TACIT: OPTIMIZATION MODELS THAT LEARN FROM THEIR MISTAKES

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world optimization problems are difficult to model accurately because many objectives and constraints reside in domain experts' tacit knowledge, making them hard to formalize. As a result, optimization models often contain miscalibrated objectives, missing constraints, or omitted decision variables, leading to solutions that fail to reflect operational realities. We address this challenge by automatically repairing misspecified formulations using historical data consisting of past solutions and subsequent user overrides. Traditional approaches such as inverse optimization and constraint learning tend to overfit sparse data and produce complex formulations. Our central idea is to combine the reasoning capabilities and prior knowledge of LLMs with the formal grounding provided by optimization. We realize this idea through two complementary paradigms. Top-down, an LLM proposes structural repairs, including new constraints and variables, whose numerical parameters are calibrated and validated through optimization. Bottom-up, optimization infers cuts from observed decisions, which the LLM contextualizes into interpretable, generalizable modeling constraints. We evaluate our approach on 38 misspecification scenarios spanning nine classes of optimization problems, several drawn from real-world applications, and show that TACIT can repair 78.9% of them (vs. 60.5% for the best baseline).

1 INTRODUCTION

Mathematical optimization is a powerful tool for decision-making, enabling organizations to make complex decisions at a scale and speed that would be difficult to achieve manually. As a result, it plays a central role in automating decision processes across applications such as supply chain and logistics, scheduling, resource allocation, and infrastructure planning. However, an optimization system is only as good as the formulation on which it is built. Constructing such a formulation requires translating a real-world decision problem into decision variables, constraints, and objectives that faithfully capture how decisions should be made in practice.

This translation is inherently challenging. It typically requires close collaboration between domain experts, who understand the operational requirements and preferences governing decisions, and optimization experts, who encode this knowledge into a mathematical model. Much of this knowledge, however, is tacit and may not be fully articulated during model development. Consequently, optimization formulations can contain misaligned objectives, missing constraints, or omitted variables, producing solutions that fail to reflect operational realities (Baxi et al., 2026).

Large Language Models (LLMs) are increasingly being used to automate parts of the formulation process by translating natural-language problem descriptions into mathematical optimization models (AhmadiTeshnizi et al., 2024; Astorga et al., 2024; Zhang et al., 2024). While recent advances have substantially improved formulation accuracy, they do not eliminate the fundamental challenge of specifying the right decision problem. Requirements absent from the problem description cannot readily be reflected in the resulting formulation, and LLMs may introduce additional formulation errors of their own. Thus, LLM-generated formulations may also be incomplete or misaligned with the decisions users intend to make.

When misaligned optimization systems are deployed in practice, deficiencies in the underlying formulation become visible through their decisions. Recommendations that violate implicit operational requirements or fail to capture user preferences may be rejected, manually overridden, or repeatedly

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.

adjusted through tedious processes. Addressing such failures thus often requires costly iterations between domain and optimization experts to diagnose the source of the mismatch and revise the formulation. Yet, these interactions also generate valuable information: rejected recommendations and user corrections provide signals about which aspects of the formulation are incomplete or misaligned. This motivates our central question:

Can we recover the formulation that governs users' decisions from their corrections, as an explicit and interpretable repair of the deployed one?

To illustrate the difficulty, consider a toy example with a manufacturer choosing the number of motors (x) to produce and machines (y) to assemble, where each machine requires one motor. Suppose the formulation mistakenly omits this dependency and captures only production capacity and handling constraints: $\max\{2x + 5y : 2x + 3y \leq 10,\ x + y \leq 4,\ x, y \in \mathbb{Z}_{\geq 0}\}$. Its optimal solution, $(0, 3)$, assembles three machines without producing any motors. The user corrects it to $(2, 2)$, which is optimal under the intended constraint $x \geq y$. However, this correction does not uniquely reveal the missing constraint: many constraints could make $(2, 2)$ optimal, while a similar result could be achieved by just changing the objective coefficients. Recovering $x \geq y$ thus requires understanding the semantic relationship between the variables, highlighting the difficulty of repairing optimization models from solution feedback alone.

In practice, users typically provide multiple feedback points as they repeatedly solve instances of the same underlying formulation, but this challenge grows significantly for real-world formulations with a substantial number of variables and constraints, often spanning distinct semantic families and different aspects of the modeled system. More fundamentally, the formulation itself may be incomplete: entire variables may be missing, including auxiliary variables used to represent nonlinear objectives or logical relationships (see Example 1). Repair may therefore require recovering not only constraints, but missing pieces of the modeling structure itself.

This toy example exposes three broader challenges that make formulation repair fundamentally hard:

- **Indirect feedback.** In practice, users such as supply chain planners or business operators typically focus on correcting the proposed solution rather than diagnosing the underlying model error.
- **Non-identifiability.** A corrected solution rarely pins down the intended model: the same feedback may be explained by different constraints, objectives, or structural changes, as in the example above. With few corrections, local fixes can explain without capturing the underlying rule.
- **Interpretability.** A repair should not merely reproduce observed decisions through an unnatural or unnecessarily complex formulation: it should have a clear real-world interpretation that users can inspect and validate. Recent work highlights the challenges practitioners face in interpreting and modifying optimization models, as well as the broader importance of aligning algorithmic systems with human judgment and organizational objectives (Chen et al., 2026; Caro et al., 2026).

Our contribution. We introduce TACIT (Templates And Cuts for Interpretable Transformations), an end-to-end framework for automatically repairing optimization formulations from feedback. We cast repair as a search over modifications of the original formulation: adding or changing constraints, variables, and objective terms. This space is vast and underdetermined: many formulations can be consistent with any finite set of corrections, but few are meaningful to a modeler. Our central idea is to combine the reasoning capabilities and prior knowledge of LLMs with the formal grounding provided by optimization. We realize this idea through two complementary paradigms that run within a single search loop. *Bottom-up*, optimization derives cuts that separate rejected solutions from accepted ones, and an LLM agent interprets, refines, and generalizes these cuts into meaningful modeling rules. *Top-down*, an LLM agent identifies a potentially missing modeling component and expresses it as a template, possibly introducing new variables, constraints, or unspecified parameters, which optimization then specifies from the feedback.

This hybrid architecture provides several advantages:

1. **Broad coverage of misspecifications.** Our framework supports a substantially broader class of repairs than local changes to coefficients or individual constraints, recovering missing, extraneous, or miscalibrated components in the objective, constraints, and variables. A single repair can introduce auxiliary variables, add or remove constraints, and modify the objective jointly.
2. **Domain generality.** The framework is not tailored to a specific application domain (e.g., transportation, scheduling) and applies across structurally diverse optimization models.

3. **No human in the loop.** The framework operates from sparse, offline signals, namely user-corrected solutions, without querying users or requiring them to diagnose the model error.
4. **Interpretability and generalization.** By searching over semantically meaningful modeling changes, the framework promotes interpretability while leveraging limited feedback to recover modeling rules that generalize beyond observed data.

Finally, we construct a comprehensive benchmark of 38 misspecification scenarios spanning nine classes of optimization problems, including several real-world applications, with errors involving constraints, variables, parameters, and objective components. Across these settings, TACIT can recover the ground-truth formulation, up to decision equivalence, in 78.9% of the scenarios from limited user feedback, against 60.5% for the best baseline. The recovered formulations generalize to unseen instances, and our ablations show that combining LLM reasoning with optimization is critical. At the same time, both top-down and bottom-up inference improve repair performance independently, but their combination achieves the best overall performance. Overall, our experiments show that sparse solution feedback can be sufficient to repair complex formulation misspecifications without requiring users to diagnose the formulation themselves.

2 Related Work

Data-driven model customization and preference modeling. The closest works to ours (Hewitt & Frejinger, 2020; Bayani et al., 2024; Abdellaoui et al., 2026) correct optimization models from prescribed and implemented solutions by learning a function that models directly how users modify the solutions. These approaches range from affine mappings (Hewitt & Frejinger, 2020) to regression and decision trees (Bayani et al., 2024), and richer predictors for routing and scheduling (Abdellaoui et al., 2026). Related work learns user preferences within an optimization problem (Maragno et al., 2025); see also Fajemisin et al. (2021). There, an ML model modeling user behavior or preference is embedded within the optimization formulation and is trained based on the optimization outcomes. While these data-driven customization methods can substantially improve decisions, the learned behavioral or preference models are encoded within the formulation using additional variables and constraints that do not possess real-world meaning. In contrast, we recover an *explicit, corrected formulation* that can be inspected and validated by humans. Moreover, some complex formulation errors cannot be corrected by such learned-function embeddings, as shown in Theorem 1 (Appendix A.1); our framework can recover the appropriate corrections even in these complex cases.

Inverse optimization and learning optimization models. Inverse optimization estimates unknown model parameters from observed decisions, including under multiple or noisy observations and imperfect information (Chan et al., 2025; Moghaddass & Terekhov, 2021; Aswani et al., 2018; Mohajerin Esfahani et al., 2018; Bulut & Ralphs, 2021). Recent methods also jointly learn objectives and feasible regions of LPs or MILPs from contextual examples (Tan et al., 2020; Kumar et al., 2021; Ren et al., 2025; Kitaoka, 2025). This literature motivates our numerical calibration components and highlights the identifiability challenges of decision-only supervision. Moreover, our setting additionally requires recovering the *semantic and structural content* of a misspecified formulation. In addition, these methods cannot naturally capture more complex structural corrections that require introducing new variables and coordinating them with new constraints or objective terms. Our framework addresses both limitations by using LLMs to inject semantic knowledge into the repair process and to propose structural changes involving previously absent variables.

Language models for optimization modeling and repair. LLMs have been used to formulate optimization models from natural-language specifications and diagnose or repair them using solver feedback and domain-specific checks (AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Astorga et al., 2024; Zhang et al., 2024; Thind et al., 2025; Jiang et al., 2025; Chen et al., 2024; Ao et al., 2026a;b); see Xiao et al. (2025) for a recent survey. This literature is complementary to our setting: autoformulation methods assume intended requirements are expressed in the specification, while diagnostic repair relies on explicit specifications, infeasibility, or predefined checks. We instead address requirements that are *missing from the formulation and its specification*, inferring them from downstream behavioral evidence and combining LLM semantic priors with optimization-based validation.

3 PROBLEM DEFINITION

Core concepts. An instance is described by a data vector d (e.g., demands, capacities, availabilities), whose dimension may vary across instances. We consider mixed-integer linear programs (MILPs), with linear programs (LPs) as a special case. A *formulation* is a pair $\mathcal{M} = (c, \mathcal{X})$ mapping instance data d to an optimization problem $\mathcal{M}(d)$: $\min c(d)^\top y$ s.t. $y \in \mathcal{X}(d)$, where $c(d)$ is the vector of objective coefficients and $\mathcal{X}(d)$ the feasibility set, defined by linear constraints and, when applicable, integrality requirements. We use $\mathcal{S}(d; \mathcal{M})$ to denote its set of optimal solutions. An *initial formulation* $\mathcal{M}_0 = (c_0, \mathcal{X}_0)$ is given but may not fully capture the underlying decision problem. There is an unknown *target formulation* $\mathcal{M}^\star = (c^\star, \mathcal{X}^\star)$ that is the ground truth that governs the decisions actually taken. The two may differ in the objective (missing terms or incorrect weights), constraints (missing, miscalibrated, or extraneous constraints), variables (e.g., auxiliary variables present only in $\mathcal{M}^\star$), or any combination thereof.

Example 1 (Vehicle routing problem (VRP))**.** *A company operates a fleet $\mathcal{L}$ of vehicles and must serve a set of customers C, with d_{ij} denoting the distance between customers i and j. Let variable $x_{ij\ell}$ indicate whether vehicle ℓ visits customer j immediately after customer i. The company seeks to minimize total distance plus a penalty for routes that exceed D miles. In this example, the initial formulation $\mathcal{M}_0$ omits this penalty, which appears in the target formulation $\mathcal{M}^\star$ (in blue):*

$$\mathcal{M}_0: \quad \min_{x} \ \sum_{\ell \in \mathcal{L}} \sum_{i,j \in C} d_{ij} x_{ij\ell} \qquad \text{s.t.} \quad x \in \mathcal{X}_{\text{VRP}}.$$

$$\mathcal{M}^\star: \quad \min_{x,z} \ \sum_{\ell \in \mathcal{L}} \sum_{i,j \in C} d_{ij} x_{ij\ell} + \lambda \sum_{\ell \in \mathcal{L}} z_\ell \qquad \text{s.t.} \quad x \in \mathcal{X}_{\text{VRP}},$$
$$\sum_{i,j \in C} d_{ij} x_{ij\ell} \le D + M z_\ell, \qquad \forall \ell \in \mathcal{L}$$
$$z_\ell \in \{0,1\}, \qquad \forall \ell \in \mathcal{L},$$

$\mathcal{X}_{\text{VRP}}$ denotes the standard VRP constraints. Repairing $\mathcal{M}_0$ requires coordinated changes: adding auxiliary variables, linking constraints, a new objective term, and estimating its coefficient λ.

We refer to the variables of the initial formulation $\mathcal{M}_0$ as *primary variables*: they encode the actionable decisions made and reported by practitioners. In Example 1, $x_{ij\ell}$ are primary, while the auxiliary variables z_ℓ introduced by $\mathcal{M}^\star$ are not. Any additional variables introduced by $\mathcal{M}^\star$ are called *auxiliary variables*. Only the primary variables of $\mathcal{M}^\star$ are observed in the user feedback. Let Π_0 be the projection that restricts a solution to the primary variables. A *feasible lifting* of x under $\mathcal{M}$ is any solution $y \in \mathcal{X}(d)$ satisfying $\Pi_0(y) = x$.

Observed data. Instances are drawn from an unknown distribution P over triples $(d, x^0, x^\star)$, where $x^0 \in \mathcal{S}(d; \mathcal{M}_0)$ is the *initial solution* and $x^\star \in \Pi_0(\mathcal{S}(d; \mathcal{M}^\star))$ the *target solution*, the projection onto the primary variables of an optimal solution of $\mathcal{M}^\star(d)$. We observe a sample $\mathcal{D} = \{(d_i, x_i^0, x_i^\star) : i \in \mathcal{I}\} \sim P^{|\mathcal{I}|}$. Optimality of the targets is a standard assumption of inverse optimization (Chan et al., 2025; Besbes et al., 2025): we use it to describe the problem and our methodology and relax it in some of our experiments. We further posit that whenever $x_i^0 \neq x_i^\star$, the target is strictly preferred, that is, every lifting of x_i^0 is infeasible or strictly suboptimal under $\mathcal{M}^\star$. This reflects practice, where the initial solution is the default and practitioners depart from it only when an alternative is strictly better.

Separation loss. Our goal is to find a formulation $\mathcal{M}$ that best explains the observed data $\mathcal{D}$. We formalize this through *separability*: $\mathcal{M}$ *separates* $\mathcal{D}$ if, for every instance i with $x_i^0 \neq x_i^\star$, $x_i^\star \in \Pi_0(\mathcal{S}(d_i; \mathcal{M}))$ and $x_i^0 \notin \Pi_0(\mathcal{S}(d_i; \mathcal{M}))$. That is, the target solution is feasible and optimal under $\mathcal{M}$, whereas the initial solution is not. By the assumptions above, $\mathcal{M}^\star$ separates $\mathcal{D}$. To quantify the degree of separation achieved by a candidate formulation $\mathcal{M} = (c, \mathcal{X})$, we measure the feasibility and optimality of a solution x on instance i through its relative optimality gap $\gamma_i(x) \triangleq \frac{Z_i(x) - Z_i^{\text{opt}}}{|Z_i^{\text{opt}}|}$, where $Z_i^{\text{opt}} = \min\{c(d_i)^\top y : y \in \mathcal{X}(d_i)\}$ and $Z_i(x) = \min\{c(d_i)^\top y : y \in \mathcal{X}(d_i), \Pi_0(y) = x\}$ denote, respectively, the optimal objective value of $\mathcal{M}(d_i)$ and the best objective value among feasible liftings of x.[1] By construction, $\gamma_i(x) = 0$ if and only if $x \in \Pi_0(\mathcal{S}(d_i; \mathcal{M}))$. Given

[1] When $Z_i^{\text{opt}} = 0$, we add a small positive constant to the denominator, and if x does not admit a feasible lifting under $\mathcal{M}(d_i)$ we define $\gamma_i(x) = 1$.

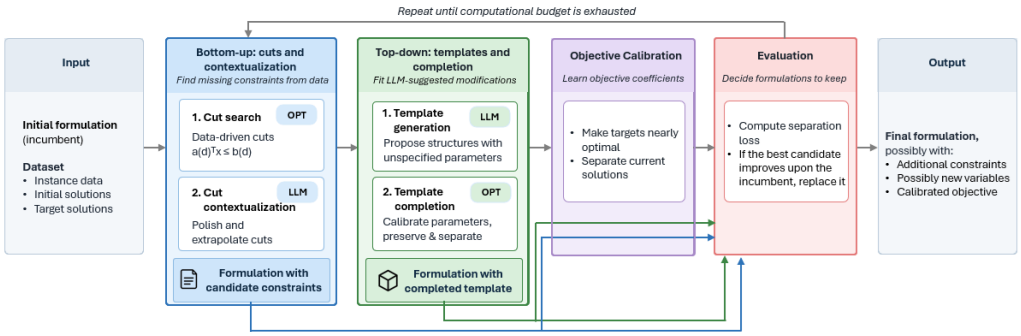


Figure 1: TACIT framework overview.

tolerances $0 \leq \delta \leq \Delta < 1$, we define the *separation loss*:

$$\mathcal{L}(\mathcal{M};\mathcal{D}) \triangleq 1 - \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} r_i(\mathcal{M}),\ r_i(\mathcal{M}) \triangleq \begin{cases} \mathbb{1}_{\{\gamma_i(x_i^\star)\leq\Delta\}} \cdot \mathbb{1}_{\{\gamma_i(x_i^0)\geq\gamma_i(x_i^\star)+\delta\}} & \text{if } x_i^0 \neq x_i^\star \\ \mathbb{1}_{\{\gamma_i(x_i^\star)\leq\Delta\}} & \text{otherwise.} \end{cases} \quad (1)$$

Here, Δ bounds the relative optimality gap of the observed target solutions, while δ is the minimum gap difference separating the initial solution from the target. Thus, $r_i(\mathcal{M}) = 1$ if $\mathcal{M}$ makes $x_i^\star$ near-optimal within tolerance Δ and, when $x_i^0 \neq x_i^\star$, prefers $x_i^\star$ to x_i^0 by at least δ. In this case, we say that $\mathcal{M}$ separates instance i. The loss $\mathcal{L}(\mathcal{M};\mathcal{D}) \in [0,1]$ is thus the fraction of instances that $\mathcal{M}$ fails to separate, and $\mathcal{L}(\mathcal{M};\mathcal{D}) = 0$ if and only if $\mathcal{M}$ separates $\mathcal{D}$.

We frame the formulation correction problem as finding a formulation minimizing the expected separation loss: $\min_{\mathcal{M}} \mathbb{E}_{\mathcal{D}\sim P}[\mathcal{L}(\mathcal{M};\mathcal{D})]$. Since P is unknown, we minimize $\mathcal{L}(\mathcal{M};\mathcal{D})$ on the observed dataset $\mathcal{D}$ and assess generalization on held-out instances. As there can be multiple equivalent formulations differing widely in size and readability, our method additionally aims for a parsimonious transformation of $\mathcal{M}_0$ that remains interpretable for practitioners maintaining $\mathcal{M}_0$.

4 Methodology

Our framework is illustrated in Figure 1. We maintain an incumbent formulation $\mathcal{M} = (c, \mathcal{X})$, initialized by removing from $\mathcal{M}_0$ any constraints that render a target solution infeasible. We then iteratively generate, evaluate, and incorporate candidate transformations. Candidates are generated via two complementary paradigms that combine optimization and LLM agents in opposite directions. In the *bottom-up* paradigm, optimization finds linear inequalities separating solutions of the incumbent formulation from the targets, which an LLM agent contextualizes and extrapolates into meaningful constraints or constraint families. In the *top-down* paradigm, an LLM agent proposes a transformation template whose learnable terms are calibrated through optimization. Candidates are evaluated using the separation loss (1), before and after objective calibration, and an improving candidate becomes the new incumbent. These steps are applied sequentially and repeated until the computational budget is exhausted; Algorithm C.1 gives the full procedure.

Finding an improved formulation presents several challenges: (i) handling multiple and general formulation errors; (ii) navigating the vast space of possible formulations; (iii) constructing inequalities and objectives that apply across different instance sizes; (iv) producing interpretable transformations; and (v) generalizing to unseen instances under limited data. We next detail the formulation transformations in our system and how they address these challenges.

4.1 Bottom-up: cuts and contextualization

Cut search. A *cut* for formulation $\mathcal{M}$ is a linear inequality satisfied by every target solution $x_i^\star$ but excludes at least one of the projected optimal solutions $\hat{x}_i$ of $\mathcal{M}$. We seek to add cuts to the current incumbent formulation $\mathcal{M}$ that is not yet fully correct. Challenges (ii) and (iii) immediately appear. To be deployed across instances, a cut has to be a function of the instance data d, which leads to a large search space. We restrict this space by considering cuts $a(d)^\top x \leq b(d)$ where the components $a(d)$ and $b(d)$ are affine functions of the data. Moreover, to handle varying instance sizes (e.g.,

VRP with different number of customers), we employ *family-level weights* to define the cuts: each coefficient has the form $a(d)_k = \alpha_{t(k)}^\top \phi_k(d)$, where $\phi_k(d)$ is the data associated with variable k and $\alpha_{t(k)}$ is the weight vector (to be estimated) for its the family $t(k)$ of this variable; similarly $b(d) = \beta^\top \phi_0(d)$, where $\phi_0(d)$ represents data not associated to a particular variable family and β are the weights to be estimated. More details about this concept can be found in Appendix E.1.

We infer α and β through a MILP (see Appendix E.1) that maximizes the cut quality score

$$\max_{\alpha,\beta,z,\eta} \quad \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} z_i - \rho\big(\|\alpha\|_0 + \|\beta\|_0\big) + \varepsilon\eta, \tag{2}$$

subject to validity for every $x_i^\star$, where z_i indicates whether $\hat{x}_i$ violates the cut by at least a margin $\eta \geq \underline{\eta} > 0$. The score therefore rewards cuts that eliminate a large fraction of the solutions from formulation $\mathcal{M}$, favors simple and sparse structure, and, through the small term $\varepsilon\eta$ with $0 < \varepsilon \ll 1$, breaks ties in favor of stronger separation.

Sparsity regularization limits overfitting, promoting cut generalization to unseen instances, and also improves interpretability, since meaningful combinatorial constraints often involve only few variables and instance features (Challenges (iv)-(v)). We sweep the parameter ρ, generating cuts from the sparsest nonempty solution to the unregularized one, and evaluate them by separation loss. These candidates are then passed to the *cut contextualization* step for polishing and extrapolation.

Cuts contextualization. The cut-search procedure identifies sparse numerical structures that explain the observed corrections, but the resulting coefficients need not directly reveal the underlying modeling intent. We therefore invoke an LLM agent to contextualize each discovered cut using the full formulation $\mathcal{M}$. The agent first *polishes* the cut by proposing semantically plausible transformations, such as rounding coefficients, removing negligible terms, and introducing contextually relevant terms. For example, the search may return the cut $0.98x_i + 1.03x_j - 1.01y_k \leq 0.97$, while the underlying combinatorial structure is more naturally expressed as $x_i + x_j - y_k \leq 1$. Thus, the search identifies the relevant geometry, while the LLM uses the formulation context to recover a cleaner symbolic representation, which is crucial for interpretability. Importantly, the LLM does not unilaterally select the repair: it proposes a collection of candidate transformations, which are evaluated using the separation loss (1), and the lowest-loss variant is retained.

The agent then *extrapolates* the accumulated cuts into indexed constraint families. For instance, $x_{3,7} \leq y_3$ may suggest the family $x_{ij} \leq y_i,\ \forall i, j$. Such families are ubiquitous (e.g., Example 1) and can encode many individual constraints. With limited data (Challenge (v)), observed corrections may reveal only a few members of the underlying family, making extrapolation important for recovering its broader structure. Table B.5 provides cut contextualization examples from our experiments.

4.2 Top-down: templates and calibration

Some target formulations introduce additional variables and therefore cannot be recovered through cuts alone (Theorem 1). The *top-down paradigm* addresses this limitation by having an LLM agent propose a *template* for the transformation, potentially introducing auxiliary variables, modifying variable domains, and adding constraint families with unspecified parameters. Optimization then calibrates these parameters to preserve target solutions while maximally separating unwanted ones.

Template generation. Rather than asking the LLM to produce a fully specified repair, we use it to propose a *structural hypothesis* for how the formulation should change. The agent receives the incumbent formulation $\mathcal{M}$, constraints removed during preprocessing, and samples of mismatched pairs $(\hat{x}_i, x_i^\star)$, where $\hat{x}_i$ is a projected optimal solution of $\mathcal{M}$, i.e., $\hat{x}_i = \Pi_0(\hat{y}_i)$ for $\hat{y}_i \in \mathcal{S}(d_i; \mathcal{M})$. It proposes a template τ specifying *what* should change, while leaving its learnable parameters unspecified. This separation is important: the agent identifies a semantically meaningful modeling mechanism, while the subsequent calibration step determines whether and how it can be instantiated without excluding the observed targets. Table B.6 provides template examples from our experiments. Formally, a template τ specifies (i) new auxiliary variables z with domain $Z_\tau(d)$; (ii) constraints $g_j(y, z, w; d) \leq 0$, $j \in J_\tau(d)$, parameterized by weights w to be calibrated; and (iii) a possibly modified domain $\mathcal{Y}_\tau(d)$ for the existing variables y. Dependence on d allows templates to generalize

across instance sizes. The functions g_j are bi-affine in (y, z) and w, yielding the updated feasible set

$$\mathcal{X}_\tau(d; w) = \{(y, z) : y \in \mathcal{Y}(d),\ z \in Z_\tau(d),\ g_j(y, z, w; d) \le 0,\ \forall j \in J_\tau(d)\}. \tag{3}$$

A template does not change the objective function and assigns zero objective coefficients to new variables; their contribution is determined separately by the objective-calibration step in Section 4.3.

For example, in the VRP setting of Example 1, the missing long-route penalty cannot be recovered through cuts over the existing variables alone. A template can introduce binary variables z_ℓ, $\ell \in \mathcal{L}$, indicating long routes and constraints $g_\ell(x, z, w; d) \le 0$, $\ell \in \mathcal{L}$, linking them to route lengths. Calibration may instantiate these as $\sum_{i,j \in C} d_{ij} x_{ij\ell} \le D + M z_\ell$. Thus, the LLM agent identifies the missing modeling structure, while optimization determines its numerical parameters, enabling repairs involving multiple connected modeling changes (Challenge (i)).

Template completion. We calibrate the template weights w to preserve all observed targets while maximally separating the optimal solutions of the current formulation $\mathcal{M}$. Preservation requires each $x_i^\star$ to admit a lifting $y_i^\star$ into the non-primary variables of $\mathcal{M}$ and an assignment of the new variables $z_i^\star$ such that $(y_i^\star, z_i^\star)$ belongs to the new feasible set $\mathcal{X}_\tau(d_i; w)$. We therefore solve an optimization problem with max-min structure to jointly calibrate the template parameters w against the best completions $\hat{z}_i$ of the new variables. We solve this problem via constraint generation and discard any template where no weights allow every target to be feasibly lifted; see Appendix E.2.

4.3 Objective calibration

Cuts and templates modify the feasible set, but misspecification may also lie in the objective. Moreover, auxiliary variables introduced by templates may matter via their cost. Thus, every candidate formulation $\mathcal{M} = (c, \mathcal{X})$ undergoes *objective calibration*, keeping $\mathcal{X}$ fixed while learning c. Calibration balances two goals: (i) each target $x_i^\star$ should admit a feasible lifting $y_i^\star$ that is nearly optimal under the calibrated objective, allowing some suboptimality to accommodate potentially suboptimal target solutions in practice, (ii) the lifted target $y_i^\star$ should outperform the current model solution $\hat{y}_i$ by as large a margin as possible. Appendix E.3 provides the resulting optimization problem.

5 Numerical Results

5.1 Experimental Setup

Dataset. We introduce a benchmark of 38 scenarios spanning nine classes of LPs and MILPs. The benchmark combines classical optimization problems with realistic applications, including scenarios motivated by resource allocation in hyperscalers. The scenarios cover a broad range of formulation errors along two dimensions: the affected component (objective, constraints, variables, domains) and nature of the error (extraneous, missing, miscalibrated, local, global). Many scenarios deliberately combine multiple forms of misspecification. Each scenario specifies an initial formulation $\mathcal{M}_0$ and a target formulation $\mathcal{M}^\star$. Appendix D gives them in full and Table D.1 summarizes the taxonomy. We will release the benchmark for reproducibility and future research on formulation repair.

Baselines and protocol. We compare our framework against (i) the initial formulation, (ii) the data-driven methods of Hewitt & Frejinger (2020); Bayani et al. (2024), which we denote AOP, and (iii) an LLM-only baseline that generates the complete optimization model as `gurobipy` code from the initial code and pairs of model and target solutions. For each scenario, every method is run 5 times using $N = 128$ training instances and evaluated on a disjoint set of 256 test instances.

Metrics. We evaluate the final incumbent formulation $\mathcal{M}$ on both training and test instances. On the training set, we report the separation loss $\mathcal{L}$ and the *separation rate*, defined as the fraction of runs with $\mathcal{L} = 0$. On the test set, we report four metrics. The *feasibility rate* is the fraction of test instances for which the solution produced by $\mathcal{M}$ is feasible under $\mathcal{M}^\star$. The *optimality rate* is the fraction for which that solution is feasible and within 1% of the optimal value under $\mathcal{M}^\star$ in relative gap. The *recovery rate* is the proportion of scenarios for which this optimality criterion holds on **all 256 test instances**. These metrics are averaged over the 5 runs of each method. Finally, the *recoverable rate* is the proportion of scenarios for which at least one of the 5 runs attains recovery, capturing the fraction of scenarios that are amenable to formulation repair by TACIT.

Table 1: TACIT against LLM baselines under different LLM backbone models.

	Model	Training $\mathcal{L}\downarrow$	Training sep. ↑	Training feas. ↑	Training opt. ↑	Test feas. ↑	Test opt. ↑	Test recovery ↑	Test recoverable ↑
AOP	AOP linear (c=16)	0.94	0.0%	0.57	0.07	0.57	0.04	0.0%	0.0%
AOP	AOP tree (c=16)	0.93	0.0%	0.58	0.10	0.57	0.07	0.0%	0.0%
LLM-only	DeepSeek-V4-F	0.64	18.9%	0.57	0.37	0.57	0.37	19.5%	31.6%
LLM-only	Kimi-K2.6	0.56	29.5%	0.66	0.46	0.66	0.47	30.0%	36.8%
LLM-only	GPT-5.6-sol	0.40	47.9%	0.74	0.59	0.75	0.59	46.8%	60.5%
TACIT	DeepSeek-V4-F	0.19	57.4%	0.79	0.64	0.78	0.61	45.3%	63.2%
TACIT	Kimi-K2.6	**0.12**	**66.8%**	0.87	0.77	0.85	0.73	55.3%	71.1%
TACIT	GPT-5.6-sol	0.12	66.3%	**0.89**	**0.79**	**0.88**	**0.76**	**64.2%**	**78.9%**

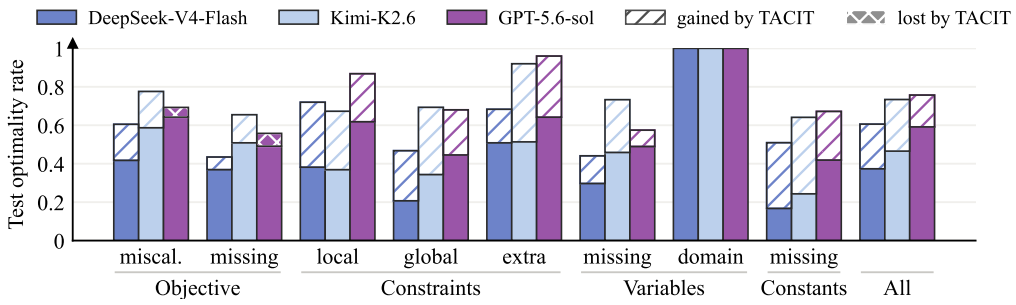


Figure 2: Change in optimality rate induced by TACIT under different LLM backbone models.

5.2 Results

We run TACIT with three LLM backbones (DeepSeek-V4-Flash, Kimi-K2.6, GPT-5.6-sol) for cut contextualization and template generation, using five iterations of the framework per run. To match this budget, the LLM-only baseline is given five attempts per run, and its best formulation by training separation loss is kept. Additional implementation details are provided in Appendix C.

TACIT substantially improves formulation repair across LLM backbones. Table 1 compares TACIT with LLM-only repair and AOP. TACIT improves every test metric across all backbones. AOP (Adapted Optimization Problem) never recovers a correct formulation: increasing its penalty weight c makes solutions significantly suboptimal before it makes them feasible (Table B.4 in the appendix). With DeepSeek-V4-Flash and Kimi-K2.6, TACIT nearly doubles the scenario recovery and recoverable rates compared to LLM-only: $19.5\% \rightarrow 45.3\%$ and $31.6\% \rightarrow 63.2\%$ respectively for DeepSeek-V4-Flash; $30\% \rightarrow 55.3\%$ and $36.8\% \rightarrow 71.1\%$, for Kimi-K2.6. Importantly, TACIT remains beneficial even with the substantially stronger GPT-5.6-sol, where TACIT obtains additional 17.4 and 18.4 percentage points, respectively, and achieves a total 64.2% recovery and 78.9% recoverable rate of scenarios. Overall, TACIT improves performance across all reasoning models, with larger gains for weaker models, and is able to lift weaker models to approach the performance of much stronger models. Table 1 further shows that feasibility is recovered reliably across all backbones, suggesting that feasibility is not the primary bottleneck in formulation recovery; the harder challenge is recovering target-optimal behavior.

TACIT provides the largest gains on structurally demanding misspecifications. Figure 2 breaks down test optimality by misspecification type, revealing a clear spectrum of difficulty. Some errors, such as variable-domain misspecifications, can be readily identified from the formulation context. In contrast, LLM-only baselines struggle when repair requires reconstructing structure absent from the initial formulation, such as missing variables or global constraints. TACIT substantially improves performance on these harder categories across backbones. For example, on missing variables, TACIT improves test optimality $0.30 \rightarrow 0.44$ with DeepSeek-V4-Flash and $0.46 \rightarrow 0.74$ with Kimi-K2.6; on global constraints, the improvements are $0.21 \rightarrow 0.47$ and $0.35 \rightarrow 0.69$. Similar gains arise for extra constraints. These results suggest that TACIT is most valuable when formulation repair goes beyond recognizing local errors and requires reconstructing missing model structure.

LLM reasoning and optimization play complementary roles. We ablate TACIT with DeepSeek-V4-Flash in Table 2. The top-down pathway is already strong, achieving recovery and recoverability

Table 2: Ablation of TACIT components using DeepSeek-V4-Flash. All variants share preprocessing and objective calibration. `Template` only uses LLM-guessed constants.

	Training				Test			
	$\mathcal{L}\downarrow$	sep. ↑	feas. ↑	opt. ↑	feas. ↑	opt. ↑	recovery ↑	recoverable ↑
Obj. calibration only	0.76	5.8%	0.41	0.23	0.41	0.23	5.3%	5.3%
Cuts	0.35	33.2%	0.57	0.33	0.54	0.27	14.7%	21.1%
Cuts + contextualize	0.35	33.2%	0.59	0.35	0.57	0.30	17.9%	26.3%
Template	0.60	22.6%	0.56	0.39	0.56	0.39	21.1%	31.6%
Template + complete	0.31	47.9%	0.77	0.59	0.77	0.58	41.1%	57.9%
TACIT	**0.19**	**57.4%**	**0.79**	**0.64**	**0.78**	**0.61**	**45.3%**	**63.2%**

of 41.1% and 57.9%, but incorporating the bottom-up pathway further improves them to 45.3% and 63.2%. The results also highlight the importance of combining LLM reasoning with optimization: the best pure approach, whether based only on optimization-driven cut generation or on LLM-only templates, achieves recovery and recoverability of 21.1% and 31.6%, less than half of the full system. Adding optimization-based calibration to the strongest pure approach, LLM-generated templates, raises recovery $21.1\% \rightarrow 41.1\%$ and recoverability $31.6\% \rightarrow 57.9\%$. The breakdown in Table B.1 helps explain this complementarity: cut-based inference is particularly useful for local constraint errors, whereas templates are stronger when the repair requires new modeling structure, such as auxiliary variables. Together, these results show that TACIT benefits both from combining LLM reasoning with optimization and from combining top-down semantic reasoning with bottom-up evidence from observed decisions.

Performance under limited training data. Table B.2 shows that top-down inference is stable across training sizes, whereas bottom-up improves by 12.6 percentage points from 32 to 256 instances. TACIT combines both behaviors, maintaining strong performance across training sizes.

Robustness to suboptimal feedback. We relax the assumption that observed target decisions are optimal, allowing up to 5% suboptimality. Test feasibility remains unchanged (0.78 to 0.79), while test recovery decreases by 4.8 percentage points. This is a deliberately challenging setting: TACIT may never observe a target-optimal solution, yet is evaluated on whether it recovers target-optimal behavior. These results suggest that imperfect feedback still conveys useful structural information, allowing TACIT to remain robust even as exact preference recovery becomes harder; see Table B.3.

The recovered transformations remain explicit and inspectable. Beyond predictive performance, TACIT returns modifications to the optimization formulation rather than an opaque correction of its outputs. Section B.2.3 illustrates this across three scenarios. In TP-Scenario2, it recovers the target exactly, adding $x_{ij} \leq 24 y_{ij}$ for all $i \in I$, $j \in J$, and $\sum_{i \in I} y_{ij} \leq 2$ for all $j \in J$. In the more challenging MCF-Scenario2, it fixes the objective function by introducing auxiliary variables and the cuts $f_{ij} \leq u_{ij} y_{ij}$ for all $i \in V, j \in O_i$, plus the valid but redundant $0 \cdot y_{ij} \leq f_{ij}$. Results are not always as clean. In GPU-Scenario4, the recovered objective $-1.90 \sum_{i \in I} s_i z_i - 0.10 \sum_{i \in I} z_i - 2.42 \sum_{i \in I} \sum_{j \in J_i} x_{ij}$ is equivalent under the problem constraints to a scaling of the target $-\sum_{i \in I} s_i z_i$ with a small tie-breaking term $-\epsilon \sum_{i \in I} z_i$. Table B.5 further shows how the cut-contextualization agent turns numerical cuts into interpretable constraints. Across inspected examples, TACIT recovers recognizable modeling constructs, such as linking constraints, combinatorial restrictions, auxiliary variables, and structured objective terms, rather than opaque instance-specific corrections. Even when non-canonical terms are present, the recovered formulations remain directly inspectable.

6 Conclusion

Formulation repair is challenging: the source of misspecification is unknown, modeling logic may be absent from the initial formulation, and sparse feedback can be consistent with many possible repairs. We introduced TACIT, an end-to-end framework combining LLM-guided semantic reasoning with optimization-based search and validation to address this challenge. Across diverse formulation errors, it substantially improves recovery over the baselines, especially on structurally demanding misspecifications, while remaining effective with limited and imperfect feedback. Our results demonstrate that user corrections provide sufficient signal to systematically improve optimization models, a step toward models that learn from their mistakes.

Reproducibility Statement

All 38 benchmark scenarios, including the base problems, initial and target formulations, and instance generation procedures, are specified in Appendix D. The cut-search MILP, template completion, and objective calibration problems are given in Appendices E.1, E.2, and E.3, respectively. Appendix C details the prompts, compute budgets, and hyperparameters used for numerical experiments. The proof of Theorem 1 is in Appendix A.1. The full code will be released upon acceptance.

AI Use Statement

Beyond their role in the proposed framework, LLMs were used to polish the writing, search for related work, and help in implementing the benchmark scenarios. All text, references, and scenarios were verified by the authors, who take full responsibility for the manuscript and its artifacts.

References

Abdelhakim Abdellaoui, Ayoub Boufous, Issmail El Hallaoui, Loubna Benabbou, Francois Aube, and Mouloud Amazouz. Learning implicit feasibility constraints for real-world routing and scheduling: Application to log transportation. *arXiv preprint arXiv:2606.05353*, 2026.

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. Optimus-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*, 2024.

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*, 2026a.

Ruicheng Ao, David Simchi-Levi, and Xinshang Wang. Solver-in-the-loop: Mdp-based benchmarks for self-correction and behavioral rationality in operations research. *arXiv preprint arXiv:2601.21008*, 2026b.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using llms. *arXiv preprint arXiv:2411.01679*, 2024.

Anil Aswani, Zuo-Jun Max Shen, and Auyon Siddiq. Inverse optimization with noisy data. *Operations Research*, 66(3):870–892, 2018. doi: 10.1287/opre.2017.1705.

Saumil Baxi, Kayla Cummings, Alexandre Jacquillat, Sean Lo, Rob McDonald, Konstantina Mellou, Ishai Menache, and Marco Molinaro. Online rack placement in large-scale data centers: Online sampling optimization and deployment. *Operations Research*, 2026. doi: 10.1287/opre.2025.1992.

M Bayani, Y Adulyasak, and LM Rousseau. Learning and modeling implicit constraints in optimization models through decision trees. *Les Cahiers du GERAD ISSN*, 711:2440, 2024.

Omar Besbes, Yuri Fonseca, and Ilan Lobel. Contextual inverse optimization: Offline and online learning. *Operations Research*, 73(1):424–443, 2025.

Aykut Bulut and Ted K. Ralphs. On the complexity of inverse mixed integer linear optimization. *SIAM Journal on Optimization*, 31(4):3014–3043, 2021. doi: 10.1137/20M1377369.

Felipe Caro, Jean-Edouard Colliard, Elena Katok, Axel Ockenfels, Nicolas Stier-Moses, Catherine Tucker, and D. J. Wu. Introduction to the special issue on the human-algorithm connection. *Management Science*, 72(1):1–13, 2026. doi: 10.1287/mnsc.2023.intro.v72.n1.

Timothy CY Chan, Rafid Mahmood, and Ian Yihang Zhu. Inverse optimization: Theory and applications. *Operations Research*, 73(2):1046–1074, 2025.

Hao Chen, Gonzalo E Constante-Flores, and Can Li. Diagnosing infeasible optimization problems using large language models. *INFOR: Information Systems and Operational Research*, 62(4): 573–587, 2024.

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, 2026. doi: 10.1287/ijds.2025.0074.

Adejuyigbe Fajemisin, Donato Maragno, and Dick den Hertog. Optimization with constraint learning: A framework and survey. *arXiv preprint arXiv:2110.02121*, 2021.

Mike Hewitt and Emma Frejinger. Data-driven optimization model customization. *European Journal of Operational Research*, 287(2):438–451, 2020.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025.

Akira Kitaoka. Inverse mixed-integer programming: Learning constraints then objective functions. *arXiv preprint arXiv:2510.04455*, 2025.

Mohit Kumar, Samuel Kolb, Luc De Raedt, and Stefano Teso. Learning mixed-integer linear programs from contextual examples. *arXiv preprint arXiv:2107.07136*, 2021.

Donato Maragno, Holly Wiberg, Dimitris Bertsimas, S. Ilker Birbil, Dick den Hertog, and Adejuyigbe O. Fajemisin. Mixed-integer optimization with constraint learning. *Operations Research*, 73(2):1011–1028, 2025. doi: 10.1287/opre.2023.2523.

Mahsa Moghaddass and Daria Terekhov. Inverse integer optimization with multiple observations. *Optimization Letters*, 15(4):1061–1079, 2021. doi: 10.1007/s11590-021-01721-4.

Peyman Mohajerin Esfahani, Soroosh Shafieezadeh-Abadeh, Grani A. Hanasusanto, and Daniel Kuhn. Data-driven inverse optimization with imperfect information. *Mathematical Programming*, 167(1):191–234, 2018. doi: 10.1007/s10107-017-1216-6.

Ke Ren, Peyman Mohajerin Esfahani, and Angelos Georghiou. Inverse optimization via learning feasible regions. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 51471–51488, 2025.

Yingcong Tan, Daria Terekhov, and Andrew Delong. Learning linear programs from optimal decisions. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. Optimai: Optimization from natural language using llm-powered ai agents. *arXiv preprint arXiv:2504.16918*, 2025.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024.

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, Qingcan Kang, Jiahui Duan, Tao Zhong, Mingxuan Yuan, Jia Zeng, Yuan Wang, Gang Chen, and Dongxiang Zhang. A survey of optimization modeling meets LLMs: Progress and future directions. *arXiv preprint arXiv:2508.10047*, 2025.

Jihai Zhang, Wei Wang, Siyan Guo, Li Wang, Fangquan Lin, Cheng Yang, and Wotao Yin. Solving general natural-language-description optimization problems with large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track)*, pp. 483–490, 2024.

A PROOFS

A.1 LIMITATION OF EMBEDDED ML MODELS FOR FORMULATION CORRECTION

As mentioned in Section 2, one approach for correcting formulations is by using an ML model to represent the user action of mapping from "incorrect" to "target" solution, and embedding this model within the optimization itself (Hewitt & Frejinger, 2020; Bayani et al., 2024; Abdellaoui et al., 2026). That is, given an initial formulation

$$\mathcal{M}_0(d) = \min \left\{ c_0(d)^\top x : x \in \mathcal{X}_0(d) \right\},$$

consider the modified problem

$$\mathcal{M}_{0,f}(d) = \min \left\{ c_0(d)^\top x : x = f(x),\ x \in \mathcal{X}_0(d) \right\},$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a machine-learning model representing the user's correction of a solution x into a solution $f(x)$. The function f is learned so that, ideally, the optimal solutions of $\mathcal{M}_{0,f}(d)$ match the optimal solutions of the target formulation $\mathcal{M}^\star(d)$ for every instance:

$$\arg\min \mathcal{M}_{0,f}(d) = \mathcal{S}(d; \mathcal{M}^\star) \qquad \text{for all } d.$$

Some approaches relax the fixed-point equality $x = f(x)$ through a Lagrangian penalty, i.e., the objective becomes $c_0(d)^\top x - \lambda \|x - f(x)\|_\infty$, but we stick to the original fixed-point equality to avoid the extra parameter λ.

The following result shows that this approach cannot represent all formulation corrections. In particular, the limitation already arises when the initial formulation is missing a nonlinear term in the objective. Although such a term can be represented in a MILP using auxiliary variables and constraints, it cannot, in general, be recovered by restricting the feasible set of the original variables. In fact, the impossibility persists even if new constraints are added. We note that through the use of templates, our proposed system is able to handle such corrections.

Theorem 1. *There exist an initial formulation $\mathcal{M}_0$ and a target formulation $\mathcal{M}^\star$ such that no function f and no set K of additional constraints make the optimal solutions of*

$$\mathcal{M}_{0,f,K}(d) \coloneqq \min \left\{ c_0(d)^\top x : x = f(x),\ x \in \mathcal{X}_0(d),\ x \in K \right\}$$

equal to the optimal solutions of $\mathcal{M}^\star(d)$ for every instance d.

Proof. Consider the family of instances parametrized by the scalar $d \in \mathbb{R}$. The target formulation is

$$\mathcal{M}^\star(d) = \min \left\{ dx + |x| : x \in \{-1, 0, 1\} \right\},$$

whereas the initial formulation, which omits the nonlinear objective term, is

$$\mathcal{M}_0(d) = \min \left\{ dx : x \in \{-1, 0, 1\} \right\}.$$

The target formulation can equivalently be written as the MILP

$$\begin{aligned} \mathcal{M}^\star(d): \qquad \min_{x, y_1, y_2} \quad & dx + y_1 + y_2 \\ \text{s.t.} \quad & x = y_1 - y_2, \\ & x \in \{-1, 0, 1\}, \\ & y_1, y_2 \in \{0, 1\}. \end{aligned}$$

Thus, $\mathcal{M}^\star$ can be obtained from $\mathcal{M}_0$ by introducing auxiliary variables and constraints together with the missing objective term.

For a function $f : \mathbb{R} \to \mathbb{R}$ and set $K \subseteq \mathbb{R}$, in this case we have

$$\mathcal{M}_{0,f,K}(d) = \min \left\{ dx : x = f(x),\ x \in \{-1, 0, 1\},\ x \in K \right\}.$$

We claim that for every f and K, there exists some d such that

$$\arg\min \mathcal{M}_{0,f,K}(d) \neq \Pi_0(\mathcal{S}(d; \mathcal{M}^\star)).$$

To see that, let

$$S \coloneqq \{x : x = f(x),\ x \in \{-1, 0, 1\},\ x \in K\}$$

be the feasible set of $\mathcal{M}_{0,f,K}(d)$. Importantly, S is independent of d, and hence $\mathcal{M}_{0,f,K}(d) = \min_{x \in S} dx$.

Consider $d = \{-2, 0, 2\}$. For these instances, the target optimal solutions are

$$\mathcal{S}(-2; \mathcal{M}^\star) = \{1\}, \qquad \mathcal{S}(0; \mathcal{M}^\star) = \{0\}, \qquad \mathcal{S}(2; \mathcal{M}^\star) = \{-1\}.$$

Suppose by contradiction that the optimal-solution sets of $\mathcal{M}_{0,f,K}(d)$ and $\mathcal{M}^\star(d)$ agree for all three values of d. Agreement at $d = -2$ requires 1 to be feasible for $\mathcal{M}_{0,f,K}(-2)$, and therefore $1 \in S$. Similarly, agreement at $d = 2$ requires $-1 \in S$. When $d = 0$, however, the objective of $\mathcal{M}_{0,f,K}(0)$ is identically zero over S, so every feasible point is optimal:

$$\arg\min \mathcal{M}_{0,f,K}(0) = S.$$

Since $-1, 1 \in S$, it follows that

$$\{-1, 1\} \subseteq \arg\min \mathcal{M}_{0,f,K}(0).$$

This contradicts $\mathcal{S}(0; \mathcal{M}^\star) = \{0\}$. Therefore no such f and K exist. □

B Extended Numerical Results

B.1 Additional Results

Table B.1: Test optimality rate per misspecification type, with DeepSeek-V4-Flash as LLM backbone.

	Objective		Constraints			Variables		Constants	
Instances	Miscal. *7*	Missing *6*	Local *10*	Global *15*	Extra *5*	Missing *10*	Domain *2*	Missing *16*	All *38*
Initial model	0.24	0.22	0.20	0.11	0.17	0.18	0.01	0.13	0.17
Calib. only	0.49	0.26	0.20	0.11	0.28	0.21	0.01	0.13	0.23
Cuts	0.52	0.20	0.41	0.07	0.40	0.12	0.00	0.06	0.27
Cuts + contextualize	0.52	0.22	0.43	0.11	0.41	0.13	0.00	0.08	0.30
Template	0.59	0.37	0.29	0.24	0.50	0.31	**1.00**	0.17	0.39
Template + complete	0.60	**0.46**	0.55	**0.49**	**0.77**	0.43	**1.00**	**0.51**	0.58
All	**0.61**	0.43	**0.72**	0.47	0.68	**0.44**	**1.00**	**0.51**	**0.61**

Table B.2: Recovery rate of the final formulation against the number of training instances per scenario.

Training instances	32	64	128	256
Calibration	4.7%	5.8%	5.3%	6.3%
Cuts + contextualize	7.9%	13.7%	17.9%	20.5%
Template + complete	**36.8%**	43.2%	41.1%	40.5%
All	36.3%	**45.8%**	**45.3%**	**43.2%**

Table B.3 compares our framework trained on optimal targets with training on targets that are up to 5% suboptimal. Let $Z^\star(d)$ be the optimal value of $\mathcal{M}^\star(d)$ and, for a solution y, let $Z(y; d) = c^\star(d)^\top y$. For each instance whose target differs from the initial solution, we draw $x \sim U(0, \Delta)$ with $\Delta = 0.05$ and replace the target by a maximizer of $Z(y; d)$ over $\{y \in \mathcal{X}^\star(d) : Z(y; d) \leq Z^\star(d) + x|Z^\star(d)|\}$, i.e., the worst target-feasible solution within relative gap x. When x^0 admits a feasible lifting under $\mathcal{M}^\star(d)$ with best objective $Z_{\text{init}}(d)$, we add $Z(y; d) \leq Z_{\text{init}}(d) - \delta|Z^\star(d)|$ with $\delta = 0.005$, so the new target still beats the initial solution by a relative margin of δ (instances where this set is empty are discarded).

The Adapted Optimization Problem (Hewitt & Frejinger, 2020; Bayani et al., 2024) learns a mapping from the initial model's optimum to the observed plan and penalizes deviations from that prediction,

Table B.3: Performance of TACIT with DeepSeek-V4-Flash backbone on datasets with suboptimal target solutions.

	Training			Test			
Dataset	$\mathcal{L}\downarrow$	feas. ↑	opt. ↑	feas. ↑	opt. ↑	rec. % ↑	recoverable ↑
Optimal targets	0.19	0.79	0.64	0.78	0.61	45.3%	63.2%
Suboptimal targets	0.19	0.81	0.58	0.79	0.53	40.5%	52.6%

so it can only imitate the observed plans rather than recover the rule that produced them. In Table B.4, raising c makes its solutions satisfy the hidden constraint more often (feasibility from 0.51 to 0.62 and 0.52 to 0.64) but only by pulling them toward typical plans, so they stop being optimal (optimality from 0.13 to 0.01 and 0.15 to 0.04) and no run ever recovers the ground truth.

Table B.4: AOP methods across different values of the penalty parameter c.

	Training				Test			
Model	$\mathcal{L}\downarrow$	sep. ↑	feas. ↑	opt. ↑	feas. ↑	opt. ↑	rec. % ↑	recoverable ↑
Initial	**0.85**	0.0%	0.46	**0.17**	0.46	**0.17**	-	-
AOP linear (c=1)	0.90	0.0%	0.50	0.15	0.51	0.13	0.0%	0.0%
AOP linear (c=16)	0.94	0.0%	0.57	0.07	0.57	0.04	0.0%	0.0%
AOP linear (c=256)	0.98	0.0%	0.61	0.02	0.62	0.01	0.0%	0.0%
AOP tree (c=1)	0.91	0.0%	0.52	**0.17**	0.52	0.15	0.0%	0.0%
AOP tree (c=16)	0.93	0.0%	0.58	0.10	0.57	0.07	0.0%	0.0%
AOP tree (c=256)	0.95	0.0%	**0.65**	0.07	**0.64**	0.04	0.0%	0.0%

B.2 Example Repaired Formulation

B.2.1 Bottom-Up

Table B.5 shows two cases where refining a cut recovers the missing constraint exactly. In GPU Scenario 3, the raw cut has near-degenerate coefficients. Because the variables z_i are binary, it still encodes the implication $z_0 = 1 \Rightarrow z_i = 1$ for all $i > 0$. LLM rewrites it as the family in the more interpretable form $z_0 \leq z_i$ for all $i > 0$. In contrast, the template-based search never recovers a correct formulation on this scenario. In WA Scenario 3, the raw cut is already interpretable but applies to a single day and is not sufficient for out-of-sample test instances, so that raw cuts reach a test optimality of 0. The LLM refinement keeps employees 3 and 5 as fixed anchors, extends the cut over the day index d, and recovers the ground-truth constraint exactly.

Table B.5: Example of cut processing on two scenarios.

GPU Scenario 3: *demand 0 can only be satisfied whenever all other demands are satisfied.*	
Missing constraint:	$z_0 \leq z_i,\ \forall i \in I,\ i > 0$
Raw cut from ($\mathcal{P}_{\text{cut}}$):	$0.9999\, z_0 - 0.0001\, z_1 - 0.0001\, z_2 - 0.9996\, z_3 - 0.0001\, z_4 \leq 0$
LLM refined cut:	$z_0 \leq z_i,\ \forall i \in I,\ i > 0$
WA Scenario 3: *employees 3 and 5 can never work on the same day.*	
Missing constraint:	$x_{3,d} + x_{5,d} \leq 1,\ \forall d \in D$
Raw cut from ($\mathcal{P}_{\text{cut}}$):	$0.5\, x_{3,7} + 0.5\, x_{5,7} - 0.5 \leq 0$
LLM refined cut:	$x_{3,d} + x_{5,d} \leq 1,\ \forall d \in D$

B.2.2 Top-Down

Table B.6 shows three templates proposed by the LLM that recover the ground truth, and how the fitting step turns each into a concrete correction. In VRP Scenario 2, the template adds a variable z that bounds every route cost. It has no free constants so no fitting in necessary. In TP Scenario

3, the template adds both a new integer variable $y_{i,j}$ and a new constant, the batch size L. The constraint calibration ($\mathcal{P}_{\text{constr}}$) recovers the true value $L = 2$. In CFL Scenario 7, the LLM proposes a general per-pair cap $x_{i,j} \leq u_{i,j}\, y_i$ without knowing which facility is restricted. Here, the constraint calibration ($\mathcal{P}_{\text{constr}}$) fit the constants $u_{i,j}$ to capture the local rule: every cap is set to 1, which leaves the original constraint unchanged, except for facility 4, whose cap is set to 0.5.

Table B.6: Example of template processing on three scenarios. Templates proposed by the LLM, before (raw) and after fitting the new constants with ($\mathcal{P}_{\text{constr}}$). Fitted values are shown in blue.

	Template (raw)	**Template (fitted)**
CVRP Scenario 2: *the ground truth minimizes the longest route instead of the total distance.*		
Variables:	$z \in \mathbb{R}_{\geq 0}$	$z \in \mathbb{R}_{\geq 0}$
Constants:	–	–
Constraints:	$\sum_{i \in N} \sum_{\substack{j \in N \\ j \neq i}} c_{i,j}\, x_{i,j,k} \leq z,\ \forall k \in K$	$\sum_{i \in N} \sum_{\substack{j \in N \\ j \neq i}} c_{i,j}\, x_{i,j,k} \leq z,\ \forall k \in K$
TP Scenario 3: *goods are shipped in batches of a fixed size.*		
Variables:	$y_{i,j} \in \mathbb{Z},\ \forall i \in I, j \in J$	$y_{i,j} \in \mathbb{Z},\ \forall i \in I, j \in J$
Constants:	L	$L = 2$
Constraints:	$x_{i,j} = L\, y_{i,j},\ \forall i \in I, j \in J$	$x_{i,j} = 2\, y_{i,j},\ \forall i \in I, j \in J$
CFL Scenario 7: *facility 4 can serve at most half of the demand of any customer.*		
Variables:	–	–
Constants:	$u_{i,j},\ \forall i \in I, j \in J$	$u_{4,j} = 0.5,\ u_{i,j} = 1\ (i \neq 4)$
Constraints:	$x_{i,j} \leq u_{i,j}\, y_i,\ \forall i \in I, j \in J$	$x_{4,j} \leq 0.5\, y_4,\ \forall j \in J$
		$x_{i,j} \leq y_i,\ \forall i \in I \setminus \{4\}, j \in J$

B.2.3 Final formulation samples

In these examples, $\mathcal{M}'$ is the final formulation returned by TACIT.

TP Scenario 2. At most $K = 2$ origins may ship to a given destination. Here TACIT recovers the target formulation exactly, up to the big-M constant (24 instead of $M = 1000$):

$$
\begin{aligned}
\mathcal{M}^\star:\ \min_{x,z}\ & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\
\text{s.t.}\ & x \in \mathcal{X}_{\text{TP}}, \\
& x_{ij} \leq M z_{ij}, && \forall i \in I,\ j \in J, \\
& \sum_{i \in I} z_{ij} \leq K, && \forall j \in J, \\
& z_{ij} \in \{0,1\}, && \forall i \in I,\ j \in J.
\end{aligned}
\qquad
\begin{aligned}
\mathcal{M}':\ \min_{x,y}\ & \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \\
\text{s.t.}\ & x \in \mathcal{X}_{\text{TP}}, \\
& x_{ij} \leq 24\, y_{ij}, && \forall i \in I,\ j \in J, \\
& \sum_{i \in I} y_{ij} \leq 2, && \forall j \in J, \\
& y_{ij} \in \{0,1\}, && \forall i \in I,\ j \in J.
\end{aligned}
$$

GPU Scenario 4. The target values the total demand fulfilled rather than the number of requests fulfilled. Since $\mathcal{X}_{\text{GPU}}$ contains $\sum_{j \in J_i} x_{ij} = s_i z_i$, the incumbent objective is equivalent to $-4.32 \sum_{i \in I} s_i z_i - 0.10 \sum_{i \in I} z_i$, i.e. the target objective scaled by 4.32 plus a small tie-breaker on the number of requests.

$$
\begin{aligned}
\mathcal{M}^\star:\ \min_{x,y,z}\ & -\sum_{i \in I} s_i\, z_i \\
\text{s.t.}\ & (x,y,z) \in \mathcal{X}_{\text{GPU}}.
\end{aligned}
\qquad
\begin{aligned}
\mathcal{M}':\ \min_{x,y,z}\ & -1.90 \sum_{i \in I} s_i z_i - 0.10 \sum_{i \in I} z_i - 2.42 \sum_{i \in I} \sum_{j \in J_i} x_{ij} \\
\text{s.t.}\ & (x,y,z) \in \mathcal{X}_{\text{GPU}}.
\end{aligned}
$$

MCF Scenario 2. Minimize the number of used arcs. Here TACIT also recovers precisely an equivalent formulation (we note that $f \in \mathcal{X}_{\text{MCF}}$ guarantees that $f \geq 0$, and so the last inequality in $\mathcal{M}'$ is valid, albeit redundant):

$$\mathcal{M}^\star: \quad \min_{f,w} \ \sum_{i\in V}\sum_{j\in O_i} w_{ij} \quad \text{s.t.} \ f \in \mathcal{X}_{\text{MCF}}, \ f_{ij} \leq u_{ij} w_{ij}, \ \forall i \in V,\ j \in O_i, \ w_{ij} \in \{0,1\}, \ \forall i \in V,\ j \in O_i.$$

$$\mathcal{M}': \quad \min_{f,y} \ \sum_{i\in V}\sum_{j\in O_i} y_{ij} \quad \text{s.t.} \ f \in \mathcal{X}_{\text{MCF}}, \ f_{ij} \leq u_{ij} y_{ij}, \ \forall i \in V,\ j \in O_i, \ 0 \cdot y_{ij} \leq f_{ij}, \ \forall i \in V,\ j \in O_i, \ y_{ij} \in \{0,1\}, \ \forall i \in V,\ j \in O_i.$$

C Implementation Details

C.1 Routine and Subroutines

Algorithms C.1 to C.4 summarize the framework of Section 4. We write $\mathcal{M} \oplus \cdot$ for the formulation obtained by adding constraints, variables, or an instantiated template to $\mathcal{M}$, and $\mathcal{C}$ for the cuts accumulated across iterations. A single incumbent $\mathcal{M}'$ is maintained: every candidate formulation is scored by EVALUATE, which compares it to the incumbent before and after objective calibration and keeps the best. Cuts are searched on $\mathcal{M}' \oplus \mathcal{C}$ so that successive calls separate the solutions surviving the previous cuts, while templates are proposed from $\mathcal{M}_0$ to avoid compounding noise from previous modifications.

Algorithm C.1 TACIT Framework

Require: initial formulation $\mathcal{M}_0$, data $\mathcal{D}$, iteration budget T
$\mathcal{M}' \leftarrow \mathcal{M}_0$ with every constraint violated by some $x_i^\star$ removed *(preprocessing)*
$\mathcal{M}' \leftarrow$ EVALUATE$(\mathcal{M}', \mathcal{M}')$
$\mathcal{C} \leftarrow \emptyset$
for $t = 1, \ldots, T$ **while** $\mathcal{L}(\mathcal{M}'; \mathcal{D}) > 0$ **do**
  $\mathcal{M}', \mathcal{C} \leftarrow$ SEARCHCUTS$(\mathcal{M}', \mathcal{C})$ *(Sec. 4.1)*
  $\mathcal{M}' \leftarrow$ SEARCHTEMPLATES$(\mathcal{M}', \mathcal{M}_0)$ *(Sec. 4.2)*
end for
return $\mathcal{M}'$

Algorithm C.2 EVALUATE$(\mathcal{M}', \mathcal{M})$: score a candidate $\mathcal{M}$ and update the incumbent $\mathcal{M}'$

if some $x_i^\star$ has no lifting under $\mathcal{M}$ **then**
  return $\mathcal{M}'$
end if
if $\mathcal{L}(\mathcal{M}; \mathcal{D}) < \mathcal{L}(\mathcal{M}'; \mathcal{D})$ **then**
  $\mathcal{M}' \leftarrow \mathcal{M}$
end if
$\theta \leftarrow$ solution of $(\mathcal{P}_{\text{obj}})$ on $\mathcal{M}$ *(Sec. 4.3)*
$\mathcal{M}_\theta \leftarrow (c_\theta, \mathcal{X})$
if $\mathcal{L}(\mathcal{M}_\theta; \mathcal{D}) < \mathcal{L}(\mathcal{M}'; \mathcal{D})$ **then**
  $\mathcal{M}' \leftarrow \mathcal{M}_\theta$
end if
return $\mathcal{M}'$

C.2 Hyperparameters

Each run uses $N = 128$ training instances unless stated otherwise and is evaluated on 256 test instances. We use rejection sampling to ensure that at least half of target solutions differ from the initial ones. The search runs for at most 5 iterations and stops early once the training loss is optimal. Cuts are generated by the hyperplane-separation MIP with an ℓ_0 penalty starting at 128 and halved until non-empty cuts are found. Template prompts show up to 16 model/target solution pairs, the constraints removed during preprocessing, and up to 5 templates that were returned by the LLM on

Algorithm C.3 SEARCHCUTS($\mathcal{M}', \mathcal{C}$): cut search and LLM refinement

$\mathcal{M} \leftarrow \mathcal{M}' \oplus \mathcal{C}$
for ρ in $\{2^7, \dots, 2^{-7}\}$ **do**
  $\{(a^{(0)}, b^{(0)})\} \leftarrow$ solution of the cut MIP on $\mathcal{M}$ with penalty ρ *(App. E.1)*
  $\{(a^{(k)}, b^{(k)})\}_{k \geq 1} \leftarrow$ LLM-polished variants of $(a^{(0)}, b^{(0)})$ given $\mathcal{M}$
  for $k = 0, 1, \dots$ **do**
    $\mathcal{M}' \leftarrow \textsc{Evaluate}\big(\mathcal{M}', \mathcal{M} \oplus \{a^{(k)\top} x \leq b^{(k)}\}\big)$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{a^{(k)\top} x \leq b^{(k)}\}$
  end for
end for
$\mathcal{M}' \leftarrow \textsc{Evaluate}(\mathcal{M}', \mathcal{M}' \oplus \mathcal{C})$
for each constraint family $\mathcal{G}$ returned by the LLM as an extrapolation of $\mathcal{C}$ **do**
  $\mathcal{M}' \leftarrow \textsc{Evaluate}(\mathcal{M}', \mathcal{M}' \oplus \mathcal{G})$
end for
return $\mathcal{M}'$

Algorithm C.4 SEARCHTEMPLATES($\mathcal{M}', \mathcal{M}$): template generation and calibration

$\pi \leftarrow$ sample n_{pairs} mismatched pairs $(\hat{x}_i, x_i^\star)$ with $\hat{x}_i \in \Pi_0(\mathcal{S}(d_i; \mathcal{M}))$, $\hat{x}_i \neq x_i^\star$
$\{\tau_1, \dots, \tau_m\} \leftarrow$ templates returned by the LLM given $\mathcal{M}$ and π
for $j = 1, \dots, m$ **do**
  if $(\mathcal{P}_{\text{constr}})$ is infeasible **then**
    continue
  end if
  $w \leftarrow$ solution of $(\mathcal{P}_{\text{constr}})$ on $\mathcal{M} \oplus \tau_j$
  $\mathcal{M}' \leftarrow \textsc{Evaluate}(\mathcal{M}', \mathcal{M} \oplus \tau_j(w))$
end for
return $\mathcal{M}'$

previous iterations to avoid repetitions. All MIPs are solved with Gurobi on a single thread with a 30-second limit per instance. LLMs are queried with high reasoning effort and default temperature.

C.3 PROMPT TEMPLATES

We report the prompts verbatim. Fields between single angle brackets (e.g.<skeleton_str>) are filled at run time; tags between double angle brackets (e.g. «START_MODIFICATION») are the delimiters used to parse the response. The current formulation (<skeleton_str>) is always given as gurobipy-style pseudo-code.

Cut refinement. Each raw cut returned by the separation MIP has fitted numerical coefficients. The LLM rewrites it into simpler, interpretable variants (rounded coefficients, dropped terms, extension to a family), which are then evaluated like any other candidate cut.

Polish cut

```
Consider the following optimization model:
<skeleton_str>

where:
Existing variables: <variables>
Existing constants: <constants_and_parameters>

The above model is imperfect, and adding the following cut would improve it:
<raw_constraint_str>

Provide improved constraints that are easy to interpret.
Rely exclusively on the constants provided, do not introduce any new mathematical entities
↪  or sets.
Each constraint should be unambiguous.

If relevant, consider:
- scaling, equating, or rounding coefficients
- removing extraneous terms
```

```
- reordering terms
- adding contextually relevant terms
- replacing terms with equivalent expressions
- extending it to a family of constraints

Explain your reasoning, then return each improved version of the constraint in
↪  mathematical form between its own tags <<START_IMPROVED_CONSTRAINT>> and
↪  <<END_IMPROVED_CONSTRAINT>>.
Sort them from least to most parsimonious.
```

Cut generalization. Given up to 16 cuts found so far (sparse cuts are prioritized), the LLM abstracts their common pattern into constraint families over the existing sets, using only existing symbols.

Generalize cuts

```
Consider the following optimization model:
<skeleton_str>

where:
Existing variables: <variables>
Existing constants: <constants_and_parameters>

A cut generation algorithm found the following, valid cuts:
<cuts>

First, analyze the cuts. Work through this step-by-step and explain your reasoning for
↪  each step:

1. **Variable Abstraction:** Look past intermediate variables and specific variable
↪  weights. Use the baseline model constraints to derive the implications of cuts to core
↪  decision variables.

2. **Pattern Recognition:** Identify the common, underlying structural dependency across
↪  these cuts. Pay attention to specific indices acting as a fixed anchor across all the
↪  cuts, versus indices that represent a sequence.

3. **Generalization:** The cuts are incomplete samples of a broader family. Return one or
↪  more candidate generalized constraints (using $\forall$), from most to least
↪  interpretable.

Return each candidate generalized constraints in mathematical form (adding \forall
↪  quantifiers if appropriate) with a one sentence description, wrapping each candidate
↪  individually between the tags <<START_CONSTRAINT>> and <<END_CONSTRAINT>>.

Rules:
- Rely exclusively on the existing variables and constants. You may use standard
↪  mathematical operators (e.g., >, \neq) to define conditions or subsets based only on
↪  the provided sets.
- Do not introduce any new mathematical symbols, placeholder coefficients, generic
↪  multipliers, dummy variables, or abstract sets.
- Indices are plain letters bound by \forall or a sum (e.g. i, j, or k) or raw numbers for
↪  specific elements (r_{5}, v_{k, 3}): never k_0, k', k_prime.
- A candidate constraints must be missing from the current model. Do not propose
↪  constraints that are already implied by the existing model constraints.
```

Template proposal. Given the initial formulation and a set of instances where its solutions differ from the targets (only the differing variables are shown), the LLM proposes modifications that may introduce new symbolic constants, whose values are then set by calibration. The prompt also lists the constraints removed by preprocessing and the most recent modifications, to avoid repeating them.

Get templates

```
Consider the following optimization model:
<skeleton_str>

where:
Existing variables: <variables>
Existing constants: <constants_and_parameters>

The above model's solutions are infeasible or suboptimal under the ground truth model.
The ground truth model could incorporate additional unknown contant(s), variable(s), or
↪  constraint(s).
```

```
<removed_constraints>

<recent_modifications>

<cuts_str>

Below are the solutions on <n_pairs> instances where the model's solution differ from the
↪  ground truth solutions. Only variables that differ are reported under the format
↪  {var_where_mismatch: (model_value, target_value)}.
<delta_pairs>

First, analyze the difference between the solutions. Explain your analysis with specific
↪  examples.
Then, propose a model modification that explain the observed differences of the model's
↪  solutions.

Return one or more candidate modifications.

Before proposing modifications, determine whether the observed discrepancies are best
↪  explained by:
1. a missing linear constraint on existing variables;
2. an existing linear constraint with incorrect existing or missing constants;
3. a missing decision variable or auxiliary variable representing an unmodeled quantity,
↪  and related constraints;
4. an incorrect decision variables domain (e.g. continuous, integer, binary).
Analyze all four possibilities separately and include candidate modifications from
↪  multiple categories whenever plausible.

Each candidate modification must be enclosed in its own separate block:
<<START_MODIFICATION>>
...
<<END_MODIFICATION>>

Each block represents one independent candidate modification that explains the observed
↪  solution differences.

For each candidate modification block, include:
- The mathematical form of every proposed constraint, with all terms on the left-hand side
↪  of the equality or inequality;
- A one sentence description of each constraint, constant, and decision variable involved.

Rules for each candidate modification:
- A modification may contain one constraint or multiple constraints that must be
↪  considered jointly.
- Constraints involved in a candidate modification must be missing from the current model.
↪  Do not propose constraints that are already implied by the existing model constraints.
- A proposed constraints must keep all target solutions feasible.
- A modification may introduce new decision variables / auxiliary variables when the
↪  observed discrepancies suggest its existence
- A modification may introduce new constants, but may *not* introduce new sets.
- Do not use numerical constants unless the value is uniquely determined. If the data only
↪  suggests the existence of one or more quantities but not their exact values, introduce
↪  new symbolic constants for them, not already present in the model; a new constant may
↪  multiply an existing parameter when the quantity plausibly scales with it.
- Do not use backets for variable or constant names. Backets must exclusively be used for
↪  mathematical indices.
```

Parsing and translation. Three additional prompts, omitted here for brevity, convert the free-text modifications and constraints returned above into executable code: one splits a modification into its new constants, variables, type changes, and constraints, the two other translate each constraint and variable into our structured representation using few-shot examples. They perform a syntactic translation only and will be available once the code is released.

LLM-only baseline. The LLM receives the initial formulation as compilable gurobipy code, the solution differences on all instance pairs, and the data format, and returns a complete corrected gurobipy model in a single query.

Get formulation (LLM-only baseline)

```
Consider the following optimization model, written as the body of a gurobipy function
↪  `get_formulation(data)` (with `import gurobipy as gp` and `from gurobipy import GRB`).
```python
<skeleton_str>
```
```

```
```

where:
Existing variables: <variables>
Existing constants and parameters: <constants_and_parameters>

The above model's solutions are infeasible or suboptimal under the ground truth model.
The ground truth model could incorporate additional unknown constant(s), variable(s), or
↪ constraint(s), or a different objective.

Below are the solutions on <n_pairs> instances where the model's solution differ from the
↪ ground truth solutions. Only variables that differ are reported under the format
↪ {var_where_mismatch: (model_value, target_value)}.
<delta_pairs>

First, analyze the difference between the solutions. Explain your analysis with specific
↪ examples.
Before writing the model, determine whether the observed discrepancies are best explained
↪ by:
1. a missing linear constraint on existing variables;
2. an existing linear constraint with incorrect existing or missing constants;
3. a missing decision variable or auxiliary variable representing an unmodeled quantity,
↪ and related constraints;
4. an incorrect decision variables domain (e.g. continuous, integer, binary).
Analyze all four possibilities separately, then write the single complete model that best
↪ explains the observed solutions.

Then, write the complete corrected model as executable Python. Return exactly one block:
<<START_MODEL>>
import gurobipy as gp
from gurobipy import GRB

def get_formulation(data):
    ...
    return model
<<END_MODEL>>

Data format. `data` is a flat dictionary with string keys holding one instance. An indexed
↪ entry is one key per index, written with the exact spelling of the code above and
↪ built with f-strings, e.g. data[f"c[{i}, {j}]"]. Indices may be integers or strings,
↪ as in the example. These are the only keys:
<data_keys>
Example instance:
<example_data>

Rules:
- Define exactly `def get_formulation(data):` building and returning a gurobipy Model
↪ created with gp.Model(). Do not solve it, do not print, do not set parameters, do not
↪ create a gp.Env, do not read files, do not use randomness.
- The model must be a mixed-integer linear program: linear constraints and a linear
↪ objective set with model.setObjective(..., GRB.MINIMIZE or GRB.MAXIMIZE) following the
↪ sense of the code above.
- Name every variable with `name=` using the exact symbol, index order and spelling of the
↪ code above: model.addVar(vtype=GRB.BINARY, name=f"y[{i}]"),
↪ model.addVar(vtype=GRB.CONTINUOUS, name=f"x[{i}, {j}]"). Every variable of the code
↪ above must exist in the model with the same name over the same index set. Names must
↪ be unique.
- gurobipy's addVar defaults to lb=0: pass lb=-GRB.INFINITY for any variable that may be
↪ negative, and state every bound explicitly.
- You may add variables, constraints and constants that are not in the code above, change
↪ variable types, and change constraint or objective coefficients. New variables must
↪ have their own names, distinct from those of the code above.
- You may hardcode numerical constants whose values you infer from the solutions above.
- The ground truth solutions above (target values) must be feasible in your model, and
↪ should be its optimal solutions.
- Do not merely return the code above: the model must explain the observed differences.
```

### C.4 Compute and runtime

Runs are parallelized on an AMD EPYC 9734 processor, one run per core with Gurobi restricted to a single thread. LLMs are queried through hosted APIs with high reasoning effort. Each method is run 5 times per scenario, each run on a dataset generated with a different random seed, for at most 5 iterations and under a time budget of 6 hours per run. For datasets with $N{=}128$ instances, the median runtime per run ranges from 2 min for calibration alone to 0.8 hours for template, 1.3 hour for cuts and refine, and 2.1 hours for the full framework. The runtime of the LLM-only baseline is 0.2 hour for the LLM-only baseline.
```

We use two models in our framework: the large model, reported in the main text (DeepSeek-V4-Flash, Kimi-K2.6 or gpt-5.6-sol), generates templates and refine cuts, while the small model (always DeepSeek-V4-Flash) handles simpler tasks: split templates into variables, constants, and constraints, and translating natural-language modifications into optimization code. Averaged across the three large models, a run of our framework consumes 154k tokens from the large model and 346k tokens from the small model. The LLM-only baseline uses on average 210k tokens per run, all from the large model.

D Benchmark Applications and Scenarios

The benchmark consists of 38 scenarios built on nine base problems. Each scenario is a pair of formulations: the initial model used as the starting point, and the target model that generated the observed solutions. Table D.1 classifies what separates the two. Below, every problem is introduced with its notation, its base formulation and how instances are drawn, and its scenarios are then listed. Unless we say otherwise, the initial model is the base formulation and the target model is the base formulation with the stated change. Every model is written as a minimization, so maximization problems appear with a negated objective. Scenarios that single out specific indices (e.g. facility 4, or employees 3 and 5) are run on instances of fixed size so that these indices always exist. Numerical constants introduced by a target model (R, L, K, ...) are hidden from the repair method, which only sees the initial model and the solutions.

D.1 Capacitated facility location (CFL)

Facilities are opened at a fixed cost and customers are assigned to open facilities, possibly splitting their demand. Let I be the facilities and J the customers. Customer j has demand d_j, facility i has capacity C_i and opening cost f_i, and serving the whole demand of j from i costs c_{ij}. Variable $y_i \in \{0,1\}$ opens facility i and $x_{ij} \in [0,1]$ is the fraction of j's demand served by i.

$$\min \quad \sum_{i\in I} f_i y_i + \sum_{i\in I}\sum_{j\in J} c_{ij} x_{ij} \tag{4}$$

$$\text{s.t.} \quad \sum_{i\in I} x_{ij} \geq 1, \qquad \forall j \in J, \tag{5}$$

$$\sum_{j\in J} d_j x_{ij} \leq C_i y_i, \qquad \forall i \in I, \tag{6}$$

$$0 \leq x_{ij} \leq 1,\ y_i \in \{0,1\}, \qquad \forall i \in I,\ j \in J. \tag{7}$$

Instances have 6 to 8 facilities and 8 to 14 customers, d_j uniform in $[1,20]$, f_i in $[50,250]$ and c_{ij} in $[1,100]$. Capacities are $C_i = \lceil \alpha_i \bar{d} \rceil$ with $\bar{d}$ the total demand per facility and α_i uniform in $[1.5, 2.5]$, so that roughly half of the facilities need to open.

- **S1: service radius.** A facility can only serve customers within distance $R = 50$. The target adds $z_{ij} \in \{0,1\}$ with $x_{ij} \leq z_{ij}$ and $c_{ij} z_{ij} \leq R$ for all i, j. Instances are patched so that every customer has at least one facility within the radius.
- **S2: overestimated capacity.** The target replaces (6) by the tighter $\sum_j d_j x_{ij} \leq 0.8\, C_i y_i$. The initial model uses the full C_i.
- **S3: underestimated capacity.** The target allows $\sum_j d_j x_{ij} \leq 1.2\, C_i y_i$, so the capacity constraint of the initial model cuts off target solutions.
- **S4: overweighted connection costs.** The target objective is $\sum_i f_i y_i + 0.5 \sum_{i,j} c_{ij} x_{ij}$.
- **S5: connection costs paid per pair.** The target charges c_{ij} once whenever facility i serves any part of customer j. It adds $z_{ij} \in \{0,1\}$ with $x_{ij} \leq z_{ij}$ and replaces the second sum in (4) by $\sum_{i,j} c_{ij} z_{ij}$.
- **S6: unsplittable demand.** The target makes x_{ij} binary.
- **S7: capped facility.** Facility 4 serves at most half of any customer: $x_{4j} \leq 0.5$ for all $j \in J$.

D.2 GPU ALLOCATION (GPU)

GPU demands are matched to compatible deployments. A deployment has some spare capacity and can be scaled up by adding instances, within the GPU budget of its location. Let I be the demands, J the deployments, $J_i \subseteq J$ the deployments compatible with demand i and I_j the demands compatible with j. Locations $L = \{\text{us-east}, \text{us-west}\}$ host GPU types $G = \{\text{H100}, \text{A100}\}$, and $G_\ell \subseteq J$ are the deployments hosted at location ℓ. Demand i needs s_i GPUs. Deployment j has A_j spare GPUs and instances of size S_j, and location ℓ has $C_{\ell g}$ GPUs of type g left. Variables: $x_{ij} \in \mathbb{Z}_+$ GPUs of j given to i, $y_j \in \mathbb{Z}_+$ new instances of j, $z_i \in \{0,1\}$ whether i is served.

$$\min \quad -100 \sum_{i \in I} z_i \tag{8}$$

$$\text{s.t.} \quad \sum_{j \in J_i} x_{ij} = s_i z_i, \qquad \forall i \in I, \tag{9}$$

$$\sum_{i \in I_j} x_{ij} \le A_j + S_j y_j, \qquad \forall j \in J, \tag{10}$$

$$\sum_{j \in G_\ell} S_j y_j \le C_{\ell g}, \qquad \forall \ell \in L,\ g \in G, \tag{11}$$

$$x_{ij}, y_j \in \mathbb{Z}_+,\ z_i \in \{0,1\}. \tag{12}$$

Instances have 3 to 6 demands and 4 to 9 deployments, three compatible deployments per demand, s_i in $[10, 200]$, $S_j \in \{1, 2, 4\}$, A_j in $[0, 15]$ and $C_{\ell g}$ in $[100, 300]$.

- **S1: cost of scaling a deployment.** The target pays one unit per deployment that gets new instances: it adds $u_j \in \{0,1\}$ with $y_j \le 500\, u_j$, and $+\sum_{j \in J} u_j$ to the objective.
- **S2: missing assignment constraints.** The initial model lacks (9). The target is the base formulation.
- **S3: precedence.** Demand 0 can only be served once every other demand is: $z_0 \le z_i$ for all $i \in I \setminus \{0\}$.
- **S4: size-weighted objective.** The target maximizes the number of GPUs served, $\min -\sum_i s_i z_i$, while the initial model counts demands, $\min -\sum_i z_i$.
- **S5: at most two demands.** The target adds $\sum_{i \in I} z_i \le 2$.

D.3 MAXIMUM INDEPENDENT SET (MIS)

Let $G = (V, E)$ be an undirected graph and N_u the neighbours of u. Variable $x_u \in \{0,1\}$ selects vertex u, and each edge is written once, with $u < v$.

$$\min \quad -\sum_{u \in V} x_u \tag{13}$$

$$\text{s.t.} \quad x_u + x_v \le 1, \qquad \forall u \in V,\ v \in N_u,\ u < v, \tag{14}$$

$$x_u \in \{0,1\}, \qquad \forall u \in V. \tag{15}$$

Graphs have 15 to 25 vertices (20 when the size is fixed) and each edge is present with probability 0.2.

- **S1: spurious path.** The initial model additionally forbids the pairs of the path $0{-}1{-}\cdots{-}6$, i.e. $x_u + x_{u+1} \le 1$ for $u = 0, \ldots, 5$, whether or not these edges are in E. The target is the base formulation.
- **S2: missing path.** The initial model drops (14) for every edge $\{u, u+1\} \in E$ with $u \le 5$. The target is the base formulation.
- **S3: conflict budget.** Up to $k = 3$ edges may have both endpoints selected. The target adds $z_{uv} \in \{0,1\}$ per edge, replaces (14) by $x_u + x_v \le 1 + z_{uv}$, and adds $\sum_{\{u,v\} \in E} z_{uv} \le k$.

D.4 MINIMUM-COST FLOW (MCF)

On a directed graph with node set V, let O_i be the heads of the arcs leaving i and P_i the tails of the arcs entering i. Arc (i, j) has capacity u_{ij} and unit cost c_{ij}, and node i has net supply b_i (negative for a demand). Variable $f_{ij} \geq 0$ is the flow on (i, j).

$$\min \quad \sum_{i \in V} \sum_{j \in O_i} c_{ij} f_{ij} \tag{16}$$

$$\text{s.t.} \quad \sum_{j \in O_i} f_{ij} - \sum_{j \in P_i} f_{ji} = b_i, \qquad \forall i \in V, \tag{17}$$

$$0 \leq f_{ij} \leq u_{ij}, \qquad \forall i \in V,\ j \in O_i. \tag{18}$$

Instances have 8 to 12 nodes. Each unordered pair is linked with probability 0.25 and a link gives an arc in both directions, each with its own u_{ij} in $[1, 20]$ and c_{ij} in $[1, 10]$. Balances are induced by a random flow that fits both directions of every link, so the reversed instance of S1 stays feasible.

- **S1: reversed balances.** The initial model has $-b_i$ on the right-hand side of (17). The target is the base formulation.
- **S2: fewest arcs.** The target adds $w_{ij} \in \{0, 1\}$ with $f_{ij} \leq u_{ij} w_{ij}$ and minimizes $\sum_i \sum_{j \in O_i} w_{ij}$ instead of the flow cost.

D.5 MULTI-PERIOD PRODUCTION PLANNING (MPP)

Three products P are produced on two machines M over three periods T. Demand is d_{pt}, product p takes u_{mp} time units on machine m, and machine m has C_{mt} time units in period t. Integer variables x_{pt}, s_{pt} and b_{pt} are the quantity produced, the end-of-period inventory and the quantity bought outside.

$$\min \quad \sum_{p \in P} \sum_{t \in T} (10 x_{pt} + 3 s_{pt} + 25 b_{pt}) \tag{19}$$

$$\text{s.t.} \quad x_{p0} + b_{p0} - s_{p0} = d_{p0}, \qquad \forall p \in P, \tag{20}$$

$$x_{pt} + b_{pt} + s_{p,t-1} - s_{pt} = d_{pt}, \qquad \forall p \in P,\ t \in T,\ t > 0, \tag{21}$$

$$\sum_{p \in P} u_{mp} x_{pt} \leq C_{mt}, \qquad \forall m \in M,\ t \in T, \tag{22}$$

$$x_{pt}, s_{pt}, b_{pt} \in \mathbb{Z}_+. \tag{23}$$

d_{pt} is uniform in $[10, 100]$, u_{mp} in $[1, 4]$ and C_{mt} in $[150, 300]$.

- **S1: free inventory.** The target objective drops the holding cost: $\sum_{p,t} (10 x_{pt} + 25 b_{pt})$.
- **S2: production cap.** The target limits total production per period: $\sum_{p \in P} x_{pt} \leq L$ for all t, with $L = 100$.

D.6 SET COVER (SC)

There are $|I| = 20$ elements and $|J| = 40$ subsets. $a_{ij} \in \{0, 1\}$ says whether subset j contains element i, $S_i = \{j : a_{ij} = 1\}$, and c_j is the cost of subset j. Variable $x_j \in \{0, 1\}$ selects subset j.

$$\min \quad \sum_{j \in J} c_j x_j \tag{24}$$

$$\text{s.t.} \quad \sum_{j \in J} a_{ij} x_j \geq 1, \qquad \forall i \in I, \tag{25}$$

$$x_j \in \{0, 1\}, \qquad \forall j \in J. \tag{26}$$

Each a_{ij} is 1 with probability 0.15, every element is covered by at least one subset, and c_j is uniform in $[1, 100]$. Elements and subsets are indexed from 0.

- **S1: double coverage.** Elements 0 and 1 must be covered twice: the target adds $\sum_j a_{ij} x_j \geq 2$ for $i \in \{0, 1\}$.
- **S2: optional elements.** The target drops (25) for $i \in \{0, 1\}$.
- **S3: subsets that do not cover.** In the target, subsets 0, 1 and 2 contribute nothing to coverage: (25) becomes $\sum_{j \geq 3} a_{ij} x_j \geq 1$.
- **S4: unweighted initial model with a premium.** The initial model minimizes the number of selected subsets, $\sum_j x_j$. The target minimizes the weighted cost (24) with c_j doubled for $j \in \{0, 1, 2\}$.
- **S5: weighted coverage.** Each element i has a hidden requirement r_i, uniform in $[1, 2]$, and each subset a hidden weight w_j, uniform in $[0, 2]$, both drawn once and shared by all instances. The target replaces (25) by $\sum_{j \in S_i} w_j x_j \geq r_i$. Instances are patched by adding subsets to elements whose covering subsets do not carry enough weight.

D.7 Transportation problem (TP)

Origins I with supplies S_i ship to destinations J with demands D_j at unit cost c_{ij}. Variable $x_{ij} \geq 0$ is the quantity shipped.

$$\min \quad \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{27}$$

$$\text{s.t.} \quad \sum_{j \in J} x_{ij} \leq S_i, \qquad \forall i \in I, \tag{28}$$

$$\sum_{i \in I} x_{ij} \geq D_j, \qquad \forall j \in J, \tag{29}$$

$$x_{ij} \geq 0. \tag{30}$$

Instances have 3 to 9 origins and destinations, D_j uniform in $[5, 25]$ and c_{ij} in $[1, 10]$. Supplies are drawn between 1.15 and 1.45 times the average demand per origin, so that origins must share destinations while the targets below remain feasible.

- **S1: minimum utilization.** Every origin ships at least half of its supply: $\sum_j x_{ij} \geq 0.5\, S_i$.
- **S2: limited suppliers.** At most $K = 2$ origins serve a destination. The target adds $z_{ij} \in \{0, 1\}$ with $x_{ij} \leq M z_{ij}$, $M = 1000$, and $\sum_i z_{ij} \leq K$ for all j.
- **S3: batches.** Shipments come in batches of $B = 2$: the target adds $y_{ij} \in \mathbb{Z}_+$ and $x_{ij} = B y_{ij}$.
- **S4: fixed route cost.** Using a route costs 10. The target adds $z_{ij} \in \{0, 1\}$ with $x_{ij} \leq M z_{ij}$, and $10 \sum_{i,j} z_{ij}$ to the objective.
- **S5: integral shipments.** The target makes x_{ij} integer. Since the constraint matrix is totally unimodular, integral data would make both models agree, so a fractional part uniform in $[0.1, 0.9]$ is subtracted from every D_j.
- **S6: tied origins.** Origins 0, 1 and 2 ship the same quantity to every destination: $x_{0j} = x_{1j}$ and $x_{1j} = x_{2j}$ for all j.

D.8 Capacitated vehicle routing (CVRP)

Nodes $N = \{0, \ldots, 6\}$ with depot 0 and customers $C = N \setminus \{0\}$ are served by vehicles $K = \{0, 1, 2\}$. Customer i has demand d_i, vehicle k capacity Q_k, and c_{ij} is the travel cost. Variable $x_{ijk} \in \{0, 1\}$ says whether k drives from i to j, and $u_{ik} \geq 0$ is the continuous Miller–Tucker–

Zemlin position of i on the route of k, with $n = |N| = 7$.

$$\min \quad \sum_{k\in K}\sum_{i\in N}\sum_{j\neq i} c_{ij}x_{ijk} \tag{31}$$

$$\text{s.t.} \quad \sum_{k\in K}\sum_{i\neq j} x_{ijk} = 1, \qquad \forall j \in C, \tag{32}$$

$$\sum_{i\in C}\sum_{j\neq i} d_i x_{ijk} \leq Q_k, \qquad \forall k \in K, \tag{33}$$

$$\sum_{i\neq h} x_{ihk} - \sum_{j\neq h} x_{hjk} = 0, \qquad \forall h \in N,\ k \in K, \tag{34}$$

$$\sum_{j\in C} x_{0jk} \leq 1, \qquad \forall k \in K, \tag{35}$$

$$u_{ik} - u_{jk} + n x_{ijk} \leq n-1, \qquad \forall k \in K,\ i,j \in C,\ i \neq j, \tag{36}$$

$$1 \leq u_{ik} \leq n, \qquad \forall k \in K,\ i \in C, \tag{37}$$

$$x_{ijk} \in \{0,1\}. \tag{38}$$

Customers are placed uniformly in a box of half-width 5 around the depot, c_{ij} is the Euclidean distance except that returning to the depot is free ($c_{i0} = 0$), d_i is uniform in $[2, 4]$ and Q_k between 0.35 and 1.1 times the total demand.

- **S1: route length.** A vehicle traverses at most $R = 4$ arcs: $\sum_{i\in N}\sum_{j\neq i} x_{ijk} \leq R$ for all k.
- **S2: min–max.** The target minimizes the longest route: it adds a continuous r with $\sum_i \sum_{j\neq i} c_{ij}x_{ijk} \leq r$ for all k and objective $\min r$.
- **S3: forbidden arc.** No vehicle drives from customer 2 to customer 5: $x_{2,5,k} = 0$ for all k.
- **S4: incompatible vehicle.** Vehicle 2 never visits customer 3: $x_{i,3,2} = 0$ for all $i \neq 3$.

D.9 Workforce assignment (WA)

Employees E are assigned to consecutive days $D = \{0, \ldots, T\}$. Day d needs r_d employees, employee e works at most m_e days and costs c_{ed} on day d. Variable $x_{ed} \in \{0, 1\}$ assigns e to d.

$$\min \quad \sum_{e\in E}\sum_{d\in D} c_{ed}x_{ed} \tag{39}$$

$$\text{s.t.} \quad \sum_{e\in E} x_{ed} \geq r_d, \qquad \forall d \in D, \tag{40}$$

$$\sum_{d\in D} x_{ed} \leq m_e, \qquad \forall e \in E, \tag{41}$$

$$x_{ed} \in \{0,1\}. \tag{42}$$

Instances have 6 or 7 employees and 10 to 14 days, m_e uniform in $[1, |D|]$, r_d in $[1, \min(|E|, \sum_e m_e)]$ and c_{ed} in $[1, 100]$.

- **S1: rest after five days.** Five consecutive workdays must be followed by two days off. The target adds, for every $e \in E$,

$$\sum_{t=d}^{d+5} x_{et} \leq 5, \qquad d = 0, \ldots, T-5, \tag{43}$$

$$\sum_{t=d}^{d+4} x_{et} + x_{e,d+6} \leq 5, \qquad d = 0, \ldots, T-6. \tag{44}$$

- **S2: workload balance.** The target adds continuous $W, w \geq 0$ with $\sum_d x_{ed} \leq W$ and $\sum_d x_{ed} \geq w$ for all e, and $100\,(W - w)$ to the objective.

- **S3: incompatible employees.** Employees 3 and 5 never work the same day: $x_{3d} + x_{5d} \leq 1$ for all d.
- **S4: no back-to-back days.** Employee 3 never works two days in a row: $x_{3d} + x_{3,d+1} \leq 1$ for $d = 0, \ldots, T-1$.

Table D.1: Taxonomy of scenario misspecifications over the 38 benchmark scenarios spanning 9 problems: CFL (capacitated facility location), GPU (GPU allocation), MIS (maximum independent set), MCF (minimum-cost flow), MPP (multi-period production planning), SC (set cover), TP (transportation), CVRP (capacitated vehicle routing), WA (workforce assignment). A constraint is local when the restriction it adds involves fixed indices (e.g. $\sum_j (x_{3,j} + x_{5,j}) \leq 1$) and global when it is stated over generic indices (e.g. $\sum_{i,j} x_{i,j} \leq 1$). Extra marks a constraint of the initial model that cuts off target-feasible or target-optimal solutions. Constraints that only define an auxiliary variable are counted under missing variable rather than as a missing constraint, and big-M bounds are not counted as missing constants.

		Objective		Constraints			Variables		Constants
	Scenario	Miscalibrated	Missing	Local	Global	Extra	Missing	Domain	Missing
CFL	S1	□	□	□	■	□	■	□	■
	S2	□	□	□	■	□	□	□	■
	S3	□	□	□	■	■	□	□	■
	S4	■	□	□	□	□	□	□	□
	S5	■	■	□	□	□	■	□	□
	S6	□	□	□	□	□	□	■	□
	S7	□	□	■	□	□	□	□	■
GPU	S1	□	■	□	□	□	■	□	□
	S2	□	□	□	■	□	□	□	□
	S3	□	□	■	□	□	□	□	□
	S4	■	□	□	□	□	□	□	□
	S5	□	□	□	■	□	□	□	■
MIS	S1	□	□	□	□	■	□	□	□
	S2	□	□	■	□	□	□	□	□
	S3	□	□	□	■	□	■	□	■
MCF	S1	□	□	□	■	■	□	□	□
	S2	■	■	□	□	□	■	□	□
MPP	S1	■	□	□	□	□	□	□	□
	S2	□	□	□	■	□	□	□	■
SC	S1	□	□	■	□	□	□	□	■
	S2	□	□	□	□	■	□	□	□
	S3	□	□	■	□	■	□	□	□
	S4	■	□	□	□	□	□	□	□
	S5	□	□	□	■	□	□	□	■
TP	S1	□	□	□	■	□	□	□	■
	S2	□	□	□	■	□	■	□	■
	S3	□	□	□	■	□	■	□	■
	S4	□	■	□	□	□	■	□	■
	S5	□	□	□	□	□	□	■	□
	S6	□	□	■	□	□	□	□	□
CVRP	S1	□	□	□	■	□	□	□	■
	S2	■	■	□	□	□	■	□	□
	S3	□	□	■	□	□	□	□	□
	S4	□	□	■	□	□	□	□	□
WA	S1	□	□	□	■	□	□	□	■
	S2	□	■	□	■	□	■	□	■
	S3	□	□	■	□	□	□	□	□
	S4	□	□	■	□	□	□	□	□

E Additional Details

E.1 Cut Search

Definition 1 (Variable family)**.** *Let $\mathcal{K}_i(\mathcal{M})$ be the set of indices of the decision variables of $\mathcal{M}(d_i)$, whose size may vary across instances. Each variable is identified by a symbol and a tuple of indices ranging over sets that are either shared by all instances or instance-dependent. The* family $t(k)$ *of variable k is obtained by dropping its indices with instance-dependent sets. We denote by $\mathcal{T}(\mathcal{M})$ the set of families. In particular, when every instance shares the same index sets, each variable forms its own family.*

The intuition behind families is that weights learned from data must be defined at the family level: a weight attached to a family applies to every instance, whereas a weight attached to an individual variable is meaningless on instances where that variable does not exist.

For instance, consider the VRP example (Example 1) where there is a fixed fleet of 5 vehicles $\mathcal{L}$ for all instances, but the customer set C varies across instances. The routing variable $x_{ij\ell}$ indicates vehicle ℓ travels from customer i to j. A variable index is a tuple $k = (i, j, l) \in N \times N \times \{1, \dots, 5\}$, whose customer components i, j are instance-dependent and whose vehicle component ℓ is shared. In this case, there is one family per vehicle, $\{x_{ijl} : (i, j) \in N \times N\}$ for $l \in \{1, \dots, 5\}$. In particular, it is not possible to learn customer-dependent weights that apply to all instances.

Let $\varphi_k(d)$ be the elements of d whose instance-dependent indices all appear among the indices of variable k. Elements indexed only by index sets identical across instances belong to every $\varphi_k(d)$, and we denote this common part by $\varphi_0(d)$. Define $\phi_k(d) = \big(1, \varphi_k(d)\big)$. A cut then reads $a(d)^\top x \le b(d)$ with $a(d)_k = \alpha_{t(k)}^\top \phi_k(d)$ and $b(d) = \beta^\top \phi_0(d)$, where $\alpha_{t(k)}$ and β are weights to be inferred. We solve the following MILP to perform this inference and find cuts with sparsity controlled by a parameter ρ:

$$
\begin{aligned}
\max_{\alpha,\beta,z,\eta} \quad & \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} z_i - \rho \left(\|\alpha\|_0 + \|\beta\|_0 \right) + \epsilon \eta && && (\mathcal{P}_{\text{cut}}) \\
\text{s.t.} \quad & a_i^\top x_i^\star - b_i \le 0, && \forall i \in \mathcal{I}, \\
& a_i^\top \hat{x}_i - b_i \ge \eta - M(1 - z_i), && \forall i \in \mathcal{I}, \\
& a_{i,k} = \alpha_{t(k)}^\top \phi_k(d_i), && \forall i \in \mathcal{I},\ \forall k \in \mathcal{K}(d_i), \\
& b_i = \beta^\top \phi_0(d_i), && \forall i \in \mathcal{I}, \\
& z \in \{0,1\}^{|\mathcal{I}|}, \quad \eta \ge \underline{\eta}, \quad \|\alpha\|_\infty \le 1, \quad \|\beta\|_\infty \le 1.
\end{aligned}
$$

The first constraint makes the cut valid for every target, and the binary z_i certifies that $\hat{x}_i$ violates it by at least the margin $\eta \ge \underline{\eta} > 0$. The objective maximizes the share of solutions of $\mathcal{M}$ that are cut off, penalizes the number of nonzero weights $\|\alpha\|_0 + \|\beta\|_0$ (linearized with binaries $u_{i,k} \ge |\alpha_{i,k}|$ and $v_i \ge |\beta_i|$, which also yield the bound $\|\alpha\|_\infty \le 1$ and $\|\beta\|_\infty \le 1$) and breaks ties with a small reward on the margin with $0 < \epsilon \ll 1$. The penalty ρ trades separation for sparsity: a ρ close to 0 yields the densest and most separating cut, whereas a large ρ yields cuts with few terms, which are more interpretable and less prone to overfitting. The cut search sweeps ρ downward to collect cuts of increasing density, from the sparsest nonempty one to the unregularized one, and evaluates each by the separation loss.

E.2 Template completion

We calibrate the template weights w to preserve all observed targets while maximally separating the optimal solutions of the current formulation $\mathcal{M}$. Preservation requires each $x_i^\star$ to admit a lifting $y_i^\star$ into the non-primary variables of $\mathcal{M}$ and an assignment of the new variables $z_i^\star$ such that $(y_i^\star, z_i^\star)$ belongs to the new feasible set $\mathcal{X}_\tau(d_i; w)$. For the second goal, define

$$
V_\tau(y, z, w; d) \triangleq \sum_{j \in J_\tau(d)} [g_j(y, z, w; d)]^+,
$$

which measures the total violation of the template constraints by a solution (y, z). Since an optimal solution $\hat{y}_i$ of $\mathcal{M}$ does not specify the new variables z, it is excluded by the template only if *every* completion $\hat{z}_i$ violates its constraints. Thus, for a given w, its violation is measured by $\min_{\hat{z}_i \in Z_\tau(d_i)} V_\tau(\hat{y}_i, \hat{z}_i, w; d_i)$, i.e., the violation under the completion of the new variables that best satisfies the template.

Combining these two requirements, we determine the template weights w by

$$\max_{w \in \mathcal{W},\, (y_i^\star, z_i^\star)_{i \in \mathcal{I}}} \quad \sum_{i \in \mathcal{I}} \min_{\hat{z}_i \in Z_\tau(d_i)} V_\tau(\hat{y}_i, \hat{z}_i, w; d_i) \tag{$\mathcal{P}_{\text{constr}}$}$$

$$\text{s.t.} \quad (y_i^\star, z_i^\star) \in \mathcal{X}_\tau(d_i; w), \qquad \forall i \in \mathcal{I}, \tag{45}$$

$$\Pi_0(y_i^\star) = x_i^\star, \qquad \forall i \in \mathcal{I}. \tag{46}$$

For scale-invariant templates, we normalize the corresponding components of w. Problem ($\mathcal{P}_{\text{constr}}$) has a max-min structure, jointly calibrating the template parameters w against the best completions $\hat{z}_i$ of the new variables. We solve this problem via constraint generation and discard any template where no weights allow every target to be feasibly lifted.

E.3 Objective Calibration

As with cuts, we parameterize objective coefficients as affine functions of instance data, with one weight vector per variable family. The objective coefficient of variable k is $c_\theta(d_i)_k = \theta_{t(k)}^\top \phi_k(d_i)$, where $\phi_k(d_i)$ denotes data associated with variable k and $\theta_{t(k)}$ is the vector to be learned for family $t(k)$.

We seek weights θ that balance two goals. First, each target solution $x_i^\star$ should admit a feasible lifting $y_i^\star$ that is nearly optimal under the calibrated objective. We allow some suboptimality to accommodate potentially suboptimal target solutions in practice. Second, the lifted target $y_i^\star$ should outperform the current model solution $\hat{y}_i$ by as large a margin as possible; equivalently, we seek an objective vector $c_\theta(d_i)$ for which $c_\theta(d_i)^\top(\hat{y}_i - y_i^\star)$ is large. We therefore calibrate θ by solving

$$\min_{\theta \in \Theta,\, (y_i^\star, \varepsilon_i)_{i \in \mathcal{I}}} \quad \sum_{i \in \mathcal{I}} \varepsilon_i - \lambda \sum_{i \in \mathcal{I}} c_\theta(d_i)^\top(\hat{y}_i - y_i^\star) \tag{$\mathcal{P}_{\text{obj}}$}$$

$$\text{s.t.} \quad c_\theta(d_i)^\top(y_i^\star - y) \le \varepsilon_i \qquad \forall i \in \mathcal{I},\ \forall y \in \mathcal{X}(d_i), \tag{47}$$

$$y_i^\star \in \mathcal{X}(d_i), \qquad \Pi_0(y_i^\star) = x_i^\star \quad \forall i \in \mathcal{I}, \tag{48}$$

$$\varepsilon_i \ge 0 \qquad \forall i \in \mathcal{I}. \tag{49}$$

At an optimum, $\varepsilon_i = c_\theta(d_i)^\top y_i^\star - \min_{y \in \mathcal{X}(d_i)} c_\theta(d_i)^\top y \ge 0$, so ϵ_i precisely captures the suboptimality of the lifted target solution under the calibrated objective. The parameter $\lambda > 0$ controls the tradeoff between target suboptimality and separation from the current model solutions. As with template calibration, we solve Problem ($\mathcal{P}_{\text{obj}}$) via constraint generation.