TRACEABILITY MATTERS: TOWARDS FAITHFUL OPTIMIZATION MODELING VIA EVIDENCE-CHAIN SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

Operations Research (OR) modeling, which translates natural-language problem descriptions into solver-executable mathematical formulations, has traditionally relied on domain experts, making it labor-intensive and difficult to scale. Recent work uses large language models (LLMs) to automate this process, but primarily optimizes for code executability and final-answer accuracy, overlooking whether each modeling decision is faithful to the problem statement. Through a failure analysis of state-of-the-art frameworks, we find that nearly 90% of failures arise from such unfaithfulness: modeling elements bound to the wrong entities or conditions (*mapping misalignment*) and stated conditions missing from the formulation (*information omission*). Human experts guard against both errors by repeatedly checking every modeling decision against the source problem text and confirming that no stated condition is left out, an ability we term *traceability awareness*. Inspired by this observation, we propose **TAGO** (**T**raceability-**A**ware **G**uided **O**ptimization), a framework that trains reasoning LLMs to ground each modeling step in explicit textual evidence. TAGO first constructs StepTrace-OR, a dataset that links every modeling step to verbatim evidence from the problem statement, and then trains models with evidence-aligned supervised fine-tuning followed by curriculum reinforcement learning, using a composite reward that targets binding validity and key-evidence recovery. Across four backbones and five OR benchmarks, TAGO consistently outperforms state-of-the-art baselines on both Pass@1 and Pass@8, improving average Pass@1 by up to **8.14** points over the strongest baseline. Further analyses also show that TAGO substantially reduces misalignment and omission errors and produces more credible evidence chains. Our code is available at `https://anonymous.4open.science/r/TAGO_code-245D/`.

1 INTRODUCTION

Operations Research (OR) studies how to make optimal decisions under limited resources and underpins applications ranging from logistics scheduling and production planning to financial investment (Cohen & Sherkat, 1987; Jayal et al., 2010). In practice, OR problems typically begin as natural-language descriptions of decisions, objectives, and constraints, which must be translated into mathematical formulations that an optimization solver can execute. This translation requires identifying the relevant entities, binding each quantity to the correct entity, and expressing every stated condition as a constraint. Because this process has traditionally been carried out manually by domain experts, optimization modeling remains labor-intensive and difficult to scale.

Large language models (LLMs), with their strong capabilities in language understanding, reasoning, and code generation, offer a promising path toward automating OR modeling (Xiao et al., 2025; Huang et al., 2025b). Existing approaches in this field fall into three main lines. Prompting-based methods such as OptiMUS (AhmadiTeshnizi et al., 2023) decompose modeling into structured, multi-step reasoning; data-centric methods curate OR-specific corpora for instruction tuning (Huang et al., 2025a; Jiang et al., 2025); and more recent work employs reinforcement learning with solver feedback or process reward models to supervise intermediate reasoning steps (Chen et al., 2025; Zhou et al., 2025; Zhao et al., 2026). Together, these efforts have substantially improved the executability and final-answer accuracy of generated OR formulations.

However, executability and answer accuracy are only indirect proxies for OR modeling correctness: they do not verify whether a formulation faithfully represents the problem as stated. A model may attach a quantity to the wrong entity or silently omit a condition, yet still produce code that runs. To characterize such errors, we perform a fine-grained diagnosis of failure cases from representative state-of-the-art methods under a unified

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.

error rubric (Appendix B.1). We find that **mapping misalignment**, where a modeling element is bound to the wrong entity or condition, accounts for 56.7% of the diagnosed failures, and **information omission**, where a stated condition is missing from the formulation, accounts for another 32.6%. In other words, nearly 90% of failures stem not from coding or solver issues, but from a mismatch between the formulation and its source problem statement, a limitation that current methods largely overlook.

These two error types call for checking a formulation against the problem statement in both directions. Specifically, mapping misalignment requires checking whether an expression is tied to the correct entity or condition, while information omission requires checking whether relevant conditions have been incorporated at all. Human experts routinely perform such checks, *referring back* to the problem statement to justify each variable and constraint and to confirm that nothing has been left out. We term this ability to verify the correspondence between modeling decisions and the original problem statement in both directions **Traceability Awareness**, and argue that it is a key missing ingredient in current LLM-based optimization modeling.

Equipping LLMs with such traceability awareness, however, poses two critical challenges. First, *data*: existing OR datasets pair problems with final formulations or answers but rarely align individual modeling decisions with their supporting textual evidence, leaving no fine-grained signal from which to learn the traceback process. Second, *supervision*: existing training objectives reward executability and answer correctness, providing little signal on whether the reasoning process is consistent with the problem statement. As a result, models can fall into a regime of "successful execution with flawed logic."

To address these challenges, we propose **TAGO** (**T**raceability-**A**ware **G**uided **O**ptimization), a framework that trains reasoning LLMs to ground every modeling step in explicit textual evidence from problem statement. TAGO addresses the two challenges in two corresponding phases.

Phase 1: Step-level evidence-chain data construction. We build **StepTrace-OR**, a step-level traceable OR dataset in which every mathematical expression is explicitly linked to verbatim evidence from the problem statement. Evidence chains are generated by a strong teacher model and refined through multi-stage filtering.

Phase 2: Evidence-aligned supervision. Building on StepTrace-OR, we first perform evidence-aligned supervised fine-tuning (SFT), where each target interleaves a structured formulation, its verbatim evidence, and executable solver code, yielding an evidence-grounded reference policy. We then apply Group Relative Policy Optimization (GRPO) (Shao et al., 2024) with a carefully designed composite reward whose two traceability terms mirror the two failure modes: the *Binding Validity Rate* (BVR) measures whether each cited piece of evidence actually supports its attached expression, targeting mapping misalignment, while *Key-Evidence Recovery* (KER) measures how much annotated key evidence is recovered while penalizing unmatched citations, targeting information omission. Combined with format compliance and a graded solver-outcome reward, our design supervises traceability without sacrificing solution correctness. The RL training stage follows an easy-to-hard curriculum that first consolidates basic evidence binding ability and then shifts to harder problems where misalignment and omission are more prevalent.

The main contributions of this paper are as follows:

- **Conceptually**, through a systematic failure analysis, we show that mapping misalignment and information omission account for nearly 90% of failures in current LLM-based OR modeling, and we formalize *traceability awareness*, the bidirectional correspondence between modeling decisions and the problem statement, as a principle for addressing them for the first time.
- **Methodologically**, we construct the StepTrace-OR dataset by transforming conventional problem–answer data into stepwise data with step-level evidence chains, and propose an evidence-aligned training recipe that combines SFT with curriculum RL under traceability-aware composite reward design.
- **Empirically**, across four backbones and five benchmarks, TAGO consistently outperforms state-of-the-art baselines in both Pass@1 and Pass@8, with the largest gain in average Pass@1 over the strongest baseline reaching 8.14 points. Additional analyses further show that TAGO produces higher-quality evidence chains with fewer omission and misalignment errors.

2 Related Work

LLM for OR Modeling. Recent surveys (Xiao et al., 2025) organize LLM-based OR modeling around prompting, domain adaptation, process supervision, and multi-agent workflows. Early work such as OptiGuide (Li et al., 2023) uses LLMs to answer what-if queries over existing optimization models. Chain-of-Experts (Xiao et al., 2024), OptiMUS (AhmadiTeshnizi et al., 2023), and Autoformulation (Astorga et al.,

2025) decompose formulation into coordinated generation, verification, or search stages. ORLM (Huang et al., 2025a), LLaMoCo (Ma et al., 2024), and LLMOPT (Jiang et al., 2025) instead improve model capability through synthetic data, instruction tuning, and alignment. Recently, SAGE (Zhao et al., 2026) makes the modeling strategy explicit and optimizes it with solver-verified multi-strategy data and strategy-focused credit assignment. Across these lines, the dominant training signal is execution feedback, which can reject non-runnable or numerically incorrect programs but cannot reveal whether each set, parameter, variable, objective term, and constraint are derived from the right source facts. TAGO targets this gap by supervising the correspondence between individual modeling decisions and the problem statement directly.

Data and Supervision for OR Modeling. NL4OPT (Ramamonjison et al., 2023) casts natural-language optimization as structured semantic parsing and annotates optimization entities and relations. OR-Instruct, the data pipeline behind ORLM (Huang et al., 2025a), ReSocratic (Yang et al., 2025b), and OptMATH (Lu et al., 2025) use semi-automatic, feedback-driven, or bidirectional synthesis to scale up training data, while OR-PRM (Wang et al., 2026) adds search and process rewards for intermediate reasoning. These resources improve supervision within the formulation pipeline, yet their records remain largely organized as problem–formulation, problem–code, or problem–solution pairs. The correctness labels cannot distinguish *information omission*, where a requirement never enters the formulation, from *mapping misalignment*, where a valid source fact is attached to the wrong mathematical entity. StepTrace-OR addresses this by moving supervision from the pair level to the step level by annotating each modeling step with verbatim source evidence.

3 METHODOLOGY

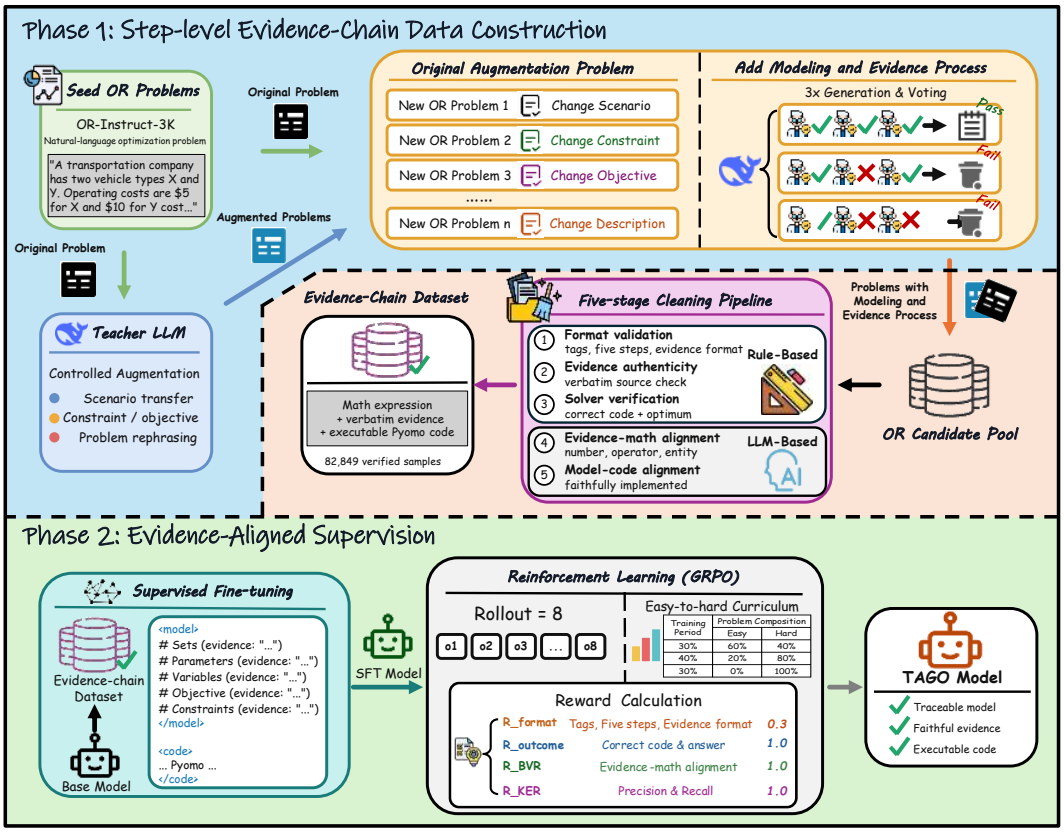


Figure 1: Overview of the TAGO training framework.

Our goal is to train an LLM to perform automating OR modeling with traceability awareness, so that every modeling decision can be checked against the problem statement. This goal imposes two important requirements. First, the model must reason in the right unit: a single modeling step paired with the textual evidence that supports it. Second, the training signal must separate three properties that execution feedback conflates:

whether the program is *executable and correct*, whether each modeling decision is *faithful* to its cited evidence, and whether the formulation is *complete*, *i.e.*, no key condition has been omitted. TAGO meets these requirements in two phases. Phase 1 (Section 3.1) constructs StepTrace-OR, whose demonstrations instantiate this unit of reasoning. Phase 2 (Section 3.2) first instills the evidence-grounded format through supervised fine-tuning, and then utilizes reinforcement learning with rewards that target the three properties separately: a solver-outcome reward for correctness, the Binding Validity Rate (BVR) for faithfulness, and Key-Evidence Recovery (KER) for completeness. Figure 1 gives an overview of our TAGO framework.

3.1 Phase 1: Step-level Evidence-Chain Data Construction

3.1.1 Seed data augmentation

We use OR-Instruct-3K (Huang et al., 2025a) as seed data and apply controlled augmentations to expand its scale and scenario coverage, so that sufficient high-quality training data remains after subsequent quality control. The augmentation modifies only the problem wording or a well-defined modeling condition, while ensuring that each augmented problem retains a checkable reference optimum and states every modeling condition explicitly in its text, so that each modeling step can later be grounded in a supporting evidence span. Using DeepSeek-v4-pro (DeepSeek-AI et al., 2026) as the teacher model, we apply three transformations: (i) **scenario transfer**, which adapts a problem to a new industry scenario and updates its optimal objective value accordingly; (ii) **constraint/objective transformation**, which modifies constraints or the optimization direction and updates the optimal value; and (iii) **problem rephrasing**, which rewrites the description while preserving the underlying mathematical model and optimal value. Details are provided in Appendix A.1.

3.1.2 Step-level Evidence Chains Generation

We adopt an OR-specific generation protocol that first expresses each problem in the canonical five-component form of *Sets*, *Parameters*, *Variables*, *Objective*, and *Constraints*. This structure yields formulations that are well-formed and solver-agnostic by construction, and each component corresponds to a distinct kind of modeling decision, making step omissions observable. For each step, the teacher must output both the mathematical expression and the verbatim source evidence that warrants it; Pyomo code is generated alongside the chain so that semantic faithfulness and executable behavior can be checked independently. To reduce dependence on a single teacher trajectory, we independently sample three responses per problem and use DeepSeek-v4-pro as the LLM judge to assess whether they yield equivalent formulations and solutions. If all three are judged equivalent, the problem and its evidence chain enter the candidate pool; otherwise, we resample for up to three rounds and discard problems that remain unresolved, rather than injecting ambiguous supervision. The construction prompts and judging criteria are provided in Appendix A.2.

3.1.3 Five-Stage Filtering

Candidate evidence chains pass through five filtering stages organized into two tiers: deterministic rule-based checks first, followed by semantic LLM-based checks. Malformed outputs, fabricated citations, and non-solving programs are removed by rules rather than by an LLM judge, and only the surviving samples undergo semantic verification. This ordering assigns each stage a single type of risk and prevents favorable semantic judgments from masking earlier validity failures.

- **Rule-based tier: Stage 1 (Format Validation)** checks tag completeness, the presence of all five components, evidence fields, and code integrity; **Stage 2 (Evidence Authenticity)** requires every normalized cited span to match the source problem exactly; and **Stage 3 (Solver Validation)** requires successful execution, a feasible solution, and an objective value within relative tolerance 10^{-4} of the reference optimum.
- **LLM-based tier:** Using DeepSeek-v4-pro as the judge, **Stage 4 (Evidence–Math Alignment)** verifies that numerical values, operator directions, and entity bindings in each expression are supported by its cited evidence; and **Stage 5 (Model–Code Alignment)** verifies that the Pyomo code correctly implements the stated sets, parameters, variable domains, objective, and constraint quantifiers.

A sample is retained only if it passes all five stages, ultimately yielding **StepTrace-OR**, a dataset of OR problems annotated with verified step-level evidence chains. Authenticity and semantic alignment are separate by design: a quoted span can be genuine yet support the wrong expression, while a semantically appropriate citation is unusable if it is not copied from the source problem. The final solver check imposes a third, complementary condition: the chain must compile into a feasible model that reproduces the reference optimal objective value. Appendix A.3 specifies the acceptance criteria.

3.2 PHASE 2: EVIDENCE-ALIGNED SUPERVISION

3.2.1 EVIDENCE-ALIGNED SUPERVISED FINE-TUNING

Data format. The filtered StepTrace-OR records form the SFT dataset $\mathcal{D}_{\text{SFT}}$. Each sample is a pair (q, y), where q is the problem statement and the target y consists of a model block followed by a code block:

$$y = (y_{\text{model}},\ y_{\text{code}}). \tag{1}$$

The model block contains five ordered steps $y_{\text{model}} = (s_1, \ldots, s_5)$, corresponding to sets, parameters, variables, objective, and constraints. Each step $s_k = \{(m_{k,j}, e_{k,j})\}_{j=1}^{n_k}$ is a collection of *bindings*, each pairing a mathematical expression $m_{k,j}$ with a verbatim evidence span $e_{k,j}$ from q. The code block contains executable Pyomo code implementing the model formulation.

Structural tags. The two blocks are delimited by `<model>...</model>` and `<code>...</code>`, and each evidence span is written inline as `(evidence: ``...'')`. These special tags serve two purposes. First, they make traceability checkable at the level of individual bindings, allowing the reward functions in Section 3.2.2 to distinguish missing components from incorrect ones. Second, they enforce a model-then-code solution order: the model first commits to a fully evidence-annotated formulation and then generates code that implements it, rather than emitting code directly from the question.

Training objective. We train the model by minimizing the standard negative log-likelihood:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\,\mathbb{E}_{(q,y)\sim\mathcal{D}_{\text{SFT}}}\big[\log \pi_\theta(y \mid q)\big]. \tag{2}$$

Beyond teaching the output structure required for reward computation, SFT yields a policy π_{ref} that already associates modeling decisions with source evidence. We use π_{ref} both to initialize RL and as its reference policy; without this initialization, RL would have to discover the output schema and the traceability behavior simultaneously from sparse rewards.

3.2.2 EVIDENCE-ALIGNED REINFORCEMENT LEARNING

In the RL stage, we turn traceability awareness into an explicit optimization target. Starting from π_{ref}, the policy generates outputs in the structured format described in Section 3.2.1. For an output y, let $\mathcal{B}(y) = \{(m_j, e_j)\}_{j=1}^{N}$ denote all its bindings across the five steps, and let $\hat{\mathcal{E}}(y)$ denote the set of cited evidence spans. We design a composite reward whose components target faithfulness, completeness, structure, and solution correctness, respectively.

Binding Validity Rate (BVR). BVR targets mapping misalignment by measuring the fraction of bindings in which the cited span supports the attached expression. An LLM judge, instantiated with DeepSeek-v4-pro checks each binding (m_j, e_j) for numerical consistency, operator direction, and entity correspondence:

$$R_{\text{BVR}}(y) = \frac{1}{N}\sum_{j=1}^{N} \mathbb{I}\big[(m_j, e_j) \text{ passes all criteria}\big], \tag{3}$$

with $R_{\text{BVR}} = 0$ when $N = 0$. Binding validity is distinct from source authenticity: a span copied verbatim from q still fails if it is bound to the wrong expression. The judging prompt is provided in Figure 5 in Appendix A.3.

Key-Evidence Recovery (KER). BVR alone can be inflated by citing only a few confident bindings. KER counteracts this by targeting information omission, comparing the predicted spans $\hat{\mathcal{E}}(y)$ with the annotated key evidence $\mathcal{E}^\star(q)$ in StepTrace-OR. After whitespace normalization, a predicted span is eligible for matching only if it is an exact substring of problem statement q; non-eligible spans are counted as false positives.

Eligible spans are aligned to annotated spans by character-level similarity,[1] retaining pairs with similarity at least 0.75, and we take a maximum-cardinality one-to-one matching $\mathcal{M}^\star$ over these pairs. Exact matching against problem statement q thus enforces authenticity, while the similarity threshold governs alignment to annotations. With $\text{TP} = |\mathcal{M}^\star|$, precision $P = \text{TP}/|\hat{\mathcal{E}}(y)|$ and recall $R = \text{TP}/|\mathcal{E}^\star(q)|$, we define

$$R_{\text{KER}}(y) = F_\beta = \frac{(1+\beta^2)\,P \cdot R}{\beta^2 P + R}, \qquad \beta^2 = 2, \tag{4}$$

[1] Computed with Python's `difflib.SequenceMatcher`.

with $R_{\text{KER}} = 0$ when $\hat{\mathcal{E}}(y)$ is empty. Here, precision penalizes superfluous or fabricated spans, while recall penalizes omissions and thus measures the completeness of the formulation. We set $\beta > 1$ to weight recall more heavily, since an omitted span can remove a required modeling element entirely and turn the model into a relaxation whose optimal solution may be infeasible for the original problem.

Format reward. We compute a bounded structural reward from the fraction of passed checks over required components, evidence annotations, and the code block:

$$R_{\text{format}}(y) = 0.3 \cdot \frac{N_{\text{pass}}}{N_{\text{check}}} \in [0, 0.3]. \tag{5}$$

Format compliance is an auxiliary structural signal rather than a measure of semantic or solver correctness. Its small range encourages adherence to the required output schema while preventing formatting alone from compensating for incorrect evidence binding, missing key evidence, or an incorrect optimization outcome.

Outcome reward. We define the following graded partial credit from solver execution:

$$R_{\text{outcome}}(y) = \begin{cases} 0, & \text{the program does not execute,} \\ 0.2, & \text{execution completes but no feasible solution is found,} \\ 0.4, & \text{a feasible solution has an incorrect objective value,} \\ 1.0, & \text{the objective matches the reference within tolerance.} \end{cases} \tag{6}$$

The total reward is then defined as:

$$R(y) = R_{\text{BVR}}(y) + R_{\text{KER}}(y) + R_{\text{format}}(y) + R_{\text{outcome}}(y). \tag{7}$$

Optimization and curriculum learning. We optimize the policy with GRPO, sampling $G = 8$ outputs per prompt. For each prompt, we compute the standard deviation s_r of the rewards over the G sampled outputs and retain the prompt only when $s_r > 0.05$ since prompts with nearly identical rewards provide no useful group-relative signal. Details are provided in Appendix A.4.1.

Training follows an easy-to-hard curriculum. We define a prompt as *easy* if the SFT reference policy π_{ref} solves it in a single rollout; otherwise it is *hard*. This criterion is computed once after SFT for every prompt. Early training emphasizes easy prompts, for which the policy can already find correct solutions, so that reward differences primarily teach it how to bind and recover evidence. The proportion of easy prompts is then reduced from 60% to 20% and finally to 0% over the first 30%, middle 40%, and final 30% of training steps, respectively, exposing the policy to harder problems where omission and misalignment are more prevalent.

4 Experiments

We organize the evaluation around four research questions: (1) whether TAGO improves optimization-modeling accuracy over prior methods (Section 4.2); (2) whether its supervised and reinforcement learning components contribute complementary gains (Section 4.3); (3) whether it improves evidence-chain quality and reduces the targeted modeling errors (Sections 4.4.1 and 4.4.2); and (4) where these gains are most pronounced across problem difficulty levels and sampling budgets (Sections 4.4.3 and 4.4.4).

4.1 Experiment Setup

Benchmarks. We evaluate TAGO on five commonly utilized optimization modeling benchmarks: NL4OPT, the EasyLP and ComplexLP subsets of MAMO, NLP4LP, and IndustryOR (Ramamonjison et al., 2023; Huang et al., 2025b; AhmadiTeshnizi et al., 2023; Huang et al., 2025a). For every method, we extract and execute the generated code in an isolated environment. A completion is correct only if execution returns a feasible solution whose objective has relative error below 10^{-4} against the reference optimum.

Baselines. We compare TAGO with several baselines including direct generation (zero-shot and chain-of-thought), offline training (ORLM (Huang et al., 2025a) and LLMOPT (Jiang et al., 2025)), online RL (OR-PRM (Wang et al., 2026) and SAGE (Zhao et al., 2026)), and two GPT-4o based agentic systems (Reflexion (Shinn et al., 2023) and OptiMUS). Direct baselines use the same Qwen2.5-3B/7B (Qwen et al., 2025) and Qwen3-4B/8B (Yang et al., 2025a) backbones. We retrain trainable baselines with the same prompts and decoding settings.

Table 1: Pass@1 results (%) on the five optimization modeling benchmarks. "Ave" is the benchmark-size-weighted average across the suite. Within each backbone and column, bold marks the best result and underline marks the second best, the same rule applies to all subsequent tables.

Backbone	Method	IndustryOR	EasyLP	ComplexLP	NL4OPT	NLP4LP	Ave
Qwen2.5-3B	Zero-Shot	0.00	9.51	0.00	25.11	23.91	12.99
	CoT	1.00	11.04	0.00	28.57	28.88	15.30
	OR-PRM	11.00	58.74	5.69	58.01	48.76	45.98
	LLMOPT	13.00	58.44	6.16	60.61	45.03	45.65
	ORLM	20.00	65.49	7.58	**67.10**	50.00	51.38
	SAGE	16.00	68.40	13.27	65.80	40.99	51.05
	TAGO	**22.00**	**77.91**	**18.01**	66.82	**55.90**	**59.52**
Qwen2.5-7B	Zero-Shot	6.00	20.71	5.69	26.84	26.09	19.73
	CoT	13.00	68.71	7.11	53.68	48.45	49.87
	OR-PRM	14.00	72.85	16.59	61.90	52.17	55.08
	LLMOPT	16.00	76.38	20.09	62.77	54.66	57.88
	ORLM	23.00	70.40	18.48	**78.79**	51.86	57.39
	SAGE	20.00	82.98	22.27	74.89	52.80	62.73
	TAGO	**29.00**	**86.66**	**23.22**	78.35	**59.32**	**66.95**
Qwen3-4B	Zero-Shot	8.00	23.47	6.16	25.54	25.78	20.85
	CoT	14.00	71.78	8.06	56.71	48.76	51.91
	OR-PRM	14.00	76.07	18.01	69.70	56.52	58.77
	LLMOPT	20.00	80.52	20.38	68.40	56.21	61.15
	ORLM	25.00	72.24	20.38	79.65	54.04	59.17
	SAGE	23.00	**86.04**	24.64	76.62	55.28	65.37
	TAGO	**32.00**	84.97	**28.44**	**80.09**	**60.87**	**67.75**
Qwen3-8B	Zero-Shot	14.00	34.51	11.85	29.44	47.20	31.93
	CoT	19.00	77.76	17.06	63.20	56.83	58.77
	OR-PRM	24.00	81.75	20.38	76.62	68.32	65.77
	LLMOPT	25.00	87.58	26.54	76.19	69.57	69.40
	ORLM	28.00	80.67	30.33	83.55	60.87	66.42
	SAGE	28.00	**88.96**	35.07	83.55	61.18	70.71
	TAGO	**38.00**	86.50	**38.39**	**88.74**	**72.36**	**73.94**
GPT-4o	Zero-Shot	27.00	59.36	35.07	52.38	42.55	49.21
	Reflexion (Agentic)	33.00	85.58	38.39	60.17	42.86	62.60
	OptiMUS (Agentic)	**37.00**	**86.50**	**45.50**	**64.50**	**46.58**	**65.70**

Implementation Details. We use DeepSeek-v4-pro to repeatedly sample candidates from the 3,000 OR-Instruct seeds across the three augmentation types, where a single seed can yield multiple retained records. After deduplication and five-stage validation through rule-based and LLM-based filters, we obtain 82,849 evidence-annotated samples from the pipeline, among which 74,564 are used for SFT and 8,285 for RL. We train Qwen2.5-3B/7B and Qwen3-4B/8B models with TRL-GRPO, setting eight trajectories per group and following the same curriculum schedule. We list all hyperparameters in Appendix A.4.

4.2 Main Result

Table 1 reports Pass@1 on the five benchmarks, and Appendix B.2 provides the complete Pass@1 and Pass@8 results.

Overall performance. TAGO achieves the best weighted average on all four LLM backbones, reaching 73.94% with Qwen3-8B. Its margin over the strongest baseline ranges from 2.38 points (Qwen3-4B) to 8.14 points (Qwen2.5-3B). The advantage is most consistent on the three most challenging benchmarks, IndustryOR, ComplexLP, and NLP4LP, where TAGO ranks first on all four backbones.

Comparison with baselines. Zero-Shot and CoT, which prompt the backbone without any training, remain well below all trained methods. Among trained baselines, SAGE is the strongest on three of the four backbones, yet it falls short of TAGO's weighted average in every case.

Comparison with agentic GPT-4o. TAGO with Qwen3-8B surpasses the agentic OptiMUS built on GPT-4o in weighted average (73.94 vs. 65.70), with large gains on NL4OPT and NLP4LP. Even with smaller backbones, TAGO exceeds OptiMUS in weighted average (66.95 with Qwen2.5-7B and 67.75 with Qwen3-

Table 2: Ablation results on Qwen2.5-7B and Qwen3-8B. Values are benchmark-size-weighted aggregate Pass@1 percentages; Δ is measured against full TAGO.

Stage	Variant	Qwen2.5-7B Ave.	Δ	Qwen3-8B Ave.	Δ
SFT+RL	**Full TAGO (2:1)**	**66.95**	0.00	**73.94**	0.00
SFT	Evidence-aligned SFT	57.65	-9.30	61.74	-12.20
	Plain SFT (w/o evidence)	52.18	-14.77	56.99	-16.95
RL	w/o format	66.23	-0.72	72.63	-1.31
	Binary outcome reward $(0, 1)$	65.43	-1.52	71.70	-2.24
	w/o R_{KER}	65.17	-1.78	70.38	-3.56
	w/o R_{BVR}	64.32	-2.63	68.60	-5.34
Weight	Balanced KER-F_β (1:1)	66.23	-0.72	72.50	-1.44
	P-heavy KER-F_β (1:2)	64.64	-2.31	66.82	-7.12

4B), showing that evidence-grounded training lets compact open models rival agentic systems built on a much larger proprietary model.

4.3 Ablation Studies

We isolate the contributions of (i) evidence annotations in SFT targets, (ii) the RL stage on top of SFT, (iii) the structured format and outcome-reward design, and (iv) the binding-validity and key-evidence-recovery rewards. Full TAGO sets $\beta^2 = 2$ in R_{KER}, corresponding to a 2:1 R-to-P emphasis within the KER score.

Table 2 summarizes the ablation results, and Appendix B.3 provides the complete benchmark-level results. (i) Plain SFT without evidence annotations degrades substantially, showing evidence-aware supervision is essential. (ii) The RL stage adds a further 9.30 and 12.20 points on top of evidence-aligned SFT, making it the largest single contributor. (iii) Replacing the graded outcome reward with a binary one costs 1.52 and 2.24 points, while removing the format reward yields a modest but consistent drop. (iv) Among the evidence-aligned rewards, removing R_{BVR} or R_{KER} lowers the weighted average by 5.34 and 3.56 points on Qwen3-8B, respectively. Finally, the recall-oriented $\beta^2 = 2$ setting outperforms both the balanced and precision-heavy alternatives, with the precision-heavy variant dropping by 7.12 points on Qwen3-8B, suggesting that preventing omissions matters more than filtering redundant evidence citations.

4.4 Further Analyses

4.4.1 Evidence-Chain Quality Experiment

To test whether evidence-aligned post-training improves evidence-chain quality alongside modeling accuracy, we draw a benchmark-stratified random sample of 300 problems from the test suite. Because baselines do not natively produce evidence chains, we apply the same post-hoc evidence extractor to the outputs of all methods, including TAGO, so that no method benefits from its native output format. Accuracy is computed on the full sample, whereas evidence-chain metrics are computed only on each method's correct outputs. Table 3 summarizes the results, and Appendix B.4.1 details the full evaluation protocol. TAGO achieves the highest accuracy (74.0%) and the highest $F_{1,\mathrm{KER}}$ (50.89%), exceeding the strongest baseline, LLMOPT, by 14.62 points in $F_{1,\mathrm{KER}}$. It also attains the highest precision and recall, indicating that our post-training method can effectively improve the evidence-chain quality of LLMs in OR problem solving. Notably, SAGE achieves higher accuracy than LLMOPT but a lower $F_{1,\mathrm{KER}}$, indicating that correct answers do not necessarily come with reliable evidence chains.

Table 3: Evidence-chain quality (%) on a shared, stratified 300-instance Qwen3-8B test sample using the unified post-hoc extractor.

Method	Accuracy	Pre_{KER}	Rec_{KER}	$F_{1,\mathrm{KER}}$
TAGO	**74.00**	**49.20**	**52.70**	**50.89**
LLMOPT	69.00	39.40	33.60	36.27
SAGE	71.00	36.80	31.50	33.94
OR-PRM	66.00	31.20	23.80	26.99
ORLM	66.00	27.60	20.40	23.46

4.4.2 Error Breakdown Experiment

To test whether evidence-chain supervision reduces the modeling errors it targets, we analyze the 135 instances in the full test suite on which TAGO, OR-PRM, and SAGE all fail. Each failure is labeled by an LLM

judge along four error categories, and Table 4 summarizes the resulting breakdown. Because a single failure can involve multiple error types, the table reports the prevalence of each category rather than a partition of failures. In Section 1, we attribute each failure to a single upstream cause to identify its root source, so code and solver errors may appear more frequently here. Appendix B.4.2 details the judge prompts.

On this shared subset, TAGO shows markedly lower prevalence of modeling errors, with 11.11% omission and 12.59% misalignment, compared with at least 34% omission and 51% misalignment for both OR-PRM and SAGE. This pattern is consistent with the intended roles of R_{BVR} and R_{KER}. In contrast, TAGO's code error rate (49.63%) and solver error rate (31.11%) remain comparable to those of the baselines, suggesting that its accuracy gains stem primarily from more faithful problem formulation rather than from improved general code generation.

Table 4: Error-type rates (%) on the same 135 shared failed instances in the Qwen3-8B test sample.

Method	Modeling errors		Coding errors	
	Omission	Misalignment	Code Error	Solver Error
TAGO	**11.11**	**12.59**	49.63	**31.11**
OR-PRM	36.30	51.85	**47.41**	34.81
SAGE	34.07	56.30	51.11	37.78

Note: The four error types are non-exclusive and may co-occur, so their percentages can sum to over 100%.

4.4.3 Difficulty Analysis Experiment

To identify where evidence-aligned training helps most, we stratify the Qwen3-8B results on IndustryOR by its native difficulty labels and compare TAGO with SAGE, the strongest baseline on this backbone. As shown in Table 5, TAGO outperforms SAGE on all three subgroups, and its lead widens with difficulty: +5.13 on Easy, +12.20 on Medium, and +15.00 on Hard. This trend matches the mechanism of step-level grounding. Harder problems require more conditions, parameters, and constraints to be mapped from the problem statement into the formulation, which raises the risk of information omission and mapping misalignment, precisely the two error types that TAGO's evidence-aligned rewards target. These results suggest that the benefit of evidence grounding increases with the complexity of the modeling task.

Table 5: IndustryOR Pass@1 (%) by native difficulty label. Δ is the TAGO–SAGE difference.

Difficulty	TAGO Pass@1	SAGE Pass@1	Δ
Easy	**48.72**	43.59	+5.13
Medium	**31.71**	19.51	+12.20
Hard	**30.00**	15.00	+15.00
Overall	**38.00**	28.00	+10.00

4.4.4 Pass@K Performance Experiment

To examine how TAGO scales with the inference sampling budget, we compare it with SAGE and OR-PRM on Qwen3-8B under the same solver protocol at $K \in \{1, 2, 4, 8, 16\}$. Figure 2 shows the weighted Overall curve, while Appendix B.4.3 reports the per-benchmark curves and corresponding values. On the weighted average, Pass@K increases with K for all methods, and TAGO leads at every budget. Its margin over SAGE and OR-PRM remains stable from $K = 1$ to $K = 16$, indicating that evidence-aligned RL broadens the coverage of correct formulations rather than merely sharpening the output distribution toward a few high-reward samples. TAGO is also more sample-efficient: it matches or exceeds SAGE with only half the sampling budget, and with only two samples it approaches the Pass@16 of OR-PRM.

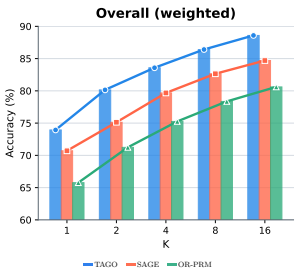


Figure 2: Pass@K results.

5 Conclusion

We presented TAGO, which is, to the best of our knowledge, the first traceability-aware framework tailored for LLM-based OR modeling. Our failure analysis shows that most errors in current methods arise not from code execution but from unfaithful modeling decisions, namely mapping misalignment and information omission, which executability and answer accuracy cannot detect. To address this, we constructed StepTrace-OR, a dataset that links every modeling step to verbatim source evidence, and trained models through evidence-aligned SFT followed by curriculum RL with rewards targeting binding validity and key-evidence recovery. With Qwen3-8B, TAGO achieves an average of 73.94% Pass@1 and 86.41% Pass@8 across five OR benchmarks, while reducing both error types and improving evidence-chain quality.

AI USE STATEMENT

Generative AI models are an integral part of our methodology. We use LLMs to synthesize the training data in the five-step formulation format, to filter candidate samples during data cleaning, and to serve as the judge that computes the binding-validity reward during RL training. LLMs are also used in our evaluation, including the post-hoc evidence extractor and error-category labeling. The models, prompts, and settings for each of these uses are documented in the corresponding sections and appendices. In addition, we used generative AI tools for language polishing and LaTeX formatting, and all such outputs are reviewed by the authors. The authors will take full responsibility for the content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The complete data construction pipeline, including the controlled augmentation templates, step-level evidence-chain extraction prompts, and acceptance criteria, is described in the appendix. The appendix also provides training details, including hyperparameters, reward coefficients, and the prompts for the LLM judge. All experiments use publicly available benchmarks: NL4OPT, MAMO (EasyLP and ComplexLP), NLP4LP, and IndustryOR. We also release our source code at `https://anonymous.4open.science/r/TAGO_code-245D/`.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Optimization modeling using mip solvers and large language models, 2023. URL https://arxiv.org/abs/2310.06116.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using llms, 2025. URL https://arxiv.org/abs/2411.01679.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed rl: Grounding large language models for authentic optimization modeling, 2025. URL https://arxiv.org/abs/2505.11792.

A.I. Cohen and V.R. Sherkat. Optimization-based methods for operations scheduling. *Proceedings of the IEEE*, 75(12):1574–1591, 1987. doi: 10.1109/PROC.1987.13928.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang, Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang

Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, November 2025a. ISSN 1526-5463. doi: 10.1287/opre.2024.1233. URL http://dx.doi.org/10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages, 2025b. URL https://arxiv.org/abs/2405.13144.

A.D. Jayal, F. Badurdeen, O.W. Dillon, and I.S. Jawahir. Sustainable manufacturing: Modeling and optimization challenges at the product, process and system levels. *CIRP Journal of Manufacturing Science and Technology*, 2(3):144–152, 2010. ISSN 1755-5817. doi: https://doi.org/10.1016/j.cirpj.2010.03.006. URL https://www.sciencedirect.com/science/article/pii/S1755581710000131. Sustainable Development of Manufacturing Systems.

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, 2025. URL https://arxiv.org/abs/2410.13213.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization, 2023. URL https://arxiv.org/abs/2307.03875.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling, 2025. URL https://arxiv.org/abs/2502.11102.

Zeyuan Ma, Hongshu Guo, Jiacheng Chen, Guojun Peng, Zhiguang Cao, Yining Ma, and Yue-Jiao Gong. Llamoco: Instruction tuning of large language models for optimization code generation, 2024. URL https://arxiv.org/abs/2403.01131.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rindranirina Ramamonjison, Timothy T. Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions, 2023. URL https://arxiv.org/abs/2303.08233.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, 2 edition, 1979.

Yilin Wang, Heng Zhou, Dongxing Mao, Linjie Li, Jingru Tan, Haochen Han, Zhengyuan Yang, Alex Jinpeng Wang, and Min Li. OR-PRM: A Process Reward Model for Algorithmic Problem in Operations Research. In *International Conference on Learning Representations*, 2026. URL https://mlanthology.org/iclr/2026/wang2026iclr-orprm/.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica 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. URL https://openreview.net/forum?id=HobyL1B9CZ.

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, 2025. URL https://arxiv.org/abs/2508.10047.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025a. URL https://arxiv.org/abs/2505.09388.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. Optibench meets resocratic: Measure and improve llms for optimization modeling, 2025b. URL https://arxiv.org/abs/2407.09887.

Ruiqing Zhao, Fengzhi Li, Yuan Zuo, Rui Liu, Yansong Liu, Yunfei Ma, Fanyu Meng, and Junlan Feng. Strategy-aware optimization modeling with reasoning llms, 2026. URL https://arxiv.org/abs/2605.02545.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. Steporlm: A self-evolving framework with generative process supervision for operations research language models, 2025. URL https://arxiv.org/abs/2509.22558.

APPENDIX CONTENTS

A Data and Prompt Details **13**

A.1 Data Construction Prompt Templates 13

A.2 Step-level Evidence Chains Prompt Templates 14

A.3 Validation and Reward Judging 15

A.4 Training Settings 17

B Experimental Results **18**

B.1 Error Diagnosis 18

B.2 Main Results 19

B.3 Ablation Studies 20

B.4 Additional Analyses 20

C Case Study **25**

C.1 Information Omission Case Study 25

C.2 Mapping Misalignment Case Study 26

C.3 Evidence-Chain Data Case Study 27

A DATA AND PROMPT DETAILS

A.1 DATA CONSTRUCTION PROMPT TEMPLATES

Figure 3 shows the augmentation interface applied repeatedly to 3,000 OR-Instruct seeds. After deduplication, preliminary checks, and full five-stage validation, 82,849 records remain.

The mutation rules supplied in `{rule}` are: *scenario transfer* renames industry entities and rewrites numerical values while preserving the constraint topology; *constraint/objective transformation* adds, removes, or alters a justified mathematical condition or objective direction and updates the optimum; and *problem rephrasing* rewrites the full question while preserving the model and optimum. We remove duplicate questions using normalized text and model signatures before the preliminary model/optimum check and subsequent evidence-chain validation.

SEED DATA AUGMENTATION PROMPT

SYSTEM PROMPT: SEED DATA AUGMENTATION

You are an expert in Operations Research dataset construction. Given a source optimization problem, generate exactly one controlled augmentation and return one JSON object. Do not include executable solver code.

Apply exactly one requested augmentation type:

1. Scenario transfer: adapt the context to a new industry scenario while preserving the optimization skill and constraint topology.
2. Constraint/objective transformation: modify a justified constraint or objective direction and update the mathematical model and optimum.
3. Problem rephrasing: rewrite the description while preserving the mathematical model and optimal value.

Then verify the LP or MILP structure, variable domains, boundedness, and feasibility of the claimed optimum. Fill augmentation_type, problem_class, change_summary, changed_facts, and feasibility_notes. Recompute a plausible numerical optimum and check that it agrees with the objective and constraints.

Do not write or run code.

If a check fails, rewrite only the first failing part before producing the final object. Return JSON only, with this schema:

{"augmentation_type":"scenario_transfer|constraint_or_objective_transformation|problem_rephrasing",
"question":"complete self-contained problem",
"problem_class":"LP|MILP",
"change_summary":" what changed",
"changed_facts":[{"before":"...","after":"..."}],
"claimed_expert_value":0.0,
"feasibility_notes":"consistency and feasibility check"}

USER PROMPT: CONTROLLED AUGMENTATION

Create one {augmentation_type} augmentation.

Mutation rule:

{rule}

Source optimum (for reference; recompute the new value):

{expert_value}

Source problem:

{question}

Return exactly the JSON object matching the required schema.

RETRY PROMPT

{original_prompt}

Your previous candidate failed automatic validation.

Validation errors:

{failure_errors}

Previous candidate:

{previous_response}

Return a corrected complete JSON object. Preserve the requested augmentation type, fix JSON escaping, the mathematical model, and theclaimed_expert_value as needed. Return JSON only.

Figure 3: Data construction prompt templates.

A.2 Step-level Evidence Chains Prompt Templates

Figure 4 shows the prompt used for step-level evidence chains generation.

The decoder is instructed to emit no explanation outside the two tagged blocks. During preprocessing, we extract the five component lines and their quoted spans, normalize whitespace, require each span to exactly occur in the normalized source, and preserve the verbatim evidence in the stored target. Three independent responses are sampled for each problem. Equivalent model–code pairs enter the candidate pool. When the responses disagree, the teacher resamples the full group; after at most three rounds, an instance that still has no agreement is discarded rather than passed to downstream validation. The agreed evidence-chain targets are then sent to Appendix A.3, where the five-stage validation pipeline applies the rule-based filter (Stages 1–3) and the LLM-based filter (Stages 4–5).

STEP-LEVEL EVIDENCE CHAINS PROMPT

SYSTEM PROMPT: EVIDENCE CHAIN CONSTRUCTION

You are an expert operations-research modeler. Given an OR problem, you must:

1. Build a complete mathematical model with sets, parameters, variables, objective, and constraints.
2. For every modeling component, attach evidence that exactly quotes the original problem statement.
3. Provide complete runnable Pyomo code implementing the model.
4. Use exactly this output structure:

<model>

#sets: ... (evidence: "...")
#parameter: ... (evidence: "...")
#variable: ... (evidence: "...")
#objective: ... (evidence: "...")
#constraint: ... (evidence: "...")

</model>

<code>

```python
... complete Pyomo code ...
```

</code>

For table evidence, use | to separate row and column. Use
(evidence: "RowName|ColumnName") for one cell,
(evidence: "RowName|ColumnName1, ColumnName2") for one row, and
(evidence: "RowName1, RowName2|ColumnName") for one column.

USER INPUT

Below is an operations-research question. Build the mathematical model,evidence chains, and corresponding Pyomo code.

Question: {question}
Response:

Figure 4: Step-level evidence chains prompt.

A.3 Validation and Reward Judging

We apply a five-stage validation pipeline to each generated candidate evidence chain. Stages 1–3 form the rule-based filter, and Stages 4–5 form the LLM-based filter. A candidate is retained only when it passes all five stages. The deterministic stages run first, so no LLM-as-Judge call is made for a candidate that is malformed, cites inauthentic evidence, fails to execute, is infeasible, or produces an incorrect objective value.

Rule-based filter (Stages 1–3).

1. **Stage 1: Format validation (rule).** Check the completeness of the `<model>` and `<code>` tags, all five components, an `(evidence:''...'')` field for every mathematical expression, and a complete Python/Pyomo program. Any failure is discarded.
2. **Stage 2: Evidence authenticity (rule).** Normalize whitespace in each plain-text span and the problem, then require the span to be an exact substring of the normalized problem; otherwise, discard the candidate. For `row|column` table citations, every row and column identifier must exactly match a source label, and the expression value must equal the identified cell value. No fuzzy threshold is used for authenticity.
3. **Stage 3: Solver validation (rule).** Run the extracted code in an isolated environment. Require successful execution and a feasible solution; compare its objective with the standard answer using relative error 10^{-4}. Candidates without a valid solution or outside the tolerance are discarded.

LLM-based filter (Stages 4–5).

4. **Stage 4: Evidence–math alignment (LLM-as-Judge: DeepSeek-v4-pro).** Check numerical consistency, operator direction (for example, "at least" implies $\geq$ and "at most" implies $\leq$), and entity binding. A statement such as "at least 30 vehicles" paired with $x \leq 30$ is marked `FAIL`. Stage 4 passes only when every evidence–expression pair is labeled `PASS`.
5. **Stage 5: Model–code alignment (LLM-as-Judge: DeepSeek-v4-pro).** Check that set definitions match, parameter keys agree with `Param` declarations, variable domains match the mathematical definition (`NonNegativeReals`, `Integers`, or `Binary`), the objective is complete, and constraint loops implement the stated quantifiers. Stage 5 passes only when the judge returns `PASS`.

Evidence–math alignment judge prompt. Figure 5 shows the Stage 4 evidence–math judge used in both the LLM-based filter and the mapping-alignment BVR reward calculation. Every evidence–expression pair must receive `PASS`; fractional aggregation is used only for RL reward scoring.

EVIDENCE–MATH ALIGNMENT JUDGE PROMPT

SYSTEM PROMPT:

You are an operations-research model auditor. Judge only the consistency between the source problem, each quoted evidence span, and the mathematical expression that claims to use it. Do not repair the model.

For every evidence-expression pair, check:

1. Numerical consistency: every number, coefficient, bound, and unit is supported by the quoted text.
2. Direction consistency: min/max, <=/>=, equality, and logical operators match the source statement.
3. Entity and index consistency: products, locations, periods, resources, and indices refer to the same entities in the source and expression.
4. Scope consistency: the evidence is sufficient for the full expression; do not infer an unstated constraint or relationship.

Return JSON only:

{"pairs":[{"id":1,"label":"PASS|FAIL ",

"reason":"short reason"}]}

Use label=FAIL whenever any required check is unsupported.

Source problem:

{question}

Generated model:

{model}

Evidence spans:

{evidence}

Figure 5: Evidence–math alignment judge prompt

Model–code alignment judge prompt. Figure 6 shows the Stage 5 model–code consistency check used in the LLM-based filter; code execution is verified separately by the rule-based check in Stage 3.

MODEL–CODE ALIGNMENT JUDGE PROMPT

SYSTEM PROMPT:

You are an operations-research implementation auditor. Determine whether the Pyomo code faithfully implements the mathematical model. Do not repair the code or infer missing model components.

Check: (1) sets and indices, (2) parameter names, keys, and values, (3)variable domains and bounds, (4) the complete objective and optimizationdirection, and (5) every constraint, its quantifier, and its loop indices.

Return JSON only:

{"label":"PASS|FAIL","reason":"short reason"}

Return PASS only if every stated component is implemented faithfully.

Mathematical model:

{model}

Pyomo code:

{code}

Figure 6: Model–code alignment judge prompt

Unified post-hoc evidence extraction protocol. Figure 7 shows the unified extractor used for every method in the evidence-chain quality evaluation. It returns verbatim supporting spans, which are matched to annotated evidence with `SequenceMatcher` at threshold 0.75.

For evaluation, evidence-chain metrics are computed by an independent judge that is separate from the judge used in data construction, reward computation, and the Stage 4–5 filtering pipeline. We also manually review a random sample of extracted evidence chains to audit evidence-span fidelity and entity binding.

STANDARDIZED POST-HOC EVIDENCE PROTOCOL PROMPT

SYSTEM PROMPT:

You are an operations-research evidence attribution auditor. Given a source problem and a completed mathematical model/code output, identify only verbatim source spans that directly support explicit modeling decisions. Do not paraphrase, infer unstated facts, or repair the model. If no direct source span supports a decision, omit that decision.

Return JSON only:

{"links":[{"decision":"short model decision",

"evidence":"exact source span"}]}

Return {"links":[]} when no grounded evidence link is available.

Source problem:

{question}

Model/code output:

{output}

Figure 7: Unified post-hoc evidence extraction protocol

A.4 Training Settings

Table 6 summarizes the SFT and RL configurations. All runs use one node with eight NVIDIA RTX 5880 (48GB) GPUs.

Table 6: Key hyperparameters for SFT and GRPO training.

Stage	Type	Parameter	Value
SFT	**Data**	Optimizer	AdamW
		Learning rate	2×10^{-4}
		Max sequence length	8192
		Training epochs	3
RL	**Algorithm**	Advantage estimator	GRPO
	Data	Batch size (per-device)	2
		Gradient accumulation	4
		Learning rate	2×10^{-6}
		Max prompt length	2048
		Max response length	8192
		Rollout number	8
	Actor/Rollout	KL loss coefficient	0.002
	Reward	R_{BVR} weight	1.0
		R_{KER} weight	1.0
		R_{format} range	$[0, 0.3]$
		R_{outcome} weight	1.0
		KER-F_β setting	$\beta^2 = 2$ ($\beta = \sqrt{2}$), R:P$= 2:1$

A.4.1 Token-Level GRPO Objective

For completeness, at token context $h_{i,t} = (q, o_{i,<t})$, the importance ratio and sampled KL estimator used in training are

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid h_{i,t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid h_{i,t})}, \qquad d_{i,t}^{\text{KL}} = \frac{\pi_{\text{ref}}(o_{i,t} \mid h_{i,t})}{\pi_\theta(o_{i,t} \mid h_{i,t})} - \log \frac{\pi_{\text{ref}}(o_{i,t} \mid h_{i,t})}{\pi_\theta(o_{i,t} \mid h_{i,t})} - 1. \tag{8}$$

The full length-normalized clipped objective is

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{\substack{q \sim P(Q), \\ \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|q)}} \Bigg[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{T_i} \sum_{t=1}^{T_i} \Big(\min\big(\rho_{i,t} \hat{A}_{i,t}, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) \hat{A}_{i,t}\big) - \lambda_{\text{KL}} d_{i,t}^{\text{KL}} \Big) \Bigg]. \tag{9}$$

B Experimental Results

B.1 Error Diagnosis

For the diagnostic analysis, we use the following definitions. **Mapping Misalignment** occurs when the model recognizes relevant semantic information but binds it to the wrong entity or symbolic relationship. **Information Omission** occurs when key information from the problem statement is not extracted into the mathematical model. **Other Errors** comprise failures that are not attributable to either of the above, primarily non-cognitive failures such as code syntax errors, API misuse, or solver execution failures. Importantly, when a downstream error (*e.g.*, a code or solver error) is caused by an upstream mapping misalignment or information omission, we attribute the failure to the upstream modeling error rather than to the downstream symptom. Figure 8 summarizes the distribution of these three categories.

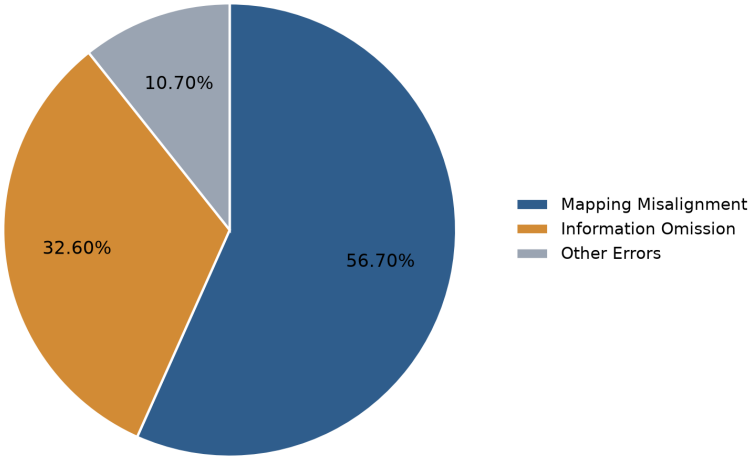


Figure 8: Distribution of diagnosed failure types. Mapping Misalignment is the largest category, followed by Information Omission and Other Errors.

B.2 Main Results

Table 7 reports the full Pass@1 and Pass@8 results for Section 4.2, where "Ave" is weighted by benchmark size rather than taken as an unweighted mean.

Table 7: Pass@1/Pass@8 results (%) across five benchmarks. "Ave" is benchmark-size-weighted; bold and underline mark the best and second-best values.

Backbone	Method	IndustryOR		EasyLP		ComplexLP		NL4OPT		NLP4LP		Ave	
		P@1	P@8	P@1	P@8	P@1	P@8	P@1	P@8	P@1	P@8	P@1	P@8
Qwen2.5-3B	Zero-Shot	0.00	3.00	9.51	32.52	0.00	0.95	25.11	42.86	23.91	47.20	12.99	30.87
	CoT	1.00	4.00	11.04	42.02	0.00	1.90	28.57	55.41	28.88	49.69	15.30	37.60
	OR-PRM	11.00	20.00	58.74	80.06	5.69	16.59	58.01	83.55	48.76	64.91	45.98	64.58
	LLMOPT	13.00	21.00	58.44	82.06	6.16	19.91	60.61	81.39	45.03	60.87	45.65	64.78
	ORLM	20.00	29.00	65.49	83.13	7.58	22.27	**67.10**	88.31	50.00	64.29	51.38	67.88
	SAGE	16.00	26.00	68.40	84.05	13.27	26.07	65.80	87.88	40.99	62.73	51.05	68.21
	TAGO	**22.00**	**34.00**	**77.91**	**87.88**	**18.01**	**27.96**	66.82	**89.17**	**55.90**	**67.08**	**59.52**	**71.76**
Qwen2.5-7B	Zero-Shot	6.00	18.00	20.71	43.71	5.69	16.11	26.84	49.78	26.09	49.69	19.73	40.37
	CoT	13.00	22.00	68.71	80.98	7.11	25.12	53.68	76.19	48.45	60.25	49.87	64.18
	OR-PRM	14.00	32.00	72.85	85.12	16.59	28.91	61.90	84.85	52.17	66.46	55.08	69.79
	LLMOPT	16.00	35.00	76.38	87.42	20.09	36.02	62.77	88.74	54.66	65.53	57.88	72.36
	ORLM	23.00	38.00	70.40	88.50	18.48	31.75	**78.79**	91.77	51.86	65.22	57.39	72.82
	SAGE	20.00	35.00	82.98	90.49	22.27	39.81	74.89	90.04	52.80	66.46	62.73	74.60
	TAGO	**29.00**	**42.00**	**86.66**	**92.64**	**23.22**	**45.02**	78.35	**92.21**	**59.32**	**69.88**	**66.95**	**77.77**
Qwen3-4B	Zero-Shot	8.00	18.00	23.47	45.40	6.16	18.96	25.54	45.89	25.78	47.83	20.85	40.50
	CoT	14.00	26.00	71.78	85.12	8.06	32.23	56.71	79.65	48.76	62.17	51.91	68.15
	OR-PRM	14.00	35.00	76.07	88.50	18.01	31.28	69.70	87.01	56.52	71.74	58.77	73.22
	LLMOPT	20.00	38.00	80.52	90.03	20.38	37.44	68.40	85.71	56.21	69.25	61.15	74.21
	ORLM	25.00	38.00	72.24	90.03	20.38	31.75	79.65	**96.10**	54.04	69.57	59.17	75.07
	SAGE	23.00	37.00	**86.04**	90.95	24.64	42.18	76.62	92.21	55.28	**72.05**	65.37	76.78
	TAGO	**32.00**	**42.00**	84.97	**92.94**	**28.44**	**48.34**	**80.09**	94.81	**60.87**	70.19	**67.75**	**78.83**
Qwen3-8B	Zero-Shot	14.00	28.00	34.51	66.72	11.85	38.86	29.44	64.50	47.20	51.55	31.93	56.73
	CoT	19.00	37.00	77.76	88.96	17.06	41.71	63.20	89.61	56.83	65.53	58.77	74.08
	OR-PRM	24.00	41.00	81.75	92.48	20.38	42.65	76.62	90.04	68.32	76.40	65.77	78.36
	LLMOPT	25.00	45.00	87.58	93.10	26.54	38.86	76.19	90.48	69.57	74.53	69.40	78.03
	ORLM	28.00	42.00	80.67	97.39	30.33	44.55	83.55	**96.97**	60.87	74.84	66.42	81.53
	SAGE	28.00	38.00	**88.96**	94.79	35.07	58.77	83.55	93.94	61.18	79.50	70.71	82.65
	TAGO	**38.00**	**47.00**	86.50	**97.55**	**38.39**	**62.56**	**88.74**	**96.97**	**72.36**	**84.16**	**73.94**	**86.41**
GPT-4o	Zero-Shot	27.00	52.00	59.36	84.51	35.07	62.56	52.38	77.92	42.55	63.35	49.21	73.81
	Reflexion(Agentic)	33.00	**59.00**	85.58	88.96	38.39	66.35	60.17	80.95	42.86	67.39	62.60	78.03
	OptiMUS(Agentic)	**37.00**	**59.00**	**86.50**	**89.42**	**45.50**	**70.14**	**64.50**	**85.71**	**46.58**	**68.01**	**65.70**	**79.62**

B.3 Ablation Studies

Table 8 reports an ablation study that isolates the effects of evidence-aligned SFT, evidence-aligned RL, the structured template, each evidence reward term, and the KER weighting. In the binary-outcome variant, a reward of 1 is assigned only to outputs that are executable and feasible and that achieve the correct objective; all other reward terms remain unchanged. KER uses $R{:}P$ ratios of 2:1, 1:1, and 1:2.

Table 8: Detailed ablation results (%) on Qwen2.5-7B and Qwen3-8B. Within each backbone and column, bold marks the best result and underline marks the second best, including ties.

Backbone	Stage	Variant	IndOR	EasyLP	CpxLP	NL4OPT	NLP4LP	Ave
Qwen2.5-7B	SFT+RL	**Full TAGO (2:1)**	**29.00**	**86.66**	**23.22**	**78.35**	**59.32**	**66.95**
	SFT	Evidence-aligned SFT	18.00	75.46	7.58	75.76	53.73	57.65
		Plain SFT (w/o evidence)	12.00	71.17	6.16	63.64	48.14	52.18
	RL	w/o format	<u>28.00</u>	<u>85.89</u>	22.27	<u>77.06</u>	**59.32**	<u>66.23</u>
		Binary outcome reward (0,1)	26.00	85.58	21.80	76.62	57.45	65.43
		w/o R_{KER}	22.00	85.28	21.33	76.19	<u>58.70</u>	65.17
		w/o R_{BVR}	25.00	83.90	19.91	75.76	57.76	64.32
	Weight	Balanced KER-F_β (1:1)	26.00	<u>85.89</u>	<u>22.75</u>	**78.35**	<u>58.70</u>	<u>66.23</u>
		P-heavy KER-F_β (1:2)	24.00	84.36	21.80	76.19	57.14	64.64
Qwen3-8B	SFT+RL	**Full TAGO (2:1)**	**38.00**	**86.50**	**38.39**	**88.74**	**72.36**	**73.94**
	SFT	Evidence-aligned SFT	27.00	77.15	23.70	75.76	56.21	61.74
		Plain SFT (w/o evidence)	22.00	72.24	20.85	71.43	50.31	56.99
	RL	w/o format	**38.00**	<u>86.20</u>	<u>34.12</u>	86.15	71.43	<u>72.63</u>
		Binary outcome reward (0,1)	35.00	86.04	<u>34.12</u>	84.42	69.57	71.70
		w/o R_{KER}	33.00	84.82	33.18	83.55	67.70	70.38
		w/o R_{BVR}	<u>36.00</u>	81.75	31.28	82.68	66.46	68.60
	Weight	Balanced KER-F_β (1:1)	**38.00**	85.74	33.65	<u>86.58</u>	<u>71.74</u>	72.50
		P-heavy KER-F_β (1:2)	34.00	79.45	31.28	80.52	64.91	66.82

B.4 Additional Analyses

This section provides the evaluation protocols and additional results for the analyses in Sections 4.4.1, 4.4.2, and 4.4.4, presented in the same order as in the main text.

B.4.1 Evidence-Chain Quality under a Unified Evaluation Protocol

Table 3 reports accuracy and evidence recovery on the same benchmark-stratified sample of 300 instances for all methods. All methods use the unified post-hoc evidence extractor detailed in Figure 7 in Appendix A.3; no method's native evidence fields or method-specific extraction protocol is used. Evidence metrics are computed only over correct outputs, and the extraction results are evaluated by an independent judge with an additional manual spot check.

Metric definition. P and R use the one-to-one evidence-span matching in Section 3.2.2. $F_{1,\text{KER}}$ uses the standard balanced F_1 formula (van Rijsbergen, 1979):

$$F_{1,\text{KER}} = 2 \times \frac{P \times R}{P + R}. \tag{10}$$

B.4.2 Error Breakdown

Table 4 compares four non-exclusive error indicators on the same 135 instances failed by all three methods. Rates use this shared set as the denominator.

To make the four non-exclusive error indicators explicit, we use separate LLM-as-Judge prompts on the same 135 shared failed instances. A case may receive multiple `FAIL` labels when the execution trace supports more than one error type.

MAPPING-MISALIGNMENT JUDGE PROMPT

SYSTEM PROMPT

You are an operations-research modeling auditor. Judge only whether each generated mathematical expression binds source information to the correct

entity, index, parameter, unit, and relationship. Do not repair the model and do not judge code execution.

For every expression and its cited evidence, check:

1. Entity binding: products, locations, periods, resources, and indices refer to the same entities in the source and expression.

2. Value binding: every number, coefficient, bound, and unit is attached to the correct source item.

3. Relation binding: operator direction, equality, min/max, and logical relations match the source statement.

4. Scope binding: quantifiers and cross-entity relationships use the correct index scope.

Use FAIL when a valid source fact is attached to the wrong entity, value, index, or relationship. Do not label a missing requirement as misalignment

unless an incorrect binding is also present.

Return JSON only: {"label":"PASS|FAIL","reason":"short reason"}

Return PASS only when every checked binding is faithful.

Source problem:

{question}

Generated model:

{model}

Evidence-expression pairs:

{evidence_pairs}

Figure 9: LLM-as-Judge prompt for mapping-misalignment errors.

INFORMATION-OMISSION JUDGE PROMPT

SYSTEM PROMPT

You are an operations-research completeness auditor. Judge whether the generated formulation represents every source requirement needed for the stated

objective and feasible solution. Do not infer unstated requirements and do not repair the model.

Check completeness of:

1. Required decisions, variables, domains, and bounds.

2. Every objective term, coefficient, and optimization direction.

3. Every source constraint, condition, table value, and logical requirement.

4. Coupling, balance, capacity, demand, and boundary conditions stated in the source.

Label FAIL when a required source item is absent from the mathematical model or code-facing formulation. If the item is present but attached to the

wrong entity, report that as misalignment rather than omission.

Return JSON only: {"label":"PASS|FAIL","missing_items":["..."],"reason":"short reason"}

Return PASS only when no key source requirement is omitted.

Source problem:

{question}

Generated model:

{model}

Evidence chain, if available:

{evidence}

Figure 10: LLM-as-Judge prompt for information-omission errors.

CODE-ERROR JUDGE PROMPT

SYSTEM PROMPT

You are an operations-research code auditor. Determine whether the submitted solver program contains a coding or implementation error, independently

of whether the mathematical model is faithful. Do not repair the code.

Check:

1. Python syntax, imports, names, and data types.

2. Pyomo/API usage, model construction, parameter keys, and variable declarations.

3. Serialization, indexing, constraint construction, objective construction, and missing implementation blocks.

4. Whether the program reaches solver invocation without a code-level exception.

Use FAIL for syntax errors, undefined names, illegal API calls, malformed declarations, or missing/incorrect implementation of a stated model

component. Do not use FAIL only because the solver reports infeasibility or because the source requirement is omitted or misaligned.

Return JSON only: {"label":"PASS|FAIL","category":"syntax|api|declaration|implementation|none","reason":"short reason"}

Source mathematical model:

{model}

Submitted code:

{code}

Execution trace:

{execution_log}

Figure 11: LLM-as-Judge prompt for code errors.

SOLVER-ERROR JUDGE PROMPT

SYSTEM PROMPT

You are an operations-research solver-execution auditor. Determine whether the failed result contains a solver or runtime error after the code has been

accepted for execution. Use the execution trace and solver status; do not repair the model or code.

Use FAIL for:

1. Solver exceptions, unavailable solver backends, or runtime termination during solve.

2. Infeasible, unbounded, undefined, numerical-failure, or timeout status when a valid solution was required.

3. Missing solution values or an invalid solver status that prevents objective evaluation.

4. A solver-returned objective/status failure caused by runtime execution rather than Python syntax or API construction.

Do not use FAIL only for a code parse/API error, a missing source requirement, or a source-to-model binding error. Those are recorded by the other

judges; categories may still co-occur when the trace supports both.

Return JSON only: {"label":"PASS|FAIL","category":"exception|infeasible|unbounded|timeout|numerical|status|none","reason":"short reason"}

Submitted code:

{code}

Solver execution trace:

{execution_log}

Reference objective/status, if available:

{reference}

Figure 12: LLM-as-Judge prompt for solver and runtime errors.

B.4.3 PASS@K SAMPLING

Table 9 reports Pass@K for $K \in \{1, 2, 4, 8, 16\}$ on all five benchmarks.

Figure 13 shows the five benchmark-specific Pass@K curves that complement the weighted Overall curve in the main text.

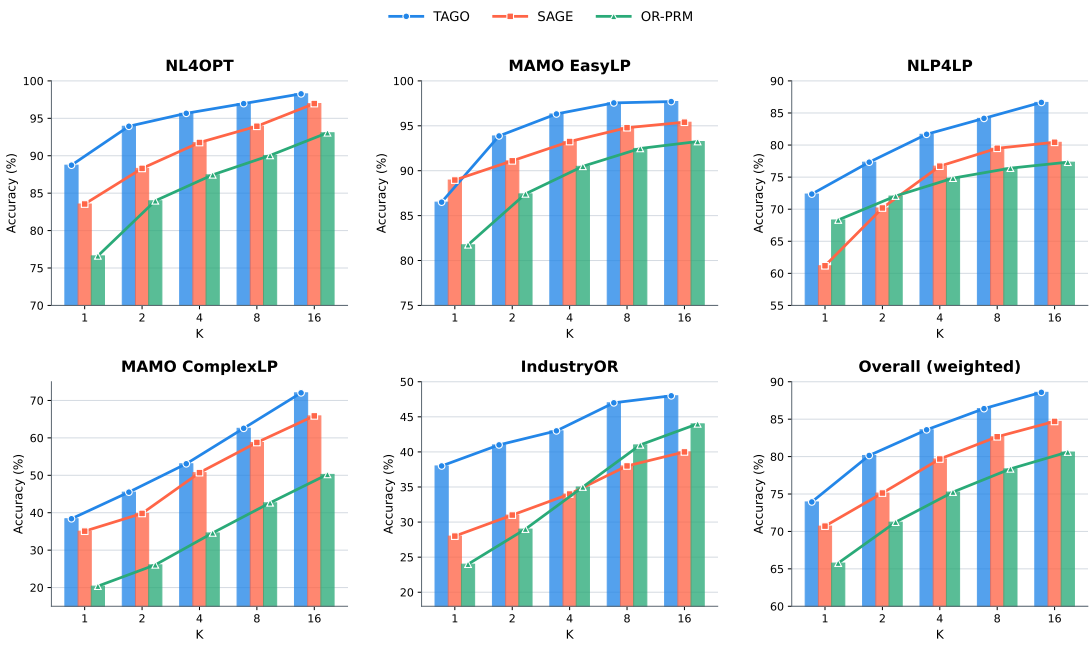


Figure 13: Benchmark-specific Pass@K accuracy for TAGO, SAGE, and OR-PRM. The weighted Overall panel is reported in the main text.

Table 9: Pass@K sampling results (%) across five benchmarks. Bold marks the best result and underline marks the second best for each dataset and sampling budget.

Dataset	K	TAGO	SAGE	OR-PRM
NL4OPT	1	**88.74**	83.55	76.62
	2	**93.94**	88.31	83.98
	4	**95.67**	91.77	87.45
	8	**96.97**	93.94	90.04
	16	**98.27**	96.97	93.07
MAMO_EasyLP	1	86.50	**88.96**	81.75
	2	**93.87**	91.10	87.42
	4	**96.32**	93.25	90.49
	8	**97.55**	94.79	92.48
	16	**97.70**	95.40	93.25
NLP4LP	1	**72.36**	61.18	68.32
	2	**77.33**	70.19	72.05
	4	**81.68**	76.71	74.84
	8	**84.16**	79.50	76.40
	16	**86.65**	80.43	77.33
MAMO_ComplexLP	1	**38.39**	35.07	20.38
	2	**45.50**	39.81	26.07
	4	**53.08**	50.71	34.60
	8	**62.56**	58.77	42.65
	16	**72.04**	65.88	50.24
IndustryOR	1	**38.00**	28.00	24.00
	2	**41.00**	31.00	29.00
	4	**43.00**	34.00	35.00
	8	**47.00**	38.00	41.00
	16	**48.00**	40.00	44.00
Overall	1	**73.94**	70.71	65.77
	2	**80.15**	75.13	71.24
	4	**83.58**	79.68	75.26
	8	**86.41**	82.65	78.36
	16	**88.59**	84.70	80.61

TAGO improves monotonically with the sampling budget on every benchmark and leads the weighted Overall row at each K. The curve shapes are informative beyond the ranking. EasyLP and NL4OPT approach saturation by $K = 8$, whereas MAMO_ComplexLP continues to gain substantially through $K = 16$, indicating that difficult problems still benefit from more search budget.

C Case Study

C.1 Information Omission Case Study

Figure 14 shows an executable transportation model that omits the transit-time requirement. The missing evidence–constraint link makes the omission explicit.

Information Omission Case

Problem Description

A logistics company needs to transport goods from three warehouses, W1, W2, and W3, to four destinations, D1, D2, D3, and D4. The inventories of W1, W2, and W3 are 500, 400, and 300 units, respectively. The demands of D1, D2, D3, and D4 are 200, 300, 400, and 300 units, respectively. The company can choose between water and air transportation. Water transportation is cheaper but slower, while air transportation is more expensive but faster. The transportation cost and transit time depend on both the warehouse-destination pair and the selected transportation mode……

Modeling

# SETS
Warehouses = {W1, W2, W3}
Destinations = {D1, D2, D3, D4}

# PARAMETERS
Supply(Warehouses) = (500, 400, 300)
Demand(Destinations) = (200, 300, 400, 300)
WaterCost(Warehouses, Destinations) =
W1: (1, 2, 3, 2)
W2: (2, 1, 2, 3)
W3: (3, 2, 1, 2)
AirCost(Warehouses, Destinations) =
W1: (15, 16, 17, 16)
W2: (16, 15, 16, 17)
W3: (17, 16, 15, 16)

VARIABLES
WaterShip(Warehouses, Destinations) ≥ 0
AirShip(Warehouses, Destinations) ≥ 0

OBJECTIVE
minimize TotalCost =
sum(WaterCost[i,j] * WaterShip[i,j] + AirCost[i,j] * AirShip[i,j]
for i in Warehouses for j in Destinations)

CONSTRAINTS
Supply Limit:
sum(WaterShip[i,j] + AirShip[i,j] for j in Destinations) ≤ Supply[i]
Demand Satisfaction:
sum(WaterShip[i,j] + AirShip[i,j] for i in Warehouses) = Demand[j]
(Lack: Time Constraint)

Coding

```
from pyomo.environ import *
m = ConcreteModel()
m.W = Set(initialize=['W1','W2','W3'])
m.D = Set(initialize=['D1','D2','D3','D4'])
m.s = Param(m.W, initialize={'W1':500,'W2':400,'W3':300})
m.d = Param(m.D, initialize={'D1':200,'D2':300,'D3':400,'D4':300})
m.cw = Param(m.W, m.D, initialize=water_cost_dict)
m.ca = Param(m.W, m.D, initialize=air_cost_dict)
m.xw = Var(m.W, m.D, domain=NonNegativeReals)
m.xa = Var(m.W, m.D, domain=NonNegativeReals)
m.obj = Objective(expr=sum(m.cw[i,j]*m.xw[i,j] + m.ca[i,j]*m.xa[i,j]
                           for i in m.W for j in m.D), sense=minimize)
m.sup = Constraint(m.W, rule=lambda m,i: sum(m.xw[i,j]+m.xa[i,j]
                                             for j in m.D) <= m.s[i])
m.dem = Constraint(m.D, rule=lambda m,j: sum(m.xw[i,j]+m.xa[i,j]
                                             for i in m.W) == m.d[j])
SolverFactory('gurobi').solve(m)
```

Figure 14: Information omission case study.

C.2 Mapping Misalignment Case Study

Figure 15 shows a valid-looking program that swaps the corn and soybean yield rates. The error propagates into the production constraint, while the evidence chain reveals that the recognized values were bound to the wrong entities.

Mapping Misalignment

Problem Description

A farm chooses nonnegative integer acres of corn and soybeans. Corn yields 1 ton per acre, costs $7 per acre, and uses 2 resource units per acre. Soybeans yield 2 tons per acre, cost $5 per acre, and use 1 resource unit per acre. The farm must produce at least 200 tons using at most 400 resource units and minimizes planting cost.

Modeling

# SETS
Crops = {Corn, Soybean}

PARAMETERS
YieldRate(Corn) = 2 → **(Output relationship mapping error)**
YieldRate(Soybean) = 1
Cost(Corn) = 7
Cost(Soybean) = 5
MinYield = 200
ResourceLimit = 400
ResourceCoef(Corn) = 2
ResourceCoef(Soybean) = 1

VARIABLES
CornArea ≥ 0, integer
SoybeanArea ≥ 0, integer

OBJECTIVE
minimize TotalCost = 7 * CornArea + 5 * SoybeanArea

CONSTRAINTS
Yield Requirement:
2 * CornArea + 1 * SoybeanArea ≥ 200 (Incorrect propagated yield constraint)
Resource Limit:
2 * CornArea + 1 * SoybeanArea ≤ 400
Integer:
CornArea, SoybeanArea ∈ $\mathbb{Z}^+$

Coding

```
from pyomo.environ import *
model = ConcreteModel()
# Sets
model.Crops = Set(initialize=['Corn', 'Soybean'])
# Parameters
…
model.YieldRate = Param(model.Crops, initialize=yield_rate)
model.Cost = Param(model.Crops, initialize=cost)
model.ResourceCoef = Param(model.Crops, initialize=resource_coef)
# Variables
model.Area = Var(model.Crops, domain=NonNegativeIntegers)
# Objective
def objective_rule(model):
    return sum(model.Cost[c] * model.Area[c] for c in model.Crops)
model.TotalCost = Objective(rule=objective_rule, sense=minimize)
# Constraints
def yield_constraint_rule(model):
    return sum(model.YieldRate[c] * model.Area[c] for c in model.Crops) ≥
200
model.YieldConstraint = Constraint(rule=yield_constraint_rule)
…
# Solver call
# solver = SolverFactory('gurobi')
# results = solver.solve(model)
```

Figure 15: Mapping misalignment case study.

C.3 Evidence-Chain Data Case Study

Figure 16 shows a training sample in which sets, parameters, variables, the objective, and constraints are linked to source spans before the formulation is compiled into Pyomo code.

Evidence-chain Data Case Study

Problem Description

A transportation company has two vehicle types, X and Y. Operating costs are $5 for X and $10 for Y. It must operate at least 30 X vehicles and at least 20 Y vehicles, with at most 100 vehicles in total. The number of vehicles must be whole numbers. What is the minimum daily operational cost?

<model>

SETS
V: vehicle types, {X, Y} (evidence: "two vehicle types, X and Y")

PARAMETER
cost[v]: operating cost per vehicle type (evidence: "Operating costs are $5 for X and $10 for Y.")
min_op[v]: minimum required vehicles per type (evidence: "It must operate at least 30 X vehicles and at least 20 Y vehicles")
max_total: maximum total vehicles (evidence: "with at most 100 vehicles in total.")

VARIABLES
$x[v] \in \mathbb{Z}_+$: number of vehicles of type v (evidence: "must be whole numbers")

OBJECTIVE
minimize sum cost[v] * x[v] (evidence: "minimum daily operational cost")

CONSTRAINTS
$x[v] \geq$ min_op[v] (evidence: "It must operate at least 30 X vehicles and at least 20 Y vehicles")
$\sum x[v] \leq$ max_total (evidence: "with at most 100 vehicles in total.")

</model>

<code>

```
import pyomo.environ as pyo
# --- Model Initialization ---
model = pyo.ConcreteModel()
# --- SETS ---
model.V = pyo.Set(initialize=['X', 'Y'])
# --- PARAMETERS ---
model.cost = pyo.Param(model.V, initialize={'X': 5, 'Y': 10})
model.min_op = pyo.Param(model.V, initialize={'X': 30, 'Y': 20})
model.max_total = pyo.Param(initialize=100)
# --- VARIABLES ---
model.x = pyo.Var(model.V, domain=pyo.NonNegativeIntegers)
# --- OBJECTIVE ---
def objective_rule(model):
    return sum(model.cost[v] * model.x[v] for v in model.V)
model.total_cost = pyo.Objective(rule=objective_rule, sense=pyo.minimize)
# --- CONSTRAINTS ---
def min_op_rule(model, v):
    return model.x[v] ≥ model.min_op[v]
model.min_op_constraint = pyo.Constraint(model.V, rule=min_op_rule)
def total_vehicles_rule(model):
    return sum(model.x[v] for v in model.V) ≤ model.max_total
model.total_vehicles_constraint = pyo.Constraint(rule=total_vehicles_rule)
# --- Solver Call ---
# solver = pyo.SolverFactory('glpk')   # or 'gurobi', 'cbc'
# results = solver.solve(model)
# --- Display Results ---
# for v in model.V:
#     print(f"x[{v}] = {pyo.value(model.x[v])}")
# print(f"Total cost = {pyo.value(model.total_cost)}")
```

</code>

Figure 16: Evidence-chain data case study.