A Hybrid Learning-to-Optimize Framework for Mixed-Integer Quadratic Programming

Viet-Anh Le VIETANH@SEAS.UPENN.EDU
Mu Xie MUX2001@SEAS.UPENN.EDU
Rahul Mangharam RAHULM@SEAS.UPENN.EDU
Department of Electrical & Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104, USA

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

In this paper, we propose a learning-to-optimize (L2O) framework to accelerate solving parametric mixed-integer quadratic programming (MIQP) problems, with a particular focus on mixed-integer model predictive control (MI-MPC) applications. The framework learns to predict integer solutions with enhanced optimality and feasibility by integrating supervised learning (for optimality), self-supervised learning (for feasibility), and a *differentiable quadratic programming (QP) layer*, resulting in a hybrid L2O framework. Specifically, a neural network (NN) is used to learn the mapping from problem parameters to optimal integer solutions, while a differentiable QP layer is integrated to compute the corresponding continuous variables given the predicted integers and problem parameters. Moreover, a *hybrid loss function* is proposed, which combines a supervised loss with respect to the global optimal solution, and a self-supervised loss derived from the problem's objective and constraints. The effectiveness of the proposed framework is demonstrated on two benchmark MI-MPC problems, with comparative results against purely supervised and self-supervised learning models.

Keywords: Learning to optimize, mixed-integer quadratic programming, mixed-integer model predictive control.

1. Introduction

Mixed-integer optimization is fundamental to many control applications involving discrete decision-making, such as autonomous driving (Quirynen et al., 2024), traffic signal coordination with connected automated vehicles (Le and Malikopoulos, 2024), multi-robot pickup and delivery (Camisa et al., 2022), motion planning and task assignment for robot fleets (Salvado et al., 2018), and signal temporal logic specifications (Belta and Sadraddini, 2019). However, solving a mixed-integer program (MIP) is NP-hard because it requires combinatorial search over discrete decision variables. Consequently, computation time can grow exponentially with problem size or constraint complexity, making MIPs generally unsuitable for real-time control or decision-making.

Advancements in machine learning and differentiable programming provide a promising opportunity to accelerate MIP solvers through learning-to-optimize (L2O) frameworks. The literature on L2O for MIP problems is still limited but has gained increasing attention in recent years. The current state of the art can be categorized into two main approaches: (i) supervised learning (SL) and (ii) self-supervised learning (SSL). Classical SL approaches, e.g., (Cauligi et al., 2021, 2022; Le et al., 2025), train neural networks (NNs) to minimize the supervised loss between predictions and the optimal integer solutions generated by an optimization solver such as `GUROBI` (Gurobi Optimization, LLC, 2021). A major drawback of SL approaches is that they do not guarantee feasibility,

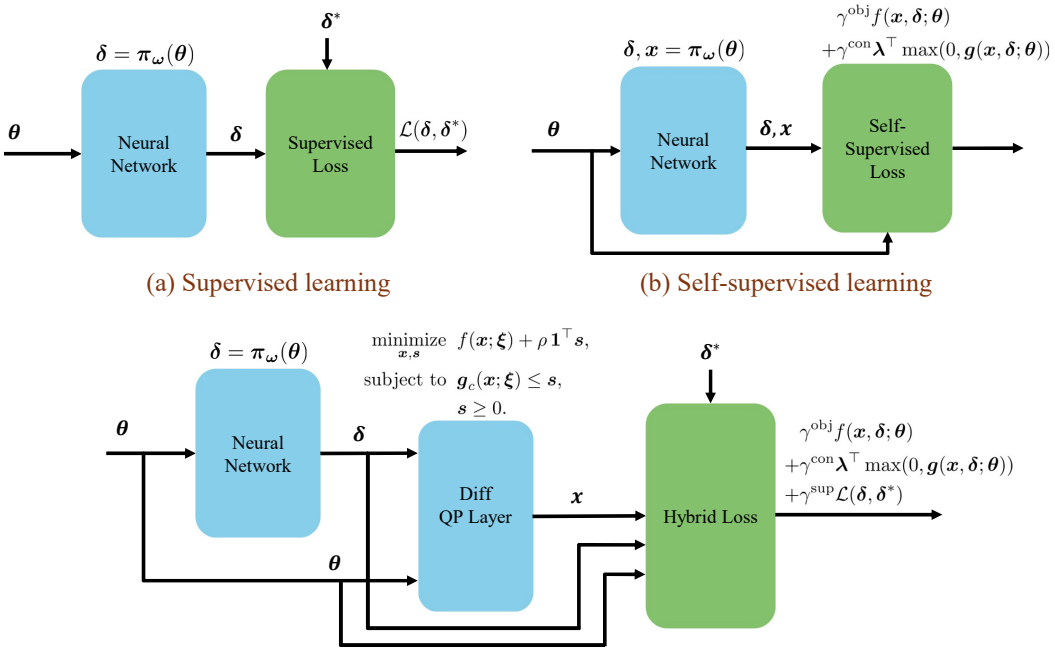

Figure 1: Architecture of the proposed hybrid framework (c) compared with supervised learning (a) and self-supervised learning (b). In our framework, the NN takes the problem parameters $\boldsymbol{\theta}$ to predict the integer solution $\boldsymbol{\delta}$, while the QP layer computes the continuous solution $\boldsymbol{x}$ based on $\boldsymbol{\theta}$ and $\boldsymbol{\delta}$. In conventional SL and SSL, the NN is trained to predict the integer solution without considering the continuous solution or to predict both the integer and continuous solutions, respectively.

i.e., the resulting convex continuous problems obtained by fixing the learned integer variables can be infeasible. SSL approaches (Tang et al., 2024; Boldockỳ et al., 2025), on the other hand, do not rely on labeled data and can improve feasibility by using a loss function that combines the objective function with a penalty for constraint violation. For example, (Tang et al., 2024) proposed a framework for mixed-integer nonlinear programming that solves the integer-relaxed problem, combined with integer correction layers to ensure integrality and a projection step to improve feasibility. (Boldockỳ et al., 2025) considered parametric MIQP problems within a differentiable predictive control framework, which constrains the integer solutions and optimal control inputs to a neural state-feedback law. However, trained SSL models may produce feasible but suboptimal solutions, since differentiable programming techniques such as gradient descent may converge to locally optimal solutions.

Addressing the limitations of both SL (infeasibility) and SSL (suboptimality), we propose a novel hybrid L2O framework that strategically combines both training paradigms with an integrated differentiable QP layer. The hybrid approach proposed in this paper shares conceptual similarities with physics-informed machine learning (PINN) by embedding the known optimization structure of the problem (the QP layer) as an inductive bias, much as PINNs embed physical laws (the PDEs). First, we propose a novel architecture in which a differentiable QP layer is integrated into the network to better incorporate the optimization structure. During training, the QP layer takes the NN predictions of the integer decision variables as input and outputs the corresponding optimal solu-

tions for the continuous variables, but it may be infeasible. To address this issue and ensure that gradients can always be computed, we propose a simple yet effective approach that introduces a differentiable layer for the relaxed QP problem. We prove that, if the penalty weight is chosen sufficiently large, the relaxed problem yields either the exact optimal solution when the original QP is feasible or the minimally infeasible solution otherwise. Second, we propose a new loss function design, called a *hybrid loss function*, defined as a weighted sum of SL and SSL losses. This approach allows the framework to balance the feasibility-optimality trade-off in L2O. The overall architecture of our framework in comparison with SL and SSL can be illustrated in Fig. 1.

Our framework differs from existing work in the relevant literature in the following aspects. First, a major difference lies in how we encode the dependency between continuous variables, integer variables, and problem parameters during training. In SL approaches (Cauligi et al., 2021, 2022), the goal is to learn the mapping from problem parameters to the optimal integer variables, while disregarding the continuous variables during training. In online prediction, the continuous variables are then obtained by solving a QP given the NN predictions of the integer variables. In contrast, SSL approaches (Tang et al., 2024; Boldocký et al., 2025) train NNs to predict both the continuous and integer variables from the problem parameters. Thus, the prediction obtained directly from the NN does not explicitly account for the dependency between continuous and integer variables. In our approach, we incorporate the continuous variables into training by integrating a QP layer, based on the fact that the optimal continuous variables are the solutions of a parametric QP given the integer variables and the MIQP problem parameters. Therefore, our approach better incorporates the underlying optimization structure into both training and prediction than supervised and self-supervised learning. Second, we combine supervised and self-supervised learning objectives to define a hybrid loss function, rather than relying on either one alone. This hybrid loss allows the framework to balance the optimality of supervised learning with training labels and the feasibility improvement of self-supervision during training. Thus, our proposed framework can be viewed as a compromise between SL and SSL. We show through numerical examples that the hybrid loss function achieves near-global optimality and minimal constraint violation for most problem instances.

2. Preliminaries

This section provides a brief discussion on the parametric MIQPs and L2O for accelerating solving MIQP problems.

2.1. Parametric MIQPs

We consider a parametric MIQP problem that takes the following form:

$$\underset{\boldsymbol{x}\in\mathcal{X},\boldsymbol{\delta}\in\mathcal{I}}{\text{minimize}} \quad f(\boldsymbol{x},\boldsymbol{\delta};\boldsymbol{\theta}), \tag{1a}$$

$$\text{subject to} \quad \boldsymbol{g}(\boldsymbol{x},\boldsymbol{\delta};\boldsymbol{\theta}) \leq 0, \tag{1b}$$

where $\boldsymbol{x}$ is the vector of continuous optimization variables, $\boldsymbol{\delta}$ is the vector of integer optimization variables, and $\boldsymbol{\theta}$ is the vector of problem parameters. We let $\mathcal{X}$ and $\mathcal{I}$ be the domains for continuous and integer optimization variables, and $\boldsymbol{g}(\cdot) = [g_1(\cdot), \ldots, g_r(\cdot)]$ be the vector of r linear constraints. We assume that $\mathcal{X}$ and $\mathcal{I}$ are non-empty. In this work, we consider a convex quadratic objective

function and linear constraints, i.e.,

$$\begin{aligned} f(\boldsymbol{x}, \boldsymbol{\delta}; \boldsymbol{\theta}) &= \frac{1}{2} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\delta} \end{bmatrix}^\top \boldsymbol{Q}(\boldsymbol{\theta}) \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\delta} \end{bmatrix} + \boldsymbol{p}(\boldsymbol{\theta})^\top \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\delta} \end{bmatrix}, \\ \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\delta}; \boldsymbol{\theta}) &= \boldsymbol{G}(\boldsymbol{\theta}) \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{\delta} \end{bmatrix} - \boldsymbol{h}(\boldsymbol{\theta}). \end{aligned} \tag{2}$$

where $\boldsymbol{Q}(\boldsymbol{\theta}) \succeq 0$. Note that given known parameters $\boldsymbol{\theta}$ and integer variables $\boldsymbol{\delta}$, the optimal solution of the continuous variables can be obtained by solving a QP problem, if it is feasible, as follows:

$$\underset{\boldsymbol{x} \in \mathcal{X}}{\text{minimize}} \quad f(\boldsymbol{x}; \boldsymbol{\xi}), \tag{3a}$$

$$\text{subject to} \quad \boldsymbol{g}_c(\boldsymbol{x}; \boldsymbol{\xi}) \leq 0, \tag{3b}$$

where $\boldsymbol{g}_c(\cdot)$ denotes the components of $\boldsymbol{g}(\cdot)$ that involve at least one continuous decision variable, and $\boldsymbol{\xi} = [\boldsymbol{\delta}^\top, \boldsymbol{\theta}^\top]^\top$. The objective function and constraint function in (3) can be expressed as:

$$f(\boldsymbol{x}; \boldsymbol{\xi}) = \frac{1}{2} \boldsymbol{x}^\top \boldsymbol{Q}_x(\boldsymbol{\xi}) \boldsymbol{x} + \boldsymbol{p}_x(\boldsymbol{\xi})^\top \boldsymbol{x}, \tag{4}$$

$$\boldsymbol{g}_c(\boldsymbol{x}; \boldsymbol{\xi}) = \boldsymbol{G}_x(\boldsymbol{\xi}) \boldsymbol{x} - \boldsymbol{h}_x(\boldsymbol{\xi}). \tag{5}$$

Mixed-integer MPC: A typical application that can greatly benefit from L2O approaches is model predictive control (MPC). In MPC, we solve a parametric optimization problem at every time step, in which the problem parameters may include, for instance, a given initial state, target state, to name a few. Using machine learning to solve or assist in solving parametric MPC problems enables real-time implementation of complex MPC, such as nonlinear MPC or mixed-integer MPC. We consider the state $\boldsymbol{x}_t \in \mathcal{X} \subset \mathbb{R}^{n_x}$ and control inputs $\boldsymbol{u}_t \in \mathcal{U} \subset \mathbb{R}^{n_u}$ as the continuous decision variables and let $\boldsymbol{\delta}_t \in \mathcal{I} \subset \mathbb{Z}^{n_\delta}$ be the integer decision variables. Let H be the control horizon length, and $\boldsymbol{x}_{0:H}, \boldsymbol{u}_{0:H-1}, \boldsymbol{\delta}_{0:H-1}$ be the concatenated vectors over the control horizon. Given a vector of problem parameters $\boldsymbol{\theta} \in \mathbb{R}^{n_p}$, a parametric MI-MPC can be written as:

$$\begin{aligned} \underset{\boldsymbol{x}_{0:H}, \boldsymbol{u}_{0:H-1}, \boldsymbol{\delta}_{0:H-1}}{\text{minimize}} \quad & c_H(\boldsymbol{x}_H; \boldsymbol{\theta}) + \sum_{t=0}^{H-1} c_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\delta}_t; \boldsymbol{\theta}), \\ \text{subject to} \quad & \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}}(\boldsymbol{\theta}), \\ & \boldsymbol{x}_{t+1} = \boldsymbol{f}(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\delta}_t; \boldsymbol{\theta}), \ t = 0, \ldots, H-1, \\ & \boldsymbol{g}_t(\boldsymbol{x}_t, \boldsymbol{u}_t, \boldsymbol{\delta}_t; \boldsymbol{\theta}) \leq 0, \ t = 0, \ldots, H-1, \\ & \boldsymbol{g}_H(\boldsymbol{x}_t; \boldsymbol{\theta}), \end{aligned} \tag{6}$$

where the stage cost $c_t(\cdot)$ and terminal cost $c_H(\cdot)$ are convex quadratic, while the dynamics $\boldsymbol{f}(\cdot)$, the inequality constraints $\boldsymbol{g}_t(\cdot)$ and $\boldsymbol{g}_H(\cdot)$ are assumed to be linear functions. The objective function and constraints are functions of the parameter vector $\boldsymbol{\theta} \in \Theta$, where $\Theta \subseteq \mathbb{R}^{n_p}$ is the admissible set of parameters.

2.2. Learning to Optimize for MIQPs

Due to the combinatorial nature, the problem (1) is computationally demanding to solve, where finding the optimal solution may scale exponentially with the problem size. Meanwhile, solving (3)

while the integers are fixed is significantly cheaper to solve than the MIQP. An interesting approach to accelerating solving problems of the form (1) is to learn a map between the vector of problem parameters $\boldsymbol{\theta}$ and the discrete optimizer $\boldsymbol{\delta}^*$ by an NN, $\boldsymbol{\delta}^* = \boldsymbol{\pi}_{\boldsymbol{\omega}}(\boldsymbol{\theta})$, where $\boldsymbol{\omega}$ is the vector of network weights. Overall, there are two main approaches for learning $\boldsymbol{\pi}_{\boldsymbol{\omega}}$, including (i) supervised learning and (ii) self-supervised learning.

Supervised Learning: The NN $\boldsymbol{\pi}_{\boldsymbol{\omega}}$ can be trained by a classical SL approach. In SL, we collect the optimal solutions $\boldsymbol{\delta}^{i,*}$ corresponding to each $\boldsymbol{\theta}^i$, for $i = 1, \ldots, M$, obtained from a solver. We then use the dataset $\{\boldsymbol{\theta}^i, \boldsymbol{\delta}^{i,*}\}$ to train a NN that minimizes the following loss function:

$$\underset{\boldsymbol{\omega}}{\text{minimize}} \ \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(\boldsymbol{\pi}_{\boldsymbol{\omega}}(\boldsymbol{\theta}^i), \boldsymbol{\delta}^{i,*}), \tag{7}$$

where $\mathcal{L}$ denotes a supervised loss (e.g., Huber or cross-entropy) between the predicted outputs and labels. However, the prediction from an SL model may not ensure that the resulting QP is feasible, although the original MIQP is feasible.

Self-Supervised Learning: Self-supervised learning, contrary to SL, does not rely on labeled data for training the model. It instead trains models by minimizing the objective function and constraint violation directly from model predictions. In generic SSL, an NN is trained to predict both the integer and continuous variables, i.e., $(\boldsymbol{\delta}^i, \boldsymbol{x}^i) = \boldsymbol{\pi}_{\boldsymbol{\omega}}(\boldsymbol{\theta}^i)$, and to train the NNs given a dataset with M training instances, the following self-supervised loss function is used:

$$\underset{\boldsymbol{\omega}}{\text{minimize}} \ \frac{1}{M} \sum_{i=1}^{M} f(\boldsymbol{x}^i, \boldsymbol{\delta}^i; \boldsymbol{\theta}^i) + \boldsymbol{\lambda}^\top \max\left(0, \boldsymbol{g}(\boldsymbol{x}^i, \boldsymbol{\delta}^i; \boldsymbol{\theta}^i)\right), \tag{8a}$$

$$\text{subject to} \ \ (\boldsymbol{\delta}^i, \boldsymbol{x}^i) = \boldsymbol{\pi}_{\boldsymbol{\omega}}(\boldsymbol{\theta}^i), \tag{8b}$$

In (8), we include a penalty for constraint violation with max penalty (implemented via a ReLU) function. $\boldsymbol{\lambda} \in \mathbb{R}^r_{>0}$ is a vector of penalty parameters that balances the trade-off between minimizing the objective function and satisfying the constraints. Although the penalty methods lack formal guarantees, they often outperform their hard constraint counterparts in practice. Training the NN with a self-supervised loss function can improve, though it does not guarantee, feasibility. Nevertheless, a major drawback of this approach is that since the self-supervised loss (8) is non-convex with respect to $\boldsymbol{\omega}$, gradient-based methods may not converge and may converge to a sub-optimal solution. Moreover, in MPC applications, designing an NN to predict the optimal continuous decision variables from the problem parameters is generally challenging, as it is difficult to enforce the system dynamics on the network outputs (Cauligi et al., 2021), unless the optimal integer and continuous solutions at each time step are constrained to follow a state-feedback law, as in differentiable predictive control (Boldocký et al., 2025).

Remark 1 ***(Differentiating through discrete operations)*** *In many approaches, the NNs are designed to directly output discrete values, using discrete operations such as rounding to produce those outputs. These discrete operations lead to non-differentiability and hinder the use of standard differentiable programming for network training. A common approach to address this issue is the straight-through estimator (STE) (Bengio et al., 2013), which enables backpropagation through discrete operations. During the forward pass, STE applies a non-differentiable operation to obtain discrete values. During the backward pass, it bypasses the non-existent gradients of these operations by replacing them with those of smooth surrogate functions. This approach was used in*

self-supervised learning frameworks for mixed-integer programming (Tang et al., 2024; Boldocký et al., 2025). We also use this technique in our framework.

We observe that the strength of SL in finding global solutions corresponds to the weakness of SSL, and vice versa, the strength of SSL in improving feasibility corresponds to the weakness of SL. Therefore, an interesting idea is to combine SL and SSL to exploit the benefits of both approaches.

3. Proposed Framework

In this section, we present an L2O framework for MIQPs in which a differentiable QP layer is incorporated, and a hybrid loss function combining SL and SSL is proposed.

3.1. Differentiable QP Layers for Feasible and Infeasible Problems

Given known $\boldsymbol{\delta}^*$ and $\boldsymbol{\theta}$, the optimal solution of the continuous variables can be obtained by solving the QP problem (3), if it is feasible. Thus, we can consider the QP problem as a differentiable layer within deep learning architectures, denoted by $\boldsymbol{x} = \mathrm{QP}(\boldsymbol{\delta}, \boldsymbol{\theta})$. Therefore, it leads to the following problem in which we approximate the integer solutions by NNs, and the continuous solutions by a QP layer:

$$\begin{aligned} \underset{\boldsymbol{\omega}}{\text{minimize}} \quad & f(\boldsymbol{x}, \boldsymbol{\delta}; \boldsymbol{\theta}), && \text{(9a)} \\ \text{subject to} \quad & \boldsymbol{\delta} = \boldsymbol{\pi}_{\boldsymbol{\omega}}(\boldsymbol{\theta}), && \text{(9b)} \\ & \boldsymbol{x} = \mathrm{QP}(\boldsymbol{\delta}, \boldsymbol{\theta}), && \text{(9c)} \\ & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{\delta}; \boldsymbol{\theta}) \leq 0. && \text{(9d)} \end{aligned}$$

To ensure the validity of the L2O framework using a QP layer and differentiable programming, we need the following assumption.

Assumption 1 *The QP problem (3) is strictly convex.*

As stated in Amos and Kolter (2017, Theorem 1), Assumption 1 is needed to ensure that the QP layer is subdifferentiable everywhere, and differentiable at all but a measure-zero set of points, because the solution of a strictly convex QP is continuous. The question is how to compute the gradient of the optimal solution $\boldsymbol{x}^*$ with respect to the argument, i.e., $\frac{\partial \boldsymbol{x}^*}{\partial \boldsymbol{\xi}}$. If the problem is feasible, these derivatives can be obtained by differentiating the KKT conditions (sufficient and necessary conditions for optimality) of (3). However, since the methods in (Amos and Kolter, 2017; Agrawal et al., 2019) rely on KKT conditions, it assumes the QP problem is feasible. On the other hand, in our framework, the optimization problems might be infeasible during training, given different values of the integer variables from the NN. Thus, we cannot directly incorporate the differentiable QP layer in (Amos and Kolter, 2017; Agrawal et al., 2019) into our framework. In this section, we present a simple yet efficient way to handle infeasibility during training, as described below.

To this end, we introduce slack variables $\boldsymbol{s}$ for the constraints, leading to the following QP:

$$\begin{aligned} \underset{\boldsymbol{x} \in \mathcal{X}, \boldsymbol{s}}{\text{minimize}} \quad & f(\boldsymbol{x}; \boldsymbol{\xi}) + \rho \, \mathbf{1}^\top \boldsymbol{s}, && \text{(10a)} \\ \text{subject to} \quad & \boldsymbol{g}_c(\boldsymbol{x}; \boldsymbol{\xi}) \leq \boldsymbol{s}, && \text{(10b)} \\ & \boldsymbol{s} \geq 0. && \text{(10c)} \end{aligned}$$

For ease of notations, in the rest of this section, we omit the argument $\boldsymbol{\xi}$ while mentioning the terms involving it. Since (10) is feasible given any realization of $\boldsymbol{\xi}$ as long as the domain for $\boldsymbol{x}$ is non-empty, we can apply the KKT conditions. First, we formulate the Lagrangian of (10) as follows:

$$L(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{\mu}, \boldsymbol{\kappa}) = \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}_x \boldsymbol{x} + \boldsymbol{p}_x^\top \boldsymbol{x} + \rho\, \mathbf{1}^\top \boldsymbol{s} + \boldsymbol{\mu}^\top (\boldsymbol{G}_x \boldsymbol{x} - \boldsymbol{h}_x - \boldsymbol{s}) - \boldsymbol{\kappa}^\top \boldsymbol{s}, \tag{11}$$

where $\boldsymbol{\mu} \geq 0$ and $\boldsymbol{\kappa} \geq 0$ are the dual variables on the constraints, and $\rho > 0$ is a penalty weight for the slack variables. The KKT conditions for stationarity, primal feasibility, and complementary slackness are

$$\boldsymbol{Q}_x \boldsymbol{x}^* + \boldsymbol{p}_x + \boldsymbol{G}_x^\top \boldsymbol{\mu}^* = 0, \tag{12a}$$
$$\rho\, \mathbf{1} - \boldsymbol{\mu}^* - \boldsymbol{\kappa}^* = 0, \tag{12b}$$
$$D(\boldsymbol{\mu}^*)\big(\boldsymbol{G}_x \boldsymbol{x}^* - \boldsymbol{h}_x - \boldsymbol{s}^*\big) = 0, \tag{12c}$$
$$D(\boldsymbol{\kappa}^*)\boldsymbol{s}^* = 0, \tag{12d}$$

where $D(\cdot)$ is the operation creating a diagonal matrix from a vector. Taking the differentials of the KKT conditions (12), we obtain

$$\mathsf{d}\boldsymbol{Q}_x \boldsymbol{x}^* + \boldsymbol{Q}_x \mathsf{d}\boldsymbol{x} + \mathsf{d}\boldsymbol{p}_x + \mathsf{d}\boldsymbol{G}_x^\top \boldsymbol{\mu}^* + \boldsymbol{G}_x^\top \mathsf{d}\boldsymbol{\mu} = 0, \tag{13a}$$
$$\mathsf{d}\boldsymbol{\mu} + \mathsf{d}\boldsymbol{\kappa} = 0, \tag{13b}$$
$$D\big(\boldsymbol{G}_x \boldsymbol{x}^* - \boldsymbol{h}_x - \boldsymbol{s}^*\big)\mathsf{d}\boldsymbol{\mu} + D(\boldsymbol{\mu}^*)(\mathsf{d}\boldsymbol{G}_x \boldsymbol{x}^* + \boldsymbol{G}_x \mathsf{d}\boldsymbol{x} - \mathsf{d}\boldsymbol{h}_x - \mathsf{d}\boldsymbol{s}) = 0, \tag{13c}$$
$$D(\boldsymbol{s}^*)\mathsf{d}\boldsymbol{\kappa} + D(\boldsymbol{\kappa}^*)\mathsf{d}\boldsymbol{s} = 0, \tag{13d}$$

or in the matrix form as follows:

$$\begin{bmatrix} \boldsymbol{Q}_x & \mathbf{0} & \boldsymbol{G}_x & \mathbf{0} \\ D(\boldsymbol{\mu}^*)\boldsymbol{G}_x & -D(\boldsymbol{\mu}^*) & D\big(\boldsymbol{G}_x \boldsymbol{x}^* - \boldsymbol{h}_x - \boldsymbol{s}^*\big) & \mathbf{0} \\ \mathbf{0} & D(\boldsymbol{\kappa}^*) & \mathbf{0} & D(\boldsymbol{s}^*) \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \mathsf{d}\boldsymbol{x} \\ \mathsf{d}\boldsymbol{s} \\ \mathsf{d}\boldsymbol{\mu} \\ \mathsf{d}\boldsymbol{\kappa} \end{bmatrix} = \begin{bmatrix} -\mathsf{d}\boldsymbol{Q}_x \boldsymbol{x}^* - \mathsf{d}\boldsymbol{p}_x - \mathsf{d}\boldsymbol{G}_x^\top \boldsymbol{\mu}^* \\ -D(\boldsymbol{\mu}^*)(\mathsf{d}\boldsymbol{G}_x \boldsymbol{x}^* - \mathsf{d}\boldsymbol{h}_x) \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{14}$$

Using these equations, we can form the Jacobians of $\boldsymbol{x}^*$ and $\boldsymbol{s}^*$ with respect to any of the problem parameters. The details can be found in (Amos and Kolter, 2017), (Agrawal et al., 2019).

Note that given any $\boldsymbol{x}$, $\boldsymbol{s}^* = \max(0, \boldsymbol{g}_c(\boldsymbol{x}))$. In other words, the use of slack variables in (10) is equivalent to using the max penalty function. The following theorem shows that if the original QP (3) is feasible, then by choosing a sufficiently large value for ρ, solving (10) yields the same solution as (3).

Theorem 2 *If the original QP (3) is feasible and let $\boldsymbol{x}'^*$ and $\boldsymbol{\mu}'^*$ be the optimal solutions and multipliers of the problem, respectively. If the penalty weight ρ is chosen such that $\rho > \left\|\boldsymbol{\mu}'^*\right\|_\infty$, then the optimal solutions of (10) and the original QP (3) are the same.*

The proof of this theorem follows directly from Proposition 5.25 in (Bertsekas, 2014) and is therefore omitted. Therefore, if the original QP is feasible, we get the exact derivative of the optimal solution by using the KKT conditions of the relaxed problem. In the infeasible case, we show in the following theorem that, under some mild conditions, the obtained solution from (10) is a point that minimizes the constraint violation, and among all the solutions with minimal constraint violation, the obtained solution also minimizes the original objective function.

Theorem 3 *If the original QP (3) is infeasible and assume that either one of the following properties hold:*

- $\boldsymbol{Q}_x \succ 0$, *which means* $f(\boldsymbol{x})$ *is coercive.*
- $\mathcal{X}$ *is compact (closed and bounded).*

Let us denote $v(\boldsymbol{x}) := \mathbf{1}^\top \max(0, \boldsymbol{g}_c(\boldsymbol{x}))$. *If the penalty weight* ρ *is chosen sufficiently large, then (i) a minimizer* $(\boldsymbol{x}_\rho^*, \boldsymbol{s}_\rho^*)$ *of (10) achieves minimal total violation, i.e.,* $v(\boldsymbol{x}_\rho^*) = v^*$ *and (ii)*

$$\boldsymbol{x}_\rho^* \in \underset{\boldsymbol{x}\in\mathcal{X}}{\arg\min} \{f(\boldsymbol{x}) \,:\, v(\boldsymbol{x}) = v^*\}. \tag{15}$$

The proof for Theorem 3 is given in Appendix A.

3.2. Hybrid Training Loss

Our training framework relies on a *hybrid loss function* that combines supervised and self-supervised losses. The main advantage of this hybrid loss is that it leverages the strengths of supervised learning (SL) in achieving global solution optimality and self-supervised learning (SSL) in improving constraint satisfaction. The hybrid loss used to train the neural network can be defined as follows:

$$\underset{\boldsymbol{\omega}}{\text{minimize}} \;\; \frac{1}{M}\sum_{i=1}^{M} \gamma^{\text{obj}} f(\boldsymbol{x}^i, \boldsymbol{\delta}^i; \boldsymbol{\theta}^i) + \gamma^{\text{con}} \boldsymbol{\lambda}^\top \max\left(0, \boldsymbol{g}(\boldsymbol{x}^i, \boldsymbol{\delta}^i; \boldsymbol{\theta}^i)\right) + \gamma^{\text{sup}} \mathcal{L}(\boldsymbol{\delta}^i, \boldsymbol{\delta}^{i,*}), \tag{16}$$

where γ^{obj}, γ^{con}, and $\gamma^{\text{sup}} \in \mathbb{R}_{\geq 0}$ are the weights for the objective value, constraint violation, and supervised loss, respectively. Note that in our framework, the constraints involving at least one continuous variable can be directly handled using the differentiable QP layer, while the constraints involving only the integer variables must be incorporated into the loss function.

4. Results and Discussions

We validate our hybrid L2O framework on two benchmark MI-MPC problems: (i) collision avoidance for robot navigation and (ii) simplified thermal energy tanks (Boldocký et al., 2025). In the first example, binary variables are used to formulate the collision-avoidance constraints, thus, there are several coupling constraints between the integer and continuous variables. Meanwhile, the second example involves integer decision variables and demonstrates the case where the integers appear in the objective function. The details of the two examples are provided in Appendix B. For each example, a multilayer perceptron network is constructed with four hidden layers, 128 neurons per layer, and ReLU activation functions. Our implementation and examples are available at https://github.com/mlab-upenn/L2O-MIQP.

We compare the proposed hybrid L2O framework with an SL model and an SSL model. Note that the SSL model considered here follows the architecture of our framework with the differentiable QP layer, rather than the conventional design in which the NN is trained to predict both integer and continuous solutions. However, the loss function used to train the SSL model does not include the supervised term. We evaluate the trained models using two metrics: constraint violation and optimality gap. We separate the violations into those associated with constraints involving continuous variables and those involving only integer variables. The optimality gap is expressed as a

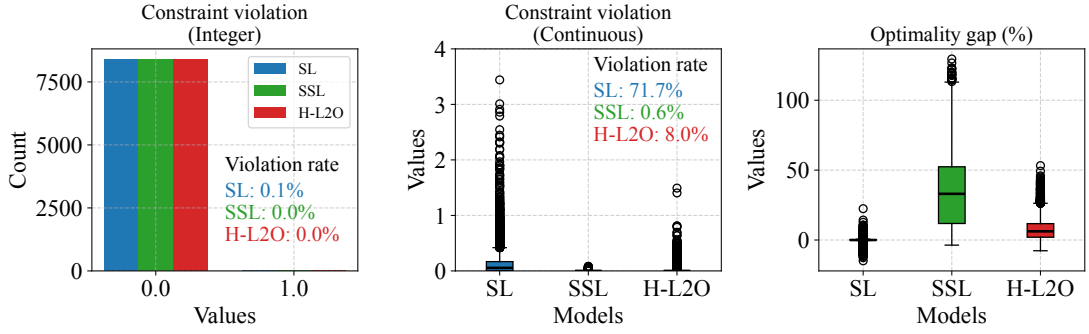

Figure 2: Statistical comparison of the three models: hybrid L2O (H-L2O), supervised learning (SL), and self-supervised learning (SSL), for the robot navigation example.

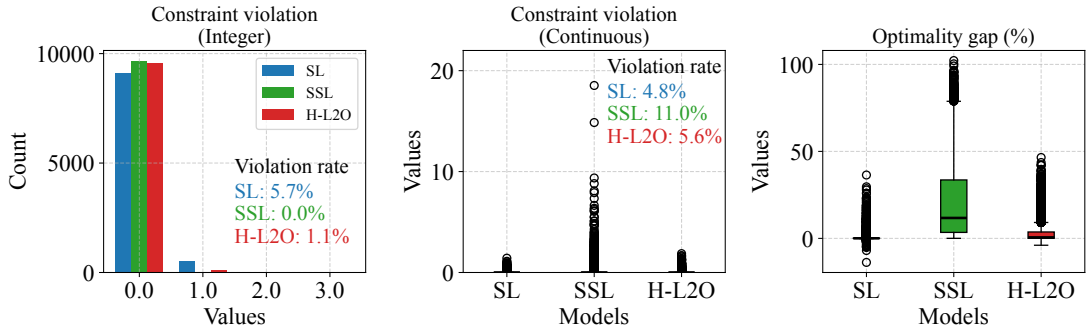

Figure 3: Statistical comparison of the three models: hybrid L2O (H-L2O), supervised learning (SL), and self-supervised learning (SSL), for thermal energy tank example.

percentage, computed as the ratio of the objective gap to the optimal objective value. We show the statistical comparison for the two examples in Figures 2 and 3, respectively. In each figure, the left panel shows the violations for integer-only constraints in the form of barplots, while the boxplots in the middle and right panels show the continuous-constraint violations and the optimality gap, respectively. For the figures showing constraint violations, we also report the violation rate, i.e., the percentage of validation problems in which the obtained solutions violate the constraints. For the robot navigation example, the results indicate that all three models satisfy the constraints involving only integer variables. However, for continuous-constraint violations, the SL model fails to ensure constraint satisfaction in 71.7% of the test cases (a consequence of SL training only for the integer solution, which does not guarantee a feasible continuous solution from the subsequent QP), whereas the hybrid L2O and SSL models exhibit much smaller violation rates of 8% and 0.6%, respectively, demonstrating improved constraint satisfaction. The plot of the optimality gap shows that the hybrid L2O model achieves better optimality than the SSL model, although it exhibits a slightly larger gap than the SL model. A similar trend in optimality is observed in the second example. Regarding constraint satisfaction, the hybrid L2O model outperforms the SL model for constraints involving only integer variables, achieving a lower violation rate of 1.1% compared to 5.7%. Meanwhile, the two models achieve comparable levels of satisfaction for continuous constraints (5.6% vs. 4.8% violation rate). In contrast, the SSL model perfectly satisfies the integer-only constraints but performs poorly on the continuous ones. Overall, the results show that the proposed hybrid L2O frame-

Table 1: Average solving time and standard deviation for GUROBI solver and the L2O solver.

Problem	Solver	Avg. time (ms)	Std. dev. (ms)
Robot navigation example	GUROBI	69.49	19.06
	L2O	7.55	1.31
Energy tank example	GUROBI	15.40	8.61
	L2O	1.31	0.39

work effectively balances feasibility and optimality, achieving optimality comparable to supervised learning while improving constraint satisfaction.

Finally, we report the computation times in Table 1, comparing the L2O approach (either SL, SSL, or the hybrid L2O) with the GUROBI solver (Gurobi Optimization, LLC, 2021). Note that the computation time for L2O approach includes both the NN prediction time and the time required to solve the relaxed QP problem. The results confirm that the L2O approach significantly reduces the overall solving time compared to a state-of-the-art MIQP solver. In particular, the L2O approach achieves approximately $9\times$ and $12\times$ faster computation for the robot navigation and energy tank examples, respectively.

Although our proposed hybrid L2O framework demonstrates some benefits over SL and SSL approaches, it still has certain limitations that can be addressed in future research. First, the choice of weights in the hybrid loss function significantly affects the optimality and feasibility performance and currently requires manual tuning. Second, integrating the differentiable QP layer considerably increases the training time compared to purely supervised learning. Finally, the current framework is limited to MIQPs, while integrating differentiable optimization layers into general mixed-integer convex or nonlinear programming problems remains an open challenge.

5. Conclusions

In this work, we developed a hybrid L2O framework for MIQPs that integrates a differentiable QP layer and a hybrid loss function combining supervised learning and self-supervised learning. We designed a model architecture in which a neural network learns the mapping from problem parameters to optimal integer variables, while the differentiable QP layer computes the corresponding continuous variables. This architecture enables the optimization structure to be incorporated into the learning process. To balance solution optimality and constraint feasibility during training, we defined a hybrid loss that linearly combines supervised and self-supervised terms. We validated the framework on two benchmark MPC examples, showing that the hybrid L2O approach effectively balances optimality and feasibility: it achieves better constraint satisfaction than purely supervised learning and better optimality than purely self-supervised learning. Future work will focus on extending the framework to mixed-integer convex and nonlinear programming problems.

Acknowledgments

This work was partially supported by US DoT Safety21 National University Transportation Center and NSF grants CISE-2431569.

References

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International conference on machine learning*, pages 136–145. PMLR, 2017.

Calin Belta and Sadra Sadraddini. Formal methods for control synthesis: An optimization perspective. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):115–140, 2019.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Dimitri P Bertsekas. *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

Ján Boldockỳ, Shahriar Dadras Javan, Martin Gulan, Martin Mönnigmann, and Ján Drgoňa. Learning to solve parametric mixed-integer optimal control problems via differentiable predictive control. *arXiv preprint arXiv:2506.19646*, 2025.

Andrea Camisa, Andrea Testa, and Giuseppe Notarstefano. Multi-robot pickup and delivery via distributed resource allocation. *IEEE Transactions on Robotics*, 39(2):1106–1118, 2022.

Abhishek Cauligi, Preston Culbertson, Edward Schmerling, Mac Schwager, Bartolomeo Stellato, and Marco Pavone. Coco: Online mixed-integer control via supervised learning. *IEEE Robotics and Automation Letters*, 7(2):1447–1454, 2021.

Abhishek Cauligi, Ankush Chakrabarty, Stefano Di Cairano, and Rien Quirynen. Prism: Recurrent neural networks and presolve methods for fast mixed-integer optimal control. In *Learning for Dynamics and Control Conference*, pages 34–46. PMLR, 2022.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2021. URL http://www.gurobi.com.

Viet-Anh Le and Andreas A Malikopoulos. Distributed Optimization for Traffic Light Control and Connected Automated Vehicle Coordination in Mixed-Traffic Intersections. *IEEE Control Systems Letters*, 8:2721–2726, 2024.

Viet-Anh Le, Panagiotis Kounatidis, and Andreas A. Malikopoulos. Combining Graph Attention Networks and Distributed Optimization for Multi-Robot Mixed-Integer Convex Programming. In *2025 64th IEEE Conference on Decision and Control*, 2025.

Rien Quirynen, Sleiman Safaoui, and Stefano Di Cairano. Real-time mixed-integer quadratic programming for vehicle decision-making and motion planning. *IEEE Transactions on Control Systems Technology*, 2024.

João Salvado, Robert Krug, Masoumeh Mansouri, and Fedorico Pecora. Motion planning and goal assignment for robot fleets using trajectory optimization. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7939–7946. IEEE, 2018.

Bo Tang, Elias B Khalil, and Ján Drgoňa. Learning to optimize for mixed-integer non-linear programming. *arXiv preprint arXiv:2410.11061*, 2024.

Appendix A. Proof of Theorem 3

Proof We first prove (ii) given assuming that $\boldsymbol{x}_\rho^* \in S_v$. Let $S_v = \{\boldsymbol{x} \,|\, v(\boldsymbol{x}) = v^*\}$ be the set of all points that achieve this minimum violation. S_v is closed since it is a level set of a continuous function. Combining with the condition that either $f(\boldsymbol{x})$ is coercive or $\mathcal{X}$ is compact, there exist a minimizer of $f(\boldsymbol{x})$ on S_v. From the definition of $\boldsymbol{x}_\rho^*$, we have

$$\boldsymbol{x}_\rho^* \in \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x}) + \rho v(\boldsymbol{x}), \tag{17}$$

which means $\boldsymbol{x}_\rho^*$ is also a minimizer of $f(\boldsymbol{x}) + \rho v(\boldsymbol{x})$ on S_v, i.e.,

$$\boldsymbol{x}_\rho^* \in \arg\min_{\boldsymbol{x}\in S_v} f(\boldsymbol{x}) + \rho v(\boldsymbol{x}) = \arg\min_{\boldsymbol{x}\in S_v} f(\boldsymbol{x}) + \rho v^*. \tag{18}$$

Since ρv^* is a constant, then $\boldsymbol{x}_\rho^* \in \arg\min_{\boldsymbol{x}\in S_v} f(\boldsymbol{x})$, or $\boldsymbol{x}_\rho^*$ is a minimizer of $f(\boldsymbol{x})$ on S_v.

For (i), we prove that for a sufficiently large but finite ρ, $\boldsymbol{x}_\rho^*$ must be in S_v. Let us assume, for the sake of contradiction, $\boldsymbol{x}_\rho^* \notin S_v$. From (17), we have

$$f(\boldsymbol{x}_\rho^*) + \rho v(\boldsymbol{x}_\rho^*) \leq f(\boldsymbol{x}^{**}) + \rho v(\boldsymbol{x}^{**}) = f(\boldsymbol{x}^{**}) + \rho v^*, \tag{19}$$

for any $\boldsymbol{x}^{**} \in S_v$, which leads to

$$\rho \leq \frac{f(\boldsymbol{x}^{**}) - f(\boldsymbol{x}_\rho^*)}{v(\boldsymbol{x}_\rho^*) - v^*}. \tag{20}$$

Next, since $v(\cdot)$ is a convex piecewise linear function, the Hoffman error bound property holds, i.e.,

$$v(\boldsymbol{x}_\rho^*) - v^* \geq c\, \mathrm{dist}(\boldsymbol{x}_\rho^*, S_v), \tag{21}$$

with $c > 0$. Moreover, $f(\boldsymbol{x})$ is a quadratic function, so it is Lipschitz continuous, and we have the following inequality

$$|f(\boldsymbol{x}^{**}) - f(\boldsymbol{x}_\rho^*)| \leq L_f \cdot \|\boldsymbol{x}^{**} - \boldsymbol{x}_\rho^*\|. \tag{22}$$

Therefore, if we choose $\boldsymbol{x}^{**} \in S_v$ such that $\mathrm{dist}(\boldsymbol{x}_\rho^*, S_v) = \|\boldsymbol{x}^{**} - \boldsymbol{x}_\rho^*\|$, i.e., $\boldsymbol{x}^{**}$ is the closest point in S_v to $\boldsymbol{x}$, then from (19), we obtain that $\rho \leq \frac{L_f}{c}$ must be satisfied. Thus, we can select ρ such that $\rho > \frac{L_f}{c}$, which leads to a contradiction. The proof is thus complete. ■

Appendix B. Details of Numerical Examples

B.1. Collision Avoidance for Robot Navigation

In this example, we consider a navigation problem for a single robot operating in an environment with stationary obstacles. The robot is required to move from its initial position to a designated

goal while avoiding collisions with obstacles. This problem is formulated as an MIQP, where binary variables are used to formulate the collision avoidance constraints between the robot and the obstacles. The corresponding MI-MPC formulation follows the setup described in (Le et al., 2025).

We consider an MPC problem with a single robot and n_o obstacles, and let $\mathcal{O}$ be the set of obstacles. We formulate the MPC problem with a control horizon of length H. Let $t \in \mathbb{Z}^+$ be the current time step. At every time step $k \in \mathbb{Z}^+$, let $\boldsymbol{p}(k) = [p^x(k), p^y(k)]^\top \in \mathbb{R}^2$, $\mathbf{v}(k) = [v^x(k), v^y(k)]^\top \in \mathbb{R}^2$, and $\boldsymbol{u}(k) = [u^x(k), u^y(k)]^\top \in \mathbb{R}^2$ be the vectors of positions, velocities, and accelerations for robot, respectively. Let $\boldsymbol{x}(k) = [\boldsymbol{p}(k), \mathbf{v}(k)]^\top$ be the state vector of robot. The dynamics of each robot are governed by a discrete-time double-integrator model as follows,

$$\begin{aligned} \boldsymbol{p}(k+1) &= \boldsymbol{p}(k) + \tau \boldsymbol{v}(k) + \frac{1}{2}\tau^2 \boldsymbol{u}(k), \\ \boldsymbol{v}(k+1) &= \boldsymbol{v}(k) + \tau \boldsymbol{u}(k), \end{aligned} \tag{23}$$

where $\tau \in \mathbb{R}_{>0}$ is the sampling time period, and compactly expressed as $\boldsymbol{x}(k+1) = \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k))$. We assume that the states and control inputs of robots are subjected to the following bound constraints:

$$\begin{aligned} p^x_{\min} \le p^x(k) \le p^x_{\max}, \quad p^y_{\min} \le p^y(k) \le p^y_{\max}, \\ -v_{\max} \le v^x(k), v^y(k) \le v_{\max}, \quad -a_{\max} \le u^x(k), u^y(k) \le a_{\max}, \end{aligned} \tag{24}$$

where $[p^x_{\min}, p^x_{\max}, p^y_{\min}, p^y_{\max}]^\top \in \mathbb{R}^4$ is the boundary of the environment, $v_{\max} \in \mathbb{R}_{>0}$ and $a_{\max} \in \mathbb{R}_{>0}$ are the maximum speed and acceleration of the robots, respectively. More compactly, (24) is expressed as $\boldsymbol{x}(k) \in \mathcal{X}$ and $\boldsymbol{u}(k) \in \mathcal{U}$.

The mixed-integer constraints for collision avoidance between the robot and obstacle $o \in \mathcal{O}$ at time-step k are formulated by big-M formulation as follows,

$$\begin{aligned} \cos\alpha_o(p^x(k+1) - p^x_o) + \sin\alpha_o(p^y(k+1) - p^y_o) &\ge L_o + d_{\min} - Mb_{1,o}(k), \\ -\sin\alpha_o(p^x(k+1) - p^x_o) + \cos\alpha_o(p^y(k+1) - p^y_o) &\ge W_o + d_{\min} - Mb_{2,o}(k), \\ -\cos\alpha_o(p^x(k+1) - p^x_o) - \sin\alpha_o(p^y(k+1) - p^y_o) &\ge L_o + d_{\min} - Mb_{3,o}(k), \\ \sin\alpha_o(p^x(k+1) - p^x_o) - \cos\alpha_o(p^y(k+1) - p^y_o) &\ge W_o + d_{\min} - Mb_{4,o}(k), \end{aligned} \tag{25}$$

where $b_{1,o}(k)$, $b_{2,o}(k)$, $b_{3,o}(k)$ and $b_{4,o}(k)$ are binary decision variables satisfying

$$b_{1,o}(k) + b_{2,o}(k) + b_{3,o}(k) + b_{4,o}(k) \le 3, \tag{26}$$

$[p^x_o, p^y_o]^\top$ is the center location, α_o is the rotation angle, and $2L_o$ and $2W_o$ are the length and width of obstacle–$o \in \mathcal{O}$, respectively, while $d_{\min}$ is the minimal distance between robot and obstacle to be considered as no collision. We define the binary decision variables $\boldsymbol{\delta}(k) \in \{0,1\}^{4n_o}$ at each time step k as the concatenated vector of $b_{1,o}(k), b_{2,o}(k), b_{3,o}(k), b_{4,o}(k)$, for all $o \in \mathcal{O}$, and rewrite all the collision avoidance constraints as $\boldsymbol{g}_o(\boldsymbol{x}_{k+1}, \boldsymbol{\delta}_k) \le 0$.

The objective for the robot is to reach the goal, i.e., minimize the distance to the goal, while maintaining the minimum effort. Thus, we consider the following MPC cost given by a weighted sum of terminal cost $\bar{c}$ and running cost c over the horizon i.e.,

$$\underset{\substack{\boldsymbol{x}(k+1)\in\mathcal{X},\\ \boldsymbol{u}(k)\in\mathcal{U}}}{\text{minimize}} \ \bar{c}\big(\boldsymbol{x}(t+H)\big) + \sum_{k=t}^{t+H-1} c\big(\boldsymbol{u}(k), \boldsymbol{x}(k)\big), \tag{27}$$

where

$$\begin{aligned}\bar{c}(\boldsymbol{x}(t+H)) &= \omega_{\mathrm{pt}} \left\|\boldsymbol{p}(t+H) - \boldsymbol{p}_{\mathrm{g}}\right\|_2^2, \\ c(\boldsymbol{u}(k), \boldsymbol{x}(k)) &= \omega_{\mathrm{p}} \left\|\boldsymbol{p}(k) - \boldsymbol{p}_{\mathrm{g}}\right\|_2^2 + \omega_{\mathrm{u}} \left\|\boldsymbol{u}(k)\right\|_2^2\end{aligned} \tag{28}$$

with $\boldsymbol{p}_{\mathrm{g}}$ being the vector of goal positions, while ω_{pt}, ω_{p}, and ω_{u} are positive weights. Consequently, the cost function is quadratic in the continuous decision variables.

Therefore, the parametric MI-MPC problem can be given by

$$\begin{aligned}\text{minimize} \quad & \bar{c}(\boldsymbol{x}_H; \boldsymbol{\theta}) + \sum_{k=0}^{H-1} c(\boldsymbol{x}_k, \boldsymbol{u}_k; \boldsymbol{\theta}) \\ \text{subject to} \quad & \boldsymbol{x}_0 = \boldsymbol{x}_{\text{init}}(\boldsymbol{\theta}), \\ & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \\ & \boldsymbol{g}_o(\boldsymbol{x}_{k+1}, \boldsymbol{\delta}_k) \leq 0, \\ & \boldsymbol{x} \in \mathcal{X}^{H+1}, \ \boldsymbol{u} \in \mathcal{U}^H, \ \boldsymbol{\delta}_k \in \{0,1\}^{4n_o}.\end{aligned} \tag{29}$$

where the parameter vector $\boldsymbol{\theta} = [\boldsymbol{x}_0^\top, \boldsymbol{p}_{\mathrm{g}}^\top]^\top \in \mathbb{R}^6$ contains the initial state and goal position. We consider an MPC problem with three obstacles and a control horizon of length 20, leading to $3 \times 20 \times 4 = 240$ binary decision variables. An NN $\boldsymbol{\pi}_{\boldsymbol{\omega}} : \mathbb{R}^6 \to \{0,1\}^{240}$ is employed to predict the binary decision variables. The parameters for the simulation are set as follows: $\tau = 0.25\,\mathrm{s}$, $[p^x_{\min}, p^x_{\max}, p^y_{\min}, p^y_{\max}]^\top = [-0.5\,\mathrm{m}, 3\,\mathrm{m}, -3\,\mathrm{m}, 0.5\,\mathrm{m}]^\top$, $v_{\max} = 0.5\,\mathrm{m/s}$, $a_{\max} = 0.5\,\mathrm{m/s^2}$, $d_{\min} = 0.25\,\mathrm{m}$, $M = 10^3$, $\omega_{\mathrm{pt}} = 10$, $\omega_{\boldsymbol{p}} = 1$, $\omega_{\mathrm{u}} = 1$, $w_s = 10^4$. The information of the three obstacles is:

1. Obstacle 1: $p_1^x = 1.0\,\mathrm{m}$, $p_1^y = 0.0\,\mathrm{m}$, $L_1 = 0.8\,\mathrm{m}$, $W_1 = 1.0\,\mathrm{m}$, $\alpha_1 = 0.0\,\mathrm{rad}$.
2. Obstacle 2: $p_2^x = 0.7\,\mathrm{m}$, $p_2^y = -1.1\,\mathrm{m}$, $L_2 = 1.0\,\mathrm{m}$, $W_2 = 0.8\,\mathrm{m}$, $\alpha_2 = 0.0\,\mathrm{rad}$.
3. Obstacle 3: $p_3^x = 0.4\,\mathrm{m}$, $p_3^y = -2.5\,\mathrm{m}$, $L_3 = 0.8\,\mathrm{m}$, $W_3 = 1.0\,\mathrm{m}$, $\alpha_3 = 0.0\,\mathrm{rad}$.

B.2. Simplified Thermal Energy Tank System

In the second example, we examine a simplified thermal energy tank system, adapted from (Boldocký et al., 2025). The system dynamics are represented by a discrete-time linear time-invariant (LTI) model with both continuous and discrete control inputs, described as follows:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}_u\boldsymbol{u}_k + \boldsymbol{B}_d\delta_k + \boldsymbol{E}\boldsymbol{d}_k, \tag{30}$$

where $\boldsymbol{x}_k \in \mathbb{R}^2$ is the state vector, $\boldsymbol{u}_k \in \mathbb{R}^2$ is the continuous control input, $\delta_k \in \{0,1,2,3\}$ is the discrete control input, and $\boldsymbol{d}_k \in \mathbb{R}^2$ represents known disturbances at time step k. The dynamics matrices $\boldsymbol{A}$, $\boldsymbol{B}_u$, $\boldsymbol{B}_d$, and $\boldsymbol{E}$ are:

$$\boldsymbol{A} = \begin{bmatrix} 0.9983 & 0.001 \\ 0 & 0.9966 \end{bmatrix}, \quad \boldsymbol{B}_u = 0.075\,\mathbb{I}_2, \quad \boldsymbol{B}_d = \begin{bmatrix} 0 \\ 0.0825 \end{bmatrix}, \quad \boldsymbol{E} = -0.0833\,\mathbb{I}_2. \tag{31}$$

The system is subject to the following state and input constraints:

$$0 \leq x_{1,k} \leq 8.4, \quad 0 \leq x_{2,k} \leq 3.6, \quad 0 \leq u_{1,k}, u_{2,k} \leq 8. \tag{32}$$

In addition to the constraints on the continuous control inputs, we also impose constraints on the changes between consecutive time steps of the discrete control input as follows,

$$-1 \leq \delta_k - \delta_{k-1} \leq 1,\ k = 1, \ldots, H-1 \tag{33}$$

The control objective is to minimize the expected cumulative cost over the prediction horizon, which includes both state tracking and control effort penalties. The stage cost function and terminal cost function are defined as

$$\begin{aligned} \bar{c}(\boldsymbol{x}_k, \boldsymbol{u}_k, \delta_k; \boldsymbol{r}_k) &= \|\boldsymbol{x}_k - \boldsymbol{r}_k\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_k\|_{\boldsymbol{R}}^2 + \rho \|\delta_k\|_2^2, \\ c(\boldsymbol{x}_H; \boldsymbol{r}_H) &= \|\boldsymbol{x}_H - \boldsymbol{r}_H\|_{\boldsymbol{Q}_t}^2, \end{aligned} \tag{34}$$

where $\boldsymbol{r}_k$ is the reference state at time step k, and $\boldsymbol{Q}$, $\boldsymbol{R}$, $\boldsymbol{Q}_t$ and ρ are weighting matrices and vectors, respectively, given by $\boldsymbol{Q} = \boldsymbol{Q}_t = \mathbb{I}_2$, $\boldsymbol{R} = 0.5\,\mathbb{I}_2$, and $\rho = 0.1$. For simplicity, we assume that the reference state $\boldsymbol{r}_k$ remains constant over the prediction horizon, i.e., $\boldsymbol{r}_k = \boldsymbol{r}$ for all k, where $\boldsymbol{r} = [4.2, 1.8]^\top$. Thus, the optimization problem is parameterized by the current state $\boldsymbol{x}_t$ and the sequence of known future disturbances over the prediction horizon, $[\boldsymbol{d}_k, \boldsymbol{d}_{k+1}, \ldots, \boldsymbol{d}_{k+H-1}]$. Accordingly, the parameter vector is defined as $\boldsymbol{\theta} = [\boldsymbol{x}_k^\top, \boldsymbol{d}_k^\top, \boldsymbol{d}_{k+1}^\top, \ldots, \boldsymbol{d}_{k+H-1}^\top] \in \mathbb{R}^{2H+2}$.

The parametric MI-MPC problem is given by

$$\begin{aligned} \text{minimize} \quad & \bar{c}(\boldsymbol{x}_H; \boldsymbol{r}_H) + \sum_{k=0}^{H-1} c(\boldsymbol{x}_k, \boldsymbol{u}_k, \delta_k; \boldsymbol{r}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}_u\boldsymbol{u}_k + \boldsymbol{B}_d\delta_k + \boldsymbol{E}\boldsymbol{d}_k, \\ & -1 \leq \delta_k - \delta_{k-1} \leq 1,\ k = 1, \ldots, H-1 \\ & \boldsymbol{x}_k \in \mathcal{X},\ \boldsymbol{u}_k \in \mathcal{U},\ \delta_k \in \{0, 1, 2, 3\}. \end{aligned} \tag{35}$$

An NN is trained to predict the integer control inputs, i.e., $\boldsymbol{\pi}_{\boldsymbol{\omega}} : \mathbb{R}^{2H+2} \to \{0, 1, 2, 3\}^H$. For this example, we set the control horizon to $H = 20$, resulting in a parameter vector $\boldsymbol{\theta} \in \mathbb{R}^{42}$ and an NN output dimension of 20.