2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Numerical evidence for sample efficiency of model-based over model-free reinforcement learning control of partial differential equations

1st Stefan Werner
SICK Sensor Intelligence
Waldkirch, Germany
stefanwerner1997@gmail.com

2nd Sebastian Peitz
Department of Computer Science, Paderborn University
Paderborn, Germany
sebastian.peitz@upb.de; ORCID: 0000-0002-3389-793X

***Abstract*—The goal of this paper is to make a strong point for the usage of dynamical models when using reinforcement learning (RL) for feedback control of dynamical systems governed by partial differential equations (PDEs). To breach the gap between the immense promises we see in RL and the applicability in complex engineering systems, the main challenges are the massive requirements in terms of the training data, as well as the lack of performance guarantees. We present a solution for the first issue using a data-driven surrogate model in the form of a convolutional Long-Short Term Memory network with actuation. We demonstrate that learning an actuated model in parallel to training the RL agent significantly reduces the total amount of required data sampled from the real system. Furthermore, we show that iteratively updating the model is of major importance to avoid biases in the RL training. Detailed ablation studies reveal the most important ingredients of the modeling process. We use the chaotic Kuramoto-Sivashinsky equation do demonstrate our findings.**

***Index Terms*—reinforcement learning, surrogate modeling, partial differential equations, feedback control**

I. Introduction

Feedback control of complex physical systems is an essential building block in almost any modern technology. We thus face the task of having to take control actions in a very short amount of time and for a system with highly complex, distributed dynamics (typically governed by nonlinear partial differential equations (PDEs)) that are difficult or even impossible to observe completely. In recent years, *reinforcement learning* [1] (RL) is gaining more and more popularity as a very powerful and real-time capable paradigm for feedback control. We have seen tremendous successes, not only in the area of games (e.g., [2]), but also in complex technical applications such as flow control [3] or nuclear fusion [4]. However, there are two drawbacks that limit the deployment of RL agents in real systems. The first point is that the amount of required training data often exceeds tens to hundreds of millions of samples [5], [6] such that the training becomes very expensive. Second, the control performance may vary strongly in between training runs, in particular for high-dimensional state and action space dimensions, where it is challenging to obtain the required amounts of training data. A popular approach to tackle these issues is the usage of model-based algorithms [7], see also [8] for an overview and a taxonomy. Therein, a surrogate model replaces the real environment to allow for a significant increase of the training on data created by said surrogate. It is widely accepted that this can reduce the amount of "real" data an agent consumes before converging in the case that accurate and generalizable surrogates can be learned [9]–[11], and in many situations, this model can be orders of magnitude faster than evaluating the environment model (e.g., numerically solving a PDE). Collecting data through *model-based rollouts* is advantageous as soon as they reduce the amount of computation time spent on numerical simulations or the number of time-consuming and impractical real-world trials. At the same time, models introduce approximation errors so that we may obtain inferior solutions [12].

Even though it seems clear that surrogates have the potential to improve the learning process, the trade off in terms of sample efficiency, computational complexity, and performance is only rarely quantified in an engineering context such as PDE control. In this paper, we thus discuss and evaluate a variety of design choices associated with model-based RL, adopt a more sophisticated optimization approach, and identify promising directions for future research. We discuss the various modeling steps (Section III) that are required in order to arrive at a model that possesses the required accuracy. We then thoroughly study the impact on the required amount of training data (Section IV), where an improvement by a factor of nine is observed for the case of the 1D Kuramoto-Sivashinsky equation. In both Sections III and IV, we conduct various ablation studies to identify the most important modeling techniques in the offline phase, as well as online updating strategies during deployment. Finally, we discuss the implications for the robustness of RL agents as well as future research directions in Section VI.

II. Reinforcement learning

Reinforcement learning (RL) aims to solve sequential decision-making problems mathematically defined as Markov Decision Processes (MDPs) (see, e.g., [1] for a detailed overview). It possesses a set $\mathcal{S}$ of system states and a set $\mathcal{A}$ of actions an *agent* may select among to exercise control. At each

Copyright ©2024 EUCA

discrete time step τ, the agent observes the state $\boldsymbol{s}_\tau \in \mathcal{S}$ of the environment and responds with an action $\boldsymbol{a}_\tau \in \mathcal{A}$. A stochastic transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ formalizes in what way the system state changes as a result thereof. In an MDP, the state transition is independent of past states and actions, meaning that it satisfies the *Markov property*. The reward signal r_τ then quantifies the quality of decision $\boldsymbol{a}_\tau$ taken in state $\boldsymbol{s}_\tau$ and is an instance of the stochastic reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathbb{R})$. Starting from an initial state $\boldsymbol{s}_0 \sim \mathcal{P}_0(\mathcal{S})$, the RL framework aims to maximize the expected sum of discounted future rewards $\mathbb{E}\left[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau\right]$, where $\gamma \in [0, 1]$ is the discount factor.

A policy π is a mapping $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ modeling the probability $\pi(\boldsymbol{a}|\boldsymbol{s})$ of taking action $\boldsymbol{a} \in \mathcal{A}$ in state $\boldsymbol{s} \in \mathcal{S}$ of the environment. Solving an MDP means finding an optimal policy $\pi^* = \arg\max_\pi \mathbb{E}_\pi \left[\sum_{\tau=0}^{\infty} \gamma^\tau r_\tau\right]$.

So-called *value functions* are useful concepts to formally define optimality conditions for policies. In substance, the *state value* $V^\pi(\boldsymbol{s}) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{\tau+k} | \boldsymbol{s}_\tau = \boldsymbol{s}]$ denotes what future rewards to expect following policy π when the environment is in state $\boldsymbol{s}$. Therefore, a policy π is optimal if its state value $V^\pi(\boldsymbol{s})$ at each state $\boldsymbol{s} \in \mathcal{S}$ is at least as large as the state values of any other policy. Similarly, *state-action values* $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ describe what rewards to expect once action $\boldsymbol{a}$ is selected in state $\boldsymbol{s}$ and policy π governs the behavior thereafter: $Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{\tau+k} | \boldsymbol{s}_\tau = \boldsymbol{s}, \boldsymbol{a}_\tau = \boldsymbol{a}]$.

In *Deep Reinforcement Learning*, deep neural networks are typically used as function approximators to learn policies and value functions. For continuous control tasks (that is, continuous state and/or action spaces as in PDE control), *policy gradient methods* (e.g., the *Proximal Policy Optimization (PPO)* [13], the *deep deterministic policy gradient (DDPG)* [14] or the *Soft Actor Critic (SAC)* [15]) are the most prominent methods.

A. Model-based reinforcement learning

Model-based algorithms use a *dynamics model* to capture the changes in the environment, i.e., they approximate the state transition map $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ by a surrogate $f_{\boldsymbol{\theta}} \approx \mathcal{T}$, where $\boldsymbol{\theta}$ are trainable parameters.

The model-based framework extends the optimization procedure of model-free reinforcement learning with additional *planning* [1] and model-learning steps. Our work focuses on models learned from data for two reasons. The phenomena defining the physics as well as their parameter values (for example mass or viscosity parameters) are often not known exactly. At the same time, computational models solving the governing PDEs often do not meet real-time constraints, which is essential for online planning.

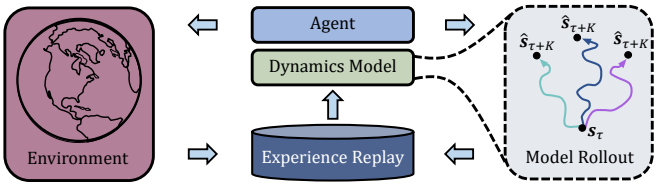

Fig. 1. In model-based RL with learned models, collected data serves two purposes. Samples are used to learn a model of the system dynamics and to estimate update targets for the model-free agent. Using rollouts of the agent in the model, algorithms collect additional data to improve their behavior. In the ideal absence of prediction errors, the model matches the environment without its downsides of real-world trials or severe computational costs.

B. Reinforcement learning for PDE control

In the literature on partial differential equations, control problems are usually defined on a continuous time scale (in contrast to the discrete-time decision-making underpinning reinforcement learning). The system state $\boldsymbol{u} : \Omega \times [0, T] \to \mathbb{R}^n$ is a function of time $t \in [0, T]$ and space $x \in \Omega$, and the dynamics is described by a nonlinear partial differential operator $\mathcal{N}$, i.e.,

$$\frac{\partial \boldsymbol{u}}{\partial t} = \mathcal{N}(\boldsymbol{u}, \boldsymbol{\phi}),$$

with $\boldsymbol{\phi} : \Omega \times [0, T] \to \mathbb{R}^m$ being the control input that may depend on both space and time. Moreover, the initial conditions are given by $\mathcal{I}(\boldsymbol{u}, \nabla_{\boldsymbol{x}} \boldsymbol{u}, \ldots)$ and the boundary conditions by $\mathcal{B}(\boldsymbol{u}, \nabla_{\boldsymbol{x}} \boldsymbol{u}, \ldots)$.

To draw the connection to RL, one can introduce a partial discretization in time with a constant step size $\Delta\tau$ and a zero-order hold on the control:

$$\mathcal{T}(\boldsymbol{u}_\tau, \boldsymbol{\phi}_\tau) = \boldsymbol{u}_\tau + \int_{t_\tau}^{t_{\tau+1}} \mathcal{N}(\boldsymbol{u}(\cdot, t), \boldsymbol{\phi}_\tau) \, \mathrm{d}t = \boldsymbol{u}_{\tau+1}.$$

Using the above considerations, the control task can be formalized as an optimal control problem of the following form:

$$\begin{aligned} \min_{\boldsymbol{\phi}} J(\boldsymbol{u}, \boldsymbol{\phi}) &= \min_{\boldsymbol{\phi}} \sum_{\tau=0}^{p} \ell(\boldsymbol{u}_\tau, \boldsymbol{\phi}_\tau) \qquad (1) \\ \text{s.t.} \quad \boldsymbol{u}_{\tau+1} &= \mathcal{T}(\boldsymbol{u}_\tau, \boldsymbol{\phi}_\tau), \qquad \tau = 0, 1, 2, \ldots, p-1, \end{aligned}$$

where J is the objective functional over the time horizon $T = p\Delta\tau$, and ℓ is the *stage cost*, e.g., a tracking term (with regularization, including penalties on the control cost)

$$\ell(\boldsymbol{u}_\tau, \boldsymbol{\phi}_\tau) = \left\| \boldsymbol{u}_\tau - \boldsymbol{u}_\tau^{\mathsf{ref}} \right\|_{L^2}^2 + \lambda \left\| \boldsymbol{\phi}_\tau \right\|_{L^2}^2 .$$

In terms of RL, the stage cost ℓ can be seen as the negative reward, the state $\boldsymbol{u}$ corresponds to $\boldsymbol{s}$ and $\boldsymbol{\phi}$ is linked to $\boldsymbol{a}$.

C. The Kuramoto-Sivashinsky equation

Due to its rich dynamical behavior, the Kuramoto-Sivashinsky equation is one of the most frequently studied PDE systems in many situations:

$$\frac{\partial \boldsymbol{u}}{\partial t} = -\nabla_{\boldsymbol{x}}^2 \boldsymbol{u} - \nabla_{\boldsymbol{x}}^4 \boldsymbol{u} - \frac{1}{2} \boldsymbol{u} \nabla_{\boldsymbol{x}} \boldsymbol{u} + \boldsymbol{\phi} \, , \qquad (2)$$

where $\boldsymbol{u}$ is the velocity and $\boldsymbol{\phi}$ is an additive forcing term. Similar to other works [16], [17], we study a one-dimensional spatial domain $\Omega = [0, L]$ with $L = 22$ and periodic boundary conditions such that the system exhibits chaotic behavior.

Again following related work [16]–[18], the control consists of a superposition of several Gaussians

$$\phi_{x,\tau\Delta\tau} = \sum_{i=1}^{4} \frac{\boldsymbol{a}_\tau[i]}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x}^{(i)})^2}{2\sigma^2} \right), \qquad (3)$$

located at spatial coordinates $\boldsymbol{x}^{(i)} \in \{0, L/4, 2L/4, 3L/4\}$. Here $\boldsymbol{a}_\tau[i] \in [-1, 1]$ is the i-th control output of the agent at time step τ and $\sigma = 0.4$. The applied control values $\boldsymbol{a}_\tau[i]$ are changed at intervals of $\Delta\tau = 0.25$ time units. Each episode simulates $T_{\max} = 100$ time units of the system, that is, 400 discrete steps, beginning with states sampled from the unforced attractor as an initial condition.

In accordance with [17], [18], our goal is to dampen the dissipation D of the solution variable while minimizing the amount of energy P spent to power the system as well as the actuation devices:

$$D = \langle \left[\nabla_{\boldsymbol{x}}^2 \boldsymbol{u}\right]^2 \rangle \quad \text{and} \quad P = \langle \left[\nabla_{\boldsymbol{x}} \boldsymbol{u}\right]^2 \rangle + \langle \boldsymbol{u}\boldsymbol{\phi} \rangle , \qquad (4)$$

where $\langle \cdot \rangle$ denotes the spatial average taken over the physical domain Ω. The reward is thus

$$r_\tau = -\int_{\tau\Delta\tau}^{\tau\Delta\tau+\Delta\tau} D(t) + P(t)\ dt\ . \qquad (5)$$

III. Learning surrogate models of forced PDEs

Before integrating our surrogate into the online learning process of model-based RL, we evaluate its design on an offline dataset. In order to do so, we discuss the different modeling steps in detail and assess their importance in an ablation study on the Kuramoto-Sivashinsky system.

For learning surrogates, the spatial extent of the physical domain Ω and the dimensionality of its discretization affects the amount of data necessary in training as well as the degree to which the surrogate matches the state evolution of the system. To achieve a dimensionality reduction, we use a *convolutional autoencoder (CAE)* architecture, consisting of an encoder and a decoder network, as illustrated by the yellow and dark blue blocks in Fig. 2. The encoder network $f_{\boldsymbol{\theta}_{\text{enc}}}$ compresses snapshots $\boldsymbol{s}_\tau$ of the system to a compact latent space $\boldsymbol{h}_\tau = f_{\boldsymbol{\theta}_{\text{enc}}}(\boldsymbol{s}_\tau)$, while the decoder network $f_{\boldsymbol{\theta}_{\text{dec}}}$ attempts to recover the original input $\hat{\boldsymbol{s}}_\tau = f_{\boldsymbol{\theta}_{\text{dec}}}(\boldsymbol{h}_\tau)$. The information bottleneck between the encoder and decoder networks regulates the degree to which the state dimensionality is reduced. More details on the specifics of the network architecture can be found in the long version of this paper, which can be found on arXiv [19].

For time stepping, we use a convolutional Long-Short Term Memory (LSTM) network model [20] (cf. [19] for more details). Similar to other works [21]–[23], the decoder network $f_{\boldsymbol{\theta}_{\text{dec}}}$ is tied into a network learning temporal residuals, i.e., it is used to predict state changes $\Delta\boldsymbol{s}_\tau = \boldsymbol{s}_{\tau+1} - \boldsymbol{s}_\tau$ of solution variables:

$$\hat{\boldsymbol{u}}_{\,\tau\Delta\tau+\Delta\tau} = \boldsymbol{u}_{\,\tau\Delta\tau} + \Delta\tau \cdot f_{\boldsymbol{\theta}_{\text{dec}}}(\boldsymbol{o}_\tau),$$

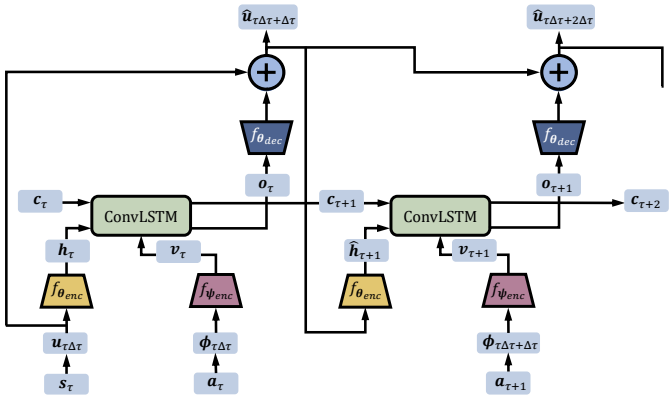

Fig. 2. Our surrogate model is composed of a state encoder $f_{\boldsymbol{\theta}_{\text{enc}}}$ (yellow), an action encoder $f_{\boldsymbol{\psi}_{\text{enc}}}$ (red) and a convolutional LSTM cell $f_{\boldsymbol{\theta}_{\text{fwd}}}$ (green) modeling the transition dynamics, as well as a decoder network $f_{\boldsymbol{\theta}_{\text{dec}}}$ (blue) that restores the spatial extent of its output to the physical domain. Starting off an initial condition (left), the model predicts the state evolution using its prior output as an input of the state variable (right). The figure does not show intermediate scaling or normalization transformations for clarity.

where $\boldsymbol{o}_\tau = f_{\boldsymbol{\theta}_{\text{fwd}}}(\boldsymbol{h}_\tau, \boldsymbol{v}_\tau; \boldsymbol{c}_\tau)$, and targets $\Delta\boldsymbol{s}_\tau/\Delta\tau \approx f_{\boldsymbol{\theta}_{\text{dec}}}(\boldsymbol{o}_\tau)$ are used to train the transition model $f_{\boldsymbol{\theta}_{\text{fwd}}}$ via end-to-end backpropagation. The variable $\boldsymbol{c}_\tau$ is the cell state of our recurrent transition model. The overall network architecture is illustrated in Fig. 2. It shows that, in each step, our model (i) transforms control outputs $\boldsymbol{a}$ to external forcing terms $\boldsymbol{\phi}$ using a known functional, (ii) encodes solution variables $\boldsymbol{u}$ and forcing terms $\boldsymbol{\phi}$ to a latent space defined on a shared spatial domain with encoder neural networks (yellow and red components), (iii) uses a transition model (green component) to infer encoded state changes $\boldsymbol{o}$, and (iv) decodes temporal changes $\boldsymbol{o}$ (blue component) before adding them to the original input. In order to use our model as a surrogate of the system, it is unrolled in time, taking the control output of an agent as well as its prior prediction for an input.

A notable benefit of learning temporal residuals is that the network's predictions obey the initial conditions of the system by construction. In fact, modeling state changes scaled with coefficient $\Delta\tau$ imposes a temporal smoothness assumption on the predictions to thwart irregular and erratic changes in the state evolution. In the context of model-based control, our temporal smoothness assumption offers a significant advantage. Since the control loop intertwines data collection, model learning, and behavior improvement, only a limited number of system snapshots are available to the surrogate at first. As we confirmed in our experiments, models predicting states in place of state changes often suffer from compounding model errors in the small data regime due to their dependence on well-functioning encoder and decoder networks.

IV. Learning to control PDEs sample efficiently

As a member of the *Dyna* family of model-based algorithms, our approach implements a model-free agent to learn behavior $\pi_{\boldsymbol{\omega}}$ and improve it over time. In essence, all descendants of Dyna follow instructions similar to those outlined in Alg. 1. In a broad sense, the algorithm intertwines data collection, model

learning, and behavior improvement for its main steps. Each iteration begins with rolling out $\pi_{\boldsymbol{\omega}}$ to collect additional samples that are later stored as a dataset $\mathcal{D}_{\text{env}}$. After training a deep neural network $f_{\boldsymbol{\theta}}$ to learn the dynamics of the environment on samples stored in $\mathcal{D}_{\text{env}}$, the model serves as an approximate MDP $\mathcal{M}'$ of the actual system $\mathcal{M}$. Supplementary samples from model-based rollouts of $\pi_{\boldsymbol{\omega}}$ are then stored in a separate dataset $\mathcal{D}_{\text{model}}$ as proxies for the actual experience. Iterations of our algorithm conclude with an update of the model-free agent. Here, data stored in $\mathcal{D}_{\text{env}}$ as well as $\mathcal{D}_{\text{model}}$ is used, and the latter amount often exceeds the other by multiple orders of magnitude.

Algorithm 1 Model-Based Reinforcement Learning

1: Initialize parameters of policy $\pi_{\boldsymbol{\omega}}$ and model $f_{\boldsymbol{\theta}}$
2: Initialize empty datasets $\mathcal{D}_{\text{env}}$ and $\mathcal{D}_{\text{model}}$
3: **while** not done **do**
4: Sample $\mathcal{M}$ using policy $\pi_{\boldsymbol{\omega}}$ $\rightarrow$ add to $\mathcal{D}_{\text{env}}$
5: Train model $f_{\boldsymbol{\theta}}$ on dataset $\mathcal{D}_{\text{env}}$
6: Sample $\mathcal{M}'$ using policy $\pi_{\boldsymbol{\omega}}$ $\rightarrow$ add to $\mathcal{D}_{\text{model}}$
7: Update $\pi_{\boldsymbol{\omega}}$ using samples of $\mathcal{D}_{\text{model}}$ and $\mathcal{D}_{\text{env}}$
8: **end while**

A. The model-based RL framework

A concrete implementation of the model-based principle is, therefore, all about (i) the design of a surrogate and its training process, (ii) the way in which $\mathcal{D}_{\text{model}}$ is populated with artificial samples, and (iii) the model-free agent updating $\pi_{\boldsymbol{\omega}}$ with data stored in $\mathcal{D}_{\text{env}}$ and fictitious samples in $\mathcal{D}_{\text{model}}$. In the following, we outline our approach to each of the above steps for model-based RL. A more detailed discussion of the individual steps can be found in the preprint [19].

1) Model learning: Unlike related work on model-based flow control, we do not collect snapshots of the system in advance using random exploration. Instead, we alternate between data collection and model learning to align the distribution of states that the agent visits with the data our surrogate is trained on. Our experiments (Section V-B) suggest that an online adaptation of the model to changes in the behavior is essential for accuracy. After each step of data collection, we train the model $f_{\boldsymbol{\theta}}$ using the samples collected in $\mathcal{D}_{\text{env}}$. To mitigate overfitting in the small data regime, we monitor the model on a validation set $\mathcal{D}_{\text{val}}$ and stop training early once the validation loss $\mathcal{L}_{\text{val}}$ converges.

2) Model-Based Rollouts: Consistent with standard practice [24], [25], we mitigate the risk of model exploitation using an ensemble $\{f_{\boldsymbol{\theta}_1}, \dots, f_{\boldsymbol{\theta}_{L_{\text{ens}}}}\}$ of dynamics models. The models deviate not only by their initial weights and the ordering of mini-batches, but also in terms of the data used for training and validation. Using our ensemble of dynamics models $\{f_{\boldsymbol{\theta}_1}, \dots, f_{\boldsymbol{\theta}_{L_{\text{ens}}}}\}$, we define an approximate MDP $\mathcal{M}'$ imitating the original system $\mathcal{M}$. Analogous to recent works on Dyna algorithms, model-based rollouts in our implementation branch off arbitrary system states sampled from dataset $\mathcal{D}_{\text{env}}$. Unlike rollouts beginning at initial conditions of the decision-making process, branching rollouts off arbitrary starting states avoids compounding model errors for states visited during later stages of episodes. In place of sampling single starting states, we select state-action sequences $(\boldsymbol{s}_{\tau-K_{\text{tf}}}, \boldsymbol{a}_{\tau-K_{\text{tf}}}, \dots, \boldsymbol{s}_{\tau}, \boldsymbol{a}_{\tau}) \sim \mathcal{D}_{\text{env}}$ of length K_{tf}. Sampling sequences enables us to warm-start the memory unit of our recurrent transition model $f_{\boldsymbol{\theta}}$. Each state-action pair of the sequence is then processed in *teacher-forcing* mode (see [26] for a definition) to guide the transitions. We match the sequence length K_{tf} to the number of teacher-forcing steps during training. In our experiments, using K_{tf} state transitions to seed the cell state was essential since we did not backpropagate gradients to its initial state at training time. Vice versa, we do not learn the initial condition of the memory unit since model-based rollouts start in different initial state, which otherwise destabilized the algorithm in our experiments. In order to implement a similar mechanism for the first K_{tf} steps of an episode, we use a simple padding method and repeat the initial state-action tuple to extend the sequence to length K_{tf}.

3) Policy optimization: As our approach belongs to the Dyna family, arbitrary model-free agents can be used to derive behavior given artificial samples. We here use the soft-actor critic [15] method, which is an off-policy algorithm such that data from previous iterations may be reused, even if the policy has changed. Our work builds on code made openly available in [27] and [28]. The soft-actor critic agent implements double Q-learning [29] to learn estimates of state-action values with separate networks $Q_{\boldsymbol{\theta}}$ and $Q_{\boldsymbol{\theta}'}$ to mitigate selection biases and uses delayed target networks $Q_{\boldsymbol{\theta}^-}$ to stabilize the optimization towards a moving target.

V. Experimental Evaluation

Corresponding to Sections III and IV, we first evaluate the capabilities of our surrogate to approximate the state evolution when the data is collected in advance, before considering the control aspect. More details on the training of our model can be found in the preprint [19].

A. Data-driven prediction of forced PDEs

To evaluate the amount of data our surrogate needs to approximate the state evolution, we collect a dataset of 100 simulation episodes ($40,000$ snapshots) with uniformly sampled actions. We split the samples into training, validation (80% / 20%), and testing episodes. All results are the aggregate of cross-validation scores with five folds. We study the evolution of state variable $\boldsymbol{u}$ for subsequences with 30 prediction steps (7.5 time units) during training and testing.

The small data regime is critical to our work, which is why we systematically decrease the amount of data throughout cross-validation from 100% down to 10%. To isolate the contribution of single design decisions for our dynamics model, we introduce several ablations and compare our results with those of a simple baseline, namely (i) the usage of fully connected (FC) networks instead of CNNs and (ii) the prediction of the state instead of a residual update. We train each model for a maximum of 250 epochs or terminate the

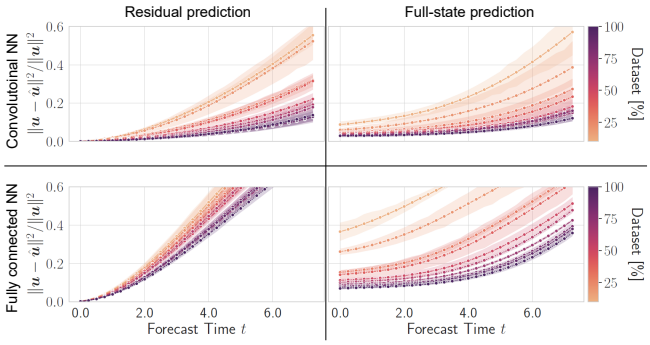

Fig. 3. Prediction error of the Kuramoto-Sivashinsky equation. Each datapoint averages the error for different testing subsequences. The models are trained on an increasing share of the dataset composed of $40,000$ snapshots. Solid lines depict the mean of five folds, while the shaded regions show the 95% confidence interval.

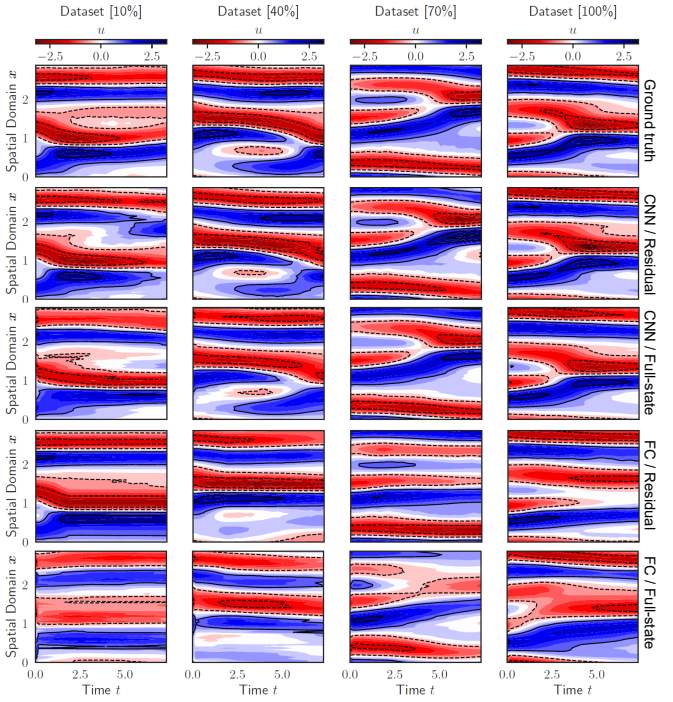

Fig. 4. Comparison of the temporal state evolution of the Kuramoto-Sivashinsky equation with forecasts of our model and the different ablations.

optimization once the monitored loss does not decrease for $P_{\text{val}} = 25$ validation epochs.

In Fig. 3, we show the normalized mean squared state prediction error, averaged over various initial conditions. Unsurprisingly, convolutional models (top row) prove far more effective than fully connected networks (bottom row) in both the small as well as large data regime. In terms of the comparison between residual and full-state prediction (left vs. right column), the former yields a notable advantage for forecasts over short time durations with error margins being negligible for several time units. In the small data regime, the full-state prediction is not yet equipped to recover state variables and, therefore, suffers from large error margins. Since our RL approach will limit model usage after a small number of prediction steps, the inferior accuracy after ~ 3.25 time units (13 discrete steps) is less relevant for the optimization. The errors in the predicted rewards (which have a significant impact on the learning performance) are very similar to the state prediction. Exemplary trajectories are shown in Fig. 4.

In general, increasing the degrees of freedom of a model enables it to learn the state evolution of snapshots in greater detail. At the same time, models of higher capacity are at a greater risk of overfitting to patterns in the training set.

B. Reinforcement learning control of PDEs

In this section, we study whether (i) learning a surrogate of the global dynamics indeed mitigates the data consumption of model-free approaches, (ii) learned control strategies can match their effectiveness, although model-based RL algorithms introduce approximation errors, (iii) examine the contribution of single components to our overall approach in isolation, and (iv) evaluate our algorithm for a number of configurations.

1) Data efficiency of learned control laws: We compare episode returns from our approach to those of the popular model-free algorithms PPO [13] and SAC [15]. These are most often encountered in the fluid mechanics literature nowadays, despite the agent discarding past experiences after each update. Since our approach integrates SAC as a main component for policy improvement, it is the candidate best suited to compare against. In both cases, we use the open-source implementations made available by the *stable-baselines3* project [30] as well as their default configuration for our experiments.

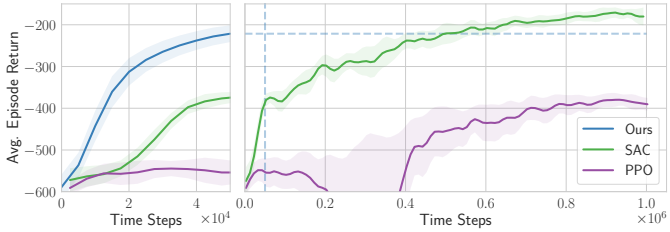

Fig. 5. Average evaluation episode returns for our approach compared to SAC and PPO for up to $50,000$ (left) and $1,000,000$ (right) steps of training. Solid lines show the mean of three trials, while shaded regions denote the standard deviation among trials. All lines are smoothed using a Gaussian filter with $\sigma = 1$. The dotted lines on the right-hand side show the average performance of our model-based algorithm after termination at $50,000$ time steps.

Fig. 5 shows the average return of evaluation episodes with respect to the number of samples collected for training thus far. Due to computational costs, we terminate the execution of our model-based approach once it obtained $50,000$ samples. For comparison, we execute the baselines for $1,000,000$ samples. On the left-hand side of the figure, we show the training curves up to the point where we terminate our algorithm, while the right-hand side shows the asymptotic behavior of SAC and PPO. We find that our model-based approach indeed learns suitable control strategies using substantially fewer samples. On average, SAC takes about $430,000$ samples to break even

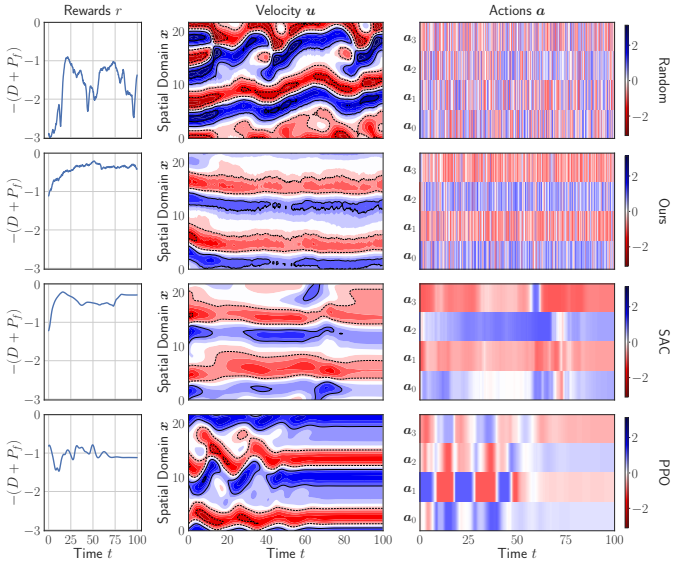

Fig. 6. The spatio-temporal evolution of the Kuramoto-Sivashinsky system (middle) and its dissipative term D and power consumption P (left) as a result of the actions (right). In each row, we show an agent starting from an initial condition sampled at random. The figure shows control strategies after their training terminated after $50,000$ or $1,000,000$ steps, respectively.

with the average episode returns of our approach at $50,000$ samples, while PPO cannot match its performance even after $1,000,000$ steps. With an almost nine-fold improvement in terms of data consumption over SAC, this demonstrates that model-based RL is indeed a promising direction to learn control strategies for systems governed by PDEs. Even though we stopped training early after $50,000$ steps, the control strategy dampened the dissipation D and power consumption P of the system by more than 63% of the values accomplished with random forcing. At the same time, SAC converges to a value of about 73%, although using 20 times the amount of data to do so. An example of what actuations the controller applies to navigate the system towards a stable state after termination is illustrated in Fig. 6.

2) Ablation study: A part of the motivation for our work are the shortcomings of past studies on model-based control for governed systems insofar as recent advances in the RL literature are not properly taken into account. To this end, our work examines the following ablations of our model-based approach.

- *Offline model training ablation:* Similar to [17], we train an RL agent on a surrogate model based on 50,000 snapshots collected in advance using random exploration.
- *Model exploitation ablation:* Similar to [31] and [17], this ablation neither uses ensembling nor rollout scheduling to diversify or truncate rollouts in an adaptative manner.
- *Surrogate ablation:* In place of our residual surrogate, we use the full-state prediction model to approximate the temporal state evolution.

Fig. 7 compares our implementation of the model-based methodology to its ablations. It shows that an improper treatment of compounding model errors often leads to unstable behavior optimization (purple line). Out of all methods we have tried for learning control strategies in our experiments, we found the *exploitation ablation* to be the least stable, as characterized by its wide standard deviation. Implementing model ensembling and using a schedule to determine the length of model-based rollouts is seemingly essential to prevent the agent from taking advantage of model errors in the small data regime. The figure also shows that a similar issue pertains to our *surrogate ablation* (green line). Although convergence is more stable overall, its approximate MDP cannot faithfully match the temporal evolution of the PDE in the small data regime. Our results also suggest that random exploration does not cover all relevant regions of the state space that policies visit during their optimization process (red line). After an initial phase of improvement, the behavior of our *offline ablation* diverges more and more from the state distribution that the model used for training, increasing prediction errors and destabilizing the optimization.

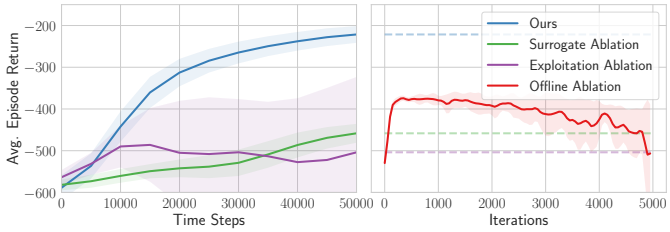

Fig. 7. Average evaluation episode returns for our model-based approach compared to different ablations. Solid lines show the mean of three trials, while shaded regions denote the standard deviation among trials. All lines are smoothed using a Gaussian filter with $\sigma = 1$. The dotted lines on the right show the average performance of the algorithms on the left after $50,000$ time steps. Since our *offline ablation* does not collect additional samples, we show its performance throughout iterations of the optimization.

VI. Discussion and Outlook

We have seen that carefully constructed surrogate models are capable of increasing the sample efficiency by roughly one order of magnitude. Furthermore, we found that an overall more sophisticated approach to model-based reinforcement learning (online model adaptation, ensembling, curriculum learning, etc.) is beneficial to stabilize the convergence, and we hope that our findings showcase the appeal of adopting best practices. However, due to the increased complexity, the performance of model-based RL is still inferior in situations where the model is comparatively easy to simulate. For future work, it will thus be highly interesting to see whether simpler yet more efficient surrogate models can be utilized. At this point, suitable candidates appear to be GRUs instead of LSTMs (see, e.g., [32]), as well as the Koopman operator [33]–[35], as it allows us to learn linear models of nonlinear systems, which is very efficient both in terms of the required training data and the run time. Finally, it might be worth looking into recent prediction error results for these methods [36]–[38] and see whether they can be transferred into guarantees for the RL process. In addition, one can

try to exploit system knowledge (in particular symmetries / invariances) in order to get smaller agents and thus to reduce the number of parameters that have to be trained [39], [40].

CODE AVAILABILITY

The source code of the conducted experiments can be obtained freely under https://github.com/stwerner97/pdecontrol.

ACKNOWLEDGMENT

S.P. acknowledges financial support by the project "SAIL: SustAInable Life-cycle of Intelligent Socio-Technical Systems" (Grant ID NW21-059D), which is funded by the program "Netzwerke 2021" of the Ministry of Culture and Science of the State of Northrhine Westphalia, Germany.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[3] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi, "Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control," *Journal of fluid mechanics*, vol. 865, pp. 281–302, 2019.

[4] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv:1611.01578*, 2016.

[7] R. S. Sutton, "Dyna, an integrated architecture for learning, planning, and reacting," *ACM Sigart Bulletin*, vol. 2, no. 4, pp. 160–163, 1991.

[8] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba, "Benchmarking model-based reinforcement learning," *arXiv:1907.02057*, 2019.

[9] C. G. Atkeson and J. C. Santamaria, "A comparison of direct and model-based reinforcement learning," in *Proceedings of international conference on robotics and automation*, vol. 4. IEEE, 1997, pp. 3557–3564.

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

[11] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv:1912.01603*, 2019.

[12] H. W. Andersen and M. Kümmel, "Evaluating estimation of gain directionality: Part 1: Methodology," *Journal of Process Control*, vol. 2, no. 2, pp. 59–66, 1992.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv:1509.02971*, 2015.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[16] M. A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin, "Control of chaotic systems by deep reinforcement learning," *Proceedings of the Royal Society A*, vol. 475, no. 2231, p. 20190351, 2019.

[17] K. Zeng, A. J. Linot, and M. D. Graham, "Data-driven control of spatiotemporal chaos with reduced-order neural ode-based models and reinforcement learning," *arXiv:2205.00579*, 2022.

[18] K. Zeng and M. D. Graham, "Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics," *Physical Review E*, vol. 104, no. 1, p. 014210, 2021.

[19] S. Werner and S. Peitz, "Learning a model is paramount for sample efficiency in reinforcement learning control of PDEs," *arXiv:2302.07160*, 2023.

[20] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[21] S. Mo, N. Zabaras, X. Shi, and J. Wu, "Deep autoregressive neural networks for high-dimensional inverse problems in groundwater contaminant source identification," *Water Resources Research*, vol. 55, no. 5, pp. 3856–3881, 2019.

[22] N. Geneva and N. Zabaras, "Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks," *Journal of Computational Physics*, vol. 403, p. 109056, 2020.

[23] P. Ren, C. Rao, Y. Liu, J.-X. Wang, and H. Sun, "Phycrnet: Physics-informed convolutional-recurrent network for solving spatiotemporal pdes," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114399, 2022.

[24] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," *arXiv:1802.10592*, 2018.

[25] K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," *Advances in neural information processing systems*, vol. 31, 2018.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[27] P. Tandon, "pytorch_sac," https://github.com/pranz24/pytorch-soft-actor-critic, 2018.

[28] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "pytorch_sac_ae," https://github.com/denisyarats/pytorch_sac_ae, 2021.

[29] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[30] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[31] X.-Y. Liu and J.-X. Wang, "Physics-informed dyna-style model-based deep reinforcement learning for dynamic control," *Proceedings of the Royal Society A*, vol. 477, no. 2255, p. 20210618, 2021.

[32] S. Pandey, P. Teutsch, P. Mäder, and J. Schumacher, "Direct data-driven forecast of local turbulent heat flux in rayleigh–bénard convection," *Physics of Fluids*, vol. 34, no. 4, p. 045106, 2022.

[33] J. L. Proctor, S. L. Brunton, and J. N. Kutz, "Dynamic mode decomposition with control," *SIAM Journal on Applied Dynamical Systems*, vol. 15, no. 1, pp. 142–161, 2016.

[34] M. Korda and I. Mezić, "Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control," *Automatica*, vol. 93, pp. 149–160, 2018.

[35] S. Peitz and S. Klus, "Koopman operator-based model reduction for switched-system control of pdes," *Automatica*, vol. 106, pp. 184–191, 2019.

[36] F. Nüske, S. Peitz, F. Philipp, M. Schaller, and K. Worthmann, "Finite-data error bounds for Koopman-based prediction and control," *Journal of Nonlinear Science*, vol. 33, p. 14, 2023.

[37] S. Peitz and K. Bieker, "On the Universal Transformation of Data-Driven Models to Control Systems," *Automatica*, vol. 149, p. 110840, 2023.

[38] C. Zhang and E. Zuazua, "A quantitative analysis of Koopman operator methods for system identification and predictions," *Comptes Rendus. Mécanique*, 2023.

[39] S. Peitz, J. Stenner, V. Chidananda, O. Wallscheid, S. L. Brunton, and K. Taira, "Distributed Control of Partial Differential Equations Using Convolutional Reinforcement Learning," *Physica D: Nonlinear Phenomena*, vol. 461, p. 134096, 2024.

[40] L. Guastoni, J. Rabault, P. Schlatter, H. Azizpour, and R. Vinuesa, "Deep reinforcement learning for turbulent drag reduction in channel flows," *The European Physical Journal. E, Soft Matter*, vol. 46, 2023.