LEARNING TO CONTROL FEASIBILITY–OBJECTIVE TRADE-OFF FOR INTEGER LINEAR PROGRAMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Solution predictors for integer linear programs (ILP) struggle to balance feasibility against objective quality, and no existing method learns to control this trade-off at inference. We introduce LibrAx, an unsupervised solution predictor that exposes this trade-off as a spectrum controllable at inference: a single forward pass yields the spectrum. Unlike soft-constrained losses, its semi-hard constrained loss keeps the penalty bounded while retaining an exactness guarantee. On the Very Hard tier of Distributional MIPLIB, LibrAx attains a $2.7\times$ lower primal gap than existing solution predictors on average, with 100% feasibility, producing solutions within 1.2 seconds. ILP solvers need 185–$453\times$ that time to match LibrAx's solution quality on average, widening to 347–$603\times$ on an unseen, harder tier.

1 INTRODUCTION

Combinatorial optimization (CO) underlies many real-world problems such as pickup-and-delivery routing (Zong et al., 2022), chip design (Mirhoseini et al., 2021), and job shop scheduling (Kotary et al., 2022). CO aims to optimize an objective function subject to constraints (Mazyavkina et al., 2021); when the objective and constraints are linear and the variables are required to be integers, the problem is an integer linear program (ILP) (Bertsimas & Tsitsiklis, 1997). ILP is NP-hard in general (Nemhauser & Wolsey, 1988): the integrality requirement makes the feasible set discrete, and the search space grows combinatorially with problem size (Floudas, 1995).

In practice, instances from the same application share recurring structure, which machine learning (ML) can learn once and exploit repeatedly (Gasse et al., 2022). One line of ML work for ILP trains models that assist exact solvers, e.g., for cut selection (Tang et al., 2020; Paulus et al., 2022), branching (Khalil et al., 2016; Kuang et al., 2024), or node selection (He et al., 2014; Labassi et al., 2022). Another line trains a *solution predictor* that directly outputs a candidate solution (Nair et al., 2020; Han et al., 2023; Huang et al., 2024a; Zeng et al., 2024; Liu et al., 2025a; Geng et al., 2025). Since a predicted solution can warm-start an exact solver (Nair et al., 2020), the two lines are complementary. When numerous instances must be solved repeatedly under short time limits (Chen et al., 2024), obtaining high-quality solutions fast matters more than proving optimality (Cantürk et al., 2024). We therefore focus on solution prediction, where speed and quality are decisive.

Most existing solution predictors are supervised, trained to imitate precomputed near-optimal target solutions (Nair et al., 2020; Yoon, 2022; Han et al., 2023; Huang et al., 2024a; Cantürk et al., 2024; Zeng et al., 2024). The predictor is trained to minimize prediction error against the targets—to mimic given solutions rather than to solve the problem—so low imitation error need not translate into high solution quality (Geng et al., 2025). Moreover, collecting such targets is itself costly, and on hard instances they cannot be certified optimal under practical time limits (Huang et al., 2024b). Recent work instead trains the predictor in an unsupervised manner, directly optimizing the ILP objective so that the loss aligns with the optimization goal without labels (Geng et al., 2025).

Despite this progress, the existing unsupervised method (Geng et al., 2025) struggles to balance two competing goals: optimizing the objective and satisfying the constraints. Since improving the objective pulls predictions toward the constraint boundary, the challenge lies in retaining feasibility while pursuing it. This method combines the two goals on an equal footing into a *soft-constrained loss*—the objective plus a weighted violation penalty—which has two limitations. First, the feasibility–objective trade-off is fixed during training with no way to steer it at inference, even though the appropriate trade-off varies across instances. Second, feasibility is enforced only by letting the penalty

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

term outweigh the objective term: a small penalty weight yields infeasible predictions, while a large one risks letting the unbounded penalty dominate the loss function—a *penalty-weight dilemma*.

We introduce **LibrAx**, whose name evokes *Libra* weighing feasibility ($\mathbf{Ax} \leq \mathbf{b}$) against the objective for ILP. LibrAx is an unsupervised ILP solution predictor that makes the feasibility–objective trade-off controllable at inference, with a graph neural network (GNN) architecture designed around active-constraint structure. The main contributions of our work are as follows:

- We expose the feasibility–objective trade-off as a spectrum controllable at inference: one forward pass generates the spectrum, and sweeping it absorbs trade-off shifts without retraining. To the best of our knowledge, LibrAx is the first ILP solution predictor to learn such control.
- We propose a *semi-hard constrained loss* that bypasses the penalty-weight dilemma by keeping the penalty scale bounded for any penalty weight, yet retains an exactness guarantee. The shape of its violation-aware gate further makes strictness adjustable independently of the penalty weight.
- We design the *Active-Constraint (AC) layer*, motivated by the premise that high-quality solutions concentrate near intersections of tight constraints. Its message passing can approximate the operator that recovers such intersections, and the layer empirically accelerates training convergence.

Across three domains at two hardness tiers, LibrAx attains a $3.2\times$ lower primal gap and $65.4\times$ faster prediction than existing solution predictors on average, retaining 100% feasibility. Warm-starting Gurobi with LibrAx's predictions reduces its primal integral by $1.5\times$ on average, and by $3.8\times$ when combined with trust-region search, both relative to standalone Gurobi.

2 PRELIMINARIES

2.1 INTEGER LINEAR PROGRAMMING

An integer linear program (ILP) can be formulated as $\min_{\mathbf{x}}\{\mathbf{c}^\top\mathbf{x} \mid \mathbf{Ax} \leq \mathbf{b},\ \mathbf{l} \leq \mathbf{x} \leq \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{x} = (x_1, \ldots, x_n)$ collects the n variables, $\mathbf{c} \in \mathbb{R}^n$ the objective coefficients, $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)^\top \in \mathbb{R}^{m\times n}$ the constraint matrix with m rows $\mathbf{a}_k^\top$, $\mathbf{b} \in \mathbb{R}^m$ the right-hand sides, $[\mathbf{l}, \mathbf{u}]$ the variable bounds, and $\mathbf{x} \in \mathbb{Z}^n$ the integrality requirement. A solution $\mathbf{x}$ is feasible if it satisfies all constraints and integrality, and optimal if it minimizes $\mathbf{c}^\top\mathbf{x}$ among feasible solutions.

The integrality requirement makes ILP NP-hard (Berthold, 2006). A common remedy is the LP relaxation, which drops this requirement, though its solutions need not satisfy integrality. A constraint $\mathbf{a}_k^\top\mathbf{x} \leq b_k$ is *active* at $\mathbf{x}$ when it holds with equality. If the feasible region is bounded and nonempty, an optimal LP solution exists, and at least one lies at a vertex that is an intersection of active constraints (Bertsimas & Tsitsiklis, 1997). While an ILP optimum need not lie at a vertex, LP-guided primal heuristics obtain integer solutions by searching around LP relaxation solutions (Berthold, 2006); LibrAx builds the same premise into its GNN architecture.

2.2 UNSUPERVISED LOSS FUNCTIONS FOR ILP

Unsupervised approaches for ILP train a solution predictor without labels, minimizing a loss over its predicted solution $\hat{\mathbf{x}} \in \mathbb{R}^n$. The loss combines two goals: an *objective term* $\mathcal{J}(\hat{\mathbf{x}})$ reflecting the ILP objective $\mathbf{c}^\top\hat{\mathbf{x}}$, and a *penalty term* $\mathcal{P}(\hat{\mathbf{x}};\gamma) \geq 0$ that vanishes exactly when $\hat{\mathbf{x}}$ satisfies the constraints, with penalty weight $\gamma > 0$. How a loss weighs these two terms defines its behavior, which we express with a gate $\omega \in [0, 1]$ on the objective term,

$$\mathcal{L} = \omega\,\mathcal{J}(\hat{\mathbf{x}}) + \mathcal{P}(\hat{\mathbf{x}};\gamma). \tag{1}$$

A *soft-constrained* loss keeps the objective always active, $\omega = 1$, as adopted in unsupervised learning for ILP (Geng et al., 2025). In effect, such losses treat feasibility as one goal among many, even though it is a requirement to be ultimately met. They enforce feasibility by letting the penalty term overwhelm the objective term, which requires a sufficiently large γ and lets the penalty scale grow without bound. At the other extreme, a loss can mirror the ILP directly by nullifying the objective whenever a constraint is violated, which we call a *hard-constrained* loss, $\omega = \mathbb{I}[\,\mathcal{P}(\hat{\mathbf{x}};\gamma) = 0\,]$. Its gate enforces feasibility exactly, but the $0/1$ switch erases the objective gradient on every infeasible point. Our *semi-hard constrained* loss lies between these two extremes, with a continuous gate to retain a usable objective gradient while pursuing feasibility at a bounded penalty scale.

3 METHODOLOGY

Figure 1 presents an overview of LibrAx. Given an ILP instance as a bipartite graph, LibrAx computes variable embeddings via Active-Constraint (AC) layers. These embeddings are then conditioned on H strictness factors that control how strictly constraints are enforced. Each trade-off head $h \in \{1, \dots, H\}$ judges its prediction $\hat{\mathbf{x}}^{(h)}$ at a different strictness level, set by the shape of its gate. To facilitate training, we augment the objective with a compressed form and adopt a feasibility-centric curriculum that gradually introduces the objective and integrality terms before intensifying the feasibility requirement. At inference, LibrAx sweeps the trade-off spectrum in a single forward pass and selects the best solution among the resulting candidates. We defer all proofs to Appendix A.

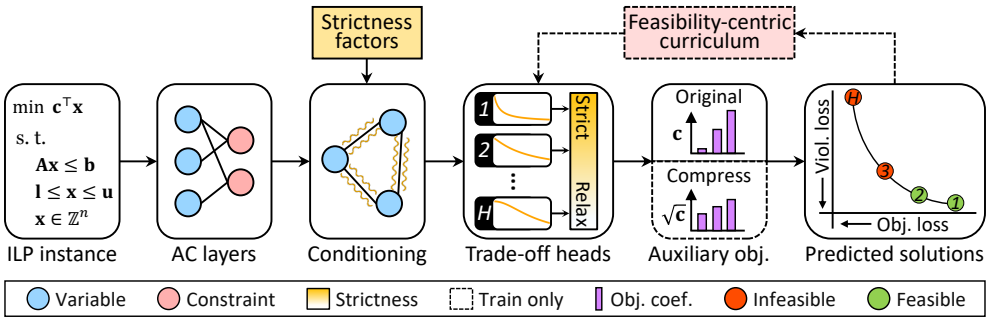

Figure 1: An overview of LibrAx.

3.1 ACTIVE-CONSTRAINT GNN ARCHITECTURE

3.1.1 ACTIVE-CONSTRAINT LAYER

Motivated by the premise from Section 2.1, we design a neural network layer whose message passing can approximate the recovery of a variable from its active constraints, which we call the Active-Constraint (AC) layer. For context, Appendix F.2 reviews existing GNN architectures for ILP solution predictors. For a given point $\mathbf{x}$, let $\mathcal{I}(\mathbf{x}) \subseteq \{1, \dots, m\}$ be the index set of active constraints at $\mathbf{x}$, i.e., those satisfied with equality, $\mathbf{a}_k^\top \mathbf{x} = b_k$ for $k \in \mathcal{I}(\mathbf{x})$. We say constraint k is incident to variable x_i if $a_{ki} \neq 0$, and define $S_i = \{\, k \in \mathcal{I}(\mathbf{x}) : a_{ki} \neq 0 \,\}$.

Lemma 1. *Fix a point* $\mathbf{x}$ *and a variable* x_i *with* $S_i \neq \emptyset$. *Then any weights* $\{s_{ki}\}_{k=1}^m$ *satisfying*

$$s_{ki} \in [0,1], \qquad \sum_{k=1}^{m} s_{ki} = 1, \qquad s_{ki} = 0 \ \textit{for}\ k \notin S_i, \tag{2}$$

recover x_i *from the remaining variables:*

$$x_i \;=\; \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j \Big). \tag{3}$$

We lift Eq. (3) to node embeddings $\{\vec{x}_i\}$. The Active-Constraint layer maps $\{\vec{x}_i\}$ to $\{\vec{x}'_i\}$ by

$$\vec{b}'_k = \varphi(\vec{b}_k) - \sum_{j=1}^{n} a_{kj}\, \psi(\vec{x}_j), \qquad \vec{x}'_i = \phi\left(\vec{x}_i + \sum_{k:\, a_{ki} \neq 0} \frac{s_{ki}}{a_{ki}}\, \vec{b}'_k \right), \quad \forall i, \tag{4}$$

where $\vec{b}_k \in \mathbb{R}^d$ is an embedding for the k-th right-hand side, $\varphi, \psi, \phi : \mathbb{R}^d \to \mathbb{R}^d$ are learnable functions, and the weights $\{s_{ki}\}$ satisfy $s_{ki} \in [0,1]$ and $\sum_{k:a_{ki}\neq 0} s_{ki} = 1$; in practice, we compute them with a GAT-style attention over incident constraints (Veličković et al., 2018) (see Appendix C.2).

Eq. (4) follows the standard message-passing form (Gilmer et al., 2017): φ transforms the right-hand-side embedding and ψ transforms variable embeddings to form the message $\vec{b}'_k$; $\sum_k \frac{s_{ki}}{a_{ki}}(\cdot)$ is the weighted aggregation over incident constraints; and ϕ is the update function.

Proposition 1. *There exists an instantiation of the AC layer (Eq.* (4)*) that exactly realizes Eq.* (3).

At $d = 1$ with identity maps, the update reproduces Eq. (3), so the AC layer can approximate the operator of Lemma 1 (Appendix C.2). For general d and learnable functions, the layer can thus build on this structure rather than learning it from scratch.

3.1.2 STRICTNESS CONDITIONING

To let the model balance constraint satisfaction against objective quality, we condition the variable embeddings $\vec{x}_i$ on *strictness factors* $\{\tau_h\}_{h=1}^{H}$, each a scalar $\tau_h \in [0,1]$ specifying how strictly constraints should be enforced. During training the τ_h are learnable parameters, parametrized via a sigmoid to stay in $[0,1]$; at inference they are set to values sampled evenly over $[0,1]$ (see Section 3.3). Each τ_h is projected to an embedding $\vec{p}_h = \mathrm{MLP}_p(\tau_h) \in \mathbb{R}^d$.

Since variables that share a constraint are coupled through it, adjusting one affects the others. To model this coupling, we propagate information along a variable–variable graph connecting variables that co-occur in a constraint. For each variable i, we compute coupling coefficients $e_{ij}^{(h)}$ with its neighbours j and a strictness-specific embedding $\vec{x}_i^{(h)}$:

$$e_{ij}^{(h)} = \tanh\big(\vec{w}_u^\top \vec{x}_i + \vec{w}_v^\top \vec{x}_j + \vec{w}_p^\top \vec{p}_h\big), \qquad \vec{x}_i^{(h)} = \vec{x}_i + \mathrm{MLP}_o\left(\frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} e_{ij}^{(h)}\,\vec{x}_j\right), \quad (5)$$

where $\vec{w}_u, \vec{w}_v, \vec{w}_p$ are learnable vectors, and $\mathcal{N}(i) = \{\, j \neq i : \exists\, k \text{ with } a_{ki} \neq 0 \text{ and } a_{kj} \neq 0 \,\}$. The strictness term $\vec{w}_p^\top \vec{p}_h$ makes the same variable pair yield different coupling coefficients for different strictness factors. A shared decoder maps these embeddings to predicted values $\hat{x}_i^{(h)} = \sigma\big(\mathrm{MLP}_{\mathrm{dec}}(\vec{x}_i^{(h)})\big)$, where σ is an activation function matching the variable domain (Appendix C.2).

3.2 LEARNING THE FEASIBILITY–OBJECTIVE TRADE-OFF

3.2.1 SEMI-HARD CONSTRAINED LOSS

We introduce a semi-hard constrained loss that suppresses the objective and emphasizes feasibility as violations grow. Since the ILP concerns integer solutions, we measure the loss at the rounded prediction. However, rounding has zero gradient almost everywhere. We therefore follow the straight-through estimator (STE) (Bengio et al., 2013; Hubara et al., 2016; van den Oord et al., 2017), treating rounding as the identity in the backward pass: $\tilde{\mathbf{x}} = \hat{\mathbf{x}} + \mathrm{sg}\big[\mathrm{round}(\hat{\mathbf{x}}) - \hat{\mathbf{x}}\big]$, with $\mathrm{sg}[\cdot]$ the stop-gradient operator. Given $\tilde{\mathbf{x}}$, we define its constraint violation and the violation-aware gate ω as

$$\mathcal{L}_{\mathrm{viol}}(\tilde{\mathbf{x}}) = \frac{1}{m}\sum_{k=1}^{m} \mathrm{ReLU}\left(\frac{\mathbf{a}_k^\top \tilde{\mathbf{x}} - b_k}{\|\mathbf{a}_k\|_2}\right), \qquad u = \frac{\mathcal{L}_{\mathrm{viol}}}{\mathcal{L}_{\mathrm{viol}} + 1}, \qquad \omega = \frac{1}{1 + \gamma\, u^\alpha} \in (0,1], \quad (6)$$

where $\mathcal{L}_{\mathrm{viol}} \geq 0$ measures how much $\tilde{\mathbf{x}}$ violates the linear constraints, $\gamma > 0$ is the penalty weight, and the exponent $\alpha \geq 1$ controls how sharply the constraint violation is penalized.

For simplicity, we present the loss design below with the original form only, i.e., the original coefficients $\mathbf{c}$; a compressed form is instantiated identically in Section 3.2.3. We define the semi-hard constrained loss, incorporating the objective and integrality terms, as

$$\mathcal{L}_{\mathrm{obj}}(\tilde{\mathbf{x}}) = \frac{\mathbf{c}^\top \tilde{\mathbf{x}}}{\|\mathbf{c}\|_1}, \quad \mathcal{L}_{\mathrm{int}}(\hat{\mathbf{x}}) = \frac{1}{n}\sum_{i=1}^{n}\big(1 - |\cos(\pi \hat{x}_i)|\big), \quad \mathcal{L} = \mathrm{sg}[\omega]\,(\mathcal{L}_{\mathrm{obj}} + \lambda \mathcal{L}_{\mathrm{int}}) + (1-\omega), \quad (7)$$

where $\mathcal{L}_{\mathrm{int}} \in [0,1]$, weighted by $\lambda > 0$, vanishes at integers and peaks at half-integers. As rounding via the STE resolves integrality, fractional predictions are guided toward the nearest integers rather than penalized as infeasible. Eq. (1) is recovered with $\mathcal{J} = \mathcal{L}_{\mathrm{obj}}$ and $\mathcal{P} = 1 - \omega$, up to the integrality term $\lambda \mathcal{L}_{\mathrm{int}}$ added for the integer domain.

The violation-aware gate ω prioritizes constraint satisfaction over the objective and integrality: as the violation grows, ω shrinks, down-weighting both terms and activating the penalty $1 - \omega$. The weight γ shapes the gate: the gate fully opens as $\gamma \to 0$ and becomes a $0/1$ switch as $\gamma \to \infty$. Proposition 2 certifies that the loss defines a well-posed target aligned with the ILP objective:

Proposition 2. *Consider the ILP form in Section 2.1 with a nonempty feasible set $\mathcal{F} \subseteq \mathbb{Z}^n$ and the variable-bound box $B = [\mathbf{l}, \mathbf{u}] \supseteq \mathcal{F}$ with $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$. Let $\mathbf{x}^\star \in \arg\min_{\mathbf{x}\in\mathcal{F}} \mathbf{c}^\top \mathbf{x}$. Define the offset loss $\mathcal{L}^\beta = \mathrm{sg}[\omega](\mathcal{L}_{\mathrm{obj}} - \beta + \lambda\mathcal{L}_{\mathrm{int}}) + (1-\omega)$, with all terms evaluated as in Eqs. (6) and (7). There exist an offset β and a finite threshold γ_{thr} such that for $\gamma > \gamma_{\mathrm{thr}}$, every minimizer of $\mathcal{L}^\beta$ over $\hat{\mathbf{x}} \in B$ rounds to a feasible solution attaining the optimal objective.*

The offset β is an analysis device that leaves training gradients unchanged (Remark 1), and the guarantee characterizes what the loss minimizes rather than whether training reaches it (Remark 2).

3.2.2 MULTI-HEAD TRADE-OFF SPECTRUM

To expose the feasibility–objective trade-off as a controllable spectrum, we instantiate the H trade-off heads, each associated with a strictness factor τ_h and an exponent α_h. Head h judges the prediction $\hat{\mathbf{x}}^{(h)}$ with the semi-hard loss in Eq. (7) at its own α_h. All heads are trained jointly, and the total loss, which incorporates the compressed objective form and the warmup, appears in Eq. (9).

An effective multi-head design should span from strict to relaxed enforcement: stricter heads penalize even small violations and favor feasibility, while more relaxed heads tolerate larger violation and favor objective quality. Each head's strictness is governed by the exponent α through the slope of the gate ω as the violation approaches zero. Writing $v = \mathcal{L}_{\text{viol}}$ for brevity, we have, for small v,

$$u = \frac{v}{v+1} \approx v, \qquad \omega = \frac{1}{1+\gamma u^{\alpha}} \approx \frac{1}{1+\gamma v^{\alpha}} \approx 1 - \gamma v^{\alpha}, \qquad \frac{d\omega}{dv} \approx -\gamma\, \alpha\, v^{\alpha-1}. \tag{8}$$

As $v \to 0$, the slope $d\omega/dv$ vanishes for $\alpha > 1$, remains finite and nonzero only at $\alpha = 1$, and diverges for $\alpha < 1$. A vanishing slope ($\alpha > 1$) barely penalizes small violations, whereas a diverging slope ($\alpha < 1$) produces an unbounded gradient that may destabilize training. We therefore space the exponents evenly over $[1, \alpha_{\text{ub}}]$. Appendix B compares the gate design against alternatives.

3.2.3 AUXILIARY OBJECTIVE

In practice, variables with large $|c_i|$ dominate the objective gradient, leaving weak signals for combinations of small-coefficient variables. To tackle this, we introduce a *compressed form* of the objective, $\mathbf{c}_{\text{cmp}} = \text{sign}(\mathbf{c}) \odot \sqrt{|\mathbf{c}|}$, which preserves sign and order while compressing the coefficient range. The compressed form serves as an auxiliary training objective, aiding the primary task through shared representations (Liebel & Körner, 2018; Liu et al., 2019). We therefore instantiate a decoder MLP_{cmp}, identical to MLP_{dec} but with separate parameters and trained with the same semi-hard loss under the compressed form, yielding $\hat{\mathbf{x}}^{(h)}_{\text{cmp}}$; unmarked predictions $\hat{\mathbf{x}}^{(h)}$ continue to denote the output from the original form. At inference, the compressed coefficients are no longer used: candidates from both decoders are retained and scored with the original objective (Section 3.3).

3.2.4 FEASIBILITY-CENTRIC CURRICULUM

Optimizing the objective and satisfying constraints compete from the outset, so learning both at once from scratch is unstable. Inspired by curriculum learning (Bengio et al., 2009; Kumar et al., 2010), we therefore organize training around feasibility in two phases: a warmup phase that gradually introduces the objective, and a subsequent phase that progressively tightens the feasibility requirement.

During the first T_{wu} warmup steps, we linearly ramp the objective and integrality terms into the loss, so that early training focuses on feasibility before the full trade-off is introduced, as follows:

$$\begin{aligned} \mathcal{L}(\hat{\mathbf{x}}; \alpha, \mathfrak{c}) &= \text{sg}[\omega]\ (\eta_{\text{o}} \mathcal{L}_{\text{obj}}(\tilde{\mathbf{x}}; \mathfrak{c}) + \eta_{\text{i}} \lambda\, \mathcal{L}_{\text{int}}(\hat{\mathbf{x}})) + (1 - \omega), \\ \mathcal{L}_{\text{total}} &= \frac{1}{2H} \sum_{h=1}^{H} \Big[\mathcal{L}\big(\hat{\mathbf{x}}^{(h)}; \alpha_h, \mathbf{c}\big) + \mathcal{L}\big(\hat{\mathbf{x}}^{(h)}_{\text{cmp}}; \alpha_h, \mathbf{c}_{\text{cmp}}\big) \Big], \end{aligned} \tag{9}$$

where the warmup factors $\eta_{\text{o}} = \min(1,\ 2t/T_{\text{wu}})$ and $\eta_{\text{i}} = \min(1,\ t/T_{\text{wu}})$ ease in the objective and integrality terms over training step t; the objective leads so that early training is not locked into premature integer assignments. $\mathcal{L}_{\text{obj}}(\tilde{\mathbf{x}}; \mathfrak{c}) = \mathfrak{c}^{\top} \tilde{\mathbf{x}} / \|\mathfrak{c}\|_1$ evaluates the objective with coefficients $\mathfrak{c} \in \{\mathbf{c}, \mathbf{c}_{\text{cmp}}\}$, and ω is the gate computed at the rounded prediction $\tilde{\mathbf{x}}$ with exponent α.

Since γ_{thr} in Proposition 2 is instance-dependent, selecting a single suitable γ is nontrivial: a large γ makes early training hard, while a small γ may fall below γ_{thr}. Our *curriculum penalty scheduler* therefore begins with a small γ and increases it as training progresses by monitoring $\mathcal{L}_{\text{total}}$:

$$\gamma \leftarrow \begin{cases} \rho\, \gamma, & \text{if } \mathcal{L}_{\text{total}} < \mathcal{L}_{\text{ref}}, \\ \gamma, & \text{otherwise}, \end{cases} \qquad \mathcal{L}_{\text{ref}} = \mathcal{L}_{\text{best}} + \epsilon_{\text{a}}\, q^2, \tag{10}$$

where $\rho > 1$ is a growth factor, $\mathcal{L}_{\text{best}}$ is the loss recorded at the most recent increase, q counts consecutive steps without improvement, and $\epsilon_{\text{a}} > 0$ keeps the schedule from freezing. Upon each increase, $\mathcal{L}_{\text{best}}$ is re-evaluated with the current predictions at the new γ, and q is reset; Appendix C.3 details the full procedure. Corollary 1 formalizes that γ surpasses the threshold γ_{thr} of Proposition 2 after sufficiently many active steps, at which point the guarantee applies.

Corollary 1. Let γ_{thr} be as in Proposition 2 and $\gamma_0 > 0$ the initial penalty weight. Suppose that predictions remain in B. Then, the update rule of Eq. (10) does not stall while the scheduler is active, and after $N = \max\big(0, \lceil \log_\rho(\gamma_{\text{thr}}/\gamma_0) \rceil + 1\big)$ increases, $\gamma > \gamma_{\text{thr}}$; N is 0 if $\gamma_{\text{thr}} = 0$.

3.3 Inference-time trade-off control

During training, the strictness factors $\{\tau_h\}_{h=1}^{H}$ are learned at H levels. At inference, sweeping $\tau \in [0, 1]$ acts as a control knob to freely adjust the feasibility–objective trade-off. We sample H' strictness values spaced evenly over $[0, 1]$. The two decoders produce $2H'$ predictions $\hat{\mathbf{x}}^{(h')}$ and $\hat{\mathbf{x}}^{(h')}_{\text{cmp}}$, which are rounded to integer candidates $\tilde{\mathbf{x}}^{(h')}$ and $\tilde{\mathbf{x}}^{(h')}_{\text{cmp}}$, yielding a candidate pool $\mathcal{C}$ of size $2H'$. LibrAx then returns the feasible candidate with the smallest $\mathbf{c}^\top \tilde{\mathbf{x}}$ over $\tilde{\mathbf{x}} \in \mathcal{C}$; if none is feasible, it instead returns the one minimizing the semi-hard loss averaged over the training exponents, $\frac{1}{H} \sum_{h=1}^{H} \mathcal{L}(\tilde{\mathbf{x}}; \alpha_h, \mathbf{c})$ with $\eta = 1$.

4 Experiments

We address four core questions: **(Q1)** How effective is LibrAx as a solution predictor in quality, feasibility, and speed? **(Q2)** How well does it generalize to unseen, harder instances? **(Q3)** How effectively does sweeping the strictness factor navigate the trade-off? **(Q4)** How much do predictions accelerate an ILP solver? Further setup details and analyses appear in Appendices D and E.

4.1 Experimental setup

Baselines. We compare against state-of-the-art *ILP solution predictors* that report the feasibility of their predictions: DDIM (Zeng et al., 2024) and DiffILO (Geng et al., 2025). We exclude *partial*-solution predictors (Nair et al., 2020; Han et al., 2023; Huang et al., 2024a; Liu et al., 2025a): they fail to generate feasible solutions on their own across most instances (Zeng et al., 2024; Geng et al., 2025; Lee & Kim, 2026), falling outside our setting. We also include LP-R, which rounds the LP relaxation solution, a learning-free use of the premise in Section 2.1. The commercial solver Gurobi and the open-source solver SCIP further serve as learning-free references: we compare against them via the time they take to match each method's solutions (TTM; defined below). All learned baselines use their default settings, except for resource-driven adjustments (Appendix E.1). At inference, all learned methods predict 50 candidate solutions ($H' = 25$ for LibrAx).

Benchmarks. In the main experiments, we evaluate on NP-hard ILP problems used in both learned baseline works (Zeng et al., 2024; Geng et al., 2025): maximum independent set (MIS) (Bergman et al., 2016), combinatorial auctions (CA) (Leyton-Brown et al., 2000), and set covering (SC) (Balas & Ho, 1980). For each, we draw datasets from Distributional MIPLIB (Huang et al., 2024b), which categorizes distributions by hardness tier. We use the distributions labeled *Very Hard* (VH), in which no instance is solved to optimality within one hour. Additionally, we use the *Very Hard2* (VH2) distributions, the hardest tier for each domain. Each dataset contains 1,000 instances: 800 for training, 100 for validation, and 100 for test. For the VH2 datasets, we use only the test set.

Metrics. Since both quality and speed are decisive, as discussed in Section 1, we evaluate them across four angles: quality via the *primal gap* (PG) (Berthold, 2013; Huang et al., 2024a) and the *feasibility rate* (FR) (Zeng et al., 2024; Lee & Kim, 2026); speed via the *time to produce* (TTP); both jointly via the *time to match* (TTM). PG measures the gap between the method's objective value z and the reference z^*, where z^* is the best value found by standalone Gurobi within 60 minutes: $\text{PG}(z) = \frac{|z - z^*|}{\max(|z|, |z^*|, \epsilon)}$, which is set to 1 when no feasible solution is found (Berthold, 2013) and reported as a percentage. FR is the percentage of test instances for which a feasible solution is found. TTP is the wall-clock time in seconds to produce solutions, from data loading through rounding. TTM is the wall-clock time in seconds for a standalone solver to reach an objective value at least as good as the method's best solution; unmatched runs count as the time limit, and TTM is 0 when the method finds no feasible solution. Aggregate improvements report geometric means over datasets.

4.2 Solution prediction performance

Table 1 (left) reports the results on the VH tier. Averaged across domains, LibrAx attains a PG $4.3\times$, $1.9\times$, and $3.8\times$ lower than LP-R, DDIM, and DiffILO, respectively. It also produces solutions within 1.2 seconds, $8.1\times$, $760.1\times$, and $3.9\times$ faster, respectively. These gains come despite

Table 1: Solution prediction performance on the VH and VH2 tiers. Learned methods are trained on VH only. TTP and TTM are in seconds. TTM is per solver (G: Gurobi, S: SCIP). Best values per hardness tier and domain are in bold, and second-best are underlined; FR is not highlighted.

Domain	Method	Very Hard (VH)					Very Hard2 (VH2)				
		FR↑	PG↓	TTP↓	TTM_G↑	TTM_S↑	FR↑	PG↓	TTP↓	TTM_G↑	TTM_S↑
MIS	LP-R	100	99.3	0.63	1.0	1.0	100	99.3	1.96	1.0	1.0
	DDIM	32	74.9	411.80	0.3	0.3	0	100.0	1430.62	0.0	0.0
	DiffILO	0	100.0	0.78	0.0	0.0	0	100.0	2.21	0.0	0.0
	LibrAx	100	**6.5**	**0.12**	**648.3**	**793.3**	100	**5.9**	**0.25**	**2486.5**	**2841.8**
CA	LP-R	100	48.1	5.67	1.0	1.0	100	47.4	35.72	1.0	1.0
	DDIM	100	27.0	94.48	2.1	1.0	100	32.5	392.05	1.0	1.0
	DiffILO	0	100.0	0.23	0.0	0.0	0	100.0	0.44	0.0	0.0
	LibrAx	100	**20.0**	**0.06**	**4.8**	**151.1**	100	**20.3**	**0.10**	**33.9**	**317.4**
SC	LP-R	0	100.0	1.22	0.0	0.0	0	100.0	**2.06**	0.0	0.0
	DDIM	100	**21.4**	92.05	**17.0**	**6.3**	100	66.8	342.24	**31.1**	15.2
	DiffILO	95	32.8	2.75	15.9	6.0	100	62.6	4.94	**31.1**	**15.3**
	LibrAx	100	45.7	**1.18**	16.5	**6.3**	100	**57.1**	2.52	**31.1**	**15.3**

a lower total training cost, as DDIM and DiffILO require $46\times$ and $11\times$ LibrAx's cost, respectively (Appendix E.1); DDIM's gap stems mostly from collecting labels with a solver, which unsupervised training avoids. Gurobi and SCIP need $185\times$ and $453\times$ the time LibrAx spends to reach the same solution quality. Within LibrAx's candidate pool, the loss-minimizing candidate is feasible on every instance (Appendix E.2). On SC, although the learned baselines attain a better PG, their TTM is comparable to LibrAx's as the solver overtakes all three solutions at nearly the same time.

DiffILO fails to produce any feasible solution on MIS and CA. Its implementation caps the adaptive penalty weight to avoid penalty dominance, yet the cap can leave the weight below the threshold its guarantee requires—the penalty-weight dilemma. On the failing domains, the weight indeed reaches its cap, and removing it on MIS lets the penalty dominate the loss (Appendix E.3). Our semi-hard constrained loss bypasses the dilemma in two ways. First, the gated penalty sharpens with γ without inflating the loss scale, so γ can grow freely, up to 2.4×10^8 on SC (Appendix E.4). Second, heads with $\alpha > 1$ have a vanishing gate slope near zero violation (Eq. (8)), so the penalty dominates their gradients less even at large γ. DDIM incurs the longest TTP, up to 412 seconds on MIS, as its quality depends on many sequential sampling steps (Zeng et al., 2024); LibrAx generates all 50 candidates in one forward pass. Yet, DDIM yields a worse PG than LibrAx on MIS and CA, even on the subset of MIS instances where it predicts feasible solutions (21.7% vs. 6.5% for LibrAx). The learned baselines thus fail to pursue the objective while maintaining feasibility, and LP-R shares LibrAx's premise but ignores the integrality requirement. LibrAx closes both gaps by making the trade-off controllable (Section 4.4) and by learning to predict solutions that stay feasible after rounding.

4.3 Generalization to harder instances

Table 1 (right) reports the results on the VH2 tier, which no learned method has seen during training. LibrAx attains the best PG on all domains while maintaining a 100% FR. DDIM loses feasibility entirely on MIS, and DiffILO still finds no feasible solution on MIS and CA, while its FR on SC rises from 95% to 100%. The baselines' in-distribution advantage on SC does not transfer: the PG ranking flips in LibrAx's favor. The value of LibrAx's predictions also grows with hardness: on VH2, Gurobi and SCIP need $347\times$ and $603\times$ LibrAx's time to match its solution quality on average, up from $185\times$ and $453\times$ on VH. Section 4.4 attributes this robustness to the spectrum.

4.4 Controllability of the trade-off spectrum

Figure 2 shows the sweep over the strictness factor τ. As τ grows, the violation falls nearly monotonically to zero while the objective is sacrificed: the per-instance Spearman correlation between τ and the violation is below -0.99 on average where the violation is nonzero, and that between τ and the objective value exceeds 0.99. Once a prediction becomes feasible, a larger τ maintains feasibility for 100% of such points on CA and SC, and 99.7% on MIS. The first feasible point (the knee) thus marks the loosest strictness that feasibility permits. Its location varies across instances, forming the transition zone. Notably, on CA, the threshold $\tau^* = 0.83$ for 100% FR lies beyond the range of trained τ ($[0.16, 0.77]$), yet the sweep still attains full feasibility by extrapolation (Appendix E.5).

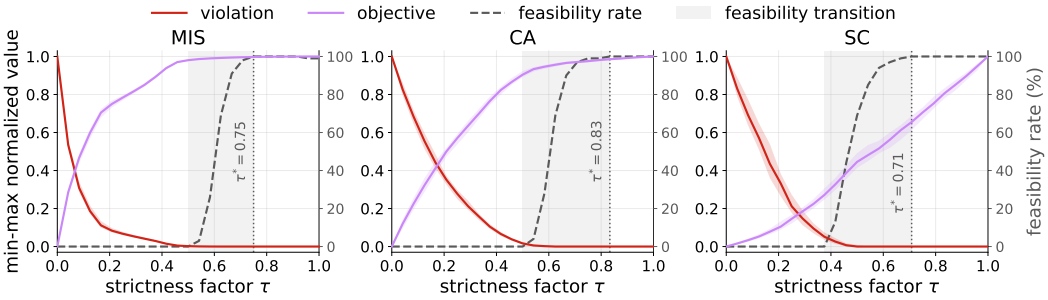

Figure 2: Feasibility–objective spectrum swept by τ on the VH tier. Solid lines (with IQR bands) show min–max normalized values. Dashed lines (right axis) and shaded regions denote feasibility rate and transition zone ($0 \rightarrow 100\%$). Dotted lines mark τ^*, the threshold achieving 100% FR.

Table 2: A fixed operating point selected on the validation set (Fix) versus the sweep (Swp).

Metric	MIS				CA				SC			
	VH		VH2		VH		VH2		VH		VH2	
	Fix	Swp	Fix	Swp	Fix	Swp	Fix	Swp	Fix	Swp	Fix	Swp
FR ↑	100	100	99	100	100	100	100	100	100	100	100	100
PG ↓	7.0	6.5	7.2	5.9	21.4	20.0	21.4	20.3	60.5	45.7	75.1	57.1

The knee, however, is unknown in advance. Taking τ^* estimated on the validation set as a fixed operating point (Fix) underperforms sweeping (Swp) on all domains and tiers (Table 2). Fix also remains vulnerable to distribution shift despite the conservative selection on validation (FR drops to 99% on MIS-VH2), mirroring the baselines that commit to a single operating point (Table 1). Since the required operating point varies by instance and difficulty, it should not be fixed at a single value. LibrAx keeps the operating point adjustable, generating the spectrum in a single forward pass.

4.5 Solver Acceleration

We compare Gurobi with and without LibrAx's prediction on the VH2 tier under a one-hour budget; LibrAx's predictions consistently outperform the baselines' as trust-region centers under a short budget (Appendix E.6). Predictions are supplied either as an initial incumbent alone (Start) or additionally with a trust region (Start+TR) following Geng et al. (2025). We evaluate performance using the primal integral (PI) (Berthold, 2013), calculated as the sum over the one-hour budget of the incumbent's gap to the per-instance best objective value across all runs. All timings for warm-started runs include LibrAx's TTP. Details of the trust region and PI computation appear in Appendix D.

Figure 3 traces the incumbent objective over time, averaged over test instances, with the PI per domain. Warm-starting gives an immediate advantage over standalone Gurobi in all domains, improving PI by $1.5\times$ (Start) and $3.8\times$ (Start+TR) on average. On MIS, where the solver struggles most (Table 1), the prediction and the trust region act synergistically: on average, Start+TR reaches in 124 seconds the objective that standalone Gurobi attains only after 3,407 seconds. On CA and SC, the gains are modest and concentrated in the early phase, and the trust region adds little on SC.

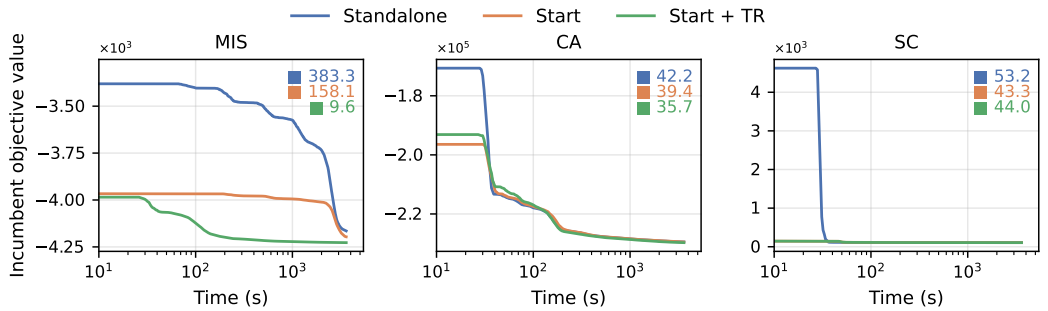

Figure 3: Incumbent objective value over time on the VH2 tier for standalone Gurobi (Standalone) and the warm-started variants (Start and Start+TR). Each panel reports the PI of each method.

4.6 Ablation study

Table 3 reports the ablation results on CA-VH. A single head at the relaxed end (α=3) never finds a feasible solution, one at the strict end (α=1) is always feasible but yields 100% PG, and the middle one (α=2) attains merely 6% FR. Random strictness values (rand τ) leave the model uninformed of which strictness it serves, degrading PG to 28.3%. Hence, both heterogeneous heads and conditioning are needed to realize the spectrum. A soft-constrained loss in place of ours (w/ soft) keeps full feasibility but raises PG to 22.8%, remaining prone to the penalty-weight dilemma (Appendix E.7). Disabling the penalty scheduler (w/o PS) loses full feasibility (FR 95%) and nearly doubles PG (37.4%), underscoring the value of gradually tightening the feasibility requirement. Dropping the integrality term (w/o INT) collapses (PG 100%). The remaining two (w/o AUX and w/o AC) barely change the final PG but converge more slowly (Appendix E.8).

Table 3: Ablation results on CA-VH. SH(α): single head trained at a fixed exponent α; rand τ: τ_h sampled uniformly from $[0, 1)$ during training, with α_h kept; w/ soft: a soft-constrained loss in place of ours; w/o PS: penalty scheduler disabled; w/o INT: no integrality term ($\lambda = 0$); w/o AUX: no auxiliary objective; w/o AC: the bipartite GNN (Gasse et al., 2019) in place of AC layers.

Metric	LibrAx	SH(3)	SH(2)	SH(1)	rand τ	w/ soft	w/o PS	w/o INT	w/o AUX	w/o AC
FR $\uparrow$	100	0	6	100	99	100	95	100	100	100
PG $\downarrow$	20.0	100.0	96.7	100.0	28.3	22.8	37.4	100.0	20.3	20.2

5 Related work

Solution prediction for ILP. Most ILP solution predictors are supervised, imitating near-optimal targets that require costly solver runs to collect (Nair et al., 2020; Han et al., 2023; Huang et al., 2024a; Liu et al., 2025a). Unsupervised training is more recent for ILP: losses that penalize constraint violation were first studied for specific CO problems (Karalias & Loukas, 2020; Gaile et al., 2022; Schuetz et al., 2022; Wang & Li, 2023) and later extended to ILP by Geng et al. (2025). These losses are soft-constrained and thus subject to the penalty-weight dilemma. Our semi-hard constrained loss instead bypasses the dilemma with a bounded penalty scale and a vanishing gate slope for $\alpha > 1$. Appendix F provides a broader overview of ML for ILP and GNN architectures.

Controllable trade-offs. In multi-objective learning, preference vectors steer models across the Pareto front, realizing a chosen trade-off on demand (Lin et al., 2019; Navon et al., 2021; Ruchte & Grabocka, 2021). P-MOCO brings this idea to a specific class of multi-objective CO problems (Lin et al., 2022), but feasibility is not part of what is learned—it is preserved by construction, so violation never enters the trade-off. For ILP solution predictors, by contrast, maintaining feasibility while optimizing the objective remains the central challenge. Extending this equal-footing scalarization directly to ILP—treating the violation as one more goal—yields a soft-constrained loss, which is prone to the penalty-weight dilemma. Diffusion-based predictors such as DDIM also admit inference-time adjustment via a manually tuned guidance weight (Zeng et al., 2024); yet each operating point requires a separate guided run of multiple inference steps. In contrast, LibrAx learns this control and generates the spectrum in a single forward pass. To the best of our knowledge, LibrAx is the first ILP solution predictor to learn control over the feasibility–objective trade-off.

6 Conclusion

Existing solution predictors for ILP often fail to pursue the objective while maintaining feasibility. We introduced LibrAx, an unsupervised solution predictor that closes this gap. Since the feasibility–objective trade-off varies across instances, LibrAx does not fix it but generates the spectrum in a single forward pass, controlled via strictness factors. Its semi-hard constrained loss bypasses the penalty-weight dilemma: the weight can grow without inflating the penalty scale, and the strictness toward feasibility is adjustable independently of the weight. Furthermore, its AC layers accelerate training convergence. On the Very Hard tiers of Distributional MIPLIB, LibrAx attains 100% FR on all six datasets, with quality and speed advantages that grow on the unseen, harder tier. There, sweeping the spectrum absorbs shifts in the trade-off without retraining or manual tuning. Warm-starting with LibrAx reduces PI on all tested domains, with the largest gains where the solver struggles most. Preliminary results on a real-world domain suggest that LibrAx applies to general integer variables (Appendix E.10), and improving its performance in this setting is left for future work.

AI USE STATEMENT

We used generative AI tools for the following tasks: "formulate mathematical claims", "provide critical ingredients for proving mathematical claims", "assist in the writing of proofs", "assist with translation", "provide feedback on research methodology or experiments", "identify relevant literature", and "implement methods".

We have not used generative AI tools for the following tasks: "help develop theoretical models or conceptual frameworks", "design research methodology or experiments", "propose or refine hypotheses", "support qualitative and thematic data analysis", "generate synthetic data sets", "clean and reformat dataset", and "interpret results".

Throughout this work, the authors provided the initial drafts of the manuscript and the proofs; generative AI tools refined them, and the authors performed the final revision. We have reviewed all AI-assisted work: all mathematical claims and proofs were verified by the authors, and AI-assisted code was tested for correctness. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work develops a learning-based method for predicting solutions of integer linear programs. All experiments use benchmark instances from publicly available datasets. We do not identify ethical concerns specific to our contribution beyond considerations generic to optimization technology.

REPRODUCIBILITY STATEMENT

Complete proofs of all claims, with their assumptions stated explicitly, appear in Appendix A. Implementation details—the instance representation, the network architectures, and the full training procedure in pseudocode (Algorithm 1)—are given in Appendix C. The experimental setup—dataset source, training seeds, hyperparameters, solver configurations, and measurement protocols—is described in Appendix D. The baselines are run from their public implementations with default settings, with the resource-driven adjustments listed in Appendix E.1. All datasets are drawn from the publicly available Distributional MIPLIB (see Appendix D).

REFERENCES

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.

Egon Balas and Andrew Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study. *Combinatorial optimization*, pp. 37–60, 1980.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

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

David Bergman, Andre A Cire, Willem-Jan Van Hoeve, and John Hooker. *Decision diagrams for optimization*, volume 1. Springer, 2016.

Timo Berthold. Primal heuristics for mixed integer programs. *Diploma thesis, Technische Universität Berlin*, 2006.

Timo Berthold. Measuring the impact of primal heuristics. *Operations research letters*, 41(6): 611–614, 2013.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena Scientific Belmont, MA, 1997.

Furkan Cantürk, Taha Varol, Reyhan Aydoğan, and Okan Örsan Özener. Scalable primal heuristics using graph neural networks for combinatorial optimization. *Journal of Artificial Intelligence Research*, 80:327–376, 2024.

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.

Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing mixed-integer linear programs by graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023.

Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

Elīza Gaile, Andis Draguns, Emīls Ozoliņš, and Kārlis Freivalds. Unsupervised training for neural tsp solver. In *International Conference on Learning and Intelligent Optimization*, pp. 334–346. Springer, 2022.

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.

Maxime Gasse, Simon Bowly, Quentin Cappart, Jonas Charfreitag, Laurent Charlin, Didier Chételat, Antonia Chmiela, Justin Dumouchelle, Ambros Gleixner, Aleksandr M Kazachkov, et al. The machine learning for combinatorial optimization competition (ml4co): Results and insights. In *NeurIPS 2021 competitions and demonstrations track*, pp. 220–231. PMLR, 2022.

Zijie Geng, Jie Wang, Xijun Li, Fangzhou Zhu, Jianye Hao, Bin Li, and Feng Wu. Differentiable integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pp. 1263–1272. JMLR.org, 2017.

Lacy M Greening, Mathieu Dahan, and Alan L Erera. Lead-time-constrained middle-mile consolidation network design with fixed origins and destinations. *Transportation Research Part B: Methodological*, 174:102782, 2023.

Prateek Gupta, Maxime Gasse, Elias Khalil, Pawan Mudigonda, Andrea Lodi, and Yoshua Bengio. Hybrid models for learning to branch. *Advances in neural information processing systems*, 33: 18087–18097, 2020.

Qingyu Han, Linxin Yang, Qian Chen, Xiang Zhou, Dong Zhang, Akang Wang, Ruoyu Sun, and Xiaodong Luo. A gnn-guided predict-and-search framework for mixed-integer linear programming. In *The Eleventh International Conference on Learning Representations*, 2023.

He He, Hal Daumé, and Jason Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27, 2014.

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*, pp. 13869–13890. PMLR, 2023.

Taoan Huang, Aaron M Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *Forty-first International Conference on Machine Learning*, 2024a.

Weimin Huang, Taoan Huang, Aaron M Ferber, and Bistra Dilkina. Distributional miplib: a multi-domain library for advancing ml-guided milp methods. *arXiv preprint arXiv:2406.06954*, 2024b.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

Nikolaos Karalias and Andreas Loukas. Erdos goes neural: an unsupervised learning framework for combinatorial optimization on graphs. *Advances in Neural Information Processing Systems*, 33: 6659–6672, 2020.

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*, volume 30, 2016.

James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. Fast approximations for job shop scheduling: A lagrangian dual deep learning method. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 7239–7246, 2022.

Yufei Kuang, Jie Wang, Haoyang Liu, Fangzhou Zhu, Xijun Li, Jia Zeng, Jianye Hao, Bin Li, and Feng Wu. Rethinking branching on exact combinatorial optimization solver: The first deep symbolic discovery framework. In *International Conference on Learning Representations*, volume 2024, pp. 50302–50322, 2024.

M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23, 2010.

Abdel Ghani Labassi, Didier Chételat, and Andrea Lodi. Learning to compare nodes in branch and bound with graph neural networks. *Advances in neural information processing systems*, 35: 32000–32010, 2022.

Tae-Hoon Lee and Min-Soo Kim. RL-SPH: Learning to achieve feasible solutions for integer linear programs. In *Forty-third International Conference on Machine Learning*, 2026.

Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the 2nd ACM conference on Electronic commerce*, pp. 66–76, 2000.

Lukas Liebel and Marco Körner. Auxiliary tasks in multi-task learning. *arXiv preprint arXiv:1805.06334*, 2018.

Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qing-Fu Zhang, and Sam Kwong. Pareto multi-task learning. *Advances in neural information processing systems*, 32, 2019.

Xi Lin, Zhiyuan Yang, and Qingfu Zhang. Pareto set learning for neural multi-objective combinatorial optimization. In *International Conference on Learning Representations*, 2022.

Haoyang Liu, Jie Wang, Zijie Geng, Xijun Li, Yuxuan Zong, Fangzhou Zhu, Jianye HAO, and Feng Wu. Apollo-MILP: An alternating prediction-correction neural solving framework for mixed-integer linear programming. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised generalisation with meta auxiliary learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Wenbo Liu, Akang Wang, Wenguo Yang, and Qingjiang Shi. Mixed-integer linear optimization via learning-based two-layer large neighborhood search. In *International Conference on Learning and Intelligent Optimization*, pp. 215–236. Springer, 2025b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *The 5th International Conference on Learning Representations*, 2017.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

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.

Aviv Navon, Aviv Shamsian, Gal Chechik, and Ethan Fetaya. Learning the pareto front with hypernetworks. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=NjF772F4ZZR.

George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley Series in Discrete Mathematics and Optimization. Wiley-Interscience, New York, 1988.

Max B Paulus, Giulia Zarpellon, Andreas Krause, Laurent Charlin, and Chris Maddison. Learning to cut by looking ahead: Cutting plane selection via imitation learning. In *International conference on machine learning*, pp. 17584–17600. PMLR, 2022.

Michael Ruchte and Josif Grabocka. Scalable pareto front approximation for deep multi-objective learning. In *2021 IEEE international conference on data mining (ICDM)*, pp. 1306–1311. IEEE, 2021.

Martin JA Schuetz, J Kyle Brubaker, and Helmut G Katzgraber. Combinatorial optimization with physics-inspired graph neural networks. *Nature Machine Intelligence*, 4(4):367–377, 2022.

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.

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.

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *International conference on machine learning*, pp. 9367–9376. PMLR, 2020.

John A Tomlin. *On scaling linear programming problems*. Springer, 1975.

Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Haoyu Wang and Pan Li. Unsupervised learning for combinatorial optimization needs meta-learning. *arXiv preprint arXiv:2301.03116*, 2023.

Yaoxin Wu, Wen Song, Zhiguang Cao, and Jie Zhang. Learning large neighborhood search policy for integer programming. *Advances in Neural Information Processing Systems*, 34:30075–30087, 2021.

Taehyun Yoon. Confidence threshold neural diving. *NeurIPS 2021 ML4CO competition track*, 2022.

Hao Zeng, Jiaqi Wang, Avirup Das, Junying He, Kunpeng Han, Haoyuan Hu, and Mingfei Sun. Effective generation of feasible solutions for integer programming via guided diffusion. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4107–4118, 2024.

Zefang Zong, Meng Zheng, Yong Li, and Depeng Jin. Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. In *Proceedings of the AAAI conference on artificial intelligence*, volume 36, pp. 9980–9988, 2022.

A PROOFS

Lemma 1. *Fix a point* $\mathbf{x}$ *and a variable* x_i *with* $S_i \neq \emptyset$*. Then any weights* $\{s_{ki}\}_{k=1}^m$ *satisfying*

$$s_{ki} \in [0,1], \qquad \sum_{k=1}^{m} s_{ki} = 1, \qquad s_{ki} = 0 \ \textit{for}\ k \notin S_i, \tag{2}$$

recover x_i *from the remaining variables:*

$$x_i \;=\; \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j \Big). \tag{3}$$

Proof. Let $k \in S_i$, so that $a_{ki} \neq 0$ and $\mathbf{a}_k^\top \mathbf{x} = b_k$. Separating the i-th term,

$$\mathbf{a}_k^\top \mathbf{x} = a_{ki}\, x_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j = b_k \quad \Longrightarrow \quad a_{ki}\, x_i = b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j,$$

and dividing by $a_{ki} \neq 0$ gives

$$x_i \;=\; \frac{1}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j \Big). \tag{11}$$

Let $\{s_{ki}\}$ be any weights satisfying Eq. (2). Since they are supported on S_i, we have $\sum_{k \in S_i} s_{ki} = \sum_{k=1}^m s_{ki} = 1$. Multiplying Eq. (11) by s_{ki} and summing over $k \in S_i$,

$$x_i \;=\; \Big(\sum_{k \in S_i} s_{ki} \Big) x_i \;=\; \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj}\, x_j \Big),$$

which is exactly Eq. (3).

□

Proposition 1. *There exists an instantiation of the AC layer (Eq.* (4)*) that exactly realizes Eq.* (3)*.*

Proof. Take $d = 1$ and $\phi = \varphi = \psi = \mathrm{id}$. Then Eq. (4) reduces to

$$x_i' \;=\; x_i + \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{j=1}^{n} a_{kj}\, x_j \Big).$$

Splitting $\sum_{j=1}^n a_{kj} x_j = a_{ki} x_i + \sum_{\substack{j=1 \\ j \neq i}}^n a_{kj} x_j$ and using $\sum_{k \in S_i} s_{ki} = 1$,

$$x_i' \;=\; x_i + \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj} x_j \Big) - \Big(\sum_{k \in S_i} s_{ki} \Big) x_i \;=\; \sum_{k \in S_i} \frac{s_{ki}}{a_{ki}} \Big(b_k - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{kj} x_j \Big),$$

which is exactly the right-hand side of Eq. (3). □

Proposition 2. *Consider the ILP form in Section 2.1 with a nonempty feasible set* $\mathcal{F} \subseteq \mathbb{Z}^n$ *and the variable-bound box* $B = [\mathbf{l}, \mathbf{u}] \supseteq \mathcal{F}$ *with* $\mathbf{l}, \mathbf{u} \in \mathbb{Z}^n$*. Let* $\mathbf{x}^\star \in \arg\min_{\mathbf{x} \in \mathcal{F}} \mathbf{c}^\top \mathbf{x}$*. Define the offset loss* $\mathcal{L}^\beta = \mathrm{sg}[\omega](\mathcal{L}_{\mathrm{obj}} - \beta + \lambda \mathcal{L}_{\mathrm{int}}) + (1 - \omega)$*, with all terms evaluated as in Eqs.* (6) *and* (7)*. There exist an offset* β *and a finite threshold* γ_{thr} *such that for* $\gamma > \gamma_{\mathrm{thr}}$*, every minimizer of* $\mathcal{L}^\beta$ *over* $\hat{\mathbf{x}} \in B$ *rounds to a feasible solution attaining the optimal objective.*

Proof. Fix an offset $\beta \geq \max_{z \in B} \mathcal{L}_{\text{obj}}(z)$ and let $g(\tilde{\mathbf{x}}) = \beta - \mathcal{L}_{\text{obj}}(\tilde{\mathbf{x}}) \geq 0$, $g^\star = g(\mathbf{x}^\star)$, and $M = \max_{z \in B} g(z)$. Since $\mathbb{Z}^n \cap B$ is finite and an infeasible integer point violates some linear constraint, the minimum violation $\delta = \min_{\tilde{\mathbf{x}} \in (\mathbb{Z}^n \cap B) \setminus \mathcal{F}} \mathcal{L}_{\text{viol}}(\tilde{\mathbf{x}})$ is positive whenever an infeasible integer point exists; set $u_0 = \delta/(\delta + 1)$. We prove the claim with the explicit threshold $\gamma_{\text{thr}} = (M - g^\star)/\big((1 + g^\star)\, u_0^\alpha\big)$.

Within the set of predictions sharing the same rounded value, $\{\hat{\mathbf{x}} : \text{round}(\hat{\mathbf{x}}) = \tilde{\mathbf{x}}\}$, the terms evaluated at the rounded prediction (i.e., $\mathcal{L}_{\text{obj}}$ and ω) are unchanged, while $\lambda \mathcal{L}_{\text{int}} \geq 0$ vanishes at $\hat{\mathbf{x}} = \tilde{\mathbf{x}}$. Hence $\mathcal{L}^\beta(\text{round}(\hat{\mathbf{x}})) \leq \mathcal{L}^\beta(\hat{\mathbf{x}})$ for every $\hat{\mathbf{x}} \in B$, so every minimizer of $\mathcal{L}^\beta$ over B rounds to a minimizer over $\mathbb{Z}^n \cap B$; the analysis thus reduces to minimizing $\mathcal{L}^\beta$ over $\tilde{\mathbf{x}} \in \mathbb{Z}^n \cap B$. Since detaching the gradient does not change the value and $\mathcal{L}_{\text{int}}$ vanishes on integer points, on $\mathbb{Z}^n \cap B$ we use the value form $\mathcal{L}^\beta = \omega(\mathcal{L}_{\text{obj}} - \beta) + (1 - \omega) = 1 - \omega(1 + g)$, with $\omega = 1/(1 + \gamma u^\alpha)$.

Feasible points. If $\tilde{\mathbf{x}}$ is feasible then $\mathcal{L}_{\text{viol}} = 0$ by Eq. (6), so $u = 0$ and $\omega = 1$. Substituting $\omega = 1$ into the value form gives $\mathcal{L}^\beta(\tilde{\mathbf{x}}) = 1 - (1 + g) = -g(\tilde{\mathbf{x}})$. Thus minimizing $\mathcal{L}^\beta$ over feasible points maximizes g. Since $g = \beta - \mathbf{c}^\top \tilde{\mathbf{x}}/\|\mathbf{c}\|_1$ with β and $\|\mathbf{c}\|_1$ constant, this is equivalent to minimizing $\mathbf{c}^\top \tilde{\mathbf{x}}$, i.e., solving the original ILP. Its minimum is attained at $\mathbf{x}^\star$ with loss value $\mathcal{L}^\star = -g^\star$.

Infeasible points. If $(\mathbb{Z}^n \cap B) \setminus \mathcal{F} = \emptyset$, every candidate point is feasible and the claim follows directly from the feasible-point analysis above; we therefore assume that an infeasible integer point exists, so that δ is defined. By definition of δ, every infeasible $\tilde{\mathbf{x}} \in \mathbb{Z}^n \cap B$ has $\mathcal{L}_{\text{viol}}(\tilde{\mathbf{x}}) \geq \delta$, so $u \geq u_0$, and $\omega \leq \omega_0 = 1/(1 + \gamma u_0^\alpha)$. Since $g \leq M$ on B, the value form gives

$$\mathcal{L}^\beta(\tilde{\mathbf{x}}) = 1 - \omega(1 + g) \;\geq\; 1 - \omega_0(1 + M).$$

It remains to show $1 - \omega_0(1 + M) > \mathcal{L}^\star = -g^\star$, i.e., $\omega_0(1 + M) < 1 + g^\star$. Substituting ω_0 and clearing the positive denominator, this is equivalent to

$$1 + M < (1 + g^\star)(1 + \gamma u_0^\alpha) \iff \gamma > \frac{M - g^\star}{(1 + g^\star)\, u_0^\alpha} = \gamma_{\text{thr}},$$

which holds by assumption. Hence $\mathcal{L}^\beta(\tilde{\mathbf{x}}) > -g^\star = \mathcal{L}^\star$ for every infeasible $\tilde{\mathbf{x}}$.

Combining the two cases, any minimizer of $\mathcal{L}^\beta$ over $\tilde{\mathbf{x}} \in \mathbb{Z}^n \cap B$ is feasible, and among feasible points minimizing $\mathcal{L}^\beta$ is equivalent to minimizing $\mathbf{c}^\top \tilde{\mathbf{x}}$; hence every such minimizer attains the optimal objective. By the reduction at the beginning of the proof, every minimizer of $\mathcal{L}^\beta$ over $\hat{\mathbf{x}} \in B$ rounds to a feasible solution attaining the optimal objective.

□

Remark 1. The offset β is an analysis device: it makes the gated objective term $\mathcal{L}_{\text{obj}} - \beta$ nonpositive on B, so no point can lower $\mathcal{L}^\beta$ by inflating the violation to shrink ω. At any fixed γ, it leaves the gradient with respect to the model parameters unchanged, since $\text{sg}[\omega]$ is detached and β is constant:

$$\nabla_{\hat{\mathbf{x}}} \mathcal{L}^\beta = \text{sg}[\omega]\, \nabla_{\hat{\mathbf{x}}} \big(\mathcal{L}_{\text{obj}} + \lambda \mathcal{L}_{\text{int}}\big) + \nabla_{\hat{\mathbf{x}}}(1 - \omega) = \nabla_{\hat{\mathbf{x}}} \mathcal{L}.$$

This equality holds for any β, so training with $\beta = 0$ follows exactly the gradient field of the analyzed loss at a fixed γ; the scheduler of Algorithm 1 monitors the unshifted loss value, as we set $\beta = 0$. The stop-gradient removes the $\mathcal{L}_{\text{obj}} \nabla \omega$ pathway by which gradient descent could otherwise lower the loss through increased violation.

Remark 2. Proposition 2 should be read as certifying that the loss defines a well-posed target: over all predictions in B, the minimizers of the offset loss are exactly the ILP optima. This rules out the failure mode in which an infeasible integer point attains a lower loss than the optimum. In practice, the restriction $\hat{\mathbf{x}} \in B$ holds via sigmoid or clipping. General finite bounds normalize to $\lceil \mathbf{l} \rceil, \lfloor \mathbf{u} \rfloor$ without changing $\mathcal{F}$, so integer bounds are without loss of generality; their integrality ensures $\text{round}(\hat{\mathbf{x}}) \in B$ for every $\hat{\mathbf{x}} \in B$.

Corollary 1. Let γ_{thr} be as in Proposition 2 and $\gamma_0 > 0$ the initial penalty weight. Suppose that predictions remain in B. Then, the update rule of Eq. (10) does not stall while the scheduler is active, and after $N = \max\big(0, \lceil \log_\rho(\gamma_{\text{thr}}/\gamma_0) \rceil + 1\big)$ increases, $\gamma > \gamma_{\text{thr}}$; N is 0 if $\gamma_{\text{thr}} = 0$.

Proof. Since predictions remain in B, $\mathcal{L}_{\text{obj}}$ is bounded. In addition, the penalty $1 - \omega$ is at most one. Thus, $\mathcal{L}_{\text{total}}$ is bounded above by some $C < \infty$.

We first show that γ does not stall while the scheduler is active. Suppose for contradiction that γ stops increasing, and let T be the last step at which an increase occurs. For every subsequent active step, no improvement occurs, so $\mathcal{L}_{\text{best}}$ stays fixed and q increments at each step. By Eq. (10), $\mathcal{L}_{\text{ref}} = \mathcal{L}_{\text{best}} + \epsilon_{\text{a}}\, q^2$ grows without bound as q increases. Hence $\mathcal{L}_{\text{ref}}$ eventually exceeds $C \geq \mathcal{L}_{\text{total}}$, so $\mathcal{L}_{\text{total}} < \mathcal{L}_{\text{ref}}$ triggers an increase of γ—contradicting that T was the last increase.

If $\gamma_{\text{thr}} = 0$, any $\gamma_0 > 0$ already exceeds it, so assume $\gamma_{\text{thr}} > 0$. Each increase multiplies γ by $\rho > 1$, and γ is never decreased, so after N increases $\gamma = \rho^N \gamma_0$. By definition, $N \geq \lceil \log_\rho(\gamma_{\text{thr}}/\gamma_0) \rceil + 1 > \log_\rho(\gamma_{\text{thr}}/\gamma_0)$. Since $\rho > 1$, applying $\rho^{(\cdot)}$ preserves the inequality, so $\rho^N > \gamma_{\text{thr}}/\gamma_0$. Multiplying both sides by $\gamma_0 > 0$ gives $\rho^N \gamma_0 > \gamma_{\text{thr}}$. As γ does not stall while the scheduler is active, these N increases occur provided the scheduler remains active for sufficiently many steps. Since γ is nondecreasing, $\gamma > \gamma_{\text{thr}}$ holds thereafter. □

B Choice of the violation-aware gate

We compare the candidate gates by the slope structure they impose near zero violation—a design criterion for spanning strictness, not a claim about learned behavior. Figure 4 compares four candidates, and the bottom row zooms into the region where the violation approaches zero. Design (a), $\omega = \alpha/(\alpha + \gamma v^2)$, has a vanishing slope at $v = 0$ for every head, so the loss provides no gradient pushing small violations to zero. Design (b), $\omega = (1 + \gamma v)^{-\alpha}$, has a nonzero slope for every head, so all heads penalize even small violations, leaving no spread toward the relaxed end. Design (c), $\omega = 1/(1 + \gamma v^\alpha)$, spreads the slopes well but the ordering collapses: since $v^\alpha = 1$ at $v = 1$ for every α, all heads meet there and the ordering flips beyond it. Design (d), $\omega = 1/(1 + \gamma u^\alpha)$ with $u = v/(v + 1)$, spreads the slopes well and, since $u < 1$ for all finite v, keeps a consistent ordering without collapse. We adopt Design (d), with the exponents spanning $[1, \alpha_{\text{ub}}]$.

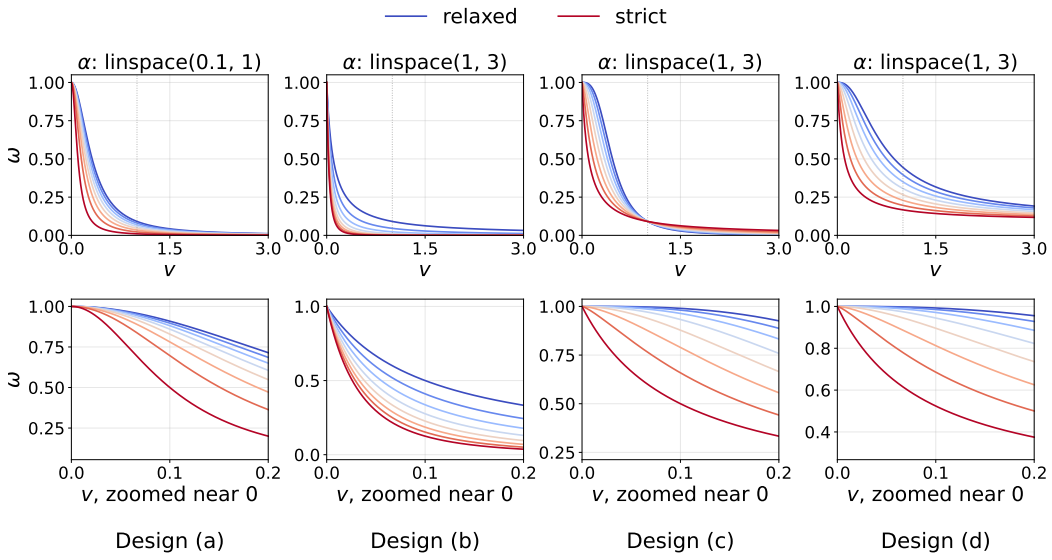

Figure 4: Comparison of candidate violation-aware gates ω as a function of the violation v (top: full view; bottom: zoomed near zero). Colors run from strict (red) to relaxed (blue) heads.

C Implementation details

C.1 Instance representation

Each ILP instance is encoded as a variable–constraint bipartite graph, where variable i and constraint k are joined by an edge whenever $a_{ki} \neq 0$, following prior studies (Nair et al., 2020; Zeng et al., 2024; Geng et al., 2025). We write $\mathcal{C}_i = \{k : a_{ki} \neq 0\}$ for the constraints incident to variable i and $\mathcal{V}_k = \{i : a_{ki} \neq 0\}$ for the variables in constraint k, with degrees $d_i^v = |\mathcal{C}_i|$ and $d_k^c = |\mathcal{V}_k|$.

Without loss of generality, we assume $\mathbf{a}_k \neq \mathbf{0}$ for every constraint and $\mathbf{c} \neq \mathbf{0}$. This ensures all normalizations in Eqs. (6) and (7) and below are well defined.

Following Lee & Kim (2026), our neural network operates on equilibration-scaled data (Tomlin, 1975) to stabilize training under widely varying coefficient scales. This scaling multiplies each constraint by a positive scalar while preserving problem equivalence. It yields $\tilde{a}_{ki} = a_{ki}/M_k$ and $\tilde{b}_k = b_k/M_k$, where $M_k = \max_i |a_{ki}|$. The objective is scaled by its largest magnitude, $\tilde{c}_i = c_i / \max_j |c_j|$. At evaluation, predicted solutions are assessed on the original, unscaled problem.

Following common practice (Han et al., 2023; Zeng et al., 2024; Geng et al., 2025), each node is described by features derived from the objective, the constraints, and the graph structure, listed in Table 4. Several statistics are normalized to a bounded range, denoted $\mathrm{nrm}(\cdot)$: nonnegative count features are divided by their maximum over nodes, signed features by the maximum absolute value (floored at 1), and nodes with no incident edges receive 0. Each constraint node receives a sinusoidal positional encoding (Vaswani et al., 2017) of its index within the instance, added to its initial embedding with dropout of 0.1. All features serve as the node and edge inputs to the network.

Table 4: Input features of our bipartite graph representation.

Definition	Description
Variable node	
$\tilde{c}_i = c_i / \max_j \lvert c_j \rvert$	Objective coefficient, globally scaled by the largest magnitude.
$\max_{k \in \mathcal{C}_i} \tilde{a}_{ki}$	Largest incident constraint coefficient.
$\min_{k \in \mathcal{C}_i} \tilde{a}_{ki}$	Smallest incident constraint coefficient.
$\frac{1}{d_i^v} \sum_{k \in \mathcal{C}_i} \tilde{a}_{ki}$	Mean of the coefficients on incident constraints.
$\mathrm{std}_{k \in \mathcal{C}_i} \tilde{a}_{ki}$	Standard deviation of the incident constraint coefficients.
$\mathrm{nrm}(d_i^v)$	Number of incident constraints.
$\mathrm{nrm}\big(\sum_{k \in \mathcal{C}_i} d_k^c\big)$	Sum of degrees over incident constraints.
$\mathrm{nrm}\big(\frac{1}{d_i^v} \sum_{k \in \mathcal{C}_i} d_k^c\big)$	Mean degree of neighboring constraints.
$\tilde{c}_i / \max(d_i^v, 1)$	Objective coefficient per incident constraint.
$\max_{k \in \mathcal{C}_i} \max_{j \in \mathcal{V}_k} \tilde{c}_j$	Largest objective coefficient reachable through a shared constraint.
$\mathrm{rank}(\tilde{c}_i)/n$	Rank of the objective coefficient among all n variables.
$\mathbb{I}[\, i \text{ binary} \,]$	Binary-variable indicator.
$\mathbb{I}[\, i \text{ integer} \,]$	Integer-variable indicator.
Constraint node	
$\tilde{b}_k = b_k/M_k$	Right-hand side, scaled by the constraint's largest absolute coefficient M_k.
$\max_{i \in \mathcal{V}_k} \tilde{a}_{ki}$	Largest coefficient in the constraint.
$\min_{i \in \mathcal{V}_k} \tilde{a}_{ki}$	Smallest coefficient in the constraint.
$\frac{1}{d_k^c} \sum_{i \in \mathcal{V}_k} \tilde{a}_{ki}$	Mean of the constraint's coefficients.
$\mathrm{std}_{i \in \mathcal{V}_k} \tilde{a}_{ki}$	Standard deviation of the constraint's coefficients.
$\mathrm{nrm}(d_k^c)$	Number of variables in the constraint.
$\mathrm{nrm}\big(\sum_{i \in \mathcal{V}_k} d_i^v\big)$	Sum of degrees over member variables.
$\mathrm{nrm}\big(\frac{1}{d_k^c} \sum_{i \in \mathcal{V}_k} d_i^v\big)$	Mean degree of member variables.
$\mathrm{nrm}\big(\tilde{b}_k / \max(d_k^c, 1)\big)$	Right-hand side per member variable.
$\max_{i \in \mathcal{V}_k} \tilde{c}_i$	Largest objective coefficient among member variables.
$\mathrm{nrm}\big(\sum_{i \in \mathcal{V}_k} \tilde{c}_i\big)$	Total objective weight carried by member variables.
$\frac{\tilde{\mathbf{a}}_k^\top \tilde{\mathbf{c}}}{\lVert\tilde{\mathbf{a}}_k\rVert_2 \, \lVert\tilde{\mathbf{c}}\rVert_2}$	Cosine similarity between the constraint row and the objective.
Edge	
$\tilde{a}_{ki} = a_{ki}/M_k$	Constraint coefficient on edge (k, i), scaled per constraint.

C.2 NETWORK ARCHITECTURE DETAILS

AC layer. The weights s_{ki} in Eq. (4) are computed by attention over the constraints incident to x_i. With learnable projections $\mathbf{W}_x$, $\mathbf{W}_b$, an attention vector $\vec{w}$, and $\|$ denoting concatenation,

$$s_{ki} = \frac{\exp\big(\mathrm{LeakyReLU}(\vec{w}^\top[\,\mathbf{W}_x\vec{x}_i \,\|\, \mathbf{W}_b\vec{b}'_k\,])\big)}{\sum_{\ell:\, a_{\ell i}\neq 0} \exp\big(\mathrm{LeakyReLU}(\vec{w}^\top[\,\mathbf{W}_x\vec{x}_i \,\|\, \mathbf{W}_b\vec{b}'_\ell\,])\big)}. \tag{12}$$

The normalization over $\{\ell : a_{\ell i} \neq 0\}$ ensures $s_{ki} \in [0, 1]$ and $\sum_{k:\, a_{ki}\neq 0} s_{ki} = 1$, as required in Eq. (4). The softmax weights in Eq. (12) are positive at finite logits, so weights on inactive incident constraints vanish only in the limit. In practice, to keep the term s_{ki}/a_{ki} bounded, we clamp $|a_{ki}|$ from below at $\epsilon = 10^{-2}$ in the divisor. The implementation therefore approximates the recovery operator of Lemma 1, and we regard it as an inductive bias toward that operator (Section 3.1.1).

The functions φ and ψ in Eq. (4) are single linear layers that project $\mathbb{R}^d$ to one d-dimensional vector per attention head. The aggregated messages of all heads are averaged before the update function ϕ, which applies layer normalization followed by two linear layers with hidden size $2d$, a LeakyReLU (slope 0.2), and dropout of 0.1 in between.

Conditioning and decoders. The strictness factors are initialized by applying the sigmoid to H values evenly spaced over $[-1, 1]$. $\mathrm{MLP}_{\mathrm{dec}}$ and $\mathrm{MLP}_{\mathrm{cmp}}$ share the same architecture but not the parameters: two hidden layers of size d with ReLU activations and dropout of 0.1, followed by a linear layer with scalar output. MLP_p maps the scalar τ_h through two linear layers with a ReLU in between, followed by layer normalization. MLP_o applies layer normalization to its input, followed by two linear layers with a LeakyReLU (slope 0.2) and dropout of 0.1 in between. The function σ is the sigmoid for binary variables and, for general integer variables, $\sigma(z) = l_i + \mathrm{softplus}(z)$, clamped at the upper bound u_i when it is finite; variables without a finite lower bound are not supported.

C.3 TRAINING DETAILS

Following Geng et al. (2025), we adopt a cosine annealing learning-rate schedule with warm restarts (Loshchilov & Hutter, 2017), which periodically decays the learning rate and then raises it again, sweeping across a wide range of learning rates. A single unit T_{u} in Algorithm 1 defines

Algorithm 1 Training under the feasibility-centric curriculum

Input: ILP instance, total steps T_{total}, initial penalty weight γ_0, growth factor $\rho > 1$, easing rate $\epsilon_{\mathrm{a}} > 0$, window unit T_{u}
Initialize: $\gamma \leftarrow \gamma_0$, $\mathcal{L}_{\mathrm{best}} \leftarrow \infty$, $\mathcal{L}_{\mathrm{ref}} \leftarrow \infty$, $q \leftarrow 0$
for each training step t **do**
  $\hat{\mathbf{x}}, \hat{\mathbf{x}}_{\mathrm{cmp}} \leftarrow$ predicted solutions from the model
  $\eta_{\mathrm{o}} \leftarrow \min(1,\ 2t/T_{\mathrm{u}})$
  $\eta_{\mathrm{i}} \leftarrow \min(1,\ t/T_{\mathrm{u}})$
  compute $\mathcal{L}_{\mathrm{total}}$ using η_{o}, η_{i} and current γ {Equation (9)}
  update model parameters by backpropagating $\mathcal{L}_{\mathrm{total}}$
  if $T_{\mathrm{u}} < t < T_{\mathrm{total}} - T_{\mathrm{u}}$ **then**
    if $\mathcal{L}_{\mathrm{total}} < \mathcal{L}_{\mathrm{ref}}$ **then**
      $q \leftarrow 0$
      $\gamma \leftarrow \rho\,\gamma$
      $\mathcal{L}_{\mathrm{best}} \leftarrow$ loss re-evaluated at the new γ with the current $\hat{\mathbf{x}}$ and $\hat{\mathbf{x}}_{\mathrm{cmp}}$
      $\mathcal{L}_{\mathrm{ref}} \leftarrow \mathcal{L}_{\mathrm{best}}$
    else
      $q \leftarrow q + 1$
      $\mathcal{L}_{\mathrm{ref}} \leftarrow \mathcal{L}_{\mathrm{best}} + \epsilon_{\mathrm{a}} q^2$
    end if
  end if
end for

the warmup length, the restart period, and the γ-freezing window, keeping the schedule controlled by one interval. In particular, T_{wu} in Section 3.2.4 coincides with T_{u}. We set T_{u} to 10% of the total steps. A linear warmup ramps up the learning rate and the warmup factors over the first T_{u} steps, and the restart period is $T_0 = T_{\mathrm{u}}$ with $T_{\mathrm{mult}} = 1$. Gradients are clipped to unit norm. The curriculum penalty scheduler is kept inactive during the first and last T_{u} steps; γ thus grows only after early training stabilizes and before final convergence. We use the initial penalty weight $\gamma_0 = 10$, the growth factor $\rho = 1.02$, and the easing rate $\epsilon_{\mathrm{a}} = 10^{-8}$. We select the model checkpoint by FR on the validation set, with the average objective value over feasible instances as a tiebreaker. A checkpoint is saved only when the model finds a feasible solution on the validation set; if none is ever saved, the final checkpoint is evaluated.

D SETUP DETAILS

Dataset sizes. All instances are obtained from the official Distributional MIPLIB release[1]. Table 5 shows the average instance sizes per dataset. Distributional MIPLIB (Huang et al., 2024b) labels a distribution *Very Hard* when none of its instances is solved to optimality within one hour. The *Very Hard2* distributions have twice as many variables as the corresponding *Very Hard* ones. Middle-Mile Consolidation Network (MMCN) (Greening et al., 2023), a real-world domain with general integer variables, is evaluated only in Appendix E.10. In total, our evaluation spans eight datasets: four domains, each at two tiers.

Table 5: Average instance sizes per benchmark and hardness tier.

Domain	Hardness	#Variables (binary)	#Variables (integer)	#Constraints
MIS	Very Hard	6,000	0	23,994.82
	Very Hard2	12,000	0	48,009.62
CA	Very Hard	4,000	0	2,676.32
	Very Hard2	8,000	0	5,354.08
SC	Very Hard	4,000	0	5,000
	Very Hard2	8,000	0	5,000
MMCN	Medium	1,156.94	263.23	437.81
	Hard	2,074.76	346.39	642.57

Hyperparameters. We distinguish hyperparameters shared across datasets used in the main experiments from those set per dataset. All hyperparameters were selected on the CA-VH validation set and reused for the other datasets, with only those in Table 6 adjusted where necessary. The shared hyperparameters are as follows: all training-related hyperparameters specified in Appendix C.3, 100 training epochs, the exponent upper bound $\alpha_{\mathrm{ub}} = 3$, the number of trade-off heads $H = 8$, the hidden size $d = 32$, eight attention heads per AC layer, and the AdamW optimizer with learning rate 3×10^{-4} and weight decay 10^{-2}. All results use training seed 1, except Figures 8 and 9, which averages over seeds 1–3. Table 6 lists the hyperparameters that differ across datasets. SC-VH performs best with a single AC layer; DiffILO likewise uses its smallest depth for SC, suggesting that shallow models suit this domain. The integrality weight λ is set per domain using the validation set, though performance is stable over a broad range of λ (Appendix E.9).

Table 6: Hyperparameters that differ across datasets.

Hyperparameter	Dataset			Description
	MIS-VH	CA-VH	SC-VH	
num_layers	4	4	1	Number of AC layers.
λ	0.5	0.5	0.1	Weight of the integrality term.

[1] https://huggingface.co/datasets/weiminhu/D-MIPLIB

ILP solvers. Because our evaluation targets primal solution quality rather than proofs of optimality, we configure both solvers to focus on finding better primal solutions, following Han et al. (2023); Huang et al. (2024a); Geng et al. (2025): `MIPFocus` $= 1$ in Gurobi and the aggressive heuristic emphasis in SCIP. This strengthens the solvers on the axis we compare. All solver runs are restricted to a single thread, following Han et al. (2023); the public implementations of DDIM (Zeng et al., 2024) and DiffILO (Geng et al., 2025) likewise adopt this setting. LP-R solves the LP relaxation with Gurobi under the same settings and rounds the solution to the nearest integers. All remaining parameters for the ILP solvers stay at their defaults. Following Geng et al. (2025), the trust region is imposed as the constraint $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 \leq \Delta$ for the entire run, so Start+TR is a restricted primal search that does not preserve global optimality, unlike Start alone. The trust-region radius in Section 4.5 is $\Delta = 1{,}560$ on MIS, $1{,}040$ on CA, and 800 on SC, obtained from DiffILO's per-domain ratios of Δ to the number of variables (0.13, 0.13, and 0.1, respectively).

Computing environment. All experiments run on a machine equipped with two AMD EPYC 7302 CPUs @ 3.0 GHz, 2048 GB of RAM, and three NVIDIA A100 GPUs. Our implementation is based on PyTorch 2.6.0, torch_geometric 2.7.0, and torch_sparse 0.6.18. We use Gurobi 13.0.2 and SCIP 10.0 as ILP solvers. Each run uses a single GPU, and all learned methods restrict PyTorch to a single CPU thread at inference.

Measurement details. TTP covers all computation from loading the ILP instance to producing the rounded candidates, including input-feature construction, graph building, and model inference; the subsequent feasibility check and best-candidate selection are excluded for all methods alike. Timings are end-to-end wall-clock with no warm-up excluded. TTM is extracted from the solver logs for both solvers; since Gurobi's log timestamps have a one-second resolution, SCIP's finer-grained log is forward-filled onto the same one-second grid for consistency. For a run with incumbent value z_t at time t, we compute $\mathrm{PI} = \sum_{t=1}^{T} \mathrm{pg}(z_t)$ with $T = 3600$ seconds and a one-second grid, where pg is the primal gap evaluated against the per-instance best objective value. Following Berthold (2013); Huang et al. (2024a), pg is set to 1 when no incumbent exists or when z_t and the reference differ in sign. For warm-started runs, the timings include LibrAx's TTP, so no incumbent exists during the first TTP seconds.

E Additional results

E.1 Training cost comparison

All learned baselines are trained on each domain with their default settings, with two resource-driven exceptions. First, DiffILO's default 1,200 epochs on MIS would require over 400 hours of training, as estimated from the measured per-epoch cost, which is impractical. We instead cap training at 150 epochs (≈54 hours), still over $15\times$ LibrAx's training budget; its training loss shows no improvement over the final 20% of training. Second, DDIM's training batch sizes are adjusted to fit GPU memory: for CISP pretraining, 16 on SC; for diffusion training, 16 on MIS and SC.

Table 7 summarizes the resulting training costs. Following Geng et al. (2025), data collection (DC) denotes the machine time spent obtaining labels with an ILP solver on training and validation instances, and we count it toward the total training cost. We extend DC to include feature preprocessing performed in advance. For DDIM, DC comprises label collection plus input-graph construction per instance; following its default setting, labels are collected by solving each instance under a

Table 7: Training cost in hours. Data collection (DC) covers feature preprocessing performed in advance and label generation by a solver for supervised methods; DC is cumulative machine time, summed over instances. The lowest total per domain is in bold.

Approach	Method	MIS-VH			CA-VH			SC-VH		
		DC	Train	Total	DC	Train	Total	DC	Train	Total
Supervised	DDIM	309.8	1.4	311.2	254.1	0.7	254.8	253.1	3.8	256.9
Unsupervised	DiffILO	0.7	54.0	54.7	0.1	68.7	68.8	0.7	75.7	76.4
	LibrAx	0.0	3.7	**3.7**	0.0	1.8	**1.8**	0.0	30.9	**30.9**

1,000-second time limit. For DiffILO, DC corresponds to its feature preprocessing. LibrAx computes features on the fly, so their cost is included in its training time. Overall, the training costs of DDIM and DiffILO exceed LibrAx's on every domain, by 46× and 11× on average (geometric mean). The DDIM's gap stems from label collection, which LibrAx's unsupervised training avoids.

E.2 Loss–solution alignment

Figure 5 visualizes how the loss ranks the candidates in the finite pool produced by the inference sweep, on an instance drawn at random from each domain. For every rounded candidate, we compute the candidate loss—the semi-hard loss averaged over the H training exponents ($\eta = 1$)—at the final penalty weight γ reached by the curriculum. Among feasible candidates the ranking is fixed by construction: the gate is fully open ($\omega = 1$) and the integrality term vanishes at integer points, so the loss reduces to the normalized objective. We therefore focus on how infeasible candidates are ranked relative to feasible ones, and make two observations across all test instances. First, the loss-minimizing candidate, marked by the star in Figure 5, is feasible on every instance and domain. Second, the feasible candidates are separated from the infeasible ones in loss on every instance on MIS and SC and on 98% of CA instances. These observations are consistent with the large values of γ (Appendix E.4), at which the gate approaches the 0/1 switch of the hard-constrained regime discussed in Section 2.2. This pushes infeasible candidates above the feasible cluster.

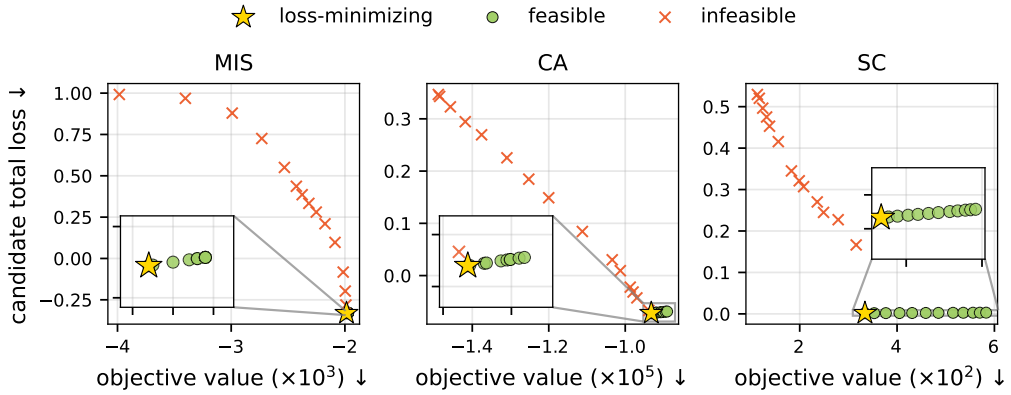

Figure 5: Candidate loss versus objective value per domain on the VH tier, evaluated at the final penalty weight. Insets magnify the feasible cluster. The star marks the loss-minimizing candidate, which is feasible on every instance. For clarity, only candidates from the original-form decoder are shown; the plotted instance is drawn at random.

E.3 Penalty-weight dilemma of soft-constrained losses

DiffILO trains with a soft-constrained loss whose penalty term scales linearly with the penalty weight, and its exactness guarantee (Theorem 3 of Geng et al. (2025)) holds only for a sufficiently large weight. Because the penalty scale grows with the weight, the public implementation adapts the weight during training with a domain-specific upper bound. We inspect the penalty weight at the end of training. On MIS and CA, the penalty weight sits at its default cap (150 and 10, respectively) and no feasible solution is produced. On SC, it remains at 3.68, below the cap of 5.0, and training succeeds. These caps might suffice for the instance distributions evaluated by Geng et al. (2025), with 1,500–2,000 variables, whereas the VH distributions have 4,000–6,000 variables (Table 5). Since the threshold required by the guarantee is instance-dependent, a cap set for one distribution can fall short of it on another.

We retrain DiffILO on MIS with the cap removed. The penalty weight grows to 1.5×10^4, and training converges to feasible solutions with almost no objective value (FR 100% and PG 99.8% on MIS-VH), indicating that the penalty term dominates the loss in this setting. This illustrates the dilemma discussed in Section 4.2: the cap guards against penalty dominance but falls below the threshold its exactness guarantee requires in the evaluated configuration. Appendix E.7 isolates the loss form within LibrAx's pipeline and shows that the dilemma persists there, albeit partly alleviated by the spectrum and the penalty scheduler.

E.4 Training dynamics under penalty-weight growth

Figure 6 shows the training loss and the penalty weight γ during training on each domain. Each point is a snapshot of the last batch of an epoch; we overlay an exponential moving average ($\alpha = 0.8$) for readability. Shaded regions mark the warmup and cooldown windows of the curriculum scheduler, during which γ is frozen.

In the initial freeze window, the violation loss stays low because the objective and integrality terms are barely active with the small warmup factors. As the warmup factors grow, the objective term pushes the violation loss up. Once the scheduler activates, increasing γ progressively strengthens the feasibility enforcement, and the violation loss is driven back down for the remainder of training.

The scheduler drives γ to large final values ($\gamma \approx 2.4 \times 10^4$ on MIS, 8.0×10^2 on CA, and 2.4×10^8 on SC). Since the violation penalty enters only through the bounded gate $\omega \in (0, 1]$ (Eq. (6)), this growth sharpens the feasibility pressure without inflating the loss scale. This contrasts with soft-constrained losses, where the penalty scales linearly with the weight and can dominate the loss, as observed for DiffILO (Appendix E.3) and for the soft-constrained variant under the same pipeline as LibrAx (Appendix E.7).

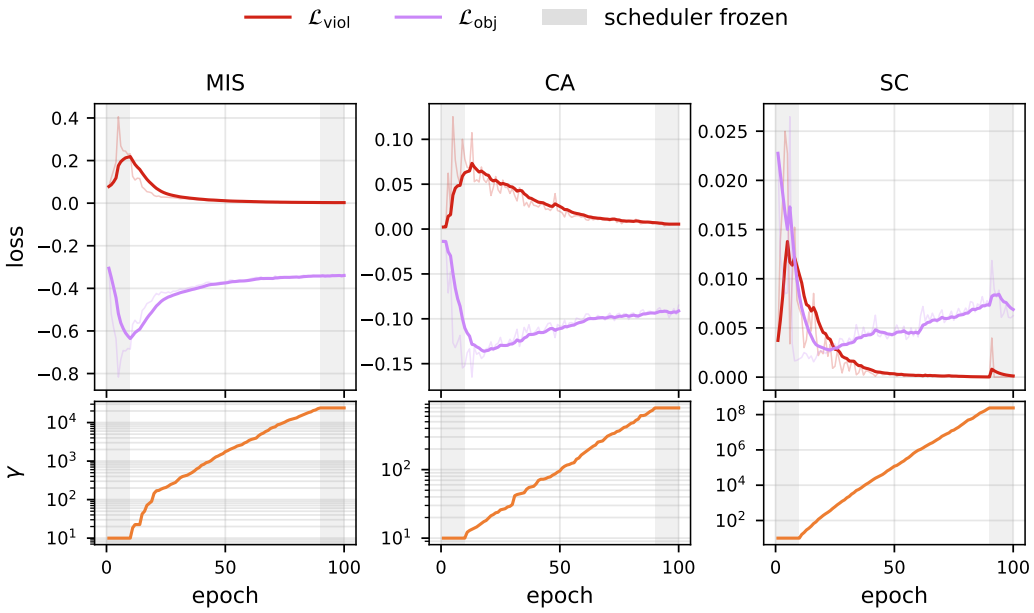

Figure 6: Training dynamics on MIS, CA, and SC. Top: training loss; bottom: penalty weight γ (log scale). Shaded regions mark the scheduler's freeze windows.

E.5 Trained strictness factors and the swept spectrum

Figure 7 visualizes the candidate pool for a single test instance per domain in the violation–objective plane, comparing the $H = 8$ trained strictness factors against 25 inference values. Two key patterns emerge across all domains. First, the candidates trace a trade-off curve on the plotted instances: reducing the violation consistently comes at the cost of objective quality, without crossings. This is consistent with the Spearman correlations reported in Section 4.4, which indicate that both the violation and the objective are nearly monotone in τ. Second, the inference values smoothly interpolate between the trained factors and extrapolate beyond them at both ends along the same curve; the trained factors span $[0.24, 0.77]$ on MIS, $[0.16, 0.77]$ on CA, and $[0.24, 0.72]$ on SC. Extrapolating toward the relaxed end yields candidates with larger violations, whereas extrapolating toward the strict end produces feasible candidates that sacrifice objective quality.

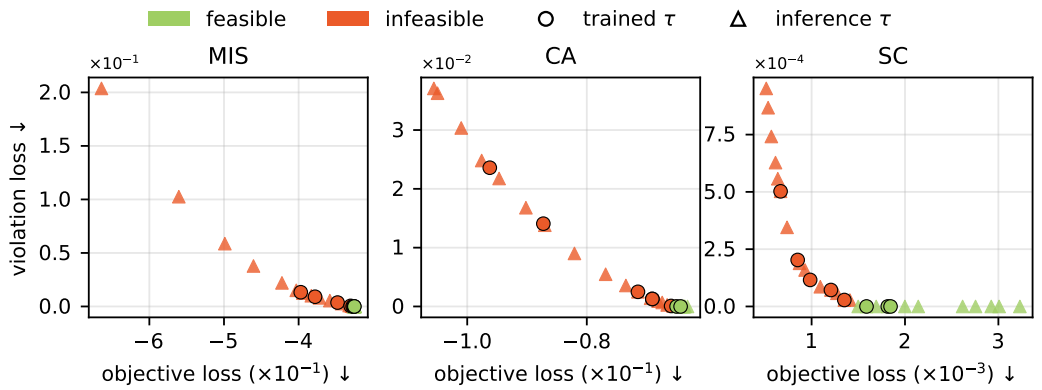

Figure 7: The feasibility–objective trade-off spectrum swept on the VH test instance drawn at random per domain. Circles mark the $H = 8$ trained strictness factors; triangles mark 25 inference values. Candidates from the original-form decoder are shown for clarity.

E.6 Predictions as trust-region centers

A predicted solution can remain valuable even if infeasible, provided that it incurs only minor constraint violations and lies in a region near high-quality feasible solutions. To evaluate how effectively each method locates such regions, we use its best solution as the center of a trust region and allow Gurobi to search within its neighborhood. We increase the trust-region radius Δ from small to large, which probes how far each prediction deviates from a strong feasible solution, and impose a time limit of 30 seconds per instance. This time limit excludes the time to produce the solutions (TTP) because the experiment concerns the quality of the prediction as a center rather than the speed of producing it. For each method, the center is selected as the feasible candidate with the best objective value; for LP-R, its single candidate serves as the center. When no candidate is feasible, DiffILO and LibrAx follow their fallback protocols, selecting the candidate with the lowest loss. DDIM lacks such a criterion because its supervised loss requires labels that are unavailable at test time. We therefore select its candidate with the smallest violation, breaking ties by objective value.

Table 8 shows that LibrAx achieves the best PG at every Δ within the given time limit. This indicates that searching around LibrAx's center reaches better solutions compared to the baselines under the same budget of time and Δ. Interestingly, while DDIM attains a 0% FR on MIS-VH2 on its own (Table 1), its feasibility recovers to 45% at $\Delta = 5$ and reaches 100% at $\Delta = 10$. This

Table 8: Trust-region search with Gurobi around each method's best solution on the VH2 tier under a 30-second time limit, varying the trust-region radius Δ. Best PG per dataset and Δ in bold.

Dataset	Method	$\Delta = 1$		$\Delta = 5$		$\Delta = 10$		$\Delta = 50$	
		FR ↑	PG ↓	FR ↑	PG ↓	FR ↑	PG ↓	FR ↑	PG ↓
MIS-VH2	LP-R	100	99.3	100	99.2	100	99.1	100	98.1
	DDIM	0	100.0	45	64.6	100	21.2	100	20.2
	DiffILO	0	100.0	0	100.0	0	100.0	0	100.0
	LibrAx	100	**5.9**	100	**5.8**	100	**5.7**	100	**4.9**
CA-VH2	LP-R	100	46.9	100	45.3	100	43.5	100	33.6
	DDIM	100	32.3	100	31.5	100	30.7	100	26.4
	DiffILO	0	100.0	0	100.0	0	100.0	0	100.0
	LibrAx	100	**20.0**	100	**19.3**	100	**18.6**	100	**14.9**
SC-VH2	LP-R	0	100.0	0	100.0	0	100.0	100	89.7
	DDIM	100	64.2	100	64.3	100	63.2	100	57.4
	DiffILO	100	62.1	100	62.1	100	61.6	100	56.4
	LibrAx	100	**56.4**	100	**56.4**	100	**55.6**	100	**48.4**

suggests that DDIM's predictions on MIS-VH2 lie close to the feasible region. Nevertheless, its PG still lags behind LibrAx's, implying that an accurate initial prediction matters so that the Δ budget is spent on improving the objective rather than on recovering feasibility. Meanwhile, LP-R becomes fully feasible on SC-VH2 only at $\Delta = 50$, whereas DiffILO remains infeasible on MIS-VH2 and CA-VH2 even at $\Delta = 50$. On SC-VH2, all three learned methods yield feasible solutions on their own (Table 1), eliminating the need to expend any Δ budget on feasibility repair. LibrAx still leads by 6–9 percentage points at every Δ, demonstrating that its advantage stems from the quality of the center itself.

E.7 Training dynamics of the soft-constrained variant

The w/ soft variant of Section 4.6 replaces the semi-hard constrained loss with a soft-constrained one of the form $\mathcal{L} = \mathcal{L}_{\text{obj}} + \gamma_h \mathcal{L}_{\text{viol}} + \lambda \mathcal{L}_{\text{int}}$, keeping everything else identical. Per-head weights γ_h are evenly spaced on a log scale over $[10^{-1.5}, 10^{1.5}]$, so that the range covers the initial penalty weight γ_0 of the semi-hard constrained loss. Concretely, γ_h spans from 0.03 to 31.6, and $\gamma_0 = 10$. All γ_h follow the same schedule as LibrAx's γ (Eq. (10)), monitoring the soft-constrained loss.

Figure 8 compares the training dynamics of the two losses on CA-VH, averaged over three training seeds (1–3). Under the soft-constrained loss, the objective term dominates early training: $\mathcal{L}_{\text{obj}}$ drops sharply while $\mathcal{L}_{\text{viol}}$ rises to about ten times the level reached under the semi-hard constrained loss. The violation returns to zero only as the weight grows, and $\mathcal{L}_{\text{obj}}$ rises back along the way.

Over the final 20 epochs where both losses have plateaued, the two reach a comparable violation level, yet the semi-hard constrained loss settles at a lower objective loss with non-overlapping error bars across the three seeds (Figure 8 (right)). This matches the lower PG of LibrAx (20.0% vs. 22.8%; Table 3), which outperforms the w/ soft variant on 95 of 100 test instances (Wilcoxon signed-rank test, $p < 10^{-17}$). The plateauing of $\mathcal{L}_{\text{obj}}$ at an inferior value indicates penalty dominance once the weight grows large, suppressing further optimization of the objective—the penalty-weight dilemma (Appendix E.3). It is worth noting that the variant still attains full feasibility; given that removing the spectrum or the penalty scheduler degrades LibrAx (Table 3), we attribute this to the spectrum of per-head weights and the scheduler. The remaining gap thus isolates the gate itself. The semi-hard constrained loss bounds the penalty term and instead suppresses the objective term when violations are severe. The soft-constrained loss sets strictness solely by the penalty weight, whereas the relaxed heads of our loss keep a small gate slope even under violations. This avoids excessive repulsion from the constraint boundary and likely leads to a better objective value.

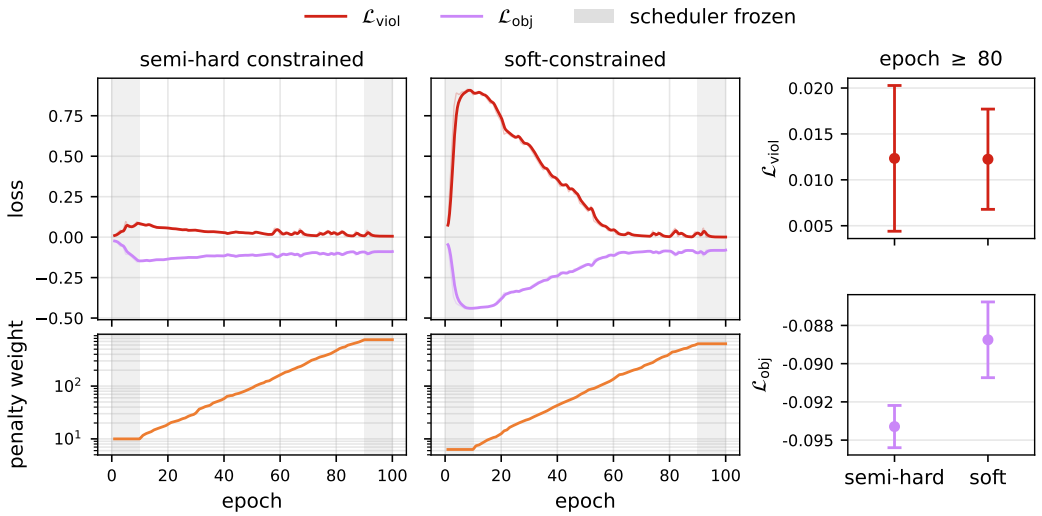

Figure 8: Training dynamics of the semi-hard constrained loss (LibrAx) and the soft-constrained variant on CA-VH, averaged over three seeds. Left and middle: loss terms (top) and penalty weight (bottom; γ for LibrAx, the mean of γ_h over heads for the soft-constrained variant). Shaded regions mark epochs where the scheduler is frozen. Right: loss averaged over the final 20 epochs; mean $\pm$ standard deviation across three seeds.

E.8 Convergence comparison with ablated variants

Figure 9 compares the best validation objective over training between LibrAx and the w/o AC and w/o AUX variants of Section 4.6. Validation objective values are measured on the equilibration-scaled problem (Appendix C.1). For each run, the curve tracks the best validation objective at each epoch; curves are averaged over three training seeds (1–3), and segments are dashed while the validation FR of any seed is below 100%. The w/o AC variant is identical to LibrAx, except that the AC layers are replaced by the standard bipartite GNN layers with an enlarged hidden size. The hidden size is adjusted so that its parameter count (241K on MIS/CA and 133K on SC) is comparable to LibrAx's (220K and 127K) without being smaller.

On MIS and CA, LibrAx reaches its converged objective level earlier than the w/o AC variant, which lags behind at a worse objective before catching up; on MIS, its curve remains above LibrAx's throughout training. On SC, where LibrAx uses a single AC layer (Table 6), the two architectures are compared with minimal confounding from depth. There, the variant plateaus at a worse objective, failing to reach LibrAx's final quality within the same training budget. We attribute these gaps to the AC layer's built-in recovery operator (Proposition 1), which spares the network from learning active-constraint structure from scratch. It is worth noting that fitting solutions with the standard bipartite GNN is known to be parameter-hungry (Chen et al., 2023); see Appendix F.2.

The w/o AUX variant approaches LibrAx's objective level only toward the end of training on CA. On SC, it plateaus at a worse objective than LibrAx, like the w/o AC variant. This matches the motivation of Section 3.2.3: the compressed objective supplies gradient to small-coefficient variables. The effect matters on CA and SC, where the ratio of the largest to the smallest nonzero $|c_i|$ within an instance is 3.8×10^3 and 100 on geometric average, respectively. MIS is excluded from this analysis since its coefficients are uniform, making the compressed objective coincide with the original.

Overall, both components accelerate training convergence. This acceleration also leads to a better final validation objective when the training budget cannot compensate for the slower convergence of the ablated variants, as on SC.

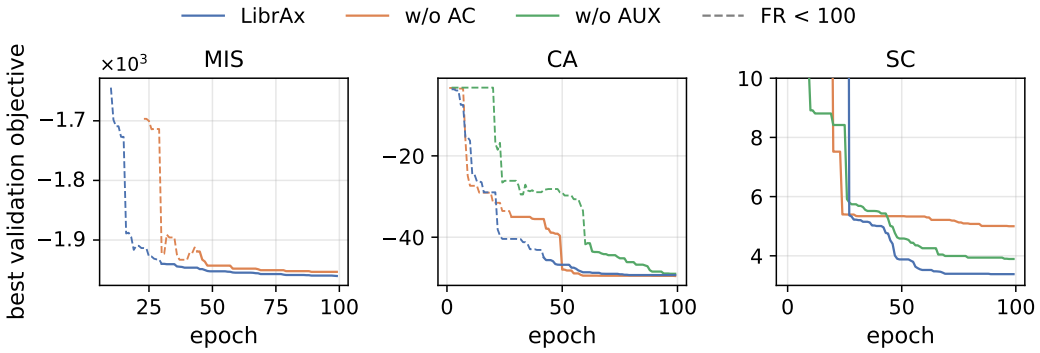

Figure 9: Best validation objective during training for LibrAx and its ablated variants on the VH tier, averaged over three seeds. The w/o AUX variant is shown for CA and SC only, as the compressed objective coincides with the original on MIS. Dashed segments mark epochs where the validation FR is below 100%. The y-axis on SC is truncated at 10.

E.9 Hyperparameter sensitivity

As shown in Table 6, all loss-related hyperparameters except the integrality weight λ are fixed across domains while maintaining strong performance, suggesting limited sensitivity; thus, λ warrants the most attention. Table 9 reports a sweep of $\lambda \in \{0.1, 0.3, \ldots, 0.9\}$ on MIS and CA. On both domains, PG varies only mildly for $\lambda \in [0.3, 0.9]$, and LibrAx outperforms the baselines at all of these values (Table 1). At $\lambda = 0.1$, PG degrades to 15.4% and 48.9%, respectively. This degradation is moderate compared with the collapse at $\lambda = 0$, which corresponds to the w/o INT ablated variant in Section 4.6. This indicates that guiding predictions toward integer values is essential, and $\lambda \geq 0.3$ suffices on both domains in our experiments, though the suitable value may differ across domains.

Table 9: Sweep of the integrality weight λ on MIS and CA. Our default ($\lambda = 0.5$) is marked with †.

Dataset	Metric	λ				
		0.1	0.3	$0.5^\dagger$	0.7	0.9
MIS-VH	FR↑	94	100	100	100	100
	PG↓	15.4	7.5	6.5	7.0	7.3
CA-VH	FR↑	100	100	100	100	100
	PG↓	48.9	21.1	20.0	21.1	23.3

E.10 Extension to general integer variables

Most ILP solution predictors focus on binary variables (Han et al., 2023; Huang et al., 2024a; Zeng et al., 2024; Liu et al., 2025a; Geng et al., 2025). For instance, DDIM and DiffILO, our baselines, are formulated for binary variables and do not apply to general integer variables: DDIM restricts its formulation to $\mathbf{x} \in \{0, 1\}^n$, and DiffILO relies on a relaxed Bernoulli distribution. In contrast, LibrAx's components are not tied to binary variables: the AC layer and the semi-hard constrained loss are both defined for general integer variables. To test this applicability, we evaluate on Middle-Mile Consolidation Network (MMCN) (Greening et al., 2023), a real-world domain from Distributional MIPLIB with general integer variables (Table 5), where LP-R is the only applicable baseline. For general integer variables, the decoder activation σ is softplus without clamping (Appendix C.2), as all general integer variables in MMCN have bounds $[0, \infty)$. Hyperparameters follow SC-VH (Appendix D), except $\lambda = 0.9$, the hidden size $d = 64$, and 300 training epochs.

Table 10 shows that LibrAx reaches 99% and 100% FR on MMCN at the Medium and Hard tiers, respectively, where LP-R yields no feasible solution, suggesting LibrAx's applicability to general integer variables. There remains room for improvement in solution quality, which we attribute to two factors. First, the decoder is shared across variable types in this experiment, so a single decoder must serve the sigmoid scale of binary variables and the unbounded scale of general integer variables. Second, MMCN lies outside the scope of Proposition 2: its general integer variables have no finite upper bounds, so predictions are not confined to a bounded box. This experiment thus serves as a stress test beyond the setting the theory covers, yet these feasibility rates suggest that our loss retains most of its effectiveness rather than failing outright. Designing dedicated decoders for each variable type and deriving implied variable bounds to close this gap is beyond the scope of this applicability test, and we leave improving LibrAx on general integer variables for future work.

Table 10: Results on the Medium and Hard tiers of MMCN, which contains general integer variables.

Domain	Hardness	Method	FR↑	PG↓	TTP↓
MMCN	Medium	LP-R	0	100.0	0.01
		LibrAx	99	65.7	0.03
	Hard	LP-R	0	100.0	0.02
		LibrAx	100	66.3	0.04

F Additional related work

F.1 Machine learning for ILP

Learning-based approaches to ILP fall into two complementary lines, differing in what is learned. The first learns decisions within the solving process, e.g., for cut selection (Tang et al., 2020; Paulus et al., 2022), variable selection (Khalil et al., 2016; Alvarez et al., 2017; Gasse et al., 2019; Gupta et al., 2020; Kuang et al., 2024), and node selection (He et al., 2014; Labassi et al., 2022) in the branch-and-bound loop, or through learned large neighborhood search (Song et al., 2020; Sonnerat et al., 2021; Wu et al., 2021; Huang et al., 2023; Liu et al., 2025b). The second learns the solution itself: a predictor directly outputs a candidate solution, serving as a learned primal heuristic. Most

predictors are supervised: they imitate near-optimal target solutions collected by solver runs (Nair et al., 2020; Yoon, 2022; Han et al., 2023; Huang et al., 2024a; Zeng et al., 2024; Liu et al., 2025a), and refine the prediction with hard fixing (Nair et al., 2020; Yoon, 2022) or a trust-region search around the prediction (Han et al., 2023; Huang et al., 2024a; Liu et al., 2025a; Geng et al., 2025). Obtaining these labels is itself expensive: near-optimal targets may require running Gurobi with a time limit of one hour (Han et al., 2023; Huang et al., 2024a), and training requires hundreds of such instances. This cost grows further with instance size, limiting practicality on large NP-hard problems (Cantürk et al., 2024). Lee & Kim (2026) propose a label-free alternative and position it as complementary to solution predictors, as it iteratively explores feasible solutions rather than predicting variable values directly.

Another line removes the need for labels entirely, directly optimizing a differentiable surrogate of the problem itself. Penalty-based unsupervised losses were studied for specific CO problems (Karalias & Loukas, 2020; Schuetz et al., 2022; Wang & Li, 2023; Gaile et al., 2022). Geng et al. (2025) brought this paradigm to ILP, directly optimizing the ILP objective with a weighted violation penalty. These losses are all soft-constrained: feasibility is enforced only by letting an additive penalty outweigh the objective, leaving them prone to the penalty-weight dilemma. Moreover, the trade-off between the two goals is fixed once training ends. LibrAx differs in both respects. Its gated penalty remains bounded for any penalty weight, yet the global minimizers of its offset form round to ILP optima above a finite threshold (Proposition 2). Its strictness conditioning further exposes the trade-off as a spectrum controllable at inference.

F.2 GNN ARCHITECTURE FOR SOLUTION PREDICTION

Most learned predictors for ILP follow the standard bipartite GNN (Gasse et al., 2019): they encode an instance as a variable–constraint bipartite graph and apply learned message passing that alternates variable-to-constraint and constraint-to-variable updates (Nair et al., 2020; Yoon, 2022; Han et al., 2023; Huang et al., 2024a; Zeng et al., 2024; Liu et al., 2025a; Geng et al., 2025). This design is theoretically supported: GNNs on this encoding can approximate ILP solution mappings, given random features or restricted instance classes (Chen et al., 2023). In practice, however, the approximation with the standard bipartite GNN is parameter-hungry; even on small instances ($m = 6$, $n = 20$), a GNN trained to fit the optimal solution needs orders of magnitude more parameters than one trained to fit only the optimal objective value (Chen et al., 2023). Indeed, the bipartite GNN architecture for ILP solution predictors has been noted as still simple and open to further research (Geng et al., 2025). A well-established remedy is to match the architecture's inductive biases to the structure of the task, which aids generalization and data efficiency (Battaglia et al., 2018). The AC layer follows this principle for ILP: its message passing can approximate the active-constraint recovery operator (Proposition 1), so the network need not learn that structure from scratch.