Data-Centric Engineering (2025), 6: e51
doi:10.1017/dce.2025.10027

RESEARCH ARTICLE

Interpretable and efficient data-driven discovery and control of distributed systems

Florian Wolf[1,2,3], Nicolò Botteghi[2], Urban Fasel[4] and Andrea Manzoni[2]

[1]Department of Mathematics, Technical University of Darmstadt, Darmstadt, Germany
[2]Department of Mathematics, MOX, Politecnico di Milano, Milan, Italy
[3]The Computing + Mathematical Sciences Department, California Institute of Technology, Pasadena, USA
[4]Department of Aeronautics, Imperial College London, London, UK
Corresponding author: Florian Wolf; Email: florian.wolf@stud.tu-darmstadt.de

Received: 24 May 2025; **Revised:** 25 August 2025; **Accepted:** 06 September 2025

Keywords: autoencoder; model-based deep reinforcement learning; sparse identification of nonlinear dynamics

Abstract

Effectively controlling systems governed by partial differential equations (PDEs) is crucial in several fields of applied sciences and engineering. These systems usually yield significant challenges to conventional control schemes due to their nonlinear dynamics, partial observability, high-dimensionality once discretized, distributed nature, and the requirement for low-latency feedback control. Reinforcement learning (RL), particularly deep RL (DRL), has recently emerged as a promising control paradigm for such systems, demonstrating exceptional capabilities in managing high-dimensional, nonlinear dynamics. However, DRL faces challenges, including sample inefficiency, robustness issues, and an overall lack of interpretability. To address these challenges, we propose a data-efficient, interpretable, and scalable Dyna-style model-based RL framework specifically tailored for PDE control. Our approach integrates Sparse Identification of Nonlinear Dynamics with Control within an Autoencoder-based dimensionality reduction scheme for PDE states and actions (AE+SINDy-C). This combination enables fast rollouts with significantly fewer environment interactions while providing an interpretable latent space representation of the PDE dynamics, facilitating insight into the control process. We validate our method on two PDE problems describing fluid flows—namely, the 1D Burgers equation and 2D Navier–Stokes equations—comparing it against a model-free baseline. Our extensive analysis highlights improved sample efficiency, stability, and interpretability in controlling complex PDE systems.

Impact Statement

Controlling complex physical systems described by partial differential equations is central to many applications in engineering and applied sciences. In recent years, data-driven control strategies, particularly reinforcement learning, have gained attention due to their effectiveness and real-time applicability. However, these techniques often struggle with sample inefficiency and lack of interpretability, especially when dealing with high-dimensional systems. The proposed data-driven approach combines autoencoders with sparse model discovery, enabling scalable, robust, and interpretable control of high-dimensional systems. Our method significantly reduces the need for expensive simulations or experiments while uncovering low-dimensional and interpretable dynamics models. These advances open new possibilities for the design and deployment of reliable, data-

This research article was awarded Open Data badge for transparent practices. See the Data Availability Statement for details.

© The Author(s), 2025. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/4.0), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

efficient control strategies across industries, such as energy, aerospace, and manufacturing, and provide a foundation for more transparent and trustworthy AI-based control in safety-critical systems.

1. Introduction

Feedback control for complex physical systems is essential in many fields of engineering and applied sciences, which are typically governed by partial differential equations (PDEs). In these cases, the state of the systems is often challenging or even impossible to observe completely; the systems exhibit nonlinear dynamics and require low-latency feedback control (Brunton et al., 2020; Peitz and Klus, 2020; Kim and Jeong, 2021). Consequently, effectively controlling these systems is a computationally intensive task. For instance, significant efforts have been devoted in the last decade to the investigation of optimal control problems governed by PDEs (Hinze et al., 2008; Manzoni et al., 2022); however, classical feedback control strategies face limitations with such highly complex dynamical systems. For instance, (nonlinear) model predictive control (MPC) (Grün and Pannek, 2017) has emerged as an effective and important control paradigm. MPC utilizes an internal model of the dynamics to create a feedback loop and provide optimal controls, resulting in a difficult trade-off between model accuracy and computational performance. Despite its impressive success in disciplines such as robotics (Williams et al., 2018) and controlling PDEs (Altmüller, 2014), MPC struggles with real-time applicability in providing low-latency actuation, due to the need for solving complex optimization problems.

In recent years, reinforcement learning (RL), particularly deep RL (DRL) (Sutton and Barto, 2018), an extension of RL relying on deep neural networks, has gained popularity as a powerful and real-time applicable control paradigm. Especially in the context of solving FPDEs, DRL has demonstrated outstanding capabilities in controlling complex and high-dimensional dynamical systems at low latency (Yousif et al., n.d.; Botteghi and Fasel, 2024; Peitz et al., 2024; Vinuesa, 2024). Additionally, recent community contributions of benchmark environments such as *ControlGym* (https://github.com/xiangyuan-zhang/controlgym) (Zhang et al., n.d.), *PDEControlGym* (https://github.com/lukebhan/PDEControlGym) (Bhan et al., 2024), and *HydroGym* (https://github.com/dynamicslab/hydrogym), all wrapped under the uniform *Gym* (Brockman et al., 2016) interface, highlight the importance of DRL in the context of controlling PDEs. Despite its impressive success, DRL faces three major challenges:

1. **Data usage:** DRL algorithms are known to be sample inefficient, requiring a large number of environment interactions, resulting in long training times and high computational requirements (Zoph and Le, 2016). This is particularly problematic in the context of controlling PDEs, as generating training data often requires either long simulations (Zolman et al., n.d., section B.3) or would necessitate very expensive real-world experiments (OpenAI, 2020).
2. **Robustness of the control performance:** In high-dimensional state-action spaces, as in the context of (discretized) PDEs, it is challenging to generate sufficient training data for the agent to adequately cover the state-action space. This is crucial for ensuring a reliable and trustworthy controller, impeding the application to safety-critical use-cases like nuclear fusion (Zhi et al., 2018) or wind energy (Aubru et al., 2017).
3. **Black-box model:** DRL methods often result in policies and learned dynamics that are difficult to interpret. While interpretability may not be critical when the sole objective is control performance, it becomes important in scenarios where understanding the underlying PDE dynamics or ensuring transparent and verifiable decision-making is necessary. This includes applications in scientific discovery, safety-critical systems, or when diagnosing and improving controllers. In the context of PDE control, interpretability aids both in gaining insights into complex dynamics and in building trust in the learned controllers (Alla et al., 2023; Botteghi and Fasel, 2024).

A popular approach to address the aforementioned problem of sample inefficiency is the use of model-based algorithms, specifically *model-based RL* (MBRL) (Sutton, 1990; Deisenroth and Rasmussen, 2011;

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Chua et al., 2018; Clavera et al., 2018; Wang et al., 2019; Hafner et al., 2020). One of its various forms is the *Dyna-style* MBRL algorithm (Sutton, 1991), which iteratively collects samples from the full-order environment to learn a surrogate dynamics model. The agent then alternates between interacting with the surrogate model and the actual environment, significantly reducing the amount of required training data, allowing faster rollouts. Recent contributions in MBRL for PDEs include the use of convolutional long-short term memory with actuation networks (Werner and Peitz, 2023) and Bayesian linear regression for model identification in the context of the State-Dependent Riccati Equation control approach (Alla et al., 2023).

Sparse dictionary learning is a class of data-driven methods that seek to approximate a nonlinear function using a sparse linear combination of candidate dictionary functions, for example, polynomials of degree d or trigonometric functions. In the context of identification of dynamical systems, sparse dictionary learning is used by the *Sparse Identification of Nonlinear Systems* (SINDy) method (cf. Section 2.3). SINDy is a very powerful method to identify a parsimonious, that is, with a limited number of terms, dynamics model and resolve the issue of lacking interpretability, for example, of deep neural networks. The *PySINDy* (https://github.com/dynamicslab/pysindy) package (Kaptanoglu et al., 2021) makes implementations easy and fast. With its active community, SINDy has been extended to various forms, such as ensemble versions (Fasel et al., 2022), uncertainty quantification for SINDy (Hirsh et al., 2022), Bayesian autoencoder (AE)-based SINDy (Gao and Kutz, 2024) and applied to a variety of different applications, including turbulence closures (Zanna and Bolton, 2020), PDEs (Rudy et al., 2016), continuation methods (Conti et al., 2023), and variational architectures (Conti et al., 2024). In practice, problems governed by PDEs usually require a fine enough discretization in order to be solved effectively, leading to high-dimensional dynamical systems (order of magnitude 50×10^3, or even larger). However, SINDy does not scale to high-dimensional systems, which is why we learn the latent representation of the high-dimensional states and actions of the PDE with two AEs. This dimensionality-reduction step allows for using SINDy and improves the robustness (we learn a low-dimensional representation of the PDE states) of the control policies (Lesort et al., 2018; Botteghi et al., 2025).

Our contribution can be summarized as follows:

- With AE+SINDy-C, we present a novel combination of the AE framework and the SINDy-C algorithm, incorporating controls into the AE framework as a Dyna-style MBRL method for controlling PDEs.
- AE+SINDy-C enables fast rollouts, significantly reducing the required number of full-order environment interactions and provides an interpretable, low-dimensional latent representation of the dynamical system.
- We demonstrate the feasibility of our approach by solving two different fluid-flow PDEs, comparing it to a state-of-the-art model-free baseline algorithm.
- We provide an extensive analysis of the learned dynamics and numerical ablation studies.

In particular, our work relies on two SINDy (Brunton et al., 2016) extensions:

1. For control applications, SINDy has been extended to work with controls, namely *SINDy-C*, MPCs (Kaiser et al., 2018; Fasel et al., 2021), and importantly, in the context of RL in the work of Arora et al. (Arora et al., 2022), and more recently by Zolman et al. (Zolman et al., n.d.) (see Section 2.3.1 for more details).
2. The work of Champion et al. (Champion et al., 2019) enables data-driven discovery using SINDy and an AE framework for high-dimensional dynamical systems (see Section 2.3.2 for more details), which has been extended to the continuation of parameter-dependent PDE solutions by Conti et al. (Conti et al., 2023).

The remainder of this paper is structured as follows: Section 2 provides a brief overview of the general problem setting, related with control strategies relying on DRL, and the two aforementioned SINDy

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

extensions upon which AE+SINDy-C is based. Section 3 explains the proposed method in detail and highlights the usage of AE+SINDy-C in the context of DRL for controlling PDEs. In Section 4, we explain the two PDE benchmark cases and analyze the numerical results in detail. Section 5 provides an overview of possible interesting directions for follow-up work and variations of AE+SINDy-C. All details about the used environments, DRL, and the training of AE+SINDy-C can be found in Appendix A.

2. Background and related work

2.1. Problem setting

In this work, we address the problem of controlling a nonlinear, distributed, dynamical system described by the equation:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}(t) = f(\mathbf{x}(t), \mathbf{u}(t)),$$

where the state $\mathbf{x}(t) \in \mathbb{R}^{N_x}$ and control input $\mathbf{u}(t) \in \mathbb{R}^{N_u}$ are continuous variables, with potentially very large dimensions N_x and N_u, respectively. The function $f : \mathbb{R}^{N_x} \times \mathbb{R}^{N_u} \to \mathbb{R}^{N_x}$ is assumed to be unknown, but we can observe a time-discrete evolution of the system, resulting in a sequence of (partially or fully) observable measurements $\mathbf{m}_t, \mathbf{m}_{t+1}, \ldots, \mathbf{m}_{t+H}$ of the state over a horizon $H \in \mathbb{N}_+$. A system is fully observable (FO) if the observations $\mathbf{m}_i \in \mathbb{R}^{N_x}$ allow to observe each (full) state of the system directly, that is, $N_x^{\mathrm{Obs}} = N_x$ and $\mathbf{m}_i = \mathbf{x}_i$. On the other hand, a system is partially observable (PO) if only a limited number of sensors is available, resulting in a lower-dimensional observation space $\mathbf{m}_i \in \mathbb{R}^{N_x^{\mathrm{Obs}}}$, where $N_x^{\mathrm{Obs}} \ll N_x$ that does not capture the full state of the PDE. In this work, we consider both FO and PO systems. Similarly, we assume a limited number of actuators are available to control the system. The combination of nonlinear system dynamics with a limited number of state sensors and control actuators entails a complex and challenging problem in PDE control.

2.2. Reinforcement learning

RL is a general framework for solving sequential decision-making processes. RL has been applied to a variety of different tasks, including natural language processing for dialogue generation (OpenAI Team, 2024) and text summarization (Li et al., 2016), computer vision for object detection and image classification (Mnih et al., 2013), robotics for autonomous control and manipulation (Levine et al., 2016), finance for portfolio management and trading strategies (Jiang et al., 2017), and game playing for mastering complex strategic games (Silver et al., 2016). These applications highlight the broad versatility and significant impact of RL across multiple fields.

RL is the subset of machine learning that focuses on training agents to make decisions by interacting with an environment. The RL framework is typically modeled as a Markov decision process (MDP), defined by the tuple $\mathcal{M} := (\mathcal{X}, \mathcal{U}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{X} \subseteq \mathbb{R}^{N_x}$ is the set of observable states, $\mathcal{U} \subseteq \mathbb{R}^{N_u}$ is the set of actions, $\mathcal{P} : \mathcal{X} \times \mathcal{X} \times \mathcal{U} \to [0,1]$ is the transition probability kernel with $\mathcal{P}(\mathbf{x}', \mathbf{x}, \mathbf{u})$ representing the probability of reaching the state $\mathbf{x}' \in \mathcal{X}$ while being in the state $\mathbf{x} \in \mathcal{X}$ and applying action $\mathbf{u} \in \mathcal{U}$, $\mathcal{R} : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the reward function, and $\gamma \in (0,1]$ is the discount factor.

The goal of an RL agent is to learn an optimal policy π that maps states $\mathbf{x}$ of the environment to actions $\mathbf{u}$ in a way that maximizes the expected cumulative reward over time a control horizon H:

$$R_H = \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t\right], \tag{2.1}$$

with rewards r_t collected at timesteps t and a control horizon H being finite or infinite—in this work we focus on finite control horizons. RL agents often optimize the policy by approximating either the value function $V : \mathcal{X} \to \mathbb{R}$ or the action-value function $Q : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ to quantify, and optimize the cumulative reward (eq. [2.1]) for a state $\mathbf{x} \in \mathcal{X}$ or a state-action pair $(\mathbf{x}, \mathbf{u}) \in \mathcal{X} \times \mathcal{U}$, respectively. When dealing with

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

high-dimensional state spaces, continuous actions, and nonlinear dynamics, the estimation of the value function becomes a very challenging and data-inefficient optimization problem.

In general, we distinguish between model-free and model-based RL algorithms. Model-free algorithms do not assume any explicit model of the system dynamics, and aim at optimizing the policy directly by interacting with the environment. This approach offers more flexibility and is more robust to model inaccuracies but usually requires a large number of interactions to achieve good performance, making it difficult to apply in cases of data sparsity or expensive interactions. On the other hand, model-based RL algorithms internally create a model of the environment's dynamics, that is, the transition probability kernel $\mathcal{P}$ and the reward function $\mathcal{R}$ (note that in this work we only consider the case of a known full-order reward functions). The agent leverages this model to simulate the environment and plan its actions accordingly, typically resulting in more sample-efficient training. We focus on Dyna-style (Sutton, 1991) RL algorithms, which learn a surrogate model of the system dynamics, allowing the agent to train on an approximation of the environment and thus requiring fewer data.

A general scheme of the RL cycle we used for training, including a Dyna-style surrogate dynamics model of the environment is shown in Figure 1. Since this work focuses on Dyna-style RL algorithms with the surrogate model being the center of this work, we use the proximal policy optimization (PPO) algorithm (Schulman et al., 2017) as a state-of-the-art actor-critic algorithm. Actor-critic methods learn both the value-function, that is, the critic, and the policy, that is, the actor, at the same time and have shown very promising results in the last years.

2.3. SINDy: Sparse Identification of Nonlinear Dynamics

We briefly review two versions of *Sparse Identification of Nonlinear Dynamics* (SINDy) used in the SINDy-RL (Zolman et al., n.d.) and AE+SINDy (Champion et al., 2019) frameworks. SINDy (Brunton et al., 2016) is an extremely versatile and popular dictionary learning method, that is, a data-driven algorithm aiming at approximating a nonlinear function through a dictionary of user-defined functions. In particular, SINDy assumes that this approximation takes the form of a sparse linear combination of (potentially nonlinear) candidate functions, such as polynomials or trigonometric functions.

In its general formulation, given a set of N data points $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$ with $\mathbf{x}_k \in \mathbb{R}^m$ and $\mathbf{y}_k = f(\mathbf{x}_k) \in \mathbb{R}^n$, $k = 1, \ldots, N$, and collected as

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top \in \mathbb{R}^{N \times m}, \quad \mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^\top \in \mathbb{R}^{N \times n},$$

and a set of d candidate functions $\mathbf{\Theta}(\mathbf{X}) = [\theta_1(\mathbf{x}), \ldots, \theta_d(\mathbf{x})] \in \mathbb{R}^{N \times d}$, we aim to find a representation of the form

$$\mathbf{Y} = \mathbf{\Theta}(\mathbf{X}) \cdot \mathbf{\Xi}, \tag{2.2}$$

where $\mathbf{\Xi} \in \mathbb{R}^{d \times n}$ is the coefficients to be fit. These are usually trained in a Lasso-style optimization (Tibshirani, 2018) since we assume sparsity, that is, the system dynamics can be sufficiently represented by a small subset of terms in the library. Overall, we optimize (a variation of) the following loss:

$$\mathbf{\Xi} = \arg\min_{\widehat{\mathbf{\Xi}}} \left\| \mathbf{Y} - \mathbf{\Theta}(\mathbf{X}) \widehat{\mathbf{\Xi}} \right\|_F + \mathcal{L}\left(\widehat{\mathbf{\Xi}}\right), \tag{2.3}$$

with a regularization term $\mathcal{L}$ promoting sparsity, that is, usually $\mathcal{L}(\mathbf{\Xi}) = \|\mathbf{\Xi}\|_1$, and $\|\cdot\|_F$ is the Frobenius norm defined as $\|A\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2}$ for a matrix $A \in \mathbb{R}^{m \times n}$. While in Champion et al. (2019)) and Zolman et al. (n.d.)), the loss eq. (2.3) is an $\|\cdot\|_2$-penalty with sequential thresholding least squares (STLS), we use *PyTorch*'s automatic differentiation framework (Paszke et al., 2017) and optimize the $\|\cdot\|_1$-loss as in (Brunton et al., 2016).

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

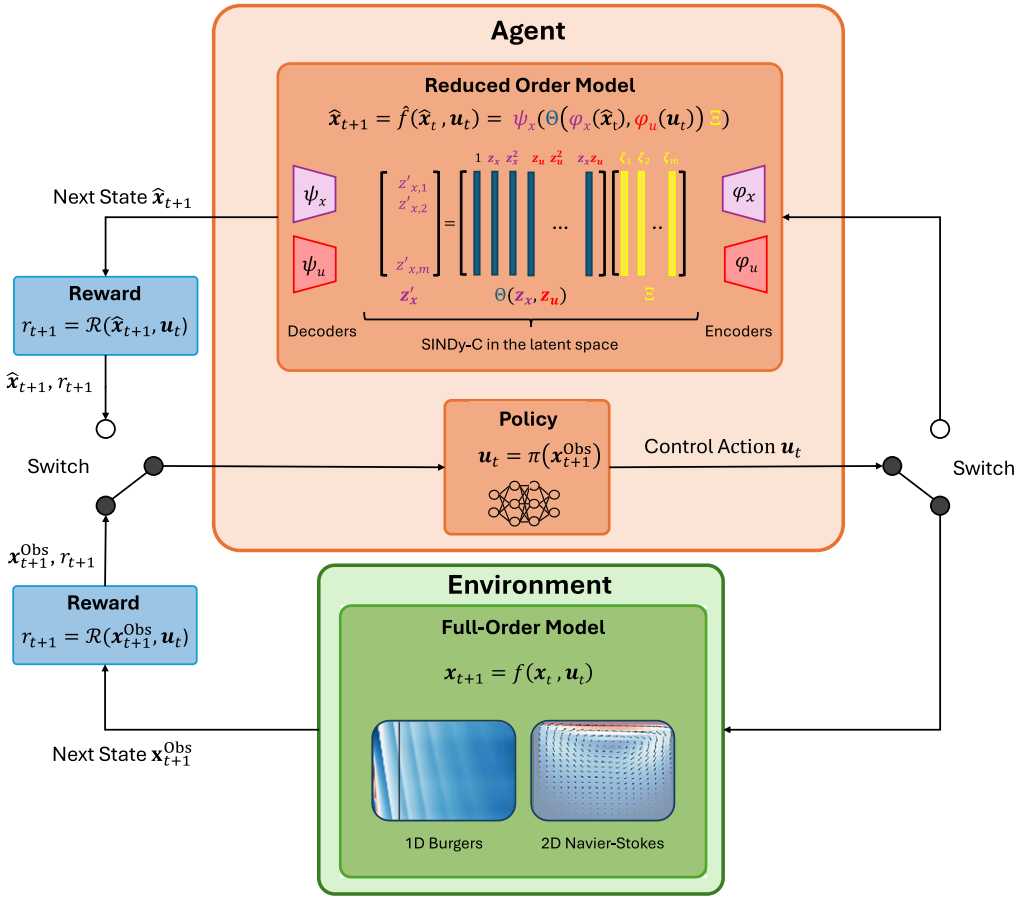


Figure 1. *Overview of the RL training loop. In Dyna-style algorithms, we choose if the agent interacts with the full-order model, requiring (expensive) environment rollouts or the learned surrogate, that is reduced order, model, providing fast approximated rollouts. In this work, we focus on the setting where the full-order reward is (analytically) known and only the dynamics are approximated. In general, the observed state is computed by* $\mathbb{R}^{N_x^{\mathrm{Obs}}} \ni \mathbf{x}_{t+1}^{\mathrm{Obs}} = C \cdot \mathbf{x}_{t+1}$. *In the partially observable (PO) case, the projection matrix* $C \in \{0,1\}^{N_x \times N_x^{\mathrm{Obs}}}$ *is structured with a single 1 per row and zero elsewhere, that is,* $N_x^{\mathrm{Obs}} \ll N_x$. *In the fully observable case* $C \equiv \mathrm{Id}_{\mathbb{R}^{N_x}}$, *that is,* $N_x^{\mathrm{Obs}} = N_x$.

2.3.1. SINDy-RL

SINDy-RL is a very recent extension of SINDy that combines SINDy with an RL algorithm for Dyna-style RL (Zolman et al., n.d.; Arora et al., 2022). This approach involves using SINDy to learn a surrogate model of the environment and then train the RL agent using this approximation of the full-order model. This method results in a drastically-reduced number of necessary full-order interactions and very fast convergence. Specifically, Zolman et al. (Zolman et al., n.d.) work with the discrete SINDy-C formulation, that is, including controls, for the discovery task $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$, where $\mathbf{X} = (\mathbf{x}(t_k), \mathbf{u}(t_k))_{k=1,\ldots,N}$ and $\mathbf{Y} = (\mathbf{x}(t_{k+1}))_{k=1,\ldots,N}$ in the notation of eq. (2.2). Note that since in our work we focus on enabling efficient surrogate representations for controlling distributed systems, we do not further review the reward approximation nor the policy approximation aspects addressed in Zolman et al. (n.d.)).

Despite significantly reducing the number of full-order interactions, not requiring derivatives of (potentially noisy) measurements, and incorporating controls in the surrogate model, so far SINDy-RL lacks the scalability to high-dimensional systems (see Zolman et al., n.d., section 6). While it shows

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

convincing results for state spaces with up to eight dimensions and action spaces with up to two dimensions, controlling distributed systems require much larger state and action spaces to be solved and controlled accurately.

2.3.2. AE+SINDy: Autoencoder framework for data-driven discovery in high-dimensional spaces

In their work, Champion et al. (2019)) rely on the classical SINDy formulation for the discovery task in the latent space. Indeed, they generalize the concept of data-driven dynamics discovery to high-dimensional dynamical systems by combining the classical SINDy formulation with an AE framework (Hinton and Salakhutdinov, 2006; Bengio et al., 2012; Goodfellow et al., 2016) to allow for nonlinear dimensionality reduction. Hence, instead of directly working with high-dimensional systems, the SINDy algorithm is applied to the compressed, lower-dimensional representation of the system with $N_x^{\text{Lat}} \ll N_x$.

Given the encoder $\varphi(\cdot, \mathbf{W}_\varphi) : \mathbb{R}^{N_x} \to \mathbb{R}^{N_x^{\text{Lat}}}$ and the decoder network $\psi(\cdot, \mathbf{W}_\psi) : \mathbb{R}^{N_x^{\text{Lat}}} \to \mathbb{R}^{N_x}$, as well as the latent representation $\mathbf{z}(t) = \varphi(\mathbf{x}(t)) \in \mathbb{R}^{N_x^{\text{Lat}}}$, the following loss is minimized:

$$\begin{aligned} \min_{\mathbf{W}_\varphi, \mathbf{W}_\psi, \boldsymbol{\Xi}} & \underbrace{\|\mathbf{x} - \psi(\mathbf{z})\|_2^2}_{\text{reconstruction loss}} + \underbrace{\lambda_1 \left\|\dot{\mathbf{x}} - (\nabla_{\mathbf{z}} \psi(\mathbf{z}))\left(\boldsymbol{\Theta}\left(\mathbf{z}^\top\right)\boldsymbol{\Xi}\right)\right\|_2^2}_{\text{SINDy loss in } \dot{\mathbf{x}}} \\ & + \underbrace{\lambda_2 \left\|\dot{\mathbf{z}} - \boldsymbol{\Theta}\left(\mathbf{z}^\top\right)\boldsymbol{\Xi}\right\|_2^2}_{\text{SINDy loss in the latent space}} + \underbrace{\lambda_3 \|\boldsymbol{\Xi}\|_1}_{\text{sparse regularization}}, \end{aligned} \tag{2.4}$$

with $\dot{\mathbf{z}} = (\nabla_{\mathbf{x}} \mathbf{z})\dot{\mathbf{x}}$. The SINDy loss in $\dot{\mathbf{x}}$, that is, the *consistency loss*, ensures that the time derivatives of the prediction matches the input time derivative $\dot{\mathbf{x}}$; we refer to figure 1 in Champion et al. (2019)) for more details (the AE+SINDy algorithm could be seen as the upper half of Figure 2 with a different loss). As described in the supplementary material of Champion et al. (2019)), due to its nonconvexity and to obtain a parsimonious dynamical model, the loss in eq. (2.4) is optimized via STLS.

AE+SINDy provides very promising results for (re-)discovering the underlying true low-dimensional dynamics representation of high-dimensional dynamical systems. Further extensions of the AE+SINDy framework have been recently proposed in Conti et al. (2023)) and Bakarji et al. (2023)). However, to be applicable in a control setting with (potentially noisy) data, AE+SINDy suffers from the necessity of requiring derivatives of the observed data and the inability to include controls in the SINDy framework. In the next section, we will combine and generalize the approaches of SINDy-RL and AE+SINDy to develop AE+SINDy-C within an RL setting.

3. AE+SINDy-C: Low-dimensional sparse dictionary learning for simultaneous discovery and control of distributed systems

3.1. Latent model discovery

To efficiently scale SINDy to high-dimensional state spaces, incorporate controls, and eliminate reliance on derivatives, we introduce AE+SINDy-C. This approach aims to accelerate DRL for control tasks involving distributed—potentially large-scale—systems, by combining the derivative-free Dyna-style DRL training in the SINDy-C case, as demonstrated by (Zolman et al., n.d.; Arora et al., 2022), with the scalable approach to high-dimensional state- and action-spaces from (Champion et al., 2019).

Our method approximates the environment dynamics to speed up computationally-expensive simulations, significantly reducing the need for interactions with the full-order model. Additionally, it yields an interpretable and parsimonious dynamics model within a low-dimensional surrogate space. When setting the dimensions of the latent spaces, N_x^{Lat} and N_u^{Lat}, representing the reduced-order state and control, respectively, we distinguish between two scenarios:

- **Intrinsic dimension known a priori.** In some applications, domain knowledge or prior analysis provides insight into the intrinsic dimension of the system's solution manifold. For example, in Section 4.1, Burgers' equation evolves over a one-dimensional spatial domain and time, suggesting

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

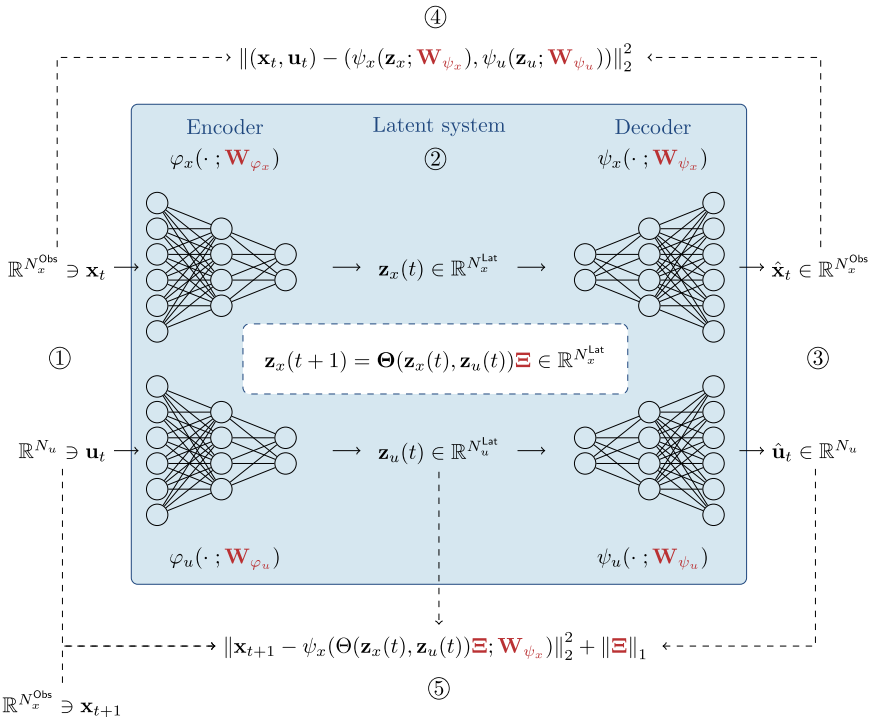


Figure 2. *AE architecture and loss function used during the training stage. Trainable parameters are highlighted in red. The different stages of the training scheme can be listed as follows. (1) the current state* $\mathbf{x}_t$*, applied control* $\mathbf{u}_t$*, and the next state* $\mathbf{x}_{t+1}$ *are provided as input data. (2) After compressing both the current state and the control vector, the SINDy-C algorithm is applied in the latent space, yielding a low-dimensional representation of the prediction for the next state. (3) The latent space representations of the current state, the control, and the next state prediction are decoded. (4) The classical AE loss is computed. (5) The SINDy-C loss and a regularization term to promote sparsity are computed. Figure inspired by Conti et al. (2023, figure 1).*

a low-dimensional solution manifold of intrinsic dimension two. Accordingly, we choose $N_x^{\text{Lat}} = 2$ and set $N_u^{\text{Lat}} = 2$ for reasons of symmetry. This choice is validated by the model's ability to accurately reconstruct the dynamics.

- **Intrinsic dimension unknown.** In the absence of prior information, N_x^{Lat} and N_u^{Lat} are treated as hyperparameters to be inferred from data. In this setting, we adopt a data-driven strategy to explore and tune these dimensions, aiming to approximate the minimal latent space that captures the essential dynamics and control dependencies. For example, in Section 4.2, we experiment with multiple choices of latent dimensions to estimate a lower bound on the dimensionality of the surrogate representation. This tuning process enables the discovery of a compact latent model solely from observational data.

In support of this approach, our results in Section 4 show that AE+SINDy-C is capable of identifying latent dimensions that are consistent with the underlying structure of the control space, even when no prior information is used.

We operate in the (discrete-time) SINDy-C setting (see Section 2.3.1) with $\mathbf{x}_k = (\mathbf{x}(t_k), \mathbf{u}(t_k))$ and $\mathbf{y}_k = \mathbf{x}(t_{k+1})$. Here, $\mathbf{x}(t_k) \in \mathbb{R}^{N_x^{\text{Obs}}}$, where $N_x^{\text{Obs}} = N_x$ in the fully observable case, and $N_x^{\text{Obs}} \ll N_x$ in the partially observable case (compared to Figure 1, we simplify the $\mathbf{x}_t^{\text{Obs}}$ notation by omitting the superscript), respectively, and $\mathbf{u}(t_k) \in \mathbb{R}^{N_u}$. Overall, we are interested in the discovery task $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$.

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Instead of applying SINDy-C directly on our measurements, we follow the idea of Champion et al. (Champion et al., 2019) to first compress the states and actions using the encoder, then apply SINDy-C in the low-dimensional latent space, predict the next state, and subsequently decompress the obtained prediction back to the observation space using the decoder.

3.1.1. Offline training

As already described in the previous section, the overall framework is shown in Figure 2 and can be divided into three steps:

(i) **Encoding:** The observed state $\mathbf{x}_t \in \mathbb{R}^{N_x^{\text{Obs}}}$ and the current control $\mathbf{u}_t \in \mathbb{R}^{N_u}$ are individually compressed by two separated encoder networks $\varphi_x(\cdot;\mathbf{W}_{\varphi_x}) : \mathbb{R}^{N_x^{\text{Obs}}} \to \mathbb{R}^{N_x^{\text{Lat}}}$ and $\varphi_u(\cdot;\mathbf{W}_{\varphi_u}) : \mathbb{R}^{N_u} \to \mathbb{R}^{N_u^{\text{Lat}}}$, yielding their low-dimensional latent-space equivalent $\mathbf{z}_x(t) \in \mathbb{R}^{N_x^{\text{Lat}}}$ and $\mathbf{z}_u(t) \in \mathbb{R}^{N_u^{\text{Lat}}}$, respectively.

(ii) **Discrete SINDy-C in the latent space:** Given a set of d dictionary functions (e.g., polynomials, trigonometric functions) the dictionary $\mathbf{\Theta}(\mathbf{z}_x(t),\mathbf{z}_u(t)) = \mathbf{\Theta}(\varphi_x(\mathbf{x}_t;\mathbf{W}_{\varphi_x}),\varphi_u(\mathbf{u}_t;\mathbf{W}_{\varphi_u})) \in \mathbb{R}^d$ is evaluated and multiplied by $\mathbf{\Xi} \in \mathbb{R}^{d \times N_x^{\text{Lat}}}$, that is, the coefficients to be fit.

(iii) **Decoding:** The result $\mathbf{z}_x(t+1) = \mathbf{\Theta}(\mathbf{z}_x(t),\mathbf{z}_u(t))\mathbf{\Xi} \in \mathbb{R}^{N_x^{\text{Lat}}}$ is then decoded to obtain a prediction for the next state of the high-dimensional system by using the state-decoder $\psi_x(\cdot;\mathbf{W}_{\psi_x}) : \mathbb{R}^{N_x^{\text{Lat}}} \to \mathbb{R}^{N_x^{\text{Obs}}}$. To train the AE, the compressed state and action are also decoded and fed into the loss function by using the state-decoder and the control-decoder $\psi_u(\cdot;\mathbf{W}_{\psi_u}) : \mathbb{R}^{N_u^{\text{Lat}}} \to \mathbb{R}^{N_u}$.

Overall, we obtain the following loss function:

$$\begin{aligned}\min_{\mathbf{W}_{\varphi_x},\mathbf{W}_{\varphi_u},\mathbf{W}_{\psi_x},\mathbf{W}_{\psi_u},\mathbf{\Xi}} \quad & \underbrace{\left\|\mathbf{x}_{t+1} - \psi_x\left(\mathbf{\Theta}\left(\varphi_x(\mathbf{x}_t;\mathbf{W}_{\varphi_x}),\varphi_u(\mathbf{u}_t;\mathbf{W}_{\varphi_u})\right)\cdot\mathbf{\Xi};\mathbf{W}_{\psi_x}\right)\right\|_2^2}_{\text{forward SINDy-C prediction loss}} \\ & + \lambda_1 \underbrace{\left\|\mathbf{x}_t - \psi_x\left(\varphi_x(\mathbf{x}_t;\mathbf{W}_{\varphi_x});\mathbf{W}_{\psi_x}\right)\right\|_2^2}_{\text{autoencoder loss state}} \\ & + \lambda_2 \underbrace{\left\|\mathbf{u}_t - \psi_u\left(\varphi_u(\mathbf{u}_t;\mathbf{W}_{\varphi_u});\mathbf{W}_{\psi_u}\right)\right\|_2^2}_{\text{autoencoder loss control}} \\ & + \lambda_3 \underbrace{\|\mathbf{\Xi}\|_1}_{\text{sparsity regularization}} ,\end{aligned} \tag{3.1}$$

representing the loss of the decoded forward prediction in the SINDy-C latent space formulation, that is, parts (ii) and (iii), the classical AE loss and also the regularization loss to promote sparsity in the SINDy-coefficients. Since our entire pipeline is implemented in the *PyTorch* library (Paszke et al., 2017), we train eq. (3.1) by using the automatic differentiation framework and thus do not rely on STLS. We use Kaptanoglu et al. (2021)) to create the set of d dictionary functions once in the beginning of the pipeline. The parameters $\lambda_1, \lambda_2 \in \mathbb{R}_{>0}$ are hyperparameters to individually weight the contribution of each term.

3.1.2. Online deployment

Following the training stage, where the parameters $\mathbf{W}_{\varphi_x}$, $\mathbf{W}_{\varphi_u}$, $\mathbf{W}_{\psi_x}$, $\mathbf{W}_{\psi_u}$, and $\mathbf{\Xi}$ are learned, the trained network can be deployed as a Dyna-style environment approximation to train the DRL agent. The surrogate environment enables extremely fast inferences and, if well-trained, provides accurate predictions of the system dynamics. Given a current state $\mathbf{x}_t \in \mathbb{R}^{N_x^{\text{Obs}}}$ (either observed or simulated) and a control $\mathbf{u}_t \sim \pi(\cdot|\mathbf{x}_t)$, the prediction of the next state is computed as follows:

$$\widehat{\mathbf{x}}_{t+1} = \psi_x\left(\mathbf{\Theta}\left(\varphi_x(\mathbf{x}_t;\mathbf{W}_{\varphi_x}),\varphi_u(\mathbf{u}_t;\mathbf{W}_{\varphi_u})\right)\cdot\mathbf{\Xi};\mathbf{W}_{\psi_x}\right) \approx f(\mathbf{x}_t,\mathbf{u}_t),$$

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

which involves only one matrix multiplication, the evaluation of d dictionary functions, and three neural network forward passes, resulting in exceptionally low inference times.

3.2. DRL training procedure

We describe the usage of AE+SINDy-C in a Dyna-style MBRL in Algorithm 1. A key hyperparameter in this approach is k_{dyn} (line 7), which controls how many times the agent is trained on the surrogate model before collecting new data from the full-order environment. Specifically, the agent is trained $k_{\text{dyn}} - 1$ times per iteration on the surrogate. Choosing k_{dyn} involves a trade-off: a larger k_{dyn} increases sample efficiency by maximizing learning from the surrogate but may lead to instability or degraded performance if the surrogate model is imperfect. Conversely, a smaller k_{dyn} reduces this risk at the price of more frequent interactions with the environment, increasing computational cost and data requirements. In our experiments, k_{dyn} was selected via preliminary tuning to achieve a balance between stable convergence, learning speed, and efficient data usage. Future work could explore adaptive strategies that adjust k_{dyn} dynamically based on surrogate model accuracy or policy performance metrics to optimize training. We use the PPO algorithm (Schulman et al., 2017) as a state-of-the-art actor-critic policy. As in Zolman et al. (n.d.)), we also use the PPO gradients to update the parameters of the autoencoder surrogate model.

To also correctly emulate the partial observability, that is, $N_x^{\text{Obs}} < N_x$, in the reward function, we project the target state into the lower-dimensional observation space and compute the projected reward. Namely, for Burgers' equation, we compute $\left(\mathbf{x}_t^{\text{Obs}} - C \cdot \mathbf{x}_{\text{ref}}\right)^\top Q^{\text{Proj}} \left(\mathbf{x}_t^{\text{Obs}} - C \cdot \mathbf{x}_{\text{ref}}\right)$, where Q^{Proj} is a scaled identity matrix in the lower-dimensional observation space. This procedure makes the training more challenging since we only obtain partial information via the reward function. In the case of $N_x^{\text{Obs}} = N_x$ and for the evaluation of all final models, we use the closed form of the full-order reward function.

1: **Algorithm 1** Dyna-style MBRL using AE+SINDy-C.
Require: $\mathcal{E}$ full-order env.(FOE), π_0 init.policy, $\Theta, N_{\text{off}}, N_{\text{collect}}, n_{\text{batch}}, k_{\text{dyn}}$.
2: $D_{\text{off}} = \text{CollectData}(\mathcal{E}, \pi_0, N_{\text{off}})$ `// Off-policy data using FOE`
3: $D = \text{InitializeDatastore}(D_{\text{off}})$
4: $\hat{\mathcal{E}} = \text{AE} + \text{SINDy-C}(D, \Theta)$ `// Fit surrogate env.`
5: $\pi = \text{InitializePolicy}()$
6: **while** not done **do**
7: **for** $k_{\text{dyn}} - 1$ **steps do**
8: $\pi = \text{PPO.update}\left(\hat{\mathcal{E}}, \pi, n_{\text{batch}}\right)$ `// Train agent with surrogate env.`
9: **end for**
10: $D_{\text{on}} = \text{CollectData}(\mathcal{E}, \pi, N_{\text{collect}})$ `// Collect on-policy data`
11: $D = \text{UpdateStore}(D, D_{\text{on}})$
12: $\hat{\mathcal{E}} = \text{AE} + \text{SINDy-C}(D, \Theta)$ `// Update surrogate env.`
13: **end while**
14: **return** π

4. Numerical experiments

To validate our approach, we study Burgers' equation and the Navier–Stokes equations on two test benchmark implementations provided by the well-established *ControlGym* library (Zhang et al., n.d.) and the *PDEControlGym* library (Bhan et al., 2024), respectively. In both cases, we compare our Dyna-style MBRL Algorithm 1 with the model-free PPO baseline which only interacts with the full-order environment. Burgers' equation serves as an initial example to highlight the data efficiency of the algorithm, explore both partially and fully observable cases, examine robustness to noisy observations, and experiment with discovering control dimensions in the latent space. While the model-free PPO baseline achieves slightly better final performance (cf. Table 1), AE+SINDy-C reaches competitive results using

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Table 1. *Performance comparison of the Dyna-style AE+SINDy-C method for Burgers' equation*

	Baseline		AE+SINDy-C, $k_{\text{dyn}} = 5$		AE+SINDy-C, $k_{\text{dyn}} = 10$	
	PO48 × 8	FO256 × 8	PO48 × 8	FO256 × 8	PO48 × 8	FO256 × 8
#FOM interactions	12,000	12,000	2400	2400	**1200**	**1200**
Off-policy	-	-	200	200	200	200
On-poliy	12,000	12,000	2200	2200	100	1000
Reward $\mathcal{R}$						
Random Init ($\mu \pm \sigma^2$)						
$t \in [0\text{s}, 1\text{s}]$	−85.42	**−8.52**	−51.95	−17.24	−82.52	−17.22
	±37.13	**±3.47**	±25.03	±12.81	±24.77	±12.39
$t \in (1\text{s}, 5\text{s}]$	−14.52	**−2.61**	−6.50	−10.62	−10.39	−45.29
	±8.09	**±1.51**	±4.54	±3.86	±8.22	±7.65
Bell Function Init ($\mu \pm \sigma^2$)						
$t \in [1\text{s}, 2\text{s}]$	**−31,551.70**	−32,817.89	−44,054.61	−42,032.25	−47,417.93	−45,027.72
	±10,772.44	±11,513.03	±13,382.10	±13,418.30	±14,071.58	±13,936.04
$t \in (2\text{s}, 6\text{s}]$	**−2642.00**	−2778.67	−23165.13	−15,335.65	−35,431.61	−25,302.46
	±1055.71	±1313.07	±8099.89	±5967.48	±10,610.13	±8719.55
Total #parameters	**13,705**	66,953	14,905	78,727	14,905	78,727
AE + SINDy-C	-	-	1200	11,774	1200	11,774
Actor + Critic	13,705	66,953	13,705	66,953	13,705	66,953

Note: We test $k_{\text{dyn}} = 5, 10$ against the full-order baseline for the partially observable (PO) and FO case. The models correspond to the dashed vertical lines in Figure 3 and represent all models after 100 epochs. We compare the number of full-order model (FOM) interactions, the performance for a random initialization (cf. Appendix A.1), the bell-shape initialization and $v = 0.01$ (cf. Section 4.1.2), and the total number of parameters. Best performances (bold) are highlighted row-wise. For the evaluation the performance over five fixed random seeds is used.

significantly fewer full-order interactions, demonstrating its advantage in terms of sample efficiency. Additionally, we emphasize the method's strengths in out-of-distribution generalization for the initial condition, interpretability of the learned surrogate dynamics, and details about the AE's training, which are covered in Section 4.1. In contrast, the Navier–Stokes equations offer a more challenging example, with only boundary controls and a much higher-dimensional state space, underscoring AE+SINDy-C's scalability. This case will showcase how the method can be applied to complex, high-dimensional systems, discussed in Section 4.2.

Since AE+SINDy-C serves as a Dyna-style MBRL algorithm, we analyze the efficiency of the proposed framework with respect to the number of full-order model interactions. For all our experiments, we use the *Ray RLLib* package (Liang et al., 2017); AE+SINDy-C is implemented in *PyTorch* (Paszke et al., 2017). All details about the PDE environments parameters are available in Appendix B. For further details on the DRL training, we refer to Appendix C; specific details about the AE training can be found in Appendix D.

4.1. Burgers' equation (ControlGym)

The viscous Burgers' equation can be interpreted as a one-dimensional, simplified version of the Navier–Stokes equations, describing fluid dynamics and capturing key phenomena such as shock formations in water waves and gas dynamics. Solving Burgers' equation is particularly challenging for RL and other numerical methods due to its nonlinearity and the presence of sharp gradients and discontinuities. Given a spatial domain $\Omega = [0, L] \subset \mathbb{R}$ and a time horizon T, we consider the evolution of continuous fields $\mathbf{x}(x,t) : \Omega \times [0,T] \to \mathbb{R}$ under a the temporal dynamics:

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

$$\begin{aligned} \partial_t \mathbf{x}(x,t) + \mathbf{x}(x,t)\partial_x \mathbf{x}(x,t) - \nu \partial_{xx}^2 \mathbf{x}(x,t) &= \mathbf{u}(x,t) \\ \mathbf{x}(x,0) &= \mathbf{x}_{\text{init}}(x) \end{aligned}, \tag{4.1}$$

given an initial state $\mathbf{x}_{\text{init}} : \Omega \to \mathbb{R}$, a constant diffusivity $\nu > 0$, a source term (also called forcing function) $\mathbf{u}(x,t) : \Omega \times [0,T] \to \mathbb{R}$, and periodic boundary conditions. Internally, *ControlGym* discretizes the PDE in space and time, assuming uniformly spatially distributed control inputs $\mathbf{u}_t$, which are piecewise constant over time. This results in a discrete-time finite-dimensional nonlinear system of the form:

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t; \mathbf{w}_k), \tag{4.2}$$

including (optional) Gaussian noise $\mathbf{w}_k$ (to be precise, the discrete dynamics f also depend on the mesh size of the discretization). In this work, the internally used solution parameters are set to their default and for more details we refer to Zhang et al. (n.d.), p. 6) and Appendix B. As previously mentioned, since our new encoding-decoding scheme does not rely on derivatives, we explicitly want to test the robustness of AE+SINDy-C with respect to observation noise. Thus, we follow the observation process by *ControlGym* given by

$$\mathbf{x}_t^{\text{Obs}} = C \cdot \mathbf{x}_t + \mathbf{v}_t,$$

where the matrix $C \in \mathbb{R}^{N_x^{\text{Obs}} \times N_x}$ is as described in Figure 1 and $\mathbf{v}_t \sim \mathcal{N}(0, \Sigma_v)$ is zero-mean Gaussian noise with symmetric positive definite covariance matrix $\Sigma_v = \sigma^2 \cdot \text{Id}_{N_x^{\text{Obs}}} \in \mathbb{R}^{N_x^{\text{Obs}} \times N_x^{\text{Obs}}}, \sigma^2 > 0$. For both problems, we define a target state $\mathbf{x}_{\text{ref}}$ and use the following objective function to be maximized:

$$\mathcal{J}(\mathbf{x}_t, \mathbf{u}_t) = -\mathbb{E}\left\{ \sum_{t=0}^{K-1} \left[\left(\mathbf{x}_t^{\text{Obs}} - \mathbf{x}_{\text{ref}}\right)^\top Q \left(\mathbf{x}_t^{\text{Obs}} - \mathbf{x}_{\text{ref}}\right) + \mathbf{u}_t^\top R \mathbf{u}_t \right] \right\},$$

with positive definite matrices $Q \in \mathbb{R}^{N_x^{\text{Obs}} \times N_x^{\text{Obs}}}$ and $R \in \mathbb{R}^{N_u \times N_u}$ balancing the control effort and the tracking performance, and $K \in \mathbb{N}$ denotes the number of discrete time steps. As reference state and controls, we both use the zero-vector. The inherent solution manifold of eq. (4.1) is two-dimensional, representing both space and time.

In this work, the diffusivity constant $\nu = 1.0$ is fixed and we choose $\Omega = (0,1)$ as spatial domain, that is, we select $L = 1$. In this example, we set the reduced dimension to $N_x^{\text{Lat}} = 2$, as Burgers' equation depends on both space and time and is known to exhibit a solution manifold of (approximately) intrinsic dimension two. For symmetry, we choose $N_u^{\text{Lat}} = 2$. In cases where such prior information is not available, these dimensions can be treated as tunable parameters and inferred from data, as discussed in Section 3.1.

The target state of $\mathbf{x}_{\text{ref}} \equiv \mathbf{0} \in \mathbb{R}^{N_x^{\text{Obs}}}$ is used. In Section 4.1.3, we will see how AE+SINDy-C is independently able to compress the controls even further.

For Burgers' equation, we trained AE+SINDy-C with different full-order model update frequencies $k_{\text{dyn}} = 5, 10, 15$ to analyze the sample efficiency and the sensitivity of the surrogate model. In the case $k_{\text{dyn}} = 15$, the internal modeling bias was too big for a successful DRL training, which is the reason why this case is excluded from the analysis. This appears to be a natural limitation since the number of full-order interactions was too low and thus not sufficient to adequately model the system dynamics under investigation.

4.1.1. Sample efficiency and speed-ups

As in the training, the initial condition is given by a uniform $\mathcal{U}\left([-1,1]^{N_x}\right)$ distribution; for more details, we refer to Appendix A.1. We use an evaluation time horizon of $T_{\text{eval}} = 1\text{s}$ and let the agent extrapolate over time for four more seconds, that is, $T_{\text{extrap}} = 4\text{s}$, resulting in $T = 5\text{s}$, four times more than the training horizon. Figure 3 shows the average reward ($\pm$ one standard deviation) given the number of full-order interactions each model has observed. For a quantitative overview, we refer to Table 1, where we display the results with further detail. We separately illustrate the performance on the evaluation horizon $[0s, 1s]$, that is, the time horizon the agent interacts for during the training, and the extrapolation horizon $(1s, 5s]$. The vertical dashed lines in Figure 3 represent the models after 100 epochs and are

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

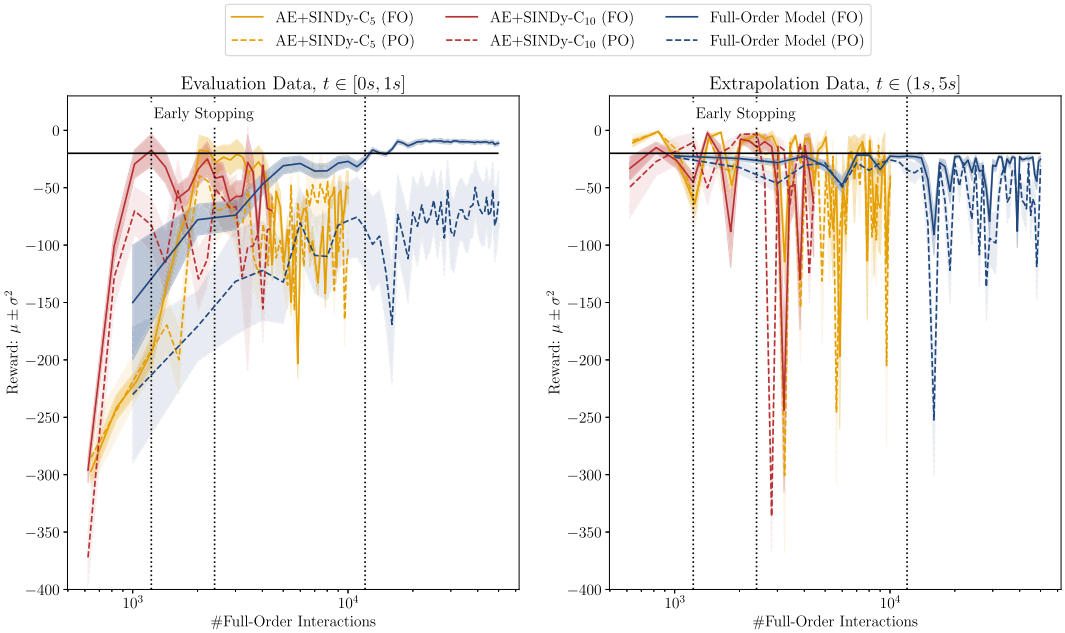


Figure 3. *Sample efficiency of the Dyna-style AE+SINDy-C method for Burgers' equation. We test* $k_{\text{dyn}} = 5, 10$ *against the full-order baseline for the fully observable (solid line) and partially observable (dashed line). The dashed vertical lines indicate the point of early stopping for each of the model classes (FO + PO) after 100 epochs and represent the models which are evaluated in detail in Appendix A.1. For the evaluation, the performance over five fixed random seeds is used.*

deployed to evaluate the agent's performance, that is, Table 1. We save all intermediate models and stop the training for all models after exactly 100 epochs, which is approximately the point where the extrapolation performance drops, that is, performing early stopping. In the case of the AE+SINDy-C algorithm, this performance drop is also evident in the evaluation data, caused by the high sensitivity of the online training of the AE.

In general, we can see that all of the models overfit the dynamics. While the performance on the evaluation time interval still improves, the model becomes very unstable when extrapolating in time and at the end of the training interval almost all model solutions diverge. As expected, generally the model fully observing the PDE outperforms the model that only observes partial measurements. Additionally, the partially observable cases exhibit a higher standard deviation and thus higher variability in their performance.

Partially observable. Generally, the models only partially observing the system exhibit worse performances and suffer under higher variability. At the end of the training horizon, in both cases, the AE+SINDy-C method overfits the dynamics model, resulting in divergent solutions when extrapolating in time. The highest performance for each of the PO cases is almost similar, although AE+SINDy-C with $k_{\text{dyn}} = 10$ requires nearly 10 times less data compared to the full-order model, and with $k_{\text{dyn}} = 5$ close to five times less data, respectively (cf. Table 1)—matching the intuition of sample efficiency of Algorithm 1 based on k_{dyn}.

Fully observable. As already mentioned, a direct comparison all of the trained models in the FO case outperforms their PO counterparts, quantitatively and qualitatively. As visualized in Table 1, and similarly to the PO case, we achieve nearly the same level of performance with 10 times, respectively, five times, less data.

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

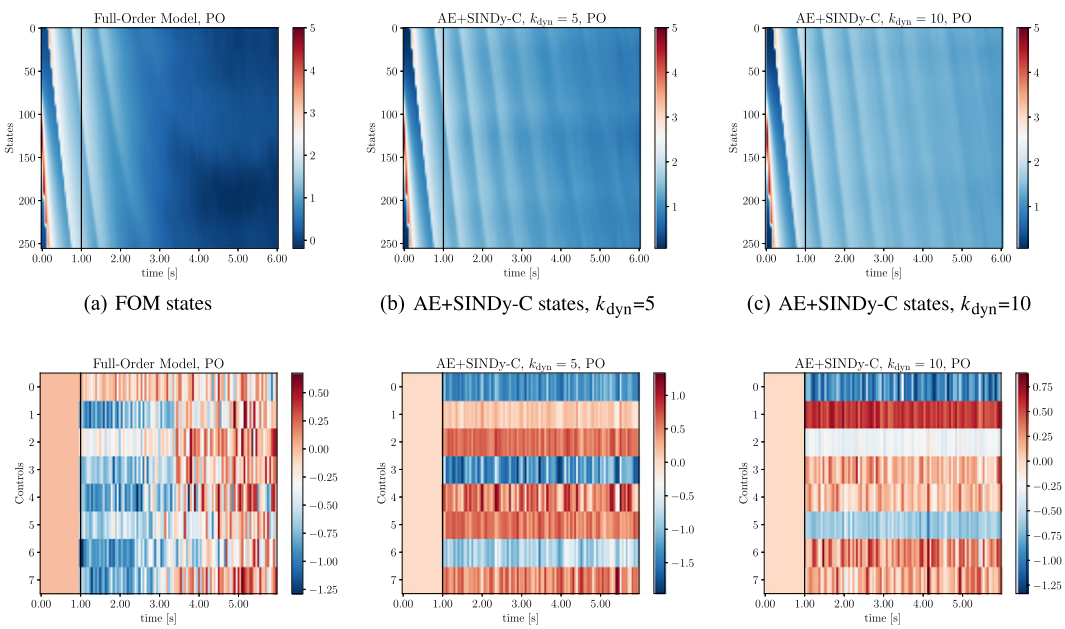
4.1.2. Generalization: Variation of initial condition and diffusivity constant

Compared to the $\mathcal{U}\left([-1,1]^{N_x}\right)$ distribution, we used during the training phase and the detailed analysis in Appendix A.1, we are also interested in the generalization capabilities of the agents trained with AE +SINDy-C on out-of-distribution initial conditions with more regularity. Given N_{spatial} discretization points in space, we modify the default initial condition of the *ControlGym* library and define the initial state

$$(\mathbf{x}_{\text{init}})_i = 5 \cdot \frac{1}{\cosh(10(\Delta_i - \alpha))} \in [0,5], \quad \Delta_i = \frac{i}{N_{\text{spatial}}}, i = 0, \ldots, N_{\text{spatial}}, \tag{4.3}$$

for our second test case, exhibiting a higher degree of regularity. The term $\alpha \sim \mathcal{U}[0.25, 0.75]$ randomly moves the peak of the curve, enabling the ablation studies presented in Table 1. For the plots reported in Figures 4 and 5, we centered the function in the domain by taking $\alpha = 0.5$. With a nonnegative domain of $[0,5]$, this initial state clearly exceed the training domain of $[-1,1]$. To make the test case more interesting and more realistic, we also change the diffusivity constant to $\nu = 0.01$, that is, two orders of magnitude smaller than on the training data, and we let the PDE evolve uncontrolled for 1 s before the controller starts actuating for five more seconds, that is, a delayed start and a total observation horizon of $T = 6$s of which the controller is activated in $[1\text{s}, 6\text{s}]$.

Clearly, this task is much more challenging than the one previously considered, As in the first test setting, the fully observable case seems to be overall easier for all of the models, although the differences between the fully and partially observable case are more significant. Most importantly, as discussed in Appendix A.1, working with a regular initial state, the agent also generates regular control trajectories. Although we did not apply actuation bounds, the order of magnitude of the controls is almost the same as in the previous test case, indicating a very robust generalization of the policy.

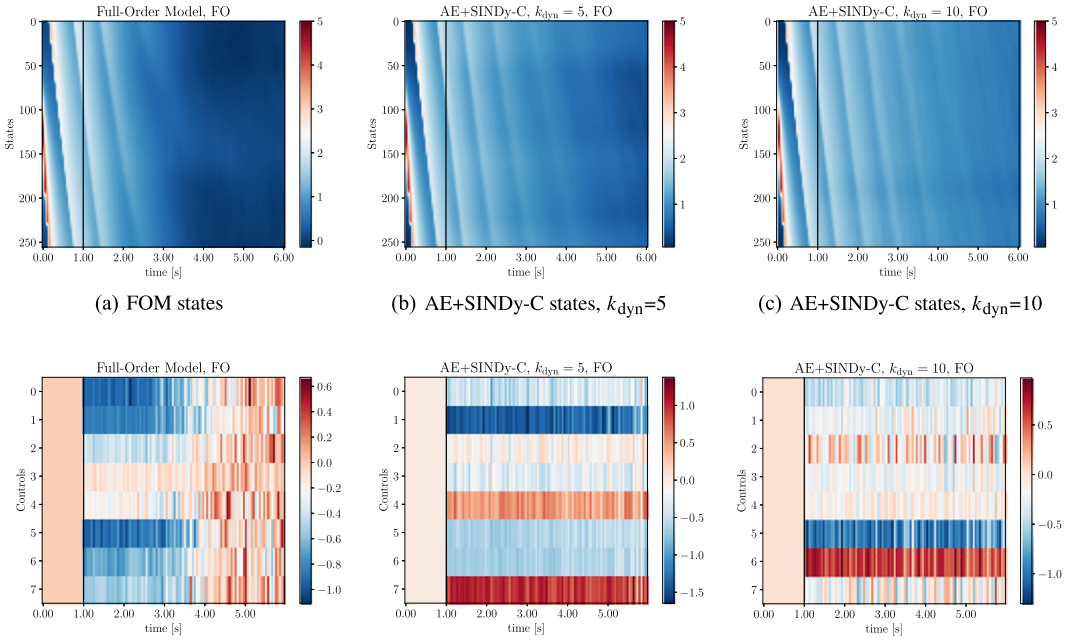


(a) FOM states (b) AE+SINDy-C states, k_{dyn}=5 (c) AE+SINDy-C states, k_{dyn}=10

(d) FOM controls (e) AE+SINDy-C controls, k_{dyn}=5 (f) AE+SINDy-C controls, k_{dyn}=10

Figure 4. *State and control trajectories for Burgers' equation in the PO case. The initial condition is a bell-shaped hyperbolic cosine (eq. [4.3] with $\alpha = 0.5$ fixed), we use $\nu = 0.01$ (two orders of magnitude smaller compared to the training phase), and the black solid line indicates the timestep t when the controller is activated. (a) FOM states. (b) AE + SINDy-C states, $k_{\text{dyn}} = 5$. (c) AE + SINDy-C states, $k_{\text{dyn}} = 10$. (d) FOM controls. (e) AE + SINDy-C controls, $k_{\text{dyn}} = 5$. (f) AE + SINDy-C controls, $k_{\text{dyn}} = 10$.*

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press



(a) FOM states (b) AE+SINDy-C states, k_{dyn}=5 (c) AE+SINDy-C states, k_{dyn}=10

(d) FOM controls (e) AE+SINDy-C controls, k_{dyn}=5 (f) AE+SINDy-C controls, k_{dyn}=10

Figure 5. *State and control trajectories for Burgers' equation in the FO case. The initial condition is a bell-shaped hyperbolic cosine (eq. [4.3] with $\alpha = 0.5$ fixed), we use $\nu = 0.01$ (two orders of magnitude smaller compared to the training phase), and the black solid line indicates the timestep t when the controller is activated. (a) FOM states. (b) AE + SINDy-C states, $k_{\text{dyn}} = 5$. (c) AE + SINDy-C states, $k_{\text{dyn}} = 10$. (d) FOM controls. (e) AE + SINDy-C controls, $k_{\text{dyn}} = 5$. (f) AE + SINDy-C controls, $k_{\text{dyn}} = 10$.*

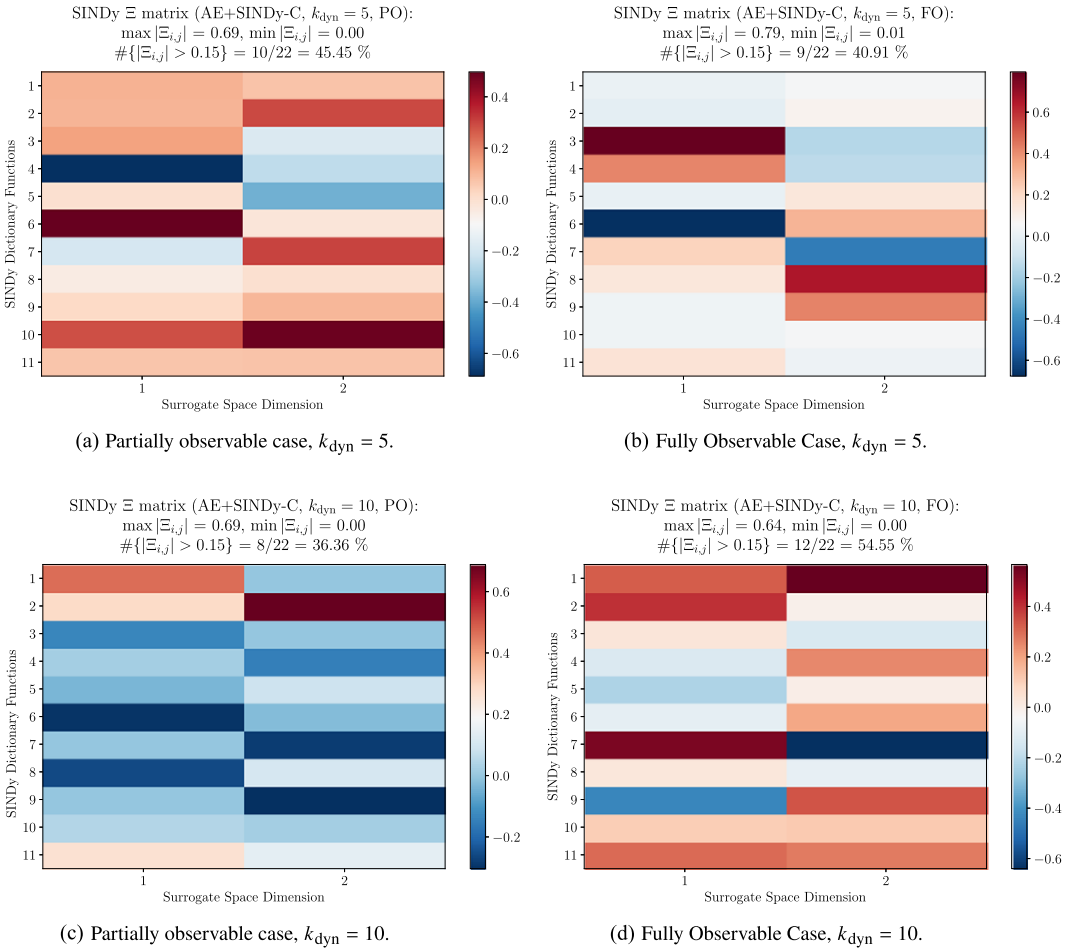
Partially observable. In the PO case (cf. Figure 4), the baseline model clearly outperforms the AE +SINDy-C method. While, for both values of $k_{\text{dyn}} = 5, 10$, the agent is able to stabilize the PDE toward a steady state and slowly decreases it is constant, only the baseline agent fully converges toward to the desired zero-state. Interestingly, the AE+SINDy-C controllers seem to focus on channel-wise high and low control values, while the full-order baseline agents exhibit a switching behavior after circa 3 s for seven out of eight actuators. Similar to the case of the uniform initial condition, the AE+SINDy-C with $k_{\text{dyn}} = 10$ does not seem to rely on actuator two.

Fully observable. As in the PO case, the FO test case (cf. Figure 5) shows a similar pattern. The baseline model outperforms both the AE+SINDy-C agents. In contrast to the PO case, now also the two agents trained with the surrogate model are close to the zero-state target, with $k_{\text{dyn}} = 5$ slightly outperforming the $k_{\text{dyn}} = 10$ agent. Notably, parts of the controls in all models exhibit zero-like actuators, the baseline model again showing a switching behavior after circa three and a half seconds. It seems like having access to more state measurements requires the usage of less actuators, resulting in a higher reward, as displayed in Table 1. For real-world applications, this can be really useful when designing position and size of controllers for an actual system.

4.1.3. Dynamics in the latent space

A relevant aspect of AE+SINDy-C is the compression of high-dimensional PDE discretization into a low-dimensional surrogate space, resulting in a closed-form and interpretable representation and the potential to discover unknown system dynamics. The coefficient matrices $\Xi \in \mathbb{R}^{d \times N_x^{\text{Obs}}}$ are visualized in Figure 6 for Burgers' case, highlighting the coefficients as a heatmap as well as some key points of interests (e.g., sparsity).

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press



(a) Partially observable case, $k_{\text{dyn}} = 5$.

(b) Fully Observable Case, $k_{\text{dyn}} = 5$.

(c) Partially observable case, $k_{\text{dyn}} = 10$.

(d) Fully Observable Case, $k_{\text{dyn}} = 10$.

Figure 6. *Analysis of the coefficient matrix $\boldsymbol{\Xi} \in \mathbb{R}^{d \times N_x^{\text{Obs}}}$ for Burgers' equation. (a) Partially observable case, $k_{\text{dyn}} = 5$. (b) Fully observable case, $k_{\text{dyn}} = 5$. (c) Partially observable case, $k_{\text{dyn}} = 10$. (d) Fully observable case, $k_{\text{dyn}} = 10$.*

Based on the coefficients $\boldsymbol{\Xi}$, one can compute the closed-form representation of the SINDy-C dynamics model in the surrogate space. Since our algorithm is not trained with sequential thresholding, we chose a threshold of 0.15 to cutoff nonsignificant contributions and improve readability. Exemplarily, we obtain the following surrogate space dynamics for $k_{\text{dyn}} = 5$:

- PO, $k_{\text{dyn}} = 5$:

$$\begin{aligned}
\mathbf{z}_{x,1}(t+1) &= -0.688 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,2}(t) + 0.497 \cdot \mathbf{z}_{x,1}(t)^3 \\
&\quad -0.195 \cdot \mathbf{z}_{x,1}(t)^2\mathbf{z}_{x,2}(t) + 0.288 \cdot \mathbf{z}_{u,1}(t) \\
\mathbf{z}_{x,2}(t+1) &= +0.297 \cdot \mathbf{z}_{x,2}(t) - 0.180 \cdot \mathbf{z}_{x,1}(t)^2 - 0.245 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,2}(t) \\
&\quad -0.381 \cdot \mathbf{z}_{x,2}(t)^2 + 0.304 \cdot \mathbf{z}_{x,1}(t)^2\mathbf{z}_{x,2}(t) + 0.487 \cdot \mathbf{z}_{u,1}(t).
\end{aligned}$$

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

• FO, $k_{\text{dyn}} = 5$:

$$\begin{aligned}\mathbf{z}_{x,1}(t+1) &= +0.795 \cdot \mathbf{z}_{x,1}(t)^2 + 0.420 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,2}(t) - 0.676 \cdot \mathbf{z}_{x,1}(t)^3 \\ &\quad + 0.224 \cdot \mathbf{z}_{x,1}(t)^2 \mathbf{z}_{x,2}(t) + 0.159 \cdot \mathbf{z}_{u,2}(t) \\ \mathbf{z}_{x,2}(t+1) &= +0.310 \cdot \mathbf{z}_{x,1}(t)^3 - 0.454 \cdot \mathbf{z}_{x,1}(t)^2 \mathbf{z}_{x,2}(t) \\ &\quad + 0.655 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,2}(t)^2 + 0.426 \cdot \mathbf{z}_{x,2}(t)^3.\end{aligned}$$

Denoting by $\mathbf{z}_{x,i}$ and $\mathbf{z}_{u,i}$ the ith component of the state and the control in the latent space, respectively, SINDy individually compressed both state and action into a two-dimensional latent space representation. While for the state the corresponding AE requires both dimensions, the control AE correctly compressed the control into a one-dimensional representation. Overall, each of the representations is different, which can be explained by (a) a different training procedure, (b) different full-order interactions observed by the models, and (c) the nonuniqueness of the representation.

4.1.4. Training of AE+SINDy-C

To analyze the goodness of fit of the internal surrogate model, we provide in Table 2 the average training and validation error, that is, eq. (3.1), of the last iteration of each surrogate model update, that is, step 12. We clearly see that keeping the training epochs of AE+SINDy-C low helps to reduce overfitting, and for each of the four methods the order of magnitude of the training and validation loss is the same (internally, we use an 80/20 splitting for training and validation and train for 100 epochs. Debugging plots show that in all cases an upper bound of 30 epochs would be enough to train the AE+SINDy-C representation). Additionally, Table 2 also displays the average duration of a surrogate model update, that is, step 12, which is due to the small number of parameters.

4.2. Incompressible Navier–Stokes Equations (PDEControlGym)

This second example focuses on a much more challenging equation and control setting. The temporal dynamics of the 2D velocity field $\mathbf{x}(y_1, y_2, t) : \Omega \times [0, T] \to \mathbb{R}^2$, and the pressure field $p(y_1, y_2, t) : \Omega \times [0, T] \to \mathbb{R}$ is given by

$$\begin{aligned}&\nabla \cdot \mathbf{x} = 0, \\ &\partial_t \mathbf{x} + \mathbf{x} \cdot \nabla \mathbf{x} = -\frac{1}{\rho} \nabla p + \nu \nabla^2 \mathbf{x},\end{aligned} \tag{4.4}$$

Table 2. *Training time and overview of the loss distribution of eq. (3.1) during the training phase of the Dyna-style AE+SINDy-C method for Burgers' equation*

	AE+SINDy-C, $k_{\text{dyn}} = 5$		AE+SINDy-C, $k_{\text{dyn}} = 10$	
	PO 48×8	FO 256×8	PO 48×8	FO 256×8
Training Time $[s](\mu \pm \sigma^2)$	3.80 ± 1.21	3.80 ± 2.55	4.55 ± 0.71	4.11 ± 0.41
Loss eq. (3.1)$(\mu \pm \sigma^2)$				
Training	$7.53 \cdot 10^{-3}$ $\pm 1.01 \cdot 10^{-4}$	$7.27 \cdot 10^{-3}$ $\pm 1.33 \cdot 10^{-4}$	$8.01 \cdot 10^{-3}$ $\pm 3.21 \cdot 10^{-4}$	$7.81 \cdot 10^{-3}$ $\pm 1.34 \cdot 10^{-4}$
Validation	$8.43 \cdot 10^{-3}$ $\pm 2.88 \cdot 10^{-4}$	$7.44 \cdot 10^{-3}$ $\pm 1.64 \cdot 10^{-4}$	$7.94 \cdot 10^{-3}$ $\pm 6.24 \cdot 10^{-4}$	$9.33 \cdot 10^{-3}$ $\pm 5.22 \cdot 10^{-4}$

Note: Mean and variance are computed over all training epochs. The training was performed on a MacBook M1 (2021, 16GB RAM).

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

in $\Omega \times [0,T]$, denoting the kinematic viscosity of the fluid by ν, and its density by ρ. For the experiments, the equation is fully observable, we consider Dirichlet boundary conditions with velocities set to 0, and we control along the top boundary $u_t = u_t(y_1) = \mathbf{x}(y_1, 1, t)$, for $y_1 \in \Omega$, that is, $y_2 = 1$ is fixed. Following the benchmark of Bhan et al. (2024, section 5.3), the domain $\Omega = (0,1)^2$ and the time interval $T = 0.2$s are used, parameters for the solution procedure are kept. The equation is noise-free fully observed. The DRL agents are trained with a discretized version of the following objective function:

$$\mathcal{J}(u_t, \mathbf{x}) = -\frac{1}{2}\int_0^T \int_\Omega \|\mathbf{x}(y_1,y_2,t) - \mathbf{x}_{\text{ref}}(y_1,y_2,t)\|^2 \mathrm{d}(y_1,y_2)\mathrm{d}t + \frac{\gamma}{2}\int_0^T \left\|u_t - u_t^{\text{ref}}\right\|^2 \mathrm{d}t,$$

where the reference solution $\mathbf{x}_{\text{ref}}$ is given by the resulting velocity field when applying the controls $u : [0,T] \to \mathbb{R}, t \mapsto 3 - 5t$. For the reference controls in the objective function, $u_t^{\text{ref}} \equiv 2.0$ is used.

Based on the superior results from Burgers' equation example, we initially explored larger values of k_{dyn} for the Navier–Stokes case. However, for $k_{\text{dyn}} = 10$, training stability deteriorated and policy performance declined, likely due to the increased complexity of the dynamics and accumulation of surrogate model errors. Consequently, we focus on $k_{\text{dyn}} = 5$ for the Navier–Stokes experiments, which achieves a favorable balance between sample efficiency and training stability. We then compare the Dyna-style MBRL scheme at this setting with the model-free benchmark. To investigate the sensitivity of AE+SINDy-C to the choice of latent dimensions and to estimate a lower bound on the intrinsic dimensionality of the solution manifold, we conduct experiments with varying latent space sizes. Specifically, we train models with three different latent architectures:

- $(882,84,6) \times (1,4,2)$, corresponding to a six-dimensional state and two-dimensional control representation, that is, an eight-dimensional latent space;
- $(882,52,3) \times (1,4,1)$, corresponding to a three-dimensional state and one-dimensional control representation, that is, a four-dimensional latent space; and
- $(882,30,1) \times (1,4,1)$, corresponding to a one-dimensional state and one-dimensional control representation, that is, a two-dimensional latent space.

As expected, reducing the number of degrees of freedom in the latent space negatively impacts the surrogate model's expressiveness and fidelity. In particular, with only two latent state dimensions, the learned representation failed to capture sufficient dynamics for the agent to effectively learn a control policy. As a result, this configuration is excluded from the analysis. These experiments demonstrate how latent space dimensionality can be treated as a tunable parameter, enabling a data-driven estimation of the minimal dimensionality required for successful policy learning.

4.2.1. Sample efficiency and scalability

Figure 7 and Table 3 clearly highlights the sample efficiency of AE+SINDy-C compared to the model-free baseline in the case of an eight-dimensional surrogate space. Not only does AE+SINDy-C need approximately five times less data but simultaneously outperforms the baseline—a clear indication that the internal dynamics model in the latent space helps the agent to understand the underlying system dynamics. The bumps in performances of AE+SINDy-C at approximately 12 k and 20 k FOM interactions represent the point where the dynamics model correctly represents the directions of the flow. Before that, AE+SINDy-C was able to correctly control the magnitude of the flow field and

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

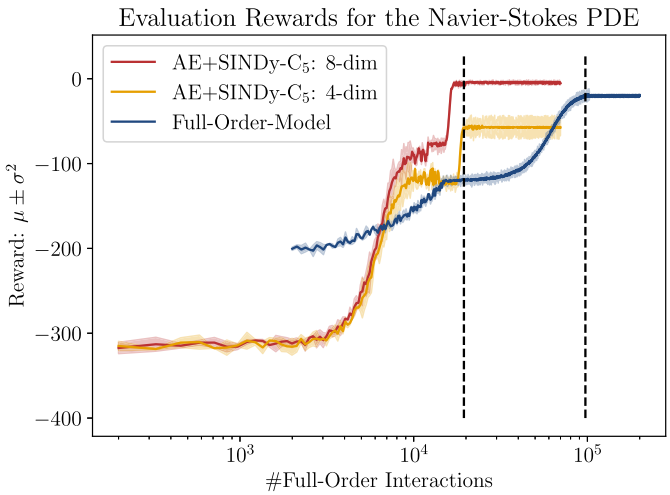


Figure 7. *Sample efficiency of the Dyna-style AE+SINDy-C method for the Navier–Stokes equations. We test $k_{\text{dyn}} = 5$ against the full-order model-free baseline. The dashed vertical lines indicate the point of early stopping for each of the model after 750 epochs and represent the models which are evaluated in detail in Section 4.2.2. For the evaluation, the performance over five fixed random seeds is used.*

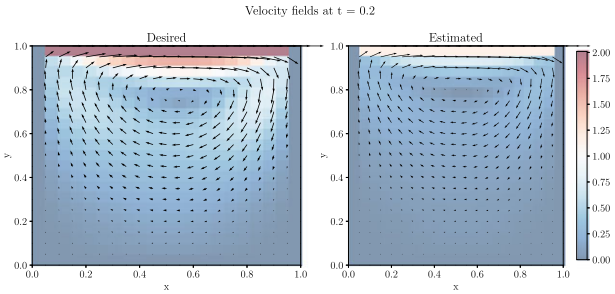
Table 3. *Performance comparison of the Dyna-style AE+SINDy-C method for the Navier–Stokes equations*

		AE + SINDy-C, $k_{\text{dyn}} = 5, (882 \times 1)$	
Surrogate Space Dim	Baseline(882×1)	8	4
# FOM interactions	97,280	**19,456**	**19,456**
Off-policy	-	2000	2000
On-policy	97,280	17,456	17,456
Reward $\mathcal{R}(\mu \pm \sigma^2)$	-24.88 ± 5.77	$\mathbf{-3.61 \pm 1.64}$	-60.09 ± 12.11
Total # parameters	**226,306**	377,601	319,421
AE+SINDy-C	-	151,295	93,115
Actor + Critic	226,306	226,306	226,306

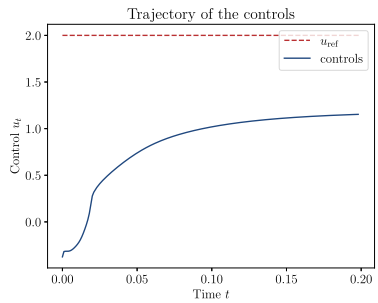
Note: We test $k_{\text{dyn}} = 5$ against the full-order baseline (both $N_x^{\text{Obs}} \times N_u = 882 \times 1$). The models correspond to the dashed vertical lines in Figure 7 and represent all models after 750 epochs. We compare the number of FOM interactions, the reward using five fixed random seeds and the total number of parameters. Best performances (bold) are highlighted row-wise.

adjust the controls, but the orientation of the vector field was reversed. The four-dimensional version is not outperforming the baseline and the final model exhibits high uncertainty, also suffering under very fuzzy controls (cf. Figure 8f). Since the corresponding velocity-field cannot be considered a valid solution (cf. Figure 8e), it is thus excluded from the following analysis. The plateau at around 15 k interactions corresponds to very fuzzy controls and only with a big delay the baseline model can stabilize the controls, missing at the end the correct magnitude of the flow field (cf. Figure 8a). Interestingly, the fuzzy overfitting at the end of the training procedure for Burgers' equation does not appear, which is most probably due to simpler initial conditions and thus more regular systems dynamics, although the dynamics in general are harder to capture, that is, more full-order interactions are needed.

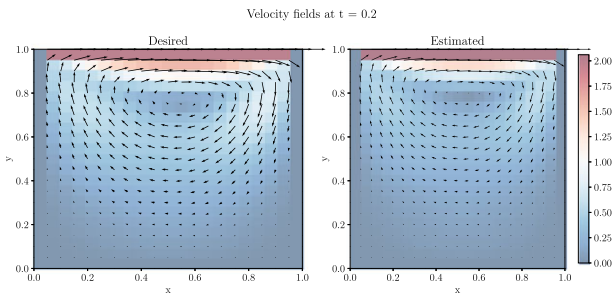
https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press



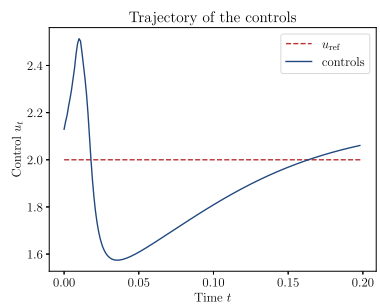
(a) Full-order baseline model, Velocity field at $t = 0.2$.



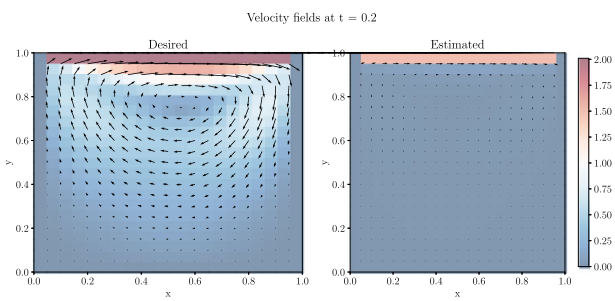
(b) Full-order baseline model, control trajectory.



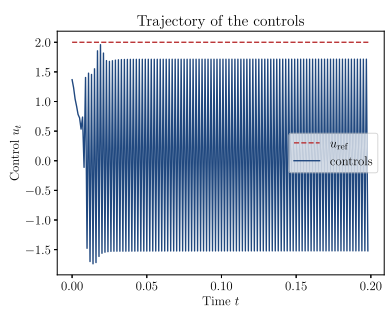
(c) AE+SINDy-C with $k_{\mathrm{dyn}} = 5$ and an 8-dimensional latent space, velocity field at $t = 0.2$.



(d) AE+SINDy-C with $k_{\mathrm{dyn}} = 5$ and an 8-dimensional latent space, control trajectory.



(e) AE+SINDy-C with $k_{\mathrm{dyn}} = 5$ and an 4-dimensional latent space, velocity field at $t = 0.2$.



(f) AE+SINDy-C with $k_{\mathrm{dyn}} = 5$ and an 4-dimensional latent space, control trajectory.

Figure 8. *Velocity field and control trajectories for the model-free baseline and AE+SINDy-C for the Navier–Stokes equations. Black arrows represent the velocity fields and the background color the magnitude of the velocity vector. (a) Full-order baseline model, Velocity field at t = 0.2. (b) Full-order baseline model, control trajectory. (c) AE + SINDy-C with $k_{\mathrm{dyn}} = 5$ and an eight-dimensional latent space, velocity field at ? = 0.2. (d) AE + SINDy-C with $k_{\mathrm{dyn}} = 5$ and an eight-dimensional latent space, control trajectory. (e) AE + SINDy-C with $k_{\mathrm{dyn}} = 5$ and a four-dimensional latent space, velocity field at t = 0.2. (f) AE + SINDy-C with $k_{\mathrm{dyn}} = 5$ and a four-dimensional latent space, control trajectory.*

4.2.2. Velocity field and control analysis

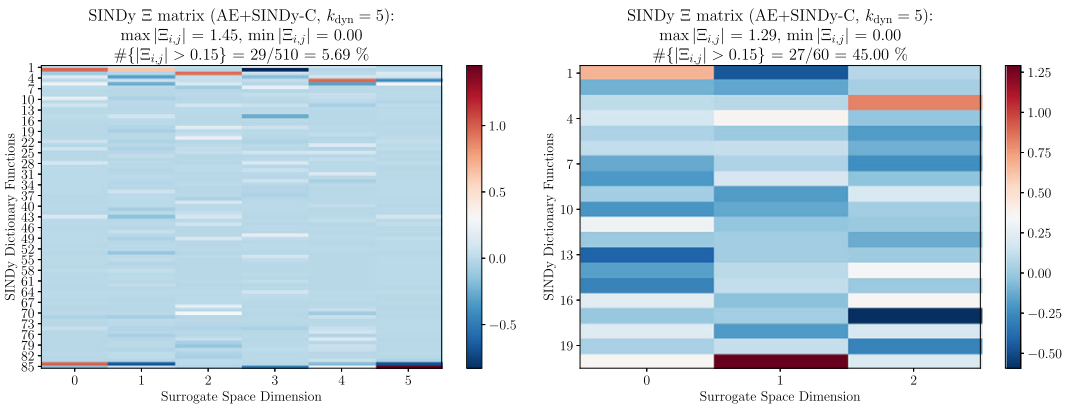
In Figure 8 the resulting velocity fields at the end of the observation horizon are visualized (Bhan et al., 2024, cf. figure 4]. It is evident that AE+SINDy-C nearly perfectly matches the desired flow and clearly outperforms the model-free baseline which has difficulties finding the correct magnitude of the velocity field, although the general directions are correct (since we use with *Ray RLLib* (Liang et al., 2017) instead of *Stable-Baselines3* (Raffin et al., 2021) a different RL-engine and we run PPO

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

with different settings, we are not slightly able to achieve the performance presented in (Bhan et al., 2024, table 3) with our baseline model. But, our AE+SINDy-C outperforms the presented baseline algorithms). Thus, imposing a Dyna-style dynamics model does not only significantly decrease reducing the number of full-order model interactions, however it helps the agent to grasp the general dynamics faster. While the controls of the agent AE+SINDy-C stay close to the reference value, the baseline algorithm has difficulties in smoothly increasing the controls (previous epochs exhibit strongly oscillating behavior, similar to bang-bang controls) (Table 3).

4.2.3. Dynamics in the latent space

As in Burgers' equation example, we visualize the coefficient matrix $\boldsymbol{\Xi} \in \mathbb{R}^{d \times N_x^{\text{Lat}}}$ for the eight-dimensional latent space version. Figure 9 shows the sparse nature of the dynamics model. Interestingly, with 29, and respectively 27, coefficients with absolute values above the threshold of 0.15, both representations need almost the same amount of basis functions, indicating the amount of information carried by the state and action of the PDE.



(a) 8-dimensional Latent space. (b) 4-dimensional Latent space.

Figure 9. *Analysis of the coefficient matrix $\boldsymbol{\Xi} \in \mathbb{R}^{d \times N_x^{\text{Obs}}}$ for the Navier–Stokes equations. (a) Eight-dimensional latent space. (b) Four-dimensional latent space.*

Table 4. *Analysis of the internal loss distribution for the training and validation data during the AE training as well as the training time for the Navier–Stokes equations, trained on a MacBook M1 (2021, 16GB RAM)*

	AE+SINDy-C, $k_{\text{dyn}} = 5, (882 \times 1)$	
Surrogate Dimension	8	4
Training Time $[s] (\mu \pm \sigma^2)$	12.65 ± 5.23	9.81 ± 2.92
Loss eq. (3.1) $(\mu \pm \sigma^2)$		
Training	$1.53 \cdot 10^{-4} \pm 3.01 \cdot 10^{-4}$	$7.57 \cdot 10^{-3} \pm 3.36 \cdot 10^{-4}$
Validation	$9.33 \cdot 10^{-4} \pm 3.82 \cdot 10^{-4}$	$9.11 \cdot 10^{-3} \pm 8.43 \cdot 10^{-4}$

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

We obtain the following dynamics equations for the eight-dimensional latent space:

$$
\begin{aligned}
\mathbf{z}_{x,1}(t+1) &= +0.960 \cdot \mathbf{z}_{x,2}(t) + 0.165 \cdot \mathbf{z}_{x,4}(t) + 0.251 \cdot \mathbf{z}_{x,6}(t) \\
&\quad + 0.228 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,4}(t) + 0.981 \cdot \mathbf{z}_{u,1}(t) \\
\mathbf{z}_{x,2}(t+1) &= +0.631 \cdot \mathbf{z}_{x,2}(t) - 0.308 \cdot \mathbf{z}_{x,4}(t) - 0.245 \cdot \mathbf{z}_{x,6}(t) - 0.652 \cdot \mathbf{z}_{u,1}(t) \\
\mathbf{z}_{x,3}(t+1) &= +0.941 \cdot \mathbf{z}_{x,3}(t) + 0.166 \cdot \mathbf{z}_{x,3}(t)^2 + 0.265 \cdot \mathbf{z}_{x,3}(t)\mathbf{z}_{x,6}(t) \\
&\quad + 0.190 \cdot \mathbf{z}_{x,3}(t)\mathbf{z}_{x,4}(t)^2 + 0.286 \cdot \mathbf{z}_{x,3}(t)\mathbf{z}_{x,4}(t)\mathbf{z}_{x,6}(t) \\
\mathbf{z}_{x,4}(t+1) &= +0.208 \cdot \mathbf{z}_{x,1}(t) - 0.828 \cdot \mathbf{z}_{x,2}(t) - 0.158 \cdot \mathbf{z}_{x,4}(t) \\
&\quad + 0.216 \cdot \mathbf{z}_{x,1}(t)^2 - 0.235 \cdot \mathbf{z}_{x,2}(t)\mathbf{z}_{x,4}(t) \\
&\quad + 0.175 \cdot \mathbf{z}_{x,1}(t)\mathbf{z}_{x,6}(t)^2 - 0.464 \cdot \mathbf{z}_{u,2}(t) \\
\mathbf{z}_{x,5}(t+1) &= +0.949 \cdot \mathbf{z}_{x,5}(t) - 0.290 \cdot \mathbf{z}_{x,6}(t) - 0.177 \cdot \mathbf{z}_{u,1}(t) + 0.310 \cdot \mathbf{z}_{u,2}(t) \\
\mathbf{z}_{x,6}(t+1) &= -0.396 \cdot \mathbf{z}_{x,5}(t) + 0.314 \cdot \mathbf{z}_{x,6}(t) - 0.695 \cdot \mathbf{z}_{u,1}(t) + 1.451 \cdot \mathbf{z}_{u,2}(t)
\end{aligned}
$$

4.2.4. Training of AE+SINDy-C

For the sake of completeness, the goodness of fit in terms of the average training and validation error are provided in Table 4 for the Navier–Stokes case. Compared to Burgers' equation, the training and validation loss are of the same order of magnitude. Since the AE+SINDy-C model is much larger in this case, the training takes longer. Interestingly, while in Burgers' equation case our internal logging metrics show that usually even 20–30 epochs would be enough, in the more complex Navier–Stokes equations case on average at least 70–80 epochs are needed before the loss converges.

5. Discussion and conclusion

We propose a data-efficient and interpretable Dyna-style MBRL framework for controlling distributed systems, such as those governed by PDEs. Combining SINDy-C with an AE framework not only scales to high-dimensional systems but also provides a low-dimensional learned representation as a dynamical system in the latent space. The proposed controllers have proven to be effective and robust in both PO and FO cases. Additionally, we showed how the proposed framework can be used to estimate an approximate lower bound for a low-dimensional surrogate representation of the dynamics.

A limitation of our method is the training of an AE online. Problems such as overfitting (Goodfellow et al., 2016; Zhang et al., 2021) and catastrophic forgetting (McCloskey and Cohen, 1989; Kirkpatrick et al., 2017) lead to decreasing performance if the AE is trained for too long and on too little data. Potential options to overcome this issue could include the usage of shorter roll-outs (Janner et al., 2019), ensemble methods (Zolman et al., n.d.), or training the AE using sequential thresholding to enforce zero-valued coefficients. Burgers' PDE example demonstrated that, provided the initial condition is sufficiently regular, the complexity of learning a reduced representation can be significantly mitigated. In contrast, the Navier–Stokes equation example highlights the critical importance of selecting an appropriate latent space dimensionality. These experiments underscore the need to actively explore different surrogate space configurations in order to avoid inadequate model expressiveness and reduced policy learning performance.

Further research should address these limitations to enhance the robustness and generalizability of our framework across diverse PDE scenarios. Beyond the current improvements, integrating SINDy-C with AEs opens new avenues in multiple fields. In future work, we plan to extend the latent space representation to include parameter dependencies, as seen in (Conti et al., 2023), enabling more effective learning of parametric systems, such as parameterized PDEs.

Data availability statement. The code for AE+SINDy-C is publicly available under https://doi.org/10.5281/zenodo.17098125 (Wolf, 2025).

Acknowledgements. AM acknowledges 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 and the project FAIR (Future Artificial Intelligence Research), funded by the NextGenerationEU program within the PNRR-PE-AI scheme (M4C2, Investment 1.3, Line on Artificial Intelligence).

Author contribution. Conceptualization: F.W., A.M.; Methodology: F.W.; Software: F.W.; Validation: F.W.; Formal analysis: F.W.; Investigation: F.W.; Writing – original draft: F.W.; Writing – review and editing: F.W., A.M., N.B., and U.F.; Visualization: F.W.; Supervision: A.M., N.B., and U.F. All authors approved the final submitted draft.

Funding statement. FW was supported by a scholarship from the Italian Ministry of Foreign Affairs and International Cooperation.

Competing interests. AM is member of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM). The other authors declare none.

Ethical standard. The research meets all ethical guidelines, including adherence to the legal requirements of the study country.

References

Alla A, **Pacifico A**, **Palladino M and Pesare A** (2023) Online identification and control of pdes via reinforcement learning methods. https://link.springer.com/article/10.1007/s10444-024-10167-y

Altmüller N (2014) *Model predictive control for partial differential equations.* https://dl.acm.org/doi/10.5555/AAI29034536

Arora R, **da Silva BC and Moss E** (2022) Model-based reinforcement learning with sindy. https://arxiv.org/abs/2208.14501

Aubrun S, **Leroy A and Devinant P** (2017) A review of wind turbine-oriented active flow control strategies. *Experiments in Fluids 58*, 134.

Bakarji J, **Kathleen Champion JNK and Brunton SL** (2023) Discovering governing equations from partial measurements with deep delay autoencoders. *Proceedings of the Royal Society A 479*(2276), 20230422.

Bengio Y, **Courville AC and Vincent P** (2012) Unsupervised feature learning and deep learning: A review and new perspectives. *arXiv Preprint* arXiv:1206.5538.

Bhan L, **Bian Y**, **Krstic M and Shi Y** (2024) *Pde control gym: A benchmark for data-driven boundary control of partial differential equations.* https://proceedings.mlr.press/v242/bhan24a/bhan24a.pdf.

Botteghi N and Fasel U (2024) *Parametric pde control with deep reinforcement learning and l 0 sparse polynomial policies*. *2024 IEEE 63rd Conference on Decision and Control (CDC)*. Milan, Italy: IEEE, pp. 6108–6115.

Botteghi N, **Poel M and Brune C** (2025) Unsupervised representation learning in deep reinforcement learning: A review. *IEEE Control Systems 45*(2), 26–68.

Brockman G, **Cheung V**, **Pettersson L**, **Schneider J**, **Schulman J**, **Tang J and Zaremba W** (2016) *OpenAI gym.* https://github.com/openai/gym.

Brunton SL, **Noack BR and Koumoutsakos P** (2020) Machine learning for fluid mechanics. *Annual Review of Fluid Mechanics 52*(1), 477–508.

Brunton SL, **Proctor JL and Kutz JN** (2016) Sparse identification of nonlinear dynamics with control (sindyc). *IFAC-PapersOnLine 49*(18), 710–715.

Champion K, **Lusch B**, **Kutz JN and Brunton SL** (2019) Data-driven discovery of coordinates and governing equations. *116*(45), 22445–22451.

Chua K, **Calandra R**, **McAllister R and Levine S** (2018) Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N and Garnett R (eds), *Advances in Neural Information Processing Systems*. Montréal, Canada: Curran Associates, Inc, Vol. *31*.

Clavera I, **Rothfuss J**, **Schulman J**, **Fujita Y**, **Asfour T and Abbeel P** (2018) *Model-based reinforcement learning via meta-policy optimization*, *Conference on Robot Learning*. Zürich, Switzerland: PMLR, pp. 617–629.

Conti P, **Gobat G**, **Fresca S**, **Manzoni A and Frangi A** (2023) Reduced order modeling of parametrized systems through autoencoders and SINDy approach: Continuation of periodic solutions. *Computer Methods in Applied Mechanics and Engineering 411*, 116072.

Conti P, **Kneifl J**, **Manzoni A**, **Frangi A**, **Fehr J**, **Brunton SL and Kutz JN** (2024) *Veni, vindy, vici: a variational reduced-order modeling framework with uncertainty quantification.*

Deisenroth M and Rasmussen CE (2011) *Pilco: A model-based and data-efficient approach to policy search*. *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pp. 465–472.

Fasel U, **Kaiser E**, **Kutz JN**, **Brunton BW and Brunton SL** (2021) *Sindy with control: A tutorial*. *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 16–21.

Fasel U, **Kutz JN**, **Brunton BW and Brunton SL** (2022) Ensemble-sindy: Robust sparse model discovery in the low-data, high-noise limit, with active learning and control. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 478*(2260), 20210904.

Goodfellow I, **Bengio Y and Courville A** (2016) *Deep Learning*. MIT Press. https://www.deeplearningbook.org/

Grüne L and Pannek J (2017) *Nonlinear model predictive control: Theory and algorithms*. In *Communications and Control Engineering*. Cham: Springer, 2nd Edn.

Hafner D, **Lillicrap TP**, **Norouzi M and Ba J** (2020) Mastering atari with discrete world models. CoRR abs/2010.02193. https://arxiv.org/abs/2010.02193.

Hinton G and Salakhutdinov R (2006) Reducing the dimensionality of data with neural networks. *Science 313*(5786), 504–507.

Hinze M, **Pinnau R**, **Ulbrich M and Ulbrich S** (2008) *Optimization with PDE constraints*. Dordrecht, Netherlands: Springer Science & Business Media, Vol. *23*.

Hirsh SM, **Barajas-Solano DA and Kutz JN** (2022) Sparsifying priors for Bayesian uncertainty quantification in model discovery. *Royal Society Open Science 9*(2), 211823.

Janner M, **Fu J**, **Zhang M and Levine S** (2019) When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems 32*, 12519–12530.

Jiang Z, **Xu D and Liang J** (2017) A deep reinforcement learning framework for the financial portfolio management problem. *arXiv Preprint* arXiv:1706.10059.

Kaiser E, **Kutz JN and Brunton SL** (2018) Sparse identification of nonlinear dynamics for model predictive control in the low-data limit. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 474*(2219), 20180335.

Kaptanoglu AA, **de Silva BM**, **Fasel U**, **Kaheman K**, **Goldschmidt AJ**, **Callaham JL**, **Delahunt CB**, **Nicolaou ZG**, **Champion K**, **Loiseau JC**, **Kutz JN and Brunton SL** (2021) Pysindy: A comprehensive python package for robust sparse system identification, *arXiv preprint* arXiv:2111.08481.

Kim D-K and Jeong H (2021) Deep reinforcement learning for feedback control in a collective flashing ratchet. *Physical Review Research 3*(2), L022002.

Kingma D and Ba J (2015) *Adam: A Method for Stochastic Optimization*. San Diega, CA: International Conference on Learning Representations (ICLR).

Kirkpatrick J, **Pascanu R**, **Rabinowitz N**, **Veness J**, **Desjardins G**, **Rusu AA**, **John Quan KM**, **Ramalho T**, **Grabska-Barwinska A**, **Hassabis D**, **Clopath C**, **Kumaran D and Hadsell R**. (2017) Overcoming catastrophic forgetting in neural networks, *Proceedings of the National Academy of Sciences 114*(13) 3521–3526.

Lesort T, **Díaz-Rodríguez N**, **Goudou J-Franoȷs and Filliat D** (2018) Jean-Franois Goudou, and David Filliat, *state representation learning for control: An overview. Neural Networks 108*, 379–392.

Levine S, **Finn C**, **Darrell T and Abbeel P** (2016) End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research 17*(1), 1334–1373.

Li J, **Monroe W**, **Shi T**, **Jean S**, **Ritter A and Jurafsky D** (2016) Deep reinforcement learning for dialogue generation. *arXiv Preprint*. https://aclanthology.org/D16-1127/.

Liang E, **Liaw R**, **Nishihara R**, **Moritz P**, **Fox R**, **Gonzalez J**, **Goldberg K and Stoica I** (2017) Ray RLLlib: A composable and scalable reinforcement learning library. *arXiv Preprint 85*, 245. arXiv:1712.09381.

Manzoni A, **Quarteroni A and Salsa S** (2022) *Optimal control of partial differential equations: Analysis, approximation, and applications*. Cham: Springer International Publishing.

Gao LM and Kutz JN (2024) Bayesian autoencoders for data-driven discovery of coordinates, governing equations and fundamental constants. *Proceedings of the Royal Society A 480*(2286), 20230506.

McCloskey M and Cohen NJ (1989) Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of Learning and Motivation 24*, 109–165.

Mnih V, **Kavukcuoglu K**, **Silver D**, **Graves A**, **Antonoglou I**, **Wierstra D and Riedmiller M** (2013) Playing atari with deep reinforcement learning. *arXiv Preprint* arXiv:1312.5602.

OpenAI (2020) Learning dexterous in-hand manipulation, *The International Journal of Robotics Research 39*(1), 3–20.

Paszke A, **Gross S**, **Chintala S**, **Chanan G**, **Yang E**, **DeVito Z**, **Lin Z**, **Desmaison A**, **Antiga L and Lerer A** (2017) *Automatic differentiation in pytorch*, NIPS-W. https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Peitz S and Klus S (2020) *Feedback control of nonlinear pdes using data-efficient reduced order models based on the koopman operator*. Cham: Springer International Publishing, pp. 257–282.

Peitz S, **Stenner J**, **Chidananda V**, **Wallscheid O**, **Brunton SL and Taira K** (2024) Distributed control of partial differential equations using convolutional reinforcement learning. *Physica D: Nonlinear Phenomena*, *461*, 134096.

Raffin A, **Hill A**, **Gleave A**, **Kanervisto A**, **Ernestus M and Dormann N** (2021) Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research 22*(268), 1–8.

Rudy SH, **Brunton SL**, **Proctor JL and Kutz JN** (2016) *Data-driven discovery of partial differential equations.* https://openreview.net/forum?id=BJJsrmfCZ.

Schulman J, **Wolski F**, **Dhariwal P**, **Radford A and Klimov O** (2017) *Proximal policy optimization algorithms*, CoRR abs/1707.06347.

Silver D, **Huang A**, **Maddison CJ**, **Guez A**, **Sifre L**, **Driessche G**, **Schrittwieser J**, **Antonoglou I**, **Panneershelvam V**, **Lanctot M**, *et al.* (2016) Mastering the game of go with deep neural networks and tree search. *Nature 529*(7587), 484–489. https://www.science.org/doi/10.1126/sciadv.1602614.

Sutton RS (1990) Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Porter B and Mooney R (eds), *Machine Learning Proceedings 1990*. San Francisco, CA: Morgan Kaufmann, pp. 216–224.

Sutton RS (1991) Dyna, an integrated architecture for learning, planning, and reacting. ACM *SIGART Bulletin 2*(4), 160–163.

Sutton RS and Barto AG (2018) *Reinforcement learning: An introduction*. 2nd Edn. Cambridge, MA, USA: The MIT Press.

OpenAI Team (2024) *Gpt-4 technical report.* https://dl.acm.org/doi/abs/10.1177/0278364919887447.

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Tibshirani R (2018) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological) 58*(1), 267–288.

Vinuesa R (2024) Perspectives on predicting and controlling turbulent flows through deep learning. *Physics of Fluids 36*(3), 031401.

Wang T, **Bao X**, **Clavera I**, **Hoang J**, **Wen Y**, **Langlois ED**, **Zhang S**, **Zhang G**, **Abbeel P and Ba J** (2019) Benchmarking model-based reinforcement learning. *arXiv Preprint* arXiv:1907.02057.

Werner S and Peitz S (2023) Learning a model is paramount for sample efficiency in reinforcement learning control of pdes. ArXiv abs/2302.07160.

Williams GP, **Aldrich A**, **Roderick M**, **Blackmore L**, **Kuwata Y**, **Burnell S**, **Vleugels J**, **Weiss R**, **Hauser J**, **Alonso-Mora J**, *et al.* (2018) Information-theoretic model predictive control: Theory and applications to autonomous driving. *IEEE Transactions on Robotics 34*(6), 1603–1622. https://ieeexplore.ieee.org/document/7989202.

Wolf F (2025) *AE-SINDy-C: Interpretable and efficient data-driven discovery and control of distributed systems.* https://zenodo.org/records/17098126.

Yousif MZ, **Kolesova P**, **Yang Y**, **Zhang M**, **Yu L**, **Rabault J**, **Vinuesa R and Lim H-C** Optimizing flow control with deep reinforcement learning: Plasma actuator placement around a square cylinder. *Physics of Fluids 35*(12), 125101.

Zanna L and Bolton T (2020) Data-driven equation discovery of ocean mesoscale closures. *Geophysical Research Letters 47*(17), e2020GL088376.

Zhang C, **Bengio S**, **Hardt M**, **Recht B and Vinyals O** (2021) Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM 64*(3), 107–115

Zhang X, **Mo W**, **Mowlavi S**, **Benosman M and Başar T** (n.d.) *Controlgym: Large-scale safety-critical control environments for benchmarking reinforcement learning algorithms.* https://proceedings.mlr.press/v242/zhang24b/zhang24b.pdf.

Zhi W, **Fan D**, **Zhou Y**, **Li R and Noack B** (2018) Jet mixing optimization using machine learning control. *Experiments in Fluids 59*, 131.

Zolman N, **Fasel U**, **Kutz JN, and Brunton SL** (n.d.) *SINDy-RL: Interpretable and efficient model-based reinforcement learning.* https://arxiv.org/abs/2403.09110.

Zoph B and Le QV (2016) *Neural architecture search with reinforcement learning*, CoRR abs/1611.01578. https://arxiv.org/abs/1611.01578.

A. Burgers' equation: Additional evaluation

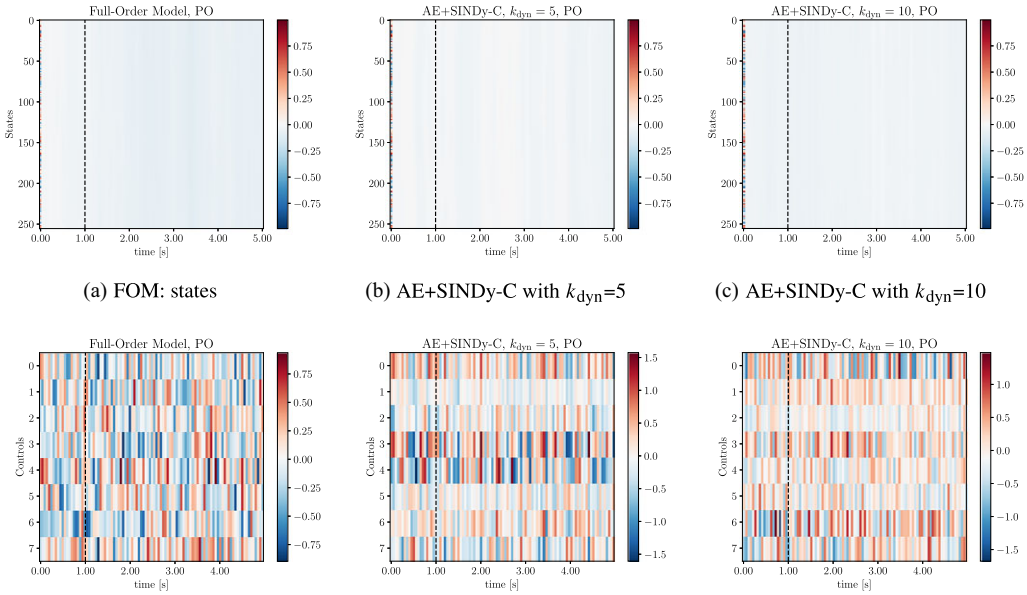
A.1. Random initial condition: State and control trajectories

We analyze the models indicated by the dashed line in Figure 3 with the same fixed random seed for one specific initial condition. As in the training, the initial condition is drawn from a uniform $\mathcal{U}\left([-1,1]^{N_x}\right)$ distribution. The idea of training the agent in this way might seem counterintuitive at first due to the non-regularity of the initial state, but turns out to be a very effective strategy, as we will see in Section 4.1.2. We impose low penalties on the controls (cf. Appendix B), resulting in very aggressive control strategies. Figures A1 and A2 visualize the results for the partially respectively fully observable case. Both of them confirm the results we have already seen from Figure 3. Overall, the controls show a very chaotic behavior and are not regular. This can clearly be explained by the nonregularity of the initial state distribution and the resulting state trajectories, as well as the low penalty on the control itself. The issue of regularity has been discussed in detail in Section 4.1.2, when we consider regular, but out-of-distribution, initial conditions. With a regular initial condition, the problem of chaotic controls does not appear. In a model-by-model comparison, the FO case is overall solved more effectively and with less variation between different random seeds—confirming the intuition that, in general, more measurement points increase the performance of the agent.

Partially observable. Compared to the FO case, the PO case (cf. Figure A1) is as expected more challenging for all of the methods. All three controllers struggle to correctly capture the system dynamics and effectively regulate the system—represented by lower rewards in general and higher standard deviations (cf. Table 1). In the PO case, the baseline model is outperformed by the AE+SINDy-C method, although taking into account the standard deviation, the difference is not significant. Interestingly, while the baseline model seems to rely on all of the controls, both, the $k_{\text{dyn}} = 5$ and $k_{\text{dyn}} = 10$ exhibit one u_i with close to zero controls over the entire time horizon (cf. Figure A1e index one and Figure A2f index two). Overall, the PDE is aggressively controlled and successfully regulated toward the zero-state.

Fully observable. In the FO case (cf. Figure A2), the baseline model outperforms the AE+SINDy-C method. Nevertheless, Table 1 highlights that the differences between the models are marginal although the baseline model specifically stands out by a much lower standard deviation and thus can be trusted more. The performance of AE+SINDy-C for $k_{\text{dyn}} = 5$ and $k_{\text{dyn}} = 10$ are similar, also regarding their standard deviations, even though at the end of the extrapolation horizon in the case of $k_{\text{dyn}} = 5$ the DRL agent seems to slightly overshoot the target while in the case of $k_{\text{dyn}} = 10$ the agent undershoots the target (see Figure A2b and c, respectively).

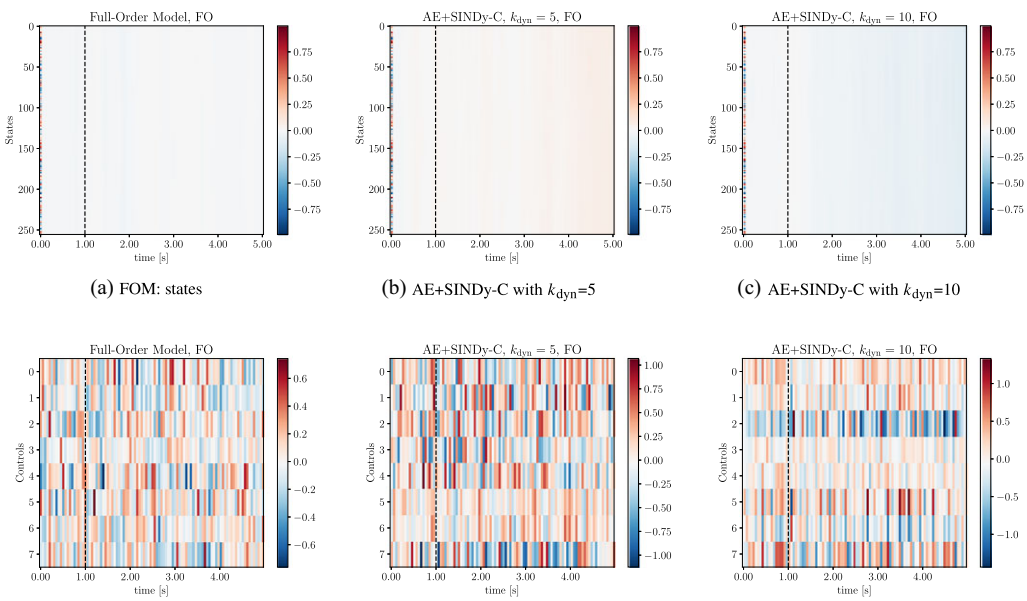
https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press



(a) FOM: states (b) AE+SINDy-C with k_{dyn}=5 (c) AE+SINDy-C with k_{dyn}=10

(d) FOM: controls (e) AE+SINDy-C with k_{dyn}=5 (f) AE+SINDy-C with k_{dyn}=5

Figure A1. *State and control trajectories for Burgers' equation in the partially observable (PO) case with* $\mathcal{U}\left([-1,1]^{N_x}\right)$ *as initial distribution. The black dashed line indicates the timestep t of extrapolation in time. (a) FOM: states. (b) AE + SINDy-C with* $k_{\text{dyn}} = 5$. *(c) AE + SINDy-C with* $k_{\text{dyn}} = 10$. *(d) FOM: controls. (e) AE + SINDy-C with* $k_{\text{dyn}} = 5$. *(f) AE + SINDy-C with* $k_{\text{dyn}} = 5$.



(a) FOM: states (b) AE+SINDy-C with k_{dyn}=5 (c) AE+SINDy-C with k_{dyn}=10

(d) FOM: controls (e) AE+SINDy-C with k_{dyn}=5 (f) AE+SINDy-C with k_{dyn}=10

Figure A2. *State and control trajectories for Burgers' equation in the fully observable (FO) case with* $\mathcal{U}\left([-1,1]^{N_x}\right)$ *as initial distribution. The black dashed line indicates the timestep t of extrapolation in time. a) FOM: states. (b) AE + SINDy-C with* $k_{\text{dyn}} = 5$. *(c) AE + SINDy-C with* $k_{\text{dyn}} = 10$. *(d) FOM: controls. (e) AE + SINDy-C with* $k_{\text{dyn}} = 5$. *(f) AE + SINDy-C with* $k_{\text{dyn}} = 10$.

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

B. Environments: Hyperparameters

Table B1. *Environment details for* ControlGym*'s implemention of Burgers' equation, the diffusivity constant* $\nu = 1.0$ *is fixed*

	Partially observable	Fully observable
Domain size	1.0	1.0
Spatial discretization	256	256
Observable states N_s	48	256
Time: train + extrapolation	1 s + 4 s	1 s + 4 s
Control discretization N_u	8	8
Control support width	0.125	0.125
Additive Gaussian noise (FOM) σ	0.25	0.25
Penalty state Q_w	100.0	100.0
Penalty control R_w	0.01	0.01

Table B2. *Environment details for* PDEControlGym*'s implemention of the Navier–Stokes equations*

	Fully observable
Spatial domain	$[0,1]^2$
Spatial discretization step Δ_x	0.05
Spatial discretization step Δ_y	0.05
Time discretization step Δ_t	0.001
Time: train + extrapolation	0.2 s
Penalty γ controls	0.1

C. Deep RL: Hyperparameters

Table C1. *DRL algorithm configuration details for Burgers' equation experiment*

	Partially observable	Fully observable
Network class	PPO	PPO
Batch size	256	256
Hidden layer size	128	128
Learning rate	$3.0 \cdot 10^{-4}$	$3.0 \cdot 10^{-4}$
GAE λ	0.95	0.95
Discount factor γ	0.99	0.99
Gradient clipping	0.5	0.5

Note: We use Ray RLLib (Liang et al., 2017) to train our models. The PPO (Schulman et al., 2017) policy is trained by using the Adam algorithm (Kingma and Ba, 2015).

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Table C2. *DRL algorithm configuration details for the Navier–Stokes equations experiment*

	Partially observable
Network class	PPO
Batch size	128
Hidden layer size	128
Learning rate	$3.0 \cdot 10^{-4}$
GAE λ	0.95
Discount factor γ	0.99
Gradient clipping	0.5

Note: We use Ray RLLib (Liang et al., 2017) to train our models. The PPO (Schulman et al., 2017) policy is trained by using the Adam algorithm (Kingma and Ba, 2015).

D. Autoencoder: Hyperparameters and training details

Internally, an 80/20 splitting is used for training and validation, and once new data are available, the surrogate model is trained for 100 epochs. In all of the cases, we computed the number of neurons of the hidden layer such that the ratio between the size of the input and the hidden layer is the same as the ratio between the size of the hidden and the output layer. Only for the controls of the Navier–Stokes equations we went for an increase in neurons to find an effective latent space representation (cf. Table D2).

Table D1. *Details of the AE+SINDy-C surrogate model for Burgers' equation*

	Partially observable	Fully observable
Layer shapes (state, control)	$(48,8) \times (10,4) \times (2,2)$	$(256,8) \times (22,4) \times (2,2)$
#Parameters: AE + Ξ-matrix	1178 + 22	11,752 + 22
Off-policy buffer size	200	200
On-policy buffer size	2400	2400
SINDy Polynomial degrees (state, control)	(3, 1)	(3, 1)
SINDy dictionary size d	11	11
FOM data update frequency	5 and 10	5 and 10
Internal epochs	100	100
Adam learning rate	10^{-3}	10^{-3}
Batch size	64	64
Activation function	Softmax	Softmax
Loss function	$\lambda_i = 1.0, i = 1,2$	$\lambda_i = 1.0, i = 1,2$
Clip gradient norm	1.0	1.0

Note: We use *PySINDy* (Kaptanoglu et al., 2021) to generate the set of dictionary functions.

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press

Table D2. *Details of the AE+SINDy-C surrogate model for the Navier–Stokes equations*

	8-dim Latent space	4-dim Latent space
Layer shapes (state, control)	$(882,1) \times (84,4) \times (6,2)$	$(882,1) \times (52,4) \times (3,1)$
#Parameters: AE + Ξ-matrix	150,785 + 510	93,055 + 60
Off-policy buffer size	200	200
On-policy buffer size	2400	2400
SINDy polynomial degrees (state, control)	(3,1)	(3,1)
SINDy dictionary size d	85	20
FOM data update frequency	5	5
Internal epochs	100	100
Adam learning rate	10^{-3}	10^{-3}
Batch size	64	64
Activation function	Softmax	Softmax
Loss function	$\lambda_i = 1.0, i = 1,2$	$\lambda_i = 1.0, i = 1,2$
Clip gradient norm	1.0	1.0

Note: We use *PySINDy* (Kaptanoglu et al., 2021) to generate the set of dictionary functions.

Cite this article: Wolf F, Botteghi N, Fasel U and Manzoni A (2025). Interpretable and efficient data-driven discovery and control of distributed systems. *Data-Centric Engineering*, 6, e51. doi:10.1017/dce.2025.10027

https://doi.org/10.1017/dce.2025.10027 Published online by Cambridge University Press