LEARNING TO ALLOCATE SOLVER FREEDOM: ADAPTIVE REOPTIMIZATION FOR MIXED-INTEGER PROGRAMMING

Anonymous authors
Paper under double-blind review

ABSTRACT

Can correction signals learned on smaller mixed-integer programs (MIPs) guide reoptimization at million-variable scale? Given a feasible anchor, we learn which integer decisions to release and adapt the amount of solver freedom to each instance. We study this question through *anchor-conditioned reoptimization*. A quality–freedom frontier characterizes the tradeoff between attainable solution quality and the number of anchor decisions left free. A lightweight gradient-boosted decision tree ranks variables using structural, root-LP, and anchor-conditioned features; a cumulative-score rule selects the free set, and a MIP solver jointly optimizes the released variables and continuous decisions. We derive a correction-coverage bound under marginal correction probabilities and show why high partial recall alone cannot guarantee objective improvement. Across three structurally distinct MIP families, our method improves gap–time tradeoffs and reduces mean optimality gaps by 14%–94% on larger instances relative to matched-budget unrestricted solving. On MILPBench, family-specific predictors trained using feasible references on 20,000–40,000-variable instances transfer to problems with up to two million variables without target-scale retraining or retuning, achieving competitive objective values in 175–397 seconds of measured online runtime. The framework operates entirely on CPUs without GPU acceleration, providing a practical and scalable interface between learning and MIP solving.

1 INTRODUCTION

Mixed-integer programming (MIP) underlies discrete decision making in scheduling, allocation, network design, and many other applications (Wolsey, 1998). Modern solvers often find useful feasible solutions well before completing optimization or certification. This raises a practical question: once such a solution is available, how should limited computation be allocated to improve it, and can correction signals learned on smaller instances remain useful at much larger scales?

Classical and learned neighborhood methods improve incumbents by restricting subsequent search (Fischetti & Lodi, 2003; Danna et al., 2005; Fischetti & Monaci, 2014; Song et al., 2020; Wu et al., 2021; Sonnerat et al., 2021; Huang et al., 2023; Yuan et al., 2025). Their effectiveness depends on two coupled choices: *which* decisions to reconsider and *how many*. We study these choices through *anchor-conditioned reoptimization*. Given a feasible anchor x^A, a *quality–freedom frontier* characterizes the minimum number of anchor decisions that must remain free to attain a target objective quality. At optimality, this yields a minimum correction support; more generally, any better feasible reference provides a sufficient correction support for training.

We turn this formulation into a lightweight learned policy (Figure 1). A gradient-boosted decision tree ranks integer variables by correction evidence, and a cumulative-score rule adaptively determines the release-set size. The remaining integer variables are fixed to the anchor, while the MIP solver jointly optimizes the released variables and continuous decisions. Learning therefore allocates solver freedom rather than predicting replacement values. We also provide a coverage interpretation of cumulative correction mass and show why high partial correction recall alone need not imply objective improvement.

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.

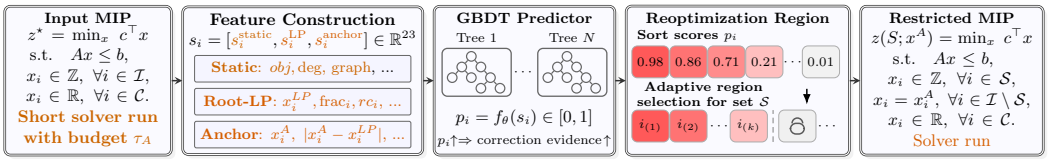

Figure 1: Anchor-conditioned reoptimization. A short solver run produces a feasible anchor. A lightweight predictor ranks integer variables by correction evidence, cumulative correction mass selects the free set, and an off-the-shelf solver optimizes the resulting restricted MIP.

Across three controlled MIP families, the method improves gap–time tradeoffs and reduces mean gaps by 14%–94% under larger-scale transfer settings relative to matched-budget unrestricted solving. On MILPBench, family-specific predictors trained on 20,000–40,000-variable instances using feasible-reference supervision transfer to problems with up to two million variables without target-scale retraining or retuning.

Our contributions are: (i) a quality–freedom characterization and coverage analysis linking correction supports to attainable quality; (ii) a lightweight learned policy combining correction ranking with adaptive release-set sizing; and (iii) empirical evidence of cross-scale transfer from smaller training instances to million-variable MIPs.

2 Problem Formulation

2.1 Anchor-Conditioned Reoptimization

We consider a mixed-integer linear program in minimization form,

$$z^\star = \min_x \; c^\top x \quad \text{s.t.} \quad Ax \le b,\; x_i \in \mathbb{Z}\;\forall i \in \mathcal{I},\; x_i \in \mathbb{R}\;\forall i \in \mathcal{C}, \tag{1}$$

where $\mathcal{I}$ and $\mathcal{C}$ index integer and continuous variables. Maximization is handled by negating the objective. We assume that the MIP has a finite, attainable optimum. Let $\mathcal{X}$ be the feasible set and $x^A \in \mathcal{X}$ a feasible *anchor* obtained from a short solver run. For a subset $S \subseteq \mathcal{I}$, we retain solver freedom only on the integer variables in S and fix all remaining integer variables to their anchor values:

$$\mathcal{X}(S; x^A) = \left\{ x \in \mathcal{X} : x_i = x_i^A,\; \forall i \in \mathcal{I} \setminus S \right\}. \tag{2}$$

The resulting restricted objective is

$$z(S; x^A) = \min_{x \in \mathcal{X}(S; x^A)} c^\top x. \tag{3}$$

All continuous variables remain free. Because $x^A \in \mathcal{X}(S; x^A)$ for every S, every restricted problem remains feasible and satisfies $z^\star \le z(S; x^A) \le c^\top x^A$. Moreover, if $S_1 \subseteq S_2$, then $\mathcal{X}(S_1; x^A) \subseteq \mathcal{X}(S_2; x^A)$, and therefore $z(S_2; x^A) \le z(S_1; x^A)$. Thus, releasing additional variables may increase computational cost, but cannot worsen the *best attainable* restricted objective.

2.2 Quality–Freedom Frontier

We formalize the relationship between retained solver freedom and attainable solution quality. For a free-variable budget $k \in \{0, \ldots, |\mathcal{I}|\}$, define

$$F(k; x^A) = \min_{S \subseteq \mathcal{I},\, |S| \le k} z(S; x^A), \tag{4}$$

which is the best objective attainable when solver freedom is retained on at most k integer variables. Conversely, for a target objective quality $\beta \ge z^\star$, define

$$K(\beta; x^A) = \min_{S \subseteq \mathcal{I}} \left\{ |S| : z(S; x^A) \le \beta \right\}, \tag{5}$$

which measures the minimum solver freedom required to make a solution meeting target quality β reachable. For integer assignments, define the Hamming distance from the anchor as

$$d_H(x_\mathcal{I}, x_\mathcal{I}^A) = \sum\nolimits_{i \in \mathcal{I}} \mathbf{1}[x_i \ne x_i^A]. \tag{6}$$

The two frontier functions admit equivalent solution-space characterizations,

$$F(k; x^A) = \min_{x \in \mathcal{X},\, d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}) \le k} c^\top x, \tag{7}$$

$$K(\beta; x^A) = \min_{x \in \mathcal{X},\, c^\top x \le \beta} d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}). \tag{8}$$

These expressions identify the best quality reachable within a Hamming-distance budget and the nearest feasible solution meeting a target quality, respectively. Consequently,

$$F(k; x^A) \le \beta \quad \Longleftrightarrow \quad K(\beta; x^A) \le k. \tag{9}$$

Thus, F and K provide complementary views of a *quality–freedom frontier*: one asks what quality is attainable under a freedom budget, while the other asks how much freedom is needed for a target quality. Derivations are provided in Appendix A. The frontier also connects our formulation to classical neighborhood search. For binary MIPs, $F(k; x^A)$ coincides with the optimal objective of radius-k Local Branching around x^A (Fischetti & Lodi, 2003). For general integer variables, Equation (7) gives the analogous coordinate-wise Hamming neighborhood. Whereas Local Branching restricts the number of deviations from the anchor, our method learns a variable-specific release set and adaptively determines its size. Importantly, the frontier characterizes attainable quality rather than finite-time solver performance. A larger free set may be computationally harder to optimize despite having a no-worse restricted optimum. The frontier therefore serves as an ideal characterization of solver-freedom allocation, rather than a procedure that our method computes explicitly.

2.3 Minimum Correction and Practical Supervision

At the optimality endpoint $\beta = z^\star$, let $\mathcal{X}^\star$ denote the set of globally optimal solutions, and select an optimal solution nearest to the anchor:

$$x^{\star,A} \in \arg\min_{x \in \mathcal{X}^\star} d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}). \tag{10}$$

Define its *minimum correction support* as

$$D_{\min}(x^A) = \left\{ i \in \mathcal{I} : x_i^{\star,A} \neq x_i^A \right\}. \tag{11}$$

When multiple nearest optimal solutions exist, $x^{\star,A}$ denotes any one of them. Different nearest optima may induce different minimum-cardinality supports, but all such supports have cardinality $K(z^\star; x^A)$ and are equally valid oracle targets. By Equation (8),

$$|D_{\min}(x^A)| = K(z^\star; x^A). \tag{12}$$

Hence $D_{\min}$ is a minimum-cardinality release set that preserves at least one globally optimal solution. Every $S \supseteq D_{\min}(x^A)$ satisfies $z(S; x^A) = z^\star$, whereas any release set with $|S| < |D_{\min}(x^A)|$ cannot preserve a global optimum. Exact $D_{\min}$ provides an ideal, optimization-grounded correction target, but it is not required by the learning framework. Given any feasible reference solution $x^R \in \mathcal{X}$, define its correction support relative to the anchor as

$$D_R(x^A) = \left\{ i \in \mathcal{I} : x_i^R \neq x_i^A \right\}. \tag{13}$$

Because x^R remains feasible when variables outside D_R are fixed to the anchor, $z(D_R; x^A) \le c^\top x^R$ and $K(c^\top x^R; x^A) \le |D_R(x^A)|$. A feasible reference therefore supplies a sufficient, possibly nonminimal, release set for recovering a solution no worse than x^R. Even an optimal reference need not be nearest to the anchor. In the experiments using synthetic MIP benchmarks (Gasse et al., 2019), we use exact minimum-correction labels

$$y_i = \mathbf{1}[i \in D_{\min}(x^A)], \tag{14}$$

whereas the million-scale MILPBench experiments use reference-derived labels

$$y_i^R = \mathbf{1}[i \in D_R(x^A)]. \tag{15}$$

This separates the theoretical role of $D_{\min}$ from the practical requirements of training. Rather than computing the ideal frontier or minimum correction support at inference time, our method learns variable-level correction evidence and uses the cumulative mass of predicted correction scores to allocate solver freedom adaptively.

3 METHOD

3.1 ANCHOR AND VARIABLE REPRESENTATION

Each integer variable has features $s_i = [s_i^{\text{static}}, s_i^{\text{LP}}, s_i^{\text{anchor}}]$. The controlled representation has 23 components: 15 static, 6 root-LP, and 2 anchor-conditioned features (Appendix B). Static features describe objective coefficients, bounds, constraint incidence, neighboring variables, PageRank, instance size, and three family indicators. LP features include the relaxation value, fractionality, confidence, reduced cost, and bound indicators. Anchor-conditioned features are x_i^A and $\Delta_i^{A,\text{LP}} = |x_i^A - x_i^{\text{LP}}|$, allowing scores to depend on the anchor and its relation to the LP solution.

3.2 LEARNING THE FREEDOM RANKING

We train a GBDT classifier using Equation (14) or (15). Its positive-class score $p_i = f_\theta(s_i) \in [0, 1]$ indicates correction evidence. Variables are ranked as $p_{(1)} \geq p_{(2)} \geq \cdots \geq p_{(|\mathcal{I}|)}$. The scores guide release rather than replacement values. The selection rule uses their relative magnitudes; calibrated probabilities are not required to execute it. Model settings are in Appendix C.

3.3 ADAPTIVE SOLVER-FREEDOM ALLOCATION

To adapt region size to the score distribution, we retain a specified fraction of total correction mass. For a global correction-mass threshold $\rho \in (0, 1]$ and positive total score mass, define

$$k_\rho = \min \left\{ k : \frac{\sum_{j=1}^{k} p_{(j)}}{\sum_{i \in \mathcal{I}} p_i} \geq \rho \right\}, \tag{16}$$

and construct the free-variable set $S_\rho = \{i_{(1)}, \ldots, i_{(k_\rho)}\}$. If the total score mass is zero, the implementation releases a single variable using deterministic tie-breaking. Concentrated scores yield smaller sets than diffuse scores at the same ρ. We select ρ on development instances for the quality–time tradeoff and freeze it for evaluation. This rule is sensitive to score magnitudes, not only their ordering.

3.4 RESTRICTED REOPTIMIZATION

We fix $x_i = x_i^A$ for $i \in \mathcal{I} \setminus S_\rho$, leave all continuous variables free, and supply the anchor as a feasible warm start. The same solver used to obtain the anchor then jointly optimizes the released variables, using its native branching, cuts, and primal heuristics.

3.5 CORRECTION COVERAGE AND ITS LIMITS

The cumulative-score rule minimizes the number of released variables needed to retain a prescribed fraction of score mass. Its relation to solution quality requires an additional link between scores and correction events. Consider a random feasible reference of quality at most β, with correction support D. Conditional on the information $\mathcal{H}$ used for selection, let $q_i = \Pr(i \in D \mid \mathcal{H})$ and let S be fixed. Here the instance, anchor, and target may also be random; the reference must meet its associated target almost surely. All probabilities below are conditional on $\mathcal{H}$. No independence assumption among the correction events is required.

Theorem 1 (Coverage from marginal correction probabilities)**.** *For any finite index set and $q_i \in [0, 1]$, let $\mathcal{P}(q)$ contain all distributions on subsets D with these marginals. With $[t]_+ = \max\{t, 0\}$,*

$$\inf_{P \in \mathcal{P}(q)} \Pr_P(D \subseteq S) = \left[1 - \sum_{i \notin S} q_i \right]_+ . \tag{17}$$

For $0 \leq \delta < 1$, a smallest set ensuring coverage at least $1 - \delta$ for every $P \in \mathcal{P}(q)$ is the descending-q prefix of length

$$k_\delta = \min \left\{ k \in \{0, \ldots, |\mathcal{I}|\} : \sum_{j > k} q_{(j)} \leq \delta \right\}. \tag{18}$$

The result follows from a sharp union bound and an exchange argument; Appendix A.4 gives the proof. It is tight over abstract support distributions with the given marginals, not necessarily over supports realizable by a particular MIP. Since $D \subseteq S$ preserves the reference, the same failure bound suffices for target-quality attainability. Coverage of this particular support is not necessary: a different solution may attain the target.

For learned scores with $P_\Sigma = \sum_i p_i > 0$, set $\varepsilon_{\text{out}} = \sum_{i \notin S_\rho} |p_i - q_i|$. The selection rule implies

$$\Pr\big(z(S_\rho; x^A) > \beta \mid \mathcal{H}\big) \le \min\{1, (1-\rho)P_\Sigma + \varepsilon_{\text{out}}\}. \tag{19}$$

Thus, ρ is a score-mass fraction, not a success probability. Our implementation does not estimate the conditional score-to-marginal discrepancy ε_{out}, so Equation (19) provides an interpretation of the selection rule rather than a certified guarantee for the reported GBDTs. It concerns the restricted optimum, not finite-time solver output. Conversely, arbitrarily high partial correction recall need not yield any objective improvement: binary variables constrained to be equal cannot change unless all are released. Proposition 3 formalizes this limitation. This motivates evaluating optimization quality directly rather than inferring it from variable-level recall.

4 Evaluation

We study correction structure, adaptive solver-freedom allocation, transfer across generator settings, and scalability to million-variable MILPs.

4.1 Setup

Synthetic benchmarks. We use Set Cover, combinatorial auctions, and capacitated facility-style instances generated with standard benchmark generators (Gasse et al., 2019). For each family, 80 medium-scale instances are used for training and 20 disjoint instances are split equally between development and held-out testing. We additionally evaluate five larger-scale instances per family for transfer. Each instance uses three anchor seeds, giving 240 training runs, 30 medium test runs, and 15 transfer runs per family. Instance statistics are reported in Appendix D.

Protocol and baselines. The GBDT uses exact minimum-correction labels. Anchor budgets are 0.5 s for medium instances and 1 s for transfer instances. A single development-selected threshold $\rho = 0.875$ is frozen thereafter. Static features are precomputed, while root-LP features and inference are computed online.

We use SCIP 10.0.3 as the underlying solver for the controlled experiments and compare against seven baselines: (i) *Unrestricted Solving*; (ii) *Local Branching* (Fischetti & Lodi, 2003), which searches an anchor-centered Hamming neighborhood; (iii) *Value-Fixing*, which directly predicts variable values using the same features, GBDT class, and free-variable budget; (iv) *RINS* (Danna et al., 2005), which fixes variables using anchor–LP agreement before solving a sub-MIP; (v) *Random-LNS* (Song et al., 2020), which releases a random subset of variables for reoptimization; (vi) *ACP* (Ye et al., 2022), which constructs neighborhoods through adaptive constraint partitioning; and (vii) *EnCore-PS* (Li et al., 2026), which uses predicted early-to-final consistency within a Predict-and-Search neighborhood. Ours, Local Branching, Value-Fixing, and RINS use the same feasible anchor; other baselines follow their native incumbent-generation procedures.

Local Branching uses development-selected radii of 2%, 40%, and 70% for Set Cover, Auction, and Facility, respectively, specifying the maximum fraction of integer variables allowed to differ from the anchor. Full Solve is used only as an offline optimal reference.

Metrics and runtime. We report relative objective gap to the offline optimal reference $z^\star$ and measured online runtime

$$T_{\text{online}} = T_{\text{anchor}} + T_{\text{LP}} + T_{\text{GBDT}} + T_{\text{reduced}}, \tag{20}$$

where the terms denote anchor generation, root-LP/state construction, GBDT inference, and restricted reoptimization, respectively. Baselines are timed over the online stages they require under the same wall-clock budget. Training, static-feature preprocessing, label generation, and offline reference computation are excluded. Unless otherwise stated, results are means over test runs.

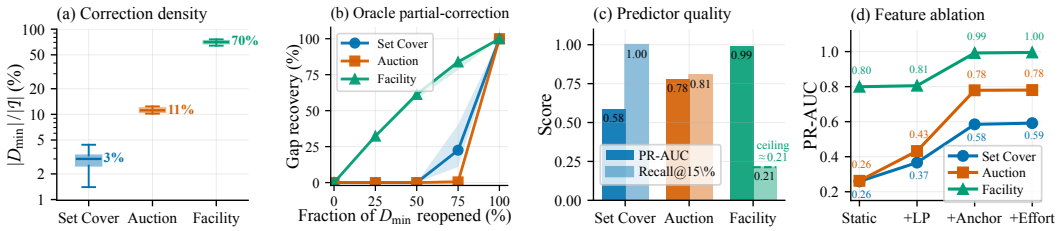

Figure 2: **Correction structure and learnability.** (a) Mean minimum-correction density across MIP families. (b) Objective improvement as increasing fractions of the oracle correction set are released. (c) Correction-variable ranking quality. (d) Predictive performance under feature ablations.

4.2 Understanding Correction Structure

Figure 2 reveals substantial heterogeneity in correction structure. The mean minimum-correction density is approximately 3% for Set Cover, 11% for Auction, and 70% for Facility. Releasing increasing fractions of the oracle correction set further reveals different correction geometries: Auction exhibits strong complementarity, whereas Facility improves more smoothly as additional correction variables are freed. Despite these differences, the anchor-conditioned predictor concentrates useful corrections near the top of its ranking. On Facility, the relatively low Recall@15% approaches the maximum attainable under a 15% free-variable budget, given its much larger correction support. Feature ablations show that root-LP and anchor-conditioned information provide substantial predictive gains. Adding offline solver-effort features, which summarize each variable's attributed branching effort and rank in a full SCIP search tree, yields little further improvement. These results support the use of the deployable 23-dimensional representation without requiring offline solver dynamics.

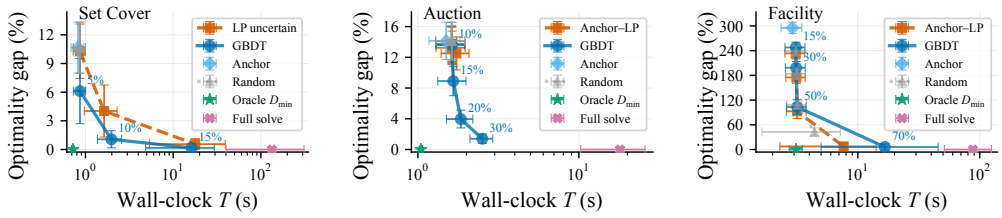

Figure 3: **Gap–time tradeoff under fixed free-variable budgets.** Mean optimality gap versus mean online wall-clock time for Set Cover, Auction, and Facility. Each marker aggregates 10 held-out development instances with three solver seeds per instance (30 runs per point). Error bars show ± 1 sample standard deviation across runs for gap and time; for Oracle $D_{\min}$ and Full Solve, only time error bars are shown. Labels indicate the fraction of integer variables kept free by GBDT-based reoptimization. Oracle $D_{\min}$ and Full Solve are offline reference points.

Figure 3 sweeps fixed free-variable budgets on development instances. Set Cover reaches roughly 1% gap at 10% freedom and approaches the optimum at 15%. Auction's gap falls from roughly 14% to below 2% as freedom increases from 10% to 30%. Facility retains a substantial gap at 50% freedom but improves sharply at 70%, with greater computation. The GBDT generally improves the tradeoff over the plotted LP heuristics, especially on Set Cover and Auction. The oracle illustrates the potential of a well-selected restricted region, but requires offline labels. The different useful free fractions motivate instance-dependent sizing.

4.3 Adaptive Reoptimization

Figure 4(a) shows that the same global threshold induces substantially different region sizes. At the development-selected $\rho = 0.875$, the mean fractions of integer variables left free are approximately 10%, 43%, and 71% for Set Cover, Auction, and Facility, respectively. These differences reflect the distribution of correction scores: concentrated scores require fewer released variables to reach the same cumulative-mass threshold, whereas diffuse scores retain a larger fraction of solver freedom.

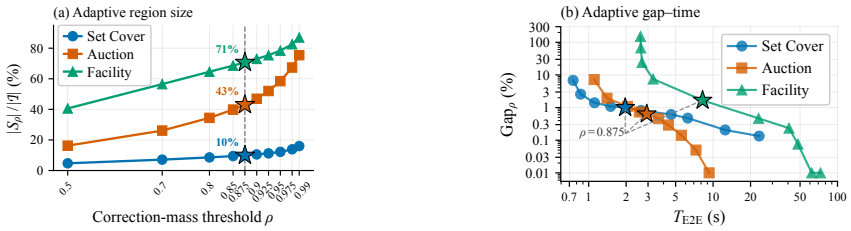

Figure 4: **Adaptive correction-mass allocation.** (a) Mean free-variable fraction as the correction-mass threshold ρ increases. (b) Mean optimality gap versus measured online runtime under different thresholds; both axes use logarithmic scales.

Table 1: **Online performance and cross-scale transfer.** Mean optimality gaps and measured online runtimes across three MIP families. Bold indicates the lowest mean gap among the online methods within each setting. We use SCIP as the underlying MIP solver.

Method	Set Cover		Auction		Facility	
	Gap (%) ↓	Time (s) ↓	Gap (%) ↓	Time (s) ↓	Gap (%) ↓	Time (s) ↓
Medium Instances						
Full Solve	0.00	394.899	0.00	38.300	0.00	174.780
Unrestricted Solve	3.94	2.274	1.02	3.103	17.26	8.265
Local Branching (Fischetti & Lodi, 2003)	3.61	2.271	1.17	3.101	64.85	8.240
RINS (Danna et al., 2005)	6.23	2.268	2.89	1.684	145.11	8.418
Value-Fixing (GBDT)	3.39	0.902	12.31	1.053	94.81	2.968
Random-LNS (Song et al., 2020)	4.91	2.195	1.05	3.047	7.66	8.430
ACP (Ye et al., 2022)	9.60	2.256	1.03	3.099	149.71	8.441
EnCore-PS (Li et al., 2026)	11.48	2.246	1.04	3.097	36.36	8.389
Ours	**1.00**	2.239	**0.44**	3.096	**1.67**	8.386
Larger-Scale Instances (Frozen Transfer)						
Full Solve	0.00	903.140	0.00	345.841	0.00	208.033
Unrestricted Solve	2.54	9.493	0.35	18.425	78.48	12.102
Local Branching (Fischetti & Lodi, 2003)	1.29	6.204	0.66	18.425	170.26	12.102
Value-Fixing (GBDT)	10.82	1.502	2.17	2.512	226.36	12.219
RINS (Danna et al., 2005)	3.17	9.546	2.43	4.313	223.15	12.180
Random-LNS (Song et al., 2020)	1.98	9.448	1.00	18.377	12.70	12.185
ACP (Ye et al., 2022)	3.79	9.532	0.97	18.431	224.83	12.201
EnCore-PS (Li et al., 2026)	21.17	9.492	0.67	18.426	120.07	12.103
Ours	**0.87**	9.488	**0.30**	18.425	**4.84**	12.095

Figure 4(b) shows the quality–time tradeoff. Appendix E compares adaptive and fixed release-set sizes while keeping the learned ranking unchanged. When each Fixed run is capped at its paired Ours reoptimization time, approximately mean-matched Fixed has mean gaps of 2.17% versus 1.00% on Set Cover and 35.97% versus 1.67% on Facility; Auction remains close. This post-hoc time control supports adaptive sizing on the former families, rather than superiority at arbitrary predefined end-to-end budgets.

Online comparison. Table 1 shows that adaptive freedom allocation improves solution quality under matched online computation budgets. Our method achieves mean optimality gaps of 1.00%, 0.44%, and 1.67% on Set Cover, Auction, and Facility, respectively, compared with 3.94%, 1.02%, and 17.26% for unrestricted solving.

Value-Fixing uses the same features, model class, and free-variable count, but fixes variables to predicted values rather than feasible anchor values. Its larger gaps, especially on Auction and Facility, favor anchor-based restriction in these experiments. Because the training target and fixed assignments also differ, this comparison does not isolate neighborhood sizing or ranking alone.

Our method achieves the best objective among the strictly matched-budget methods in all six settings. All matched baselines receive the same per-instance online wall-clock budget; neighborhood-based baselines are implemented with SCIP subproblems where applicable. These results demonstrate effectiveness under the evaluated short-budget regime rather than superiority across all configurations or longer horizons.

4.4 CROSS-SCALE TRANSFER

The GBDT and threshold are frozen after medium-scale development. The transfer settings increase Set Cover constraints from 1,000 to 2,000, Auction integer variables from 1,000 to 1,500, and Facility's continuous part while retaining 100 integer variables (Appendix D). These settings therefore test cross-scale transfer under distinct within-family structural changes, rather than a uniform increase in integer problem size or exact solving difficulty. On the five larger-scale transfer instances per family, Ours reduces mean gaps from 2.54% to 0.87% (Set Cover), 0.35% to 0.30% (Auction), and 78.48% to 4.84% (Facility) relative to matched-budget unrestricted solving. Relative reductions are 66%, 14%, and 94%; Auction's absolute gain is 0.05 percentage points. No retraining or retuning is performed for the transfer settings.

4.5 CROSS-SCALE TRANSFER TO MILLION-VARIABLE MILPS

We test within-family transfer from smaller training instances to million-variable MILPs using feasible-reference labels. SC grows from 40,000 to 2,000,000 variables, while MVC and MIS grow from 20,000 to 1,000,000. These are separately trained MILPBench models rather than transfers of the controlled-benchmark models.

Large-scale setup. We use the Set Cover (SC), Minimum Vertex Cover (MVC), and Maximum Independent Set (MIS) families from MILPBench (Ye et al., 2025a). The GBDT models are trained on easy-scale instances using reference-derived correction sets D_R rather than exact $D_{\min}$ labels. The dataset-provided reference gap estimates are approximately 4.1%, 10.3%, and 8.6% for easy-scale SC, MVC, and MIS, respectively, so training does not rely on proven-optimal reference solutions. The models are then frozen and evaluated on medium- and hard-scale instances without retraining or retuning. Instance statistics are provided in Appendix D. Each of the six evaluation settings uses five official test instances. We use Gurobi 12.0.3 as the underlying solver for all methods in Table 3.

Scalable feature representation. At million-variable scale, root-LP features may be costly and poorly discriminative. We therefore use a heuristic gate between an LP-aware GBDT and an LP-free (NoLP) companion model. NoLP is selected if no usable LP assignment is obtained or if more than half of binary variables satisfy $|x_i^{\mathrm{LP}} - 0.5| \leq 10^{-6}$. The NoLP model omits LP-derived features and anchor–LP disagreement. All medium settings and SC hard use LP-aware scoring, while MVC and MIS hard use NoLP. The easy-scale development set selects $\rho_{\mathrm{LP}} = 0.96$ and $\rho_{\mathrm{NoLP}} = 0.825$, which are frozen before medium/hard evaluation. Appendix F provides activation diagnostics.

Table 2: **Offline training cost on MILPBench.** Feature collection covers 25 easy-scale training instances per family. LP-aware and NoLP fitting times are reported separately. Feature collection is performed once per family; both GBDTs are fitted using the resulting training data.

Family	Feature collection (s)	LP-aware (s)	NoLP (s)	Total fit (s)
SC	2136.9	5.42	7.13	12.55
MVC	1564.9	5.36	3.57	8.93
MIS	1566.3	4.33	2.58	6.91

Offline training cost. Table 2 reports the cost of training the LP-aware and NoLP models on 25 easy-scale instances per family. Feature collection takes approximately 26–36 minutes per family, whereas fitting both GBDTs requires only 7–13 seconds. Most of the measured offline cost therefore lies in feature collection rather than model fitting. These costs exclude the original generation of the dataset-provided reference solutions. The same fitted models are reused for both medium- and hard-scale evaluation.

Matched-budget comparison. Table 3 compares all methods under the same local evaluation environment. Our method achieves the best objective among the strictly matched-budget methods in all six settings. The gains are particularly large on hard SC, MVC, and MIS, where our method substantially outperforms unrestricted solving, RINS, ACP, Random-LNS, and EnCore-PS under comparable online runtimes. Longer unrestricted runs obtain slightly better objectives only on medium MVC and MIS, but require substantially more computation. Literature-reported MILPBench baselines are presented separately in Appendix G for broader context. These results show that learned solver-freedom allocation provides a strong quality–time tradeoff when transferred from easy-scale training instances to much larger MILPs.

Table 3: **MILPBench performance.** Mean objective values and realized wall-clock runtimes are reported. All methods use Gurobi 12.0.3 in the same local environment. Unrestricted Solve (long budget) uses nominal 2,000/4,000 s limits; matched baselines use per-instance caps equal to the measured Ours runtime. Reported times may differ slightly from the nominal cap. EnCore-PS uses a matched downstream solver limit but is not end-to-end time matched; see Appendix C. Bold marks the best objective among strictly matched-budget methods.

	SC (medium) ↓		SC (hard) ↓		MVC (medium) ↓		MVC (hard) ↓		MIS (medium) ↑		MIS (hard) ↑	
Method	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)
Unrestricted Solve (long budget)	24289.0	2053.44	319782.2	4010.06	26793.0	2062.13	330929.0	4011.82	23154.3	2175.48	169048.2	4011.86
Unrestricted Solve (matched budget)	24289.0	245.9	320046.6	376.5	27532.6	311.7	330909.7	200.6	22610.6	397.1	169048.2	196.4
RINS (Danna et al., 2005)	24276.0	239.7	320046.6	393.9	27510.3	311.9	329671.0	188.7	22676.6	397.2	170308.3	194.1
ACP (Ye et al., 2022)	16022.3	239.3	320046.6	379.6	27095.1	311.1	330909.7	178.6	22843.4	396.7	169048.2	178.5
Random-LNS (Song et al., 2020)	16613.6	238.9	319937.2	379.8	27336.4	311.1	295605.0	177.9	22648.2	396.8	203299.5	179.6
EnCore-PS (Li et al., 2026)	24289.1	238.5	320046.6	369.8	27731.3	311.1	330911.0	175.2	22187.1	396.5	169045.7	176.0
Ours	**15937.7**	238.5	**159664.2**	369.8	**26895.8**	311.2	**274382.4**	175.2	**23081.3**	396.5	**226003.4**	176.0

5 Related Work

Learning guides branching and cutting (Khalil et al., 2016; Gasse et al., 2019; Tang et al., 2020), solution prediction (Nair et al., 2020; Han et al., 2023; Huang et al., 2024; Liu et al., 2025), and neighborhood construction (Song et al., 2020; Wu et al., 2021; Sonnerat et al., 2021; Huang et al., 2023; Yuan et al., 2025). Classical incumbent-based methods include Local Branching, RINS, and proximity search (Fischetti & Lodi, 2003; Danna et al., 2005; Fischetti & Monaci, 2014; Rodrigues et al., 2021); learned Local Branching also adapts neighborhood radii (Liu et al., 2022). Our method constructs a single restricted MIP by combining anchor-conditioned correction ranking with cumulative-score sizing of solver freedom. EnCore (Li et al., 2026) predicts early-to-final consistency with an incumbent-conditioned GNN. For a shared anchor–reference pair, its consistency labels complement our correction labels. Our emphasis is on quality–freedom characterization, minimum correction supports, cumulative-score sizing, and lightweight scale transfer. Appendix H provides additional comparison.

6 Conclusion

We studied whether correction signals learned on smaller MIPs can guide reoptimization at substantially larger scales. Anchor-conditioned reoptimization formulates this problem as solver-freedom allocation: a quality–freedom frontier characterizes how much freedom is required for a target solution quality, a lightweight GBDT ranks where that freedom is most valuable, and cumulative correction mass adapts the released set to each instance. Our analysis gives a coverage interpretation of the score-mass rule while showing why high partial correction recall alone need not imply objective improvement. Across controlled benchmarks, the resulting restricted reoptimization improves gap–time tradeoffs and transfers across scale settings. On MILPBench, family-specific predictors trained on 20,000–40,000-variable instances using feasible-reference supervision transfer to problems with up to two million variables without target-scale retraining or retuning. The resulting CPU-based pipeline demonstrates that lightweight learned solver-freedom allocation can remain effective as MIP scale increases by up to $50\times$.

Limitations. The coverage analysis depends on the discrepancy between learned scores and marginal correction probabilities and characterizes attainable restricted quality rather than finite-time solver performance. Our empirical study uses family-specific training, a limited number of large-scale test instances, and precomputed static features in the controlled timing protocol. The cross-scale experiments establish within-family transfer rather than cross-family generalization. Literature-reported MILPBench results in Appendix G follow different evaluation environments and are included for contextual comparison rather than hardware-matched runtime claims.

Reproducibility Statement

The method and online timing convention are specified in Sections 3 and 4.1. Appendix A provides the theoretical derivations and label-generation formulation; Appendices B–F document features, implementation details, benchmark configurations, sizing ablations, and the MILPBench reliability gate. Table 3 reports local MILPBench comparisons under a common Gurobi solver environment, while literature-reported results are provided separately in Appendix G.

AI Use Statement

Generative AI tools assisted with manuscript drafting and editing, and literature comparison. They also assisted with LaTeX and figure preparation. The authors are responsible for the final text, mathematical claims, experimental claims, and artifacts.

References

Emilie Danna, Edward Rothberg, and Claude Le Pape. Exploring relaxation induced neighborhoods to improve MIP solutions. *Mathematical Programming*, 102(1):71–90, 2005. doi: 10.1007/s10107-004-0518-7.

Matteo Fischetti and Andrea Lodi. Local branching. *Mathematical Programming*, 98(1–3):23–47, 2003. doi: 10.1007/s10107-003-0395-5.

Matteo Fischetti and Michele Monaci. Proximity search for 0–1 mixed-integer convex programming. *Journal of Heuristics*, 20(6):709–731, 2014. doi: 10.1007/s10732-014-9266-x.

Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. Exact combinatorial optimization with graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

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

Taoan Huang, Aaron M. Ferber, Yuandong Tian, Bistra Dilkina, and Benoit Steiner. Searching large neighborhoods for integer linear programs with contrastive learning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 13869–13890. PMLR, 2023.

Taoan Huang, Aaron M. Ferber, Arman Zharmagambetov, Yuandong Tian, and Bistra Dilkina. Contrastive predict-and-search for mixed integer linear programs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 19757–19771. PMLR, 2024.

Elias B. Khalil, Pierre Le Bodic, Le Song, George L. Nemhauser, and Bistra Dilkina. Learning to branch in mixed integer programming. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. doi: 10.1609/aaai.v30i1.10080.

Guanlin Li, Chengrui Gao, Chenguang Wang, Haopu Shang, Zherong Zhang, Ke Xue, Jixiang Lu, Weiyong Yang, and Chao Qian. Learning early-to-final solution consistency for MILP acceleration. *arXiv preprint arXiv:2608.19953*, 2026.

Defeng Liu, Matteo Fischetti, and Andrea Lodi. Learning to search in local branching. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 3796–3803, 2022. doi: 10.1609/aaai.v36i4.20294.

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

Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, Ravichandra Addanki, Tharindi Hapuarachchi, Thomas Keck, James Keeling, Pushmeet Kohli, Ira Ktena, Yujia Li, Oriol Vinyals, and Yori Zwols. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

Filipe Rodrigues, Agostinho Agra, Lars Magnus Hvattum, and Cristina Requejo. Weighted proximity search. *Journal of Heuristics*, 27:459–496, 2021. doi: 10.1007/s10732-021-09466-0.

Jialin Song, Ravi Lanka, Yisong Yue, and Bistra Dilkina. A general large neighborhood search framework for solving integer linear programs. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

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

Yunhao Tang, Shipra Agrawal, and Yuri Faenza. Reinforcement learning for integer programming: Learning to cut. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 9367–9376. PMLR, 2020.

Laurence A. Wolsey. *Integer Programming*. Wiley, 1998.

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

Huigen Ye, Hongyan Wang, Hua Xu, Chengming Wang, and Yu Jiang. Adaptive constraint partition based optimization framework for large-scale integer linear programming (student abstract). *arXiv preprint arXiv:2211.11564*, 2022.

Huigen Ye, Hua Xu, Hongyan Wang, Chengming Wang, and Yu Jiang. Gnn&gbdt-guided fast optimizing framework for large-scale integer programming. In *International conference on machine learning*, pp. 39864–39878. PMLR, 2023.

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

Huigen Ye, Yaoyang Cheng, Hua Xu, Zhiguang Cao, and Hanzhang Qin. Milpbench: A large-scale benchmark test suite for mixed integer linear programming problems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 94–103, 2025a.

Huigen Ye, Hua Xu, An Yan, and Yaoyang Cheng. Large language model-driven large neighborhood search for large-scale milp problems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 72131–72180, 2025b.

Hao Yuan, Wenli Ouyang, Changwen Zhang, Yong Sun, Liming Gong, and Junchi Yan. BTBS-LNS: Binarized-tightening, branch and search on learning LNS policies for MIP. In *International Conference on Learning Representations*, 2025.

A Supplementary Derivations

This appendix derives the quality–freedom characterizations used in Section 2.2 and establishes the sufficiency of feasible-reference correction supports used in Section 2.3.

A.1 Derivation: $F(k)$ as a Hamming-Budget Minimum

For $k \in \{0, \dots, |\mathcal{I}|\}$, define the Hamming neighborhood

$$\mathcal{B}_k(x^A) = \left\{ x \in \mathcal{X} : d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}) \le k \right\}. \tag{21}$$

We show that $\mathcal{B}_k(x^A)$ is exactly the union of all restricted feasible sets with at most k released integer variables:

$$\mathcal{B}_k(x^A) = \bigcup\nolimits_{S \subseteq \mathcal{I}, |S| \le k} \mathcal{X}(S; x^A). \tag{22}$$

Derivation of Equation (22). If $x \in \mathcal{X}(S; x^A)$ with $|S| \le k$, then only variables in S can differ from the anchor. Therefore, $d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}) \le |S| \le k$, and hence $x \in \mathcal{B}_k(x^A)$. Conversely, let $x \in \mathcal{B}_k(x^A)$ and define

$$S(x) = \left\{ i \in \mathcal{I} : x_i \ne x^A_i \right\}.$$

Then $|S(x)| = d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}) \le k$. Because x agrees with the anchor outside $S(x)$, we have $x \in \mathcal{X}(S(x); x^A)$. This establishes Equation (22). Combining Equation (22) with the definition of F gives

$$F(k; x^A) = \min\nolimits_{x \in \mathcal{X}, d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}) \le k} c^\top x. \tag{23}$$

For binary MIPs, this is the optimal objective of radius-k Local Branching around x^A.

A.2 Derivation: $K(\beta)$ as a Hamming Distance

For any $\beta \ge z^\star$, define the target-quality set

$$\mathcal{X}_\beta = \left\{ x \in \mathcal{X} : c^\top x \le \beta \right\}.$$

Select a solution nearest to the anchor:

$$x^{\beta,A} \in \arg\min_{x \in \mathcal{X}_\beta} d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}), \tag{24}$$

and define its correction support as

$$D_\beta(x^A) = \left\{ i \in \mathcal{I} : x^{\beta,A}_i \ne x^A_i \right\}. \tag{25}$$

We show that

$$K(\beta; x^A) = |D_\beta(x^A)| = \min_{x \in \mathcal{X}_\beta} d_H(x_{\mathcal{I}}, x^A_{\mathcal{I}}). \tag{26}$$

Derivation of Equation (26). By construction, $x^{\beta,A}_i = x^A_i$ for every $i \notin D_\beta(x^A)$. Thus, $x^{\beta,A} \in \mathcal{X}(D_\beta(x^A); x^A)$. Since $x^{\beta,A} \in \mathcal{X}_\beta$, we have $c^\top x^{\beta,A} \le \beta$, and therefore

$$z(D_\beta(x^A); x^A) \le c^\top x^{\beta,A} \le \beta.$$

By the definition of K,

$$K(\beta; x^A) \le |D_\beta(x^A)|. \tag{27}$$

For the reverse direction, consider any $S \subseteq \mathcal{I}$ satisfying $z(S; x^A) \le \beta$. There exists some $\bar{x} \in \mathcal{X}(S; x^A)$ with $c^\top \bar{x} \le \beta$, so $\bar{x} \in \mathcal{X}_\beta$. Because $\bar{x}$ agrees with the anchor outside S,

$$d_H(\bar{x}_{\mathcal{I}}, x^A_{\mathcal{I}}) \le |S|.$$

By the definition of $x^{\beta,A}$,

$$|D_\beta(x^A)| = d_H(x^{\beta,A}_{\mathcal{I}}, x^A_{\mathcal{I}}) \le d_H(\bar{x}_{\mathcal{I}}, x^A_{\mathcal{I}}) \le |S|.$$

Since this holds for every release set capable of attaining quality β,

$$|D_\beta(x^A)| \le K(\beta; x^A).$$

Combining both directions establishes Equation (26). The threshold equivalence

$$F(k; x^A) \le \beta \quad \Longleftrightarrow \quad K(\beta; x^A) \le k$$

follows immediately from the two solution-space characterizations: both inequalities hold precisely when there exists a feasible solution with objective at most β and Hamming distance at most k from the anchor. At the optimality endpoint $\beta = z^\star$, $\mathcal{X}_\beta = \mathcal{X}^\star$. Hence

$$K(z^\star; x^A) = \min_{x \in \mathcal{X}^\star} d_H(x_\mathcal{I}, x^A_\mathcal{I}) = |D_{\min}(x^A)|. \tag{28}$$

Although the nearest optimal solution and its correction support need not be unique, all minimum correction supports have the same cardinality.

A.3 Reference-Solution Sufficiency

The million-scale experiments replace exact minimum-correction labels with labels derived from feasible reference solutions. We establish the corresponding release-set sufficiency property. Let $x^R \in \mathcal{X}$ be any feasible reference and recall

$$D_R(x^A) = \left\{ i \in \mathcal{I} : x^R_i \ne x^A_i \right\}.$$

By definition, $x^R_i = x^A_i$ for all $i \notin D_R(x^A)$. Consequently,

$$x^R \in \mathcal{X}(D_R(x^A); x^A).$$

Since the restricted optimum cannot be worse than the objective of a feasible solution within its region,

$$z(D_R(x^A); x^A) \le c^\top x^R. \tag{29}$$

By the definition of K,

$$K(c^\top x^R; x^A) \le |D_R(x^A)|. \tag{30}$$

Thus, D_R is a sufficient, though not necessarily minimal, release set for retaining the reference objective. When the reference is optimal, D_R preserves at least one global optimum but may contain more variables than a minimum correction support. When the reference is merely feasible, the corresponding target quality is $c^\top x^R$, and the sufficiency statement applies to that quality rather than necessarily to global optimality.

These properties motivate using reference-derived correction labels when exact minimum-correction labels are impractical to obtain. They characterize the attainable objective of a restricted MIP whose release set contains the reference correction support; they do not guarantee that a learned release set will contain that support or that a time-limited solver will attain the restricted optimum.

A.4 Coverage Analysis and Proofs

Probability model. The instance, anchor, and feasible reference can be generated jointly at random. Each reference must have objective at most its associated target β almost surely. Condition on an information set $\mathcal{H}$ that determines the variable index set, scores, and release set; write $q_i = \Pr(i \in D \mid \mathcal{H})$. If $\mathcal{H}$ already determines D, these probabilities are zero or one. Otherwise, they express the uncertainty remaining after conditioning. Neither independence of correction events nor independence of the anchor and reference is assumed. A target event for a maximization problem is interpreted after negating the objective, as in Section 2.1.

Proof of Theorem 1. Fix $\mathcal{H}$ and let $J = \mathcal{I} \setminus S$. Since $D \not\subseteq S$ is the union of events $\{i \in D\}$ over $i \in J$, the union bound yields

$$\Pr(D \not\subseteq S \mid \mathcal{H}) \le \min\left\{ 1, \sum_{i \in J} q_i \right\}.$$

To establish sharpness over $\mathcal{P}(q)$, enumerate $J = \{j_1, \ldots, j_m\}$ and let $a_0 = 0$, $a_\ell = \sum_{r=1}^{\ell} q_{j_r}$. On a probability space with U uniform on $[0, 1)$, define $j_\ell \in D$ when U belongs to the image of $[a_{\ell-1}, a_\ell)$ modulo one. Each interval has length at most one, so its image has measure q_{j_ℓ}. The union is the image of $[0, a_m)$, of measure $\min\{1, a_m\}$. For $i \in S$, choose any measurable event of measure q_i, for example $\{U < q_i\}$. This constructs a distribution with all required marginals attaining the bound. The empty-J case gives coverage one directly.

For $0 \le \delta < 1$, the sharp bound is at least $1 - \delta$ if and only if $\sum_{i \notin S} q_i \le \delta$. Among sets of cardinality k, a descending-q prefix maximizes $\sum_{i \in S} q_i$ and hence minimizes the excluded mass: replacing a selected smaller score by an unselected larger one cannot increase the excluded mass. The first prefix satisfying the constraint therefore has minimum cardinality. Ties can be resolved in any fixed order. □

The extremal construction allows arbitrary support dependencies. It need not describe the correction supports available in a fixed MIP. Consequently, sharpness is an information-level statement about support coverage; it does not establish a necessary condition for preserving a particular MIP's target quality.

Corollary 2 (Quality retention with score error)**.** *Under the reference-quality assumption above, suppose $P_\Sigma = \sum_i p_i > 0$ and S_ρ is defined by Equation (16). With $\varepsilon_{\text{out}} = \sum_{i \notin S_\rho} |p_i - q_i|$, Equation (19) holds.*

Proof. Reference sufficiency gives $\{z(S_\rho; x^A) > \beta\} \subseteq \{D \not\subseteq S_\rho\}$. Furthermore,

$$\sum_{i \notin S_\rho} q_i \le \sum_{i \notin S_\rho} p_i + \varepsilon_{\text{out}} \le (1 - \rho) P_\Sigma + \varepsilon_{\text{out}}.$$

Apply the union bound and cap the resulting probability bound at one. □

Interpretation of cumulative mass. For any nonnegative scores with positive total mass, S_ρ solves

$$\min_{S \subseteq \mathcal{I}} |S| \quad \text{subject to} \quad \sum_{i \notin S} p_i \le (1 - \rho) \sum_i p_i.$$

This deterministic statement follows from the same exchange argument and requires no probability calibration. If $p_i = q_i$, the excluded mass is the expected number of missed reference corrections, and the rule also meets the corresponding marginal-only coverage constraint when $(1 - \rho) \sum_i q_i < 1$. The normalized threshold is not itself a coverage probability. For example, total mass 100 and $\rho = 0.875$ yield a failure bound capped at one even with perfect probabilities. General monotone transformations of scores preserve ranking but can change S_ρ. The result therefore explains the selection rule without supplying an unmeasured calibration or out-of-distribution guarantee.

A.5 Limits of Partial Correction Recall

Proposition 3 (High recall need not improve the objective)**.** *For any $\epsilon \in (0, 1)$ and $M > 0$, there exist a binary linear program, a feasible anchor x^A, and a release set S such that*

$$\frac{|S \cap D_{\min}(x^A)|}{|D_{\min}(x^A)|} \ge 1 - \epsilon, \qquad z(S; x^A) = c^\top x^A, \qquad z(S; x^A) - z^\star = M.$$

Proof. Choose an integer $n \ge 2$ with $1/n \le \epsilon$. Minimize $-Mx_1$ subject to $x_i \in \{0, 1\}$ and $x_i = x_1$ for $i = 2, \ldots, n$. With anchor $\mathbf{0}$, the unique optimum is $\mathbf{1}$ and $D_{\min} = \{1, \ldots, n\}$. Releasing any $n - 1$ variables leaves one fixed at zero, forcing all variables to zero. Recall is $1 - 1/n$, the restricted objective remains zero, and the unrestricted optimum is $-M$. □

The same construction with $n = 2$ shows that the attainable improvement $G(S) = c^\top x^A - z(S; x^A)$ need not be submodular. Both singleton improvements and $G(\varnothing)$ are zero, whereas $G(\{1, 2\}) = M$, violating the submodular inequality. Hence general MIPs do not admit an objective-improvement guarantee based on high partial recall or a submodular-greedy argument alone. This construction illustrates a possible source of complementarity; it is not a claim that every experimental family has this exact structure.

Table 4: **Variable-level feature inventory.** The controlled three-family representation contains 15 static features, 6 root-LP features, and 2 anchor-conditioned features.

No.	Feature	Block	Description
1	`obj_norm`	Static	Objective coefficient normalized by the largest absolute objective coefficient in the instance.
2	`degree`	Static	Number of constraints incident to the variable in the variable–constraint bipartite graph.
3	`deg_cent`	Static	Degree-based centrality of the variable in the bipartite graph.
4	`frac_eq`	Static	Fraction of incident constraints that are equalities.
5	`frac_leq`	Static	Fraction of incident constraints with a less-than-or-equal sense.
6	`frac_geq`	Static	Fraction of incident constraints with a greater-than-or-equal sense.
7	`bound_width_norm`	Static	Normalized width of the variable's feasible bound interval.
8	`vartype_code`	Static	Numerical encoding of the variable type.
9	`nbr_obj_mean`	Static	Mean objective information among structurally neighboring variables.
10	`n_vars`	Static	Number of variables in the MIP instance.
11	`n_cons`	Static	Number of constraints in the MIP instance.
12	`pagerank`	Static	PageRank score of the variable in the variable–constraint bipartite graph.
13–15	`fam_`$\star$	Static	Three family-indicator features identifying the controlled problem family.
16	`x_lp`	Root-LP	Value of the variable in the root-LP relaxation.
17	`lp_fractionality`	Root-LP	Distance of the root-LP value from integrality.
18	`lp_confidence`	Root-LP	Confidence derived from the proximity of the root-LP value to an integral assignment.
19	`lp_reduced_cost`	Root-LP	Reduced cost of the variable in the root-LP solution.
20	`lp_at_lb`	Root-LP	Indicator that the root-LP value lies at the variable's lower bound.
21	`lp_at_ub`	Root-LP	Indicator that the root-LP value lies at the variable's upper bound.
22	`x_anchor`	Anchor	Value of the variable in the feasible anchor solution x^A.
23	`anchor_lp_absdiff`	Anchor	Absolute disagreement between the anchor and root-LP values, $\lvert x_i^A - x_i^{\mathrm{LP}}\rvert$.

A.6 Minimum-Correction Certification

For binary integer variables and a certified optimal objective $z^\star$, a nearest optimum can be defined by the auxiliary MIP

$$\min_{x\in\mathcal{X},\ c^\top x\le z^\star} \left(\sum_{i:x_i^A=0} x_i + \sum_{i:x_i^A=1} (1-x_i) \right). \tag{31}$$

Its objective is the integer-variable Hamming distance to the anchor. Our controlled labels use this two-stage procedure, with optimality certified for both objectives. Certifying only the original objective would not establish minimum correction cardinality. With an objective tolerance $\eta > 0$, the relaxed constraint $c^\top x \le z^\star + \eta$ instead targets $K(z^\star + \eta; x^A)$. This distinction also applies to numerical solver tolerances.

B Feature Definitions

This appendix summarizes the variable-level features used in the experiments. The controlled representation contains 23 features: 15 static features, 6 root-LP features, and 2 anchor-conditioned features. Table 4 lists the complete feature set. Most features are self-explanatory from Table 4. We further specify three features whose definitions involve normalization or implementation choices.

Normalized objective coefficient. The normalized objective feature `obj_norm` is $\frac{c_i}{\max_j |c_j|}$, when the denominator is positive.

Graph-based structural feature. The `pagerank` feature is computed on the variable–constraint bipartite graph, with an edge whenever the corresponding constraint coefficient is nonzero. Unlike local degree statistics, PageRank captures recursive structural prominence in the constraint system. We use a damping factor of 0.85 and 40 iterations in the controlled experiments. For MILPBench,

we disable the PageRank feature because computing it on the large incidence graphs adds minutes of preprocessing overhead. The remaining static features are retained unchanged.

LP confidence. For binary variables, the LP-confidence feature `lp_confidence` is $2\left|x_i^{\mathrm{LP}} - \frac{1}{2}\right|$. Values near zero correspond to LP assignments near 0.5, whereas values near one indicate proximity to an integral binary assignment. For general integer variables, the confidence score is derived from LP fractionality.

C Implementation and Baseline Configuration

GBDT Predictors. Our method and Value-Fixing use `HistGradientBoostingClassifier` with maximum depth 6, learning rate 0.08, at most 300 boosting iterations, ℓ_2 regularization 0.1, and class weighting for label imbalance. Hyperparameters are fixed across evaluation. MILPBench uses separate family-specific LP-aware and NoLP models trained on the official easy-scale training instances.

Solvers and compute. Experiments run on a Dell Pro Max Tower T2 with an Intel Core Ultra 9 285K CPU (24 cores), 128 GB RAM, and an NVIDIA GeForce RTX 5090 GPU (32 GB), running Ubuntu 26.04.1 LTS. Controlled experiments use SCIP 10.0.3 through PySCIPOpt 6.2.1, while MILPBench experiments use Gurobi 12.0.3. Solver runs use up to eight CPU threads. Our method does not use GPU acceleration; the GPU is used only by EnCore-PS inference.

Timing and tuning. Controlled online time includes anchor generation, root-LP computation, prediction, and restricted reoptimization. MILPBench runtime is measured end-to-end from anchor generation through restricted reoptimization. The controlled correction-mass threshold is $\rho = 0.875$. Local Branching radii are 0.02, 0.40, and 0.70 for Set Cover, Auction, and Facility, respectively. MILPBench uses $\rho_{\mathrm{LP}} = 0.96$ and $\rho_{\mathrm{NoLP}} = 0.825$. All reported thresholds are selected before held-out evaluation and then frozen.

Neighborhood baselines. RINS is implemented as a one-shot root-LP neighborhood: integer variables agreeing with the anchor within 10^{-6} are fixed, and the remaining variables are reoptimized. Random-LNS follows the variable-partitioning strategy of Song et al. (2020) with three blocks. ACP follows Ye et al. (2022); for the short controlled budgets, its subproblem schedule is scaled to the available wall-clock budget. Local Branching uses the development-selected radii above. Value-Fixing uses the same feature groups, GBDT class, and free-variable count as our correction policy, but predicts values rather than release scores.

EnCore-PS. EnCore-PS uses the official implementation with Predict-and-Search and a single early feasible solution. Published schedules are used where available; fixed local schedules are used otherwise. Its downstream solver limit is paired with our online budget, but its graph construction, GPU inference, and separate early-solution generation are outside that timer. We therefore treat EnCore-PS as a reproduced reference rather than a strictly end-to-end time-matched baseline.

D Benchmark Instance Statistics

Table 5 summarizes the instance sizes used in the controlled three-family experiments and the large-scale MILPBench evaluation.

E Sizing Ablation under Paired Reoptimization Time Limits

We isolate release-set sizing while keeping the GBDT ranking fixed. Each configuration is evaluated on 10 instances per family and three seeds per instance (30 paired runs). Ours uses cumulative-mass sizing with $\rho = 0.875$. Fixed (dev-tuned) uses family-specific release fractions selected on the development set by minimum mean gap: 0.30 for Set Cover, 0.60 for Auction, and 0.85 for Facility. A second Fixed configuration uses approximately mean-matched fractions of 0.099, 0.430, and 0.708, respectively. Matching mean freedom does not match the number released in each run, so this control tests instance-dependent allocation rather than variable ranking.

Table 5: **Benchmark instance statistics.** Numbers of variables (n), integer variables ($|\mathcal{I}|$), and constraints (m). Scale labels are defined separately within each benchmark. SC, MVC, and MIS denote Set Cover, minimum vertex cover, and maximum independent set, respectively.

Family	Scale	n	$\lvert\mathcal{I}\rvert$	m
Controlled benchmarks				
Set Cover	Medium	1,000	1,000	1,000
	Hard	1,000	1,000	2,000
Auction	Medium	1,000	1,000	386
	Hard	1,500	1,500	577
Facility	Medium	20,100	100	20,301
	Hard	40,100	100	40,501
MILPBench				
Set Cover	Easy	40,000	40,000	40,000
	Medium	200,000	200,000	200,000
	Hard	2,000,000	2,000,000	2,000,000
MVC	Easy	20,000	20,000	60,000
	Medium	100,000	100,000	300,000
	Hard	1,000,000	1,000,000	3,000,000
MIS	Easy	20,000	20,000	60,000
	Medium	100,000	100,000	300,000
	Hard	1,000,000	1,000,000	3,000,000

For each instance–seed pair, we first record the gap and reoptimization wall time t^{ours} from the Ours run without the additional paired time cap. We then set each Fixed run's solver time limit to t^{ours}. The paired runs use the same instance, anchor, GBDT ranking, and solver seed. Each Fixed run receives the same feasible anchor as a warm start and retains its best feasible incumbent at termination, falling back to the anchor if no improvement is found. The timing and warm-start conventions were checked across policies. Actual observed wall times can differ from the limits because of early termination and time-limit enforcement overhead. This is a post-hoc, Ours-runtime-derived control of the reoptimization stage, not a prespecified common end-to-end budget. Ours gaps are taken from the original runs, not from a new capped rerun.

Fixed fractions are not retuned for the paired time limits, so the comparison concerns these configurations rather than a time-budget-optimized fixed policy.

Table 6: **Sizing under paired reoptimization time limits.** Each Fixed solver limit equals the corresponding observed Ours reoptimization time. Gap statistics are over 10 instances $\times$ 3 seeds; time is mean observed reoptimization wall time, not total online time. Capped counts runs truncated by the imposed limit. Ours is the uncapped reference run.

Family	Policy	Mean gap (%)	Median gap (%)	Time (s)	Capped
Set Cover	Ours	1.00	0.76	1.519	0/30
	Fixed ($r = 0.30$)	4.23	4.45	1.551	30/30
	Fixed ($r = 0.099$)	2.17	1.97	0.973	12/30
Auction	Ours	0.44	0.46	2.118	0/30
	Fixed ($r = 0.60$)	0.56	0.45	2.211	30/30
	Fixed ($r = 0.430$)	0.55	0.42	1.881	14/30
Facility	Ours	1.67	1.35	5.892	0/30
	Fixed ($r = 0.85$)	144.67	23.22	5.924	30/30
	Fixed ($r = 0.708$)	35.97	5.67	3.727	10/30

Tables 6–7 show family-dependent effects. On Set Cover, Ours beats approximately mean-matched Fixed in 16 pairs, loses none, and ties 14. On Facility it wins 24 pairs, loses one, and ties five; both mean and median gaps favor Ours. The large mean gap of Fixed $r = 0.85$ is sensitive to poor incumbents, but its median gap (23.22%) also exceeds that of Ours (1.35%). Its 12 paired wins nevertheless show heterogeneous behavior. Auction remains close: Ours has a slightly lower mean

Table 7: **Paired gap differences under the time control.** $\Delta g = g_{\text{Fixed}} - g_{\text{Ours}}$; positive values favor Ours. Mean differences are computed before rounding the displayed gaps. Counts are Ours wins / Fixed wins / ties. Runs share instances across seeds; counts are descriptive, not independent-sample significance tests.

Family	Control	Mean Δg (pp)	Wins / losses / ties
Set Cover	Fixed ($r = 0.30$)	$+3.23$	26 / 2 / 2
	Fixed ($r = 0.099$)	$+1.17$	16 / 0 / 14
Auction	Fixed ($r = 0.60$)	$+0.11$	16 / 13 / 1
	Fixed ($r = 0.430$)	$+0.11$	9 / 11 / 10
Facility	Fixed ($r = 0.85$)	$+143.00$	18 / 12 / 0
	Fixed ($r = 0.708$)	$+34.30$	24 / 1 / 5

Table 8: Root-LP geometry on representative MILPBench instances. "Near half" denotes $|x_i^{\text{LP}} - 0.5| \leq 0.05$, whereas "Exact half" denotes $|x_i^{\text{LP}} - 0.5| \leq 10^{-6}$. Near-half statistics are reported only to characterize LP geometry; the reliability gate uses only the exact-half criterion. Hard MVC and MIS exhibit a distinct LP state in which more than 90% of binary variables collapse numerically to 0.5.

Setting	Mean $\lvert x^{\text{LP}} - 0.5\rvert$	Near half (%)	Exact half (%)
SC Medium	0.408	3.9	0.0
SC Hard	0.410	3.7	0.0
MVC Medium	0.043	91.4	0.5
MIS Medium	0.043	91.5	0.6
MVC Hard	0.041	91.8	91.8
MIS Hard	0.041	91.8	91.8

gap, while Fixed has slightly lower medians and mixed paired outcomes. Thus, on Set Cover and Facility, the observed benefits are not explained solely by allowing Ours more reoptimization time than these fixed configurations. The experiment does not establish uniform gains across families or budgets.

F MILPBench Root-LP Reliability Gate

The controlled experiments use root-LP information as part of the anchor-conditioned variable representation. On MILPBench, however, a numerically valid LP state does not necessarily provide discriminative variable-level information. This appendix describes the reliability gate used in Section 4.5.

F.1 Half-Integral LP-State Collapse

For binary graph problems such as MIS and MVC, LP relaxations can place a large fraction of variables near 0.5. Near-half values alone are not necessarily problematic for our predictor: as can be seen in Table 8, at medium scale, most binary variables lie close to 0.5, yet retain small variable-specific deviations that can provide discriminative information. At hard scale, however, we observe a qualitatively different regime in which approximately 92% of binary variables collapse numerically to 0.5 within a tight tolerance.

This exact-half collapse directly affects several features used by the LP-aware predictor. For a binary variable, these include the root-LP value x_i^{LP} (`x_lp`), its distance from integrality $|x_i^{\text{LP}} - \text{round}(x_i^{\text{LP}})|$ (`lp_fractionality`), and the LP confidence $2|x_i^{\text{LP}} - 0.5|$ (`lp_confidence`). The representation also contains the anchor–LP disagreement $|x_i^A - x_i^{\text{LP}}|$ (`anchor_lp_absdiff`), together with reduced-cost and LP-bound features. When $x_i^{\text{LP}} = 0.5$ and the anchor $x_i^A \in \{0, 1\}$, these four quantities become

$$x_i^{\text{LP}} = 0.5, \qquad |x_i^{\text{LP}} - \text{round}(x_i^{\text{LP}})| = 0.5, \qquad 2|x_i^{\text{LP}} - 0.5| = 0, \qquad |x_i^A - x_i^{\text{LP}}| = 0.5.$$

Thus, several LP-derived and anchor–LP features become identical across the affected variables and lose their variable-level discrimination, even though other LP-derived features, such as reduced costs, may still vary. This differs from ordinary near-half fractionality, where small deviations from 0.5 remain available to the predictor. Importantly, exact-half collapse is not an LP solve failure. The returned primal vector can be numerically valid and usable while providing substantially less discriminative variable-level information for the learned ranking model. We therefore distinguish *LP availability* from *LP discriminativeness*: the former determines whether an LP state can be used at all, whereas the latter determines whether the LP-aware representation should be trusted for scoring.

F.2 Reliability Statistic and Gate

For an instance with binary integer variables, define the exact-half ratio

$$r_{\text{half}} = \frac{1}{|\mathcal{I}_b|} \sum_{i \in \mathcal{I}_b} \mathbf{1}\left[|x_i^{\text{LP}} - 0.5| \le \epsilon\right], \tag{32}$$

where $\mathcal{I}_b$ denotes the set of binary variables. We use $\epsilon = 10^{-6}$ and declare an available LP state collapsed when

$$r_{\text{half}} > \tau_{\text{half}}, \qquad \tau_{\text{half}} = 0.5. \tag{33}$$

The threshold is intended as a regime detector rather than a finely tuned decision boundary. Its purpose is not to distinguish moderately fractional from integral LP solutions, but to detect the regime in which a majority of binary variables receive the same numerically exact half-integral value. Mean distance from 0.5 alone does not identify this collapse: medium and hard MVC/MIS exhibit similar mean deviations, despite sharply different exact-half ratios. We therefore gate on the concentration of numerically exact half-integral values rather than average fractionality. In the reported experiments, hard MVC/MIS instances have $r_{\text{half}} \approx 92\%$, whereas medium MVC/MIS instances have exact-half ratios below 1%. SC instances remain far from the collapsed regime. Consequently, the gate does not react to ordinary LP fractionality. We additionally define $a_{\text{LP}} \in \{0, 1\}$ to indicate whether a usable root-LP primal assignment is available. A timeout alone does not imply $a_{\text{LP}} = 0$: if the LP procedure returns a usable primal assignment, the LP-aware path remains eligible. If no usable assignment is obtained, the framework uses NoLP without evaluating r_{half}.

F.3 Companion Models and Correction-Mass Sizing

We train two companion predictors on the same easy-scale instances so that each scoring path uses the feature representation seen during training:

- an *LP-aware* GBDT using static, anchor-conditioned, and LP-derived features; and
- a *NoLP* GBDT using static features and LP-independent anchor features.

The scoring path is selected at the instance level:

$$p_i = \begin{cases} f_{\text{LP}}(s_i), & a_{\text{LP}} = 1 \ \wedge \ r_{\text{half}} \le \tau_{\text{half}}, \\ f_{\text{NoLP}}(s_i^{\text{NoLP}}), & \text{otherwise.} \end{cases} \tag{34}$$

The two scoring paths use development-selected correction-mass thresholds

$$\rho_{\text{LP}} = 0.96, \qquad \rho_{\text{NoLP}} = 0.825.$$

Both thresholds are fixed before medium- and hard-scale evaluation. The threshold follows the selected representation path rather than the problem family or instance scale. The gate changes the scoring representation and its associated correction-mass threshold, but not the underlying reoptimization mechanism. In both paths, variables are ranked by correction scores, the cumulative correction-mass rule determines the free set, and the resulting restricted MIP is solved by the same underlying MIP solver. The quality–freedom formulation is unaffected.

F.4 Activation Diagnostics

The two NoLP activation paths serve different purposes. The unavailable-LP fallback handles cases in which the LP procedure produces no usable primal assignment. The collapse gate handles cases in which an LP assignment is available but its exact-half concentration exceeds τ_{half}. In the reported MILPBench experiments, no instance activates the unavailable-LP fallback. All medium-scale instances and SC hard instances use LP-aware scoring. Hard MVC and MIS instances activate NoLP because their exact-half ratios are approximately 0.92. These observations illustrate why the gate is based on the observed LP state rather than problem family or instance scale. In particular, SC hard instances retain LP-aware scoring despite their larger size, whereas hard MVC and MIS switch to static and anchor-conditioned evidence when several LP-derived features lose variable-level discrimination.

G Comparison with Published MILPBench Results

Table 9 provides published results as context, separately from the local experiments in Table 3. The first four rows are transcribed from prior work; Ours is repeated for convenience. Hardware, solver configurations, and timing conventions need not match. Reported times are retained as supplied rather than interpreted as a common evaluation horizon. In particular, the LLM-LNS row reports 2,000/8,000 seconds for medium/hard settings. Cross-source runtime ratios are not controlled speedups.

Our SC objectives are lower than the listed published results. On MVC and MIS, our objectives are within 0.5% of LLM-LNS at medium scale and 2.1% at hard scale. These are descriptive comparisons of reported means, not statistical superiority claims.

Table 9: **Published MILPBench results for context.** SC/MVC are minimized and MIS maximized. The first four rows reproduce literature results; Ours repeats local measurements. Timing conventions and hardware differ. "–" denotes reported CL-LNS sampling failure. Bold marks the best listed mean objective.

	SC (medium) ↓		SC (hard) ↓		MVC (medium) ↓		MVC (hard) ↓		MIS (medium) ↑		MIS (hard) ↑	
Method	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)	Obj.	Time (s)
CL-LNS (Huang et al., 2023)	–	–	–	–	31285.0	2000	–	–	15000.0	2000	–	–
GNN&GBDT (Ye et al., 2023)	17503.4	5041.6	252797.2	>60000	27329.9	29320.5	274600.8	21397.3	22530.1	4227.1	215393.6	27952.9
Light-MILPOpt (Ye et al., 2024)	16108.1	177.8	160015.5	1795.4	26950.7	193.8	269571.5	1503.3	22966.5	223.5	230432.9	2062.7
LLM-LNS (Ye et al., 2025b)	15950.2	2000	161732.8	8000	**26763.4**	2000	**268825.5**	8000	**23137.2**	2000	**230682.8**	8000
Ours	**15937.7**	238.5	**159664.2**	369.8	26895.8	311.2	274382.4	175.2	23081.3	396.5	226003.4	176.0

H Additional Related Work

We provide additional context on methods related to anchor-conditioned reoptimization.

Incumbent-centered neighborhood search. Classical primal heuristics exploit feasible incumbents by restricting subsequent optimization to a nearby region. Local Branching (Fischetti & Lodi, 2003) searches within a prespecified Hamming-distance neighborhood, while RINS (Danna et al., 2005) fixes variables on which an incumbent and an LP-relaxation solution agree. Proximity Search (Fischetti & Monaci, 2014) favors solutions close to an incumbent while requiring objective improvement, and Weighted Proximity Search (Rodrigues et al., 2021) assigns nonuniform weights to different coordinate changes. Our formulation instead associates a target solution quality with the amount of coordinate-wise solver freedom retained around an anchor, and learns which anchor decisions should remain free.

Learned neighborhood construction. Learning has also been used to construct or adapt neighborhoods within large-neighborhood search. Song et al. (2020) learn variable subsets for iterative LNS, Wu et al. (2021) learn destroy operators through reinforcement learning, and subsequent work learns related neighborhood or search policies (Sonnerat et al., 2021; Huang et al., 2023; Yuan et al., 2025). Liu et al. (2022) additionally learn instance-dependent Local Branching radii and their adaptation. Our method likewise adapts both the identity and number of released variables, but derives them from anchor-conditioned correction evidence and constructs a single restricted reoptimization problem rather than an iterative destroy–repair policy.

Solution prediction and variable fixing. Another line of work predicts variable assignments and uses them to reduce the MIP search space. Neural Diving (Nair et al., 2020) predicts partial assignments completed by a MIP solver, while Predict-and-Search (Han et al., 2023) constructs search regions around learned solution predictions. Other approaches use confidence estimates or iterative prediction and correction to determine which predicted values can be fixed (Huang et al., 2024; Liu et al., 2025). Anchor-conditioned reoptimization instead predicts where solver freedom is useful rather than predicting replacement values; released variables remain jointly determined by the underlying MIP solver.

Early-to-final consistency. EnCore (Li et al., 2026) predicts agreement between early assignments and full-budget references using an incumbent-conditioned GNN. For a shared anchor–reference pair, its consistency labels and our reference-correction labels are complementary:

$$\mathbf{1}[x_i^R \neq x_i^A] = 1 - \mathbf{1}[x_i^R = x_i^A].$$

Our minimum-correction target additionally selects a nearest optimal solution in Hamming distance, while our learned policy uses cumulative-score sizing followed by a single restricted solve. EnCore reports cross-solver and zero-shot cross-family transfer. We study a different transfer axis: within-family cross-scale transfer from 20,000–40,000-variable MILPBench instances to problems with up to one to two million variables. These settings are complementary and do not establish general superiority in transfer, runtime, or memory.