End-to-End Reinforcement Learning of Koopman Model Predictive Control

Ende-zu-Ende bestärkendes Lernen von Koopman modellprädiktiver Regelung

Von der Fakultät für Maschinenwesen der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Daniel Georg Mayfrank

Berichter: Universitätsprofessor Alexander Mitsos, Ph.D.
Universitätsprofessor Dr.-Ing. Sergio Lucia

Tag der mündlichen Prüfung: 14.11.2025

Titel: End-to-End Reinforcement Learning of Koopman Model Predictive Controllers

Autor: Daniel G. Mayfrank

Reihe: Aachener Verfahrenstechnik Series
AVT.SVT - Process Systems Engineering
Band 42 (2025)

Herausgeber: Aachener Verfahrenstechnik
Forckenbeckstraße 51
52074 Aachen
Tel.: +49 (0)241 80 97717
Fax.: +49 (0)241 80 92326
E-Mail: secretary.svt@avt.rwth-aachen.de
https://www.avt.rwth-aachen.de

Volltext verfügbar: 10.18154/RWTH-2025-09976

Preface

Although this dissertation lists only one author, many people contributed, directly and indirectly, to its completion. I am deeply grateful for the guidance, support, and inspiration I received throughout this journey.

First and foremost, I thank my doctoral advisor, Alexander Mitsos, whose valuable input and remarkable efficiency have shaped both this work and my approach to research. His trust in me, the freedom he gave me to pursue my own ideas, and his consistently fast and thoughtful feedback created an environment in which I could grow. I always felt that differing opinions were welcomed and taken seriously, and this openness has been invaluable for my personal and professional development.

My deepest gratitude also goes to my mentor and closest collaborator, Manuel Dahmen. It is hard to imagine a better direct supervisor. He was always available when I needed insight, yet never imposed himself unnecessarily — a balance that made working with him exceptionally pleasant. His perfectionist approach taught me a great deal about both research and writing, and his efficiency played a crucial role in helping me progress smoothly through this project.

I conducted my research at ICE-1 at Forschungszentrum Jülich (FZJ) and was part of the Helmholtz School for Data Science in Life, Earth, and Energy (HDS-LEE). I am grateful for the freedom these institutions provided to shape my doctoral project according to my interests. Through FZJ and HDS-LEE, I also met many inspiring people, two of whom are now among my closest friends. My sincere thanks go again to Alexander and Manuel for enabling and supporting this environment.

A special thank you goes to Danimir Doncevic, who first sparked my interest in many of the topics that eventually led to this dissertation. As my former supervisor and later colleague, he showed me that deep learning is far more than calling fit on a universal-approximator model; his perspective ignited several of the ideas explored in this work. He was an excellent mentor and played a central role in motivating me to pursue a PhD.

I would also like to thank my colleagues, especially Mehmet, Moritz, Eike, and Maurizio for the great time we shared. Working with my students Dion, Na Young, Kayra and Lukas, was equally rewarding; collaborating with you was both fun and intellectually enriching. Your contributions enabled parts of this work that I could not have done alone.

My heartfelt thanks go to my wife Hila, for sharing countless home-office days with me and for providing emotional support, patience, and encouragement, especially during moments of frustration. I am also deeply grateful to my family for their continuous interest in the content of this dissertation and their genuine attempts to engage with topics that can be difficult to access from the outside.

Berlin, November 2025 *Daniel Mayfrank*

Contents

Abbreviations

ASU	Air Separation Unit
BPTT	Backpropagation Through Time
CSTR	Continuous Stirred-Tank Reactor
DAE	Differential-Algebraic Equation
(e)(N)MPC	(economic) (Nonlinear) Model Predictive Control
HPC	High Pressure Column
HVAC	Heating, Ventilation and Air Conditioning
KKT	Karush-Kuhn-Tucker
LMPC	Linear Model Predictive Control
LICQ	Linear Independence Constraint Qualification
MAC	Main Air Compressor
MBPO	Model-Based Policy Optimization
MDP	Markov Decision Process
MHE	Moving Horizon Estimation
MLP	Multilayer Perceptron
MSE	Mean Squared Error
NN	Neural Network
OCP	Optimal Control Problem
PDE	Partial Differential Equation
PHX	Pre-Heat Exchanger
PINN	Physics-Informed Neural Network
PIRL	Physics-Informed Reinforcement Learning
PO-GRL	Partially Observable Guided Reinforcement Learning
PPO	Proximal Policy Optimization
QP	Quadratic Program
RHS	Right-Hand Side of equation
RL	Reinforcement Learning
SAC	Soft Actor-Critic
SCS	Strict Complementarity Slackness
SHAC	Short-Horizon Actor-Critic
SI	System Identification
SOSC	Second-Order Sufficient Condition
TRPO	Trust Region Policy Optimization

Kurzfassung

Modellbasierte Regelungsverfahren wie modellprädiktive Regelung (MPC) und Varianten davon wie z.B. ökonomische nichtlineare MPC (eNMPC) sind in der chemischen Industrie nach wie vor unverzichtbar. Mechanistische Modelle sind jedoch oft nicht verfügbar oder zu rechenintensiv für den Einsatz als Teil einer (e)NMPC. Datengetriebene Modelle, die meist mittels Systemidentifikation (SI) gelernt werden, bieten eine recheneffiziente Alternative. Allerdings maximiert SI die durchschnittliche Vorhersagegenauigkeit und kann dadurch in (e)NMPCs zu suboptimalem Verhalten führen. Im Gegensatz dazu nutzt aktuelle Forschung bestärkendes Lernen (RL) um datengetriebene Modelle Ende-zu-Ende für optimales Verhalten als Teil eines Reglers für spezifische Anwendungen zu trainieren. Diese Arbeit leistet einen Beitrag zu diesem aufstrebenden Forschungsfeld, indem sie Methoden zum RL-basierten Ende-zu-Ende-Lernen von Koopman-Modellen für (e)NMPC-Regler entwickelt.

Koopman-Modelle können die Dynamik nichtlinearer Systeme abbilden und führen gleichzeitig zu konvexen optimalen Regelungsproblemen (OCPs), wenn sie in einer (e)NMPC verwendet werden, womit sie eine günstige Balance zwischen Darstellungskapazität und Berechnungseffizienz bieten. Mittels post-optimaler Sensitivitätsanalyse entwickeln wir eine Methode zur Konstruktion automatisch differenzierbarer Koopman-basierter (e)NMPC-Regler, die über die lernbaren Parameter des Koopman-Modells optimiert werden können. Wir nutzen den RL-Algorithmus Proximal Policy Optimization (PPO) um die Koopman-(e)NMPC-Regler für spezifische Regelungsaufgaben zu optimieren.

Unter der Annahme der Verfügbarkeit vollständiger Zustandsmessungen demonstrieren wir die Effektivität unserer Methode in NMPC und eNMPC Fallstudien. Diese basieren auf (i) einem kleinen kontinuierlichen Rührkesselreaktormodell mit zwei Differentialzuständen und zwei Stellgrößen, sowie (ii) einer Luftzerlegungsanlage mit 119 Differentialzuständen und etwa 2300 algebraischen Zuständen. Die Ergebnisse zeigen, dass die vorgeschlagene Methode in Bezug auf die Regelungsleistung im Vergleich zu traditionellen Benchmark-Verfahren vorteilhaft abschneidet. Außerdem zeigen wir, dass die (e)NMPC-Regler im Gegensatz zu RL-trainierten Reglern in Form künstlicher neuronaler Netze auf bestimmte Änderungen in der Regelungsumgebung reagieren können, ohne erneut trainiert werden zu müssen. Allerdings beobachten wir (i) Konvergenzprobleme aufgrund ungenauer Gradientenschätzungen und (ii) eine geringe Stichprobeneffizienz.

Wir adressieren das Konvergenzproblem (i), indem wir die automatische Differenzierbarkeit von Trainingsumgebungen auf Basis mechanistischer Simulationsmodelle nutzen. Die Stichprobeneffizienz (ii) erhöhen wir durch die Integration unseres RL-Trainingsansatzes für Koopman-(e)NMPC-Regler mit modellbasiertem RL. Zudem zeigen wir, dass physikinformiertes Modelllernen die Stichprobeneffizienz weiter erhöhen kann, wenn partielles Vorwissen über die Systemdynamik vorhanden ist.

Insgesamt leistet diese Arbeit einen Beitrag zum Gebiet der datengetriebenen Regelung und zeigt Wege zu leistungsfähigeren, echtzeitfähigen, datengetriebenen (e)NMPCs auf.

Summary

Model-based control methods such as Model Predictive Control (MPC) and variants thereof, e.g., economic nonlinear MPC (eNMPC), remain indispensable in the chemical industry. However, mechanistic models are often unavailable or too computationally expensive for use in (e)NMPC. Data-driven models, usually trained using system identification (SI) approaches, can serve as a computationally cheap alternative to mechanistic models. However, SI focuses narrowly on maximizing average prediction accuracy, which can result in suboptimal performance when the model is used as part of a policy. In contrast, recent research has explored training data-driven models end-to-end for optimal performance in predictive control policies using reinforcement learning (RL) approaches. This thesis contributes to this emerging research field by developing methods for RL-based end-to-end learning of Koopman models for (e)NMPC policies.

Koopman models can accurately represent the dynamics of nonlinear systems while resulting in convex optimal control problems (OCPs) when used in (e)NMPC, thus striking a favorable balance between representational capacity and computational efficiency. By performing post-optimal sensitivity analysis on the resulting OCPs, we develop a method for constructing automatically-differentiable Koopman-based (e)NMPC policies, which can be optimized via the learnable parameters of the Koopman model. We optimize the (e)NMPC policies for specific control tasks using the state-of-the-art actor-critic RL algorithm Proximal Policy Optimization (PPO).

Assuming the availability of full state measurements, we demonstrate the effectiveness of our method in NMPC (setpoint tracking) and eNMPC (demand response) case studies. These are based on (i) a small continuous stirred-tank reactor model with two differential states and two control inputs and (ii) an air separation unit with 119 differential states and approximately 2300 algebraic states. The results show that the proposed method performs favorably in terms of the control performance of the resulting policies compared to traditional benchmarks, including neural network policies trained using RL and Koopman-based eNMPC policies trained via system identification. Furthermore, we show that, in contrast to the neural network policies, the (e)NMPC policies can react to certain changes in the control setting without retraining. However, we observe (i) convergence problems resulting from inaccurate policy gradient estimates and (ii) low sample efficiency.

To address the former issue, we exploit the automatic differentiability of training environments based on mechanistic simulation models to aid the policy optimization, resulting in substantially improved convergence and control performance. Furthermore, we improve the sample efficiency of the learning process by integrating our method for RL-based training of Koopman (e)NMPC policies with Dyna-style model-based RL. We also show that when leveraging model-based RL, the sample efficiency can be increased further by utilizing partial prior knowledge about the system dynamics via physics-informed model learning.

In sum, this thesis contributes to the field of data-driven control and shows avenues toward higher-performance, real-time-capable, data-driven (e)NMPCs.

Publications and copyrights

This thesis is a result of the research performed by the author during his time at the IEK-10/ICE-1. Parts of this thesis have already been published in journals or conference proceedings. The publications are integrated into the thesis as described in the following paragraphs. Parts of each publication have also been integrated into the introduction (Chapter 1) to motivate our work, and into the conclusion (Chapter 8) to summarize our work. Manuel Dahmen and Alexander Mitsos provided ideas, guidance and edits to the initial drafts of all listed publications. Contributions of other co-authors are stated below the corresponding publications.

- (Mayfrank et al., 2024) Mayfrank, D., Mitsos, A., and Dahmen, M. (2024). End-to-end reinforcement learning of Koopman models for economic nonlinear model predictive control. *Computers & Chemical Engineering*, 190, 108824. Copyright © 2024 The Authors. Published by Elsevier Ltd.

  The content of this publication has also been presented at a scientific conference (Mayfrank et al. (2023)).

  Large parts of the publication have been incorporated into Chapter 4. The according supporting information is included in Appendix A.

- (Mayfrank et al., 2025a) Mayfrank, D., Ahn, N. Y., Mitsos, A., and Dahmen, M. (2025). Task-optimal data-driven surrogate models for eNMPC via differentiable simulation and optimization. 14th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems (DYCOPS) 2025 (in press). Copyright © 2025 The Authors. Published by Elsevier Ltd.

  Large parts of the publication have been incorporated into Chapter 6.

  Co-authors' contributions: Na Young Ahn implemented the code for the SHAC algorithm and reviewed and edited the manuscript.

- (Mayfrank et al., 2025d) Mayfrank, D., Velioglu, M., Mitsos, A., and Dahmen, M. (2024). End-to-end reinforcement learning of Koopman models for economic nonlinear model predictive control. *Computers & Chemical Engineering*, 109240, in press. Copyright © 2025 The Authors. Published by Elsevier Ltd.

  The content of this publication will also be presented at a scientific conference (Mayfrank et al. (2025c)).

  Large parts of the publication have been incorporated into Chapter 7.

  Co-authors' contributions: Mehmet Velioglu implemented the code for the physics-informed neural network (PINN) ensemble, wrote the draft for the PINN-specific parts of Sections 3.1 and 3.2 in the article, and reviewed and edited the manuscript.

- (Mayfrank et al., 2025b) Mayfrank, D., Dernek, K., Lang, L., Mitsos, A., and Dahmen, M. End-to-End Reinforcement Learning of Koopman Models for eNMPC of an Air Separation Unit. arXiv preprint arXiv:2511.04522

  Large parts of the manuscript have been incorporated into Chapter 5.

  Co-authors' contributions: Kayra Dernek implemented an reinforcement learning (RL) environment accessible from Python based on the Modelica model of the air separation unit (ASU). Furthermore, he developed and implemented the iterative data generation and system identification approach, as well as the modifications to the Koopman model architecture and the model upscaling. He also reviewed and edited the manuscript. Laura Lang co-supervised Kayra Dernek during his thesis project, wrote the draft for the section on the ASU demand response RL environment, and reviewed and edited the manuscript.

During his time at IEK-10/ICE-1, the author co-supervised the student theses of Na Young Ahn, Kayra Dernek, and Lukas Kesper. In addition, the authors worked with the student research assistants Dion Jakobs and Kayra Dernek. The valuable work of all students is strongly acknowledged as they assisted in exploratory work, algorithm implementation, and process modeling. Involvement of students is credited with co-authorship in the corresponding journal publications and included in the previously stated author contributions.

Declarations

Declaration of Competing Interest

We have no conflict of interest.

Acknowledgements

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.

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

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

1. Introduction

Model-based optimal control has been an active research topic since the 1960s (Sieben-thal and Aris (1964); Gould et al. (1970)). Model Predictive Control (MPC) subsequently emerged as the dominant real-time realisation of these ideas and has been deployed widely in industrial practice since the 1980s (Qin and Badgwell (2003); Prett and Morari (2013)). Today, MPC and its variants, e.g., economic nonlinear MPC (eNMPC), are more relevant than ever, as volatile market conditions increasingly reward economically optimized and flexible operation (Klaucke et al. (2017); Mitsos et al. (2018)). Yet despite decades of methodological advances and successful applications, fundamental challenges remain: Because of the large scales, nonlinear dynamics, and unknown parameters of process systems, first-principles models are often difficult to derive and reduce (Tang and Daoutidis (2022)). Moreover, (e)NMPC requires solving optimal control problems (OCPs) online, which – given a mechanistic model of an industrial chemical production process – is often computationally intractable (Mitsos et al. (2018)).

Data-driven modeling techniques can help alleviate the challenges associated with traditional model-based process control: If no mechanistic model is available for a given process, data-driven models can be learned via system identification (SI) (Ljung (1998)) from historical operating data or data produced through intentional perturbation of the process. Furthermore, if a mechanistic process model is available but not suitable for control purposes due to computational limitations, computationally cheap data-driven *surrogate* models can be trained to approximate the behavior of the mechanistic model (e.g., McBride and Sundmacher (2019); Fiedler et al. (2020); Johnson et al. (2025)). Popular types of data-driven dynamic models in the chemical process engineering literature are parameterized polynomials (Forrester and Keane (2009)), Gaussian Processes (GPs) (Williams and Rasmussen (1995)), artificial neural networks (NNs) (Goodfellow et al. (2016)), and, more recently, Koopman models (Koopman (1931)).

This thesis aims to develop and apply methods for learning data-driven dynamic models that perform optimally as part of (e)NMPC, i.e., to train models that result in OCPs which are computationally tractable in real-time applications while resulting in good control performance, e.g., accurately tracking setpoints or successfully minimizing production costs while avoiding constraint violations. To this end, we combine methods from Koopman theory (Koopman (1931); Korda and Mezić (2018)), post-optimal sensitivity analysis (Fiacco and Ishizuka (1990)), and reinforcement learning (RL) (Sutton and Barto (2018)). Furthermore, we investigate how these methods can be utilized to optimize the performance of (e)NMPC policies without adapting the underlying dynamic model, i.e., by tuning parameters in the policy's objective function or constraints, such as state bounds.

1.1. RL-based learning of Koopman eNMPCs

As described above, data-driven dynamic models present a promising avenue for rendering eNMPC tractable for complex processes. System identification (SI) represents the predominant methodology employed for training data-driven dynamic models. Depending on the chosen model structure, SI can yield linear models (Ljung (1998)), relatively simple nonlinear models such as polynomials (Forrester and Keane (2009)), probabilistic nonlinear models such as Gaussian Processes (Williams and Rasmussen (1995)), or universal approximators such as highly parameterized NNs (Goodfellow et al. (2016)). However, SI narrowly focuses on optimizing average prediction accuracy on a collection of samples. In contrast, recent studies have demonstrated the superiority of dynamic models, trained explicitly for optimal performance in control applications via reinforcement learning (RL) techniques, over SI-trained models (Amos et al. (2018); Yin et al. (2022); Iwata and Kawahara (2022); Chen et al. (2019); Gros and Zanon (2019)) (see Figure 1.1a). Nonetheless, these findings were limited to the learning of linear models (Chen et al. (2019)), applications devoid of state constraints (Amos et al. (2018); Yin et al. (2022); Iwata and Kawahara (2022); Chen et al. (2019)), or primarily concentrated on the adaptation of bounds and cost function rather than the dynamic MPC model itself (Gros and Zanon (2019)).

RL research primarily focuses on learning control policies that directly map from states to actions and thus eschew the utilization of system state predictions to dictate control actions. In contrast, learning a dynamic model and employing its predictions to derive a control law, as exemplified in eNMPC, has crucial advantages: (i) retraining is rendered unnecessary should constraints or objective functions be altered, provided that the system dynamics persist unaltered; (ii) learning the system dynamics may be more sample-efficient than learning a control policy in the form of a direct state-action mapping (Amos et al. (2018); Chen et al. (2019)); (iii) MPC has a well-established theory regarding performance and stability guarantees, especially for linear models and recent publications aim to extend this theory to include (learned) eNMPC (Gros and Zanon (2019); Angeli et al. (2011)).

Using RL to train dynamic models promises to combine the aforementioned advantages of model-based policies with the typical advantage of end-to-end learning over SI, i.e., the task-optimal performance of the resulting model. To train dynamic models using RL algorithms, (eN)MPC is viewed as a policy whose control outputs can be differentiated with respect to the parameters of the model (see Figure 1.1b). This requires differentiating a solution point of a parametric optimization problem with respect to the parameters (Fiacco and Kyparisis (1985); Fiacco and Ishizuka (1990); Ralph and Dempe (1995)).

Under suitable assumptions, which we discuss in Section 2.5, solutions to parametric optimization problems depend smoothly on the parameters. In particular, if second-order sufficient conditions (SOSC) hold, differentiability of the solution with respect to the parameters is given. In convex parametric programs with strictly convex objectives, the second-order conditions are satisfied automatically when the Karush-Kuhn-Tucker (KKT) conditions (Karush (1939); Boyd et al. (2004)) hold, making verification straightforward in practice. Therefore, for end-to-end learning of dynamic models for MPC, it is desirable to learn dynamic models that lead to convex optimal control problems (OCPs). A simple condition for convex OCPs are formulations with linear dynamics and convex constraints. Koopman theory (Koopman (1931)) aims to find globally-valid linear approximations of nonlinear process dynamics through a nonlinear coordinate transformation into a higher-

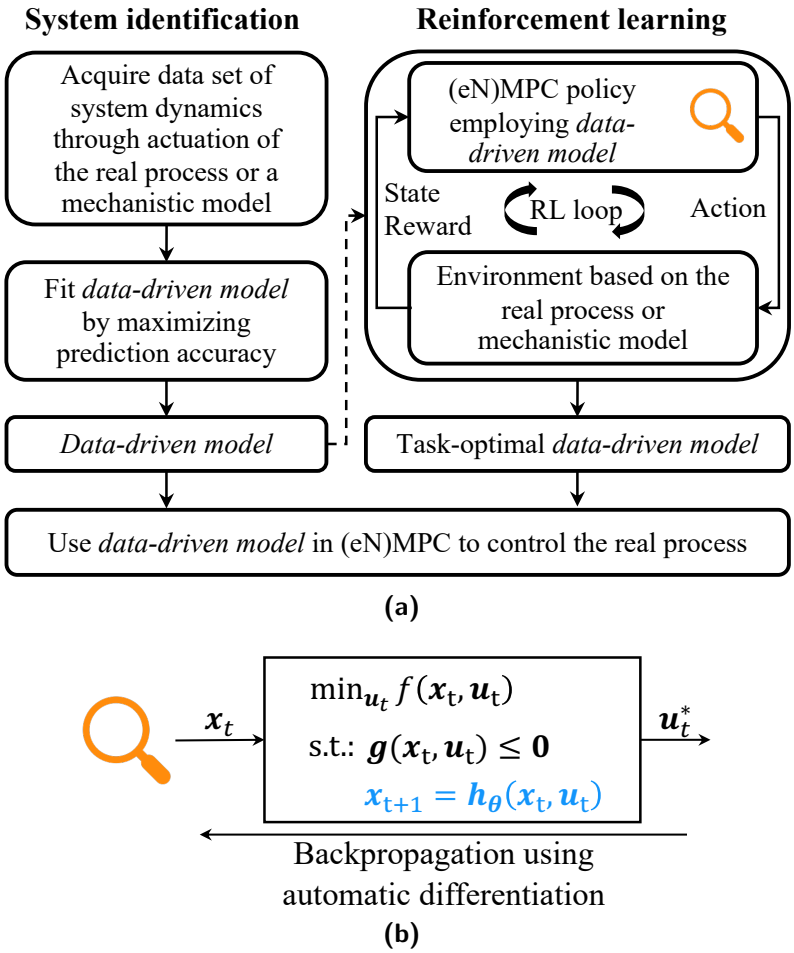


Figure 1.1.: (a) Comparison of two paradigms for the training of data-driven dynamic models for use in eNMPC: SI vs. RL. While RL can start from a randomly parameterized model, leading to a clean split between the methodologies, it is often more practical to use a model identified via SI as an initial guess for the RL procedure. This is pointed out by the dashed arrow. (b) The differentiable eNMPC policy takes as input the current state $\boldsymbol{x}_t$ and computes the optimal control action $\boldsymbol{u}_t^*$ based on a cost function f, inequality constraints $\boldsymbol{g}$, and the learnable dynamic model $\boldsymbol{h}_{\boldsymbol{\theta}}$ (highlighted in blue font), which is parameterized by $\boldsymbol{\theta}$.

dimensional linear state space. Koopman theory and its extension to actuated systems for control applications (Korda and Mezić (2018), therefore, provides a powerful framework to obtain data-driven models that can accurately represent continuous nonlinear dynamical systems while still resulting in convex OCPs, especially when combined with modern deep learning algorithms (Lusch et al. (2018)).

A major impediment to RL-based training of (e)NMPC policies is the notoriously low sample efficiency of RL. This is especially true for the *model-free* RL algorithms that have so far been used for RL-based training of (eN)MPCs, 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. In this context, it is important to clarify that, 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 2.3 for the exact definitions that we use in this work. *Model-based* RL algorithms (e.g., Sutton (1991)) aim to increase the sample efficiency of RL by concurrently learning a policy and a model of the environments' dynamics. For a long time, these algorithms could not compete with their model-free counterparts in terms of the resulting control performance. However, in the last years, a series of innovations (Kurutach et al. (2018); Janner et al. (2019); Frauenknecht et al. (2024)) have led to model-based RL algorithms that match the control performance of state-of-the-art model-free RL algorithms on various benchmark problems while requiring only a fraction of the training samples. Still, the adoption of RL algorithms into the process control community is in its infancy and has largely been limited to model-free RL (Faria et al. (2022); Dogru et al. (2024)). 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.

This thesis develops methods for RL-based training of (e)NMPC policies for process control applications featuring nonlinear dynamics and which may involve hard constraints on states. First, Chapter 2 introduces necessary preliminaries for the remaining chapters, while Chapter 3 reviews prior work that is related to this thesis, though not directly foundational to its development. Chapter 4 addresses how (e)NMPCs that employ Koopman models can be viewed as automatically differentiable policies that can be trained via RL methods by optimizing the Koopman model for a specific control task. We apply our method to an NMPC and an eNMPC application derived from an established nonlinear continuous stirred-tank reactor model. We show that the resulting controllers outperform Koopman (e)NMPCs whose models were trained only using SI. The numerical experiments presented in Chapter 4 are based on a well-studied, nonlinear, but relatively small continuous stirred-tank reactor (CSTR) model (Flores-Tlacuahuac and Grossmann (2006)). As the CSTR is far less complex than many industrial systems of interest, the results should only be seen as successful proofs of concepts of the developed ideas. Chapter 5 introduces a demand response case study based on the benchmark air separation unit (ASU) process presented in Caspari et al. (2020). Due to their high electricity demand, flexible operation strategies such as demand response can substantially reduce the operating cost of ASUs. The model is a nonlinear differential-algebraic equations (DAE) system with 2327 algebraic and 119 differential states. We show that – assuming a perfect state estimator, i.e., by providing full state information to the policy – our base method (Chapter 4) can be applied successfully to this large-scale case study without necessitating any substantial modifications. In Chapters 6 and 7 we revisit the CSTR eNMPC case study introduced in Chapter 4 to further develop our base method (Chapter 4): In Chapter 6 we extend our original method specifically for applications where the goal is to train a data-driven *surrogate* model for a mechanistic model. We exploit the differentiability of the mechanistic model to aid the optimization of the Koopman eNMPC policy, thereby achieving substantially improved control performance compared to our original approach. Chapter 7 addresses the issue of sample efficiency of RL-based training of (e)NMCPs by (i) com-

bining our original method with a model-based RL algorithm (Janner et al. (2019)) and by (ii) utilizing partial physics-knowledge to further increase sample efficiency via physics-informed model learning (Raissi et al. (2019); Velioglu et al. (2025)). Thus, Chapter 7 aims to extend RL-based training of (e)NMPC policies towards complex real-world control problems where no mechanistic model of the environment is available and interactions with the real environment are expensive. Finally, Chapter 8 summarizes the results of this thesis, draws final conclusions, and discusses open questions and possible directions for future research.

2. Preliminaries

This chapter introduces preliminaries that directly enable our following work. This chapter does not provide a comprehensive treatment of these subjects. Instead, we refer the reader to standard textbooks (e.g., Sutton and Barto (2018); Morales (2020); Brunton and Kutz (2022); Boyd et al. (2004); Nocedal and Wright (1999)) for details that are not directly relevant to the methods developed in this thesis.

2.1. Markov decision processes

The Markov decision process (MDP) (Sutton and Barto (2018)) is a framework for modeling discrete-time stochastic decision processes. An MDP is defined by a tuple $(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{U}}, \boldsymbol{\mathcal{P}}, \mathcal{R})$. Herein, $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^n$ is the state space and $\boldsymbol{\mathcal{U}} \subseteq \mathbb{R}^m$ is the action space with associated states $\boldsymbol{x}_t \in \boldsymbol{\mathcal{X}}$ and actions $\boldsymbol{u}_t \in \boldsymbol{\mathcal{U}}$ at time step t, respectively. $\mathcal{R} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{U}} \to \mathbb{R}$ is a scalar reward function, and $\boldsymbol{\mathcal{P}} : \boldsymbol{\mathcal{X}} \times \boldsymbol{\mathcal{U}} \to \mathrm{Dist}(\boldsymbol{\mathcal{X}})$ a stochastic transition function that describes the probability distribution for the next state, given the current state and action.

2.2. Model predictive control

Model predictive control (MPC) (Rawlings et al. (2017)) refers to a family of solution approaches for control problems that can be represented by MDPs. MPC is based on solving optimal control problems (OCPs) to determine the optimal control actions $\boldsymbol{u}_t$ based on a (usually deterministic) model $\boldsymbol{\mathcal{F}} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ of the dynamics of the system. This model can be derived from first principles or learned from data. OCPs are optimization problems where the objective is to determine optimal control profiles over a finite time horizon for a given control task, e.g., tracking a series of system state setpoints or minimizing the operating cost of a production system while respecting product quality constraints.

Given a time horizon of t_f time steps, $\mathrm{T} \coloneqq \{0, \dots, t_f\}$ and $\mathrm{T}_{-1} \coloneqq \{0, \dots, t_f - 1\}$, we consider OCPs of the form

$$\min_{(\boldsymbol{u}_t)_{t \in \mathrm{T}_{-1}}} \sum_{t \in \mathrm{T}} \Phi(\boldsymbol{x}_t, \boldsymbol{u}_t) + V_f(\boldsymbol{x}_{t_f}), \tag{2.1a}$$

$$\text{s.t. } \boldsymbol{x}_0 = \boldsymbol{x}_{\mathrm{init}}, \tag{2.1b}$$

$$\boldsymbol{x}_{t+1} = \boldsymbol{\mathcal{F}}(\boldsymbol{x}_t, \boldsymbol{u}_t), \ \forall t \in \mathrm{T}_{-1}, \tag{2.1c}$$

$$\boldsymbol{h}_{\mathrm{f}}(\boldsymbol{x}_{t_f}) \leq 0, \tag{2.1d}$$

$$\underline{\boldsymbol{x}} \leq \boldsymbol{x}_t \leq \bar{\boldsymbol{x}}, \ \forall t \in \mathrm{T}, \tag{2.1e}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_k \leq \bar{\boldsymbol{u}}, \ \forall t \in \mathrm{T}_{-1}, \tag{2.1f}$$

where $\boldsymbol{x}_{\text{init}}$ (Eq. (2.1b)) is the initial state of the system at $t = 0$. $\boldsymbol{u}_t$ and $\boldsymbol{x}_t$ are the control inputs and system states at each time step t, respectively. The objective function (Eq. (2.1a)), consists of a stage cost $\Phi(\boldsymbol{x}_t, \boldsymbol{u}_t)$ penalizing states and controls at each time step, and a terminal cost $V_f(\boldsymbol{x}_{t_f})$ that penalizes the state at the final time step. Lower and upper bounds are imposed on the state (Eq. (2.1e)) and control (Eq. (2.1f)) variables. Eq. (2.1c) accounts for the system dynamics. The terminal constraint (Eq. (2.1d)) is typically used to enforce that the system can always return to a safe region.

When used in the context of MPC (Rawlings et al. (2017)), OCPs are typically solved using a receding horizon strategy to alleviate model errors, i.e., after solving the OCP at the current time step, only the first control input is applied to the system. Then, the optimization is repeated at the next time step with updated state information.

Traditionally, MPC (Rawlings et al. (2017)) focuses on following reference trajectories, i.e., the objective function of the OCP (2.1a) typically penalizes the distance of the systems states and the control effort quadratically. If combined with a linear model of the dynamics (LMPC), such an objective function results in OCPs that take the form of convex quadratic programs (QPs), which can be solved reliably and efficiently (Boyd et al. (2004)). However, for control problems with nonlinear dynamics, LMPC may not produce satisfactory control performance across the entire state space. For such problems, nonlinear MPC (NMPC) can improve the control performance by utilizing a nonlinear dynamic model. Typically, NMPC involves nonconvex OCPs, since the model $\boldsymbol{\mathcal{F}}$ is part of an equality constraint in the the OCP (see Eq. (2.1c)) and nonlinear equality constraints lead to a nonconvex feasible set. Thus, for high-frequency control or for large systems that result in OCPs with many optimization variables, NMPC can quickly become computationally intractable.

In contrast to (N)MPC, which aims to follow a reference trajectory for the system states, economic (N)MPC (e(N)MPC) (Ellis et al. (2018)) directly optimizes the economic performance of the process by integrating economic objectives into the OCP. For instance, the objective function may minimize the operating cost of a production process, while quality and safety are ensured by the constraints.

2.3. Deep reinforcement learning with continuous action spaces

Like MPC and its variants, RL (Sutton and Barto (2018)) is a framework for solving control problems represented by MDPs. Deep RL utilizes nonlinear function approximators, such as NNs, to learn optimal control policies through sequential feedback from trial-and-error actuation. An *environment* is the MDP of a specific RL problem. An *episode* refers to a sequence of interactions between a policy and its environment, starting from an initial state, involving a series of control inputs, and leading to a terminal state. Given a reward function

$$r_{t+1} = \mathcal{R}(\boldsymbol{u}_t, \boldsymbol{x}_{t+1}), \tag{2.2}$$

the *return*

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

is the (discounted) sum of rewards over an episode with $T+1$ time steps, starting at time step t, with the parameter γ, $0 \leq \gamma \leq 1$, being the discount rate. A policy $\boldsymbol{\pi_\theta}(\boldsymbol{u}_t|\boldsymbol{x}_t)\colon \mathbb{R}^n \to \mathbb{R}^m$ is a function, parameterized by $\boldsymbol{\theta}$, mapping states $\boldsymbol{x}_t$ to (probability distributions over) control inputs $\boldsymbol{u}_t$. The goal of RL is to maximize the expected future return.

In engineering applications with continuous action spaces, RL methods that involve agents learning parameterized policies $\boldsymbol{\pi_\theta}(\boldsymbol{u}_t|\boldsymbol{x}_t)\colon \mathbb{R}^n \mapsto \mathbb{R}^m$, directly mapping states to actions, are the most successful (Sutton and Barto (2018)). These methods can be classified as either policy-gradient or actor-critic methods (see Figure 2.1). Policy-gradient methods

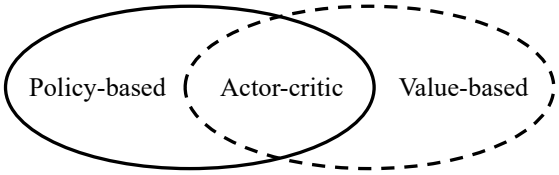
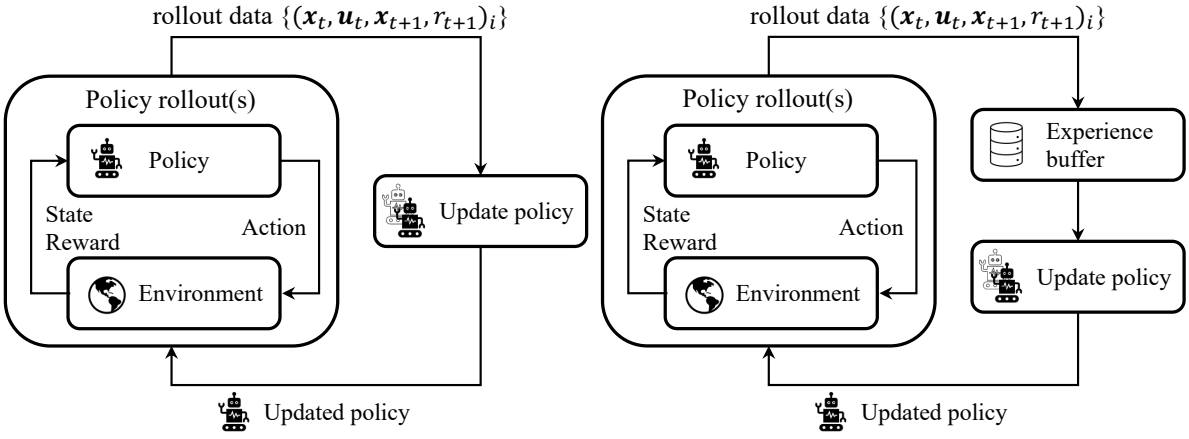


Figure 2.1.: Classification of deep RL algorithms. Pure value-based methods are usually not well-suited for problems with continuous action spaces. Reproduced from Morales (2020).

(e.g., Williams (1992)), which learn only a policy, suffer from high variance in the targets used for policy-gradient updates. Actor-critic methods learn both a policy and a value function. The value function assesses the policies' action selections when updating the policy and is therefore called critic. Using a critic introduces bias into the policy update but substantially reduces the variance of the updates (Sutton and Barto (2018)). In the last decade, actor-critic methods have been the focus of intense research efforts (Silver et al. (2014); Fujimoto et al. (2018); Schulman et al. (2015a, 2017); Haarnoja et al. (2018)) and today's state-of-the-art algorithms typically vastly outperform pure policy-gradient methods.

Actor-critic algorithms can be separated into *on-policy* (e.g., Schulman et al. (2015a, 2017)) and *off-policy* (e.g., Fujimoto et al. (2018); Haarnoja et al. (2018)) algorithms (see Figure 2.2). In on-policy algorithms, actor and critic are updated only using data collected from



(a) On-policy RL algorithms **(b)** Off-policy RL algorithms

Figure 2.2.: Comparison of on-policy and off-policy RL algorithms. Reproduced from Levine et al. (2020).

the current policy. Thus, they require continuous interaction with the environment for learning. Off-policy methods can learn from data generated by policies other than the current policy. Thus, experiences can be stored in a replay buffer and reused throughout training. Therefore, off-policy methods are generally more sample efficient than on-policy methods. However, the "deadly triad" combination (Sutton and Barto (2018)) of function approximation, bootstrapping, and off-policy updates makes modern actor-critic off-policy algorithms unstable. Therefore, on-policy methods can be more suitable if interacting with the environment to get new samples under the current policy is comparatively cheap, e.g., when training in a simulation.

2.3.1. On-policy actor-critic algorithms

Between each update to the probabilistic actor and the critic, on-policy algorithms generate data by running the current policy in the environment for N steps. The critic $V_{\boldsymbol{\phi}}(\boldsymbol{x})\colon \mathbb{R}^n \mapsto \mathbb{R}$ with trainable parameters $\boldsymbol{\phi}$ is trained in a supervised manner with the loss being the mean squared Monte Carlo error

$$L_{\text{critic}}(\boldsymbol{\phi}) = \frac{1}{N}\sum_{t=1}^{N}(G_t - V_{\boldsymbol{\phi}}(\boldsymbol{x}_t))^2. \tag{2.3}$$

Herein, G_t represents the discounted cumulative reward received after t during the same episode

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \cdots + \gamma^{T-t} r_T, \tag{2.4}$$

where γ is the discount factor and T is the episode length. Thus, the critic is trained to approximate the expected cumulative future reward for any given state, given the current policy. The critic serves the training procedure of the policy by providing a measure for the expected improvement of taking a specific action $\boldsymbol{u}_t$ in state $\boldsymbol{x}_t$ over taking the average action of the current policy in that state. This measure is referred to as the advantage $A(\boldsymbol{x}_t, \boldsymbol{u}_t)$. Schulman et al. (2015b) thoroughly discuss methods for using a critic to approximate the advantage.

Let

$$p_t(\boldsymbol{\theta}) = \frac{\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}{\boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{u}_t|\boldsymbol{x}_t)} \tag{2.5}$$

be the probability ratio of taking action $\boldsymbol{u}_t$ at state $\boldsymbol{x}_t$ with a new policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, compared to an old policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}$, so $p(\boldsymbol{\theta}_{\text{old}}) = 1$. Given an advantage estimate $\hat{A}_t$, most actor-critic methods then aim to maximize

$$L(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\left[\frac{\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}{\boldsymbol{\pi}_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{u}_t|\boldsymbol{x}_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[p_t(\boldsymbol{\theta})\hat{A}_t\right]. \tag{2.6}$$

This loss has an intuitive appeal because it results in gradient steps in parameter space that most increase the probability of taking $\boldsymbol{u}_t$ at $\boldsymbol{x}_t$, scaled by the advantage, i.e., if the advantage is negative this probability will be decreased. However, maximizing (2.6) can lead to destructively large policy changes, especially when taking multiple consecutive gradient steps without generating new data with the updated policy.

Today, due to its favorable balance between performance, ease of implementation, and ease of tuning, Proximal Policy Optimization (PPO) (Schulman et al. (2017)) is the most

commonly used on-policy actor-critic method. The key idea of PPO is to modify the loss function (2.6) in a way that removes the incentive to modify the policy too much, i.e., to move $p_t(\boldsymbol{\theta})$ outside of the interval $[1-\epsilon, 1+\epsilon]$, where ϵ is a hyperparameter:

$$L_{\text{actor}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t\big[\min(p_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(p_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t)\big]. \tag{2.7}$$

Using (2.7) as the actor loss function allows PPO to perform multiple optimization steps for each batch of collected experiences, while preventing destructively large changes to the policy. Thereby, PPO improves its sample efficiency compared to other on-policy actor-critic methods, while maintaining stable learning progress, thus making it applicable to various tasks.

2.3.2. Dyna-style model-based RL

Model-based RL (MBRL) (Moerland et al. (2023)) algorithms are designed to increase the sample efficiency of RL by concurrently learning a policy and a model 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 Figure 2.3):

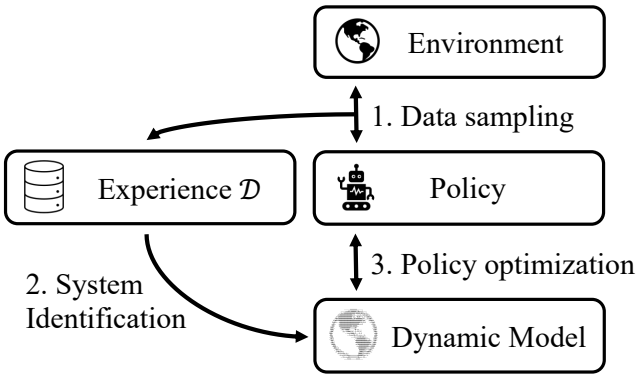


Figure 2.3.: Dyna-style (Sutton (1991)) model-based RL framework. The three steps are repeated for a predefined number of steps, or until satisfactory control performance is reached.

(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); 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 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 Figure 2.3): (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 (Figure 2.3, 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. Figure 2.3) 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); Haarnoja et al. (2018)).

2.3.3. Short-Horizon Actor-Critic algorithm

RL algorithms are a class of policy optimization algorithms that enable learning control policies through trial-and-error actuation of real-world or simulated environments (Mahmood et al. (2018)). Standard RL algorithms view environments as black boxes and do not use derivative information regarding the environment dynamics or the reward signals, even though many RL publications use simulated environments where analytical gradients of dynamics and rewards could be available. Policy gradient algorithms are the most commonly used class of RL algorithms for end-to-end learning of dynamic models for control, see, e.g., (Chen et al. (2019); Gros and Zanon (2019)). However, fundamental issues arise from the fact that policy gradient algorithms do *not* leverage analytical gradients from the environment. These issues concern both our understanding of the algorithms' empirical behavior (Ilyas et al. (2018); Wu et al. (2022)) and their performance (Islam et al. (2017); Henderson et al. (2018a,b)) (see Section 3.4).

SHAC (Xu et al. (2022)) is a policy optimization algorithm that uses derivative information from a differentiable simulation environment, i.e., an environment wherein the transition

function and the reward function (see Section 2.1) are known differentiable functions, to train control policies. Via concatenation using the chain rule, gradients can be propagated through an entire episode (Werbos (1990)), thus enabling policy optimization via gradient ascent on returns if the policy is also differentiable. This approach results in the BPTT algorithm for policy optimization, upon which SHAC is based. The concatenation of many backpropagation steps through a differentiable transition function and a differentiable policy, however, leads to exploding/vanishing gradients and to a noisy optimization landscape with many local optima (Xu et al. (2022)), making BPTT unsuitable for long-horizon tasks. SHAC addresses those challenges by shortening the learning horizon, i.e., instead of backpropagating gradients through an entire episode, the episode is split into short-horizon subepisodes for learning. Gradients are not allowed to flow from one subepisode to another. To ensure that the training results in a farsighted policy despite training with short-horizon subepisodes, SHAC uses a value function (critic) $V_{\boldsymbol{\phi}}$ with learnable parameters $\boldsymbol{\phi}$ to estimate future rewards based on the last state of a subepisode. The critic also serves to smooth the rugged reward landscape observed in BPTT. In practice, multiple environment instances can be run in parallel to perform batch updates on the policy and the value function (Xu et al. (2022)).

For N trajectories being sampled in parallel from different environment instances and a subepisode length of h, the policy loss is calculated as

$$\mathcal{L}_\theta = -\frac{1}{Nh}\sum_{i=1}^{N}\left[\Big(\sum_{t=t_0}^{t_0+h-1}\gamma^{t-t_0}r_t^i\Big) + \gamma^h V_\phi(\boldsymbol{x}_{t_0+h}^i)\right].$$

Herein, t_0 is the time index of the start of the current subepisode and $\boldsymbol{x}_{t_0+h}^i$ is the final state of the i-th trajectory at the current subepisode.

Using the same trajectories, the value function loss can be calculated as

$$\mathcal{L}_\phi = \|V_\phi(\boldsymbol{x}_t^i) - \tilde{V}(\boldsymbol{x}_t^i)\|_2.$$

Herein, $\tilde{V}(\boldsymbol{x}_t^i)$ is the estimated value of $\boldsymbol{x}_t^i$ which is calculated via a td-λ approach (Sutton and Barto (2018)) using the current critic and the data of the current subepisode.

2.4. Koopman theory for control

In 1931, Koopman showed that the behavior of any continuous nonlinear dynamical system can be described by the infinite-dimensional linear Koopman operator K, acting on the space of all possible measurement functions of the system state (Koopman (1931)). Given an autoregressive discrete-time nonlinear dynamical system $\boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t)$, the Koopman operator advances the vector of possible measurement functions $\boldsymbol{g}$ of $\boldsymbol{x}$ in time:

$$\boldsymbol{g}(\boldsymbol{x}_{t+1}) = \mathrm{K}\,\boldsymbol{g}(\boldsymbol{x}_t). \tag{2.8}$$

Since identifying and using an infinite-dimensional linear operator is intractable, learning finite-dimensional matrix approximations of K and the associated sets of measurement functions has been the focus of intense research efforts (Brunton et al. (2021)). Deep learning is a promising and straightforward tool to automatically discover appropriate measurement functions that transform the system state $\boldsymbol{x}$ into a (typically higher dimensional)

latent space $\boldsymbol{z}$ with approximately linear dynamics (Lusch et al. (2018)). Autoencoders, in particular, are well-suited to learning a nonlinear coordinate transformation together with its inverse.

Multiple ways to extend Koopman theory to controlled systems for use in MPC applications have been proposed (e.g., Proctor et al. (2018); Williams et al. (2016); Korda and Mezić (2018)). We use the approach by Korda and Mezić (2018) as it results in convex OCPs, under the easily verifiable condition that the objective function and inequality constraint functions are convex. The resulting models are of the form

$$\boldsymbol{z}_0 = \boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_0), \tag{2.9a}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{A}_{\boldsymbol{\theta}}\boldsymbol{z}_t + \boldsymbol{B}_{\boldsymbol{\theta}}\boldsymbol{u}_t, \tag{2.9b}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{C}_{\boldsymbol{\theta}}\boldsymbol{z}_t, \tag{2.9c}$$

where $\boldsymbol{z}_t \in \mathbb{R}^N$ is the lifted latent space vector, $\hat{\boldsymbol{x}}_t$ is the prediction of the states $\boldsymbol{x}_t \in \mathbb{R}^n$, and $\boldsymbol{u}_t \in \mathbb{R}^m$ is the control input vector. $\boldsymbol{A}_{\boldsymbol{\theta}} \in \mathbb{R}^{N\times N}$ and $\boldsymbol{B}_{\boldsymbol{\theta}} \in \mathbb{R}^{N\times m}$ together form an approximation to the Koopman operator for an actuated system, and $\boldsymbol{C}_{\boldsymbol{\theta}} \in \mathbb{R}^{n\times N}$ is a linear decoder. The initial condition of the predictor is obtained by passing the initial condition $\boldsymbol{x}_0$ through a nonlinear encoder $\boldsymbol{\psi}_{\boldsymbol{\theta}} \colon \mathbb{R}^n \mapsto \mathbb{R}^N$, where typically $N \gg n$.

All model components can be learned from data by adjusting the model parameters $\boldsymbol{\theta}$. Lusch et al. (2018) identify three requirements for autoregressive Koopman models. When performing SI, these requirements result in a training loss function that is a weighted combination of three loss terms. The extension to controlled systems and the model structure by Korda and Mezić (2018) results in the following three loss terms:

1. An autoencoder loss term that promotes the identification of nonlinear lifting functions $\boldsymbol{\psi}_{\boldsymbol{\theta}}$, which allow for a linear reconstruction of the system state through $\boldsymbol{C}_{\boldsymbol{\theta}}$:

$$||\boldsymbol{C}_{\boldsymbol{\theta}}\boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_t) - \boldsymbol{x}_t||_2 \tag{2.10}$$

2. A prediction loss term for the identification of the linear latent space dynamics:

$$||\boldsymbol{A}_{\boldsymbol{\theta}}\boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_t) + \boldsymbol{B}_{\boldsymbol{\theta}}\boldsymbol{u}_t - \boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_{t+1})||_2 \tag{2.11}$$

3. A loss term combining all elements of the model for the prediction of the original system state:

$$||\boldsymbol{C}_{\boldsymbol{\theta}}(\boldsymbol{A}_{\boldsymbol{\theta}}\boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_t) + \boldsymbol{B}_{\boldsymbol{\theta}}\boldsymbol{u}_t) - \boldsymbol{x}_{t+1}||_2 \tag{2.12}$$

When using models of the form given by Equations (2.9a) - (2.9c) as part of (e)NMPC, the nonlinear encoding of the initial state into the higher dimensional Koopman space can be performed before solving the OCP and, therefore, outside of the scope of the OCP optimizer (Korda and Mezić (2018)). Given an initial Koopman state $\boldsymbol{z}_0 = \boldsymbol{\psi}_{\boldsymbol{\theta}}(\boldsymbol{x}_0)$, the OCP optimizer sees only the linear components of the Koopman model, i.e., $\boldsymbol{A}_{\boldsymbol{\theta}}$ and $\boldsymbol{B}_{\boldsymbol{\theta}}$ for the dynamics, and $\boldsymbol{C}_{\boldsymbol{\theta}}$ to evaluate the objective function or constraints in case they depend on state variables. Thus, all equality constraints imposed by the Koopman model are linear. Therefore, the resulting OCPs are convex if the objective function and constraints on control and system state variables are convex. This unique combination of nonlinear model dynamics and convex OCPs, when using the model in (e)NMPC, makes Koopman models promising candidates for control problems where (e)MPC does not deliver satisfactory control performance, while alternative (e)NMPC approaches may not be real-time capable.

2.5. Post-optimal sensitivity analysis of convex optimization problems

Obtaining post-optimal sensitivities of solution mappings of optimization problems has been a topic of study for several decades (e.g., Fiacco and Kyparisis (1985); Ralph and Dempe (1995); Agrawal et al. (2019b)). Given a parametric nonlinear program with parameters $\boldsymbol{\theta}$, and a solution point $\boldsymbol{a}^*$ for a given $\boldsymbol{\theta}^*$, the solution point is differentiable with respect to the parameters in a neighborhood of $\boldsymbol{\theta}^*$ under the following assumptions (Pistikopoulos et al. (2020); Fiacco and Ishizuka (1990)): (i) The objective function and constraints of the problem are twice continuously differentiable in $\boldsymbol{a}$. (ii) The gradients of objective function and constraints with respect to $\boldsymbol{a}$ and the constraints are once continuously differentiable in $\boldsymbol{\theta}$. (iii) The second-order sufficient conditions (SOSC) for a local minimum hold at $\boldsymbol{a}^*$. (iv) The linear independence constraint qualification (LICQ) is satisfied at $\boldsymbol{a}^*$. (v) Strict complementary slackness (SCS) holds at $\boldsymbol{a}^*$. Note that the above conditions also imply that the first-order KKT conditions hold (Karush (1939); Boyd et al. (2004)), and that $\boldsymbol{a}^*$ is an isolated local minimum (Fiacco and Ishizuka (1990)). For a convex parametric problem, if the objective is strictly convex in a neighborhood of $\boldsymbol{a}^*$, this means that the solution map of the optimization problem becomes single-valued, i.e., the solution map is locally an implicit function of the problem parameters.

A straightforward but costly way to obtain the derivatives of a solution point $\boldsymbol{a}^*$ that satisfies conditions (i-v) with respect to the parameters $\boldsymbol{\theta}$ is to unroll the computational graph of an iterative optimization algorithm and backpropagate through every step of the entire algorithm using algorithmic differentiation (e.g., Domke (2012)). Alternatively, it is possible to apply the implicit function theorem (Nocedal and Wright (1999)) to the first-order KKT conditions of the optimization problem to cheaply obtain gradients (e.g., Fiacco and Ishizuka (1990)). Using the implicit function theorem, Amos and Kolter (2017) and Agrawal et al. (2019a) connected solvers for quadratic and convex problems, respectively, to PyTorch (Paszke et al. (2019)) and Tensorflow (Abadi et al. (2016)). In doing so, they connected the most popular deep learning frameworks with established techniques from the field of parametric optimization. Thereby, they lowered the required effort to use post-optimal sensitivity analysis in a deep learning project substantially.

2.6. Physics-informed neural networks

Physics-Informed Neural Networks (PINNs) (Raissi et al. (2019)) are a class of NNs that integrate physical laws, expressed as partial differential equations (PDEs), into the training process. In their original formulation (Raissi et al. (2019)), PINNs take spatiotemporal coordinates as the input and predict the corresponding system states. By embedding the known dynamics as soft constraints in the loss function, PINNs ensure that predictions adhere to physical laws while simultaneously learning unknown dynamics from data. Usually, derivatives of system states with respect to the spatiotemporal coordinates have to be calculated to compute the physics-based loss term, which is done via automatic differentiation (Naumann (2011)). This hybrid approach is particularly useful in scenarios where (i) only partial knowledge of the system dynamics is available and (ii) data is scarce or expensive to obtain.

In process control, PINNs can be employed to model system dynamics by incorporating first-principles knowledge (e.g., Patel et al. (2024)), thereby reducing the reliance on large labeled datasets compared to purely data-driven models. However, in their original formulation (Raissi et al. (2019)), PINNs are not suitable for control applications since they do not account for varying initial states and control inputs. Antonelo et al. (2024) extended PINNs to control applications by adding the initial system states and control variables to the model inputs.

3. Related work

This chapter discusses prior work that is related to this thesis but does not directly enable our work. Thus, this chapter is intended to enable the reader to situate our work in a wider research context.

3.1. Online tuning of model-based controllers

Chemical process operation has long relied on model- and optimization-based controllers, such as plant scheduling (Kondili et al. (1993); Bassett et al. (1996)), Real-Time Optimization (Marlin et al. (1997)), and MPC (Qin and Badgwell (2003)). These methods rely on predicting future plant behavior based on mathematical models and selecting control actions by solving optimization problems that minimize an economic or operational objective subject to process constraints. However, any mathematical model – mechanistic or data-driven – is inherently an approximation of the actual plant. Various sources of plant-model mismatch exist, including structural discrepancies between the real plant and the model structure, uncertainties in model parameters, and external disturbances affecting the plant. Such mismatches lead to suboptimal plant operation, motivating strategies to correct or mitigate their effects.

Several strategies have been proposed to handle plant-model mismatch. A straightforward approach is to iteratively re-identify the model parameters from real-time operational data (Jang et al. (1987); Chen and Joseph (1987)). As more data becomes available through ongoing interactions between the controller and the plant, model accuracy can progressively improve. While conceptually simple, continuous model re-identification often does not produce a model-based controller whose optimal control inputs are also optimal for the true plant (Marchetti et al. (2009)). Identifying such a model is especially difficult if (i) the controller does not lead to sufficient excitation of the system to estimate the uncertain model parameters from the resulting operating data, or if (ii) a large structural mismatch between the model and the plant makes finding "true" model parameters impossible (Marchetti et al. (2009)).

Forbes et al. (1994) state the requirements that a model has to fulfill so that a model-based controller deploying that model produces plant-optimal behavior. Due to the problems associated with model re-identification aiming for predictive accuracy, methods were developed that aim to adapt the model so that the necessary optimality conditions of the resulting model-based controller align with those of the real plant. A noteworthy algorithm is the Integrated System Optimization and Parameter Estimation (ISOPE) algorithm (Roberts (1979); Marchetti et al. (2016)). ISOPE requires measurements of the plant's outputs as well as estimates of the gradients of the plant outputs with respect to the inputs. As an alternative to re-identifying the model, parameters within the controller's objective function or constraints can be tuned in order to match the necessary optimality

conditions between the controller and plant while keeping the imperfect model fixed. One method originating from this idea is Modifier Adaptation (MA) (Marchetti et al. (2016)). The advantage of MA compared to methods that re-identify the model is its ability to steer the controller towards the plant optimum despite structural mismatch between the plant and the model.

From a high-level perspective, ISOPE and MA share conceptual similarities with actor-critic policy gradient methods from RL (Sutton et al. (1999); Sutton and Barto (2018)), as all these approaches iteratively refine control policies based on gradient information derived from real-world data. Nevertheless, there are methodological differences: Policy gradient methods leverage the policy gradient theorem (see Section 2.3) to estimate the gradients of the policy's parameters with respect to a reward function whose maximization is expected to yield optimal controller behavior. Since the reward function can be any function defined by the user, policy gradient methods are applicable to arbitrary MDP problems. Policy gradient methods are typically used to optimize high-dimensional black-box policies like artificial neural networks. In contrast, MA and ISOPE minimize discrepancies between predicted gradients and gradients estimated from plant measurements by optimizing a limited number of structured parameters of a policy (Marchetti et al. (2016)). Moreover, MA and ISOPE typically address steady-state optimization problems with applications including CSTRs (e.g., Marchetti et al. (2010)) and distillation columns (e.g., Rodriguez-Blanco et al. (2015)).

As mentioned above, there are conceptual similarities between certain RL methods and MA and ISOPE. In the context of this thesis, these similarities are especially obvious since we use RL methods to optimize (e)NMPC policies by adapting the model (see Chapters 4, 6, 5) as is done in ISOPE, or the constraint functions (see Chapter 7, 5) as in MA. However, the methods used in this thesis are more flexible than MA or ISOPE since they can be applied to any policy (model-based or model-free) whose outputs can be differentiated with respect to the policy's parameters and to any MDP problem for which a suitable reward function can be specified.

3.2. Combining MPC and RL

Data-driven MPC and RL are distinct solution approaches for MDPs developed by separate research communities. Both methodologies are widely used in real-world control tasks (Schwenzer et al. (2021), Dogru et al. (2024)). Recently, economic MPC has gained increased attention (e.g., Angeli et al. (2011)), expanding the applicability of MPC to a broader range of MDPs and bridging the gap between the two communities. Some researchers even consider data-driven MPC with online model learning as a variant of model-based RL (Wang et al. (2019)): While other model-based RL approaches learn an environment model and use it offline to optimize a policy, data-driven MPC derives a policy directly from the learned model by incorporating it into online OCPs.

Reiter et al. (2025) and Görges (2017) highlight that the strengths and weaknesses of RL and data-driven MPC often contrast sharply. For instance, model-free RL is generally applicable only in scenarios with abundant training data, while data-driven MPC is effective in situations where data is limited or expensive. Additionally, MPC offers strong theoretical foundations regarding stability and uncertainty, naturally integrates constraints, and

can adapt to changes in the control environment that do not affect the dynamics without the need for retraining. RL lacks all of these properties. On the other hand, RL does not require solving OCPs during policy inference, making it more scalable for problems where the computational cost of MPC would be prohibitively high.

Due to the complementary strengths and weaknesses of RL and MPC, recently, a lot of work has gone into trying to combine both methodologies. The methods developed in this thesis (Chapters 4-7) are a part of this larger effort. Reiter et al. (2025) provide a thorough review and organize contributions that aim to synthesize RL and MPC into three categories:

(i) MPC can be used as an expert actor during policy training. To this end, an imitation learning algorithm based on supervised learning can be used to train the policy to mimic the behavior of the MPC (e.g., Åkesson and Toivonen (2006); Hertneck et al. (2018); Karg and Lucia (2021); Lüken et al. (2023)). Alternatively, the MPC can be used to guide the exploration process of the policy during training (e.g., Mordatch and Todorov (2014); Levine et al. (2016); Schulz et al. (2024)).

(ii) MPC can be used as part of the critic in an actor-critic RL or imitation learning algorithms. Ghezzi et al. (2023) propose using a fixed MPC as the critic for training a neural network policy. Assuming sufficient representational capacity of the policy and good convergence, this approach ultimately results in a policy that matches the performance of the MPC. A parameterized MPC can also be used as a learnable critic (e.g., Bhardwaj et al. (2020)). Such an approach can provide a good initial guess for the critic based on prior knowledge about the system while retaining the ability to learn from experience.

(iii) MPC can be used as (a part of) the policy. Many contributions that fall into this category use RL to optimize a parameterized and automatically differentiable MPC policy (e.g., Gros and Zanon (2019)). The original research conducted for this thesis (Chapters 4 - 5) is part of this bigger research effort. The parameters that are optimized via RL may be part of the model, or they may appear in the objective function or constraints of the MPC. Learning a highly parameterized dynamic model via RL offers great flexibility and may result in exceptional control performance (see Chapters 6, 5). However, this approach leads to policy optimization problems with many variables, which may result in bad convergence (see Chapter 4). Alternatively, optimizing the (e)NMPC via parameters in the objective function or constraints may be advantageous since the behavior of the (e)NMPC policy can be tuned by optimizing a very small number of learnable parameters, which makes policy optimization easier (see Chapter 7). Lewis and Vrabie (2009) and Lewis et al. (2012) first show that RL can be used to learn goal-directed dynamic models by connecting RL to the linear quadratic regulator (Brunton and Kutz (2022)), a model predictive feedback controller for control settings with linear dynamics and quadratic cost functions. Amos et al. (2018) and Gros and Zanon (2019) are the first to learn (e)MPC controllers end-to-end by viewing these as differentiable policies. Gros and Zanon (2019) use RL in the form of deep Q-learning (Sutton and Barto (2018)), a popular RL algorithm for environments with discrete action spaces and the first RL algorithm to claim superhuman performance on an Atari game (Mnih et al. (2013)), to learn linear MPC and eNMPC controllers. In the nonlinear case, they focus on learning advantageous parameterizations of the constraints and cost function and only parameterize the dynamic model with a learnable constant added to the transition function. Amos et al. (2018) use imitation learning (Hussein et al. (2017)), a supervised learning approach aiming to train an NMPC policy

through expert demonstrations. Amos et al. (2018) only consider control settings without inequality constraints on state variables. Moreover, they learn a parameterized version of the true underlying dynamic equations, which requires an a priori known structure of the system dynamics. Chen et al. (2019) use both imitation learning and RL together with the differentiable MPC solver by Amos et al. (2018) to learn task-optimal linear dynamic models for HVAC control without constraints on state variables. Brandner et al. (2023) use RL to optimize parameters that appear in the cost function and constraints of an MCP that is used to operate a flash separation unit. Another idea that is also part of category (iii) – using an MPC as part of the policy – but which does not necessarily entail online learning of a parameterized MPC policy, is to use MPC as a filter for control actions proposed by a neural network policy, e.g., to increase safety during exploration (e.g., Dalal et al. (2018); Wabersich and Zeilinger (2021)).

We refer the reader to Reiter et al. (2025) for a more detailed comparison and analysis of the potentials for the synthesis of MPC and RL. Mesbah et al. (2022) provide a complementary perspective, analyzing possibilities for integrating machine learning into MPC, with a particular focus on uncertainty-aware control.

3.3. Applications of Koopman (e)NMPC

Koopman-based MPCs have seen application in various domains due to their favorable balance between high representational capacity and comparatively low computational demand in solving the resulting OCPs. These applications range from ground vehicle (Cibulka et al. (2020)) to air vehicle control (Folkestad and Burdick (2021)), flow control (Arbabi et al. (2018)), and soft robotics (Bruder et al. (2019)). Some contributions have also studied the application of Koopman-based MPCs to the control of chemical processes. Narasingam and Kwon (2019) integrate Koopman models with Lyapunov-based MPC, thus guaranteeing controller feasibility and closed-loop stability. Narasingam and Kwon (2020) integrate a priori system knowledge in the choice of the Koopman basis functions and achieve successful data-driven control of a hydraulic fracturing process. Using a learned disturbance estimator, Son et al. (2022b) develop offset-free Koopman Lyapunov-based MPC to compensate for the unavoidable plant-model-mismatch resulting from a finite-dimensional approximation of the Koopman operator. They showcase their approach on the control of a batch pulp digester (Son et al. (2021)). For hybrid systems with differing local dynamics, Son et al. (2022a) develop a hybrid Koopman MPC approach by clustering the training data and by training one local Koopman model for each cluster. Furthermore, Albalawi and Hameed (2023) demonstrate the efficacy of Koopman-based economic MPC on a continuous stirred-tank reactor (CSTR) case study.

Recent contributions have also explored combining Koopman-based MPC and RL. Iwata and Kawahara (2022) and Yin et al. (2022) both use RL for end-to-end learning of task-optimal Koopman models to obtain gain matrices that act on the Koopman invariant subspaces, minimizing the squared distance of the system state from a desired target state. However, similarly to linear-quadratic regulator (LQR) control (Kalman et al. (1960)), this approach cannot easily be extended to (e)NMPC applications or applications with hard bounds on state variables. Weissenbacher et al. (2022) use Koopman models in the context of offline RL (Levine et al. (2020)), i.e., RL based on a static data set without the ability to

interact with the environment, to learn a latent representation of the system's dynamics, enabling the discovery of symmetries that can be used to augment the dataset. Thereby, they improve the generalization of neural network policies to out-of-distribution states. Rozwood et al. (2024) utilize Koopman theory to linearly approximate the evolution of the value function of a given MDP. They integrate this approximation into two well-known entropy-maximizing RL algorithms (Haarnoja et al. (2018); Hazan et al. (2019)), thereby increasing the interpretability of the resulting neural network policies while retaining the performance of the baseline algorithms.

3.4. Motivation for differentiable RL environments

Various state-of-the-art actor-critic algorithms (e.g., Schulman et al. (2015a, 2017); Haarnoja et al. (2018)) are based on the policy gradient theorem (Sutton et al. (1999); Sutton and Barto (2018)). Algorithms that utilize the policy gradient theorem do not rely on differentiable environment dynamics, treating the environment as a non-differentiable black box. Consequently, they estimate policy gradients using sampled trajectories rather than utilizing analytical gradients. While such algorithms have yielded impressive empirical results (e.g., Hwangbo et al. (2019)), they remain poorly understood (Ilyas et al. (2018); Wu et al. (2022)). Ilyas et al. (2018) demonstrate that the empirical behavior of these algorithms often does not reflect the ideas that motivated their development. They detect poor gradient estimates across different environments when using state-of-the-art policy gradient algorithms. They also show that gradient estimates decay with rising task complexity. Additionally, they uncover a problem that arises specifically in modern *on-policy* policy optimization algorithms: The state-of-the-art algorithms Trust Region Policy Optimization (TRPO) (Schulman et al. (2015a)) and Proximal Policy Optimization (PPO) (Schulman et al. (2017)) use a surrogate objective function for policy optimization, constraining the size of each policy update, thus mitigating the risk of catastrophic policy updates even when performing multiple consecutive updates per data sample. However, Ilyas et al. (2018) show that the alignment of the surrogate objective function and the true reward landscape consistently worsens across different environments as training progresses. Poor gradient estimation and a surrogate objective function that does not match the true reward landscape can slow down learning and reduce the terminal performance of the learned policies. Other publications call the effectiveness of modern policy gradient algorithms for continuous control tasks into question: Henderson et al. (2018b) report narrow effective hyperparameter windows of standard numerical optimizers when used in policy gradient algorithms. Islam et al. (2017) and Henderson et al. (2018a) show that the performance of trained policies on continuous control benchmark problems can vary drastically along different benchmark implementations of the same policy gradient algorithms. Mania et al. (2018) develop a simple random search algorithm acting on the parameter space of linear policies that can compete with modern policy optimization algorithms on continuous control benchmark problems.

Differentiable simulators (e.g., Chen et al. (2018); Rackauckas et al. (2019); Lienen and Günnemann (2022)) can be used to construct simulated RL environments with automatically differentiable dynamics and reward functions, thus enabling the use of analytic gradients for policy optimization. Backpropagation Through Time (BPTT) (Werbos (1990)) has been used in conjunction with differentiable simulators to train controllers (e.g., Hu

et al. (2019); Huang et al. (2021)); however, applying BPTT to long-horizon and multi-stage tasks is impractical due to noisy optimization landscapes and exploding/vanishing gradients (Huang et al. (2021); Xu et al. (2022)). Recently, Mora et al. (2021) and Xu et al. (2022) designed policy optimization algorithms that avoid the well-known problems of BPTT while still making use of gradient information from the environment. Their algorithms exhibit increased training wall-clock time efficiency and terminal performance compared to state-of-the-art actor-critic policy gradient algorithms. However, the Policy Optimization via Differentiable Simulation (PODS) algorithm by Mora et al. (2021) requires the Hessian of the simulation, whereas the Short-Horizon Actor-Critic (SHAC) algorithm by Xu et al. (2022) only requires first-order gradients, making the latter computationally cheaper. Moreover, PODS is designed for monotonic policy improvement, thereby excelling in exploitation without any exploration. Therefore, PODS would need to be combined with other RL methods that feature exploration to make it applicable to tasks that feature local optima in policy space. SHAC features exploration via a stochastic actor, as is common in actor-critic policy gradient algorithms, and is scalable to high-dimensional control problems with local policy optima.

These developments show that incorporating gradient information from differentiable environments can enhance policy optimization compared to state-of-the-art actor-critic RL algorithms, as demonstrated for NN policies. There is no inherent reason why similar benefits should not extend to RL-based learning of (e)NMPC policies.

4. RL-based end-to-end learning of Koopman (e)NMPCs

This chapter presents a method for end-to-end RL of Koopman models for optimal performance in (e)NMPC applications with hard constraints on states. Using the *cvxpylayers* package (Agrawal et al. (2019a)) to construct differentiable Koopman-MPC policies, we use Proximal Policy Optimization (PPO) (Schulman et al. (2017)), a prominent RL algorithm for tasks with continuous action spaces, to learn task-optimal Koopman models. We test our approach on two distinct case studies derived from a continuous stirred-tank reactor model from the literature (Flores-Tlacuahuac and Grossmann (2006); Du et al. (2015)): (i) an NMPC case study, wherein the controller objective is to stabilize the product concentration in the presence of an externally varying product flow rate, and (ii) a demand response case study, in which eNMPC seeks to minimize electricity costs without violating constraints on state variables. We assess the resulting control performance by comparing it to that of dynamic models trained exclusively with SI as well as model-free policies trained by RL.

Our findings demonstrate that end-to-end learning consistently outperforms SI in learning data-driven models of the mechanistic CSTR model in our control applications. Additionally, we investigate to what extent the learned models and policies still function in the case of modified constraints. Such a change in the control setting is an example of so-called *distribution shift* (Zhang et al. (2021a)), which is a major challenge in machine learning applications (Quinonero-Candela et al. (2008); Zhang et al. (2021a); Yao et al. (2022)). We show that, unlike model-free policies, MPCs employing end-to-end learned dynamic surrogate models may successfully adapt to such a change in the control setting without a need for retraining. To our knowledge, this work is the first to connect end-to-end learning of Koopman models to control applications with constraints on system variables and the first to study the ability of end-to-end learned dynamic models to adapt to a distribution shift in control.

The remainder of this chapter is organized as follows: Section 4.1 presents our method, Section 4.2 provides numerical examples on a case study, and Section 4.3 discusses the conclusions and directions for future work.

4.1. Method

Figure 4.1 summarizes our proposed workflow for training task-optimal Koopman surrogate models. We start with the standard SI procedure for dynamic surrogate models, given a mechanistic model: First, we generate a data set $\mathcal{D}$ from which the system dynamics can be learned using supervised learning, by (randomly) actuating the mechanistic model. Second, we fit a dynamic surrogate model, in our case a Koopman model with learnable

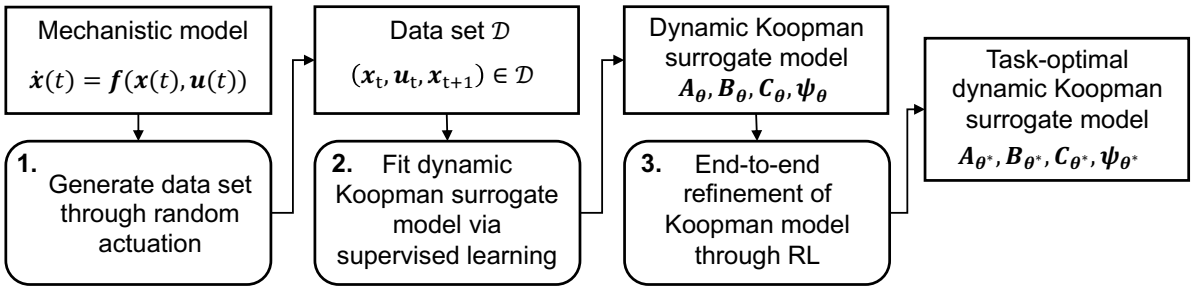


Figure 4.1.: Workflow from mechanistic model to task-optimal dynamic Koopman surrogate model. The mechanistic model is a continuous-time model, whereas the Koopman model is a discrete-time model. In principle, this workflow would also be applicable to learning a task-optimal Koopman model directly via interactions with the real process instead of with a mechanistic model.

parameters $\boldsymbol{\theta}$, to the data. As described in Section 2.4, our Koopman models consist of a nonlinear encoder $\boldsymbol{\psi_\theta}$, the matrices $\boldsymbol{A_\theta}$ and $\boldsymbol{B_\theta}$ that represent the linear dynamics inside the Koopman space, and a decoder matrix $\boldsymbol{C_\theta}$. We perform SI by minimizing the sum of the three loss functions that correspond to the three fundamental requirements that a Koopman model should fulfill (see Section 2.4): reconstruction of states through the autoencoder part of the model ((2.10)), linear prediction of the dynamics in the latent space ((2.11)), and prediction of system states by combining all elements of the model ((2.12)). We use multi-step versions of the prediction loss (Equation 2.11) and the combined loss (Equation 2.12) to improve model accuracy when predicting over multiple time steps in closed-loop fashion. We follow standard SI practices, such as using a separate validation data set for early stopping and normalizing the inputs and outputs of the model.

The main contribution of this work is our method for the third and final step depicted in Figure 6.1, the end-to-end refinement of Koopman models for optimal performance as part of MPC in a specific control task. Figure 4.2 summarizes our approach towards end-to-end refinement of Koopman models for optimal performance in (e)NMPC applications through RL. We use the actor-critic PPO algorithm (Schulman et al. (2017)) to learn the optimal model parameters $\boldsymbol{\theta}$ for a given control task. Note that the refinement applies to all elements of the Koopman model, i.e., the predictor consisting of $\boldsymbol{A_\theta}$ and $\boldsymbol{B_\theta}$, and the autoencoder given by $\boldsymbol{\psi_\theta}$ and $\boldsymbol{C_\theta}$. Using the *cvxpylayers* package (Agrawal et al. (2019a)), we construct automatically differentiable stochastic Koopman-MPC policies $\boldsymbol{\pi_\theta}(\boldsymbol{u}_t|\boldsymbol{x}_t)\colon \mathbb{R}^n \mapsto \mathbb{R}^m$ that serve as actors. These policies take a state $\boldsymbol{x}_t$ as input and compute the Koopman state $\boldsymbol{z}_t$ through a multilayer perceptron encoder $\boldsymbol{\psi_\theta}$. Then, by evaluating objective function and constraints through the decoder, any (e)MPC problem with a convex objective function and convex constraints (in state space $\boldsymbol{x}$) can be formulated and solved using *cvxpylayers* to obtain the optimal control solution $\boldsymbol{u}_t^*$. As is common in applying MPC and suitable for an RL approach, our MPC policies are implemented using a receding horizon approach and only return the control solution for the first control step in the MPC horizon. Given the current time step t and an MPC horizon of $t_f + 1$ steps with the corresponding sets $\mathrm{T}_{+1} = \{t, \ldots, t + t_f\}$ and $\mathrm{T} = \{t, \ldots, t + t_f - 1\}$,

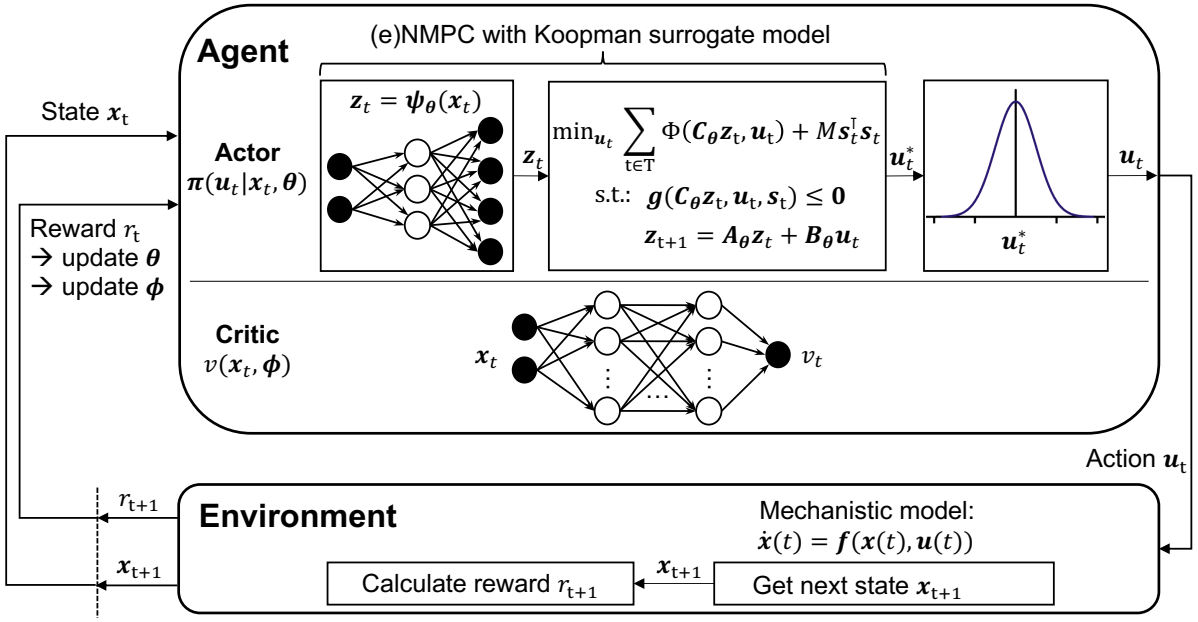


Figure 4.2.: Method for end-to-end refinement of dynamic Koopman surrogate model. The RL agent consists of a stochastic actor and a critic. The actor is an MPC policy utilizing a dynamic Koopman surrogate model. The critic is a feedforward neural network. The environment consists of the mechanistic model of the system that is to be controlled, and a reward function that depends upon the controllers task.

the OCPs that are solved inside the Koopman-MPC policies take the form

$$\min_{(\boldsymbol{u}_t)_{t\in \mathrm{T}}} \sum_{t\in \mathrm{T}_{+1}} \Phi(\boldsymbol{C_\theta} \boldsymbol{z}_t, \boldsymbol{u}_t) + M \boldsymbol{s}_t^\intercal \boldsymbol{s}_t, \tag{4.1a}$$

$$\text{s.t. } \boldsymbol{z}_{t+1} = \boldsymbol{A_\theta} \boldsymbol{z}_t + \boldsymbol{B_\theta} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{4.1b}$$

$$\underline{\boldsymbol{x}} - \boldsymbol{s}_t \le \boldsymbol{C_\theta} \boldsymbol{z}_t \le \bar{\boldsymbol{x}} + \boldsymbol{s}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{4.1c}$$

$$\underline{\boldsymbol{u}} \le \boldsymbol{u}_k \le \bar{\boldsymbol{u}} \quad \forall t \in \mathrm{T}, \tag{4.1d}$$

$$\boldsymbol{g}(\boldsymbol{C_\theta} \boldsymbol{z}_t, \boldsymbol{u}_t, \boldsymbol{s}_t) \le 0 \quad \forall t \in \mathrm{T}_{+1}. \tag{4.1e}$$

Both state variables $\boldsymbol{x}_t$ and control variables $\boldsymbol{u}_t$ can have upper and lower bounds (see Eqs. (4.1c) and (4.1d)). To guarantee the feasibility of the OCPs, we add slack variables $\boldsymbol{s}_t$ to the state bounds and penalize their use quadratically in the objective function (Eq. (4.1a)) with a penalty factor M. The stage cost Φ of the objective function can be any convex function and depends on the aim of the policy. The linear transition function in Koopman space is given by Equation (4.1b). Depending on the application, the OCPs can also include further convex inequality constraints (see Eq. (4.1e)).

As PPO is an on-policy RL method, the actor must be stochastic during training to ensure exploration. Therefore, during training, we sample the action that the actor will take from a normal distribution $\boldsymbol{u}_t \sim \boldsymbol{\mathcal{N}}(\boldsymbol{u}_t^*, \boldsymbol{\sigma}^2)$ with the mean $\boldsymbol{u}_t^*$ being the deterministic solution of the Koopman-MPC (see Figure 4.2). Depending on the application, it can be beneficial to use a constant standard deviation $\boldsymbol{\sigma}$, or start with a large standard deviation and gradually reduce the amount of exploration, e.g., by reducing $\boldsymbol{\sigma}$ linearly up to a specific value, as the policy becomes more competent (e.g., Chen et al. (2019)).

We use a critic in the form of a multilayer perceptron (MLP) with learnable parameters $\boldsymbol{\phi}$ and generalized advantage estimation (Schulman et al. (2015b)) to calculate the advantage estimates during training. These are used to compute the actor loss via the clipped PPO loss function and to update the actor parameters $\boldsymbol{\theta}$ accordingly (Schulman et al. (2017)). We also use global gradient clipping to stabilize the training, as described by Engstrom et al. (2020). We use the Adam optimizer (Kingma and Ba (2014)) to train both the actor and the critic.

We use the running average cumulative reward of a fixed number of preceding episodes as a performance metric during training. We choose the number of episodes to ensure a relatively small variance while limiting the influence of outdated policy parameters due to training updates (cf. Sections 4.2.3 and 4.2.4). At the end of a training run, we choose the agent that obtained the highest average cumulative reward as the final agent. We then evaluate the performance of that agent by testing it without any exploration, i.e., $\boldsymbol{u}_t = \boldsymbol{u}_t^*$.

Due to the convex nature of the OCPs, our Koopman-MPC policies are not only automatically differentiable but also computationally cheap compared to many alternatives that utilize general nonlinear dynamic models. We expect that Koopman-MPC policies are real-time feasible for a potentially vast range of systems.

4.2. Numerical experiments

4.2.1. Case study description

We base our numerical examples on a dimensionless benchmark continuous stirred-tank reactor (CSTR) model (Flores-Tlacuahuac and Grossmann (2006); Du et al. (2015)). The dynamics of the CSTR are given by two nonlinear ordinary differential equations:

$$\dot{c}(t) = (1 - c(t))\frac{\rho(t)}{V} - c(t)ke^{-\frac{N}{T(t)}}, \tag{4.2a}$$

$$\dot{T}(t) = (T_f - T(t))\frac{\rho(t)}{V} + c(t)ke^{-\frac{N}{T(t)}} - F(t)\alpha_c(T(t) - T_c) \tag{4.2b}$$

The system states $\boldsymbol{x}$ are the product concentration c and the temperature T. The control inputs $\boldsymbol{u}$ are the production rate ρ and the coolant flow rate F. All parameters are given in Table 4.1.

Table 4.1.: CSTR model parameters (Flores-Tlacuahuac and Grossmann (2006); Du et al. (2015))

	symbol	value
volume	V	20
reaction constant	k	$300\frac{1}{h}$
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

Based on the CSTR model, we construct both NMPC and eNMPC applications: In NMPC (Section 4.2.3), the controller shall stabilize the product concentration c and temperature T at given target values, given random perturbations to the production rate ρ. The goal in our eNMPC case (Section 4.2.4) is to optimize process economics while satisfying process constraints, given electricity price predictions. Table 4.2 presents the bounds of the system states and control inputs as well as the target values of the system states in NMPC.

Table 4.2.: Lower (lb) and upper (ub) bounds of system states and control inputs, target values of system states in NMPC, and steady-state (ss) values used to evaluate the economic benefit of flexible production in eNMPC.

	NMPC (Section 4.2.3)			eNMPC (Section 4.2.4)		
variable	lb	ub	target	lb	ub	ss
c	-	-	0.1367	0.1231	0.1504	0.1367
T	-	-	0.7293	0.6	0.8	0.7293
ρ	$0.8\frac{1}{h}$	$1.2\frac{1}{h}$	-	$0.8\frac{1}{h}$	$1.2\frac{1}{h}$	$1.0\frac{1}{h}$
F	$0.0\frac{1}{h}$	$700.0\frac{1}{h}$	-	$0.0\frac{1}{h}$	$700.0\frac{1}{h}$	$390.0\frac{1}{h}$

We use discrete time steps of length $\Delta t_{\text{discr}} = 15\,\text{min}$ and control steps of length $\Delta t_{\text{ctrl}} = 60\,\text{min}$. Given a current state $\boldsymbol{x}_t$ and control action $\boldsymbol{u}_t$, we use Equations (4.2a) - (4.2b) and the RK45 solver (Dormand and Prince (1980)) in SciPy (Virtanen et al. (2020)) to calculate the next state $\boldsymbol{x}_{t+1}$ in the environment.

We compare the performances of three different controller types:

1. *Koopman-SI*: MPC controller using a Koopman model to approximate the CSTR dynamics. The model is trained using SI on a data set generated using the mechanistic model (Equations (4.2a) - (4.2b)) and random control inputs. We provide a more detailed explanation of the data sampling and SI procedure in Subsection 4.2.2.
2. *Koopman-RL*: MPC controller using a Koopman model to approximate the CSTR dynamics. We use the model of the *Koopman-SI* controller as an initial guess and tune it for task-optimal MPC performance using the approach described in Section 4.1.
3. *MLP*: A model-free neural network policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)$ in the form of a multilayer perceptron (MLP), trained by RL. Similarly to the *Koopman-RL*, each action is sampled from a normal distribution $\boldsymbol{u}_t \sim \boldsymbol{\mathcal{N}}(\boldsymbol{u}_t^*, \boldsymbol{\sigma}^2)$, where the neural network produces $\boldsymbol{u}_t^*$ given the current state of the environment.

Matching our proposed workflow (see Figure 6.1), the remainder of this section is structured as follows: Subsection 4.2.2 elaborates on the data sampling procedure using the mechanistic model and discusses the subsequent SI process using the Koopman model architecture. Subsections 4.2.3 and 4.2.4 describe the end-to-end refinement of the Koopman models and report the performances of all three controller types for the NMPC and eNMPC applications, respectively. Finally, Subsection 4.2.5 presents our findings on a modified eNMPC, which we use to study the adaptability of end-to-end refined Koopman MPCs to modifications of constraints after training has been completed.

4.2.2. Data sampling and system identification

Given randomly sampled control inputs $\boldsymbol{u}$ within their respective bounds, the product concentration c typically quickly moves away from its feasible range in eNMPC and also away from its target value in NMPC (see Table 4.2). Therefore, such a sampling strategy would not produce a data set particularly suitable for learning a good dynamic surrogate model, as most data would be in irrelevant parts of the state space. Instead of actuating the mechanistic model by randomly sampling both control inputs ρ and F, we, therefore, take a different approach: We generate multiple trajectories using the mechanistic model, each trajectory starting from the target state of the NMPC, which is also at the center of the feasible range of the eNMPC. For each trajectory, we randomly generate a series of control inputs for either ρ or F. Given this series of inputs, we then solve an OCP using the mechanistic model to determine a corresponding series of values for the respective other control variable, which leads to c not diverging much from its feasible range in eNMPC. As in NMPC and eNMPC, we use control steps of length $\Delta t_{\text{ctrl}} = 60\,\text{min}$, i.e., ρ and F change every four time steps.

Given an OCP horizon of t_f time steps with the corresponding set $\mathrm{T} = \{0, \dots, t_f - 1\}$, and a randomly generated trajectory for the production rate $\rho_{\text{target},t} \forall t \in \mathrm{T}$, the OCP minimizes

$$\sum_{t \in \mathrm{T}} (c_{\text{target}} - c_t)^2 + w_\rho (\rho_{\text{target},t} - \rho_t)^2, \tag{4.3}$$

subject to the mechanistic model (Equations (4.2a) - (4.2b)), and subject to the given bounds on control variables (see Table 4.2), with c_{target} being the target value for c in NMPC (see Table 4.2) and $w_\rho = 10$. Equivalently, given a sampled trajectory for the coolant flow rate $F_{\text{target},t} \forall t \in \mathrm{T}$, the OCP minimizes

$$\sum_{t \in \mathrm{T}} (c_{\text{target}} - c_t)^2 + w_F \cdot (F_{\text{target},t} - F_t)^2, \tag{4.4}$$

with $w_F = 0.1$. We choose w_ρ and w_F so that the resulting data set has good coverage of the feasible part of the state space (see Table 4.2, eNMPC). We use the GEKKO Optimization Suite (Beal et al. (2018)) with default settings to formulate and solve the OCPs. Given the resulting trajectories for the control inputs $\boldsymbol{u}$, we use the RK45 solver (Dormand and Prince (1980)) in SciPy (Virtanen et al. (2020)) to compute the system response and generate the data set for SI. We generate 84 trajectories for training and validation purposes. Each of those trajectories has a length of 5 days, i.e., 480 time steps. We use 63 of those trajectories for training and the remaining 21 for validation.

We perform SI by minimizing the sum of three loss functions (see Equations 2.10 - 2.12), corresponding to the three requirements to a dynamic Koopman model (see Section 2.4). To obtain Koopman models that perform well over multiple time steps of closed-loop prediction, we take a stochastic curriculum learning approach (Bengio et al. (2015); Zaremba and Sutskever (2014)), i.e., we start the training procedure by minimizing the mean-squared error (MSE) of predictions over a single time step, and, progress onto minimizing the average MSE over multiple time steps of continuous closed-loop prediction in later epochs. Such a curriculum learning procedure is designed to result in models that make accurate long-term predictions, but to avoid exploding/vanishing gradients resulting from concatenating predictions of an untrained model in the first epochs. Specifically, in the first epoch,

we use the one-step MSE as the loss function. After 250 epochs, we use the average MSE over 240 time steps, i.e., 2.5 days, of closed-loop prediction in every epoch. Between the first and the 250th epoch, we decrease the probability of taking the one-step loss linearly from 100 % to 0 % in favor of the 240-step loss. We use the Adam optimizer (Kingma and Ba (2014)) with a learning rate of $0.5 \cdot 10^{-4}$, a mini-batch size of 64 samples, and a maximum number of 5,000 epochs. We stop training early after at least 350 epochs if the validation loss has not reached a new minimum for 100 consecutive epochs.

A pragmatic and common practice when training ML models is to oversize the models and then use regularization techniques and early stopping to prevent overfitting (Goodfellow et al. (2016)). However, we aim to obtain models that strike a good balance between accuracy and computational tractability. Therefore, we avoid unnecessary oversizing of our Koopman surrogate models by starting with a small model and repeatedly training larger models until the performance gains become negligible. All results shown in the following are obtained using a Koopman model with a latent space dimensionality of eight, i.e., $\boldsymbol{A_\theta} \in \mathbb{R}^{8\times8}, \boldsymbol{B_\theta} \in \mathbb{R}^{8\times2}, \boldsymbol{C_\theta} \in \mathbb{R}^{2\times8}$, and an MLP encoder $\boldsymbol{\psi}\colon \mathbb{R}^2 \mapsto \mathbb{R}^8$ with two hidden layers (four and six neurons, respectively), and hyperbolic tangent activation functions.

4.2.3. NMPC

Here, the controller task is to stabilize the product concentration c and temperature T at constant target values (see Table 4.2). Every eight hours, the production rate ρ is set to a random value within its bounds for the next eight hours. The controller can respond by varying the coolant flow rate F within its bounds to stabilize c and T. Thus, the production rate becomes an external input ρ_{ext}, and the coolant flow rate F is the control input.

During RL, we define the reward r_t for each control action as proportional to the negative squared relative deviation between c, T, and their respective steady-state values:

$$r_t = -((c_t - c_{\text{target}})/(c_{\text{ub}} - c_{\text{lb}}))^2 - ((T_t - T_{\text{target}})/(T_{\text{ub}} - T_{\text{lb}}))^2 \tag{4.5}$$

Relative deviations are achieved by normalizing each variable using the size of its feasible range (see Table 4.2). Aptly, using the feasible range of each variable, we also normalize the inputs and outputs of the Koopman models to values between zero and one. At each control step, the MPC controller solves an OCP in Koopman space, minimizing the squared distance from the target state $\boldsymbol{x}_{\text{target}}$:

$$\min_{(F_t)_{t\in\mathrm{T}}} \sum_{t\in\mathrm{T}_{+1}} ||C_{\boldsymbol{\theta}}\boldsymbol{z}_t - \boldsymbol{x}_{\text{ss}}||, \tag{4.6a}$$

$$\text{s.t. } \boldsymbol{z}_0 = \boldsymbol{\psi_\theta}(\boldsymbol{x}_0), \tag{4.6b}$$

$$\boldsymbol{z}_{t+1} = A_{\boldsymbol{\theta}}\boldsymbol{z}_t + B_{\boldsymbol{\theta}} \begin{pmatrix} \rho_{\text{ext}} \\ F_t \end{pmatrix} \qquad \forall t \in \mathrm{T}, \tag{4.6c}$$

$$\underline{F}_t \le F_t \le \bar{F}_t \qquad \forall t \in \mathrm{T}. \tag{4.6d}$$

We use an MPC horizon of $t_f = 12$ quarter hours, i.e., three hours, and episodes with a duration of 288 time steps, i.e., three days. As the production rate ρ_{ext} is adjusted every eight hours, each episode includes eight step changes.

We repeat the RL training of the *Koopman-RL* and *MLP* controllers over ten random seeds. The values of training hyperparameters are given in Appendix A. The MLP controllers have two fully connected hidden layers, each with 256 neurons, followed by an output layer with one neuron for F. All layers have hyperbolic tangent activation functions. As a performance indicator during the training process, we use the running average sum of rewards during the previous 30 episodes, i.e., 2,160 control steps. As the policies are updated every 2,048 control steps, taking the running average of 30 episodes has a relatively low variance while not being overly strongly influenced by actions resulting from outdated policy parameters. In the following, we call the undiscounted cumulative reward of an episode the *score* of that episode.

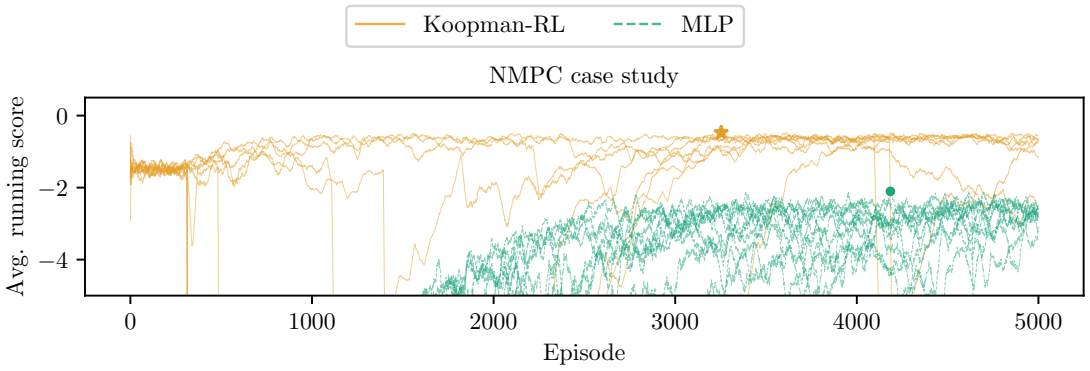


Figure 4.3.: Summary of training runs in NMPC. Each training configuration is run ten times with different random seeds. We average the running score over the last 30 episodes. The lines represent individual training runs. Additionally, we highlight the highest average running score achieved by a policy type (star for *Koopman-RL*, dot for *MLP*).

Figure 4.3 summarizes the RL training results for both RL-trained controller types. We show the running average of training scores of each training run. Due to using a pretrained Koopman model, the *Koopman-RL* controllers start the RL training at comparatively high scores. Still, after around 5,000 episodes, most *Koopman-RL* controllers have significantly increased their average scores compared to the SI baseline, and the performance reaches a plateau. Some training runs show rather steadily improving scores from the start, reaching the plateau after about 1,000 episodes. Other trainings show an initial drop in performance, followed by a recovery. The MLP controllers are randomly initialized and obtain low scores at the beginning of the training. The performance improves gradually and plateaus after about 3,000 episodes at average scores below those of the SI-pretrained Koopman controllers.

After completing all training runs, we test the policies that achieved the highest average running scores without exploration, i.e., we set $\boldsymbol{u}_t = \boldsymbol{u}_t^*$, on 100 test episodes. Table 4.3 shows the results of this test. A direct comparison of the three controllers, given the same randomly generated production rate trajectory, is given in Figure 4.4. Note that as exploration is turned off in these tests, the performance may be higher than during training (cf. Figure 4.3).

All three controllers exhibit sensible behavior. However, Table 4.3 shows that the *Koopman-RL* controller manages to keep the state of the CSTR on average closer to the target than the other two controllers. This finding is confirmed by Figure 4.4, which shows

Table 4.3.: NMPC: test results over 100 episodes. Scores represent the negative summed squared relative deviation from the target values, i.e., higher is better, and a score of 0 is the upper performance bound.

Score	avg.	std.	min.	max.
MLP	-2.21	0.73	-4.40	-0.43
Koopman-SI	-1.35	0.62	-3.06	-0.29
Koopman-RL	-0.50	0.19	-1.27	-0.12

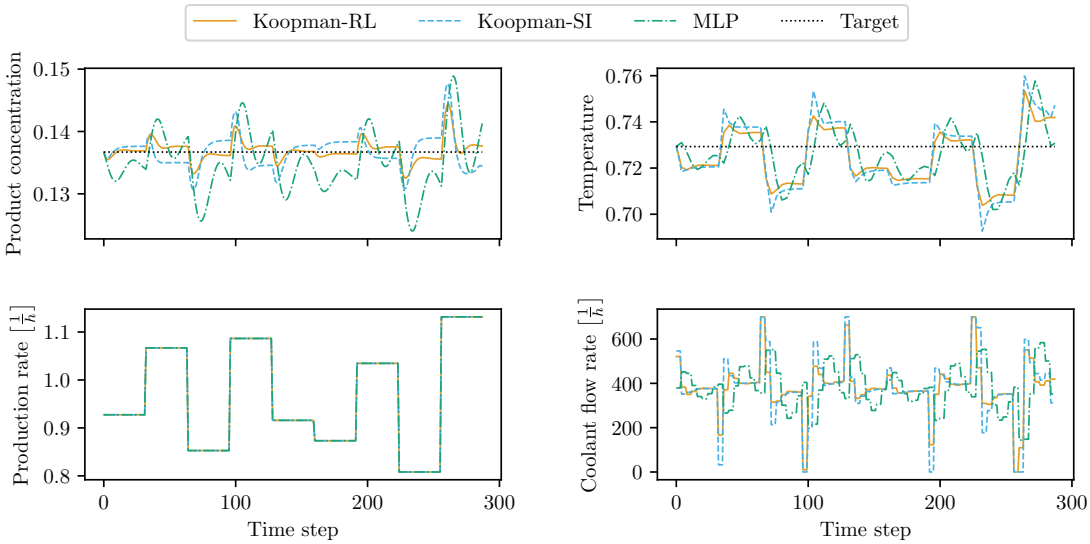


Figure 4.4.: NMPC: comparison of the controller behavior, given the same randomly generated production rate trajectory. Best viewed in color.

that the *Koopman-RL* controller manages to stabilize the product concentration c quicker and much closer to the target than the other two controllers. The *MLP* controller does not stabilize c at or near the target; instead, it makes c oscillate around its target. Regarding the temperature variable T, all three controllers produce an average deviation from T_{target} that is substantially larger than the average deviation of c from c_{target}. This has two reasons: Firstly, the feasible range of T is larger than that of c (see Table 4.2), which affects the reward calculation (Eq. 4.5) as we normalize deviations using the feasible range of each variable. Secondly, the objective function in the Koopman OCPs (Eq. 4.6a) is also affected due to the normalization of the inputs of the Koopman models. The overall performance regarding T is very similar for all three controllers. Again, the *MLP* controller produces oscillating behavior, whereas both Koopman controllers establish steady states at some distance to the target after each step change.

4.2.4. eNMPC

Here, the controller task is to optimize process economics through demand response, i.e., shifting electricity consumption in time to benefit from intervals with relatively low prices, given electricity price profiles. The state variables c and T are subject to box constraints,

with upper and lower bound values given in Table 4.2. The control inputs $\boldsymbol{u}_t = (\rho_t, F_t)^\intercal$ can be varied freely within their bounds. For simplicity, we assume that electric power consumption is proportional to the coolant flow rate F. Thus, the controller can improve the process economics by shifting process cooling to low-price intervals. To enable flexible operation, we assume a product storage with filling level l_t and a maximum capacity of six hours of steady-state production. Because the number of variables in an OCP scales with the prediction horizon, RL training with a differentiable MPC controller and a long time horizon can quickly become computationally infeasible. We identify a price prediction horizon that results in a performant eNMPC by repeatedly solving eNMPC control problems using the mechanistic model (Equations (4.2a) - (4.2b)). Based on the results, we choose a price prediction horizon of nine hours, as longer horizons yield no substantial improvement in control performance.

We use historic day-ahead electricity prices from the Austrian market (Open Power System Data (2020)) from March 29, 2015, to March 25, 2018, to train the controllers. The final testing is performed on Austrian electricity price data from March 26, 2018, to September 30, 2018. Our goal is to train controllers that perform well when used continuously over an arbitrarily long time horizon. In addition, when testing controllers on short episodes in the eNMPC setting, the economic performance of a controller can be heavily influenced by how the controller utilizes the product storage. For example, a controller that fills up the storage initially to obtain flexibility might incur higher costs in a short episode than a steady-state production but might achieve economic savings in the long run. Therefore, we test the final performance of the controllers in a single continuous episode that covers half a year. However, during training, it makes sense to use shorter episodes to allow for more frequent restarts. This is especially important at the beginning of the training procedure when controllers are likely to steer the system to undesirable states from which little can be learned. Therefore, as in the NMPC case study, we choose an episode length of 288 control steps, i.e., three days, for training. At the start of each training episode, the state variables get reset to their steady-state values, the storage level is initialized randomly between one and two hours of steady-state production volume, and a connected series of electricity prices of length T is sampled from the data. In the roughly half-year-long test episode, we start with an empty storage to have a consistent starting point for the three different controllers. The product concentration c and the temperature T are initialized at their steady-state values.

For RL, we choose a reward that depends on the electricity cost savings compared to steady-state production (see Table 4.2) and, on whether any bounds on state variables were violated during the previous control interval. First, we check for constraint violations and compute the steady-state costs

$$c_{\mathrm{ss}} = F_{\mathrm{ss}} \cdot p_t \cdot \Delta t_{\mathrm{ctrl}}, \tag{4.7}$$

and the costs induced by the current control action

$$c_t = F_t \cdot p_t \cdot \Delta t_{\mathrm{ctrl}}, \tag{4.8}$$

using the electricity price p_t. Then, we calculate the reward

$$r_t = \begin{cases} -1, & \text{if constraint violation} \\ (c_{\mathrm{ss}} - c_t)\beta, & \text{otherwise,} \end{cases} \tag{4.9}$$

wherein β is a hyperparameter used to balance the influence of cost savings and constraint violations.

At each control step, the eMPC controller solves an OCP in Koopman space, aiming to minimize the operational costs while satisfying all constraints:

$$\min_{(\rho_t, F_t)_{t\in \mathrm{T}}} \sum_{t\in \mathrm{T}_{+1}} (F_t p_t \Delta t_{\mathrm{ctrl}} + M \boldsymbol{s}_t^\intercal \boldsymbol{s}_t), \tag{4.10a}$$

$$\text{s.t. } \boldsymbol{z}_0 = \boldsymbol{\psi}_{\boldsymbol{\theta}} \begin{pmatrix} c_0 \\ T_0 \end{pmatrix}, \tag{4.10b}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{A}_{\boldsymbol{\theta}} \boldsymbol{z}_t + \boldsymbol{B}_{\boldsymbol{\theta}} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{4.10c}$$

$$l_{t+1} = l_t + (\rho_t - \rho_{\mathrm{ss}}) \Delta t_{\mathrm{discr}} \quad \forall t \in \mathrm{T}, \tag{4.10d}$$

$$\boldsymbol{x}_t = \boldsymbol{C}_{\boldsymbol{\theta}} \boldsymbol{z}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{4.10e}$$

$$\underline{\boldsymbol{x}}_t - \boldsymbol{s}_{\boldsymbol{x},t} \leq \boldsymbol{x}_t \leq \bar{\boldsymbol{x}}_t + \boldsymbol{s}_{\boldsymbol{x},t} \quad \forall t \in \mathrm{T}_{+1}, \tag{4.10f}$$

$$0 - s_{l,t} \leq l_t \leq 6.0 + s_{l,t} \quad \forall t \in \mathrm{T}_{+1}, \tag{4.10g}$$

$$\boldsymbol{s}_t = \begin{pmatrix} \boldsymbol{s}_{\boldsymbol{x},t} \\ s_{l,t} \end{pmatrix} \quad \forall t \in \mathrm{T}_{+1}, \tag{4.10h}$$

$$\boldsymbol{0} \leq \boldsymbol{s}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{4.10i}$$

$$\underline{\boldsymbol{u}}_t \leq \boldsymbol{u}_t \leq \bar{\boldsymbol{u}}_t \quad \forall t \in \mathrm{T}. \tag{4.10j}$$

As explained in Section 4.1, we introduce slack variables $\boldsymbol{s}_{\boldsymbol{x},t}$ for the state variables and $s_{l,t}$ for the product storage to ensure feasibility of the policy but penalize their use in the objective function.

The values of the training hyperparameters are given in the Appendix A. The MLP controllers have separate input layers for $\boldsymbol{x}_t$, l_t, and the trajectory of future electricity prices. Each of those input layers is followed by a hidden layer, with 100, 56, and 100 neurons, respectively. The outputs of those layers are concatenated and passed through two fully connected layers, each of size 256 neurons. The output layer has two neurons, one each for ρ and F. All layers have hyperbolic tangent activation functions.

As in NMPC, we repeat the RL training of the *Koopman-RL* and *MLP* controllers over ten random seeds. Again, we use the average running score during the previous 30 episodes, i.e., 2,160 control steps, as a performance indicator during the training process. Figure 4.5 summarizes the RL training results for both controller types. We show the running average of training scores of each training run. Unlike in NMPC, the SI pretraining of the Koopman models does not lead to high initial scores, as the resulting Koopman MPCs frequently produce minor constraint violations, resulting in low rewards (see Equation 4.9). The performance of the *Koopman-RL* controllers improves and saturates after about 2,500 episodes. After 5,000 episodes, the maximum running scores of the *Koopman-RL* controllers have reached values between 1.4 and 3.8. The running scores of the *MLP* controllers rise to around zero after only about 750 episodes. Afterward, the running scores continue to rise slowly, saturating only after about 20,000 episodes at maximum running scores between 4.6 and 5.7. We stopped the RL refinement of the *Koopman-RL* controllers at 5,000 episodes because of the high computational cost of the training procedure compared to the *MLP* controllers, which is due to the necessity to solve and backpropagate through OCPs repeatedly. Because our case study is relatively small, analyzing training

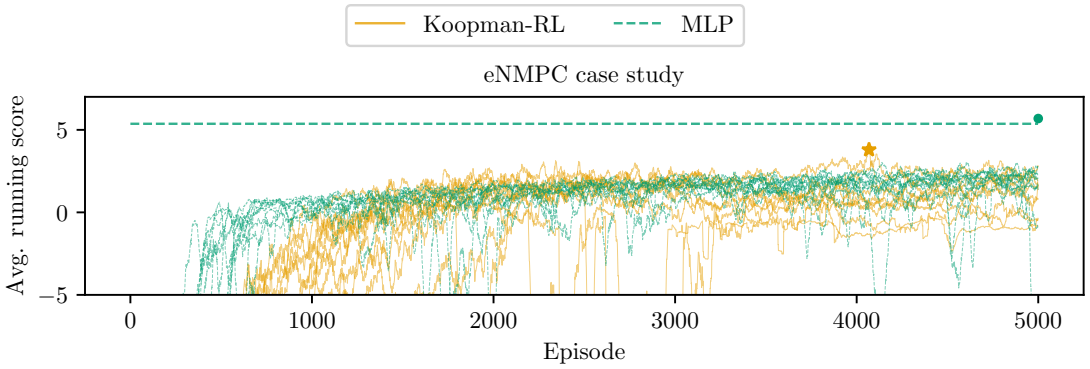


Figure 4.5.: Summary of RL training runs in eNMPC. Each training configuration is run ten times with different random seeds. We average the running score over the previous 30 episodes. The lines represent individual training runs. The constant dashed green line represents the median maximum performance when training the MLP policies for 20,000 episodes. Additionally, we highlight the highest average running score achieved by a policy type (star for *Koopman-RL*, dot for *MLP*). Best viewed in color.

runtimes would give little insight into the expected runtime for a practical control problem. Thus, we defer this analysis for future work on more realistic systems.

We observe that convergence to high average scores is unstable, especially for the Koopman-RL policies. This is a common finding in policy optimization algorithms (Schulman et al. (2015a, 2017)), stemming from the fact that small perturbations to the parameters $\boldsymbol{\theta}$ of a policy can have outsized effects on the resulting actions, and thus also the rewards. We hypothesize that this effect is aggravated when learning the parameters of an MPC policy, as small changes in trajectory predictions can lead to very different active sets of constraints, resulting in vastly different control outputs.

Similar to our analysis in Section 4.2.3, after completing the controller training runs, we test the policies which achieved the highest ever running scores during training. We perform this testing without exploration, i.e., we set $\boldsymbol{u}_t = \boldsymbol{u}_t^*$, and we report the performance metrics in Table 4.4. Additionally, we show the control behavior of the three controllers in a three-day test episode using a randomly sampled continuous electricity price profile from the test set in Figure 4.6 for illustration purposes.

Table 4.4.: eNMPC: test results over a single roughly half-year long test episode with electricity price data from March 26, 2018 to September 30, 2018. Costs are stated relative to steady-state production. The constraint violation column reports the percentage of control steps that result in constraint violations.

	Cost	Constr. viols. [%]	Avg. storage level [h]
MLP	0.91	0.24	0.80
Koopman-SI	0.90	8.84	0.28
Koopman-RL	0.94	0.22	5.32

Table 4.4 shows that even though the *Koopman-SI* controller achieves the highest cost savings, it does so at the expense of causing many constraint violations. The *Koopman-RL*

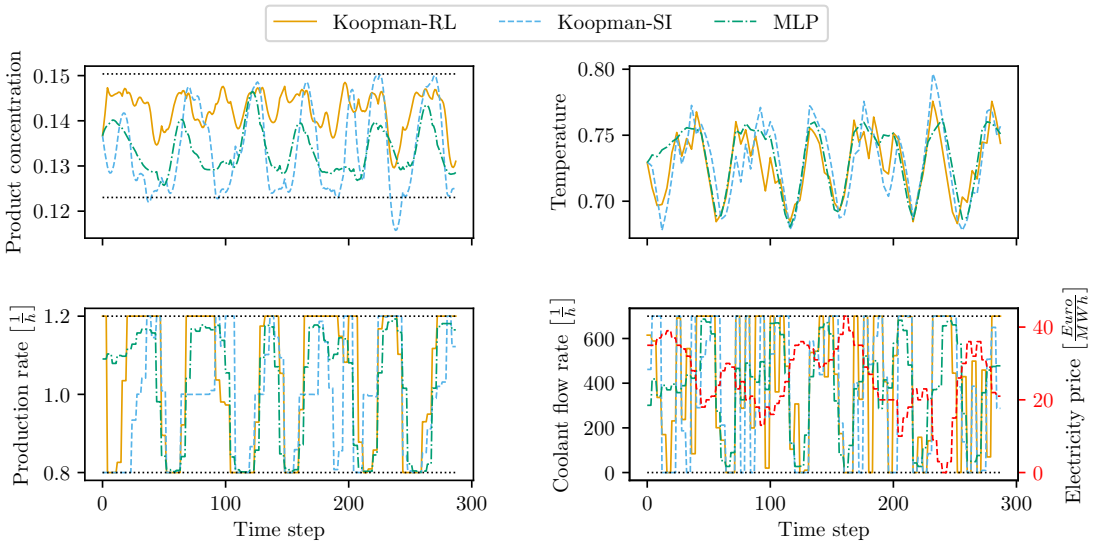


Figure 4.6.: eNMPC: comparison of the controller behavior, given a randomly sampled continuous electricity price trajectory from the test set. The dotted lines represent the bounds (Table 4.2). Best viewed in color.

controller and the *MLP* controller cause few constraint violations. In case of the *Koopman-RL* controller, all violations are related to the bounds of the concentration variable c. The *MLP* controller achieves higher cost savings than the *Koopman-RL* controller. Interestingly, all constraint violations by the *MLP* controller are violations of the non-negativity constraint of the product storage. Apparently, the controllers utilize the product storage quite differently. While the *Koopman-SI* controller mostly keeps the storage near-empty, the *Koopman-RL* controller exhibits the opposite behavior, generally operating close to the maximum storage capacity of six hours, which could partially explain the difference in economic performance compared to the *MLP* controller. Most importantly, end-to-end training clearly improved the overall performance of the Koopman MPC in this application.

4.2.4.1. Analysis of Koopman embedding before and after end-to-end training

A central challenge in applied Koopman theory is identifying a suitable set of functions for mapping between the original state space and the lifted space where the dynamics can be approximated linearly with high accuracy. Here, we analyze how the proposed end-to-end refinement of a Koopman model for task-optimal performance in eNMPC affects the autoencoder part of the model, which defines the mapping between the original and lifted domains and which is pre-trained by SI and then fine-tuned by RL. To this end, we generate a data set of 100,000 steps in the demand response environment using the full mechanistic model by applying the *Koopman-SI* controller. This data set contains 8,302 steps with constraint violations in either c or T, and 91,698 steps without constraint violations. We use this data set to calculate the autoencoder MSE for both the *Koopman-SI* and the *Koopman-RL* model as a measure for the reconstruction loss. Furthermore, we pass the 100,000 points through the autoencoders of both models. Then, we calculate the rates at which points with violations of state bounds get correctly mapped to the exterior of the feasible region defined by the state bounds (sensitivity). To determine the specificity, we

proceed analogously for the points without constraint violations.

Table 4.5.: Analysis of autoencoder performance and constraint violation detection rates for the *Koopman-SI* and *Koopman-RL* models. The table displays the autoencoder mean-squared error (MSE) as a measure for the reconstruction loss, sensitivity (true positive rate), and specificity (true negative rate) for different combinations of encoder and decoder from *Koopman-SI* (SI) and *Koopman-RL* (RL) models. Herein, *positive* refers to a point with a constraint violation in c or T, and *negative* to a point without a constraint violation.

Encoder model	Decoder model	Autoencoder MSE	True pos. (sensitivity)	True neg. (specificity)
SI	SI	$3.2 \cdot 10^{-5}$	0.844	0.995
RL	RL	$3.5 \cdot 10^{-4}$	0.980	0.947
SI	RL	$3.5 \cdot 10^{-5}$	0.844	0.995
RL	SI	$4.1 \cdot 10^{-4}$	0.971	0.947

As can be seen from the upper part of Table 4.5, the end-to-end refinement leads to a tenfold increase in the reconstruction loss; still, the loss remains at a relatively low level. Importantly, the autoencoder of the *Koopman-RL* model shows a much higher sensitivity than the autoencoder of the *Koopman-SI* model, i.e., the rate at which a point that violates a constraint on c or T still does so after being passed through the autoencoder rises from 84.4 % to 98.0 %. When using the Koopman models as part of an eNMPC, this increase in sensitivity and the related decrease in specificity leads to more conservative controller behavior. The higher sensitivity and specificity are expected as the Koopman model was refined using a reward function that heavily punishes constraint violations (see Eq. 4.9).

To identify which part of the autoencoder model was modified to what extent during the refinement, we combine the encoder of one model with the decoder of the other model, i.e., the encoder of the *Koopman-SI* model with the decoder of the *Koopman-RL* model and vice versa. As can be seen from the lower part of Table 4.5, the combinations of encoders and decoders from different models still yield low reconstruction losses, suggesting that the Koopman models before and after the refinement operate with very similar lifted spaces. Interestingly, using the encoder of the *Koopman-SI* model together with the decoder of the *Koopman-RL* model produces almost identical results to using the full *Koopman-SI* autoencoder. Conversely, using the *Koopman-RL* encoder in conjunction with the *Koopman-SI* decoder produces very similar results to using the full *Koopman-RL* autoencoder. These observations indicate that the end-to-end refinement affected the encoder to a larger extent than the decoder.

4.2.5. eNMPC with adapted bounds

One key advantage of MPC controllers over model-free policies is that they may be able to adapt to changing control settings, such as shifted bounds or a change in the objective function. Model-free policies cannot adapt to such changes, as constraints and objective function are learned implicitly by the policy through the reward signal of the environment. Therefore, any change to the environment would require a retraining of the policy.

We shift the bounds of the product concentration variable c and rerun the test (cf. Subsection 4.2.4) without retraining. We examine three different constraint adaptations: relaxed bounds, tightened bounds, and a shifted feasible region of c, i.e., a different target product. Table 4.6 shows the adapted bounds of c in all three cases and reports the frequency of constraint violations that result from applying the controllers. Additionally, Figure 4.7 illustrates the resulting trajectories of c when using the controllers on a three-day test episode for the different bound adaptations. We waive reporting the production costs in Table 4.6 as they become meaningless in the presence of vast differences in the rates of constraint violations.

Table 4.6.: Adapted lower (lb) and upper (ub) bounds of c and constraint violations resulting from applying the controllers without retraining. The constraint violation column reports the percentage of control steps that result in constraint violations.

	Adapted bounds of c		Constr. viols. [%]		
	lb	ub	*MLP*	*Koopman-SI*	*Koopman-RL*
Tightened	0.1299	0.1435	49.20	19.53	1.73
Relaxed	0.1162	0.1572	0.24	4.41	0.13
Shifted	0.1504	0.1777	100.00	14.94	5.83

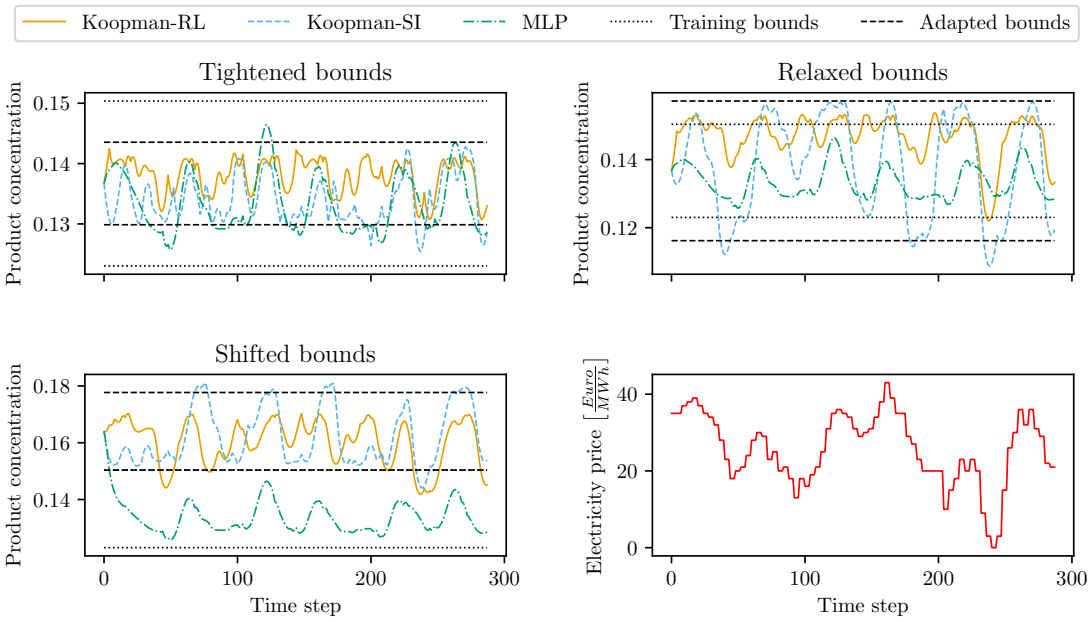


Figure 4.7.: eNMPC with adapted c bounds: comparison of the controller behavior, given a randomly sampled continuous electricity price trajectory. The original bounds used during RL training are shown in dotted black lines, whereas the shifted bounds are shown in dashed black lines. Best viewed in color.

Both Table 4.6 and Figure 4.7 show that the *Koopman-RL* controller is best at adapting to a change in the control setting, even though RL trained it for optimal performance in a somewhat different setting. Even though the *tightened bounds* and *shifted bounds*

cases lead to more frequent constraint violations than in the original setting (cf. Table 4.4), the *Koopman-RL* controller apparently still profits from its end-to-end refinement, as constraint violations are markedly rarer than for the *Koopman-SI* controller. In the *tightened bounds* and *relaxed bounds* cases, the relative difference in the frequency of constraint violations compared to the *Koopman-SI* controller stays similar to the original case (cf. Table 4.4). In the *shifted bounds* case, the *Koopman-RL* controller produces more constraint violations than in the other two cases, which is expected, as the RL training discouraged ever entering the *shifted bounds* range.

Unsurprisingly, the *MLP* controller is agnostic to a change in the control setting and fails in the *tightened bounds* and *shifted bounds* cases. In the *relaxed bounds* case, it causes few constraint violations due to its unaltered behavior. Again, all constraint violations are violations of the non-negativity constraint of the product storage, which we did not relax. As the *MLP* controller respects the original training bounds of the state variables c and T, it does not take advantage of the enhanced flexibility potential of the process in the *relaxed bounds* case.

4.3. Conclusion

This chapter presents a method for end-to-end RL of Koopman models for optimal performance in (e)NMPC applications. We exploit the modeling approach by Korda and Mezić (2018) to construct Koopman-MPC controllers with convex underlying OCPs, and *cvxpylayers* (Agrawal et al. (2019a)) for post-optimal sensitivity analysis, thus turning the Koopman-MPC controllers into automatically differentiable policies that can be trained using state-of-the-art RL methods like PPO (Schulman et al. (2017)). Applying a trained Koopman-MPC controller has a comparatively low computational cost due to the convexity of the OCPs. Therefore, our method should be able to produce predictive controllers that are real-time capable for potentially large control systems.

We demonstrate the effectiveness of our method by comparing the control performance to those of MPC utilizing models trained by system identification and model-free neural network controllers trained by RL. Specifically, we investigate two control applications derived from a CSTR model: (i) a setpoint tracking problem (NMPC, Section 4.2.3), and (ii) a demand response problem with hard constraints on state variables (eNMPC, Section 4.2.4). Our RL-trained Koopman models consistently outperform their counterparts trained by system identification.

Additionally, we alter the feasible region of a state variable in eNMPC and test the controllers without retraining (Section 4.2.5). Such changes in the control setting present a significant challenge to data-driven control. Due to the explicit constraints in the Koopman-eNMPCs, end-to-end training using RL preserves the controllers' inherent adaptability to changes in the environment. Thus, even though the Koopman-eNMPCs lose some of the gains in performance from the RL refinement procedure, they can adapt better to the changes to the environment than the Koopman-eNMPCs trained by system identification alone. As expected, the neural network controllers cannot adapt as they implicitly learn the constraints of their environment through the rewards received during training.

We validated our method in NMPC and eNMPC settings using a nonlinear, yet comparatively simple, CSTR model, far from the complexity encountered in many real-world

systems. In the next chapter, we therefore evaluate the method on a substantially more complex demand response scenario involving a high-dimensional model of an air separation unit (ASU).

As discussed in Section 4.2, convergence to high rewards is quite unstable given the Koopman eNMPC policies. Unstable convergence is a common finding in modern actor-critic RL algorithms and can be a result of inaccurate policy gradient estimates (see Section 3.4). Given eNMPC policies, estimating policy gradients is further complicated by the presence of inequality constraints in the underlying OCPs, which can be active or inactive. However, if a *surrogate* model is to be learned from an automatically differentiable mechanistic model, analytic simulation gradients become available. Therefore, in Chapter 6, we study how such analytic simulation gradients can aid end-to-end learning of Koopman eNMPC policies.

Another limitation of the method presented in this chapter is its low sample efficiency. Sample efficiency becomes a critical issue when interactions with the environment are costly, e.g., when learning from interactions with a real-world environment or due to computationally intensive simulations. Modern model-based RL algorithms (e.g., Janner et al. (2019); Frauenknecht et al. (2024)) can match the asymptotic performance of state-of-the-art actor-critic RL algorithms like PPO in many learning environments while being vastly more sample efficient. Moreover, when learning from a real-world system with only partial knowledge of its dynamics, these algorithms readily accommodate physics-informed model learning. In Chapter 7, we therefore integrate the differentiable Koopman eNMPC policies developed here with a physics-informed version of a state-of-the-art model-based RL algorithm (Janner et al. (2019)).

5. RL-based end-to-end learning of Koopman (e)NMPCs: application to an air separation unit model

In Chapter 4, we introduce an RL-based method for end-to-end learning of Koopman models (Koopman (1931); Korda and Mezić (2018)) for (e)NMPC applications. We demonstrate the efficacy of our method in two simulated case studies (NMPC and eNMPC) based on a small model of a continuous stirred-tank reactor (CSTR) (Flores-Tlacuahuac and Grossmann (2006)) comprised of just two ordinary differential equations. In this chapter, we demonstrate the scalability of our method in a demand response case study based on a large-scale index-one nonlinear semi-explicit differential-algebraic equations (DAE) model of an air separation unit (ASU) with approx. 2327 algebraic and 119 differential states (Caspari et al. (2020); Schulze et al. (2023)). We assume full observability of the ASU, i.e., we provide the policy with all 119 differential states and a few selected algebraic states (see Section 5.2.1). We find that our method produces highly performant and easily real-time capable controllers that clearly outperform SI-based Koopman eNMPCs. This means that, given identically-sized Koopman models, our controllers exhibit similar economic performance but, unlike the SI-trained controllers, they avoid constraint violations.

The remainder of this chapter is organized as follows: First, the ASU demand response case study is introduced in Sec. 5.1. Then, Sec. 5.2 describes adjustments to the Koopman model architecture and the system identification procedure we employed in this chapter compared to Chapter 4. Sec. 5.3 presents the results of the numerical experiments. Finally, Sec. 5.4 discusses the conclusions and directions for future work.

5.1. Demand response of an air separation unit

We consider demand response of a single-product air separation plant for the production of purified nitrogen based on the benchmark process presented in Caspari et al. (2020), illustrated in Figure 5.1. It is a mid-level complexity air separation unit. RL approaches often need many policy-environment interactions to produce good results. To reduce the environment response time, we therefore construct a demand response RL environment using the modified model version presented in Schulze et al. (2023). The modified model is a nonlinear DAE system with 2327 algebraic and 119 differential states, implemented in Modelica. This section provides a description of the process and how we construct an RL environment based on the model.

Ambient air is compressed in the main air compressor (MAC) and subsequently passes through a pre-cooler and a multi-stream heat exchanger, where it is cooled against ambient air and the cold streams exiting the process. The multi-stream heat exchanger is divided

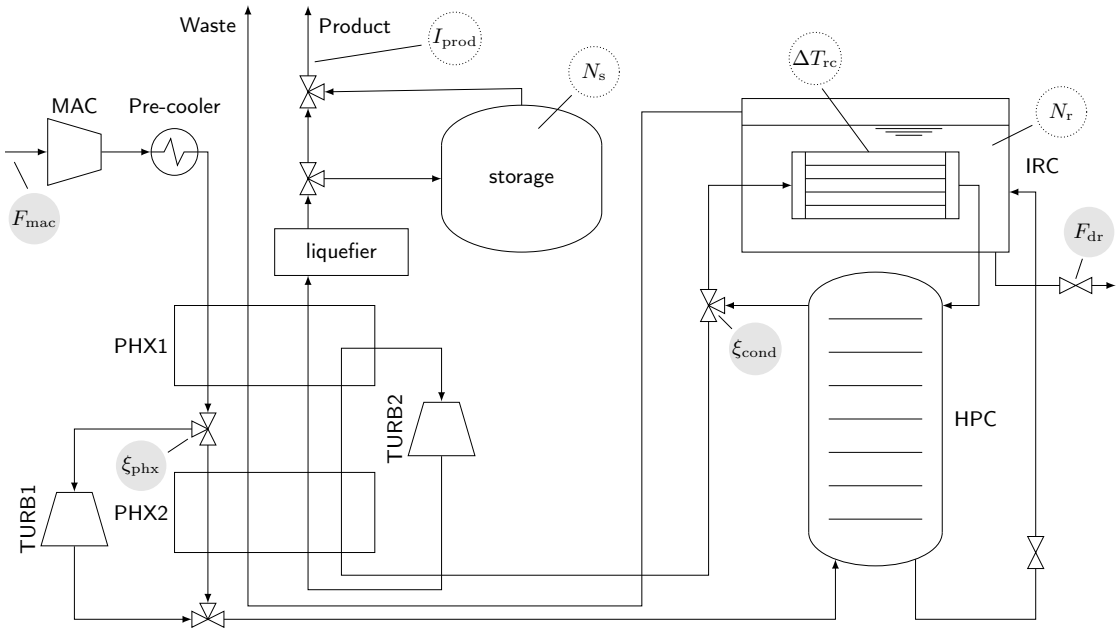


Figure 5.1.: Air separation process flowsheet. The following manipulated variables are shown with a grey background: inlet air flow rate F_{mac}, air fraction passing through turbine 1 ξ_{phx}, distillation column reflux ratio ξ_{cond}, and drain stream F_{dr}. The controlled variables are depicted with dotted circles: product impurity I_{prod}, molar holdup in storage N_{s} and reboiler N_{r}, temperature difference between reboiler and condenser ΔT_{rc}.

into two parts. After the first section (PHX1), a fraction of the entering air stream is used for electricity generation in turbine 1, while the remainder is liquefied in the second part of the heat exchanger (PHX2). Both streams are recombined before entering the high-pressure distillation column (HPC). The oxygen-rich bottom product of the column is expanded and used in an integrated reboiler condenser (IRC) to cool the column reflux stream. Liquid can be withdrawn from the reboiler via a drain stream. The vapor that exits the reboiler passes through the multi-stream heat exchangers before it leaves the process as a waste stream. At the top of the column, a nitrogen-rich stream is withdrawn. A fraction of this stream becomes the product stream and is directed through the multi-stream heat exchangers and turbine 2. We consider a case where liquid nitrogen shall be delivered. Therefore, the product stream passes through a liquefier before leaving the process. To enable flexible operation, the process includes a storage tank, where the liquid product can be stored for later delivery.

The task of the eNMPC is to minimize operational cost by exploiting variations in the electricity price, while fulfilling a constant demand for liquid nitrogen and avoiding constraint violations. The operational cost is given by the overall power consumption E of the ASU multiplied by the electricity price. For the overall power consumption, the energy demand of the MAC and the liquefier, as well as the electricity generation from the turbines, are taken into account. Operational constraints regarding the product purity I_{prod}, the molar holdups in the storage tank N_s and the reboiler N_r, as well as the temperature difference between reboiler and condenser ΔT_{rc} must be satisfied. Control inputs that can be manipulated are the inlet air flow rate F_{mac}, the fraction of the entering air stream passing through turbine 1 ξ_{phx}, the reflux ratio of the distillation column ξ_{cond} and the drain

stream F_{dr}. The values for the lower and upper bounds of the controlled and manipulated variables are given in Table 5.1.

Variable	lb	ub	Constraint type
I_{prod} [ppm]	0	1800	path
ΔT_{rc} [K]	2	5	path
N_r [kmol]	2	10	path
N_s [-]	0	6	path
F_{mac} [mol/s]	30	50	input
F_{dr} [mol/s]	0	2	input
ξ_{phx} [kmol]	0	0.1	input
ξ_{cond} [-]	0.51	0.54	input

Table 5.1.: Summary of lower (lb) and upper (ub) bounds of the operational and input variables.

To use the Modelica model as part of an RL environment, we export it as a functional mock-up unit that can be simulated within Python code. At each control step in the environment, the policy receives the current state of the ASU and an accurate electricity price prediction for the upcoming 9 hours. After receiving a control input from the policy, the state of the ASU is updated by simulating the model for a time step of 15 min. Furthermore, analogous to the reward calculation in Chapter 4, we calculate a reward based on constraint violations and electricity cost savings compared to steady-state production.

5.2. Modifications to the method compared to Chapter 4

5.2.1. Koopman architecture

In the previous chapters, we utilize Koopman models of the form proposed by Korda and Mezić (2018) (see Section 2.4): (i) A nonlinear state observation function $\boldsymbol{\psi_\theta}\colon \mathbb{R}^{n_{\boldsymbol{x}}} \mapsto \mathbb{R}^{n_{\boldsymbol{z}}}$ that transforms the initial system state $\boldsymbol{x}_0 \in \mathbb{R}^{n_{\boldsymbol{x}}}$ into the initial Koopman state $\boldsymbol{z}_0 \in \mathbb{R}^{n_{\boldsymbol{z}}}$, where typically $n_{\boldsymbol{z}} \gg n_{\boldsymbol{x}}$: $\boldsymbol{z}_0 = \boldsymbol{\psi_\theta}(\boldsymbol{x}_0)$. (ii) The $\boldsymbol{A_\theta} \in \mathbb{R}^{n_{\boldsymbol{z}} \times n_{\boldsymbol{z}}}$ and $\boldsymbol{B_\theta} \in \mathbb{R}^{n_{\boldsymbol{z}} \times n_{\boldsymbol{u}}}$ matrices, which linearly approximate the evolution of the Koopman state, driven by external control inputs $\boldsymbol{u}_t \in \mathbb{R}^{n_{\boldsymbol{u}}}$: $\boldsymbol{z}_{t+1} = \boldsymbol{A_\theta z}_t + \boldsymbol{B_\theta u}_t$. (iii) The $\boldsymbol{C_\theta} \in \mathbb{R}^{n_{\boldsymbol{x}} \times n_{\boldsymbol{z}}}$ matrix, which linearly transforms the Koopman state $\boldsymbol{z}_t$ into a predicted system state $\hat{\boldsymbol{x}}_t$: $\hat{\boldsymbol{x}}_t = \boldsymbol{C_\theta z}_t$. Such models can be trained by adjusting the parameters $\boldsymbol{\theta}$.

Since the ASU model is a DAE system, it features system outputs $\boldsymbol{y}_t$ that exhibit discontinuities when control inputs change non-continuously. The Koopman model architecture proposed by Korda and Mezić (2018), which we used in Chapter 4, cannot reflect such behavior since the control inputs enter directly into the predictor-part of the model (see Equation 2.9b), which produces the latent space vector one time step into the future. To enable the Koopman models to represent instantaneous output responses to input changes, we extend the framework of Korda and Mezić (2018) by adding a second decoder, i.e., $\boldsymbol{y}_t = \boldsymbol{D_\theta z}_t + \boldsymbol{E_\theta u}_t$, with learnable $\boldsymbol{D_\theta} \in \mathbb{R}^{n_{\boldsymbol{y}} \times n_{\boldsymbol{z}}}$ and $\boldsymbol{E_\theta} \in \mathbb{R}^{n_{\boldsymbol{y}} \times n_{\boldsymbol{u}}}$ matrices. Note that this second decoder was not needed in our previous work (Mayfrank et al., 2024)

because the CSTR investigated there was described by an ordinary differential equation (ODE) system for which discontinuities in control inputs do not cause discontinuities in states/outputs. The overall architecture of the Koopman model is visualized in Figure 5.2.

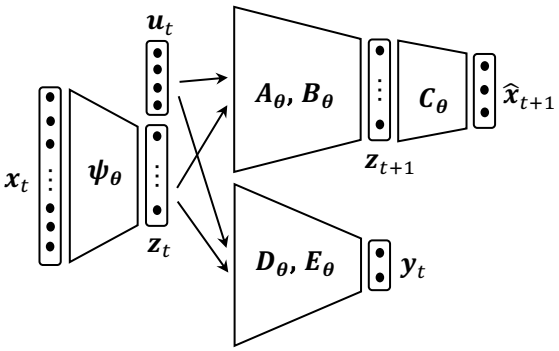


Figure 5.2.: Koopman model architecture used in this chapter. Outputs which do not exhibit discontinuous jumps in response to changes in the control input are computed by obtaining a prediction $\boldsymbol{z}_{t+1}$ of the latent space vector via $\boldsymbol{A}_{\boldsymbol{\theta}}$ and $\boldsymbol{B}_{\boldsymbol{\theta}}$ and decoding this prediction via $\boldsymbol{C}_{\boldsymbol{\theta}}$. Outputs that jump when the control inputs change are computed by directly decoding a concatenation of the current latent space vector $\boldsymbol{z}_t$ and the control inputs $\boldsymbol{u}_t$ via $\boldsymbol{D}_{\boldsymbol{\theta}}$ and $\boldsymbol{E}_{\boldsymbol{\theta}}$.

We assume full observability of the ASU, i.e., all 119 differential states are included in $\boldsymbol{x}_t$. Additionally, we also include in $\boldsymbol{x}_t$ the product impurity I_{prod}, the temperature difference between reboiler and condenser ΔT_{rc}, the reboiler holdup N_r, the molar enthalpy of the product, and the product concentrations. Based on this information, the model constructs a latent representation $\boldsymbol{z}_t$ of the state of the ASU. The vector of control inputs $\boldsymbol{u}_t$ consists of the four inputs listed in Table 5.1. The model predicts the evolution of the control-relevant entries of $\boldsymbol{x}_t$ through the $\boldsymbol{A}_{\boldsymbol{\theta}}$, $\boldsymbol{B}_{\boldsymbol{\theta}}$, and $\boldsymbol{C}_{\boldsymbol{\theta}}$ matrices. This vector $\hat{\boldsymbol{x}}_t \in \mathbb{R}^3$ consists of I_{prod}, ΔT_{rc}, and N_r. The control-relevant, discontinuous outputs $\boldsymbol{y}_t \in \mathbb{R}^2$ are the energy demand E of the process and the production rate $\dot{n}_{product}$. The latter is used to calculate the molar holdup N_{s} of the storage tank.

We choose a latent space dimensionality of 10 for the Koopman model, i.e., $\boldsymbol{z}_t \in \mathbb{R}^{10}$. Altogether, the Koopman model thus consists of the matrices $\boldsymbol{A}_{\boldsymbol{\theta}} \in \mathbb{R}^{10\times10}$, $\boldsymbol{B}_{\boldsymbol{\theta}} \in \mathbb{R}^{10\times4}$, $\boldsymbol{C}_{\boldsymbol{\theta}} \in \mathbb{R}^{3\times10}$, $\boldsymbol{D}_{\boldsymbol{\theta}} \in \mathbb{R}^{2\times10}$, $\boldsymbol{E}_{\boldsymbol{\theta}} \in \mathbb{R}^{2\times4}$, and an encoder $\boldsymbol{\psi}_{\boldsymbol{\theta}} \colon \mathbb{R}^{126} \to \mathbb{R}^{10}$. The encoder is a feed-forward neural network with two hidden layers with 87 and 48 neurons, respectively, and tanh activation functions.

5.2.2. System identification procedure

Using the two-step SI approach employed in Chapter 4, i.e., data generation using random control inputs followed by model fitting, we do not obtain a good initial guess for the Koopman model in the more complex case study of this chapter. Therefore, in this chapter, we employ the iterative data sampling and SI approach depicted in Figure 5.3: We start by generating data through random actuation of the mechanistic model, followed by fitting the parameters $\boldsymbol{\theta}$ of the Koopman model to that data. Then, we construct an eNMPC policy based on the Koopman model and let the policy interact with the environment for 2880 time steps, i.e., 30 simulated days, to extend the training data set for the Koopman

model, and we retrain the model on this larger data set. We repeat this procedure until the maximum average reward obtained during data sampling in one iteration does not improve for five iterations. The model of the policy that produced the maximum average reward is then used as the initial guess for the RL-based end-to-end learning procedure.

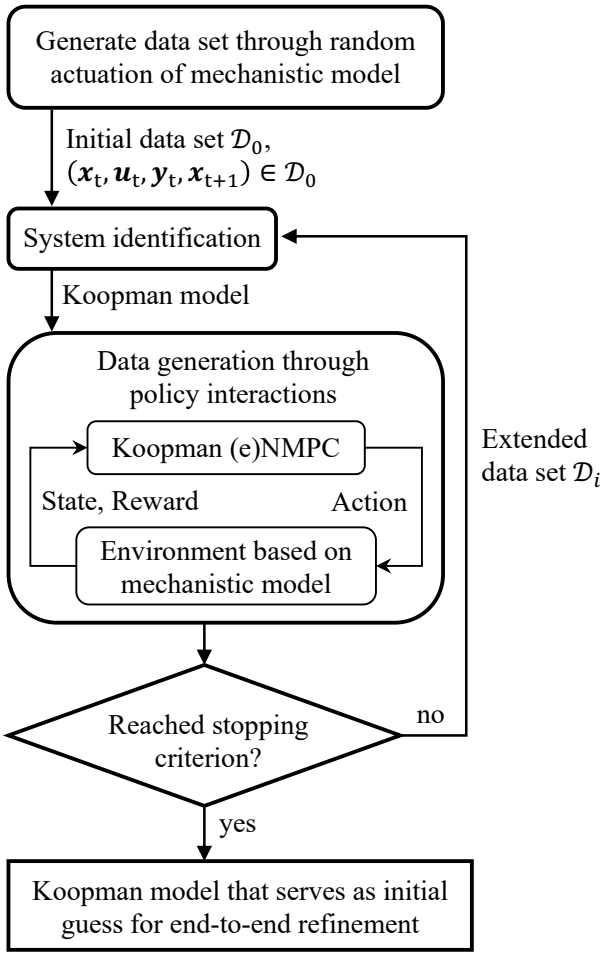


Figure 5.3.: Iterative data sampling and system identification procedure.

RL-based end-to-end refinement of the Koopman model requires solving and backpropagating through the OCPs numerous times. To decrease the associated computational burden, it makes sense to minimize the number of optimization variables in the OCPs, i.e., to maximize the time step duration Δt of the Koopman model and the resulting eNMPC controller. We observe that a discretization of the eNMPC controller using 15 min time steps is short enough to enable the controller to make use of the full feasible space of control inputs without causing constraint violations due to too fast process dynamics. However, during preliminary testing of system identification, we observe that we obtain more accurate predictions by chaining the predictions of models predicting shorter time steps. Therefore, we take the following approach: During system identification, we generate data with a 5 min discretization and we fit the Koopman model to that data. Before using the model as part of eNMPC (in data sampling or end-to-end refinement), we transform the prediction components of the Koopman model $(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{D}, \boldsymbol{E})$ to predict 15 min time steps instead of 5 min steps, i.e., a one-step prediction of the transformed model is

equivalent to chaining three prediction steps with constant control input in the original model. The following equations are used for this exact transformation:

$$\boldsymbol{A}_{15\text{min}} = \boldsymbol{A}^3, \tag{5.1a}$$
$$\boldsymbol{B}_{15\text{min}} = \boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}, \tag{5.1b}$$
$$\boldsymbol{D}_{15\text{min}} = \boldsymbol{D}\boldsymbol{A}^3, \tag{5.1c}$$
$$\boldsymbol{E}_{15\text{min}} = \boldsymbol{D}\boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{D}\boldsymbol{A}\boldsymbol{B} + \boldsymbol{D}\boldsymbol{B} + \boldsymbol{E}. \tag{5.1d}$$

Eqs. 5.1 are derived in Eqs. 5.2 and Eqs. 5.3. Please note that the eNMPC has control steps of 15 min length, and therefore, $\boldsymbol{u}_t = \boldsymbol{u}_{t+1} = \boldsymbol{u}_{t+2}$ in the 5 min discretization Koopman model. To perform the upscaling of the Koopman model from a 5 min time step to a 15 min time step, we derive the transformed system dynamics by concatenating the predictions of the original model over three consecutive time steps with constant input. The following equations describe the stepwise progression for the latent state $\boldsymbol{z}_t$ and output $\boldsymbol{y}_t$ variables over three time steps, with constant control input $\boldsymbol{u}_t$ over the interval. This allows us to predict the system's behavior at the larger time scale while maintaining consistency with the original model's dynamics.

$$\boldsymbol{z}_{t+1} = \boldsymbol{A}\boldsymbol{z}_t + \boldsymbol{B}\boldsymbol{u}_t \tag{5.2a}$$
$$\boldsymbol{z}_{t+2} = \boldsymbol{A}(\boldsymbol{A}\boldsymbol{z}_t + \boldsymbol{B}\boldsymbol{u}_t) + \boldsymbol{B}\boldsymbol{u}_t = \boldsymbol{A}^2\boldsymbol{z}_t + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{B}\boldsymbol{u}_t \tag{5.2b}$$
$$\begin{aligned}\boldsymbol{z}_{t+3} &= \boldsymbol{A}(\boldsymbol{A}^2\boldsymbol{z}_t + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{B}\boldsymbol{u}_t) + \boldsymbol{B}\boldsymbol{u}_t = \boldsymbol{A}^3\boldsymbol{z}_t + \boldsymbol{A}^2\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{B}\boldsymbol{u}_t \\ &= \boldsymbol{A}^3\boldsymbol{z}_t + (\boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B})\boldsymbol{u}_t = \boldsymbol{A}_{15\text{min}}\boldsymbol{z}_t + \boldsymbol{B}_{15\text{min}}\boldsymbol{u}_t\end{aligned} \tag{5.2c}$$
$$\Rightarrow \boldsymbol{A}_{15\text{min}} = \boldsymbol{A}^3, \boldsymbol{B}_{15\text{min}} = \boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B} \tag{5.2d}$$

$$\boldsymbol{y}_t = \boldsymbol{D}\boldsymbol{z}_t + \boldsymbol{E}\boldsymbol{u}_t \tag{5.3a}$$
$$\boldsymbol{y}_{t+1} = \boldsymbol{D}\boldsymbol{z}_{t+1} + \boldsymbol{E}\boldsymbol{u}_t \tag{5.3b}$$
$$\boldsymbol{y}_{t+2} = \boldsymbol{D}\boldsymbol{z}_{t+2} + \boldsymbol{E}\boldsymbol{u}_t \tag{5.3c}$$
$$\begin{aligned}\boldsymbol{y}_{t+3} &= \boldsymbol{D}\boldsymbol{z}_{t+3} + \boldsymbol{E}\boldsymbol{u}_t = \boldsymbol{D}(\boldsymbol{A}^3\boldsymbol{z}_t + \boldsymbol{A}^2\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{A}\boldsymbol{B}\boldsymbol{u}_t + \boldsymbol{B}\boldsymbol{u}_t) + \boldsymbol{E}\boldsymbol{u}_t \\ &= \boldsymbol{D}\boldsymbol{A}^3\boldsymbol{z}_t + (\boldsymbol{D}\boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{D}\boldsymbol{A}\boldsymbol{B} + \boldsymbol{D}\boldsymbol{B} + \boldsymbol{E})\boldsymbol{u}_t = \boldsymbol{D}_{15\text{min}}\boldsymbol{z}_t + \boldsymbol{E}_{15\text{min}}\boldsymbol{u}_t\end{aligned} \tag{5.3d}$$
$$\Rightarrow \boldsymbol{D}_{15\text{min}} = \boldsymbol{D}\boldsymbol{A}^3, \boldsymbol{E}_{15\text{min}} = \boldsymbol{D}\boldsymbol{A}^2\boldsymbol{B} + \boldsymbol{D}\boldsymbol{A}\boldsymbol{B} + \boldsymbol{D}\boldsymbol{B} + \boldsymbol{E} \tag{5.3e}$$

5.3. Numerical experiments

We obtain an initial guess for the Koopman model via the iterative data sampling and system identification approach outlined in Sec. 5.2.2. Then, we repeat the end-to-end refinement five times using different fixed seeds in every training run. During training, we cycle through the historic German day-ahead electricity prices from 2023-01-01 to 2023-12-31 (EPEX (2023)).

Figure 5.4 illustrates the evolution of the reward over 200,000 environment steps for each training run. Each training run takes approximately two days on a Windows 11 workstation with an Intel Core i7-14700 CPU. After training has completed, our final performance evaluation is based on the policy that attained the highest average reward during a policy

rollout. Consequently, our primary metric of interest is the maximum average reward achieved by each training run, rather than the reward at the final training step. Four out of five training runs demonstrate a substantial improvement in the policy's average reward compared to the initial guess. After training, we test the performance of the policy

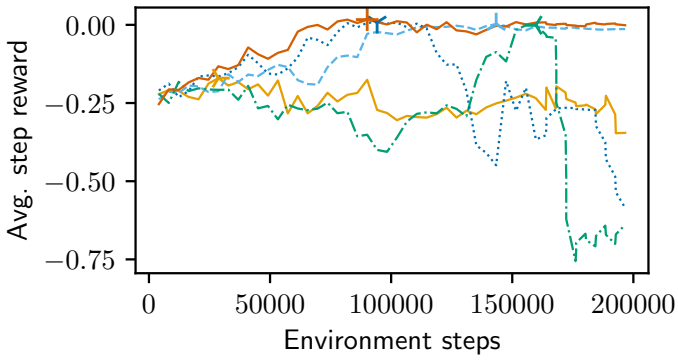


Figure 5.4.: Learning progress during end-to-end model refinement. Each line represents one training run. For each training run, a marker depicts the highest average reward achieved in one policy rollout.

that obtained the highest average reward in a three-day test episode. The electricity price trajectory used for testing was generated using the method by Papadimitriou et al. (2024), which constructs representative price profiles from historical data. Specifically, hourly prices from the entire year 2023 were averaged and then scaled to preserve key statistical properties, including the mean and variance of price fluctuations. Importantly, these representative prices are not identical to any specific consecutive days from the 2023 historical data used for training, thus preventing overly optimistic results due to overfitting to particular past events.

In the test episode, the eNMPC employing the Koopman model obtained solely via system identification (from hereon called *Koopman-SI* policy) saves 5 % of electricity cost compared to steady-state production while producing small constraint violations in 17.7 % of the time steps, resulting in an average reward of −0.14. The eNMPC employing the end-to-end refined Koopman model (*Koopman-PPO*) also produces 5 % cost savings, however, it does so without violating any constraints in the test episode, thus yielding an average reward of 0.03.

Figure 5.5 illustrates the behavior of the policies in the three-day test episode. We show the evolution of ΔT_{IRC} and N_R, since these are the only states where constraint violations occur, as well as the evolution of the energy demand E. Both policies exhibit an intuitive inverse relationship between the electricity price and the energy demand. It can be seen that the overall behavior of the policies is quite similar, although the ΔT_{IRC} and E trajectories of *Koopman-SI* exhibit high-frequency oscillations which do not occur in *Koopman-PPO*. Moreover, it is apparent that most constraint violations by *Koopman-SI* are violations of the lower bound of ΔT_{IRC}, with only one constraint violation occurring regarding the lower bound of N_R.

As stated above, the architecture of the Koopman models that we use (see Figure 5.2) results in convex OCPs, which are relatively easy to solve. In the 72 hour test episode,

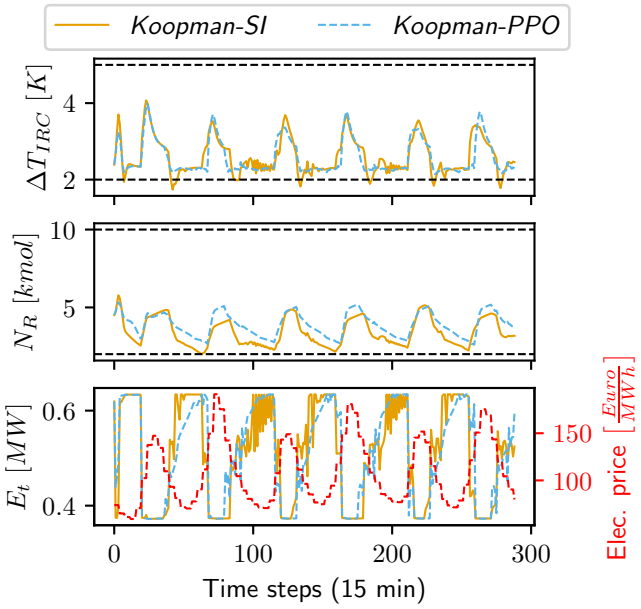


Figure 5.5.: Comparison of the control behavior of *Koopman-SI* and *Koopman-PPO*.

i.e., 288 control steps of 15 min length, the inference time of the *Koopman-PPO* policy was between 0.3 s and 9.2 s with an average time of 1.5 s.

5.4. Conclusion

We apply our previously published method (Mayfrank et al. (2024)) for end-to-end learning of Koopman surrogate models for (e)NMPC applications to a high-dimensional nonlinear demand response problem based on a mechanistic model of an ASU. We assume full observability of the ASU, which, in a real-world application, would not be available. In such a setting, our approach would either have to operate using only measurable outputs, or be coupled with state estimation techniques like Kalman filtering (Kalman et al. (1960)) or Moving Horizon Estimation (Allgöwer et al. (1999)), thus making the overall control problem more difficult.

We find that out of the five independent end-to-end training runs, four improve the performance of the resulting eNMPC policy substantially compared to the system identification baseline, avoiding the constraint violations caused by the baseline policy while retaining the economic performance. Still, our method does not provide any safety guarantees.

Our method produces data-driven predictive control policies that strike an exceptional balance between control performance (high economic performance and consistent constraint satisfaction) and computational efficiency (policy inference time $\ll$ sampling time). By demonstrating its scalability to large problems, we show that our method could be applicable to complex real-world control problems where mechanistic predictive control policies are not real-time capable and system identification-based data-driven controllers produce unsatisfactory performance.

While this chapter shows that, in principle, end-to-end learning of Koopman eNMPCs could be scalable to interesting real-world use cases, the critical issues of poor convergence

and low sample efficiency identified in Chapter 4 remain. The following two chapters therefore investigate strategies to overcome these limitations. Chapter 6 explores how the use of differentiable simulation environments can improve training convergence. Chapter 7 shifts the focus to improving sample efficiency through integration of model-based RL and physics-informed model training.

6. Leveraging differentiable simulation for end-to-end learning of Koopman eNMPCs

In the last two chapters, we showed that RL can be used to learn data-driven MPC-based policies that outperform similar policies whose models are learned solely via system identification (SI). Like other methods for RL-based learning of MPC-based policies (e.g., Chen et al. (2019); Yin et al. (2022); Iwata and Kawahara (2022)), we used an (actor-critic) policy gradient algorithm (Schulman et al. (2017)). However, as outlined in Section 3.4, the empirical behavior of these algorithms is not well understood and problems can arise from poor policy gradient estimates. We suspect that accurate policy gradient estimation may be especially difficult given an MPC-based policy, since minimal changes to parameters or system states can lead to constraints becoming active/inactive, resulting in a sudden change in the direction of the policy gradient. This may explain why, even though we improved upon the purely SI-based policy, policy training using PPO did not exhibit good convergence (see Figure 4.5).

In contrast to algorithms that use policy gradient estimates based on the policy gradient theorem, recently, policy optimization algorithms that leverage the derivative information from automatically differentiable simulated environments were designed, e.g., the Policy Optimization via Differentiable Simulation (PODS) algorithm (Mora et al. (2021)) and the SHAC algorithm (Xu et al. (2022)) (see Section 2.3.3). These algorithms manage to avoid the well-known problems of Backpropagation Through Time (BPTT) (Werbos (1990)), i.e., noisy optimization landscapes and exploding/vanishing gradients (Huang et al. (2021); Xu et al. (2022)), and have shown enhanced training wall-clock time efficiency and increased terminal performance compared to state-of-the-art RL algorithms that do not exploit derivative information from the environment. These algorithmic advances could also benefit the learning of dynamic *surrogate* models for (eN)MPC if the mechanistic simulation model is differentiable. To this end, differentiable simulators, e.g., (Chen et al. (2018)), can be used to construct simulated RL environments with automatically differentiable dynamics and reward functions, thus enabling the use of analytic gradients for policy optimization. Nevertheless, policy optimization using differentiable environments has yet to be established for the learning of dynamic surrogate models for (eN)MPC.

By combining our previously proposed method for end-to-end learning of task-optimal Koopman models in (e)NMPC applications (Mayfrank et al. (2024)) with the SHAC algorithm (Xu et al. (2022)), this chapter presents a method for end-to-end optimization of Koopman *surrogate* models. Crucially, our method exploits the differentiability of simulated environments, distinguishing it from previous contributions, which are based on RL (Chen et al. (2019); Gros and Zanon (2019); Mayfrank et al. (2024)) or imitation learn-

ing (Amos et al. (2018)). We evaluate the resulting control performance on an eNMPC case study derived from a literature-known continuous stirred-tank reactor model (Flores-Tlacuahuac and Grossmann (2006)). We compare the performance to that of eNMPCs employing Koopman surrogate models that were trained either using (*i*) system identification or (*ii*) RL. We find that the novel combination of a Koopman-eNMPC trained using SHAC exhibits superior performance compared to the other options. This finding confirms our expectation that the advantages of policy optimization algorithms that leverage derivative information from differentiable environments can apply to the end-to-end training of dynamic surrogate models for predictive control applications. Thus, our work constitutes a step towards more performant, real-time capable optimal control policies for large-scale, nonlinear systems, where optimal control policies based on a mechanistic model are not real-time capable.

We structure the remainder of this chapter as follows: Section 6.1 presents our method. Section 6.2 showcases the performance of our method on a simulated case study and discusses the results. Section 6.3 draws some final conclusions.

6.1. Method

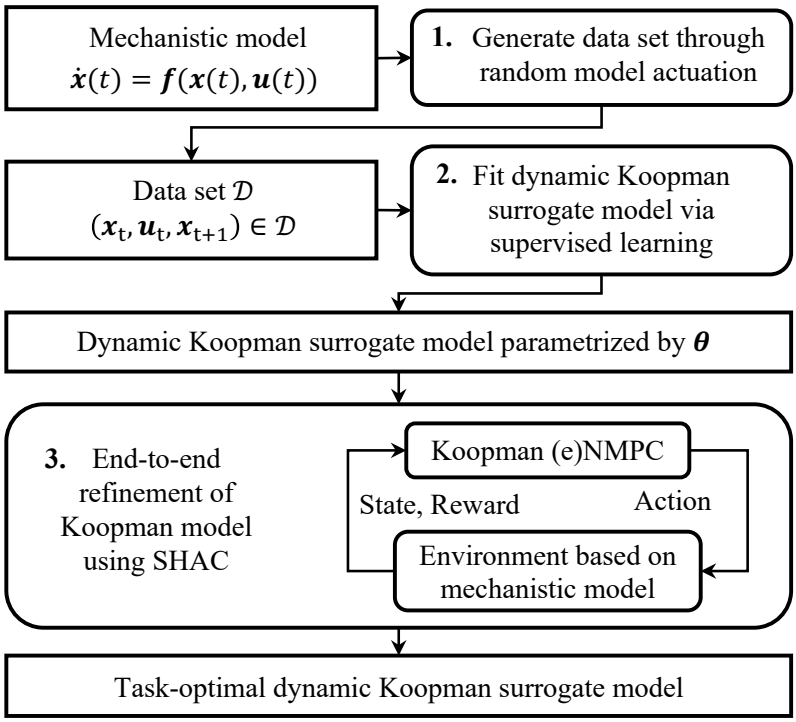


Figure 6.1.: Workflow from mechanistic model to task-optimal dynamic Koopman surrogate model. SHAC relies on simulation gradients from an automatically differentiable training environment. Therefore, this approach is only feasible if a differentiable (mechanistic) process model is available. Adapted from Mayfrank et al. (2024).

We aim to exploit the differentiability of continuous-time mechanistic models of the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{6.1}$$

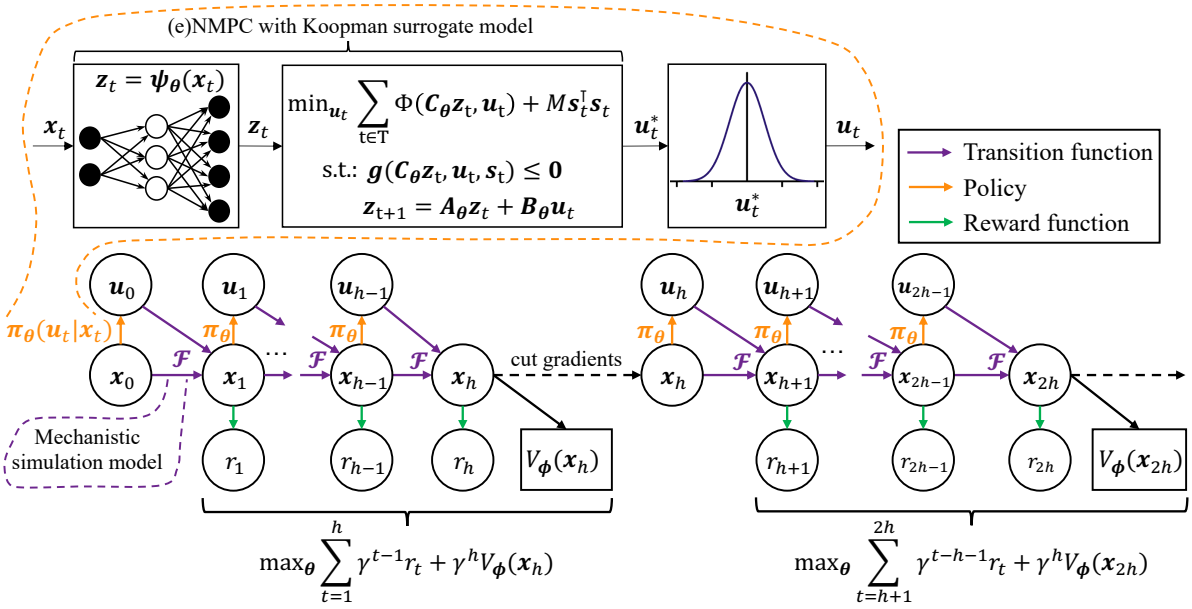


Figure 6.2.: Using SHAC to train a task-optimal Koopman surrogate model for the transition function. This figure can be interpreted as a SHAC-specific unrolled version of the typical RL loop shown in the third step in Figure 6.1. The policy is optimized by adjusting the parameters $\boldsymbol{\theta}$ of the dynamic Koopman surrogate model. Φ is a convex function for the stage cost of the objective function. To ensure the feasibility of the resulting optimal control problems, we add slack variables $\boldsymbol{s}_t$ to the state bounds (Mayfrank et al. (2024)). Their use is penalized quadratically using a penalty factor M. Due to the use of PyTorch (Paszke et al. (2019)) and *cvxpylayers* (Agrawal et al. (2019a)), the output $\boldsymbol{u}_t$ of the policy is differentiable with respect to $\boldsymbol{x}_t$ and $\boldsymbol{\theta}$. The critic is a feedforward neural network with trainable parameters $\boldsymbol{\phi}$. To increase the clarity of the figure, we omit the direct dependence of r_{t+1} with respect to $\boldsymbol{u}_t$.

for the end-to-end learning of task-optimal discrete-time dynamic surrogate models. In our previous publication (Mayfrank et al. (2024)), we present a method for end-to-end RL of data-driven Koopman models for optimal performance in (e)NMPC applications, based on viewing the predictive controller as a differentiable policy and training it using RL. This method is independent of the specific policy optimization algorithm. Therefore, by replacing the RL algorithm (Schulman et al. (2017)) with SHAC (Xu et al. (2022)), we can take advantage of policy optimization algorithms that exploit the differentiability of simulated environments in the learning of surrogate models for dynamic optimization. Analogous to the approach we took in (Mayfrank et al. (2024)), the overall workflow (visualized in Figure 6.1) from a mechanistic model to a task-optimal dynamic Koopman surrogate model consists of three steps: (*i*) We generate a data set of the system dynamics by simulating the mechanistic model using randomly generated control inputs. (*ii*) Following the model structure proposed by Korda and Mezić (2018), we fit a Koopman model (Koopman (1931)) with learnable parameters $\boldsymbol{\theta}$ to the data. (*iii*) Using the mechanistic process model (Eq. 6.1) and a differentiable simulator (Chen et al. (2018)), we construct a differentiable RL environment whose reward formulation incentivizes task-optimal controller performance on a specific control task. For instance, in an eNMPC application, the task-specific reward should depend on operating costs and potential constraint violations, not on the prediction accuracy of the dynamic model, which is used as part of the predictive controller.

Using the differentiable environment, we fine-tune the Koopman model for task-optimal performance as part of a predictive controller. Figure 6.2 provides a conceptual sketch of this fine-tuning process. To ensure exploration in the training process, we add Gaussian noise to the – otherwise deterministic – output of the Koopman eNMPC policy. For a detailed description of steps one and two in Figure 6.1 (data generation and SI) and how to construct a differentiable eNMPC policy from a Koopman surrogate model, we refer the reader to our previous work (Mayfrank et al. (2024)). Note that the data generation and SI steps do not need to be highly optimized since they only produce an initial guess for the end-to-end refinement step, which mainly determines the final control performance.

6.2. Numerical experiments

6.2.1. Case study description

Analogous to Chapter 4, we construct an eNMPC case study based on the dimensionless CSTR model by Flores-Tlacuahuac and Grossmann (2006). Please refer to Section 4.2.1 for a detailed case study description.

To enable the use of analytic gradients in policy optimization, we use the automatically differentiable ODE integrator by Chen et al. (2018) inside the RL training environment.

6.2.2. Training setup

We compare the performance of the following three training paradigms:

1. *Koopman-SI*: eNMPC controller using a Koopman surrogate model trained solely using SI.
2. *Koopman-PPO*: eNMPC controller using a Koopman surrogate model pretrained using SI and refined for task-optimal performance using the state-of-the-art policy gradient algorithm Proximal Policy Optimization (PPO) (Schulman et al. (2017)), like we did in Mayfrank et al. (2024).
3. *Koopman-SHAC* (main contribution of this work): eNMPC controller using a Koopman surrogate model pretrained using SI and refined for task-optimal performance using the SHAC algorithm (Xu et al. (2022)).

Our goal is to train dynamic surrogate models of fixed size for optimal performance as part of eNMPC. All results presented in Subsection 6.2.3 are obtained using a Koopman model with a latent space dimensionality of eight, i.e., $\boldsymbol{A_\theta} \in \mathbb{R}^{8\times8}, \boldsymbol{B_\theta} \in \mathbb{R}^{8\times2}, \boldsymbol{C_\theta} \in \mathbb{R}^{2\times8}$, and an encoder $\boldsymbol{\psi}\colon \mathbb{R}^2 \to \mathbb{R}^8$ in the form of a feedforward neural network with two hidden layers, four and six neurons, respectively and hyperbolic tangent activation functions. We use an eNMPC horizon of nine hours.

We use the same data set as in (Mayfrank et al. (2024)) for the SI pretraining of the Koopman surrogate model. This data set consists of 84 trajectories, each of a length of 5 days, i.e., 480 time steps, using a step length of 15 minutes. Of those 84 trajectories, we use 63 for training and the remaining 21 for validation. We perform SI by minimizing the sum of the three loss terms presented in Subsection 2.4 (Eqs. (2.10)-(2.12)). We repeat SI ten times using random seeds. We use the model with the lowest validation loss for

the *Koopman-SI* controller. The same model is used as the initial guess when training the *Koopman-PPO* and *Koopman-SHAC* controllers.

In order to use a policy optimization algorithm such as SHAC, which makes use of derivative information from a simulated environment, not only the dynamic model but also the reward function must be differentiable. For our case study, we choose a reward function that calculates the reward at time step t based on whether any constraints were violated at that time step, and on the electricity cost savings compared to the steady-state production at nominal rate between $t-1$ and t. The constraint component r_t^{con} of the reward quadratically penalizes violations of the bounds of c, T, and the product storage, i.e., $r_t^{\text{con}} \geq 0$, and $r_t^{\text{con}} = 0$ if no constraint violation occurs at t. The cost-component r_t^{cost} of the reward is calculated by taking the difference between the cost at nominal production and the actual production cost between $t-1$ and t, i.e.,

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

where p_{t-1} is the electricity price between $t-1$ and t, and Δt_{ctrl} is the time between $t-1$ and t for which the controls are held constant. We balance the influence of the two components on the overall reward using a hyperparameter α:

$$r_t = \alpha \cdot r_t^{\text{cost}} - r_t^{\text{con}}$$

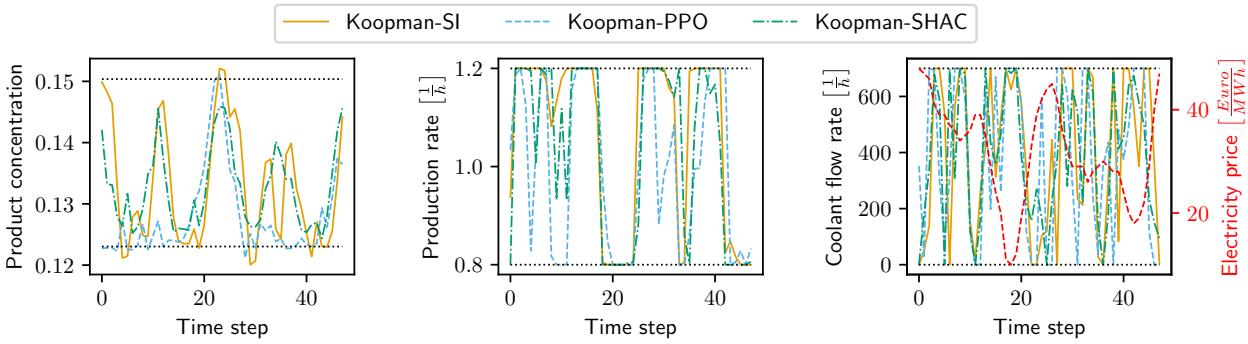


Figure 6.3.: Comparison of the controllers. We show a randomly chosen two-day interval from the test episode. The dotted black lines represent the bounds.

We train the policies using day-ahead electricity prices from the Austrian market from March 29, 2015, to March 25, 2018 (Open Power System Data (2020)). Using random seeds, we repeat the controller training ten times for each combination of policy and training algorithm (except for the *Koopman-SI* controller, whose Koopman model is not trained any further after SI).

We use the same hyperparameters for *Koopman-PPO* and *Koopman-SHAC* whenever possible, i.e., for all hyperparameters which are not part of PPO or SHAC. For the hyperparameters of PPO and SHAC, we do not perform extensive hyperparameter tuning. Instead, we mostly rely on the standard values. In both algorithms, we use the Adam optimizer Kingma and Ba (2014) with a small learning rate of 10^{-5} as the behavior of the Koopman eNMPCs is highly sensitive to small changes in the parameters. We used the *Stable-Baselines3* (Raffin et al. (2021)) implementation of PPO but implemented our own version

of SHAC following the description in (Xu et al. (2022)). All code used for training the controllers, including the hyperparameters that were used to obtain the results presented in Subsection 6.2.3, is publicly available[1]. In addition to the code used to obtain the results presented in the following sections, the linked repository also contains code for training pure neural network policies for our case study using PPO and SHAC. The results of the neural network policies do not influence the narrative of this work in a meaningful way. Therefore, and to maximize the conciseness of our contribution, we do not discuss the results of the neural network policies further.

6.2.3. Results

For *Koopman-PPO* and *Koopman-SHAC*, we identify the controller (and the associated set of parameters) that achieved the highest average reward between two consecutive parameter updates. We test their performance and that of the *Koopman-SI* controller on a continuous, roughly half-year-long test episode using Austrian day-ahead electricity price data from March 26, 2018, to September 30, 2018 (Open Power System Data (2020)). Unlike the training process, we perform this testing without exploration, i.e., we waive adding Gaussian noise to the controller output (cf. Section 6.1). We initialize the test episode for each controller at the steady state of the CSTR and with empty product storage. The results are presented in Table 6.1. We also visualize the behavior of each controller by randomly choosing a two-day interval from the test episode and plotting the product concentration c, the production rate ρ, and the coolant flow rate F during this interval in Figure 6.3. It can be noted that all controllers leverage the entire feasible range of the control variables and the product concentration, and that the results exhibit an intuitive inverse relationship between electricity prices and coolant flow rate. We forego including the temperature T in Figure 6.3 as T never reaches its bounds in any of the test episodes.

Table 6.1.: Test results: The economic cost is stated relative to the nominal production cost, i.e., we report the cost incurred by the respective controller, divided by the cost incurred by steady-state production at nominal rate given the same electricity price trajectory. The percentage of control steps that result in constraint violations is given. The average size of a constraint violation is given relative to the size of the feasible range of the variable whose bound was violated.

	Economic cost	Constr. viols. [%]	Avg. size constr. viol.
Koopman-SI	**0.88**	19.88	$5.1 \cdot 10^{-2}$
Koopman-PPO	0.90	17.57	$1.3 \cdot 10^{-2}$
Koopman-SHAC	0.90	**0.0**	–

As can be seen from Table 6.1, *Koopman-SHAC* is the only controller that does not produce any constraint violations. *Koopman-SI* achieves slightly higher cost savings than *Koopman-SHAC*, however, it frequently produces constraint violations. Therefore, we consider *Koopman-SHAC* preferable in most real-world applications where constraint-satisfaction is of high importance.

[1] `https://jugit.fz-juelich.de/iek-10/public/optimization/shac4koopmanenmpc`

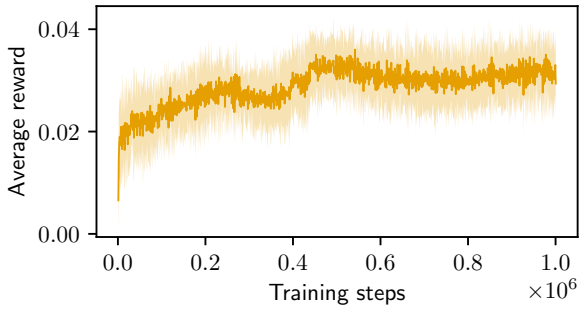


Figure 6.4.: Learning progress in the *Koopman-SHAC* training runs. The dark orange line indicates the running average reward over the previous 1024 steps in the environment, averaged over all ten training runs. The light orange region indicates one standard deviation of the performance variance between the training runs.

It is noticeable that *Koopman-SI* and *Koopman-PPO* cause substantially more constraint violations than the other controllers. In the case of *Koopman-SI*, this is not surprising since the employed Koopman surrogate model received no end-to-end training for task-optimal performance. Here, frequent but minor constraint violations show that the controller is trying to operate at the borders of the feasible range, which is normal behavior for a predictive controller. However, in the case of *Koopman-PPO*, the results are unsatisfactory and rather unexpected. Here, the end-to-end training reduced the average size of the constraint violations by roughly a factor of four, but only led to a small improvement in the frequency of constraint violations. We observe that some training runs did not improve performance compared to *Koopman-SI* at all. Moreover, those training runs that did improve performance did not show stable convergence to high rewards. The unstable convergence of the *Koopman-PPO* controllers is in line with the results in our previous work (Mayfrank et al. (2024)). There, we used a non-differentiable reward formulation with a high constant penalty incurred by any constraint violation. Using that reward formulation, we managed to substantially reduce the frequency of constraint violations, unlike here. However, that approach (Mayfrank et al. (2024)) severely affected the resulting economic performance and thus produced overly conservative eNMPC controllers. In contrast to *Koopman-PPO*, *Koopman-SHAC* exhibits a relatively stable convergence to high rewards in our case study (Figure 6.4) and produces superior terminal performance (Table 6.1, Figure 6.3). The control performance improves relatively evenly in all ten training runs without ever dropping for extended periods.

Due to the small size of the case study under consideration, a detailed analysis of the training runtimes would provide little insight about the expected runtime on a control problem of more practical relevance. Therefore, we leave such an analysis for future work on larger systems.

6.2.3.1. Policy gradient analysis

The fundamental difference between the two policy optimization algorithms SHAC (Xu et al. (2022)) and PPO (Schulman et al. (2017)) is that SHAC utilizes analytic policy gradients from an automatically differentiable environment, whereas PPO estimates policy gradients via the policy gradient theorem (Sutton and Barto (2018)). Ilyas et al. (2018)

show that given common and practical hyperparameter configurations, the PPO-estimated policy gradients can incur a high variance which can lead to unstable training convergence, as we observe in our case.

We analyze empirically whether there is a meaningful difference in the variance of the policy gradients produced by PPO and SHAC when applied to our case study. Even though SHAC uses analytical policy gradients, these still incur some variance due to the exploration noise in the policy. Following Ilyas et al. (2018), we investigate how "tightly concentrated" the gradients are, i.e., how similar the direction of multiple policy gradients are given a fixed policy parameterization. To this end, we take the following three-step approach for each combination of policy type and training algorithm: (i) For a given policy, we first fit the critic to the policy by running the training algorithm for 100 updates *without* updating the policy. This first step is performed to prevent a bad (randomly initialized) critic from distorting the results of the following analysis. In our case, pretraining the critic for 100 updates is more than enough for the critic to converge to low losses. (ii) In a second step, we continue running the training algorithm for another 100 updates. Again, we do *not* update the policy. Instead, we record the 100 resulting policy gradients, which would have been used for updates during normal training. (iii) Finally, we calculate how similar the 100 recorded gradients are by computing their average pairwise cosine similarity. The cosine similarity is a measure of the similarity of two vectors, which only depends on their direction, not on their length. It takes a value of one if both vectors point in the same direction, zero if they are orthogonal to each other, and minus one if they point in exactly opposite directions. We fix the policy parameters to those of the *Koopman-SI* controller for this analysis, as both *Koopman-PPO* and *Koopman-SHAC* use this parameterization as the initial guess in their training runs.

Given the approach described above, PPO produces an average cosine similarity of 0.22, whereas SHAC results in an average cosine similarity of 0.94. As expected, for a static policy, the directions of policy gradients produced by SHAC are much more tightly concentrated than those resulting from PPO. We assume that the low similarity of policy gradient directions produced by PPO is the reason for the unstable convergence to high rewards and thus the relatively bad performance of *Koopman-PPO* (see Table 6.1).

6.3. Conclusion

We combine our previously presented method for turning Koopman-(e)NMPC controllers into differentiable policies (Chapter 4) with the policy optimization algorithm SHAC (Xu et al. (2022)). Our approach leverages derivative information from automatically differentiable simulated environments, differentiating it from previously published methods for end-to-end training of dynamic models for control. We find that SHAC produces a stable convergence to high control performance across all independent training instances, translating to superior control performance compared to our original approach (Chapter 4), which was based on the PPO algorithm. Our analysis indicates that this superior performance could be due to a substantially lower variance in the direction of the policy gradient estimates, which results from the exploitation of analytic derivative information from an automatically differentiable environment. Note that even though our method achieves perfect constraint satisfaction in our case study, it does not provide any safety guarantees.

The results can be understood as a successful proof of concept. By training data-driven surrogate models for optimal performance in a specific control task, our method utilizes the representational capacity of the model efficiently, thus avoiding unnecessarily large and computationally expensive surrogate models. We view our approach as a promising avenue toward more performant real-time capable data-driven (e)NMPCs. Still, the computational burden of backpropagation through mechanistic simulations and optimal control problems, and thus the cost of each training iteration, is naturally linked to the size of the mechanistic simulation model, meaning that our method could become computationally intractable for very large models. Thus, future work should investigate the application of our method to larger mechanistic simulation models and more challenging control problems, presumably necessitating training for more iterations.

The method developed in this chapter is promising for applications where a task-optimal surrogate model for a control application is to be learned based on a differentiable mechanistic model. However, a mechanistic model may not always be available, necessitating learning directly from the real process. In such a setting, the sample efficiency becomes critical since interactions with a physical process for learning purposes are costly. Therefore, in the next chapter, we study how the sample efficiency can be maximized if no or only partial prior knowledge about the environment dynamics is available.

7. Accelerating end-to-end learning of Koopman eNMPC using physics-informed model-based RL

As shown in the previous two chapters, using RL, data-driven (eN)MPCs can be trained for optimal performance in a specific control task (see Figure 7.1a). To this end, a differen-

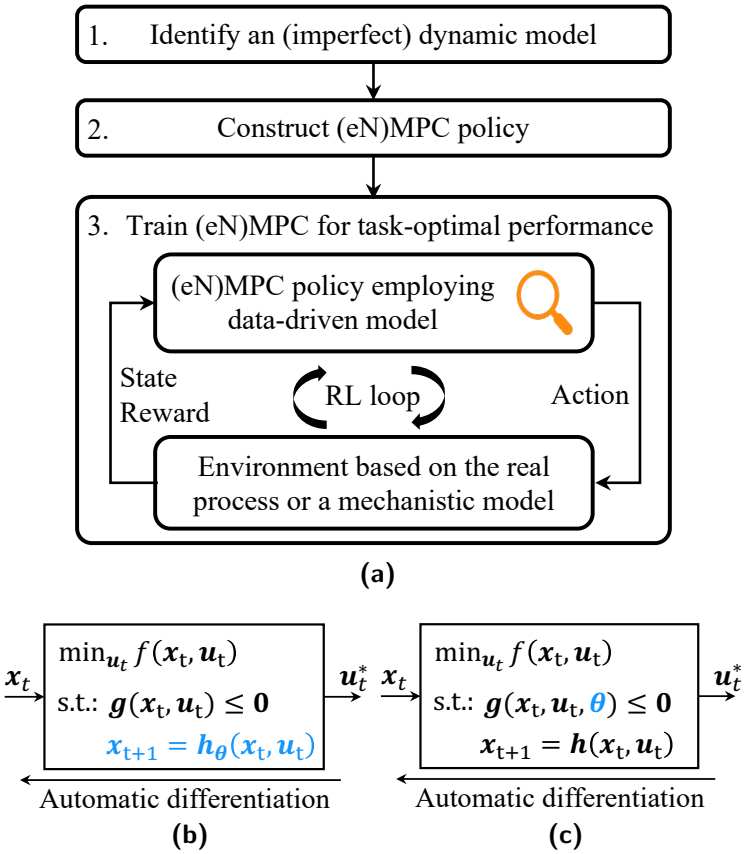


Figure 7.1.: (a) Procedure for RL-based training of an (eN)MPC. Analogous to Figure 4.1. Subfigures (b) and (c) show alternative learnable policies which can be used in (a): (b) shows a differentiable eNMPC policy parameterized by the parameters $\boldsymbol{\theta}$ of the dynamic model. The policy takes as input the current state $\boldsymbol{x}_t$ and computes the optimal control action $\boldsymbol{u}_t^*$ based on the minimization of a cost function f, subject to inequality constraints $\boldsymbol{g}$, and the learnable dynamic model $\boldsymbol{h}_{\boldsymbol{\theta}}$. (c) shows a differentiable eNMPC policy with parameterized inequality constraints $\boldsymbol{g}$, e.g., state bounds. Training this policy leaves the underlying dynamic model $\boldsymbol{h}$ unchanged but adapts the inequality constraints to counteract model-plant mismatch.

tiable (e)NMPC policy is constructed, wherein the learnable parameters can be parameters

of the dynamic model (see Figure 7.1b, e.g., Mayfrank et al. (2024, 2025a)), 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 Figure 7.1c). 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 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.

The adoption of RL algorithms into the process control community is still in a relatively early stage 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. (2021b); Gao and Wang (2023); 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); Mayfrank et al. (2024, 2025a)). 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 Chapter 4, we introduced a method for RL training of (e)NMPCs that utilize data-driven Koopman models. However, due to its poor sample efficiency, 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 settings, where the agent has to learn from (costly) interactions with the physical system. In this chapter, we extend the applicability of RL-based training of Koopman (e)NMPCs to settings where sample efficiency is critical by combining our previously presented 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 chapter, 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. (2024)) 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 chapter is structured as follows: Section 7.1 presents our method. Section 7.2 presents the results of the numerical experiments that we conducted on a simulated case study. Section 7.3 draws final conclusions and discusses promising directions for future research.

7.1. Method

In Chapters 4 and 6, we aim to optimize a Koopman model of the form of Eqs. (2.9) for optimal performance as part of an (e)NMPC in a specific control task. However, as noted before, 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 remainder of this chapter, $\boldsymbol{\theta}_\mathrm{K}$ refers to the parameters of the Koopman model, which appear as $\boldsymbol{\theta}$ in Eqs. (2.9). Additionally, we introduce the parameters $\boldsymbol{\theta}_\mathrm{B}$, which modify the state bounds of the eNMPC. Both types of parameters influence the behavior of the policy, i.e., $\boldsymbol{\theta} = [\boldsymbol{\theta}_\mathrm{K}^\intercal, \boldsymbol{\theta}_\mathrm{B}^\intercal]^\intercal$ and $\boldsymbol{\pi}_{\boldsymbol{\theta}}(\boldsymbol{u}_t|\boldsymbol{x}_t)\colon \mathbb{R}^n \mapsto \mathbb{R}^m$.

Integrating the idea of state bound adaptation into the differentiable Koopman (e)NMPC framework (Chapter 4) is straightforward. Given an (e)NMPC horizon of $t_f + 1$ steps with the corresponding sets $\mathrm{T}_{+1} = \{t, \dots, t + t_f\}$ and $\mathrm{T} = \{t, \dots, t + t_f - 1\}$, a convex OCP is

solved to obtain the optimal action $\boldsymbol{u}_t^*$:

$$\min_{(\boldsymbol{u}_t)_{t\in \mathrm{T}}} \sum_{t\in \mathrm{T}_{+1}} \Phi(\boldsymbol{C}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{z}_t, \boldsymbol{u}_t) + M \boldsymbol{s}_t^\intercal \boldsymbol{s}_t, \tag{7.1a}$$

$$\text{s.t. } \boldsymbol{z}_{t+1} = \boldsymbol{A}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{z}_t + \boldsymbol{B}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{7.1b}$$

$$\boldsymbol{g}(\boldsymbol{C}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{z}_t, \boldsymbol{u}_t, \boldsymbol{s}_t, \boldsymbol{\theta}_\mathrm{B}) \leq \boldsymbol{0} \quad \forall t \in \mathrm{T}_{+1} \tag{7.1c}$$

Φ is a convex function representing the stage cost of the objective function, and $\boldsymbol{g}$ 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. (2019a)), the output $\boldsymbol{u}_t$ of the policy is automatically differentiable with respect to $\boldsymbol{x}_t$, $\boldsymbol{\theta}_\mathrm{K}$, and $\boldsymbol{\theta}_\mathrm{B}$.

7.1.1. Model-based policy optimization of Koopman eNMPCs

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 7.2.2.

Our method iterates between the three typical Dyna steps and is visualized in Figure 7.2. 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 $\boldsymbol{u}_t \sim \boldsymbol{\mathcal{N}}(\boldsymbol{u}_t^*, \boldsymbol{\sigma}^2)$. 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. If (incomplete) physics knowledge is available, it is possible to train PINNs (Raissi et al. (2019)). Throughout this work, each ensemble member $\boldsymbol{NN}_{i,\boldsymbol{\omega}_i} \forall i \in \{1, 2, \ldots, n\}$ is a PINN parameterized by $\boldsymbol{\omega}_i$. Furthermore, we fit the parameters $\boldsymbol{\theta}_\mathrm{K}$ 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 Figure 7.4). Since the behavior of the overall Koopman eNMPC policy is optimized with respect to the physics-informed NN ensemble anyway (see Figure 7.2a, step three), and to keep our method as simple as possible, we decided against a physics-informed system identification

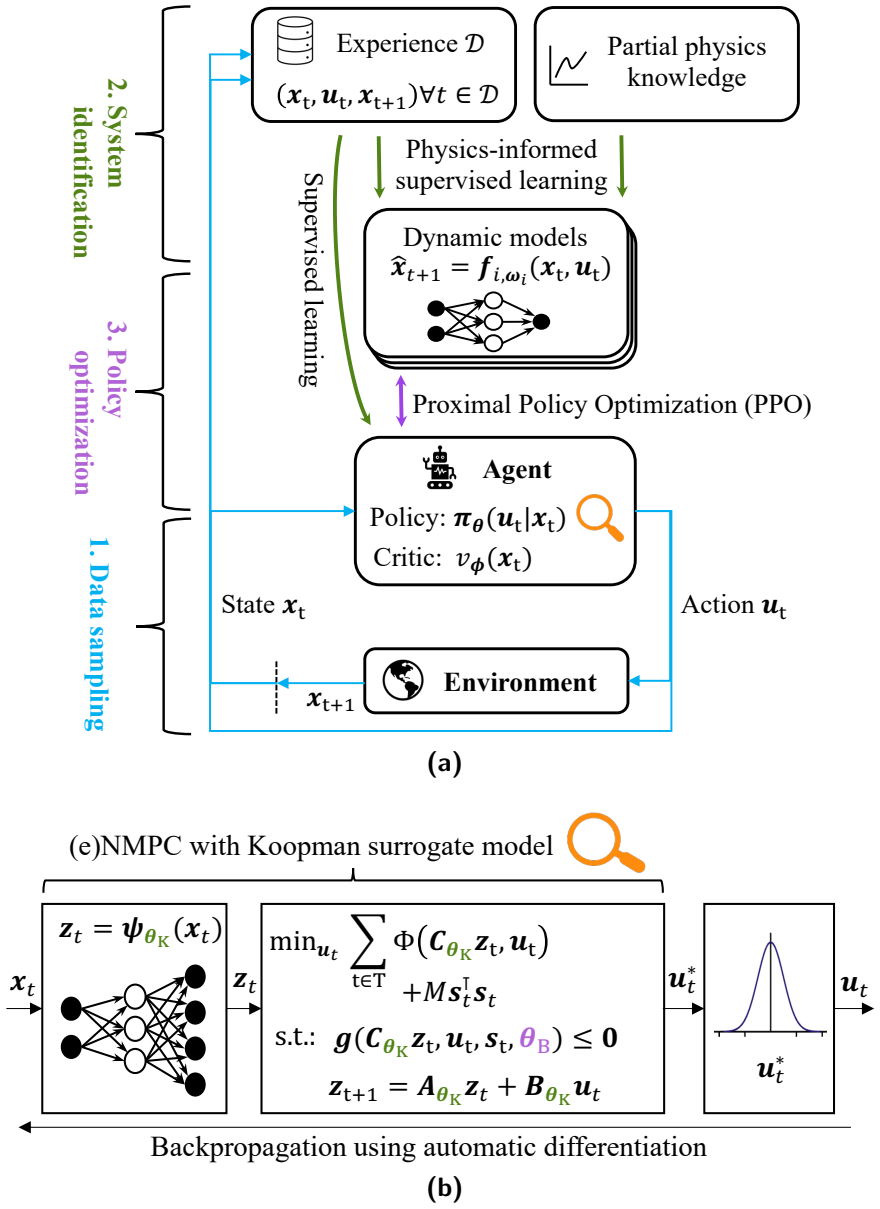


Figure 7.2.: Using MBPO to train a task-optimal Koopman (e)NMPC controller. (a) The training algorithm. The following three steps are executed in a loop until a stopping criterion is reached: First, the Koopman (e)NMPC interacts with the environment to gather data about the dynamics. Second, all data collected up to the current step is used to fit the Koopman model (parameters $\boldsymbol{\theta}_K$) and the PINN ensemble (parameters $\boldsymbol{\omega}_i \forall i \in \{1, 2, \ldots, n\}$). Third, a surrogate RL environment is constructed using the NN ensemble, and the Koopman (e)NMPC is optimized by tuning the parameters $\boldsymbol{\theta}_B$, i.e., the state bounds. (b) The automatically differentiable Koopman (e)NMPC whose behavior is defined by the parameters of the Koopman model ($\boldsymbol{\theta}_K$) and the parameters modifying the state bounds ($\boldsymbol{\theta}_B$). The parameters are color-coded to match the colors of the corresponding optimization steps in Figure 7.2a.

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. (2024) for a detailed description of how Koopman models in the form proposed by Korda and Mezić (2018) (Eq. (2.9)) can be trained using system identification.

The third step is based upon our previously published method (Mayfrank et al. (2024)) on viewing Koopman (e)NMPC policies as automatically differentiable policies (Figure 7.2b): 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 $\boldsymbol{\theta}_{\mathrm{B}}$ 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 7.2.2.4. The overall learning process can continue for a predefined number of steps in the real environment or until a satisfactory performance is achieved.

We emphasize that the behavior of the resulting (e)NMPC is defined by $\boldsymbol{\theta}_{\mathrm{K}}$ and $\boldsymbol{\theta}_{\mathrm{B}}$ and that both parameter types are optimized in each iteration of the overall algorithm (Figure 7.2): In the SI step, $\boldsymbol{\theta}_{\mathrm{K}}$ 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 $\boldsymbol{\theta}_{\mathrm{B}}$, 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 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 a lack of sufficient 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. (7.1)) 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 $\boldsymbol{\theta}_{\mathrm{B}}$ 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.

7.2. Numerical experiments

7.2.1. Case study description

Analogous to Chapters 4 and 6, we construct an eNMPC case study based on the dimensionless CSTR model by Flores-Tlacuahuac and Grossmann (2006). Please refer to Section 4.2.1 for a detailed case study description.

In this chapter, we study how to maximize the sample efficiency of RL-based end-to-end learning of Koopman eNMPCs given none or partial prior knowledge about the process dynamics. Partial knowledge of the dynamics can be utilized via physics-informed model learning (SI step in Figure 7.2a). If no prior knowledge about the dynamics is available, then the PINNs are replaced by purely data-driven NNs. For the training of the PINN ensemble in the SI step, 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. (4.2) is unknown. Thus, the physics equations for the PINN are

$$\dot{c}(t) = (1 - c(t))\frac{\rho(t)}{V} - R(t), \tag{7.2a}$$

$$\dot{T}(t) = (T_f - T(t))\frac{\rho(t)}{V} + R(t) - F(t)\alpha_c(T(t) - T_c), \tag{7.2b}$$

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

7.2.2. Implementation details

This subsection explains the implementation details of our method (Section 7.1.1) when it is applied to this case study. Section 7.2.2.1 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 7.2.2.2), system identification (Section 7.2.2.3), and policy optimization (Section 7.2.2.4). Section 7.2.2.5 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 7.2.3, is available online[1].

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

7.2.2.1. Model architecture

As in Mayfrank et al. (2024), 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 Eqs. (4.2)), this means that $\boldsymbol{A}_{\boldsymbol{\theta}_\mathrm{K}} \in \mathbb{R}^{8\times 8}, \boldsymbol{B}_{\boldsymbol{\theta}_\mathrm{K}} \in \mathbb{R}^{8\times 2}, \boldsymbol{C}_{\boldsymbol{\theta}_\mathrm{K}} \in \mathbb{R}^{2\times 8}$ (see Eqs. (2.9)). The encoder $\boldsymbol{\psi}_{\boldsymbol{\theta}_\mathrm{K}} \colon \mathbb{R}^2 \mapsto \mathbb{R}^8$ 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 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. (7.1)). Analogous to our earlier work on model-free RL of Koopman eNMPCs (Mayfrank et al. (2024)), given a prediction for the evolution of the electricity prices $\boldsymbol{p}_\mathrm{eNMPC} = (p_t)_{t\in\mathrm{T}_{+1}}$, 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., $\boldsymbol{z}_0 = \boldsymbol{\psi}_{\boldsymbol{\theta}_\mathrm{K}}(\boldsymbol{x}_0)$. Then, the following OCP is solved:

$$\min_{(\rho_t, F_t)_{t\in\mathrm{T}}} \sum_{t\in\mathrm{T}_{+1}} (F_t p_t \Delta t_\mathrm{ctrl} + M \boldsymbol{s}_t^\intercal \boldsymbol{s}_t), \tag{7.3a}$$

$$\text{s.t. } \boldsymbol{z}_{t+1} = \boldsymbol{A}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{z}_t + \boldsymbol{B}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{u}_t \quad \forall t \in \mathrm{T}, \tag{7.3b}$$

$$l_{t+1} = l_t + (\rho_t - \rho_\mathrm{ss}) \Delta t_\mathrm{ctrl} \quad \forall t \in \mathrm{T}, \tag{7.3c}$$

$$\boldsymbol{x}_t = \boldsymbol{C}_{\boldsymbol{\theta}_\mathrm{K}} \boldsymbol{z}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{7.3d}$$

$$\underline{\boldsymbol{x}}_t - \boldsymbol{s}_{\boldsymbol{x},t} + \boldsymbol{\theta}_{\mathrm{B},\underline{\boldsymbol{x}}} \le \boldsymbol{x}_t \le \bar{\boldsymbol{x}}_t + \boldsymbol{s}_{\boldsymbol{x},t} + \boldsymbol{\theta}_{\mathrm{B},\bar{\boldsymbol{x}}} \quad \forall t \in \mathrm{T}_{+1}, \tag{7.3e}$$

$$0 - s_{l,t} + \theta_{\mathrm{B},\underline{l}} \le l_t \le 6.0 + s_{l,t} + \theta_{\mathrm{B},\bar{l}} \quad \forall t \in \mathrm{T}_{+1}, \tag{7.3f}$$

$$\boldsymbol{s}_t = \begin{pmatrix} \boldsymbol{s}_{\boldsymbol{x},t} \\ s_{l,t} \end{pmatrix} \quad \forall t \in \mathrm{T}_{+1}, \tag{7.3g}$$

$$\boldsymbol{0} \le \boldsymbol{s}_t \quad \forall t \in \mathrm{T}_{+1}, \tag{7.3h}$$

$$\underline{\boldsymbol{u}}_t \le \boldsymbol{u}_t \le \bar{\boldsymbol{u}}_t \quad \forall t \in \mathrm{T} \tag{7.3i}$$

Note that $\boldsymbol{\theta}_\mathrm{B}$ appears only in the constraints regarding the state bounds of the CSTR (Eq. (7.3e)) and the storage level (Eq. (7.3f)), 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 Figure 7.3. 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., $\Delta(\boldsymbol{p}_\mathrm{eNMPC}) = \max_{t\in\mathrm{T}_{+1}}(p_t) - \min_{t\in\mathrm{T}_{+1}}(p_t)$, 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 for the output layer, which does not have an activation function, all layers have hyperbolic tangent activation functions. We picked such an architecture since it yielded substantially better results than fully connected architectures of similar overall size in preliminary testing.

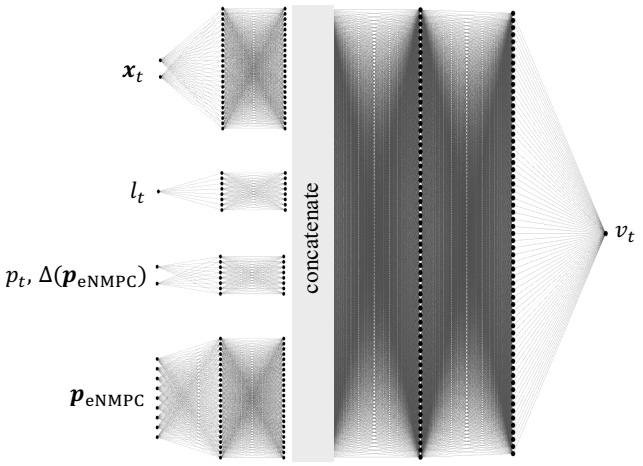


Figure 7.3.: Critic architecture. As for the policy, the system states are scaled so that the feasible range is in [-1,1], whereas the product storage and the electricity prices are left unscaled.

In the policy optimization step, we use an ensemble of $n = 10$ PINNs to model the dynamics of the real environment. Each PINN $\boldsymbol{NN}_{i,\boldsymbol{\omega}_i} \forall i \in \{1, 2, \ldots, 10\}$ 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 $\boldsymbol{x} = [c, T]^\intercal$ and one for the algebraic state $\boldsymbol{y} = [R]$. A schematic of the PINN architecture is shown in Figure 7.4. Further details on the PINN modeling approach used throughout this paper can be found in Velioglu et al. (2025).

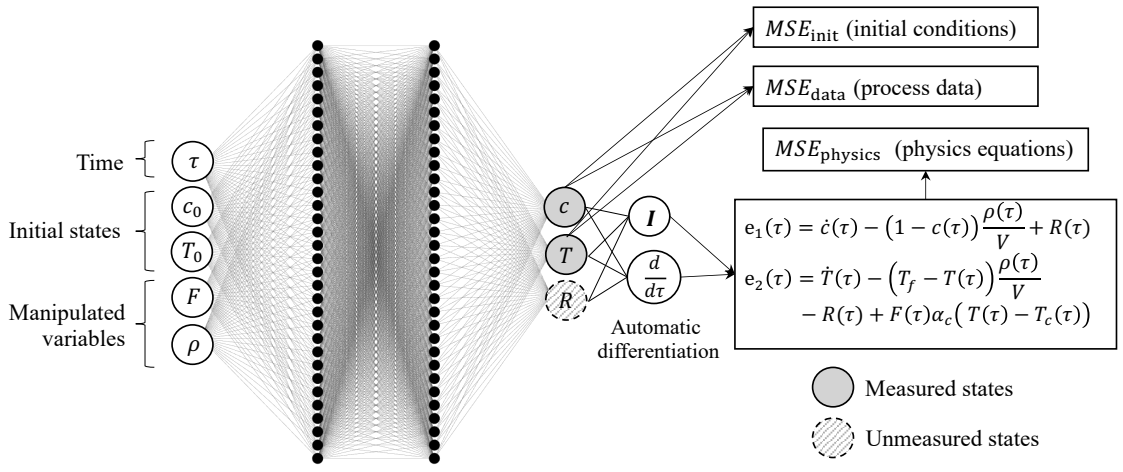


Figure 7.4.: General schematic of the PINN models used in the CSTR case study.

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.

For the purpose of data-driven modeling, we rescale the system states and control inputs (see Table 4.2) linearly so that the lower and upper bounds of each variable correspond to -1.0 and 1.0, respectively.

7.2.2.2. 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 Figure 7.2a). 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 4.2). 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 7.2.2.3), we split $\mathcal{D}$ into a training and a validation data set, i.e., $\mathcal{D} = (\mathcal{D}_\mathrm{train}, \mathcal{D}_\mathrm{val})$. In the first MBPO iteration, the data from the first episode is added to $\mathcal{D}_\mathrm{train}$, and the data from the second episode to $\mathcal{D}_\mathrm{val}$. Thereafter, at the start of each episode, we randomly determine whether the data of this episode will be added to $\mathcal{D}_\mathrm{train}$ (with a chance of 75 %) or $\mathcal{D}_\mathrm{val}$ (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 $\boldsymbol{u}_t \sim \boldsymbol{\mathcal{N}}(\boldsymbol{u}_t^*, \boldsymbol{\sigma}^2)$, 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.

The PINN requires two additional data sets for training: (i) A physics data set $\mathcal{D}_\mathrm{physics}$ is used to calculate the physics residuals. It contains unlabeled data, i.e., no output observations from the environment are needed. We sample $|\mathcal{D}_\mathrm{physics}| = 2000$ collocation points for the PINN inputs using the lower and upper bounds specified in Table 4.2 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 4.2. 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.

7.2.2.3. System identification

In the SI step (second step in Figure 7.2a), 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. (2.10) - (2.12) 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:

$$MSE_{\text{total}} = MSE_{\text{physics}} + \lambda_1 MSE_{\text{data}} + \lambda_2 MSE_{\text{init}}, \tag{7.4a}$$

$$\text{with } MSE_{\text{data}} = \frac{1}{n|\mathcal{D}_{\text{train}}|} \sum_{j=1}^{|\mathcal{D}_{\text{train}}|} (\hat{\boldsymbol{x}}(\tau_j) - \boldsymbol{x}(\tau_j))^2, \tag{7.4b}$$

$$MSE_{\text{physics}} = \frac{1}{n|\mathcal{D}_{\text{physics}}|} \sum_{j=1}^{|\mathcal{D}_{\text{physics}}|} \left(\dot{\hat{\boldsymbol{x}}}(\tau_j) - \boldsymbol{f}(\hat{\boldsymbol{x}}(\tau_j), \hat{\boldsymbol{y}}(\tau_j), \boldsymbol{u}_j)\right)^2, \tag{7.4c}$$

$$MSE_{\text{init}} = \frac{1}{n|\mathcal{D}_{\text{init}}|} \sum_{j=1}^{|\mathcal{D}_{\text{init}}|} (\hat{\boldsymbol{x}}_j(0) - \boldsymbol{x}_j(0))^2 \tag{7.4d}$$

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. (7.2)), 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. (7.4a) 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. (7.4) 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.

7.2.2.4. Policy optimization

The policy optimization step (third step in Figure 7.2a) adjusts the parameters $\boldsymbol{\theta}_\mathrm{B}$ towards 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 by evaluating it at $\tau = \Delta t_\mathrm{ctrl}$. 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 7.2.2.2), 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 $\mathcal{D}_\mathrm{train}$, 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. (2.2)) is calculated via

$$r_t = \alpha \cdot r_t^\mathrm{cost} - r_t^\mathrm{con,rel} - r_t^\mathrm{con,bool} + 1. \tag{7.5}$$

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:

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

Constraint violations are penalized twofold: $r_t^\mathrm{con,rel}$ penalizes constraint violations quadratically, i.e., $r_t^\mathrm{con,rel} \geq 0$, and $r_t^\mathrm{con,rel} = 0$ if no constraint violation occurs at t. $r_t^\mathrm{con,bool}$ imposes an additional small constant penalty whenever a constraint violation occurs, irrespective of the magnitude of the violation, i.e., $r_t^\mathrm{con,bool} = 0.1$ if there is a constraint violation, and $r_t^\mathrm{con,bool} = 0$ 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. (2024)) 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 7.1.1, 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 state transition function. 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.

7.2.2.5. Ablation and benchmark variants

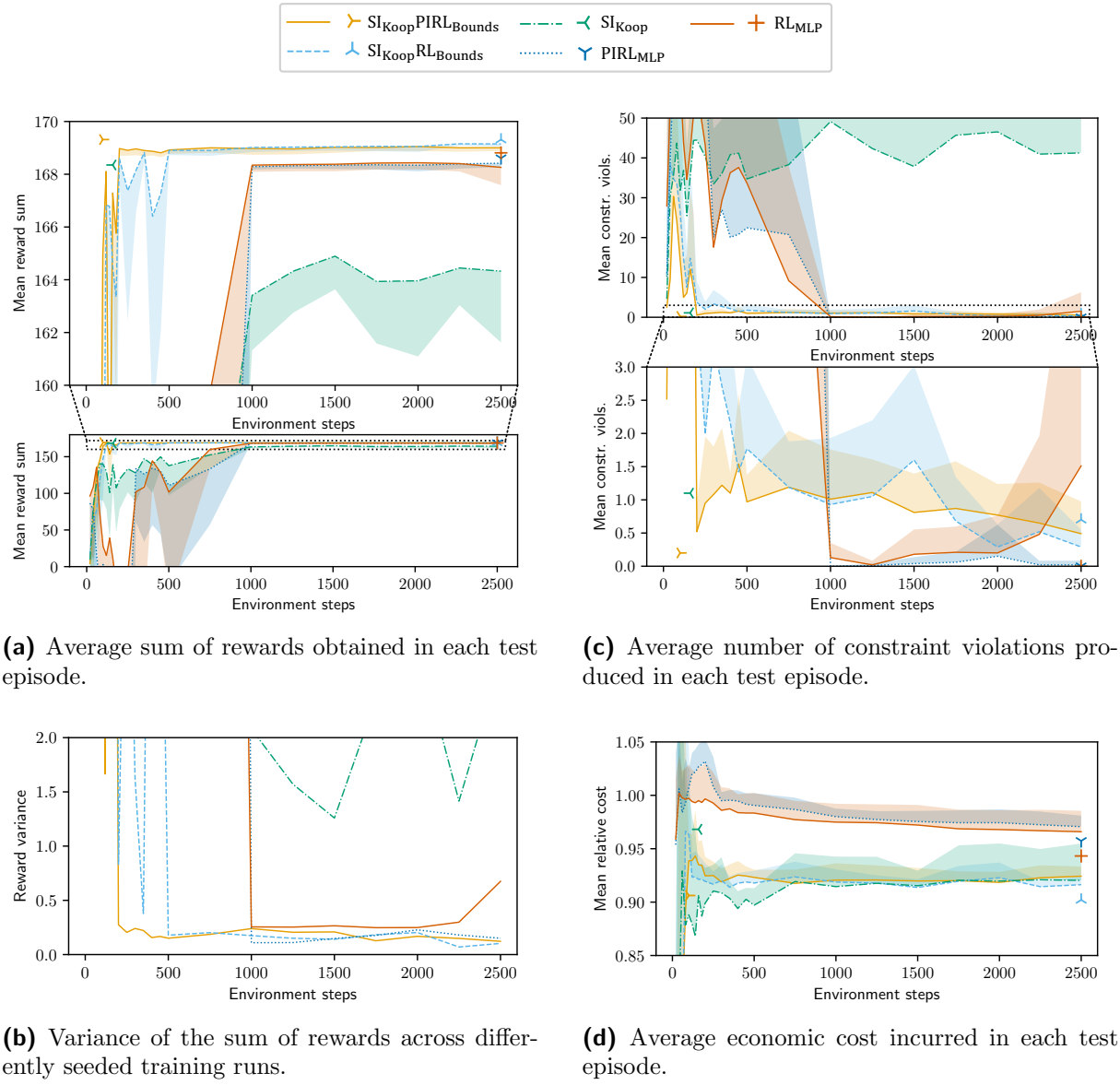
To evaluate the performance of our proposed method and to analyze how different components of the method contribute to the overall performance, we compare the performance of the following controller types. To avoid unnecessary repetition, we focus our description on the differences compared to the main 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 7.1.1 with implementation details explained in Section 7.2.2.1 - 7.2.2.4. 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 $||\boldsymbol{NN}_{i,\boldsymbol{\omega}_i}(\boldsymbol{x}_t, \boldsymbol{u}_t) - \boldsymbol{x}_{t+1}||$ for each $i \in \{1, 2, \ldots, 10\}$. The architecture of the models is identical to that of the PINN models (see Figure 7.4), 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 7.2.2.1 (Figure 7.3), 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.

7.2.3. 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 7.2.2. 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



(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.

Figure 7.5.: 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.

these tests can be viewed in Figure 7.5. Figure 7.6 shows some randomly chosen control trajectories that 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 Figure 7.6. 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. (7.5)), 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. Figure 7.5a shows that $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ (i) reach the highest reward values and that they do so with (ii) exceptional sample efficiency and (iii) low variance (see also Figure 7.5b) across different training runs. $\mathrm{PIRL_{MLP}}$ and $\mathrm{RL_{MLP}}$ 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 $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$. $\mathrm{SI_{Koop}}$ 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: $\mathrm{SI_{Koop}PIRL_{Bounds}}$ performs better than $\mathrm{SI_{Koop}RL_{Bounds}}$ between 200 and 500 environment steps. Thereafter, both variants perform comparably well. The two MLP controller variants, $\mathrm{PIRL_{MLP}}$ and $\mathrm{RL_{MLP}}$, 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 Figure 7.5c, we can see that initially (up to around 1000 steps) $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ are best at avoiding constraint violations. In particular, $\mathrm{SI_{Koop}PIRL_{Bounds}}$ 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 intially cause many constraint violations, however, after around 1000 environment steps they have learned to avoid constraint violations almost perfectly. Towards the end of the training, some of the $\mathrm{RL_{MLP}}$ controllers seem to lose this capability partially. $\mathrm{SI_{Koop}}$ struggles with constraint satisfaction across the full length of all training runs.

Figure 7.5d 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 become comparable to that of the Koopman eNMPC-based controllers if given a higher training budget.

The control trajectories in Figure 7.6 align with the findings derived thus far from Figure 7.5. All controllers show an intuitive inverse relationship between electricity prices and coolant flow rate F, although this relationship is noticeably weaker for $\mathrm{PIRL_{MLP}}$ and $\mathrm{RL_{MLP}}$. 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, $\mathrm{SI_{Koop}PIRL_{Bounds}}$ and $\mathrm{SI_{Koop}RL_{Bounds}}$ effectively utilize the full feasible range without violating bounds. In contrast, $\mathrm{SI_{Koop}}$ causes minor constraint violations. $\mathrm{PIRL_{MLP}}$ and $\mathrm{RL_{MLP}}$ avoid violations by maintaining c well

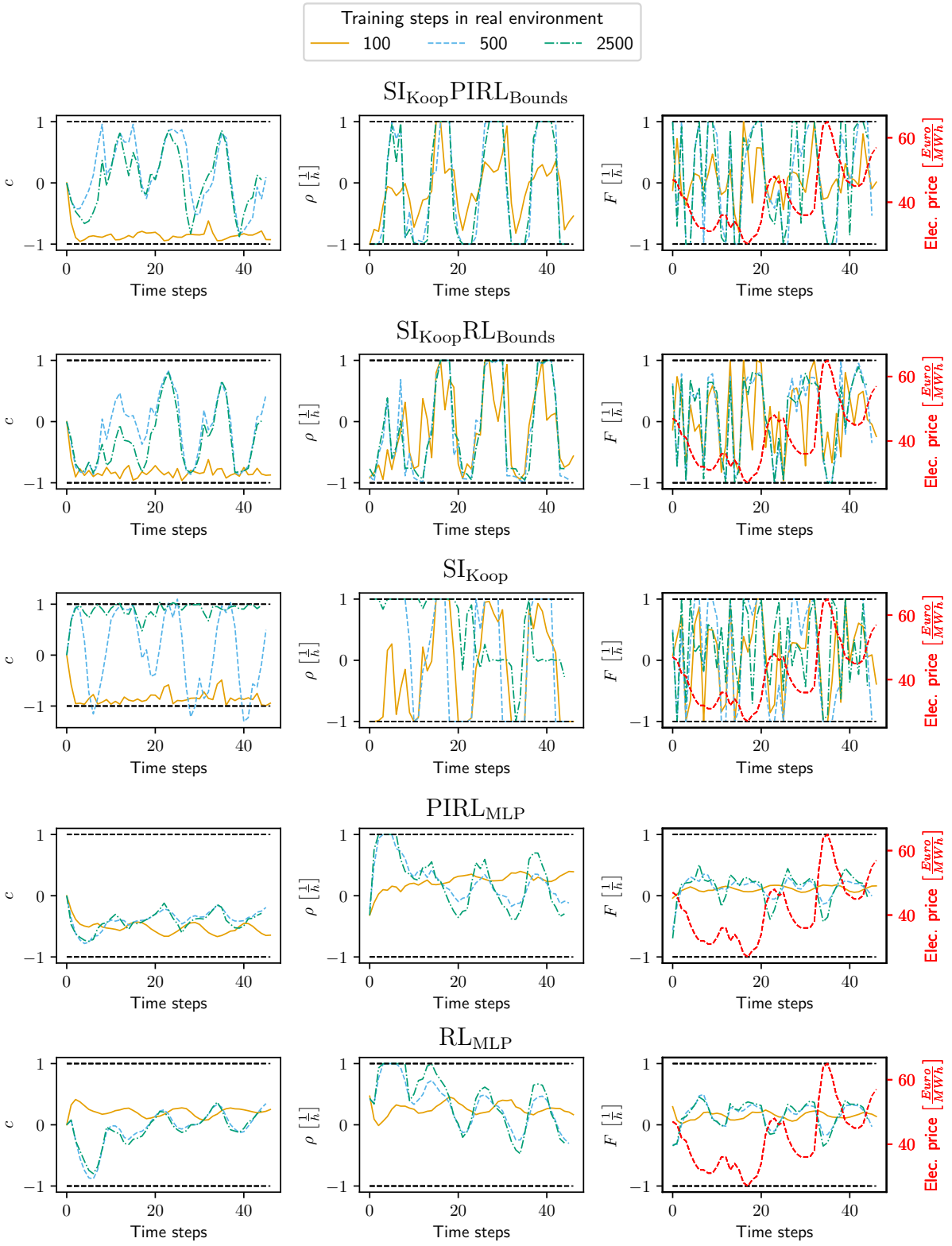


Figure 7.6.: Control trajectory comparison. The bounds of each variable (see Table 4.2) are used for scaling to the [-1, 1] range. We omit the temperature T since it never reaches its bounds in any of the test episodes.

within bounds but sacrifice process flexibility, limiting economic performance.

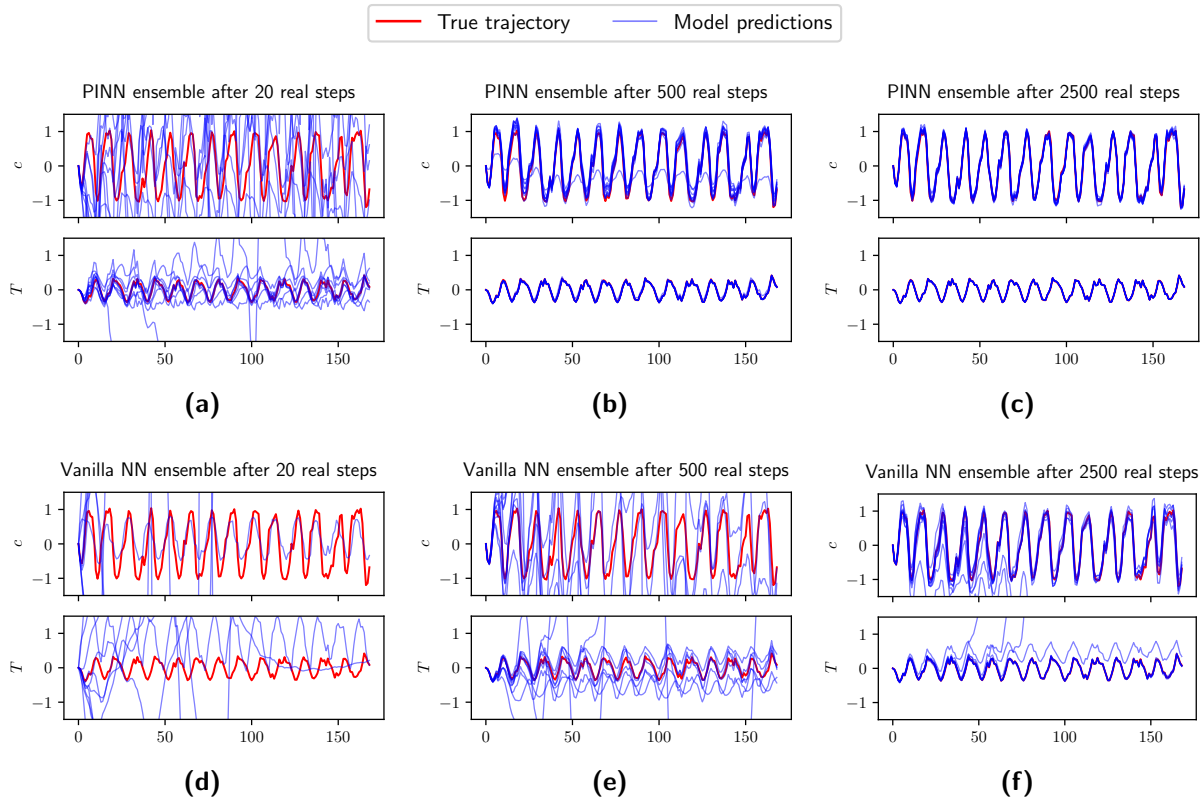


Figure 7.7.: Comparison of closed-loop prediction results of a PINN ensemble and a vanilla NN ensemble after different numbers of data sampling steps in the real environment (Figure 7.2a, first step). The bounds of each variable (see Table 4.2) are used for scaling to the [-1, 1] range. The red line 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.

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 ($\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$, PIRL_{MLP}) compared to their non-physics-informed counterparts ($\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$, RL_{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 $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ controller. Then, we randomly pick one of the $\text{SI}_{\text{Koop}}\text{PIRL}_{\text{Bounds}}$ and $\text{SI}_{\text{Koop}}\text{RL}_{\text{Bounds}}$ training runs. We test the ensembles that were produced by the chosen training runs after 20, 500, and 2500 environment steps. Figure 7.7 shows the results of these tests. It is evident from Figure 7.7a and 7.7d 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 (Figure 7.7b and 7.7e), 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 7.2.2.4), 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. Figure 7.5 and Figure 7.7).

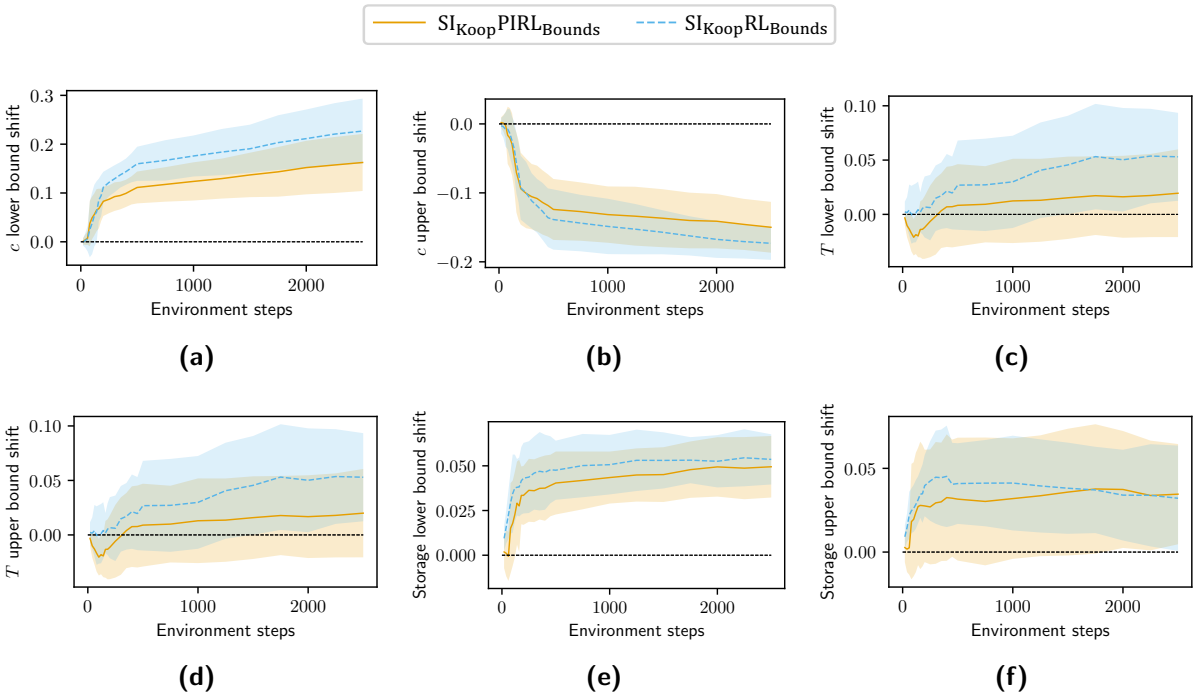


Figure 7.8.: Evolution of the parameters $\boldsymbol{\theta}_{\text{B}}$ during training. Lines depict the average of the respective parameter over 10 training runs; the shaded regions depict one standard deviation across the different training runs. The adaptation of the bounds is performed with respect to the scaled c and T variables (both variables scaled to the [-1,1] range using their bounds given in Table 4.2). We leave the product storage in its original [0,6] range.

As explained in Section 7.2.2, in training the SI$_{\text{Koop}}$PIRL$_{\text{Bounds}}$ and SI$_{\text{Koop}}$RL$_{\text{Bounds}}$ controllers, $\boldsymbol{\theta}_{\text{K}}$ are trained only via SI, whereas $\boldsymbol{\theta}_{\text{B}}$ are trained only via PPO using imagined policy rollouts using the learned model ensemble. This means that besides their good performance (see Figure 7.5), the SI$_{\text{Koop}}$PIRL$_{\text{Bounds}}$ and SI$_{\text{Koop}}$RL$_{\text{Bounds}}$ controllers have another valuable property: the (very few) parameters $\boldsymbol{\theta}_{\text{B}}$ that are used to optimize the controller for task-optimal control performance are intuitively interpretable. Each parameter in $\boldsymbol{\theta}_{\text{B}}$ modifies one of the bounds in the OCPs of the eNMPC (see Eq. (7.3)), i.e., the lower and upper bounds of c, T, and the storage level. Figure 7.8 shows the evolution of $\boldsymbol{\theta}_{\text{B}}$ during the SI$_{\text{Koop}}$PIRL$_{\text{Bounds}}$ and SI$_{\text{Koop}}$RL$_{\text{Bounds}}$ training runs. For both variants and across all trained eNMPCs, the bounds of c are tightened (see Figure 7.8a and 7.8b), thus decreasing the likelihood of constraint violations. Figure 7.8c and 7.8d, 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 directions of the adaptations do not align across all training instances. Since in our controller tests, the bounds of c were violated frequently (see Figure 7.6), whereas no violations of the T bounds were observed, it makes intuitive sense that optimizing $\boldsymbol{\theta}_{\mathrm{B}}$ leads to a tightening of the c bounds and a less decisive adaptation of the T bounds. Figure 7.8f 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 Figure 7.8e). This is not surprising since, like the bounds of T, the upper storage bound is not violated by any of the controllers during training.

7.3. 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. (2024)) 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 Figure 7.2a): 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. (2024)) 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 7.1.1) 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 Figure 7.2a), 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 Figure 7.2a). 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, 2022b)): 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.

8. Conclusion

Model-based control strategies, such as MPC or variants thereof, rely on solving dynamic optimization problems with embedded dynamic models in real time. Due to the complexities of process systems, mechanistic dynamic models (i) are not always available, or (ii) may not be usable as part of real-time capable optimizations. In such cases, models can be learned from data generated by the real process or using a mechanistic model that cannot be used for control directly.

This thesis develops methods for end-to-end learning of data-driven dynamic models for optimal performance as part of predictive control policies. While previous contributions to the literature had also addressed end-to-end learning of dynamic models for specific control applications, open questions remained: Some works (e.g., Iwata and Kawahara (2022); Yin et al. (2022)) are limited to setpoint tracking problems that can be addressed using model-based policies that can be represented by a closed-form solution, e.g., LQR policies. Other works (e.g., Amos et al. (2018); Chen et al. (2019)) cannot handle constraints on the system states. Finally, some prior contributions (e.g., Gros and Zanon (2019); Brandner et al. (2023)) circumvent learning purely data-driven dynamic models with many parameters by optimizing highly structured models with few parameters, which requires a priori system knowledge.

In the following, this chapter summarizes this thesis and its contributions in Section 8.1 and presents ideas on possible future work in Section 8.2.

8.1. Summary

This thesis aims to develop and apply RL methods to train Koopman models for optimal performance in nonlinear, constrained control tasks. Chapter 1 introduces current needs of the chemical industry as a motivation and establishes foundational concepts linking (e)NMPC, RL, and Koopman theory. Chapter 2 provides the necessary preliminaries, including Markov decision processes, MPC formulations, RL algorithms, and Koopman theory for control. Chapter 3 reviews related literature on online controller tuning, integration of MPC and RL, and applications of Koopman (e)NMPC. Chapters 4 through 7 present the core methodological contributions of this thesis:

In Chapter 4, we present a method for end-to-end RL of Koopman surrogate models for optimal performance as part of (e)NMPC. Using an automatically differentiable solver for convex optimization problems (Agrawal et al. (2019a)), we construct (e)NMPC policies whose control outputs are differentiable with respect to the parameters of the Koopman model. Thus, we can use state-of-the-art actor-critic policy gradient algorithms (e.g., Schulman et al. (2017)) to optimize the parameters of the Koopman model for maximum performance in a specific control task. We apply our method to two applications derived from an established nonlinear continuous stirred-tank reactor model (Flores-Tlacuahuac

and Grossmann (2006)). The controller performance is compared to that of (e)NMPCs utilizing models trained using system identification, and model-free neural network controllers trained using RL. We demonstrate that end-to-end trained models outperform those trained using system identification in (e)NMPC, and that, in contrast to neural network controllers, (e)NMPC controllers can react to changes in the control setting without requiring retraining. Our approach is applicable to a wide range of control problems that can be addressed with (e)NMPC policies featuring arbitrary convex objective functions and convex constraints. By using Koopman models, we obtain policies that can perform excellently in environments with nonlinear dynamics while remaining real-time capable due to the convexity of the OCPs in the eNMPC.

In Chapter 5, we introduce a more challenging demand response case study built on a large-scale air separation unit (ASU) model, and we validate our base method (Chapter 4) on it. Similarly to our approach in Chapter 4, we compare the performance of the resulting policies to that of a Koopman eNMPC obtained through system identification and find that our method produces policies that avoid constraint violations while exhibiting similar economic performance. By demonstrating the scalability of our method to large problems, we show that it could be applicable to complex real-world control problems where mechanistic predictive control policies are not real-time capable and system identification-based data-driven controllers produce unsatisfactory performance.

Chapter 6 further develops our method from Chapter 4: Standard RL algorithms are designed to enable learning from real-world, physical systems and thus from pure input-output data. However, when learning data-driven *surrogate* models for mechanistic models using RL, the dynamics of the environment may be differentiable, which can be exploited for policy optimization. Therefore, we combine our previously developed method for creating automatically differentiable Koopman (e)NMPC policies with the Short-Horizon Actor-Critic (SHAC) (Xu et al. (2022)) algorithm, which leverages the differentiability of the learning environment for policy optimization. We evaluate the performance of our method by comparing it to that of other training algorithms on the small-scale eNMPC case study that we also used in Chapter 4. Compared to the benchmark methods, which include a Koopman eNMPC learned end-to-end using a state-of-the-art RL algorithm that views the learning environment as a non-differentiable black-box (Schulman et al. (2017)), our method produces similar economic performance while eliminating constraint violations. Our analysis indicates that this superior performance could be due to a substantially lower variance in the direction of the policy gradient estimates, which results from the exploitation of analytic derivative information from an automatically differentiable environment. Chapter 6 thus offers a promising path toward more performant controllers that employ dynamic surrogate models.

In Chapter 7, we continue the development of our method for learning task-optimal Koopman (e)NMPCs by addressing the issue of sample efficiency. Standard RL algorithms are notoriously sample inefficient. When training in inexpensive in-silico environments, this may not be a problem; however, when simulating the environment becomes expensive or when learning from interactions with physical systems, sample efficiency becomes critical. To improve it, we combine the model-based RL algorithm Model-Based Policy Optimization (MBPO) (Janner et al. (2019)) with our previously developed method. Like other Dyna-style model-based RL algorithms, MBPO uses the experience acquired from interactions with the environment to learn a model of the environment, which is then used as

a surrogate learning environment for policy optimization. MBPO utilizes a learned model ensemble and truncated simulated policy rollouts to mitigate the impact of overexploiting model errors in policy optimization. We validate our approach using the established CSTR eNMPC case study. 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, we show that utilizing partial prior knowledge about the system dynamics via physics-informed learning of the model ensemble further increases sample efficiency. Due to the combination of exceptional sample efficiency and high resulting control performance, our approach could offer concrete benefits for the control of various real-world systems where mechanistic models cannot be used for predictive control or may not even be available.

8.2. Outlook

This section presents three directions of possible future work.

Dealing with model plant mismatch

In Chapters 4 and 6, we developed methods for learning task-optimal Koopman surrogate models for control applications. These methods rely on constructing an RL environment based on a mechanistic process model and training an eNMPC policy, parameterized by the Koopman surrogate model, within this environment. The performance of the resulting eNMPC policies is evaluated exclusively within this mechanistic model-based environment, which implicitly assumes that the mechanistic model accurately captures the true plant dynamics.

However, in practical applications, what ultimately matters is the control performance achieved when deploying the policy on the physical plant. A mismatch between the mechanistic model and the physical plant may cause a policy optimized for the mechanistic model to perform poorly in reality. Thus, future research should analyze the sensitivity of the developed methods with respect to model-plant mismatch and investigate how to facilitate good policy performance in the physical plant.

One obvious way to sidestep this issue is to learn directly from interactions with the real plant instead of training the policy in a simulated environment based on an inaccurate mechanistic model. However, in practice this may not be a good option: Even highly sample-efficient RL algorithms need to explore different behaviors to improve a policy, which will always be costly given a real-world plant and could even be dangerous depending on the process. Moreover, training within a simulated environment offers the key advantage that simulations can be run at much faster than real-time speeds, thus accelerating the learning process by enabling the RL algorithm to gather vast amounts of data in a short period. As a result, while training directly on the plant may yield policies that are more accurate in terms of real-world performance, the use of simulation for initial training still offers practical benefits in terms of efficiency, safety, and cost-effectiveness.

One strategy to address model-plant mismatch is to explicitly incorporate uncertainty into the RL training environment. This can be achieved by randomly sampling uncertain

model parameters from a distribution of plausible values, or by introducing disturbances that affect the system in a known way, such as random noise on the model outputs. Such approaches could allow the policy to be trained in a robust fashion, effectively learning to perform well under a range of possible dynamics variations and thereby reducing sensitivity to model inaccuracies.

Another possible approach is meta-learning, where the RL agent is trained across multiple variants of the mechanistic model with different parameterizations or structural perturbations. This approach aims to produce a policy that can be fine-tuned to the true plant dynamics in a few-shot manner.

Advancing research along these directions will enhance the resilience of learned Koopman eNMPC controllers against model errors. Successfully addressing model-plant mismatch will be crucial for translating the promising control performance demonstrated in simulated environments to complex industrial chemical processes in real-world settings.

Safety

Safety is a fundamental challenge in potential applications of RL to industrial chemical processes. In applications where training is done in silico, safe behavior is desired when training has finished and the policy starts interacting with the real process. In our work, we addressed this by penalizing constraint violations using the reward function of the learning environment. While such an approach often produces empirically good results, it does not provide any safety guarantees. In applications where no model of the system is available and the real process is used as a learning environment, safety-critical constraints need to be met even during learning.

A promising idea for enabling safety-aware RL is model predictive safety certification (Wabersich and Zeilinger (2018)). Gronauer et al. (2024) propose Ensemble Model Predictive Safety Certification (X-MPSC), which combines an ensemble of probabilistic neural network models with tube-based MPC to certify and correct potentially unsafe actions suggested by an RL agent. Given sufficient offline data from a safe controller for initialization, X-MPSC can reduce constraint violations during training and inference to near zero. The differentiable Koopman-eNMPC policies developed in this thesis can be naturally integrated into X-MPSC. Another interesting possibility is to employ Koopman models in the model ensemble used to certify actions proposed by a neural network policy. Such an approach could improve the real-time feasibility of X-MPSC since safety certification requires solving OPCs in real-time, which is easier when using Koopman models than when using neural network models.

Future research regarding safety during both training and inference could enable the use of RL-based policies directly on physical plants without compromising safety. Furthermore, such research could open the door to continuous online policy adaptation in changing environments, e.g., due to the physical degradation of production plants.

Partially observable systems

The experiments conducted in this thesis assume full observability of the control systems. However, many practical systems operate under partial observability, where not all state

variables are directly measurable. Partial observability poses challenges for reinforcement learning and model predictive control, as the lack of complete state information can lead to suboptimal control actions. In some cases, a naive approach that ignores partial observability and provides only the measurable states as inputs to the policy may suffice. However, in many applications, more sophisticated ways of dealing with partial observability are necessary.

State estimation techniques such as Moving Horizon Estimation (MHE) or Kalman filtering are commonly employed in control to reconstruct unmeasured states from noisy observations, enabling controllers to operate effectively despite incomplete data. Such approaches, which enable the controller to infer hidden states and make informed decisions despite incomplete information, could be integrated with the RL-based Koopman eNMPC framework developed in this thesis.

An alternative strategy to dealing with partial observability is proposed by Weigand et al. (2021), who developed Partially Observable Guided Reinforcement Learning (PO-GRL). This method leverages full-state information during the early stages of training and gradually transitions to using only partial observations, thus resulting in a policy that does not require full-state information or explicit state estimation during inference. Such an approach is viable in applications where training can be performed in silico using a mechanistic model, e.g., when learning data-driven surrogate models. PO-GRL has been demonstrated successfully in challenging partially observable tasks, including robotics applications. Applying PO-GRL concepts within our Koopman eNMPC framework could guide the training of Koopman models and eNMPC policies to be robust under partial observability.

Exploring these directions offers the potential to extend the practical applicability of RL-trained Koopman eNMPC controllers substantially. By combining observer-based estimation or guided observability training, future research could enable robust, high-performance control using end-to-end learned Koopman eNMPC policies in scenarios with limited or noisy sensor data. This could enable the deployment of such policies in complex real-world systems, such as chemical plants, where full observability is usually not given.

Appendix A.

Hyperparameters in Chapter 4

Table A.1.: Hyperparameters in NMPC. Where possible, the notation is consistent with the PPO paper (Schulman et al. (2017)).

Hyperparameter	Value	Description
General		
$\boldsymbol{\sigma}$	$(0.05, 0.05)^\intercal$	standard deviation for action selection
γ	0.95	reward discount factor
λ	0.95	generalized advantage estimation hyperparameter
ϵ	0.2	clipping hyperparameter
N_{PPO}	1	number of parallel actors
T_{PPO}	2,048	control steps between updates to actor and critic
M_{PPO}	64	minibatch size
optimizer	Adam	optimizer used for updates to actor and critic
α_{actor}	10^{-4}	learning rate of actor
α_{critic}	$3 \cdot 10^{-4}$	learning rate of critic
episodes	5,000	number of episodes per training run
Koopman MPC policies		
K_{PPO}	5	number of epochs per update
max. gradient norm	100.0	gradient clipping value for actor update
solver	SCS	solver for Koopman OCPs (O'Donoghue et al. (2016))
max. iters.	500	maximum number of iterations in Koopman OCP solver
MLP policies		
K_{PPO}	10	number of epochs per update
max. gradient norm	40.0	gradient clipping value for actor update

Table A.2.: Hyperparameters in eNMPC. Where possible, the notation is consistent with the PPO paper (Schulman et al. (2017)).

Hyperparameter	Value	Description
General		
β	$5 \cdot 10^{-5}$	reward calculation hyperparameter
$\boldsymbol{\sigma}$	$(0.05, 0.05)^\intercal$	standard deviation for action selection
γ	0.95	reward discount factor
λ	0.95	generalized advantage estimation hyperparameter
ϵ	0.2	clipping hyperparameter
N_{PPO}	1	number of parallel actors
T_{PPO}	2,048	control steps between updates to actor and critic
M_{PPO}	64	minibatch size
optimizer	Adam	optimizer used for updates to actor and critic
Koopman MPC policies		
episodes	5,000	number of episodes per training run
K_{PPO}	5	number of epochs per update
α_{actor}	10^{-5}	learning rate of actor
α_{critic}	$2 \cdot 10^{-5}$	learning rate of critic
max. gradient norm	100.0	gradient clipping value for actor update
solver	ECOS	solver for Koopman OCPs (Domahidi et al. (2013))
max. iters.	500	maximum number of iterations in Koopman OCP solver
M	10,000	penalty factor for slack variable usage
MLP policies		
episodes	20,000	number of episodes per training run
K_{PPO}	10	number of epochs per update
max. gradient norm	40.0	gradient clipping value for actor update
α_{actor}	10^{-4}	learning rate of actor
α_{critic}	$3 \cdot 10^{-4}$	learning rate of critic

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). TensorFlow: a system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. (2019a). Differentiable convex optimization layers. *Advances in Neural Information Processing Systems*, 32:9558–9570.

Agrawal, A., Barratt, S., Boyd, S., Busseti, E., and Moursi, W. M. (2019b). Differentiating through a cone program. *arXiv preprint arXiv:1904.09043*.

Åkesson, B. M. and Toivonen, H. T. (2006). A neural network model predictive controller. *Journal of Process Control*, 16(9):937–946.

Albalawi, F. and Hameed, S. W. (2023). Koopman-based economic model predictive control for nonlinear systems. In *2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, pages 822–828.

Allgöwer, F., Badgwell, T. A., Qin, J. S., Rawlings, J. B., and Wright, S. J. (1999). Nonlinear predictive control and moving horizon estimation—an introductory overview. *Advances in control: Highlights of ECC'99*, pages 391–449.

Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. (2018). Differentiable MPC for end-to-end planning and control. *Advances in Neural Information Processing Systems*, 31:8299–8310.

Amos, B. and Kolter, J. Z. (2017). Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145.

Angeli, D., Amrit, R., and Rawlings, J. B. (2011). On average performance and stability of economic model predictive control. *IEEE Transactions on Automatic Control*, 57(7):1615–1626.

Antonelo, E. A., Camponogara, E., Seman, L. O., Jordanou, J. P., de Souza, E. R., and Hübner, J. F. (2024). Physics-informed neural nets for control of dynamical systems. *Neurocomputing*, 579:127419.

Arbabi, H., Korda, M., and Mezic, I. (2018). A data-driven Koopman model predictive control framework for nonlinear flows. *arXiv preprint arXiv:1804.05291*.

Bassett, M. H., Pekny, J. F., and Reklaitis, G. V. (1996). Decomposition techniques for the solution of large-scale scheduling problems. *AIChE Journal*, 42(12):3373–3387.

Beal, L., Hill, D., Martin, R., and Hedengren, J. (2018). Gekko optimization suite. *Processes*, 6(8):106.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *Advances in Neural Information Processing Systems*, 28:1171–1179.

Bhardwaj, M., Choudhury, S., and Boots, B. (2020). Blending MPC & value function approximation for efficient reinforcement learning. *arXiv preprint arXiv:2012.05909*.

Boyd, S., Boyd, S. P., and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Brandner, D. and 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., and Lucia, S. (2023). Reinforcement learning combined with model predictive control to optimally operate a flash separation unit. In *Computer Aided Chemical Engineering*, volume 52, pages 595–600. Elsevier.

Bruder, D., Gillespie, B., Remy, C. D., and Vasudevan, R. (2019). Modeling and control of soft robots using the Koopman operator and model predictive control. *arXiv preprint arXiv:1902.02827*.

Brunton, S. L., Budišić, M., Kaiser, E., and Kutz, J. N. (2021). Modern Koopman theory for dynamical systems. *arXiv preprint arXiv:2102.12086*.

Brunton, S. L. and Kutz, J. N. (2022). *Data-driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press.

Caspari, A., Tsay, C., Mhamdi, A., Baldea, M., and Mitsos, A. (2020). The integration of scheduling and control: Top-down vs. bottom-up. *Journal of Process Control*, 91:50–62.

Chen, B., Cai, Z., and 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*, pages 316–325.

Chen, C. Y. and Joseph, B. (1987). On-line optimization using a two-phase approach: An application study. *Industrial & Engineering Chemistry Research*, 26(9):1924–1930.

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. (2018). Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 31:6572–6583.

Cibulka, V., Haniš, T., Korda, M., and Hromčík, M. (2020). Model predictive control of a vehicle using Koopman operator. *IFAC-PapersOnLine*, 53(2):4228–4233. 21st IFAC World Congress.

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

Dalal, G., Dvijotham, K., Vecerik, M., Hester, T., Paduraru, C., and Tassa, Y. (2018). Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*.

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

Domahidi, A., Chu, E., and Boyd, S. (2013). ECOS: An SOCP solver for embedded systems. In *2013 European control conference (ECC)*, pages 3071–3076. IEEE.

Domke, J. (2012). Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pages 318–326.

Dormand, J. R. and Prince, P. J. (1980). A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26.

Du, J., Park, J., Harjunkoski, I., and Baldea, M. (2015). A time scale-bridging approach for integrating production scheduling and process control. *Computers & Chemical Engineering*, 79:59–69.

Ellis, M., Liu, J., and Christofides, P. D. (2018). *Economic Model Predictive Control: Theory, Formulations and Chemical Process Applications*. Springer Publishing Company, Incorporated, 1st edition.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep policy gradients: A case study on PPO and TRPO. *arXiv preprint arXiv:2005.12729*.

EPEX (2023). Epex spot market data. `https://www.epexspot.com`. Accessed: 2025-01-02.

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

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

Fiacco, A. V. and Ishizuka, Y. (1990). Sensitivity and stability analysis for nonlinear programming. *Annals of Operations Research*, 27(1):215–235.

Fiacco, A. V. and Kyparisis, J. (1985). Sensitivity analysis in nonlinear programming under second order assumptions. In *Systems and Optimization*, pages 74–97. Springer.

Fiedler, F., Cominola, A., and Lucia, S. (2020). Economic nonlinear predictive control of water distribution networks based on surrogate modeling and automatic clustering. *IFAC-PapersOnLine*, 53(2):16636–16643.

Flores-Tlacuahuac, A. and Grossmann, I. E. (2006). Simultaneous cyclic scheduling and control of a multiproduct CSTR. *Industrial & Engineering Chemistry Research*, 45(20):6698–6712.

Folkestad, C. and Burdick, J. W. (2021). Koopman NMPC: Koopman-based learning and nonlinear model predictive control of control-affine systems. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7350–7356.

Forbes, J., Marlin, T., and MacGregor, J. (1994). Model adequacy requirements for optimizing plant operations. *Computers & chemical engineering*, 18(6):497–510.

Forrester, A. I. and Keane, A. J. (2009). Recent advances in surrogate-based optimization. *Progress in aerospace sciences*, 45(1-3):50–79.

Frauenknecht, B., Eisele, A., Subhasish, D., Solowjow, F., and 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., and Trimpe, S. (2025). On rollouts in model-based reinforcement learning. *arXiv preprint arXiv:2501.16918*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596.

Gao, C. and Wang, D. (2023). Comparative study of model-based and model-free reinforcement learning control performance in HVAC systems. *Journal of Building Engineering*, 74:106852.

Ghezzi, A., Hoffman, J., Frey, J., Boedecker, J., and Diehl, M. (2023). Imitation learning from nonlinear mpc via the exact q-loss and its gauss-newton approximation. In *2023 62nd IEEE Conference on Decision and Control (CDC)*, pages 4766–4771. IEEE.

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

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

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

Görges, D. (2017). Relations between model predictive control and reinforcement learning. *IFAC-PapersOnLine*, 50(1):4920–4928.

Gould, L., Evans, L., and Kurihara, H. (1970). Optimal control of fluid catalytic cracking processes. *Automatica*, 6(5):695–703.

Gronauer, S., Haider, T., da Roza, F. S., and Diepold, K. (2024). Reinforcement learning with ensemble model predictive safety certification. *arXiv preprint arXiv:2402.04182*.

Gros, S. and Zanon, M. (2019). Data-driven economic NMPC using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648.

Haarnoja, T., Zhou, A., Abbeel, P., and 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 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR.

Hazan, E., Kakade, S., Singh, K., and Van Soest, A. (2019). Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pages 2681–2691. PMLR.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018a). Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, pages 3207–3214.

Henderson, P., Romoff, J., and Pineau, J. (2018b). Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods. *arXiv preprint arXiv:1810.02525*.

Hertneck, M., Köhler, J., Trimpe, S., and Allgöwer, F. (2018). Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3):543–548.

Hu, Y., Liu, J., Spielberg, A., Tenenbaum, J. B., Freeman, W. T., Wu, J., Rus, D., and Matusik, W. (2019). Chainqueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE.

Huang, Z., Hu, Y., Du, T., Zhou, S., Su, H., Tenenbaum, J. B., and Gan, C. (2021). Plasticinelab: A soft-body manipulation benchmark with differentiable physics. *arXiv preprint arXiv:2104.03311*.

Hussein, A., Gaber, M. M., Elyan, E., and Jayne, C. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35.

Hwangbo, J., Lee, J., Dosovitskiy, A., Bellicoso, D., Tsounis, V., Koltun, V., and Hutter, M. (2019). Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26):eaau5872.

Ilyas, A., Engstrom, L., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2018). A closer look at deep policy gradients. *arXiv preprint arXiv:1811.02553*.

Iman, R. L., Helton, J. C., and Campbell, J. E. (1981). An Approach to Sensitivity Analysis of Computer Models: Part I—Introduction, Input Variable Selection and Preliminary Variable Assessment. *Journal of Quality Technology*, 13(3):174–183.

Islam, R., Henderson, P., Gomrokchi, M., and Precup, D. (2017). Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*.

Iwata, T. and Kawahara, Y. (2022). Data-driven end-to-end learning of pole placement control for nonlinear dynamics via Koopman invariant subspaces. *arXiv preprint arXiv:2208.08883*.

Jang, S.-S., Joseph, B., and Mukai, H. (1987). On-line optimization of constrained multivariable chemical processes. *AIChE Journal*, 33(1):26–35.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32:12498–12509.

Johnson, C. R., Wohlgemuth, K., and Lucia, S. (2025). A tutorial overview of model predictive control for continuous crystallization: current possibilities and future perspectives. *arXiv preprint arXiv:2506.17146*.

Kalman, R. E. et al. (1960). Contributions to the theory of optimal control. *Boletín de la Sociedad Matemática Mexicana*, 5(2):102–119.

Karg, B. and Lucia, S. (2021). Reinforced approximate robust nonlinear model predictive control. In *2021 23rd International Conference on Process Control (PC)*, pages 149–156. IEEE.

Karush, W. (1939). Minima of functions of several variables with inequalities as side constraints. *M. Sc. Dissertation. Department of Mathematics, University of Chicago*.

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

Klaucke, F., Karsten, T., Holtrup, F., Esche, E., Morosuk, T., Tsatsaronis, G., and Repke, J.-U. (2017). Demand response potentials for the chemical industry. *Chemie Ingenieur Technik*, 89(9):1133–1141.

Kondili, E., Pantelides, C. C., and Sargent, R. W. (1993). A general algorithm for short-term scheduling of batch operations—i. milp formulation. *Computers & Chemical Engineering*, 17(2):211–227.

Koopman, B. O. (1931). Hamiltonian systems and transformation in Hilbert space. *Proceedings of the National Academy of Sciences*, 17(5):315–318.

Korda, M. and 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., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Levine, S., Kumar, A., Tucker, G., and Fu, J. (2020). Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*.

Lewis, F. L. and Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. *IEEE Circuits and Systems Magazine*, 9(3):32–50.

Lewis, F. L., Vrabie, D., and Vamvoudakis, K. G. (2012). Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. *IEEE Control Systems Magazine*, 32(6):76–105.

Lienen, M. and Günnemann, S. (2022). torchode: A parallel ode solver for pytorch. *arXiv preprint arXiv:2210.12375*.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528.

Liu, X.-Y. and Wang, J.-X. (2021). Physics-informed dyna-style model-based deep reinforcement learning for dynamic control. *Proceedings of the Royal Society A*, 477(2255):20210618.

Ljung, L. (1998). System identification. In *Signal analysis and prediction*, pages 163–173. Springer.

Lüken, L., Brandner, D., and Lucia, S. (2023). Sobolev training for data-efficient approximate nonlinear MPC. *IFAC-PapersOnLine*, 56(2):5765–5772.

Lusch, B., Kutz, J. N., and Brunton, S. L. (2018). Deep learning for universal linear embeddings of nonlinear dynamics. *Nature Communications*, 9(1):1–10.

Maddu, S., Sturm, D., Müller, C. L., and Sbalzarini, I. F. (2022). Inverse Dirichlet weighting enables reliable training of physics informed neural networks. *Machine Learning: Science and Technology*, 3(1):015026.

Mahmood, A. R., Korenkevych, D., Vasan, G., Ma, W., and Bergstra, J. (2018). Benchmarking reinforcement learning algorithms on real-world robots. In *Conference on Robot Learning*, pages 561–591. PMLR.

Mania, H., Guy, A., and Recht, B. (2018). Simple random search of static linear policies is competitive for reinforcement learning. *Advances in Neural Information Processing Systems*, 31:1805–1814.

Marchetti, A., Chachuat, B., and Bonvin, D. (2009). Modifier-adaptation methodology for real-time optimization. *Industrial & Engineering Chemistry Research*, 48(13):6022–6033.

Marchetti, A., Chachuat, B., and Bonvin, D. (2010). A dual modifier-adaptation approach for real-time optimization. *Journal of Process Control*, 20(9):1027–1037.

Marchetti, A. G., François, G., Faulwasser, T., and Bonvin, D. (2016). Modifier adaptation for real-time optimization—methods and applications. *Processes*, 4(4):55.

Marlin, T. E., Hrymak, A. N., et al. (1997). Real-time operations optimization of continuous processes. In *AIChE Symposium Series*, volume 93, pages 156–164. New York, NY: American Institute of Chemical Engineers, 1971-c2002.

Mayfrank, D., Ahn, N. Y., Mitsos, A., and Dahmen, M. (2025a). Task-optimal data-driven surrogate models for eNMPC via differentiable simulation and optimization. In *14th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems (DYCOPS) 2025 (in press)*.

Mayfrank, D., Dernek, K., Lang, L., Mitsos, A., and Dahmen, M. (2025b). End-to-end reinforcement learning of Koopman models for eNMPC of an air separation unit. *arXiv preprint arXiv:2511.04522*.

Mayfrank, D., Mitsos, A., and Dahmen, M. (2023). End-to-end reinforcement learning of Koopman models for economic model predictive control. In *AIChE Annual Meeting*. AIChE.

Mayfrank, D., Mitsos, A., and Dahmen, M. (2024). End-to-end reinforcement learning of Koopman models for economic nonlinear model predictive control. *Computers & Chemical Engineering*, 190:108824.

Mayfrank, D., Velioglu, M., Mitsos, A., and Dahmen, M. (2025c). Physics-informed model-based policy optimization of Koopman economic NMPC policies. In *AIChE Annual Meeting*. AIChE.

Mayfrank, D., Velioglu, M., Mitsos, A., and Dahmen, M. (2025d). Sample-efficient reinforcement learning of Koopman eNMPC. *Computers & Chemical Engineering (in press)*, page 109240.

McBride, K. and Sundmacher, K. (2019). Overview of surrogate modeling in chemical process engineering. *Chemie Ingenieur Technik*, 91(3):228–239.

Mesbah, A., Wabersich, K. P., Schoellig, A. P., Zeilinger, M. N., Lucia, S., Badgwell, T. A., and Paulson, J. A. (2022). Fusion of machine learning and MPC under uncertainty: What advances are on the horizon? In *2022 American Control Conference (ACC)*, pages 342–357. IEEE.

Mitsos, A., Asprion, N., Floudas, C. A., Bortz, M., Baldea, M., Bonvin, D., Caspari, A., and Schäfer, P. (2018). Challenges in process optimization for new feedstocks and energy sources. *Computers & Chemical Engineering*, 113:209–221.

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

Moerland, T. M., Broekens, J., Plaat, A., Jonker, C. M., et al. (2023). Model-based reinforcement learning: A survey. *Foundations and Trends in Machine Learning*, 16(1):1–118.

Mora, M. A. Z., Peychev, M., Ha, S., Vechev, M., and Coros, S. (2021). Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR.

Morales, M. (2020). *Grokking deep reinforcement learning*. Manning.

Mordatch, I. and Todorov, E. (2014). Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, volume 4, page 23.

Narasingam, A. and Kwon, J. S.-I. (2019). Koopman Lyapunov-based model predictive control of nonlinear chemical process systems. *AIChE Journal*, 65(11):e16743.

Narasingam, A. and Kwon, J. S.-I. (2020). Application of Koopman operator for model-based control of fracture propagation and proppant transport in hydraulic fracturing operation. *Journal of Process Control*, 91:25–36.

Naumann, U. (2011). *The Art of Differentiating Computer Programs*. Society for Industrial and Applied Mathematics.

Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer.

O'Donoghue, B., Chu, E., Parikh, N., and Boyd, S. (2016). Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068.

Open Power System Data (2020). Open power system data. `https://data.open-power-system-data.org/time_series/` (accessed on 2022-08-29).

Papadimitriou, C., Schulze, J. C., and Mitsos, A. (2024). Representative electricity price profiles for european day-ahead and intraday spot markets. *arXiv preprint arXiv:2405.14403*.

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. *Advances in Neural Information Processing Systems*, 32:8024–8035.

Patel, R., Bhartiya, S., and Gudi, R. (2024). Model predictive control using physics informed neural networks for process systems. *IFAC-PapersOnLine*, 58(14):775–780.

Pistikopoulos, E. N., Diangelakis, N. A., and Oberdieck, R. (2020). *Multi-Parametric Optimization and Control*. John Wiley & Sons.

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

Prett, D. M. and Morari, M. (2013). *The shell process control workshop*. Elsevier.

Proctor, J. L., Brunton, S. L., and Kutz, J. N. (2018). Generalizing Koopman theory to allow for inputs and control. *SIAM Journal on Applied Dynamical Systems*, 17(1):909–930.

Qin, S. J. and Badgwell, T. A. (2003). A survey of industrial model predictive control technology. *Control engineering practice*, 11(7):733–764.

Quinonero-Candela, J., Sugiyama, M., Schwaighofer, A., and Lawrence, N. D. (2008). *Dataset Shift in Machine Learning*. MIT Press.

Rackauckas, C., Innes, M., Ma, Y., Bettencourt, J., White, L., and Dixit, V. (2019). Diffeqflux. jl-a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*.

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

Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707.

Ralph, D. and Dempe, S. (1995). Directional derivatives of the solution of a parametric nonlinear program. *Mathematical Programming*, 70(1):159–172.

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

Rawlings, J. B., Mayne, D. Q., Diehl, M., et al. (2017). *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI.

Reiter, R., Hoffmann, J., Reinhardt, D., Messerer, F., BaumgÃĪrtner, K., Sawant, S., Boedecker, J., Diehl, M., and Gros, S. (2025). Synthesis of model predictive control and reinforcement learning: Survey and classification. *arXiv preprint arXiv:2502.02133*.

Roberts, P. (1979). An algorithm for steady-state system optimization and parameter estimation. *International Journal of Systems Science*, 10(7):719–734.

Rodriguez-Blanco, T., Sarabia, D., Navia, D., and De Prada, C. (2015). Modifier-adaptation methodology for rto applied to distillation columns. *IFAC-PapersOnLine*, 48(8):223–228.

Rozwood, P., Mehrez, E., Paehler, L., Sun, W., and Brunton, S. L. (2024). Koopman-assisted reinforcement learning. *arXiv preprint arXiv:2403.02290*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

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

Schulz, F., Hoffmann, J., Zhang, Y., and Boedecker, J. (2024). Learning when to trust the expert for guided exploration in RL. In *ICML 2024 Workshop: Foundations of Reinforcement Learning and Control – Connections and Perspectives*.

Schulze, J. C., Doncevic, D. T., Erwes, N., and Mitsos, A. (2023). Data-driven model reduction and nonlinear model predictive control of an air separation unit by applied koopman theory. *arXiv preprint arXiv:2309.05386*.

Schwenzer, M., Ay, M., Bergs, T., and Abel, D. (2021). Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349.

Siebenthal, C. and Aris, R. (1964). Studies in optimization—vi the application of pontryagin's methods to the control of a stirred reactor. *Chemical Engineering Science*, 19(10):729–746.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic policy gradient algorithms. In *International Conference on Machine Learning*, pages 387–395.

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

Son, S. H., Choi, H.-K., Moon, J., and Kwon, J. S.-I. (2022a). Hybrid Koopman model predictive control of nonlinear systems using multiple EDMD models: An application to a batch pulp digester with feed fluctuation. *Control Engineering Practice*, 118:104956.

Son, S. H., Narasingam, A., and Kwon, J. S.-I. (2022b). 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. *Journal of Process Control*, 118:26–36.

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

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063.

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

Velioglu, M., Zhai, S., Rupprecht, S., Mitsos, A., Jupke, A., and Dahmen, M. (2025). Physics-informed neural networks for dynamic process operations with limited physical knowledge and data. *Computers & Chemical Engineering*, 192:108899.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272.

Wabersich, K. P. and Zeilinger, M. N. (2018). Linear model predictive safety certification for learning-based control. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 7130–7135. IEEE.

Wabersich, K. P. and Zeilinger, M. N. (2021). A predictive safety filter for learning-based control of constrained nonlinear dynamical systems. *Automatica*, 129:109597.

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

Weigand, S., Klink, P., Peters, J., and Pajarinen, J. (2021). Reinforcement learning using guided observability. *arXiv preprint arXiv:2104.10986*.

Weissenbacher, M., Sinha, S., Garg, A., and Yoshinobu, K. (2022). Koopman q-learning: Offline reinforcement learning via symmetries of dynamics. In *International Conference on Machine Learning*, pages 23645–23667. PMLR.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

Williams, C. and Rasmussen, C. (1995). Gaussian processes for regression. *Advances in neural information processing systems*, 8.

Williams, M. O., Hemati, M. S., Dawson, S. T., Kevrekidis, I. G., and Rowley, C. W. (2016). Extending data-driven Koopman analysis to actuated systems. *IFAC-PapersOnLine*, 49(18):704–709.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256.

Wu, S., Shi, L., Wang, J., and Tian, G. (2022). Understanding policy gradient algorithms: A sensitivity-based approach. In *International Conference on Machine Learning*, pages 24131–24149. PMLR.

Xu, J., Makoviychuk, V., Narang, Y., Ramos, F., Matusik, W., Garg, A., and Macklin, M. (2022). Accelerated policy learning with parallel differentiable simulation. *arXiv preprint arXiv:2204.07137*.

Yao, H., Choi, C., Cao, B., Lee, Y., Koh, P. W. W., and Finn, C. (2022). Wild-time: A benchmark of in-the-wild distribution shift over time. *Advances in Neural Information Processing Systems*, 35:10309–10324.

Yin, H., Welle, M. C., and Kragic, D. (2022). Embedding Koopman optimal control in robot policy learning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 13392–13399. IEEE.

Zaremba, W. and Sutskever, I. (2014). Learning to execute. *arXiv preprint arXiv:1410.4615*.

Zhang, M., Marklund, H., Dhawan, N., Gupta, A., Levine, S., and Finn, C. (2021a). Adaptive risk minimization: Learning to adapt to domain shift. *Advances in Neural Information Processing Systems*, 34:23664–23678.

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