PROCEEDINGS A

royalsocietypublishing.org/journal/rspa

Research

Cite this article: Liu X-Y, Wang J-X. 2021 Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control. *Proc. R. Soc. A* **477**: 20210618. https://doi.org/10.1098/rspa.2021.0618

Received: 3 August 2021
Accepted: 18 October 2021

Subject Areas:
computational physics, artificial intelligence, applied mathematics

Keywords:
reinforcement learning, physics-informed neural networks, flow control, Kuramoto–Sivashinsky

Author for correspondence:
Jian-Xun Wang
e-mail: jwang33@nd.edu

Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control

Xin-Yang Liu and Jian-Xun Wang

Department of Aerospace and Mechanical Engineering, College of Engineering, University of Notre Dame, Notre Dame, IN, USA

J-XW, 0000-0002-9030-1733

Model-based reinforcement learning (MBRL) is believed to have much higher sample efficiency compared with model-free algorithms by learning a predictive model of the environment. However, the performance of MBRL highly relies on the quality of the learned model, which is usually built in a black-box manner and may have poor predictive accuracy outside of the data distribution. The deficiencies of the learned model may prevent the policy from being fully optimized. Although some uncertainty analysis-based remedies have been proposed to alleviate this issue, model bias still poses a great challenge for MBRL. In this work, we propose to leverage the prior knowledge of underlying physics of the environment, where the governing laws are (partially) known. In particular, we developed a physics-informed MBRL framework, where governing equations and physical constraints are used to inform the model learning and policy search. By incorporating the prior information of the environment, the quality of the learned model can be notably improved, while the required interactions with the environment are significantly reduced, leading to better sample efficiency and learning performance. The effectiveness and merit have been demonstrated over a handful of classic control problems, where the environments are governed by canonical ordinary/partial differential equations.

1. Introduction

Reinforcement learning (RL) is a class of artificial intelligence (AI) techniques that train an AI agent to learn the optimal control strategy by interacting

THE ROYAL SOCIETY PUBLISHING

with the surrounding environment. Over the past few years, with the rapid development of deep learning (DL), deep reinforcement learning (DRL) techniques have been witnessing tremendous success in a variety of applications. In particular, DRL has demonstrated superhuman performance at playing Go [1,2] and Atari games from pixels [3]. Most recently, there has been growing interest in applying DRL for dynamic control of complex physical systems, e.g. laminar/turbulent flows [4–10], active matter [11], fish swimmers [12–14], unmanned aerial vehicles [15–17] and robotics [18–20]. Moreover, people also applied DRL in passive control and shape optimization [21,22].

In general, most state-of-the-art RL agents learn the desired tasks by gathering experience directly from the environment. Namely, the optimal action strategy is derived by interacting with the real physical system in a trial-and-error manner. This class of RL methods is known as *model-free reinforcement learning* (MFRL). Owing to the ease of implementation and no need for prior knowledge of the dynamic transitions, MFRL has been widely applied to many tasks, mainly in playing computer games [3,23,24]. Despite their popularity, MFRL methods usually have *low sample efficiency*, i.e. requiring a massive amount of interactions with the environment. Although the low sample efficiency and slow convergence rate might be acceptable for training an agent in gaming applications since the environment interactions are nearly costless, these shortcomings will significantly limit the DRL applications for dynamic control of physical/mechanical systems (e.g. flights or robotics). First, real-world interactions of a mechanical system can be very expensive and time-consuming. For instance, considering a flow control problem with plasma actuators, it is very costly or even infeasible to train a controller by conducting a huge amount of wind tunnel experiments with enormous control trials, which is unlike training an AI game agent that can be done by playing the computer games for a vast amount of times (episodes). Second, the mechanical systems can be easily worn out from extensive action trials, and thus the exploration of optimal control strategy in MFRL is highly restricted to avoid possible damage to the system in real-world industry settings. Although the off-policy MFRL algorithms with a replay buffer (e.g. deep Q-Networks [3] and their actor–critic (AC) extensions [25–27]) can better use historical data than on-policy MFRL algorithms (e.g. trust region policy optimization [28] and proximal policy optimization [29]), the data efficiency is still far from sufficient.

One way to improve data efficiency is to augment the data collected from real-world interactions with a learned transition model. This is the general idea of the other class of DRL algorithms: *model-based reinforcement learning* (MBRL) [30,31]. Using a learned model to reason about the future can avoid the irreversible consequence of trial-and-error in the real environment and has great potential to significantly improve data efficiency, which is thus more appealing in applications of complex mechanical systems. In addition, the learned transition model is independent of rewards and thus can be transferred to other control problems in the same/similar environments. Many existing MBRL methods rely on simple function approximators, such as Gaussian process (GP), linear models and Gaussian mixture models [32–34]. However, the limited expressibility of the simple models prevents them from handling high-dimensional problems with complex dynamic transitions. Thanks to the rapid developments of deep learning, more and more complex high-dimensional function approximators based on neural networks have been applied to design more powerful MBRL algorithms. For example, Racanière *et al.* [35] proposed a novel MBRL framework, Imagination-Augmented Agent (I2A), where the environment model is constructed by a recurrent network architecture for generating imagined trajectories to inform agent's decisions. Kaiser *et al.* [36] presented a complete MBRL method (SimPLe) using a convolutional neural network to successfully solve Atari games with significantly fewer interactions than MFRL methods. Hafner *et al.* [37] developed the Deep Planning Network (PlaNet) that learns the latent dynamics of the environment directly from images using a variational autoencoder and a recurrent latent network. The effectiveness of PlaNet has been demonstrated by successfully solving a number of continuous control tasks from pixels. Hafner *et al.* [38] further extended the PlaNet by developing a novel AC based MBRL method (Dreamer), which learns long-horizon behaviours from images purely by latent imagination. Dreamer has

been evaluated on the DeepMind Control Suite and outperforms most state-of-the-art MBRL and MFRL algorithms in every aspect.

Despite the great promise, most commonly used MBRL approaches suffer from model inaccuracy (i.e. model bias), preventing them from matching the success of their model-free counterparts [30]. This is particularly true when it comes to learning complex dynamics with high-capacity models (e.g. deep neural networks), which are prone to overfitting in data-sparse and out-of-sample regimes [31]. In particular, the model bias can be significantly exacerbated for predicting long rollout horizons because of the 'compound error' effect. To mitigate this issue, rather than learning the transition deterministically, people built the dynamic models in a probabilistic manner, where the unknown model bias is treated as the epistemic uncertainty (i.e. model-form uncertainty) and can be modelled in several different ways. For example, Depeweg *et al.* [39] employed Bayesian neural networks (BNNs) to learn the probabilistic dynamic transition and update the policy over an ensemble of models sampled from the trained BNNs. Kurutach *et al.* [40] proposed to use an ensemble of models to estimate the model-form uncertainty and regularize the trust region policy optimization (TRPO). Nonetheless, the model-form uncertainty is notoriously difficult to quantify, especially for black-box deep learning models [41,42]. Most recently, a more promising strategy known as physics-informed deep learning (PIDL) has attracted increasing attention in the scientific machine learning (SciML) community, aiming to leverage both the advantages of deep learning and prior knowledge of underlying physics to enable data-scarce learning. Instead of learning solely from labelled data, the model training process is also guided by physics laws and knowledge, which could provide rigorous constraints to the model output, alleviate overfitting issues, and improve the robustness of the trained model in data-scarce and out-of-sample regimes. This idea has been recently explored for solving PDEs or modelling complex physical systems. For example, researchers have incorporated physical constraints (e.g. realizability, symmetry, invariance) into SciML models to develop physics-informed, data-driven turbulence models [43–45]. People have also used governing equations of the physical systems to inform or directly train deep neural networks, i.e. physics-informed neural networks (PINNs) [46], where the violation of the physical laws is penalized by incorporating the equation residuals into the network loss function. This simple idea has been applied in many scientific and engineering problems [47–51].

In this work, we leverage the idea of PIDL and propose physics-informed model-based reinforcement learning (PiMBRL), an innovative MBRL framework for complex dynamic control that incorporates the physical laws/constraints of the system to alleviate the issue of model bias, reduce the real-world interactions and significantly improve the data efficiency. Specifically, a novel autoencoding-based recurrent network architecture is constructed to learn the dynamic transition in the Dyna-style MBRL framework [52], which is a commonly used MBRL formulation. The governing physics of the environment are assumed to be known and are used to inform the model learning and RL agent optimization. State-of-the-art off-policy AC algorithms, e.g. Twin Delayed Deep Deterministic Policy Gradients (TD3) [26], are used for value/policy optimization. We have demonstrated the effectiveness and merit of the proposed PiMBRL on a few classic dynamic control problems, where the environments are governed by canonical ordinary/partial differential equations (ODEs/PDEs), including cart-pole, pendulum, viscous fluid dynamics governed by Burgers' equation and chaotic/turbulent dynamics governed by Kuramoto–Sivashinsky (KS) Equation. The performance of the proposed PiMBRL algorithms is compared with their MBRL and MFRL counterparts, and significant improvements in terms of sample efficiency and model accuracy are observed. The novel contributions of this work are summarized as follows: (i) we propose a physics-informed model-based RL framework based on a novel encoder–decoder recurrent network architecture; (ii) embed the physics of the environment into the MBRL using discretized PIDL formulation [53]; (iii) demonstrate the effectiveness of proposed methods on a variety of dynamic control problems, particularly including nonlinear spatio-temporal chaotic systems, e.g. the KS equation, which exhibits a wide range of dynamics from the steady to chaotic/turbulent regimes, shedding lights on developing controllers for more challenging fluid systems governed by Navier–Stokes equations;

(iv) compare the proposed method with state-of-the-art MBRL and MFRL in terms of accuracy and sample complexity. This work is the first attempt to use physical laws to inform the MBRL agent optimization to the best of the authors' knowledge.

The rest of the paper is organized as follows. The background of MFRL/MBRL and our proposed PiMBRL algorithms are introduced in §2. Numerical results on classic dynamic control problems are presented and discussed in §3. The influence of model rollout length and accuracy threshold is further discussed in §4. Finally, §5 concludes the paper.

2. Methodology

(a) Problem formulation and background

We consider dynamical systems equipped with localized control inputs (i.e. actuators)

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \mathscr{F}(\boldsymbol{u}, \boldsymbol{a}; \boldsymbol{\mu}), \tag{2.1}$$

where $\boldsymbol{u}(\boldsymbol{x}, t) \in \mathbb{R}^{d_u}$ denotes the state variable of the system in the spatial domain Ω and temporal domain $t \in [0, T]$, $\boldsymbol{a}(\boldsymbol{x}, t) \in \mathbb{R}^{d_a}$ represents the action variable (i.e. control inputs), and $\mathscr{F}(\cdot)$ is a nonlinear differential operator parametrized by $\boldsymbol{\mu}$. In many cases, the systems can be assumed to possess the Markov property, referred to as Markov Decision Processes (MDP). The discrete form can be written as

$$\boldsymbol{u}_{t+1} = \mathcal{F}(\boldsymbol{u}_t, \boldsymbol{a}_t; \boldsymbol{\mu}), \tag{2.2}$$

where the state $\boldsymbol{u}_{t+1}$ at next time $t+1$ only depends on the state $\boldsymbol{u}_t$ and action $\boldsymbol{a}_t$ at current time step t, and the time-invariant transition dynamics $\mathcal{F}$ of the environment is a nonlinear differential functional. In the optimal control problem, the goal is to find a series of action signals (a.k.a., policy π) that maximizes the expected return $R(\pi)$,

$$R(\pi) = \int_0^T \mathop{\mathbb{E}}_{\boldsymbol{u}_t \sim \pi_t} [r(\boldsymbol{u}_t)], \tag{2.3}$$

where $r(\boldsymbol{u}_t)$ denotes the reward function of the state at time t, which is a signal to assess the control agent locally. This optimal control problem can be solved by deep reinforcement learning (DRL), either in a model-free or model-based manner.

(i) Value function, policy function and Bellman equation

Before putting forth the proposed DRL algorithms, we introduce several important concepts in DRL, including value and policy functions and Bellman equation. Value functions are functions of a state (or a state-action pair) that estimate the total return starting from that particular state (state-action pair). Value function $v(\boldsymbol{u})$ of a state $\boldsymbol{u}$ is known as state-value function, while value function $q(\boldsymbol{u}, \boldsymbol{a})$ of a state-action pair $(\boldsymbol{u}, \boldsymbol{a})$ is known as action-value function. The state-value and action-value functions are formally defined as

$$v(\boldsymbol{u}) \doteq \mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^k r(\boldsymbol{u}_{t+k}) \middle| \boldsymbol{u}_t = \boldsymbol{u}\right] \tag{2.4a}$$

and

$$q(\boldsymbol{u}, \boldsymbol{a}) \doteq \mathbb{E}\left[\sum_{k=1}^{\infty} \gamma^k r(\boldsymbol{u}_{t+k}) \middle| \boldsymbol{u}_t = \boldsymbol{u}, \boldsymbol{a}_t = \boldsymbol{a}\right], \tag{2.4b}$$

where $\gamma \leq 1$ is the discount rate. A policy function π maps states to actions (or probabilities of actions). Namely, a policy function π can be defined either as a deterministic function $\pi(\boldsymbol{u}) = \boldsymbol{a}$ or a probability measure $\pi(\boldsymbol{a}|\boldsymbol{u})$. Owing to the nature of MDP, value functions can be estimated

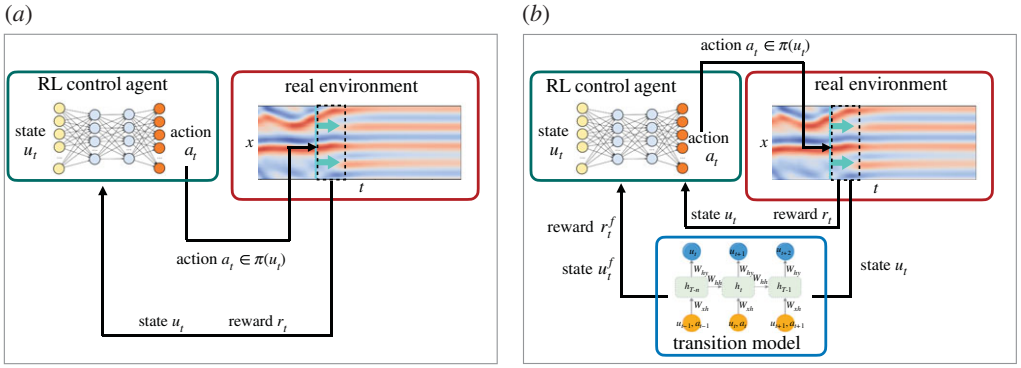

Figure 1. (*a*,*b*) Schematics of model-free reinforcement learning (MFRL) and model-based reinforcement learning (MBRL). (Online version in colour.)

recursively based on the Bellman equations [52]

$$v(\boldsymbol{u}_t) = \mathop{\mathbb{E}_{\pi}}_{\boldsymbol{u}_{t+1}\sim\mathcal{P}} [r(\boldsymbol{u}_t, \boldsymbol{a}_t) + \gamma v(\boldsymbol{u}_{t+1})] \tag{2.5a}$$

and

$$q(\boldsymbol{u}_t, \boldsymbol{a}_t) = \mathop{\mathbb{E}}_{\boldsymbol{u}_{t+1}\sim\mathcal{P}} \left[r(\boldsymbol{u}_t, \boldsymbol{a}_t) + \gamma \mathop{\mathbb{E}}_{\boldsymbol{a}_{t+1}\sim\pi} q(\boldsymbol{u}_{t+1}, \boldsymbol{a}_{t+1}) \right], \tag{2.5b}$$

where $\mathcal{P} = \mathcal{P}(\boldsymbol{u}_{t+1}|\boldsymbol{u}_t, \boldsymbol{a}_t)$ is the transition probability, describing the dynamics of the environment.

(ii) Model-free and model-based reinforcement learning

As mentioned earlier, RL aims to find a series of actions (i.e. optimal policy) that maximize the total return by estimating the value and/or policy function based on the Bellman equation. Depending on whether or not learning and using a model of the transition dynamics of the environment, RL can be classified into two categories: model-free reinforcement learning (MFRL) and model-based reinforcement learning (MBRL). In MFRL, the optimization process is conducted by repeatedly interacting with the environment with a trial-and-error search, and the model of the environment is not required (figure 1*a*). Namely, the state dynamics of the environment are (partially) observed as exploring different policy strategies, and the best policy will be identified after massive trials. MBRL, on the other hand, leverages a model $\tilde{\mathcal{F}}$ that approximates the real environment $\mathcal{F}$ and predicts the dynamic transition ($\tilde{\mathcal{F}}: \boldsymbol{u}_t, \boldsymbol{a}_t \rightarrow \boldsymbol{u}_{t+1}$), which can be learned from the interactions with the real environment. The RL agent is then optimized by the interactions not only with the real environment but also with the virtual environment constructed by the model (figure 1*b*). The learned model $\tilde{\mathcal{F}}$ can be used for planning with its gradient information (e.g. stochastic value gradients [54], guided policy search [55]) or synthesizing imagined samples to augment real samples for better sample efficiency. The latter is known as the Dyna-like MBRL [36,40,52,56] that can directly leverage cutting-edge MBRL algorithms.

The algorithms of the DRL agent optimization can be grouped into three classes: (i) actor-only, (ii) critic-only and (iii) actor-critic methods. Actor-only (i.e. policy-gradient) methods directly optimize the policy function $\tilde{\pi}(\boldsymbol{u}; \boldsymbol{\theta}_\pi)$, which is often parametrized by a deep neural network by calculating the policy gradient with respect to network parameters $\boldsymbol{\theta}_\pi$. The optimization can be solved based on stochastic gradient descent (SGD) [57] or its variants [58–61]. Examples of policy-gradient methods include REINFORCE [62], TRPO [28] and PPO [29]. Critic-only (value-based) methods are a family of RL algorithms that learn an DNN-approximated value-action function $\tilde{q}(\boldsymbol{u}, \boldsymbol{a}; \boldsymbol{\theta}_q)$ based on the optimal Bellman equation. Examples include Q-Learning [63], DQN [3], Dueling DQN [64], etc. Since the optimization in critic-only methods is always performed

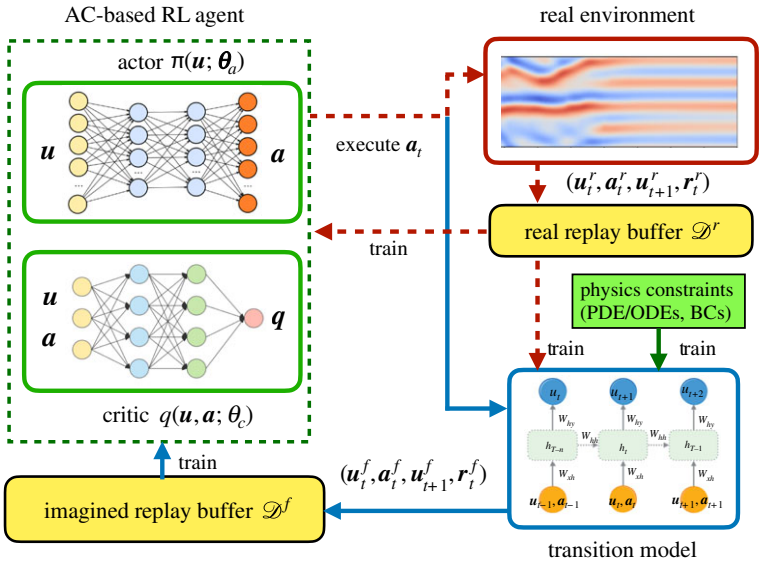

Figure 2. Schematics of Dyna-style physics-informed model-based actor–critic algorithm. (Online version in colour.)

off-policy, they are more sample-efficient than the actor-only methods that are often on-policy. However, since critic-only methods optimize the policy indirectly, they are less stable compared with the actor-only methods. The AC methods learn a value function to support the policy gradient optimization, and thus the AC family combines the strengths of both actor and critic methods. As such, AC-based methods will be used in the proposed PiMBRL for value/policy optimization.

(b) Physics-informed model-based reinforcement learning

We propose a physics-informed model-based reinforcement learning (PiMBRL) framework, where the physics knowledge (e.g. conservation laws, governing equations and boundary conditions) of the environment is incorporated to inform the model learning and RL optimization. In this work, we focus on the Dyna-style MBRL formulation with the off-policy AC-based optimization. The proposed framework will retain the generality and optimality of model-free AC-based DRL methods, while significantly reducing the real-world interactions by learning a reliable environment model based on physics-informed discrete learning. Specifically, an AC-based RL agent will be initialized and starts interacting with the real environment. State-action data pairs and corresponding rewards $(\boldsymbol{u}_t^r, \boldsymbol{a}_t^r, \boldsymbol{u}_{t+1}^r, r_t^r)$ are iteratively collected from the real environment and saved into the real replay buffer $\mathscr{D}^r$. These real samples are used to train the actor-critic agent and the transition model simultaneously. The model is constructed by an auto-encoding recurrent network, where boundary conditions (BCs) of the system are strictly encoded, and the governing physics are imposed softly by minimizing the violation of the conservation laws. Synthetic samples $(\boldsymbol{u}_t^f, \boldsymbol{a}_t^f, \boldsymbol{u}_{t+1}^f, r_t^f)$ are generated by the model and are collected into the imagined replay buffer $\mathscr{D}^f$, which will be leveraged to augment the real samples for the RL update. The model training, data generation, environment interaction and RL agent optimization are conducted iteratively in an online manner. The overall schematic of the proposed PiMBRL is shown in figure 2, and the detailed algorithm is given by algorithm 1. More details of AC optimization and physics-informed model construction will be elaborated as follows.

[1]In this paper, TD3 is used as a demonstration (details of the TD3 is given in algorithm 3), but other off-policy algorithms such as Deep Deterministic Policy Gradient (DDPG) and Soft Actor–Critic (SAC) are also applicable.

Algorithm 1. Physics-informed model-based reinforcement learning (PiMBRL).

Randomly initialize policy (actor) network $\pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$, value (critic) network(s) $q(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_q)$, transition model $\tilde{\mathcal{F}}(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_F)$, and replay buffers $\mathscr{D}^r$, $\mathscr{D}^f$ for real and fictitious environments.
Randomly initialize state $\boldsymbol{u}_0$ (or observed state $\boldsymbol{u}_0^o$), reward r_0, and done signal d_0 for real environment.
for episode$=1, M$ **do**
  for $i=0, T$ **do**
    Execute action $\boldsymbol{a}_i=\pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$ in the real environment $\mathcal{F}$;
    Save new data pair $(\boldsymbol{u}_i^o, \boldsymbol{a}_i, \boldsymbol{u}_{i+1}^o, r_i, d_i)$ to the real buffer $\mathscr{D}^r$;
    if episode ends **then** Reset the environment $\mathcal{F}$;
    end if
  end for
  if sufficient (n_{s_M}) state-action pairs stored in the real buffer $\mathscr{D}^r$ **then**
    Sample a batch of real state-action pairs, $\{(\boldsymbol{u}_i^o, \boldsymbol{a}_i, \boldsymbol{u}_{i+1}^o)\}_{i=1}^{n_{b_r}}$, from $\mathscr{D}^r$
    Update the transition model $\tilde{\mathcal{F}}(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_F)$ using the data loss L_D on the batch $\mathscr{D}^r$;
  end if
  if transition model meets the accuracy threshold (data loss $L_D<\lambda$) **then**
    for model prediction length $=1, l_M$ **do**
      Sample a batch of states $\{(\boldsymbol{u}_{j+1})\}$ from $\{\mathscr{D}^r, \mathscr{D}^f\}$
      Execute actions $\{\boldsymbol{a}_{j+1}=\pi(\boldsymbol{u}_{j+1};\boldsymbol{\theta}_\pi)\}_{j=1}^{n_{b_f}}$ in transition model $\tilde{\mathcal{F}}$;
      Save new data pairs $\{(\boldsymbol{u}_{j+1}^o, \boldsymbol{a}_{j+1}, \boldsymbol{u}_{j+2}^o, r_{j+2}, d_{j+2})\}_{j=1}^{n_{b_f}}$ to buffer $\mathscr{D}^f$;
    end for
  end if
  if enough (n_{s_R}) state-action pairs stored in $\{\mathscr{D}^r, \mathscr{D}^f\}$ **then**
    Sample a batch of state-action pairs $\{(\boldsymbol{u}_k^o, \boldsymbol{a}_k, \boldsymbol{u}_{k+1}^o)\}$ from the fake buffer $\mathscr{D}^f$
    Update model $\tilde{\mathcal{F}}$ according to physical loss L_E on sampled state-action pairs;
  end if
  Sample a batch of $\{(\boldsymbol{u}_l^o, \boldsymbol{a}_l, \boldsymbol{u}_{l+1}^o, r_l, d_l)\}$ from the augmented buffer $\{\mathscr{D}^r, \mathscr{D}^f\}$
  Update policy network $\pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$ and value network $q(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_q)$ on the sampled state-action pairs using off-policy algorithms[1] (see algorithm 3 in appendix A).
end for

(i) Dyna-style model-based actor–critic optimization

We consider a generic Dyna-style model-based actor–critic optimization algorithm, which can be easily adapted to any state-of-the-art off-policy AC methods, e.g. DDPG, TD3 or SAC. As mentioned earlier, in the Dyna-style formulation, model is used to augment real samples, and thus model-free AC-based optimization can be directly leveraged. For a AC-based RL agent, two neural networks $\tilde{\pi}(\boldsymbol{u};\boldsymbol{\theta}_\pi)$ and $\tilde{q}(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_q)$ are constructed to represent the policy and value functions, respectively. Based on samples from both the real environment and virtual environment simulated by the model, the policy network $\tilde{\pi}(\boldsymbol{u};\boldsymbol{\theta}_\pi)$ is iteratively updated by

$$\boldsymbol{\theta}_\pi^{k+1}=\boldsymbol{\theta}_\pi^k+\alpha_\pi\nabla_{\boldsymbol{\theta}_\pi}J(\boldsymbol{\theta}_\pi^k), \tag{2.6}$$

where α_π is the learning rate and $\nabla_{\boldsymbol{\theta}_\pi}J(\boldsymbol{\theta}_\pi)$ represents the policy gradient with respect to actor network parameters $\boldsymbol{\theta}_\pi$, which can be calculated based on the critic network

$$\nabla_{\boldsymbol{\theta}_\pi}J(\boldsymbol{\theta}_\pi^k)=\mathbb{E}\left[\sum_{t=0}^{T}\nabla_{\boldsymbol{\theta}_\pi}\log\tilde{\pi}(\boldsymbol{u}_t;\boldsymbol{\theta}_\pi^k)\cdot\tilde{q}(\boldsymbol{u}_t,\boldsymbol{a}_t;\boldsymbol{\theta}_q^k)\right]. \tag{2.7}$$

The critic network $\tilde{q}(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_q)$ is optimized by minimizing the temporal difference (TD)-based loss function

$$\boldsymbol{\theta}_q^*=\arg\min_{\boldsymbol{\theta}_q}||q_t'-\tilde{q}(\boldsymbol{u}_t,\boldsymbol{a}_t;\boldsymbol{\theta}_q)||_{L_2}, \tag{2.8}$$

where $||\cdot||_{L_2}$ represent L2 norm and q_t' is estimated based on the optimal Bellman equation

$$q_t'=r_t+\gamma\tilde{q}(\boldsymbol{u}_{t+1},\tilde{\pi}(\boldsymbol{u}_{t+1};\boldsymbol{\theta}_\pi);\boldsymbol{\theta}_q). \tag{2.9}$$

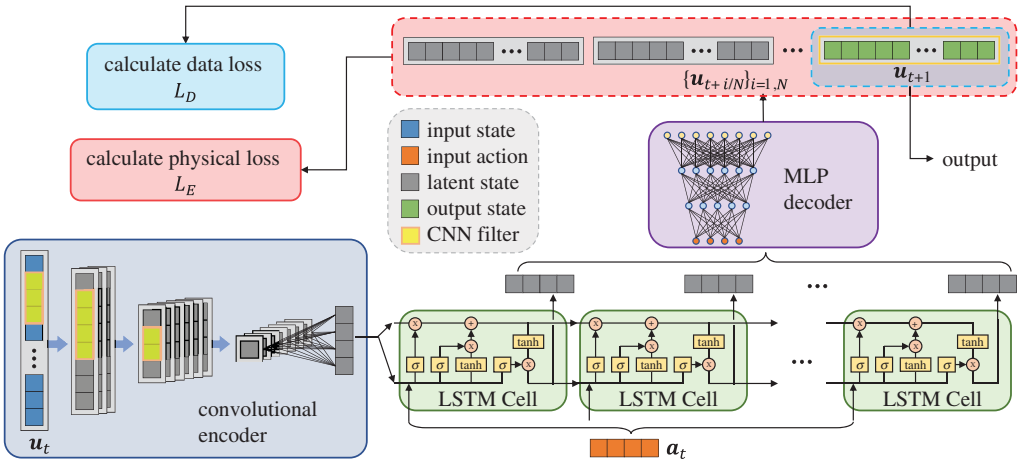

Figure 3. Schematics of LSTM-based neural network architecture for transition model. (Online version in colour.)

(ii) Physics-informed learning architecture for transition dynamics

We develop a physics-informed recurrent neural network to learn the dynamics transition, aiming to map the current states and actions to the states at the next control step ($\tilde{\mathcal{F}}: \boldsymbol{u}_t, \boldsymbol{a}_t \rightarrow \boldsymbol{u}_{t+1}$). To better capture the spatiotemporal dependencies, a convolutional encoder, multi-layer perceptron (MLP) decoder and long-short term memory (LSTM) blocks are used to build the learning architecture. As shown in figure 3, the high-dimensional state vector ($\boldsymbol{u}_t$) at the current control step is encoded into the latent space by a convolutional encoder. Together with the input actions ($\boldsymbol{a}_t$), the latent state vector is fed into the LSTM-based transition network, which outputs latent intermediate transition states between the two control steps. After a multi-layer perceptron (MLP) decoder, the latent outputs are decoded to the full-order physical states. The network is trained based on both data and physics constraints. The data loss L_D is defined by the mismatch between the model prediction $\tilde{\boldsymbol{u}}_{t+1}$ and labelled data $\boldsymbol{u}_{t+1}$ obtained from interactions with the real environment

$$L_D = \frac{1}{n_b} \sum_{j=1}^{n_b} ||\tilde{\boldsymbol{u}}_{t+1}^{(j)} - \boldsymbol{u}_{t+1}^{(j)}||_{L_2}, \tag{2.10}$$

where n_b is the batch size. The physical loss L_E is constructed based on the conservation laws of the system in their discretized form. The residuals of the governing equations is minimized on multiple discrete spatio-temporal snapshots. To this end, the network outputs include the states at N intermediate time steps ($\{\boldsymbol{u}_{t+i/N}\}_{i=1,N}$) as well as the state at the next control step. The physical loss is then obtained by taking the averaged residuals of the governing equations

$$L_E = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{n_b} \left\| \frac{\mathrm{d}\tilde{\boldsymbol{u}}_{t+i/N}^{(j)}}{\mathrm{d}t} - \mathcal{F}(\tilde{\boldsymbol{u}}_{t+i/N}^{(j)}, \boldsymbol{a}_{t+i/N})^{n_b} \right\|_{L_2}, \tag{2.11}$$

where spatial derivatives in $\mathcal{F}$ are computed using high-order finite-difference-based spatial filtering, and temporal derivatives $\mathrm{d}\boldsymbol{u}_{t+i/N}/\mathrm{d}t$ are approximated by forward Euler method.

3. Results

In this section, we test the proposed PiMBRL on a number of classic control problems that are governed by ODEs or PDEs. The performance is compared against the baseline model-free and model-based counterparts. The standard model-free TD3 algorithm is used as the MFRL baseline (see algorithm 2 in appendix A), while the purely data-driven Dyna-like model-based TD3 is

deemed as the MBRL baseline. The hyperparameters used in the following experiments are summarized by tables 1 and 2 in appendix A. Note that since this work aims to demonstrate the merit of incorporating physics prior in MBRL, the comparison study is confined within the MFBL, MBRL, and PiMBRL variants of the same policy optimization algorithm. The horizontal comparisons among different MFRL/MBRL baselines have been studied in, e.g. refs of [65,66], which is not the focus here.

(a) ODE governed environments

We first evaluate PiMBRL on two classic dynamic control benchmarks, Cart-Pole and Pendulum, which are available in the OpenAI Gym environment. The physics of both systems are known, which can be described by a set of ODEs. Since there is no spatial dependence and the dimension of the system is low, we use a low-capacity two-layer MLP with 256 neurons per layer to directly learning the transition dynamics $\boldsymbol{u}_{t+1} = \tilde{\mathcal{F}}(\boldsymbol{u}_t, \boldsymbol{a}_t)$ and model rollout length (l_M) is set equal to the trajectory length.

(i) Cart-Pole

We start with the Cart-Pole benchmark problem (i.e. 'CartPole-v0' environment provided in OpenAi gym), where a cart moves along a frictionless track with a pole attached to the top of it via an unactuated joint, as shown in figure 4*a*. The control goal is to keep the pole from falling over by acting a horizontal force on the cart. The reward is +1 for each time step as long as the pole is upright and the cart remains in a certain region. The physics of this system is governed by

$$\left.\begin{aligned} \ddot{x} &= \frac{f + m_p\dot{\theta}^2 \sin\theta - m_p l\ddot{\theta}}{m_p + m_c} \\ \ddot{\theta} &= \frac{g\sin\theta - \cos\theta(f + m_p\dot{\theta}^2\sin\theta)/(m_p + m_c)}{l(4/3 - (m_p\cos^2\theta/(m_c + m_p)))}, \end{aligned}\right\} \tag{3.1}$$

where x is the spatial coordinate of the cart, θ represents the angle of the pole from vertical, and f is the force that the RL agent applies on the cart. m_c, m_p are the mass of the cart and pole, respectively. One episode is considered to be ended if the pole deviates too much from vertical position (i.e. $|\theta| > \pi/12$) or the cart leaves the designated area (i.e. $|x| > 2.4$) or one episode has more than 200 control steps. The state observation of this environment is a four-dimensional vector, $\boldsymbol{u} = (x, \dot{x}, \theta, \dot{\theta})$, while the action space is discrete, consisting of two valid values $\{-10, 10\}$. Each episode begins at a random state $\boldsymbol{u}_0 = (x_0, \dot{x}_0, \theta_0, \dot{\theta}_0)$.

Figure 4*b* compares the performance curves of the MFRL, MBRL and PiMBRL. The proposed PiMBRL reaches the total return of 200 only after about 3000 time steps in the real environment, while the vanilla Dyna-like MBRL counterpart needs much longer to achieve so and its performance is not stable as well. For the MFRL counterpart, the total return is still below 120 even after 15 000 time steps. Although both MBRL and MFRL are able to achieve the same performance with sufficient time steps, PiMBRL only uses about 45.2% and 9.7% time steps needed by its MBRL and MFRL counterparts, respectively. Therefore, to achieve the same level of performance, PiMBRL can significantly reduce the required number of interactions with the real environment, compared with the original model-free TD3 (i.e. MFRL) and Dyna-like model-based TD3 (i.e. MBRL).

(ii) Pendulum

The second test case is the Pendulum-v0 available in the OpenAi gym. In this environment (figure 5*a*), one end of the pendulum is fixed, while the other end can swing freely. θ denotes the angle of the pendulum from vertical position. The state contains the angle and its time derivative, i.e. $\boldsymbol{u} = (\theta, \dot{\theta})$. In each episode, the pendulum starts from a random state $\boldsymbol{u}_0 \in (-1, 1) \times (-1, 1)$. Besides, the angular velocity is constrained as $\dot{\theta} \in [-8, 8]$, and any $\dot{\theta}$ out of this range will be

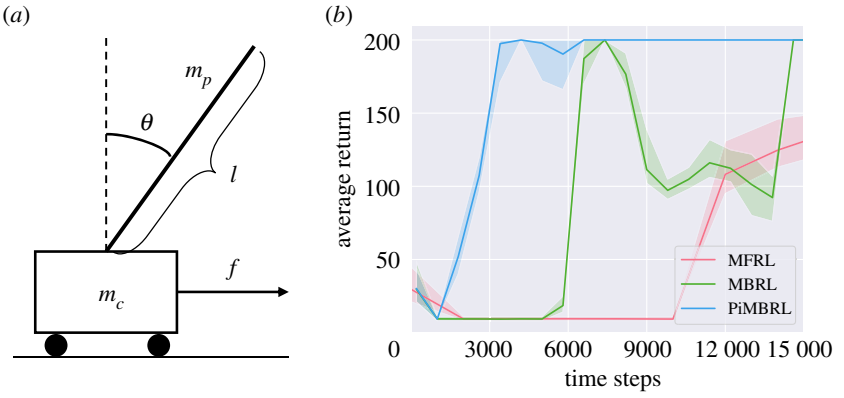

Figure 4. (*a*) Schematic diagram of the Cart-Pole environment. (*b*) Performance curve of PiMBRL versus standard model-free TD3 (MFRL) and Dyna-like model-based TD3 (MBRL) in Cart-Pole environment. The solid lines indicate averaged returns of 100 randomly selected test episodes, while the shaded area represents the return distribution of all test samples. (*a*) Cart-Pole, (*b*) RL Performance. (Online version in colour.)

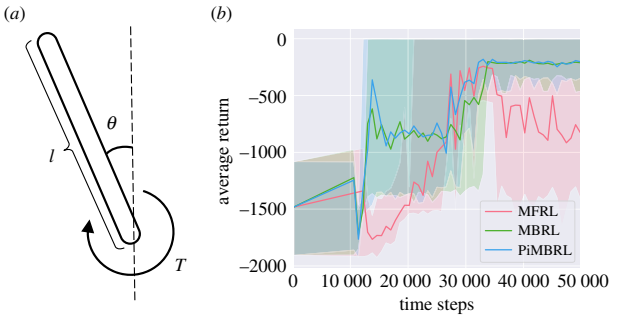

Figure 5. (*a*) Schematic diagram of pendulum environment. (*b*) Performance curve of PiMBRL versus standard model-free TD3 (MFRL) and Dyna-like model-based TD3 (MBRL) in the pendulum environment. The solid lines indicate averaged returns of 100 randomly selected test episodes, while the shaded area represents the return distribution of all test samples. (*a*) Pendulum, (*b*) RL performance. (Online version in colour.)

capped by the boundary value (−8 or 8). The dynamics of the pendulum system is governed by,

$$\ddot{\theta} = -\frac{3g}{2l}\sin(\theta + \pi) + \frac{3}{ml^2}T, \tag{3.2}$$

where $g = 10$ is the acceleration of gravity, $T \in [-2, 2]$ denotes the torque that the agent applies to the pendulum, $l = 1$ and $m = 1$ are the length and mass of the pendulum, respectively. The control goal here is to swing the pendulum up and make it stays upright, meanwhile consuming as less energy as possible. To achieve this, the reward function is defined as

$$r = -\theta^2 - 0.1\dot{\theta}^2 - 0.001T^2. \tag{3.3}$$

The performance curve is shown in figure 5*b*. After about 3000 time steps, both the PiMBRL and MBRL achieve averaged total return of –200 with reduced uncertainty. By contrast, the total return of the MFRL largely fluctuates and the average value remains less than –800. Although PiMBRL shows greater sample efficiency over the MFRL baseline, it does not show a notable advantage over the MBRL counterpart in the Cart-Pole case. This might be due to the following two factors. First, compared with the Cart-Pole problem, the dynamics of the pendulum environment is easier for the model to learn, since the observation space is much smaller than that of the Cart-Pole environment (two dimensions versus four dimensions). Second, the model

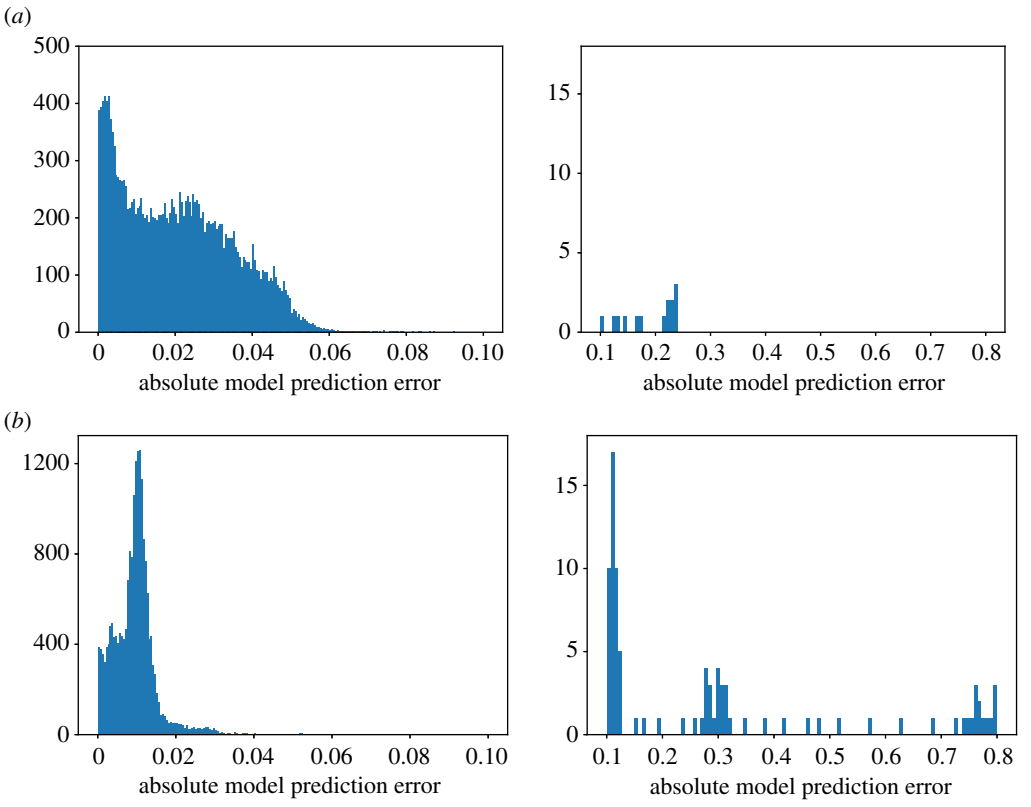

Figure 6. Histograms of the prediction errors of the transition models for the pendulum environment, (*a*) trained with physics-informed loss in PiMBRL and (*b*) trained with the data loss only in MBRL. (Online version in colour.)

network is benefited less from the physical loss, since when the state $\dot{\theta}$ exceeds the limits $[-8, 8]$ and capped by the boundary value, the governing equation (3.2) is no longer satisfied.

Figure 6 compares the histograms of the prediction errors of the models trained by PiMBRL and MBRL, respectively. Figure 6*a* shows the prediction error of the model trained with the physics-informed loss (equation loss + data loss), while figure 6*b* shows the model prediction error in MBRL where only the labelled data are used for training. Although the model does achieve higher accuracy on average without using physics constraints (see two sub-figures in the left column), in the out-of-sample regime (away from the training set), the physics-informed model shows better performance and robustness (see two sub-figures in the right column). Overall, the models in both PiMBRL and MBRL are learned sufficiently well to achieve a roughly similar RL performance.

(b) PDE governed environments

Unlike the environments that can be described by ODEs, the systems governed by PDEs are much more complicated in terms of the dimension of spatiotemporal solution space, dramatically increasing the level of difficulty in learning the dynamic model as well as the policy and value functions. In this section, we evaluate the proposed PiMBRL structure on two continuous control problems in the environments governed by Burgers' equation and KS equation, respectively. The length of each control step (i.e. refresh rate) is set at the order of 10^{-1} second based on common flow sensor response time [67]. The learned transition model is only used to predict a limited rollout length (l_M) in each trajectory to control the model error accumulation, making it remain at a relatively low level.

(i) Burgers' equation

For the first PDE-based control problem, we consider a one-dimensional Burgers' equation with periodic boundary condition,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} + f(x,t), \quad x \in [0,l],\ t \in [0,2\pi], \tag{3.4}$$

where x is the spatial coordinate, $\nu = 0.01$ is the kinematic viscosity, and $f(x,t)$ denotes the source term, defined as

$$f(x,t) = a_1(t)\exp\left[\left(-15\left(\frac{x}{l} - 0.25\right)\right)^2\right] + a_2(t)\exp\left[\left(-15\left(\frac{x}{l} - 0.75\right)\right)^2\right] \tag{3.5}$$

with the control parameters $\boldsymbol{a} = (a_1, a_2) \in [-0.025, 0.075]^2$.

The control problem is defined as training the RL agent to match a reference trajectory. Namely, the RL agent is trained to find the optimal strategy of controlling the source term with two control parameters a_1, a_2, in order to match a predefined reference trajectory profile $\boldsymbol{u}_{\mathrm{re}}$,

$$u_{re}(x,t) = 0.05\sin t + 0.5, \quad t \in [0,2\pi]. \tag{3.6}$$

Each episode starts from a randomly generated initial condition,

$$u(x,0) = 0.2\,c\exp\left[\left(-5\left(\frac{x}{l} - 0.5\right)\right)^2\right] + 0.2\,(1-c)\left(0.5\sin 4\pi\frac{x}{l} + 0.5\right), \tag{3.7}$$

where c is randomly sampled from a uniform distribution on $[0,1)$. That is, the trained RL is expected to finally match the reference trajectory, starting from any randomly generated initial state by equation (3.6). The observation is set as the discrepancy between the PDE state and reference state at the same control step $\boldsymbol{u}^o = \boldsymbol{u} - \boldsymbol{u}_{re}$. The environment is simulated numerically based on the finite difference methods, where the convection term and diffusion term are discretized by the second-order upwind scheme and the fourth-order central difference scheme, respectively. Euler method is used for the time integration. The simulated environment is defined on a spatial mesh of 150 grid points and the numerical time step is set as 0.01. The control signal is applied every 500 numerical steps and one episode contains 60 control steps. The reward function is defined as $-10||\boldsymbol{u}^o||_{L_2}$. Without the RL training, random control signals are applied to the system, and the corresponding spatiotemporal state surface $\boldsymbol{u}$ is shown in figure 7*a*. We can see that the state surface is unsmooth, and a large discrepancy remains between the uncontrolled state with the reference state. Figure 7*c* shows one of the test episodes controlled by the trained PiMBRL agent. The corresponding actions and rewards are given in figure 7*d*. Although the initial state is far from the reference, the controlled surface gradually approaches and finally matches the reference state after $t = 40$. The corresponding action curves are smooth, suggesting that the RL agent successfully learns an effective control strategy.

Figure 8 shows the performance curves of the PiMBRL, MBRL, MFRL tested on 100 randomly selected initial conditions. The PiMBRL reaches the total return of 0.1 only after 800 time steps, while it takes the MBRL about 1300 time steps to achieve a similar level of performance. The MFRL counterpart cannot reach the same level of performance within 1400 time steps. Again, our PiMBRL shows significant advantages in terms of sample efficiency, since it only uses about 65% of time steps required by MBRL and 46.7% time steps required by MFRL to achieve the control goal.

(ii) Kuramoto–Sivashinsky equation

In the last case, we evaluate the proposed PiMBRL on the control of a nonlinear, chaotic dynamic system governed by the one-dimensional KS equation, which is more challenging. The system governed by the KS equation often exhibits spatio-temporally chaotic or weakly turbulent behaviour, and thus the KS equation is widely used as a model system for turbulence study [68]. In this case, the KS environment is controlled by four actuators distributed equally in space to

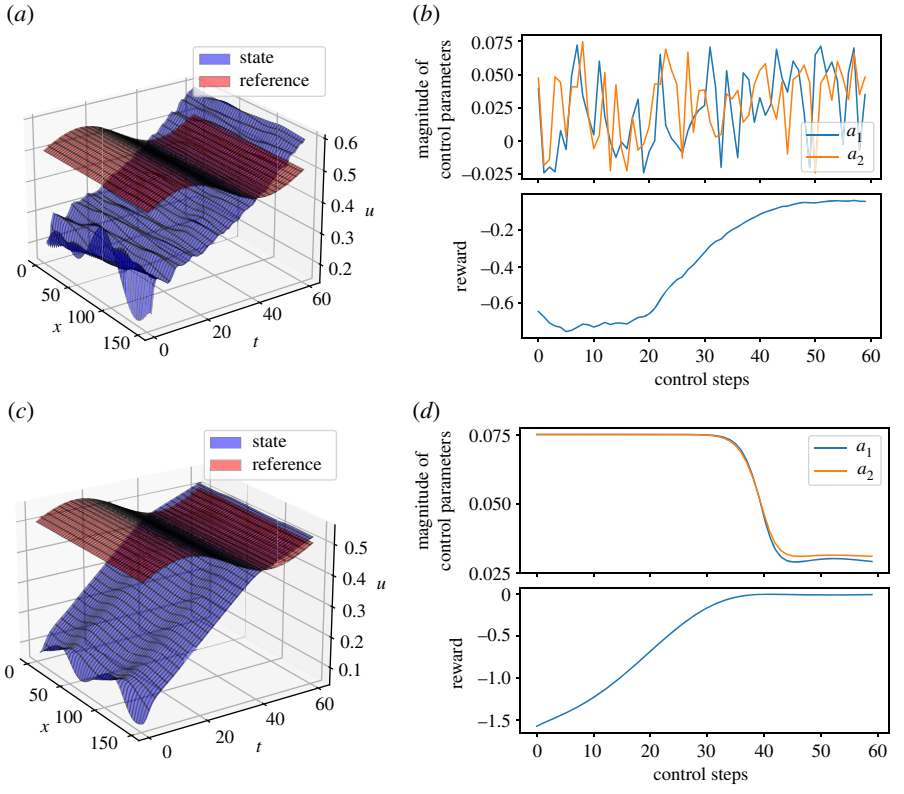

Figure 7. Results of one test episode (*a*) with random control signals and (*c*) with trained RL controller. The corresponding actions and reward curves of the (*b*) uncontrolled episode and (*d*) RL controlled episode. (*a*) Uncontrolled, (*b*) uncontrolled actions and rewards, (*c*) controlled, (*d*) controlled actions and rewards. (Online version in colour.)

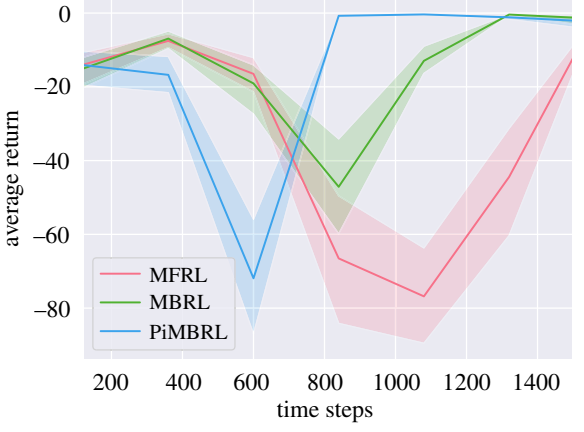

Figure 8. Performance curve of PiMBRL versus standard MFRL and MBRL in the Burgers' equation environment. The solid lines indicate averaged returns of 100 test episodes, while the shaded area represents the return distribution of all test samples. (Online version in colour.)

minimize the energy dissipation and total input power. The physics of this system is governed by the KS equation

$$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} + \frac{\partial^4 u}{\partial x^4} + u\frac{\partial u}{\partial x} = f(x,t),\ x \in [0,l],\ t \in [0,\infty), \tag{3.8}$$

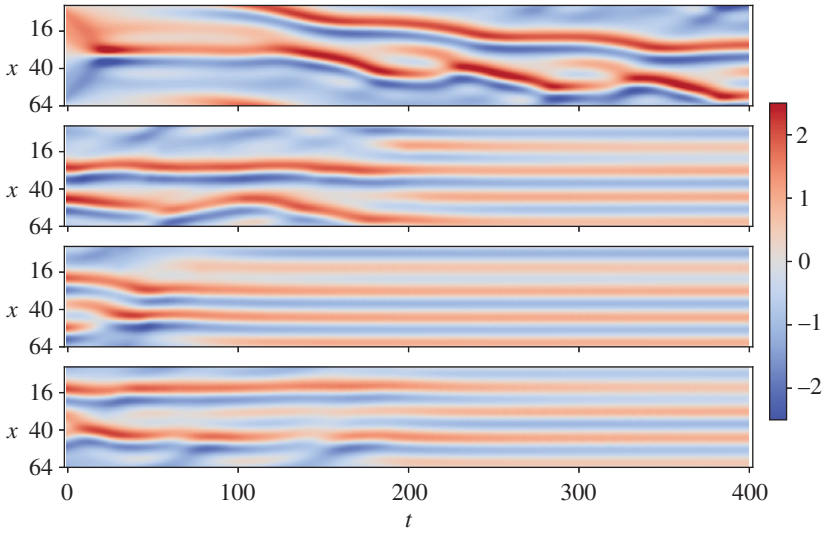

Figure 9. Contours of spatio-temporal state u of four randomly selected episodes. The top one is an uncontrolled episode, while the other three are controlled by the trained RL agent. (Online version in colour.)

where u is the state variable, and f represents the source term (i.e. actuator) defined by

$$f(x,t)=\sum_{i=1}^{4}\frac{a_i(t)\,\mathrm{e}^{-(x-x_i)^2/2}}{\sqrt{2\pi}}, \tag{3.9}$$

where $x_i\in\{0,\,l/4,\,l/2,\,3l/4\}$ is the spatial locations of the actuator, and $\boldsymbol{a}=\{a_i(t)\}_{i=1,2,3,4}\in[-0.5,0.5]^4$ defines the control parameters. To achieve the control goal, the reward function is defined as follows:

$$r=-\frac{1}{Tl}\int_{t_0}^{t_0+T}\int_0^l\left(\left(\frac{\partial^2 u}{\partial x^2}\right)^2+\left(\frac{\partial u}{\partial x}\right)^2+uf\right)\mathrm{d}x\,\mathrm{d}t, \tag{3.10}$$

where T is the time length of one control step. The environment is simulated numerically based on the finite difference method, where the convection term is discretized by the second-order upwind scheme, the second and fourth derivatives are discretized by the sixth-order central difference scheme, and the fourth-order Runge–Kutta scheme is used for time integration with time stepping size of 0.001 on the one-dimensional domain $l=8\pi$ discretized by a mesh of 64 grid points. Each control step contains 250 numerical steps, and one episode consists of 400 control steps. Each episode starts with a random initial condition sampled from the attractor of the unforced KS equation. Figure 9 shows the spatiotemporal states of four test episodes with randomly sampled initial states. The top one is an uncontrolled episode, where nonlinear chaotic behaviour is developed along the time axis. By contrast, the 'turbulence' in the other three episodes controlled by the agent can be quickly stabilized after 200 time steps, showing the effectiveness of the PiMBRL controller.

Figure 10 shows the performance curve of PiMBRL versus that of the MFRL. It is clear that the PiMBRL agent reaches higher averaged total returns with fewer time steps than the MFRL counterpart does. The PiMBRL performance curve is consistently above that of the MFRL approach, and meanwhile, less uncertainty is observed. As shown by figure 10*b*, MFRL needs more than 400 000 time steps to barely reach the return level of −55, deemed as the threshold level of an acceptable policy in KS environments. In stark contrast, the PiMBRL agent only uses about 15% of the time steps required by its MFRL counterpart to reach the averaged total return of −55. As for the uncertainty region, both the lower and upper envelopes of the PiMBRL (blue region) are notably higher than those of the MFRL (red region) at almost every time step. In this

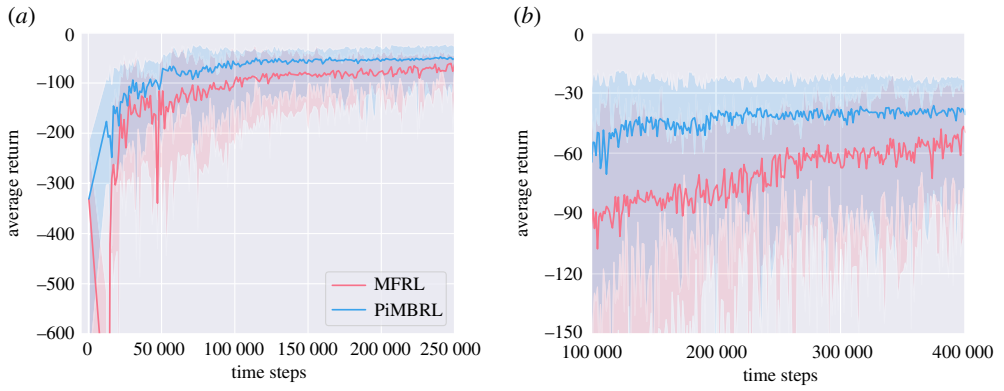

Figure 10. Performance curve of PiMBRL versus MFRL and MBRL in KS equation environment. Solid lines shows the average return of all the test episodes while the shaded area represents the distribution range of the 200 test episodes. (*a*) 0 to 2.5×10^5 time steps, (*b*) 1.0×10^6 to 4.0×10^6 time steps. (Online version in colour.)

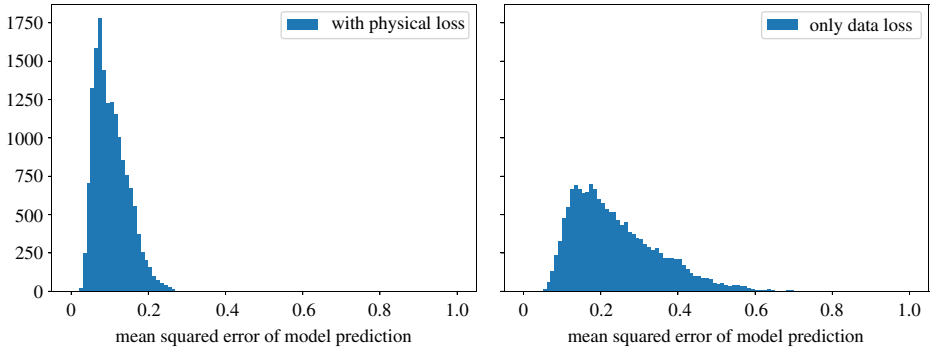

Figure 11. Histograms of the prediction errors (MSE) of the transition models for the KS system, (*a*) trained with physics-informed loss in PiMBRL, and (*b*) trained with the data loss only in MBRL. Both are trained on the same real buffer with 10^5 time steps stored. (Online version in colour.)

case, a model-free fine-tuning approach [69] is applied when the average return is above −55 to further improve the PiMBRL performance. This is because when the RL agent is trained to achieve a high accuracy level, the model-based exploration does not help too much while the model bias becomes the bottleneck. At this point, the RL agent can be fine-tuned by interacting with the real environment without using the model [69]. As such, the PiMBRL and MFRL performance curves will eventually converge to each other with a large number of training time steps.

The MBRL result is not plotted here because the accuracy (L_D) of the learned model purely based on labelled data does not meet the accuracy threshold $\lambda = 0.01$, and thus the MBRL performance curve is identical to that of the MFRL counterpart. In contrast to the ODE-based pendulum system shown above, the spatiotemporal dynamics of the chaotic system is much more challenging for the model to learn purely based on the limited amount of labelled data. Incorporating physics constraints can significantly help this scenario and improve the model learning performance. This can be clearly seen in figure 11, where the comparison of model prediction errors of PiMBRL (*a*) and MBRL (*b*) are shown. The mean squared error distribution of the model trained based on the physics-informed loss is much less than that of the model trained purely based on the labelled data.

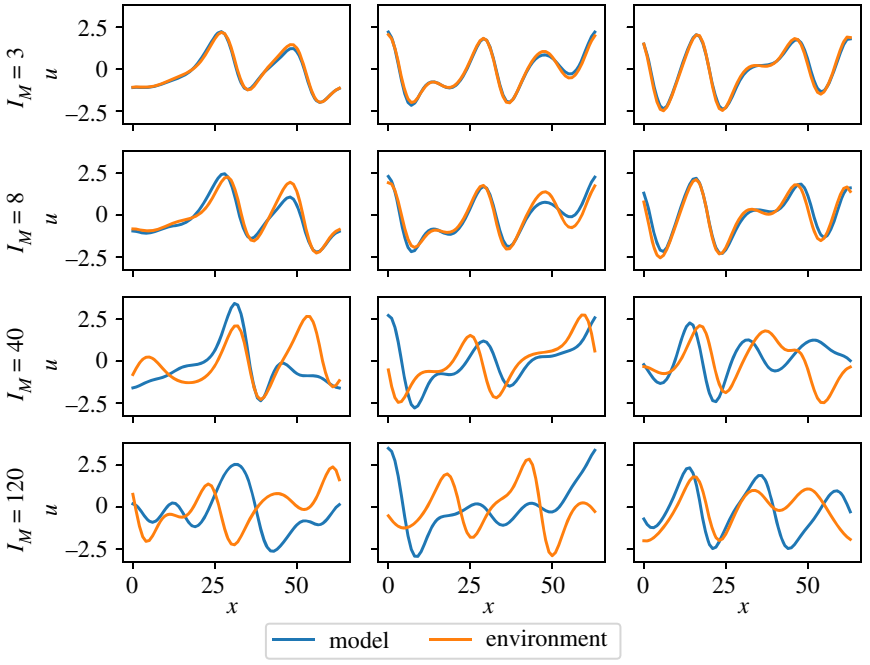

Figure 12. Model predicted snapshots of the KS environment with different rollout lengths compared with the true environment at three randomly selected initials. (Online version in colour.)

4. Discussion

(a) Influence of model rollout length

The rollout length of the model is important to the RL performance. On the one hand, a long-term model prediction could help the agent see 'deeper and further', improving the exploration rate and increasing the sample efficiency in the Dyna-like MBRL framework. On the other hand, accurately predicting a long trajectory is always challenging due to the error accumulation effect, and inaccurate prediction data could be harmful to the RL training, which is a trade-off. Figure 12 shows the model prediction performance for the KS environment with four different rollout lengths ($l_M = 3, 8, 40, 120$). For all three randomly selected initials, we can see that the states predicted by the model agree well with the ground truth when the rollout length is small ($l_M \leq 8$). However, large discrepancies can be observed when $l_M \geq 40$ and the model predictions significantly deviate from the ground truth when the rollout length is 120.

We are more interested in how the model rollout length affects the PiMBRL performance, so we investigated the influence of four different rollout lengths on the reward curves. The case setting is the same as that used in §3b(ii) except for the rollout length, and the RL performance is evaluated on 200 randomly generated episodes. Figure 13 shows the performance curves of PiMBRL with different model prediction lengths. As expected, the RL performance slightly deteriorates if the rollout length of the model is either too short or too long. For the four rollout lengths ($l_M = 3, 8, 40, 120$), the RL agent achieves the best performance with $l_M = 8$ at almost any stage of the entire training process. When $l_M \leq 8$ or $l_M \geq 8$, the RL convergence speed is relatively slow before entering the fine-tuning stage due to the trade-off between exploration and model bias. However, even after the model-free fine tuning, the RL agent with a longer rollout length ($l_M \geq 40$) still suffers from the low-quality model prediction data and is very difficult to be further improved (see the comparison of the green and blue curves in figure 13).

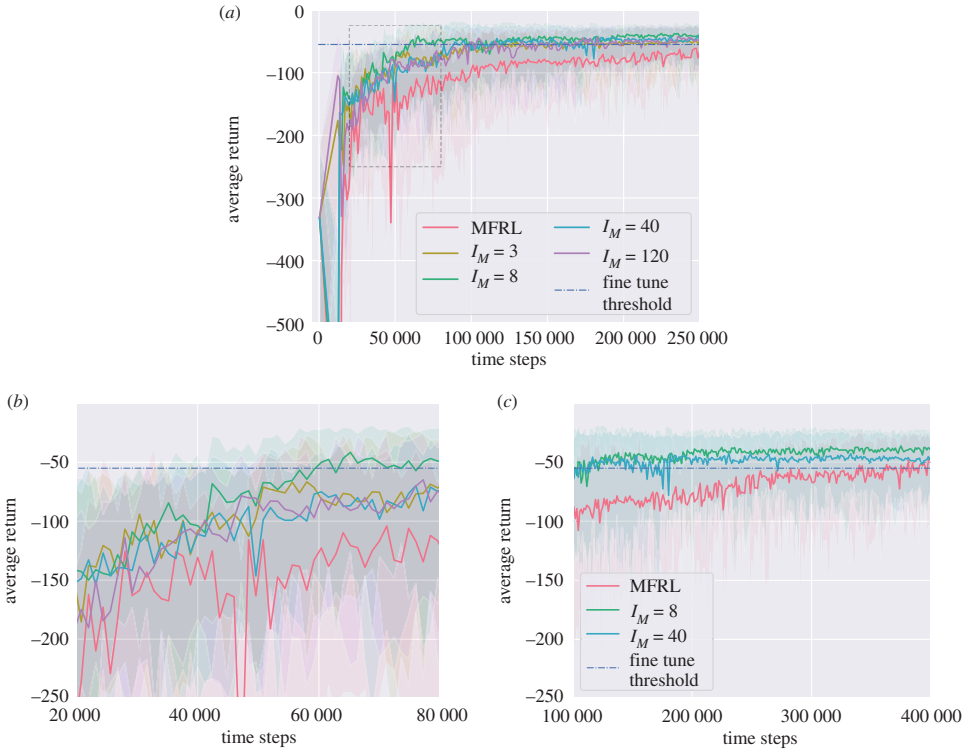

Figure 13. Performance curves of PiMBRL with four different model rollout lengths compared with MFRL base line. (*b*) Zoom-in view of the dashed box. (*c*) Zoom-in view of the range between 1×10^5 and 4×10^5 time steps. (Online version in colour.)

(b) Influence of model accuracy threshold

In model-based RL, the transition model is trained along with the RL agent (i.e. value and policy networks) from scratch, and the model-generated data usually can be effectively used once the model is trained to reach a certain level of accuracy. As mentioned above, we use the model accuracy threshold parameter (λ) to determine when the model-predicted data should be used for the RL training. Here, we would like to study how this parameter of λ affects the PiMBRL performance.

Using the KS environment as an example, figure 14 shows the performance curves of PiMBRL with three different threshold values, i.e. $\lambda = 0.02, 0.01, 0.005$. Overall, the influence of λ values on the final performance of the RL agent is negligible since all cases converge to the same level of total return. This is because the model is trained together with the RL agent and can be improved over the entire RL training process. Actually, with the same amount of training time steps in the real environment, the models can reach a similar accuracy level regardless of threshold values λ. However, the RL agent with a relatively large threshold value ($\lambda = 0.02$) performs slightly better and has a relatively faster convergence rate at the early stage of the training (time step 40 000–60 000). A higher threshold allows the agent to access more data generated by the model earlier, which leads to a better exploration rate and thus slightly higher sample efficiency (figure 14*b*).

(c) Limitations and perspectives

Compared with MFRL algorithms, MBRL is believed to have higher sample efficiency and thus more attractive to application scenarios where repeatedly interacting with the real environment is costly or impossible. However, for certain cases where the environment interactions are cheap and fast (e.g. via high-frequency sensors), MFRL could be more efficient than its Dyna-MBRL

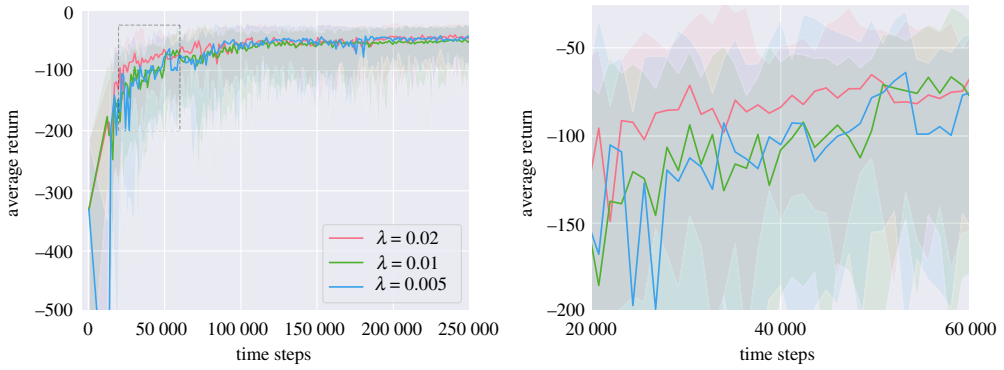

Figure 14. Performance curves of PiMBRL with three different model accuracy threshold values $\lambda = 0.02, 0.01, 0.005$, where (*a*) is the full view of the entire training history and (*b*) shows the zoom-in view of the dashed box area in (*a*). (*a*) Full view, (*b*) zoom-in view. (Online version in colour.)

counterpart since it does not suffer from model-bias issues. Moreover, deploying multiple controllers in parallel using a multi-agent setting may also have the potential to alleviate the sample complexity issues in MFRL, particularly for systems with strong invariance [70].

This work tackles the challenges in MBRL from a different angle. The main contribution lies in incorporating prior knowledge into model learning to constrain model bias for the Dyna-style MBRL structure, and significant improvements have been demonstrated on several classic control problems. Nonetheless, the current development has several limitations. First, although the proposed PiMBRL has been tested on several classic control problems for proof-of-concept, further studies are still needed to fill gaps for ultimately solving complex real-world challenges, e.g. three-dimensional turbulence control. Second, this work focuses on systems with well-posed physics (i.e. systems with known governing equations and physical parameters). However, for some systems whose underlying physics remains unknown or too complex to be expressed as explicit PDEs/ODEs, the PiMBRL in its current form is not directly applicable, and further extensions are needed. For example, one common scenario is that the governing equations are known, but parameters are not given. In this case, the PiMBRL can be extended by assimilating data collected from interactions with the environment to infer the unknown parameters during the training, since physics-informed networks are good at solving both forward and inverse problems in a unified manner [48,71]. A more challenging scenario is that the system's governing equations are not known explicitly. In this case, the proposed framework can also be extended to discover the explicit equation forms by coupling the sparsity-promoting learning techniques. For example, the pointwise PINN has been combined with library-based sparse regression techniques to discover explicit equation forms from sparse and noise observations [72].

Lastly, we acknowledge that we have not conducted a throughout, comprehensive parametric study, which is not the focus of this work. There are many factors that can be studied in the future to further improve the performance. For example, the current PiMBRL is built upon TD3, but the proposed framework is generic and can be coupled with other policy learning algorithms, e.g. DDPG and SAC. Moreover, the generic RL hyperparameters remain the same for all test cases and have not been fine-tuned. Adaptive parameter tuning for a specific application may be able to further improve the performance.

5. Conclusion

In this work, we presented an innovative model-based reinforcement learning framework (PiMBRL) for dynamic control, which leverages the physical laws and constraints of the environment to alleviate the model-bias issue and significantly improve the sample efficiency. In particular, an autoencoding-based recurrent network structure is devised to learn the

spatiotemporal dynamics of the environments. The merit and effectiveness of the proposed PiMBRL framework have been demonstrated over a set of classic dynamic control problems, where the environments are governed by canonical ODEs or PDEs, including viscous Burgers' equations and KS equations with chaotic behaviours. Compared with the model-free or purely data-driven model-based reinforcement learning counterparts, our PiMBRL shows a significant improvement in model predictive accuracy and RL sample efficiency. Moreover, the effects of different hyper-parameters used in PiMBRL (e.g. model rollout length l_M and model accuracy threshold λ) are studied, and how these parameters affect the RL performance is discussed.

Data accessibility. The data that support the findings of this study will be openly available in GitHub at https://github.com/Jianxun-Wang/PIMBRLuponpublication.

Authors' contributions. X.-Y.L.: participated in the design of the study, implemented the entire framework, carried out all numerical experiments and drafted the manuscript. J.-X.W.: conceived of the study, drafted and revised the manuscript, and supervised the project.

Competing interests. We declare we have no competing interests.

Funding. This work is funded by the National Science Foundation under award nos. CMMI-1934300 and OAC-2047127 and start-up funds from the College of Engineering at University of Notre Dame.

Acknowledgements. The authors thank two anonymous reviewers for their insightful comments and suggestions to improve the quality of this work.

Appendix A

(a) Twin-delayed deep deterministic policy gradient (TD3)

Algorithm 2. Model-free twin-delayed deep deterministic policy gradient (TD3).

Initialize policy (actor) network $\pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$, value (critic) networks $q_1(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_{q_1})$, $q_2(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_{q_2})$, empty the replay buffer $\mathscr{D}^r$ and reset the environment $\mathcal{F}$.
Make a copy of policy and value networks as target networks $\pi_{targ} \leftarrow \pi$, $q_{targ,1} \leftarrow q_1$, $q_{targ,2} \leftarrow q_2$
for time steps $t = 1, N$ **do**
  Execute action $\boldsymbol{a}_i = \pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$ in the environment $\mathcal{F}$;
  Save new data pair $(\boldsymbol{u}_i^o, \boldsymbol{a}_i, \boldsymbol{u}_{i+1}^o, r_i, d_i)$ to buffer $\mathscr{D}^r$;
  if episode ends **then** Reset the environment $\mathcal{F}$;
  end if
  if t mod `update_every` $== 0$ **then**
    if enough state-action pairs stored in $\{\mathscr{D}^r, \mathscr{D}^f\}$ **then**
      for $k = 1, I_{RL}$ **do**
        Sample J state-action pairs $\{(\boldsymbol{u}_j^o, \boldsymbol{a}_j, \boldsymbol{u}_{j+1}^o, r_j, d_j)\}$ from buffer $\mathscr{D}^r$
        Compute target

$$Q_j = r_j + \gamma(1 - d_j) \min_{i=1,2} \left\{ q_{targ,i}(\boldsymbol{u}_{j+1}^o, \pi_{targ}(\boldsymbol{u}_{j+1}^o)) \right\}$$

        Update value networks via gradient descent,

$$\nabla_{\boldsymbol{\theta}_{q_i}} \frac{1}{J} \sum_{j=1}^{J} \left[q_i(\boldsymbol{u}_j^o, \boldsymbol{a}_j) - Q_j \right]^2, \qquad i = 1, 2$$

        if k mod $2 == 0$ **then**
          Update policy network via gradient descent:

$$\nabla_{\boldsymbol{\theta}_\pi} \frac{1}{J} \sum_{j=1}^{J} q_1(\boldsymbol{u}_j^o, \pi(\boldsymbol{u}_j^o))$$

          Update target networks:

$$\boldsymbol{\theta}_{q_{targ,i}} \leftarrow \rho \boldsymbol{\theta}_{q_{targ,i}} + (1 - \rho)\boldsymbol{\theta}_{q_i}, \qquad i = 1, 2$$

$$\boldsymbol{\theta}_{\pi_{targ}} \leftarrow \rho \boldsymbol{\theta} \pi_{targ} + (1 - \rho)\boldsymbol{\theta}\pi$$

        end if
      end for
    end if
  end if
end for

Algorithm 3. TD3-based policy and value network update algorithm used in MBRL/PiMBRL.

With buffer $\mathscr{D}^r$, and $\mathscr{D}^f$, policy network $\pi(\boldsymbol{u};\boldsymbol{\theta}_\pi)$, value networks $q_1(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_{q_1})$, $q_2(\boldsymbol{u},\boldsymbol{a};\boldsymbol{\theta}_{q_2})$ and their corresponding target networks $\pi_{targ}, q_{targ,1}, q_{targ,2}$

for $k = 1, I_{RL}$ **do**

Sample J state-action pairs $\{(\boldsymbol{u}_j^o, \boldsymbol{a}_j, \boldsymbol{u}_{j+1}^o, r_j, d_j)\}$ from buffer $\{\mathscr{D}^r, \mathscr{D}^f\}$

Compute target

$$Q_j = r_j + \gamma(1-d_j)\min_{i=1,2}\left\{q_{targ,i}(\boldsymbol{u}_{j+1}^o, \pi_{targ}(\boldsymbol{u}_{j+1}^o))\right\}$$

Update value networks by gradient descent:

$$\nabla_{\boldsymbol{\theta}_{q_i}}\frac{1}{J}\sum_{j=1}^{J}\left[q_i(\boldsymbol{u}_j^o, \boldsymbol{a}_j) - Q_j\right]^2, \qquad i = 1,2$$

if k mod 2 == 0 **then**

Update policy network by gradient descent:

$$\nabla_{\boldsymbol{\theta}_\pi}\frac{1}{J}\sum_{j=1}^{J} q_1(\boldsymbol{u}_j^o, \pi(\boldsymbol{u}_j^o))$$

Update target networks:

$$\boldsymbol{\theta}_{q_{targ,i}} \leftarrow \rho\boldsymbol{\theta}_{q_{targ,i}} + (1-\rho)\boldsymbol{\theta}_{q_i}, \qquad i = 1,2$$

$$\boldsymbol{\theta}_{\pi_{targ}} \leftarrow \rho\boldsymbol{\theta}\pi_{targ} + (1-\rho)\boldsymbol{\theta}\pi$$

end if

end for

(b) Hyper-parameter settings

Table 1. Generic hyper-parameters of TD3 algorithm.

parameter	discount rate γ	P_D	I_{RL}	policy and Q-value network optimizer
value	0.99 (0.977 in KS)	2	50	Adam
parameter	polyak ρ	action noise	replay buffer size	policy and Q-value network learning rate
value	0.995	20%	5×10^5	1×10^{-3}

Table 2. Hyper-parameters of PiMBRL (§3).

environment	model accuracy threshold L_D	model rollout length l_M	maximum episode length	n_{s_M}
Cart-Pole	1×10^{-4}	200 (full length)	200	800
Pendulum	1×10^{-2}	200 (full length)	200	6000
Burgers'	1×10^{-2}	1	60	120
KS	1×10^{-2}	3	400	6000
environment	RL batch size	model batch size	update every	n_{s_R}
Cart-Pole	128	200	128	1000
Pendulum	128	200	128	12 000
Burgers'	120	120	120	120
KS	128	400	400	12 000

References

1. Silver D *et al.* 2017 Mastering the game of go without human knowledge. *Nature* **550**, 354–359. (doi:10.1038/nature24270)
2. Silver D *et al.* 2018 A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **362**, 1140–1144. (doi:10.1126/science.aar6404)
3. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M. 2013 Playing Atari with deep reinforcement learning. (http://arxiv.org/abs/1312.5602)
4. Rabault J, Kuchta M, Jensen A, Réglade U, Cerardi N. 2019 Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *J. Fluid Mech.* **865**, 281–302. (doi:10.1017/jfm.2019.62)
5. Ghraieb H, Viquerat J, Larcher A, Meliga P, Hachem E. 2021 Single-step deep reinforcement learning for open-loop control of laminar and turbulent flows. *Phys. Rev. Fluids* **6**, 053902. (doi:10.1103/PhysRevFluids.6.053902)
6. Ren F, Rabault J, Tang H. 2021 Applying deep reinforcement learning to active flow control in weakly turbulent conditions. *Phys. Fluids* **33**, 037121. (doi:10.1063/5.0037371)
7. Fan D, Yang L, Wang Z, Triantafyllou MS, Karniadakis GE. 2020 Reinforcement learning for bluff body active flow control in experiments and simulations. *Proc. Natl Acad. Sci. USA* **117**, 26 091–26 098. (doi:10.1073/pnas.2004939117)
8. Bucci MA, Semeraro O, Allauzen A, Wisniewski G, Cordier L, Mathelin L. 2019 Control of chaotic systems by deep reinforcement learning. *Proc. R. Soc. A* **475**, 20190351. (doi:10.1098/rspa.2019.0351)
9. Beintema G, Corbetta A, Biferale L, Toschi F. 2020 Controlling Rayleigh–Bénard convection via reinforcement learning. *J. Turbul.* **21**, 585–605. (doi:10.1080/14685248.2020.1797059)
10. Garnier P, Viquerat J, Rabault J, Larcher A, Kuhnle A, Hachem E. 2021 A review on deep reinforcement learning for fluid mechanics. *Comput. Fluids* **225**, 104973. (doi:10.1016/j.compfluid.2021.104973)
11. Falk MJ, Alizadehyazdi V, Jaeger H, Murugan A. 2021 Learning to control active matter. (http://arxiv.org/abs/2105.04641)
12. Gustavsson K, Biferale L, Celani A, Colabrese S. 2017 Finding efficient swimming strategies in a three-dimensional chaotic flow by reinforcement learning. *Eur. Phys. J. E* **40**, 1–6. (doi:10.1140/epje/i2017-11602-9)
13. Verma S, Novati G, Koumoutsakos P. 2018 Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proc. Natl Acad. Sci. USA* **115**, 5849–5854. (doi:10.1073/pnas.1800923115)
14. Zhu Y, Tian F-B, Young J, Liao JC, Lai JCS. 2021 A numerical study of fish adaption behaviors in complex environments with a deep reinforcement learning and immersed boundary–lattice Boltzmann method. *Sci. Rep.* **11**, 1–20. (doi:10.1038/s41598-020-79139-8)
15. Hwangbo J, Sa I, Siegwart R, Hutter M. 2017 Control of a quadrotor with reinforcement learning. *IEEE Rob. Autom. Lett.* **2**, 2096–2103. (doi:10.1109/LRA.2017.2720851)
16. Wada D, Araujo-Estrada SA, Windsor S. 2021 Unmanned aerial vehicle pitch control using deep reinforcement learning with discrete actions in wind tunnel test. *Aerospace* **8**, 18. (doi:10.3390/aerospace8010018)
17. Deng Y, Liu T, Zhao D. 2021 Event-triggered output-feedback adaptive tracking control of autonomous underwater vehicles using reinforcement learning. *Appl. Ocean Res.* **113**, 102676. (doi:10.1016/j.apor.2021.102676)
18. Bhagat S, Banerjee H, Ho Tse ZT, Ren H. 2019 Deep reinforcement learning for soft, flexible robots: brief review with impending challenges. *Robotics* **8**, 4. (doi:10.3390/robotics8010004)
19. Nagabandi A, Konolige K, Levine S, Kumar V. 2020 Deep dynamics models for learning dexterous manipulation. In *Conf. on Robot Learning, Proc. of Machine learning Research, 16–18 November*, pp. 1101–1112. New York, NY: PMLR.
20. Li T, Lambert N, Calandra R, Meier F, Rai A. 2020 Learning generalizable locomotion skills with hierarchical reinforcement learning. In *2020 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 413–419. IEEE.
21. Hachem E, Ghraieb H, Viquerat J, Larcher A, Meliga P. 2021 Deep reinforcement learning for the control of conjugate heat transfer. *J. Comput. Phys.* **436**, 110317. (doi:10.1016/j.jcp.2021.110317)

22. Viquerat J, Rabault J, Kuhnle A, Ghraieb H, Larcher A, Hachem E. 2021 Direct shape optimization through deep reinforcement learning. *J. Comput. Phys.* **428**, 110080. (doi:10.1016/j.jcp.2020.110080)
23. Botvinick M, Ritter S, Wang JX, Kurth-Nelson Z, Blundell C, Hassabis D. 2019 Reinforcement learning, fast and slow. *Trends Cogn. Sci.* **23**, 408–422. (doi:10.1016/j.tics.2019.02.006)
24. Hessel M et al. 2018 Rainbow: combining improvements in deep reinforcement learning. In *Proc. of the AAAI Conf. on Artificial Intelligence*, vol. 32. Menlo Park, CA: AAAI.
25. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D. 2015 Continuous control with deep reinforcement learning. (http://arxiv.org/abs/1509.02971)
26. Fujimoto S, Hoof H, Meger D. 2018 Addressing function approximation error in actor-critic methods. In *Int. Conf. on Machine Learning*, pp. 1587–1596. New York, NY: PMLR.
27. Haarnoja T, Zhou A, Abbeel P, Levine S. 2018 Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Int. Conf. on Machine Learning*, pp. 1861–1870. New York, NY: PMLR.
28. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. 2015 Trust region policy optimization. In *Int. Conf. on machine learning*, pp. 1889–1897. New York, NY: PMLR.
29. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. 2017 Proximal policy optimization algorithms. (http://arxiv.org/abs/1707.06347)
30. Moerland TM, Broekens J, Jonker CM. 2020 Model-based reinforcement learning: a survey. (http://arxiv.org/abs/2006.16712)
31. Plaat A, Kosters W, Preuss M. 2020 Model-based deep reinforcement learning for high-dimensional problems, a survey. (http://arxiv.org/abs/2008.05598)
32. Deisenroth M, Rasmussen CE. 2011 Pilco: A model-based and data-efficient approach to policy search. In *Proc. of the 28th Int. Conf. on machine learning (ICML-11)*, pp. 465–472. Citeseer.
33. Tassa Y, Erez T, Todorov E. 2012 Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 4906–4913. Piscataway, NJ: IEEE.
34. Levine S, Koltun V. 2013 Guided policy search. In *Int. Conf. on machine learning*, pp. 1–9. PMLR.
35. Racanière S *et al.* 2017 Imagination-augmented agents for deep reinforcement learning. In *Proc. of the 31st Int. Conf. on Neural Information Processing Systems*, pp. 5694–5705. New York, NY: Curran Associates.
36. Kaiser L *et al.* 2020 Model based reinforcement learning for Atari. In *Int. Conf. on Learning Representations*. See https://openreview.net/forum?id=S1xCPJHtDB.
37. Hafner D, Lillicrap T, Fischer I, Villegas R, Ha D, Lee H, Davidson J. 2019 Learning latent dynamics for planning from pixels. In *Int. Conf. on Machine Learning*, pp. 2555–2565. PMLR.
38. Hafner D, Lillicrap T, Ba J, Norouzi M. 2020 Dream to control: Learning behaviors by latent imagination. In *Int. Conf. on Learning Representations*. See https://openreview.net/forum?id=S1lOTC4tDS.
39. Depeweg S, Hernández-Lobato JM, Doshi-Velez F, Udluft S. 2016 Learning and policy search in stochastic dynamical systems with Bayesian neural networks. (http://arxiv.org/abs/1605.07127).
40. Kurutach T, Clavera I, Duan Y, Tamar A, Abbeel P. 2018 Model-ensemble trust-region policy optimization. In *Int. Conf. on Learning Representations*. See https://openreview.net/forum?id=SJJinbWRZ
41. Abdar M *et al.* 2021 A review of uncertainty quantification in deep learning: techniques, applications and challenges. *Inf. Fusion* **76**, 243–297. (doi:10.1016/j.inffus.2021.05.008)
42. Wang J-X, Roy CJ, Xiao H. 2018 Propagation of input uncertainty in presence of model-form uncertainty: a multifidelity approach for computational fluid dynamics applications. *ASCE-ASME J. Risk and Uncert. Eng. Syst. Part B Mech. Eng.* **4**, 011002. (doi:10.1115/1.4037452)
43. Wang J-X, Wu J-L, Xiao H. 2017 Physics-informed machine learning approach for reconstructing Reynolds stress modeling discrepancies based on DNS data. *Phys. Rev. Fluids* **2**, 034603. (doi:10.1103/PhysRevFluids.2.034603)
44. Ling J, Kurzawski A, Templeton J. 2016 Reynolds averaged turbulence modelling using deep neural networks with embedded invariance. *J. Fluid Mech.* **807**, 155–166. (doi:10.1017/jfm.2016.615)
45. Duraisamy K, Iaccarino G, Xiao H. 2019 Turbulence modeling in the age of data. *Annu. Rev. Fluid Mech.* **51**, 357–377. (doi:10.1146/annurev-fluid-010518-040547)

46. Raissi M, Perdikaris P, Karniadakis GE. 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686–707. (doi:10.1016/j.jcp.2018.10.045)
47. Sun L, Gao H, Pan S, Wang J-X. 2020 Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. *Comput. Methods Appl. Mech. Eng.* **361**, 112732. (doi:10.1016/j.cma.2019.112732)
48. Gao H, Sun L, Wang J-X. 2021 Super-resolution and denoising of fluid flow using physics-informed convolutional neural networks without high-resolution labels. *Phys. Fluids* **33**, 073603. (doi:10.1063/5.0054312)
49. Arzani A, Wang J-X, D'Souza RM. 2021 Uncovering near-wall blood flow from sparse data with physics-informed neural networks. *Phys. Fluids* **33**, 071905. (doi:10.1063/5.0055600)
50. Sahli Costabal F, Yang Y, Perdikaris P, Hurtado DE, Kuhl E. 2020 Physics-informed neural networks for cardiac activation mapping. *Front. Phys.* **8**, 42. (doi:10.3389/fphy.2020.00042)
51. Rao C, Sun H, Liu Y. 2021 Physics-informed deep learning for computational elastodynamics without labeled data. *J. Eng. Mech.* **147**, 04021043. (doi:10.1061/(ASCE)EM.1943-7889.0001947)
52. Sutton RS, Barto AG. 2018 *Reinforcement learning: an introduction*. Cambridge, MA: MIT press.
53. Gao H, Sun L, Wang J-X. 2021 PhyGeoNet: physics-informed geometry-adaptive convolutional neural networks for solving parameterized steady-state PDEs on irregular domain. *J. Comput. Phys.* **428**, 110079. (doi:10.1016/j.jcp.2020.110079)
54. Heess N, Wayne G, Silver D, Lillicrap T, Erez T, Tassa Y. 2015 Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems* (eds C Cortes, N Lawrence, D Lee, M Sugiyama, R Garnett), vol. 28. New York, NY: Curran Associates, Inc. (https://proceedings.neurips.cc/paper/2015/file/148510031349642de5ca0c544f31b2ef-Paper.pdf)
55. Levine S, Abbeel P. 2014 Learning neural network policies with guided policy search under unknown dynamics. In *Advances in neural information processing systems* (eds Z Ghahramani, M Welling, C Cortes, N Lawrence, KQ Weinberger), vol. 27, pp. 1071–1079. Red Hook, NY: Curran Associates.
56. Luo Y, Xu H, Li Y, Tian Y, Darrell T, Ma T. 2018 Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. (http://arxiv.org/abs/1807.03858)
57. Amari S. 1967 A theory of adaptive pattern classifiers. *IEEE Trans. Electron. Comput.* **EC-16**, 299–307. (doi:10.1109/PGEC.1967.264666)
58. Konečnáżş J, Richtárik P. 2013 Semi-stochastic gradient descent methods. (http://arxiv.org/abs/1312.1666)
59. Johnson R, Zhang T. 2013 Accelerating stochastic gradient descent using predictive variance reduction. *Adv. Neural Inf. Process. Syst.* **26**, 315–323.
60. Defazio A, Bach F, Lacoste-Julien S. 2014 SAGA: a fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in neural information processing systems* (eds Z Ghahramani, M Welling, C Cortes, N Lawrence, KQ Weinberger), pp. 1646–1654. Red Hook, NY: Curran Associates.
61. Schmidt M, Le Roux N, Bach F. 2017 Minimizing finite sums with the stochastic average gradient. *Math. Programm.* **162**, 83–112. (doi:10.1007/s10107-016-1030-6)
62. Williams RJ. 1992 Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **8**, 229–256. (doi:10.1007/BF00992696)
63. Watkins CJCH, Dayan P. 1992 Q-learning. *Mach. Learn.* **8**, 279–292. (doi:10.1007/BF00992698)
64. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N. 2016 Dueling network architectures for deep reinforcement learning. In *Proc. of the 33rd Int. Conf. on Machine Learning*, pp. 1995–2003 (eds MF Balcan, K Weinberger), vol. 48. New York, NY: PMLR.
65. Janner M, Fu J, Zhang M, Levine S. 2019 When to trust your model: model-based policy optimization. (http://arxiv.org/abs/1906.08253)
66. Wang T, Bao X, Clavera I, Hoang J, Wen Y, Langlois E, Zhang S, Zhang G, Abbeel P, Ba J. 2019 Benchmarking model-based reinforcement learning. (http://arxiv.org/abs/1907.02057)
67. Wiklund D, Peluso M. 2002 Quantifying and specifying the dynamic response of flowmeters. *Technical Papers-ISA* **422**, 463–476.
68. Cvitanović P, Davidchack RL, Siminos E. 2010 On the state space geometry of the Kuramoto–Sivashinsky flow in a periodic domain. *SIAM J. Appl. Dyn. Syst.* **9**, 1–33. (doi:10.1137/070705623)

69. Nagabandi A, Kahn G, Fearing RS, Levine S. 2018 Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, pp. 7559–7566. IEEE.
70. Belus V, Rabault J, Viquerat J, Che Z, Hachem E, Reglade U. 2019 Exploiting locality and translational invariance to design effective deep reinforcement learning control of the 1-dimensional unstable falling liquid film. *AIP Adv.* **9**, 125014. (doi:10.1063/1.5132378)
71. Gao H, Zahr MJ, Wang J-X. 2021 Physics-informed graph neural galerkin networks: a unified framework for solving pde-governed forward and inverse problems. (http://arxiv.org/abs/2107.12146)
72. Chen Z, Liu Y, Sun H. 2021 Physics-informed learning of governing equations from scarce data. *Nat. Commun.* **12**, 1–13.