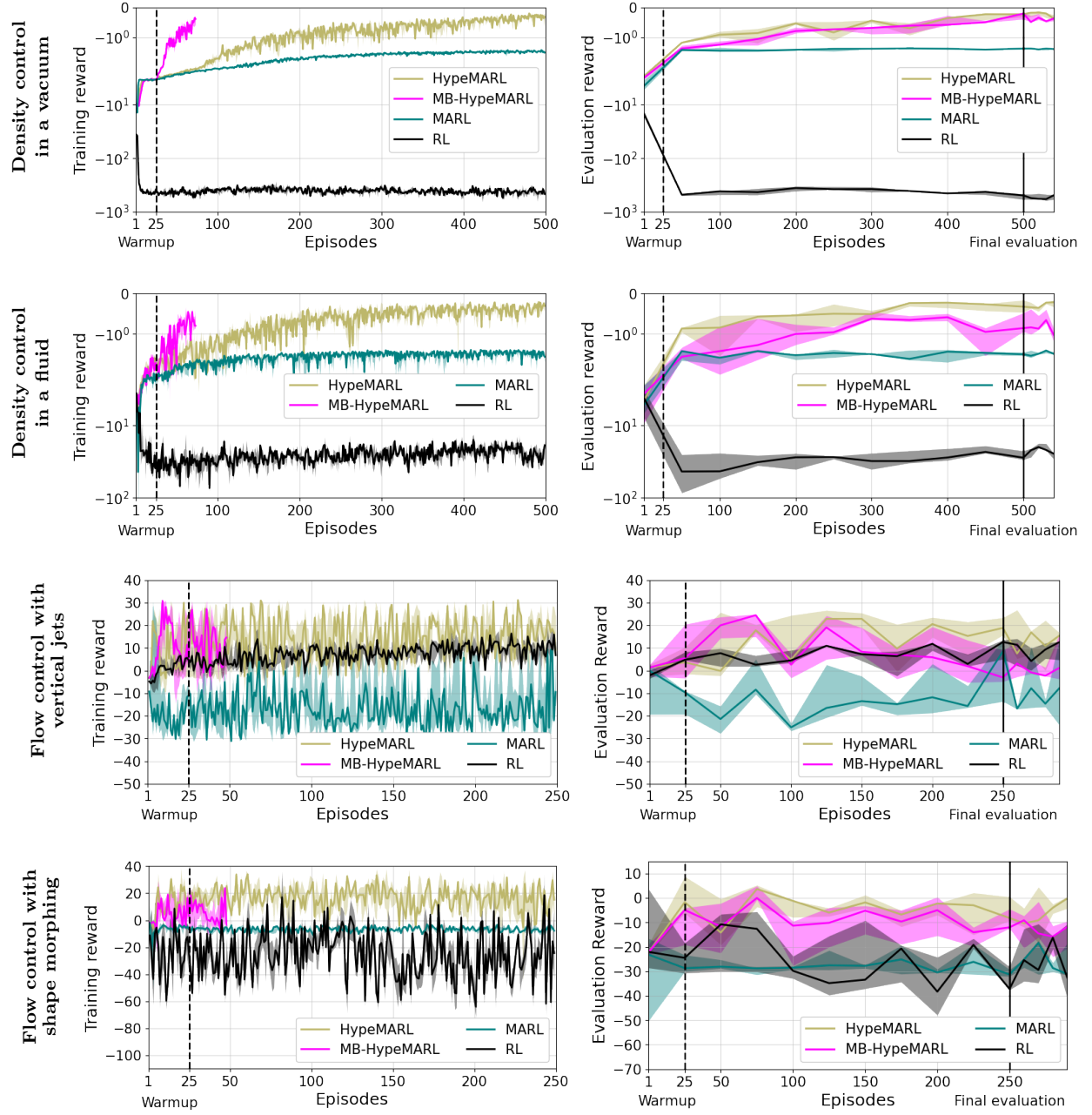


Figure A.12. Comparison with single-agent RL. Median and interquartile range of HypeMARL, MB-HypeMARL, MARL and single-agent RL training end evaluation rewards.

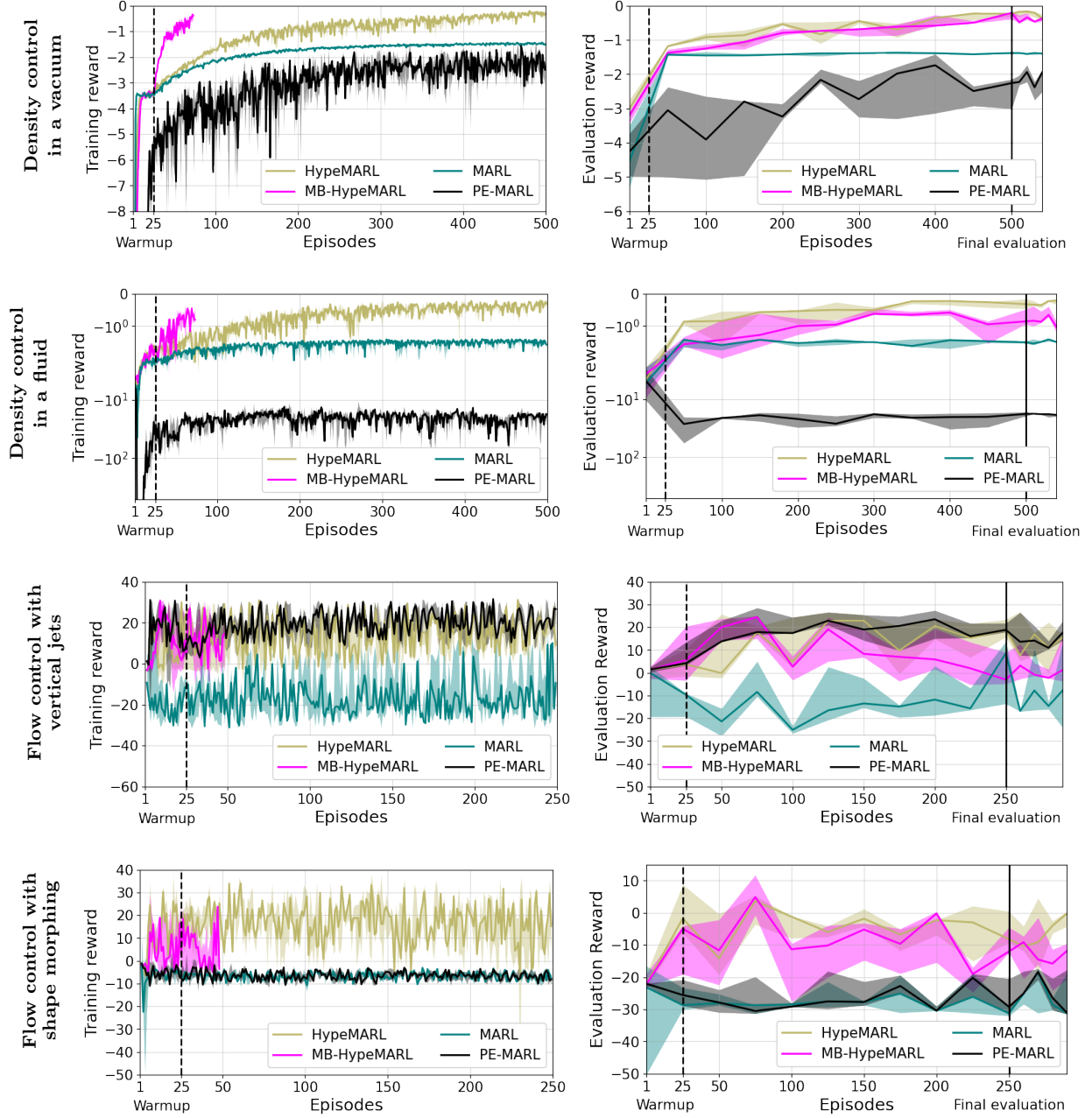


Figure B.13. Comparison with PE-MARL baseline. Median and interquartile range of HypeMARL, MB-HypeMARL, MARL and PE-MARL training end evaluation reward.

Appendix C. Neural Network Architectures

In this section, we discuss the neural network architectures used by the different agents. All approaches, namely HypeMARL, MARL, PE-MARL, and RL estimates the action-value functions and policies, while MB-HypeMARL in addition learns a local forward model of the dynamics.

The main policy network $\pi(\mathbf{y}_{i,t}; \boldsymbol{\theta}_{\pi,i})$ used by HypeMARL and MB-HypeMARL is composed of one input layer of dimension equal to the local state N_{y_i} , one hidden layer with 256 neurons, and an output layer of dimension equal to the control action dimension N_{u_i} . We can write the main policy network as:

$$\begin{aligned}\mathbf{x}_{i,t} &= \text{ReLU}\left((1 + \mathbf{g}^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi})) \odot \mathbf{y}_{i,t} W^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi}) + \mathbf{b}^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi})\right), \\ \mathbf{u}_{i,t} &= \tanh\left((1 + \mathbf{g}^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi})) \odot \mathbf{x}_{i,t} W^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi}) + \mathbf{b}^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi})\right),\end{aligned}$$

where $\mathbf{g}^{(1)}, W^{(1)}, \mathbf{b}^{(1)}$ and $\mathbf{g}^{(2)}, W^{(2)}, \mathbf{b}^{(2)}$ are respectively the gain vectors, the weights matrices, and the biases of the two layers of the main networks that are learned via the hypernetwork $H_\pi(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_\pi})$, i.e., the hypernetworks outputs, $\boldsymbol{\theta}_{\pi,i} = [\mathbf{g}^{(1)}, W^{(1)}, \mathbf{b}^{(1)}, \mathbf{g}^{(2)}, W^{(2)}, \mathbf{b}^{(2)}]$, and $\odot$ indicates the element-wise product. This type of architecture is motivated by the work in [86]. Similarly to the policy, the main value function networks $Q_1(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\theta}_{Q_1})$ and $Q_2(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}; \boldsymbol{\theta}_{Q_2})$ are composed of one input layer of dimension $N_{y_i} + N_{u_i}$, one hidden layer with 256 neurons, and an output layer of dimension 1. Assuming $\mathbf{z}_{i,t} = [\mathbf{y}_{i,t}, \mathbf{u}_{i,t}]$, we can write each value function as:

$$\begin{aligned}\chi_{i,t} &= \text{ReLU}\left((1 + \mathbf{g}^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q})) \odot \mathbf{z}_{i,t} W^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q}) + \mathbf{b}^{(1)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q})\right), \\ q_{i,t} &= (1 + \mathbf{g}^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q})) \odot \chi_{i,t} W^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q}) + \mathbf{b}^{(2)}(\mathbf{PE}(p_i), \boldsymbol{\mu}; \boldsymbol{\theta}_{H_Q}),\end{aligned}$$

where for the sake of a concise notation, we dropped the index distinguishing the two value functions that TD3 estimates. The hypernetworks use the same architecture and weight initialization proposed in [63, 67]. Each hypernetwork is composed of three blocks that map $\mathbf{PE}(p_i), \boldsymbol{\mu}$ to $\mathbf{g}^{(1)}, W^{(1)}, \mathbf{b}^{(1)}$ and $\mathbf{g}^{(2)}, W^{(2)}, \mathbf{b}^{(2)}$. Each block is composed of a linear layer and two residual blocks and each residual block is composed of two linear layers with ReLU activation and a skip connection.

On the other hand, MARL, PE-MARL, and RL use the standard actor-critic architectures, where the actor is composed of 2 fully connected hidden layers with 256 neurons each with ReLU activation, and, in the case of the policy network a tanh activation at the output layer to bound the actions in a suitable range.

Eventually, MB-HypeMARL approximates the local dynamics model with neural network with two hidden layers with 256 neurons each and ELU activations.

Appendix D. Pseudocode of HypeMARL and MB-HypeMARL

We report the pseudocode of HypeMARL and MB-HypeMARL in Algorithm 1.

Algorithm 1 HypeMARL and MB-HypeMARL

```

1: Initialize models' parameters and noise variance  $\sigma^2$ 
2: Set maximum number of episodes  $E_{\max}$  and time steps  $T_{\max}$ 
3: for  $e = 1 : E_{\max}$  do
4:   Sample  $\mu$ 
5:   Get initial measurement  $\mathbf{y}_{i,1}$  for  $i = 1 : N$ 
6:   Set  $\tilde{\mathbf{y}}_{i,1} = \mathbf{y}_{i,1}$  for  $i = 1 : N$ 
7:   for  $t = 1 : T_{\max}$  do
8:     Sample  $\theta_{\pi,i} = H_{\pi}(\mathbf{PE}(p_i), \mu; \theta_{H_{\pi}})$  for  $i = 1 : N$ 
9:     if interact with real environment then
10:      Sample  $\mathbf{u}_{i,t} \sim \pi(\mathbf{y}_{i,t}; \theta_{\pi,i}) + \epsilon$ , where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$  for  $i = 1 : N$ 
11:      Apply joint action  $\mathbf{u}_t = [\mathbf{u}_{1,t}, \dots, \mathbf{u}_{N,t}]^\top$ 
12:      Observe  $\mathbf{y}_{i,t+1} = F(\mathbf{y}_t, \mathbf{u}_t, \mu)$  for  $i = 1 : N$ 
13:      Compute local rewards  $r_{i,t}$  for  $i = 1 : N$ 
14:      if train models then
15:        Update  $\pi$  and  $Q$  using the tuple  $(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}, r_{i,t}, \mathbf{y}_{i,t+1}, \mu)_{i=1:N}$ 
16:        Update  $\tilde{F}$  using the tuple  $(\mathbf{y}_{i,t}, \mathbf{u}_{i,t}, r_{i,t}, \mathbf{y}_{i,t+1}, \mu)_{i=1:N}$ 
17:      end if
18:     else if interact with surrogate environment then
19:       Sample  $\mathbf{u}_{i,t} \sim \pi(\tilde{\mathbf{y}}_{i,t}; \theta_{\pi,i}) + \epsilon$ , where  $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2)$  for  $i = 1 : N$ 
20:       Apply joint action  $\mathbf{u}_t = [\mathbf{u}_{1,t}, \dots, \mathbf{u}_{N,t}]^\top$ 
21:       Observe  $\tilde{\mathbf{y}}_{i,t+1} = \tilde{F}(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}, \mu; \theta_{\tilde{F}})$  for  $i = 1 : N$ 
22:       Compute rewards  $\tilde{r}_{i,t}$  for  $i = 1, \dots, N$ 
23:       if train models then
24:        Update  $\pi$  and  $Q$  using the tuple  $(\tilde{\mathbf{y}}_{i,t}, \mathbf{u}_{i,t}, \tilde{r}_{i,t}, \tilde{\mathbf{y}}_{i,t+1}, \mu)_{i=1:N}$ 
25:       end if
26:     end if
27:   end for
28: end for

```

Appendix E. Hyperparameters of the numerical experiments

In this section, we report the value of the hyperparameters of our experiments. In particular, we list in Table E.1 the hyperparameters of HypeMARL, MB-HypeMARL, MARL, PE-MARL, and RL.

Table E.1. Agents

	HypeMARL	MB-HypeMARL	MARL	PE-MARL	RL
RL algorithm	TD3	TD3	TD3	TD3	TD3
batch size	32	32	32	32	32
positional encoding d	2048	2048	—	2048	—
positional encoding n	1000	1000	—	1000	—
discount γ	0.99	0.99	0.99	0.99	0.99
target network update ρ	0.005	0.005	0.005	0.005	0.005
action noise std σ	0.25	0.25	0.25	0.25	0.25
decay std σ	linear	linear	linear	linear	linear
value updates per training episode	100	100	100	100	100
policy updates per training episode	50	50	50	50	50
value-to-policy update ratio	2	2	2	2	2
learning rate actor π	10^{-6}	10^{-6}	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
learning rate critic Q	$5 \cdot 10^{-5}$	$5 \cdot 10^{-5}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$	$3 \cdot 10^{-4}$
num. hidden layer actor π	1	1	2	2	2
num. hidden layer critic Q	1	1	2	2	2
hidden dim. actor π	256	256	256	256	256
hidden dim. critic Q	256	256	256	256	256
learning rate forward dynamics F	—	10^{-4}	—	—	—
num. layer forward dynamics F	—	1	—	—	—
hidden dim. forward dynamics F	—	256	—	—	—

MOX Technical Reports, last issues

Dipartimento di Matematica
Politecnico di Milano, Via Bonardi 9 - 20133 Milano (Italy)

- 74/2026** Brivio, S.; Franco, N. R.
Deep Symmetric Autoencoders from the Eckart-Young-Schmidt Perspective
- 73/2026** Corti, M.
A weighted polygonal discontinuous Galerkin method for hierarchically coupled reaction–diffusion systems with cubic interactions
- 72/2026** Lorenzon, G.; Regazzoni, F.
Optimal Transport Dropout for Structured Predictive Uncertainty
- 71/2026** Radisic, I.; Regazzoni, F.; Dede', L.; Quarteroni, A.
A mathematical model for irreversible damage of the collagen scaffold in the myocardium
- 70/2026** Micheletti, S.
Longitudinal versus quasi-P waves in weakly anisotropic elasticity: operator comparison and long-time dephasing
- 68/2026** Patanè, G.; Greven, S.; Menafoglio, A.
Random mixtures in Bayes Hilbert spaces
- 69/2026** Zecchi, A. A.; Ferro, N.; Perotto, S.
Complexity-controlled QUBO formulation for topology optimization via anisotropic mesh adaptation
- 67/2026** Ciardulli, S.; Fontana, N.; Vantini, S.; Ieva F.
Generalized propensity score weighting for functional causal inference framework
- 66/2026** Antonietti, P.F.; Bonizzoni, F.; Corti, M.; De March, N.; Di Noto, S.; Regazzoni, F.
A stability-preserving polytopal discontinuous Galerkin method for the Fisher-Kolmogorov model with applications to neurodegenerative diseases
- 65/2026** De Sanctis, M.F.; Arnone, E.; Ieva, F.; Sangalli, L.M.
Modeling group heterogeneity in spatio-temporal data via physics-informed regression