# Learning to Optimize for Mixed-Integer Non-linear Programming with Feasibility Guarantees

**Bo Tang**
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, ON M5S 1A1, Canada
bo.tang@mail.utoronto.ca

**Elias B. Khalil**
Department of Mechanical and Industrial Engineering
University of Toronto
Toronto, ON M5S 1A1, Canada
elias.khalil@mie.utoronto.ca

**Ján Drgoňa**
The Ralph O'Connor Sustainable Energy Institute
Johns Hopkins University
Baltimore, MD 21218, USA
jdrgona1@jh.edu

## Abstract

Mixed-integer nonlinear programs (MINLPs) arise in domains as diverse as energy systems and transportation, but are notoriously difficult to solve, particularly at scale. While learning-to-optimize (L2O) methods have been successful at continuous optimization, extending them to MINLPs is challenging due to integer constraints. To overcome this, we propose a novel L2O approach with two integer correction layers to ensure the integrality of the solution and a projection step to ensure the feasibility of the solution. We prove that the projection step converges, providing a theoretical guarantee for our method. Our experiments show that our methods efficiently solve MINLPs with up to tens of thousands of variables, providing high-quality solutions within milliseconds, even for problems where traditional solvers and heuristics fail. This is the first general L2O method for parametric MINLPs, finding solutions to some of the largest instances reported to date.

## 1 Introduction

Mixed-integer optimization emerges in a broad spectrum of real-world applications involving discrete decisions, such as pricing [1], battery dispatch [2], transportation [3], and optimal control [4]. While mixed-integer *linear* programs (MILPs) benefit from decades of development in both exact algorithms [5] and heuristic methods [6, 7], many practical problems are inherently nonlinear. Unlike MILPs, mixed-integer nonlinear programs (MINLPs) pose greater challenges due to the interplay between discrete variables and nonlinear constraints or objectives. Despite the availability of classical techniques such as outer approximation [8], spatial branch-and-bound [9], and decomposition techniques [10], these approaches often fail to scale efficiently to large or real-time applications.

Preprint.

In many applications, such as real-time bidding, energy dispatch, and robotics, MINLPs is required to solve repeatedly under strict time constraints and varying input parameters. This has motivated a growing interest in *learning-to-optimize* (L2O) methods, which leverage machine learning to improve or even replace traditional optimization pipelines. Among these, end-to-end optimization methods aim to directly learn a mapping from input features or problem parameters to decision variables [11–13], thus eliminating the need for computationally intensive solvers, offering substantial gains in computational efficiency and scalability.

A critical challenge in deploying L2O methods to real-world problems lies in satisfying hard constraints. This has led to a growing body of research on improving feasibility in learning-based optimization frameworks. Prior efforts include encoding linear constraints into neural architectures [14], applying penalty-based relaxation techniques for soft constraint satisfaction [15, 16], incorporating gauge maps to respect geometric constraints [17], and performing projections onto feasible sets [18]. More recent approaches explore domain-specific architectures for structured outputs [19]. However, these techniques largely assume continuous domains and cannot be directly extended to handle integrality requirements, which remain a central barrier in applying L2O to MINLPs.

To bridge the gap between integer feasibility and learning-based methods, we propose the first general-purpose L2O framework for solving parametric MINLPs in an end-to-end fashion. A key challenge in this setting lies in the inherent non-differentiability w.r.t. integer variables, which undermines gradient-based training and has thus been largely overlooked in prior L2O literature. To overcome this, we design two novel differentiable correction layers that enable gradient flow through integer outputs, allowing neural networks to generate and refine discrete solutions via backpropagation. In addition, we introduce a lightweight projection heuristic that uses gradient signals from the correction layers to further improve constraint satisfaction. We further provide theoretical guarantees on feasibility recovery and empirically validate our approach on challenging benchmark problems, demonstrating its capability to generate high-quality solutions at unprecedented scales.

## 2 Related Work

**End-to-end optimization.** End-to-end optimization aims to directly predict optimization solutions using machine learning models, which bypasses traditional solvers to gain computational efficiency. Early work by Hopfield and Tank [20] used Hopfield networks with Lagrangian penalties to solve the traveling salesperson problem. More recently, Fioretto et al. [21] extended the penalty function to nonlinear energy systems, and Pan et al. [22] further embedded certain constraints directly into neural network architecture and output. While these supervised methods reduce inference latency significantly, they rely heavily on large datasets of precomputed optimal solutions [23, 24], which can be prohibitively expensive to generate for large-scale problems. This challenge has driven the development of self-supervised learning approaches [18], which train models by minimizing the objective function and constraint violation directly from model predictions, without optimal solutions as ground-truth labels.

Our work is the first to apply self-supervised learning to MINLPs, enabling fast, label-free training and real-time inference, even at a large scale.

**Constrained neural architectures.** While learning-based optimization has seen increasing interest, the handling of constraints remains a challenging and relatively underexplored area. One line of work attempts to incorporate hard constraints directly into network architectures. For example, Hendriks et al. [14] incorporated linear operator constraints, while Vinyals et al. [25] and Dai et al. [26] leveraged the graph structure to construct feasible solutions for the traveling salesperson problem. Additionally, Kervadec et al. [27] demonstrated that using the log-barrier method for inequality constraints can improve accuracy, constraint satisfaction, and training stability. In contrast to the above hard constraints, penalty methods [15, 16], which incorporate inequality constraints through regularization terms in the loss function, have also gained popularity. Although these methods lack formal guarantees, they often outperform their hard constraint counterparts in practice [28]. Building on penalty methods, Donti et al. [18] proposed differentiable correction and completion layers to handle equality and inequality constraints.

Unlike existing approaches focused on continuous domains, our method introduces two differentiable correction layers to enforce integrality, along with a post-hoc projection step that further ensures both integrality and constraint satisfaction.

**Learning for mixed-integer programming.** A substantial body of work applies ML to accelerate exact mixed-integer linear programming (MILP) solvers. This includes parameter tuning [29], preprocessing [30], branching variable selection [31–34], node selection [35], heuristic selection [36], and cut selection and generation [37, 38]. Another line of research relates to learning to heuristically generate solutions for integer linear programs [39–46]. We refer to the surveys of Bengio et al. [47] and Zhang et al. [48] for more details. In contrast, learning methods for MINLP remain relatively underexplored. Existing work includes supervised approaches for cut selection in quadratic programs [49], graph neural networks for solving quadratic assignment problems [50], and classification-based strategies for linearizing MIQPs [51]. Surrogate-based approximations have also been explored, such as the linearization method proposed in SurCO [52]. NeuralQP [53] combines solution prediction for integer QPs with a solver-based refinement step.

In contrast to prior methods, our approach targets general parametric MINLPs, generates fully end-to-end integer solutions without any optimization solvers, thus supporting highly efficient training and inference.

**Differentiable optimization.** A related but distinct category of methods integrates optimization solvers directly into neural networks as differentiable layers [54, 55]. These methods enable gradient-based training for problems involving linear [56], quadratic [57, 58], stochastic [59], and submodular optimization [60], as well as integer linear programs [61–64]. These solver-in-the-loop approaches are effective at improving decision quality, especially for prediction under uncertainty. However, they require solving an optimization problem during every training step, which incurs significant computational cost. As noted by Tang and Khalil [65], this limits their practicality for large-scale or real-time applications.

Rather than relying on solver-based layers as in differentiable optimization, our method trains a solver-free solution predictor that achieves high efficiency and scalability.

## 3 Preliminaries

**Learning problem formulation.** A generic training formulation for learning-to-optimize with parametric MINLPs is given by:

$$\min_{\Theta} \mathbb{E}\big[f(\hat{\mathbf{x}}, \boldsymbol{\xi})\big] \approx \frac{1}{m}\sum_{i=1}^{m} f(\hat{\mathbf{x}}^i, \boldsymbol{\xi}^i) \quad \text{s.t. } \mathbf{g}(\hat{\mathbf{x}}^i, \boldsymbol{\xi}^i) \leq 0,\ \hat{\mathbf{x}}^i \in \mathbb{R}^{n_r} \times \mathbb{Z}^{n_z},\ \hat{\mathbf{x}}^i = \boldsymbol{\psi}_\Theta(\boldsymbol{\xi}^i),\ \forall i \in [m].$$

Here, $\boldsymbol{\xi}^i \in \mathbb{R}^{n_\xi}$ represents the parameters of training instance $i$. The neural network $\boldsymbol{\psi}_\Theta(\boldsymbol{\xi}^i)$, parameterized by weights $\Theta$, predicts a solution $\hat{\mathbf{x}}^i = (\hat{\mathbf{x}}^i_r, \hat{\mathbf{x}}^i_z)$ for the mixed-integer decision variables, where $\hat{\mathbf{x}}^i_r \in \mathbb{R}^{n_r}$ denotes the continuous part and $\hat{\mathbf{x}}^i_z \in \mathbb{Z}^{n_z}$ denotes the integer part. The objective is to learn the weights $\Theta$ that minimize an empirical approximation to the expected objective function while satisfying the inequality constraints $\mathbf{g}(\hat{\mathbf{x}}^i, \boldsymbol{\xi}^i) \leq 0$, where $\mathbf{g}(\cdot)$ is a vector-valued function. We assume that the objective and constraint functions are differentiable.

**Loss function.** Our approach is self-supervised because the loss calculation does not rely on labeled data, which is particularly advantageous given the inherent difficulty of computing optimal or feasible solutions to MINLPs. The average value of the objective function $f(\cdot)$ serves as a natural loss function. However, solely minimizing the objective is insufficient when solutions violate the constraints. Therefore, similarly to Donti et al. [18], we incorporate penalty terms to account for constraint violations. This results in a soft-constrained empirical risk minimization loss, given as:

$$\mathcal{L}(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\Big[f(\hat{\mathbf{x}}^i, \boldsymbol{\xi}^i) + \lambda \cdot \|\mathbf{g}(\hat{\mathbf{x}}^i, \boldsymbol{\xi}^i)_+\|_1\Big], \tag{2}$$

where $\hat{\mathbf{x}}^i = \boldsymbol{\psi}_\Theta(\boldsymbol{\xi}^i)$ are the neural network outputs, $\|\,(\cdot)_+\,\|_1$ penalizes only positive constraint violations (implemented via a `ReLU` function), and $\lambda > 0$ is a penalty hyperparameter that balances the trade-off between minimizing the objective function and satisfying the constraints.

**Differentiating through discrete operations.** Since MINLP involves integer decision variables, neural networks are required to produce discrete outputs. However, discrete operations like rounding are inherently non-differentiable, which obstructs the use of standard gradient-based training. To overcome this challenge, we employ the *Straight-Through Estimator* (STE) [66], a widely used technique for enabling backpropagation through discrete mappings. 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. Specifically, for our floor function that rounds the number down and indicator function that decides rounding directions, STE uses the gradient of the identity function during backpropagation.

# 4 Methodology

In this section, we introduce our novel L2O methodology for solving parametric MINLP problems. As illustrated in Figure 1, the approach consists of two core components: (i) integer correction layers and (ii) an integer feasibility projection. For clarity, we omit the parametric index $i$ throughout.

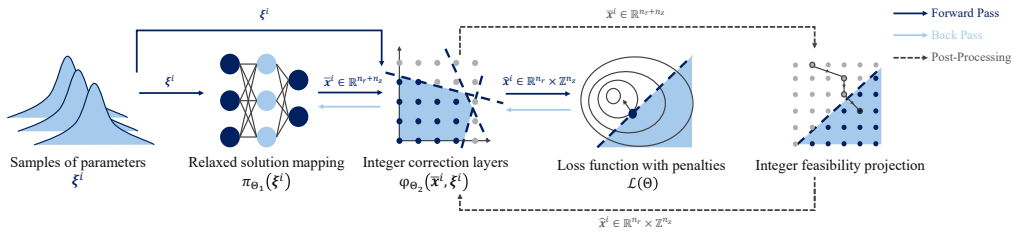


Figure 1: Conceptual diagram for our self-supervised learning-to-optimize approach for parametric MINLP: Given a parameter $\boldsymbol{\xi}$, the model predicts a relaxed solution $\bar{\mathbf{x}}$, which is then processed through integer correction to enforce discrete outputs $\hat{\mathbf{x}}$. The model is trained in a self-supervised manner using a loss that combines objective value and constraint violations. At inference time, an iterative feasibility projection is applied to improve constraint satisfaction.

## 4.1 Integer Correction Layers

To handle the integer decision variables in MINLPs, we propose two learnable correction layers: *Rounding Classification* (RC) and *Learnable Threshold* (LT). The mapping $\psi_\Theta : \mathbb{R}^{n_\xi} \mapsto \mathbb{R}^{n_r} \times \mathbb{Z}^{n_z}$ from an instance parameter vector $\boldsymbol{\xi}$ to a mixed-integer solution $\hat{\mathbf{x}}$ is performed in two stages:

1. **Relaxed Solution Mapping:** The first step consists in applying a learnable mapping $\pi_{\Theta_1} : \mathbb{R}^{n_\xi} \mapsto \mathbb{R}^{n_r+n_z}$, parameterized by weights $\Theta_1$. It outputs a relaxed solution $\bar{\mathbf{x}} \in \mathbb{R}^{n_r+n_z}$ without enforcing integrality, i.e., $\bar{\mathbf{x}} = \pi_{\Theta_1}(\boldsymbol{\xi})$.
2. **Integer Correction:** The second step is a correction module $\varphi_{\Theta_2} : \mathbb{R}^{n_r+n_z} \times \mathbb{R}^{n_\xi} \mapsto \mathbb{R}^{n_r} \times \mathbb{Z}^{n_z}$ that refines the relaxed solution $\bar{\mathbf{x}}$ into a mixed-integer solution $\hat{\mathbf{x}}$. It includes a trainable layer $\delta_{\Theta_2}$ with learnable weights $\Theta_2$, which adaptively learns rounding decisions conditioned on both the parameter $\boldsymbol{\xi}$ and the relaxed solution $\bar{\mathbf{x}}$, i.e., $\hat{\mathbf{x}} = \varphi_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$.

Algorithm 1 summarizes the integer correction layer. Our implementation leverages the STE for differentiability through discrete operations. While STE has been widely used in quantized neural networks, this is, to our knowledge, its first application in learning-to-optimize. Instead of fixed rounding, our correction layers learn adaptive rounding strategies conditioned on each parametric instance $\boldsymbol{\xi}$. During training, the loss function in Equation (2) is used to jointly optimize the neural network weights $\Theta = \Theta_1 \cup \Theta_2$, accounting for both the objective function value and constraint violations of the predicted mixed-integer solution $\hat{\mathbf{x}}$. Implementation details are provided in Appendix A, while visualizations of solution evolution during training are in Appendix B.

Conceptually, our integer correction layer is a learnable, end-to-end extension of the Relaxation Enforced Neighborhood Search (RENS) [67], implicitly exploring the neighborhood of relaxed solutions. Its simplicity enables fast training and inference, even on large-scale MINLPs.

**Algorithm 1** Integer Correction $\varphi_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$

**Input:** initial relaxed solution $\bar{\mathbf{x}}$, parameters $\boldsymbol{\xi}$, and neural network $\delta_{\Theta_2}(\cdot)$
Obtain hidden states $\mathbf{h} \leftarrow \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$
Update continuous variables $\hat{\mathbf{x}}_r \leftarrow \bar{\mathbf{x}}_r + \mathbf{h}_r$
Round integer variables down $\hat{\mathbf{x}}_z \leftarrow \lfloor \bar{\mathbf{x}}_z \rfloor$
**if** using *Rounding Classification* (RC) **then**
    Obtain values $\mathbf{v} \leftarrow \text{Gumbel-Sigmoid}(\mathbf{h}_z)$
**else if** using *Learnable Threshold* (LT) **then**
    Obtain logits $\mathbf{r} \leftarrow (\bar{\mathbf{x}}_z - \hat{\mathbf{x}}_z) - \mathbf{h}$
    Obtain values $\mathbf{v} \leftarrow \text{Sigmoid}(10 \cdot \mathbf{r})$
**end if**
Obtain rounding directions $\mathbf{b} \leftarrow \mathbb{I}\big(\mathbf{v} > 0.5\big)$
Update integer variables $\hat{\mathbf{x}}_z \leftarrow \hat{\mathbf{x}}_z + \mathbf{b}$
**Output:** a mixed-integer solution $\hat{\mathbf{x}}$

**Algorithm 2** Integer Feasibility Projection $\phi(\bar{\mathbf{x}}, \boldsymbol{\xi})$

1: **Input:** initial relaxed solution $\bar{\mathbf{x}}$, parameters $\boldsymbol{\xi}$, integer correction layer $\varphi_{\Theta_2}(\cdot)$, and step size $\eta$
2: **while** True **do**
3:  Obtain updated integers $\hat{\mathbf{x}} \leftarrow \varphi_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$
4:  Compute violations $\mathcal{V}(\hat{\mathbf{x}}, \boldsymbol{\xi}) \leftarrow \|\mathbf{g}(\hat{\mathbf{x}}, \boldsymbol{\xi})_+\|_1$
5:  **if** $\mathcal{V}(\hat{\mathbf{x}}, \boldsymbol{\xi}) = 0$ **then**
6:  Break
7:  **else**
8:  Compute gradients $\mathbf{d} \leftarrow \nabla_{\bar{\mathbf{x}}} \mathcal{V}(\hat{\mathbf{x}}, \boldsymbol{\xi})$
9:  Update relaxed solution $\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} - \eta \mathbf{d}$
10:  **end if**
11: **end while**
12: **Output:** a mixed-integer solution $\hat{\mathbf{x}}$

### 4.2 Integer Feasibility Projection

Despite the effectiveness of the proposed integer correction layers in ensuring integrality, penalty-based methods cannot fully guarantee constraint satisfaction. Therefore, we incorporate a gradient-based projection method as a post-processing step. This procedure iteratively updates a relaxed solution $\bar{\mathbf{x}}$ to reduce constraint violations, while preserving integrality through repeated application of the correction layer $\varphi_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$.

Our approach efficiently alternates two lightweight steps: integer correction and gradient-based updates. In Algorithm 2, line 3 applies the correction layer to enforce integrality, while line 9—following Donti et al. [18]—performs gradient descent to reduce constraint violations $\mathcal{V}(\hat{\mathbf{x}})$. But unlike Donti et al. [18], who integrate feasibility projection during training, we apply it only at inference. Doing so avoids the need to retain deep computation graphs or compute second-order derivatives through repeated projections, preserving training efficiency and stability. This iterative structure resembles the classical "Feasibility Pump" [68], which also alternates between rounding and projection to find feasible solutions. However, unlike the original method that requires solving constrained subproblems, our correction step is learned from data through a trainable network.

### 4.3 Convergence Guarantees for Integer Feasibility Projection

For feasibility guarantees, we analyze the convergence of our integer feasibility projection $\phi(\mathbf{x}, \boldsymbol{\xi})$ for a single parametric instance $\boldsymbol{\xi}$ which remains fixed during the inference. Hence, for simplicity of exposition, we omit the parameters $\boldsymbol{\xi}$ and denote the integer correction layer by $\varphi(\mathbf{x})$, and constraint functions by $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_{n_c}(\mathbf{x})]$, and `ReLU` function by $(\cdot)_+$. Theorem 1 provides conditions for asymptotic convergence, while Theorem 2 provides non-asymptotic convergence to the approximate feasible set.

**Theorem 1** (Asymptotic Convergence of Integer Feasibility Projection)**.** *Let* $\mathcal{V}(\mathbf{x}) := \|\mathbf{g}(\varphi(\mathbf{x}))_+\|_1 = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$, *where* $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n_c}$ *and* $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ *are continuously differentiable with gradients* $\nabla \mathbf{g}$, $\nabla \varphi$ *with Lipschitz constants* $L_{\mathbf{g}}$, $L_\varphi$ *and bounded Jacobians* $\|\nabla \mathbf{g}\| \le G_{\mathbf{g}}$, $\|\nabla \varphi\| \le G_\varphi$. *Assume all gradient descent iterates* $\mathbf{x}^{(k)} \in \mathcal{D} := \{\mathbf{x} \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) > 0\}$, *and the number of active constraints is uniformly bounded* $|I_{\mathbf{x}}| := |\{j : g_j(\varphi(\mathbf{x})) > 0\}| \le \bar{n}_c$. *Then for step size* $\eta \in \big(0, \frac{1}{L}\big]$, *where* $L := \bar{n}_c(G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}})$, *the following hold:*

(i) ***L-smoothness:*** $\mathcal{V} \in C^1(\mathcal{D})$, *with* $\nabla \mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} \nabla \varphi(\mathbf{x})^\top \nabla g_j(\varphi(\mathbf{x}))$, *and* $\nabla \mathcal{V}$ *is Lipschitz continuous on compact subsets of* $\mathcal{D}$ *with Lipschitz constant at most* $L$.

(ii) ***Descent and vanishing gradient:*** *Gradient descent generates a non-increasing sequence* $\mathcal{V}(\mathbf{x}^{(k)}) \to \mathcal{V}^\star \ge 0$, *and* $\lim_{k \to \infty} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| = 0$.

(iii) ***Convergence to feasibility:*** *If every* $\mathbf{x}^* \in \mathcal{D}$ *with* $\mathcal{V}(\mathbf{x}^*) > 0$ *satisfies* $\exists j \in I_{\mathbf{x}^*}$ *such that* $\nabla g_j(\varphi(\mathbf{x}^*)) \neq 0$, *then* $\lim_{k \to \infty} \mathcal{V}(\mathbf{x}^{(k)}) = 0$.

**Theorem 2** (Non-Asymptotic Convergence of Integer Feasibility Projection)**.** *Under the assumptions of Theorem 1, suppose gradient descent is applied to the function* $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$*, with fixed step size* $\eta \in \left(0, \frac{1}{L}\right]$*, where* $L := \bar{n}_c(G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}})$ *is an upper bound on the Lipschitz constant of* $\nabla\mathcal{V}$ *over the region* $\mathcal{D} := \{x : \mathcal{V}(\mathbf{x}) > 0\}$*. Then for any number of iterations* $K \geq 1$*, the minimum gradient norm over the first* $K$ *iterates satisfies*

$$\min_{0 \leq k < K} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2 \leq \frac{2}{\eta K} \left[\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}^\star\right],$$

*where* $\mathcal{V}^\star := \inf_{x \in \mathcal{D}} \mathcal{V}(\mathbf{x}) \geq 0$*. In particular, to ensure* $\min_{0 \leq k < K} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| \leq \delta$*, it suffices to run* $K \geq \frac{2}{\eta \delta^2}(\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}^\star)$ *iterations with complexity* $K = \mathcal{O}\left(\frac{1}{\delta^2}\right)$*. Furthermore, if* $\mathcal{V}^\star = 0$*, then for any* $\epsilon > 0$ *this implies approximate feasibility* $\mathcal{V}(\mathbf{x}^{(k)}) < \epsilon \quad$ *for all* $k \geq K_\epsilon$ *for some* $K_\epsilon$*.*

The proofs of the above theorems, iteration complexity, an alternative asymptotic convergence via Łojasiewicz inequality, and other supplementary theoretical analysis can be found in Appendix C.

# 5 Experimental Results

## 5.1 Experimental Setup

**Methods.** Table 1 provides an overview of all the methods used in the following experiments. A uniform 1000-second time limit is imposed for all methods. The experiments evaluate learning-based methods, Rounding Classification (RC) and Learnable Threshold (LT), which are trained to predict integer solutions directly. we also evaluate their enhanced variants with integer feasibility projection, RC-P and LT-P to improve feasibility. We compare these methods against a broad set of baselines: Exact solvers (EX), implemented via Gurobi for convex problems and SCIP for nonconvex problems, search optimal solutions when tractable, but often incur high computational cost. Heuristic baselines include Rounding after Relaxation (RR), which directly rounds continuous solutions, and root node solution (N1), which extracts the first feasible solution from the solver. Note that baselines EX and N1 include a wide range of heuristics embedded in the solver and executed in conjunction with the tree search procedure; we are also implicitly comparing these heuristics, not just to the exact search. Altogether, these methods span a wide spectrum of strategies—from exact to learning-based and heuristic—enabling a comprehensive evaluation in terms of solution quality, feasibility, and runtime.

Table 1: Summary of Methods. Methods with "*" use a trained model.

| **Method** | **Abbr** | **Description** |
|---|---|---|
| Rounding Classification* | RC* | Learns to produce probabilities to classify rounding directions. |
| Learnable Threshold* | LT* | Learns to predict thresholds to guide integer variable rounding. |
| Exact Solver | EX | Solves problems exactly using Gurobi and SCIP + Ipopt. |
| Rounding after Relaxation | RR | Rounds solutions of the continuous relaxation to its nearest integers. |
| Root Node Solution | N1 | Finds the first feasible solution from the root node of the solver. |
| **Enhanced Methods:** RC-P* and LT-P* that extend RC and LT with integer feasibility projection. | | |

In addition, we conduct two ablation studies to evaluate the effectiveness of the correction layers $\varphi_{\Theta_2}$. The first baseline, Rounding after Learning (RL), removes the correction module entirely and applies naive rounding after training. The second, Rounding with STE (RS), uses a fixed rounding operator during training. These variants isolate the effect of removing learnable rounding behavior from our framework. Results in Appendix G.1 show that both RL and RS significantly underperform our full model in terms of feasibility and objective value, highlighting the importance of end-to-end, data-driven correction.

**Problem classes.** We tested the methods on a variety of optimization problems, including integer quadratic problems, integer non-convex problems, and mixed-integer Rosenbrock problems. They were selected to cover both convex and non-convex scenarios and evaluate the scalability of the

methods in higher-dimensional settings. Further details on the mathematical formulation and data generation process are provided in Appendix D. The problem classes are as follows:

- **Integer Quadratic Problems (IQPs).** Based on [18], modified to include integer variables and reformulated by removing equality constraints.
- **Integer Non-convex Problems (INPs).** Extend IQPs with trigonometric terms in the objective and parameterized constraint matrices.
- **Mixed-integer Rosenbrock Problems (MIRBs).** A new large-scale MINLP benchmark with nonlinear constraints and parametric variation in both objectives and constraints.

In addition, we evaluated our methods on integer linear programs (MILPs) using the dataset from the MIP Workshop 2023 Computational Competition [69]. These experiments primarily serve to demonstrate that our methods can also handle integer linear cases with details provided in Appendix G.4.

**Training configuration.** Each model is trained on 8,000 instances, validated on 1,000, and evaluated on a test set of 100 unseen samples. Architecture and training hyperparameters are detailed in Appendix E.2. Our code is available at https://anonymous.4open.science/r/L2O-MINLP.

All experiments are conducted on a workstation equipped with NVIDIA V100 GPUs. Exact solvers include Gurobi (for convex problems) and SCIP with Ipopt (for nonconvex problems); further setup details are in Appendix E.1. Notably, Gurobi and SCIP are widely recognized as state-of-the-art solvers for MINLP. As highlighted in the comprehensive benchmarking study by [70]: "It is clear, however, that the global solvers Antigone, BARON, Couenne, and SCIP are the most efficient at finding the correct primal solution when regarding the total time limit. [...] Gurobi also is very efficient when considering that it only supports a little over half of the total number of problems!"

Thanks to the self-supervised nature of our framework, which does not require labeled solutions, scaling up the training dataset incurs no additional labeling cost. To examine this benefit, we analyze the effect of training set size in Appendix G.3.

**Overall results.** As illustrated in Figure 2, exact solvers such as Gurobi find better solutions over time but can be slow. For more complex problem instances, these solvers may fail to find feasible solutions within strict time limits. In contrast, our proposed methods consistently achieve high-quality feasible solutions within milliseconds. To the best of our knowledge, this is the first general approach for efficiently solving parametric MINLPs with up to tens of thousands of variables.

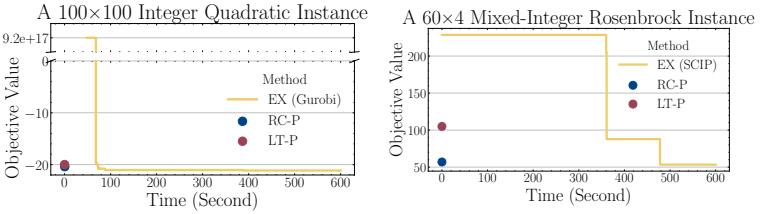


Figure 2: Illustration of objective value evolution for a $100 \times 100$ Integer Quadratic and $60 \times 4$ Mixed-Integer Rosenbrock over 600 seconds. The RC-P and LT-P methods achieve subsecond solutions comparable to those found by exact solvers in hundreds of seconds.

Even when accounting for training time (100 seconds), the overall efficiency of RC and LT remains substantially better. Importantly, once trained, the models effectively generalize to unseen problem instances, making them ideal for repeated problem-solving scenarios where the training cost is amortized [71]. Furthermore, RC and LT can generate high-quality initial solutions for exact solvers, reducing the search space and accelerating the convergence of traditional methods.

### 5.2 Empirical Evaluation

We evaluate our methods on a range of problem instances. For IQPs and INPs, we tested problem sizes from $20 \times 20$ (20 decision variables and 20 constraints) to $1000 \times 1000$, while for MIRBs, we experimented with instances ranging from 2 to 20,000 decision variables with the number of

constraints fixed at 4. The results are summarized in Table 3, Table 4, and Table 5, corresponding to IQPs, INPs, and MIRBs, respectively. For methods that rely on exact solvers (EX, N1, and RR), the solver may fail to find any solution within the time limit. To account for this, we report "%Solved", which indicates the proportion of instances obtained within the given computational budget.

Result for IQPs, INPs and MIRBs. Each problem size is evaluated on a test set of 100 instances. "Obj Mean" and "Obj Med" represent the mean and median objective values for this minimization problem, with smaller values being better. "Feasible" denotes the fraction of feasible solutions, "Solved" denotes the percentage of instances where a solution, whether feasible or infeasible, was found within the time limit, and "Time " is the average solving/inference time per instance. The "—" symbol indicates that no solution is found for any instance within 1000 seconds. For methods achieving 100% feasibility, we highlight in bold the best-performing metrics across problem sizes.

Table 3: Results for IQP

| | Metric | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|---|
| RC | Obj Mean | −4.237 | −12.20 | −13.54 | −31.62 | −73.31 | −142.7 |
| | Obj Med | −4.307 | −12.20 | −13.60 | −31.71 | −73.38 | −142.7 |
| | Feasible | 99% | 99% | 96% | 97% | 86% | 82% |
| | Time | 0.0019 | 0.0019 | 0.0022 | 0.0021 | 0.0025 | 0.0042 |
| RC-P | Obj Mean | −4.238 | −12.20 | −13.54 | **−31.62** | **−73.31** | **−142.7** |
| | Obj Med | −4.307 | −12.20 | −13.57 | **−31.71** | **−73.38** | **−142.7** |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time | **0.0045** | 0.0055 | **0.0050** | **0.0050** | 0.0065 | 0.0090 |
| LT | Obj Mean | −4.302 | −12.98 | −13.65 | −31.34 | −72.36 | −142.6 |
| | Obj Med | −4.319 | −13.03 | −13.77 | −31.61 | −72.48 | −142.6 |
| | Feasible | 98% | 98% | 93% | 95% | 94% | 100% |
| | Time | 0.0020 | 0.0020 | 0.0023 | 0.0022 | 0.0026 | 0.0047 |
| LT-P | Obj Mean | −4.301 | −12.98 | −13.65 | −31.34 | −72.36 | −142.6 |
| | Obj Med | −4.316 | −13.03 | −13.77 | −31.61 | −72.48 | −142.6 |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time | 0.0056 | **0.0055** | 0.0100 | 0.0064 | **0.0063** | **0.0086** |
| EX | Obj Mean | **−5.120** | **−15.93** | **−20.79** | — | — | — |
| | Obj Med | **−5.130** | **−15.96** | **−20.78** | — | — | — |
| | Feasible | 100% | 100% | 100% | — | — | — |
| | Solved | 100% | 100% | 100% | 0% | 0% | 0% |
| | Time | 8.728 | 1520 | 1237 | — | — | — |
| RR | Obj Mean | −5.179 | −16.17 | −21.92 | −46.73 | −106.5 | −213.3 |
| | Obj Median | −5.217 | −16.21 | −21.89 | −46.76 | −106.5 | −213.3 |
| | Feasible | 0% | 0% | 0% | 0% | 0% | 0% |
| | Solved | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time | 0.417 | 0.440 | 0.583 | 0.846 | 2.639 | 8.874 |
| N1 | Obj Mean | 9.8e7 | 1.7e17 | 1.5e18 | — | — | — |
| | Obj Med | 9.600 | 2.4e17 | 1.4e18 | — | — | — |
| | Feasible | 100% | 100% | 100% | — | — | — |
| | Solved | 100% | 100% | 100% | 0% | 0% | 0% |
| | Time | 0.415 | 0.498 | 104.2 | — | — | — |

Table 4: Results for INP

| | Metric | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|---|
| RC | Obj Mean | 0.228 | 0.771 | 1.664 | 1.472 | 0.526 | 1.422 |
| | Obj Med | 0.217 | 0.752 | 1.594 | 1.436 | 0.526 | 0.809 |
| | Feasible | 100% | 98% | 100% | 99% | 96% | 97% |
| | Time | **0.0019** | 0.0020 | 0.0022 | 0.0022 | 0.0029 | 0.0040 |
| RC-P | Obj Mean | 0.228 | 0.772 | 1.664 | 1.471 | 0.524 | 1.423 |
| | Obj Median | 0.217 | 0.752 | 1.594 | 1.436 | 0.526 | 0.809 |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time | 0.0045 | 0.0058 | 0.0060 | 0.0054 | **0.0061** | **0.0115** |
| LT | Obj Mean | 0.195 | 0.580 | 0.669 | **−0.356** | −1.374 | −3.744 |
| | Obj Med | 0.175 | 0.566 | 0.649 | **−0.373** | −1.594 | −3.716 |
| | Feasible | 99% | 98% | 96% | 100% | 98% | 99% |
| | Time | 0.0019 | 0.0020 | 0.0021 | **0.0023** | 0.0029 | 0.0050 |
| LT-P | Obj Mean | 0.195 | **0.580** | **0.669** | **−0.356** | **−1.374** | **−3.744** |
| | Obj Median | 0.175 | 0.566 | **0.649** | **−0.373** | **−1.594** | **−3.716** |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time | 0.0048 | **0.0050** | **0.0058** | 0.0056 | 0.0072 | 0.0117 |
| EX | Obj Mean | **−0.453** | 1.649 | 256.93 | — | — | — |
| | Obj Med | **−0.463** | **−0.052** | 134.62 | — | — | — |
| | Feasible | 100% | 100% | 14% | — | — | — |
| | Solved | 100% | 100% | 14% | 0% | 0% | 0% |
| | Time | 0.9949 | 1001 | 1001 | — | — | — |
| RR | Obj Mean | −0.464 | −1.039 | −2.068 | −3.990 | −9.391 | — |
| | Obj Med | −0.476 | −1.215 | −2.307 | −4.327 | −9.221 | — |
| | Feasible | 3% | 0% | 0% | 0% | 0% | — |
| | Solved | 100% | 100% | 100% | 100% | 100% | 0% |
| | Time | 0.996 | 1.189 | 4.600 | 54.01 | 449.0 | — |
| N1 | Obj Mean | 2.1e4 | 3.7e6 | 4411 | — | — | — |
| | Obj Med | 2.222 | 45.85 | 155.2 | — | — | — |
| | Feasible | 100% | 100% | 14% | — | — | — |
| | Solved | 100% | 100% | 14% | 0% | 0% | 0% |
| | Time | 0.144 | 8.968 | 940.4 | — | — | — |

Table 5: Results for MIRB

| | Metric | 2×4 | 20×4 | 200×4 | 2000×4 | 20000×4 |
|---|---|---|---|---|---|---|
| RC | Obj Mean | 23.27 | **59.39** | 503.5 | 5938 | 6.7e4 |
| | Obj Med | 21.48 | **48.86** | 461.7 | 5792 | 6.7e4 |
| | Feasible | 97% | 100% | 99% | 99% | 76% |
| | Time | 0.0019 | **0.0019** | 0.0021 | 0.0033 | 0.0121 |
| LT | Obj Mean | 23.18 | 62.51 | 622.8 | 5612 | 4.8e4 |
| | Obj Med | 20.80 | 63.40 | 626.0 | 5558 | 3.5e4 |
| | Feasible | 98% | 100% | 100% | 97% | 66% |
| | Time | 0.0019 | 0.0020 | 0.0026 | 0.0030 | 0.0127 |
| EX | Obj Mean | **19.62** | 64.67 | 8.4e5 | 4.7e10 | 1.1e15 |
| | Obj Med | **18.20** | 59.16 | 908.8 | 9262 | 1.0e5 |
| | Feasible | 100% | 100% | 100% | 96% | 78% |
| | Solved | 100% | 100% | 100% | 96% | 78% |
| | Time | 3.5090 | 1005 | 1002 | 1002 | 1040 |
| N1 | Obj Mean | 40.37 | 87.83 | 3.7e8 | 8.3e12 | 1.2e15 |
| | Obj Med | 27.93 | 77.34 | 957.4 | 9379 | 1.0e5 |
| | Feasible | 100% | 100% | 100% | 95% | 78% |
| | Solved | 100% | 100% | 100% | 95% | 78% |
| | Time | 0.0323 | 0.0813 | 0.2608 | 71.91 | 782.1 |

| | Metric | 2×4 | 20×4 | 200×4 | 2000×4 | 20000×4 |
|---|---|---|---|---|---|---|
| RC-P | Obj Mean | 23.50 | **59.39** | **504.2** | 5942 | 9.8e4 |
| | Obj Med | 21.48 | **48.86** | **461.7** | 5792 | 7.3e4 |
| | Feasible | 100% | 100% | 100% | 100% | 100% |
| | Time | **0.0062** | 0.0048 | **0.0052** | **0.0070** | 0.0824 |
| LT-P | Obj Mean | 23.33 | 62.51 | 622.8 | **5615** | **8.0e4** |
| | Obj Med | 20.80 | 63.40 | 626.0 | **5558** | **4.5e4** |
| | Feasible | 100% | 100% | 100% | 100% | 100% |
| | Time | **0.0062** | 0.0055 | 0.0062 | 0.0071 | **0.0639** |
| RR | Obj Mean | 22.24 | 1.2e4 | 1.4e4 | 2.1e6 | 1.7e8 |
| | Obj Med | 22.19 | 51.17 | 501.9 | 5437 | 7.0e6 |
| | Feasible | 55% | 59% | 40% | 6% | 18% |
| | Solved | 100% | 100% | 58% | 7% | 22% |
| | Time | 0.1805 | 0.5570 | 1.2396 | 9.2334 | 1064 |

It is worth noting that some of the objective values are extremely large. This occurs when the baseline methods, such as EX and N1, generate poor-quality feasible solutions, particularly for larger problem instances. Due to the absence of explicit bounds on the decision variables, the baselines occasionally produce trivial yet suboptimal solutions, leading to inflated objective values. This issue is not confined to this particular case but also appears in other problem instances, further underscoring the limitations of the baseline methods in handling larger-scale optimization tasks effectively.

**Q1. How do learning-based methods compare to traditional solvers and heuristics?** Traditional methods (EX, RR, and N1) struggle on larger instances, often failing to return solutions within the 1000-second limit, whereas RC and LT remain effective. While N1 can find feasible solutions quickly for small instances, it suffers from numerical instability and breaks down on larger problems. Similarly, RR, which relies on relaxation rounding, struggles with feasibility across all problem scales. In contrast, RC and LT achieve objective values close to the exact solver (EX) while maintaining lower infeasibility rates and achieving several orders of magnitude speed-ups. For IQPs and INPs, RC and LT outperform heuristic baselines such as N1 and RR, especially as problem sizes increase. In MIRBs, they even surpass EX in most cases. Overall, our learning-based approaches provide substantial advantages in scalability, speed, and solution quality over other methods.

**Q2. How effective is the integer feasibility projection?** RC-P and LT-P successfully find feasible solutions for all test instances. As shown in Appendix F.1, constraint violations in RC and LT in IQPs and INPs are sparse and minor, which allows the projection step to correct infeasibilities with negligible impact on the objective value. For MIRBs, feasibility projection plays an even more crucial role, while feasibility rates decline significantly for RC and LT as the problem size grows to 20,000 variables, applying feasibility projection allows RC-P and LT-P to satisfy all constraints across all instances. Even though feasibility projection introduces additional computational overhead, inference remains highly efficient. Even with projection, total inference time remains below one second, preserving a substantial speed advantage over other methods. Thus, these results demonstrate both the effectiveness and computational efficiency of our gradient-based projection step.

**Q3. How does the choice of penalty weight affect performance?** The penalty weight ($\lambda$ in Equation (2)) balances objective minimization and constraint satisfaction. We assess its impact by varying $\lambda$ from 0.1 to 1000 on $1000 \times 1000$ INPs using RC, LT, and their projection-enhanced variants RC-P and LT-P. Additional results for IQPs, MIRBs, and smaller-scale instances are reported in Appendix G.2. As shown in Figure 3, smaller penalty values typically yield better objective values but lead to a higher proportion of infeasible solutions in RC and LT, while larger values improve feasibility at the expense of suboptimality. Remarkably, applying our projection step—capped at a maximum of 1000 iterations—restores feasibility even under severe violations from small penalty weights, while largely preserving the low objective values. This trend holds consistently across benchmarks, suggesting that RC-P and LT-P can benefit from lower penalty weights than those used in the main experiments.

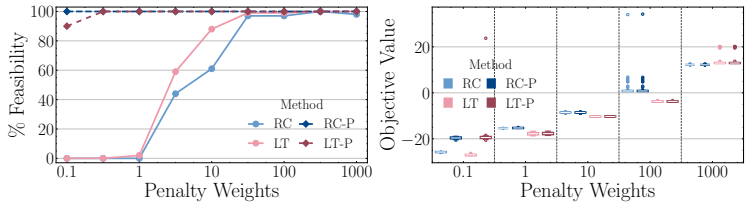


Figure 3: Illustration of the proportion of feasible solutions (Top) and objective value (Bottom) for $1000 \times 1000$ INC on the test set. As the penalty weight increases, the fraction of feasible solutions increases while the objective value generally deteriorates.

**Q4. How long is the training time?** In addition to evaluating solution quality, feasibility, and solving/inference times, we also measured the offline training times on different problem sizes. These results are presented in Appendix F.2, where it is evident that the training times for the learning-based methods scale well with problem size. Compared to the runtime of exact solvers, the training process is relatively short, often requiring only a few minutes for smaller instances and less than 30 minutes for the largest problems. For larger instances, training even takes less time than finding the first feasible solution for a single instance with the exact solver. Moreover, when repeatedly solving multiple instances, the training cost can be amortized, making our approach particularly advantageous in real-world scenarios where rapid deployment and large-scale optimization are required.

## 6 Conclusion

We introduced the first general learning-to-optimize (L2O) method for parametric MINLPs, featuring integer correction layers that enable neural networks to generate feasible, high-quality solutions. Our self-supervised approach does not require collecting optimal solutions as labels, substantially reducing data preparation efforts. Furthermore, we proposed a feasibility projection as post-processing with rigorous constraint satisfaction guarantees and negligible computational overhead.

Our learning-based methods outperform traditional solvers and heuristics across diverse problem types, maintaining strong feasibility and solution quality even in high-dimensional settings where traditional approaches often fail. To our knowledge, this is the first work to address L2O for parametric MINLPs and successfully solve one of the largest MINLPs reported to date.

Although our methods demonstrate strong performance, certain limitations remain. First, our feasibility guarantees rely on specific albeit generic assumptions. Extending these guarantees to broader classes of MINLPs is an important avenue for future work. Additionally, future work could explore alternative strategies to enhance feasibility. For example, in certain problem classes, a subset of constraints could be directly handled using differentiable optimization layers [54] while others are incorporated into the loss function. Additionally, specialized neural network architectures, such as those proposed by Pan et al. [22] and Tordesillas et al. [72], could be designed to satisfy certain types of constraints inherently.

## References


[1] Thomas Kleinert, Martine Labbé, Ivana Ljubić, and Martin Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization*, 9:100007, 2021.

[2] Nawaf Nazir and Mads Almassalkhi. Guaranteeing a physically realizable battery dispatch without charge-discharge complementarity constraints. *IEEE Transactions on Smart Grid*, 14 (3):2473–2476, 2021.

[3] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *2001 European control conference (ECC)*, pages 2603–2608. IEEE, 2001.

[4] Tobia Marcucci and Russ Tedrake. Warm start of mixed-integer programs for model predictive control of hybrid systems. *IEEE Transactions on Automatic Control*, 66(6):2433–2448, 2020.

[5] Ailsa H Land and Alison G Doig. *An automatic method for solving discrete programming problems*. Springer, 2010.

[6] Yves Crama, Antoon WJ Kolen, and EJ Pesch. Local search in combinatorial optimization. *Artificial Neural Networks: An Introduction to ANN Theory and Practice*, pages 157–174, 2005.

[7] David S Johnson and Lyle A McGeoch. The traveling salesman problem: a case study. *Local search in combinatorial optimization*, pages 215–310, 1997.

[8] Roger Fletcher and Sven Leyffer. Solving mixed integer nonlinear programs by outer approximation. *Mathematical programming*, 66:327–349, 1994.

[9] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods & Software*, 24 (4-5):597–634, 2009.

[10] Ivo Nowak. *Relaxation and decomposition methods for mixed integer nonlinear programming*, volume 152. Springer Science & Business Media, 2005.

[11] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. End-to-end constrained optimization learning: A survey. *arXiv preprint arXiv:2103.16378*, 2021.

[12] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[13] Pascal Van Hentenryck. Optimization learning, 2025. URL `https://arxiv.org/abs/2501.03443`.

[14] Johannes Hendriks, Carl Jidling, Adrian Wills, and Thomas Schön. Linearly constrained neural networks. *arXiv preprint arXiv:2002.01600*, 2020.

[15] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1796–1804, 2015.

[16] Zhipeng Jia, Xingyi Huang, I Eric, Chao Chang, and Yan Xu. Constrained deep weak supervision for histopathology image segmentation. *IEEE transactions on medical imaging*, 36(11):2376–2388, 2017.

[17] Meiyi Li, Soheil Kolouri, and Javad Mohammadi. Learning to solve optimization problems with hard linear constraints. *IEEE Access*, 11:59995–60004, 2022. URL `https://api.semanticscholar.org/CorpusID:251741205`.

[18] Priya Donti, David Rolnick, and J Zico Kolter. DC3: A learning method for optimization with hard constraints. In *International Conference on Learning Representations*, 2021.

[19] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. End-to-end feasible optimization proxies for large-scale economic dispatch. *IEEE Transactions on Power Systems*, 39(2):4723–4734, 2024. doi: 10.1109/TPWRS.2023.3317352.

[20] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[21] Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods. In *Proceedings of the AAAI conference on artificial intelligence*, 2020.

[22] Xiang Pan, Tianyu Zhao, Minghua Chen, and Shengyu Zhang. Deepopf: A deep neural network approach for security-constrained dc optimal power flow. *IEEE Transactions on Power Systems*, 36(3):1725–1735, 2020.

[23] Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, et al. Miplib 2017: data-driven compilation of the 6th mixed-integer programming library. *Mathematical Programming Computation*, 13(3):443–490, 2021.

[24] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Learning hard optimization problems: A data generation perspective. *Advances in Neural Information Processing Systems*, 34:24981–24992, 2021.

[25] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[26] Hanjun Dai, Elias Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in neural information processing systems*, 30, 2017.

[27] Hoel Kervadec, Jose Dolz, Jing Yuan, Christian Desrosiers, Eric Granger, and Ismail Ben Ayed. Constrained deep networks: Lagrangian optimization via log-barrier extensions. In *2022 30th European Signal Processing Conference (EUSIPCO)*, pages 962–966. IEEE, 2022.

[28] Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025*, 2017.

[29] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Hydra-mip: Automated algorithm configuration and selection for mixed integer programming. In *RCRA workshop on experimental evaluation of algorithms for solving problems with combinatorial explosion at the international joint conference on artificial intelligence (IJCAI)*, pages 16–30, 2011.

[30] Timo Berthold and Gregor Hendel. Learning to scale mixed-integer programs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[31] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[32] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. A machine learning-based approximation of strong branching. *INFORMS Journal on Computing*, 29(1):185–195, 2017.

[33] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. *Advances in neural information processing systems*, 32, 2019.

[34] Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. Parameterizing branch-and-bound search trees to learn branching policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[35] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

[36] Antonia Chmiela, Elias Khalil, Ambros Gleixner, Andrea Lodi, and Sebastian Pokutta. Learning to schedule heuristics in branch and bound. *Advances in Neural Information Processing Systems*, 34:24235–24246, 2021.

[37] Arnaud Deza and Elias B. Khalil. Machine learning for cutting planes in integer programming: A survey. In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence*, IJCAI-2023. International Joint Conferences on Artificial Intelligence Organization, August 2023. doi: 10.24963/ijcai.2023/739. URL `http://dx.doi.org/10.24963/IJCAI.2023/739`.

[38] Gabriele Dragotto, Stefan Clarke, Jaime Fernández Fisac, and Bartolomeo Stellato. Differentiable cutting-plane layers for mixed-integer linear optimization, 2023. URL `https://arxiv.org/abs/2311.03350`.

[39] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[40] Elias Khalil, Christopher Morris, and Andrea Lodi. MIP-GNN: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

[41] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020.

[42] Nicolas Sonnerat, Pengming Wang, Ira Ktena, Sergey Bartunov, and Vinod Nair. Learning a large neighborhood search algorithm for mixed integer programs. *arXiv preprint arXiv:2107.10201*, 2021.

[43] Jialin Song, Yisong Yue, Bistra Dilkina, et al. A general large neighborhood search framework for solving integer linear programs. *Advances in Neural Information Processing Systems*, 33: 20012–20023, 2020.

[44] Dimitris Bertsimas and Bartolomeo Stellato. Online mixed-integer optimization in milliseconds. *INFORMS Journal on Computing*, 34(4):2229–2248, 2022.

[45] Taoan Huang, Aaron M Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *International Conference on Machine Learning*, pages 13869–13890. PMLR, 2023.

[46] Huigen Ye, Hua Xu, and Hongyan Wang. Light-milpopt: Solving large-scale mixed integer linear programs with lightweight optimizer and small-scale training dataset. In *The Twelfth International Conference on Learning Representations*, 2024.

[47] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[48] Jiayi Zhang, Chang Liu, Xijun Li, Hui-Ling Zhen, Mingxuan Yuan, Yawen Li, and Junchi Yan. A survey for solving mixed integer programming via machine learning. *Neurocomputing*, 519: 205–217, 2023.

[49] Radu Baltean-Lugojan, Pierre Bonami, Ruth Misener, and Andrea Tramontani. Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks. `https://optimization-online.org/2018/11/6943/`, 2019.

[50] Alex Nowak, Soledad Villar, Afonso S Bandeira, and Joan Bruna. Revised note on learning quadratic assignment with graph neural networks. In *2018 IEEE Data Science Workshop (DSW)*, pages 1–5. IEEE, 2018.

[51] Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. A classifier to decide on the linearization of mixed-integer quadratic problems in cplex. *Operations research*, 70(6):3303–3320, 2022.

[52] Aaron M Ferber, Taoan Huang, Daochen Zha, Martin Schubert, Benoit Steiner, Bistra Dilkina, and Yuandong Tian. Surco: Learning linear surrogates for combinatorial nonlinear optimization problems. In *International Conference on Machine Learning*, pages 10034–10052. PMLR, 2023.

[53] Zhixiao Xiong, Fangyu Zong, Huigen Ye, and Hua Xu. Neuralqp: A general hypergraph-based optimization framework for large-scale qcqps. *arXiv preprint arXiv:2410.03720*, 2024.

[54] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and Zico Kolter. Differentiable convex optimization layers. *ArXiv*, abs/1910.12430, 2019.

[55] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *Advances in neural information processing systems*, 35:5230–5242, 2022.

[56] Adam N Elmachtoub and Paul Grigas. Smart "predict, then optimize". *Management Science*, 68(1):9–26, 2022.

[57] 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.

[58] Rajiv Sambharya, Georgina Hall, Brandon Amos, and Bartolomeo Stellato. End-to-end learning to warm-start for real-time quadratic optimization. In *Learning for Dynamics and Control Conference*, pages 220–234. PMLR, 2023.

[59] Priya Donti, Brandon Amos, and J Zico Kolter. Task-based end-to-end model learning in stochastic optimization. *Advances in neural information processing systems*, 30, 2017.

[60] Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. *Advances in Neural Information Processing Systems*, 30, 2017.

[61] Bryan Wilder, Bistra Dilkina, and Milind Tambe. Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[62] Quentin Berthet, Mathieu Blondel, Olivier Teboul, Marco Cuturi, Jean-Philippe Vert, and Francis Bach. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

[63] Marin Vlastelica Pogančić, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2020.

[64] Aaron Ferber, Bryan Wilder, Bistra Dilkina, and Milind Tambe. Mipaal: Mixed integer program as a layer. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 1504–1511, 2020.

[65] Bo Tang and Elias B Khalil. Cave: A cone-aligned approach for fast predict-then-optimize with binary linear programs. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 193–210. Springer, 2024.

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

[67] Timo Berthold. Rens: the optimal rounding. *Mathematical Programming Computation*, 6: 33–54, 2014.

[68] Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.

[69] Suresh Bolusani, Mathieu Besançon, Ambros Gleixner, Timo Berthold, Claudia D'Ambrosio, Gonzalo Muñoz, Joseph Paat, and Dimitri Thomopulos. The MIP Workshop 2023 computational competition on reoptimization, 2023. URL `http://arxiv.org/abs/2311.14834`.

[70] Andreas Lundell and Jan Kronqvist. Polyhedral approximation strategies for nonconvex mixed-integer nonlinear programming in shot. *Journal of Global Optimization*, 82(4):863–896, 2022.

[71] Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *CoRR*, abs/2202.00665, 2022. URL https://arxiv.org/abs/2202.00665.

[72] Jesus Tordesillas, Jonathan P How, and Marco Hutter. Rayen: Imposition of hard convex constraints on neural networks. *arXiv preprint arXiv:2307.08336*, 2023.

[73] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[74] Aladin Virmaux and Kevin Scaman. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/d54e99a6c03704e95e6965532dec148b-Paper.pdf.

[75] Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael J. Cree. Regularisation of neural networks by enforcing lipschitz continuity. *Mach. Learn.*, 110(2):393–416, February 2021. ISSN 0885-6125. doi: 10.1007/s10994-020-05929-w. URL https://doi.org/10.1007/s10994-020-05929-w.

[76] Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/95e1533eb1b20a97777749fb94fdb944-Paper.pdf.

[77] Ján Drgoňa, Aaron Tuor, Soumya Vasisht, and Draguna Vrabie. Dissipative deep neural dynamical systems. *IEEE Open Journal of Control Systems*, 1:100–112, 2022. doi: 10.1109/OJCSYS.2022.3186838.

[78] Jérôme Bolte, Aris Daniilidis, Adrian S Lewis, and Masahiro Shiota. Clarke subgradients of stratifiable functions. *SIAM Journal on Optimization*, 18(2):556–572, 2007.

[79] Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137:91–129, 2013.

[80] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pages 1724–1732. PMLR, 2017.

[81] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[82] Jan Drgona, Aaron Tuor, James Koch, Madelyn Shapiro, Bruno Jacob, and Draguna Vrabie. Neuromancer: Neural modules with adaptive nonlinear constraints and efficient regularizations, 2023. URL https://github.com/pnnl/neuromancer.

[83] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021. URL https://www.gurobi.com.

[84] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, et al. The scip optimization suite 8.0. *arXiv preprint arXiv:2112.08872*, 2021.

[85] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106:25–57, 2006.

# A Details of Correction Layers

The following subsections detail the two distinct approaches for designing the correction layer $\varphi_{\Theta_2}$ while sharing the same network $\pi_{\Theta_1}$ for generating relaxed solutions $\bar{\mathbf{x}}^i$ in both methods. RC and LT in Algorithm 1 differ in how they determine the rounding direction: RC adopts a probabilistic approach to decide the rounding direction for each integer variable, while LT yields a threshold vector to control the rounding process. Both methods are differentiable, easy to train using gradient descent, and computationally efficient during inference. The workflow for each approach is illustrated in Figure 4.

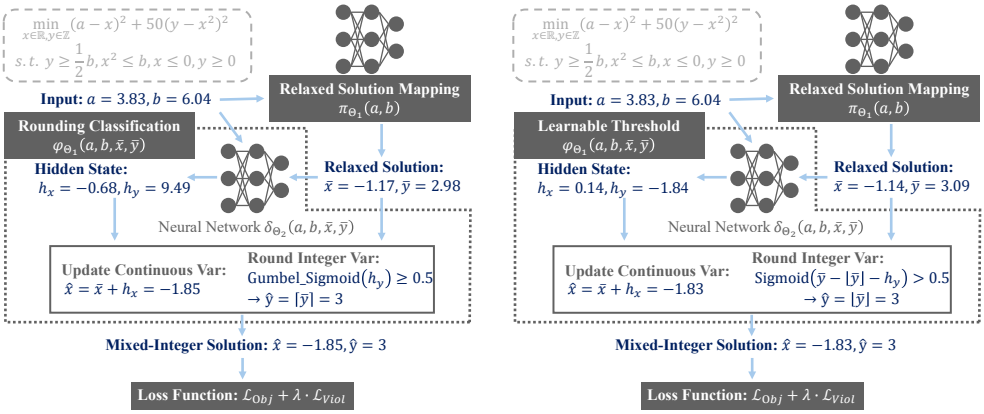


Figure 4: Examples of the two integer correction layers: Rounding Classification (left) uses a classification-based rounding strategy, while Learnable Threshold (right) yields a threshold to guide rounding decisions.

## A.1 Rounding Classification

**Forward Pass.** The key step of the *Rounding Classification* (RC) approach is performed in line 6 of Algorithm 1. For the integer variables, RC applies a stochastic soft-rounding mechanism to the neural network output $\mathbf{h}_z = \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})$, producing a binary vector $\mathbf{b} \in \{0,1\}^{n_z}$. Each entry of $\mathbf{b}$ determines whether the fractional part of the relaxed value $\bar{\mathbf{x}}_z$ is rounded down (0) or up (1).

To introduce stochasticity and enhance exploration during training, the Gumbel-noise method [73] is employed. Specifically, logits $\mathbf{h}_z$ are perturbed by noise sampled from the Gumbel distribution:

$$\epsilon = -\log(-\log(U)), \quad U \sim \text{Uniform}(0,1).$$

where $U$ is a random variable drawn from the uniform distribution over the interval $[0,1]$. The perturbed logits are then passed through the Sigmoid function to produce a soft decision (probability) $\mathbf{v}$ for rounding. Hence our binary Gumbel-Sigmoid($\mathbf{h}$) layer is defined as:

$$\mathbf{v} = \frac{1}{1+\exp\left(-\frac{\mathbf{h}+\boldsymbol{\epsilon}_1-\boldsymbol{\epsilon}_2}{\tau}\right)}$$

where $\boldsymbol{\epsilon}_1$ and $\boldsymbol{\epsilon}_2$ are vectors of independent Gumbel samples, and $\tau > 0$ is the scalar temperature parameter controlling the smoothness of the approximation. A smaller $\tau$ produces sharper transitions, approaching a hard step function, while a larger $\tau$ yields smoother probabilistic behavior. In our experiments, we set $\tau = 1$ for simplicity.

To obtain binary decisions, we apply the following indicator function in the forward pass:

$$\mathbf{b} = \mathbb{I}(\mathbf{v} > 0.5).$$

The correction layer $\varphi_{\Theta_2}$ for RC produces the final integer output as:

$$\hat{\mathbf{x}}_z = \lfloor \bar{\mathbf{x}}_z \rfloor + \mathbf{b}.$$

**Backward Pass.** Because the binarization operation is non-differentiable, the gradient of $\mathbf{b}$ w.r.t. the $\mathbf{v}$ is approximated with STE. Hence, the final gradient of $\mathbf{b}$ w.r.t. the input logit $\mathbf{h}$ is approximated by the gradient of the Gumbel-Sigmoid given as:

$$\frac{\partial \mathbf{b}}{\partial \mathbf{h}} := \frac{\partial \mathbf{v}}{\partial \mathbf{h}} = \frac{1}{\tau} \cdot \mathbf{v} \odot (1 - \mathbf{v})$$

where the gradient expression is computed elementwise, with $\odot$ denoting the Hadamard (elementwise) product. This formulation allows gradients to flow during backpropagation despite the non-differentiable binarization.

**Lipschitz Smoothness $L_\varphi$ of the Gradient.** Now applying the chain rule, the gradient of the binary decision vector $\mathbf{b}$ with respect to the inputs $\bar{\mathbf{x}}_z$ of the network $\delta_{\Theta_2}$ is given by:

$$\frac{\partial \mathbf{b}}{\partial \bar{\mathbf{x}}_z} = \frac{\partial \mathbf{b}}{\partial \mathbf{h}_z} \cdot \frac{\partial \mathbf{h}_z}{\partial \bar{\mathbf{x}}_z} \approx \frac{\partial \mathbf{v}}{\partial \mathbf{h}_z} \cdot \frac{\partial \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})}{\partial \bar{\mathbf{x}}_z} = \frac{1}{\tau} \cdot \mathbf{v} \odot (1 - \mathbf{v}) \cdot \frac{\partial \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})}{\partial \bar{\mathbf{x}}_z}.$$

Meanwhile, since the floor operation $\lfloor \bar{\mathbf{x}}_z \rfloor$ is non-differentiable, we again apply the STE by treating it as the identity function during backpropagation:

$$\frac{\partial \lfloor \bar{\mathbf{x}}_z \rfloor}{\partial \bar{\mathbf{x}}_z} := \mathbf{I},$$

hence, contributing a Lipschitz constant of 1 to $\bar{\mathbf{x}}_z$.

The Jacobian of the RC correction layer to the neural network input $\bar{\mathbf{x}}$ can thus be approximated as:

$$\nabla_{\bar{\mathbf{x}}} \varphi_{\Theta_2}(\bar{\mathbf{x}}) \approx \mathbf{I} + \frac{1}{\tau} \cdot \mathrm{diag}(\mathbf{v} \odot (1 - \mathbf{v})) \cdot \nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi}).$$

We now analyze the Lipschitz constant of this gradient map. Define the scalar function. Let us define the scalar function:

$$g(h) = \frac{1}{\tau} \cdot \sigma(z)\,(1 - \sigma(z))\,,$$

where $\sigma(\cdot)$is the sigmoid function, $z = \frac{h + \epsilon_1 - \epsilon_2}{\tau}$, and $\epsilon_1, \epsilon_2 \sim \text{Gumbel}(0, 1)$ are independent samples. This function corresponds to the elementwise gradient of the Gumbel-Sigmoid output with respect to its input logit $h$, under the STE approximation we use in the backward pass. It reflects the sensitivity of the soft relaxation $\mathbf{v}$ to changes in the perturbed logits $\mathbf{h}$. The shape and boundedness of $g(h)$ directly influence the stability and smoothness of our optimization process.

The maximum absolute value of this derivative over all $z \in \mathbb{R}$ determines the Lipschitz constant. The product $\sigma(z)(1-\sigma(z))(1-2\sigma(z))$ attains its maximum absolute value at $\sigma(z) = \frac{1}{2} \pm \frac{1}{2\sqrt{3}}$, yielding:

$$|g'(h)| \leq \frac{1}{6\sqrt{3}\tau^2} \approx \frac{0.0962}{\tau^2}$$

Hence, the Lipschitz constant of the STE-approximated gradient of the Gumbel-Sigmoid layer is bounded by:

$$L_{\text{Gumbel}} \leq \frac{0.0962}{\tau^2}$$

This implies that as the temperature $\tau$ decreases (to make the sampling sharper), the gradient becomes more sensitive to changes in $h$, which can affect training stability.

We now estimate the Lipschitz constant of the approximate Jacobian $\nabla_{\bar{\mathbf{x}}} \varphi_{\Theta_2}(\bar{\mathbf{x}})$, which is central to the convergence analysis of the integer feasibility projection (see Theorem 1). Since both the Gumbel modulation term and the neural network are Lipschitz continuous, the local Lipschitz constant is bounded by

$$L_\varphi \leq L_{\text{Gumbel}} \cdot \left\| \nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi}) \right\|,$$

where $\|\cdot\|$ denotes the spectral (operator) norm, i.e., the largest singular value of the Jacobian. For a global Lipschitz estimate, we have

$$L_\varphi^{\text{global}} \leq \frac{0.0962}{\tau^2} \cdot \sup_{\bar{\mathbf{x}}} \left\| \nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi}) \right\|.$$

This bound highlights how the temperature parameter $\tau$ and the smoothness of the logit network jointly affect the stability of the correction layer. In our setup with $\tau = 1$, this yields a concrete local bound $L_\varphi \leq 0.0962 \cdot \|\nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})\|$.

### A.2 Learnable Threshold

**Forward Pass.** The *Learnable Threshold* (LT) approach, detailed in of Algorithm 1, provides an alternative correction strategy. Instead of relying on probability as in RC, LT learns to predict a threshold vector $\mathbf{h}^i \in [0,1]^{n_z}$ by applying a Sigmoid activation, which guides rounding decisions for each integer variable. These thresholds $\mathbf{h}$ are then compared against the fractional part of the relaxed integer variables. Specifically, a variable is rounded up if its fractional part $\bar{\mathbf{x}}_z - \hat{\mathbf{x}}_z$ exceeds the threshold $\mathbf{h}$, and rounded down otherwise. Thus, the binary decision in the forward pass is computed as:

$$\mathbf{b} = \mathbb{I}(\bar{\mathbf{x}}_z - \hat{\mathbf{x}}_z - \mathbf{h} > 0).$$

**Backward Pass.** Although the forward pass applies a hard threshold, the backward pass approximates the gradient from the following smoothed Sigmoid surrogate:

$$\mathbf{v} = \frac{1}{1 + \exp\left(-\beta \cdot (\bar{\mathbf{x}}_z - \hat{\mathbf{x}}_z - \mathbf{h})\right)},$$

where $\beta > 0$ controls the steepness of the approximation. A higher $\beta$ yields sharper transitions. We use $\beta = 10$ in our experiments.

Thus, the approximated partial derivatives of $\mathbf{b}$ w.r.t. the threshold $\mathbf{h}$ are:

$$\frac{\partial \mathbf{b}}{\partial \mathbf{h}} := \frac{\partial \mathbf{v}}{\partial \mathbf{h}} = -\beta \cdot \mathbf{v} \odot (1 - \mathbf{v}).$$

**Lipschitz Smoothness $L_\varphi$ of the Gradient.** Since the gradient of the LT correction layers is approximated with a scaled sigmoid, let's analyze its maximum slope. The product $\sigma(z)(1-\sigma(z))$ is maximized at $z = 0$, where:

$$\sigma(0) = 0.5 \quad \Rightarrow \quad \sigma(0)(1-\sigma(0)) = 0.25$$

So the maximum value of the derivative of the scaled sigmoid function is:

$$\max_x \left| \frac{d}{dx}\sigma(\beta x) \right| = \beta \cdot \max_z \sigma(z)(1-\sigma(z)) = \beta \cdot 0.25 = \frac{\beta}{4}$$

Hence, similar to the RC method, the local Lipschitz constant of the approximate Jacobian $\nabla_{\bar{\mathbf{x}}}\varphi_{\Theta_2}$ for the LT correction layer is bounded by:

$$L_\varphi \leq \frac{\beta}{4} \cdot \left\| \nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi}) \right\|,$$

while the global Lipschitz constant is bounded by:

$$L_\varphi^{\text{global}} \leq \frac{\beta}{4} \cdot \sup_{\bar{\mathbf{x}}} \left\| \nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi}) \right\|.$$

In our setup with $\beta = 10$, this yields the concrete local bound $L_\varphi \leq 2.5 \cdot \|\nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})\|$.

*Remark* 1 (The role of Lipschitz constants). These concrete estimates for the Lipschitz constants of the approximate gradients in both integer correction layers enable the choice of appropriate step sizes $\eta \in (0, 1/L]$ during feasibility projection defined by Algorithm 2. For theoretical background, we refer readers to Appendix C. For practical methods to estimate or constrain the Lipschitz constants or operator norms of neural network gradients $\|\nabla_{\bar{\mathbf{x}}} \delta_{\Theta_2}(\bar{\mathbf{x}}, \boldsymbol{\xi})\|$, we refer to prior work such as [74–77].

## B Example Illustration

This section illustrates the process of our Integer Correction and Feasibility Projection components in producing solutions. We present a two-dimensional Mixed-Integer Rosenbrock Benchmark (MIRB) instance, formulated as follows:

$$\begin{aligned} \min_{x \in \mathbb{R}, y \in \mathbb{Z}} \quad & (a - x)^2 + 50(y - x^2)^2 \\ \text{subject to} \quad & y \geq b/2, \quad x^2 \leq b, \quad x \leq 0, \quad y \geq 0. \end{aligned}$$

In this formulation, $x$ is a continuous decision variable, while $y$ is an integer decision variable. Both variables are subject to linear constraints. The instance parameters $a$ and $b$ serve as input features to the neural network.

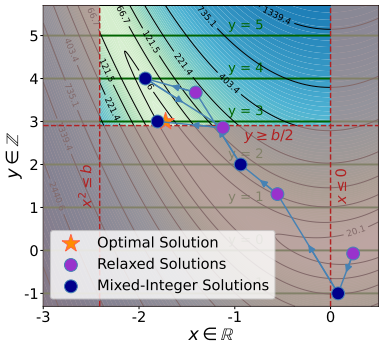


Figure 5: Example of the relaxed solutions $\bar{x}, \bar{y}$ and the mixed-integer solutions $\hat{x}, \hat{y}$ across different epochs of training for the same sample instance using RC.

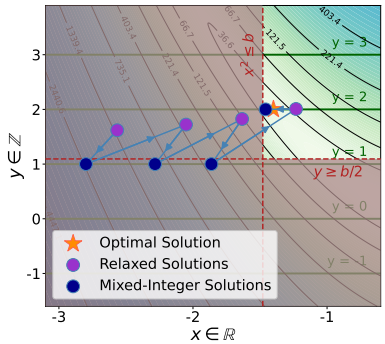


Figure 6: Example of the relaxed solutions $\bar{x}, \bar{y}$ and the mixed-integer solutions $\hat{x}, \hat{y}$ across different iterations of feasibility projection to refine an infeasible solution.

**Integer Correction Layer.** Figure 5 illustrates the progression of relaxed solutions $(\bar{x}, \bar{y})$ and their corresponding mixed-integer solutions $(\hat{x}, \hat{y})$ across different training epochs. In this example, the instance parameters are set to $a = 3.83$ and $b = 6.04$. Throughout training, the relaxed solutions $(\bar{x}, \bar{y})$ are iteratively adjusted by the neural network, and the integer correction layer transforms them into mixed-integer solutions. As training progresses, the model learns to generate solutions that are not only integer-feasible but also of high quality in terms of the objective function and constraint satisfaction.

**Integer Feasible Projection.** Figure 6 illustrates the iterative refinement process of the feasibility projection step, applied to an initially infeasible solution. In this example, the instance parameters are set to $a = 4.16$ and $b = 2.19$. The relaxed solution $(\bar{x}, \bar{y})$ is iteratively adjusted through gradient-based updates, reducing constraint violations while preserving the integer feasibility enforced by the correction layer. As the projection process progresses, the solution moves towards the feasible region, forcing that the final mixed-integer solution $(\hat{x}, \hat{y})$ satisfies all constraints.

# C Theoretical Guarantees for Integer Feasibility Projection

## C.1 Asymptotic Convergence

*Proof.* Now we prove Theorem 1 in the following steps.

**L-smoothness of $\nabla\mathcal{V}$ on compact subsets of $\mathcal{D}$.** Define the region of interest as $\mathcal{D} := \{x \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) > 0\}$, which consists of all points where at least one constraint is violated. That is, there exists $j \in \{1, \ldots, n_c\}$ such that $g_j(\varphi(\mathbf{x})) > 0$. The penalty function is given by

$$\mathcal{V}(\mathbf{x}) = \|\mathbf{g}(\varphi(\mathbf{x}))_+\|_1 = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x}))).$$

For any $x \in \mathcal{D}$, define the active set:

$$I_{\mathbf{x}} := \{j \in \{1, \ldots, n_c\} : g_j(\varphi(\mathbf{x})) > 0\}.$$

On this set, $\max(0, g_j(\varphi(\mathbf{x}))) = g_j(\varphi(\mathbf{x}))$, which is smooth since both $g_j$ and $\varphi$ are continuously differentiable. The terms with $g_j(\varphi(\mathbf{x})) \leq 0$ contribute a constant zero and hence do not affect differentiability. Thus, the function simplifies to:

$$\mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} g_j(\varphi(\mathbf{x})),$$

and is differentiable on $\mathcal{D}$. By the chain rule, its gradient is:

$$\nabla\mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} \nabla\left[g_j(\varphi(\mathbf{x}))\right] = \sum_{j \in I_{\mathbf{x}}} \nabla\varphi(\mathbf{x})^\top \nabla g_j(\varphi(\mathbf{x}))$$

Let $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{D}$ belong to a compact subset $\mathcal{K} \subset \mathcal{D}$. Define the union of their active sets $I_{\mathbf{x}_1} \cup I_{\mathbf{x}_2}$, and let $|I_{\mathbf{x}_1} \cup I_{\mathbf{x}_2}|$ denote the number of active constraints in this union. We compare:

$$\|\nabla \mathcal{V}(\mathbf{x}_1) - \nabla \mathcal{V}(\mathbf{x}_2)\| \leq \sum_{j \in I_{\mathbf{x}}} \left\| \nabla \varphi(\mathbf{x}_1)^\top \nabla g_j(\varphi(\mathbf{x}_1)) - \nabla \varphi(\mathbf{x}_2)^\top \nabla g_j(\varphi(\mathbf{x}_2)) \right\|.$$

Each term is bounded by:

$$\begin{aligned} &\left\| \nabla \varphi(\mathbf{x}_1)^\top \nabla g_j(\varphi(\mathbf{x}_1)) - \nabla \varphi(\mathbf{x}_2)^\top \nabla g_j(\varphi(\mathbf{x}_2)) \right\| \\ &\quad \leq \|\nabla \varphi(\mathbf{x}_1) - \nabla \varphi(\mathbf{x}_2)\| \cdot \|\nabla g_j(\varphi(\mathbf{x}_1))\| \\ &\qquad + \|\nabla \varphi(\mathbf{x}_2)\| \cdot \|\nabla g_j(\varphi(\mathbf{x}_1)) - \nabla g_j(\varphi(\mathbf{x}_2))\|. \end{aligned}$$

By the assumptions:

- $\|\nabla \varphi(\mathbf{x}_1) - \nabla \varphi(\mathbf{x}_2)\| \leq L_\varphi \|\mathbf{x}_1 - \mathbf{x}_2\|$,
- $\|\varphi(\mathbf{x}_1) - \varphi(\mathbf{x}_2)\| \leq G_\varphi \|\mathbf{x}_1 - \mathbf{x}_2\|$,
- $\|\nabla g_j(\varphi(\mathbf{x}_1)) - \nabla g_j(\varphi(\mathbf{x}_2))\| \leq L_{\mathbf{g}} G_\varphi \|\mathbf{x}_1 - \mathbf{x}_2\|$,
- $\|\nabla g_j(\varphi(\mathbf{x}))\| \leq G_{\mathbf{g}}$, $\|\nabla \varphi(\mathbf{x})\| \leq G_\varphi$.

Putting it together:

$$\|\nabla \mathcal{V}(\mathbf{x}_1) - \nabla \mathcal{V}(\mathbf{x}_2)\| \leq |I_{\mathbf{x}_1} \cup I_{\mathbf{x}_2}| \cdot (G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}}) \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|.$$

Therefore, $\nabla \mathcal{V}$ is Lipschitz continuous on compact subsets of $\mathcal{D}$, with local Lipschitz constant:

$$L(\mathbf{x}_1, \mathbf{x}_2) := |I_{\mathbf{x}_1} \cup I_{\mathbf{x}_2}| \cdot (G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}}).$$

In particular, the Lipschitz constant of $\nabla \mathcal{V}$ depends only on the number of active constraints at $\mathbf{x}_1$ and $\mathbf{x}_2$. If the number of active constraints $|I_{\mathbf{x}}|$ is uniformly bounded by $\bar{n}_c \ll n_c$, i.e., only a few constraints are typically active, the global Lipschitz constant estimate $L = \bar{n}_c (G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}})$ becomes significantly tighter than the worst-case constant $L = n_c (G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}})$.

**Descent lemma and vanishing gradient norm.** Because $\mathcal{V}$ is differentiable with $L$-Lipschitz gradient, the standard descent lemma for gradient descent implies:

$$\mathcal{V}(\mathbf{x}^{(k+1)}) \leq \mathcal{V}(\mathbf{x}^{(k)}) - \eta \left(1 - \frac{L\eta}{2}\right) \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

If $\eta \in (0, 1/L]$, then $1 - \frac{L\eta}{2} \geq 1/2$, and so:

$$\mathcal{V}(\mathbf{x}^{(k+1)}) \leq \mathcal{V}(\mathbf{x}^{(k)}) - \frac{\eta}{2} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

This shows $\{\mathcal{V}(\mathbf{x}^{(k)})\}$ is non-increasing and bounded below by 0, hence converges to some finite $\mathcal{V}^\star \geq 0$.
Summing from $k = 0$ to $K - 1$:

$$\sum_{k=0}^{K-1} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2 \leq \frac{2}{\eta} \left( \mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \right) \leq \frac{2}{\eta} \mathcal{V}(\mathbf{x}^{(0)}).$$

Therefore:

$$\sum_{k=0}^{\infty} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2 < \infty \quad \Rightarrow \quad \lim_{k \to \infty} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| = 0.$$

**Convergence to Feasibility.** Suppose by contradiction that $\mathcal{V}^\star > 0$. Then based on the descent lemma and vanishing gradient there exists a subsequence $\{\mathbf{x}^{(k_j)}\}$ converging to $\mathbf{x}^* \in \mathcal{D}$ such that $\mathcal{V}(\mathbf{x}^*) > 0$, and $\nabla \mathcal{V}(\mathbf{x}^*) = 0$. Since the ReLU penalty is smooth at points where $g_j(\varphi(\mathbf{x}^*)) > 0$, the gradient of $\mathcal{V}$ at $\mathbf{x}^*$ is given by:

$$\nabla \mathcal{V}(\mathbf{x}^*) = \sum_{j \in I_{\mathbf{x}^*}} \nabla \varphi(\mathbf{x}^*)^\top \nabla g_j(\varphi(\mathbf{x}^*)).$$

Now recall that $\mathcal{V}^\star > 0$ only if $\mathbf{g}(\varphi(\mathbf{x}^*)) > 0$ componentwise, i.e., some inequality constraints are violated after surrogate rounding. Now, assuming at least one of the violated constraints has a nonzero gradient:

$$\exists j \in \{1, \ldots, n_c\} \text{ such that } g_j(\varphi(\mathbf{x}^*)) > 0 \text{ and } \nabla g_j(\varphi(\mathbf{x}^*)) \neq 0.$$

Because the sum $\sum_{j \in I_{\mathbf{x}^*}} \nabla\varphi(\mathbf{x}^*)^\top \nabla g_j(\varphi(\mathbf{x}^*)) = 0$ and at least one $\nabla g_j(\varphi(\mathbf{x}^*)) \neq 0$, the only way for the expression to vanish is if $\nabla\varphi(\mathbf{x}^*) = 0$. Therefore:

$$\nabla\mathcal{V}(\mathbf{x}^*) = 0 \Rightarrow \nabla\varphi(\mathbf{x}^*) = 0.$$

But by Theorem 4, gradient descent cannot converge to any critical point $\mathbf{x}^* \in \mathcal{D}$ with $\mathcal{V}(\mathbf{x}^*) > 0$ that is not a local minimum of $\mathcal{V}$. Under this framework, the structural result of Theorem 5 further ensures that strict local minima of $\varphi$ do not correspond to local minima of $\mathcal{V}$, and hence cannot act as attractors. Therefore, such points $\mathbf{x}^*$ cannot be limit points of the iterates. This contradiction shows that no accumulation point of $\{\mathbf{x}^{(k)}\}$ can satisfy $\mathcal{V}(\mathbf{x}^*) > 0$, so we must have:

$$\lim_{k\to\infty} \mathcal{V}(\mathbf{x}^{(k)}) = 0.$$

□

### C.2 Non-Asymptotic Convergence

*Proof.* Now we prove Theorem 2.

Since $\mathcal{V}$ is differentiable and has $L$-Lipschitz gradient on $\mathcal{D}$, and gradient descent is performed with step size $\eta \leq \frac{1}{L}$, we have:

$$\mathcal{V}(\mathbf{x}^{(k+1)}) \leq \mathcal{V}(\mathbf{x}^{(k)}) - \frac{\eta}{2}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Summing from $k = 0$ to $K - 1$ gives:

$$\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \geq \frac{\eta}{2} \sum_{k=0}^{K-1} \|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

So:

$$\sum_{k=0}^{K-1} \|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2 \leq \frac{2}{\eta}\left(\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}^\star\right).$$

Dividing by $K$, we obtain:

$$\min_{0\leq k<K} \|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2 \leq \frac{2}{\eta K}\left(\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}^\star\right).$$

To ensure $\min_k \|\nabla\mathcal{V}(\mathbf{x}^{(k)})\| \leq \delta$, it suffices to choose

$$K \geq \frac{2}{\eta\delta^2}\left(\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}^\star\right).$$

Finally, if $\mathcal{V}^\star = 0$, then $\mathcal{V}(\mathbf{x}^{(k)}) \to 0$, so for any $\epsilon > 0$, there exists $K_\epsilon$ such that

$$\mathcal{V}(\mathbf{x}^{(k)}) < \epsilon \quad \text{for all } k \geq K_\epsilon.$$

Thus, the iterates eventually enter and remain in the approximate feasible region $S_\epsilon := \{x \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) < \epsilon\}$. □

*Corollary* 3 (Iteration Complexity for Approximate Feasibility via Integer Feasibility Projection). Suppose the conditions of Theorem 2 hold and that the infimum value satisfies $\mathcal{V}^\star = 0$. Then for any tolerance $\epsilon > 0$, gradient descent with step size $\eta \in \left(0, \frac{1}{L}\right]$ will produce an iterate $\mathbf{x}^{(k)}$ satisfying:

$$\mathcal{V}(\mathbf{x}^{(k)}) < \epsilon$$

after at most

$$K_\epsilon := \left\lceil \frac{2}{\eta\epsilon}\mathcal{V}(\mathbf{x}^{(0)}) \right\rceil$$

iterations. That is,

$$\mathbf{x}^{(k)} \in S_\epsilon := \{x \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) < \epsilon\} \quad \text{for all } k \geq K_\epsilon.$$

*Proof.* Now we prove Corollary 3.

We begin with the inequality established in the proof of Theorem 2, which follows from the descent lemma for gradient descent with an $L$-Lipschitz smooth function and step size $\eta \leq \frac{1}{L}$:

$$\mathcal{V}(\mathbf{x}^{(k+1)}) \leq \mathcal{V}(\mathbf{x}^{(k)}) - \frac{\eta}{2}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Summing this inequality from $k = 0$ to $K - 1$ yields:

$$\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \geq \frac{\eta}{2}\sum_{k=0}^{K-1}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Since $\mathcal{V}(\mathbf{x}^{(K)}) \geq 0$, we have:

$$\mathcal{V}(\mathbf{x}^{(K)}) \leq \mathcal{V}(\mathbf{x}^{(0)}) - \frac{\eta}{2}\sum_{k=0}^{K-1}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Now assume for contradiction that $\mathcal{V}(\mathbf{x}^{(k)}) \geq \epsilon$ for all $k < K$. Then, since $\mathcal{V}$ is non-increasing, this implies $\mathcal{V}(\mathbf{x}^{(k)}) \geq \epsilon$ for all $k < K$, and so:

$$\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \geq \mathcal{V}(\mathbf{x}^{(0)}) - \epsilon.$$

At the same time, from the descent inequality:

$$\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \geq \frac{\eta}{2}\sum_{k=0}^{K-1}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

We bound the sum from below by using the minimum value over the sequence:

$$\sum_{k=0}^{K-1}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2 \geq K \cdot \min_{0\leq k<K}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Then

$$\mathcal{V}(\mathbf{x}^{(0)}) - \mathcal{V}(\mathbf{x}^{(K)}) \geq \frac{\eta}{2}K \cdot \min_{0\leq k<K}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Since $\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\| \geq 0$ and $\mathcal{V}(\mathbf{x}^{(k)}) \geq \epsilon$, the total descent must continue until $\mathcal{V}(\mathbf{x}^{(K)}) < \epsilon$. Solving for $K$, the smallest $K$ such that $\mathcal{V}(\mathbf{x}^{(K)}) < \epsilon$ must satisfy:

$$\mathcal{V}(\mathbf{x}^{(0)}) - \epsilon < \frac{\eta}{2}K \cdot \min_{0\leq k<K}\|\nabla\mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

To guarantee this, a sufficient condition is to ensure:

$$\mathcal{V}(\mathbf{x}^{(K)}) < \epsilon \quad \text{whenever} \quad K \geq \frac{2}{\eta\epsilon}\mathcal{V}(\mathbf{x}^{(0)}).$$

Therefore, setting

$$K_\epsilon := \left\lceil \frac{2}{\eta\epsilon}\mathcal{V}(\mathbf{x}^{(0)}) \right\rceil$$

guarantees that $\mathcal{V}(\mathbf{x}^{(k)}) < \epsilon$ for all $k \geq K_\epsilon$, or in other words $\mathbf{x}^{(k)} \in S_\epsilon$ for all $k \geq K_\epsilon$. □

*Remark* 2 (On Regularity and Practical Tightness of Lipschitz Assumptions). The convergence analysis in Theorem 1 and Theorem 2 assumes that $\mathbf{g}$ and $\varphi$ are $C^1$ with Lipschitz continuous gradients and bounded Jacobians. These conditions are realistic in practice:

- ReLU penalty function is smooth on $\mathcal{D} := \{\mathbf{x} : \mathcal{V}(\mathbf{x}) > 0\}$, and thus does not interfere with differentiability of $\mathcal{V}$.
- The constraint map $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n_c}$ consists of smooth nonlinear functions (e.g., polynomials, exponentials, or trigonometric expressions) defined on a compact domain such as $[0, 1]^n$. Under this assumption, $\mathbf{g} \in C^2$ and its Jacobian $\nabla\mathbf{g}$ is Lipschitz continuous and bounded.

- Surrogate rounding $\varphi$ is in our case approximated with Gumbel-Sigmoid and scaled sigmoid functions, which are smooth and parameterized by a temperature or sharpness term. When this parameter is finite and bounded away from zero, $\varphi$ is $C^2$, with bounded and Lipschitz continuous gradient. Smoothness and boundedness are easily satisfied when the neural component of $\varphi$ is implemented using standard smooth activations over a compact domain.

The bound $L := \bar{n}_c(G_{\mathbf{g}} L_\varphi + G_\varphi L_{\mathbf{g}})$ used in Theorem 1 is conservative. In many applications, $\mathbf{g}(\varphi(\mathbf{x}))$ is sparse or low-rank, and individual $g_j$ depend on a few coordinates of $\varphi(\mathbf{x})$, allowing for significantly smaller effective Lipschitz constants. Exploiting such a structure can yield tighter complexity bounds and improved convergence in practice.

### C.3 Asymptotic Convergence Based on the Łojasiewicz Inequality

Theorem 1 relies on mild regularity assumptions, as noted in Remark 2. Alternatively, convergence can be shown using the Łojasiewicz inequality by leveraging the analytic or subanalytic structure of $\mathbf{g}$ and $\varphi$, without requiring explicit smoothness or curvature bounds.

**Theorem 4** (Feasibility Convergence of Integer Feasibility Projection via Łojasiewicz Inequality)**.** *Let $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$, where $\mathbf{g} \in C^\omega$ (real analytic) and $\varphi \in C^\infty$ are composed with piecewise-linear ReLU, and suppose $\mathcal{V}(\mathbf{x}) > 0$ on the region $\mathcal{D} \subset \mathbb{R}^n$. Then $\mathcal{V}$ is subanalytic and satisfies the Łojasiewicz gradient inequality at every critical point $\mathbf{x}^* \in \mathcal{D}$. As a result, for any initialization $\mathbf{x}^{(0)} \in \mathcal{D}$, the gradient descent iterates*

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \nabla \mathcal{V}(\mathbf{x}^{(k)}), \quad \eta \in (0, 1/L],$$

*converge to a critical point $\mathbf{x}^\infty \in \mathcal{D}$, and $\mathcal{V}(\mathbf{x}^{(k)}) \to \mathcal{V}(\mathbf{x}^\infty)$. If further, all critical points $\mathbf{x}^* \in \mathcal{D}$ with $\mathcal{V}(\mathbf{x}^*) > 0$ are not local minima of $\mathcal{V}$, then*

$$\lim_{k \to \infty} \mathcal{V}(\mathbf{x}^{(k)}) = 0.$$

*Proof.* We divide the proof of Theorem 4 into two parts: convergence of gradient descent to a critical point, and convergence to feasibility under a non-minimality assumption.

**Subanalyticity and Łojasiewicz inequality.** Since $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n_c}$ is real analytic and $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ is $C^\infty$, their composition $\mathbf{g} \circ \varphi$ is real analytic as well. The ReLU function $\max(0, z_j)$ is piecewise analytic and semialgebraic. Therefore, the function $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$ is a finite sum of semialgebraic (hence subanalytic) functions composed with analytic mappings, and is itself a *subanalytic* and $C^1$ function on the open region

$$\mathcal{D} := \{\mathbf{x} \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) > 0\}.$$

By a standard result in nonsmooth analysis (see [78]), every $C^1$ subanalytic function satisfies the *Łojasiewicz gradient inequality* at all its critical points. That is, for each critical point $\mathbf{x}^* \in \mathcal{D}$, there exist constants $C > 0$, $\theta \in [0, 1)$, and a neighborhood $\mathcal{U} \subset \mathcal{D}$ of $\mathbf{x}^*$ such that:

$$\|\nabla \mathcal{V}(\mathbf{x})\| \geq C(\mathcal{V}(\mathbf{x}) - \mathcal{V}(\mathbf{x}^*))^\theta, \quad \forall x \in \mathcal{U}.$$

**Convergence of gradient descent to a critical point.** Let $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \nabla \mathcal{V}(\mathbf{x}^{(k)})$ be the gradient descent update with constant step size $\eta \in (0, 1/L]$, where $L$ is a Lipschitz constant for $\nabla \mathcal{V}$ on compact subsets of $\mathcal{D}$ (as shown in Theorem 1). The descent lemma implies:

$$\mathcal{V}(\mathbf{x}^{(k+1)}) \leq \mathcal{V}(\mathbf{x}^{(k)}) - \frac{\eta}{2} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2.$$

Since $\mathcal{V}(\mathbf{x}^{(k)}) \geq 0$ and is non-increasing, it converges to a finite limit $\mathcal{V}^\star \geq 0$. Furthermore, summing the descent inequality gives:

$$\sum_{k=0}^{\infty} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\|^2 < \infty, \quad \Rightarrow \quad \lim_{k \to \infty} \|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| = 0.$$

Hence, the sequence $\{\mathbf{x}^{(k)}\}$ has at least one accumulation point $\mathbf{x}^\infty \in \mathcal{D}$ with $\nabla \mathcal{V}(\mathbf{x}^\infty) = 0$, i.e., a critical point. Since $\mathcal{V}$ is subanalytic and satisfies the Łojasiewicz inequality at all critical points, the full sequence $\mathbf{x}^{(k)} \to \mathbf{x}^\infty$, as shown by the convergence theorem for gradient descent on Łojasiewicz functions (see [79]).

**Convergence to feasibility under non-minimality.** Suppose now that $\mathbf{x}^\infty \in \mathcal{D}$ is a critical point with $\mathcal{V}(\mathbf{x}^\infty) > 0$. Then, by the Łojasiewicz inequality:

$$\|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| \geq C(\mathcal{V}(\mathbf{x}^{(k)}) - \mathcal{V}(\mathbf{x}^\infty))^\theta.$$

But this contradicts the fact that $\|\nabla \mathcal{V}(\mathbf{x}^{(k)})\| \to 0$ unless $\mathcal{V}(\mathbf{x}^{(k)}) \to \mathcal{V}(\mathbf{x}^\infty)$. Thus $\mathcal{V}(\mathbf{x}^{(k)}) \to \mathcal{V}(\mathbf{x}^\infty) > 0$. To rule this out, we assume that no such point $\mathbf{x}^\infty \in \mathcal{D}$ with $\mathcal{V}(\mathbf{x}^\infty) > 0$ is a *local minimum* of $\mathcal{V}$. That is, every critical point with $\mathcal{V}(\mathbf{x}^*) > 0$ is a saddle or otherwise unstable. This ensures that the gradient descent trajectory cannot converge to any such point $\mathbf{x}^\infty$. Hence, it must converge to a point where $\mathcal{V}(\mathbf{x}^\infty) = 0$, i.e., feasibility is achieved:

$$\lim_{k \to \infty} \mathcal{V}(\mathbf{x}^{(k)}) = 0.$$

□

*Remark* 3 (Relationship Between Theorems 1 and 4). Theorem 1 shows that under a mild structural condition, gradient descent converges to feasibility with vanishing gradient norm. Theorem 4 provides a complementary convergence result using the Łojaśiewicz gradient inequality, which is automatically satisfied due to the analytic structure of the composite function $\mathcal{V}$ representing our integer feasibility projection. This yields convergence to feasibility without requiring explicit curvature conditions. Hence, generalizing the convergence guarantees to non-smooth but subanalytic neural surrogates of the rounding operation $\varphi$, such as those with ReLU activation functions. However, this comes at the expense of losing explicit rates.

### C.4 Feasibility Convergence Despite Local Minima in the Correction Layer

While Theorem 4 guarantees convergence to feasibility under a general non-minimality condition, we now show that this condition is satisfied for a wide class of problematic critical points, namely, strict local minima of the integer correction layer $\varphi$.

**Theorem 5** (Strict Local Minima of $\varphi$ Do Not Trap Gradient Descent with ReLU-L1 Penalty)**.** *Let $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ be twice continuously differentiable, and let $\mathbf{g} : \mathbb{R}^n \to \mathbb{R}^{n_c}$ be continuously differentiable. Define the ReLU-based penalty function:*

$$\mathcal{V}(\mathbf{x}) := \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x}))).$$

*Suppose $\mathbf{x}^* \in \mathbb{R}^n$ is a strict local minimum of $\varphi$, i.e.,*

$$\nabla \varphi(\mathbf{x}^*) = 0, \quad \nabla^2 \varphi(\mathbf{x}^*) \succ 0,$$

*and that at least one constraint is violated at $\mathbf{x}^*$, i.e., $g_j(\varphi(\mathbf{x}^*)) > 0$ for some $j$. Then*

1. *$\nabla \mathcal{V}(\mathbf{x}^*) = 0$, so $\mathbf{x}^*$ is a stationary point of $\mathcal{V}$,*
2. *$\mathbf{x}^*$ is not a local minimum of $\mathcal{V}$,*
3. *For small perturbations $\delta$, $\nabla \mathcal{V}(\mathbf{x}^* + \delta) \neq 0$ generically,*
4. *Therefore, gradient descent initialized near $\mathbf{x}^*$ will escape.*

*Proof.* Now we prove Theorem 5 in the following steps.

**Stationarity.** We define the active index set at $\mathbf{x}^*$ as:

$$I_{\mathbf{x}} := \{j : g_j(\varphi(\mathbf{x}^*)) > 0\}.$$

Since $g_j(\varphi(\mathbf{x}^*)) > 0$, the ReLU is smooth at those points, and we have:

$$\mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} g_j(\varphi(\mathbf{x})) \quad \text{in a neighborhood of } \mathbf{x}^*.$$

Thus, $\mathcal{V}$ is differentiable at $\mathbf{x}^*$, and by the chain rule:

$$\nabla \mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} \nabla \varphi(\mathbf{x})^\top \nabla g_j(\varphi(\mathbf{x})).$$

At $\mathbf{x}^*$, $\nabla \varphi(\mathbf{x}^*) = 0$, so:

$$\nabla \mathcal{V}(\mathbf{x}^*) = 0.$$

**Not a Local Minimum of $\mathcal{V}$.** We now analyze the second-order behavior at $\mathbf{x}^*$. Since:

$$\mathcal{V}(\mathbf{x}) = \sum_{j \in I_{\mathbf{x}}} g_j(\varphi(\mathbf{x})),$$

we compute the Hessian:

$$\nabla^2 \mathcal{V}(\mathbf{x}^*) = \sum_{j \in I_{\mathbf{x}}} \nabla^2 (g_j \circ \varphi)(\mathbf{x}^*).$$

Each term expands as:

$$\nabla^2 (g_j \circ \varphi)(\mathbf{x}^*) = \nabla^2 \varphi(\mathbf{x}^*)^\top \nabla g_j(\varphi(\mathbf{x}^*)) + \nabla \varphi(\mathbf{x}^*)^\top \nabla^2 g_j(\varphi(\mathbf{x}^*)) \nabla \varphi(\mathbf{x}^*).$$

Since $\nabla \varphi(\mathbf{x}^*) = 0$, the second term vanishes, and we get:

$$\nabla^2 \mathcal{V}(\mathbf{x}^*) = \sum_{j \in I_{\mathbf{x}}} \nabla^2 \varphi(\mathbf{x}^*)^\top \nabla g_j(\varphi(\mathbf{x}^*)).$$

Now observe that this expression is a weighted sum of the positive definite matrix $\nabla^2 \varphi(\mathbf{x}^*)$ scaled by the (possibly signed) vectors $\nabla g_j(\varphi(\mathbf{x}^*))$. Unless all $\nabla g_j(\varphi(\mathbf{x}^*))$ are positively aligned with the curvature of $\varphi$, this sum may introduce directions of negative or zero curvature. In particular, if the set $\{\nabla g_j(\varphi(\mathbf{x}^*))\}_{j \in I_{\mathbf{x}}}$ spans directions that are not all strictly aligned with the eigenvectors of $\nabla^2 \varphi(\mathbf{x}^*)$, then the resulting Hessian $\nabla^2 \mathcal{V}(\mathbf{x}^*)$ will be indefinite. A sufficient condition for $\mathbf{x}^*$ to *not* be a local minimum of $\mathcal{V}$ is that there exists $j \in I_{\mathbf{x}}$ and a direction $v \in \mathbb{R}^n$ such that:

$$v^\top \nabla^2 \varphi(\mathbf{x}^*) v > 0 \quad \text{and} \quad v^\top \nabla g_j(\varphi(\mathbf{x}^*)) < 0.$$

In this case, the term $\nabla^2 \varphi(\mathbf{x}^*)^\top \nabla g_j(\varphi(\mathbf{x}^*))$ contributes negative curvature along $v$. Therefore, unless all gradients $\nabla g_j(\varphi(\mathbf{x}^*))$ are strictly positive multiples of a single direction compatible with the curvature of $\varphi$, the composite Hessian $\nabla^2 \mathcal{V}(\mathbf{x}^*)$ will be indefinite, and $\mathbf{x}^*$ will not be a local minimum of $\mathcal{V}$.

**Perturbation Analysis.** For a small perturbation $\delta$, the gradient of $\varphi$ becomes:

$$\nabla \varphi(\mathbf{x}^* + \delta) = \nabla^2 \varphi(\mathbf{x}^*) \delta + o(\|\delta\|) \neq 0,$$

since $\nabla^2 \varphi(\mathbf{x}^*) \succ 0$. Also, by continuity:

$$\nabla g_j(\varphi(\mathbf{x}^* + \delta)) \to \nabla g_j(\varphi(\mathbf{x}^*)).$$

Hence,

$$\nabla \mathcal{V}(\mathbf{x}^* + \delta) = \sum_{j \in I_{\mathbf{x}}} \nabla \varphi(\mathbf{x}^* + \delta)^\top \nabla g_j(\varphi(\mathbf{x}^* + \delta)) \neq 0$$

for generic small $\delta$.

**Escape from $\mathbf{x}^*$.** Since $\nabla \mathcal{V}(\mathbf{x}) \neq 0$ in a neighborhood around $\mathbf{x}^*$, gradient descent will not be trapped at $\mathbf{x}^*$. Any initialization near $\mathbf{x}^*$ will result in descent away from the point. □

*Remark* 4 (Generic Nondegeneracy of Constraint Gradients and Curvature Alignment). Two structural assumptions play a key role in the analysis of convergence to feasibility:

**Nonvanishing gradients at violated constraints.** The assumption in Theorem 1 that some active constraint $g_j(\varphi(\mathbf{x}^*)) > 0$ satisfies $\nabla g_j(\varphi(\mathbf{x}^*)) \neq 0$ is generic. If each $g_j$ is real analytic or $C^1$ and non-constant, then the set

$$Z_j := \{\mathbf{x} \in \mathbb{R}^n : g_j(\mathbf{x}) > 0 \text{ and } \nabla g_j(\mathbf{x}) = 0\}$$

has measure zero. Hence, the probability that a randomly initialized or dynamically reached point $\varphi(\mathbf{x}^*)$ lies in such a degenerate set is zero in the measure-theoretic sense. So the assumption holds almost surely.

**Misalignment of constraint gradients.** The condition that all $\nabla g_j(\varphi(\mathbf{x}^*))$ align with the curvature of $\varphi$ is highly non-generic. In practice, $\nabla g_j$ typically vary independently across $j$, resulting in misalignment. As a result, the composite Hessian

$$\nabla^2 \mathcal{V}(\mathbf{x}^*) = \sum_{j \in I_{\mathbf{x}^*}} \nabla^2 (g_j \circ \varphi)(\mathbf{x}^*)$$

is generically indefinite at $\mathbf{x}^* \in \mathcal{D}$ because the presence of even a single direction $v \in \mathbb{R}^n$ such that $v^\top \nabla^2 \varphi(\mathbf{x}^*) v > 0$ and $v^\top \nabla g_j(\varphi(\mathbf{x}^*)) < 0$ causes the composite Hessian to be indefinite. Therefore, under mild and generic assumptions on $\mathbf{g}$, any point $\mathbf{x}^*$ that is a strict local minimum of $\varphi$ and satisfies $\mathcal{V}(\mathbf{x}^*) > 0$ will not be a local minimum of $\mathcal{V}$. This ensures that gradient descent will escape from such points.

### C.5 Exclusion of Degenerate Convergence Scenarios

*Remark* 5 (Plateau Behavior and Subanalyticity). One may wonder whether the penalty function $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$ can exhibit plateau-like behavior, i.e., regions where $\mathcal{V}$ remains constant and positive, within the infeasible region $\mathcal{D} := \{\mathbf{x} : \mathcal{V}(\mathbf{x}) > 0\}$. We address two such scenarios below.

**Flat critical manifolds.** A flat manifold is a set $M \subset \mathcal{D}$ where $\nabla \mathcal{V}(\mathbf{x}) = 0$ and $\mathcal{V}(\mathbf{x}) = c > 0$ for all $x \in M$. These sets could trap gradient descent if they existed with positive measure. However, because $\mathcal{V}$ is subanalytic and $C^1$ on $\mathcal{D}$, it satisfies the Kurdyka–Łojasiewicz (KL) inequality near all critical points [78]. This rules out the existence of non-isolated flat critical sets unless $\mathcal{V}$ is locally constant, which we now argue is also structurally implausible. Moreover, known convergence results for KL functions [79] imply that gradient descent cannot asymptotically converge to a non-isolated flat critical manifold unless it is initialized there. In typical smooth machine learning problems, such events occur with probability zero under random initialization. Therefore, the subanalyticity of $\mathcal{V}$ implies that flat critical manifolds are unstable under gradient descent.

**Constant regions.** Suppose, for contradiction, that $\mathcal{V}(\mathbf{x}) = c > 0$ on an open subset $U \subset \mathcal{D}$. Then each active term $j \in I_{\mathbf{x}} := \{j : g_j(\varphi(\mathbf{x})) > 0\}$ must be constant over $U$, implying that the compositions $g_j \circ \varphi$ are locally constant. This in turn forces their gradients to vanish: $\nabla(g_j \circ \varphi)(\mathbf{x}) = 0$ for all $x \in U$. Unless $g_j \circ \varphi$ is identically constant—a non-generic scenario—this condition fails on open sets.

**Implication.** Together, subanalytic regularity and mild structural assumptions that $g_j \circ \varphi$ are not constant functions ensure that $\mathcal{V}$ cannot be locally constant on any open subset of $\mathcal{D}$. Therefore, genuine plateaus or flat manifolds that could trap gradient descent do not arise in typical settings.

*Remark* 6 (Critical Points and Optimization Challenges). The penalty function $\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x})))$ is piecewise smooth and subanalytic on the infeasible region $\mathcal{D} := \{\mathbf{x} : \mathcal{V}(\mathbf{x}) > 0\}$. A natural question is whether gradient descent could become trapped at infeasible critical points. Two classes of critical points could, in principle, obstruct convergence.

**Non-degenerate local minima.** Points $\mathbf{x}^* \in \mathcal{D}$ where $\nabla \mathcal{V}(\mathbf{x}^*) = 0$, $\nabla^2 \mathcal{V}(\mathbf{x}^*) \succ 0$, and $\mathcal{V}(\mathbf{x}) > \mathcal{V}(\mathbf{x}^*)$ locally. These may arise if the active constraint set $I_{\mathbf{x}^*} := \{j : g_j(\varphi(\mathbf{x}^*)) > 0\}$ is fixed and the composite Hessian

$$\nabla^2 \mathcal{V}(\mathbf{x}^*) = \sum_{j \in I_{\mathbf{x}^*}} \nabla^2 (g_j \circ \varphi)(\mathbf{x}^*)$$

is positive definite. However, such configurations require fine alignment between $\nabla g_j$ and the curvature of $\varphi$, which is highly non-generic.

**Flat saddles.** Isolated critical points where $\nabla \mathcal{V}(\mathbf{x}^*) = 0$ and the Hessian is degenerate (e.g., zero eigenvalues). These may occur under degeneracy or saturation in $\varphi$, or when multiple $\nabla g_j(\varphi(\mathbf{x}))$ vanish. While subanalyticity rules out flat critical *manifolds*, it does not preclude such isolated saddles.

#### C.5.1 Taxonomy of Feasibility Convergence

Feasibility convergence is ensured by one of two assumptions:

- *Structural:* If every $\mathbf{x}^* \in \mathcal{D}$ with $\mathcal{V}(\mathbf{x}^*) > 0$ has some active constraint $g_j(\varphi(\mathbf{x}^*)) > 0$ with $\nabla g_j(\varphi(\mathbf{x}^*)) \neq 0$, then $\nabla \mathcal{V}(\mathbf{x}^*) \neq 0$, so infeasible stationary points are excluded. This condition appears in Theorem 1 and rules out non-degenerate local minima.
- *Dynamical:* Alternatively, assume that no infeasible stationary point $\mathbf{x}^* \in \mathcal{D}$ attracts nearby trajectories under gradient descent:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \eta \nabla \mathcal{V}(\mathbf{x}^{(k)}).$$

That is, for every neighborhood $B_\rho(\mathbf{x}^*) := \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}^*\| < \rho\}$, some $\mathbf{x} \in B_\rho(\mathbf{x}^*) \cap \mathcal{D}$ generates a trajectory that does not converge to $\mathbf{x}^*$. This allows such critical points to exist but ensures they do not trap iterates, especially relevant for flat saddles or degenerate cases not excluded structurally.

**Implication.** Whether through gradient non-vanishing or non-attraction, infeasible critical points are generically avoided. Combined with the subanalytic geometry of $\mathcal{V}$, these conditions help explain why gradient descent almost always escapes infeasible regions in practice. Moreover, stochastic methods and perturbation-based algorithms [80] have been shown to escape strict and flat saddles in polynomial time under mild conditions.

*Proposition* 1 (Characterization of Critical Point Behavior). Let $\mathbf{x}^* \in \mathcal{D} := \{\mathbf{x} \in \mathbb{R}^n : \mathcal{V}(\mathbf{x}) > 0\}$ be a stationary point of the ReLU-penalized objective

$$\mathcal{V}(\mathbf{x}) = \sum_{j=1}^{n_c} \max(0, g_j(\varphi(\mathbf{x}))).$$

Then, the behavior of gradient descent near $\mathbf{x}^*$ depends on the type of critical point as follows:

| Critical Point Type | Addressed in | GD Converges? | Feasibility |
|---|---|---|---|
| Strict local minimum of $\varphi$ | Thm 5 | No | Yes |
| Strict saddle of $\mathcal{V}$ | Thm 4 | No (almost surely) | Yes |
| Non-isolated saddle of $\mathcal{V}$ | Thm 4 | No (generically) | Yes |
| Flat critical manifold, $\mathcal{V}(\mathbf{x}) > 0$ | Rem. 5 | No (subanalytic) | Yes |
| Locally constant region of $\mathcal{V}$ | Rem. 5 | No (subanalytic) | Yes |
| Non-degenerate saddle of $\mathcal{V}$ | Thm 4 | No (stable under perturbation) | Yes |
| Non-degenerate local minimum of $\mathcal{V}$ | Rem. 6 | Yes (if present but very unlikely) | No |
| Flat saddle of $\mathcal{V}$ | Rem. 6 | Unclear | Open |
| Degenerate saddle of $\mathcal{V}$ | Rem. 6 | Unclear | Open |

In summary:

- **Feasibility convergence justification.** Infeasible critical points are either excluded structurally (via gradient non-vanishing; see Theorem 1) or are assumed to be non-attracting (see Remark 6). These complementary perspectives explain why convergence to feasibility occurs in practice.
- **Strict local minima of** $\varphi$ are ruled out as attractors by Theorem 5, since they are not minima of $\mathcal{V}$.
- **Strict saddles**, **non-isolated saddles**, **flat manifolds**, and **locally constant regions** are generically avoided due to the subanalytic structure of $\mathcal{V}$, which guarantees the Łojasiewicz (KL) property and precludes convergence to non-isolated critical sets, see Theorem 4, and Remark 5.
- **Non-degenerate saddles** are generically escaped by gradient descent due to instability in directions of negative curvature.
- **Non-degenerate local minima of** $\mathcal{V}$ with $\mathcal{V}(\mathbf{x}^*) > 0$ may exist but are structurally rare and require unlikely gradient-curvature alignment, see Remark 6.
- **Degenerate or flat saddles** are not ruled out by subanalyticity alone. While rare in practice, they remain an open challenge. Their avoidance may require additional randomness or second-order mechanisms, see Remark 6.

# D MINLP Problem Setup and Parameter Sampling

## D.1 Integer Quadratic Problems

The integer quadratic problems (IQPs) used in our experiments are formulated as follows:

$$\min_{\mathbf{x} \in \mathbb{Z}^n} \ \frac{1}{2}\mathbf{x}^\intercal \mathbf{Q}\mathbf{x} + \mathbf{p}^\intercal \mathbf{x} \quad \text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

where the coefficients $\mathbf{Q} \in \mathbb{R}^{n \times n}$, $\mathbf{p} \in \mathbb{R}^n$, and $\mathbf{A} \in \mathbb{R}^{m \times n}$ were fixed, while $\mathbf{b} \in \mathbb{R}^m$ were treated as treated as a parametric input, varying between instances to represent different optimization scenarios.

To ensure convexity, $\mathbf{Q} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with entries sampled uniformly from $[0, 0.01]$. The linear coefficient vector $\mathbf{p} \in \mathbb{R}^n$ has entries drawn from a uniform distribution over $[0, 0.1]$, while the constraint matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is generated from a normal distribution with a mean of $0$ and a standard deviation of $0.1$. The parameter $\mathbf{b} \in \mathbb{R}^m$, representing the right-hand side of the inequality constraints, is sampled uniformly from $[-1, 1]$. These variations in $\mathbf{b}$ across instances ensure the parametric nature of the problem.

Compared to the original setup in Donti et al. [18], which focused on continuous optimization, we introduced integer constraints on all decision variables. Additionally, equality constraints were removed to ensure that generated instances remain feasible, as such constraints could lead to infeasibilities in the discrete space. These modifications preserve the fundamental structure of the original problems while ensuring compatibility with our proposed framework.

### D.2 Integer Non-convex Problems

The integer non-convex problem (INPs) used in the experiments is derived by modifying the IQPs from Appendix D.1 as follows:

$$\min_{\mathbf{x} \in \mathbb{Z}^n} \ \frac{1}{2}\mathbf{x}^\intercal \mathbf{Q}\mathbf{x} + \mathbf{p}^\intercal \sin(\mathbf{x}) \quad \text{subject to } \mathbf{A}\mathbf{x} \leq \mathbf{b}$$

where the sine function is applied element-wise to the decision variables $\mathbf{x}$. This introduces non-convexity into the problem, making it more challenging compared to the convex case. For the integer non-convex problems, the coefficients $\mathbf{Q}$, $\mathbf{p}$, $\mathbf{A}$, and $\mathbf{b}$ are generated in the same way as in the quadratic formulation. However, an additional parameter $\mathbf{d} \in \mathbb{R}^m$ is introduced, with each element independently sampled from a uniform distribution over $[-0.5, 0.5]$. The parameter $\mathbf{d}$ modifies the constraint matrix $\mathbf{A}$ by adding $\mathbf{d}$ to its first column and subtracting $\mathbf{d}$ from its second column. Alongside $\mathbf{d}$, the right-hand side vector $\mathbf{b}$ remains a dynamic parameter in the problem. The problem scale and experimental setup remain consistent with IQPs.

### D.3 Mixed-integer Ronsenbrock Problems

The mixed-integer Rosenbrock problem (MIRBs) used in this study is defined as:

$$\min_{\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{Z}^n} \|\mathbf{a} - \mathbf{x}\|_2^2 + 50\|\mathbf{y} - \mathbf{x}^2\|_2^2$$

$$\text{subject to } \ \|\mathbf{x}\|_2^2 \leq nb, \ \mathbf{1}^\intercal \mathbf{y} \geq \frac{nb}{2}, \ \mathbf{p}^\intercal \mathbf{x} \leq 0, \ \mathbf{Q}^\intercal \mathbf{y} \leq 0,$$

where $\mathbf{x} \in \mathbb{R}^n$ are continuous decision variables and $\mathbf{y} \in \mathbb{Z}^n$ are integer decision variables. The vectors $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{Q} \in \mathbb{R}^n$ are fixed for each instance, while the parameters $b$ and $\mathbf{a}$ vary. In detail, the vectors $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{Q} \in \mathbb{R}^n$ are generated from a standard normal distribution. The parameter $b$ is uniformly distributed over $[1, 8]$ for each instance, and the parameter $\mathbf{a} \in \mathbb{R}^n$ represents a vector where elements drawn independently from a uniform distribution over $[0.5, 4.5]$. The parameters $b$ and $\mathbf{a}$ influence the shape of the feasible region and the landscape of the objective function, serving as input features to the neural network.

## E Computational Setup

### E.1 Computational Environment.

All experiments were conducted on a workstation equipped with two Intel Silver 4216 Cascade Lake CPUs (2.1GHz), 64GB RAM, and four NVIDIA V100 GPUs. Our models were implemented in Python 3.10.13 using PyTorch 2.5.0+cu122 [81] for deep learning models, and NeuroMANCER 1.5.2 [82] for modeling parametric constrained optimization problems.

To benchmark against traditional optimization solvers, we used Gurobi 11.0.1 [83] for convex problems such as IQPs and MILPs and SCIP 9.0.0 [84] was employed, coupled with Ipopt 3.14.14 [85] for those more general mixed-integer non-convex problems such as INP and MIRB.

### E.2 Neural Network Architecture and Training.

The solution mapping $\pi_{\Theta_1}$ used across all learning-based methods—RC, LT, and ablation studies—consists of five fully connected layers with ReLU activations. The rounding correction network $\varphi_{\Theta_2}$ for RC and LT is composed of four fully connected layers, also with ReLU activations, and incorporates Batch Normalization and Dropout with a rate of $0.2$ to prevent overfitting. For feasibility projection, we set a maximum iteration limit of $1000$ for computational efficiency.

To accommodate varying problem complexities, hidden layer widths were scaled proportionally to the problem size:

- For the IQPs and INPs, the hidden layer width used in the learning-based methods was scaled accordingly, increasing from $64$, $128$ up to $2048$ for the corresponding problem sizes. Smaller problems, such as $20 \times 20$, used smaller hidden layers $64$, while larger problems, such as $1000 \times 1000$, used hidden layers with widths up to $2048$ to accommodate the complexity.
- For the MIRBs, the hidden layer width was scaled based on the number of variables: a width of $4$ was used for problems with $2$ variables, $16$ for problems with $20$ variables, and up to $1024$ for problems with $10,000$ variables.

All networks were trained using the AdamW optimizer with learning rate $10^{-3}$, batch size $64$ and $200$ epochs, with early stopping based on validation performance. The constraint penalty weight $\lambda$ was set to $100$ for all benchmark problems. For the feasibility projection step, we used a fixed step size of $\eta = 0.01$ and a maximum iteration limit of $1000$ to ensure both convergence and computational efficiency.

## F Additional Experimental Analyses

### F.1 Constraints Violations

This section examines constraint violations across three benchmark problems, focusing on both their frequency and magnitude. To aid understanding, the results are presented using heatmaps, where each heatmap (Figure 7, Figure 8, and Figure 9) displays rows as 100 test instances and columns as individual constraints, reflecting the solutions before the application of the projection step.

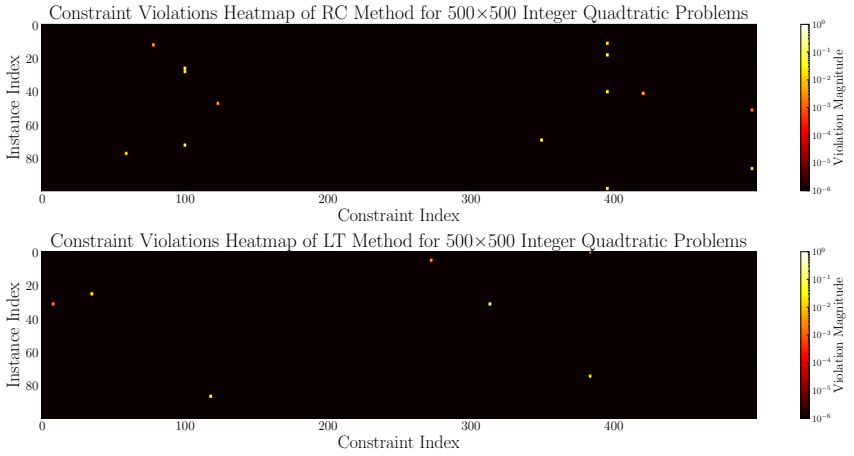


Figure 7: Illustration of Constraint Violation Heatmap of RC method (Top) and LT method (bottom) for 500×500 mixed-integer QP on 100 test instances: Each row represents an instance in the test set, while each column corresponds to a specific constraint. Color intensity indicates the magnitude of constraint violation, with lighter shades representing larger violations.

The heatmaps for the IQPs (Figure 7) and the INPs (Figure 8) reveal a sparse distribution of constraint violations, primarily concentrated in a few constraints across instances. This indicates that the

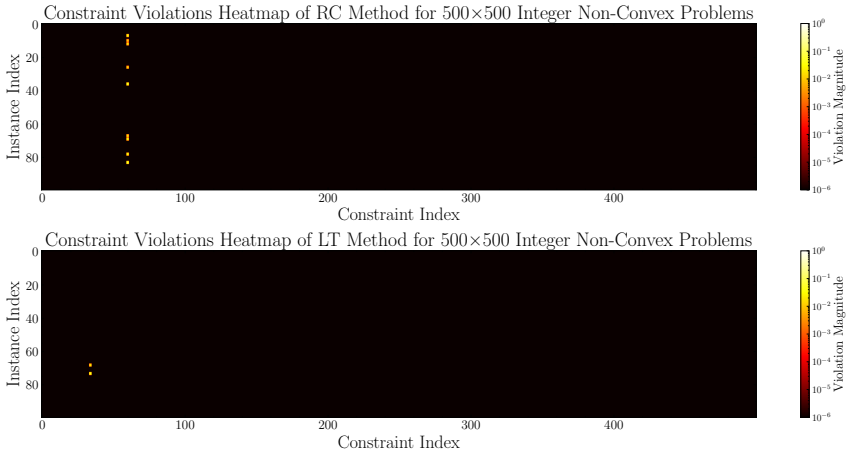


Figure 8: Illustration of Constraint Violation Heatmap of RC method (Top) and LT method (bottom) for 500×500 INPs on 100 test instances: Each row represents an instance in the test set, while each column corresponds to a specific constraint. Color intensity indicates the magnitude of constraint violation, with lighter shades representing larger violations.

majority of constraints are consistently satisfied, with violations being limited to isolated instances. Overall, the magnitude and frequency of these violations are nearly negligible. Thus, our feasibility projection effectively corrects them. In contrast, an unexpected observation is that when these near-feasible solutions are provided as warm-starting points to Gurobi or SCIP, the solvers consistently fail to recover a feasible solution, despite the minimal constraint violations.

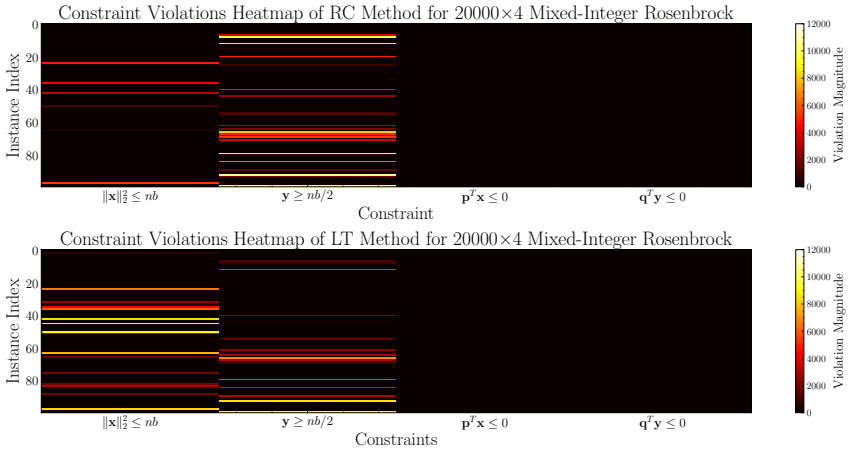


Figure 9: Illustration of Constraint Violation Heatmap of RC method (Top) and LT method (bottom) for 20000×4 MIRBs on 100 test instances: Each row represents an instance in the test set, while each column corresponds to a specific constraint. Color intensity indicates the magnitude of constraint violation, with lighter shades representing larger violations.

Figure 9 reveals a significantly denser distribution of constraint violations in the high-dimensional MIRBs, highlighting the increased complexity of this large-scale MINLP. Given the scale and difficulty of this problem, even state-of-the-art solvers fail to produce feasible solutions, and both RC and LT exhibit a high proportion of constraint violations, with magnitudes reaching the order of $10^3$. Remarkably, despite these substantial violations, Table 5 shows that the feasibility projection in RC-P and LT-P successfully restores feasibility across all 100 test instances, albeit at the cost of increased objective values.

### F.2 Training Time

This section presents the training times for the RC and LT methods across various problem sizes. All training runs were conducted using datasets of 9,000 instances for each problem, with 1,000 instances reserved for validation per epoch. While the training process was set for 200 epochs, an early stopping strategy was applied, allowing the training to terminate earlier when performance plateaued. Note that RC-P and LT-P are not included in the training time comparison, as feasibility projection is applied only as a post-processing step and does not impact the training duration.

Table 6: Training Times (in seconds) for RC and LT methods across different problem sizes for the IQPs Each method was set to train for 200 epochs, with early stopping applied.

| Method | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|
| RC | 153.98 | 237.11 | 141.15 | 149.43 | 606.23 | 727.32 |
| LT | 154.33 | 158.61 | 128.86 | 139.17 | 458.62 | 462.41 |

Table 7: Training Times (in seconds) for RC and LT methods across different problem sizes for the INPs. Each method was set to train for 200 epochs, with early stopping applied.

| Method | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|
| RC | 173.02 | 138.53 | 136.01 | 104.05 | 116.01 | 156.85 |
| LT | 104.35 | 88.41 | 111.38 | 89.24 | 230.52 | 195.67 |

Table 8: Training Times (in seconds) for RC and LT methods across different problem sizes for the MIRBs. Each method was set to train for 200 epochs, with early stopping applied.

| Method | 2×4 | 20×4 | 200×4 | 2000×4 | 20000×4 |
|---|---|---|---|---|---|
| RC | 230.68 | 112.35 | 75.49 | 106.76 | 5227.05 |
| LT | 126.60 | 125.11 | 86.43 | 84.61 | 6508.41 |

Table 6, Table 7, and Table 8 summarize the training times (in seconds) required by each method for problems of different scales. These results highlight the computational efficiency of our methods during training, with training times for most problem instances remaining within a few hundred seconds. Even for large-scale problems, such as the 20000×4 MIRBs, RC and LT required only a few hours of training—much shorter than the time exact solvers take to find just a single feasible solution for an instance.

## G Additional Experiments

### G.1 Ablation Study

To evaluate the contribution of the correction layers $\varphi_{\Theta_2}$, we perform an ablation study using two baseline methods:

- **Rounding after Learning (RL)**: This baseline trains only the first neural network $\pi_{\Theta_1}$, which predicts relaxed solutions. Rounding to the nearest integer is applied post-training, meaning that the rounding step does not participate in the training process. This isolates the effect of excluding the corrective adjustments provided by $\varphi_{\Theta_2}$. This baseline highlights the limitations of naively rounding relaxed predictions. Such rounding often leads to significant deviations in the objectives and severe violations in the constraints, emphasizing the importance of end-to-end learning where updates are guided by the ultimate loss function.
- **Rounding with STE (RS):** In this baseline, continuous values predicted by $\pi_{\Theta_1}$ are rounded during training using the STE, as shown in Algorithm 3. This approach allows gradients to flow through the rounding operator, facilitating optimization of the loss function even with integer constraints.This allows gradients to pass through the rounding operator, enabling optimization of the loss function in the presence of integer constraints. However, the

rounding mechanism relies solely on fixed nearest-integer corrections, which are determined independently of the problem parameters or the relaxed predictions. Consequently, it lacks the refinement provided by learnable correction layers, limiting its ability to adjust the rounding to improve objective values and feasibility.

**Algorithm 3** Rounding with STE: Forward Pass.

1: Training instance $\boldsymbol{\xi}^i$ and neural networks $\pi_{\Theta_1}(\cdot)$
2: Predict a continuously relaxed solution $\bar{\mathbf{x}}^i \leftarrow \pi_{\Theta_1}(\boldsymbol{\xi}^i)$
3: Round integer variables down: $\hat{\mathbf{x}}^i_z \leftarrow \lfloor \bar{\mathbf{x}}^i_z \rfloor$
4: Compute fractions $\mathbf{v}^i \leftarrow \bar{\mathbf{x}}^i_z - \hat{\mathbf{x}}^i_z$
5: Compute rounding directions $\mathbf{b}^i \leftarrow \mathbb{I}\big(\mathbf{v}^i > 0.5\big)$
6: Update integer variables: $\hat{\mathbf{x}}^i_z \leftarrow \hat{\mathbf{x}}^i_z + \mathbf{b}^i$
7: **Output:** $\hat{\mathbf{x}}^i$

**Results and Insights.** The results of the ablation experiments, summarized in Table 9, Table 10 and Table 11, demonstrate the importance of the correction layers $\varphi_{\Theta_2}$ in improving both solution quality and feasibility. The experimental setup and model parameters used are consistent with those in the main text, ensuring the results are directly comparable. RL shows a significant drop in feasibility rates, highlighting the importance of incorporating differentiable rounding during training to guide outputs. Similarly, while RS benefits from differentiability via STE, the lack of learnable layers for rounding limits its performance compared to RC and LT.

Table 9: Ablation Study for IQPs. See the caption of Table 3 for details.

| Method | Metric | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|---|
| RL | Obj Mean | −4.726 | −14.52 | −17.22 | −37.14 | −89.81 | −176.6 |
| | Obj Median | −4.716 | −14.52 | −17.27 | −37.15 | −89.81 | −176.6 |
| | Feasible | 64% | 42% | 23% | 10% | 0% | 0% |
| | Time (Sec) | 0.0004 | 0.0004 | 0.0005 | 0.0005 | 0.0005 | 0.0011 |
| RS | Obj Mean | −3.929 | −11.93 | −10.58 | −24.72 | −54.93 | −110.7 |
| | Obj Median | −3.963 | −11.96 | −10.58 | −24.72 | −54.93 | −110.6 |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time (Sec) | 0.0010 | 0.0011 | 0.0013 | 0.0012 | 0.0016 | 0.0031 |

Table 10: Ablation Study for INPs. See the caption of Table 4 for details.

| Method | Metric | 20×20 | 50×50 | 100×100 | 200×200 | 500×500 | 1000×1000 |
|---|---|---|---|---|---|---|---|
| RL | Obj Mean | −0.138 | −0.629 | −1.581 | −4.196 | −11.531 | −23.64 |
| | Obj Median | −0.148 | −0.655 | −1.554 | −4.196 | −11.531 | −23.64 |
| | Feasible | 87% | 51% | 15% | 0% | 0% | 0% |
| | Time (Sec) | 0.0005 | 0.0005 | 0.0006 | 0.0006 | 0.0006 | 0.0013 |
| RS | Obj Mean | 0.292 | 1.734 | 2.849 | 4.921 | 9.511 | 25.36 |
| | Obj Median | 0.284 | 1.736 | 2.841 | 4.907 | 9.511 | 25.36 |
| | Feasible | 100% | 100% | 100% | 100% | 100% | 100% |
| | Time (Sec) | 0.0012 | 0.0011 | 0.0012 | 0.0013 | 0.0018 | 0.0031 |

Table 11: Ablation Study for MIRBs. See the caption of Table 5 for details.

| Method | Metric | 2×4 | 20×4 | 200×4 | 2000×4 | 20000×4 |
|---|---|---|---|---|---|---|
| RL | Obj Mean | 58.34 | 63.70 | 605.9 | 6222 | 68364 |
| | Obj Median | 58.00 | 61.95 | 609.0 | 5950 | 69087 |
| | Feasible | 14% | 64% | 56% | 72% | 69% |
| | Time (Sec) | 0.0006 | 0.0005 | 0.0005 | 0.0008 | 0.0014 |
| RS | Obj Mean | 25.095 | 69.36 | 684.7 | 6852 | 72910 |
| | Obj Median | 25.353 | 68.58 | 663.1 | 6509 | 68904 |
| | Feasible | 100% | 97% | 100% | 99% | 61% |
| | Time (Sec) | 0.0010 | 0.0010 | 0.0012 | 0.0019 | 0.0103 |

### G.2 Experiments with Varying Penalty Weights

To complement the findings in the main text, we extend our analysis to the 1000×1000 IQPs and the 2000×4 MIRBs. Figure 10 shows that these problems exhibit similar trends: higher penalty weights improve feasibility at the cost of increased objective values, while feasibility projection effectively corrects solutions obtained with smaller penalty weights, preserving their lower objective values.

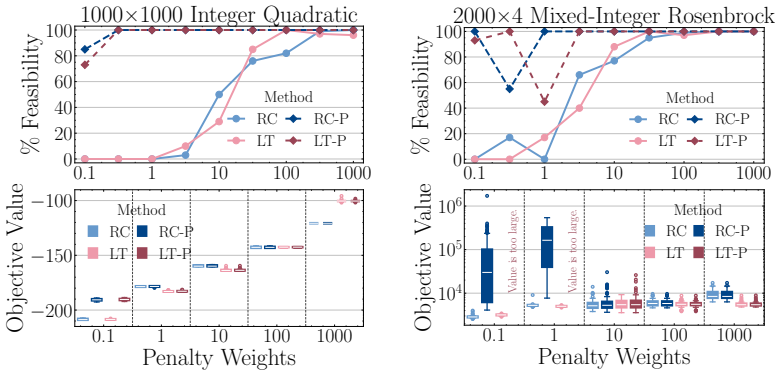


Figure 10: Illustration of the Proportion of feasible solutions (Top) and objective values (Bottom) for smaller-scale problems. For these instances, EX finds feasible solutions within 1000 seconds (leftmost values on the bottom), serving as a benchmark. Our approach, with properly tuned penalty weights, achieves comparable or even better objective values.

Additionally, we evaluate smaller-scale instances against exact solvers where exact solvers (EX) can find feasible solutions within the 1000-second time limit for all test instances. Figure 11 presents results for these instances, illustrating the proportion of feasible solutions and objective values across different penalty weights. These results further confirm that the appropriate choice of penalty weight, when combined with feasibility projection, can yield high-quality, feasible solutions. Thus, our method achieves comparable or even superior objective values to EX while solving instances orders of magnitude faster.

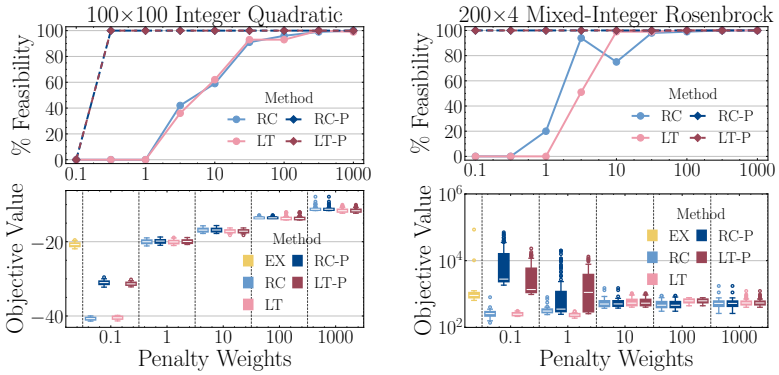


Figure 11: Illustration of the Proportion of feasible solutions (Top) and objective values (Bottom) for smaller problems. For these instances, EX can find feasible solutions within 1000 seconds (leftmost values on the bottom), serving as a benchmark. Our approach, with properly tuned penalty weights, achieves comparable or even better objective values.

### G.3 Experiments with Varying Training Sample Size

To evaluate the effect of training sample size, we trained the model on datasets containing 800, 8,000, and 80,000 instances. Training epochs were adjusted to 2,000, 200, and 20 (with early stopping enabled) to ensure a comparable number of optimization iterations across experiments. All other hyperparameters were kept consistent to isolate the effect of sample size.

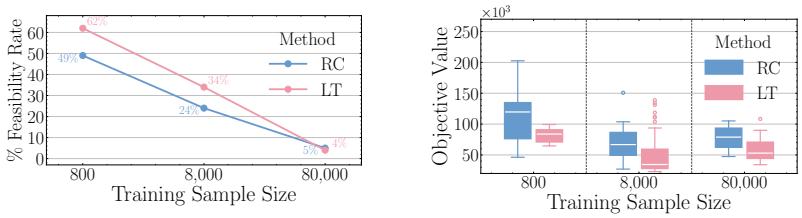


Figure 12: Illustration of the objective value (Left) and proportion of infeasible solutions (Right) of 20000×4 MIRBs on the test set. As the training sample size increases, the fraction of infeasible solutions decreases while the objective value generally deteriorates, as expected.

The result demonstrates that increasing the sample size yields significant improvements in both objective values and feasibility. For instance, training with 80,000 samples reduced the infeasibility rate to $5\%$ on the test set, compared to much higher rates with smaller datasets. This emphasizes the critical role of sufficient sample size and demonstrates the scalability advantage of our self-supervised framework.

### G.4 Experiments on Binary Linear Programs

**Dataset.** The 'Obj Series 1' dataset from the MIP Workshop 2023 Computational Competition [69] is conducted To evaluate the performance of our methods on MILPs. This dataset comprises 50 related MILP instances derived from a shared mathematical formulation. The instances differ in 120 of the 360 objective function coefficients, while all other components, including the constraints, remain consistent. Each instance includes 360 binary variables and 55 constraints, offering a structured benchmark for optimization methods.

**Model Configuration.** The neural network architecture and hyperparameters were consistent with those used in the main experiments. Specifically for the MILP problem in this study, the input dimension of the neural network was set to 120, corresponding to the number of varying objective function coefficients, and the output dimension was set to 360, representing the decision variables. The hidden layer consisted of 256 neurons.

Table 12: Comparison of Optimization Methods on the MILP. See the caption of Table 3 for details.

| Method | Obj Mean | Obj Median | Feasible | Time (Sec) |
|---|---|---|---|---|
| RC | 9745.90 | 9763.00 | 100% | 0.04 |
| LT | 14149.00 | 14149.00 | 100% | 0.04 |
| EX | 8756.80 | 8747.00 | 100% | 28.91 |
| N1 | 11901.10 | 11933.00 | 100% | 0.01 |

**Results.** Table 12 summarizes the performance of various optimization methods on the MILP benchmark: Both learning-based methods (RC and LT) demonstrate the ability to generate high-quality feasible solutions efficiently, with RC even surpassing the heuristic-based method N1 in terms of objective value. However, N1 is the fastest method overall, showcasing the robustness and efficiency of the heuristic in the MILP solver. EX achieved the best objective values but required significantly more computation time. Notably, the training time for the learning-based models is approximately 120 seconds, making them well-suited for applications requiring repeated problem-solving.