LEARNING-GUIDED INFEASIBILITY RELAXATION FOR MIXED-INTEGER LINEAR PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world Mixed-Integer Linear Programs (MILPs) frequently face infeasibility. While recent learning-based methods accelerate MILP solving for feasible ones, infeasibility relaxation at scale remains a critical, unresolved bottleneck. To address this, we present L-IR, the first Learning-guided framework for accelerating MILP Infeasibility Relaxation. By formalizing relaxation as a constraint support set coverage task, L-IR learns to intelligently isolate the subset of constraints that, in hindsight, require relaxation and filters out the rest, yielding a reduced problem. Our L-IR is trained via a novel tripartite objective: a quality-weighted binary classification loss to learn multiple high-quality relaxations, a complete-support ranking loss to encourage early ranking of complete high-quality supports, and a coverage loss to dynamically truncate high-ranked subsets during inference. Architecturally, we introduce a constraint-centric geometric representation to explicitly capture conflict topology. Moreover, to address data scarcity, we construct a comprehensive benchmark of 20,000 infeasible MILPs across 580 problem classes. Extensive evaluations show that L-IR accelerates exact solvers by up to 6.08× while maintaining relaxation quality. We further demonstrate L-IR's strong generalization across varying relaxation hardness and unseen distributions.

1 INTRODUCTION

Mixed-Integer Linear Programs (MILPs) are essential tools for modeling practical decision-making problems across diverse domains, including computational biology (Shlomi et al., 2008), scheduling (Zhou et al., 2023), and manufacturing (Belvaux & Wolsey, 2001). In practice, however, a model is rarely right the first time: conflicting requirements, overly ambitious targets, or shifting operating conditions frequently render it infeasible. At that point, the solver returns no plan at all, and the decision-maker must instead find the least costly way to relax the constraints to make it feasible.

Classical methods tackle infeasibility from two complementary angles: *diagnosis* and *repair*. Diagnosis methods (Chinneck & Dravnieks, 1991; Chen et al., 2025) extract an Irreducible Infeasible Subsystem (IIS) to explain *why* a model fails, but an explanation is not a remedy: it does not tell us which constraints to relax, or by how much. Repair methods, such as Gurobi's `feasRelax`, answer this directly by attaching a slack to every constraint and minimizing the total violation cost. This formulates an auxiliary MILP that is often as hard as the original one, which can take hours to solve, exceeding what real-time operational decisions allow. This raises a practical question: *to what extent can machine learning accelerate infeasibility relaxation for MILPs?*

Recent learning-based methods accelerate MILP solving but primarily target feasible instances, leaving infeasibility relaxation underexplored. A natural extension is then to apply methods such as predict-and-search (Huang et al., 2024a) to the auxiliary relaxation MILP. However, these methods do not exploit the structure of infeasibility relaxation and yield limited speedups in our experiments (Table 1). In this paper, we explore a more targeted opportunity: although all m constraints may be relaxed, high-quality repairs often modify only a small subset of them (e.g., 2.4%); restricting relaxation to such a candidate set yields up to 11.8× speedups (Section 4). This motivates a targeted learning task: predicting *where relaxation is needed*. By selecting a compact candidate set and fixing all remaining slacks to zero, we shrink the relaxation space and the complexity of the auxiliary MILP before optimization, and leave the solver to determine *how much to relax*.

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.

However, turning this opportunity into reliable acceleration requires more than identifying individually promising constraints. Because relaxation decisions are coupled, the central challenge is to predict a compact candidate set that covers the entire support of at least one high-quality relaxation. This calls for an instance-level *set coverage* objective that balances compactness against repair quality. Moreover, as this learning task remains unexplored, two further challenges arise: the lack of large-scale training data and of neural architectures tailored to these coupled relaxation decisions.

In this paper, we present *Learning-guided Infeasibility Relaxation* (L-IR), the first learning-guided framework for accelerating MILP infeasibility relaxation. L-IR casts candidate selection as a *constraint support coverage* task and bridges per-constraint predictions and set-level coverage with three complementary losses: (i) a quality-weighted classification loss for soft constraint scoring, (ii) a differentiable complete-support ranking loss that promotes early coverage of high-quality supports, and (iii) a coverage regression loss that predicts an instance-adaptive truncation ratio. To capture how constraints conflict, we design a constraint-centric neural network with geometry-aware features. Moreover, to overcome data scarcity, we extend the LLM-based MILP-Evolve (Li et al., 2025b) with infeasibility-based mutations and reflection-based parameter search, building a benchmark of about 20,000 infeasible MILPs across 580 problem classes and three hardness levels.

We make the following contributions: **(1)** We introduce L-IR, the first learning-guided framework for accelerating MILP infeasibility relaxation, and formalize it as a constraint support coverage task. **(2)** We propose a tripartite training objective and a constraint-centric neural architecture, which together enable instance-adaptive reduction. **(3)** We construct the first large-scale benchmark of infeasible MILPs via the infeasibility-mutation operator and reflection-based parameter search. **(4)** Extensive experiments show that L-IR accelerates Gurobi by up to 6.08× while preserving relaxation quality, outperforming classical heuristics and neural baselines on both synthetic and real-world benchmarks.

2 Related Work

Infeasibility Handling. Prior work follows two paradigms. *Diagnosis* explains infeasibility, classically by extracting an Irreducible Infeasible Subsystem (IIS) via deletion or elastic filtering (Chinneck & Dravnieks, 1991; Guieu & Chinneck, 1999), with recent LLM-based tools translating it into natural language (Chen et al., 2025; Li et al., 2025a; Ao et al., 2026). Yet an IIS only locates a contradiction, without prescribing which constraints to relax. *Repair* actively restores feasibility: exact solvers offer elastic tools (e.g., Gurobi's feasRelax and CPLEX's feasOpt) that minimize slack-based violation costs, and Zouitine et al. (2025) learn an RL policy for sequential constraint deletion, restricted to LPs and SAT and requiring a costly solver-in-the-loop feasibility check at every step. Both remain prohibitive for large-scale MILPs, motivating learning-based acceleration.

Learning-Guided MILP Solvers. Modern solvers such as Gurobi (Gurobi Optimization, LLC, 2026) and SCIP (Achterberg, 2009) build on Branch-and-Bound, whose heuristic modules are increasingly replaced by learned policies (Bengio et al., 2021; Scavuzzo et al., 2024) for branching (Khalil et al., 2016; Gasse et al., 2019; Gupta et al., 2022), cut selection (Huang et al., 2022; Wang et al., 2023; Paulus et al., 2022; Puigdemont et al., 2024), node selection (He et al., 2014), separation (Li et al., 2023), heuristic scheduling (Khalil et al., 2017), presolving (Liu et al., 2024), and large neighborhood search (Wu et al., 2021). Other method instead shrinks the search space by predict-and-search (Nair et al., 2020; Han et al., 2023; Huang et al., 2024a; Liu et al., 2025) or identifying active constraints (Li et al., 2026), powered by MILP representations evolving from bipartite graphs (Gasse et al., 2019) to richer encodings (Ding et al., 2020; Wang et al., 2026) and stronger backbones (Huang et al., 2026). However, they all assume feasible instances, leaving *infeasible* MILPs largely unexplored.

Instance generation and benchmarks. As learning-based solvers are heavily data-driven, a parallel line of research focuses on synthetic instance generation and benchmarking. Early methods rely on standard generators from Ecole (Prouvost et al., 2020) (e.g., set cover, combinatorial auctions) and evaluate on benchmarks like MIPLIB 2017 (Gleixner et al., 2021a) and Distributional MIPLIB (Huang et al., 2024b). To enhance data diversity and hardness, recent works use variational autoencoder (Geng et al., 2023; Guo et al., 2024) and Diffusion model (Zhang et al., 2024) to generate new MILP instances by mimicking the existing ones. Most recently, MILP-Evolve (Li et al., 2025b) utilizes LLMs to iteratively evolve generator code, yielding thousands of distinct instance classes. Yet, a critical gap remains: none of these pipelines produce *infeasible* instances, which are common in real-world scenarios. This motivates us to *first* build a framework to generate infeasible instances.

3 Preliminary: MILP Formulation and Infeasibility Relaxation

MILP formulation. The MILP with m constraints and n variables can be formulated as

$$\min \mathbf{c}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b}, \tag{1}$$

where $\mathbf{x} \in \mathbb{Z}^q \times \mathbb{R}^{n-q}$ is the vector of q integer decision variables and $n-q$ continuous ones. $\mathbf{c} \in \mathbb{R}^n$ is the objective coefficient vector. The constraint coefficient matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and the right-hand side $\mathbf{b} \in \mathbb{R}^m$ define m linear constraints. Many practical applications formulated as Eq.(1) often encounter infeasibility, especially at the initial stage of decision-making due to conflicting demands or overly ambitious targets. Formally, an infeasible MILP is an optimization model in which no solution $\mathbf{x}$ exists that satisfies all m constraints simultaneously.

Infeasibility relaxation. To repair the infeasibility, a standard practice is to reformulate the original MILP by relaxing constraints (Gurobi Optimization, LLC, 2026), yielding an auxiliary MILP problem:

$$\min F(\mathbf{s}) \quad \text{s.t.} \quad \mathbf{A}\mathbf{x} \leq \mathbf{b} + \mathbf{s}, \quad \mathbf{s} \in \mathbb{R}^m_+, \tag{2}$$

where the slack variables $\mathbf{s} > 0$ denotes the constraints to be relaxed, and $F(\cdot)$ is the corresponding violation cost function. In practice, the choice of $F(\cdot)$ mathematically represents specific repair philosophies (Chen et al., 2025) (e.g., the ℓ_1-norm for minimizing total violations in linear cost), which inherently dictates the resulting optimal relaxation set. While our proposed framework is broadly applicable to various cost formulations, we evaluate it under two representative definitions of $F(\cdot)$: total violations and the number of violated constraints, with details provided in Appendix A.

4 Methodology

To efficiently solve the infeasibility relaxation in Eq. (2), we propose the *Learning-guided Infeasibility Relaxation* (L-IR) framework, organized around two questions. First, *what* to learn: we formalize infeasibility relaxation as a constraint support coverage task, enabling MILP reduction backed by optimality guarantees and empirical evidence. Second, *how* to learn: we design a pipeline that combines LLM-based benchmark generation, a constraint-centric neural network with geometric features, a tripartite training objective, and instance-adaptive inference.

4.1 MILP Reduction and Learning Task

We accelerate MILP infeasibility relaxation from a natural yet unexplored angle: restricting the slack variables $\mathbf{s}$. Specifically, fixing $s_i = 0$ forces constraint i to remain satisfied, whereas allowing $s_i \geq 0$ permits its relaxation. Any feasible relaxation of Eq. (2) thus induces a *relaxation support* $S = \text{supp}(\mathbf{s}) := \{i \mid s_i > 0\}$, and we denote by S^* the support of an optimal relaxation (not necessarily unique). Given a candidate set $C \subseteq \{1, \ldots, m\}$, the restricted problem fixes $s_i = 0$ for all $i \notin C$:

$$V(C) = \min_{\mathbf{x} \in \mathbb{R}^n,\, \mathbf{s} \in \mathbb{R}^m_+} F(\mathbf{s}), \text{ s.t. } \mathbf{A}\mathbf{x} \leq \mathbf{b} + \mathbf{s};\ s_i = 0,\ \forall i \notin C. \tag{3}$$

We define $V(C) = +\infty$ if the restricted problem is infeasible. The solver then jointly optimizes the primal variables and the remaining slacks within C. Notably, inclusion in C *permits* rather than *enforces* relaxation, meaning the realized support can be any subset of C. However, overly restricting C may exclude high-quality solutions. Formally speaking, we note the following proposition:

Proposition 1 (Optimality of Support Coverage)**.** *Assume the unrestricted problem attains a finite optimum V^* and every feasible restricted problem attains its optimum. Then (i)* ***Feasibility****: the restricted problem is feasible iff C contains the support of some feasible relaxation; (ii)* ***Optimality****: $V(C) \geq V^*$, with equality iff C contains the support of some optimal relaxation.*

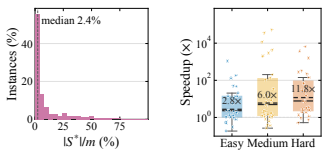

Figure 1: Optimal support set sizes and oracle speedup.

High Acceleration Potential. We now highlight the learning opportunity. Figure 1(a) reveals a favorable property: optimal supports are highly sparse, with $|S^*|/m$ having a median of only 2.4%. To quantify the resulting potential, we run an oracle experiment with $C = S^*$. As shown in Figure 1(b), this support reduction speeds up Gurobi by up to 11.8× over the unrestricted relaxation model on the *hard* validation set.

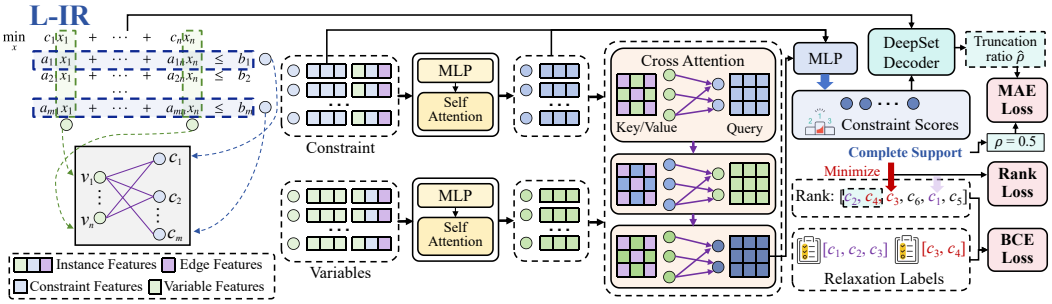

Figure 2: Overview of our L-IR framework, which includes an attention-based constraint-centric encoder with geometric features and two complementary decoders to predict constraint scores and support truncation ratios of a complete support. Our network is trained via a novel tripartite objective.

Learning Task. To realize this acceleration, the candidate set C shall fully cover at least one high-quality support ($S \subseteq C$) to preserve solution quality, while remaining compact ($|C| \ll m$) to ensure efficiency. Balancing these requirements raises three learning challenges: 1) identifying constraints likely to require relaxation; 2) ensuring that top-ranked constraints complete an entire high-quality support; and 3) truncating the ranking so that $|C| \ll m$ without breaking coverage. Our tripartite training objective addresses these three challenges, respectively.

4.2 Learning-guided Infeasibility Relaxation (L-IR)

We now present the L-IR framework. First, we overcome training data scarcity by constructing a large-scale benchmark of infeasible MILPs and using Gurobi (Gurobi Optimization, LLC, 2026) to obtain high-quality relaxation supports as supervision. Next, we develop a *constraint-centric neural network* g_θ that uses geometric representations to capture constraint conflicts and learns support predictions through a *tripartite training objective*. At inference time, g_θ ranks constraints and predicts the fraction of the ranked list needed to cover a high-quality support, defining the candidate set $\hat{C}$. By fixing the slack variables associated with the remaining $m - |\hat{C}|$ constraints to zero, L-IR produces a reduced relaxation model for the solver, thereby accelerating infeasibility relaxation.

Large-Scale Infeasible MILP Benchmark. Despite their prevalence in industrial applications, infeasible MILPs are often considered anomalous and are largely overlooked by the research community. For instance, the standard MIPLIB benchmark (Gleixner et al., 2021b) contains merely 44 infeasible instances. This lack of data fundamentally hinders data-driven methodologies. To bridge this gap, we construct the first large-scale benchmark of intrinsically infeasible MILPs, building upon the generation framework of MILP-Evolve (Li et al., 2025b). We extend this framework with tailored infeasibility-specific modifications and a novel, highly efficient reflection-based parameter search scheme. Starting from eight classic MILP seed codes from Ecole (Prouvost et al., 2020), we inject initial infeasibilities sourced from practical conflicts. We then prompt an LLM to iteratively refine the problem generator code. Alongside standard genetic operators, we introduce a bespoke *infeasibility-mutation* operator that injects distributed, non-trivial contradictions, such as capacity cascades, bound tightening, and coupled-row shifts. We ensure the quality of the generated code by: 1) requiring the LLM to explicitly articulate these injected conflicts to align with real-world scenarios, and 2) filtering every generated problem class based on feedback from Gurobi. Using the evolved generator code, we instantiate diverse MILP instances via our *reflection-based parameter search scheme*. Unlike prior random grid search methods, this scheme is guided by both solver feedback and the LLM's semantic reasoning of the code, enhancing efficiency. Finally, we successfully generate a massive, diverse benchmark comprising approximately 20,000 instances across 580 problem classes. For each infeasible MILP instance, we extract the optimal support S^* and high-quality intermediate supports to serve as training labels. Detailed generation pipelines and hyperparameters are in Appendix B.

Geometry-Aware Features. Infeasibility arises from a geometric contradiction, where the constraint hyperplanes fail to form a feasible intersection. To expose these conflicts, we encode the MILP as a constraint-variable bipartite graph (Gasse et al., 2019), using a feature set $\mathcal{G} = \{C, \mathcal{V}, \mathcal{E}, \mathcal{I}\}$ explicitly tailored to capture its underlying geometry. For constraint nodes $c_i \in C$, we augment the standard attributes, i.e., sense, coefficient, bias, objective similarity, with geometric features

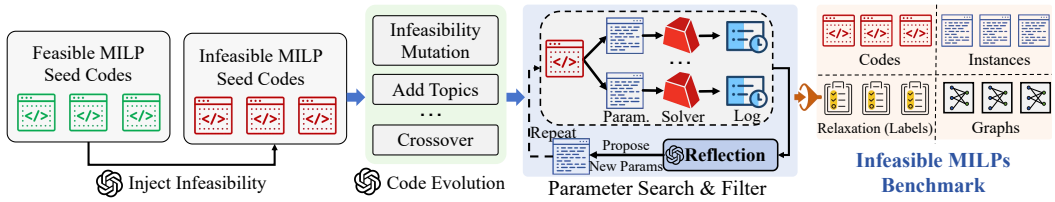

Figure 3: Pipeline of our infeasible MILP benchmark generation.

overlooked by the prior works, forming the constraint features f_C. We integrate variable composition and bound-related indicators to quantify how tightly a constraint restricts its variables, alongside inter-row interactions (e.g., parallel domination) to capture relationships invisible in isolation. For variable nodes $v_i \in \mathcal{V}$, we encode $f_\mathcal{V}$ with its coefficients, types, bounds, and objective relations, enabling the network to assess how constraints couple and conflict over shared variables. For edges $\mathcal{E} = \{e_{ij} | \mathbf{A}_{ij} \neq 0, \forall 1 \leq i \leq n, 1 \leq j \leq m\}$, we encode $f_\mathcal{E}$ with normalized coefficients in $\mathbf{A}$, preserving each variable's contribution to a constraint. Global instance features $f_\mathcal{I} = \{|\mathcal{C}|, |\mathcal{V}|, |\mathcal{E}|\}$ capture problem scale and sparsity. Together, this geometry-aware representation explicitly guides the network to pinpoint candidates for relaxation. Details are in Appendix C.

Constraint-Centric Network Architecture. Unlike prior works that treat variables and constraints symmetrically, we treat infeasibility relaxation is a constraint-driven task. We propose an asymmetric, *constraint-centric* network g_θ that explicitly biases the message-passing flow toward constraint representations. Building upon but diverging from the parallel dual-attention mechanism (Huang et al., 2026), we design a sequential architecture. We first initialize embeddings $h_\mathcal{C}^0$ and $h_\mathcal{V}^0$ by fusing local and global features via Layer Normalization (LN) and Multilayer Perceptron (MLPs):

$$h_\mathcal{C}^0 = \text{MLP}_\mathcal{C}^0(\text{LN}_\mathcal{C}^0(f_\mathcal{C})) + \text{MLP}_\mathcal{I}^0(\text{LN}_\mathcal{I}^0(f_\mathcal{I})), \quad h_\mathcal{V}^0 = \text{MLP}_\mathcal{V}^0(\text{LN}_\mathcal{V}^0(f_\mathcal{V})) + \text{MLP}_\mathcal{I}^0(\text{LN}_\mathcal{I}^0(f_\mathcal{I})). \tag{4}$$

For each layer $\ell \in [1, L]$, we compute intra-type linkages via kernel linear self-attention:

$$h_\mathcal{C}^{\ell(0)} = \text{SelfAttention}_\mathcal{C}^\ell(h_\mathcal{C}^{\ell-1}), \quad h_\mathcal{V}^{\ell(0)} = \text{SelfAttention}_\mathcal{V}^\ell(h_\mathcal{V}^{\ell-1}). \tag{5}$$

Next, we capture inter-type bipartite interactions via sparse cross-attention, restricted to existing edges ($\mathbf{A}_{ij} \neq 0$). Different from Huang et al. (2026), we compute this sequentially using $h^{\ell(0)}$) after self attention and enrich the keys and values with edge features $f_\mathcal{E}$:

$$\begin{aligned} h_\mathcal{C}^{\ell(1)} &= \text{CrossAttn}_\mathcal{C}^{\ell(1)}(W_{QC}^{\ell(1)} h_\mathcal{C}^{\ell(0)}, W_{KC}^{\ell(1)} h_\mathcal{V}^{\ell(0)} + W_{K\mathcal{E}}^{\ell(1)} f_\mathcal{E}, W_{VC}^{\ell(1)} h_\mathcal{V}^{\ell(0)} + W_{V\mathcal{E}}^{\ell(1)} f_\mathcal{E}), \\ h_\mathcal{V}^{\ell(1)} &= \text{CrossAttn}_\mathcal{V}^{\ell(1)}(W_{QV}^{\ell(1)} h_\mathcal{V}^{\ell(0)}, W_{KV}^{\ell(1)} h_\mathcal{C}^{\ell(0)} + W_{K\mathcal{E}}^{\ell(1)} f_\mathcal{E}, W_{VV}^{\ell(1)} h_\mathcal{C}^{\ell(0)} + W_{V\mathcal{E}}^{\ell(1)} f_\mathcal{E}). \end{aligned} \tag{6}$$

To enable the constraint-centric bias, we introduce a subsequent cross-attention hop designed to update only the constraint embeddings, leveraging the newly updated variable representations $h_\mathcal{V}^{\ell(1)}$:

$$h_\mathcal{C}^{\ell(2)} = \text{CrossAttn}_\mathcal{C}^{\ell(2)}(W_{QC}^{\ell(2)} h_\mathcal{C}^{\ell(1)}, W_{KC}^{\ell(2)} h_\mathcal{V}^{\ell(1)} + W_{K\mathcal{E}}^{\ell(2)} f_\mathcal{E}, W_{VC}^{\ell(2)} h_\mathcal{V}^{\ell(1)} + W_{V\mathcal{E}}^{\ell(2)} f_\mathcal{E}). \tag{7}$$

To alleviate over-smoothing, a residual module concatenates the intermediate constraint representation: $h_\mathcal{C}^L = \text{MLP}^L([h_\mathcal{C}^0; h_\mathcal{C}^{L(1)}; h_\mathcal{C}^{L(2)}])$, where $[;]$ denotes concatenation. A final projection MLP maps $h_\mathcal{C}^L$ to a score z_i for each constraint. We compute the probability p_i of each constraint belonging to the high-quality support via a sigmoid layer. While these scores provide a continuous ranking of constraints, extracting a compact candidate set requires a precise truncation. Since the size of a high-quality support varies significantly across different instances (Figure 5), the existing relying on a fixed probability threshold is inadequate. To address this and enable instance-adaptive truncation, we integrate an additional deep-sets decoder (Zaheer et al., 2017). This decoder aggregates the distribution of constraint scores z_i and combines it with global instance features to predict a dynamic truncation ratio $\hat{\rho}^\theta$. Finally, the network yields two complementary outputs: constraint scores $\boldsymbol{z}^\theta$ that define the ranking, and a ratio $\hat{\rho}^\theta$ that dictates the truncation to fully cover a high-quality support.

Tripartite Training Objectives. As formalized in Section 4.1, our learning goal is to identify a minimal candidate set C that perfectly covers at least one high-quality relaxation support. For each training instance, we collect K feasible relaxations with supports $S_k = \text{supp}(\mathbf{s}^{(k)})$ and costs $F_k = F(\mathbf{s}^{(k)})$. Let $F_{\min} = \min_k F_k$. We define the relative optimality gap as $\text{gap}_k = (F_k - F_{\min})/\max\{|F_{\min}|, \epsilon_f\}$, and let $\mathcal{K}_\delta = \{k : \text{gap}_k \leq \delta\}$ index the high-quality relaxations. To guarantee a bounded solution quality

$V(C) \leq F_{\min} + \delta \max\{|F_{\min}|, \epsilon_f\}$, the set C must satisfy the explicit coverage condition: $S_k \subseteq C$ for at least one $k \in \mathcal{K}_\delta$. While this guarantee holds trivially for any excessively large C, realizing an actual computational speedup requires C to remain highly compact ($|C| \ll m$). To seamlessly balance this coverage guarantee with compactness, our network jointly learns an *accurate constraint ranking* and a *precise truncation ratio* via a tripartite objective: 1) a *quality-weighted classification loss* to supervise individual constraint scores; 2) a *complete-support ranking loss* to push entire valid supports early in the ranking; and 3) a *coverage regression loss* to predict the instance-adaptive truncation ratio.

(i) Quality-weighted binary classification. To supervise individual constraint probabilities p_i^θ, we leverage all feasible relaxations with supports S_k. While the lowest-cost support alone already provides a valid target, it discards the information remained in other alternatives. We therefore aggregate all the obtained supports into soft targets, weighted by their solution quality:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{m}\sum_{i=1}^{m}\Big[y_i \log p_i + (1-y_i)\log(1-p_i)\Big], \quad y_i = \sum_k w_k \,\mathbf{1}\{i \in S_k\}, \quad w_k = \frac{e^{-\text{gap}_k/T}}{\sum_k e^{-\text{gap}_k/T}}, \tag{8}$$

where $T > 0$ controls the preference for lower-cost relaxations. To this end, $\mathcal{L}_{\text{BCE}}$ provides constraint-level supervision from multiple feasible relaxations. However, this loss supervises each constraint in isolation, while we want the coverage of a support as a whole: the $|S_k|$ highest-scoring constraints may be drawn from several different relaxations and complete none of them (Proposition 2). This gap motivates a second objective, which rewards the early ranking of an entire high-quality support.

(ii) Complete-support ranking. To enable this complete coverage, we propose a complete-support ranking loss $\mathcal{L}_{\text{rank}}$. Let $r(i)$ denote the descending rank of the constraint score z_i, and define a prefix candidate set $C_\kappa = \{i : r(i) \leq \kappa\}$. A specific support S_k is completely covered only when C_κ expands to include its lowest-ranked member. Thus, to ensure that at least one high-quality support is fully covered, the required prefix length must reach where the earliest support completes in the ranking: $\kappa \geq D_{\text{any}}(\mathbf{p})$, where $D_{\text{any}}(\mathbf{p}) = \min_{k\in\mathcal{K}_\delta} \max_{i\in S_k} r(i)$. We define D_{any} as the *coverage depth*. Intuitively, the exact depth required to fully cover a specific support S_k simply equals its size $|S_k|$ plus the discrete count of non-members ranked strictly above its weakest member, $\underline{z}_k = \min_{i\in S_k} z_i$. We approximate it using a smooth sigmoid surrogate, yielding our complete-support ranking loss:

$$\mathcal{L}_{\text{rank}} = \min_{k\in\mathcal{K}_\delta} \frac{1}{m}\Big(|S_k| + \sum_{j\notin S_k} \sigma((z_j - \underline{z}_k)/\tau)\Big), \tag{9}$$

where $\sigma(\cdot)$ is the sigmoid function and $\tau > 0$ is the temperature. We take the minimum to cover any one of the high-quality supports. As formally established in Lemma 3, this smooth surrogate asymptotically recovers the exact normalized depth, with $\mathcal{L}_{\text{rank}} \to D_{\text{any}}(\mathbf{p})/m$ as $\tau \to 0$.

(iii) Coverage loss. While $\mathcal{L}_{\text{rank}}$ encourages early support completion, the truncation ratio remains unknown at inference. As the actual coverage depth is computable during training, we derive the ground-truth target ratio $\rho = D_{\text{any}}(\mathbf{p})/m$ required by the current scores. We supervise the deep-sets decoder using mean absolute error in logarithmic space, penalizing relative multiplicative errors rather than absolute errors across instances, i.e., $\mathcal{L}_{\text{coverage}} = \left|\log\hat{\rho} - \log\rho\right|$. Overall, our joint training objective is $\mathcal{L} = \lambda_1 L_{\text{BCE}} + \lambda_2 \mathcal{L}_{\text{rank}} + \lambda_3 \mathcal{L}_{\text{coverage}}$, where $\lambda_1, \lambda_2, \lambda_3 \geq 0$ control the loss weights.

Instance-Adaptive Inference. Given a new instance, our network g_θ produces constraint scores $\boldsymbol{z}$ and a truncation ratio $\hat{\rho}$. We sort constraints by decreasing score z_i and select the top $\hat{\rho}$ fraction as $\hat{C}$. We then fix slacks outside $\hat{C}$ to zero and solve Eq. (3), jointly optimizing the primal variables and remaining slacks. If this restricted run generates relaxation that doesn't meet the acceptance quality requirement, we fall back to Gurobi with the full constraint set C, to align with practical deployment.

5 Experiment

Implementation Details. All experiments are conducted using an AMD Ryzen Threadripper PRO 7975WX CPU and NVIDIA RTX 4090 GPUs. Following Appendix B, we generate 17,978 training, 150 validation, and 1,500 test instances, with both evaluation sets further partitioned into in-distribution (ID) and out-of-distribution (OOD) dataset. Models are trained for up to 30 epochs with a batch size of 4, retaining the best validation checkpoint for testing. We evaluate performance across three random seeds (2027-2029). Solvers are restricted to 5 threads, with time limits strictly set to 400 seconds for our synthetic dataset and 3600 seconds for MIPLIB (Gleixner et al., 2021b). Among the

Table 1: Performance on unseen in-distribution and out-of-distribution MILPs.

Methods	Easy			Medium			Hard		
	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
Unseen In-Distribution MILPs									
Gurobi	100.0%	4.24	–	100.0%	34.93	–	98.5%	214.70	–
Random	11.5%	5.74	0.74×	14.9%	43.92	0.80×	10.8%	208.58	1.03×
LP (Farkas)	26.9%	5.33	0.80×	29.9%	35.45	0.99×	17.9%	168.13	1.28×
LP (Dual Prices)	80.2%	6.94	0.61×	72.5%	37.36	0.93×	62.3%	123.31	1.74×
LP (Elastic)	68.6%	5.10	0.83×	59.5%	27.51	1.27×	43.8%	109.70	1.96×
Chinneck (2001)	11.1%	157.16	0.03×	10.6%	241.13	0.14×	7.2%	443.87	0.48×
Li et al. (2026) ($\mathcal{V}$)	50.6%	21.12	0.20×	50.3%	91.03	0.38×	47.6%	231.78	0.93×
Huang et al. (2026) ($\mathcal{V}$)	98.0%	4.62	0.92×	99.1%	40.24	0.87×	96.3%	224.48	0.96×
Li et al. (2026) (C)	71.9%	3.47	1.22×	74.7%	22.02	1.59×	74.4%	102.92	2.09×
Huang et al. (2026) (C)	75.9%	3.09	1.37×	80.3%	20.41	1.71×	79.1%	94.62	2.27×
L-IR (Ours)	94.5%	**2.43**	**1.74**×	96.0%	**10.90**	**3.21**×	94.7%	**35.34**	**6.08**×
Out-of-Distribution (OOD) MILPs									
Gurobi	100.0%	4.55	–	100.0%	41.35	–	98.1%	218.29	–
Random	8.6%	5.62	0.81×	4.4%	44.35	0.93×	6.6%	198.79	1.10×
LP (Farkas)	28.9%	4.25	1.07×	18.3%	35.65	1.16×	11.6%	190.21	1.15×
LP (Dual Prices)	89.8%	3.31	1.37×	86.3%	19.36	2.14×	79.8%	93.53	2.33×
LP (Elastic)	75.1%	**2.51**	**1.82**×	64.5%	16.59	2.49×	54.2%	90.34	2.42×
Chinneck (2001)	21.8%	27.39	0.17×	31.9%	75.94	0.54×	29.6%	193.73	1.13×
Li et al. (2026) ($\mathcal{V}$)	26.4%	16.10	0.28×	24.6%	62.35	0.66×	13.1%	195.56	1.12×
Huang et al. (2026) ($\mathcal{V}$)	100.0%	4.42	1.03×	100.0%	40.58	1.02×	98.1%	200.62	1.09×
Li et al. (2026) (C)	86.2%	2.90	1.57×	86.0%	24.75	1.67×	91.6%	165.80	1.32×
Huang et al. (2026) (C)	84.5%	2.64	1.73×	86.8%	29.22	1.41×	93.6%	183.17	1.19×
L-IR (Ours)	92.0%	**2.51**	**1.82**×	93.9%	**13.52**	**3.06**×	95.3%	**72.26**	**3.02**×

44 infeasible MILPs, we filter out the trivial instances that Gurobi solves within 10 seconds, leaving 23 instances under the total-violation objective in Eq. (11) and 20 under the cardinality objective in Eq. (12). All the ablation studies are conducted on the ID validation dataset. The held-out test set is exclusively reserved for the main baseline comparisons presented in Table 1.

Baseline. We compare our L-IR with the following baselines: 1) *Exact solver*: Gurobi 12.0.3 (Gurobi Optimization, LLC, 2026), which relaxes over all constraints based on B&B algorithm; 2) *Classical heuristics* that construct the candidate set without learning: *Random* draws each constraint into the set with 50% probability; *LP (Farkas)*, *LP (Dual Prices)*, and *LP (Elastic)* each solve a single linear programming (LP) and respectively take the rows with a nonzero Farkas certificate, nonzero dual price, and positive slack, into the candidate set; *Iterative Deletion* (Chinneck, 2001) instead deletes rows one at a time, at each step removing the dual-price candidate whose deletion reduces total infeasibility the most, until the system turns feasible, and returns the deleted rows as the candidate set; 3) *Other learning-guided MILP reduction methods* mainly focusing on binary variable reduction: Li et al. (2026) and Huang et al. (2026), retrained and tested under Predict-and-Search with the inference parameters of their own papers and denoted with the ($\mathcal{V}$) mark. We further adapt them using our proposed MILP reduction with relaxation support, denoting with the (C) mark.

Metrics. Evaluation is performed on 900 unseen ID and 600 OOD datasets under different difficulty (Easy, Medium and Hard), using three random seeds. We report the following metrics on total 4,500 runs: 1) *Solved* is the share of runs returning a high-quality relaxation within 10% of the optimum V^* without any fallback. 2) *Time* is the 1-shifted geometric mean of the solving times in seconds(Gasse et al., 2019), including fallback to full Gurobi if no high-quality relaxation as defined above is found. 3) *Speedup* is the average runtime acceleration compared with Gurobi, i.e., $\text{Time}_{\text{Gurobi}}/\text{Time}_{\text{method}}$. 4) For an instance with m constraints, let S denote the union of all high-quality supports and C the predicted candidate set under truncation budget. We report Recall $= |C \cap S|/|S|$ and True negative rate (TNR) $= 1 - |C \setminus S|/(m - |S|)$, directly representing the set coverage and compactness, respectively.

5.1 Performance on Accelerating Infeasibility Relaxation

As shown in Table 1, L-IR achieves substantial acceleration on both unseen in-distribution and out-of-distribution datasets under the standard total violations cost (Eq. 11). Notably, this advantage scales with problem complexity: on the hardest ID instances, L-IR achieves a 6.08× speedup over

Table 2: Results on relaxation minimizing number of violated constraints.

Methods	Easy			Medium			Hard		
	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
Gurobi	97.2%	1.17	–	95.5%	19.55	–	89.4%	329.81	–
Huang et al. (2026) (*C*)	68.5%	1.12	1.04×	57.0%	12.39	1.58×	51.1%	188.88	1.75×
L-IR (ours)	84.8%	**0.88**	**1.33**×	76.2%	**9.66**	**2.02**×	68.0%	**147.31**	**2.24**×

Table 3: Results on MIPLIB instances.

$F(\cdot)$	Method	Solved ↑	Time ↓	Speedup ↑
Eq. (11)	Gurobi	68.8%	2885.5	–
	L-IR (ours)	67.2%	**2603.8**	**1.11**×
Eq. (12)	Gurobi	51.7%	2502.9	–
	L-IR (ours)	41.7%	**2134.1**	**1.17**×

Table 4: Different parameter search schemes.

Search method	Success%↑	Classes solved ↑
Random grid search	5.0%	43.3%
LLM, code only	12.9%	40.0%
LLM, code + reflection (Cross)	22.3%	43.3%
LLM, code + reflection (Ours)	**27.5%**	**60.0%**

Gurobi while maintaining a 94.7% high-quality solution rate. We note that the speedup is a net metric that already absorbs the overhead of unsolved part compared to Gurobi. Existing learning-guided methods focusing on primal variable reduction (denoted with $\mathcal{V}$) completely fail to accelerate the solver, validating our hypothesis in Section 4.1 that targeting the slack variables is superior to fixing binary variables for this task. We further adapt them to our constraint reduction paradigm (denoted with C), their performance, though improved (e.g., 2.27× by Huang et al. (2026) (C) on Hard instances), remains inferior to L-IR. Moreover, L-IR exhibits superior OOD generalizability against both classical and neural baselines, confirming its capability to identify geometric conflicts.

Adaptability to Constraint Count Cost. To evaluate the robustness of L-IR across different cost formulations, we retrain and test our model to minimize the number of violated constraints (Eq. 12) on the same test instances. As detailed in Appendix A, this objective is mathematically more challenging than penalizing total violations: the average size of reference supports drops by 75%, leaving a lower tolerance for prediction errors. Notably. Table 2 demonstrates that L-IR consistently accelerates the relaxation process over Gurobi, confirming its strong adaptability to diverse optimization goals.

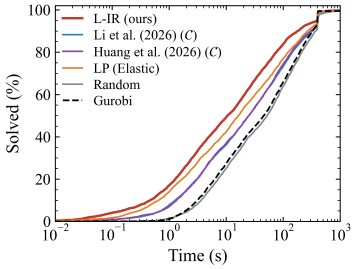

Figure 4: Cumulative solved rate over time.

Generalization to Real-World MILPs. We further test our pretrained model on the real-world benchmark MIPLIB 2017 (Gleixner et al., 2021b). While the model is never fine-tuned on this benchmark, it still reduces the solving time by 1.11× and 1.17× respectively, at a comparable rate of high-quality relaxations (67.2% against 68.8% for Gurobi under the total-violation objective). These results confirm that the learned ranking transfers to instances far outside its training distribution.

5.2 Ablation Study and Further Discussion

Model Prediction Performance. As demonstrated in Table 5, successful relaxation is highly sensitive to prediction accuracy: even a 5% error margin from a perfect oracle causes the solved rate to collapse to 29%. Driven by our tripartite training objective, L-IR successfully navigates this bottleneck, achieving a 98% recall and 82% TNR. This balances coverage with compactness, yielding a mere 4.4× larger candidate set that preserves an 89% solved rate while accelerating the solver by 4.33×.

Effects of Reflection-Based Parameter Search. Parameter search is a critical step in generating massive new instances from the evolved code (Figure 3). We evaluate our reflection-based scheme against the standard random grid search (Li et al., 2025b) using a fixed budget of 16 searches across 30 problem classes. As shown in Table 4, enabling the LLM to reflect on solver feedback increases the configuration success rate from 5.0% to 27.5%, and the solved class rate from 43.3% to 60.0% ($p = 0.001$). This confirms that iterative solver feedback, rather than static code analysis, improves search efficiency. However, we note that this feedback-derived knowledge is most effective when shared intra-class, outperforming cross-class applications. See Appendix F.1 for details.

Table 5: Oracle performance with error analysis on the hard validation dataset.

	Oracle (perfect)	**Oracle + 95% recall** & 95% TNR	**Oracle + 90% recall** & 90% TNR	**Oracle + 70% recall** & 70% TNR	**L-IR (Ours)** (98% recall + & 82% TNR)
Candidate set $\lvert C\rvert/\lvert S_{\text{ref}}\rvert$	1.0	3.9	6.8	18.3	4.4
Solved	100%	29%	25%	6%	89%
Time	17.0	168.5	166.4	226.2	45.7
Speedup	11.77×	1.19×	1.20×	0.88×	4.33×

Table 6: Effect of the enhanced features.

ID	f_C	$f_{\mathcal{V}}$	$f_{\mathcal{E}}$	f_I	**Solved ↑**	**Time ↓**	**Speedup ↑**
1	✓	✓	×	×	64.3%	79.9	2.48×
2	✓$^+$	✓$^+$	×	✓	96.4%	67.3	2.94×
3	✓$^+$	✓$^+$	✓	×	89.3%	63.2	3.13×
4 (Ours)	✓$^+$	✓$^+$	✓	✓	89.3%	**45.7**	**4.33**×

Table 7: Effects on encoder design.

ID	C**-centric**	**Res.**	**Solved ↑**	**Time ↓**	**Speedup ↑**
1	×	✓	85.7%	64.0	3.09×
2	✓	×	82.1%	60.8	3.25×
3 (Ours)	✓	✓	89.3%	**45.7**	**4.33**×

Table 8: Effect of the training objectives.

ID	$\mathcal{L}_{\text{BCE}}$	$\mathcal{L}_{\text{rank}}$	$\mathcal{L}_{\text{cov.}}$	**Recall ↑**	**TNR ↑**	**Solved ↑**	**Time ↓**	**Speedup ↑**
1	✓	×	×	0.96	0.65	78.6%	108.9	1.82×
2	×	✓	✓	0.88	0.77	85.7%	62.8	3.15×
3	✓	×	✓	0.89	**0.83**	75.0%	65.6	3.02×
4	✓	✓	×	0.95	0.81	82.1%	76.5	2.59×
5 (Ours)	✓	✓	✓	**0.98**	0.82	89.3%	**45.7**	**4.33**×

Table 9: Effect of the inference strategy.

Method	**Solved ↑**	**Time ↓**	**Speedup ↑**
Neural Diving	84.5%	148.2	1.35×
PaS	90.5%	193.1	1.04×
Apollo	51.2%	180.1	1.11×
Adaptive (Ours)	89.3%	**45.7**	**4.33**×

Effects of Geometry-Aware Features. We ablate the four feature groups of $\mathcal{G}$. Results in Table 6 and 15 show that augmenting f_C and $f_{\mathcal{V}}$ (marked with $^+$) with our geometric attributes improves the solved rate, while the edge and instance features $f_{\mathcal{E}}$, f_I raise the acceleration, indicating that node features drive support coverage, whereas edge and instance contexts improve set compactness.

Discussion on Network Architectures. Table 7 (Full results in Table 16) ablates our two core encoder designs: constraint-centric asymmetry and the residual module. While the asymmetric flow drives acceleration, it degrades the solved rate. Notably, incorporating the residual stream recovers this lost coverage and compounds the speedup on Hard instances, revealing an architectural synergy.

Effects of Different Loss Terms. Tables 8 and 17 ablate the three objective terms. The classification loss scores individual constraints reliably but alone fails to form compact supports (lowest TNR). The ranking loss improves complete support coverage (higher Recall; see also Figure 7), while the coverage loss turns this ranking into instance-adaptive truncation. Only their combination attains both the highest solved rate and the largest acceleration on Hard instances, showing that scoring, ordering, and truncation are complementary. The effect of loss weights is studied in Appendix F.2.

Advantage of Instance-Adaptive Inference. Prior works design distinct strategies to handle prediction errors, such as aggressively fixing variables (Neural Diving (Nair et al., 2020)), searching trust regions (Predict-and-Search (Han et al., 2023; Huang et al., 2024a)), or correcting via uncertainty (Apollo-MILP (Liu et al., 2025)). However, because they are exclusively tailored to primal variable reduction, their efficiency degrades significantly when adapted to our constraint-side formulation. Moreover, these methods are problem-specific, stipulating that all instances within a MILP class share the same static fixing percentage. In contrast, L-IR pioneers instance-adaptive MILP reduction, dynamically predicting the optimal truncation ratio for each unique instance (Table 9 and 18).

6 Conclusion

This paper proposes the *first* Learning-guided Infeasibility Relaxation (L-IR) framework. By formalizing relaxation as a constraint support set coverage task, L-IR learns to intelligently isolate the subset of constraints that, in hindsight, require relaxation and filters out the rest, yielding a reduced problem. L-IR is then jointly trained via a novel tripartite objective with a constraint-centric geometric representation to explicitly capture conflict topology. To address data scarcity, we construct the first infeasible MILP benchmark of 20,000 infeasible instances across 580 problem classes. Future work includes 1) applying L-IR to more practical violation cost formulations, 2) developing a unified framework for both diagnosis and repair (Ao et al., 2026), and 3) improving the distributional similarity of our synthetic data to the real-world benchmarks (Gleixner et al., 2021b; Huang et al., 2024b)

AI USE STATEMENT

We used generative AI tools for generating synthetic datasets, to polish the writing and to implement simple coding tasks. We have not used generative AI tools for retrieval and discovery, research ideation or execution and proving mathematical claims. We have reviewed all AI-assisted work. For example, LLM-generated data was verified and tested for correctness via the exact solver Gurobi (Gurobi Optimization, LLC, 2026). LLM-generated code was verified by the author. We take responsibility for the final content, including text, claims, or artifacts produced with the aid of AI.

REPRODUCIBILITY STATEMENT

We try every efforts to ensure reproducibility. The detail network architecture is in Section 4.2. The experimental settings along with all the hyperparameter during training and inference are listed in Section 5. We generate a dataset of infeasible MILPs with the details introduced in Appendix B. We promise that our datasets (including the detailed devision of the training, validation and test datasets), code and all the pretrained models will be released upon acceptance. We believe our dataset will be a valuable asset for the research community.

REFERENCES

Tobias Achterberg. SCIP: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

Ruicheng Ao, David Simchi-Levi, and Xinshang Wang. Optirepair: Closed-loop diagnosis and repair of supply chain optimization models with llm agents. *arXiv preprint arXiv:2602.19439*, 2026.

Gaetan Belvaux and Laurence A Wolsey. Modelling practical lot-sizing problems as mixed-integer programs. *Management Science*, 47(7):993–1007, 2001.

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

Hao Chen, Gonzalo Esteban Constante-Flores, Krishna Sri Ipsit Mantri, Sai Madhukiran Kompalli, Akshdeep Singh Ahluwalia, and Can Li. Optichat: Bridging optimization models and practitioners with large language models. *INFORMS Journal on Data Science*, 5(3):199–220, 2025.

John W Chinneck. Fast heuristics for the maximum feasible subsystem problem. *INFORMS Journal on Computing*, 13(3):210–223, 2001.

John W Chinneck and Erik W Dravnieks. Locating minimal infeasible constraint sets in linear programs. *ORSA Journal on Computing*, 3(2):157–168, 1991.

Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. Accelerating primal solution findings for mixed integer programs based on solution prediction. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, volume 34, pp. 1452–1459, 2020.

Matteo Fischetti and Andrea Lodi. Repairing mip infeasibility through local branching. *Computers & operations research*, 35(5):1436–1445, 2008.

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.

Zijie Geng, Xijun Li, Jie Wang, Xiao Li, Yongdong Zhang, and Feng Wu. A deep instance generative framework for milp solvers under limited data availability. *Advances in Neural Information Processing Systems*, 36:26025–26047, 2023.

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

Ambros Gleixner, Gregor Hendel, Gerald Gamrath, Tobias Achterberg, Michael Bastubbe, Timo Berthold, Philipp M. Christophel, Kati Jarck, Thorsten Koch, Jeff Linderoth, Marco Lübbecke, Hans D. Mittelmann, Derya Ozyurt, Ted K. Ralphs, Domenico Salvagnin, and Yuji Shinano. MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library. *Mathematical Programming Computation*, 2021b. doi: 10.1007/s12532-020-00194-3. URL https://doi.org/10.1007/s12532-020-00194-3.

Olivier Guieu and John W Chinneck. Analyzing infeasible mixed-integer and integer linear programs. *INFORMS Journal on Computing*, 11(1):63–77, 1999.

Ziao Guo, Yang Li, Chang Liu, Wenli Ouyang, and Junchi Yan. Acm-milp: Adaptive constraint modification via grouping and selection for hardness-preserving milp instance generation. In *Forty-first International Conference on Machine Learning*, 2024.

Prateek Gupta, Elias B Khalil, Didier Chetélat, Maxime Gasse, Yoshua Bengio, Andrea Lodi, and M Pawan Kumar. Lookback for learning to branch. *arXiv preprint arXiv:2206.14987*, 2022.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2026. URL https://www.gurobi.com.

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. *arXiv preprint arXiv:2302.05636*, 2023.

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Proceedings of the 27th Conference on Neural Information Processing Systems (NeurIPS)*, pp. 3293–3301, 2014.

Peixin Huang, Yaoxin Wu, Yining Ma, Cathy Wu, Wei Zhang, and Wen Song. A general neural backbone for mixed-integer linear optimization via dual attention. *arXiv preprint arXiv:2601.04509*, 2026.

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.

Zeren Huang, Kerong Wang, Furui Liu, Hui-Ling Zhen, Weinan Zhang, Mingxuan Yuan, Jianye Hao, Yong Yu, and Jun Wang. Learning to select cuts for efficient mixed-integer programming. *Pattern Recognition*, 123:108353, 2022.

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.

Elias B Khalil, Bistra Dilkina, George L Nemhauser, Shabbir Ahmed, and Yufen Shao. Learning to run heuristics in tree search. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 659–666, 2017.

Jiajun Li, Yixuan Li, Ran Hou, Yu Ding, Shisi Guan, Jiahui Duan, Xiongwei Han, Tao Zhong, Vincent Chau, Weiwei Wu, and Wanyuan Wang. Constraint matters: Multi-modal representation for reducing mixed-integer linear programming. 2026. URL https://arxiv.org/abs/2508.18742.

Kai Li, Ruihao Zheng, Xinye Hao, and Zhenkun Wang. Multi-objective infeasibility diagnosis for routing problems using large language models. *arXiv preprint arXiv:2508.03406*, 2025a.

Sirui Li, Wenbin Ouyang, Max B. Paulus, and Cathy Wu. Learning to configure separators in branch-and-cut. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=gf5xJVQS5p.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *International Conference on Learning Representations*, 2025b.

Chang Liu, Zhichen Dong, Haobo Ma, Weilin Luo, Xijun Li, Bowen Pang, Jia Zeng, and Junchi Yan. L2p-mip: Learning to presolve for mixed integer programming. In *The Twelfth International Conference on Learning Representations*, 2024.

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. *ICLR'25*, 2025.

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.

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.

Antoine Prouvost, Justin Dumouchelle, Lara Scavuzzo, Maxime Gasse, Didier Chételat, and Andrea Lodi. Ecole: A gym-like library for machine learning in combinatorial optimization solvers. In *Learning Meets Combinatorial Algorithms at NeurIPS2020*, 2020. URL https://openreview.net/forum?id=IVc9hqgibyB.

Pol Puigdemont, Stratis Skoulakis, Grigorios Chrysos, and Volkan Cevher. Learning to remove cuts in integer linear programming. *arXiv preprint arXiv:2406.18781*, 2024.

Lara Scavuzzo, Karen Aardal, Andrea Lodi, and Neil Yorke-Smith. Machine learning augmented branch and bound for mixed integer linear programming. *Mathematical Programming*, pp. 1–44, 2024.

Tomer Shlomi, Moran N Cabili, Markus J Herrgård, Bernhard Ø Palsson, and Eytan Ruppin. Network-based prediction of human tissue-specific metabolism. *Nature biotechnology*, 26(9):1003–1010, 2008.

Ruobing Wang, Xin Li, and Mingzhong Wang. Milpnet: A multi-scale architecture with geometric feature sequence representations for advancing milp problems. In *International Conference on Learning Representations*, volume 2026, pp. 112771–112808, 2026.

Zhihai Wang, Xijun Li, Jie Wang, Yufei Kuang, Mingxuan Yuan, Jia Zeng, Yongdong Zhang, and Feng Wu. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 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, 2021.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017.

Yahong Zhang, Chenchen Fan, Donghui Chen, Congrui Li, Wenli Ouyang, Mingda Zhu, and Junchi Yan. Milp-fbgen: Lp/milp instance generation with feasibility/boundedness. In *Forty-first International Conference on Machine Learning*, 2024.

Jianan Zhou, Yaoxin Wu, Zhiguang Cao, Wen Song, Jie Zhang, and Zhenghua Chen. Learning large neighborhood search for vehicle routing in airport ground handling. *IEEE Transactions on knowledge and data engineering*, 35(9):9769–9782, 2023.

Mehdi Zouitine, Ahmad Berjaoui, Agnès Lagnoux, Clément Pellegrini, and Emmanuel Rachelson. Learning to repair infeasible$*$ problems with deep reinforcement learning on graphs, 2025.

Contents of the Appendix

A Different Formulations for A MILP Infeasibility Relaxation Model **14**

B Details for Our Infeasible MILP Benchmark Generation **15**

C Enhanced Input Features **16**

D Network Architecture and Training Details **18**

D.1 Input Representation and Encoding 18

D.2 Attention Layers and Readout 19

D.3 Coverage Head (Deep Sets Decoder) 19

E Support preservation: formulations and proofs **19**

E.1 Quality-Weighted Binary Classification and Support Coverage 19

E.2 Ranked Prefixes and the Ranking Surrogate 21

F Additional Results **22**

F.1 Full Results of Reflection-Based Parameter Search 22

F.2 Impact of Tripartite Loss Weights 22

F.3 Full Results of Ablation Studies 23

F.4 More Results on Different Inference Strategy 23

F.5 Overhead Discussion 23

F.6 Distributional Shift and Generalization to MIPLIB 23

F.7 Case Study 24

A Different Formulations for A MILP Infeasibility Relaxation Model

Equation (2) leaves the violation cost $F(\cdot)$ abstract by design: it mathematically encodes a modeler's definition of acceptable violations. Consequently, different choices of $F(\cdot)$ yield fundamentally different relaxations for the exact same infeasible model. Standard elastic relaxation tools (e.g., Gurobi's feasRelax and CPLEX's FeasOpt) typically support three cost formulations: total violation, the number of violated constraints, and squared violation. While they share identical feasible regions, their distinct objectives drive entirely different relaxation behaviors. We evaluate our framework on the first two as representative paradigms, detailing the third for completeness. Among them, total violation is our primary setting (Appendix B), complemented by the constraint count cost (Section 5). Notably, our learning formulation (Section 4.1) remains strictly invariant across both. Throughout, $\mu_i > 0$ denotes the penalty assigned to constraint i. In our analysis, we uniformly set $\mu_i = 1$ to isolate the impact of the violation aggregation mechanism from individual constraint weighting.

The Common Elastic Reformulation. Let $\mathbf{a}_i^\top$ denote the i-th row of $\mathbf{A}$. To accommodate mixed-sense constraints, Eq. (2) introduces a single non-negative slack variable $s_i \geq 0$ to absorb the violation of each corresponding constraint in Eq. (1):

$$\mathbf{a}_i^\top \mathbf{x} \leq b_i + s_i, \qquad \mathbf{a}_i^\top \mathbf{x} \geq b_i - s_i, \qquad -s_i \leq \mathbf{a}_i^\top \mathbf{x} - b_i \leq s_i. \tag{10}$$

We note that he original variable bounds and integrality conditions on $\mathbf{x}$ are strictly retained. For any violation cost F that is strictly increasing in $\mathbf{s}$, the optimal slack s_i exactly matches the realized violation. Consequently, the support $S = \text{supp}(\mathbf{s})$ precisely identifies the set of relaxed constraints. By restricting elasticity entirely to the constraints, we align the label space with the constraint-based candidate set C introduced in Section 4.1. Finally, because Proposition 1 requires only a finite V^* and attainable optima for a generic F, its theoretical guarantees for feasibility and optimality preservation hold verbatim for all three cost formulations discussed below.

Total Violation. The first formulation minimizes the absolute sum of violations:

$$F_{\text{sum}}(\mathbf{s}) = \sum_{i=1}^{m} \mu_i s_i. \tag{11}$$

As the default objective in standard elastic solvers, it assumes that violation is *fungible* with a constant marginal cost. Since this cost is inherently indifferent to how the violation is spread, the optimal support is rarely unique. Instead, near-optimal supports of very different shapes are cheaply available from the solver's solution pool: the same instance may be relaxed by granting one large overrun or by trimming several demands at the exact same total cost. Note that our framework explicitly exploits non-uniqueness, as it allows us to label each instance with a family of alternative feasible supports (Appendix B) rather than a single arbitrary target. This multiplicity is precisely what the complete-support ranking loss $\mathcal{L}_{\text{rank}}$ (Section 4.2) exploits: the network does not need to memorize one rigid ground truth, but can instead naturally minimize the required truncation depth by covering whichever high-quality support is ranked earliest.

Number of Violated Constraints. The second formulation penalizes the number of violated constraints, disregarding the violation magnitude (Chinneck, 2001; Fischetti & Lodi, 2008):

$$F_{\text{count}}(\mathbf{s}) = \sum_{i=1}^{m} \mu_i \mathbf{1}\{s_i > 0\}. \tag{12}$$

Since this cost is non-linear in $\mathbf{s}$, it requires introducing a binary indicator ζ_i for each constraint:

$$\min_{\mathbf{x},\, \mathbf{s}\geq \mathbf{0},\, \boldsymbol{\zeta}\in\{0,1\}^m} \sum_{i=1}^{m} \mu_i \zeta_i \quad \text{s.t.} \quad \text{Eq. (10)}, \qquad s_i \leq M_i \zeta_i, \ \forall i \in \{1, \ldots, m\}, \tag{13}$$

where M_i is a Big-M constant bounding the maximum possible violation of constraint i. While M_i is finite for bounded variable domains, extracting a tight, universally valid bound is non-trivial: underestimating M_i based on a single known relaxation can inadvertently cut off better optimal repairs. Consequently, this reformulation is an MILP with m additional binaries, making it computationally expensive. Furthermore, since the objective no longer grows with s_i once $\zeta_i = 1$, the slack of an opened constraint need not equal its realized violation; thus, the support is explicitly defined by $\boldsymbol{\zeta}$. Conceptually, this cost encodes a violation as a discrete event with a fixed penalty.

Squared Violation. The third formulation penalizes the squared violations: $F_{\text{sq}}(\mathbf{s}) = \sum_i \mu_i s_i^2$, operating over the same feasible set as Eq. (10). By imposing an increasing marginal cost, it heavily penalizes large isolated violations, forcing the solver to distribute the shortfall across multiple constraints rather than concentrating it. This least-squares approach aligns with escalating penalty structures in practice. While our model reduction remains mathematically well-defined under this cost, since satisfied constraints still yield $s_i = 0$, the resulting reformulation is a Mixed-Integer Quadratic Program (MIQP). Solving an MIQP invokes fundamentally different algorithmic paths in exact solvers compared to MILPs. Consequently, benchmarking speedups under this cost would conflate the benefits of our candidate reduction with the solver's internal MIQP heuristics. Thus, while theoretically compatible, we exclude it from our empirical evaluation.

B Details for Our Infeasible MILP Benchmark Generation

Seed classes and evolutionary operators. We start from eight classic MILP generators provided by Ecole (Prouvost et al., 2020) (capacitated facility location, combinatorial auction, fixed-charge multi-commodity network flow, generalized independent set, independent set, knapsack, maximum satisfiability, and set cover), and inject initial infeasibilities into each of them by tightening resources, demands, or coupling rows. We then use an LLM to evolve this population for up to 2,000 rounds, expanding at most 108 classes per round, where parents are sampled by code-embedding diversity and code complexity to prevent the population from collapsing onto a single template. Alongside the standard operators inherited from MILP-Evolve (Li et al., 2025b), such as crossover, mutation, formulation rewriting, topic transfer, and deletion, we introduce an *infeasibility-mutation* operator that injects distributed contradictions such as capacity cascades, bound tightening, and coupled-row shifts. This operator carries the largest sampling weight (2.0, versus 1.0 for crossover and mutation and 0.5 for deletion), since infeasibility, rather than difficulty alone, is the property we need.

Reflection-based parameter search. A compiled class does not automatically yield useful instances as default parameters often produce trivially infeasible or intractable models. While MILP-Evolve (Li et al., 2025b) tunes parameters via random grid search with predefined ranges for each parameter, it is highly inefficient in generating infeasible instances. To improve efficiency, we replace random search with an LLM that iteratively *reflects* on exact solver feedback. Operating over four rounds of four configurations, the LLM analyzes the generator source alongside a performance history of prior attempts, including parameter choices, timing, model sizes, and specific failure modes. Results in Appendix F.1 demonstrate that our reflection scheme improves the success rate by more than 5 times.

Acceptance criteria. An instance is retained only if Gurobi proves the original model infeasible and all of the following hold: 1) the infeasibility is detected within 30 seconds; 2) the subsequent relaxation takes between 30 and 400 seconds (as heavy parallelization during generation artificially inflates these times, test and validation instances are re-evaluated in isolation across three seeds for accurate difficulty stratification); 3) at most 50% of the rows are relaxed; and 4) the model has at most 10^6 rows, 10^6 columns, and 2×10^7 nonzeros.

Label extraction. For each retained instance, we solve the elastic reformulation (Eq. 2) using Gurobi FeasRelax to minimize Eq.(11) and Eq.(12), respectively. In the solver's solution pool, each solution $\mathbf{s}^{(k)}$ yields a reference support $S_k = \text{supp}(\mathbf{s}^{(k)})$ (identifying rows with slack $> 10^{-6}$) and an associated cost F_k. This constructs the label set $\{(S_k, F_k)\}_{k=1}^K$ used throughout Section 4.2 to derive $F_{\min}$ and the high-quality subset $\mathcal{K}_\delta$.

Dataset split. We retain 19,628 instances across 580 classes (Table 10). To ensure rigorous evaluation, validation and test sets are stratified evenly across three difficulty bands based on unrestricted relaxation time: easy (1-10s), medium (10-100s), and hard ($\geq$ 100s). For out-of-distribution (OOD) evaluation, the Generalized Independent Set (GISP) family is strictly isolated.

Dataset statistics. We analyze the instance statistics in Figure 5. Instances span four orders of magnitude, reaching 10^6 rows and 10^7 nonzeros (medians: 8,085 rows, 26,703 columns), with unrestricted relaxations taking a median of 54 seconds. Notably, high-quality relaxation supports are highly sparse, touching a median of only 2.4% of rows. If we switch the optimization target to Eq.(12), the average size of reference supports even drops by 75%. This extreme sparsity empirically validates our compact candidate set reduction. Furthermore, relaxation supports are non-unique: at $\delta = 5\%$, 69.1% of instances offer multiple high-quality relaxations, with 15.0% reaching the

Table 10: Dataset split. Difficulty bands are defined by unrestricted relaxation time (1–10s, 10–100s, ≥ 100s). The OOD sets isolate a single problem family strictly unseen during training.

Split	#Instances	#Classes	Per band	Distribution
Training	17,978	526	4,623/6,326/7,029	seen classes
Validation (ID)	90	77	30/30/30	seen classes
Validation (OOD)	60	35	20/20/20	unseen family
Test (ID)	900	428	300/300/300	seen classes
Test (OOD)	600	54	200/200/200	unseen family

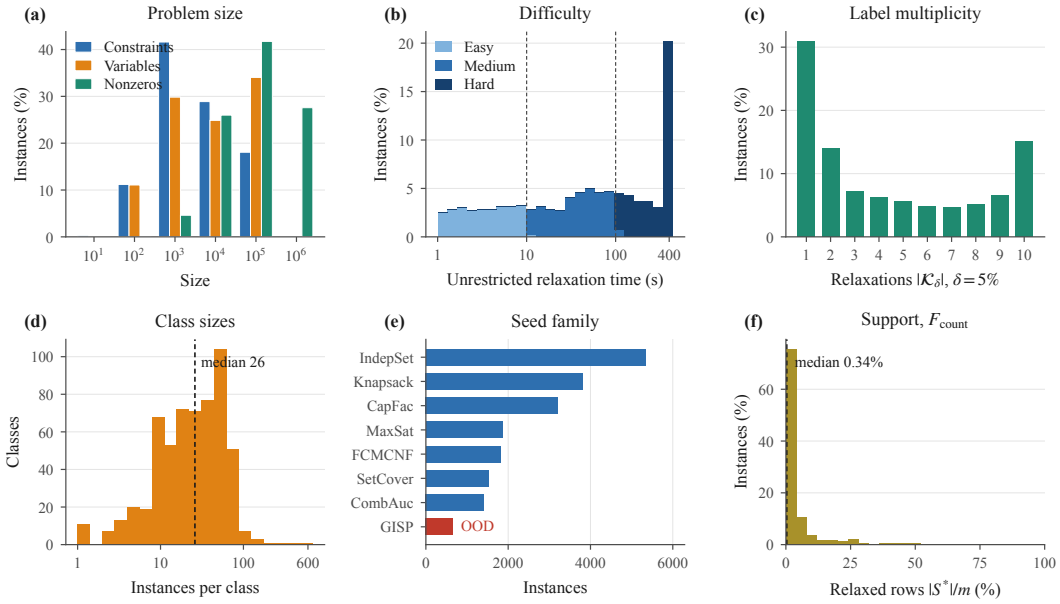

Figure 5: Dataset statistics. (a) Problem size (log scale). (b) Unrestricted relaxation time, shaded by difficulty band (dashed lines at 10s and 100s). (c) Multiplicity of high-quality labels $|\mathcal{K}_\delta|$ ($\delta = 5\%$), capped at 10 by the solution pool. (d) Instances per class. (e) Instances per seed family (OOD family GISP in red). (f) Fraction of rows relaxed by the smallest high-quality support S^* under the count cost F_{count} of Eq.(12), over the instances that record a support under that cost.

solver's pool capacity of 10. This severe multiplicity renders rigid single-target matching ineffective, justifying our set-based ranking loss.

Class genealogy and OOD scope. By tracking the lineage of each generated class, we reconstruct an evolutionary forest rooted at the eight base seeds (Figure 6). This topology is characteristically broad rather than deep. The OOD family mirrors this flat structure, as 41 of the 54 GISP classes are direct seed descendants. However, as all instances stem from these eight seeds, we strictly scope our OOD claims: we demonstrate transfer to *unseen problem classes* within known lineages, rather than zero-shot generalization to entirely novel MILP families.

C Enhanced Input Features

This section expands on the geometry-aware feature set $\mathcal{G} = \{C, \mathcal{V}, \mathcal{E}, \mathcal{I}\}$ introduced in Section 4.2 and summarized in Table 11. Beyond standard attributes (sense, right-hand side, and coefficient statistics), the constraint features f_C explicitly encode variable type composition, bound-activity intervals, objective similarity, and a dedicated set of *certificate features* that activate strictly upon structural proofs of infeasibility. The variable features $f_{\mathcal{V}}$ capture objective coefficients, declared types and bounds, alongside column-wise statistics that reflect topological constraint participation.

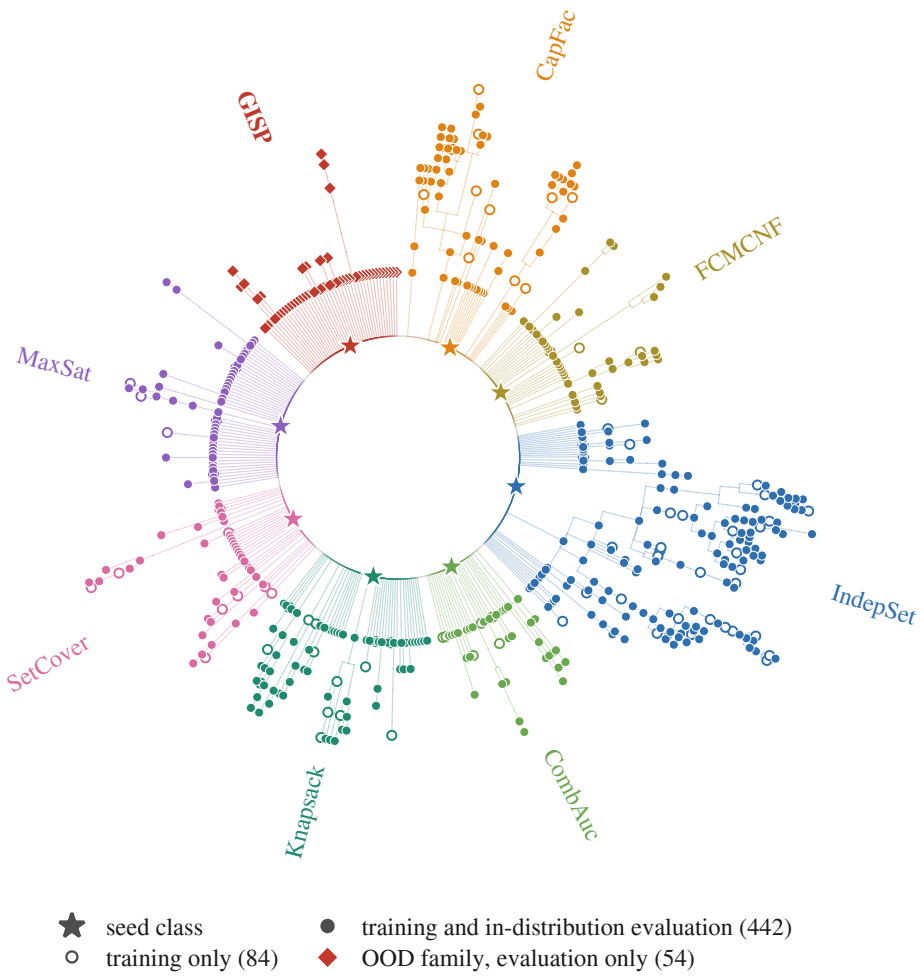

Figure 6: Genealogy of the problem classes.

Finally, the edge features $f_{\mathcal{E}}$ store row-normalized coefficients to preserve local constraint geometries, while the instance features $f_{\mathcal{I}}$ summarize the macroscopic graph scale.

Orientation. To standardize the constraint matrix $\mathbf{A}$, we convert all $\geq$ rows to the $\leq$ format ($-\mathbf{a}_i^\top \mathbf{x} \leq -b_i$). However, the *original* inequality direction (the declared sense) fundamentally encodes the modeler's intent, offering a strong predictive signal for whether a constraint is likely to be relaxed. While prior work (Gasse et al., 2019) destroys this critical prior by exclusively retaining the standardized $\leq$ format, we explicitly preserve the declared constraint sense within f_C. Similarly, for variable features $f_{\mathcal{V}}$, we compute the attributes from Han et al. (2023) using the original declared coefficients, but derive all subsequent column statistics from the standardized matrix.

Bound activity. To capture constraint tightness, f_C computes the achievable interval $[L_i, U_i]$ of each row based on its variable bounds:

$$L_i = \sum_{j \in \mathcal{N}(i)} \min_{x_j \in [l_j, u_j]} \mathbf{A}_{ij} x_j, \qquad U_i = \sum_{j \in \mathcal{N}(i)} \max_{x_j \in [l_j, u_j]} \mathbf{A}_{ij} x_j. \tag{14}$$

This interval is well-defined as long as all participating variables are bounded. If b_i falls outside this interval, the constraint cannot be satisfied on its own, and the gap exactly quantifies its violation depth. Consequently, any valid relaxation must modify this row. This provides the neural network with a strict mathematical guarantee, rather than just a statistical hint.

Certificates. Conflict and domination features identify structural contradictions among parallel constraints. By mapping each row to a primitive integer vector $\mathbf{q}$, we project constraints sharing this

key onto a common scalar $\mathbf{q}^\top \mathbf{x}$, effectively reducing them to 1D intervals. An empty intersection among these intervals deterministically proves instance infeasibility and guarantees support inclusion. Conversely, a constraint is flagged as dominated if relaxing it yields no expansion of the group's feasible interval. To eliminate false positives from numerical noise, all bounding operations and Diophantine equality checks are executed in exact rational arithmetic, rendering these certificates mathematically airtight.

Edges and instance. Raw coefficients $\mathbf{A}_{ij}$ can mislead the network if constraints are scaled arbitrarily (e.g., $2x \leq 4$ vs. $x \leq 2$). To enforce scale invariance, $f_{\mathcal{E}}$ normalizes edge weights by dividing by $\|\mathbf{a}_i\|_2$. This ensures the network evaluates a constraint's true geometric shape rather than its units. Furthermore, instance features $f_I = \{\log(1 + |\mathcal{C}|), \log(1 + |\mathcal{V}|), \log(1 + |\mathcal{E}|)\}$ simply capture the graph dimensions, using a log scale to handle drastically varying problem sizes. Importantly, this entire feature set is extracted directly from the static inputs $(\mathbf{A}, \mathbf{b}, \mathbf{c})$ and variable bounds in $O(|\mathcal{E}|)$ time. This guarantees that constructing our geometry-aware graph $\mathcal{G}$ incurs no expensive solver operations.

Table 11: Input features of the bipartite graph. Notations: $\mathcal{N}(\cdot)$ denotes the non-zero incidence neighborhood, $\text{slog}(z) = \text{sign}(z)\log(1+|z|)$, and $[L_i, U_i]$ is the bounding envelope of row i. Constraints are standardized to $\leq$ format with the rewritten RHS denoted as b_i; original orientations are retained as the declared sense. All features are extracted statically in $O(|\mathcal{E}|)$ time without solver calls.

Group	Features
Constraint node i	
Sense	Declared sense (one-hot) and a scalar sense code
Right-hand side	$\log(1 + \lvert b_i\rvert)$; $\text{sign}(b_i)$; normalized bias $b_i/\|\mathbf{a}_i\|_2$ and its instance-wise percentile
Support	Degree $\log(1 + \lvert\mathcal{N}(i)\rvert)$; normalized degree $\log(1 + \lvert\mathcal{N}(i)\rvert) - \log(1 + n)$
Coefficients	$\log(1+\|\mathbf{a}_i\|_\infty)$; $\log(1+\text{median}_j\,\lvert a_{ij}\rvert)$; slog of mean, min, and max a_{ij}; fraction of positive a_{ij}; log ratio of the largest to second-largest $\lvert a_{ij}\rvert$
Variable types	Fractions of binary/integer variables in $\mathcal{N}(i)$; log mean $\lvert a_{ij}\rvert$ per variable type (binary, integer, continuous); binary indicator for the max $\lvert a_{ij}\rvert$
Objective	$\cos(\mathbf{a}_i, \mathbf{c})$
Bound activity	Tightness $\text{clip}_{[0,1]}((b_i - L_i)/(U_i - L_i))$; redundant ($U_i \leq b_i$); contradictory (interval misses b_i); finiteness indicators for L_i, U_i
Certificates	Violation depth $\log(1 + d_i/\|\mathbf{a}_i\|_2)$ where d_i is the exact shortfall; parallel conflict (flag & depth); parallel domination; Diophantine contradiction (gcd test)
Node type	Objective node indicator ($\mathbf{c}$ enters as an extra constraint node)
Variable node j	
Han et al. (2023)	On declared coefficients: clipped c_j, column degree, integrality, and mean/min/max of a_{ij}. (All min-max scaled to $[10^{-5}, 1]$)
Gasse et al. (2019)	Type (one-hot); normalized obj $c_j/\|\mathbf{c}\|_2$; bound finiteness indicators for l_j, u_j
Column statistics	On rewritten coefficients: degree $\log(1 + \lvert\mathcal{N}(j)\rvert)$; $\text{slog}(\min_i a_{ij})$; $\log(1 + \max_i \lvert a_{ij}\rvert)$; $\log(1 + \sum_i \lvert a_{ij}\rvert)$; fraction of positive a_{ij}. On declared senses: fractions of $\geq$ and $=$
Global	$\log(1 + m)$, $\log(1 + n)$, $\log(1 + \text{nnz}(\mathbf{A}))$
Edge (i, j)	$a_{ij}/\|\mathbf{a}_i\|_2$

D Network Architecture and Training Details

This section expands on the constraint-centric network g_θ introduced in Section 4.2, detailing the forward pass, layer configurations, and architectural hyperparameters. Table 12 summarizes the default network dimensions and training configurations.

D.1 Input Representation and Encoding

Following Gasse et al. (2019), the objective function $\mathbf{c}$ is appended to the constraint matrix $\mathbf{A}$ as one additional constraint node, making the total number of constraint nodes $|\mathcal{C}| = m + 1$. This objective node participates in all attention mechanisms but is explicitly masked out during the final score readout and coverage prediction, leaving the output scores $\boldsymbol{z} \in \mathbb{R}^m$.

Feature normalization. As introduced in Eq.(4), initial embeddings $h_{\mathcal{C}}^0$ and $h_{\mathcal{V}}^0$ are projected to a hidden dimension d. For constraint features $f_{\mathcal{C}}$, the scalar sense code and the one-hot sense indicators

are inherently bounded, while other statistics vary by orders of magnitude. To prevent the latter from dominating, layer normalization (LN_C^0) is applied block-wise, and the results are concatenated before the MLP projection. All MLPs in the encoder utilize ReLU activations.

Edge encoding. Edeg features are crucial for bipartite interactions. Normalizing a single scalar is mathematically degenerate, so we deterministically expand the row-normalized coefficient into three scale-invariant channels without introducing learnable parameters:

$$f_{\mathcal{E}}(e_{ij}) = \left[\mathrm{sgn}(\tilde{a}_{ij}), \log(1+|\tilde{a}_{ij}|), \mathrm{sgn}(\tilde{a}_{ij})\log(1+|\tilde{a}_{ij}|)\right], \quad \text{where} \quad \tilde{a}_{ij} = \frac{\mathbf{A}_{ij}}{\|\mathbf{a}_i\|_2}. \tag{15}$$

D.2 Attention Layers and Readout

Each message-passing layer ℓ (width d, H_a heads) operates in three sequential phases:

Kernel linear self-attention. Following Huang et al. (2026), the intra-type self-attention bypasses the $O(N^2)$ dense attention matrix. For a stream $\mathcal{S} \in \{C, \mathcal{V}\}$ with embeddings h_u, queries and keys are mapped strictly positive via a customized activation $\sigma(\cdot)$, while values remain unconstrained:

$$q_u = \sigma(W_Q h_u), \quad k_u = \sigma(W_K h_u), \quad v_u = W_V h_u. \tag{16}$$

The output is computed using an efficient associative property, normalized by the degree, and closed with a residual connection and layer normalization to yield $h_{\mathcal{S}}^{\ell(0)}$.

Sparse cross-attention. The bipartite cross-attention aggregates information strictly over the incidence neighborhood $\mathcal{N}(\cdot)$. Notably, the edge features $f_{\mathcal{E}}$ enter as an *additive bias* to both keys and values, allowing edge semantics to shift neighbor contributions without squashing them multiplicatively. Both constraint and variable streams read from the same state $h^{\ell(0)}$ independently to produce $h^{\ell(1)}$. This is followed by a position-wise Feed-Forward Network (FFN) with hidden width d_{ff}.

Constraint-centric hop & Readout. As defined in the main text, the asymmetric hop explicitly updates $h_C^{\ell(2)}$ by attending to the newly updated variables $h_{\mathcal{V}}^{\ell(1)}$. This state bypasses the FFN and directly feeds the readout. The fusion MLP concatenates the initial, intermediate, and final constraint representations to combat over-smoothing, outputting the final pre-sigmoid score $z_i \in \mathbb{R}$.

D.3 Coverage Head (Deep Sets Decoder)

The coverage head predicts the truncation log-ratio $\hat{t} = \log \hat{\rho}^{\theta}$. To ensure shift- and scale-invariance, the raw scores z_i are first standardized instance-wise using their median and median absolute deviation:

$$u_i = \frac{z_i - \mathrm{med}(z)}{\max\{\mathrm{med}_j |z_j - \mathrm{med}(z)|, \epsilon_D\}}. \tag{17}$$

These normalized scores are processed by a point-wise MLP ϕ, permutation-invariantly aggregated (mean-pooled), and concatenated with global side features b_I (comprising instance scale $\log(|C|), \log(|\mathcal{V}|), \log(|\mathcal{E}|)$ and structural certificate firing counts). Finally, a top-level MLP outputs $\hat{t}$:

$$\hat{t} = \mathrm{MLP}_{\mathrm{decoder}}\left(\left[\frac{1}{m}\sum_{i=1}^{m}\phi(u_i); b_I\right]\right). \tag{18}$$

E Support preservation: formulations and proofs

This section provides a theoretical justification for the complete-support ranking loss. We first show that quality-weighted binary classification alone can yield rankings with poor support-coverage depth, even when the soft targets are predicted exactly. We then establish how the ranking surrogate relates to this discrete depth.

E.1 Quality-Weighted Binary Classification and Support Coverage

Our quality-weighted binary classification loss aggregates all recorded feasible relaxations, assigning their contributions according to solution quality. However, matching these soft targets does not

Table 12: Configuration of the network architecture and training hyperparameters.

Hyperparameter	Value
Embedding width d / Layers L	64 / 1
Attention heads H_a	2
Self-attention / Cross-attention head widths	$d_s = 64$ / $d_c = 32$
Feed-forward hidden width d_{ff}	128
Cross-attention dropout	0.05
Quality temperature T / Tolerance δ	0.25 / 0.05
Loss weights λ_{rank} / λ_{cov}	1.0 / 0.1
Numerical floors ϵ_D	10^{-3}

necessarily yield early coverage of a high-quality support. The following proposition shows that structural overlap among supports can make the resulting coverage depth arbitrarily worse than that of an optimal ranking, even when the soft targets are predicted exactly.

Proposition 2 (Soft targets need not yield early support coverage)**.** *For every integer $q \geq 2$, there exists an infeasible MILP (with total-violation cost $F(\mathbf{s}) = \sum_i s_i$) with $m = q + 4$ rows and four recorded feasible relaxations, three of which are optimal, such that the minimum high-quality support size is two. For every temperature $T > 0$, any ranking consistent with the quality-weighted soft targets of Eq. (8), computed over all four recorded relaxations, satisfies $D_{\text{any}} \geq q + 1$, whereas an optimal ranking of the same rows achieves $D_{\text{any}} = 2$. Here consistency means $y_i > y_j \Rightarrow r(i) < r(j)$. The same conclusion holds for uniform weights over all four relaxations.*

Proof. Let Z contain q distinct row indices and let e_1, e_2, d_1, d_2 be distinct indices outside Z. Define four recorded supports

$$S_0 = \{e_1, e_2\}, \qquad S_1 = Z \cup \{d_1\}, \qquad S_2 = Z \cup \{d_2\}, \qquad S_3 = Z \cup \{d_1, d_2\}.$$

Their union is $U = Z \cup \{e_1, e_2, d_1, d_2\}$, with $|U| = q + 4$. The construction below realizes relaxation costs $F_0 = F_1 = F_2 = 1$ and $F_3 = 2$, with optimal cost one. All four relaxations enter the BCE targets. Since the first three have equal costs, their normalized quality weights are equal: write $w_0 = w_1 = w_2 = a > 0$ and $w_3 = b > 0$, where $3a + b = 1$. This holds for every $T > 0$ under Eq. (8). The soft targets are

$$y_z = 2a + b \quad (z \in Z), \qquad y_{d_1} = y_{d_2} = a + b, \qquad y_{e_1} = y_{e_2} = a.$$

Every row in Z has a strictly larger target than every row outside Z. Any consistent ranking therefore places Z in the first q positions. Completing S_1 or S_2 requires at least one additional row, while completing S_0 or S_3 requires at least two. Hence $D_{\text{any}} \geq q + 1$, regardless of whether the quality threshold also admits S_3. Conversely, placing e_1, e_2 first completes the optimal support S_0 at depth two, which is minimal because every recorded support has size at least two. Uniform weights give $a = b = 1/4$ and the same ordering.

To realize these supports and costs in an infeasible instance of Eq. (2), use two binary variables η_1, η_2 and the relaxation cost $F(\mathbf{s}) = \sum_i s_i$. Assign one row per element of U:

$$\begin{aligned} \tfrac{1}{2}(1 - \eta_1 - \eta_2) \leq 0 \quad & \text{for } e_1, e_2, \\ \tfrac{1}{q+1}\eta_t \leq 0 \quad & \text{for } d_t, \quad t \in \{1, 2\}, \\ \tfrac{1}{q+1}(\eta_1 + \eta_2) \leq 0 \quad & \text{for each } z \in Z. \end{aligned}$$

The e_1, e_2 rows require $\eta_1 + \eta_2 \geq 1$, while the d_1, d_2 rows require $\eta_1 = \eta_2 = 0$, making the original instance infeasible. For each binary assignment, record the feasible relaxation with componentwise minimal nonnegative slacks. Assignments $(0, 0)$, $(1, 0)$, and $(0, 1)$ yield supports S_0, S_1, S_2, respectively, each with total slack cost one; assignment $(1, 1)$ yields S_3 with cost two. These exhaust the binary assignments, so the optimal relaxation cost is one. Thus S_0, S_1, S_2 are high-quality supports, while S_3 remains included in the BCE targets irrespective of its membership in the high-quality subset. □

The ratio between the coverage depth of a soft-target-consistent ranking and the optimal depth is at least $(q + 1)/2$, which grows unboundedly with q. The gap arises despite exact soft-target

predictions and persists when the lower-quality relaxation is included with its quality weight. Thus, quality-weighted constraint-level supervision alone does not guarantee early coverage of a high-quality support. This motivates supplementing the classification objective with a loss that explicitly targets support-coverage depth. We next characterize this depth for ranked prefixes and establish its relationship to our complete-support ranking loss.

E.2 Ranked Prefixes and the Ranking Surrogate

As established in Section 4.2, our framework aims to identify a compact, high-quality relaxation support S_k. However, evaluating the exact compactness of a predicted candidate set requires discrete sorting and truncation operations, making the true coverage depth non-differentiable. To bridge this gap, we proposed the continuous ranking surrogate loss $\mathcal{L}_{\text{rank}}$. This section formally defines the discrete coverage depth D_{any} required to extract a valid support, and mathematically proves that our surrogate loss serves as a rigorous bound and asymptotic equivalent to this true combinatorial depth.

Formally, let $r : [m] \to \{1, \dots, m\}$ denote the descending ranking of the predicted probabilities p_i (or equivalently, scores z_i) , and define the ranked prefix as $C_\kappa = \{i : r(i) \le \kappa\}$. A target support set S_k is fully covered by C_κ if and only if the prefix extends up to the lowest-ranked member of S_k. Therefore, $S_k \subseteq C_\kappa \iff \kappa \ge D_k := \max_{i \in S_k} r(i)$. Consequently, the instance-level coverage condition is satisfied if and only if $\kappa \ge \min_k D_k = D_{\text{any}}(\mathbf{p})$. Since $C_{D_{\text{any}}}$ must contain at least one valid support S_k, we have $D_{\text{any}}(\mathbf{p}) \ge |S|_{\min}$. Equality is attained by any ideal ranking that prioritizes the smallest high-quality support at the very top.

Lemma 3 (The ranking surrogate and D_{any})**.** *For any temperature $\tau > 0$ and score vector $\mathbf{z} = (z_1, \dots, z_m)$, the surrogate loss is bounded by $\mathcal{L}_{\text{rank}} \ge D_{\text{any}}(\mathbf{p})/(2m)$. Furthermore, provided no non-member of S_k exactly ties with its last-ranked member $\underline{z}_k = \min_{i \in S_k} z_i$ for all $k \in \mathcal{K}_\delta$, we have $\mathcal{L}_{\text{rank}} \to D_{\text{any}}(\mathbf{p})/m$ as $\tau \to 0$. More precisely, defining the margin $\gamma = \min_k \min_{j \notin S_k} |z_j - \underline{z}_k| > 0$ (with an empty minimum interpreted as $+\infty$), the approximation error is bounded by $|\mathcal{L}_{\text{rank}} - D_{\text{any}}(\mathbf{p})/m| \le \sigma(-\gamma/\tau)$.*

Proof. Fix a support index k and define $N_k = |\{j \notin S_k : z_j > \underline{z}_k\}|$ alongside the number of ties $T_k = |\{j \notin S_k : z_j = \underline{z}_k\}|$. In a descending sort, the last-ranked member of S_k is preceded by all rows with strictly larger scores, the other members of S_k holding the same score $\underline{z}_k$, and potentially some of the T_k tied non-members. Thus, the rank depth is bounded by $|S_k| + N_k \le D_k \le |S_k| + N_k + T_k$. Let $\Sigma_k = \sum_{j \notin S_k} \sigma((z_j - \underline{z}_k)/\tau)$, such that $m\mathcal{L}_{\text{rank}} = \min_k(|S_k| + \Sigma_k)$.

Since $\sigma(t) \ge 1/2$ for $t \ge 0$, it follows that $\Sigma_k \ge (N_k + T_k)/2$ and $|S_k| + \Sigma_k \ge D_k/2$. Taking the minimum over k establishes the global lower bound (the floor).

Assuming $\gamma > 0$, we have $T_k = 0$ and exact depth $D_k = |S_k| + N_k$. Each non-member with strictly higher score $z_j > \underline{z}_k$ contributes a term in $[1 - \sigma(-\gamma/\tau), 1]$, and each with lower score $z_j < \underline{z}_k$ contributes a term in $(0, \sigma(-\gamma/\tau)]$. Consequently, the deviation $|(|S_k| + \Sigma_k) - D_k| \le (m - |S_k|)\sigma(-\gamma/\tau) \le m\sigma(-\gamma/\tau)$ holds for all k. Since the minima of termwise-close families remain close, dividing by m yields the final error bound, which exponentially vanishes as $\tau \to 0$. □

In practice, the margin γ is computed over all non-members and can be negligible for instances with thousands of rows, making the asymptotic rate less informative than the strict limit and lower bound. Notably, this surrogate is not strictly monotone with respect to the discrete depth. For example, consider $m = 3$, a single high-quality support $S_1 = \{1\}$, and $\tau = 1$. Shifting the scores (z_1, z_2, z_3) from $(0, -0.1, -0.2)$ to $(0, 0.1, -10)$ reduces the surrogate loss from 0.642 to 0.508, even though the discrete depth D_{any} worsens from one to two: row 2 overtakes the support, but row 3 is pushed far below it, and the second effect dominates the sum. The lemma connects the ranking loss to support-coverage depth: twice the loss upper-bounds the normalized depth, and the loss converges to that depth as the temperature vanishes for fixed scores satisfying the no-cross-tie condition. *Empirically, Figure 7 validates that training with this loss improves complete-support coverage at matched candidate budgets, with the advantage growing as problem complexity increases.*

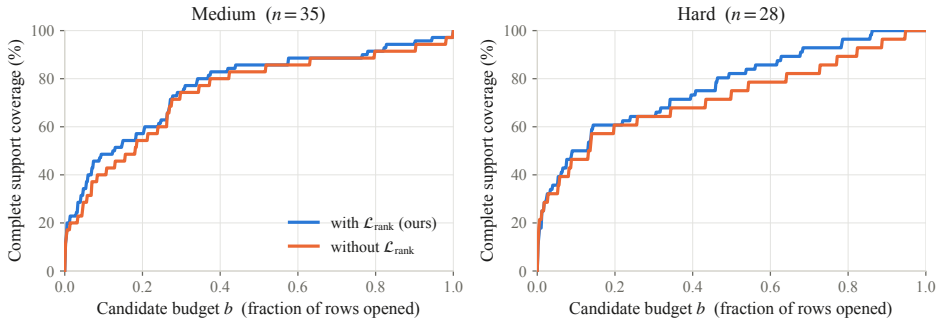

Figure 7: Complete support coverage of the network trained with and without $\mathcal{L}_{\text{rank}}$ over different fixed candidate set sizes. Results are evaluated on ID validation datasets.

F Additional Results

F.1 Full Results of Reflection-Based Parameter Search

We evaluate our reflection-based scheme on 30 random problem classes with 16 configurations and identical seeds per class. We compare standard random grid search (Li et al., 2025b) against three LLM variants: static code analysis (one-shot generation), iterative solver feedback (intra-class reflection, ours), and reflection augmented with offline cross-class rules. As Table 13 details, relying solely on static code analysis only marginally improves the configuration success rate over random search from 5.0% to 12.9% ($p = 0.34$). In contrast, iterative solver feedback significantly boosts the success rate to 27.5% ($p = 0.001$). Notably, successes accumulate steadily across the four reflection rounds from 19 to 46, proving that the LLM actively localizes the target feasible regime rather than merely sampling with better priors. Furthermore, the LLM strictly respects instance size constraints, whereas random search frequently produces unusable outliers (9 out of 24 successful configurations violated size limits). Finally, appending cross-class rules yields no further gain (dropping to 22.3%, $p = 0.18$), empirically confirming our observation that optimal tuning heuristics are highly class-specific and do not effectively transfer as generic verbal guidance.

Table 13: Full results of reflection-based parameter search. *All classes* are the 30 drawn classes; *productive classes* are the 21 in which at least one variant found a successful configuration. Success rate is over all $16n$ attempts, Successes/class is the mean per class (out of 16), and Solved class rate is the fraction of classes with at least one valid configuration.

Search method	**All classes** ($n = 30$)			**Productive classes** ($n = 21$)		
	Success% ↑	**Successes/class ↑**	**Solved class rate ↑**	**Success% ↑**	**Successes/class ↑**	**Solved class rate ↑**
Random grid search (Li et al., 2025b)	5.0%	0.80	43.3%	7.1%	1.14	61.9%
LLM, static code analysis only	12.9%	2.07	40.0%	18.5%	2.95	57.1%
LLM, iterative feedback (Cross-class rules)	22.3%	3.57	43.3%	31.8%	5.10	61.9%
LLM, iterative feedback (Intra-class, ours)	**27.5%**	**4.40**	**60.0%**	**39.3%**	**6.29**	**85.7%**

F.2 Impact of Tripartite Loss Weights

Recall that L-IR is trained via a tripartite objective: quality-weighted binary classification, complete-support ranking, and coverage regression. To prevent components with larger intrinsic numerical scales from dominating the gradients, we dynamically normalize each loss term. Specifically, the actual weight λ_k applied to loss $\mathcal{L}_k$ is scaled by its stop-gradient magnitude: $\lambda_k = \eta_k / \max(|\text{sg}(\mathcal{L}_k)|, \epsilon)$, where $\text{sg}(\cdot)$ is the stop-gradient operator, $\epsilon = 10^{-6}$, and η_k represents the targeted relative importance.

Table 14 evaluates the impact of these target magnitudes (η_1, η_2, η_3). Rather than relying on exhaustive hyperparameter tuning, our default configuration ($\eta = \{1, 1, 0.1\}$) structurally reflects the task hierarchy: it equalizes the primary classification and ranking objectives at a magnitude of 1, while down-weighting the auxiliary coverage prediction to 0.1. As the results show, this structured balancing yields the best overall scalability, achieving a commanding 4.33× speedup on Hard instances with a high solved rate of 89.3%. Breaking this hierarchy leads to suboptimal outcomes. Over-emphasizing

the classification loss slightly overfits to easier instances (Medium) at the expense of Hard-instance generalization. Conversely, arbitrarily over-weighting the ranking loss or the auxiliary coverage loss disrupts the network's training dynamics, collapsing the acceleration. This confirms that L-IR is highly robust when the loss scales properly reflect the primary-versus-auxiliary task priorities.

Table 14: Results on different weights of the three loss terms.

ID	η_1	η_2	η_3	Medium			Hard		
				Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
1	2	1	0.1	82.9%	**11.4**	**2.87**×	82.1%	63.9	3.10×
2	1	2	0.1	80.0%	18.7	1.75×	67.9%	82.8	2.39×
3	1	1	0.2	94.3%	13.2	2.47×	92.9%	57.3	3.45×
4	1	1	1	68.6%	20.0	1.64×	78.6%	68.0	2.91×
5 (Ours)	1	1	0.1	91.4%	12.8	2.55×	89.3%	**45.7**	**4.33**×

F.3 Full Results of Ablation Studies

Tables 15–18 report the complete ablation results on both Medium and Hard instances; the main text (Tables 6–9) shows the Hard columns only. The conclusions are consistent across the two difficulty bands: the full feature set, the constraint-centric encoder with the residual stream, the tripartite objective, and the instance-adaptive inference give the lowest time and the largest speedup on Hard instances, while remaining competitive on Medium instances.

F.4 More Results on Different Inference Strategy

Table 19 exposes the structural mismatch of applying standard variable-reduction baselines to infeasibility relaxation. While state-of-the-art frameworks like PaS and Apollo excel in feasible MILPs, they actively degrade solver performance here, resulting in severe computational regressions (yielding 0.96× and 0.89× slowdowns on Hard instances, respectively). Fixed Neural Diving marginally accelerates solving (1.27×) but fundamentally fractures the problem, causing the solved rate to plummet to an unacceptable 83.4%. In contrast, L-IR demonstrates overwhelming scalability. By surgically restricting the constraint space, L-IR achieves a commanding 6.08× speedup on Hard instances, nearly 5 times faster than the best surviving baseline, while robustly preserving a 94.7% solved rate.

F.5 Overhead Discussion

As detailed in Table 20, the computational overhead introduced by L-IR is negligible compared to the downstream Gurobi solver. Specifically, a forward pass of our GNN on a single RTX 4090 requires only 0.015 seconds on average (SGM), representing a mere 0.007% of the solver's time. Even when accounting for the CPU-bound static graph construction, the entire pre-solving pipeline consumes just ∼1.5% of Gurobi's baseline budget. This strictly bounds our upfront costs, confirming that the search space reduction achieved by L-IR translates seamlessly into pure end-to-end wall-clock acceleration.

F.6 Distributional Shift and Generalization to MIPLIB

While the instance generator MILP-Evolve (Li et al., 2025b) claims strong statistical alignment with MIPLIB, our analysis reveals that this alignment primarily holds for *feasible* instances. As shown in Figure 8, the natively infeasible instances in MIPLIB 2017 exhibit a starkly different structural footprint compared to our generated training set. This highlights a critical distributional gap between feasible and infeasible MILP instances. To investigate the limits of our model under this severe out-of-distribution (OOD) shift, we evaluate L-IR directly on the MIPLIB infeasible instances (Table 3). Despite the fundamentally disjoint data distributions, L-IR still extracts consistent acceleration over Gurobi, albeit modestly compared to our in-distribution results. For the objective in Eq.(11), L-IR achieves a 1.11× speedup, reducing mean time from 2885.5s to 2603.8s, while maintaining a highly competitive solved rate (67.2% vs. Gurobi's 68.8%). Under Eq.(12), L-IR yields a 1.17× speedup from 2502.9s to 2134.1s, accompanied by a solved rate drop from 51.7% to 41.7%. These results indicates a limitation of our approach: while the geometry-aware architecture is robust enough to

Table 15: Effect of the enhanced features.

ID	f_C	$f_{\mathcal{V}}$	$f_{\mathcal{E}}$	$f_{\mathcal{I}}$	Medium			Hard		
					Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
1	✓	✓	×	×	68.6%	18.1	1.80×	64.3%	79.9	2.48×
2	✓$^+$	✓$^+$	×	✓	94.3%	15.9	2.06×	96.4%	67.3	2.94×
3	✓$^+$	✓$^+$	✓	×	88.6%	14.1	2.32×	89.3%	63.2	3.13×
4 (Ours)	✓$^+$	✓$^+$	✓	✓	91.4%	**12.8**	**2.55**×	89.3%	**45.7**	**4.33**×

Table 16: Effects on encoder design.

ID	C-centric	Res.	Medium			Hard		
			Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
1	×	✓	91.4%	15.7	2.09×	85.7%	64.0	3.09×
2	✓	×	85.7%	**12.3**	**2.67**×	82.1%	60.8	3.25×
3 (Ours)	✓	✓	91.4%	12.8	2.55×	89.3%	**45.7**	**4.33**×

Table 17: Effect of the training objectives.

ID	$\mathcal{L}_{\text{BCE}}$	$\mathcal{L}_{\text{rank}}$	$\mathcal{L}_{\text{cov.}}$	Medium			Hard		
				Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
1	✓	×	×	85.7%	24.9	1.31×	78.6%	108.9	1.82×
2	×	✓	✓	82.9%	17.2	1.90×	85.7%	62.8	3.15×
3	✓	×	✓	82.9%	11.0	2.97×	75.0%	65.6	3.02×
4	✓	✓	×	91.4%	**10.5**	**3.12**×	82.1%	76.5	2.59×
5 (Ours)	✓	✓	✓	91.4%	12.8	2.55×	89.3%	**45.7**	**4.33**×

Table 18: Effect of the inference strategy.

Method	Medium			Hard		
	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
Fixed (Neural Diving)	81.0%	30.8	1.08×	84.5%	148.2	1.35×
PaS	97.1%	43.0	0.78×	**90.5%**	193.1	1.04×
Apollo	57.1%	48.7	0.68×	51.2%	180.1	1.11×
Adaptive (Ours)	**91.4%**	**12.8**	**2.55**×	89.3%	**45.7**	**4.33**×

Table 19: Inference strategies on the Dual-Attention baseline (Huang et al., 2026) using the ID test dataset. The reduction column says what the prediction restricts: constraints (C) or variables ($\mathcal{V}$).

Methods	Easy			Medium			Hard		
	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑	Solved ↑	Time ↓	Speedup ↑
Gurobi	100.0%	4.24	1.00×	100.0%	34.93	1.00×	98.5%	214.70	1.00×
Fixed (C)	75.9%	3.09	1.37×	80.3%	20.41	1.71×	79.1%	94.62	2.27×
Fixed / Neural Diving ($\mathcal{V}$)	87.5%	4.36	1.05×	86.9%	34.14	1.14×	83.4%	178.24	1.27×
PaS ($\mathcal{V}$)	98.0%	4.62	0.92×	99.1%	40.24	0.87×	96.3%	224.48	0.96×
Apollo ($\mathcal{V}$)	96.8%	14.16	0.30×	93.3%	96.40	0.36×	90.2%	242.14	0.89×
L-IR (Ours)	94.5%	**2.43**	**1.74**×	96.0%	**10.90**	**3.21**×	94.7%	**35.34**	**6.08**×

generalize and accelerate unseen distributions, closing the performance gap on MIPLIB ultimately requires future work to align the generator's distribution.

F.7 Case Study

To investigate the physical meaning of our predicted relaxation under the definition of Eq.(11), we take an urban transport model with $m = 20{,}724$ rows as an example. A conflict arises between 180 route assignment requirements ($\sum_z x_{r,z} \geq 1$) and 24 zone carbon caps ($\sum_r e_r x_{r,z} \leq C_z$): the routes emit 52,448 in total against a combined cap of 36,027. Because assigning even the cleanest routes admits

Table 20: Overhead breakdown of the components against the solver, on the unseen in-distribution hard instances. *Graph construction* builds the bipartite graph; *Model prediction* is one forward pass of our GNN on a single RTX 4090; *Gurobi* is the feasRelax solve. All times are in seconds.

	Graph construction	**Model prediction**	**Gurobi**
1-shifted geometric mean	3.12	0.015	208.22
Median	2.12	0.012	207.68
P90	22.06	0.119	372.47
Max	93.41	0.360	400.56
Share of Gurobi (SGM)	1.50%	0.007%	–

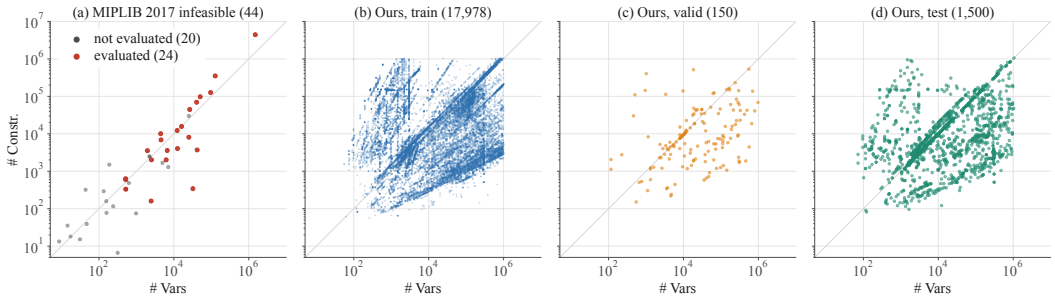

Figure 8: Instance statistics from different data sources.

at most 143 routes, at least 37 routes must be dropped. The relaxation resolves this by relaxing exactly 37 assignment rows by one unit each ($\sum_i s_i = 37$), sharply retiring the dirtiest fifth of the network (emissions ≥ 403.6) instead of buying additional carbon budget (see Figure 9 for a toy illustration).

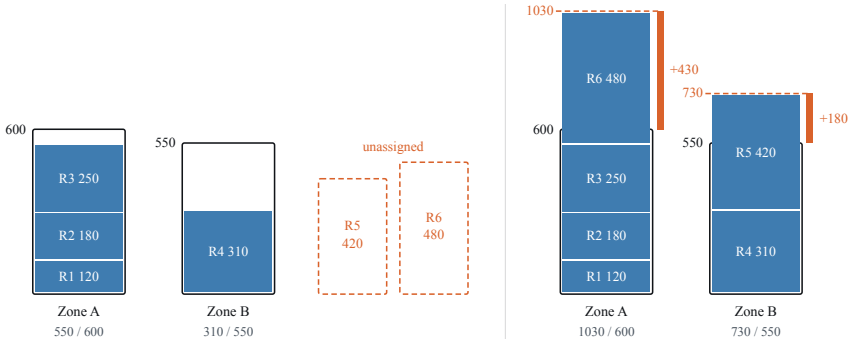

Figure 9: A simplified version of an infeasible urban transport model. The routes emit 120 to 480 (1,760 in total) and the two zones cap emissions at 600 and 550 (1,150 in total), so no assignment is feasible. Relaxing the two assignment rows of the dirtiest routes costs a slack of 2 (left), while operating every route and raising both caps costs a slack of 610 (right). Slack is drawn in orange.