Computers and Chemical Engineering 201 (2025) 109240

Contents lists available at ScienceDirect

Computers and Chemical Engineering

journal homepage: www.elsevier.com/locate/cace

Sample-efficient reinforcement learning of Koopman eNMPC

Daniel Mayfrank [a,b], Mehmet Velioglu [a,b], Alexander Mitsos [c,a,d], Manuel Dahmen [a,*]

[a] Forschungszentrum Jülich GmbH, Institute of Climate and Energy Systems, Energy Systems Engineering (ICE-1), Jülich 52425, Germany
[b] RWTH Aachen University, Aachen 52062, Germany
[c] JARA-ENERGY, Jülich 52425, Germany
[d] RWTH Aachen University, Process Systems Engineering (AVT.SVT), Aachen 52074, Germany

ARTICLE INFO

Keywords:
Economic model predictive control
Koopman
Model-based reinforcement learning
Policy optimization
Process control
Physics-informed neural networks
Demand response

ABSTRACT

Reinforcement learning (RL) can be used to tune data-driven (economic) nonlinear model predictive controllers ((e)NMPCs) for optimal performance in a specific control task by optimizing the dynamic model or parameters in the policy's objective function or constraints, such as state bounds. However, the sample efficiency of RL is crucial, and to improve it, we combine a model-based RL algorithm with our published method that turns Koopman (e)NMPCs into automatically differentiable policies. We apply our approach to an eNMPC case study of a continuous stirred-tank reactor (CSTR) model from the literature. The approach outperforms benchmark methods, i.e., data-driven eNMPCs using models based on system identification without further RL tuning of the resulting policy, and neural network controllers trained with model-based RL, by achieving superior control performance and higher sample efficiency. Furthermore, utilizing partial prior knowledge about the system dynamics via physics-informed learning further increases sample efficiency.

1. Introduction

Model predictive control (MPC) and its variants, e.g., economic nonlinear MPC (eNMPC), rely on dynamic models that (i) are accurate and (ii) lead to optimal control problems (OCPs) which are solvable in real-time. In process systems engineering, obtaining mechanistic models that fulfill these requirements can be difficult due to large scales, unknown parameters, and nonlinearities (Tang and Daoutidis, 2022). As an alternative to mechanistic models, data-driven approaches can be employed. These approaches are typically based on system identification (SI) using either historical operating data or data generated with a mechanistic model of the physical system (Tang and Daoutidis, 2022).

Alternatively, using reinforcement learning (RL) methods, data-driven (eN)MPCs can be trained for optimal performance in specific control tasks (see Fig. 1(a)), which may produce superior control performance compared to SI (see, e.g., Chen et al. (2019), Gros and Zanon (2019), Mayfrank et al. (2024b) and Mayfrank et al. (2024a)). To this end, a differentiable (e)NMPC policy is constructed, wherein the learnable parameters can be parameters of the dynamic model (see Fig. 1(b), e.g., Mayfrank et al. (2024b) and Mayfrank et al. (2024a)), or other parameters that appear in the objective function or constraints of the (e)NMPC (Gros and Zanon, 2019; Brandner et al., 2023; Brandner and Lucia, 2024), e.g., the state bounds (see Fig. 1(c)). The latter approach optimizes the policy by compensating for model errors, e.g., via bounds adaptation. Thereby, RL-based training/refinement of a highly parameterized dynamic model such as an artificial neural network can be avoided by optimizing few (interpretable) parameters, e.g., parameters that modify the state bounds. Therefore, this approach may lead to better convergence, compared to RL-based (re)training of the dynamic model itself. However, even positing good training convergence, the model-free[1] RL algorithms that have so far been used for RL-based training of (eN)MPCs remain notoriously sample inefficient, essentially rendering them inapplicable to domains where interacting with the physical environment (sampling) is expensive (Gopaluni et al., 2020), e.g., in industrial chemical process control applications.

Model-based RL (MBRL) algorithms are designed to increase the sample efficiency of RL by concurrently learning a policy and a model

[*] Corresponding author.
E-mail address: m.dahmen@fz-juelich.de (M. Dahmen).

[1] Due to the multitude of ways in which models can be used in RL, the distinction between model-free RL, model-based RL, and other control approaches is not consistent across the literature. This distinction might be especially confusing in the context of the present work since we study the training of eNMPC policies, i.e., inherently model-based policies. However, the distinction between model-free and model-based, which is most relevant to our work, is about whether the *training algorithm* uses an additional learned model of the environment to train the policy. Please refer to Section 2.1 for the exact definitions that we use in this work.

https://doi.org/10.1016/j.compchemeng.2025.109240
Received 25 March 2025; Received in revised form 14 May 2025; Accepted 9 June 2025
Available online 21 June 2025

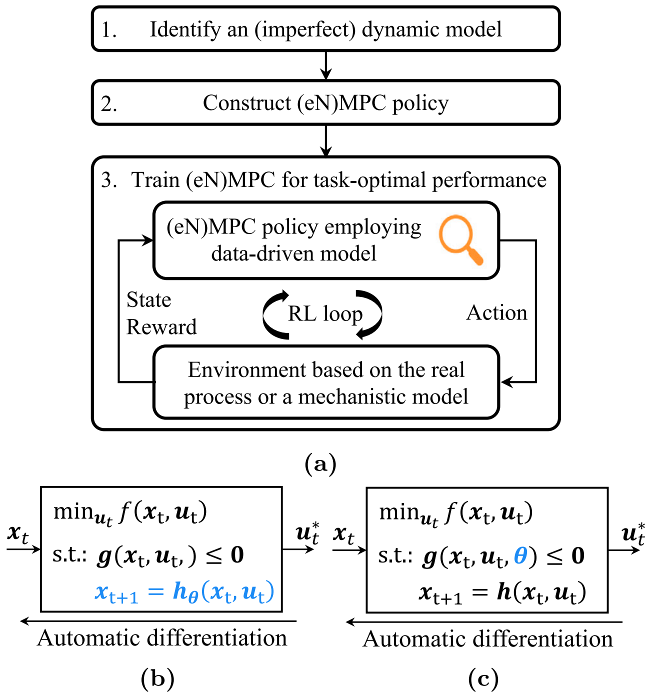

1. Identify an (imperfect) dynamic model
2. Construct (eN)MPC policy
3. Train (eN)MPC for task-optimal performance
(eN)MPC policy employing data-driven model
State
Reward
RL loop
Action
Environment based on the real process or a mechanistic model
(a)
$\min_{u_t} f(x_t, u_t)$
s.t.: $g(x_t, u_t,) \leq 0$
$x_{t+1} = h_\theta(x_t, u_t)$
x_t
u_t^*
Automatic differentiation
(b)
$\min_{u_t} f(x_t, u_t)$
s.t.: $g(x_t, u_t, \theta) \leq 0$
$x_{t+1} = h(x_t, u_t)$
x_t
u_t^*
Automatic differentiation
(c)

Fig. 1. (a) Procedure for RL-based training of an (eN)MPC. (b) A differentiable eNMPC policy parameterized by the parameters θ of the dynamic model; takes as input the current state x_t and computes the optimal control action u_t^* based on the minimization of a cost function f, subject to inequality constraints g, and the learnable dynamic model h_θ. (c) Differentiable eNMPC policy with parameterized inequality constraints g, e.g., state bounds. Training this policy leaves the underlying dynamic model h unchanged but adapts the inequality constraints to counteract model-plant mismatch.

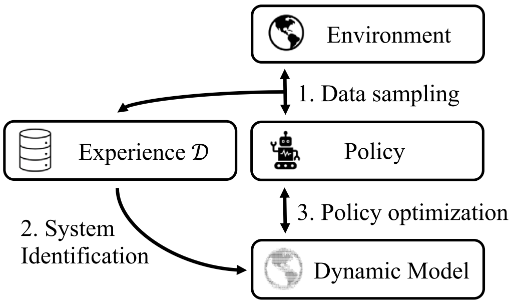


Fig. 2. Dyna-style (Sutton, 1991) model-based RL framework. The three steps are repeated for a predefined number of steps, or until satisfactory control performance is reached.

of the environment, i.e., a function that allows the prediction of future states and rewards from state–action pairs. The seminal Dyna algorithm (Sutton, 1991) iterates between three steps (see Fig. 2): (i) The current policy interacts with the real environment to gather data about the system dynamics. (ii) Using the acquired data, a data-driven dynamic model of the environment is learned via SI. (iii) The learned model is used to optimize the policy via any suitable RL algorithm, e.g., Proximal Policy Optimization (PPO) (Schulman et al., 2017) or Soft Actor-Critic (Haarnoja et al., 2018). Numerous "Dyna-style algorithms", i.e., algorithms that follow this basic framework, have been developed over the years. However, in this framework, the policy can learn to exploit the errors of the dynamic model, leading to overly-optimistic simulated results and corresponding policy failures in the real world (Kurutach et al., 2018). However, recent contributions (Kurutach et al., 2018; Clavera et al., 2018; Janner et al., 2019) have showcased ways to counteract this problem based on learning ensembles of dynamic models, leading to algorithms that can match the asymptotic control performance of model-free RL on certain problems while requiring orders of magnitude fewer interactions with the real environment. Ultimately, to maximize sample efficiency, any Dyna-style algorithm relies on learning reasonably well-generalizing dynamic models of the environment with as little data as possible. Therefore, another intuitive way to improve the performance of Dyna-style algorithms is to incorporate prior knowledge of the dynamics of the environment into the models, e.g., using physics-informed learning. Multiple contributions (e.g., Liu and Wang (2021) and Ramesh and Ravindran (2023)) have shown that using physics-informed neural networks (PINNs) (Raissi et al., 2019) in Dyna-style algorithms can increase both the sample efficiency and the performance of the resulting policies.

The adoption of RL algorithms into the process control community is still in its infancy and has largely been limited to model-free RL (Faria et al., 2022; Dogru et al., 2024). Gopaluni et al. (2020) describe model-free RL algorithms as insufficiently data-efficient for industrial process control applications. They identify the unification of model-based and model-free methods as a research area with tremendous potential to redefine automation in the process industry. Ponse et al. (2024) provide a comprehensive review on the applications of RL in the field of sustainable energy systems control, which is adjacent to process control and rate model-based methods as "underexplored". Contributions that did leverage model-based RL methods for process control or energy systems control used neural network policies that map directly from states to actions (e.g., Zhang et al. (2021), Gao and Wang (2023) and Faridi et al. (2024)). On the other hand, numerous contributions have showcased the potential benefits of RL-based training of (eN)MPCs for control compared to SI (e.g., Chen et al. (2019), Gros and Zanon (2019) and Mayfrank et al. (2024b,a)). However, these contributions used model-free RL algorithms. Due to the substantial benefits of model-based RL algorithms compared to model-free variants regarding sample efficiency, it is tempting to examine whether model-based RL can accelerate RL-based (eN)MPC training, thus making it potentially feasible in a wider range of applications. Such an analysis has not been conducted thus far.

In our previous publication (Mayfrank et al., 2024b), we introduced a method for RL training of (e)NMPCs that utilize data-driven Koopman *surrogate* models. That method rests on the availability of a simulated RL environment based on an accurate mechanistic system model. When training in a simulated environment, sample efficiency is less critical than in many potential real-world RL setting, where the agent has to learn from (costly) interactions with the physical system. In the current work, we extend the applicability of RL-based training of Koopman (e)NMPCs to settings where sample efficiency is critical by combining our previously published approach with the Model-Based Policy Optimization (MBPO) (Janner et al., 2019) algorithm. Moreover, we further increase the sample efficiency by modifying MBPO to utilize partial prior knowledge of the dynamics of the controlled system through physics-informed learning. To our knowledge, this work is the first to connect RL-based training of (eN)MPCs for task-optimal performance in specific control tasks to Dyna-style model-based RL.

In this work, we choose the MBPO algorithm (Janner et al., 2019), since it is a state-of-the-art Dyna-style RL algorithm. Many model-based RL algorithms exist and it is impossible to know a priori which algorithm will perform best in a specific task (Wang et al., 2019). While MBPO is one of the most promising algorithms available, our approach should be compatible with any Dyna-style algorithm that does not require a specific policy architecture.

We test our proposed method on an eNMPC case study (Mayfrank et al., 2024b) based on a continuous stirred-tank reactor (CSTR) model from Flores-Tlacuahuac and Grossmann (2006). We assess the performance of our approach by comparing it to that of (i) Koopman eNMPCs trained iteratively via SI and (ii) neural network policies trained using (physics-informed) MBPO. We find that through the combination of iterative SI of the Koopman model that is utilized in the eNMPC and RL-based adaptation of the state bounds in the eNMPC via the MBPO algorithm, our method outperforms the benchmarks for this case study. Additionally, we find that physics-informed learning of the model ensemble that is used in MBPO offers benefits for sample efficiency and that it can prevent policy degradation during training. These findings confirm our expectation that model-based RL can be successfully integrated with training data-driven (eN)MPCs. Thus, our work is a step toward making RL-based training of predictive controllers feasible for complex real-world control problems where no simulator of the environment is available a priori and interactions with the real environment are expensive, making sample efficiency absolutely crucial. Considering the previously mentioned contributions showcasing the advantages of RL compared to pure SI when learning data-driven predictive controllers, our work, therefore, shows an avenue toward more capable and efficient predictive controllers.

The remainder of this paper is structured as follows: Section 2 provides the theoretical background to our work and presents our method. Section 3 presents the results of the numerical experiments that we conducted on a simulated case study. Section 4 draws final conclusions and discusses promising directions for future research.

2. Method

Section 2.1 introduces notation and definitions for policy optimization using RL and provides a brief explanation of the MBPO algorithm (Janner et al., 2019). Section 2.2 explains how we set up differentiable Koopman (e)NMPCs that are trainable using RL. Subsequently, we present our method for sample-efficient learning of task-optimal Koopman (e)NMPCs for control (Section 2.3).

2.1. Model-based policy optimization

RL is a framework for learning how to map situations to actions in order to maximize a numerical reward signal (Sutton and Barto, 2018). In contrast to supervised learning tasks where learning is based on labeled data sets, RL is based on sequential feedback from trial and error actuation of an *environment*. The environment is represented by a Markov Decision Process (MDP) with associated states $\boldsymbol{x}_t \in \mathbb{R}^n$, control inputs $\boldsymbol{u}_t \in \mathbb{R}^m$, a transition function $\boldsymbol{\mathcal{F}} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$,

$$\boldsymbol{x}_{t+1} = \boldsymbol{\mathcal{F}}(\boldsymbol{x}_t, \boldsymbol{u}_t), \tag{1}$$

and a scalar reward function $\mathcal{R} : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$,

$$r_{t+1} = \mathcal{R}(\boldsymbol{x}_{t+1}, \boldsymbol{u}_t). \tag{2}$$

The goal of RL is to maximize the (discounted) sum of expected future rewards. For applications with continuous action spaces, actor-critic RL methods (see, e.g., Fujimoto et al. (2018), Schulman et al. (2017) and Haarnoja et al. (2018)) that optimize parameterized policies $\boldsymbol{\pi}_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t) : \mathbb{R}^n \mapsto \mathbb{R}^m$ directly mapping from states to (probability distributions over) actions, are most suitable (Sutton and Barto, 2018).

The Model-Based Policy Optimization (MBPO) (Janner et al., 2019) algorithm is a state-of-the-art Dyna-style RL algorithm. As any other Dyna-style algorithm, it iterates between the following three steps (see Fig. 2): (i) collect experience in the real environment, (ii) learn a model of the environment, (iii) train the policy using the environment model and a suitable model-free RL algorithm. Building upon the work by Kurutach et al. (2018), MBPO aims to overcome the critical problem of model exploitation by learning an ensemble of models and choosing one of the models at random for each environment step when training the policy (Fig. 2, third step). Ideally, learning a model ensemble maintains an adequate level of uncertainty in the policy optimization step, thus preventing overfitting the policy to the errors of a specific model. However, model errors still compound over multiple simulated steps, causing problems in policy learning for long-horizon tasks. In MBPO, Janner et al. (2019) address this issue by introducing two modifications compared to the standard Dyna framework: (i) Instead of using simulated rollouts with l discrete time steps corresponding to the length of episodes in the real environment, they shorten the simulated rollouts to a length of $k \ll l$ steps. (ii) The initial state of each simulated rollout is determined by randomly sampling from the experience $\mathcal{D}$ (cf. Fig. 2) instead of sampling from the initial state distribution of the real environment. Thus, every state that was encountered by the policy in the real environment can serve as an initial state in a simulated rollout. These modifications disentangle the length of simulated rollouts during policy training from the episode length in the original task. Janner et al. (2019) showed that in multiple continuous control benchmark problems, MBPO vastly improves sample efficiency while producing policies of similar performance compared to state-of-the-art model-free RL algorithms (e.g., Schulman et al. (2017) and Haarnoja et al. (2018)).

2.2. Differentiable Koopman (e)NMPC

In Mayfrank et al. (2024b), we introduce a method for constructing automatically differentiable stochastic (e)NMPC policies $\boldsymbol{\pi}_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t) : \mathbb{R}^n \mapsto \mathbb{R}^m$ from Koopman models of the form proposed by Korda and Mezić (2018). Such models are of the form

$$\boldsymbol{z}_0 = \boldsymbol{\psi}_\theta(\boldsymbol{x}_0), \tag{3a}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{A}_\theta \boldsymbol{z}_t + \boldsymbol{B}_\theta \boldsymbol{u}_t, \tag{3b}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{C}_\theta \boldsymbol{z}_t, \tag{3c}$$

where $\boldsymbol{z}_t \in \mathbb{R}^N$ is the vector of Koopman states and $\hat{\boldsymbol{x}}_t \in \mathbb{R}^n$ is the model prediction of the system state at time step t. The model has the following components: $\boldsymbol{\psi}_\theta : \mathbb{R}^n \mapsto \mathbb{R}^N$, where typically $N \gg n$, defines the nonlinear state observation function that transforms the initial condition $\boldsymbol{x}_0$ into the Koopman space. $\boldsymbol{A}_\theta \in \mathbb{R}^{N\times N}$ and $\boldsymbol{B}_\theta \in \mathbb{R}^{N\times m}$ linearly advance the Koopman state vector forward in time. $\boldsymbol{C}_\theta \in \mathbb{R}^{n\times N}$ linearly maps a prediction of the Koopman state to a prediction of the system state.

Given a data set describing the dynamics of some system, such models can be trained via SI by minimizing the sum of three loss functions (Lusch et al., 2018; Mayfrank et al., 2024b). These loss terms correspond to the requirements that the Koopman model needs to fulfill: (i) reconstructing states passed through the autoencoder, (ii) predicting the evolution of the lifted Koopman state, and (iii) predicting the evolution of the system states. The associated loss terms are:

$$\|\boldsymbol{C}_\theta \boldsymbol{\psi}_\theta(\boldsymbol{x}_t) - \boldsymbol{x}_t\|, \tag{4a}$$

$$\|\boldsymbol{A}_\theta \boldsymbol{\psi}_\theta(\boldsymbol{x}_t) + \boldsymbol{B}_\theta \boldsymbol{u}_t - \boldsymbol{\psi}_\theta(\boldsymbol{x}_{t+1})\|, \tag{4b}$$

$$\|\boldsymbol{C}_\theta(\boldsymbol{A}_\theta \boldsymbol{\psi}_\theta(\boldsymbol{x}_t) + \boldsymbol{B}_\theta \boldsymbol{u}_t) - \boldsymbol{x}_{t+1}\| \tag{4c}$$

In Mayfrank et al. (2024b), we aim to optimize a Koopman model for optimal performance as part of an (e)NMPC in a specific control task. However, as noted in Section 1, in RL-based training of an (eN)MPC, it may be beneficial to keep an imperfect model unchanged and instead optimize a small number of parameters in the objective function or inequality constraints which compensate for model errors. To be able to differentiate between different kinds of learnable parameters of the Koopman-eNMPC policy, we therefore rename the parameters: In the following, $\boldsymbol{\theta}_\text{K}$ refers to the parameters of the Koopman model, which appear as $\boldsymbol{\theta}$ in Eq. (3). Additionally, we introduce the parameters $\boldsymbol{\theta}_\text{B}$, which modify the state bounds of the eNMPC. Both types of parameters influence the behavior of the policy, i.e., $\boldsymbol{\theta} = [\boldsymbol{\theta}_\text{K}^\intercal, \boldsymbol{\theta}_\text{B}^\intercal]^\intercal$ and $\boldsymbol{\pi}_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t) : \mathbb{R}^n \mapsto \mathbb{R}^m$.

Integrating the idea of state bound adaptation into the differentiable Koopman (e)NMPC framework (Mayfrank et al., 2024b) is straightforward. Given an (e)NMPC horizon of $t_f + 1$ steps with the corresponding sets $\text{T}_{+1} = \{t, \dots, t+t_f\}$ and $\text{T} = \{t, \dots, t+t_f-1\}$, a convex OCP is solved to obtain the optimal action $\boldsymbol{u}_t^*$:

$$\min_{(\boldsymbol{u}_t)_{t\in\text{T}}} \sum_{t\in\text{T}_{+1}} \Phi(\boldsymbol{C}_{\theta_\text{K}} \boldsymbol{z}_t, \boldsymbol{u}_t) + M \boldsymbol{s}_t^\intercal \boldsymbol{s}_t, \tag{5a}$$

$$\text{s.t. } \boldsymbol{z}_{t+1} = \boldsymbol{A}_{\theta_{\mathrm{K}}} \boldsymbol{z}_t + \boldsymbol{B}_{\theta_{\mathrm{K}}} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{5b}$$

$$g(\boldsymbol{C}_{\theta_{\mathrm{K}}} \boldsymbol{z}_t, \boldsymbol{u}_t, \boldsymbol{s}_t, \theta_{\mathrm{B}}) \leq \boldsymbol{0} \quad \forall t \in \mathrm{T}_{+1} \tag{5c}$$

Φ is a convex function representing the stage cost of the objective function, and g are convex inequality constraint functions that can also include bounds on control and state variables. In the latter case, slack variables $\boldsymbol{s}_t$ are added to the state bounds to ensure the feasibility of the OCPs. The use of slack variables is penalized quadratically with a penalty factor M. Using PyTorch (Paszke et al., 2019) and *cvxpylayers* (Agrawal et al., 2019), the output $\boldsymbol{u}_t$ of the policy is automatically differentiable with respect to $\boldsymbol{x}_t$, θ_{K}, and θ_{B}.

2.3. Physics-informed MBPO of Koopman models for control

This section describes our general framework for sample-efficient learning of task-optimal Koopman (e)NMPC policies. A more in-depth description of the implementation details of our method when applied to our specific case study is provided in Section 3.2.

Our method iterates between the three typical Dyna steps and is visualized in Fig. 3. First, the Koopman (e)NMPC policy interacts with the environment for a predefined number of steps to gather data $\mathcal{D}$ about the system dynamics. To ensure exploration, we sample the (otherwise deterministic) action $\boldsymbol{u}_t$ from a normal distribution $\boldsymbol{u}_t \sim \mathcal{N}(\boldsymbol{u}_t^*, \sigma^2)$. We assume that the reward function of the environment is known. Thus, rewards r_t do not need to be recorded in $\mathcal{D}$. In the very first iteration of the overall algorithm the Koopman model is still randomly initialized and the eNMPC outputs will, therefore, not be meaningful. Therefore, in the data sampling step of the first MBPO iteration, we randomly sample the actions $\boldsymbol{u}_t$ from a uniform distribution over the action space. Likewise, any other controller type, e.g., a PID controller, may be used in the first MBPO iteration, although the quality of the resulting training data will be influenced by how diverse the control actions produced by the controller are.

Second, an ensemble of n data-driven dynamic models, i.e., neural networks (NNs), is learned based on $\mathcal{D}$ via SI to approximate the dynamics of the environment (c.f. Eq. (1)). If (incomplete) physics knowledge is available, it is possible to train PINNs (Raissi et al., 2019). Throughout this work, each ensemble member $\boldsymbol{NN}_{i,\omega_i} \forall i \in \{1, 2, \dots, n\}$ is a PINN parameterized by $\boldsymbol{\omega}_i$. Furthermore, we fit the parameters θ_{K} of the Koopman model to $\mathcal{D}$ via SI. Note that we do not use a physics-informed training method for the Koopman model. Physics-informed training of the Koopman model would, in principle, also be possible. However, the representational capacity of the Koopman model is limited because it is used as part of the real-time eNMPC policy and physics-informed training would necessitate using some of that representational capacity to predict outputs that are not necessary for the eNMPC application, i.e., the *a priori* unknown physics terms (see Fig. 5). Since the behavior of the overall Koopman eNMPC policy is optimized with respect to the physics-informed NN ensemble anyway (see Fig. 3(a), step three), and to keep our method as simple as possible, we decided against a physics-informed system identification approach for the Koopman model. In fitting the PINN ensemble and the Koopman model, we follow standard SI practices, such as splitting $\mathcal{D}$ into a training data set and a validation data set used for early stopping and normalizing the inputs and outputs of the models. We refer the reader to Mayfrank et al. (2024b) for a detailed description of how Koopman models in the form proposed by Korda and Mezić (2018) (Eq. (3)) can be trained using system identification.

The third step is based upon our previously published method (Mayfrank et al., 2024b) on viewing Koopman (e)NMPC policies as automatically differentiable policies (Fig. 3(b)): Using the ensemble in conjunction with the known reward function as a simulator of the environment, we train the Koopman (e)NMPC for task-optimal performance by optimizing the parameters θ_{B} via the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017). During policy optimization using PPO, one of the n PINNs is chosen randomly for each step in the simulated environment. The critic is a feedforward neural network parameterized by $\boldsymbol{\phi}$. We use the approach described by Kurutach et al. (2018) to determine when to stop the policy optimization and return to the first step (data sampling): In regular intervals, the performance of the policy is evaluated separately on all n learned models. Once the ratio of models under which the policy improves drops below a certain threshold for too many consecutive PPO iterations, we terminate the policy optimization and return to the data sampling step. A more detailed description of the termination criterion for policy optimization is given in Section 3.2.4. The overall learning process can continue for a predefined number of steps in the real environment or until a satisfactory performance is achieved.

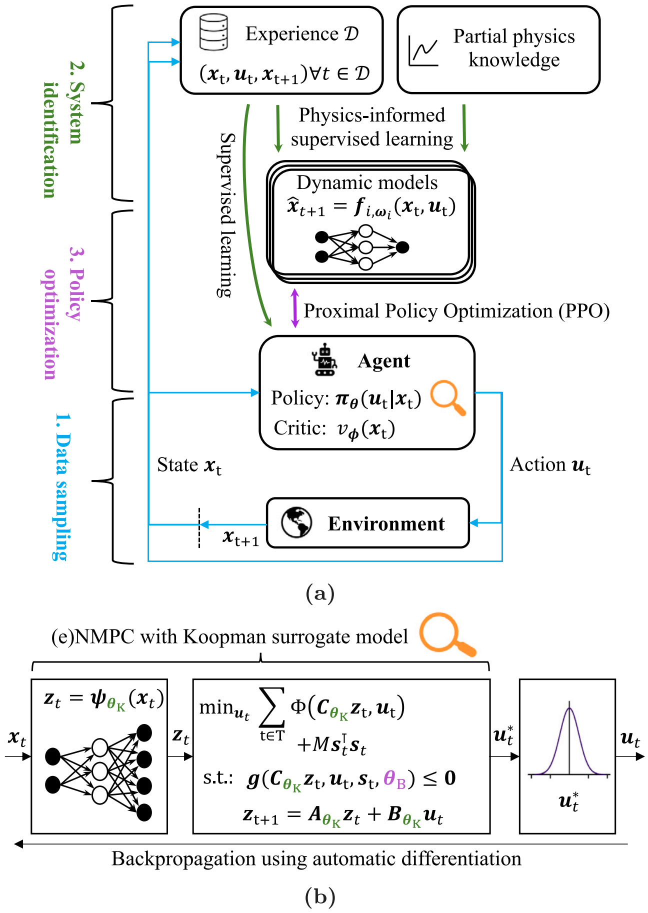

2. System identification
3. Policy optimization
1. Data sampling
Experience $\mathcal{D}$
$(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{x}_{t+1}) \forall t \in \mathcal{D}$
Partial physics knowledge
Physics-informed supervised learning
Supervised learning
Dynamic models
$\hat{\boldsymbol{x}}_{t+1} = \boldsymbol{f}_{i,\omega_i}(\boldsymbol{x}_t, \boldsymbol{u}_t)$
Proximal Policy Optimization (PPO)
Agent
Policy: $\pi_\theta(\boldsymbol{u}_t|\boldsymbol{x}_t)$
Critic: $v_\phi(\boldsymbol{x}_t)$
State $\boldsymbol{x}_t$
Action $\boldsymbol{u}_t$
Environment
$\boldsymbol{x}_{t+1}$
(a)
(e)NMPC with Koopman surrogate model
$\boldsymbol{x}_t$
$\boldsymbol{z}_t = \boldsymbol{\psi}_{\theta_{\mathrm{K}}}(\boldsymbol{x}_t)$
$\boldsymbol{z}_t$
$\min_{\boldsymbol{u}_t} \sum_{t \in \mathrm{T}} \Phi(\boldsymbol{C}_{\theta_{\mathrm{K}}} \boldsymbol{z}_t, \boldsymbol{u}_t) + M \boldsymbol{s}_t^{\intercal} \boldsymbol{s}_t$
s.t.: $\boldsymbol{g}(\boldsymbol{C}_{\theta_{\mathrm{K}}} \boldsymbol{z}_t, \boldsymbol{u}_t, \boldsymbol{s}_t, \theta_{\mathrm{B}}) \leq \boldsymbol{0}$
$\boldsymbol{z}_{t+1} = \boldsymbol{A}_{\theta_{\mathrm{K}}} \boldsymbol{z}_t + \boldsymbol{B}_{\theta_{\mathrm{K}}} \boldsymbol{u}_t$
$\boldsymbol{u}_t^*$
$\boldsymbol{u}_t^*$
$\boldsymbol{u}_t$
Backpropagation using automatic differentiation
(b)

Fig. 3. Using MBPO to train a task-optimal Koopman (e)NMPC controller. (a) The training algorithm. The following three steps are executed in a loop until a stopping criterion is reached: First, the Koopman (e)NMPC interacts with the environment to gather data about the dynamics. Second, all data collected up to the current step is used to fit the Koopman model (parameters θ_{K}) and the PINN ensemble (parameters $\omega_i \forall i \in \{1, 2, \dots, n\}$). Third, a surrogate RL environment is constructed using the NN ensemble and the Koopman (e)NMPC is optimized by tuning the parameters θ_{B}, i.e., the state bounds. (b) The automatically differentiable Koopman (e)NMPC whose behavior is defined by the parameters of the Koopman model (θ_{K}) and the parameters modifying the state bounds (θ_{B}). The parameters are color-coded to match the colors of the corresponding optimization steps in Fig. 3(a).

We emphasize that the behavior of the resulting (e)NMPC is defined by θ_{K} and θ_{B} and that both parameter types are optimized in each iteration of the overall algorithm (Fig. 3): In the SI step, θ_{K} is trained to maximize the agreement of the Koopman model to $\mathcal{D}$, i.e., all data available at the time. Then, the policy optimization step optimizes θ_{B}, i.e., the state bounds, using PPO and simulated interactions with the PINN ensemble. Please note that the policy optimization step does not necessarily result in a tightening of the state bounds: Depending on (i) the disagreement between the Koopman model and the PINN

Table 1
CSTR model parameters (Flores-Tlacuahuac and Grossmann, 2006; Du et al., 2015). All parameters except the reaction constant k are dimensionless.

	Symbol	Value
Volume	V	20
Eeaction constant	k	$300\,\frac{1}{\mathrm{h}}$
Activation energy	N	5
Feed temperature	T_f	0.3947
Heat transfer coefficient	α_c	$1.95 \cdot 10^{-4}$
Coolant temperature	T_c	0.3816

Table 2
Lower (lb) and upper (ub) bounds of system states and control inputs and steady-state (ss) values used to evaluate the economic benefit of flexible production in eNMPC.

Variable	lb	ub	ss
c	0.1231	0.1504	0.1367
T	0.6	0.8	0.7293
ρ	$0.8\,\frac{1}{h}$	$1.2\,\frac{1}{h}$	$1.0\,\frac{1}{h}$
F	$0.0\,\frac{1}{h}$	$700.0\,\frac{1}{h}$	$390.0\,\frac{1}{h}$

ensemble, and (ii) the reward function specified by the user, it may lead to tightened or relaxed bounds. Therefore, if a sensible reward function is specified, the policy optimization step should not be detrimental to the economic performance of the overall policy.

Intuitive justification for our approach: The policy optimization step optimizes the Koopman (e)NMPC policy using a surrogate RL environment. This environment is based on an ensemble of PINNs learned via SI using the same data as the Koopman model. In the following, we provide three rationales for why such an approach offers benefits compared to simply learning a Koopman model via SI and using it inside an (e)NMPC policy without further RL-based optimization of the policy: (i) The PINN ensemble captures the epistemic uncertainty of the learned dynamics (Janner et al., 2019), i.e., the uncertainty that arises from insufficient amounts of data. Through RL, the (e)NMPC is forced to behave in a way that is robust with respect to every ensemble member. Without RL, a single learned Koopman model would define the (e)NMPC behavior, which increases the risk of policy failures due to epistemic uncertainty. (ii) The representational capacity of the Koopman model that is utilized in the (e)NMPC is limited since the resulting OCPs (Eq. (5)) have to be solved in real-time. Thus, given sufficiently complex system dynamics, the Koopman model might not be able to accurately capture the system dynamics everywhere. In contrast, the representational capacity of the PINN ensemble members has a much higher limit since the ensemble models are not used in online optimization. RL tuning of θ_B can, therefore, help to compensate for the limitations of (e)NMPC that stem from a limited representational capacity of the utilized Koopman model. (iii) Using RL, the (e)NMPC can be tuned to the requirements of a specific control problem. Specifically, by weighting different parts of the reward function of the simulated RL environment, RL offers a way to tune the behavior of the policy, e.g., to prioritize cost savings or constraint satisfaction.

3. Numerical experiments

3.1. Case study description

We demonstrate our method on a demand response case study (Mayfrank et al., 2024b) based on a benchmark continuous stirred-tank reactor (CSTR) model (Flores-Tlacuahuac and Grossmann, 2006; Du et al., 2015). The following case study description is based on (Mayfrank et al., 2024b), where a more detailed explanation can be found. The states $\boldsymbol{x}$ of the model are the dimensionless product concentration c and the dimensionless reactor temperature T. The control inputs $\boldsymbol{u}$ are the production rate $\rho\ \left[\frac{1}{\mathrm{h}}\right]$ and the coolant flow rate $F\ \left[\frac{1}{\mathrm{h}}\right]$. Two nonlinear ordinary differential equations define the dynamics of the system:

$$\dot{c}(t) = (1 - c(t))\frac{\rho(t)}{V} - c(t)ke^{-\frac{N}{T(t)}}, \tag{6a}$$

$$\dot{T}(t) = (T_f - T(t))\frac{\rho(t)}{V} + c(t)ke^{-\frac{N}{T(t)}} - F(t)\alpha_c(T(t) - T_c) \tag{6b}$$

Table 1 lists all parameters appearing in Eq. (6).

For the training of the PINN ensemble in the SI step (see Fig. 3(a)), we assume that the concentration and temperature changes due to inlet/outlet flows and cooling are known. However, we assume that the constitutive expression for the reaction in Eqs. (6) is unknown. Thus, the physics equations for the PINN are

$$\dot{c}(t) = (1 - c(t))\frac{\rho(t)}{V} - R(t), \tag{7a}$$

$$\dot{T}(t) = (T_f - T(t))\frac{\rho(t)}{V} + R(t) - F(t)\alpha_c(T(t) - T_c), \tag{7b}$$

with the *unknown* and unmeasured reaction rate $R(t) = c(t)ke^{-\frac{N}{T(t)}}$.

The goal is to minimize production costs. To enable flexible operation taking advantage of electricity price fluctuations, we assume the existence of a product storage with filling level l and a maximum capacity of six hours of steady-state production. Given electricity price predictions, the controller aims to minimize production costs while ensuring that a steady product demand is met and adhering to bounds imposed on the system states. Production costs can be influenced by altering the process cooling as the electric power consumption is assumed to be proportional to the coolant flow rate F. Table 2 presents the state bounds and the steady-state values of the model (see Eq. (6)). Matching the hourly structure of the day-ahead electricity market, we choose control steps of length $\Delta t_\mathrm{ctrl} = 1$ h. We use historic day-ahead electricity prices from the Austrian market (Open Power System Data, 2020). During training, we use the prices from March 29, 2015 to March 25, 2018. For the final evaluation of the trained policies, we use the prices from March 26, 2018 to September 30, 2018.

3.2. Implementation details

This subsection explains the implementation details of our method (Section 2.3) when it is applied to this case study. Section 3.2.1 presents our architecture choices for the agent and the dynamic models. Thereafter, three subsections address the iterative three-step approach of our method, i.e., data sampling (Section 3.2.2), system identification (Section 3.2.3), and policy optimization (Section 3.2.4). Section 3.2.5 briefly describes alternative methods for learning a controller, e.g., learning a neural network policy via (physics-informed) MBPO, which we use to rate the performance of our proposed method. All training code including the hyperparameters that were used to obtain the results presented in Section 3.3 is available online.[2]

3.2.1. Model architecture

As in Mayfrank et al. (2024b), we choose a latent space dimensionality of eight for the Koopman model that is part of the eNMPC policy. Since the CSTR model has two states and two control inputs (see Eq. (6)), this means that $\boldsymbol{A}_{\theta_\mathrm{K}} \in \mathbb{R}^{8\times 8}, \boldsymbol{B}_{\theta_\mathrm{K}} \in \mathbb{R}^{8\times 2}, \boldsymbol{C}_{\theta_\mathrm{K}} \in \mathbb{R}^{2\times 8}$ (see Eq. (3)). The encoder $\boldsymbol{\psi}_{\theta_\mathrm{K}} : \mathbb{R}^2 \mapsto \mathbb{R}^8$ is a multilayer perceptron (MLP) with two hidden layers (four and six neurons, respectively) and hyperbolic tangent activation functions.

RL-based training of (e)NMPC controllers requires solving and differentiating through many OCP instances. Since the number of variables in an OCP grows linearly with the number of time steps, RL training of (e)NMPCs is computationally challenging given long prediction horizons. To balance control performance and computational

[2] https://jugit.fz-juelich.de/iek-10/public/optimization/pi-mbpo4koopmanenmpc.

tractability, we determine an effective eNMPC prediction horizon by repeatedly solving eNMPC problems using the mechanistic CSTR model while varying the prediction horizon. We select a horizon t_f of nine hours, as longer horizons do not produce substantial performance gains in the mechanistic eNMPC. Thus, $\mathrm{T}_{+1} = \{t, \dots, t+9\}$ and $\mathrm{T} = \{t, \dots, t+8\}$ (see Eq. (5)). Analogous to our earlier work on model-free RL of Koopman eNMPCs (Mayfrank et al., 2024b), given a prediction for the evolution of the electricity prices $\boldsymbol{p}_{\text{eNMPC}} = (p_t)_{t \in \mathrm{T}_{+1}}$, the policy aims to minimize the production cost while satisfying the bounds of the states and the product storage. To that end it first calculates the initial latent state $\boldsymbol{z}_0$ by passing the initial state $\boldsymbol{x}_0 = (c_0, T_0)^\top$ through the encoder, i.e., $\boldsymbol{z}_0 = \boldsymbol{\psi}_{\theta_\mathrm{K}}(\boldsymbol{x}_0)$. Then, the following OCP is solved:

$$\min_{(\rho_t, F_t)_{t\in\mathrm{T}}} \sum_{t\in\mathrm{T}_{+1}} (F_t p_t \Delta t_{\text{ctrl}} + M \boldsymbol{s}_t^\top \boldsymbol{s}_t), \tag{8a}$$

$$\text{s.t. } \boldsymbol{z}_{t+1} = \boldsymbol{A}_{\theta_\mathrm{K}} \boldsymbol{z}_t + \boldsymbol{B}_{\theta_\mathrm{K}} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{8b}$$

$$l_{t+1} = l_t + (\rho_t - \rho_{\text{ss}}) \Delta t_{\text{ctrl}} \quad \forall t \in \mathrm{T}, \tag{8c}$$

$$\boldsymbol{x}_t = \boldsymbol{C}_{\theta_\mathrm{K}} \boldsymbol{z}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{8d}$$

$$\underline{\boldsymbol{x}}_t - \boldsymbol{s}_{\boldsymbol{x},t} + \boldsymbol{\theta}_{\mathrm{B},\underline{\boldsymbol{x}}} \le \boldsymbol{x}_t \le \bar{\boldsymbol{x}}_t + \boldsymbol{s}_{\boldsymbol{x},t} + \boldsymbol{\theta}_{\mathrm{B},\bar{\boldsymbol{x}}} \quad \forall t \in \mathrm{T}_{+1}, \tag{8e}$$

$$0 - s_{l,t} + \theta_{\mathrm{B},\underline{l}} \le l_t \le 6.0 + s_{l,t} + \theta_{\mathrm{B},\bar{l}} \quad \forall t \in \mathrm{T}_{+1}, \tag{8f}$$

$$\boldsymbol{s}_t = \begin{pmatrix} \boldsymbol{s}_{\boldsymbol{x},t} \\ s_{l,t} \end{pmatrix} \quad \forall t \in \mathrm{T}_{+1}, \tag{8g}$$

$$\boldsymbol{0} \le \boldsymbol{s}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{8h}$$

$$\underline{\boldsymbol{u}}_t \le \boldsymbol{u}_t \le \bar{\boldsymbol{u}}_t \quad \forall t \in \mathrm{T} \tag{8i}$$

Note that $\boldsymbol{\theta}_\mathrm{B}$ appears only in the constraints regarding the state bounds of the CSTR (Eq. (8e)) and the storage level (Eq. (8f)), i.e., $\boldsymbol{\theta}_\mathrm{B}$ merely serves to tighten or relax those bounds.

We use a NN with parameters $\boldsymbol{\phi}$ as the critic in the policy optimization step. The architecture of the critic is shown in Fig. 4. It has four separate input layers for (i) $\boldsymbol{x}_t$, (ii) l_t, (iii) a two-element vector that includes the electricity price at the current time step and the difference between the highest and the lowest electricity price in the current MPC prediction horizon, i.e., $\Delta(\boldsymbol{p}_{\text{eNMPC}}) = \max_{t\in\mathrm{T}_{+1}}(p_t) - \min_{t\in\mathrm{T}_{+1}}(p_t)$, and (iv) a vector of all electricity prices in the current MPC prediction horizon. Each of those input layers is followed by two equally sized hidden layers, with 24, 8, 8, and 24 neurons, respectively. The output of all second hidden layers is then concatenated and passed through two fully connected layers, each of size 64 neurons. The output layer has a single neuron for the value of the current state. Except the output layer, which does not have an activation function, all layers have hyperbolic tangent activation functions. We pick such an architecture since it yielded substantially better results than fully connected architectures of similar overall size in preliminary testing.

In the policy optimization step, we use an ensemble of $n = 10$ PINNs to model the dynamics of the real environment. Each PINN $\boldsymbol{NN}_{i,\boldsymbol{\omega}_i} \forall i \in \{1, 2, \dots, 10\}$ has five separate input features that are concatenated to a single input layer (i) the PINN time τ, (ii) two initial states c_0, T_0, and (iii) two control inputs ρ, F. Here, the PINN time is chosen to be $\tau \in [0, \Delta t_{\text{ctrl}}]$ such that the PINN can be trained with constant control inputs and the PINN time domain matches the size of a control step. The input layer is followed by two equal-sized hidden layers with 32 neurons each. The output layer consists of three output features: two for the differential states $\boldsymbol{x} = [c, T]^\top$ and one for the algebraic state $\boldsymbol{y} = [R]$. A schematic of the PINN architecture is shown in Fig. 5. Further details on the PINN modeling approach used throughout this paper can be found in Velioglu et al. (2025).

At the beginning of a training run, we initialize $\boldsymbol{\theta}_\mathrm{B}$ with zeros, i.e., our initial guess is that the bounds do not need to be adapted. The learnable parameters of the PINN models ($\boldsymbol{\omega}$) are initialized with the Xavier normal distribution (Glorot and Bengio, 2010) since we observed that it leads to better performance for PINNs in our preliminary studies. All other learnable parameters ($\boldsymbol{\theta}_\mathrm{K}, \boldsymbol{\phi}$) are initialized randomly using the default PyTorch (Paszke et al., 2019) parameter initialization method.

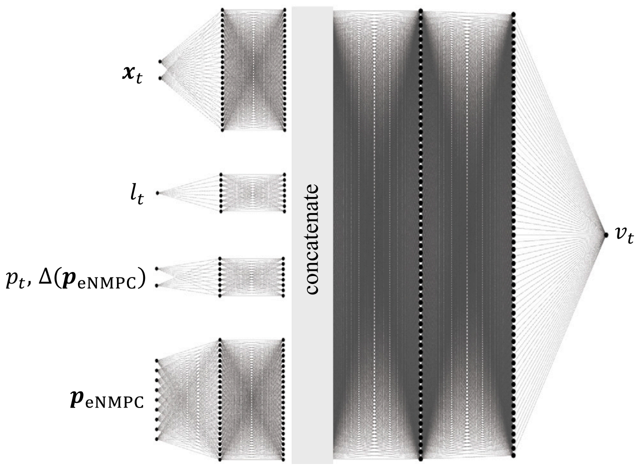

$\boldsymbol{x}_t$
l_t
p_t, $\Delta(\boldsymbol{p}_{\text{eNMPC}})$
$\boldsymbol{p}_{\text{eNMPC}}$
concatenate
v_t

Fig. 4. Critic architecture. As for the policy, the system states are scaled so that the feasible range is in [−1,1], whereas the product storage and the electricity prices are left unscaled.

For the purpose of data-driven modeling, we rescale the system states and control inputs (see Table 2) linearly so that the lower and upper bounds of each variable correspond to −1.0 and 1.0, respectively.

3.2.2. Data sampling

Each iteration of the MBPO algorithm (Janner et al., 2019) starts with the current policy interacting with the environment to gather data about the system dynamics (first step in Fig. 3(a)). This data is then added to the data set $\mathcal{D}$. We gradually increase the number of steps that the policy takes at each MBPO iteration: In the first 10 iterations, we let the policy take 20 steps in each iteration, i.e., until $\mathcal{D}$ contains 200 steps. Then, until up to 500 overall steps, we increase the number of steps taken at each iteration to 50. Finally, we increase this number to 250 steps per iteration until we reach an overall number of 2500 steps in $\mathcal{D}$. Here, we terminate each training run.

In each MBPO iteration, we start the data sampling step by resetting the environment, i.e., we set the system states to their steady-state values, we randomly initialize the storage filling level between one and two hours of steady-state production, and we sample a series of electricity prices for the current episode. An episode ends given one of two conditions: (i) the episode reaches its maximum number of 167 steps, i.e., one week of uninterrupted closed-loop operation, or (ii) a constraint violation occurs in one of the states, where the distance of the variable from the violated bound is bigger than the feasible interval of the associated state (see Table 2). Upon termination of an episode, the environment resets and a new episode starts. All state transitions are added to $\mathcal{D}$. Data sampling continues until the desired number of steps in the environment in the current MBPO iteration has been reached.

To enable early stopping in the SI step (see Section 3.2.3), we split $\mathcal{D}$ into a training and a validation data set, i.e., $\mathcal{D} = (\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}})$. In the first MBPO iteration, the data from the first episode is added to $\mathcal{D}_{\text{train}}$, and the data from the second episode to $\mathcal{D}_{\text{val}}$. Thereafter, at the start of each episode, we randomly determine whether the data of this episode will be added to $\mathcal{D}_{\text{train}}$ (with a chance of 75%) or $\mathcal{D}_{\text{val}}$ (25% chance).

In the data sampling step of the first MBPO iteration, the policy still has randomly initialized parameters $\boldsymbol{\theta}_\mathrm{K}$. Herein, we, therefore, randomly sample the actions $\boldsymbol{u}_t$ from a uniform distribution over the action space. In all subsequent MBPO iterations, we sample the action $\boldsymbol{u}_t$ from a normal distribution $\boldsymbol{u}_t \sim \mathcal{N}(\boldsymbol{u}_t^*, \boldsymbol{\sigma}^2)$, where $\boldsymbol{u}_t^*$ is the deterministic output of the policy. At the start of each new episode, $\boldsymbol{\sigma}$ is randomly sampled from a uniform distribution between 0.0 and 0.1.

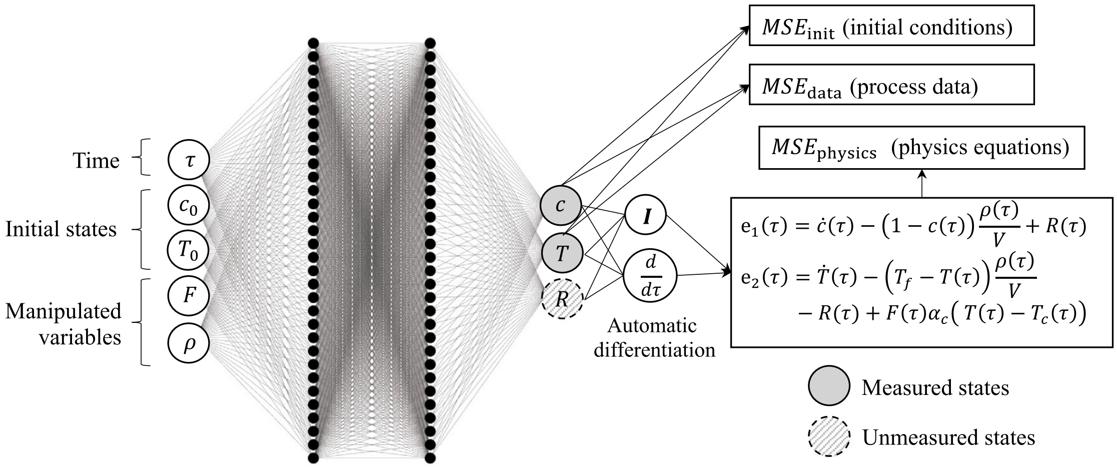

MSE_{init} (initial conditions)
MSE_{data} (process data)
MSE_{physics} (physics equations)
Time
Initial states
Manipulated variables
τ
c_0
T_0
F
ρ
c
T
R
I
$\frac{d}{d\tau}$
Automatic differentiation
$e_1(\tau) = \dot{c}(\tau) - (1 - c(\tau))\frac{\rho(\tau)}{V} + R(\tau)$
$e_2(\tau) = \dot{T}(\tau) - (T_f - T(\tau))\frac{\rho(\tau)}{V} - R(\tau) + F(\tau)\alpha_c(T(\tau) - T_c(\tau))$
Measured states
Unmeasured states

Fig. 5. General schematic of the PINN models used in the CSTR case study.

The PINN requires two additional data sets for training: (i) A physics data set $\mathcal{D}_{\text{physics}}$ is used to calculate the physics residuals. It contains unlabeled data, i.e., no output observations from the environment are needed. We sample $|\mathcal{D}_{\text{physics}}| = 2000$ collocation points for the PINN inputs using the lower and upper bounds specified in Table 2 for the initial states and controls, and $\tau \in [0, \Delta t_{\text{ctrl}} = 1\,\text{h}]$ for the PINN time. (ii) An initial state data set $\mathcal{D}_{\text{init}}$ is used to teach the PINN to match the initial state at $t = t_0$. Although this data set is labeled, the output states are identical to the initial states. Thus, no interaction with the environment is required to assemble $\mathcal{D}_{\text{init}}$. We sample $|\mathcal{D}_{\text{init}}| = 100$ data points for the state and control variables within the bounds specified in Table 2. Both $\mathcal{D}_{\text{physics}}$ and $\mathcal{D}_{\text{init}}$ are generated uniquely for each PINN model $\boldsymbol{NN}_{i,\boldsymbol{\omega}_i} \forall i \in \{1, 2, \ldots, 10\}$ in the ensemble but remain unchanged throughout the training. We use Latin Hypercube Sampling (LHS) (Iman et al., 1981) to ensure coverage of the PINN input domain.

3.2.3. System identification

In the SI step (second step in Fig. 3(a)), we fit the trainable parameters of the Koopman model ($\boldsymbol{\theta}_{\text{K}}$) and of the model ensemble ($\boldsymbol{\omega}_i \forall i \in \{1, 2, \ldots, 10\}$) to the data in $\mathcal{D}_{\text{train}}$.

For the Koopman model, we use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-4} and a mini-batch size of 64 samples. We choose a maximum number of 5000 epochs; however, we stop SI early if the sum of Eqs. (4a)–(4c) with respect to $\mathcal{D}_{\text{val}}$ does not reach a new minimum for 25 consecutive epochs.

The PINN models in the ensemble are trained by minimizing the following loss function, where n is the number of states:

$$MSE_{\text{total}} = MSE_{\text{physics}} + \lambda_1 MSE_{\text{data}} + \lambda_2 MSE_{\text{init}}, \tag{9a}$$

$$\text{with } MSE_{\text{data}} = \frac{1}{n|\mathcal{D}_{\text{train}}|} \sum_{j=1}^{|\mathcal{D}_{\text{train}}|} (\hat{\boldsymbol{x}}(\tau_j) - \boldsymbol{x}(\tau_j))^2, \tag{9b}$$

$$MSE_{\text{physics}} = \frac{1}{n|\mathcal{D}_{\text{physics}}|} \sum_{j=1}^{|\mathcal{D}_{\text{physics}}|} \left(\dot{\hat{\boldsymbol{x}}}(\tau_j) - \boldsymbol{f}(\hat{\boldsymbol{x}}(\tau_j), \hat{\boldsymbol{y}}(\tau_j), \boldsymbol{u}_j)\right)^2, \tag{9c}$$

$$MSE_{\text{init}} = \frac{1}{n|\mathcal{D}_{\text{init}}|} \sum_{j=1}^{|\mathcal{D}_{\text{init}}|} (\hat{\boldsymbol{x}}_j(0) - \boldsymbol{x}_j(0))^2 \tag{9d}$$

Here, MSE_{data} corresponds to the loss term for measurement data, MSE_{physics} corresponds to the physics regularization loss stemming from (incomplete) physics knowledge on system dynamics (c.f. Eq. (7)), and MSE_{init} corresponds to a loss term that ensures that the predictions at $\tau = 0$ are consistent with the initial states. λ_1 and λ_2 denote the weights of the measurement data and initial condition loss terms.

The PINN models are trained in a two-stage manner, similar to Velioglu et al. (2025). In the first stage, we use the Adam optimizer (Kingma and Ba, 2014) for 1000 epochs, with a learning rate of 10^{-3} and a mini-batch size of 64 samples. Here, we use inverse Dirichlet weighting (Maddu et al., 2022) to obtain the weights λ_1, λ_2 in Eq. (9a) dynamically. In the second stage, we use the LBFG-S optimizer (Liu and Nocedal, 1989) with a full batch for a maximum number of 300 epochs; however, we stop early if the loss in Eq. (9) with respect to $\mathcal{D}_{\text{val}}$ does not reach a new minimum for 25 consecutive epochs. Note that in the second stage, we fix the weights λ_1, λ_2 according to the last value attained in the first stage. At each MBPO iteration, the trainable parameters of the model ensemble ($\boldsymbol{\omega}_i \,\forall\, i \in \{1, 2, \ldots, 10\}$) have a 1/3 chance of resetting to prevent getting stuck in a sub-optimal local minimum over many consecutive iterations.

3.2.4. Policy optimization

The policy optimization step (third step in Fig. 3(a)) adjusts the parameters $\boldsymbol{\theta}_{\text{B}}$ toward task-optimal performance of the eNMPC policy, given the Koopman model that was identified through SI. Using the PINN ensemble, we construct a data-driven surrogate RL environment. In each step taken in this surrogate environment, one of the PINNs is chosen randomly as an approximation of the state transition function (Eq. (1)) by evaluating it at $\tau = \Delta t_{\text{ctrl}}$. As is typical practice in MBPO (Janner et al., 2019), we shorten episodes in the surrogate environment (to a maximum of eight steps) to prevent compounding prediction errors of the PINNs from negatively influencing the policy optimization. As in the data sampling step (see Section 3.2.2), we still terminate an episode earlier whenever an outsized constraint violation occurs. Whenever an episode ends and the surrogate environment resets, we randomly sample the state of the CSTR from all states in $\mathcal{D}_{\text{train}}$, thus decoupling the length of simulated episodes from the state values that can be reached during an episode.

To incentivize the desired controller behavior, we choose a reward that promotes cost savings compared to steady-state production while punishing constraint violations. The overall reward at each step (Eq. (2)) is calculated via

$$r_t = \alpha \cdot r_t^{\text{cost}} - r_t^{\text{con,rel}} - r_t^{\text{con,bool}} + 1. \tag{10}$$

Herein, r_t^{cost} incentivizes the minimization of the production costs by giving positive rewards if the costs are lower than those of a steady-state production regime:

$$r_t^{\text{cost}} = (F_{\text{ss}} - F_{t-1}) \cdot p_{t-1} \cdot \Delta t_{\text{ctrl}}$$

Constraint violations are penalized twofold: $r_t^{\text{con,rel}}$ penalizes constraint violations quadratically, i.e., $r_t^{\text{con,rel}} \geq 0$, and $r_t^{\text{con,rel}} = 0$ if no constraint violation occurs at t. $r_t^{\text{con,bool}}$ imposes an additional small constant penalty whenever a constraint violation occurs, irrespective of the magnitude of the violation, i.e., $r_t^{\text{con,bool}} = 0.1$ if there is a constraint violation, and $r_t^{\text{con,bool}} = 0$ otherwise. At every step, we add a constant reward of 1 to ensure that the sum of rewards of an episode keeps rising as long as the episode continues, i.e., we give a reward for not producing a constraint violation that is large enough to cause an environment reset. $\alpha = 5 \cdot 10^{-6}$ is a hyperparameter used to balance the influence of r_t^{cost} compared to all other components of the overall reward.

We use our previously published method for automatic differentiation of Koopman (e)NMPCs (Mayfrank et al., 2024b) in conjunction with the *Stable-Baselines3* (Raffin et al., 2021) implementation of the PPO algorithm (Schulman et al., 2017) for policy optimization. In each PPO iteration, we sample 2048 steps in the surrogate environment, and we set the batch size for the policy and critic updates to 256 samples. We use the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 10^{-3}, and we clip the gradient norms of the policy and the critic to a maximum of 0.5.

As explained in Section 2.3, we do not terminate the policy optimization step after a predefined number of PPO iterations. Instead, following the idea of Kurutach et al. (2018), we implement the following performance-based stopping criterion: After every five iterations of the PPO algorithm, we evaluate the performance of the policy separately on all 10 learned models. To this end, we set up 10 validation environments corresponding to the 10 learned models. In each of these environments, only the corresponding model is used to represent the transition function (Eq. (1)). Then, we compute the ratio of validation environments in which the policy improves by running the policy for five episodes in each environment. Specifically, we check the ratio of validation environments in which the policy has reached a new highest average reward in the last 25 PPO iterations. When this ratio falls below 70%, we terminate policy optimization. Then, the next MBPO iteration begins with the data sampling step, or the overall training process stops if we have already reached 2500 steps in the real environment.

3.2.5. Ablation and benchmark variants

To evaluate the performance of our proposed method and to analyze how different components of the method contribute to the overall performance, we compare the performance of the following controller types. To avoid unnecessary repetition, we focus our description on the differences compared to the main method, e.g., unless explicitly stated otherwise, all variants use MBPO (Janner et al., 2019).

1. $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ (main contribution): Our method, outlined in Section 2.3 with implementation details explained in Sections 3.2.1–3.2.4. The name refers to the iterative process of SI of a Koopman model, followed by physics-informed (PI) RL of the bounds in the eNMPC.
2. $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$: Here, we train the model ensemble without assuming any prior physics knowledge. Each model in this ensemble is a vanilla NN. We train the models by minimizing the discrete-time L_2 prediction loss, i.e., given $\mathcal{D}$, we minimize $\| \boldsymbol{NN}_{i,\omega_i}(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{x}_{t+1} \|$ for each $i \in \{1, 2, \ldots, 10\}$. The architecture of the models is identical to that of the PINN models (see Fig. 5), except that we remove the time τ from the inputs and the reaction rate R from the outputs.
3. SI_{Koop}: This variant can be thought of as adaptive Koopman eNMPC. Compared to $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$, we keep the data sampling step and the iterative SI of the Koopman model unchanged, but we omit any further (model-based) policy optimization.
4. PIRL_{MLP}: Here, we use a neural network policy in form of an MLP instead of a Koopman eNMPC. This policy has the same architecture as the critic described in Section 3.2.1 (Fig. 4), except that its output layer has a size of two, corresponding to the two-dimensional action space of the environment.
5. RL_{MLP}: Same as PIRL_{MLP} but without physics-informed model training.

3.3. Results

For each type of controller, we repeat the training ten times using different fixed seeds in every training run. We train each controller type for 2500 steps in the real environment as specified in Section 3.2. After every full MBPO iteration, we save the agent and the model ensemble for testing purposes.

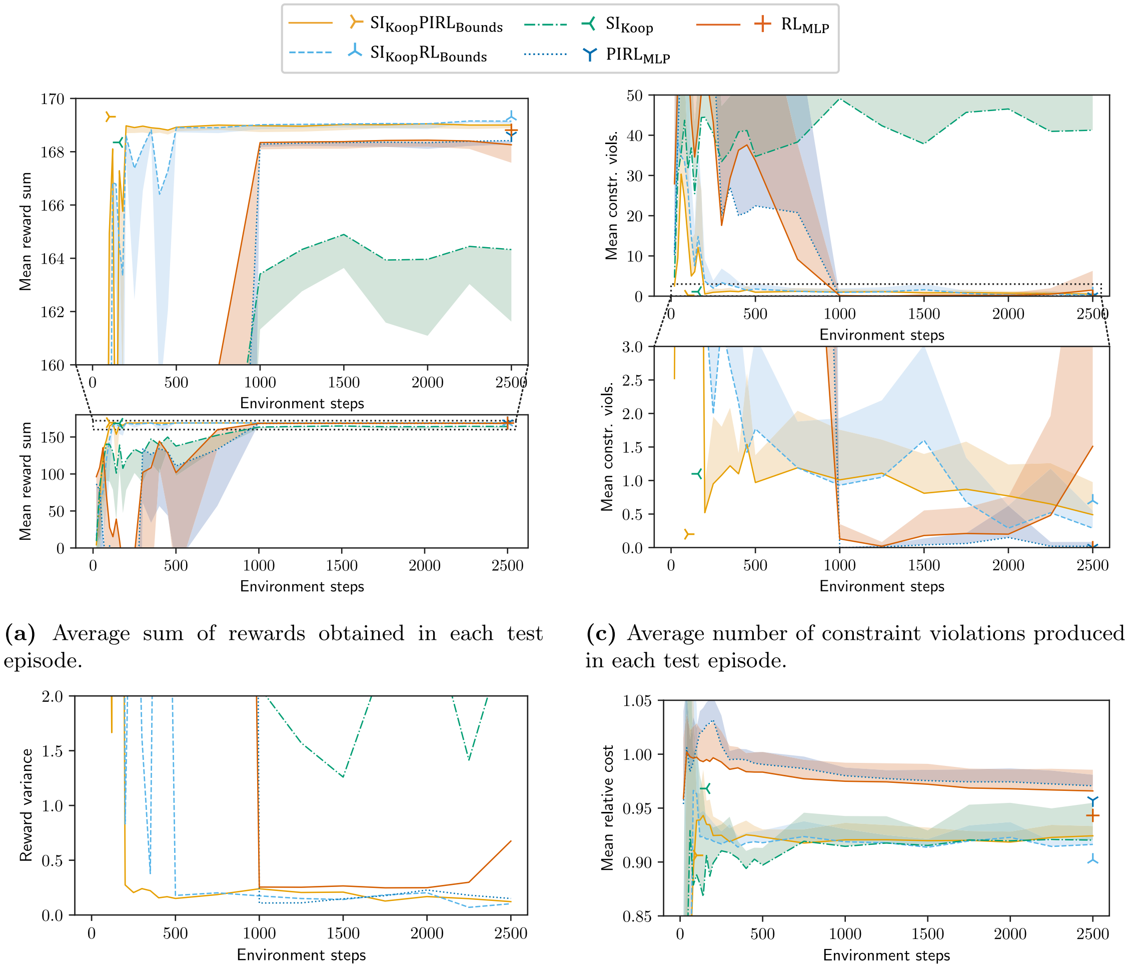
We test each controller by running it without exploration noise, i.e., $\boldsymbol{u}_t = \boldsymbol{u}_t^*$, for 10 one-week-long episodes (168 steps in each episode) using electricity price trajectories that were not used during training. Each test is performed using the same 10 electricity price trajectories, ensuring comparable results between the tests. The aggregated results of these tests can be viewed in Fig. 6. Fig. 7 shows some randomly chosen control trajectories which were part of these tests. Since our focus is on sample efficiency rather than final policy performance, we show control performance at different numbers of environment steps. We randomly select one training run for each of the depicted controller types and show its control performance in Fig. 7. Therein, we randomly select one of the 10 test electricity price trajectories and show the first 48 time steps of that test episode for each controller type.

To evaluate the performance of the controllers, we analyze the obtained rewards (since this is the metric that is maximized by MBPO), and the two metrics which we are primarily interested in and which together produce the reward (see Eq. (10)), i.e., the constraint violations and the economic performance. Throughout this section, we report economic performance via the economic cost incurred relative to the nominal production cost, i.e., we report the total cost incurred by the respective controller divided by the cost of steady-state production at nominal rate given the same electricity price trajectory. Fig. 6(a) shows that $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ and $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$(i) reach the highest reward values and that they do so with (ii) exceptional sample efficiency and (iii) low variance (see also Fig. 6(b)) across different training runs. PIRL_{MLP} and RL_{MLP} also achieve low performance variance across training runs, albeit with lower sample efficiency and at a lower performance level (when measured in average rewards) compared to $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ and $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$. SI_{Koop} performs worse, not converging to high and stable rewards within the given budget of 2500 environment steps.

A direct comparison of the physics-informed vs. the purely data-driven variants shows a small but consistent benefit resulting from the utilization of partial physics knowledge in training the model ensemble: $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ performs better than $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$ between 200 and 500 environment steps. Thereafter, both variants perform comparably well. The two MLP controller variants, PIRL_{MLP} and RL_{MLP}, show similar sample efficiency. Still, the PINN ensemble seems to benefit the stability of the learning once relatively high rewards have been reached.

Looking at Fig. 6(c), we can see that initially (up to around 1000 steps) $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ and $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$ are best at avoiding constraint violations. In particular, $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ quickly learns to avoid constraint violations. Both variants continue to make relatively steady progress in this regard up until the end of the training runs. The MLP controllers initially cause many constraint violations, however, after around 1000 environment steps they have learned to avoid constraint violations almost perfectly. Toward the end of the training, some of the RL_{MLP} controllers seem to loose this capability partially. SI_{Koop} struggles with constraint satisfaction across the full length of all training runs.

Fig. 6(d) shows that, when looking at the economic performance, there is a clear difference between the Koopman eNMPC-based controllers and the MLP controllers: After around 750 environment steps, the Koopman eNMPC-based controllers all produce average costs of around 91% to 93% of nominal production costs. After that, no big improvement happens. The MLP controllers incur substantially higher production costs (between 96% and 102% after 750 environment steps); however, they keep improving until the end of the training. Thus, it is possible that their economic performance would eventually



(a) Average sum of rewards obtained in each test episode.

(c) Average number of constraint violations produced in each test episode.

(b) Variance of the sum of rewards across differently seeded training runs.

(d) Average economic cost incurred in each test episode.

Fig. 6. Performance metrics for different controller types. Each line represents the mean metric across 10 test episodes, averaged over 10 controllers trained with different random seeds. Shaded areas denote one standard deviation from the mean, shown in a single direction (toward worse performance) for clarity. We add a point marker for each controller type to indicate the value obtained by the controller that achieved the highest average reward over the 10 test episodes.

become comparable to that of the Koopman eNMPC-based controllers if given a higher training budget.

The control trajectories in Fig. 7 align with the findings derived thus far from Fig. 6. All controllers show an intuitive inverse relationship between electricity prices and coolant flow rate F, although this relationship is noticeably weaker for $\mathrm{PIRL}_{\mathrm{MLP}}$ and $\mathrm{RL}_{\mathrm{MLP}}$. Moreover, the MLP controllers do not utilize the full range of the control inputs frequently. These observations match the lower cost savings of the MLP controllers. Regarding the evolution of the product concentration c, $\mathrm{SI}_{\mathrm{Koop}}\mathrm{PIRL}_{\mathrm{Bounds}}$ and $\mathrm{SI}_{\mathrm{Koop}}\mathrm{RL}_{\mathrm{Bounds}}$ effectively utilize the full feasible range without violating bounds. In contrast, $\mathrm{SI}_{\mathrm{Koop}}$ causes minor constraint violations. $\mathrm{PIRL}_{\mathrm{MLP}}$ and $\mathrm{RL}_{\mathrm{MLP}}$ avoid violations by maintaining c well within bounds but sacrifice process flexibility, limiting economic performance.

The sample efficiency of MBPO is foremost dependent on how many interactions with the real environment are needed until the model ensemble becomes an accurate surrogate of the real environment. Compared to purely data-driven models, PINNs can thrive in settings where few training data are available (Raissi et al., 2019). Here, we aim to confirm that the generally improved performance of the physics-informed variants ($\mathrm{SI}_{\mathrm{Koop}}\mathrm{PIRL}_{\mathrm{Bounds}}$, $\mathrm{PIRL}_{\mathrm{MLP}}$) compared to their non-physics-informed counterparts ($\mathrm{SI}_{\mathrm{Koop}}\mathrm{RL}_{\mathrm{Bounds}}$, $\mathrm{RL}_{\mathrm{MLP}}$) is indeed due to the PINNs becoming accurate predictors of the real system behavior more quickly than the vanilla NNs. We randomly pick one of the test trajectories produced by a fully trained $\mathrm{SI}_{\mathrm{Koop}}\mathrm{PIRL}_{\mathrm{Bounds}}$ controller. Then, we randomly pick one of the $\mathrm{SI}_{\mathrm{Koop}}\mathrm{PIRL}_{\mathrm{Bounds}}$ and $\mathrm{SI}_{\mathrm{Koop}}\mathrm{RL}_{\mathrm{Bounds}}$ training runs. We test the ensembles that were produced by the chosen training runs after 20, 500, and 2500 environment steps. Fig. 8 shows the results of these tests. It is evident from Figs. 8(a) and 8(d) that the data from 20 steps is not sufficient to reliably learn accurate models. However, the predictions of the PINN ensemble diverge less strongly than those of the vanilla NN ensemble. After 500 steps in the real environment (Figs. 8(b) and 8(e)), all but one of the PINNs provide highly accurate predictions, whereas the predictions of the vanilla NN ensemble look comparable to those of the PINN ensemble after 20 steps. After 2500 steps, all members of the PINN ensemble provide accurate predictions over the full 168-step horizon of the test episode. Most members of the vanilla NN ensemble also remain accurate over the full episode; however, some still diverge from the true trajectory after some time. Note that (i) during MBPO policy optimization (see Section 3.2.4), model predictions are chained only up to eight times and that (ii) for each prediction, a different ensemble member is randomly chosen. Thus, MBPO is relatively robust with respect to compounding model errors. This explains why one can expect to obtain good control performance well before one can expect the ensemble to converge to accurate closed-loop predictions over a long time horizon (cf. Figs. 6 and 8).

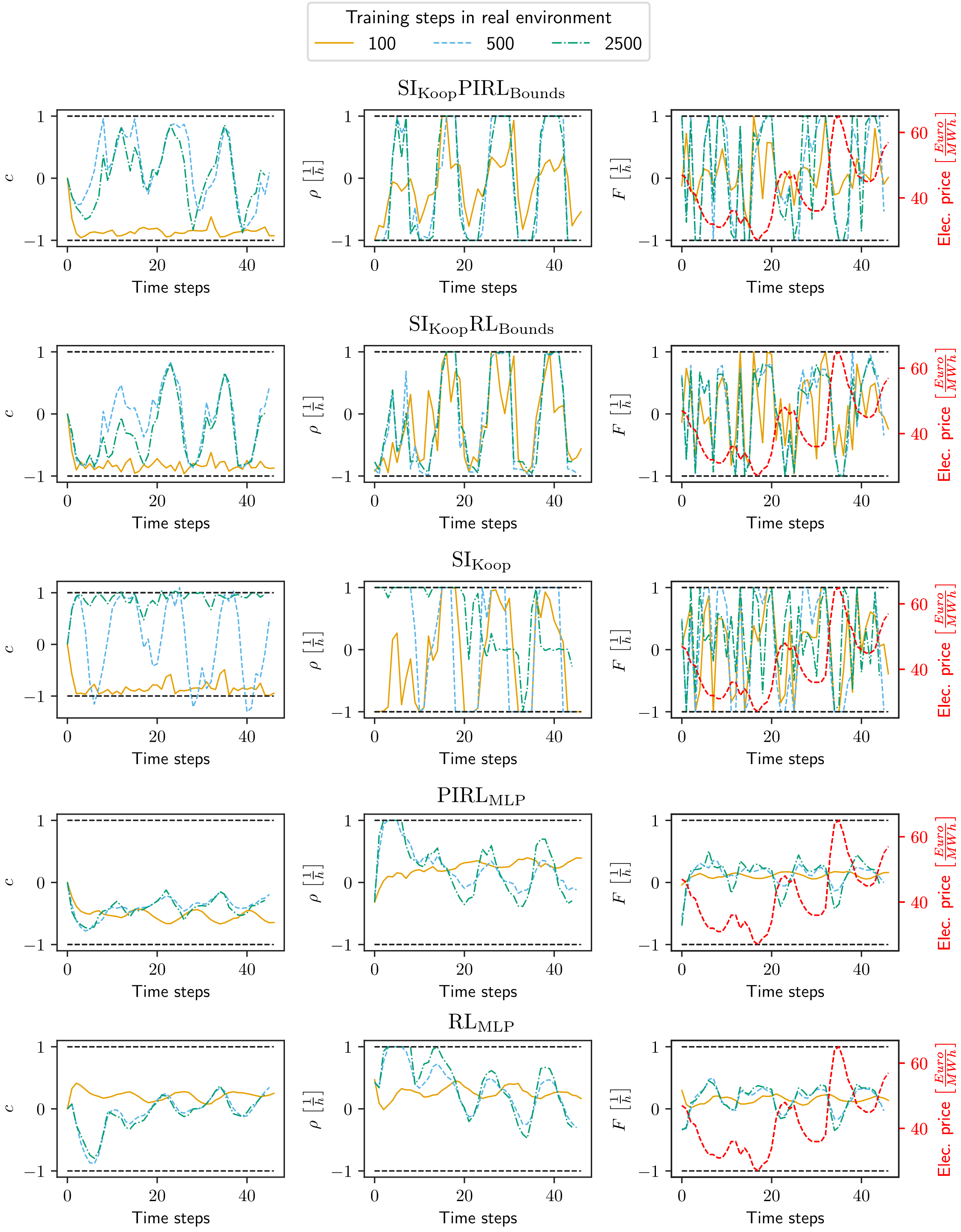


Fig. 7. Control trajectory comparison. The bounds of each variable (see Table 2) are used for scaling to the [−1, 1] range. We omit the temperature T since it never reaches its bounds in any of the test episodes.

As explained in Section 3.2, in training the $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ controllers, θ_K are trained only via SI, whereas θ_B are trained only via PPO using imagined policy rollouts using the learned model ensemble. This means that besides their good performance (see Fig. 6), the $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ controllers have another valuable property: the (very few) parameters θ_B that are used to optimize the controller for task-optimal control performance are intuitively interpretable. Each parameter in θ_B modifies one of the bounds in the OCPs of the eNMPC (see Eq. (8)), i.e., the lower and upper bounds of c, T, and the storage level. Fig. 9 shows the evolution of θ_B during the $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ training runs. For both variants and across all trained eNMPCs, the bounds of c are tightened (see Figs. 9(a) and 9(b)), thus decreasing the likelihood of constraint violations. Figs. 9(c) and 9(d), which depict the adaptation of the bounds of T, are less conclusive. Here, the observed adaptations to the bounds are smaller than those observed for c. Furthermore, the

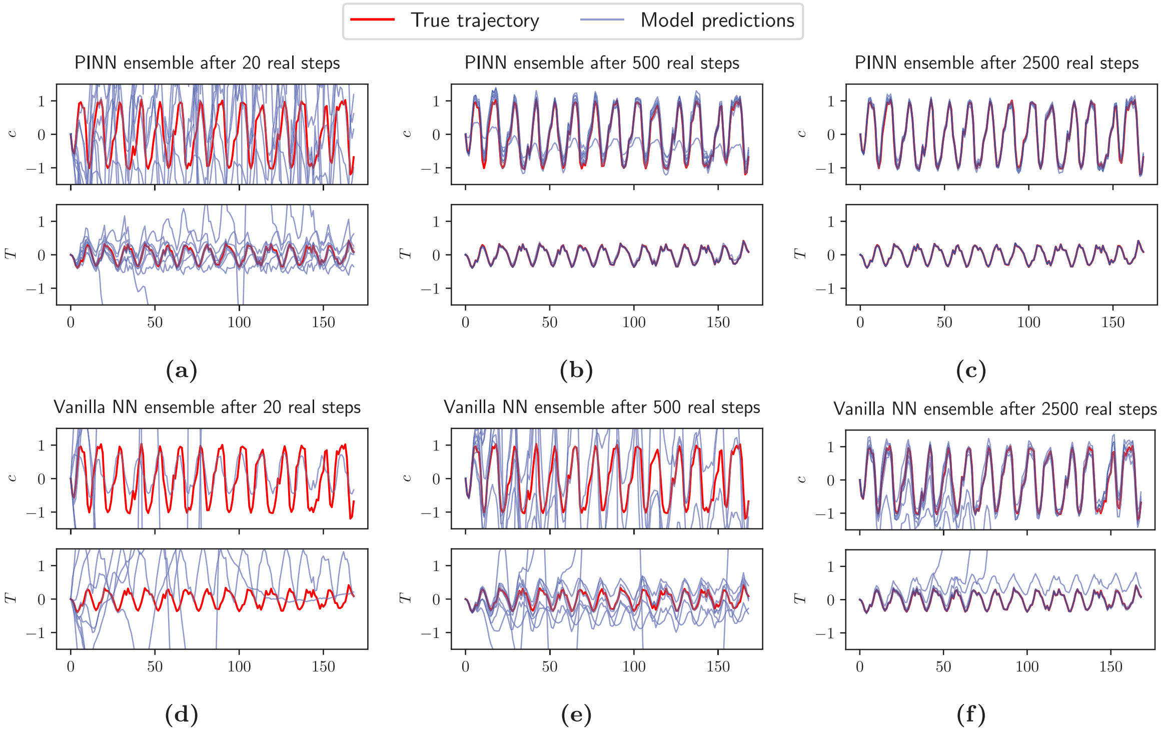


Fig. 8. Comparison of closed-loop prediction results of a PINN ensemble and a vanilla NN ensemble after different numbers of data sampling steps in the real environment (Fig. 3(a), first step). The bounds of each variable (see Table 2) are used for scaling to the [−1, 1] range. The red line depicts the true trajectory of c and T. Each blue line corresponds to the closed-loop prediction of one ensemble member. Predictions are chained over the full horizon (168 steps) of the episode.

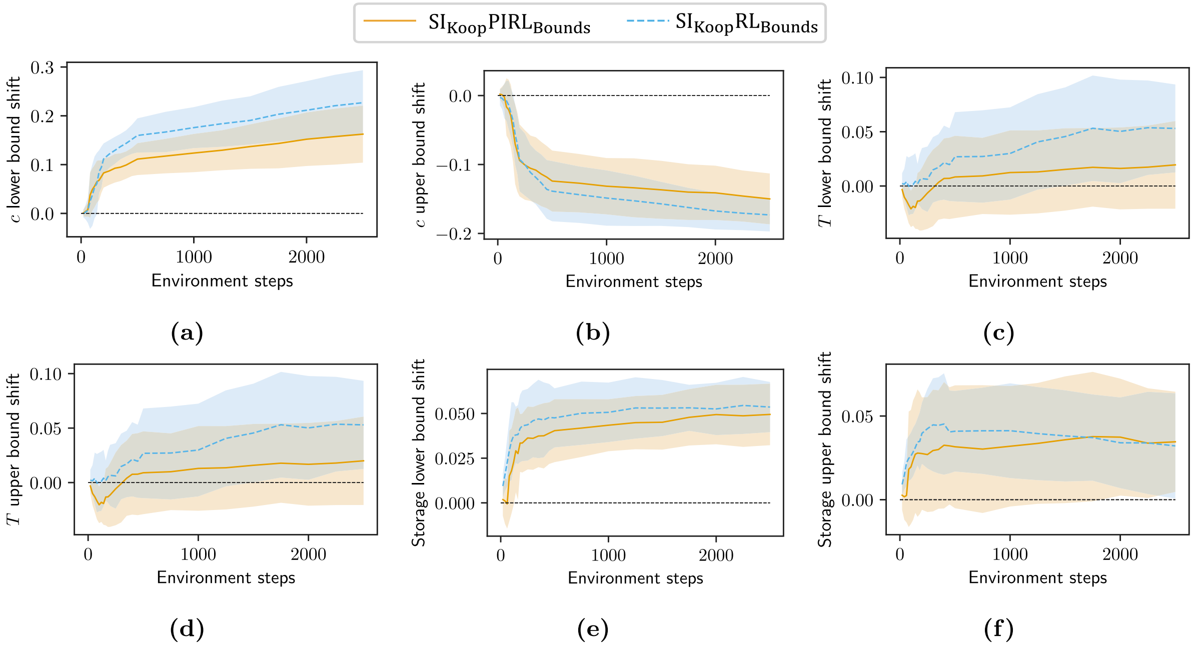


Fig. 9. Evolution of the parameters θ_B during training. Lines depict the average of the respective parameter over 10 training runs; the shaded regions depict one standard deviation across the different training runs. The adaptation of the bounds is performed with respect to the scaled c and T variables (both variables scaled to the [−1,1] range using their bounds given in Table 2). We leave the product storage in its original [0,6] range.

directions of the adaptations do not align across all training instances. Since in our controller tests, the bounds of c were violated frequently (see Fig. 7), whereas no violations of the T bounds were observed, it makes intuitive sense that optimizing θ_{B} leads to a tightening of the c bounds and a less decisive adaptation of the T bounds. Fig. 9(f) shows that a small but consistent back-off from the lower bound of the storage is learned. The adaptation of the upper bound of the storage is less decisive (see Fig. 9(e)). This is not surprising since, like the bounds of T, the upper storage bound is not violated by any of the controllers during training.

4. Conclusion

We present a method that aims to increase the sample efficiency of RL-based training of data-driven (e)NMPCs for specific control tasks. To that end, we combine our previously published approach (Mayfrank et al., 2024b) for turning Koopman (e)NMPCs into automatically differentiable policies that can be trained using RL methods with a physics-informed version of the MBPO algorithm (Janner et al., 2019), a state-of-the-art Dyna-style model-based RL algorithm. Our method iterates between three steps (see Fig. 3(a)): First, the Koopman (e)NMPC gathers data about the system dynamics by interacting with the physical system. Second, the Koopman model that is used in the (e)NMPC is fitted to the data via SI. Furthermore, an ensemble of NNs is fitted to the data. If (partial) knowledge of the system dynamics is available, physics-informed training can be utilized to increase the accuracy of the NN ensemble. Third, a surrogate environment is constructed using the NN ensemble to simulate the dynamics of the environment. In conjunction with the PPO algorithm (Schulman et al., 2017), this surrogate environment is used to optimize the Koopman (e)NMPC by adapting the variable bounds that are imposed in the OCPs of the (e)NMPC.

We validate our method using a demand response case study (Mayfrank et al., 2024b) based on a benchmark CSTR model (Flores-Tlacuahuac and Grossmann, 2006). The case study involves nonlinear dynamics and hard constraints on system variables; however, due to its small scale it is far less complex than many real-world systems. We compare the performance of our method to that of Koopman eNMPCs trained solely via iterative SI and to NN policies trained via (physics-informed) model-based RL. We find that our method (see Section 2.3) outperforms all other tested approaches when applied to our case study: It reaches higher rewards and does so with better sample efficiency and lower variance between differently seeded training instances, resulting in improved economic performance and constraint satisfaction compared to the benchmark methods.

Although our method for the training of task-optimal Koopman (e)NMPCs achieves excellent sample efficiency in our case study, the training process incurs a high computational cost. This cost is mainly driven by the policy optimization step (see Fig. 3(a)), which involves (i) numerous interactions of the controller with the learned surrogate environment and (ii) differentiating through the OPCs of the (e)NMPC in order to execute policy gradient updates. However, these computationally intensive steps of our approach *do not* need to be executed in real-time. The only step where computations need to be done in real-time is the data sampling step (see Fig. 3(a)). However, while this step involves inference of the current version of the Koopman (e)NMPC, it does not involve backpropagation and parameter updates. The computational burden of merely evaluating a Koopman (e)NMPC is relatively low since it is predominantly driven by the need to solve a convex OCP. Therefore, we assume that our method should be scalable to large systems. Our approach could offer concrete benefits for the control of various real-world systems where mechanistic models cannot be used for predictive control. Future work should, therefore, validate our method on case studies matching the scale and complexity of challenging real-world control problems.

Recent works aim to improve upon the MBPO algorithm (e.g., Frauenknecht et al. (2024)) and Dyna-style model-based RL in general (e.g., Frauenknecht et al. (2025)). These methods do not require a specific policy architecture. Therefore, they do not interfere with our approach and could be combined with our method for potentially even better performance. Another avenue of possible future research is combining our method with approaches for learning disturbance estimators for offset-free Koopman MPC (e.g., Son et al. (2021, 2022)): Instead of learning modifications to the state bounds, a task-optimal disturbance estimator could be learned to estimate the disagreement between the Koopman model and the PINN ensemble.

CRediT authorship contribution statement

Daniel Mayfrank: Writing – review & editing, Writing – original draft, Visualization, Software, Methodology, Investigation, Conceptualization. **Mehmet Velioglu:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Alexander Mitsos:** Writing – review & editing, Supervision, Funding acquisition, Conceptualization. **Manuel Dahmen:** Writing – review & editing, Supervision, Methodology, Funding acquisition, Conceptualization.

Nomenclature

Abbreviations

NN	Neural network
CSTR	Continuous stirred-tank reactor
DAE	Differential–algebraic equation
(e)(N)MPC	(Economic) (nonlinear) model Predictive control
MBPO	Model-based policy optimization
MDP	Markov decision process
MLP	Multilayer perceptron
MSE	Mean squared error
OCP	Optimal control problem
PDE	Partial differential equation
PINN	Physics-informed neural network
PPO	Proximal policy optimization
RHS	Right-hand side of equation
RL	Reinforcement learning
SI	System identification

Greek symbols

α	Reward calculation hyperparameter
$\boldsymbol{\theta}$	Learnable parameters of controller
λ	PINN loss weight hyperparameter
$\boldsymbol{\mu}$	Expected value for action selection
$\boldsymbol{\pi}$	Policy
ρ	CSTR production rate
$\boldsymbol{\sigma}$	Standard deviation for action selection
τ	PINN time
$\boldsymbol{\phi}$	Learnable parameters of critic
$\boldsymbol{\Phi}$	MPC stage cost
$\boldsymbol{\psi}$	Encoder MLP
$\boldsymbol{\omega}$	Learnable parameters of dynamic model

Latin symbols

$\boldsymbol{A}$	Autoregressive part of Koopman dynamics matrix
$\boldsymbol{B}$	External input part of Koopman dynamics matrix
c	CSTR product concentration
$\boldsymbol{C}$	Decoder matrix of Koopman model
$\mathcal{D}$	State transition database
F	CSTR coolant flow rate
$\boldsymbol{g}$	Inequality constraints
$\boldsymbol{h}$	Equality constraints
l	Storage filling level
m	Control input dimensionality
M	Penalty factor for slack variables
n	State dimensionality
N	Koopman state dimensionality
$\mathcal{N}$	Normal distribution
$\boldsymbol{NN}$	Neural network function representation
p	Electricity price
r	Reward
R	Reaction rate
$\boldsymbol{s}$	Slack variables
t	Time
T	CSTR temperature
$\mathbb{T}$	Set of discrete time steps
$\boldsymbol{u}$	Control variables
$\boldsymbol{x}$	System state variables
$\boldsymbol{z}$	Koopman state variables

Subscripts

ss	Steady-state
t	Discrete time step

Superscripts

$\dot{}$	Time derivative
$*$	Indicates optimality
$\hat{}$	Denotes model prediction

Declaration of generative AI and AI-assisted technologies in the writing process

During the preparation of this work Daniel Mayfrank used Grammarly in order to correct grammar and spelling and to improve style of writing. After using this tool, all authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centres.

Part of this work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), Germany – 466656378 – within the Priority Programme “SPP 2331:Machine Learning in Chemical Engineering”.

Data availability

Data sharing not applicable to this article.

References

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., Kolter, J.Z., 2019. Differentiable convex optimization layers. Adv. Neural Inf. Process. Syst. 32, 9558–9570.

Brandner, D., Lucia, S., 2024. Reinforced model predictive control via trust-region quasi-Newton policy optimization. arXiv preprint arXiv:2405.17983.

Brandner, D., Talis, T., Esche, E., Repke, J.-U., Lucia, S., 2023. Reinforcement learning combined with model predictive control to optimally operate a flash separation unit. In: Computer Aided Chemical Engineering. Vol. 52, Elsevier, pp. 595–600.

Chen, B., Cai, Z., Bergés, M., 2019. Gnu-RL: A precocial reinforcement learning solution for building HVAC control using a differentiable MPC policy. In: Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. pp. 316–325.

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

Dogru, O., Xie, J., Prakash, O., Chiplunkar, R., Soesanto, J.F., Chen, H., Velswamy, K., Ibrahim, F., Huang, B., 2024. Reinforcement learning in process industries: Review and perspective. IEEE CAA J. Autom. Sin. 11 (2), 283–300.

Du, J., Park, J., Harjunkoski, I., Baldea, M., 2015. A time scale-bridging approach for integrating production scheduling and process control. Comput. Chem. Eng. 79, 59–69.

Faria, R.d.R., Capron, B.D.O., Secchi, A.R., de Souza, Jr., M.B., 2022. Where reinforcement learning meets process control: Review and guidelines. Processes 10 (11), 2311.

Faridi, I.K., Tsotsas, E., Kharaghani, A., 2024. Advancing process control in fluidized bed biomass gasification using model-based deep reinforcement learning. Processes 12 (2), 254.

Flores-Tlacuahuac, A., Grossmann, I.E., 2006. Simultaneous cyclic scheduling and control of a multiproduct CSTR. Ind. Eng. Chem. Res. 45 (20), 6698–6712.

Frauenknecht, B., Eisele, A., Subhasish, D., Solowjow, F., Trimpe, S., 2024. Trust the model where it trusts itself–model-based actor-critic with uncertainty-aware rollout adaption. arXiv preprint arXiv:2405.19014.

Frauenknecht, B., Subhasish, D., Solowjow, F., Trimpe, S., 2025. On rollouts in model-based reinforcement learning. arXiv preprint arXiv:2501.16918.

Fujimoto, S., Hoof, H., Meger, D., 2018. Addressing function approximation error in actor-critic methods. In: International Conference on Machine Learning. pp. 1587–1596.

Gao, C., Wang, D., 2023. Comparative study of model-based and model-free reinforcement learning control performance in HVAC systems. J. Build. Eng. 74, 106852.

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. In: Teh, Y.W., Titterington, M. (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. In: Proceedings of Machine Learning Research, Vol. 9, PMLR, pp. 249–256.

Gopaluni, R.B., Tulsyan, A., Chachuat, B., Huang, B., Lee, J.M., Amjad, F., Damarla, S.K., Kim, J.W., Lawrence, N.P., 2020. Modern machine learning tools for monitoring and control of industrial processes: A survey. IFAC- Pap. 53 (2), 218–229.

Gros, S., Zanon, M., 2019. Data-driven economic NMPC using reinforcement learning. IEEE Trans. Autom. Control 65 (2), 636–648.

Haarnoja, T., Zhou, A., Abbeel, P., Levine, S., 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July (2018) 10-15. In: Proceedings of Machine Learning Research, Vol. 80, PMLR, pp. 1856–1865.

Iman, R.L., Helton, J.C., Campbell, J.E., 1981. An approach to sensitivity analysis of computer models: Part I—Introduction, input variable selection and preliminary variable assessment. J. Qual. Technol. 13 (3), 174–183.

Janner, M., Fu, J., Zhang, M., Levine, S., 2019. When to trust your model: Model-based policy optimization. Adv. Neural Inf. Process. Syst. 32, 12498–12509.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Korda, M., Mezić, I., 2018. Linear predictors for nonlinear dynamical systems: Koopman operator meets model predictive control. Automatica 93, 149–160.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., Abbeel, P., 2018. Model-ensemble trust-region policy optimization. arXiv preprint arXiv:1802.10592.

Liu, D.C., Nocedal, J., 1989. On the limited memory BFGS method for large scale optimization. Math. Program. 45 (1–3), 503–528.

Liu, X.-Y., Wang, J.-X., 2021. Physics-informed dyna-style model-based deep reinforcement learning for dynamic control. Proc. R. Soc. A 477 (2255), 20210618.

Lusch, B., Kutz, J.N., Brunton, S.L., 2018. Deep learning for universal linear embeddings of nonlinear dynamics. Nat. Commun. 9 (1), 1–10.

Maddu, S., Sturm, D., Müller, C.L., Sbalzarini, I.F., 2022. Inverse Dirichlet weighting enables reliable training of physics informed neural networks. Mach. Learn.: Sci. Technol. 3 (1), 015026.

Mayfrank, D., Ahn, N.Y., Mitsos, A., Dahmen, M., 2024a. Task-optimal data-driven surrogate models for eNMPC via differentiable simulation and optimization. arXiv preprint arXiv:2403.14425.

Mayfrank, D., Mitsos, A., Dahmen, M., 2024b. End-to-end reinforcement learning of koopman models for economic nonlinear model predictive control. Comput. Chem. Eng. 190, 108824.

Open Power System Data, 2020. Open power system data. https://data.open-power-system-data.org/time_series/ (Accessed 29 August 2022).

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst. 32, 8024–8035.

Ponse, K., Kleuker, F., Fejér, M., Serra-Gómez, Á., Plaat, A., Moerland, T., 2024. Reinforcement learning for sustainable energy: A survey. arXiv preprint arXiv:2407.18597.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N., 2021. Stable-baselines 3, Reliable reinforcement learning implementations. J. Mach. Learn. Res. 22 (268), 1–8.

Raissi, M., Perdikaris, P., Karniadakis, G.E., 2019. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J. Comput. Phys. 378, 686–707.

Ramesh, A., Ravindran, B., 2023. Physics-informed model-based reinforcement learning. In: Matni, N., Morari, M., Pappas, G.J. (Eds.), Learning for Dynamics and Control Conference, L4DC 2023 15-16 2023, Philadelphia, PA, USA. In: Proceedings of Machine Learning Research, volume 211, PMLR, pp. 26–37.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Son, S.H., Choi, H.-K., Kwon, J.S.-I., 2021. Application of offset-free Koopman-based model predictive control to a batch pulp digester. AIChE J. 67 (9), e17301.

Son, S.H., Narasingam, A., Kwon, J.S.-I., 2022. Development of offset-free Koopman Lyapunov-based model predictive control and mathematical analysis for zero steady-state offset condition considering influence of Lyapunov constraints on equilibrium point. J. Process Control 118, 26–36.

Sutton, R.S., 1991. Dyna, an integrated architecture for learning, planning, and reacting. ACM Sigart Bull. 2 (4), 160–163.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.

Tang, W., Daoutidis, P., 2022. Data-driven control: Overview and perspectives. In: 2022 American Control Conference. ACC, IEEE, pp. 1048–1064.

Velioglu, M., Zhai, S., Rupprecht, S., Mitsos, A., Jupke, A., Dahmen, M., 2025. Physics-informed neural networks for dynamic process operations with limited physical knowledge and data. Comput. Chem. Eng. 192, 108899.

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. arXiv preprint arXiv:1907.02057.

Zhang, W., Cao, X., Yao, Y., An, Z., Xiao, X., Luo, D., 2021. Robust model-based reinforcement learning for autonomous greenhouse control. In: Asian Conference on Machine Learning. PMLR, pp. 1208–1223.