TU Delft

Delft University of Technology

Document Version
Final published version

Citation (APA)
Oliveira da Silva, C. F. (2026). *Learning-based Model Predictive Control for Mixed-Logical Dynamical and Linear Systems*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:aca1ea61-ff31-43fd-a22c-9ed677bcc1cc

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.

Learning-based Model Predictive Control

for Mixed-Logical Dynamical and Linear Systems



Caio Fabio Oliveira da Silva

LEARNING-BASED MODEL PREDICTIVE CONTROL FOR MIXED-LOGICAL DYNAMICAL AND LINEAR SYSTEMS

LEARNING-BASED MODEL PREDICTIVE CONTROL FOR MIXED-LOGICAL DYNAMICAL AND LINEAR SYSTEMS

Dissertation

for the purpose of obtaining the degree of doctor
at Delft University of Technology,
by the authority of the Rector Magnificus,
Prof. dr. ir. H. Bijl,
chair of the Board for Doctorates,
to be defended publicly on
Monday 16 November 2026 at 12:30

by

Caio Fabio OLIVEIRA DA SILVA

This dissertation has been approved by the (co)promotors.

Composition of the doctoral committee:

Rector Magnificus,	chairperson
Prof. dr. ir. B. De Schutter,	Delft University of Technology, promotor
Dr. A. Dabiri,	Delft University of Technology, copromotor
Independent Members:	
Prof. dr. T. Keviczky,	Delft University of Technology
Prof. dr. M. M. de Weerdt,	Delft University of Technology
Prof. dr. ir. B. Besselink,	University of Groningen
Prof. dr. M. Zanon,	IMT School for Advanced Studies Lucca, Italy
Prof. dr. C. Ocampo-Martinez,	Universitat Politécnica de Catalunya, Spain

The work in this dissertation has been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101018826 - CLariNet).







Keywords: Model predictive control, learning-based control, reinforcement learning, machine learning, hybrid systems, power systems, transportation systems, embedded systems

Cover by: C. F. O. da Silva and F. Portugal

Front cover: A simplified silhouette of Brazil filled with the polyhedral state-space partition induced by the explicit model predictive control law of a double-integrator system.

Printed by: Ridderprint, www.ridderprint.nl

Copyright © 2026 by C. F. O. da Silva

ISBN/EAN: 978-94-6384-964-7 (Paperback)

ISBN/EAN: 978-94-6518-335-0 (E-book)

The research data and code supporting the work in this dissertation are available at:
`https://doi.org/10.4121/277b2054-94e1-4d6a-b16c-db448bd8c4c5`

An electronic version of this dissertation is available at:
`http://repository.tudelft.nl/`.

The first principle is that you must not fool yourself and you are the easiest person to fool.

Richard P. Feynman

CONTENTS

Summary **xi**

Acknowledgements **xiii**

1 Introduction **1**
1.1 Motivation . . . 1
1.1.1 Infrastructure systems . . . 1
1.1.2 Urban rail transit systems . . . 2
1.1.3 Embedded systems . . . 2
1.2 Technical background . . . 3
1.3 Contributions . . . 3
1.4 Outline . . . 5

2 Integrating RL and MPC for mixed-logical dynamical systems **7**
2.1 Introduction . . . 7
2.1.1 Background . . . 7
2.1.2 Related work . . . 9
2.1.3 Contributions . . . 11
2.2 Control problem . . . 11
2.3 A novel integration of reinforcement learning and MPC . . . 14
2.3.1 Decoupling of decision variables . . . 14
2.3.2 Role of reinforcement learning . . . 15
2.4 Illustrative example: microgrid control . . . 19
2.4.1 Setup . . . 20
2.4.2 Simulation results . . . 22
2.5 Discussion . . . 26
2.6 Conclusions . . . 29
Appendix . . . 30
2.A System description . . . 30
2.A.1 Storage unit . . . 30
2.A.2 Generation units . . . 31
2.A.3 Main grid . . . 31
2.A.4 Assumptions . . . 32
2.A.5 Control scheme . . . 32
2.A.6 Control problem . . . 32
2.B Supervised learning approach . . . 34

3 Learning-based MPC for real-time train rescheduling 37
3.1 Introduction . . . 37
3.1.1 Passenger-oriented train scheduling . . . 38
3.1.2 MPC for real-time train scheduling . . . 39
3.1.3 Learning-based train scheduling . . . 40
3.1.4 Chapter contributions and structure . . . 41
3.2 Problem description and explanations . . . 42
3.3 Mathematical formulation . . . 45
3.3.1 Notations . . . 45
3.3.2 Train operation constraints . . . 45
3.3.3 Rolling stock circulation constraints . . . 48
3.3.4 Passenger flow constraints . . . 50
3.4 Model predictive control for train rescheduling . . . 52
3.4.1 Problem formulation . . . 52
3.4.2 MINLP-based MPC for real-time train rescheduling . . . 53
3.4.3 MILP-based MPC for real-time train rescheduling . . . 55
3.5 Learning-based MPC for train rescheduling . . . 56
3.5.1 Presolve techniques . . . 57
3.5.2 Learning setting . . . 57
3.5.3 Decoupling the decision variables . . . 58
3.5.4 Offline training . . . 58
3.5.5 State reduction approach . . . 60
3.5.6 Online evaluation . . . 62
3.6 Case study . . . 63
3.6.1 Simulation setup . . . 63
3.6.2 Simulation results without state reduction . . . 67
3.6.3 Simulation results with state reduction . . . 73
3.7 Conclusions . . . 75

4 A learning framework for approximate explicit MPC 77
4.1 Introduction . . . 77
4.2 Preliminaries . . . 80
4.3 Learning-based approximation approach . . . 83
4.3.1 Framework description . . . 83
4.3.2 Data acquisition . . . 83
4.3.3 Classifier training . . . 84
4.3.4 Regressor training . . . 86
4.3.5 Online evaluation . . . 86
4.3.6 Closed-loop fine-tuning . . . 88
4.4 Analysis of the proposed method . . . 89
4.4.1 Probabilistic feasibility validation . . . 89
4.4.2 Complexity analysis . . . 90
4.5 Simulation results . . . 92
4.5.1 Benchmarks . . . 93
4.5.2 Results . . . 94
4.5.3 Discussion . . . 98

4.6 Conclusions 99

5 Conclusions **101**
5.1 Research contributions 101
5.2 Scientific and societal impacts 102
5.3 Recommendations for future research 103

Curriculum Vitæ **119**

List of Publications **121**

SUMMARY

Infrastructure systems such as energy and transportation networks are essential to the quality of life and economic prosperity in modern society. Their large-scale nature and complex dynamics pose significant challenges for modeling and control. Capturing system behavior often requires accounting for both continuous and discrete dynamics, while control design must address network interactions, forecast demands, and operational constraints. In model predictive control (MPC), discrete decision variables introduce combinatorial complexity, leading to mixed-integer optimization problems whose online solution may become computationally intractable. At the same time, even in purely continuous settings, the strict real-time and memory constraints of embedded hardware can limit the practical deployment of MPC. As societal demands continue to grow and infrastructure and embedded systems become increasingly complex, control approaches must evolve to ensure timely, efficient, and implementable decision-making.

In this context, this PhD thesis proposes learning-based MPC frameworks in which machine learning techniques are systematically integrated to alleviate the online computational burden of MPC. The thesis comprises three main contributions: a reinforcement learning (RL)-based MPC scheme for assigning the discrete decision variables in mixed-logical dynamical (MLD) systems; a supervised learning-based MPC approach for train rescheduling in urban rail transit networks, and a learning-based framework for approximating explicit MPC for constrained linear systems. Across all contributions, the central principle is to shift the computational effort offline by learning models to predict discrete decision variables, affine controllers, or value functions, thereby enabling efficient real-time MPC implementations.

In the first part of the thesis, we present a novel framework integrating RL and MPC for MLD systems. An RL policy is trained through repeated interaction with the environment to assign the discrete decision variables, while the continuous decision variables are computed via constrained optimization. During online operation, the discrete variables are removed from the MPC problem, substantially decreasing the online computational burden associated with mixed-integer formulations. A central contribution is the partitioning of the state-action value function across the prediction horizon, a key element in rendering the learning problem tractable. Without this partitioning, the large combinatorial discrete action space would hinder efficient exploration and training, degrading the closed-loop performance of the resulting RL policy. Simulation results on a microgrid system using real-world data demonstrate that the proposed approach significantly reduces the online computation time compared to a commercial mixed-integer programming solver while maintaining a low optimality gap and a high feasibility rate.

In the second part of the thesis, we address the problem of passenger-oriented real-time train rescheduling on urban rail transit networks. A centralized supervised learning-based MPC scheme is employed to determine the train timetable and train composition and to reduce passenger delays and operational costs while satisfying the system con-

straints and considering forecast passenger demand. The MPC problem can be formulated as a mixed-integer nonlinear or linear program, depending on whether the exact objective function or a linear approximation is used. However, the discrete decision variables significantly increase the online computational complexity. To avoid the solution of mixed-integer programs online, we employ a supervised learning-based MPC scheme where the discrete decision variables are determined by tailored presolve techniques and a recurrent neural network. The continuous decision variables are then computed via the MPC optimizer to update the train timetable. A state reduction procedure is also introduced, where a lower-dimensional state is used for learning and the full state is employed for optimization. Numerical examples based on real-world data reveal a significant reduction in computation time with minimal loss of performance compared with a commercial mixed-integer programming solver.

In the third part of this thesis, we propose a learning-based framework to approximate the MPC control law for linear constrained systems. In the offline phase, a classifier is trained to predict a state-dependent set of feedback gains derived from solutions to the implicit MPC problem, while a regressor is trained to approximate the corresponding cost-to-go function. The k-nearest neighbors algorithm is considered to exploit the structure of the MPC control law, and a neural network is selected for cost-to-go approximation. During online operation, the classifier is evaluated at the current system state to generate a set of state feedback gains, from which input candidates are computed. The cost associated with each input is computed by combining the immediate input cost with the cost-to-go predicted by the regressor, and an admissible input with the lowest cost is selected. This two-stage architecture enables the approximate MPC control law to be stored in a compact form and efficiently evaluated, thereby reducing computational and memory requirements for implementation. Simulation experiments illustrate the improvements of the proposed approach over explicit MPC, implicit MPC, and alternative approximate methods in terms of online computation time and memory footprint, while offering competitive closed-loop performance.

In summary, the thesis investigates the systematic integration of learning methods into MPC for MLD and constrained linear systems. The proposed learning-based MPC frameworks reduce the online computational burden of MPC, including RL-based MPC for MLD systems, supervised learning-based MPC for train rescheduling, and approximate learning-based explicit MPC. The results demonstrate that these frameworks can reduce the computational bottlenecks associated with discrete decision variables and resource-limited embedded implementations, supporting the efficient and practical deployment of MPC in real-time applications.

ACKNOWLEDGEMENTS

The past four years have gone by fairly quickly, reflecting the intense, challenging, and transformative experiences that shaped this journey. From the excitement of seeing research ideas come to life to the distressing moments of uncertainty and aimlessness, this period has been filled with ups and downs. Looking back, it is clear that reaching the conclusion of my Ph.D. would not have been possible without the many people who were directly or indirectly part of this journey. I would therefore like to take this opportunity to express my gratitude to all who accompanied me along the way.

This journey would not have been possible without my supervisors, Bart and Azita. Thank you for giving me the opportunity to start this journey, for keeping me from going astray along the way, and for patiently guiding me to the finish line. I greatly appreciated the freedom to follow my research interests, the valuable guidance to select the most sensible directions, and the numerous discussions that sharpened my critical thinking. Your supervision shaped me into the researcher I am today, and I will be forever grateful for it. Thank you, Bart, for the rigorous, pragmatic, and detailed feedback and your prompt availability whenever I needed it. Thank you, Azita, for your insightful perspectives in many of our discussions.

I extend my gratitude to my colleagues in the Delft Center for Systems and Control. I would like to give special thanks to my office mates, Filippo, Alessandro, and Kanghui, who shared this journey with me from the very beginning, supported me along the way, and inspired me to be a sharper thinker. Thanks to the other members of the ClariNet project, Francesco, Sam, Gianpietro, Shengling, Anil, and Changrui, for the academic discussions, support, dinners, and chats. Thank you, Xiaoyu, for your dedication during our research collaboration. Thank you, Pedro, Athina, Leila, Maarten, Reza, David, Sander, Gabriel, Lars, and Amin, for being part of this journey in several different ways.

Thank you, Prof. Alberto Bemporad, for hosting me at IMT Lucca, Italy, during my research visit and for the fruitful research discussions. My experience in Italy would not have been nearly as good without my colleagues Chiara, Yuxuan, Francesco, and Sampath, who warmly welcomed me and helped me to quickly feel at ease in a new research group.

Thanks, Claudia, Sanjana, and Alessandro, for being great housemates during my time in Delft. Thanks, Gabriel, Vitor, Arthur, Lucas, Matheus, Alex, Rafael, and João, for the enduring friendship since our bachelor's studies. Remembering our past experiences whenever we meet in Brasília is always a joy. Witnessing you grow every time I visit genuinely cheers me up. Thanks, Iago, for our sporadic, long, and enjoyable conversations about our lives, and for being someone who can truly relate to my life trajectory since high school.

Warm thanks to everyone in the forró community of The Hague. Dancing has been my main passion outside academia, and it has been a regular reminder that there is life beyond research. Forró introduced me to many incredible people who have been an essential part of making the Netherlands feel like home. I am especially grateful to Rafael

for being such a reliable, accepting, and entertaining friend, and to Bruna, Mike, Nathalia, Thanja, and Erick for the many dances, conversations, and fun moments we shared. I also extend my thanks to friends I made at festivals all over Europe: Stephanie, Daniel, Bibiane, Luiz, and Genario. You have all been a refuge during stressful times.

I want to thank my girlfriend, Francisca Portugal, for her daily support and companionship. Thank you for sharing so many good moments with me over these years. Thanks for being the person I could lean on during difficult moments. Thanks for your attentive ears when I complained about the Ph.D. journey. Thanks for your playful silliness and positive outlook that often helped take my mind away from my worries.

I would like to express my profound gratitude to my family for their unwavering support, unconditional love, and encouragement. Thank you, Mom, for always believing in me, showing me the meaning of love and stability, and working hard so I could have access to a good education. Thank you, Father, for your relentless and selfless dedication to supporting me and everyone around you. Thank you, Grandma, for teaching me how to read my first words and for your loving presence. Thank you, brothers, for sharing my life journey with me and for helping me reconnect with myself every time I come back to Brazil. Thank you, my nephew, for bringing more joy to our family. I carry the support of all of you with me as I move on to the next chapter.

Caio Fabio Oliveira da Silva
The Hague, March 2026

1

INTRODUCTION

Model predictive control (MPC) is an optimization-based control strategy that enables the consideration of a sequence of future control actions based on model-based predictions of future states. By implementing only the first control action and resolving the corresponding optimization problem at each time step using updated state measurements, closed-loop feedback is incorporated into the control law and constraints are handled by design. However, this control strategy can be particularly computationally demanding for systems with hybrid, i.e., mixed continuous and discrete, dynamics due to the presence of discrete decision variables [17]. Moreover, applications with limited hardware resources may impose strict constraints on the memory footprint and computational complexity of the controller. This chapter discusses motivating applications in which MPC is particularly suitable, but its computational complexity poses challenges for practical implementation. The chapter then presents the main contributions and concludes with an outline of the remainder of the thesis.

1.1. MOTIVATION

This thesis is motivated by three distinct multidisciplinary problems, involving mixed-logical dynamical systems, urban rail transit systems, and embedded systems. In the following, each problem, along with its challenges, is separately described.

1.1.1. INFRASTRUCTURE SYSTEMS

Ensuring that power generation matches consumers' demand at every instant is an incessant and monumental task. The power grid is just an example of many critical infrastructure systems that keep the modern world running. Other examples include transportation and water networks, which also provide indispensable services to our daily lives. Along with continuous dynamics, modeling such systems may require the use of discrete variables to represent mode transitions, logic statements, topology changes, on/off switching, and other discrete phenomena. Controlling such systems is challenging due to their large-scale, complex dynamics, operational constraints, and multi-objective

cost functions, which should consider several stakeholders.

In this thesis, the mixed-logical dynamical (MLD) system formalism is considered due to its tractability for optimization-based control and its capacity to represent a wide range of hybrid systems [17]. For control of MLD systems, MPC stands out for its ability to naturally handle continuous and discrete dynamics, continuous and discrete inputs, multi-objective cost functions, and constraints. However, the application of MPC in this setting requires the solution of mixed-integer linear and quadratic programs, whose worst-case computational complexity scales exponentially with the number of discrete variables [114]. The importance of complex infrastructure systems and the existing computational obstacles inspire the investigation of computationally efficient control strategies at the intersection of MPC and MLD systems.

1.1.2. URBAN RAIL TRANSIT SYSTEMS

In many cities, urban rail transit systems play an important role in reducing urban congestion and pollution, becoming an indispensable component of sustainable urban development. These transportation systems provide safe and efficient mobility at a large scale. The operation of urban rail transit systems involves consideration of a multitude of factors, including demands, operational costs, constraints, delays, and disruptions. The increasing passenger demand and the rapid expansion of urban rail networks pose challenges for their operation. The main goal of operating urban rail networks is to improve passenger satisfaction while reducing operational costs and satisfying operational constraints. The passenger demand at each platform of an urban rail network is inherently time-varying. Consequently, the timetable for arrivals and departures should be frequently rescheduled to ensure reliable and efficient service for the passengers. The train transport capacity, e.g., the maximum number of passengers per train service, is also a key discrete decision variable in adjusting the operation of the urban rail network to time-varying operating conditions. The exploitation of the urban rail network's resources for societal benefit requires a systematic decision-making procedure to consider the several factors that affect its operation.

Several works have employed MPC in urban transit systems for its capacity of handling nonlinear dynamics, constraints, and multi-objective formulations [6], [27], [31], [43]. However, formulations of the urban traffic management problem often lead to mixed-integer programs whose computational complexity scales poorly with the number of discrete variables required to model such discrete phenomena (e.g., train composition, ordering, transfer connectivity) in large-scale urban traffic networks. Hence, online computational complexity remains one of the main obstacles to jointly incorporating flexible train composition and rolling stock circulation in real-time train rescheduling.

1.1.3. EMBEDDED SYSTEMS

Over the last decades, advances in numerical optimization and electronics allowed MPC to be implemented in embedded systems, including automotive, aerospace, and bioengineering applications [7], [16], [32]. For such systems with a fast sampling rate and limited computational resources, the MPC implementation should satisfy tight memory footprint specifications while guaranteeing the solution of an optimization problem within a narrow time window. For deploying the MPC controller in real-world systems, the controller's

evaluation should be numerically robust in single precision arithmetic, and the code should be simple enough to enable verification and certification [69], [75]. Consequently, the demanding computational requirements remain an obstacle that prevents the broader adoption of MPC in embedded systems [37].

Many of these control applications in embedded systems can be modeled as constrained linear systems. For this class of systems, several techniques have been developed to reduce the computational complexity of online evaluation, such as explicit MPC and approximate methods. Each of these alternatives provides a distinct trade-off between computational complexity, memory footprint, performance, and guarantees for properties such as stability and feasibility. The relative importance of each of these factors in the trade-off is highly dependent on the application of interest. In particular, this thesis investigates an approximate method that favors closed-loop performance and the reduction in memory footprint over other factors.

1.2. TECHNICAL BACKGROUND

In this thesis, we explore the intersection of machine learning and MPC to develop efficient control schemes in terms of computational complexity and memory footprint. Herein, machine learning techniques are employed for three main reasons. First, they can leverage datasets generated from repeated offline solutions of the MPC problem. Second, their online evaluation complexity is typically low. Third, they offer strong function approximation capabilities, enabling the representation of complex mappings between the system state and the corresponding control actions, control laws, and value functions.

In particular, this thesis employs two paradigms of machine learning, namely reinforcement learning and supervised learning. Reinforcement learning (RL) is a framework in which an agent adapts a policy through repeated interaction with an environment to learn optimal sequential decision-making strategies [129]. In this thesis, deep Q-learning is utilized, where the central component is the Q-function, which estimates the expected return of a given state-action pair, provided that a specified policy is followed thereafter [96]. Supervised learning is a paradigm in which labeled input-output pairs are employed to devise a model that predicts outputs for unseen inputs [54]. In particular, we employ neural networks for regression and the k-nearest neighbors algorithm for set-valued classification [41], [54]. In this thesis, several control schemes are proposed in which these machine learning paradigms are integrated into MPC to reduce its computational load.

1.3. CONTRIBUTIONS

The common thread that weaves through the problems that motivate this thesis is that each one exhibits demanding computational requirements, particularly computation time and memory footprint. This leads to the unifying theme of this thesis, which is the exploration of techniques to reduce the computational requirements for implementing MPC in linear and MLD systems. To achieve this goal, this thesis proposes several control schemes in which machine learning techniques are leveraged to reduce both computation time and memory footprint in MPC implementations.

The main contributions of the thesis are listed below:

1. We propose a control scheme for MLD systems that integrates reinforcement learn-

ing (RL) and MPC. The main objective is to reduce the computational complexity of solving mixed-integer linear or quadratic programs arising from MPC formulations for MLD systems. A fundamental concept in this control scheme is the decoupling of the decision on the discrete and continuous variables. During training, an RL agent repeatedly interacts with the environment, which consists of the system dynamics and an MPC controller that determines the continuous variables, to learn a policy to determine the discrete decision variables. To render the learning problem tractable in a combinatorial action space, we introduce the decoupled Q-function, which splits the Q-function associated with the RL policy across each time step of the prediction horizon. Since the discrete decision variables are fixed by the RL policy during online evaluation, this control scheme effectively simplifies the mixed-integer linear or quadratic programs to linear or quadratic programs, substantially reducing the computational complexity of the MPC controller.

2. We combine supervised learning and MPC to propose a control scheme for passenger-oriented real-time train rescheduling, considering flexible train composition and rolling stock circulation to address time-varying passenger demands. This approach also employs the decoupling of the decision on the discrete and continuous variables. Unlike the previous RL-based method, a supervised learning classifier is trained to predict assignments for the discrete decision variables based on offline solutions of the mixed-integer nonlinear program arising from the MPC formulation. Presolve techniques are designed to reduce the number of decision variables by exploiting the coupling between the variables and constraints. Moreover, a state reduction procedure is introduced to further improve online evaluation speed and to reduce the use of computational resources during offline training. In the resulting control scheme, a lower-dimensional state is used for learning, and the full state is employed in the MPC controller. Numerical simulations based on real-world data reveal the effectiveness of the proposed method in turning the mixed-integer nonlinear program into a nonlinear program containing only real-valued decision variables, thereby significantly reducing computation time.

3. We develop a learning-based framework for approximate explicit MPC for constrained linear systems. This framework is particularly targeted at embedded systems where memory footprint and computational complexity are the main design concerns. In the offline phase, a classifier is trained to output a set of affine state feedback controllers, and a regressor is trained to approximate the cost-to-go of the explicit MPC controller. In both cases, training is carried out using supervised learning. During online evaluation, a set of control input candidates is generated based on the affine controllers predicted by the classifier. Then, the control inputs that violate the input and state constraints are discarded, and a regressor is used to estimate the total cost of applying each of the remaining input candidates. Finally, the control input that is associated with the lowest cost is selected. We demonstrate through numerical simulations that the proposed framework offers competitive computational complexity, memory footprint, and closed-loop performance compared to implicit MPC, explicit MPC, and several approximate methods.

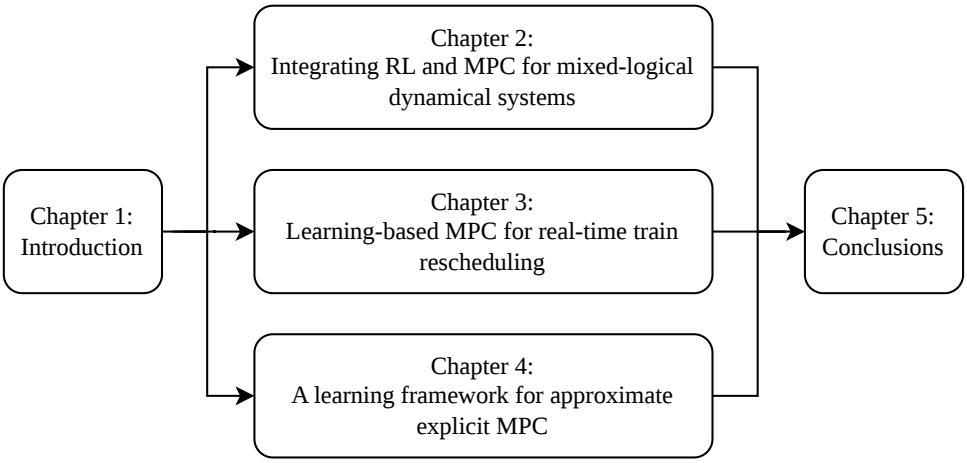


Figure 1.1: Structure of the thesis.

1.4. OUTLINE

The structure of this thesis is illustrated in Figure 1.1. The remainder of the thesis is divided into four main chapters. Chapters 2, 3, and 4 correspond to published or submitted journal papers. Chapter 2 focuses on the description of the RL-based approach aimed at reducing the computation time for implementing MPC in MLD systems. Chapter 3 addresses the problem of passenger-oriented train rescheduling. To simplify the online evaluation of the MPC controller, it presents a method that leverages supervised learning based on offline solutions of the original problem. Chapter 4 introduces a learning-based framework to approximate the explicit MPC control law for applications with limited hardware resources, such as embedded systems. Finally, Chapter 5 concludes the thesis with the main contributions, the technical and societal impacts of the thesis, and the recommendations for future research.

2

INTEGRATING RL AND MPC FOR MIXED-LOGICAL DYNAMICAL SYSTEMS

This chapter proposes an approach that integrates reinforcement learning (RL) and model predictive control (MPC) to solve finite-horizon optimal control problems in mixed-logical dynamical systems efficiently. Optimization-based control of such systems with discrete and continuous decision variables entails the online solution of mixed-integer linear programs, which suffer from the curse of dimensionality. In the proposed approach, by repeated interaction with a simulator of the system, a reinforcement learning agent is trained to provide a policy for the discrete decision variables. During online operation, the RL policy simplifies the online optimization problem of the MPC controller from a mixed-integer linear program to a linear program, significantly reducing the computation time. A fundamental contribution of this chapter is the definition of the decoupled Q-function, which plays a crucial role in making the learning problem tractable in a combinatorial action space. We motivate the use of recurrent neural networks to approximate the decoupled Q-function and show how they can be employed in a reinforcement learning setting. A microgrid system is used as an illustrative example where real-world data is used to demonstrate that the proposed method substantially reduces the maximum online computation time of MPC (up to 20×*) while maintaining high feasibility and average optimality gap lower than* 1.1%.

2.1. INTRODUCTION

2.1.1. BACKGROUND

Complex infrastructure systems, such as energy, transportation, and water networks, are pervasive in our modern world. Analysis and control design of such systems is very

This chapter is based on the paper [122].

challenging due to their size and intricate behavior. Such systems are often governed by a combination of continuous and discrete dynamics and a hybrid action space. Examples of discrete elements include mode transitions, topology changes, on/off switching, logic statements, and discrete inputs. To capture these features, several works have considered hybrid system models for critical infrastructure networks [83], [102], [105].

For the control of hybrid systems, model predictive control (MPC) arises as a promising technique due to its ability to handle constrained complex systems with discrete and continuous decision variables [25]. Furthermore, MPC has solid theoretical foundations with regard to stability, performance, and safety [111]. However, these advantages often come at the cost of intense online computational requirements, limiting the use of MPC due to hardware constraints or execution time limitations. This bottleneck is particularly pronounced in MPC for hybrid systems, where the optimizer has also to consider a sequence of discrete decision variables over the prediction horizon. In this case, the optimization problem then becomes a mixed-integer program, which is NP-hard [23].

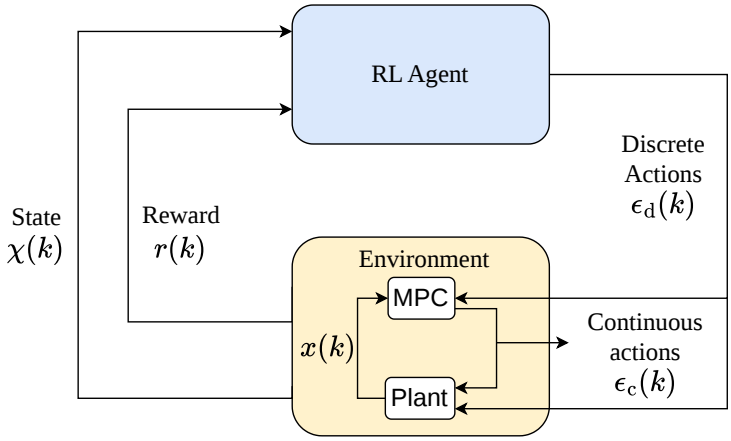


Figure 2.1: A depiction of the proposed control scheme that integrates reinforcement learning into an MPC framework for mixed-logical dynamical system. The agent's goal is to maximize its long-term reward. It learns to adapt its policy by repeatedly interacting with the environment, that is, by sending a discrete action $\epsilon_d(k)$ and by receiving the extended state $\chi(k)$ and immediate reward $r(k)$. The MPC controller, which is lumped in the environment, receives this discrete action $\epsilon_d(k)$ and then solves an optimization problem to determine the continuous action $\epsilon_c(k)$. Finally, the input $\epsilon(k)$ is fed to the system, and the next state is computed.

The most widely used technique to solve mixed-integer programs in modern solvers is the branch-and-bound [114] algorithm, which has worst-case exponential time complexity. Branch-and-bound algorithms mitigate this issue by efficiently pruning branches of the search tree by estimating lower and upper bounds of the objective function and by proposing cutting planes, which reduce the feasible set of the mixed-integer programs by introducing linear inequalities as additional constraints [39]. One alternative to branch-and-bound for control of hybrid systems is to pre-compute the MPC control law offline and simply evaluate this function online. For hybrid systems, the offline computation of the explicit MPC control law via multi-parametric programming and dynamic programming was explored in [23]. However, this approach, also referred to as explicit MPC, can

be only successfully applied to low-dimensional linear systems. Despite improvements in branch-and-bound and explicit MPC, solving mixed-integer programs with a significant number of discrete decision variables remains fundamentally difficult.

To reduce the computational load of MPC for hybrid systems, this chapter proposes the integration of reinforcement learning (RL) into the MPC framework for mixed-logical dynamical (MLD) systems. In our approach, deep RL is employed to train a policy to assign the discrete decision variables of the control problem. In the offline phase, the RL agent repeatedly interacts with a simulator of the system and adapts its policy through a reward signal, as represented in Fig. 2.1. For online inference, along a paradigm that is similar to that of recent learning-based approaches [29], [30], [92], [93], [112], the RL policy fixes the discrete decision variables and simplifies the mixed-integer program to a optimization problem involving only continuous variables, significantly reducing the online computational load of the MPC controller.

In our setting, the action space of the RL agent is combinatorial since it involves the determination of a sequence of discrete decision variables over a prediction horizon. Applying deep RL in combinatorial action spaces can be very challenging given that the neural networks representing the state-action value function and the policy need to have as many units in the output layer as the number of actions in the action space [45], [48], [124], [132]. To make RL a viable option for mixed-integer MPC, the proposed approach learns a state-action value function (Q-function) for each time step of the prediction window. Furthermore, such Q-functions are efficiently approximated in a recursive fashion by a recurrent neural network. The main benefit is that the action space is decoupled across time steps, preventing the combinatorial growth with respect to the prediction horizon. Consequently, the learning problem becomes more tractable. Hereafter, the Q-function for each time step of the prediction window is referred to as the decoupled Q-function.

2.1.2. RELATED WORK

Recently, supervised learning (SL) has been explored in several ways when combined with hybrid MPC to reduce the online computation time of the resulting mixed-integer programs [29], [30], [92], [93], [112]. Although the structure of these methods is very similar to that of the proposed approach, they differ in their learning objectives: while the proposed approach seeks to maximize the cumulative reward, SL-based approaches exclusively aim to predict the optimal action. Hence, in an RL setting, infeasible actions can be explicitly discouraged through the reward function. Moreover, the proposed approach does not require the solution of mixed-integer programs. This is particularly critical in cases where obtaining labeled data is impractical or prohibitively expensive [19]. Additionaly, in SL-based approaches, the dataset is generated using the optimal policy, meaning that the classifier or regressor is trained on the corresponding optimal state distribution. On the other hand, the proposed approach relies on data collected under the RL policy itself, resulting in a state distribution that more accurately reflects the one encountered during online operation. However, RL-based approaches often face some challenges such as sensitivity to hyperparameters, sample inefficiency, exploration, and reward design [19]. For a more general and nuanced discussion on RL and SL, the reader is referred to [19].

Several approaches have been developed to handle combinatorial action spaces in RL [45], [48], [124], [132]. However, these methods rely on exploiting problem-specific structures and do not address settings in which continuous decision variables are also a component of the action space.

Some works have applied learning to reduce the solution time of mixed-integer programs by embedding learning into the branch-and-bound algorithm to substitute expert-designed heuristics with learned rules. In many works, learning has been used to improve cutting plane rules [131], branching variable selection [151], and node selection [8]. In [21] and [20], the computational burden of MPC is reduced by using an RL policy to determine a state-dependent prediction horizon and a state-dependent recomputation policy, respectively. A hybrid approach is developed in [98], where a neural network is trained to imitate a branching heuristic and another is employed to compute multiple partial assignments of the discrete variables. These methods still rely on a branch-and-bound solver during online operation, which can be prohibitive for certain applications due to computational and implementation complexity [50].

In [95], the state-action value function (Q-function) of time-invariant mixed-logical dynamics systems is approximated. The core of their approach is to parameterize the Q-function with Benders cuts and estimate the Q-function from a lower bound. However, the scope of application of this approach is limited since it does not consider discrete states and inputs.

An MPC controller is used as a function approximator of the state-action value in [57], where the parameters of the controller are tuned by policy gradient methods. In [56], the same approach was extended to mixed-integer optimal control. The primary focus of these works is to leverage learning to improve performance rather than to reduce the computational cost of the MPC controller.

Under the light of the survey paper [113] on the integration of MPC and RL, the presented approach can be characterized as a blend of MPC as an expert actor, since it is used to train the RL agent, and of MPC for post-processing, as the MPC controller is used during online operation to determine the continuous decision variables after the assignment of discrete decision variables by the RL agent. Additionally, in the categories presented in [19], the proposed approach falls into the category of methods that learn a policy by experience since no labeled data is used during training. The methods discussed in the survey concern the use of experience to train a policy either to improve problem-specific heuristics or to improve solver heuristics. In contrast, our approach addresses the use of experience to directly determine the discrete decision variables within the MPC framework.

In [47], [91], [99], RL is applied to jointly learn discrete and continuous policies. In the artificial intelligence community, this problem is explored under the framework of Markov decision processes with parameterized actions. The problem is very similar to that of control of hybrid systems due to the nature of the action space, which has discrete and continuous elements. These works offer useful insight into the parameterization and training of policies for a hybrid action space. However, in cases where a reasonably accurate model is available, the use of MPC to determine the continuous actions can improve performance since the continuous actions solution is optimal with respect to the choice of discrete actions. Moreover, with MPC in the online evaluation, constraints can

be explicitly enforced in the optimization problem and uncertainties can be addressed with stochastic or robust formulations.

Outside the domain of hybrid systems, other works have also integrated learning into the MPC framework in various schemes. The most popular approaches include the adaptation of the system model, the use of MPC as a safety filter for RL, and the online tuning of the cost and constraint functions for performance, see [55], [64] for more methods on the interplay of learning and MPC. These approaches are aimed towards improving performance, safety, and robustness. Nevertheless, these approaches do not address the computational issues, which is the main concern of this chapter.

2.1.3. CONTRIBUTIONS

We propose an integrated reinforcement learning and MPC method that solves mixed-integer linear programs with low computational footprint, low optimality gap, and high feasibility rate. We build on the existing idea of decoupling the decision on the discrete and continuous variables with learning and MPC [29], [30], [92], [93], [112]. However, we explore a novel paradigm by employing reinforcement learning in place of supervised learning to directly optimize for control performance – instead of minimizing the classification error – and to avoid the use of branch-and-bound in both the offline and online phases of the algorithm.

The main contributions of the chapter with regard to the literature are:

- We propose a novel integrated reinforcement learning and MPC framework for control of mixed-logical dynamical systems. During online operation, the reinforcement learning agent simplifies a mixed-integer linear program into a linear program by fixing the discrete decision variables.
- The Q-function is partitioned across the prediction horizon, and the definition of decoupled Q-functions is conceived to make the learning problem tractable. The decoupled Q-function is approximated by a recurrent neural network, and its role in the reinforcement learning algorithm is described.
- The approach is evaluated on an illustrative example that jointly addresses the unit commitment and economic dispatch problems in a microgrid. The maximum computation time is reduced by up to a factor of 20 with respect to a state-of-the-art MILP solver. The comparison between the proposed approach and a method based on supervised learning reveals a trade-off: while the former outperforms in terms of feasibility, the latter outperforms in terms of optimality.

2.2. CONTROL PROBLEM

We consider MPC for its capacity to handle multivariable constrained hybrid systems. In this context, mixed-logical dynamical (MLD) systems [17] are typically used to formulate open-loop finite-horizon optimal control problems [25]. Moreover, the equivalence of MLD systems and other hybrid system modeling frameworks was established in [61], showing their broad applicability. When MLD systems are used to formulate the MPC problem and the cost function and the constraints are linear, the resulting optimization problem is a mixed-integer linear program (MILP). In this section, a general description

of the MPC optimization problem for an MLD system is first addressed to set the stage for the formulation of the control problem as an MILP.

Consider the MLD system

$$\begin{aligned} x(k+1) &= Ax(k) + B_1 u(k) + B_2 \delta(k) + B_3 z(k) + B_5, \\ E_2\delta(k) + E_3 z(k) &\leq E_1 u(k) + E_4 x(k) + E_5, \end{aligned} \tag{2.1}$$

where $x \in \mathbb{R}^{n_c} \times \{0,1\}^{n_d}$ is a vector containing the continuous and discrete system states, $u \in \mathbb{R}^{m_c} \times \{0,1\}^{m_d}$ are the continuous and discrete inputs, $\delta \in \{0,1\}^{r_d}$ is a vector with the auxiliary discrete variables, $z \in \mathbb{R}^{r_c}$ is a vector with the continuous auxiliary variables arising from the MLD modeling, and $A \in \mathbb{R}^{(n_c+n_d)\times(n_c+n_d)}$, $B_1 \in \mathbb{R}^{(n_c+n_d)\times(m_c+m_d)}$, $B_2 \in \mathbb{R}^{(n_c+n_d)\times r_d}$, $B_3 \in \mathbb{R}^{(n_c+n_d)\times r_c}$, $E_1 \in \mathbb{R}^{n_{cons}\times(m_c+m_d)}$, $E_2 \in \mathbb{R}^{n_{cons}\times r_d}$, $E_3 \in \mathbb{R}^{n_{cons}\times r_c}$, $E_4 \in \mathbb{R}^{n_{cons}\times(n_c+n_d)}$, and $E_5 \in \mathbb{R}^{n_{cons}}$. Note that the linear constraints may represent logical constraints, a byproduct of MLD modeling, or system operating constraints. The total number of constraints is denoted by n_{cons} and it accounts for state, input, and auxiliary variable constraints. For a demonstration of the application of the discrete variables and auxiliary variables to represent logical constraints, the reader is referred to Appendix 2.A, where the model for the illustrative example is characterized.

The goal is to control the system (2.1) with a receding-horizon strategy, where at each time step a finite-horizon optimal control problem is solved. Such an optimization problem can be formulated as follows:

$$\begin{aligned} \min_{\mathbf{x}(k),\epsilon_c(k),\epsilon_d(k)} \quad & J(\mathbf{x}(k),\epsilon_c(k),\epsilon_d(k)) \\ \text{s.t. } & x(k) = x'(k), \\ & x(k+l+1) = Ax(k+l) + B_1 u(k+l) + B_2\delta(k+l) + B_3 z(k+l) + B_5, \\ & E_2\delta(k+l) + E_3 z(k+l) \leq E_1 u(k+l) + E_4 x(k+l) + E_5, \\ & \text{for } l = 0,\dots,N_p-1, \end{aligned} \tag{2.2}$$

where the cost function is defined by

$$J(\mathbf{x}(k),\epsilon_c(k),\epsilon_d(k)) = \sum_{l=0}^{N_p-1} \big(\ell(x(k+l),u(k+l),\delta(k+l),z(k+l))\big) + V_f(x(k+N_p)), \tag{2.3}$$

the prediction horizon is denoted by N_p, the variable l indexes the time along the prediction horizon, the measured current state is $x'(k)$, and the final cost is represented by $V_f(\cdot)$. The predicted state trajectory over the prediction horizon is denoted by $\mathbf{x}(k) = [x^\intercal(k),\dots,x^\intercal(k+N_p)]^\intercal$. Let the predicted input be explicitly divided into continuous and discrete components $u(k) = [u_c^\intercal(k),\ u_d^\intercal(k)]^\intercal$. The stacked continuous input and auxiliary variables over the prediction horizon are represented in vector $\epsilon_c(k) = [u_c^\intercal(k), z^\intercal(k),\dots,u_c^\intercal(k+N_p-1),\ z^\intercal(k+N_p-1)]^\intercal$. Similarly, the stacked discrete inputs and auxiliary variables over the prediction horizon are expressed by $\epsilon_d(k) = [u_d^\intercal(k),\delta^\intercal(k),\dots,u_d^\intercal(k+N_p-1),\ \delta^\intercal(k+N_p-1)]^\intercal$. For the sake of simplicity, the constraints are assumed to be polyhedral, as in (2.1).

At each time step, the optimization (2.2) is solved, and the first entry of the decision variables is applied to the system. Let $(\mathbf{x}^*(k),\ \epsilon_c^*(k),\ \epsilon_d^*(k))$ be the optimal solution of (2.2).

The MPC control law then is defined by

$$u_{\mathrm{MPC}}(k) = [\epsilon^{*,\mathsf{T}}_{\mathrm{c},0}(k),\ \epsilon^{*,\mathsf{T}}_{\mathrm{d},0}(k)]^{\mathsf{T}}, \tag{2.4}$$

where the first entries of the solution vector are expressed as $[\epsilon^{*,T}_{\mathrm{c},0}(k),\ \epsilon^{*,T}_{\mathrm{d},0}(k)]^{\mathsf{T}} = [u^{*,T}_{\mathrm{c}}(k),\ z^{*,T}(k),\ u^{*,T}_{\mathrm{d}}(k),\ \delta^{*,T}(k)]^{\mathsf{T}}$.

If the cost function (2.3) is linear, e.g., state and terminal costs based on the 1-norm or the ∞-norm, then the optimization problem (2.2) can be conveniently recast as a mixed integer linear program [25]. The corresponding linear optimization problem in a compact form can be stated as follows:

$$\begin{aligned} &\min_{\epsilon(k)} \ c^{\mathsf{T}}\epsilon(k) \\ &\text{s.t. } G\epsilon(k) \leq w + Sx(k), \end{aligned} \tag{2.5}$$

where $\epsilon(k) = [\epsilon^{\mathsf{T}}_{\mathrm{c}}(k),\ \epsilon^{\mathsf{T}}_{\mathrm{d}}(k)]^{\mathsf{T}}$, $c \in \mathbb{R}^{N_{\mathrm{p}}\cdot(m_{\mathrm{c}}+r_{\mathrm{c}}+m_{\mathrm{d}}+r_{\mathrm{d}})}$, $G \in \mathbb{R}^{N_{\mathrm{p}}n_{\mathrm{cons}}\times N_{\mathrm{p}}\cdot(m_{\mathrm{c}}+r_{\mathrm{c}}+m_{\mathrm{d}}+r_{\mathrm{d}})}$, $w \in \mathbb{R}^{N_{\mathrm{p}}cons}$, $S \in \mathbb{R}^{N_{\mathrm{p}}n_{\mathrm{cons}}\times(n_{\mathrm{c}}+n_{\mathrm{d}})}$.

Note that the optimization problem (2.5) is parameterized by the state $x(k)$. Besides, the system state has an effect on the objective through the auxiliary variable $z(k)$, which is part of the decision variable $\epsilon(k)$. Similarly, the resulting optimization problem can also possibly depend on an exogenous variable $\gamma(k)$, i.e., a measured external variable outside the influence of the controller. The effect of $\gamma(k)$ on the optimization problem is to further parameterize the mixed-integer program. In principle, the exogenous variable can potentially change any of the matrices c, G, w, and S that define (2.5). In our problem formulation, we restrict the parametrization to the cost vectors c and the constraint matrix w.

The problem (2.5) can then be extended to reflect the addition of the exogenous variable:

$$\begin{aligned} &\min_{\epsilon(k)} \ c^{\mathsf{T}}(\gamma(k))\epsilon(k) \\ &\text{s.t. } G\epsilon(k) \leq w(\gamma(k)) + Sx(k). \end{aligned} \tag{2.6}$$

For ease of notation, we leave out the explicit dependence of such parameters on the exogenous signal hereafter. Since (2.6) is parameterized both by the system state $x(k)$ and the exogenous signal $\gamma(k)$, we define the augmented state

$$\chi(k) = [x^{\mathsf{T}}(k),\ \gamma^{\mathsf{T}}(k)]^{\mathsf{T}}$$

to fully characterize the parameters that define the parametric mixed-integer linear program.

The exogenous variables are necessary for the modeling of some applications, as shown in the illustrative example in Section 2.4, where such variables are comprised of electricity prices, renewable energy generation, and energy demand forecasts.

Remark 2.1. The problem formulation can be adapted for non-binary categorical variables if binary encoding is employed.

Remark 2.2. The assumption on the linearity of the objective function does not play any fundamental role in the development of the proposed approach in the next section. The choice for a linear objective is to simplify the notation and the presentation of this chapter. In principle, even a nonconvex objective cost could be chosen. Such a choice would, however, result in more complex optimization problems to be solved offline for learning and solved online for inference. Similarly, even though (2.6) can be formulated in our proposed approach with a more general parametrization in the exogenous variable $\gamma(k)$, we restrict the parametrization for the sake of simplicity and alignment with the illustrative example.

2.3. A NOVEL INTEGRATION OF REINFORCEMENT LEARNING AND MPC

In this section, we describe the training and the online evaluation of the proposed approach, which utilizes a reinforcement learning (RL) policy to assign discrete decision variables for the MILP (2.6), avoiding the online computation of the solution of this MILP. Moreover, by assigning Q-functions to each time step of the prediction window, referred to as decoupled Q-functions, the RL problem is made more tractable. We also present the efficient approximation of such decoupled Q-functions using recurrent neural networks.

Remark 2.3. In the proposed approach, the RL agent is trained entirely offline through repeated interaction with a simulator that accurately captures the system dynamics. Our framework enables the use of stochastic and robust MPC formulations to address uncertainties. Furthermore, online adaptation of the RL policy is also supported without requiring any modifications to the proposed approach, which can be beneficial for systems with time-varying dynamics.

2.3.1. DECOUPLING OF DECISION VARIABLES

The mixed-integer linear program (MILP) (2.6) is parametric with respect to the initial state $x(k)$ and possibly an exogenous signal $\gamma(k)$. Consider the set of these values in which (2.6) is feasible. In this set, the solution of (2.6) maps the signal $(x(k),\ \gamma(k))$ to their corresponding optimal decision variables $(\epsilon_c(k)^*,\ \epsilon_d(k)^*)$.

Consider the scenario where the discrete part of the optimal solution $\epsilon_d(k)^*$ is known. Then the continuous part of the solution $\epsilon_c(k)^*$ can be easily recovered by solving the problem (2.6), which becomes a linear program (LP) with the fixed discrete optimization variables, as shown below:

$$\begin{aligned} V(x(k),\ \gamma(k), \epsilon_d(k)^*) = \min_{\epsilon_c(k)}\ & c(\gamma(k))^\intercal [\epsilon_c^\intercal(k),\ \epsilon_d^\intercal(k)^*]^\intercal \\ \text{s.t. } & G[\epsilon_c^\intercal(k),\ \epsilon_d^\intercal(k)^*]^\intercal \leq w(\gamma(k)) + Sx(k) \end{aligned} . \tag{2.7}$$

In this manner, the optimal continuous decision variables can be found, provided that the optimal discrete decision variables are known. Evidently, assuming the knowledge of the discrete decision variables is not realistic. Nevertheless, we make use of this decoupling procedure by using learning approaches to provide approximate solutions to the discrete decision variables and by letting the optimization determine the continuous variables, as in (2.7). One of the main goals of this chapter is to show that, with sufficient training

data, an effective approximator for the discrete decision variables can be trained and that solving (2.7) with such approximated discrete variables yields solutions for the continuous variables without much sacrifice of optimality.

2.3.2. ROLE OF REINFORCEMENT LEARNING

Reinforcement learning is a general learning framework where the agent learns a control policy based on its interaction with the environment. It has received increasing attention in control applications due to its capacity to learn complex policies and for its low demand for online computation [129]. Next, we describe an approach that exploits the benefits of RL to efficiently determine the discrete decision variables of (2.6).

In the mixed-integer linear program (2.6), the main computational complexity stems from the discrete decision variables. In the proposed approach, reinforcement learning is used to learn a policy that maps the system state to the discrete decision variables. The training of RL policy is done offline by repeated interaction with a model of the system and historical data. During online operation, the original MILP problem (2.6) is turned into an LP problem (2.7) after the RL policy provides an approximate solution to the discrete decision variables given the current measured system state and exogenous variable. Solving such LPs of the form (2.7) is computationally cheap with state-of-the-art specific-purpose solvers. The immediate benefit of the continuous variables being determined by an optimization-based control, such as MPC, is constraint satisfaction and optimality subject to the assignment of discrete variables and the feasibility of (2.7).

In order to formulate the problem as a Markov decision process (MDP), the system and the MPC controller are lumped together to form the environment. This abstraction allows the decoupling of the discrete and continuous decision variables, which is a key component of the proposed approach. For a graphical representation of the reinforcement learning setting, see Fig. 2.1. Any reinforcement learning algorithm can be employed to train the agent. For simplicity, we use Deep Q-Learning (DQN) [96] to present our approach.

Herein, the Q-function is defined as the expected reward over a finite horizon when the agent takes action $\epsilon_\mathrm{d}(k)$ at the current time step k and then follows the policy $\epsilon_\mathrm{d}(k+l) = \pi(\epsilon_\mathrm{d} \mid \chi(k+l))$ over the remaining steps of the horizon:

$$\begin{aligned} Q^\pi(\chi,\, \epsilon_\mathrm{d}) = \mathbb{E}_\pi \Bigg[\sum_{l=0}^{N_\mathrm{p}-1} \alpha^l r(k+l) \;\Big|\; \chi(k) = \chi, \\ \epsilon_\mathrm{d}(k) = \epsilon_\mathrm{d},\, \epsilon_\mathrm{d}(k+l) = \pi(\epsilon_\mathrm{d} \mid \chi(k+l)),\; l \in \{1, \ldots, N_\mathrm{p}-1\} \Bigg], \end{aligned} \tag{2.8}$$

where α is the discount factor and the reward is defined as a function of the objective value of (2.7):

$$r(k) = f_\mathrm{reward}\big(V(x(k),\, \gamma(k),\, \epsilon_\mathrm{d}(k))\big),$$

where the function $f_\mathrm{reward}(\cdot)$ represents any transformation used in the reward design and it scales the output to the range $[0,\, 1]$, preventing the gradients from becoming large during training, which aids learning stability. If the action $\epsilon_\mathrm{d}(k)$ causes the LP (2.7) to be infeasible, then the reward is set to $r = -1$. Note that these values can be adjusted depending on the problem. To approximate the undiscounted control problem, i.e., to

appropriately consider long-term rewards, the discount factor γ is typically chosen close to the upper limit of the interval $[0, 1)$ [129].

In standard deep RL approaches, a global Q-function is employed to estimate the return for each discrete action. In a combinatorial action space, this often requires an intractable number of output units in the neural network representing the Q-function. For instance, consider a sequential decision-making problem, where the discrete state space has N_b binary actions per time step and the prediction horizon is N_p. In this case, the number of possible actions in the combinatorial action space would be $2^{N_\mathrm{b} \cdot N_\mathrm{p}}$, which can become prohibitively large even for modest values of N_b and N_p.

In this chapter, we split the action vector across time and a different Q-function is assigned to each time step of the prediction window, see Fig. 2.2. Consequently, the decision-making process for each sub-action is decoupled; therefore, we refer to the Q-functions of the sub-actions as decoupled Q-functions. The main benefit of using decoupled Q-functions is that it prevents the combinatorial growth, with respect to the prediction horizon, in the number of output units required by the neural network. This effectively allows RL to be applied to problems where the action space is combinatorial with respect to time. For the same example used for the standard deep RL approach, using decoupled Q-functions would require 2^{N_b} output units instead of $2^{N_\mathrm{b} \cdot N_\mathrm{p}}$.

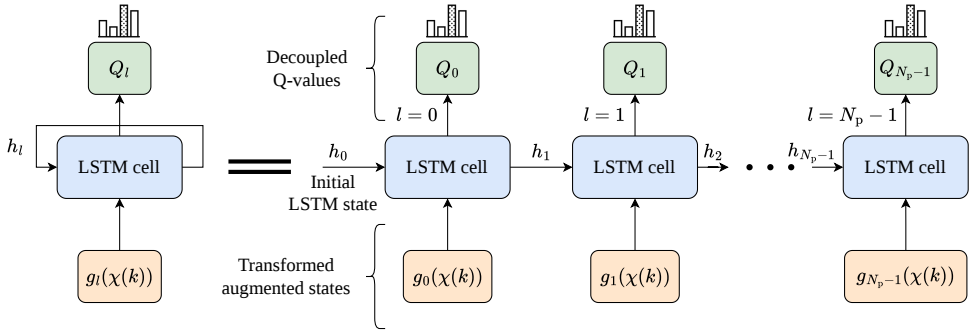


Figure 2.2: A representation of the recurrent LSTM network on the left-hand side and the unrolled LSTM network on the right-hand side. Note that the LSTM is unrolled for the duration of the prediction window N_p. Moreover, at each time step k, the augmented state $\chi(k)$ can go through the operation $g_k(\cdot)$, changing the manner in which the augmented state is presented to the LSTM network at time step k. This can be interpreted as a preprocessing technique to better exploit the structure of our problem.

The vector $\epsilon_\mathrm{d}(k)$ contains the entire sequence of discrete variables over the prediction horizon. For each time step of the prediction window $l = 0, \ldots, N_\mathrm{p} - 1$, we can represent the discrete sub-action per time step with $\varepsilon_{\mathrm{d},l}(k)$. Then $\epsilon_\mathrm{d}(k)$ can be rewritten as

$$\epsilon_\mathrm{d}(k) = \left[\varepsilon_{\mathrm{d},0}^\mathsf{T}(k),\ \varepsilon_{\mathrm{d},1}^\mathsf{T}(k),\ \ldots,\ \varepsilon_{\mathrm{d},N_\mathrm{p}-1}^\mathsf{T}(k) \right]^\mathsf{T} .$$

Instead of using RL to approximate the Q-function related to the joint action $\epsilon_\mathrm{d}(k)$, as defined in (2.8), our approach shifts the goal toward learning decoupled Q-functions $\{Q_l(\chi,\ \varepsilon_l)\}_{l=0}^{N_\mathrm{p}-1}$ associated with the sub-actions $\{\varepsilon_{\mathrm{d},l}(k)\}_{l=0}^{N_\mathrm{p}-1}$. Formally, the decoupled Q-function under a policy $\pi(\epsilon_\mathrm{d} \mid \chi(k))$ for the time step $l \in \{0, \ldots, N_\mathrm{p} - 1\}$ of the prediction

window is defined as

$$
\begin{aligned}
Q_l^{\pi}(\chi, \varepsilon) = \mathbb{E}_{\pi}\Bigg[\sum_{i=0}^{N_\mathrm{p}-1} \alpha^i r(i) \;\Big|\; & \chi(0) = \chi,\ \varepsilon_{\mathrm{d},l}(0) = \varepsilon, \\
& \varepsilon_{\mathrm{d},j}(0) = [\pi(\epsilon_\mathrm{d} \mid \chi(i))]_j \ \ \forall\, j \neq l, \\
& \varepsilon_{\mathrm{d},j}(i) = [\pi(\epsilon_\mathrm{d} \mid \chi(i))]_j, \ \ j \in \{1, \ldots, N_\mathrm{p}-1\}, \\
& \epsilon_\mathrm{c}(i) \text{ is the solution of (2.7)}, \\
& \epsilon(i) = [\epsilon_\mathrm{d}^\intercal(i),\ \epsilon_\mathrm{c}^\intercal(i)]^\intercal, \\
& r(i) = f_\mathrm{reward}(\chi(i),\ \epsilon(i)), \\
& \chi(i+1) = f_\mathrm{MDP}(\chi(i),\ \epsilon(i)) \\
& i \in \{1, \ldots, N_\mathrm{p}-1\} \Bigg],
\end{aligned}
\tag{2.9}
$$

where $[\pi(\epsilon_\mathrm{d} \mid \chi(i))]_j$ is the jth element of the discrete decision variables vector. More specifically, the decoupled Q-function is defined as the expected return of sub-action ε in state χ given that the first sub-action at index l is $\varepsilon_{d,l}(0) = \varepsilon$, the initial state is $\chi(0) = \chi$, the other sub-actions $j \neq l$ are chosen under policy $\varepsilon_{\mathrm{d},j}(0) = [\pi(\epsilon_\mathrm{d} \mid \chi(k))]_j$ for the first time step ($k = 0$) and that the system is guided by policy $\epsilon_\mathrm{d}(k) = \pi(\epsilon_\mathrm{d} \mid \chi(k))$ for $k \in \{1, \ldots, N_\mathrm{p}-1\}$. The MDP dynamics $\chi(k+1) = f_\mathrm{MDP}(\chi(k),\ \epsilon(k))$ applies the system dynamics of (2.1) on $x(k)$ and updates the exogenous signal $\gamma(k)$. The decoupled Q-functions can then be used to determine the policies for each of the sub-actions, e.g., the greedy policies are defined as

$$
\varepsilon_{\mathrm{d},l}(k) = \arg\max_{\varepsilon}\ Q_l^{\pi}(\chi(k),\ \varepsilon) \text{ for } l = 0,\ \ldots,\ N_\mathrm{p} - 1. \tag{2.10}
$$

The definition (2.9) is not tractable, but it can be useful for the design of a suitable approximator. The goal is to approximate each of the decoupled Q-functions $\{\tilde{Q}_l(\cdot)\}_{l=0}^{N_\mathrm{p}-1}$ over the prediction horizon. By definition, the decoupled Q-function of time step j depends on the sub-actions $l \in \{0, \ldots, N_\mathrm{p}-1\} \setminus \{j\}$ taken over the prediction horizon via the policy $\pi(\cdot)$. Herein, we assume that the past sub-actions are more relevant to the current sub-action than the future sub-actions. Therefore, to compute the decoupled Q-function of the time step j, only the sub-actions up to time step j of the prediction window $\{\varepsilon_{d,l}(k)\}_{l=0}^{j-1}$ are considered. We propose the use of recurrent neural networks (RNNs) to recursively compute each of the decoupled Q-values for its capacity of carrying over information from the past through an internal state. In this fashion, the decoupled Q-function receives information from the past sub-actions. Based on this assumption, an approximation of the decoupled Q-function with an RNN can be formulated as

$$
Q_l(\chi(k),\ \varepsilon_{\mathrm{d,l}}(k)) \approx \tilde{Q}_l(\chi(k),\ \varepsilon_{\mathrm{d,l}}(k),\ h_{l-1}(k)), \tag{2.11}
$$

where the hidden state $h_{l-1}(k)$ condenses information about past sub-actions. In particular, the long short-term memory (LSTM) network is employed for its capacity of capturing long-term dependencies in data and its structure, which attenuates the vanishing gradient problem [51]. Notably, the online evaluation of this approximator involves only the forward pass of the LSTM network, ensuring computational efficiency and tractability.

Other recurrent neural network architectures may be used, provided that they address the vanishing gradient problem and allow efficient online evaluation. For ease of notation, the dependence of the decoupled Q-functions on the hidden state h is omitted henceforth.

The approximation of the decoupled Q-functions is illustrated in Fig. 2.2, which depicts the forward pass of the unrolled LSTM network. The input of the LSTM network at time step l is the hidden state h_l, which encodes past information from previous steps of the prediction window, and the pre-processed augmented state $g_l(\chi(k))$. For instance, the function $g_l(\cdot)$ may extract information relevant only to time step l and apply normalization. The LSTM network outputs the vector of decoupled Q-values $\tilde{Q}_l$, which contains one value for each possible sub-action $\varepsilon_{\mathrm{d},l}$. Finally, during inference, the sub-actions can be selected in a decoupled manner by a greedy policy $\varepsilon_{\mathrm{d},l} = \arg\max_{\varepsilon} \tilde{Q}_l(\chi(k), \varepsilon)$ for $l = 0, \ldots, N_\mathrm{p} - 1$.

While each step of the prediction window has its own decoupled Q-function $Q_l(\cdot)$, the signal $h_{l-1}(k)$ retains the dependence of the previous sub-actions. In this way, some degree of interdependence between the sub-actions is modeled, and the learning problem is simplified due to the decoupling of the Q-functions. Moreover, the discrete and continuous decision variables are also not entirely decoupled since the training of the approximator is based on the augmented state and reward signals, which depend on all the decision variables. Therefore, in both cases, decoupling aids tractability while maintaining some degree of interconnection, which is arguably desirable in our setting.

With the approximator described, the training procedure using Deep Q-Learning [96] is discussed. Each episode of the training procedure starts with a randomly chosen initial state $\chi(k)$. Then, the RL agent repeatedly proposes a discrete action $\epsilon_\mathrm{d}(k)$, sends it to the environment, where the continuous decision variables $\epsilon_\mathrm{c}(k)$ are determined by MPC (2.7), and collects the reward $r(k)$ and the next state $\chi(k+1)$. An episode ends when a pre-determined time limit is reached or when the RL agent suggests a discrete action that makes the optimization problem (2.7) infeasible. At each time step, the transition $(\chi(k), \epsilon_\mathrm{d}(k), r(k), \chi(k+1))$ is stored in a data buffer $\mathscr{D}$, which only holds a finite number of the most recent transitions. After the end of each episode, N_batch transitions are randomly sampled from $\mathscr{D}$ and the weights of the LSTM network are updated by gradient descent on the following loss function:

$$L = \frac{1}{N_\mathrm{batch}} \sum_{k=1}^{N_\mathrm{batch}} \sum_{l=0}^{N_\mathrm{p}-1} \left(y_l(k) - \tilde{Q}_l(\chi(k), \varepsilon_{\mathrm{d},l}(k)) \right)^2,$$

where the temporal difference (TD) target is defined as

$$y_l(k) = r(k) + \alpha \max_{\varepsilon} \tilde{Q}_l\left(\chi(k+1), \varepsilon\right).$$

Some alternative definitions of the TD target can be found in [132]. The training of the RL agent is deemed complete when a predetermined number of episodes or number of transitions is reached.

Once training is completed, the RL agent is employed to suggest the discrete action $\epsilon_\mathrm{d}(k)$ and, consequently, the MPC optimization problem (2.2) is simplified to the LP described in (2.7). As a result, the solution time of the MILP is replaced by the time of inference of an LSTM network and the solution time of an LP, which can be efficiently solved by modern solvers, greatly reducing the computation time.

The offline training and the online inference are described in Algorithms 1 and 2, respectively.

Algorithm 1 Offline Training

Initialize recurrent neural network that estimates the decoupled Q-function $\tilde{Q}(\chi,\, \epsilon;\, \theta)$ with random weights
Initialize databuffer $\mathscr{D} = \{\}$
for episode = $1, ..., N_{\text{episodes}}$ **do**
  Set random state $\chi(0)$
  for $k = 1, ..., N_{\text{steps}}$ **do**
    for l from 0 to $N_{\text{p}} - 1$ **do**
      $\epsilon_{\text{d},l}(k) \sim \text{softmax}\, \tilde{Q}_l(\chi(k),\, \epsilon_{\text{d},l};\, \theta)$
    end for
    Set $\epsilon_{\text{d}}(k) = [\epsilon_{\text{d},0}^{\intercal}(k),\, \epsilon_{\text{d},1}^{\intercal}(k),\, \ldots,\, \epsilon_{\text{d},N_{\text{p}}-1}^{\intercal}(k)]^{\intercal}$
    Get $\epsilon_{\text{c}}(k)$ by solving the linear program (2.7)
    Set $\epsilon(k) = [\epsilon_{\text{d}}^{\intercal}(k),\, \epsilon_{\text{c}}^{\intercal}(k)]^{\intercal}$
    Get reward $r(k) = f_{\text{reward}}(\chi(k),\, \epsilon(k))$
    Get next state $\chi(k+1) = f_{\text{MDP}}(\chi(k),\, \epsilon(k))$
    Store $(\chi(k), \epsilon_{\text{d}}(k), r(k), \chi(k+1))$ in $\mathscr{D}$
    Sample a minibatch of transitions $\left\{\left(\chi^j, \epsilon_{\text{d}}^j, r^j, \chi^{j+1}\right)\right\}_{j=1}^{N_{\text{batch}}}$ from $\mathscr{D}$
    Set $y_l^j = r^j$ for terminal χ^{j+1} or $y_l^j = r_l^j + \alpha \max_{\epsilon} Q_l\left(\chi^{j+1}, \epsilon;\, \theta\right)$ for non-terminal χ^{j+1} for $l = 0, ..., N_{\text{p}} - 1$ and $j = 1, ..., N_{\text{batch}}$
    Perform a gradient descent step on $\sum_{j=1}^{N_{\text{batch}}} \sum_{l=0}^{N_{\text{p}}-1} \left(y_l^j - \tilde{Q}_l(\chi^j, \epsilon_{\text{d},l}^j;\, \theta)\right)^2$ to update the network weights θ
  end for
end for

2.4. ILLUSTRATIVE EXAMPLE: MICROGRID CONTROL

The use of mixed-logical dynamical systems for modeling and control of microgrids has been considered in [105], [107], [93], and [92]. In these works, an MLD system is employed to cast a mixed-integer linear problem for microgrid operation optimization. A similar model is used in the illustrative example of this chapter to solve the unit commitment problem, i.e., planning the schedule of the generation units, and the economic dispatch problem, i.e., controlling the power flow to fulfill the power demand of the loads while minimizing the operation cost. The system is described in Appendix 2.A and depicted in Fig. 2.3.

The knowledge of the market energy prices and the forecasts of the power demand power generation from renewable energy sources – solar and wind energy are considered – allows the operator to plan and optimally schedule the operation of the microgrid over a time horizon. To accomplish this goal, the microgrid operator has to determine a set of discrete and continuous actions over a finite prediction horizon. The discrete control actions are the following: buying or selling from grid $\delta_{\text{grid}}(k)$, charging or discharging

Algorithm 2 Online Inference

1: Measure state $\chi(k)$
2: **for** l from 0 to $N_\mathrm{p}-1$ **do** ▷ *Compute sub-actions*
3: $\quad \varepsilon_{\mathrm{d},l}(k) = \arg\max_{\varepsilon}\ \tilde{Q}_l(\chi(k),\, \varepsilon)$
4: **end for**
5: Set $= \epsilon_\mathrm{d}(k) = \left[\varepsilon_{\mathrm{d},0}^\intercal(k),\, \varepsilon_{\mathrm{d},1}^\intercal(k),\, \ldots,\, \varepsilon_{\mathrm{d},N_\mathrm{p}-1}^\intercal(k)\right]^\intercal$
6: Get $\epsilon_\mathrm{c}(k)$ by solving the linear program (2.7)
7: **return** $\epsilon_\mathrm{d}(k), \epsilon_\mathrm{c}(k)$

the energy storage system $\delta_\mathrm{b}(k)$ and turning on or turning off the generators $\{\delta_i^\mathrm{dis}(k)\}_{i=1}^{N_\mathrm{gen}}$ at time step k. The continuous actions are the power exchange with the main grid $P_\mathrm{grid}(k)$, power exchange with the energy storage system $P_\mathrm{b}(k)$, and the power provided by the dispatchable generators $P_i^\mathrm{dis}(k)$. For a summary of the decision variables, see Table 2.1. In this setting, the decision variables are represented by the vectors $\epsilon_\mathrm{c}(k)$ and $\epsilon_\mathrm{d}(k)$, which are formed by the concatenation of all discrete and continuous decision variables, respectively, over the prediction horizon. Furthermore, the exogenous signal $\gamma(k)$ represents the forecasts over the prediction horizon for the: purchasing, sale, and production prices; power generation from the renewable energy sources; and power demand from the loads.

Variable	Description
δ_b	status of the storage unit: charging(1)/discharging(0)
δ_grid	mode of connection with main grid: importing(1)/exporting(0)
δ_i^dis	state of dispatchable unit i: on(1)/off(0)
P_b	power exchanged with the storage unit [kW]
P_grid	power exchanged with the main grid [kW]
P_i^dis	power generated by dispatchable unit i [kW]

Table 2.1: Decision variables and their descriptions.

We consider the receding horizon control strategy laid out in Section 2.2. Consequently, the control problem can be formulated as the MILP of (2.6). Efficiently solving such an optimization problem is at the core of the problem. In this context, we compare our RL-based approach against standard MPC for MLD systems, where the optimization problem (2.2) is solved to optimality with branch-and-bound, and against the supervised learning (SL) proposed in [29] and briefly described in the last section and in Appendix 2.B.

2.4.1. SETUP

The renewable energy generation and load profiles were taken from the actual operation of the Dutch main grid, which is publicly available at [46]. Since the microgrid considered in this illustrative example does not have the capacity to meet the demand of the entire

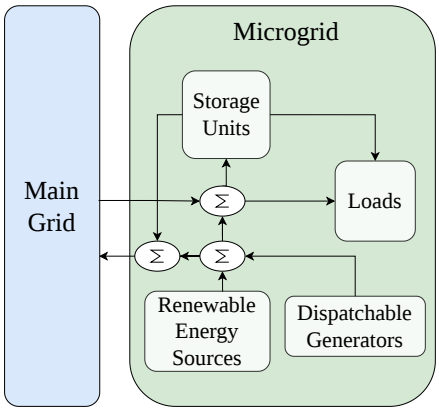


Figure 2.3: Depiction of the elements of a microgrid and a bidirectional connection with the main grid. The pointed arrows indicate the possibility of power flow between two elements.

grid, the real power profiles were linearly downscaled to adjust the values to our illustrative example. This is the only preprocessing operation applied to these signals, so the shapes of the renewable energy generation and load profiles remain unchanged. The RL-based and SL-based algorithms are trained on one year of data (2022) and their performances are evaluated in a different year (2021) to assess the capacity of the approaches to generalize and adapt to new data.

The price profiles – c_{prod}, c_{buy}, c_{sell} – are not publicly available, and, consequently, they were synthetically generated by three different normal distributions that obey a basic principle: on average, producing energy is cheaper than purchasing energy from the main grid and higher than selling energy to the main grid – examples of the cost profiles can be seen in Fig. 2.10 in Appendix 2.A. Due to the randomness of the prices, which mimic real market conditions, the price order can change, and the microgrid operator can take advantage of some occasions. For instance, when the production price is smaller than the sell price, it is profitable to generate power that exceeds the load demand sell the power surplus to the main grid. The dispatchable units are assumed to be identical, and, consequently, the same production price is used for all of them.

Furthermore, the microgrid model, which is described in Appendix 2.A, is used for the training of the supervised learning and reinforcement learning agents, for the MPC controller, and for the system simulation. This ensures that performance differences arise solely from the algorithms. For the RL-based approach, Boltzmann exploration is used during training so that exploration occurs according to the decoupled Q-values. In this type of exploration, the probability of a given sub-action being chosen is proportional to the softmax of its decoupled Q-value. The policy used for exploration is described below

$$\pi^{\text{RL}}(\varepsilon_{\text{d},l,k}(k) \mid \chi(k)) \propto e^{\xi \cdot Q_l(\chi(k),\, \varepsilon_{\text{d},l,k}(k))},$$

where $\pi^{\text{RL}}(\varepsilon_{\text{d},l,k}(k) \mid \chi(k))$ represents the probability of selection of the sub-action $\varepsilon_{\text{d},l,k}(k)$ in state $\chi(k)$, and ξ is the exploration temperature. At the beginning of training, the temperature is chosen to $\xi = 0$ so that the probabilities of selecting the sub-actions are

equal, which is equivalent to random exploration. As training progresses, the temperature is gradually raised to incentivize greedier behavior.

To ensure fair comparison, we use the general-purpose solver GUROBI [59] for the corresponding LPs and MILPs. The dimension of the most relevant variables per time step is reported in Table 2.2. Note that the values should be scaled by the prediction horizon N_p.

$\dim(\chi)$	$m_c + r_c$	$m_d + r_d$	n_{cons}
6	7	5	36

Table 2.2: The MILP information per time step: $\dim(\chi)$ is the dimensionality of the augmented state, $m_c + r_c$ represents the number of continuous variables, $m_d + r_d$ indicates the number of discrete variables, and n_{cons} denotes the number of constraints.

The proposed method does not guarantee feasibility. In the illustrative example, feasibility can be restored by a simple heuristic in which power is produced by the local generators as much as possible to match local consumer demand. If power demand exceeds local power generation, the power deficit is bought from the main grid.

For the SL and RL approaches, the neural network architecture consists of a single layer of an LSTM network, whose outputs are connected to a fully connected layer. The approaches are then trained with several sizes of the hidden state of the LSTM network in the set $\{64, 128, 256, 512\}$. While larger hidden sizes offer greater representational capacity, smaller sizes enable faster evaluation and reduce the risk of overfitting. The output of the LSTM network, i.e., the number of possible actions per time step, is $2^{(m_d+r_d)} = 32$. To simplify the presentation of the results, we only report the best-performing trained neural networks after hyperparameter tuning.

The methods are simulated with four prediction horizons $N_p \in \{4, 12, 24, 48\}$, corresponding to look-ahead periods of 2, 6, 12, and 24 hours, respectively, given the sampling time of $T_s = 30$ minutes. The aim of these experiments is to assess how the effectiveness of the methods scales with the increase of the number of binary variables. The initial state is sampled from an uniform distribution defined over its lower and upper operation limits. The initial exogenous state is retrieved from the dataset corresponding to a randomly selected day and time of the year 2022 and 2021 for training and evaluation, respectively. The code and data used in this chapter are publicly available[1].

2.4.2. SIMULATION RESULTS

The open-loop simulation results for the methods and different prediction horizons are shown in Table 2.3 and Figs. 2.4, 2.5, and 2.6. As performance metrics, we analyze the optimality gap, infeasibility rate, and computation time. The optimality gap is the relative distance to the optimal solution computed by branch-and-bound, i.e.

$$\text{optimality gap} = \frac{J_{\text{learning}} - J_{\text{optimal}}}{J_{\text{optimal}}} \cdot 100,$$

[1] https://github.com/fabcaio/microgrid_RL

where J_{learning} is the cost of the linear program (2.7) when a learning method – either RL or SL – is used to determine the discrete variables and J_{optimal} is the cost of the mixed-integer program (2.6) when it is solved to optimality with branch-and-bound. The infeasibility rate is an empirical probability of the learning methods providing discrete sequences $\epsilon_{\text{d}}(k)$ that result in an infeasible linear program (2.7). Computation time is also relative to the time necessary for the branch-and-bound solver to reach optimality, i.e.,

$$\text{reduction factor} = \frac{T^{\max}_{\text{optimal}}}{T^{\max}_{\text{i}}} \quad \text{for} \quad i \in \{\text{RL, SL}\},$$

where $T^{\max}_{\text{i}}$ is the maximum of the inference time of the LSTM added to the solution time of the linear program (2.7) and $T^{\max}_{\text{optimal}}$ is the maximum solution time of the mixed-integer program (2.6).

The simulation experiments, conducted over 3000 randomly selected initial states and summarized in Table 2.3 demonstrate the effectiveness of learning-based approaches in significantly reducing the online computation time of MPC. In control applications, often the maximum computation is a more important criterion than the average computation time for hardware specification. As shown in in Fig. 2.6, which reports the reduction in the maximum computation time, the learning-based approaches are 8 to 20 faster than the branch-and-bound solver in the worst-case.

Prediction horizon	Method	Optimality gap (%)	Infeasibility rate (1/1000)	Computation time mean	max	std
$N_{\text{p}} = 4$	RL	0.32	0.67	1.2	2.1	0.1
	SL	0.15	64.3	1.3	2.3	0.1
	Optimal	0.00	0.0	3.4	19.7	0.7
$N_{\text{p}} = 12$	RL	0.39	4.0	2.0	3.6	0.2
	SL	0.20	36.67	2.1	3.2	0.2
	Optimal	0.00	0.0	8.0	46.5	2.3
$N_{\text{p}} = 24$	RL	0.74	17.7	4.8	9.4	0.4
	SL	0.43	91.1	4.3	10.8	0.4
	Optimal	0.00	0.0	23.6	84.8	3.3
$N_{\text{p}} = 48$	RL	0.98	25.7	11.0	13.5	0.5
	SL	0.55	137.0	10.2	15.9	0.6
	Optimal	0.00	0.0	118.3	272.8	9.4

Table 2.3: Simulation results for the reinforcement learning (RL), supervised learning (SL), and branch-and-bound (Optimal) approaches for solving the mixed-integer linear program (2.6). The optimality gap is the relative distance between the objectives of the linear program (2.7) using the learning approaches and the objective of the mixed-integer program (2.6) solved to optimality with branch-and-bound. The infeasibility rate is the frequency at which the learning approaches propose discrete actions that cause linear program (2.7) to be infeasible. The mean, maximum, and standard deviation of the computation time are reported.

The reduction in computation time comes at the price of a small optimality gap – under 1% – and some instances of infeasibility, represented in Fig. 2.5. The reported values for the optimality gap account for the use of the heuristic rule to guarantee feasibility.

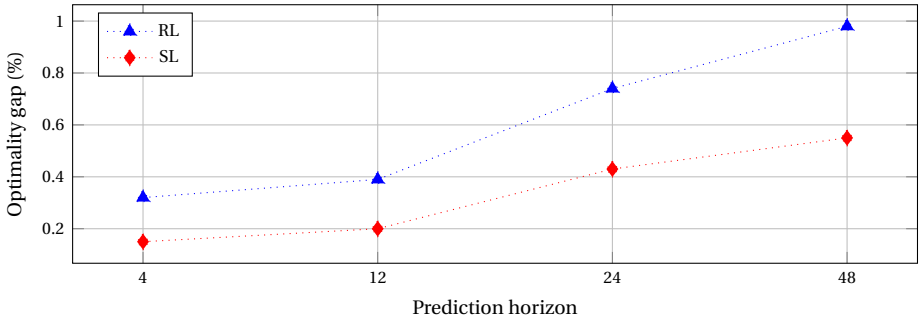


Figure 2.4: Optimality gap for different prediction horizons for the reinforcement learning (RL) and supervised learning (SL) approaches in the illustrative example.

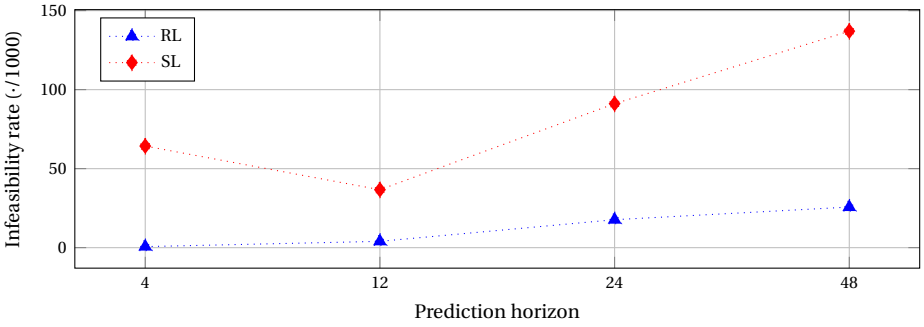


Figure 2.5: Infeasibility rate for different prediction horizons for the reinforcement learning (RL) and supervised learning (SL) approaches in the illustrative example.

The proposed approach consistently has significantly lower infeasibility rates than that of the supervised learning method across all the considered prediction horizons. The difference is more notable for $N_p = 48$, where the infeasibility rate for the proposed approach is 25.7/1000 and that of the SL approach is 137.0/1000. However, the supervised learning method has a slightly lower optimality gap than that of the proposed approach. This reveals a trade-off between both learning approaches in our illustrative example: while supervised learning favors optimality, reinforcement learning outperforms in terms of feasibility. Further investigation is required to determine whether this observation extends to other problem domains.

Concerning the scalability for an increasing prediction horizon, as seen in Fig. 2.4, the optimality of the proposed approach remains below 0.98% and that of the SL-based approach remains under 0.55%, showing only small losses in optimality. The analysis of the infeasibility rate in Fig. 2.5 reveals that the RL approach scales considerably better than the SL method regarding the infeasibility rate. The contrast is especially evident at $N_p = 48$, where the infeasibility rate of the proposed approach is 5.33 times smaller. As shown in Fig. 2.6, the reduction in computation time is consistent across the prediction horizons, achieving its peak at $N_p = 48$ where the maximum computation time is reduced

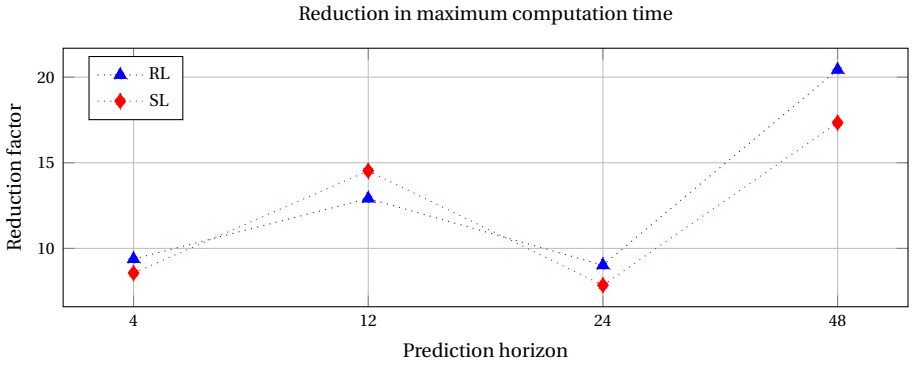


Figure 2.6: Reduction factor of the maximum computation time $\frac{T^{\max}_{\text{optimal}}}{T^{\max}_{i}}$, $i \in \{\text{RL, SL}\}$, for different prediction horizons for the reinforcement learning (RL) and supervised learning (SL) approaches in the illustrative example.

by a factor of 20. Finally, a box plot for the computation time for all approaches is shown in Fig. 2.7.

To assess the performance of the approaches in the presence of model mismatch, errors are intentionally introduced in the charging and discharging efficiencies of the storage unit η_c and η_d, respectively. While the learning-based approaches are trained with $(\eta_c, \eta_d) = (0.9, 0.99)$, the parameters of the real system are set to $(\eta_c, \eta_d) = (0.86, 0.95)$. Furthermore, uncertainty is also introduced into the forecasts of power demand and power generation by renewable sources. Only the local generation cost and day-ahead prices for buying and selling from the main grid are assumed to be known in advance. The heurisic rule is used to maintain feasibility during online operation.

The results are obtained for closed-loop simulations during 3 days from 100 different randomly selected initial states chosen in the same manner as the previous experiments. The prediction horizon is set to $N_p = 48$. For this setting, the average optimality gaps of the RL and SL approaches are 1.07% and 0.75%, respectively. The infeasibility rates for the RL and SL approaches are 8.7/1000 and 100.6/1000, respectively. This shows that both methods can generalize well and can handle some degree of uncertainty. The difference in infeasibility rate between the learning-based approaches is even more pronounced during closed-loop operation, meaning that the SL-based method relies considerably more on the safe controller. This may further negatively impact performance in applications where the safe controller is highly suboptimal.

A closed-loop simulation of the power exchanges during online operation of the microgrid is shown in Fig. 2.9, where both the behavior of the optimal controller and of the RL-based approach are displayed. Regarding performance, the optimality gap between the optimal solution (MILP) and the RL-based approach is 1.09%. The corresponding forecasts for the power demand and the power generation by renewable sources is represented in in Fig. 2.8. The results suggest that the RL policy tends to underutilize the storage unit, which may explain part of the optimality gap. The associated costs for producing, selling, and buying energy are shown in Fig. 2.10 in Appendix 2.A. For brevity, discrete variables are omitted, as they can be directly inferred from the continuous signals,

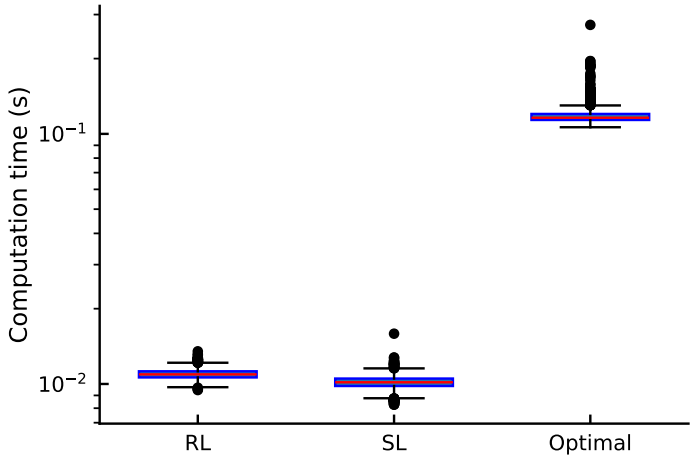


Figure 2.7: A box plot with logarithmic y-axis containing the computation time for the RL-based, SL-based, and MILP approaches. The median is highlighted in red. The blue box, extends from the first quartile to the fourth quartile, meaning that 50% of the data lies in this region. The black points represent outliers. The black vertical lines represent the range of data, excluding outliers.

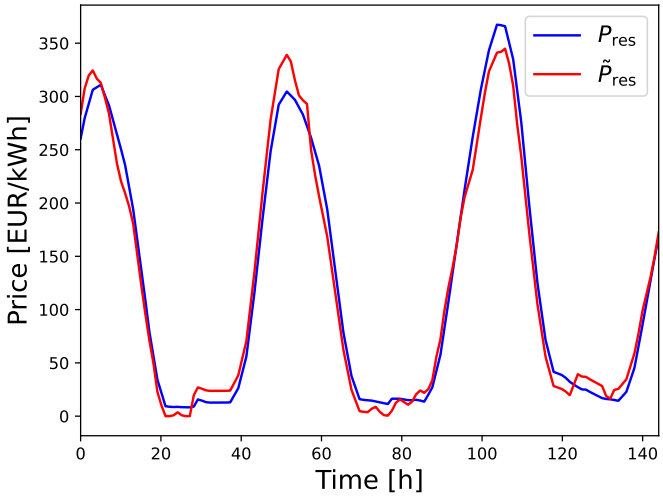
e.g., $\delta_{\mathrm{b}}(k) = 1 \iff P_{\mathrm{b}}(k) \geq 0$ and $\delta_{\mathrm{grid}}(k) = 1 \iff P_{\mathrm{grid}}(k) \geq 0$.

2.5. DISCUSSION

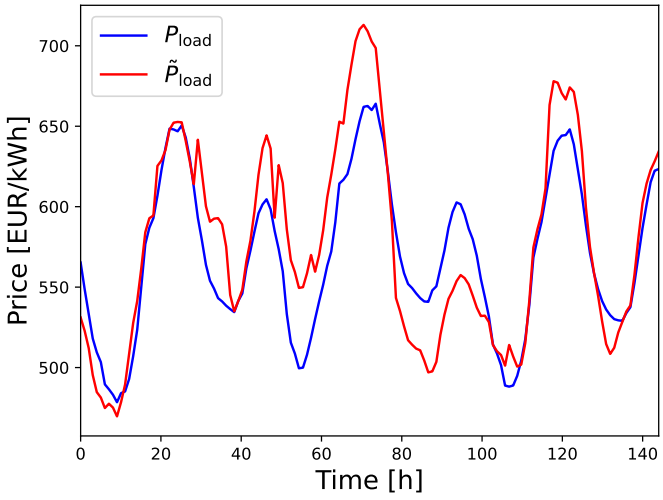
The experiments presented in the illustrative example in the previous section demonstrate the potential of the proposed framework for efficiently controlling MLD systems, achieving a favorable balance between computation time, optimality, and feasibility. This section brings to discussion several important aspects of the proposed approach.

Even though discrete actions $\epsilon_{\mathrm{d}}(k)$ that lead to the infeasibility of the LP (2.7) are discouraged in the reward function, the agent may still select infeasible actions. Since feasibility is not guaranteed by the proposed approach, it is particularly well-suited for application domains in which a safe backup controller can be designed based on expert knowledge. When that is not possible, the mixed-integer linear program (2.6) can be used to compute a feasible solution within the given computational constraints. Alternatively, the proposed method can be used to warm-start the MILP solver and reduce its average computation time. Lastly, one can consider learning-based methods that produce, by design, feasible discrete decision variables to warm-start the MILP solver [90], [150].

The sampling time of the illustrative example is relatively large compared to the average solution time of the MILP. However, the proposed approach offers advantages beyond the average reduction in computation time. In control applications, it is crucial to ensure that the maximum solution time is predictable and remains within a reasonable bound, i.e., a solution must be computed before the next time step. For mixed-integer linear program solvers, the worst-case complexity is typically large in comparison to the average solution time. In contrast, the proposed approach only requires evaluating a recurrent

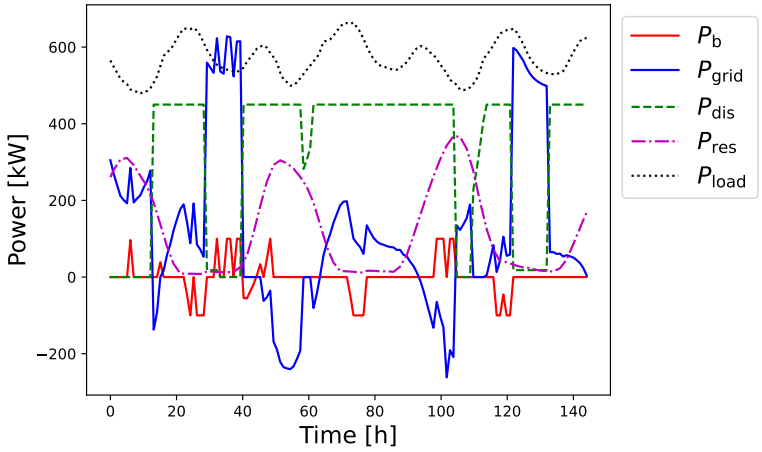


(a)

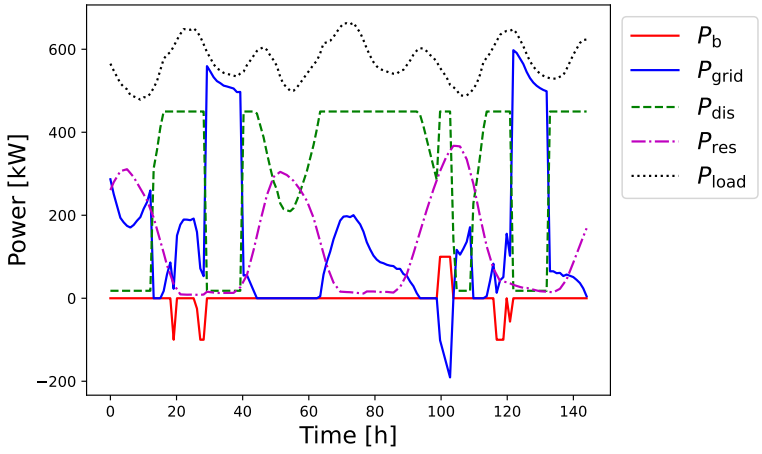


(b)

Figure 2.8: This figure provides examples of (a) the true generation of renewable energy sources P_{res} and the predicted generation of renewable energy sources $\tilde{P}_{\text{res}}$, and (b) the true power demand P_{load} and the predicted power demand $\tilde{P}_{\text{load}}$. The time window corresponds to 144 hours (3 days) randomly selected in the year 2021.



(a) Optimal solution (MILP).



(b) RL-based approach solution.

Figure 2.9: The figure represents the closed-loop operation of the microgrid for the optimal policy (top) and for the proposed approach (bottom) during a time window of 144 hours (3 days) randomly selected from the testing dataset. The following continuous signals are shown: the power exchanged with the storage unit P_b, the power exchanged between the microgrid and the main grid P_{grid}, the total power generated by the microgrid P_{dis}, the power generated by renewable energy sources P_{res}, and the power demand of the microgrid P_{load}.

neural network and solving a linear program, both of which exhibit predictable computational behavior. Hence, it is relatively easier to estimate a reasonable worst-case bound on the computation time needed to determine the control input for the proposed approach. Due to the relatively smaller worst-case bound, the proposed approach requires the allocation of a shorter time window for computing the control signal. As a consequence, the proposed approach can utilize more recent and accurate measurements of the system and exogenous signal, potentially improving the solution quality. Finally, for larger systems with more decision variables and possibly nonlinear objectives, dynamics or constraints, the disparity in the worst-case computation times is expected to increase, as the complexity of the optimization problem grows at each node of the branch-and-bound tree.

The proposed approach is agnostic to the choice of deep RL algorithm. In this chapter, Deep Q-Learning (DQN) was selected for its simplicity. Nonetheless, numerous algorithms, such as Rainbow DQN [63] and PPO [119], have shown better performance than that of DQN across several RL benchmarks. Therefore, it is likely that both optimality and feasibility rates could be further improved by employing more advanced RL algorithms.

Model mismatch and uncertainties can be explicitly handled in the proposed approach by using robust or stochastic MPC formulations. On the learning side, techniques such as domain randomization [133] and dataset augmentation [54] can also be applied to enhance robustness.

The benefits of the learning-based methods are evident during online operation. However, training these methods requires accurate modeling and offline computational resources. For supervised learning, a dataset with the optimal solution of the control problem must first be obtained, after which the classifier weights are trained. In contrast, for reinforcement learning, the agent repeatedly interacts with a simulator of the system to obtain data to adjust the decoupled Q-function approximator – possibly also a policy approximator – via temporal-difference learning. Fundamentally, for training, the supervised learning approach requires the solution of mixed-integer programs, while its reinforcement learning counterpart involves the optimization of continuous decision variables. As a result, the computation cost in the reinforcement learning approach is smaller per iteration; however, it requires more iterations due to exploration. Moreover, the determination of the best approach regarding offline computation load depends heavily on the type of available computational resources (e.g., CPU, GPU, and RAM) as well as the specific implementation of the data acquisition and training algorithms, particularly the extent to which the algorithm is designed to exploit existing hardware. Given these factors, the comparison of the computation load required by the different methods is complex and beyond the scope of this chapter.

2.6. CONCLUSIONS

This chapter presented a novel approach that combines reinforcement learning and model predictive control for mixed-logical dynamical systems. The online computation time of the MPC controller can be significantly reduced by fixing the discrete decision variables with a policy trained by RL and then by optimizing over the continuous decision variables. This procedure effectively simplifies a mixed-integer linear program into a linear program.

For the training of the RL policy, we conceived the definition of the decoupled Q-function to decouple the discrete decision variables over the prediction horizon. Moreover, we showed how LSTM networks can be trained to approximate such a function.

Simulation experiments in a microgrid system revealed that our RL-based approach achieves a favorable trade-off between optimality, feasibility, and computation time. This suggests that the decoupled Q-function effectively captured the return of the sub-actions. Moreover, the computation time of the proposed approach was up to 20 times faster than the standard MPC, whose solver was based on branch-and-bound.

Several potential research directions for enhancing the performance of the proposed approach were discussed in the previous section. Additionally, the framework can be readily extended to control problems involving mixed-integer nonlinear programs, enabling its application to even more complex systems.

Future research will also investigate the design of hybrid schemes where the RL and SL approaches are employed concurrently. For instance, the discrete action provided by the RL and SL approaches could be used sequentially, e.g., using SL when RL is not feasible, or in parallel, by evaluating both actions and selecting the one that results in the lowest cost. Alternatively, a policy trained by SL can be used to initialize the training of the RL agent.

APPENDIX

2.A System description

In the illustrative example, we address the economic dispatch problem in a microgrid system. This appendix describes the constituent elements of the microgrid, shown in Fig. 2.3, and their modeling as a mixed-logical dynamical (MLD) system. This modeling framework is used for its capacity to capture the behavior of continuous and discrete dynamics and decision variables. Besides, MLD modeling paves the way for the formulation of the economic dispatch problem as a mixed-integer program using an MPC approach. The use of MLD modeling and MPC for the microgrid operation optimization was explored in [105], where the problem is cast as a mixed-integer linear program. The modeling of the microgrid was further simplified in [107] and [92] for the design of a ruled-based control policy and a learning-based control rule, respectively. Herein, we use the same simplified microgrid modeling framework to assess the performance of the proposed approach.

2.A.1 Storage unit

The energy storage unit is described by the following equations:

$$x_{\mathrm{b}}(k+1) = \begin{cases} x_{\mathrm{b}}(k) + \dfrac{T_{\mathrm{s}}}{\eta_{\mathrm{d}}} P_{\mathrm{b}}(k) & \text{if } P_{\mathrm{b}}(k) < 0 \\ x_{\mathrm{b}}(k) + T_{\mathrm{s}} \eta_{\mathrm{c}} P_{\mathrm{b}}(k) & \text{if } P_{\mathrm{b}}(k) \geq 0 \end{cases}, \tag{2.12}$$

where $x_{\mathrm{b}}(k)$ is the energy level in the storage unit at time step k, η_{c} and η_{d} are the charging and discharging efficiencies, $P_{\mathrm{b}}(k)$ is the power exchanged with the storage unit at time step k and T_{s} is the sampling time of the discrete-time system. At a given time step, the storage unit can be either discharging or charging, depending on the sign of $P_{\mathrm{b}}(k)$. In order

to capture this hybrid behavior, we model the storage unit as a mixed-logical dynamical (MLD) system [17]. A binary variable $\delta_{\mathrm{b}}(k)$ is introduced to signal whether the storage unit is charging or discharging, $\delta_b(k) = 1 \iff P_{\mathrm{b}}(k) \geq 0$ and $\delta_b(k) = 0 \iff P_{\mathrm{b}}(k) < 0$, respectively. By defining a continuous auxiliary variable $z_{\mathrm{b}}(k) = \delta_{\mathrm{b}}(k)P_{\mathrm{b}}(k)$, (2.12) can be simplified and conveniently written as a single linear equation:

$$x_{\mathrm{b}}(k+1) = x_{\mathrm{b}}(k) + T_{\mathrm{s}}(\eta_{\mathrm{c}} - \frac{1}{\eta_{\mathrm{d}}})z_{\mathrm{b}}(k) + \frac{T_{\mathrm{s}}}{\eta_{\mathrm{d}}}P_{\mathrm{b}}(k).$$

2.A.2 Generation units

In the microgrid, energy can be locally produced either by dispatchable units or by renewable energy sources. The dispatchable units can be turned on/off, and their power output levels can be arbitrarily chosen by the microgrid operator within operating constraints. Concerning the renewable energy sources, solar and wind energy sources are considered.

The cost for locally producing energy at time step k is

$$C_{\mathrm{prod}}(k) = c_{\mathrm{prod}}(k) \sum_{i=1}^{N_{\mathrm{gen}}} P_i^{\mathrm{dis}}(k), \tag{2.13}$$

where $c_{\mathrm{prod}}(k)$ is the cost of producing energy at time step k, $P_i^{\mathrm{dis}}(k)$ represents the power generated by dispatchable unit i at time step k, and N_{gen} is the total number of dispatchable units. Besides, a binary variable $\delta_i^{\mathrm{dis}}(k)$ is introduced to represent whether the generator i is turned on (1) or off (0). This will be useful in the next section when the operating constraints are introduced and the optimization problem is formulated.

Renewable energy sources are excluded from the expression (2.13) because we assume zero cost for utilizing them when available. Moreover, due to their nature, the power generated by renewable sources is not controllable. The total power generated by renewable sources at time step k is represented by $P_{\mathrm{res}}(k)$.

2.A.3 Main grid

At any given time step, the microgrid can buy or sell energy from the main grid. The power exchange at time step k is represented by $P_{\mathrm{grid}}(\mathrm{k})$. If this variable is nonnegative, the microgrid is set to import energy from the main grid. If it is negative, the microgrid is in export mode. Then, from the microgrid operator's perspective, at time step k, the operation cost is represented as

$$C_{\mathrm{grid}}(k) = \begin{cases} c_{\mathrm{sell}}(k)P_{\mathrm{grid}}(k) & \iff P_{\mathrm{grid}}(k) < 0 \\ c_{\mathrm{buy}}(k)P_{\mathrm{grid}}(k) & \iff P_{\mathrm{grid}}(k) \geq 0 \end{cases}, \tag{2.14}$$

where $c_{\mathrm{sell}}(k)$ and $c_{\mathrm{buy}}(k)$ are the prices for selling and buying energy to/from the main grid, respectively, at time step k.

Consider the discrete auxiliary variable $[\delta_{\mathrm{grid}}(k) = 1] \iff [P_{\mathrm{grid}}(k) \geq 0]$ and the continuous auxiliary variable $z_{\mathrm{grid}}(k) = \delta_{\mathrm{grid}}(k)P_{\mathrm{grid}}(k)$. Now, with the use of MLD modeling, the operation cost between the microgrid and the main grid can be expressed in a single linear equation

$$C_{\mathrm{grid}}(k) = c_{\mathrm{buy}}(k)z_{\mathrm{grid}}(k) - c_{\mathrm{sell}}(k)z_{\mathrm{grid}}(k) + c_{\mathrm{sell}}(k)P_{\mathrm{grid}}(k). \tag{2.15}$$

Note that the operation cost is negative in export mode, i.e., the microgrid operator profits by selling energy to the grid.

2.A.4 Assumptions

We only consider uncontrollable loads, i.e., the microgrid operator does not affect the power demanded by them. Furthermore, at the current operation time, the actual load and its forecast over the future are assumed to be known. Similarly, the current power generated by renewable energy sources and its forecast are assumed to be known. These are not strong assumptions, since all of these values can be estimated with the use of historical data. We also assume knowledge of the market energy prices for buying and selling energy from the main grid and for locally producing energy with the dispatchable generators.

2.A.5 Control scheme

We consider a hierarchical control structure. A high-level controller is concerned with planning the operation schedule: i) which generation units are turned on and their power outputs; ii) whether the storage unit is charging or discharging and the corresponding power exchange; and iii) the mode of operation of the microgrid with respect to the main grid – importing or exporting – and the amount of power flowing between them. The variables determined by the high-level controller are reported in Table 2.1. A low-level controller is responsible for keeping voltage, frequency, and phase within the operation range, and it operates in a faster timescale. In this scheme, the high-level controller is responsible for generating the set points for the microgrid elements so that the load demand is satisfied, and the operation cost is minimized. On the other hand, the low-level controller has the objective of tracking these set points computed by the high-level controller. In this chapter, we assume that the low-level controller is already in place, and we focus exclusively on the design of the high-level controller.

2.A.6 Control problem

The operation cost is defined as the sum of the costs for locally producing energy and the costs for exchanging power with the main grid over a prediction horizon N_p. Accordingly, the cost is defined as follows:

$$J(\mathbf{x}_\mathrm{b}(k), \gamma(k), \epsilon_\mathrm{c}(k), \epsilon_\mathrm{d}(k)) = \sum_{l=0}^{N_\mathrm{p}-1} \left(C_\mathrm{prod}(k+l) + C_\mathrm{grid}(k+l) \right) = c^\mathsf{T} \epsilon_c(k), \tag{2.16}$$

where $\mathbf{x}_\mathrm{b}(k) = [x_\mathrm{b}^\mathsf{T}(0), \ldots, x_\mathrm{b}^\mathsf{T}(N)]^\mathsf{T}$, $C_\mathrm{prod}(k)$ and $C_\mathrm{grid}(k)$ are defined in (2.14) and (2.13), and $\epsilon_\mathrm{c}(k)$ is a stacked vector with the continuous decision variables over the prediction horizon. Furthermore, c is a weighing vector that contains the market energy prices over the prediction horizon, and it is represented by

$$c = [c_0^\mathsf{T}, \ \ldots, \ c_{\mathrm{N_p}-1}^\mathsf{T}]^\mathsf{T},$$

where

$$c_k = [0, \ c_\mathrm{sell}(k), \ c_\mathrm{prod}(k) \cdot 1_{1 \times N_\mathrm{gen}}, \ 0, \ c_\mathrm{buy}(0) - c_\mathrm{sell}(k)]^\mathsf{T}.$$

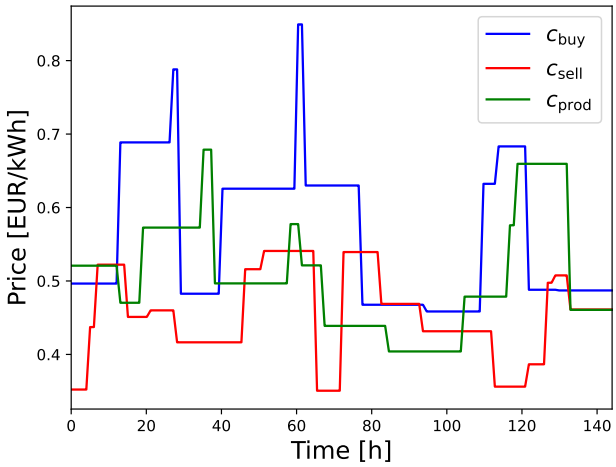


Figure 2.10: Price profiles for buying (c_{buy}), selling (c_{sell}), and producing (c_{prod}) energy over a period of 144 hours (3 days).

Note that the cost function $J(\cdot)$ is linear with respect to $\epsilon_{\text{c}}(k)$ and that it can be negative in the scenario where the microgrid operator profits from selling energy to the main grid. With the objective function defined, the optimization problem for the MPC controller can now be addressed. Along with minimizing the operation cost, the microgrid operator also has to satisfy some operating constraints, which are discussed next. Consider the following finite-horizon optimal control problem relative to the MPC controller:

$$\min_{\mathbf{x}_{\text{b}}(k),\epsilon_{\text{c}}(k),\epsilon_{\text{d}}(k)} J(\mathbf{x}_{\text{b}}(k),\gamma(k),\epsilon_{\text{c}}(k),\epsilon_{\text{d}}(k))$$

$$\text{s.t. } x_{\text{b}}(k+l+1) = x_{\text{b}}(k+l) + T_{\text{s}}(\eta_{\text{c}} - \frac{1}{\eta_{\text{d}}}) z_{\text{b}}(k+l) + \frac{T_{\text{s}}}{\eta_{\text{d}}} P_{\text{b}}(k+l), \tag{2.17a}$$

$$E_2\delta(k+l) + E_3 z(k+l) \leq E_1 u(k+l) + E_4 x_{\text{b}}(k) + E_5, \tag{2.17b}$$

$$P_{\text{b}}(k+l) = \sum_{j=1}^{N_{\text{gen}}} (P_j^{\text{dis}}(k+l)) P_{\text{res}}(k+l) + P_{\text{grid}}(k+l) - P_{\text{load}}(k+l), \tag{2.17c}$$

$$\underline{P}_{\text{b}} \leq P_{\text{b}}(k+l) \leq \bar{P}_{\text{b}}, \tag{2.17d}$$

$$\underline{P}_{\text{grid}}(k+l) \leq P_{\text{grid}}(k+l) \leq \bar{P}_{\text{grid}}, \tag{2.17e}$$

$$\delta_j^{\text{dis}}(k+l)\underline{P}_j^{\text{dis}} \leq P_j^{\text{dis}}(k+l) \leq \delta_j^{\text{dis}}(k+l)\bar{P}_j^{\text{dis}}, \tag{2.17f}$$

$$\text{for } j = 1,\ldots,N_{\text{gen}}, \text{ and}$$

$$\text{for } l = 0,\ldots,N_{\text{p}}-,$$

where the cost $J(\cdot)$ is defined by (2.16) and the decision variables and parameters are described in Tables 2.1 and 2.4. The first constraint (2.17a) concerns the dynamic of the storage unit. The equations (2.17b) arise from the introduction of the discrete and auxiliary variables introduced for the MLD modeling. The power balance of the microgrid and its connection with the main grid is enforced by constraint (2.17c) – the generation

must correspond to the load at all time steps. The lower and upper limits for the power exchange with the storage unit and with the main grid are represented in (2.17d) and (2.17e), respectively. At last, the dispatchable generation units have minimum and maximum operation levels, which are represented by (2.17f).

The objective function and the constraints of (2.17) are linear with respect to the optimization variables. Since the model and the constraints are linear, they can be conveniently represented in MLD form, as in (2.1). As a result, the optimization problem (2.17) can be recast as the mixed-integer linear program (2.6).

Parameter	Description	Value
$\bar{x}_{\mathrm{b}}$	maximum storage unit level	250 [kWh]
$\underline{x}_{\mathrm{b}}$	minimum storage unit level	25 [kWh]
$\bar{P}_{\mathrm{grid}}$	maximum power exchange with main grid	1000 [kW]
$\underline{P}_{\mathrm{grid}}$	minimum power exchange with main grid	-1000 [kW]
$\bar{P}_{\mathrm{b}}$	maximum power exchange with storage unit	100 [kW]
$\underline{P}_{\mathrm{b}}$	minimum power exchange with storage unit	-100 [kW]
$\bar{P}_i^{\mathrm{dis}}$	maximum power output of dispatchable unit i	100 [kW]
$\underline{P}_i^{\mathrm{dis}}$	minimum power output of dispatchable unit i	100 [kW]
η_c	charging efficiency of storage unit	0.9
η_d	discharging efficiency of storage unit	0.99
N_{gen}	number of generators	3

Table 2.4: Microgrid parameters, their descriptions, and their values.

2.B Supervised learning approach

The training procedure of the SL approach presented in [29] is briefly summarized to highlight fundamental differences between both RL and SL approaches. The approach presented in [29] employs an LSTM network for classification, which is the same neural network architecture used in our RL approach for the policy approximation, facilitating the comparison between the performances of both methods. Let the optimal solution of the MPC problem in (2.6) for a given state $\chi(k)$ be defined by the tuple $(\epsilon_d^*(k),\, \epsilon_c^*(k))$ containing the discrete and continuous optimization variables. The dataset for training is created by solving the MPC problem to optimality with branch-and-bound a number N_{data} of times with different initial augmented states, which are randomly sampled across the system's operating region of the state-space. The resulting pairs $\{(\chi(k),\, \epsilon_d^*(k))\}_{k=1}^{N_{\mathrm{data}}}$ are stored in a data buffer $\mathscr{D}$. Note that the continuous optimal solutions are not stored in the data buffer because the learning objective only concerns the prediction of the discrete optimal solutions. Having the dataset, it is straightforward to tune the parameters of the LSTM network by gradient descent on the following expression:

$$L = \frac{1}{T}\sum_{k=1}^{T}\sum_{l=0}^{N_{\mathrm{p}}-1} g\left((y_l(k) - [\phi_\theta^{\mathrm{LSTM}}(\chi(k))]_l)\right),$$

where $g(\cdot)$ is a loss function, e.g., cross entropy function, T is the size of the mini-batch, the target $y_l(k)$ is defined as the one-hot encoding of the optimal sub-action $\epsilon_{\mathrm{d},l}^*(k)$ and

the l-th output of the unrolled LSTM network with weights θ is denoted by $[\phi_\theta^{\text{LSTM}}(\cdot)]_k$. For more details, the reader is referred to [29].

3

LEARNING-BASED MPC FOR REAL-TIME TRAIN RESCHEDULING

This chapter focuses on passenger-oriented real-time train rescheduling, considering flexible train composition and rolling stock circulation, by integrating learning-based and optimization-based approaches. A learning-based model predictive control (MPC) approach is developed for real-time train rescheduling with flexible train composition and rolling stock circulation to address time-varying passenger demands. In the proposed approach, the values of the integer variables are obtained by pre-trained long short-term memory (LSTM) networks, while the continuous variables are determined through nonlinear constrained optimization. The learning-based MPC approach enables us to jointly consider efficiency and constraint satisfaction by combining learning-based and optimization-based approaches. In order to reduce the number of integer variables, four presolve techniques are developed to prune a subset of integer decision variables. Numerical simulations based on real-life data from the Beijing urban rail transit system are conducted to illustrate the effectiveness of the developed learning-based MPC approach.

3.1. INTRODUCTION

Urban rail transit has become increasingly important in large cities due to its reliability, high capacity, and eco-friendly characteristics. Urban rail transit systems prioritize safe and efficient train operations while providing high-quality service to passengers. Effective real-time train scheduling is essential for enhancing passenger satisfaction and minimizing operational costs within infrastructure limitations. However, the rapid expansion of urban rail transit systems and the increasing passenger demands pose significant challenges to real-time scheduling. Advanced scheduling models and control strategies are required to develop efficient timetables and to improve the overall performance of urban

This chapter is based on [87], [123]. C. F. O. da Silva shares first authorship in [87] and contributed to the methodology design, the integration of learning and control approaches, implementation, and writing of the manuscript.

rail transit systems.

3.1.1. PASSENGER-ORIENTED TRAIN SCHEDULING

In urban rail transit systems, passenger demands vary throughout the day, necessitating adjustments to the train schedules to accommodate these demand variations while considering operational costs. One direction addresses time-varying passenger demands by optimizing the departure and arrival times of trains at each station, while considering several attributes of train operations and infrastructure restrictions, e.g., train stopping plans (the set of station stops for each train) [26], [109], rolling stock circulations [140], and train speed levels [66], [142]. [109] optimized train stopping plans and timetables of a high-speed railway line considering time-varying passenger demands by formulating a mixed-integer linear programming (MILP) problem, and the aim is to find a solution that considers passenger preferences for departure times while ensuring trains operate within capacity limits. Considering the passenger load of trains, [144] minimized the passenger waiting time and the energy consumption by developing a heuristic algorithm to solve the resulting nonlinear integer programming problem. [142] formulated a mixed-integer nonlinear programming (MINLP) problem to minimize train energy consumption and passenger waiting times by optimizing train speed levels and headway deviations. To address train scheduling under uncertain passenger demands, [26] introduced a protection level on the departure and arrival times, and solved an MILP to obtain the train schedule and the stopping plan. [67] investigated minimization of operation costs, passenger waiting time, and passenger travel time, and developed a variable neighborhood search approach to solve the resulting problem. However, the above studies optimize train schedules within the fixed transport capacity of a line, such as fixed train compositions or fixed train departure frequencies. As transport capacity directly impacts passenger flows, further research is required to improve passenger satisfaction by including transport capacity as a decision variable.

Another direction for train scheduling problems addresses time-varying passenger demands by optimizing transport capacity explicitly. Several studies focus on optimizing transport capacity by adjusting train departure frequencies, with higher frequencies during peak hours and lower frequencies during off-peak hours [28], [85], [86]. To handle uncertain passenger demands, [85] developed a scenario-based distributed control approach, in which a scenario reduction technique is applied to reduce the computational burden. [108] developed a two-step approach in which the first step determines the nominal line plan and the second step adjusts the time plan based on the actual measured passenger demands. Instead of changing the departure frequency, which would significantly affect the schedule, recently, many researchers have focused on optimizing the composition of the train [148], [152]. [103] developed a column-generation-based approach to optimize the timetable, train composition, and rolling stock circulation plan of an urban rail transit line. Their paper concludes that flexible train composition can provide additional adaptability to better match time-varying passenger demands. [136] investigated flexible train composition and rolling stock circulation of an urban rail transit line, and solved the resulting MILP problem by developing an approximation approach and a two-stage meta-heuristic algorithm. [146] investigated the train scheduling problem with flexible train composition for an urban rail transit line, and they applied an

adaptive large neighborhood search algorithm to solve the resulting integer programming problem. As the inclusion of train composition optimization and rolling stock circulation planning introduces additional integer variables, the above studies indicate that the online computational complexity is the main challenge of incorporating flexible train composition into the real-time train scheduling problem.

3.1.2. MPC FOR REAL-TIME TRAIN SCHEDULING

Model predictive control (MPC) has been widely adopted in various applications for its ability to handle multivariable constrained control problems [58], [94], [110]. The train scheduling problem is a typical constrained control problem, and many studies have applied MPC for real-time train scheduling. [43] first applied MPC in the train scheduling problem to minimize train delays by adjusting transfer connections. [27] developed an MPC algorithm to optimize timetables, transfer connections, and train assignment plans in complex station areas. [31] proposed an MPC approach for train rescheduling during disruptions and delays, where in each step the resulting MILP problem is solved by combining bi-level heuristics and distributed optimization. The above studies only handle operator-related factors, leaving an open gap in including passenger demands in real-time train scheduling problems to improve the service quality of urban rail transit systems.

In recent years, several studies have focused on MPC for real-time passenger-oriented train scheduling. [6] applied MPC to compute the train timetable of a metro line, considering train headway and passenger load, where a linear programming problem is solved at each step. Based on a state-space model, [82] developed a robust MPC approach to minimize the upper bound of the timetable deviation from the nominal timetable under uncertain disturbances. By solving linear matrix inequalities, they constructed a Lyapunov function to ensure the attenuation of the timetable deviation. An event-triggered MPC approach is further developed by [138] to reduce the computational burden of updating control variables in each step. However, these studies do not explicitly consider train capacity limitations and time-varying passenger demands, and the results are based on the assumption that the maximum number of passengers does not exceed the maximum train capacity. [35] investigated real-time train rescheduling and stop-skipping strategies under disturbances and dynamic passenger flows. A mixed-integer quadratic programming (MIQP) problem is solved at each MPC step to minimize the total deviation from the original timetable and the number of waiting passengers. In the follow-up work, [36] applied distributed MPC to address the real-time train regulation problem, using Dantzig-Wolfe decomposition to solve the resulting optimization problem. [84] explicitly incorporated time-varying passenger demands and train capacity limitations into the real-time train scheduling problem. In [84], the time-varying passenger demand is approximated as a piecewise constant function by dividing the planning time window into several time intervals of equal length, and an MILP-based MPC approach is adopted for real-time train scheduling. The main challenge of applying MPC in real time is the online computational burden. Including additional attributes, such as train capacity, train composition, and rolling stock circulation, will further increase the computational burden. Therefore, further research is required to develop efficient MPC approaches for real-time passenger-oriented train scheduling.

3.1.3. LEARNING-BASED TRAIN SCHEDULING

Learning-based approaches are considered effective methodologies for reducing the online computational burden. Learning-based approaches, including deep learning and reinforcement learning (RL), have also been applied in train scheduling problems in recent years [130]. SL trains models on labeled data to make accurate predictions or classifications, while RL trains an agent to make decisions through trial-and-error using rewards and penalties. [79] applied a deep learning approach to an energy-efficient timetable rescheduling problem, where a long short-term memory (LSTM) network is trained to select the optimal operation mode. [120] applied Q-learning in the train rescheduling problem to reduce delays caused by disturbances and disruptions, and they illustrated their method on a single-lane track with three trains. [147] proposed an approximate dynamic programming approach to address train rescheduling problems, aiming to reduce passenger delays, total travel time, and train energy consumption. In [147], the states include disturbance information, train arrival times, the number of boarding passengers, and the number of waiting passengers, while the actions include rescheduling the dwell times and running times. [76] applied RL to determine track allocations and timetables of bidirectional railway lines to minimize the priority-weighted delay. [148] developed a proximal policy optimization approach for the train scheduling problem in an urban rail transit line, considering flexible train composition. In [148], the control policy and the value function are parameterized by artificial neural networks, and scheduling constraints are handled by a devised mask scheme. Simulation results show that this approach reduces the computational burden and improves solution quality compared to the genetic algorithm and differential evolution. [100] introduced an adaptive rollout surrogate approximate dynamic programming (ADP) framework for rail transit network operations, where the value function is approximated by a surrogate function, and the solution is obtained via a cross-entropy-based search, encompassing decisions related to service operations and flexible train unit deployment. [149] proposed a multi-agent deep reinforcement learning approach for disruption management with short-turning operations in a service corridor, enabling adaptive rescheduling under time-varying passenger demand. [137] formulated adaptive network dispatching as a partially observable Markov decision process and applied a deep recurrent Q network to derive dispatching policies with flexible fleet deployment. These recent studies have shown that learning-based frameworks can support train scheduling and may incorporate time-varying passenger demands in rail networks. However, explicitly modeling time-varying passenger demands in the train scheduling problem further tightens the coupling among timetable adjustments, capacity allocation via flexible train composition, and rolling-stock circulation. This coupling enlarges the discrete decision space and increases online computational complexity, while operational feasibility still needs to be guaranteed for real-world implementation. Therefore, scalability and constraint satisfaction remain key challenges when developing learning-based train scheduling approaches.

In the above studies, scalability and constraint satisfaction are the main challenges in developing learning-based train scheduling approaches. The train scheduling problem is typically formulated as an MILP or MINLP problem, and the computational complexity increases rapidly as the number of integer variables increases. In recent years, some research has combined learning-based and optimization-based approaches for MILP or

MINLP problems by using learning-based approaches to obtain the integer variables [29], [117], [122]. Having the integer variables fixed, the continuous variables are then obtained by solving a linear or nonlinear programming problem. This idea aims to combine the advantages of both learning-based and optimization-based approaches, i.e., the online computational efficiency of learning-based approaches and the feasibility guarantees of constrained optimization [29], [117]. Promising results of learning-based train scheduling and the recent learning-based approaches for mixed integer programming problems have inspired us to develop an optimization-driven learning-based framework for passenger-oriented real-time train scheduling.

3.1.4. CHAPTER CONTRIBUTIONS AND STRUCTURE

The studies introduced in Sections 3.1.1, 3.1.2, and 3.1.3 are summarized in Tables 3.1 and 3.2, categorized by network details, train composition, train order, rolling stock circulation, passenger demands, and solution method. From the above literature review, it can be found that flexible train composition combined with rolling stock circulation planning is an effective approach to balance operational costs and service quality. Shorter trains can be deployed during off-peak hours and longer trains during peak hours, thereby balancing operational costs and passenger waiting times. The passenger-oriented train scheduling problem is typically formulated as a mixed-integer programming problem. Various approaches can be applied to solve the resulting scheduling problem. Among them, optimization-based methods often encounter computational challenges for large-scale networks, necessitating additional efforts such as heuristics, two-step frameworks, and distributed approaches to reduce the computational burden. Learning-based methods are considered promising for addressing computational complexity. However, available learning-based methods often face scalability challenges, and existing passenger-oriented train scheduling approaches are generally limited to small-scale scenarios involving a single rail line. Learning-based methods are considered promising for addressing computational complexity. However, developing learning-based methods that are both scalable and constraint-satisfying for real-time passenger-oriented scheduling remains challenging, especially when time-varying passenger demand is explicitly considered and jointly optimized with flexible train composition and rolling-stock circulation in network settings.

The current chapter, therefore, addresses the learning-based train rescheduling problem in rail networks, considering time-varying passenger demands, flexible train composition, and rolling stock circulation. Unlike existing train rescheduling methods in the literature, the proposed approach integrates optimization and deep learning to combine the constraint satisfaction advantages of optimization-based methods with the fast online computation advantages of learning-based approaches. The main contributions of the chapter are listed as follows:

1. A passenger-oriented train rescheduling model in [84] is extended to include time-varying sectional passenger demands, flexible train composition, and rolling stock circulation. The time-varying passenger demands can be approximated as a piecewise constant function by dividing the prediction horizon into several time intervals.

2. Four presolve techniques are developed to streamline optimization processes by

pruning a subset of integer decision variables. After implementing the presolve techniques, long short-term memory (LSTM) networks are applied to obtain the remaining integer variables with reduced dimensions.

3. A deep-learning-based MPC approach is developed for real-time train rescheduling. To improve the online computational efficiency of MPC, the learning-based approach is applied to obtain integer variables, while the detailed timetable is obtained by solving a constrained optimization problem with the fixed integer variables.

The remaining part of the chapter is organized as follows: Section 3.2 provides the problem description and general explanations. Section 3.3 introduces the passenger-oriented train rescheduling model. In Section 3.4, the problem formulation and the MINLP-based MPC approach are presented. In Section 3.5, we propose a learning-based MPC approach for real-time train rescheduling. Section 3.6 provides an illustrative case study. Section 3.7 concludes the chapter.

3.2. PROBLEM DESCRIPTION AND EXPLANATIONS

In this section, we present the problem description of train rescheduling with flexible train composition. Some general explanations and assumptions used in the problem formulation are summarized as follows.

(1) In urban rail transit, a *line* is defined as a fixed route that connects an ordered sequence of stations and is served by trains operating in both directions.

(2) A *train unit* is the minimum rolling-stock element that can operate independently and can be coupled/decoupled; multiple train units form a train composition.

(3) A *train service* is defined as a train departure from its origin station, visiting every station in the line, and finally returning to the depot (or connecting to other train services at the depot). As illustrated in Fig. 3.1, a train service consists of one or several train units, and the composition can be changed at the station connected to a depot by adding or removing train units.

(4) Since passenger volumes are typically large, the approximation error incurred by relaxing passenger counts to continuous variables is relatively small [84]. Hence, variables representing the number of passengers are modeled as nonnegative real-valued variables.

In this chapter, we jointly optimize train compositions and timetables considering time-varying passenger demands and rolling stock circulation. We focus on the train rescheduling problem with the objective of optimizing departure times, arrival times, and train composition. In this context, both train travel times and transport capacity are adjusted to accommodate the time-varying passenger demands. We aim to minimize the total waiting time of passengers and the train energy consumption, and the control actions are the train composition, train service orders at the platforms connected to the same depot, and departure/arrival times. Flexible train composition and rolling stock circulation relate to the set of integer variables, while train timetables relate to the set of continuous variables, such as departure and arrival times. Trains operate under several constraints, including train capacity constraints, train availability in a depot, and headway constraints. By applying MPC, we solve the resulting mixed-integer programming problem in a moving horizon manner for real-time train rescheduling. To

Table 3.1: Problem characteristics of relevant studies on passenger-oriented train rescheduling.

Reference	Infrastructure	Train composition	Train order	Rolling stock circulation	Passenger demands
[120]	single line	fixed	yes	no	no
[82]	single line	fixed	no	no	yes
[76]	single line	fixed	yes	no	no
[26]	single line	fixed	yes	no	yes
[79]	single line	fixed	no	no	no
[108]	single line	fixed	no	no	yes
[142]	single line	fixed	no	yes	yes
[148]	single line	flexible	no	yes	yes
[31]	network	fixed	yes	no	no
[138]	single line	fixed	no	no	yes
[35]	single line	fixed	no	yes	yes
[84]	network	fixed	no	no	yes
[103]	single line	flexible	yes	yes	yes
[86]	network	fixed	yes	yes	yes
[100]	network	flexible	no	yes	yes
[149]	single line	flexible	no	yes	yes
[85]	network	fixed	no	yes	yes
[136]	single line	flexible	no	yes	yes
[146]	single line	flexible	no	yes	yes
[137]	network	flexible	no	yes	yes
[67]	single line	fixed	no	yes	yes
Current chapter	network	flexible	yes	yes	yes

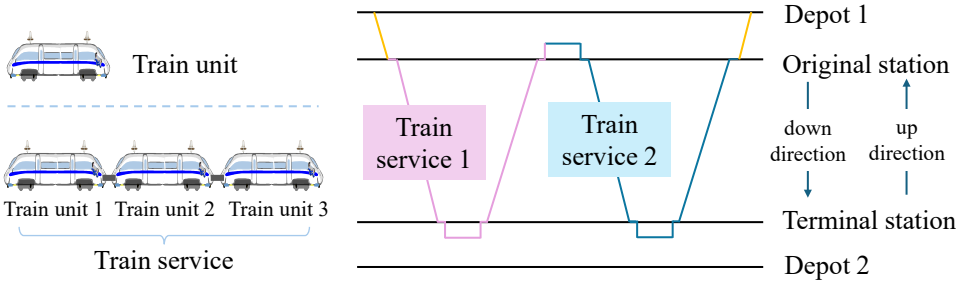


Figure 3.1: Illustration of definitions used in this chapter.

Table 3.2: Solution methods and objectives of relevant studies on passenger-oriented train rescheduling.

Ref.	Solution method	Objective(s)
[120]	Q learning	minimize the total delay of trains
[82]	robust MPC	minimize the deviations from the nominal timetable
[76]	Q learning	minimize the priority-weighted delay
[26]	robust MPC	minimize train travel time and unsatisfied passenger demands
[79]	deep learning	minimize energy consumption
[108]	Lagrangian relaxation	minimize operational costs and total passenger travel time
[142]	MILP	minimize train energy consumption and passenger waiting times
[148]	deep reinforcement learning	minimize the operation cost and passenger waiting times
[31]	bi-level heuristics, MPC	minimize train delays
[138]	event-triggered MPC	minimize the deviations from the nominal timetable
[35]	MPC	minimize total train deviation and the number of waiting passengers
[84]	MPC	minimize the waiting time of passengers
[103]	column-generation-based heuristic	minimize passenger waiting times and the usage of train units
[86]	bi-level MPC	minimize passenger travel times and the total operation costs
[100]	rollout surrogate-ADP	minimize total waiting time of passengers
[149]	multi-agent deep RL	minimize passenger travel times and the total operation costs
[85]	scenario-based distributed MPC	minimize passenger travel times and the total operation costs
[136]	two-stage heuristic	reduce the number of stranded passengers, the number of rolling stocks, and the operational costs
[146]	adaptive large neighborhood search	minimize total waiting time of passengers and the number of train units used
[137]	deep RL	minimize passenger travel times and the total operation costs
[67]	variable neighborhood search	minimize passenger waiting time, travel time and operation costs
Current chapter	deep-learning-based MPC	minimize passenger waiting times and operation costs

improve the online computational efficiency of MPC, we use the learning-based approach to obtain integer variables, i.e., train compositions and train orders; then, we optimize the detailed timetable with the fixed integer variables by solving a constrained optimization problem.

3.3. MATHEMATICAL FORMULATION

In this section, we develop a passenger-oriented train rescheduling model for urban rail transit systems. The notations for the model formulation are provided in Section 3.3.1. The train operation constraints of the model are introduced in Section 3.3.2. In Section 3.3.3, the rolling stock circulation constraints related to the model are introduced. In Section 3.3.4, passenger flow constraints of the model are presented.

3.3.1. NOTATIONS

Tables 3.3, 3.4, and 3.5 list the indices and input parameters, the decision variables, and the output variables of the model formulations, respectively.

3.3.2. TRAIN OPERATION CONSTRAINTS

In urban rail transit systems, each line typically comprises two directions, i.e., the up direction and the down direction, as shown for a line of P stations in Fig. 3.2. For each line, a train service can be defined as a train running from the starting platform to the terminal platform, e.g., from Platform 1 to Platform $2P$ in Fig. 3.2. Trains generally operate

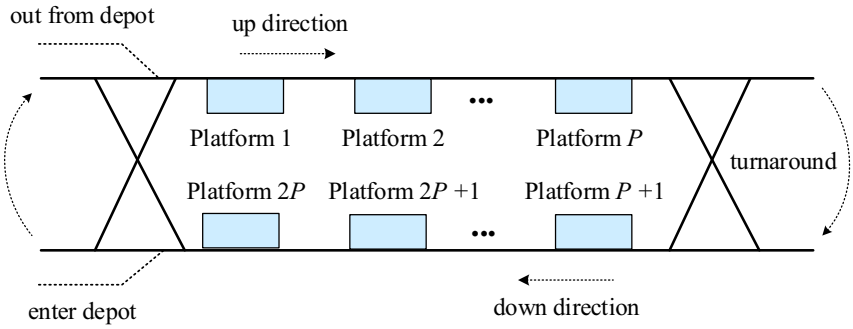


Figure 3.2: Layout of a bidirectional urban rail transit line.

following a predetermined timetable, and the predetermined departure time of train service k_p at platform p is represented by $d_p^{\text{pre}}(k_p)$.

In practice, premature departure is usually not permitted; thus, the actual departure time cannot be earlier than the predetermined departure time. In general, passengers prefer a predictable timetable at each platform so that they can conveniently plan their trips. Therefore, in this chapter, we keep the planned service schedule unchanged, i.e., we neither insert nor cancel train services when adjusting the timetable and train compositions.

Notations	Definition
p	Index of platforms, $p \in \mathscr{P}$; $\mathscr{P}$ is the set of platforms
k_p	Index of train services at platform p, $k_p \in \mathscr{I}_p$; $\mathscr{I}_p$ is the set of train services departing from platform p
z	Index of depots, $z \in \mathscr{Z}$; $\mathscr{Z}$ is the set of depots
$\mathrm{p}^{\mathrm{pla}}(p)$	Predecessor platform of platform p
$\mathrm{s}^{\mathrm{pla}}(p)$	Successor platform of platform p
$d_p^{\mathrm{pre}}(k_p)$	Predetermined departure time of train service k_p at platform p
$h_p^{\min}$	Minimum departure-arrival headway at platform p
$r_p^{\min}$	Minimum running time of trains from platform p to its succeeding platform
$r_p^{\max}$	Maximum running time of trains from platform p to its succeeding platform
$C_{\max}$	Maximum capacity of a train unit
$\ell^{\min}$	Minimum number of train units allowed to be included in any train service
$\ell^{\max}$	Maximum number of train units allowed to be included in any train service
σ_p	Parameter indicating whether the train composition can be adjusted at platform p, i.e., if train composition can be adjusted at platform p, then, $\sigma_p = 1$, otherwise, $\sigma_p = 0$.
$\tau_p^{\min}$	Minimum dwell time of a train service at platform p
t_p^{comp}	Time required for changing the train composition at platform p
$t_{p,p'}^{\mathrm{roll}}$	Time for train units from platform p to other platform p' corresponding to the same depot
$\mathrm{dep}(z)$	Set of platforms directly connected with depot z
$\mathrm{pla}(p)$	Set of platforms belonging to the same station as platform p
$\rho_p(k_p)$	Passenger demands from platform p to its successor platform during $d_{k-1,p}^{\mathrm{pre}}$ to $d_{k,p}^{\mathrm{pre}}$
N_z^{train}	Total number of train units available at depot z
$\chi_{k_q,q,k_p,p}$	Binary parameter, which can be determined based on the predetermined timetable; if train k_q at platform q has transfer connection with train k_p at platform p, $\chi_{k_q,q,k_p,p} = 1$; otherwise, $\chi_{k_q,q,k_p,p} = 0$.
$\beta_{q,p}$	Transfer rate from platform q to platform p
t_q^{trans}	Average transfer time from platform q to the platforms at the same station
E_p^{energy}	Average energy consumption for a train unit running from platform p to its successor platform
E_p^{add}	Additional cost of changing train composition at platform p

Table 3.3: Indices and input parameters.

Notations	Definition
$d_p(k_p)$	Departure time of train service k_p at platform p
$a_p(k_p)$	Arrival time of train service k_p at platform p
$\ell_p(k_p)$	Number of train units included in train service k_p at platform p, $\ell_p(k_p) \in \mathbb{Z}_+$
$\xi_{k_p,k_{p'},p,p'}$	Binary variable; if the train units from train service $k_{p'}$ at platform p' can be used for train service k_p at platform p, $\xi_{k_p,k_{p'},p,p'} = 1$; otherwise, $\xi_{k_p,k_{p'},p,p'} = 0$.
$\eta_{k_p,p}$	Binary variable; if the composition of train service k_p is changed at platform p, $\eta_{k_p,p} = 1$; otherwise, $\eta_{k_p,p} = 0$.

Table 3.4: Decision variables

Notations	Definition
$\tau_p(k_p)$	Dwell time of train service k_p at platform p
$h_p(k_p)$	Departure-arrival headway between train service k_p and train service k_p+1 at platform p
$r_p(k_p)$	Running time of train service k_p from platform p to its successor platform
$r_p^{\text{turn}}(k_p)$	Turnaround time of train service k_p at platform p
$\tau_p^{\text{add}}(k_p)$	Additional time required for changing the composition of train service k_p at platform p
$y_p(k_p)$	Number of train units coming to/from the depot for train service k_p, $y_p(k_p) \in \mathbb{Z}$
$n_p(k_p)$	Number of passengers waiting at platform p at time $d_p^{\text{pre}}(k_p)$
$n_p^{\text{trans}}(k_p)$	Number of transfer passengers arriving at platform p for train k_p
$n_p^{\text{depart}}(k_p)$	Number of passengers departing from platform p with train service k_p
$n_p^{\text{arrive}}(k_p)$	Number of passengers arriving at platform p with train k_p from the predecessor platform $\text{p}^{\text{pla}}(p)$
$n_p^{\text{before}}(k_p)$	Number of passengers waiting at platform p immediately before the departure of train service k_p
$C_p(k_p)$	Total capacity of train service k_p at platform p
$n_p^{\text{after}}(k_p)$	Number of passengers waiting at platform p immediately after train service k_p departs from platform p.

Table 3.5: Output variables

Hence, the actual departure time is constrained by

$$d_p^{\text{pre}}(k_p) \le d_p(k_p) < d_p^{\text{pre}}(k_p+1), \tag{3.1}$$

where $d_p(k_p)$ is the actual departure time of train service k_p at platform p determined by

$$d_p(k_p) = a_p(k_p) + \tau_p(k_p), \tag{3.2}$$

where $a_p(k_p)$ and $\tau_p(k_p)$ are the arrival time and dwell time of train service k_p at platform p.

For the safe operation of trains, the headway constraint should be satisfied:

$$a_p(k_p+1) = d_p(k_p) + h_p(k_p), \tag{3.3}$$

$$h_p(k_p) \ge h_p^{\min}, \tag{3.4}$$

where $h_p(k_p)$ is headway of train service k_p at platform p , and $h_p^{\min}$ denotes the minimum headway.

The arrival time of train service k_p at the successor platform of platform p should also satisfy

$$a_{\text{s}^{\text{pla}}(p)}(k_p) = d_p(k_p) + r_p(k_p), \tag{3.5}$$

$$r_p^{\min} \le r_p(k_p) \le r_p^{\max}, \tag{3.6}$$

where $r_p(k_p)$ is the running time of train service k_p from platform p to platform $\text{s}^{\text{pla}}(p)$, and $r_p^{\min}$ and $r_p^{\max}$ are the minimum and maximum running time from platform p to platform $\text{s}^{\text{pla}}(p)$.

3.3.3. ROLLING STOCK CIRCULATION CONSTRAINTS

At the terminal station, a turnaround action is required for the continuation of the train service. The turnaround constraints can be formulated as:

$$a_{\text{s}^{\text{pla}}(p)}(k_{\text{s}^{\text{pla}}(p)}) = d_p(k_p) + r_p^{\text{turn}}(k_p), \tag{3.7}$$

$$r_p^{\text{turn,min}} \le r_p^{\text{turn}}(k_p) \le r_p^{\text{turn,max}}, \tag{3.8}$$

where $r_p^{\text{turn}}(k_p)$ represents the turnaround time of train service k_p at platform p, and $r_{p,\min}^{\text{turn}}$ and $r_{p,\max}^{\text{turn}}$ denote the minimum and maximum turnaround times at platform p.

An urban rail transit line typically has a limited number of train units that either operate on the line or are stored in the depot. The train composition can be adjusted at the platform that is linked with the depot, and the number of train units $\ell_p(k_p) \in \mathbb{Z}_+$ for train service k_p at platform p is determined by

$$\ell_p(k_p) = \ell_{\text{p}^{\text{pla}}(p)}(k_p) + \sigma_p y_p(k_p), \tag{3.9}$$

$$\ell_{\min} \le \ell_p(k_p) \le \ell_{\max}, \tag{3.10}$$

where σ_p is a parameter related to the network layout indicating whether the train composition can be adjusted at platform p, i.e., if the train composition can be adjusted at

platform p, then $\sigma_p = 1$, otherwise, $\sigma_p = 0$. Moreover, $y_p(k_p) \in \mathbb{Z}$ represents the number of train units coming to/from the depot for train service k_p; specifically, if $y_p(k_p) > 0$, then, $y_p(k_p)$ extra train units will come from the depot to be added to train service k_p; if $y_p(k_p) < 0$, train service k_p will be decomposed, and $|y_p(k_p)|$ train units will return to the depot; if $y_p(k_p) = 0$ the composition of train service k_p will not be changed at platform p. Furthermore, $\ell_{\min}$ and $\ell_{\max}$ represent the minimum and the maximum number of train units allowed to be included in any train service.

Remark 3.1. If the depot is linked with the terminal platform (e.g., Platform 1 in Fig. 3.2) and train service k_p is performed by the turnaround train units of the previous train service, then (3.9) becomes $\ell_1(k_1) = \ell_{2P}(k_{2P} - 1) + \sigma_1 y_1(k_1)$, i.e., we set $\ell_1(k_1) = \ell_{2P}(k_{2P} - 1)$.

To capture the changes of train composition at platform p, we introduce a binary variable $\eta_{k_p,p}$ as

$$\eta_{k_p,p} = \begin{cases} 1, & \text{if} \quad |y_p(k_p)| > 0; \\ 0, & \text{otherwise}, \end{cases} \tag{3.11}$$

where $\eta_{k_p,p} = 1$ indicates the composition of train service k_p is changed at platform p; otherwise, $\eta_{k_p,p} = 0$.

In general, additional dwell time is required when changing the train composition; thus, the dwell time $\tau_p(k_p)$ of train service k_p at platform p should satisfy:

$$\tau_p(k_p) \geq \tau_p^{\min} + \sigma_p \tau_p^{\text{add}}(k_p), \tag{3.12}$$

where $\tau_p^{\min}$ is the minimum dwell time at platform p, and $\tau_p^{\text{add}}(k_p)$ represents the additional time required for changing the composition of train service k_p at platform p, which can be determined by

$$\tau_p^{\text{add}}(k_p) = \eta_{k_p,p} \cdot t_p^{\text{comp}}, \tag{3.13}$$

where t_p^{comp} represents the time required for changing the train composition at platform p.

A depot typically connects to at least one platform, and the departure order of train services at the corresponding platforms influences the number of available train units in a depot, i.e., if a newly arriving train service requires changing its composition, the total number of train units in the corresponding depot will change. To represent the relation of departure time of train services corresponding to the same depot, we define a binary variable $\xi_{k_p,k_{p'},p,p'}$ as

$$\xi_{k_p,k_{p'},p,p'} = \begin{cases} 1, & \text{if} \quad d_p(k_p) \geq d_{p'}(k_{p'}) + t_{p',p}^{\text{roll}}; \\ 0, & \text{otherwise}, \end{cases} \tag{3.14}$$

where $d_p(k_p)$ is the departure time of train service k_p at platform p, $d_{p'}(k_{p'})$ is the departure time of train service $k_{p'}$ at platform p', and $t_{p',p}^{\text{roll}}$ is the time for trains from platform p' to platform p corresponding to the same depot. In (3.14), $\xi_{k_p,k_{p'},p,p'} = 1$ indicates train units assigned to train service $k_{p'}$ from platform p' for train service k_p at platform p; otherwise, $\xi_{k_p,k_{p'},p,p'} = 0$.

Let $\mathscr{I}_p$ denote the set of train services departing from platform p, and let $\mathscr{I}_p(k_p) = \{k \in \mathscr{I}_p \mid k \le k_p\}$ denote the set of services on platform p up to service k_p. The rolling-stock circulation constraint is

$$\sum_{k'_p \in \mathscr{I}_p(k_p)} y_p(k'_p) + \sum_{p' \in \mathrm{dep}(z)\setminus\{p\}} \sum_{k_{p'} \in \mathscr{I}_{p'}} \xi_{k_p,k_{p'},p,p'}\, y_{p'}(k_{p'}) \le N_z^{\mathrm{train}}. \tag{3.15}$$

Here, z is the index of the depot, $\mathrm{dep}(z)$ defines the set of platforms directly connected with depot z, and N_z^{train} denotes the total number of train units in depot z. Eq. (3.15) ensures that, for each service k_p, the depot inventory remains feasible under both dispatching ($y_p(k_p) > 0$) and returning ($y_p(k_p) < 0$) of train units, and the total number of train units in depot z is bounded by N_z^{train}. In this context, rolling stock can be reused across different services within the same line through the circulation constraints. At depot-linked platforms, a train unit can either return to the depot or be assigned to subsequent services on the same line as long as the rolling stock circulation constraints are satisfied.

3.3.4. PASSENGER FLOW CONSTRAINTS

A predetermined timetable is generally designed based on time-varying passenger demand to guide daily train operations, and passenger flows are updated over consecutive predetermined departure events. At each platform, the predetermined timetable naturally divides the planning time window into several time intervals, with the predetermined train departure times at the platforms as the partition points. Following the previous research in [84], which demonstrated that the piecewise constant approximation of the time-varying passenger demand can significantly reduce computational complexity while preserving comparable accuracy, in this chapter, we also approximate the passenger arrival rates as piecewise constant functions. The resulting piecewise approximation is shown in Fig. 3.3 where $d_p^{\mathrm{pre}}(k_p)$ is the predetermined departure time of train service k_p; and $\rho_p(k_p)$ denotes passenger arrival rate during $d_p^{\mathrm{pre}}(k_p-1)$ to $d_p^{\mathrm{pre}}(k_p)$ for passengers aiming to leave platform p with train service k_p.

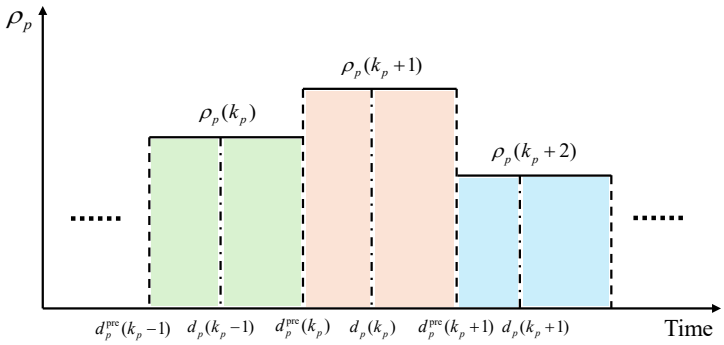


Figure 3.3: Illustration of piecewise approximation of passenger demands for platform p.

The number of passengers waiting at a platform immediately after the predetermined

departure time of train service k_p+1 at platform p can be calculated by

$$n_p(k_p+1) = n_p(k_p)+\rho_p(k_p+1)\left(d_p^{\text{pre}}(k_p+1)-d_p^{\text{pre}}(k_p)\right)+ \tag{3.16}$$

$$+n_p^{\text{trans}}(k_p)-n_p^{\text{depart}}(k_p), \tag{3.17}$$

where $n_p(k_p)$ represents the number of passengers waiting at platform p immediately before the predetermined departure time $d_p^{\text{pre}}(k_p)$; the term

$$\rho_p(k_p+1)\left(d_p^{\text{pre}}(k_p+1)-d_p^{\text{pre}}(k_p)\right)$$

gives the number of passengers arriving at platform p between $d_p^{\text{pre}}(k_p)$ and $d_p^{\text{pre}}(k_p+1)$; $n_p^{\text{trans}}(k_p)$ denotes the number of transfer passengers arriving at platform p who intend to board train service k_p, and $n_p^{\text{depart}}(k_p)$ denotes the number of passengers departing from platform p on train service k_p.

Since the departure time of each train service is adjusted according to (3.1), there exists one departure in each time interval. As shown in Fig. 3.3, the number of passengers $n_p^{\text{before}}(k_p)$ waiting at platform p immediately before the departure of train service k_p can be calculated by

$$n_p^{\text{before}}(k_p) = n_p(k_p)+\rho_p(k_p+1)\left(d_p(k_p)-d_p^{\text{pre}}(k_p)\right)+n_p^{\text{trans}}(k_p). \tag{3.18}$$

The number of passengers $n_p^{\text{depart}}(k_p)$ departing from platform p with train service k_p should satisfy

$$n_p^{\text{depart}}(k_p) \le n_p^{\text{before}}(k_p), \tag{3.19a}$$

$$n_p^{\text{depart}}(k_p) \le C_p(k_p), \tag{3.19b}$$

where $C_p(k_p)$ is the total capacity of train service k_p at platform p. Constraint (3.19) provides a standard linear approximation of

$$n_p^{\text{depart}}(k_p) = \min\{n_p^{\text{before}}(k_p), C_p(k_p)\},$$

which is commonly used in related studies (e.g., [88], [103], [136]).

The total capacity of train service k_p at platform p is computed by

$$C_p(k_p) = \ell_p(k_p)C_{\max}, \tag{3.20}$$

where $\ell_p(k_p)$ is the number of train units composing train service k_p at platform p.

The number of passengers $n_p^{\text{arrive}}(k_p)$ arriving at platform p with train k_p from the predecessor platform $\text{p}^{\text{pla}}(p)$ can be calculated by

$$n_p^{\text{arrive}}(k_p) = n_{\text{p}^{\text{pla}}(p)}^{\text{depart}}(k_p). \tag{3.21}$$

Then, the number of passengers $n_p^{\text{trans}}(k_p)$ transferring to platform p for train k_p is computed by

$$n_p^{\text{trans}}(k_p) = \sum_{q\in \text{pla}(p)} \sum_{k_q\in\mathscr{I}_q} \chi_{k_q,q,k_p,p}\beta_{q,p}n_q^{\text{arrive}}(k_q), \tag{3.22}$$

where pla(p) defines the set of platforms belonging to the same station as platform p, $\beta_{q,p}$ is the parameter defines the transfer rate from platform q to platform p, and $\chi_{k_q,q,k_p,p}$ is the binary parameter denoting transfer connection between train k_q at platform q and train k_p at platform p, which is defined as

$$\chi_{k_q,q,k_p,p} = \begin{cases} 1, & \text{if } d_p^{\text{pre}}(k_p-1) < d_q^{\text{pre}}(k_q) + t_q^{\text{trans}} \le d_p^{\text{pre}}(k_p); \\ 0, & \text{otherwise,} \end{cases} \tag{3.23}$$

where t_q^{trans} represents the average transfer time from platform q to the corresponding platforms at the same station. To improve the accuracy of the passenger flow model, (3.22) can be directly extended by making the transfer-rate parameter $\beta_{q,p}$ time-dependent, i.e., using $\beta_{q,p}(k_p)$, which can be estimated online from real-time passenger flow data.

After train service k_p departs from platform p, the number of passengers waiting at the platform can be calculated by

$$n_p^{\text{after}}(k_p) = n_p^{\text{before}}(k_p) - n_p^{\text{depart}}(k_p), \tag{3.24}$$

where $n_p^{\text{after}}(k_p)$ represents the number of passengers waiting at platform p immediately after train service k_p departs from platform p.

In summary, passenger flows are updated over consecutive predetermined departure events at each platform. For platform p, consider two consecutive predetermined departure times $d_p^{\text{pre}}(k_p)$ and $d_p^{\text{pre}}(k_p{+}1)$. The number of arriving passengers in this interval is computed by $\rho_p(k_p{+}1)\big(d_p^{\text{pre}}(k_p{+}1) - d_p^{\text{pre}}(k_p)\big)$. Consequently, the number of passengers waiting before train service $k_p{+}1$ equals the passengers remaining after service k_p plus the arrivals in this interval. The number of boarded (departing) passengers $n_p^{\text{depart}}(k_p)$ is limited by the available capacity implied by the train composition decision; any unboarded passengers remain waiting and are carried over to the next departure, which is captured by $n_p^{\text{after}}(k_p)$. For interchange platforms, transfer passengers are explicitly represented by $n_p^{\text{trans}}(k_p)$, and their impact is captured by the same update and reflected in the passenger-oriented objective. This departure-event update is used to implement the event-based MPC scheme indexed by κ in the next section.

3.4. MODEL PREDICTIVE CONTROL FOR TRAIN RESCHEDULING

3.4.1. PROBLEM FORMULATION

Based on the developed model, we can formulate the problem to minimize passenger delays and operational costs. The passenger delays corresponding to train service k_p at platform p can be formulated as

$$J_p^{\text{pass}}(k_p) = n_p(k_p)\big(d_p(k_p) - d_p^{\text{pre}}(k_p)\big) + n_p^{\text{after}}(k_p)\big(d_p^{\text{pre}}(k_p+1) - d_p(k_p)\big). \tag{3.25}$$

In this chapter, we focus on the additional waiting time relative to predetermined departures in an event-based manner. Specifically, $n_p(k_p)\big(d_p(k_p) - d_p^{\text{pre}}(k_p)\big)$ measures the additional delay of passengers waiting at platform p immediately before $d_p^{\text{pre}}(k_p)$ due to the rescheduled departure time $d_p(k_p)$. In addition, $n_p^{\text{after}}(k_p)\big(d_p^{\text{pre}}(k_p{+}1) - d_p(k_p)\big)$ measures the additional waiting time of passengers who intend to board service k_p but

cannot due to limited capacity and thus have to wait for the next predetermined departure $d_p^{\text{pre}}(k_p+1)$. Passengers arriving after $d_p^{\text{pre}}(k_p)$ are accounted for through the queue update at subsequent predetermined departures.

Assigning more train units to a train service can increase the capacity for transporting passengers while leading to higher energy consumption. Furthermore, changing the train composition may require additional workload from operators and thus lead to additional costs. The operational costs of train service k_p running from platform p to its successor platform can be expressed as

$$J_p^{\text{cost}}(k_p) = \ell_p(k_p) E_p^{\text{energy}} + \eta_{k_p,p} E_p^{\text{add}}, \tag{3.26}$$

where E_p^{energy} represents the average energy consumption for a train unit running from platform p to its successor platform, $\ell_p(k_p)E_p^{\text{energy}}$ denotes the approximate energy consumption for $\ell_p(k_p)$ train units running from platform p to its successor platform, and E_p^{add} denotes the additional cost for changing the train composition of a train service at platform p.

In urban rail transit systems, a train service typically departs from a depot, visiting each platform along a line before returning to the depot. Based on the planned departures at each platform, we adopt an event-driven MPC scheme and update the controller at planned departure events. The length of the control step, denoted as T_{ctrl}, is defined as the time interval between two consecutive train departures at a platform, and κ denotes the index of the control step[1].

Therefore, the optimization problem for train rescheduling over the control step κ_0 is

$$\begin{aligned} &\min_{\boldsymbol{g}(\kappa_0)} \; J(\kappa_0) := \sum_{p\in\mathscr{P}} \sum_{k_p\in\mathscr{N}_p(\kappa_0)} \Big(w_1 J_p^{\text{pass}}(k_p) + w_2 J_p^{\text{cost}}(k_p) \Big), \\ &\text{s.t.} \quad (3.1)-(3.26), \end{aligned} \tag{3.27}$$

where $\boldsymbol{g}(\kappa_0)$ represents a vector collecting all the variables of problem (3.27), $\mathscr{P}$ is the set collecting all the platforms of the line, $\mathscr{N}_p(\kappa_0)$ is the set indices of trains departing from platform p within the prediction time window starting at step κ_0, and w_1 and w_2 are weights balancing two objectives.

3.4.2. MINLP-BASED MPC FOR REAL-TIME TRAIN RESCHEDULING

Problem (3.27) contains piecewise constant ("if-then") constraints in (3.11) and (3.14). We apply the following transformation properties to convert (3.11) into a mixed logical dynamical (MLD) system [143].

Transformation property 3.1: If we introduce an auxiliary continuous variable $o_{k,p}$ and an auxiliary binary variable $\gamma_{k_p,p}$ with $\gamma_{k_p,p}=1 \Leftrightarrow o_{k_p,p}=y_p(k_p)$ and $\gamma_{k_p,p}=0 \Leftrightarrow o_{k_p,p}=-y_p(k_p)$, and then $o_{k_p,p}=|y_p(k_p)|$ is equivalent to

$$\begin{cases} o_{k_p,p} - y_p(k_p) \geq 0, \\ o_{k_p,p} - y_p(k_p) \leq 2Y_{\max}(1-\gamma_{k_p,p}), \\ o_{k_p,p} + y_p(k_p) \geq 0, \\ o_{k_p,p} + y_p(k_p) \leq 2Y_{\max}\gamma_{k_p,p}, \end{cases} \tag{3.28}$$

[1] In this chapter, the planned departure intervals are identical across lines. If different lines have different planned departure intervals, T_{ctrl} can be chosen as the least common multiple of the line-specific intervals after time discretization.

where $Y_{\max}$ denotes the maximum value of $y_p(k_p)$.

Transformation property 3.2: Based on *Transformation property 3.1*, (3.11) is equivalent to $\eta_{k_p,p} = \begin{cases} 1, & \text{if } o_{k_p,p} > 0; \\ 0, & \text{otherwise,} \end{cases}$, which can be converted to

$$\begin{cases} o_{k_p,p} \le \eta_{k_p,p} O_{\max}, \\ o_{k_p,p} \ge \epsilon + (1 - \eta_{k_p,p})(O_{\min} - \epsilon). \end{cases} \tag{3.29}$$

where $O_{\max}$ and $O_{\min}$ are the minimum and maximum values of $o_{k_p,p}$, respectively, and ϵ is a sufficiently small number, typically representing machine precision.

Transformation property 3.3: If we define m_{a} and M_{a} as the minimum and maximum values of $a_{p'}(k_{p'})$, respectively, then following the transformation property in [17], (3.14) is equivalent to the following inequalities

$$\begin{cases} a_{p'}(k_{p'}) - a_p(k_p) \le \left(1 - \xi_{k_{p'},k_p,p',p}\right)\left(M_{\mathrm{a}} - a_p(k_p)\right), \\ a_{p'}(k_{p'}) - a_p(k_p) \ge \varepsilon + \xi_{k_p,k_{p'},p,p'}\left(m_{\mathrm{a}} - a_p(k_p) - \varepsilon\right). \end{cases} \tag{3.30}$$

Based on the transformations described above, we can convert problem (3.27) into an MINLP problem. The nonlinearity in the MINLP arises from the nonlinear objective function. The integer variables in this problem encompass the train composition variables ($y_p(k_p)$ and $\ell_p(k_p)$), the train ordering variable ($\xi_{k_p,k_{p'},p,p'}$), and auxiliary binary variables ($\gamma_{k_p,p}$ and $\eta_{k_p,p}$).

For compactness, we rewrite the resulting MINLP problem in the following form:

$$\min_{\boldsymbol{x}(\kappa_0),\boldsymbol{u}(\kappa_0),\boldsymbol{\delta}(\kappa_0)} J(\kappa_0) := \sum_{\kappa=\kappa_0}^{\kappa_0+N-1} L(x(\kappa), u(\kappa), \delta(\kappa)) \tag{3.31a}$$

$$\text{s.t.} \quad x(\kappa+1) = A_\kappa x(\kappa) + B_{1,\kappa} u(\kappa) + B_{2,\kappa}\delta(\kappa), \tag{3.31b}$$

$$D_{3,\kappa} x(\kappa) + D_{1,\kappa} u(\kappa) + D_{2,\kappa}\delta(\kappa) \le D_{4,\kappa}, \tag{3.31c}$$

$$\kappa = \kappa_0, \cdots, \kappa_0 + N - 1,$$

where N is the total number of control steps, $x(\kappa)$, $u(\kappa)$, and $\delta(\kappa)$ collect all the independent variables, continuous decision variables, and discrete decision variables for control step κ, respectively, and $\boldsymbol{x}(\kappa_0) = [x^\top(\kappa_0), x^\top(\kappa_0+1), \ldots, x^\top(\kappa_0+N-1)]^\top$, $\boldsymbol{u}(\kappa_0) = [u^\top(\kappa_0), u^\top(\kappa_0+1), \ldots, u^\top(\kappa_0+N-1)]^\top$, and $\boldsymbol{\delta}(\kappa_0) = [\delta^\top(\kappa_0), \delta^\top(\kappa_0+1), \ldots, \delta^\top(\kappa_0+N-1)]^\top$. In (3.31), $L(x(\kappa), u(\kappa), \delta(\kappa))$ represents the nonlinear objective function for control step κ, (3.31b) collects all equality constraints, and (3.31c) collects all inequality constraints.

The MPC update index κ is event-driven. At step κ, the state contains depot inventory and train units availability, and circulation/depot constraints are enforced over the prediction horizon. Thus, the impact of a decision is propagated to later services via train units availability, and the transition from κ to $\kappa+1$ depends only on the current state and decision. Solving (3.31) leads to a series of continuous decision variables and discrete decision variables, and only the decision variables at control step κ_0 are applied. At the next step, the prediction time window is shifted for one step, and a new optimization is formulated. The framework is depicted in Fig. 3.4.

Lemma 3.2. (Recursive Feasibility)*: If problem (3.31) is feasible at control step κ with initial state $x(\kappa)$, then problem (3.31) is also feasible at control step $\kappa+1$.*

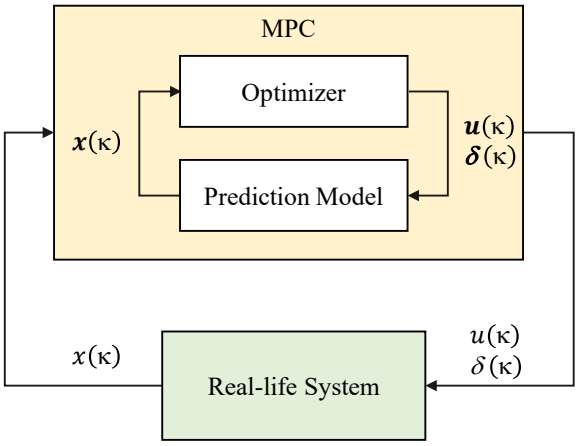


Figure 3.4: Model predictive control for real-time train rescheduling.

Proof. The proof relies on finding a feasible solution for control step $\kappa+1$. Recall that the planning time window at each platform is divided into several intervals of equal length, with a train service departing from the platform at each interval according to (3.1). In general, there are two types of depots: (i) depots connected to the terminal platform (e.g., Platform 1 in Fig. 3.2), and (ii) depots connected to intermediate platforms (e.g., Platform 2 in Fig. 3.2).

i) For a depot connected to the terminal platform: if the problem (3.31) is feasible at control step κ, then at each control step, a train service returns to the depot from the opposite direction of the line. In this context, the new train service departing from the terminal platform at control step $\kappa+1$ can directly utilize the train units by performing a turnaround from the opposite direction of the line following (3.7). Moreover, utilizing one train unit when performing the turnaround action is always feasible.

ii) For any depot connected to an intermediate platform: a feasible solution is obtained by maintaining the composition of each train service the same as it was at control step κ. □

Remark 3.3. *(Complexity Analysis).* There are three categories of variables in (3.31), i.e., continuous variables, binary variables, and integer variables (excluding binaries). The total number of variables and constraints is listed in Table 3.6, where $\mathscr{P}$ and $\mathscr{Z}$ are the set of platforms and depots, respectively, and $|\cdot|$ denotes the cardinality of a set.

3.4.3. MILP-BASED MPC FOR REAL-TIME TRAIN RESCHEDULING

In Section 3.4.1, a nonlinear objective function has been defined in (3.25) to calculate the passenger delays. This nonlinear objective function results in the MINLP-based MPC approach formulated in Section 3.4.2. In general, the nonlinear term significantly increases the computational burden. In what follows, we simplify the nonlinear objective function to reduce the computational burden.

Table 3.6: Numbers of variables and constraints in problem (3.31)

Variables or constraints	Maximum possible total number
Continuous variables	$(12\cdot\lvert\mathscr{P}\rvert+4\cdot\lvert\mathscr{Z}\rvert)\cdot N$
Binary variables	$(6+N)\cdot\lvert\mathscr{Z}\rvert\cdot N$
Integer variables (excluding binaries)	$2\cdot N\cdot\lvert\mathscr{Z}\rvert$
Constraints	$(33+N)\cdot N\cdot\lvert\mathscr{Z}\rvert$

Since we divide the planning time window as indicated in Fig. 3.3 and the actual departure time $d_p(k_p)$ is constrained by $d_p^{\text{pre}}(k_p)\le d_p(k_p)<d_p^{\text{pre}}(k_p+1)$, by using the upper bound and lower bound of $d_p(k_p)$ we approximate the nonlinear objective function (3.25) by

$$\begin{aligned}J_p^{\text{pass}}(k_p)\approx w_3 n_p(k_p)\left(d_p^{\text{pre}}(k_p+1)-d_p^{\text{pre}}(k_p)\right)+\\ +n_p^{\text{after}}(k_p)\left(d_p^{\text{pre}}(k_p+1)-d_p^{\text{pre}}(k_p)\right),\end{aligned}\tag{3.32}$$

where w_3 is a weight used to balance the approximated errors. In particular, w_3 can be defined as

$$w_3=\frac{1}{\sum_{p\in\mathscr{P}}\lvert\mathscr{I}_p\rvert}\sum_{p\in\mathscr{P}}\sum_{k_p\in\mathscr{I}_p}\frac{\bar{d}_p(k_p)-d_p^{\text{pre}}(k_p)}{d_p^{\text{pre}}(k_p+1)-\bar{d}_p(k_p)},\tag{3.33}$$

where $\lvert\mathscr{I}_p\rvert$ represents the cardinality of $\mathscr{I}_p$, and $\bar{d}_p(k_p)$ represents the average value of $d_p(k_p)$ from historical data.

Other settings are identical to those of Section 3.4.2. With this approximation, a linear objective function (3.32) is obtained, and due to the objective function and constraints being all linear, we obtain an MILP-based MPC approach for real-time train rescheduling. Since Section 3.4.3 modifies only the objective function and keeps the constraint set identical to Section 3.4.2, *Lemma 3.2 (recursive feasibility)* applies to the MILP-based MPC as well.

3.5. LEARNING-BASED MPC FOR TRAIN RESCHEDULING

For the MINLP-based MPC and MILP-based MPC introduced in Section 3.4, a mixed-integer optimization problem must be solved at each control step. The presence of integer variables significantly increases the computational complexity, which often renders these approaches unsuitable for real-time implementation. To address this computational issue in a different context, [29] proposed a learning-based approach with presolve techniques (PRISM) to solve mixed-integer programs, in which an LSTM network generates the integer decision variables, followed by the solution of a convex optimization problem to compute the continuous variables. Motivated by this framework, we develop a learning-based MPC approach with presolve techniques tailored to the train rescheduling problem with flexible train composition. In the proposed method, the presolve techniques and an ensemble of neural networks composed of feedforward layers and an LSTM network are employed to assign the discrete variables of the MPC problem. As a result, only a continuous nonlinear (or linear) optimization problem needs to be solved to compute

the continuous variables used to generate the updated timetable. The main differences between the proposed approach and PRISM lie in the design of the presolve techniques, the use of an neural network ensemble to improve feasibility, and the offline training procedure tailored to the train rescheduling problem.

3.5.1. PRESOLVE TECHNIQUES

Presolve techniques streamline optimization processes by pruning a subset of decision variables with values predetermined by other coupled variables and constraints, setting them to predefined values. In this chapter, we develop the following presolve techniques.

Presolve technique 3.1: As we adjust the departure time of train service k_p at platform p according to $d_p^{\text{pre}}(k_p) \le d_p(k_p) < d_p^{\text{pre}}(k_p+1)$, the departure order between some trains has already been determined: If $d_p^{\text{pre}}(k_p) \ge d_{p'}^{\text{pre}}(k_{p'}+1) + t_{p'}^{\text{roll}}$, then $\xi_{k_p,k_{p'},p,p'} = 1$. If $d_p^{\text{pre}}(k_p+1) \le d_{p'}^{\text{pre}}(k_{p'}) + t_{p'}^{\text{roll}}$, then $\xi_{k_p,k_{p'},p,p'} = 0$.

Presolve technique 3.2: According to the definition of $\xi_{k_p,k_{p'},p,p'}$ in (3.14), the order of trains at the same platform should be kept consistent, i.e., $\xi_{k_p+1,k_{p'},p,p'} \ge \xi_{k_p,k_{p'},p,p'}$.

Presolve technique 3.3: The train composition cannot be changed at a station that is not linked with a depot: If $\sigma_p = 0$, then $y_p(k_p) = 0$.

Presolve technique 3.4: Let $t_0 = \kappa_0 T$ represent the current time. If train service k_p has already departed from station p at time t_0, the composition cannot be changed at that station: If $d_p(k_p) \le t_0$, $y_p(k_p) = y_p^*(k_p)$, $\forall\, p \in \mathscr{P}$, where $y_p^*(k_p)$ represents the value of $y_p(k_p)$ obtained before train k departs from platform p and $\mathscr{P}$ denotes the set of all platforms of the line.

3.5.2. LEARNING SETTING

The environment of the learning-based MPC algorithm includes the system and an MPC optimization problem. The state and variables that interact with the environment are defined as follows:

State $\boldsymbol{s}(\kappa) \in \mathscr{S}$: The system state is defined to include all information required to characterize the system at control step κ and to serve as input to the neural network. Accordingly, the state is given by

$$\boldsymbol{s}(\kappa) = [\boldsymbol{n}^\top(\kappa), \boldsymbol{\rho}^\top(\kappa), \boldsymbol{N}^\top(\kappa)]^\top, \tag{3.34}$$

where $\boldsymbol{n}(\kappa)$ includes the variables $n_p(k_p)$ for each train service k_p at its corresponding platform p with $\kappa T \in [d_p^{\text{pre}}(k_p-1), d_p^{\text{pre}}(k_p))$, $\boldsymbol{\rho}(\kappa)$ collects the passenger demands $\rho_p(k_p)$ for all train services k_p departing from all platforms p from time $t = \kappa T$ to the end of the prediction time window, and $\boldsymbol{N}(\kappa)$ collects the number of available trains for all depots at time $t = \kappa T$.

Discrete variables: The variable $\boldsymbol{\delta}(\kappa) \in A$ represents the discrete variables of the MPC optimization problem (3.31) at control step κ.

Continuous variables: The variable $\boldsymbol{u}(\kappa) \in U$ denotes the continuous variables of the MPC optimization problem (3.31) at control step κ.

3.5.3. DECOUPLING THE DECISION VARIABLES

The learning problem is formulated as a multi-class classification task, where each feasible combination of discrete control actions encountered during data acquisition is mapped to a unique class label. In the forward pass, one class label is predicted for each time step of the prediction horizon, all drawn from a shared label set defined by the feasible combinations observed in the training data. This formulation enables the a priori exclusion of infeasible, suboptimal, or internally inconsistent discrete action combinations, as only labels corresponding to admissible MPC solutions are included in the label set.

The main objective of the learning process is to identify a parametric mapping

$$\phi_\theta : \mathscr{S} \to \mathscr{L}^N,$$

with parameter vector $\theta \in \mathbb{R}^{n_\theta}$ that maps the system state $\boldsymbol{s}(\kappa) \in \mathscr{S}$ to a sequence of labels, one for each time step of the prediction horizon, all drawn from a shared label set $\mathscr{L}$. The output label is subsequently decoded into a corresponding sequence of discrete decision variables. This parametric model approximates the behavior of the MPC policy in assigning the discrete variables over the prediction horizon $\delta(\kappa + k)$, for $k = 0, \ldots, N-1$, which includes the train composition and ordering decisions.

To this end, supervised learning is employed to adjust the parameter vector θ, where the model $\phi_\theta(\cdot)$ is trained offline using a labeled set of system states and corresponding discrete decisions from simulations of the MPC policy. Under this strategy, the solution of the MINLPs or MILPs, as in (3.31), is required only in the offline phase to generate the dataset for training. This design explicitly shifts the combinatorial complexity associated with the discrete variables from the online evaluation to the offline training phase. During online evaluation, the trained model $\phi_\theta(\cdot)$ is used to assign the discrete variables $\boldsymbol{\delta}(\kappa_0)$. Then the continuous decision variable $\boldsymbol{u}(\kappa_0)$, which is used to determine the departure and arrival times, can be determined by the solution of the following nonlinear optimization problem:

$$\begin{aligned}
&\min_{\boldsymbol{x}(\kappa_0), \boldsymbol{u}(\kappa_0)} J(\kappa_0, \boldsymbol{\delta}(\kappa_0)) := \sum_{\kappa=\kappa_0}^{\kappa_0+N-1} L(x(\kappa), u(\kappa), \delta(\kappa)) \\
\text{s.t.} \quad & [\delta^\top(\kappa_0), \delta^\top(\kappa_0+1), \ldots, \delta^\top(\kappa_0+N-1)]^\top = \boldsymbol{\delta}(\kappa_0), \\
& x(\kappa+1) = A_\kappa x(\kappa) + B_{1,\kappa} u(\kappa) + B_{2,\kappa} \delta(\kappa), \\
& D_{3,\kappa} x(\kappa) + D_{1,\kappa} u(\kappa) + D_{2,\kappa} \delta(\kappa) \le D_{4,\kappa}, \\
& \kappa = \kappa_0, \cdots, \kappa_0 + N - 1,
\end{aligned} \tag{3.35}$$

where the variable $\boldsymbol{\delta}(\kappa_0)$ has been fixed and it is no longer an optimization variable. Note that a linear formulation is possible by using the linearized cost in (3.32) to compose the objective function.

3.5.4. OFFLINE TRAINING

In the following, the data acquisition procedure, the architecture of the parametric model, and the training process are described.

For diverse realizations of the initial system state $\boldsymbol{s}(\kappa)$, the training dataset is obtained from solutions of the MINLP (or MILP) formulation as in (3.31) of the MPC controller. Let

$$\mathscr{D} = \{(\boldsymbol{s}_i,\ C_i)\}_{i=1}^{N_{\text{data}}}$$

denote the resulting training dataset, where $\boldsymbol{s}_i \in S$ represents the system state at sample i, and $C_i \in \mathscr{L}^N$ is the corresponding sequence of labels, denoted as $\{c_i^k\}_{k=0}^{N-1}$, associated with the discrete variables of the solution of the MPC problem. To enhance robustness regarding unobserved operating conditions, the dataset is further augmented with noise, capturing variability in passenger demand and delay profiles. In this learning framework, the MINLP (or MILP) formulation of the MPC controller acts as an oracle that determines the discrete decision variables associated with each state. Before training, the dataset is partitioned into three disjoint subsets, $\mathscr{D} = \mathscr{D}_{\text{train}} \cup \mathscr{D}_{\text{val}} \cup \mathscr{D}_{\text{test}}$, which are used for training, validation, and testing, respectively.

In practice, the train schedules across consecutive time intervals are interdependent due to the headway relation between trains and the physical connections between stations. The neural network should be able to capture and retain essential information over sequences of time intervals. Therefore, a long short-term memory (LSTM) network [29], [65] is selected as the parametric model $\phi_\theta(\cdot)$. As a deep recurrent neural network (RNN), the LSTM architecture enables the network to remember the dynamic interdependencies within train schedules, ensuring effective adaptation and learning in response to evolving temporal dynamics.

The architecture of the neural network representing the parametric model $\phi_\theta(\cdot)$ is illustrated in Fig. 3.5. The network inputs consist of the pre-processed current system state, represented as $g_k(\boldsymbol{s}(\kappa))$, where k indexes the time step over the prediction horizon. Each input vector is first mapped into a latent representation through a feedforward layer, represented by FC_1. The resulting latent features are then processed by an LSTM network, which captures temporal dependencies by maintaining a hidden state $h_k \in \mathbb{R}^{n_h}$ and a cell state $o_k \in \mathbb{R}^{n_h}$. Both states are initialized to zero at the beginning of the forward pass. Each time step of the prediction horizon is associated with a potentially distinct pre-processing function $g_k(\cdot)$, allowing considerable flexibility in the representation of temporal information. In this chapter, we employ zero-padding as the pre-processing strategy, with $g_0 = \boldsymbol{s}(\kappa)$ and $g_k = 0$ for $k = 1, \ldots, N-1$. This choice is motivated by its simplicity and empirical effectiveness. The rationale is to provide all relevant information regarding the system state and forecasted passenger demand at the initial time step and to allow the LSTM network to propagate this information forward through its internal states across the prediction horizon. Finally, the output of the LSTM network is passed through a second feedforward layer, FC_2, whose output neurons correspond to the possible combinations of discrete decision variables, yielding a categorical probability distribution over the shared label set at each time step after the application of the softmax activation function.

Each ground truth label $c_i^k \in \{0,1\}^{|\mathscr{L}|}$ corresponds to a discrete sequence and is represented using one-hot encoding, where $|\mathscr{L}|$ denotes the number of distinct discrete sequences in the shared label set. The same encoding is used to represent the outputs of the parametric model for each time step $\hat{c}_i^k = [\phi_\theta(\boldsymbol{s}_i)]_k \in \mathbb{R}^{|\mathscr{L}|}$, where the operator $[\cdot]_k$ extracts the k-th output of the neural network. Each $\hat{c}_i^k$ represents a categorical probability

distribution over the shared label set for time step k and state $\boldsymbol{s}_i$. The training procedure minimizes the mean squared error (MSE) between the predicted and target labels over the prediction horizon by stochastic gradient descent. Let $\mathcal{J} \subset \mathbb{N}^{N_{\text{batch}}}$ denote a mini-batch of size N_{batch} sampled from $\{1, \dots, N_{\text{data}}\}$. The loss function is defined as

$$L(\theta) = \sum_{j \in \mathcal{J}} \sum_{k=0}^{N-1} \| c_j^k - \hat{c}_j^k \|_2^2.$$

While cross-entropy loss is standard for classification tasks, MSE provides a smooth measure of the discrepancy between the predicted and target probability distributions. Furthermore, preliminary experiments indicated that MSE resulted in higher accuracy and more stable training in the considered setting.

The parametric model $\phi_\theta(\cdot)$ cannot guarantee the feasibility of the resulting continuous MPC problem by design. This constitutes a notable limitation of learning-based approaches, such as [29]. Unlike related approaches, we employ an ensemble of neural networks, denoted by $\{\phi_\theta^i\}_{i=1}^{N_{\text{ens}}}$, to improve feasibility. Although this strategy does not eliminate the inherent feasibility issue, it provides an efficient mechanism to mitigate it. If the ensemble of neural networks fails to provide feasible discrete variables, feasibility during online operation can be guaranteed via the heuristic strategy described in *Lemma 3.2*.

In the learning context, classification accuracy and performance are correlated; however, the ultimate objective of the learning-based framework is control performance, namely the minimization of passenger delays and operational costs. For this reason, the validation loss alone is not an adequate metric to evaluate the effectiveness of the trained models. Instead, states from the validation set $\mathcal{D}_{\text{val}}$ are used to evaluate the performance of the trained model in terms of average open-loop objective value achieved by the corresponding MPC problem. More specifically, for each state $\boldsymbol{s}_i \in \mathcal{D}_{\text{val}}$, the parametric model is evaluated as $\phi_\theta(\boldsymbol{s}_i)$, and its outputs are used to assign the discrete variables in the MINLP (or MILP). The resulting continuous MPC subproblem is solved, yielding the objective value $J(\kappa_i;\ \boldsymbol{s}_i,\ \phi_\theta(\boldsymbol{s}_i))$, where κ_i denotes the index associated with the system state $\boldsymbol{s}_i$. The average open-loop objective value of the trained model ϕ_θ over all $\boldsymbol{s}_i \in \mathcal{D}_{\text{val}}$ is represented by $\bar{J}_{\phi_\theta}$.

This performance metric is subsequently used in the hyperparameter optimization stage of the training pipeline. Multiple neural networks are trained using varying weight initializations and hyperparameters. Let $\Phi = \{\phi_{\theta_1}, \dots, \phi_{\theta_{N_{train}}}\}$ denote the trained neural networks obtained through the hyperparameter optimization procedure (e.g., grid-search, random search, and Bayesian optimization). The corresponding average open-loop objective values of the trained models $\{\bar{J}_{\phi_\theta^i}\}_{i=1}^{N_{\text{train}}}$ is then obtained and ordered. A subset $\Phi_{\text{ens}} \subseteq \Phi$ of the neural networks consisting of the N_{ens} models with the lowest average objective values is selected to compose the form the ensemble. In this manner, the ensemble of neural networks is obtained as a byproduct of the hyperparameter optimization process. Finally, the system states in $\mathcal{D}_{\text{test}}$ are used to assess the open-loop and closed-loop performances of the learning-based framework.

3.5.5. STATE REDUCTION APPROACH

This section introduces a state reduction approach for learning-based MPC for train rescheduling. Specifically, a multi-resolution state design is integrated into the control

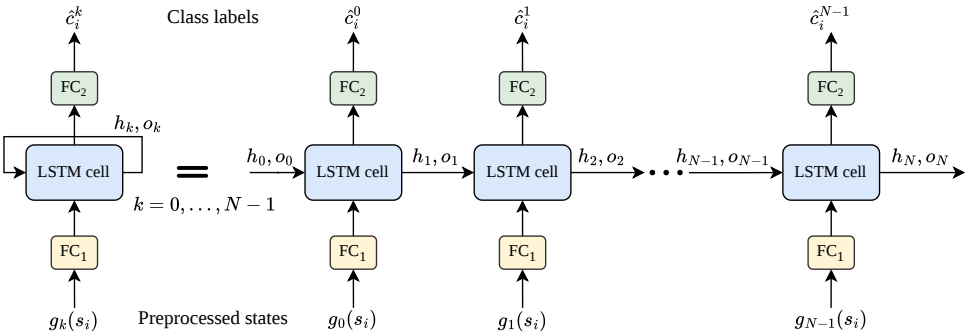


Figure 3.5: Unrolled LSTM network architecture with feedforward input (FC_1) and output (FC_2) layers. Each block represents an identical LSTM cell with shared parameters across time steps, and the network is unrolled over the prediction horizon to capture temporal dependencies in the discrete decision variables.

framework, where a low-resolution state space is utilized for learning, while the full state space is employed in MPC.

State dimensionality reduction in neural networks has several benefits, from training to inference. In a simpler state space, the neural network typically generalizes better, resulting in improved accuracy [33]. During training, the usage of computational resources can be lowered, as a lower-dimensional dataset occupies less memory. Alternatively, while maintaining the storage requirement identical, the number of points in the training set can be increased for a lower-dimensional state, which may enhance accuracy. Moreover, a lower-dimensional state space allows for a simpler neural network architecture since fewer neurons are required in each layer, allowing faster inference times. These benefits highlight the importance of proper design of the state space for learning. In what follows, we discuss a state reduction method that is tailored to the train rescheduling problem.

The core idea is to partition the vector $\boldsymbol{\rho}(\kappa)$ into a predetermined number N_s of segments, and then compute the average passenger flow for each segment. To illustrate this process, we consider the passenger flow at platform p over a number $M = H \cdot N_\mathrm{s}$ of train services $\boldsymbol{\rho}_p = [\rho_p(k_p), ..., \rho_p(k_p + M - 1)]$, where H is the length of the segments. Let $[\boldsymbol{\rho}_p]_j$ be the jth element of the vector $\boldsymbol{\rho}_p$ and $[\boldsymbol{\rho}_p]_{i:j}$ a slice of the vector $\boldsymbol{\rho}_p$ between the indices i and j provided that $i < j$. The state reduction procedure can be expressed as

$$[\boldsymbol{\rho}_p^{\text{reduced}}]_k = \text{mean}([\boldsymbol{\rho}_p]_{k\cdot H:(k+1)\cdot H}) \text{ for } k = 0, ..., N_\mathrm{s}, \tag{3.36}$$

where $\text{mean}([\boldsymbol{\rho}_p]_{i:j}) = \frac{1}{j-i}\sum_{q=i}^{j-1}[\boldsymbol{\rho}_p]_q$.

This operation reduces the number of elements in the original vector by a factor of H. The state reduction preserves the mean of the original vector, thereby ensuring no error in the computation of the total number of passengers in the prediction window. Hereafter, the collection of the passenger flow $\boldsymbol{\rho}_p^{\text{reduced}}$ across all platforms of a train line is denoted by $\boldsymbol{\rho}^{\text{reduced}}$.

The discrete decision variable $\boldsymbol{\delta}$, which represents the train composition of the train services over the prediction horizon, is relatively insensitive to fast changes in passenger

flow over time. In this chapter, we consider that train compositions can only be modified at depots located at the terminal stations of the line. Thus, once the composition of the train of a particular train service is established, it has a lasting effect on the optimization problem (3.35) as it cannot be changed on any of the intermediate platforms. Consequently, the variable $\boldsymbol{\delta}$ is more sensitive to long temporal patterns of the passenger flow rather than short ones. Therefore, we argue that a reduced passenger flow vector is more suitable for learning due to its lower dimensionality and its capacity to adequately capture the essential characteristics of the passenger flow required to estimate the optimal train composition. Hence, the state described in (3.34) is redefined for the classifier ϕ as

$$\boldsymbol{s}_{\text{reduced}}(\kappa) = [\boldsymbol{n}^{\mathsf{T}}(\kappa), (\boldsymbol{\rho}^{\text{reduced}}(\kappa))^{\mathsf{T}}(\kappa), \boldsymbol{N}^{\mathsf{T}}(\kappa)]^{\mathsf{T}}. \tag{3.37}$$

On the other hand, in the optimization problem (3.35), the unreduced passenger flow $\boldsymbol{\rho}$ is required to accurately model system dynamics. Furthermore, the continuous decision variable $\boldsymbol{u}$, including departure and arrival times, has a relatively faster effect on (3.35) and can react to faster changes in passenger flows. Hence, passenger flow $\boldsymbol{\rho}$ with the original resolution is more appropriate to be included in the MPC formulation of (3.35) that computes $\boldsymbol{u}$.

In essence, we advocate for the use of states with different levels of resolution for learning and optimization, leveraging the fact that the decision variables exhibit different sensitivities to changes in passenger flow. A low-resolution state is employed in learning to determine the discrete actions, while the original state representation is used in the MPC formulation to compute the continuous actions. In this manner, the state representation is tailored to the specific requirements of each component of the framework.

3.5.6. ONLINE EVALUATION

The outcome of the training process is a neural network ensemble $\Phi_{\text{ens}} = \{\phi_{\theta_1}, \dots, \phi_{\theta_{N_{\text{ens}}}}\}$. During online operation, the state of the urban rail transit system is measured and provided as input to the ensemble, which is evaluated sequentially. The $(i+1)$-th neural network is only evaluated if the i-th and all preceding networks fail to produce a feasible assignment of the discrete variables. While alternative strategies, such as parallel evaluation, are possible, sequential evaluation is adopted to reduce the average number of forward passes in real time. The ensemble size N_{ens} is fixed and relatively small, and the computational cost of each forward pass is negligible compared to that of solving the continuous MPC subproblem. As a result, even in the worst-case scenario where all neural networks are evaluated, the additional computational overhead remains compatible with real-time operation. Before implementing the discrete actions provided by the ensemble, the presolve techniques described in Section 3.5.1 are applied to prune infeasible discrete variables by fixing them to predetermined values. The resulting discrete assignments are then fixed in the MPC formulation, and the corresponding continuous (or linear) optimization problem (3.35) is solved to obtain the continuous decision variables, which are used to generate the updated timetable. By construction, feasibility also implies that all the constraints are satisfied and that the continuous decision variables $\boldsymbol{u}$ are optimal with respect to the choice of $\boldsymbol{\delta}$. Although the ensemble can significantly increase the likelihood of feasible discrete variables, feasibility cannot be guaranteed by design. If no feasible assignment is identified, the heuristic strategy in *Lemma 3.2* is applied to ensure

recursive feasibility. The online evaluation procedure is described in Alg. 3 and illustrated in Fig. 3.6.

Algorithm 3 Online Inference

Input: Neural network ensemble Φ_{ens}, system state $\boldsymbol{s}(\kappa)$, `state_reduction`
$s \leftarrow \boldsymbol{s}(\kappa)$
`solution_found` $\leftarrow$ `False`
if `state_reduction == True` **then**
    $s \leftarrow \boldsymbol{s}_{\text{reduced}}(\kappa)$ (via (3.36) and (3.37))
end if
for $\phi_{\theta_i} \in \Phi_{\text{ens}}$ **do**
    $\boldsymbol{\delta} \leftarrow \phi_{\theta_i}(s)$
    if (3.35) is feasible for ($\boldsymbol{s}$, $\boldsymbol{\delta}$) **then**
        Retrieve $\boldsymbol{u}$
        `solution_found` $\leftarrow$ `True`
        Break for loop
    end if
end for
if `solution_found == False` **then**
    Use Lemma 3.2 to find a feasible $\boldsymbol{\delta}$
    Get $\boldsymbol{u}$ by solving (3.35)
end if
Output: $\boldsymbol{\delta}$, $\boldsymbol{u}$

3.6. CASE STUDY

3.6.1. SIMULATION SETUP

In this section, we illustrate the proposed approaches based on real-life data from a network of three lines in the Beijing urban rail transit network. As shown in Fig. 3.7, the network includes 3 bi-directional lines with 45 stations. There are 3 transfer stations, i.e., Station ZXZ, Station XEQ, and Station HY, where passengers can transfer from one line to another. For each line, there is a depot connected with the starting station of the line, i.e., Station CPX for Changping Line, Station XZM for Line 13, and Station ZXZ for Line 8. The values of parameters for the case study are given in Table 3.7. The original timetable is generated based on the regular headway, regular dwell time, and average running times in Table 3.7. According to the definition, the length of a control step is the time interval between two consecutive train departures at a platform, i.e., the sum of the regular headway and the regular dwell time. The number of train units in the depot for each line has been selected as a random integer number with the value varying within the range given in Table 3.7. In practice, t_p^{comp} may vary across platforms. In this case study, we set $t_p^{\text{comp}} = 60$ s for all platforms, which does not affect the comparison since all methods use the same parameter setting.

The length between every two consecutive stations is openly accessible on the website

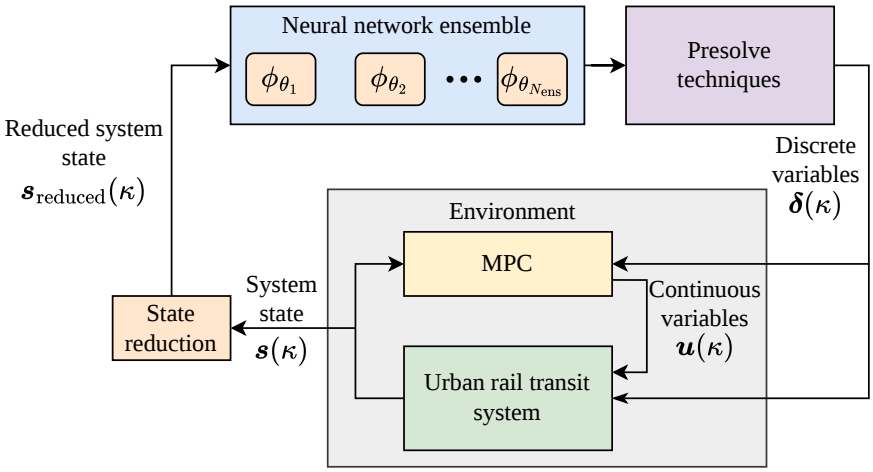


Figure 3.6: Closed-loop implementation of the proposed learning-based MPC framework. The system state is provided to the neural network ensemble to generate discrete decision variables, which are refined by presolve techniques and fixed in the MPC formulation. The resulting continuous optimization problem is solved to compute the continuous decision variables and update the timetable.

of Beijing Subway[2]. In the case study, the average running time and the average energy consumption of a train between every two consecutive stations are calculated using the method in [141] with the maximum acceleration of 0.75 m/s^2, the maximum deceleration of 0.7 m/s^2, and the cruising speed of 70 km/h, respectively. The sectional passenger demands are obtained based on real-life passenger flow data from the Beijing urban rail transit network, collected in January 2020. We have selected data from 6:00 AM to 10:00 PM for simulation, so the data contains both peak hours and off-peak hours. At interchange stations, the passengers include both passengers entering from outside the system and transfer passengers arriving from other lines. Hence, transfer impacts are reflected in the passenger-oriented results through the rescheduled departure times and the available capacity implied by train composition decisions. For training agents and simulation, we have generated passenger demands based on a Poisson distribution by using real-life passenger flow data as the expected value.

The simulations have been conducted using Python as a programming language, PyTorch as the machine learning library, and `gurobi` to solve optimization problems. Adam [77] is applied in the offline training process to minimize MSE, and dropout [126] is used to handle the over-fitting issue. Moreover, the experiments were conducted on a computing cluster with Intel XEON E5-6248R CPUs. The dataset consists of 96000 states and the corresponding optimal solutions, and it has been built using 60 CPU cores and 240GB of RAM in 24 hours. The training and hyperparameter tuning processes have been conducted using 144 CPU cores, 864GB of RAM, using more than 200000 iterations for 24 hours. The code used to generate the results in this section is available online[3].

[2] `https://www.bjsubway.com/station/zjgls/`
[3] `https://github.com/fabcaio/railwaynet_learning/`

Table 3.7: Main parameters for the case study

Parameter	Symbol	Value
Regular headway	h_p^{regular}	180 s
Minimum headway	$h_p^{\min}$	120 s
Regular dwell time	τ_p^{regular}	60 s
Minimum dwell time	$\tau_p^{\min}$	30 s
Average turnaround time	r_p^{turn}	52.9 s
Minimum running time	$r_p^{\min}$	$0.8 \cdot r_p^{\text{avrg}}$
Maximum running time	$r_p^{\max}$	$1.2 \cdot r_p^{\text{avrg}}$
Time for changing train composition	t_p^{comp}	60 s
Time for rolling stock circulation	$t_{p,p'}^{\text{roll}}$	240 s
Transfer rate at a transfer station	$\beta_{q,p}$	10%
Capacity of a train unit	$C_{\max}$	400 persons
Regular train composition of a train service	ℓ_p^{regular}	2 train units
Minimum number of train units included in a train service	$\ell_p^{\min}$	1 train unit
Maximum number of train units included in a train service	$\ell_p^{\max}$	4 train units
Weighted term	w_1	10^{-4}
Weighted term	w_2	10^{-1}
Weighted term	w_3	10^{-1}
Number of train units for Changping Line	N_z^{train}	[55, 75]
Number of train units for Line 13	N_z^{train}	[70, 90]
Number of train units for Line 8	N_z^{train}	[60, 80]
Prediction horizon of MPC	N	40

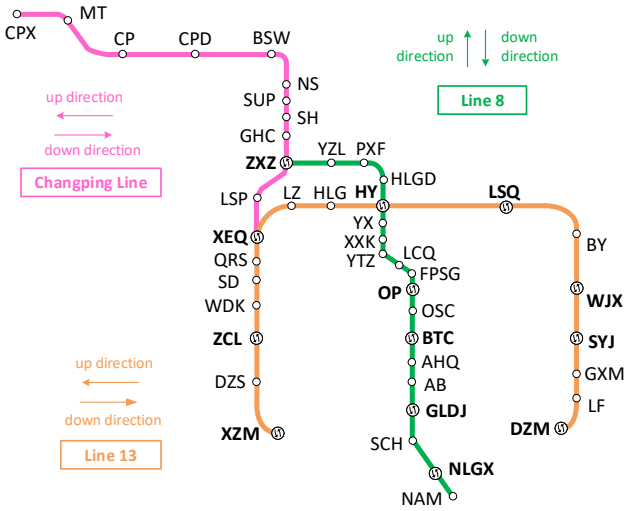


Figure 3.7: The layout of the considered urban rail transit network (with 3 lines).

To reduce the solution time of the MINLP solver without significantly compromising optimality, a simple early termination criterion was employed: if the optimality gap does not decrease by 0.5% within 10 seconds, the solution process terminates, and the solver outputs the best solution found. Moreover, as the MILP-based approach typically has a significantly shorter solution time than the MINLP-based approach, the integer variables generated by the MILP-based approach are used as a warm-start rule for the MINLP-based approach.

In this section, the developed train rescheduling approaches, i.e., MINLP-based MPC, warm-start-MINLP-based MPC, MILP-based MPC, learning-NLP-based MPC, and learning-LP-based MPC, are evaluated. As defined in Section 3.3, the length of a control step is 240 s. Hence, to ensure that a solution can be obtained for each step, we set the maximum solution time for each approach as 240 s. In addition, as a longer solution time typically yields a better objective function value, we use the MINLP approach with a warm-start and a longer maximum solution time, i.e., 600 s, as a benchmark to evaluate the performance of the developed approaches.

To improve the overall feasibility of the learning-based approach, the ensemble of 15 neural networks is trained separately using different weight initializations and hyperparameters. Each neural network in the ensemble is evaluated sequentially, as described in the previous section, until a feasible assignment of the discrete variables is identified. If a feasible solution is not found, then *Lemma 3.2* is used to ensure feasibility.

Hyperparameter optimization is performed using a grid search over a predefined set of candidate values for the hidden size, dropout rate, learning rate scheduling, and output masking. While more sophisticated hyperparameter optimization techniques, such as Bayesian methods, may perform better in large search spaces, grid search is an effective and reliable approach for the low-dimensional search space considered in this chapter.

The ensemble is a byproduct of the hyperparameter optimization procedure and is composed of neural networks trained over the following hyperparameter configurations: hidden size {512, 1024}, dropout rate {0, 0.5}, learning rate scheduling {ON, OFF}, and output masking {ON, OFF}. The input size of the input feedforward layer is constant and corresponds to the state dimension, while its output size matches the hidden size of the LSTM network. The output size of the output feedforward layer is constant and is identical to the shared label set cardinality, and its input size is equal to the hidden size.

Dropout is applied to the weights that connect the input feedforward layer to the LSTM network and to the weights that connect the output of the LSTM to the output feedforward layer. Learning rate scheduling, when enabled, halves the learning rate whenever a plateau in the validation loss is detected, defined as a lack of improvement over five consecutive epochs. Output masking, when enabled, prevents the neural network from assigning labels that were not observed during the data acquisition phase.

3.6.2. SIMULATION RESULTS WITHOUT STATE REDUCTION

Table 3.8: Comparison of the open-loop performance of different approaches for real-time train rescheduling.

Approach	Warm-start	Optimality gap (%)				CPU time (s)				Feasibility rate
		mean	std	min	max	mean	std	min	max	
Benchmark	yes	-	-	-	-	222.18	280.53	3.67	600.10	100%
MINLP	no	7.22	14.14	0	100.81	239.84	5.87	52.47	240.14	100%
Warm-start MINLP	yes	0.04	3.14	-0.24	12.96	96.71	111.35	3.58	240.10	100%
MILP	no	0.47	1.11	-33.73	1.54	8.77	30.96	0.37	240.01	100%
Learning + NLP	no	-0.11	1.18	-34.28	3.48	6.89	8.27	1.80	112.22	98.94%
Learning + LP	no	0.22	1.10	-33.42	1.73	0.13	0.01	0.11	0.25	98.55%

We conducted simulations to train the learning algorithm, and the training process for one of the neural networks in the ensemble is shown in Fig. 3.8, where a 1000-step moving average is applied to smooth the loss curves. As shown in Fig. 3.8, the loss decreases rapidly during the first 10^5 iterations and then gradually converges over the remaining training iterations.

We perform simulations for both open-loop optimization, where the MPC optimization problem is solved for one step, and closed-loop MPC, involving real-time MPC optimization over 30 steps. For open-loop optimization, we have performed simulations using the developed learning-based approaches on 10 separate cores for 24 hours (resulting in 93 scenarios). For comparison, simulations have also been performed with the benchmark approach, the MINLP approach, the warm-start MINLP approach, and the MILP approach, where the benchmark approach corresponds to the MINLP approach with warm-start and a maximum solution time of 600 s.

To compare the performance of the proposed learning-based approach with that of the MINLP and MILP formulations of the MPC controller, three performance metrics are analyzed, namely optimality gap, CPU time, and feasibility rate. The optimality gap is defined as the relative difference between the average open-loop objective value of a given approach and that of a benchmark method, i.e.,

$$\text{optimality gap} = \frac{J - J_{\text{ben}}}{J_{\text{ben}}} \cdot 100,$$

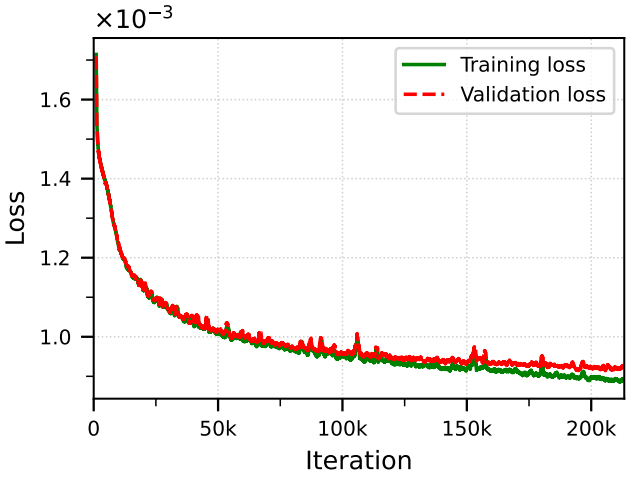


Figure 3.8: Training and validation loss as a function of the mini-batch gradient descent iterations. The curves represent moving-average smoothed values.

where J and J_{ben} denote the average open-loop objective values achieved by the approach of interest and the benchmark approach, respectively. The benchmark method is defined as the MINLP formulation of the MPC controller solved with a maximum time limit of 10 minutes, which exceeds the control step duration of 4 minutes. As a result, the benchmark solution cannot be guaranteed to be globally optimal. Nevertheless, the extended computation time allows the solver, in principle, to identify an improved local minimum. The feasibility rate is defined as the percentage of instances in which the neural network ensemble outputs discrete actions that yield a feasible continuous MPC subproblem.

The optimality gap, CPU time, and feasibility rate obtained from the simulations are given in Table 3.8. The MINLP approach without warm-start has the worst optimality gap and the worst CPU times. In general, the nonlinear objective function and the presence of integer decision variables significantly influence the performance of MINLP. By applying the warm start, the optimality gap and the solution time are reduced; however, the maximum solution time and the average solution time of the warm-start MINLP approach are still large with values of 240.10 s and 96.71 s, respectively. By approximating the nonlinear objective function, the solution time of the MILP approach is further reduced without significant performance degradation. However, the MILP also reached the maximum allowed solution time in some cases, i.e., the maximum CPU time of MILP in Table 3.8 still reached 240 s.

In the open-loop optimization, the developed learning-based NLP approach (Learning + NLP) and learning-based LP approach (Learning + LP) achieve comparable performance with an average optimality gap of -0.11% and 0.22% among the feasible cases, while significantly reducing the solution time to an average of 6.89 s and 0.13 s, respectively. Furthermore, by approximating the nonlinear objective function, the learning-based LP approach further reduces the solution time, enabling the optimized timetable to be ob-

tained in under 1 second. Both the learning-based NLP and LP approaches demonstrate high feasibility rates, at 98.94% and 98.55%, respectively. At first glance, the occurrence of negative optimality gaps may seem counterintuitive. However, the benchmark solution does not guarantee global optimality due to the imposed solver time limit. Consequently, negative optimality gaps may occur when alternative approaches achieve lower objective values than the benchmark.

During online operation, the system state may deviate from the distribution represented in the training dataset due to time-varying passenger demands and unforeseen disruptions, which can negatively affect the performance of the neural network ensemble. To improve generalization and robustness to distribution shift, data augmentation, dropout, and an ensemble of neural networks are employed during training. To deal with long-term shifts in passenger behavior, the ensemble can be periodically retrained using more recent data, and imitation learning can also be used for fine-tuning. Under severe disruption scenarios that substantially differ from the training conditions, the performance of the ensemble may deteriorate. In such cases, feasible suboptimal actions provided by *Lemma 3.2* or backup controllers can be applied to ensure recursive feasibility.

To further demonstrate the performance of the learning-based MPC approach, we conduct closed-loop control tests on 10 separate cores over 24 hours, with each episode consisting of 30 control steps. If the integer variables generated by the learning agents are infeasible, the heuristic described in *Lemma 3.2* is applied to obtain the train decomposition and to generate a feasible timetable. The simulation results are presented in Table 3.9. It can be observed that after applying the heuristic rule described in *Lemma 3.2*, the feasibility rate reached 100%. Compared to the warm-start MINLP and MILP approaches, the optimality of learning-based approaches is reduced in some cases because heuristic rules typically generate suboptimal solutions. In the closed-loop case, the maximum solution time for traditional approaches, i.e., warm-start MINLP and MILP, reaches 240 s, making them unsuitable for real-life applications where operators require an optimized solution as soon as possible for real-time timetable rescheduling. In contrast, the solution times for learning-based NLP and LP approaches are significantly lower, with average solution times of 6.50 s and 0.13 s, respectively. Table 3.9 indicates that learning-based approaches significantly enhance solution efficiency, with an acceptable optimality sacrifice of approximately 5%.

Table 3.9: Comparison of the closed-loop performance of different approaches for real-time train rescheduling.

Approach	Warm-start	Optimality gap (%)				CPU time (s)				Feasibility rate
		mean	std	min	max	mean	std	min	max	
Benchmark	yes	-	-	-	-	227.18	275.52	0.05	600.12	100%
Warm-start MINLP	yes	-0.22	3.42	-60.95	13.03	51.36	62.62	0.05	240.06	100%
MILP	no	0.46	1.53	-35.61	21.18	6.83	29.82	0.06	240.05	100%
Learning + NLP	no	4.92	10.21	-11.02	93.43	6.50	6.28	1.84	91.20	100%
Learning + LP	no	5.50	11.75	-9.99	126.71	0.15	0.03	0.11	0.31	100%

As an illustrative example, Fig. 3.9 presents the simulation results for one episode, showing the objective function values at each step for the benchmark, warm-start MINLP, MILP, learning-based NLP, and learning-based LP approaches. The episode begins at

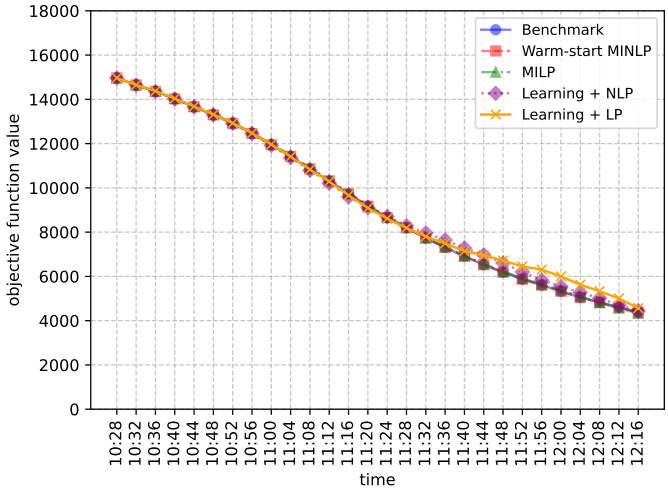


Figure 3.9: Objective function value at each MPC step for one episode.

10:30 and spans 30 MPC steps. Fig. 3.9 indicates that the developed learning-based approaches achieve comparable performance concerning objective function values. The timetables for the Changping Line, covering from Station CPX to Station XEQ, are depicted in Figs. 3.10–3.14 for the benchmark approach, the warm-start MINLP, MILP, learning-based NLP, and learning-based LP approaches, respectively, where the line width indicates the number of train units included in each train service. From Figs. 3.10–3.14, it can be observed that all approaches deploy long trains from 10:30 to 10:50 to transport passengers. After 10:50, the benchmark, warm-start MINLP, and MILP approaches continue using long trains, whereas the learning-based NLP approach initially uses short trains before switching to long trains, resulting in a slight difference in the objective function value shown in Fig. 3.9. In contrast, the learning-based LP approach continues using short trains until 11:40, resulting in the worst performance in Fig. 3.9.

In general, the rail operator expects to generate a timetable as soon as possible to ensure an efficient rolling-stock circulation that satisfies depot/turnaround constraints and limits unnecessary coupling/decoupling operations, thereby enabling effective management of real-time traffic conditions and passenger flows. The simulation results indicate that the developed learning-based MPC approach significantly reduces solution times with an acceptable trade-off in objective function performance. The developed learning-based MPC approach can generate timetables in real time with the train composition and rolling stock circulation plan generated by learning, and the detailed departure and arrival times generated by nonlinear or linear programming. It should be noted that solving MINLP or MILP problems typically requires commercial solvers, such as `gurobi` and `cplex`, to generate solutions in a computationally efficient way. By generating integer variables with well-trained learning agents at each MPC step, the resulting continuous linear or nonlinear programming problems can be efficiently solved using various available

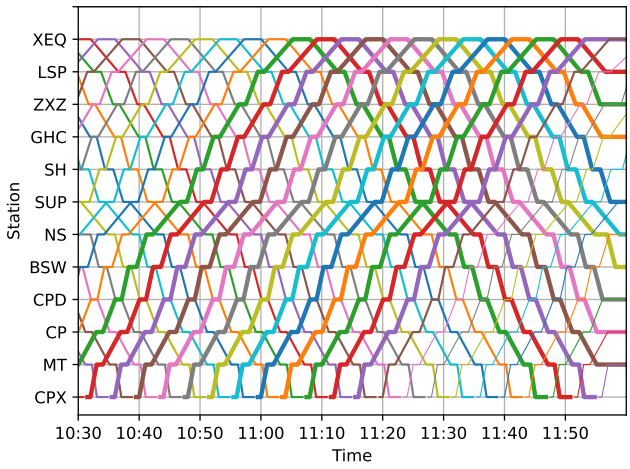


Figure 3.10: Timetable for Changping Line of the benchmark approach

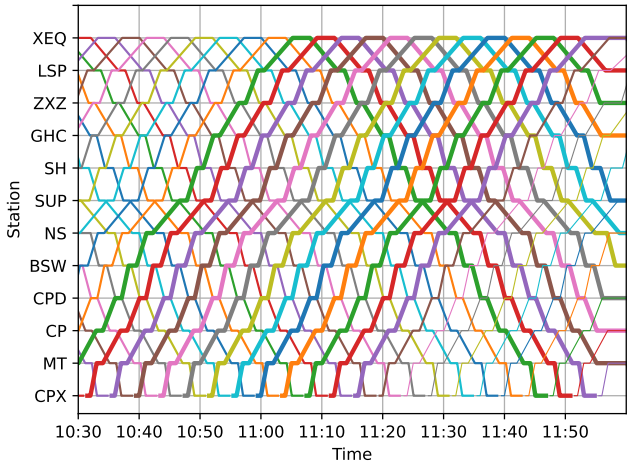


Figure 3.11: Timetable for Changping Line of the warm-start MINLP approach

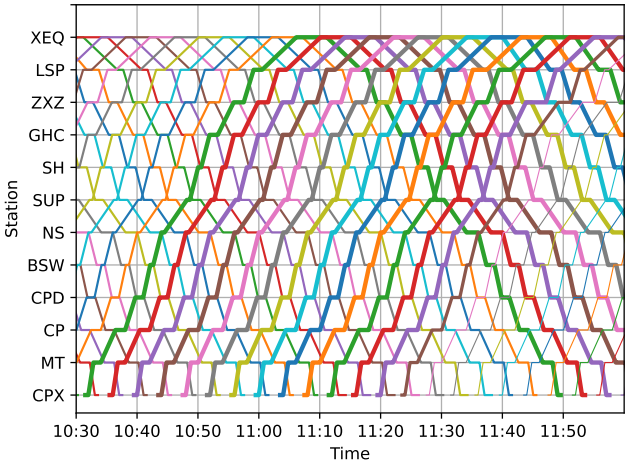


Figure 3.12: Timetable for Changping Line of the MILP approach

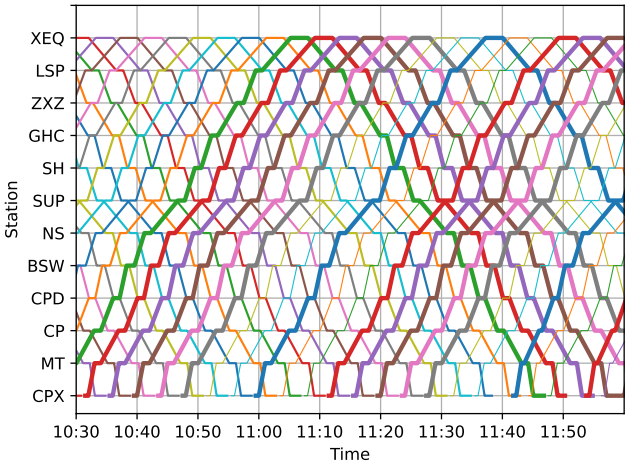


Figure 3.13: Timetable for Changping Line of the learning-based NLP approach

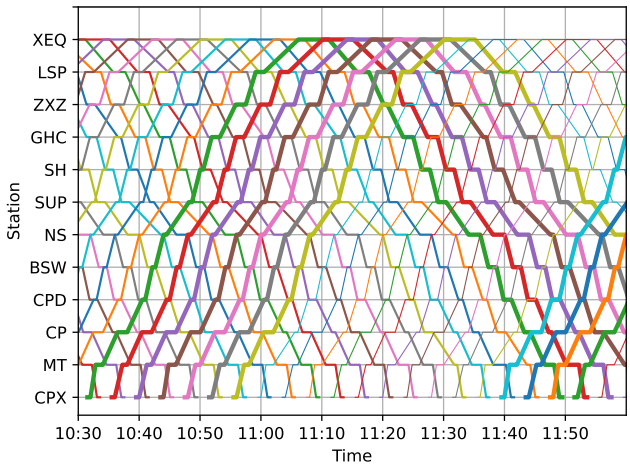


Figure 3.14: Timetable for Changping Line of the learning-based LP approach

solvers and algorithms.

3.6.3. SIMULATION RESULTS WITH STATE REDUCTION

This section reports the results of more extensive simulations of the case study using the state reduction procedure described in Section 3.5.5. Compared to the results presented in the previous section, the simulations are conducted with a longer maximum simulation length (80, previously 30), and a larger dataset (175 trajectories and 7864 samples, previously 93 trajectories and 2306 samples). The main purpose of this experiment is to assess the effect of the state reduction approach on the learning-based MPC method by comparing the results obtained without state reduction. The code and datasets used for this part of the results are publicly available[4].

The dataset characteristics for the full and reduced states, obtained offline, are represented in Table 3.10. In our case, we have the two originally contrasting goals of (i) improving accuracy by enlarging the dataset and (ii) reducing memory requirements, which is one of the computational bottlenecks for training. Both goals are achieved by applying the state reduced approach (3.36) with $N_s = 4$ and $H = 9$, which are adjusted via grid-search with validation data. Consequently, the state dimension is reduced from 4002 to 748. Moreover, the number of data points could be increased by a factor of 2.65, while decreasing the memory usage by a factor of 1.5. The datasets are then used for the training of 64 neural networks with varying hyperparameter configurations. Subsequently, the validation dataset is used to select a subset of 15 networks to form the neural network ensemble.

The comparison of the closed-loop operation of seven different methods averaged over 175 trajectories is shown in Table 3.11. The control time step and the MPC prediction

[4] https://github.com/fabcaio/railway_conf

Table 3.10: Dataset information.

State	State dimension	Number of points	Memory (GB)
Full	4002	94871	5.00
Reduced	748	252207	3.29

horizon are set to $T = 4$ minutes and $N_p = 40$, respectively. The initial state of each closed-loop simulation is randomly chosen within the operational range and it is consistent across the methods.

The maximum simulation length is chosen to be 80 time steps. In methods (I)–(III), both the discrete and continuous variables are determined by optimization. The benchmark (I) corresponds to the solution of (3.31) with a maximum solution time of 10 minutes, rendering the method unachievable since the MPC controller must find a solution within the control time step of 4 minutes. Hence, benchmark (I) is used solely to define the optimality gap, measured as the percentage difference between the performance of a given method and that of benchmark (I).

The methods (II) and (III) represent the solution of (3.31) with a maximum solution time of 4 minutes – equal to the control time step – and nonlinear and linear performance indices, respectively. In contrast, methods (IV)–(VII) employ the decoupling procedure described in Section 3.5.4, where the discrete variables are determined by an ensemble of neural networks, as described by Alg. 3 and Fig. 3.6, and the continuous variables are computed by the solution of the optimization problem (3.35) with a maximum solution time of 4 minutes. The approaches (IV) and (V) use the original state representation, while (VI) and (VII) employ the reduced state described in (3.37). Although methods (V) and (VII) use the linear performance index (3.32), their solutions are evaluated on the nonlinear performance index (3.25), ensuring consistency in the evaluation of all the methods.

As illustrated by Table 3.11, the optimality gaps of the learning-based approaches with full state (IV) and (V) are significantly reduced by the proposed approaches (VI) and (VII). More specifically, comparing (IV) to (VI) the optimality gap dropped from 13.06% to 0.25%, and comparing (V) to (VII) the optimality gap decreases from 12.82% to 0.00%, showing the significant effect of the state reduction procedure (3.36). The learning-based approaches (IV)–(VII) reduce the computation time considerably compared to the optimization-based approaches (I)–(III). Particularly, the learning-based approach (VII) with the linear performance index (3.32) has the best overall performance considering optimality and computation time. In methods (IV) and (VI), the NLP (3.35) is not solved to optimality, as an early termination criterion is employed to reduce the CPU time. This heuristic considerably reduces the computation at the expense of a minimal increase in the optimality gap. This accounts for the slightly larger optimality gap of (VI) when compared to that of (VII). The feasibility rate for all methods is 100% due to the application of the heuristics rules described in Lemma 3.2.

Hence, the simulations demonstrate that the state reduction approach significantly improves the closed-loop performance of the learning-based MPC control scheme for

real-time train rescheduling, while maintaining a similar evaluation time compared to the control scheme without state reduction. Moreover, state reduction considerably improves the reliability of the controller by reducing the maximum optimality gap and its variation over different initial states. From a training perspective, a lower state dimension reduces the input size of the neural network and consequently decreases the number of parameters that have to be trained. Furthermore, it reduces the memory requirements, reducing the computational effort for training.

Method	Optimality gap (%)				CPU time (s)			
	mean	std	min	max	mean	std	min	max
(I) Benchmark	0.00	0.00	0.00	0.00	258.18	287.56	0.04	600.41
(II) MINLP	0.28	2.89	-21.06	22.98	41.07	47.53	0.04	240.11
(III) MILP	0.59	5.83	-31.13	65.05	3.66	10.75	0.04	145.10
(IV) Learning + NLP without state reduction	13.06	9.41	-24.11	53.82	7.04	6.47	0.04	79.50
(V) Learning + LP without state reduction	12.82	8.17	-25.18	40.04	0.15	0.03	0.04	0.43
(VI) Learning + NLP with state reduction	**0.25**	**3.32**	**-31.37**	**5.11**	**7.24**	**6.36**	**0.04**	**101.61**
(VII) Learning + LP with state reduction	**0.00**	**3.24**	**-31.52**	**4.39**	**0.15**	**0.03**	**0.04**	**0.52**

Table 3.11: Comparison of the closed-loop performance for several approaches, with the state-reduced approach highlighted in bold.

3.7. CONCLUSIONS

In this chapter, we have investigated the passenger-oriented train rescheduling problem considering flexible train composition and rolling stock circulation. The passenger-oriented train rescheduling model of [84] has been extended to include train compositions and rolling stock circulation, considering time-varying passenger demands. To improve the online computational efficiency of model predictive control, we have combined the optimization-based and learning-based approaches, where the learning-based approach obtains integer variables, i.e., train compositions and train orders, by using pre-trained long short-term memory networks; then, the detailed timetables are optimized by solving a constrained optimization problem with the fixed integer variables. Several presolve techniques to prune the subset of integer decision variables have been developed. Moreover, the chapter has introduced a state reduction scheme tailored for learning-based MPC train rescheduling, where different levels of resolution are used to represent the system state for learning and control. Simulation results indicate that the developed learning-based framework can achieve comparable performance compared to the exact approach with an acceptable loss of feasibility, while the solution time is significantly reduced. Furthermore, the proposed state reduction procedure led to a significant decrease in the closed-loop optimality gap and offline training memory usage, thereby confirming its effectiveness.

In the future, we will investigate the integration of reinforcement learning (instead of deep learning) and model predictive control for real-time train rescheduling to improve the learning ability of the approach. Alternative state reduction mechanisms such as principal component analysis, autoencoders, or any combination thereof can potentially be explored. Among several directions, multi-agent learning-based approaches can also be a promising research direction for handling large-scale urban rail transit networks.

4

A LEARNING FRAMEWORK FOR APPROXIMATE EXPLICIT MPC

Model predictive control (MPC) is a popular control strategy due to its ability to handle constrained multivariable systems. However, its online computational and memory requirements can be demanding, often hindering its deployment in systems with fast dynamics and limited hardware resources. This chapter presents a two-stage learning-based method for approximating the explicit MPC control law for constrained linear systems. First, a set-valued classifier is used to construct a set of affine state feedback controllers. Second, a regressor is employed to approximate the cost-to-go, which is then used to select the controller associated with the lowest cost. The proposed approach does not require online optimization and exploits the piecewise affine structure of the optimal solution by design. Since the set-valued classifier provides multiple input candidates, feasibility can be explicitly encouraged by selecting inputs that satisfy prescribed constraints. Numerical simulations demonstrate that the proposed approach achieves a tractable worst-case computational complexity and a lower memory footprint than competing approximate methods, while exhibiting minimal performance degradation in one benchmark and outperforming implicit MPC in the other.

4.1. INTRODUCTION

Model predictive control (MPC) is a well-established control strategy widely adopted in industry for its ability to handle multivariable systems subject to constraints [10], [110]. Since MPC requires solving an optimization problem at every sampling instant, its implementation can be computationally demanding, especially for systems with fast dynamics [25]. For this reason, early applications of MPC were limited to systems with slow dynamics, such as chemical plants and oil refineries [110]. Recently, advances in numerical optimization and microelectronics enabled the use of MPC in embedded systems, impacting the automotive, aerospace, bioengineering, and energy systems [1],

This chapter is based on the paper [121].

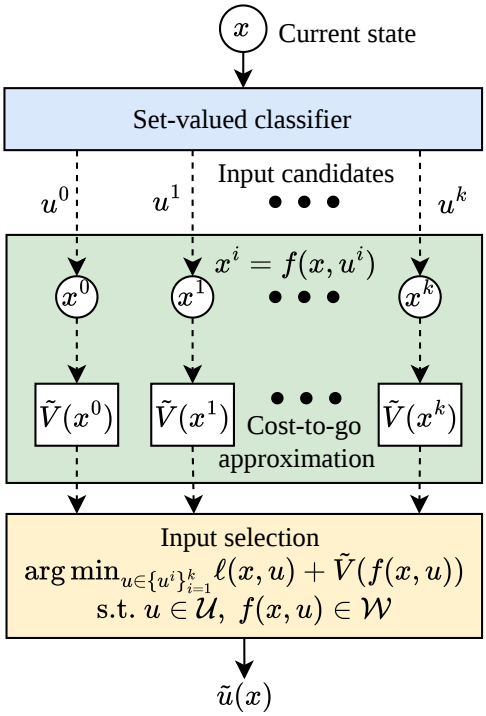


Figure 4.1: A schematic of the approximate control law evaluation. The set-valued classifier generates a set of input candidates $\{u^i\}_{i=1}^k$. For each input u^i, the predicted next state is computed via the system dynamics $x^i = f(x, u^i)$, and the corresponding approximate cost-to-go is evaluated using the regressor $\tilde{V}(x^i)$. The control input is selected by enumerating the candidate set and choosing the input associated with the lowest total cost, defined as the sum of the stage cost $\ell(x, u)$ and the approximate cost-to-go, subject to the input constraint $u \in \mathcal{U}$ and the predicted next state constraint $f(x, u) \in \mathcal{W}$.

[7], [16], [32], [38], [42] fields. However, the implementation of MPC in embedded systems remains a challenge due to high frequency sampling and limited computational and storage capacities [37].

For linear systems with quadratic costs and linear constraints, the constrained finite-horizon optimal control problem associated with MPC can be cast into a quadratic program (QP) for a given state [25]. There are several approaches to solve such a QP, including active-set methods [5], [14], [49], [53], gradient projection algorithms [106], operator splitting approaches [68], [101], [127], and interior-point methods [139]. For embedded systems, where the size of the QP is typically small, active-set methods are the most suitable choice with respect to speed, accuracy, and easiness of implementation [37].

One approach to reduce the online computational effort is to precompute the MPC control law offline using multiparametric programming [13]. This approach, known as explicit MPC, yields a continuous, piecewise affine (PWA) control law defined over a polyhedral partition of the state space [13]. However, as the number of constraints in the

multiparametric program increases, the number of polyhedral regions and associated controller gains can grow exponentially in the worst case, leading to significant computational and storage requirements [2], [13]. Several works have proposed measures to reduce the time complexity by employing binary search trees [134], hash tables [11], semi-explicit schemes [24], or by leveraging the structure of the value function and optimal control law [9]. Other schemes have been proposed to address the storage complexity by using an region-free method [81], by merging neighboring regions [52], or by applying a blend of techniques to simplify the polyhedral partition [80]. Nonetheless, explicit MPC remains most suitable for systems with few states, inputs, constraints, and short prediction horizons [2], [15].

To address the computational and storage issues of explicit MPC, a variety of approximate approaches have been proposed in the literature. In [18], canonical PWA functions are used to approximate the explicit control law over a simplicial partition of a region of interest. The method of [2] stores the most frequently visited or largest regions of the explicit MPC control law and then evaluates the control input via interpolation or extrapolation. A lattice-based PWA approximation is introduced in [145], whereas an approach based on interpolation and bilevel optimization is presented in [71]. In [72], the value function is approximated by the double-description method and the control input is obtained via barycentric interpolation. A hierarchical gridding scheme with multiscale bases is proposed in [128], and an approach based on the partition of the state-space in an orthogonal search tree is presented in [70].

Recent works have also employed learning techniques in several ways, including warm-starting active-set methods [78], approximating the value function followed by one-step-ahead optimization [60], and approximating the explicit control law using neural networks [34], [44], [74], [89], [104] or radial basis functions [40].

Each of the aforementioned approximate explicit MPC approaches entails distinct trade-offs in terms of computational complexity, memory requirements, closed-loop performance, and feasibility and stability guarantees. Among these often conflicting criteria, the goal of this chapter is to develop a learning-based framework for approximating the explicit MPC control law that achieves a low memory footprint and minimal performance degradation, while maintaining a tractable worst-case computational complexity for resource-constrained hardware.

The proposed approach consists of two main components. First, a set-valued classifier is trained to predict a set of affine state feedback controllers, which are designed to correspond to regions of the explicit MPC solution in the neighborhood of the current state. Second, a regressor is trained to approximate the cost-to-go and it is subsequently used to evaluate the cost of applying each of the affine state feedback controllers. Finally, during evaluation, pre-specified constraints can be imposed on the control input and the predicted next system state, thereby promoting feasibility during closed-loop operation. The online evaluation of the proposed approach is depicted in Fig. 4.1.

In contrast to approaches such as [18], [70], [71], [72], [128], which rely on the explicit construction of polytopic regions or the computation of the explicit control law, the proposed method avoids both procedures. Moreover, unlike learning-based approaches such as [60], [89], the proposed framework does not require online optimization. Finally, in contrast to black-box learning-based methods that directly approximate the explicit

MPC control law [34], [40], [44], [74], [104], the proposed approach explicitly exploits the piecewise affine (PWA) structure of the optimal solution.

The main contributions with respect to the literature are presented next:

- We present a learning-based framework for approximating the MPC control law for constrained linear systems. In contrast to previous work, the approximate control input is obtained by selecting a single affine controller from a state-dependent subset of a collection of affine controllers generated offline by repeated solution of the implicit MPC problem.

- We introduce set-valued classifier based on the k-nearest neighbor (k-NN) algorithm that maps the state to a set of affine controllers. To reduce the memory footprint, we propose a dataset reduction procedure that substantially decreases the cardinality of the k-NN dataset without noticeable loss in approximation accuracy or feasibility.

- The proposed approach is evaluated against explicit MPC, implicit MPC, and several state-of-the-art approximate MPC methods on two benchmark systems. The results show that the proposed approach reduces the worst-case computational complexity relative to optimal approaches, while maintaining a lower memory footprint than competing approximate methods. Closed-loop performance is nearly identical, and in one case superior, to that of the implicit MPC controller that generates the training data.

4.2. PRELIMINARIES

Consider the following finite-horizon optimal control problem for a constrained discrete-time time-invariant linear system:

$$\begin{aligned} V^*(x(t)) = \min_{U} \quad & \ell_N(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k), \\ \text{s.t.} \quad & x_0 = x(t), \\ & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1, \\ & x_k \in \mathcal{X},\ u_k \in \mathcal{U}, \quad k = 0, \dots, N-1, \\ & x_N \in \mathcal{X}_N, \end{aligned} \tag{4.1}$$

where N is the prediction horizon; $u_k \in \mathbb{R}^{n_\mathrm{u}}$ is the vector of control variables at prediction time step k; $U \triangleq [u_0^\mathsf{T},\ u_1^\mathsf{T}, \dots,\ u_{N-1}^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^m$ with $m \triangleq N \cdot n_\mathrm{u}$ is the vector of optimization variables over the prediction window; $x(t) \in \mathbb{R}^{n_\mathrm{x}}$ is the current state; $x_k \in \mathbb{R}^{n_\mathrm{x}}$ denotes the predicted state at time step $t+k$ given the current state $x(t)$ and the system dynamics; $\ell_N(\cdot)$ and $\ell(\cdot,\cdot)$ represent the terminal cost and the stage cost, respectively; $\mathcal{X} \subseteq \mathbb{R}^{n_\mathrm{x}}$, $\mathcal{U} \subseteq \mathbb{R}^m$, and $\mathcal{X}_\mathrm{N} \subseteq \mathcal{X}$ are convex polyhedra. The pair (A, B) is assumed to be stabilizable. We focus on the class of quadratic cost functions

$$\ell_N(x_N) = x_N^\mathsf{T} P x_N \quad \text{and} \quad \ell(x_k, u_k) = x_k^\mathsf{T} Q x_k + u_k^\mathsf{T} R u_k. \tag{4.2}$$

By using the relation $x_k = A^k x_0 + \sum_{j=0}^{k-1} A^j B u_{k-1-j}$ and applying algebraic manipulations [25], problem (4.1) can be cast into a multiparametric quadratic program of the form

$$\begin{aligned} \min_{U} \quad & \frac{1}{2} U^\top H U + x^\top(t) F^\top U, \\ \text{s.t.} \quad & GU \le w + Sx(t), \end{aligned} \tag{4.3}$$

where $H \in \mathbb{R}^{m \times m}$ is the Hessian matrix of the quadratic objective function, which is positive definite under the assumptions $P \succeq 0$, $Q \succeq 0$, and $R \succ 0$; $F \in \mathbb{R}^{m \times n_x}$ is the matrix defining the linear term of the objective function; and $G \in \mathbb{R}^{q \times m}$, $w \in \mathbb{R}^q$, and $S \in \mathbb{R}^{q \times n_x}$ represent the constraints over the prediction horizon. Alternatively, cost functions based on the 1-norm or the ∞-norm can also be used, rendering (4.3) a multiparametric linear program. In this case, $\ell_N(x_N) = \|P x_N\|_p$ and $\ell(x_k, u_k) = \|Q x_k\|_p + \|R u_k\|_p$ with $p = 1$ or $p = \infty$, respectively, where P, Q, and R are assumed to be full column rank matrices to guarantee positive definiteness of the cost functions.

In MPC, the control problem is solved in a receding horizon fashion, where at each time step t the current state $x(t)$ is measured, the optimal solution $U^*\big(x(t)\big) = \big[\big(u_0^*\big)^\top, \big(u_1^*\big)^\top, \ldots, \big(u_{N-1}^*\big)^\top\big]^\top$ is computed by solving problem (4.3), and then the first element of the optimal control sequence is applied to the system, defining the MPC control law

$$u(t) = u_{\text{MPC}}\big(x(t)\big) = \big[\mathbf{I}_{n_u}\ \mathbf{0}_{n_u}\ \ldots\ \mathbf{0}_{n_u}\big]\, U^*\big(x(t)\big), \tag{4.4}$$

where $\mathbf{I}_{n_u}$ is the identity matrix of size $n_u \times n_u$ and $\mathbf{0}_{n_u}$ denotes the square zero matrix of size $n_u \times n_u$.

Hereafter, x and $x(t)$ are used interchangeably whenever this is unambiguous. Let the set of states such that the problem (4.3) has a feasible solution be represented by

$$\mathscr{X}_f = \{x \mid \exists U \text{ s.t. } GU \le w + Sx\}.$$

It has been established in [13] that $\mathscr{X}_f$ is a convex polyhedron and the optimizer $U^*(x) : \mathscr{X}_f \to \mathbb{R}^m$ is continuous and piecewise affine over the set $\mathscr{X}_f$. Hence, the MPC control law can be represented as

$$u_{\text{MPC}}(x) = K^{(*,j)} x + g^{(*,j)} \text{ if } x \in \mathscr{P}^j, \tag{4.5}$$

where $\{\mathscr{P}^j\}_{j=1}^{N_{\text{par}}}$ is a non-overlapping convex polyhedral partition of the set $\mathscr{X}_f$, i.e., $\mathscr{P}^j \neq \emptyset\ \forall j \in \{1, \ldots, N_{\text{par}}\}$, $\text{int}(\mathscr{P}^i) \cap \text{int}(P^j) = \emptyset\ \forall i, j \in \{1, \ldots, N_{\text{par}}\},\ i \neq j$, and $\mathscr{X}_f = \cup_{j=1}^{N_{\text{par}}} \mathscr{P}^j$; and the tuple $(K^{(*,j)}, g^{(*,j)})$ represents the optimal controller gains over the region $\mathscr{P}^j$ of the polyhedral partition.

The optimal controller gains $(K^*(x),\ g^*(x))$ for a given state $x \in \mathscr{X}_f$ can be obtained via the Karush-Kuhn-Tucker (KKT) conditions of the problem (4.3) [2]. Consider the optimal solution $U^*(x)$ of problem (4.3) and let the set of active constraints be defined by

$$\mathscr{A}^*(x) = \{i \in \{1, \ldots, q\} \mid G_{[i]} U^*(x) = w_{[i]} + S_{[i]} x\},$$

where $G_{[i]}$, $w_{[i]}$, and $S_{[i]}$ denote the i-th row of G, w, and S, respectively. Then the optimal controller gains for a given state x can be computed as [13]

$$\begin{aligned} K^*(x) &= H^{-1}\left(W_{\mathscr{A}^*(x)}\left(S_{\mathscr{A}^*(x)} + G_{\mathscr{A}^*(x)} H^{-1} F\right) - F\right), \\ g^*(x) &= H^{-1} W_{\mathscr{A}^*(x)}\, w_{\mathscr{A}^*(x)}, \end{aligned} \tag{4.6}$$

where $G_{\mathscr{A}^*(x)} \in \mathbb{R}^{|\mathscr{A}^*(x)| \times m}$ is a matrix resulting from the concatenation of the rows corresponding to the active constraints (the same holds for $S_{\mathscr{A}^*(x)}$ and $w_{\mathscr{A}^*(x)}$) and $W_{\mathscr{A}^*(x)} = G_{\mathscr{A}^*(x)}^{\mathsf{T}}(G_{\mathscr{A}^*(x)} H^{-1} G_{\mathscr{A}^*(x)}^{\mathsf{T}})^{-1}$. Although x is inherently associated with some region $\mathscr{P}^j$ and the tuples $(K^*(x),\ g^*(x))$ and $(K^{(*,j)}, g^{(*,j)})$ are equivalent, the former notation emphasizes that the computation of $\mathscr{P}^j$ is not required, whereas the latter highlights that the tuple is optimal over $\mathscr{P}^j$. Throughout the chapter, the polyhedral partition $\{\mathscr{P}^j\}_{j=1}^{N_{\text{par}}}$ is used for conceptual clarity and is not computed explicitly.

Remark 4.1. The MPC formulation (4.1) can be modified to address tracking problems. In this case, for the system output $y \in \mathbb{R}^{n_y}$ and the constant known reference $r(t) \in \mathbb{R}^{n_y}$, the finite-horizon optimal control problem is cast as

$$
\begin{aligned}
& V^*(x(t), u(t-1), r(t)) = \\
& \quad \min_{\Delta U} \sum_{k=0}^{N-1} \|Q_y(y_k - r(t))\|_p + \|R\Delta u_k\|_p, \\
\text{s.t} \quad & x_0 = x(t),\ u_{-1} = u(t-1), \\
& x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, N-1, \\
& y_k = Cx_k, \quad k = 0, \dots, N-1, \\
& u_k = u_{k-1} + \Delta u_k, \quad k = 0, \dots, N-1, \\
& \Delta u_k = 0, \quad k = N_{\text{u}}, \dots, N-1, \\
& \Delta u_k \in \mathscr{U}_\Delta,\ u_k \in \mathscr{U}, \quad k = 0, \dots, N_{\text{u}}-1, \\
& y_k \in \mathscr{Y}, \quad k = 0, \dots, N-1,
\end{aligned}
\tag{4.7}
$$

where $Q_y \in \mathbb{R}^{n_y \times n_y}$ weighs the tracking error cost, $\Delta u \in \mathbb{R}^{n_u}$ is the input increment, and the vector with the optimization variables is defined as $\Delta U \triangleq [\Delta u_0^{\mathsf{T}},\ \Delta u_1^{\mathsf{T}},\ \dots,\ \Delta u_{N_{\text{u}}-1}^{\mathsf{T}}]^{\mathsf{T}}$. To reduce the computational cost, the input increments are constrained to zero beyond the control horizon N_{u}.

Similarly to the regulation case, the optimization problem (4.7) can be rewritten as a quadratic multiparametric program for the squared Euclidean norm ($p = 2$):

$$
\begin{aligned}
\min_{\Delta U} \quad & \frac{1}{2}\Delta U^{\mathsf{T}} H \Delta U + [x^{\mathsf{T}}(t)\ u(t-1)^{\mathsf{T}}\ r^{\mathsf{T}}(t)] F^{\mathsf{T}} \Delta U \\
\text{s.t.} \quad & G\Delta U \le w + S \begin{bmatrix} x(t) \\ u(t-1) \\ r(t) \end{bmatrix}
\end{aligned}
\quad . \tag{4.8}
$$

Note that now the explicit solution is a continuous piecewise affine function of the parameter vector $[x^{\mathsf{T}}(t)\ u^{\mathsf{T}}(t-1)\ r^{\mathsf{T}}(t)]^{\mathsf{T}}$ [13]. The optimal controller gains for a given parameter vector can also be computed by (4.6).

Remark 4.2. If the matrix $G_{\mathscr{A}^*(x)}$ is not full row rank, i.e., $\text{rank}(G_{\mathscr{A}^*(x)}) = r < p$, then r independent constraints can be selected from $G_{\mathscr{A}^*(x)}$, and the matrix corresponding to the reduced set of indices can be used to compute the optimal controller gains in (4.6) [13].

4.3. LEARNING-BASED APPROXIMATION APPROACH

This section presents the proposed framework for approximating the MPC control law (4.4). The method combines a set-valued classifier for generating control input candidates with a cost-to-go regressor for input selection. The components of the framework are formally defined, and their training and evaluation procedures are described in the following subsections.

4.3.1. FRAMEWORK DESCRIPTION

The classifier is modeled as a set-valued mapping

$$\phi : \mathscr{X} \rightrightarrows \mathscr{I}, \quad |\phi(x)| \leq k, \tag{4.9}$$

where $\mathscr{I} = \{1, \dots, N_{\text{ctrl}}\}$, $N_{\text{ctrl}} \leq N_{\text{par}}$, represents the set of indices associated with a collection of affine controllers defined by $\mathscr{D}^K = \{(K^{(*,j)}, g^{(*,j)})\}_{j=1}^{N_{\text{ctrl}}}$. Moreover, the constraint $|\phi(x)| \leq k$ imposes a limit on the cardinality of the classifier output. In essence, for each $x \in \mathscr{X}$, the image $\phi(x)$ represents a finite set of indices that can be mapped to the set of controller gains $\{(K^{(*,j)}, g^{(*,j)})\}_{j \in \phi(x)}$.

After the classifier is trained, the set of control input candidates is expressed as

$$\mathscr{U}_{\text{c}}(x) = \left\{ u^j \;\middle|\; \begin{array}{l} u^j = K^{(*,j)}x + g^{(*,j)}, \quad j \in \phi(x), \\ u^j \in \mathscr{U},\ Ax + Bu^j \in \mathscr{W} \end{array} \right\}, \tag{4.10}$$

where $\mathscr{W} \in \mathbb{R}^{n_x}$ is a convex polyhedron used to constrain the predicted next state. Sensible choices for the set $\mathscr{W}$ include the set of admissible states $\mathscr{W} = \mathscr{X}$ and the (maximal) control invariant set $\mathscr{W} = \mathscr{C} \subseteq \mathscr{X}$ (or $\mathscr{W} = \mathscr{C}_\infty \subseteq \mathscr{X}$).

The regressor is defined as a function $\tilde{V} : \mathscr{X}_{\text{f}} \to \mathbb{R}$ that approximates the optimal cost-to-go $V^*(x)$. Within the proposed framework, $\tilde{V}(\cdot)$ is used to select one of the control input candidates, defining the approximate control law as follows

$$\tilde{u}(x) = \arg\min_{u \in \mathscr{U}_{\text{c}}(x)} \ell(x,u) + \tilde{V}(f(x,u)), \tag{4.11}$$

where $f(x,u) = Ax + Bu$ denotes the system dynamics.

Intuitively, the proposed approach replaces the explicit MPC region search by first predicting a small subset of affine controllers that are likely to be optimal in the neighborhood of the current state and then selecting the most suitable one using the approximate cost-to-go.

In the remainder of this section, we describe the training, evaluation, and fine-tuning of the classifier $\phi(\cdot)$ and the regressor $\tilde{V}(\cdot)$, with the objective of achieving comparable performance to that of the MPC control law $u_{\text{MPC}}(\cdot)$.

4.3.2. DATA ACQUISITION

This subsection describes the offline construction of the datasets used to train the set-valued classifier and the regressor. The datasets are obtained by repeatedly solving the QP in (4.3) for different states sampled from the set of feasible states $\mathscr{X}_{\text{f}}$.

Let $\mathscr{D}$ and $\mathscr{D}^K$ denote datasets used to store the data required to train the classifier and the regressor. Both datasets are initialized as $\mathscr{D} \leftarrow \emptyset$ and $\mathscr{D}^K \leftarrow \emptyset$. For each sampled state

$x_{(i)}$, the problem (4.3) is solved, which outputs the optimal sequence of inputs $U^*(x_{(i)})$ and the optimal cost-to-go $V^*(x_{(i)})$. The optimal input $u^*(x_{(i)}) = u_{\text{MPC}}(x_{(i)})$ is obtained via (4.4) and the optimal controller gains $(K^*(x_{(i)}), g^*(x_{(i)}))$ are computed by extracting the set of active constraints $\mathscr{A}(x_{(i)})$ and evaluating (4.6). The index $j_{(i)}$ is defined to associate the state $x_{(i)}$ with its corresponding optimal controller gains in the dataset $\mathscr{D}^K$ such that

$$(K^*(x_{(i)}), g^*(x_{(i)})) = (K^{(*,j_{(i)})}, g^{(*,j_{(i)})}).$$

If the tuple of controller gains is not contained in $\mathscr{D}^K$ (within numerical tolerance), i.e., $(K^*(x_{(i)}), g^*(x_{(i)})) \notin \mathscr{D}^K$, the tuple is added to the dataset $\mathscr{D}^K$ and the corresponding index $j_{(i)}$ is set as the cardinality of the updated set, i.e.,

$$\mathscr{D}^K \leftarrow \mathscr{D}^K \cup \{(K^*(x_{(i)}), g^*(x_{(i)}))\}$$

, with $j_{(i)} = |\mathscr{D}^K|$. Consequently, the set $\mathscr{D}^K$ contains distinct tuples of controller gains, i.e., $K^{(*,j_1)} \neq K^{(*,j_2)}$ or $g^{(*,j_1)} \neq g^{(*,j_2)}$ for all $j_1 \neq j_2$. From the perspective of explicit MPC, the index $j_{(i)}$ can be interpreted as labeling polyhedral regions that would share the same affine controller in the explicit solution. The dataset $\mathscr{D}$ is then updated as

$$\mathscr{D} \leftarrow \mathscr{D} \cup \left\{\left((x_{(i)}),\ (j_{(i)},\ u^*_{(i)},\ V^*_{(i)})\right)\right\}.$$

This process is repeated for N_{d} samples. The resulting datasets are expressed as

$$\mathscr{D} = \left\{\left((x_{(i)}),\ (j_{(i)},\ u^*_{(i)},\ V^*_{(i)})\right)\right\}_{i=1}^{N_{\text{d}}} \tag{4.12}$$

and $\mathscr{D}^K = \{(K^{(*,j)}, g^{(*,j)})\}_{j=1}^{N_{\text{ctrl}}}$, where N_{d} and N_{ctrl} represent the number of elements in the datasets $\mathscr{D}$ and $\mathscr{D}^K$, respectively.

It is crucial that the N_{d} states $\{x_{(i)}\}_{i=1}^{N_{\text{d}}}$ are chosen such that the set of feasible initial states $\mathscr{X}_{\text{f}}$ is explored adequately. This can be achieved in several ways, such as (i) random sampling, (ii) equidistant gridding, and (iii) closed-loop trajectories with the MPC control law. For the first option, a state x is sampled randomly from $\mathscr{X}$ and it is added to $\mathscr{D}$ and $\mathscr{D}^{\text{K}}$ if the problem (4.3) is feasible, i.e., if $x \in \mathscr{X}_{\text{f}}$. For the second option, the states are sampled from an equidistant grid based on the state constraint set $\mathscr{X}$ or on an outer approximation of the set of initial feasible states $\mathscr{X}_{\text{f}}$. Similarly, the point x is only added to $\mathscr{D}$ and $\mathscr{D}^{\text{K}}$ if problem (4.3) is feasible. For the third option, closed-loop trajectories of length N_{cl} are simulated with the MPC control law (4.4) and the visited points along the trajectory are added to $\mathscr{D}$ and $\mathscr{D}^{\text{K}}$. The initial state for each trajectory can be chosen via random sampling or equidistant gridding, as described previously. Domain knowledge may be used to influence the sampling in all aforementioned approaches. In the absence of state constraints, the sampling region must be designed to cover the operating regions of interest.

4.3.3. CLASSIFIER TRAINING

In principle, any set-valued classifier of the form (4.9) can be employed within the proposed framework. In this chapter, we focus on the k-nearest neighbors (k-NN) algorithm

[41], [97] due to its conceptual simplicity and its inherent ability to generate set-valued outputs, conforming to (4.9).

A key advantage of the k-NN algorithm is that it does not require parametric training. Nevertheless, the dataset $\mathscr{D}$ obtained by the procedure described in the previous subsection typically contains a large number of elements, as adequate coverage of the state space is required. This leads to two fundamental limitations: (i) a large memory footprint for storing $\mathscr{D}$ and (ii) increased computational cost during online inference.

In order to turn the k-NN algorithm into a viable option for our approach, the number of data points in $\mathscr{D}$ must be reduced. To attain this objective, a data reduction procedure is now proposed to produce two reduced datasets, namely the reduced dataset and the reduced set of optimal controller gains:

$$\begin{aligned} \mathscr{D}^{\mathrm{r}} &= \left\{\left((x_{(i)}),\ (j_{(i)})\right)\right\}_{i=1}^{N_{\mathrm{d}}^{\mathrm{r}}} \quad \text{and} \\ \mathscr{D}^{K^{\mathrm{r}}} &= \left\{\left(K^{(*,j)}, g^{(*,j)}\right)\right\}_{j=1}^{N_{\mathrm{ctrl}}^{\mathrm{r}}}. \end{aligned} \tag{4.13}$$

The parameters $N_{\mathrm{d}}^{\mathrm{r}}$ and $N_{\mathrm{ctrl}}^{\mathrm{r}}$ represent the number of points in the reduced training dataset $\mathscr{D}^{\mathrm{r}}$ and the number of distinct optimal controllers in $\mathscr{D}^{K^{\mathrm{r}}}$. The main goal is to strike a favorable balance between classification accuracy, memory footprint, and inference speed.

We now formally define the k-NN classifier based on the reduced dataset. Given the dataset $\mathscr{D}^{\mathrm{r}}$ and a query point x, the Euclidean distances to all data points in $\mathscr{D}^{\mathrm{r}}$ are computed as

$$d_i(x) = \|x - x^{(i)}\|_2, \quad i = 1, \ldots, N_{\mathrm{d}}^{\mathrm{r}}. \tag{4.14}$$

Let $\pi(x)$ denote the index ordering that arranges the distances in nondecreasing order, i.e., $d_{\pi_1(x)}(x) \leq \cdots \leq d_{\pi_{N_{\mathrm{d}}^{\mathrm{r}}}(x)}(x)$. The classifier is defined by the indices of the k-nearest neighbors, i.e.,

$$\phi_{\mathrm{kNN}}(x) = \{\pi_1(x), \ldots, \pi_k(x)\}. \tag{4.15}$$

The core idea of the data reduction procedure is to eliminate the dataset elements whose optimal input $u^*_{(i)}$ can be recovered, within a predefined tolerance, from at least one of the optimal controller gains associated with the neighbors of $x_{(i)}$. More specifically, the following conditions must hold for at least one neighbor $j \in \phi_{\mathrm{kNN}}(x_{(i)})$:

$$\begin{aligned} &\exists\, u^j \text{ such that } u^j = K^{(*,j)} x_{(i)} + g^{(*,j)},\ u^* = u^*_{(i)}, \\ &\left| u^j_{[l]} - u^*_{[l]} \right| \leq a_{\mathrm{tol}} + r_{\mathrm{tol}} \max\left\{ \left| u^j_{[l]} \right|, \left| u^*_{[l]} \right| \right\}, \\ &\forall\, l \in \{1, \ldots, n_{\mathrm{u}}\},\ u^j \in \mathscr{U},\ Ax_{(i)} + Bu^j \in \mathscr{W}, \end{aligned} \tag{4.16}$$

where $u^j = K^{(*,j)} x_{(i)} + g^{(*,j)}$ denotes the input produced at $x_{(i)}$ by the controller gains of the j-th neighbor. The parameters a_{tol} and r_{tol} denote the absolute and relative tolerances, respectively. Moreover, the subscript $[l]$ indicates the l-th element of a vector, implying that the comparison between the control input candidate u^j and the optimal control input u^* is done in an element-wise fashion, as the components of the input vector may have significantly different value ranges.

At the start of the data reduction procedure, the reduced training set is initialized as the empty set $\mathscr{D}^{\mathrm{r}} \leftarrow \emptyset$. For each data point $x_{(i)}$ in the dataset $\mathscr{D}$, the condition (4.16) is checked, and if it is not satisfied, then the state and its corresponding index for its optimal controller gains are added to the dataset $\mathscr{D}^{\mathrm{r}} \leftarrow \mathscr{D}^{\mathrm{r}} \cup \{(x_{(i)},\ j_{(i)})\}$. If the condition (4.16) is satisfied by the state $x_{(i)}$, then the point is redundant in the sense that its associated optimal input can be closely approximated by at least one of the neighboring optimal controller gains. Typically multiple iterations through all the points of the dataset $\mathscr{D}$ are required to ensure that condition (4.16) holds for all $x_{(i)} \in \mathscr{D}$. The data reduction procedure terminates when the number of data points in $\mathscr{D}^{\mathrm{r}}$ does not change between two successive iterations or when a predefined maximum number of iterations is reached. Lastly, the optimal controller gains that are not used by any point in the reduced dataset, i.e., $(K^{(*,j)}, g^{(*,j)})$ such that $j \notin \mathscr{D}^{\mathrm{r}}$, are removed from $\mathscr{D}^{K}$ to form the reduced set of optimal controller gains $\mathscr{D}^{K^{\mathrm{r}}}$.

As a result, for every query point $x_{(i)} \in \mathscr{D}$, the application of the k-NN algorithm in the reduced dataset $\mathscr{D}^{\mathrm{r}}$ can guarantee that the optimal input $u^*_{(i)}$ is approximated, within a predefined threshold, by at least one of the control laws defined by the neighboring optimal controller gains, i.e., condition (4.16) is satisfied for all $x_{(i)} \in \mathscr{D}$.

Example 4.1. A two-dimensional system is used to demonstrate the dataset reduction procedure. The system matrices are $A = \left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$, $B = \left[\begin{smallmatrix} 1 \\ 0.5 \end{smallmatrix}\right]$. The cost function is quadratic with cost matrices $Q = \mathbf{I}_2$ and $R = 1$. The prediction horizon is set to $N = 15$. The state, input, and terminal constraints are $\|x\|_\infty \leq 5$, $\|u\|_1 \leq 0.5$, and $\mathscr{X}_{\mathrm{N}} = \mathbb{R}^2$. The terminal cost matrix P is the solution of the discrete algebraic Riccati equation. The full dataset $\mathscr{D}$ is obtained by solving the QP (4.3) in open loop for $N_{\mathrm{d}} = 3.7 \cdot 10^4$ uniformly sampled initial states from $\mathscr{X}_{\mathrm{f}}$. The number of neighbors, absolute tolerance, and relative tolerance are set to $k = 5$, $a_{\mathrm{tol}} = 10^{-2}$, and $r_{\mathrm{tol}} = 0$. The constraint on the predicted next state is chosen as the maximal control invariant set $\mathscr{W} = \mathscr{C}_\infty$. For this setting, only $N^{\mathrm{r}}_{\mathrm{d}} = 18$ samples are required in the reduced dataset $\mathscr{D}^{\mathrm{r}}$, shown in Fig. 4.2, such that the condition in (4.16) is fulfilled for all states in the full dataset.

4.3.4. REGRESSOR TRAINING

The regressor $\tilde{V}(\cdot)$ is trained using the dataset

$$\mathscr{D}_{\mathrm{V}} = \left\{\left(x_{(i)}, V^*(x_{(i)})\right)\right\}_{i=1}^{N_{\mathrm{d}}},$$

which is extracted from the dataset $\mathscr{D}$ defined in (4.12). The proposed framework admits different choices of regression models for approximating the optimal cost-to-go. In this chapter, neural networks are employed due to their expressive power, compact representation, and modest computational cost for evaluation.

4.3.5. ONLINE EVALUATION

Given the state x, the classifier $\phi(x)$ is evaluated to generate the set of control input candidates $\mathscr{U}_{\mathrm{c}}(x)$ according to (4.10). For each control input candidate $u \in \mathscr{U}_{\mathrm{c}}(x)$, the approximate cost-to-go $\tilde{V}(f(x,u))$ is evaluated and the corresponding total cost is computed. Since $\mathscr{U}_{\mathrm{c}}(x)$ is finite, the minimization in (4.11) is implemented by enumerating the control input candidates and selecting the input associated with the lowest cost.

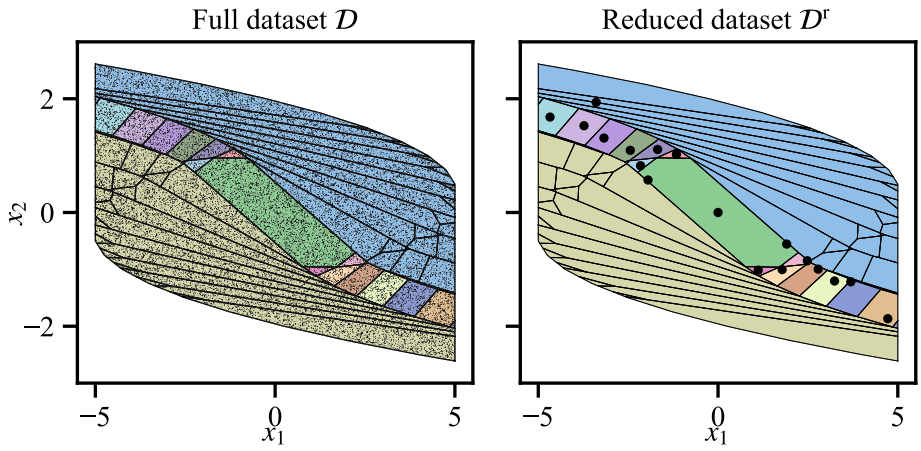


Figure 4.2: A visualization of the dataset reduction procedure for Example 4.1. The black circles represent the states contained in the datasets. The polyhedral partition of the explicit MPC solution is shown in the background, where different colors indicate distinct controller gains. After the dataset reduction, the cardinality is decreased from $3.7 \cdot 10^4$ to 18, showing that the proposed procedure preserves key features of the explicit solution using a sparse dataset.

Consider the specific case of input box constraints, i.e.,

$$\mathcal{U} = \{\, u \in \mathbb{R}^{n_{\mathrm{u}}} \mid \underline{u}_{[i]} \leq u_{[i]} \leq \bar{u}_{[i]} \;\forall i \in \{1, \ldots, n_{\mathrm{u}}\}\},$$

where $\underline{u}_{[i]}$ and $\bar{u}_{[i]}$ represent element-wise lower and upper bounds, respectively, and $u_{[i]}$ indicates the i-th element of the vector u. In this case, input constraints can be enforced by applying element-wise saturation to all the control input candidates. Consequently, the set of control input candidates can be appropriately rewritten as

$$\bar{\mathcal{U}}_{\mathrm{c}}(x) = \left\{ u^{\mathrm{s}} \;\middle|\; \begin{array}{l} j \in \phi(x),\; u = K^{(*,j)}x + g^{(*,j)}, \\ u^{\mathrm{s}}_{[l]} = \min\{\bar{u}_{[l]},\, \max\{\underline{u}_{[l]},\, u_{[l]}\}\} \\ \forall l \in \{1, \ldots, n_{\mathrm{u}}\},\; Ax + Bu^s \in \mathcal{W} \end{array} \right\}. \tag{4.17}$$

Alternatively, for the k-NN classifier (4.15), the selection criterion can be based on distance and constraint satisfaction. Let $\tilde{j}$ denote the index of the nearest neighbor in the reduced dataset $\mathcal{D}^{\mathrm{r}}$ such that its corresponding saturated control input candidate satisfies the constraint imposed in the predicted subsequent state, i.e.,

$$\tilde{j} = \arg\min_{j} \left\{ d_j(x) \;\middle|\; \begin{array}{l} j \in \phi(x),\; u = K^{(*,j)}x + g^{(*,j)}, \\ u^{\mathrm{s}}_{[l]} = \min\{\bar{u}_{[l]},\, \max\{\underline{u}_{[l]},\, u_{[l]}\}\} \\ \forall l \in \{1, \ldots, n_{\mathrm{u}}\},\; Ax + Bu^s \in \mathcal{W} \end{array} \right\}. \tag{4.18}$$

Then, the approximate control law is written as

$$\tilde{u}(x) = K^{(*,\tilde{j})}x + g^{(*,\tilde{j})}. \tag{4.19}$$

We introduce a selection flag $\sigma \in \{\text{cost}, \text{distance}\}$ that specifies the criterion used to select the control input from the candidate set $\mathscr{U}_c(x)$. For $\sigma = \text{cost}$, the candidate minimizing the approximate cost (4.11) is selected, whereas for $\sigma = \text{distance}$, the nearest candidate that satisfies the prescribed constraints is chosen according to (4.18) and (4.19).

Remark 4.3. For $\sigma = \text{cost}$, the approximate control input is obtained by minimizing the immediate stage cost in addition to the approximate cost-to-go over the subsequent N steps. Accordingly, the objective function in (4.11) represents the cost accumulated over a horizon of $N+1$ time steps, rather than the N-step prediction horizon in (4.1). Consequently, the proposed approach optimizes a slightly different objective than the nominal MPC formulation, which can induce differences in the closed-loop behavior. This design choice favors closed-loop performance when the extended horizon is beneficial, over exact approximation of the MPC control law.

Remark 4.4. Although the proposed approach promotes feasibility by selecting a control input candidate that satisfies the constraints $u^j \in \mathscr{U}$ and $Ax + Bu^j \in \mathscr{W}$, the set of control input candidates in (4.10) or (4.17) may not contain any element that fulfills these conditions. Hence, recursive feasibility cannot be guaranteed by design. In the case of infeasibility, i.e., when the set of control input candidates is empty, feasibility can be recovered by (i) projecting one of the control inputs provided by the set-valued classifier onto the set $\mathscr{W}$ using QP-based projection [74] or Dykstra's projection algorithm [34], or (ii) employing a safe backup controller.

4.3.6. CLOSED-LOOP FINE-TUNING

The offline training relies on data generated by an implicit MPC controller. Consequently, during closed-loop operation of the approximate policy, the classifier and the regressor may be evaluated at states that differ from those encountered during training, which may negatively impact the performance. This state distribution shift can be mitigated by updating the classifier and regressor using data collected from the closed-loop behavior of the approximate policy.

For a set of N_t initial states $\{x_{(l)}\}_{l=1}^{N_t}$, $x_{(l)} \in \mathscr{X}_f \; \forall l \in \{1, \dots, N_t\}$, let the l-th closed-loop trajectory of the system under the approximate control law $\tilde{u}(\cdot)$ be represented by

$$\tilde{\mathscr{T}}_l = \left\{ x(t) \;\middle|\; \begin{array}{l} x(0) = x_{(l)}, \; t = 0, \dots, N_{cl} - 1, \\ x(t+1) = Ax(t) + B\tilde{u}(x(t)) \end{array} \right\}, \tag{4.20}$$

where N_{cl} is the length of each trajectory. The same sampling strategies discussed in Section 4.3.2 for the set of initial conditions also apply here. The new set of acquired states over all trajectories is denoted by $\tilde{\mathscr{T}} = \cup_{i=1}^{N_t} \tilde{\mathscr{T}}_i$.

For the case of the k-NN classifier, which does not require parametric training, newly acquired data points that do not satisfy the reduction criterion (4.16) can be directly incorporated into the reduced datasets $\mathscr{D}^r$ and $\mathscr{D}^{K^r}$, further improving performance and feasibility with minimal impact on the memory footprint. More specifically, for each state $x_{(i)} \in \tilde{\mathscr{T}}$ for which the condition (4.16) is not satisfied, the reduced datasets $\mathscr{D}^r$ and $\mathscr{D}^{K^r}$ are augmented as follows:

$$\mathscr{D}^r \leftarrow \mathscr{D}^r \cup \{(x_{(i)}, j_{(i)})\}, \; \mathscr{D}^{K^r} \leftarrow \mathscr{D}^{K^r} \cup \{(K^{(*,j_{(i)})}, g^{(*,j_{(i)})})\},$$

where $j_{(i)}$ denotes the index associated with the optimal controller gains at $x_{(i)}$ in the possibly augmented dataset $\mathscr{D}^{K^{\mathrm{r}}}$. The augmentation of $\mathscr{D}^{K^{\mathrm{r}}}$ only occurs when the controller gains $(K^{(*,j)}, g^{(*,j)})$ are not contained in $\mathscr{D}^{K^{\mathrm{r}}}$; in this case, $j_{(i)}$ is set to $|\mathscr{D}^{K^{\mathrm{r}}}|$.

Furthermore, an imitation learning algorithm such as DAgger [116] can be employed to refine the cost-to-go approximation. In this case, the objective value (4.1) or (4.7) is evaluated for states encountered during the offline closed-loop rollouts of the approximate policy, yielding the updated dataset

$$\mathscr{D}_{\mathrm{V_u}} = \{(x,\ V^*(x)) \mid x \in \tilde{\mathscr{T}}\},$$

which is employed to further train the regressor. This procedure typically involves multiple rounds of offline closed-loop data acquisition (4.20) and retraining. This closed-loop fine-tuning step can be performed prior to validation and deployment of the approximate controller.

4.4. ANALYSIS OF THE PROPOSED METHOD

4.4.1. PROBABILISTIC FEASIBILITY VALIDATION

This subsection introduces a statistical procedure to quantify the closed-loop feasibility of the proposed approach in a probabilistic sense.

Consider a set of N_{t} initial conditions $\{x_{(l)}\}_{l=1}^{N_{\mathrm{t}}}$, $x_{(l)} \in \mathscr{X}_{\mathrm{f}}$ for $l = 1, \ldots, N_{\mathrm{t}}$, drawn independently and identically from a probability distribution defined on $\mathscr{X}_{\mathrm{f}}$. For each $x_{(l)}$, let $\tilde{\mathscr{T}}_l$ denote the l-th trajectory of length N_{cl} under the approximate control law $\tilde{u}(\cdot)$ as defined in (4.20).

The indicator function

$$I(\tilde{\mathscr{T}}_l) = \begin{cases} 1, & \text{if } \big(x(t), \tilde{u}(x(t))\big) \in \mathscr{X} \times \mathscr{U}, \\ & \forall\, t \in \{0, \ldots, N_{\mathrm{cl}} - 1\}, \\ 0, & \text{otherwise.} \end{cases} \tag{4.21}$$

characterizes the closed-loop feasibility of a trajectory, i.e., all states and inputs satisfy the constraints over the entire trajectory.

Under the i.i.d. sampling assumption on the initial states, the (unknown) probability that a closed-loop trajectory is feasible over the horizon N_{cl} is expressed as

$$p = \Pr\big(I(\tilde{\mathscr{T}}) = 1\big). \tag{4.22}$$

Each realization of the indicator function over the N_{t} trajectories is modeled as a Bernoulli random variable with success probability p, yielding $N_{\mathrm{s}} = \sum_{l=1}^{N_{\mathrm{t}}} I(\tilde{\mathscr{T}}_l)$ feasible trajectories. In the absence of prior information on the feasibility probability p, a non-informative uniform prior over $[0, 1]$ is adopted, i.e., $p \sim \mathrm{Beta}(1, 1)$. The resulting posterior distribution is given by [22]

$$p \mid N_{\mathrm{s}} \sim \mathrm{Beta}(N_{\mathrm{s}} + 1,\ N_{\mathrm{t}} - N_{\mathrm{s}} + 1). \tag{4.23}$$

Let $\alpha \in (0, 1)$, a $(1 - \alpha)$ credible interval for p is obtained from the posterior quantiles as [22]

$$\begin{aligned} \big[p_{\mathrm{L}}, p_{\mathrm{U}}\big] = \big[&F^{-1}_{\mathrm{Beta}}(\alpha/2;\ N_{\mathrm{s}} + 1,\ N_{\mathrm{t}} - N_{\mathrm{s}} + 1), \\ &F^{-1}_{\mathrm{Beta}}(1 - \alpha/2;\ N_{\mathrm{s}} + 1,\ N_{\mathrm{t}} - N_{\mathrm{s}} + 1)\big], \end{aligned} \tag{4.24}$$

where $F^{-1}_{\text{Beta}}(\cdot; a, b)$ denotes the inverse cumulative distribution function of the Beta(a, b) distribution.

In particular, with posterior probability $1 - \alpha$, the true feasibility probability p of the closed-loop system under the approximate control law is greater than or equal to p_{L}. Hence, the lower bound p_{L} can be interpreted as a conservative estimate of the feasibility probability of a closed-loop trajectory of length N_{cl} under the approximate control law.

4.4.2. COMPLEXITY ANALYSIS

Complexity certification is an important prerequisite in embedded MPC, as it is crucial to ensure that the control input can be computed within the sampling time [115]. In addition, certification also provides important information for hardware selection.

The offline and online computational complexities of the proposed framework depend on the chosen classifier and regressor, as well as on their implementations. As a representative example, we consider the k-NN classifier and the neural network regressor described in Section 4.3. In the k-NN algorithm, the selection of the nearest neighbors is implemented using a max-heap, which reduces the worst-case complexity relative to exhaustive sorting. The neural network regressor employs rectified linear units (ReLUs) in all hidden layers due to their low computational cost.

OFFLINE TRAINING

The offline computational complexity is dominated by (i) solving N_{d} QPs to generate the dataset in (4.12), and (ii) training the classifier and regressor. Active-set methods are a suitable choice to solve the QP (4.3) for their ability to exactly determine the set of active constraints, which are necessary to compute the affine controllers according to (4.6). The worst-case complexity of several active-set methods has been characterized in the literature [3], [4], [5], [37], while the average computational complexity is typically polynomial in practice [125]. The proposed k-NN classifier does not require parametric training; its offline computational cost is limited to the data reduction procedure described in Section 4.3.3. For shallow and narrow neural networks, the training complexity of the cost-to-go regressor scales mildly with the dataset size. Under these design choices, the offline training is computationally tractable and does not represent a limiting factor for the proposed approach.

ONLINE EVALUATION

For the proposed k-NN classifier and the selection flag $\sigma \in \{\text{cost}, \text{distance}\}$, we derive the worst-case number of arithmetic operations and comparisons required to compute the approximate control law during online operation.

k-NN inference: The evaluation of the k-NN classifier (4.15) consists of the computation of N^{r}_{d} distances and the selection of the smallest k distances. The latter is implemented with a max-heap of size k. An initial max-heap is built from the distances of the first k points of the dataset. As the goal is to compute the worst-case complexity, we assume that each of the remaining $N^{\text{r}}_{\text{d}} - k$ points has a distance that is smaller than the heap root, causing heap root replacement and full heap reordering for every point. This results in

$$n_{\text{kNN}} = 3N^{\text{r}}_{\text{d}} n_{\text{x}}$$

operations and

$$C_{\text{kNN}} = (h-2)2^h + 2 + h(k-2^h+1) + (N_{\text{d}}^{\text{r}} - k)(1+2h)$$

comparisons, where $h = \lfloor \log_2 k \rfloor$ is the maximum depth of the heap.

Control input candidates set: The number of arithmetic operations required to compute the set of control input candidates in (4.10) can be divided into three steps: (i) computation of the candidate inputs u^j, (ii) computation of the corresponding predicted states $x^{(+,j)} = Ax + Bu^j$, and (iii) verification of the constraint $x^{(+,j)} \in \mathcal{W}$. The three steps are performed for all $j \in \phi(x)$, with $|\phi(x)| = k$. Let the constraint set for the predicted state be defined as $\mathcal{W} = \{x \mid H_{\mathcal{W}} x \leq g_{\mathcal{W}}\}$, with $H_{\mathcal{W}} \in \mathbb{R}^{p \times n_{\text{x}}}$ and $g_{\mathcal{W}} \in \mathbb{R}^p$. In this setting, the total number of arithmetic operations is

$$n_{\text{input}} = k\Big(2n_{\text{x}} + n_{\text{x}}(2n_{\text{x}}-1) + n_{\text{x}}(2n_{\text{u}}-1) + p(2n_{\text{x}}-1)\Big),$$

while the number of comparisons to verify the constraint on the predicted state is

$$C_{\text{input}} = kp.$$

For the case of box constraints in (4.17), implementing input saturation involves additional $2kn_{\text{u}}$ comparisons.

Input selection for σ = cost: The selection criterion described in (4.11) requires k evaluations of $\ell(\cdot,\cdot)$, k evaluations of the approximate cost-to-go $\tilde{V}(\cdot)$, and k additions.

We consider that the neural network approximating the cost-to-go has a feedforward architecture, consisting of (i) an input layer of size n_{x}, (ii) N_{layers} hidden layers of sizes $s_1, \ldots, s_{N_{\text{layers}}}$, and (iii) an output layer of size 1. This yields

$$n_{\text{NN}} = (n_{\text{x}}+1)s_1 + \sum_{l=1}^{N_{\text{layers}}-1} [(s_l+1)s_{l+1}] + s_{N_{\text{layers}}} + 1$$

operations. As ReLUs are employed as the activation functions, the forward pass of the neural network also requires

$$C_{\text{NN}} = \sum_{l=1}^{N_{\text{layers}}} s_l$$

comparisons. Furthermore, the computation of the argmin in (4.11) requires $k-1$ comparisons. Consequently, the total number of arithmetic operations is

$$n_{\text{select}} = k\big((n_{\text{u}}+1)(2n_{\text{u}}-1) + n_{\text{NN}} + 1\big),$$

accounting for the evaluation of $\ell(\cdot,\cdot)$, neural network inference, and summation of the total cost. The total number of comparisons is

$$C_{\text{select}} = kC_{\text{NN}} + k - 1.$$

Input selection for σ = distance: If the selection criterion described in (4.18) is employed, then the selection step does not introduce additional arithmetic operations

beyond those required for the construction of the control input candidates set, as the distances are available from the evaluation of the k-NN classifier. Computing the arg min in (4.18) requires $k-1$ comparisons. Therefore, the number of arithmetic operations and comparisons can be expressed as

$$n_{\text{select}} = 0 \quad \text{and} \quad C_{\text{select}} = k-1, \text{ respectively.}$$

The total number of arithmetic operations and comparisons for online evaluation is then

$$n_{\text{total}} = n_{\text{kNN}} + n_{\text{input}} + n_{\text{select}} + C_{\text{kNN}} + C_{\text{input}} + C_{\text{select}}.$$

STORAGE REQUIREMENTS

Memory footprint is an important aspect in the implementation of controllers in embedded systems. We now quantify the number of scalar values that must be stored in memory to evaluate the approximate control law, neglecting temporary variable assignments.

System parameters: The following parameters must be stored: the system dynamics matrices A and B; the state and input cost matrices Q and R, respectively; the constraint matrix $H_{\mathscr{W}}$ and vector $g_{\mathscr{W}}$; and, in the case of box constraints, the lower and upper input limits $\underline{u}$, $\bar{u}$. Hence, the number of scalar values relative to the system parameters can be expressed as

$$m_{\text{par}} = n_{\text{x}}^2 + n_{\text{x}} n_{\text{u}} + n_{\text{x}}^2 + n_{\text{u}}^2 + p(n_{\text{x}}+1) + 2n_{\text{u}}.$$

k-NN: The k-NN algorithm requires the datasets $\mathscr{D}^r$, with each data point consisting of a state vector and a label, and $\mathscr{D}^{K^{\text{r}}}$, where each data point comprises controller gains, including the affine term. These datasets have

$$m_{\text{kNN}} = N_{\text{d}}^{\text{r}}(n_{\text{x}}+1) + N_{\text{ctrl}}^{\text{r}}(n_{\text{x}}+1)$$

scalar values in total.

Neural network: The neural network representing the approximate cost-to-go has an input layer of size n_{x}, N_{layers} hidden layers of size $s_1, \ldots, s_{N_{\text{layers}}}$, and an output layer of size 1, adding up to

$$m_{\text{NN}} = (n_{\text{x}}+1)s_1 + \sum_{l=1}^{N_{\text{layers}}-1} [(s_l+1)s_{l+1}] + s_{N_{\text{layers}}} + 1$$

weights, including the bias terms. The total storage is $m_{\text{total}} = m_{\text{par}} + m_{\text{kNN}} + m_{\text{NN}}$.

4.5. SIMULATION RESULTS

In this section, we assess the proposed approach on two benchmark problems, from [128], [145] and [12], [14], [37], [73], [118], and compare its performance against optimal methods, namely explicit and implicit MPC, as well as state-of-the-art approximate schemes. For explicit MPC, we adopt the online implementation described in [9], owing to its low memory footprint and low worst-case number of flops. For implicit MPC, we employ the Goldfarb-Idnani (GI) algorithm to solve the QP (4.3), as it is an efficient active-set method for small-scale problems and its exact worst-case number of flops has been characterized in [37].

In addition, we benchmark the proposed approach against three alternative approximate methods to highlight their relative strengths and limitations. The method in [74] trains a neural network via supervised learning to approximate the mapping from the state to the optimal input. The resulting input is then projected on a convex safe set to guarantee constraint satisfaction. As the benchmark problems only feature input box constraints, we do not consider orthogonal projection using a QP solver and instead apply input saturation, which simplifies the computational complexity analysis. The approach in [145] constructs a lattice-based PWA approximation of the optimal MPC control law, leading to online computations involving comparisons of affine functions, which have a low computational complexity. Finally, the method in [128] relies on multiresolution function approximation and provides stability, feasibility, and performance guarantees.

The numerical experiments were conducted on an Intel i7-13700HX CPU running at 2.1 GHz using Python 3.10.16, except for the computation of the explicit MPC control laws and the control-invariant sets, which were performed in MATLAB 2024b using the Multi-Parametric Toolbox (MPT) 3.0 [62]. The code used in this chapter is publicly available online[1].

4.5.1. BENCHMARKS

In the following, we describe the dynamics of the two benchmark problems and the parameters used for the control design and for the training and the inference of the approximate approaches.

4-D EXAMPLE

This nominally stable, single-input, four-dimensional system has been used in [145] and in [128] to evaluate approximate MPC approaches. The control objective is to regulate the system state to the origin. The system matrices are given by

$$A = \begin{bmatrix} 0.4035 & 0.3704 & 0.2935 & -0.7258 \\ -0.2114 & 0.6405 & -0.6717 & -0.0420 \\ 0.8368 & 0.0175 & -0.2806 & 0.3808 \\ -0.0724 & 0.6001 & 0.5552 & 0.4919 \end{bmatrix},$$

$$B = \begin{bmatrix} 1.6125 & 0.4806 & -1.4512 & -0.6761 \end{bmatrix}^{\mathsf{T}}.$$

The MPC formulation employs quadratic stage costs with weight matrices $Q = \mathbf{I}_2$ and $R = 0.2$. The prediction and control horizons are set to $N = N_{\mathrm{u}} = 17$. The state constraints and the input constraints are given by $\mathscr{X} = \{x \mid -5 \leq x_{[l]} \leq 5,\ l = 1,\dots,4\}$ and $\mathscr{U} = \{u \mid -0.2 \leq u \leq 0.2\}$, respectively. The terminal weight matrix P is the solution of the discrete algebraic Riccati equation (DARE). No terminal constraints are imposed, and thus $\mathscr{X}_{\mathrm{N}} = \mathbb{R}^4$. The selection flag is set to $\sigma =$ distance.

During data acquisition, $N_{\mathrm{d}} = 2 \cdot 10^5$ samples are generated by solving (4.3) in open-loop for N_{d} uniformly sampled initial states from the set of feasible states $\mathscr{X}_{\mathrm{f}}$. For the proposed approach, the number of neighbors is set to $k = 5$. In the reduction criterion (4.16), the absolute and relative tolerances are set to $a_{\mathrm{tol}} = 0.01$ and $r_{\mathrm{tol}} = 0$, respectively. For inference, the selection criterion (4.18) is applied. The constraint on the predicted next state is set to the maximal control invariant set, i.e., $\mathscr{W} = \mathscr{C}_\infty$.

[1] https://github.com/fabcaio/learning-approximate-mpc

AFTI-16 AIRCRAFT EXAMPLE

This system corresponds to a linearized approximation of the AFTI-16 aircraft longitudinal dynamics around a fixed altitude and velocity operating point. It is a well-established benchmark, used in the Hybrid Toolbox for MATLAB [12] and in several works in the literature [14], [37], [118]. The open-loop unstable continuous-time dynamics are detailed in [73]. Unlike the previous benchmark, the control objective here is to track a constant known reference. The system has four states, two inputs, and two measured outputs. Accordingly, the augmented state used in the reference tracking MPC formulation is eight-dimensional. The model is discretized using zero-order hold with a sampling time of $T_s = 0.05$ seconds.

The prediction horizon and the control horizon are set to $N_u = 2$ and $N = 10$, respectively. The stage cost is defined by the weighing matrices $Q_y = \mathbf{I}_2$ and $R = 0.1 \cdot \mathbf{I}_2$, which are diagonal square matrices penalizing the reference tracking error and the input increments, respectively. Both control inputs are constrained to lie within $[-25, 25]$. All the parameters are identical to those used in the demo `afti16.m` of the Hybrid Toolbox for MATLAB [12].

For training the proposed approach, the initial parameter vector

$$x^a \triangleq [x^\top(t)\ u^\top(t-1)\ r^\top(t)]^\top$$

is uniformly sampled from the hyper-rectangle $x^a \in [-x^a_{\max},\ x^a_{\max}]$, where

$$x^a_{\max} = [1000, 2.5, 30, 10, 25, 25, 2.5, 10]^\top.$$

From this initial state distribution, closed-loop trajectories of the form (4.20) are collected until a total of $N_d = 2 \cdot 10^5$ datapoints is obtained. For the k-NN classifier, the number of neighbors is set to $k = 15$ to balance computational complexity, memory footprint, and performance. There are no state or output constraints ($\mathscr{W} = \mathbb{R}^4$) and the input increment candidates are saturated to enforce input constraint satisfaction, as in (4.17). The selection flag is set to $\sigma = \text{cost}$. The cost-to-go is approximated by a neural network with $N_{\text{layers}} = 2$ layers and $s_1 = s_2 = 10$ neurons each.

4.5.2. RESULTS

The performance evaluation relies primarily on three metrics: the optimality gap, the worst-case number of floating-point operations (flops), and the memory footprint. The optimality gap of the optimal and approximate approaches is defined as the relative difference between the total closed-loop cost of the selected approach and that of the reference, i.e.,

$$\text{optimality gap} = \frac{J - J_{\text{ref}}}{J_{\text{ref}}} \cdot 100, \tag{4.25}$$

where J and J_{ref} represent the total closed-loop costs of the approach of interest and reference approach, respectively. The baseline closed-loop cost J_{ref} is obtained using the implicit MPC controller, possibly with larger control and prediction horizons, and serves as the reference in the computation of the optimality gap. The total closed-loop cost is obtained by summing the stage cost $\ell(\cdot,\cdot)$ over all time steps for each of the multiple closed-loop trajectories initialized from randomly sampled states. The worst-case

number of flops is selected to assess the computational complexity since it is machine-independent and directly relevant for real-time applications, where it is crucial to ensure that the control input is computed within the sampling period. Lastly, we assume single-precision arithmetic for determining the memory footprint, which accounts for the matrices and data required for evaluation.

4-D EXAMPLE

The dataset used in the proposed approach for the k-NN classification is obtained through the dataset reduction procedure described in Section 4.3, resulting in 200 points and 187 distinct affine controllers. The training time of the proposed approach, including the cost-to-go approximation, was 37 minutes, whereas the method in [74] required 35 minutes.

The results for the 4-D example are displayed in Table[2] 4.1. Among the approximate methods, the proposed approach offers the lowest memory requirement. Compared to methods in [128] and in [74], the proposed approach has a relatively higher worst-case number of flops, although it remains amenable to real-time applications. The optimality gap is negligible for all the approximate approaches, except for the method in [128].

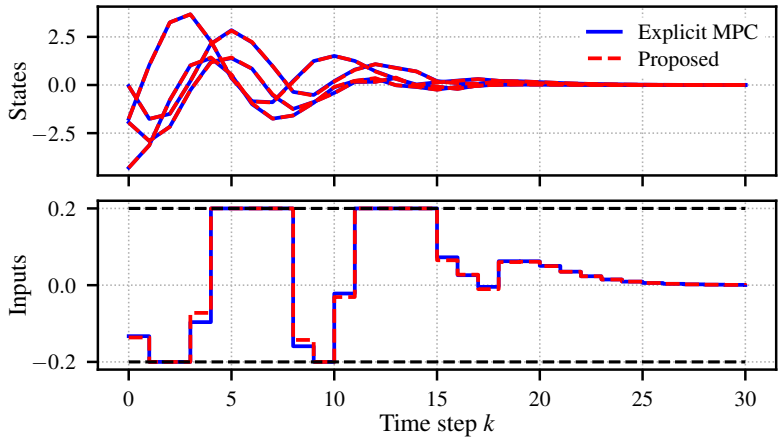


Figure 4.3: Closed-loop state and input trajectories generated by the proposed approach and the explicit MPC controller for the 4-D example.

The method in [145] exactly replicates the optimal input for this example; however, it requires significantly higher memory usage and computational cost. The scheme in [128] provides feasibility and stability guarantees and achieves the lowest worst-case number of flops, but at the expense of noticeable performance degradation and substantial memory requirements. The large computational and storage requirements of the explicit MPC scheme in [9] render it unsuitable for resource-constrained applications.

From the $5 \cdot 10^4$ simulated trajectories, the proposed approach remained feasible for all trajectories, while the method in [74] was infeasible in 29 trajectories. An example

[2] A decision tree was also implemented as an alternative set-valued classifier (4.9). However, for comparable performance, the memory footprint was substantially larger than that of the k-NN classifier, and therefore the corresponding results were omitted.

Table 4.1: Results for the 4-D example.

Method	Opt. gap (%)	Num. flops (worst case)	Storage demand (kB)	Infeas. rate ($\frac{1}{1000}$)
Proposed approach	0.071	7664	**9.55**	0
Explicit MPC [9]	**0**	445504	788.55	0
Approx. (I)[128]	<3	**119**	>35.47	0
Approx. (II) [145]	**0**	12272	51.17	0
Approx. (III) [74]	0.051	2636	9.93	0.58

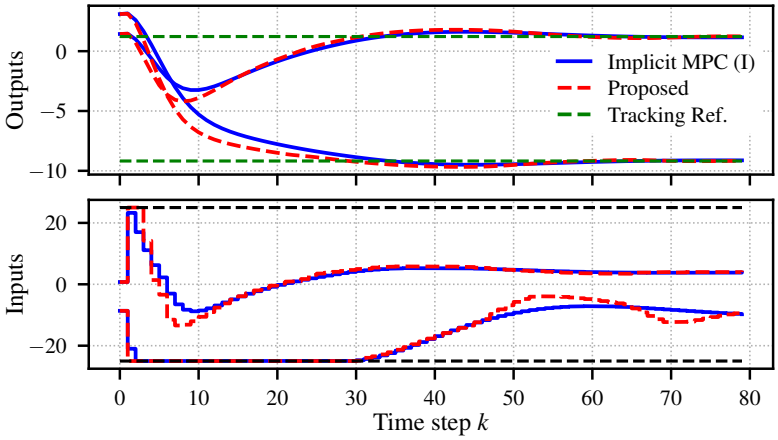


Figure 4.4: Closed-loop output and input trajectories generated by the proposed approach and the implicit MPC (I) controller for the AFTI-16 example.

of a feasible trajectory is shown in Fig. 4.3, which it is virtually identical to the optimal trajectory. Using the probabilistic validation procedure with $\alpha = 10^{-4}$ (99.99% probability), the probability that a trajectory remains feasible using the proposed approach lies within the interval $[0.9998, 1]$, while that of the method in [74] lies within the interval $[0.9988, 0.9997]$. Hence, the proposed approach exhibits a higher empirical feasibility, leading to a Bayesian credible interval with a lower bound closer to 1.

For this example, the proposed approach offers nearly optimal closed-loop performance, a low memory footprint, computational complexity suitable for resource-constrained implementations, and high feasibility.

AFTI-16 AIRCRAFT EXAMPLE

The dataset used in the proposed approach for the k-NN classification consists of 58 points and 23 distinct affine controllers. The training time of the proposed approach, including the cost-to-go approximation, was 33 minutes, whereas the method in [74] required 31 minutes.

Table 4.2: Results for the AFTI-16 Aircraft example.

Method	Opt. gap (%)	Num. flops (worst case)	Storage demand (kB)	Infeas. rate ($\frac{1}{1000}$)
Proposed approach	4.06	7397	**5.19**	0
Explicit MPC [9]	8.58	16531	142	0
Implicit MPC (I) [37]	8.58	7807	9.40	0
Implicit MPC (II) ($N_u = 3$, $N = 11$)	**0**	>7807	>9.40	0
Approx. (III) [74]	8.61	**1446**	5.40	0

The results for this example are presented in Table[2] 4.2, which reports two implicit MPC implementations with different parameter settings. The first implementation (implicit MPC (I)), with an optimality gap of 8.58%, uses control and prediction horizons of $N_u = 2$ and $N = 10$. This controller generates the dataset used to train the proposed approach and the method in [74]. The explicit MPC controller also utilizes the same settings. The second implementation (implicit MPC (II)), which is the reference for the optimality gap, is used exclusively as a performance benchmark and employs control and prediction horizons of $N_u = 3$ and $N = 11$.

The proposed approach provides the lowest memory footprint and lowest optimality gap, even outperforming the implicit MPC controller, which generated the dataset for its training. This occurs because the objective of the proposed approach is to minimize the performance cost over a horizon which is one time step longer than that of the implicit MPC (I) controller. More specifically, the proposed approach evaluation in (4.11) performs one-step minimization of a discrete set of actions considering the cost-to-go of their corresponding predicted next states over N time steps, implying that $N+1$ time steps are considered in total. An example trajectory for which the closed-loop total cost of the proposed approach is 6% lower than that of implicit MPC (I) controller is shown in Fig. 4.4. In contrast, the objective of the method in [74] is to approximate the optimal control law rather than to minimize the cost over a finite horizon. Hence, in principle, the method in [74] cannot outperform its implicit MPC (I) controller counterpart. This is reflected in Table 4.2, where the method in [74] exhibits performance degradation of 8.61% in relation to implicit MPC optimality gap of 8.58%.

For this example, only box constraints on the input are considered, meaning that no state or output constraints are imposed. Consequently, feasibility can be guaranteed by design via saturation of the control input candidates as described in Section 4.3. For $5 \cdot 10^4$ simulated trajectories, both the proposed approach and the method in [74] remained bounded and successfully tracked the reference in all cases. The probabilistic validation procedure discussed in Section 4.4.1 can be also used to assess whether the closed-loop trajectories remain bounded and whether the reference tracking objective is achieved. In this case, the indicator function (4.21) equals 1 when the energy of the states in the trajectory remains below a prescribed threshold and the reference is tracked within a

specified tolerance. For $\alpha = 10^{-4}$, the resulting Bayesian credible interval is $[0.9998, 1]$. Hence, the probabilistic analysis indicates that the approximate methods satisfy the desired closed-loop properties with high posterior probability.

For this case, the proposed approach achieves the lowest memory footprint, requires fewer worst-case number of flops than the optimal approaches, and outperforms the implicit MPC (I) approach used to generate its training dataset.

Remark 4.5. The results in Tables 4.1 and 4.2, corresponding to the approximate approaches in [128], [145] and the implicit MPC in [37], are extracted directly from the respective papers. The only exception is the approach in [74], which we implemented for comparison. In [128], only the upper bound on the optimality gap and the lower bound on the memory footprint are reported. The 4-D example is not included in [37]; therefore, Table 4.1 does not report implicit MPC results. Similarly, the AFTI-16 example is not considered in [128] or [145], and thus the results for these approximate methods do not appear in Table 4.2. Moreover, the computational complexity and storage demand of the implicit MPC used as a reference for optimality gap in Table 4.2 are not reported in [37]. For this reason, we provide the corresponding values to the implicit MPC (I) controller with shorter control and prediction horizons as conservative lower bounds. These factors explain the inconsistencies observed across Tables 4.1 and 4.2.

4.5.3. DISCUSSION

The proposed approach employs a neural network to approximate the cost-to-go (value function) associated with the MPC control problem, rather than directly approximating the MPC control law. For a fixed dataset, learning the scalar value function is typically a simpler regression task than learning the generally nonconvex, multidimensional control law. Moreover, under the standing assumptions, the MPC value function is convex, which promotes stable training and improves generalization.

The control input candidates generated by the k-NN classifier correspond to the optimal affine state feedback controllers associated with regions of the explicit MPC solution, including the optimal and/or neighboring regions, thereby exploiting the known structure of the MPC solution. This stands in contrast to black-box approaches such as approximate method (III) [74], which directly approximate the optimal control law without leveraging this structural information.

For single-precision and fixed-point implementations, the proposed approach offers improved numerical robustness relative to implicit MPC, as evaluation of the proposed approach essentially requires operations that are robust in low-precision arithmetic, such as distance calculations and matrix-vector multiplications. On the other hand, QP solvers, such as the Goldfarb-Idnani algorithm [53], require numerically sensitive operations, including matrix factorizations (Cholesky and QR), matrix inversions and divisions. Moreover, numerical robustness may be further compromised when the Hessian matrix of the QP is ill-conditioned, which accentuates errors in the aforementioned operations.

Ultimately, the choice among the optimal and approximate approaches depends on multiple factors, including system dynamics, MPC parameters, hardware constraints, and performance requirements. In the numerical examples, the proposed approach achieved a low memory footprint with negligible performance loss or even superior performance compared to the implicit MPC (I) controller that generated its training data. Moreover,

closed-loop properties such as feasibility, state boundedness, and reference tracking were validated probabilistically. These results position the proposed approach as a competitive option within the landscape of available approximate MPC methods.

Regarding tuning, the most significant parameters for the proposed approach are the number of neighbors k in the k-NN classifier and the absolute and relative tolerances used in the dataset reduction procedure. Increasing k generally decreases the number of points in the reduced dataset, but it may also increase the computational complexity, since more neighbors must be considered when constructing the set of control input candidates. Moreover, a bigger k places greater emphasis on the cost-to-go approximation when selecting the final control input. Similarly, tighter tolerances in the dataset reduction procedure lead to a more accurate approximation of the optimal input but result in a larger reduced dataset, which in turn increases both the memory footprint and the worst-case number of flops. Hence, several parameters of the proposed k-NN classifier can be tuned to align the approximate control law with given hardware constraints and performance requirements.

4.6. CONCLUSIONS

This chapter has proposed a novel learning-based framework for approximating the explicit MPC control law. The simulation results indicate that the proposed approach achieves nearly equal, and in one case superior, closed-loop performance to that of its implicit MPC counterpart, while offering a low memory footprint and a computational complexity that is suitable for resource-constrained hardware. Although feasibility cannot be guaranteed by design, this property has been assessed via a statistical validation procedure.

Future research directions include extending the method to piecewise affine and nonlinear systems, establishing theoretical guarantees for feasibility and stability, and further reducing the memory footprint and computational complexity through state dimensionality reduction techniques or alternative classifiers such as approximate k-NN methods and decision trees.

5

CONCLUSIONS

In this final chapter, we conclude the thesis with a summary of the research contributions, an outlook on the technical and societal impacts, and recommendations for future research.

5.1. RESEARCH CONTRIBUTIONS

This thesis has proposed several learning-based model predictive control (MPC) approaches aimed at reducing the online computational complexity of problems involving mixed-logical dynamical (MLD) systems, urban rail transit systems, and embedded systems. The main research contributions are summarized as follows:

- In Chapter 2, a framework integrating reinforcement learning (RL) and MPC for mixed-logical dynamical systems is developed. The key contribution is the introduction of a decoupled Q-function that mitigates the combinatorial complexity of large discrete action spaces. This decoupling renders the learning problem tractable by alleviating the combinatorial growth of the action space, which would otherwise severely hinder effective training and degrade policy performance. Within the framework, the continuous decision variables are determined via constrained optimization, preserving the constraint-enforcing and optimality-seeking properties of MPC. In a microgrid case study, the approach significantly reduces the online computational burden relative to the standard MPC formulation and achieves considerably higher feasibility than a supervised learning-based alternative, with a negligible closed-loop performance degradation.

- In Chapter 3, a learning-based MPC approach for passenger-oriented train rescheduling with flexible train composition is proposed. The core contribution is the integration of a recurrent neural network with tailored presolve techniques to predict the assignment of discrete decision variables based on offline optimal solutions of the mixed-integer MPC problem. During online operation, the trained neural network fixes the discrete variables, while the continuous variables are

computed via optimization to determine the updated train timetable. Simulation studies demonstrate that the learning-based approach achieves significant reductions in both the average and maximum online computation times, with only a minor performance degradation.

- In Chapter 4, a learning-based framework for approximate explicit MPC of constrained linear systems is introduced. The main contribution is a two-stage controller architecture that combines a classifier, a regressor, and simple selection mechanisms to exploit the piecewise affine structure of the optimal control law. Numerical examples show that the resulting approximate controller achieves low memory requirements, competitive closed-loop performance, and modest computational complexity compared to implicit MPC, explicit MPC, and alternative approximate schemes.

5.2. SCIENTIFIC AND SOCIETAL IMPACTS

Based on the aforementioned contributions, the scientific, technical and societal impacts of this thesis are discussed below:

Learning to predict discrete decision variables: In optimization-based control, the presence of discrete variables often significantly complicates the online solution of the underlying optimization problem. In Chapter 2, a decoupled Q-function is proposed to handle large combinatorial discrete action spaces and to improve the tractability of the resulting RL problem. This decoupling is not limited to the MPC formulation for MLD systems but may also be employed in other RL settings in which the discrete action space can be decomposed into smaller components by exploiting its inherent structure, for example, along temporal or spatial dimensions. In addition, the method proposed in Chapter 3 demonstrates the effectiveness of supervised learning in predicting discrete decision variables for train rescheduling. Therefore, the proposed learning-based approaches provide computationally efficient strategies for handling discrete decision variables in MPC formulations and extend the applicability of RL to problems with large discrete action spaces.

Learning to approximate MPC for constrained linear systems: Compared to existing approaches, the main contribution of the learning-based framework presented in Chapter 4 is the proposed two-stage architecture, consisting of a classifier and a regressor that exploit the piecewise affine structure of the optimal solution. As a general framework, it provides a systematic procedure for incorporating machine learning methods into the design of approximate MPC schemes. Any learning method may be employed, provided that the classifier produces a set-valued output with bounded cardinality and the regressor approximates the cost-to-go of the MPC controller. Moreover, both components should be designed to ensure low online evaluation complexity and limited memory requirements. Hence, the proposed framework enables the application of diverse learning techniques within approximate MPC to improve computational and storage efficiency for embedded implementations.

Operating infrastructure systems: The advancement of sensing, computing, and forecasting technologies has enabled the development of high-fidelity models, the deployment of optimization-based controllers, and the integration of future predictions

into decision-making across a wide range of infrastructure systems such as power grids, transportation systems, and water networks. With appropriate design, optimization-based controllers such as MPC can outperform heuristic or rule-based control strategies commonly used in practice. This capability is particularly important for the management of infrastructure systems, where the objective is to efficiently allocate resources to deliver high-quality service at low cost, while considering operational constraints and sustainability targets. However, the practical implementation of such controllers is often limited by online computational requirements, which are strongly influenced by discrete decision variables. In this context, the RL-based approach presented in Chapter 2 integrates machine learning techniques and MPC for mixed-logical dynamical systems, enabling the systematic treatment of discrete decision variables in constrained mixed-integer control problems arising in infrastructure systems and alleviating key computational limitations.

In Chapter 3, a learning-based MPC approach is proposed for the management of a large-scale urban rail transit network, explicitly considering passenger delays and operational costs. The method improves passenger comfort and promotes efficient use of energy, avoiding unnecessary consumption and supporting sustainability. Furthermore, these goals are achieved while reducing computational resource requirements, thereby lowering the associated carbon footprint and enabling scalability to larger networks. This approach can be also applied to other infrastructure systems where the online computation of discrete decision variables is challenging and sufficient offline computational resources are available to build a dataset of solutions to the control problem.

Optimizing resource utilization in embedded systems: Embedded systems are ubiquitous in modern society, with applications ranging from the automotive, aerospace, and robotics sectors to heating, ventilation, and air conditioning systems. The method proposed in Chapter 4 enables the use of MPC in scenarios where it is currently very challenging computationally. Alternatively, given its low computational and storage requirements, it reduces implementation costs and enables the deployment of MPC at a large scale. By employing MPC, the designer gains significant flexibility in specifying performance objectives and constraints, allowing important design requirements such as cost, safety, power consumption, comfort, and thermal constraints to be explicitly considered. Consequently, a broad range of applications will benefit from approximate MPC methods designed to efficiently exploit available computational resources.

5.3. RECOMMENDATIONS FOR FUTURE RESEARCH

This section presents recommendations for possible research directions.

Uncertainty management: In practical applications, control design must account for uncertainties such as model mismatch and disturbances to ensure safety, feasibility, and performance. The learning-based frameworks developed in this thesis may be extended to robust, scenario-based, or stochastic MPC formulations to handle uncertainties by design. For instance, in Chapters 2 and 3, the RL agent or the supervised learning classifier could be trained using data generated by a robust, scenario-based, or stochastic MPC controller. Accordingly, the corresponding robust, scenario-based, or stochastic MPC formulation should be used to compute the continuous decision variables during online operation. The learning-based approach presented in Chapter 4 could be readily extended to MPC formulations that explicitly consider uncertainty and can be cast as strictly convex linear

or quadratic multiparametric programs. Future work could thus focus on embedding uncertainty handling into the proposed learning-based frameworks, evaluating the trade-offs between computational complexity, performance, and robustness, and verifying whether such extensions are viable for real-time control.

Scalability: This thesis has proposed several centralized learning-based MPC approaches. In some applications, a centralized formulation may be infeasible due to the large-scale nature of the problem or constraints related to computational, communication, and cybersecurity considerations. In this context, integrating decentralized or distributed MPC approaches into the proposed methods represents a potential research direction to improve scalability and to address the aforementioned considerations. For interconnected systems, this would open up the possibility of exploiting the underlying connectivity structure in the decision-making process. For instance, the graph structure of a partitioned large-scale system could be leveraged in distributed MPC schemes and machine learning techniques such as graph neural networks.

Application to other domains: As the control scheme proposed in Chapter 2 only requires the RL agent to interact with the lumped environment comprising the system and the MPC controller, it does not rely on explicit assumptions on the system dynamics. Consequently, the scheme can be applied to nonlinear systems with discrete inputs or modes without requiring modification. Similarly, the approach presented in Chapter 3 relies on offline solutions of the MPC problem and is independent of the type of system dynamics. Hence, the approach can be applied to general MPC formulations with discrete decision variables, provided that a system model is available and the associated optimization problem can be solved to optimality. This broadens the range of potential applications of these approaches, which could be leveraged in practical settings where computational efficiency is critical.

Extension to nonlinear systems: The approximate explicit MPC approach designed in Chapter 4 can potentially be extended to nonlinear systems. In the linear case, the optimal controller gains can be computed for any given state via the KKT conditions of the QP (or LP) formulation of the MPC problem. A similar approach can be applied to piecewise affine systems, for which the structure of the explicit MPC solution is similar to that of linear systems. Although this is not possible in the general nonlinear case, a local affine approximation of the optimal control law can be obtained around any given state. These approximate local affine controllers can replace the optimal controller gains used in the linear case. In this way, the learning-based framework can be extended to nonlinear systems.

Distribution shift and online adaptation: Once a learning-based MPC controller is deployed, it induces a closed-loop state distribution that generally differs from that of the training data. This distribution shift may be further exacerbated by model mismatch, time-varying dynamics, or disturbances. Depending on the severity of the resulting mismatch between the training and closed-loop data distributions, the tunable components of the learning-based policy may need to be periodically or continuously adjusted to preserve performance and feasibility. To achieve this goal, adaptation mechanisms inspired by continual learning [135], reinforcement learning, and adaptive control can be explored. Representative examples include periodic retraining using a combination of historical and newly collected data, replay buffers to mitigate forgetting, incremental online updates

based on newly observed samples, and trust-region or regularization-based approaches to constrain deviations from the previously validated model.

Feasibility guarantees: In this thesis, the proposed learning-based MPC approaches demonstrate promising reductions in the online evaluation time and memory footprint. However, these methods cannot guarantee feasibility by design. In Chapters 2 and 3, feasibility is recovered using application-dependent heuristics. In Chapter 4, if needed, feasibility is restored either through heuristics, a safe backup controller, or by projecting an infeasible input onto a safe set. Extending these approaches to guarantee feasibility by construction would largely enhance their practical applicability. In particular, the k-NN classifier in Chapter 4 induces a polyhedral partition over the state space in which each region shares the same set of neighbors. For linear systems, this polyhedral partition could be exploited to formulate a priori feasibility checks and to identify regions of the state space that require additional sampling.

BIBLIOGRAPHY

[1] M. Aftab, C. Chen, C.-K. Chau, and T. Rahwan, "Automatic HVAC control with real-time occupancy recognition and simulation-guided model predictive control in low-cost embedded system," *Energy and Buildings*, vol. 154, pp. 141–156, 2017, ISSN: 0378-7788.

[2] A. Alessio and A. Bemporad, "A survey on explicit model predictive control," in *Nonlinear Model Predictive Control: Towards New Challenging Applications*, Springer, 2009, pp. 345–369.

[3] D. Arnström and D. Axehill, "A unifying complexity certification framework for active-set methods for convex quadratic programming," *IEEE Transactions on Automatic Control*, vol. 67, no. 6, pp. 2758–2770, 2021.

[4] D. Arnström, A. Bemporad, and D. Axehill, "Complexity certification of proximal-point methods for numerically stable quadratic programming," *IEEE Control Systems Letters*, vol. 5, no. 4, pp. 1381–1386, 2020.

[5] D. Arnström, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL^T updates," *IEEE Transactions on Automatic Control*, vol. 67, no. 8, pp. 4362–4369, 2022.

[6] W. O. Assis and B. E. Milani, "Generation of optimal schedules for metro lines using model predictive control," *Automatica*, vol. 40, no. 8, pp. 1397–1404, 2004.

[7] T. Baca, G. Loianno, and M. Saska, "Embedded model predictive control of unmanned micro aerial vehicles," in *Proc. 21st IEEE International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016, pp. 992–997.

[8] M. Balcan, T. Dick, T. Sandholm, and E. Vitercik, "Learning to branch," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 344–353.

[9] M. Baotić, F. Borrelli, A. Bemporad, and M. Morari, "Efficient on-line computation of constrained optimal control," *SIAM Journal on Control and Optimization*, vol. 47, no. 5, pp. 2470–2489, 2008.

[10] M. Bauer and I. K. Craig, "Economic assessment of advanced process control – A survey and framework," *Journal of Process Control*, vol. 18, no. 1, pp. 2–18, 2008.

[11] F. Bayat, T. A. Johansen, and A. A. Jalali, "Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control," *Automatica*, vol. 47, no. 3, pp. 571–577, 2011.

[12] A. Bemporad, *Hybrid Toolbox - User's Guide*, `http://cse.lab.imtlucca.it/~bemporad/hybrid/toolbox`, 2004.

[13] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[14] A. Bemporad, "A quadratic programming algorithm based on nonnegative least squares with applications to embedded model predictive control," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1111–1116, 2015.

[15] A. Bemporad, "Explicit model predictive control," in *Encyclopedia of Systems and Control*, Springer, 2021, pp. 744–751.

[16] A. Bemporad, D. Bernardini, R. Long, and J. Verdejo, "Model predictive control of turbocharged gasoline engines for mass production," SAE Technical Paper, Tech. Rep., 2018.

[17] A. Bemporad and M. Morari, "Control of systems integrating logic, dynamics, and constraints," *Automatica*, vol. 35, no. 3, pp. 407–427, 1999.

[18] A. Bemporad, A. Oliveri, T. Poggi, and M. Storace, "Ultra-fast stabilizing model predictive control via canonical piecewise affine approximations," *IEEE Transactions on Automatic Control*, vol. 56, no. 12, pp. 2883–2897, 2011.

[19] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021.

[20] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Optimization of the model predictive control update interval using reinforcement learning," *IFAC-PapersOnLine*, vol. 54, no. 14, pp. 257–262, 2021.

[21] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Reinforcement learning of the prediction horizon in model predictive control," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 314–320, 2021.

[22] W. M. Bolstad and J. M. Curran, *Introduction to Bayesian Statistics*. John Wiley & Sons, 2016.

[23] F. Borrelli, M. Baotić, A. Bemporad, and M. Morari, "Dynamic programming for constrained optimal control of discrete-time linear hybrid systems," *Automatica*, vol. 41, no. 10, pp. 1709–1721, 2005.

[24] F. Borrelli, M. Baotić, J. Pekar, and G. Stewart, "On the computation of linear model predictive control laws," *Automatica*, vol. 46, no. 6, pp. 1035–1041, 2010.

[25] F. Borrelli, A. Bemporad, and M. Morari, *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 2017.

[26] V. Cacchiani, J. Qi, and L. Yang, "Robust optimization models for integrated train stop planning and timetabling with passenger demand uncertainty," *Transportation Research Part B: Methodological*, vol. 136, pp. 1–29, 2020.

[27] G. Caimi, M. Fuchsberger, M. Laumanns, and M. Lüthi, "A model predictive control approach for discrete-time rescheduling in complex central railway station areas," *Computers & Operations Research*, vol. 39, no. 11, pp. 2578–2593, 2012.

[28] D. Canca, E. Barrena, A. De-Los-Santos, and J. L. Andrade-Pineda, "Setting lines frequency and capacity in dense railway rapid transit networks with simultaneous passenger assignment," *Transportation Research Part B: Methodological*, vol. 93, pp. 251–267, 2016.

[29] A. Cauligi, A. Chakrabarty, S. D. Cairano, and R. Quirynen, "PRISM: Recurrent neural networks and presolve methods for fast mixed-integer optimal control," in *Proceedings of the 4th Annual Learning for Dynamics and Control Conference*, ser. Proceedings of Machine Learning Research, vol. 168, PMLR, 2022, pp. 34–46.

[30] A. Cauligi, P. Culbertson, E. Schmerling, M. Schwager, B. Stellato, and M. Pavone, "CoCo: Online mixed-integer control via supervised learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1447–1454, 2021.

[31] G. Cavone, T. van den Boom, L. Blenkers, M. Dotoli, C. Seatzu, and B. De Schutter, "An MPC-based rescheduling algorithm for disruptions and disturbances in large-scale railway networks," *IEEE Transactions on Automation Science and Engineering*, vol. 19, no. 1, pp. 99–112, 2022.

[32] A. Chakrabarty, E. Healey, D. Shi, S. Zavitsanou, F. J. Doyle, and E. Dassau, "Embedded model predictive control for a wearable artificial pancreas," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2600–2607, 2019.

[33] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & electrical engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[34] S. Chen et al., "Approximating explicit model predictive control using constrained neural networks," in *Proc. IEEE Annual American Control Conference (ACC)*, 2018, pp. 1520–1527.

[35] Z. Chen, S. Li, A. D'Ariano, and L. Yang, "Real-time optimization for train regulation and stop-skipping adjustment strategy of urban rail transit lines," *Omega*, vol. 110, p. 102 631, 2022.

[36] Z. Chen, S. Li, H. Zhang, Y. Wang, and L. Yang, "Distributed model predictive control for real-time train regulation of metro line based on dantzig-wolfe decomposition," *Transportmetrica B: Transport Dynamics*, vol. 11, no. 1, pp. 408–433, 2023.

[37] G. Cimini and A. Bemporad, "Exact complexity certification of active-set methods for quadratic programming," *IEEE Transactions on Automatic Control*, vol. 62, no. 12, pp. 6094–6109, 2017.

[38] G. Cimini, D. Bernardini, S. Levijoki, and A. Bemporad, "Embedded model predictive control with certified real-time optimization for synchronous motors," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 2, pp. 893–900, 2020.

[39] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer Programming*. Springer, 2014.

[40] L. H. Csekő, M. Kvasnica, and B. Lantos, "Explicit MPC-based RBF neural network controller design with discrete-time actual kalman filter for semiactive suspension," *IEEE Transactions on Control Systems Technology*, vol. 23, no. 5, pp. 1736–1753, 2015.

[41] P. Cunningham and S. J. Delany, "K-nearest neighbour classifiers - A tutorial," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–25, 2021.

[42] G. R. Cygan, J. R. Verdejo, N. Jin, A. Bemporad, D. Bernardini, et al., *System and method for improving the response time of an engine using model predictive control*, US Patent 9,334,815, May 2016.

[43] B. De Schutter, T. Van den Boom, and A. Hegyi, "Model predictive control approach for recovery from delays in railway systems," *Transportation Research Record*, vol. 1793, no. 1, pp. 15–20, 2002.

[44] J. Drgoňa, D. Picard, M. Kvasnica, and L. Helsen, "Approximate model predictive building control via machine learning," *Applied Energy*, vol. 218, pp. 199–216, 2018.

[45] G. Dulac-Arnold et al., "Deep reinforcement learning in large discrete action spaces," *arXiv preprint arXiv:1512.07679*, 2016.

[46] ENTSO-E – European Network of Transmission System Operators for Electricity, *Transparency platform*, Accessed: 12-Dec-2023, 2023. [Online]. Available: `https://transparency.entsoe.eu/`.

[47] Z. Fan, R. Su, W. Zhang, and Y. Yu, "Hybrid actor-critic reinforcement learning in parameterized action space," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 2279–2285.

[48] G. Farquhar, L. Gustafson, Z. Lin, S. Whiteson, N. Usunier, and G. Synnaeve, "Growing action spaces," in *International Conference on Machine Learning*, PMLR, 2020, pp. 3040–3051.

[49] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[50] D. Frick, A. Georghiou, J. L. Jerez, A. Domahidi, and M. Morari, "Low-complexity method for hybrid MPC with local guarantees," *SIAM Journal on Control and Optimization*, vol. 57, no. 4, pp. 2328–2361, 2019.

[51] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[52] T. Geyer, F. D. Torrisi, and M. Morari, "Optimal complexity reduction of piecewise affine models based on hyperplane arrangements," in *Proc. IEEE American Control Conference (ACC)*, vol. 2, 2004, pp. 1190–1195.

[53] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming*, vol. 27, no. 1, pp. 1–33, 1983.

[54] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[55] D. Görges, "Relations between model predictive control and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 4920–4928, 2017, 20th IFAC World Congress.

[56] S. Gros and M. Zanon, "Reinforcement learning for mixed-integer problems based on MPC," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5219–5224, 2020.

[57] S. Gros and M. Zanon, "Learning for MPC with stability & safety guarantees," *Automatica*, vol. 146, 2022.

[58] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control*. Springer, 2017.

[59] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023. [Online]. Available: `https://www.gurobi.com`.

[60] K. He, S. Shi, T. van den Boom, and B. De Schutter, "Approximate dynamic programming for constrained linear systems: A piecewise quadratic approximation approach," *Automatica*, vol. 160, p. 111 456, 2024.

[61] W. P. M. H. Heemels, B. De Schutter, and A. Bemporad, "Equivalence of hybrid dynamical models," *Automatica*, vol. 37, no. 7, pp. 1085–1091, 2001.

[62] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *Proc. European Control Conference (ECC)*, `http://control.ee.ethz.ch/~mpt`, Zürich, Switzerland, Jul. 2013, pp. 502–510.

[63] M. Hessel et al., "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[64] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, pp. 269–296, 2020.

[65] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[66] Z. Hou, H. Dong, S. Gao, G. Nicholson, L. Chen, and C. Roberts, "Energy-saving metro train timetable rescheduling model considering ato profiles and dynamic passenger flow," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 7, pp. 2774–2785, 2019.

[67] Y. Huang, W. Zhou, G. Xu, and L. Deng, "Integrated demand-oriented and energy-efficiency train timetabling and rolling stock circulation planning for an urban rail transit line," *Transportation Research Part C: Emerging Technologies*, vol. 171, p. 104 993, 2025.

[68] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded online optimization for model predictive control at megahertz rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, 2014.

[69] T. A. Johansen, "Toward dependable embedded model predictive control," *IEEE Systems Journal*, vol. 11, no. 2, pp. 1208–1219, 2017.

[70] T. A. Johansen and A. Grancharova, "Approximate explicit constrained linear model predictive control via orthogonal search tree," *IEEE Transactions on Automatic Control*, vol. 48, no. 5, pp. 810–815, 2003.

[71] C. N. Jones and M. Morari, "Approximate explicit MPC using bilevel optimization," in *Proc. IEEE European Control Conference (ECC)*, 2009, pp. 2396–2401.

[72] C. N. Jones and M. Morari, "Polytopic approximation of explicit model predictive controllers," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2542–2553, 2010.

[73] P. Kapasouris, M. Athans, and G. Stein, "Design of feedback control systems for unstable plants with saturating actuators," Massachusetts Institute of Technology / NASA, NASA Contractor Report NASA-CR-183392, Nov. 1988, Accessed: 2025-12-27. [Online]. Available: https://ntrs.nasa.gov/citations/19890005006.

[74] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3866–3878, 2020.

[75] E. C. Kerrigan, J. L. Jerez, S. Longo, and G. A. Constantinides, "Number representation in predictive control," *IFAC Proceedings Volumes*, vol. 45, no. 17, pp. 60–67, 2012.

[76] H. Khadilkar, "A scalable reinforcement learning algorithm for scheduling railway lines," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 2, pp. 727–736, 2019.

[77] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[78] M. Klaučo, M. Kalúz, and M. Kvasnica, "Machine learning-based warm starting of active set methods in embedded model predictive control," *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 1–8, 2019.

[79] P. Kuppusamy, S. Venkatraman, C. Rishikeshan, and Y. P. Reddy, "Deep learning based energy efficient optimal timetable rescheduling model for intelligent metro transportation systems," *Physical Communication*, vol. 42, p. 101 131, 2020.

[80] M. Kvasnica, J. Holaza, B. Takács, and D. Ingole, "Design and verification of low-complexity explicit MPC controllers in MPT3," in *Proc. IEEE European Control Conference (ECC)*, 2015, pp. 2595–2600.

[81] M. Kvasnica, B. Takács, J. Holaza, and S. Di Cairano, "On region-free explicit model predictive control," in *Proc. 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 3669–3674.

[82] S. Li, B. De Schutter, L. Yang, and Z. Gao, "Robust model predictive control for train regulation in underground railway transportation," *IEEE Transactions on Control Systems Technology*, vol. 24, no. 3, pp. 1075–1083, 2016.

[83] X. Liu, A. Dabiri, Y. Wang, and B. De Schutter, "Modeling and efficient passenger-oriented control for urban rail transit networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 3, pp. 3325–3338, 2022.

[84] X. Liu, A. Dabiri, Y. Wang, and B. De Schutter, "Modeling and efficient passenger-oriented control for urban rail transit networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 3, pp. 3325–3338, 2023.

[85] X. Liu, A. Dabiri, Y. Wang, and B. De Schutter, "Real-time train scheduling with uncertain passenger flows: A scenario-based distributed model predictive control approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, no. 5, pp. 4219–4232, 2024.

[86] X. Liu, A. Dabiri, J. Xun, and B. De Schutter, "Bi-level model predictive control for metro networks: Integration of timetables, passenger flows, and train speed profiles," *Transportation Research Part E: Logistics and Transportation Review*, vol. 180, p. 103 339, 2023.

[87] X. Liu, C. F. O. da Silva, A. Dabiri, Y. Wang, and B. De Schutter, "Learning-based model predictive control for passenger-oriented train rescheduling with flexible train composition," *Transportation Research Part C: Emerging Technologies*, vol. 191, p. 105 841, 2026.

[88] X. Luan and F. Corman, "Passenger-oriented traffic control for rail networks: An optimization model considering crowding effects on passenger choices and train operations," *Transportation Research Part B: Methodological*, vol. 158, pp. 239–272, 2022.

[89] E. Maddalena, C. da S. Moraes, G. Waltrich, and C. Jones, "A neural network architecture to learn explicit MPC controllers from data," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11 362–11 367, 2020, 21st IFAC World Congress.

[90] D. Malyuta, B. Açıkmeşe, M. Cacan, and D. S. Bayard, "Partition-based feasible integer solution pre-computation for hybrid model predictive control," in *2019 18th European Control Conference (ECC)*, IEEE, 2019, pp. 2776–2782.

[91] W. Masson, P. Ranchod, and G. Konidaris, "Reinforcement learning with parameterized actions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[92] D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in *18th European Control Conference (ECC)*, 2019, pp. 1494–1499.

[93] D. Masti, T. Pippia, A. Bemporad, and B. De Schutter, "Learning approximate semi-explicit hybrid MPC with an application to microgrids," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5207–5212, 2020, 21st IFAC World Congress.

[94] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[95] S. Menta, J. Warrington, J. Lygeros, and M. Morari, "Learning Q-function approximations for hybrid control problems," *IEEE Control Systems Letters*, vol. 6, pp. 1364–1369, 2022.

[96] V. Mnih et al., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[97] A. Mucherino, P. Papajorgji, and P. M. Pardalos, *Data Mining in Agriculture* (Springer Optimization and Its Applications). Springer, 2009, vol. 34.

[98] V. Nair et al., "Solving mixed integer programs using neural networks," *arXiv preprint arXiv:2012.13349*, 2020.

[99] M. Neunert et al., "Continuous-discrete reinforcement learning for hybrid control in robotics," in *Conference on Robot Learning*, PMLR, 2020, pp. 735–751.

[100] H. T. Nguyen and A. H. Chow, "Adaptive rail transit network operations with a rollout surrogate-approximate dynamic programming approach," *Transportation Research Part C: Emerging Technologies*, vol. 148, p. 104 021, 2023.

[101] B. O'Donoghue, G. Stathopoulos, and S. Boyd, "A splitting method for optimal control," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2432–2442, 2013.

[102] C. Ocampo-Martinez, A. Ingimundarson, A. Bemporad, and V. Puig, "Suboptimal hybrid model predictive control: Application to sewer networks," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 10 021–10 026, 2008.

[103] H. Pan, L. Yang, and Z. Liang, "Demand-oriented integration optimization of train timetabling and rolling stock circulation planning with flexible train compositions: A column-generation-based approach," *European Journal of Operational Research*, vol. 305, no. 1, pp. 184–206, 2023.

[104] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.

[105] A. Parisio, E. Rikos, and L. Glielmo, "A model predictive control approach to microgrid operation optimization," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 5, pp. 1813–1827, 2014.

[106] P. Patrinos and A. Bemporad, "An accelerated dual gradient-projection algorithm for embedded linear model predictive control," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 18–33, 2013.

[107] T. Pippia, J. Sijs, and B. De Schutter, "A single-level rule-based model predictive control approach for energy management of grid-connected microgrids," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 6, pp. 2364–2376, 2019.

[108] S. Pu and S. Zhan, "Two-stage robust railway line-planning approach with passenger demand uncertainty," *Transportation Research Part E: Logistics and Transportation Review*, vol. 152, p. 102 372, 2021.

[109] J. Qi, V. Cacchiani, L. Yang, C. Zhang, and Z. Di, "An integer linear programming model for integrated train stop planning and timetabling with time-dependent passenger demand," *Computers & Operations Research*, vol. 136, p. 105 484, 2021.

[110] S. J. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733–764, 2003.

[111] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*. Nob Hill Publishing, 2017, ISBN: 9780975937730.

[112] R. Reiter, R. Quirynen, M. Diehl, and S. Di Cairano, "Equivariant deep learning of mixed-integer optimal control solutions for vehicle decision making and motion planning," *IEEE Transactions on Control Systems Technology*, vol. 33, no. 4, pp. 1270–1284, 2025.

[113] R. Reiter et al., "Synthesis of model predictive control and reinforcement learning: Survey and classification," *arXiv preprint arXiv:2502.02133*, 2025.

[114] A. Richards and J. How, "Mixed-integer programming for control," in *Proceedings of the American Control Conference*, IEEE, 2005, pp. 2676–2683.

[115] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1391–1403, 2011.

[116] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proc. 14th International Conference on Artificial Intelligence and Statistics*, JMLR Workshop and Conference Proceedings, 2011, pp. 627–635.

[117] L. Russo, S. H. Nair, L. Glielmo, and F. Borrelli, "Learning for online mixed-integer model predictive control with parametric optimality certificates," *IEEE Control Systems Letters*, vol. 7, pp. 2215–2220, 2023.

[118] N. Saraf and A. Bemporad, "A bounded-variable least-squares solver based on stable QR updates," *IEEE Transactions on Automatic Control*, vol. 65, no. 3, pp. 1242–1247, 2019.

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

[120] D. Šemrov, R. Marsetič, M. Žura, L. Todorovski, and A. Srdic, "Reinforcement learning approach for train rescheduling on a single-track railway," *Transportation Research Part B: Methodological*, vol. 86, pp. 250–267, 2016.

[121] C. F. O. da Silva, A. Bemporad, A. Dabiri, and B. De Schutter, "Approximate explicit model predictive control for constrained linear systems: A learning-based framework," 2026, under review.

[122] C. F. O. da Silva, A. Dabiri, and B. De Schutter, "Integrating reinforcement learning and model predictive control for mixed-logical dynamical systems," *IEEE Open Journal of Control Systems*, vol. 4, pp. 316–331, 2025.

[123] C. F. O. da Silva, X. Liu, A. Dabiri, and B. De Schutter, "A state reduction approach for learning-based model predictive control for train rescheduling," *IFAC-PapersOnLine*, vol. 59, no. 26, pp. 383–388, 2025, 7th IFAC Conference on Intelligent Control and Automation Sciences (ICONS).

[124] D. Silver et al., "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv*, 2017.

[125] D. A. Spielman and S.-H. Teng, "Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time," *Journal of the ACM (JACM)*, vol. 51, no. 3, pp. 385–463, 2004.

[126] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[127] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[128] S. Summers, C. N. Jones, J. Lygeros, and M. Morari, "A multiresolution approximation method for fast explicit model predictive control," *IEEE Transactions on Automatic Control*, vol. 56, no. 11, pp. 2530–2541, 2011.

[129] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Second. Bradford Books, 2018.

[130] R. Tang et al., "A literature review of artificial intelligence applications in railway systems," *Transportation Research Part C: Emerging Technologies*, vol. 140, p. 103 679, 2022.

[131] Y. Tang, S. Agrawal, and Y. Faenza, "Reinforcement learning for integer programming: Learning to cut," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, PMLR, 2020, pp. 9367–9376.

[132] A. Tavakoli, F. Pardo, and P. Kormushev, "Action branching architectures for deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[133] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30.

[134] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003.

[135] L. Wang, X. Zhang, H. Su, and J. Zhu, "A comprehensive survey of continual learning: Theory, method and application," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 8, pp. 5362–5383, 2024.

[136] P. Wang, X. Xiao, and N. Nießen, "Flexible rolling stock composition strategy in urban rail transit lines: The influences of various train units and station capacities," *Transportation Research Part C: Emerging Technologies*, vol. 162, p. 104 609, 2024.

[137] S.-y. Wang, A. H. Chow, and C.-s. Ying, "Adaptive and flexible rail transit network service dispatching as a partially observable markov decision process," *Transportation Research Part C: Emerging Technologies*, vol. 179, p. 105 286, 2025.

[138] X. Wang, S. Li, T. Tang, and L. Yang, "Event-triggered predictive control for automatic train regulation and passenger flow in metro rail systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 3, pp. 1782–1795, 2022.

[139] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2009.

[140] Y. Wang, A. D'Ariano, J. Yin, L. Meng, T. Tang, and B. Ning, "Passenger demand oriented train scheduling and rolling stock circulation planning for an urban rail transit line," *Transportation Research Part B: Methodological*, vol. 118, pp. 193–227, 2018.

[141] Y. Wang, B. Ning, T. Tang, T. J. van den Boom, and B. De Schutter, "Efficient real-time train scheduling for urban rail transit systems using iterative convex programming," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 6, pp. 3337–3352, 2015.

[142] Y. Wang, S. Zhu, A. D'Ariano, J. Yin, J. Miao, and L. Meng, "Energy-efficient timetabling and rolling stock circulation planning based on automatic train operation levels for metro lines," *Transportation Research Part C: Emerging Technologies*, vol. 129, p. 103 209, 2021.

[143] H. P. Williams, *Model Building in Mathematical Programming*. John Wiley & Sons, 2013.

[144] X. Wu, H. Dong, and K. T. Chi, "Multi-objective timetabling optimization for a two-way metro line under dynamic passenger demand," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 8, pp. 4853–4863, 2021.

[145] J. Xu, Y. Lou, B. De Schutter, and Z. Xiong, "Error-free approximation of explicit linear MPC through lattice piecewise affine expression," *IEEE Transactions on Automatic Control*, vol. 70, no. 3, pp. 1745–1760, 2025.

[146] L. Yang, Y. Gao, A. D'Ariano, and S. Xu, "Integrated optimization of train timetable and train unit circulation for a y-type urban rail transit system with flexible train composition mode," *Omega*, vol. 122, p. 102 968, 2024.

[147] J. Yin, T. Tang, L. Yang, Z. Gao, and B. Ran, "Energy-efficient metro train rescheduling with uncertain time-variant passenger demands: An approximate dynamic programming approach," *Transportation Research Part B: Methodological*, vol. 91, pp. 178–210, 2016.

[148] C.-S. Ying, A. H. Chow, Y.-H. Wang, and K.-S. Chin, "Adaptive metro service schedule and train composition with a proximal policy optimization approach based on deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6895–6906, 2021.

[149] C. Ying, A. H. Chow, Y. Yan, Y.-H. Kuo, and S. Wang, "Adaptive rescheduling of rail transit services with short-turnings under disruptions via a multi-agent deep reinforcement learning approach," *Transportation Research Part B: Methodological*, vol. 188, p. 103 067, 2024.

[150] M. Zamponi, E. Incerto, D. Masti, and M. Tribastone, "Certified inductive synthesis for online mixed-integer optimization," in *Proceedings of the ACM/IEEE 16th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2025)*, 2025, pp. 1–11.

[151] J. Zhang et al., "A survey for solving mixed integer programming via machine learning," *Neurocomputing*, vol. 519, pp. 205–217, 2023.

[152] Y. Zhao, D. Li, Y. Yin, and X. Zhao, "Integrated optimization of demand-driven timetable, train formation plan and rolling stock circulation with variable running times and dwell times," *Transportation Research Part E: Logistics and Transportation Review*, vol. 171, p. 103 035, 2023.

CURRICULUM VITÆ

Caio Fabio OLIVEIRA DA SILVA



Born on November 4, 1995, in Brasília, Brazil, Caio obtained his B.Sc. degree in Electrical Engineering from the University of Brasília, Brazil, and his M.Sc. degree in Automation and Control Engineering from the Politecnico di Milano, Italy, with research experience in the areas of filtering, system identification, and model predictive control.

In October 2021, he became a Ph.D. candidate at the Delft Center for Systems and Control, Delft University of Technology, the Netherlands, under the supervision of Prof. dr. ir. Bart De Schutter and Dr. Azita Dabiri. From March to June 2025, he was a visiting research fellow at the Dynamical Systems, Control, and Optimization Group at the IMT School for Advanced Studies Lucca, Italy, hosted by Prof. Alberto Bemporad. His research interests lie at the intersection of control, learning, and optimization. His doctoral research focuses on integrating model predictive control and learning to develop control strategies that improve the online computational efficiency of real-time control in mixed-logical dynamical and constrained linear systems.

List of Publications

Journals

3. **C. F. O. da Silva**, A. Bemporad, A. Dabiri, and B. De Schutter. Approximate explicit model predictive control for constrained linear systems: A learning-based framework, *under review, 2026*.

2. X. Liu, **C. F. O. da Silva**, A. Dabiri, Y. Wang and B. De Schutter. Learning-based model predictive control for passenger-oriented train rescheduling with flexible train composition, *Transportation Research Part C: Emerging Technologies*, vol. 191, p. 105 841, 2026. X. Liu and C. F. O. da Silva contributed equally.

1. **C. F. O. da Silva**, A. Dabiri, and B. De Schutter. Integrating reinforcement learning and model predictive control for mixed-logical dynamical systems, *IEEE Open Journal of Control Systems*, vol. 4, pp. 316–331, 2025.

Conferences

1. **C. F. O. da Silva**, X. Liu, A. Dabiri, and B. De Schutter. A state reduction approach for learning-based model predictive control for train rescheduling, *IFAC-PapersOnLine*, vol. 59, no. 26, pp. 383–388, 2025. 7th IFAC Conference on Intelligent Control and Automation Sciences (ICONS).