Strategy-Aware Optimization Modeling with Reasoning LLMs

Ruiqing Zhao [1 *] Fengzhi Li [2 *] Yuan Zuo [3 †] Rui Liu [1 †] Yansong Liu [1] Yunfei Ma [2] Fanyu Meng [2]
Junlan Feng [2]

Abstract

Large language models (LLMs) can generate syntactically valid optimization programs, yet often struggle to reliably choose an effective modeling strategy, leading to incorrect formulations and inefficient solver behavior. We propose SAGE, a strategy-aware framework that makes *Modeling Strategy* explicit in both data construction and post-training. SAGE builds a solver-verified multi-strategy dataset and trains a student model with supervised fine-tuning followed by Segment-Weighted GRPO using a composite reward over format compliance, correctness, and solver efficiency. Across eight benchmarks spanning synthetic and real-world settings, SAGE improves average pass@1 from 72.7 to 80.3 over the strongest open-source baseline. With multiple generations, SAGE discovers more distinct correct formulations and improves component-level diversity at pass@16 by 19–29%. At the largest scale, SAGE produces more compact constraint systems with 14.2% fewer constraints than the baseline, consistent with solver-efficient modeling. Overall, these results show that making *Modeling Strategy* explicit improves automated optimization modeling. The code and data are available at https://github.com/rachhhhing/SAGE.

1. Introduction

In operations research (OR), optimization models translate informal decision-making requirements into precise mathematical programs that can be solved by modern optimization software. In many practical workflows, however, the bottleneck is not running a solver, but formulating a solver-executable model from a natural-language description. This step requires careful design of index sets, decision variables, and constraints, and ideally produces formulations that are not only correct but also efficient to solve (Antoniou & Lu, 2007). The gap between problem intent and solver-ready formulations remains a major source of cost across scheduling, manufacturing, logistics, and finance (Cohen & Sherkat, 1987; Jayal et al., 2010; Bartolacci et al., 2012; Ponsich et al., 2012).

Large language models (LLMs) have recently demonstrated strong capabilities in language understanding, reasoning, and code generation (Brown et al., 2020), spurring rapid progress on automated optimization modeling. Existing systems generally follow two lines: prompt- and agent-based approaches decompose a problem and synthesize solver code (Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Zhang & Luo, 2025), while learning-based methods fine-tune models on optimization data to improve formulation accuracy and executability (Huang et al., 2025; Jiang et al., 2025; Chen et al., 2025; Zhou et al., 2026). Despite steady improvements, a recurring limitation is that key formulation decisions remain implicit: models are typically trained to generate mathematical formulations and solver code, but without an explicit global design that guides how a problem should be modeled.

In practice, optimization modeling is *strategy-driven*. Before writing down variables and constraints, human experts typically commit to a high-level formulation paradigm such as flow-based or assignment-based modeling, which determines the decision domain and the variable backbone that structures the entire model (Williams, 2013; Hillier, 2005). We call this paradigm-level choice the *Modeling Strategy*. It is not merely stylistic. It enforces global consistency among variables, constraints, and objectives, and it strongly shapes solver behavior in practice, since solution effort can depend critically on the chosen formulation (LINDO Systems, Inc.).

Figure 1 shows a common failure mode through a simple transportation scenario. In this problem, goods can only be shipped along a predefined set of transportation links between cities. However, a step-wise modeling pipeline may blindly introduce shipment variables for all city pairs, implicitly assuming that goods can be sent between any two

[*]Equal contribution [1]School of Computer Science and Engineering, Beihang University, Beijing, China [2]JIUTIAN Research, Beijing, China [3]MIIT Key Laboratory of Data and Decision Intelligence, Beihang University, Beijing, China. Correspondence to: Yuan Zuo <zuoyuan@buaa.edu.cn>, Rui Liu <lr@buaa.edu.cn>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

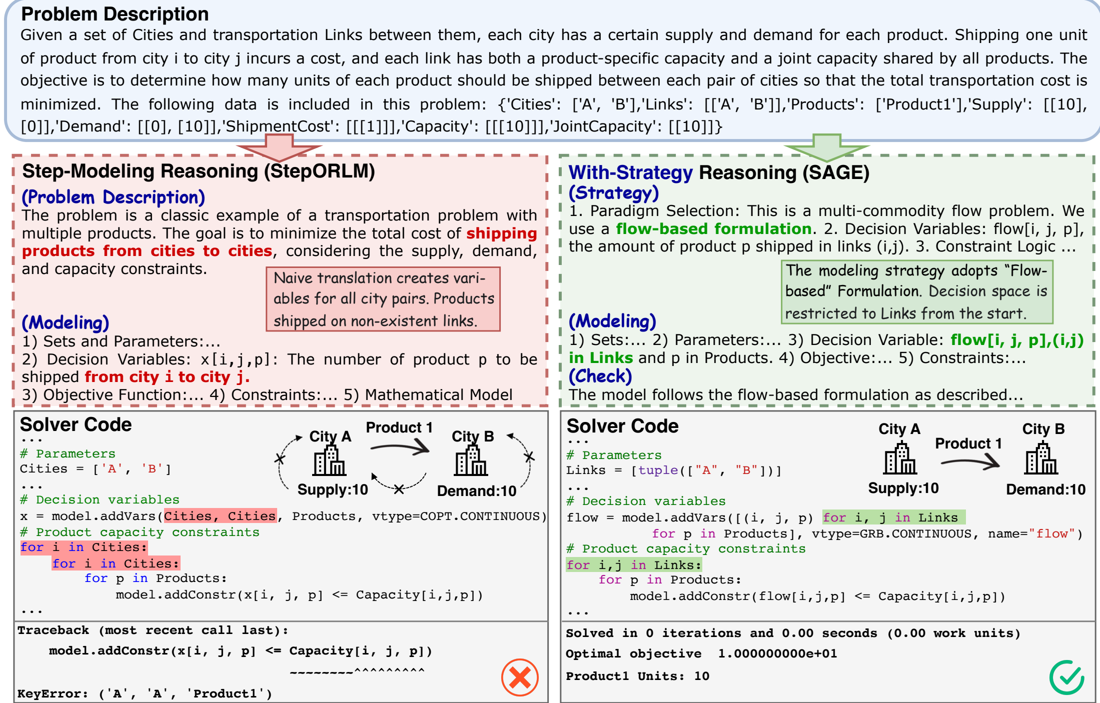

Figure 1. **Why modeling strategy matters.** A step-wise pipeline may define variables on an incorrect index space (e.g., (A, A)), creating invalid arcs and runtime failures (e.g., `KeyError`). Strategy-aware reasoning first commits to a paradigm (e.g., flow-based) and restricts the decision domain (e.g., `Links`), producing a consistent and solver-executable model.

cities. This creates decision variables for non-existent routes (e.g., (A, A)) and leads to runtime failures. In contrast, a strategy-aware approach commits early to a flow-based formulation and restricts variables to (i, j) ∈`Links`, yielding a globally consistent, solver-executable model. This highlights the first role of modeling strategy: ensuring *executability and correctness* via coherent decision domains.

Strategy also matters for *efficiency*. Many OR problems admit multiple correct formulations under different paradigms, yet they can differ substantially in model size, constraint tightness, and solver performance (Calafiore & El Ghaoui, 2014). Thus, an automation system should learn not only to generate correct and executable formulations, but also to choose strategies that lead to efficient solves (e.g., reduced solve time). This remains challenging because (i) strategy-level reasoning is rarely explicit in training data, making paradigm selection hard to learn; and (ii) prevailing objectives emphasize executability and correctness but provide limited signal for efficiency-aware strategy learning.

To address these challenges, we study *Strategy-Aware Optimization Modeling with Reasoning LLMs* and present SAGE (**S**trategy-**A**ware **G**uided r**E**asoning), a generation-based framework for optimization modeling with reasoning LLMs (Figure 2). SAGE makes modeling strategy explicit in both data construction and post-training. In Phase 1, a teacher model generates multiple candidate modeling strategies for each OR problem and then produces a reasoning trajectory together with the corresponding Gurobi code conditioned on each strategy. We execute the code and validate solutions against the ground-truth answer to filter incorrect outputs, and deduplicate semantically redundant strategies via an LLM-as-Judge to retain distinct ones. In Phase 2, we optimize the model with solver feedback via *Segment-Weighted GRPO*, which improves credit assignment by emphasizing strategy decisions, and uses a composite reward that jointly captures (i) structured format compliance, (ii) executability and correctness, and (iii) solver efficiency. Our contributions are:

- We construct a solver-verified, multi-strategy dataset of (question, reasoning, code) by generating multiple strategies per problem, filtering outputs via solver validation against ground-truth, and deduplicating redundant strategies.
- We propose *Segment-Weighted GRPO* that upweights strategy segments and optimizes a composite reward over format compliance, executability and correctness, and solver efficiency.
- Across eight benchmarks, SAGE improves average pass@1 from 72.7 to 80.3 over the strongest open-source baseline, while discovering more distinct correct formulations under pass@K and producing more solver-efficient models.

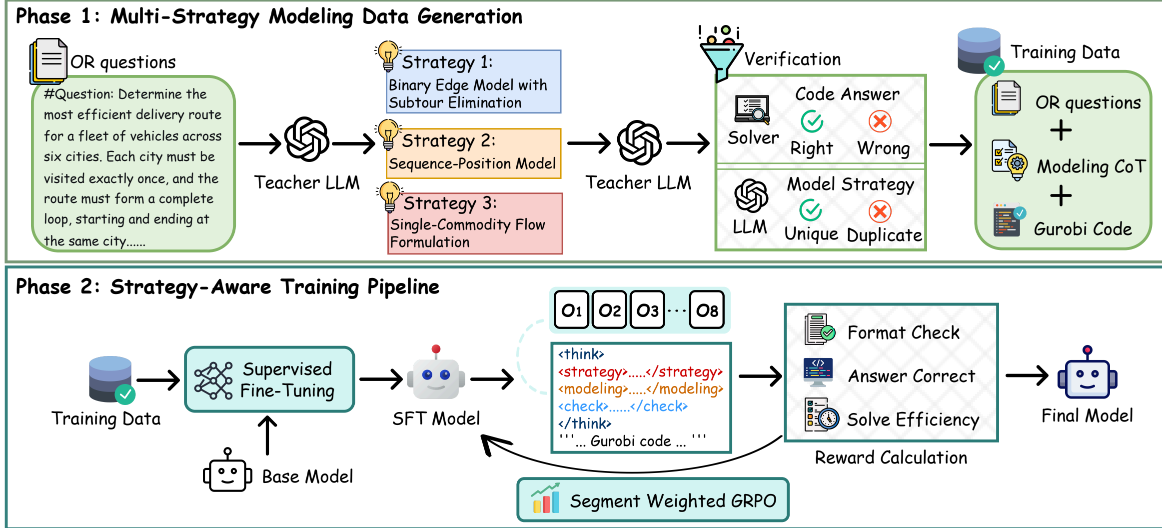

Figure 2. **Overview of SAGE.** Phase 1 builds a multi-strategy, solver-verified corpus by generating multiple candidate strategies per problem, producing strategy-conditioned reasoning and Gurobi code, filtering via solver validation against ground-truth, and deduplicating redundant strategies with an LLM-as-Judge. Phase 2 trains with supervised fine-tuning and Segment-Weighted GRPO using format, correctness, and efficiency rewards.

2. Methodology

We propose SAGE, a two-phase framework that makes *modeling strategy* explicit throughout. In Phase 1, we perform solver-verified multi-strategy data synthesis to produce *(question, modeling reasoning, code)* triples. In Phase 2, we post-train the model using supervised fine-tuning (SFT) followed by reinforcement learning (RL) with solver feedback to optimize structured reasoning, executability and correctness, and solver efficiency.

2.1. Multi-Strategy Modeling Data Synthesis

Existing optimization-modeling datasets often provide only *(question, model, code)* sequences, which expose final formulations but omit the strategy-level reasoning process. This makes it difficult for models to learn how to *choose* a formulation paradigm and consistently instantiate it. We therefore construct a dataset of *(question, modeling reasoning, code)* triples, where the *modeling reasoning* explicitly separates (i) paradigm selection from (ii) formulation instantiation.

Given a set of optimization problems $\mathcal{Q} = \{q_i\}_{i=1}^N$ with ground-truth solver answers, a strategy teacher first proposes K candidate modeling strategies for each problem:

$$\mathcal{S}(q_i) = \{s_i^{(k)}\}_{k=1}^K, \tag{1}$$

where each $s_i^{(k)}$ corresponds to a distinct formulation paradigm (e.g., assignment-based vs. flow-based vs. time-indexed).

Then, a reasoning teacher model takes $(q_i, s_i^{(k)})$ as input and produces a complete reasoning trace and a solver-executable program:

$$(r_i^{(k)}, c_i^{(k)}) \sim \pi_{\text{teach}}(r, c \mid q_i, s_i^{(k)}), \tag{2}$$

where $r_i^{(k)}$ describes the strategy-aligned modeling steps (sets/parameters/variables/constraints/objective) and $c_i^{(k)}$ is the corresponding Gurobi Python code.

Solver verification and filtering. To ensure data quality, we execute each generated program and validate it with the solver. We discard outputs that (i) fail to run, (ii) are infeasible or unbounded when a feasible optimum is expected, or (iii) return a solution that does not match the ground-truth answer within tolerance.

Strategy de-duplication. Multiple strategies may yield semantically equivalent formulations. We therefore apply an LLM-as-Judge procedure (Zheng et al., 2023) that jointly compares reasoning traces and code to remove semantically redundant strategies, retaining only distinct and meaningful paradigms.

The resulting supervised corpus is

$$\mathcal{D}_{\text{SFT}} = \bigcup_{q_i \in \mathcal{Q}} \{(q_i, r_i^{(k)}, c_i^{(k)}) \mid k \in \mathcal{K}_i\}, \tag{3}$$

where $\mathcal{K}_i$ indexes the remaining verified and de-duplicated strategies for problem q_i (possibly multiple per problem). Concrete prompt templates and examples are provided in Appendix C.2.

2.2. Supervised Fine-Tuning

After constructing $\mathcal{D}_{\text{SFT}}$, we perform full-parameter supervised fine-tuning to *distill* the teacher's strategy-conditioned modeling reasoning and code generation behavior into the student model. Given a problem description q, the student model is trained to generate a reasoning trace r followed by solver code c.

We concatenate the reasoning trace and solver code into a single target sequence $y = [r, c]$ and optimize the standard language-modeling objective:

$$\mathcal{L}_{\text{SFT}} = -\sum_{i=1}^{|\mathcal{D}_{\text{SFT}}|} \sum_{t=1}^{|y_i|} \log P_\theta\big(y_{i,t} \mid q_i, y_{i,1:t-1}\big), \quad (4)$$

where $y_{i,t}$ is the t-th token of the target sequence for sample i, and θ denotes model parameters.

2.3. Reinforcement Learning with Solver Feedback

Structured reasoning template. Unstructured "one-block" reasoning is often brittle for complex multi-stage tasks (Wei et al., 2022; Yao et al., 2023). In optimization modeling, early paradigm decisions constrain all modeling steps, making explicit structure particularly important. We therefore enforce a template in which the model emits a `<think>` block with three ordered segments:

- `<strategy>`: identify the problem class, choose a modeling paradigm, and commit to the core decision variables and constraint logic;
- `<modeling>`: instantiate the formulation (sets, parameters, variables, objective, constraints) consistent with the chosen strategy;
- `<check>`: verify global consistency (index domains, loop dimensions, logical coupling) and remove obvious redundancies.

The solver-executable code is generated after the `<think>` block. This separation strengthens the linkage between high-level paradigm selection and concrete implementation, reducing index-space mismatches and other common errors.

Segment-weighted GRPO. For RL, we adopt Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a critic-free variant of Proximal Policy Optimization (PPO) (Schulman et al., 2017), which optimizes a PPO-style clipped objective using group-relative advantage estimates over multiple sampled completions for the same prompt. GRPO can be instantiated with either outcome rewards or process rewards. In optimization modeling, solver execution naturally provides outcome-level feedback, since executability, correctness, and solver efficiency can only be verified after the full reasoning trace and solver code are generated. However, outcome-reward GRPO applies uniform token-level credit across the entire trajectory, implicitly assuming that all generation steps contribute equally to the final reward. This assumption is known to exacerbate the credit assignment problem in long-horizon reasoning tasks, where early high-level decisions may dominate downstream correctness, while later surface-level tokens have marginal impact (Lightman et al., 2024; Guo et al., 2025b). In optimization modeling, different reasoning stages play unequal roles in determining the correctness and efficiency of the final formulation. We therefore introduce *segment-weighted* GRPO, which assigns token weights according to the reasoning segment that generated them.

For a prompt, we sample a group of G trajectories. Let T_i be the trajectory length and A_i the group-relative advantage. The segment-weighted GRPO objective is

$$\mathcal{L}_{\text{SW_GRPO}} = -\textstyle\sum_{i=1}^{G}\sum_{t=1}^{T_i} \alpha_t \min(\rho_{i,t}A_i,\ \text{clip}(\rho_{i,t},\, 1-\eta,\, 1+\eta)\, A_i)\,, \quad (5)$$

where $\rho_{i,t} = \frac{\pi_\theta(a_{i,t}|s_{i,t})}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_{i,t})}$ is the likelihood ratio and η is the clipping parameter.

The token weight α_t is chosen by segment membership, with $\alpha_{\text{strategy}} > \alpha_{\text{modeling}} > \alpha_{\text{check}} > 0$. Setting all weights to 1 recovers standard GRPO.

Composite reward: format, outcome, and efficiency. We optimize not only for correctness but also for efficient formulations. For a generated response y, the total reward is

$$R(y) = R_{\text{format}}(y) + R_{\text{outcome}}(y) + R_{\text{efficiency}}(y). \quad (6)$$

Format reward. We reward adherence to the structured template by checking the presence of the tags {`<think>`, `<strategy>`, `<modeling>`, `<check>`} and a final Python code block. Each correctly detected component contributes 0.2, yielding

$$R_{\text{format}}(y) = 0.2 \times N_{\text{seg}}(y), \quad (7)$$

where $N_{\text{seg}}(y) \in \{0, \ldots, 5\}$ is the number of correctly formatted components.

Outcome reward. We execute the generated Gurobi program and assign

$$R_{\text{outcome}}(y) = \begin{cases} 0, & \text{if code execution fails,} \\ 0.2, & \text{if the solver reports infeasible,} \\ 0.4, & \text{if a solution returned is incorrect,} \\ 1.0, & \text{if the solution matches ground truth.} \end{cases} \quad (8)$$

Efficiency reward. The efficiency reward is applied only when $R_{\text{outcome}}(y) = 1.0$. Let $M(y)$ be an efficiency metric computed from solver feedback. For LP instances, we use the solver iteration count:

$$M_{\text{LP}}(y) = \text{IterationCount}(y). \quad (9)$$

Table 1. **The overall performance with Pass@1 accuracy (%).** All reproduced baseline results are obtained on the same cleaned benchmark versions used to evaluate SAGE. Scores reported from original or reproduced papers are marked with (*), while missing entries are denoted by (–). The best results are highlighted in **bold** and the second-best results are underlined.

Types	Models	Easy Tasks				Complex Tasks				Avg.
		NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
Zero-shot	GPT-4o	89.2	77.2	89.9	82.9	61.3	50.0	47.6	21.1	64.9
	DeepSeek-V3	87.8	95.2	87.6	85.1	55.9	55.6	66.7	40.4	71.8
	DeepSeek-R1	86.4	88.1	81.5	77.4	63.9	55.6	57.1	34.9	68.1
	Qwen3-32B	82.7	73.6	88.8	81.6	46.8	33.3	42.9	14.5	58.0
	Qwen2.5-72B	84.0	91.9	85.4	80.4	52.3	44.4	47.6	18.1	63.0
Agent-based	Chain-of-Experts	66.7*	94.4*	87.4*	71.2*	50.6*	57.1*	31.2*	–	–
	OptiMUS	76.2*	78.0*	88.8*	87.6*	46.8*	46.8*	45.2*	20.2*	61.2*
Offline-learning	ORLM-L3-8B	73.8*	90.4*	76.4*	61.8*	59.5*	50.0*	42.9*	2.6*	57.2*
	LLMOpt-Q2.5-14B	80.3*	89.5*	73.4*	53.8*	44.1*	35.3*	29.0*	12.5*	52.2*
	OptM-Q2.5-7B	94.7*	86.5*	–	57.9*	51.2*	–	20.0*	24.4*	–
Online-RL	SIRL-Q2.5-7B	94.8	**98.0**	96.6	89.6	81.1	44.4	50.0	27.1	72.7
	StepORLM-Q3-8B	**96.7**	95.4	97.2	88.6	79.3	50.0	52.4	13.9	71.7
	SAGE-DS-14B (Ours)	94.3	94.7	**98.9**	**93.8**	**84.7**	**61.1**	**69.0**	**45.8**	**80.3**

For MILP instances, we combine the normalized optimality gap and branch-and-bound node count:

$$M_{\text{MILP}}(y) = \beta_{\text{gap}} \cdot \text{Gap}(y) + \beta_{\text{nodes}} \cdot \text{Nodes}(y). \quad (10)$$

The efficiency reward is then defined as

$$R_{\text{efficiency}}(y) = 1 - \tanh\left(\frac{M(y)}{\alpha_{\text{eff}}}\right), \quad (11)$$

where α_{eff} is a scaling constant. The $\tanh(\cdot)$ shaping keeps the reward in $[0, 1]$ while providing smooth reward variation. A smaller $M(y)$ indicates lower solver effort and therefore yields a higher efficiency reward. Detailed normalization and scaling factors are provided in Appendix C.1.

3. Experiments

3.1. Experiment Setup

Benchmarks. We evaluate SAGE on a diverse set of optimization modeling benchmarks, including NL4OPT (Ramamonjison et al., 2022), MAMO (Huang et al., 2024), which is split into EasyLP and ComplexLP subsets, NLP4LP (AhmadiTeshnizi et al., 2024), ComplexOR (Xiao et al., 2024), IndustryOR (Huang et al., 2025), OptiBench (Yang et al., 2025b), and OptMATH (Lu et al., 2025). Additional details for each benchmark are provided in Appendix A.1.

We group these datasets into Easy and Complex categories. The Easy datasets include NL4OPT, MAMO Easy, NLP4LP, and OptiBench. They primarily contain classical linear programming problems with relatively few variables and constraints. The Complex datasets include MAMO Complex, ComplexOR, IndustryOR, and OptMATH. They cover a wider range of problem types and feature implicit or hierarchical constraints, as well as domain specific real world scenarios.

Baselines. We benchmark against a broad and representative set of baselines in four categories.

- **Zero shot generalist LLMs**. Foundation models without OR specific training, including GPT-4o (Hurst et al., 2024), DeepSeek-V3 (Liu et al., 2024), DeepSeek-R1 (Guo et al., 2025a), Qwen3-32B (Yang et al., 2025a), and Qwen2.5-72B-Instruct (Yang et al., 2024). All models are evaluated using the same prompts and decoding settings as our method.

- **Agent based methods**. Multi step orchestration frameworks that coordinate reasoning and code generation through explicit workflows. We include Chain-of-Experts (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024).

- **Offline learning baselines**. Supervised fine tuned models trained on optimization specific datasets, including ORLM-LLaMA-3-8B (Huang et al., 2025), LLMOpt-Qwen2.5-14B (Jiang et al., 2025), and OptMATH-Qwen2.5-7B (Lu et al., 2025).

- **Online RL baselines**. Models trained with reinforcement learning using solver based or process level feedback, including SIRL-Qwen2.5-7B (Chen et al., 2025) and StepORLM-Qwen3-8B (Zhou et al., 2026).

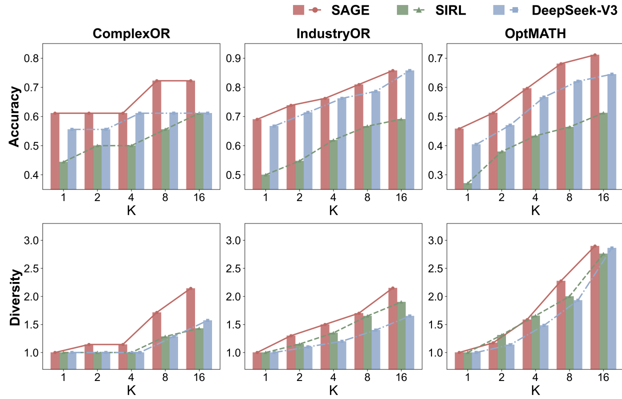

Figure 3. **Pass@K accuracy and modeling diversity.** Our method continues to discover more correct and more diverse formulations as K increases.

Implementation Details. We use DeepSeek-R1 as the teacher model during data construction and initialize our student model from DeepSeek-R1-distill-Qwen-14B. Training problems are drawn from OptMATH-201K and IndustryOR-3K, which correspond to the training splits released by the original benchmarks. This yields 100K samples for supervised fine tuning and an additional 8K samples for reinforcement learning. We also repeat the same pipeline with a smaller Qwen3-8B backbone (initializing the student from Qwen3-8B) to verify the robustness across model scales; the corresponding results are reported in Appendix B.2. We use the VeRL framework (Sheng et al., 2025) with GRPO. The detailed hyperparameter settings for both the supervised fine-tuning (SFT) and reinforcement learning (RL) stages are provided in Appendix A.3.

3.2. Main Results

Table 1 reports Pass@1 performance across eight optimization modeling benchmarks. SAGE achieves the strongest results among open-source models. It also surpasses its teacher model DeepSeek-R1, which is used during distillation, despite using far fewer parameters than the largest generalist LLMs. This result supports the effectiveness of strategy-aware post-training with solver feedback, which enables a compact model to acquire more reliable optimization modeling behavior.

Compared with online RL baselines such as SIRL-Qwen2.5-7B and StepORLM-Qwen3-8B, our model achieves comparable accuracy on Easy benchmarks and delivers substantially stronger performance on Complex benchmarks. Averaged over Complex benchmarks, SAGE improves Pass@1 by approximately **15.4%**. These results indicate that making Modeling Strategy explicit is particularly beneficial for structurally complex problems.

The framework also generalizes across backbone scales.

Table 2. **Component level modeling diversity at pass@16.** We report the average number of distinct decision variable, constraint, and objective per problem across multiple correct generations. Higher values indicate greater modeling diversity.

Model	Var. Types	Constr. Types	Obj. Types
SAGE (Ours)	**2.33**	**2.31**	**2.08**
SIRL	1.80	1.91	1.74
DeepSeek-V3	1.94	2.03	1.81

When applied to a smaller Qwen3-8B backbone, our method still achieves the best performance on the same challenging benchmarks. Detailed results are provided in Appendix B.2.

3.3. Pass@K Performance

We further evaluate Pass@K to measure modeling competence when multiple generations are allowed. This setting tests whether additional sampling enables the model to discover more correct solutions and more distinct valid formulations. We run this evaluation on three challenging benchmarks, *CpxOR*, *IndOR*, and *OptM.*, and compare against two strong baselines, SIRL and DeepSeek-V3. These baselines achieve the highest average accuracy among open-source baselines in Table 1, and all methods share the same Gurobi solver environment.

Accuracy. Across all three benchmarks, our model consistently achieves higher Pass@K accuracy than both baselines under every sampling budget, shown in figure 3. At pass@16, our method reaches 0.72 on ComplexOR, 0.86 on IndustryOR, and 0.71 on OptMATH. As K increases, the performance gap remains stable, which suggests that our approach produces correct formulations more reliably and benefits more effectively from additional sampling.

Diversity. To assess the ability to explore diverse valid formulations, we define a metric for modeling diversity. For each dataset d, we collect the subset of problems solved correctly by all compared methods at any Pass@K, denoted as $\mathcal{P}_d$. For a problem $p \in \mathcal{P}_d$, let $\mathcal{S}_p^K = \{s_{p,1}, \dots, s_{p,K}\}$ be the first K generated solutions, and let $\phi(s_{p,i})$ denote the corresponding modeling representation. We define diversity as

$$\mathrm{Div}_d(K) = \frac{1}{|\mathcal{P}_d|} \sum_{p \in \mathcal{P}_d} |\{\phi(s_{p,i}) \mid s_{p,i} \text{ is correct, } i \le K\}| . \tag{12}$$

This metric reports, for each instance, the number of correct and non-equivalent formulations among the first K generations, averaged over the benchmark. We treat two generated models as identical if they correspond to the same linear program (LP) formulation up to semantics-preserving rewrites (e.g., variable renaming and re-ordering). We use an LLM-based judge to compare the exported LP files and

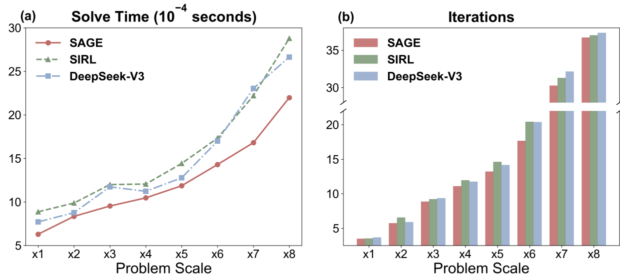

Figure 4. **Efficiency performance under increasing problem scale.** Our method yields lower solve time and fewer solver iterations, with larger gains on larger instances.

Table 3. **Structural complexity of generated formulations.** We report the average number of decision variables (#Vars), constraints (#Constr.), and non zero coefficients (NNZ) at the largest problem scale ($\times 8$). Lower values are better.

Model	#Vars	#Constr.	NNZ
SAGE (Ours)	205.7	**100.6**	**503.9**
SIRL	197.8	117.2	543.3
DeepSeek-V3	**191.8**	116.8	520.2

assess whether two formulations should be considered semantically equivalent for deduplication.

Our method also improves diversity of correct formulations, shown in figure 3. Table 2 shows higher component-level diversity, measured by the number of distinct variable designs, constraint structures, and objective formulations produced for the same problem across multiple correct generations. These results indicate that explicit Modeling Strategy improves correctness while expanding the range of valid formulation structures that the model can produce.

3.4. Efficiency Performance

We examine whether the Modeling Strategy learned by our method improves solver efficiency as problem scale increases, beyond improving formulation correctness. We focus on ComplexOR because its benchmark design separates problem descriptions from numerical data, which enables controlled scaling of problem sizes.

To reduce the influence of correctness on efficiency comparisons, we select seven representative problems that can be solved stably by all compared methods. For each selected problem, we treat the original variable scale as the base and construct larger variants by linearly increasing the number of decision variables. For each scale, we generate five random data instances under the same structural constraints.

We construct the evaluation set by using DeepSeek-V3 to generate solver-executable code and executing it with Gurobi. We retain instances whose solver-returned solutions are self-consistent across multiple sampled generations as reference solutions (Wang et al., 2022). We evaluate solver efficiency using two indicators, the solve time reported by Gurobi to reflect computational cost and the number of solver iterations to reflect convergence complexity. For each scale, we report averages over instances that are solved correctly. More scaling details are provided in Appendix A.2.

Figure 4 shows that, across all scales, our model produces formulations that solve faster and require fewer iterations than formulations produced by the baselines. The gap grows as scale increases, which suggests that the efficiency benefits induced by strategy aware modeling become more pronounced on larger instances.

To understand the source of these efficiency gains, we analyze the structural complexity of generated formulations at the largest scale ($\times 8$), as summarized in Table 3. We report three indicators. *#Vars* is the number of decision variables. *#Constr.* is the number of constraints. *NNZ* is the number of non-zero coefficients in the constraint matrix. Together they reflect the size and sparsity of the resulting LP.

As shown in Table 3, SAGE uses slightly more variables, but consistently yields fewer constraints and lower NNZ than both baselines. This matches a common OR trade-off (extended formulations): introducing auxiliary variables can simplify and sparsify the constraint system, reducing the linear-algebra workload that dominates solver time. Hence, the efficiency gains are better explained by a sparser constraint matrix rather than minimizing *#Vars*.

3.5. Ablation Study

We conduct ablation experiments to examine the contribution of the core components in our framework: (1) w/o RL: training solely with supervised fine-tuning without reinforcement learning, serving as the base policy to assess the impact of the RL stage; (2) RL w/o template: replaces the structured reasoning format with a plain instruction during the RL phase, where the model generates the formulation without explicitly outputting a modeling strategy; (3) RL w/o weighted: sets all token weights to one, effectively degenerating the objective into standard GRPO training; and (4) RL w/o efficiency: removes the efficiency term from the reward, optimizing solely for format and correctness.

Table 4 summarizes the results. The w/o RL variant shows the largest overall degradation, demonstrating that reinforcement learning is essential beyond imitation. Both the structured reasoning template and the weighted loss play crucial roles, particularly on structurally complex benchmarks such as *MAMOComplex*, *IndOR*, and *OptM.*. Removing the structured template results in an average accuracy drop of **4.2%**, while removing the weighted loss leads to a larger decline of **5.8%** across these datasets. This indicates that the structured template provides high-level organization for multi-stage

Table 4. **Ablation study of different training components.** We report pass@1 accuracy (%) on eight optimization benchmarks. Blue and red arrows indicate performance improvements and degradations relative to the full training model, respectively.

Models	NL4OPT	MAMO		NLP4LP	CpxOR	IndOR	OptiB.	OptM.	**Avg.**
		Easy	Complex						
Full Training	94.3	94.7	84.7	98.9	61.1	69.0	93.8	45.8	80.3
w/o RL	92.0 ↓2.3	89.5 ↓5.2	69.4 ↓15.3	94.9 ↓4.0	55.6 ↓5.5	57.1 ↓11.9	91.6 ↓2.2	42.2 ↓3.6	74.0 ↓6.3
RL w/o template	93.0 ↓1.3	94.1 ↓0.6	77.5 ↓7.2	97.8 ↓1.1	61.1 ↓0.0	66.7 ↓2.3	93.1 ↓0.7	42.8 ↓3.0	78.3 ↓2.0
RL w/o weighted	92.5 ↓1.8	94.7 ↓0.0	73.9 ↓10.8	96.6 ↓2.3	55.6 ↓5.5	64.3 ↓4.7	91.8 ↓2.0	44.0 ↓1.8	76.7 ↓3.6
RL w/o eff-reward	92.0 ↓2.3	95.0 ↑0.3	72.1 ↓12.6	96.1 ↓2.8	61.1 ↓0.0	66.7 ↓2.3	91.6 ↓2.2	44.6 ↓1.2	77.4 ↓2.9

Table 5. **Ablation results on IndustryOR grouped by problem difficulty.** We report execution rate (ER, %) and accuracy (AC, %) for Easy, Medium, and Hard subsets.

Models	Easy		Medium		Hard	
	ER	AC	ER	AC	ER	AC
Full Training	**100.0**	77.3	**100.0**	**75.0**	**91.7**	**50.0**
w/o RL	90.9	77.3	75.0	50.0	58.3	25.0
RL w/o template	95.5	**81.8**	87.5	**75.0**	75.0	33.3
RL w/o weighted	**100.0**	77.3	87.5	62.5	83.3	41.7
RL w/o eff-reward	95.5	**81.8**	**100.0**	**75.0**	83.3	41.7

reasoning, while the weighted loss enables more precise credit assignment to the modeling strategy segment, ensuring that key strategic decisions receive stronger optimization feedback during training.

Table 5 further confirms this observation, showing that the performance gap between the full model and its ablated variants widens as the task difficulty increases. Especially, while *RL w/o template* achieves comparable or even slightly higher performance on easier problems, its accuracy and execution rate degrade sharply on medium and hard subsets.

4. Related Work

4.1. LLMs for Automated Optimization Formulation

Large language models have been applied to automated optimization formulation and solving in operations research, leveraging their capabilities in mathematical reasoning and code generation (Brown et al., 2020; Wei et al., 2022). Early work such as NL4OPT shows that LLMs can translate natural language problem descriptions into linear optimization formulations by identifying decision variables, constraints, and objectives (Ramamonjison et al., 2022). Subsequent pipeline based approaches integrate LLMs with optimization solvers to produce solver executable models, including AutoFormulation and OptiMUS (AhmadiTeshnizi et al., 2024; Huang et al., 2025). In parallel, learning based methods fine tune LLMs on optimization specific datasets to improve formulation accuracy and executability, with representative examples including ORLM and LLMOpt (Huang et al., 2025; Jiang et al., 2025). More recent studies incorporate solver feedback during training or inference, including SIRL, which uses feasibility and optimality signals as rewards (Chen et al., 2025), StepORLM, which introduces process level supervision for optimization reasoning (Zhou et al., 2026), and OR-R1, which applies test time policy optimization to refine formulations during inference (Ding et al., 2026). Our work complements these directions by making Modeling Strategy explicit as a first class planning layer and optimizing it with solver verified data and strategy focused credit assignment.

4.2. Post Training for LLM Reasoning

Post training techniques are widely used to enhance reasoning capabilities. Prompting and few shot learning can improve performance without parameter updates, but they often remain limited on complex multi stage reasoning tasks (Brown et al., 2020). Supervised fine tuning provides a systematic approach by training models to imitate high quality reasoning trajectories. Its effectiveness can be improved by rejection sampling, which enables large scale acquisition of high quality traces without additional human annotation (Bai et al., 2022).

Beyond imitation, preference based methods such as Direct Preference Optimization provide an efficient framework for alignment, with performance that depends on the quality and diversity of preference data (Rafailov et al., 2023). Reinforcement learning further enables direct optimization with verifiable feedback. Reinforcement learning from human feedback and policy gradient methods such as Proximal Policy Optimization have achieved strong results on reasoning benchmarks (Christiano et al., 2017; Schulman et al., 2017; Ouyang et al., 2022; Cobbe et al., 2021). Group Relative Policy Optimization is a critic free variant that improves efficiency by using relative reward comparisons within groups of sampled trajectories (Shao et al., 2024). Our Segment Weighted GRPO builds on GRPO and improves credit assignment in long horizon modeling by emphasizing the strategy segment, while also incorporating solver efficiency into the training objective.

5. Conclusion

We study strategy-aware optimization modeling and propose SAGE, a framework that treats formulation-level Modeling Strategy as an explicit and trainable object rather than leaving it implicit in final code generation. By making strategy selection determine the decision domain, variable backbone, and constraint structure, SAGE operationalizes this perspective through a solver-verified multi-strategy dataset and Segment-Weighted GRPO with rewards for format compliance, correctness, and solver efficiency.

Experiments across benchmarks show improvements in pass@1 accuracy, pass@K behavior, modeling diversity, and solver efficiency. These results suggest that explicit Modeling Strategy provides benefits beyond generic reasoning or step-wise decomposition, offering a practical path toward more reliable and efficient automated optimization modeling.

6. Limitations

SAGE is intentionally specialized for automated optimization modeling. Its reasoning template, segment weighting, and efficiency reward are designed around OR-specific formulation structures and solver feedback, and may require substantial redesign to transfer to other reasoning or coding domains. The data construction stage also relies on a strong teacher model to generate strategy-explicit reasoning traces. To facilitate reproducibility and reduce the need for future work to repeat this costly teacher-based synthesis process, we have released the 108K solver-verified multi-strategy training dataset.

In open-ended real-world problems without ground-truth answers, solver feedback alone cannot always determine whether the selected modeling strategy is globally optimal. Future work may investigate stronger formulation-level verification, weaker-teacher or self-evolving data construction, adaptive multi-sampling and refinement, and extensions of strategy-explicit modeling to broader structured reasoning domains.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (72542016, 72571019). We thank the anonymous reviewers for their valuable feedback.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

AhmadiTeshnizi, A., Gao, W., and Udell, M. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Antoniou, A. and Lu, W.-S. *Practical Optimization: Algorithms and Engineering Applications*. Springer, 2007.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Bartolacci, M. R., LeBlanc, L. J., Kayikci, Y., and Grossman, T. A. Optimization modeling for logistics: options and implementations. *Journal of Business Logistics*, 33 (2):118–127, 2012.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901, 2020.

Calafiore, G. C. and El Ghaoui, L. *Optimization Models*. Cambridge University Press, 2014.

Chen, Y., Xia, J., Shao, S., Ge, D., and Ye, Y. Solver-Informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, 2025.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., and Amodei, D. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Ding, Z., Tan, Z., Zhang, J., and Chen, T. OR-R1: Automating modeling and solving of operations research optimization problem via test-time reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2026.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.

Guo, Y., Xu, L., Liu, J., Ye, D., and Qiu, S. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *arXiv preprint arXiv:2505.23564*, 2025b.

Hillier, F. S. *Introduction to operations research*. McGrawHill, 2005.

Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., and Wang, Z. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025.

Huang, X., Shen, Q., Hu, Y., Gao, A., and Wang, B. MAMO: A mathematical modeling benchmark with solvers. *arXiv e-prints*, pp. arXiv–2405, 2024.

Hurst, A., Lerer, A., Goucher, A. P., Perelman, A., Ramesh, A., Clark, A., Ostrow, A., Welihinda, A., Hayes, A., Radford, A., et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Jayal, A. D., Badurdeen, F., Dillon Jr, O., and Jawahir, I. S. 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.

Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., and Yu, Y. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, 2025.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. In *International Conference on Learning Representations*, 2024.

LINDO Systems, Inc. Formulating and solving integer programs (chapter 11). LINGO Documentation.

Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., et al. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Lu, H., Xie, Z., Wu, Y., Ren, C., Chen, Y., and Wen, Z. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pp. 27730–27744, 2022.

Ponsich, A., Jaimes, A. L., and Coello, C. A. C. A survey on multiobjective evolutionary algorithms for the solution of the portfolio optimization problem and other finance and economics applications. *IEEE Transactions on evolutionary computation*, 17(3):321–344, 2012.

Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741, 2023.

Ramamonjison, R., Yu, T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, 2022.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Sheng, G., Zhang, C., Ye, Z., Wu, X., Zhang, W., Zhang, R., Peng, Y., Lin, H., and Wu, C. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of European Conference on Computer Systems*, pp. 1279–1297, 2025.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Williams, H. P. *Model Building in Mathematical Programming*. John Wiley & Sons, 2013.

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., et al. Chain-of-Experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

Xiao, Z., Xie, J., Xu, L., Guan, S., Zhu, J., Han, X., Fu, X., Yu, W., Wu, H., Shi, W., et al. A survey of optimization modeling meets LLMs: Progress and future directions. *arXiv preprint arXiv:2508.10047*, 2025.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Yang, Z., Wang, Y., Huang, Y., Guo, Z., Shi, W., Han, X., Feng, L., Song, L., Liang, X., and Tang, J. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *International Conference on Learning Representations*, 2025b.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023.

Zhang, B. and Luo, P. OR-LLM-Agent: Automating modeling and solving of operations research optimization problem with reasoning large language model. *arXiv preprint arXiv:2503.10009*, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623, 2023.

Zhou, C., Xu, T., Lin, J., and Ge, D. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *International Conference on Learning Representations*, 2026.

A. Additional Experiment Details

A.1. Benchmark Details

Recent years have witnessed rapid progress in dataset construction for natural language optimization modeling, with a growing number of benchmarks and synthesis pipelines proposed to improve the ability of large language models to formulate and solve operations research problems. These datasets cover linear programming (LP), mixed-integer linear programming (MILP), and real-world industrial optimization scenarios. However, the increasing complexity also introduces non-negligible quality issues, including logical inconsistencies, underspecified problem statements, and incorrect reference solutions. A recent research survey systematically examined widely-used optimization datasets described in natural language, identifying and correcting a substantial portion of such issues through manual verification and revalidation (Xiao et al., 2025). In our experiments, we strictly follow this cleaned and verified benchmark protocol, and all evaluations are conducted on the corrected subsets to ensure robustness and fairness. The number of problems within each dataset is detailed in Table 6.

Specifically, we evaluate our method on the following representative benchmarks:

- **NL4Opt.** A benchmark consisting of natural language descriptions of linear programming problems, primarily focusing on the translation from textual problem statements to formal LP models. The problems span classical OR settings such as resource allocation and scheduling.
- **MAMO.** A modeling-oriented benchmark designed to assess formulation correctness rather than solution accuracy alone. It includes LP and MILP problems and is divided into EasyLP and ComplexLP subsets with distinct levels of structural difficulty.
- **NLP4LP.** A human-authored benchmark with fine-grained annotations, covering LP and MILP problems from classical OR domains such as knapsack, scheduling, and production planning. The dataset emphasizes executable and semantically correct formulations.
- **ComplexOR.** A small but challenging benchmark curated from advanced OR case studies, featuring conceptually complex mixed-integer formulations that often require hierarchical reasoning or time-indexed modeling.
- **IndustryOR.** A real-world industrial benchmark comprising optimization problems from manufacturing, logistics, finance, and energy domains. The problems span multiple formulation types and are annotated by difficulty level.
- **OptiBench** A large-scale dataset generated via reverse Socratic synthesis, where optimization demonstrations are transformed into natural language problem descriptions. Compared to earlier synthetic datasets, it provides higher-quality intermediate reasoning structures.
- **OptMATH.** A benchmark designed to overcome the limited diversity and difficulty of earlier optimization modeling datasets. It covers a wide range of optimization paradigms beyond linear formulations, including LP, MILP, nonlinear and second-order cone programming.

Table 6. Summary of evaluation datasets and number of validated instances used in our experiments.

Dataset	NL4Opt	MAMOEasy	MAMOComplex	NLP4LP	ComplexOR	IndustryOR	OptiBench	OptMATH
# Instances	213	545	111	178	18	42	403	166

A.2. Scaling Problem Details

This section provides additional details on the construction of scaled optimization problems used in the efficiency performance evaluation.

Problem Selection. To isolate solver efficiency from solution correctness, we restrict the evaluation to a subset of problems that can be solved stably and correctly by all compared methods. Specifically, we select seven problems from the ComplexOR benchmark for which SIRL, DeepSeek-V3, and SAGE consistently produce correct and feasible solutions.

Problem Types. The selected problems consist of seven representative instances drawn from the ComplexOR benchmark, covering a diverse set of classical operations research formulations. Specifically, the problems include: (i) a Car Selection Problem, (ii) a capacitated Transportation Problem, (iii) a Knapsack Problem, (iv) a Cutting Stock Problem, (v) a Diet Problem, (vi) a Project Assignment Problem, and (vii) an Aircraft Assignment Problem. This diversity ensures that the efficiency evaluation is not biased toward a specific problem class or formulation type.

Scaling Procedure. For each selected problem, the original instance is treated as the base scale. Larger instances are constructed by linearly expanding the key structural dimensions of the problem. Formally, for a given base size b and scale factor s, the scaled size is defined as $\min(b \times s, b_{\max})$, where $b_{\max}$ is a problem-specific upper bound following the standard practices. Importantly, the underlying problem structure and constraint logic remain unchanged across scales.

Instance Generation and Verification. For each problem and each scale factor $s \in \{1, \ldots, 8\}$, we generate five independent instances using a deterministic seed derived from the problem name and scale index. All instances are constructed to be feasible by design. Each generated instance is further verified by an external solver before inclusion in the evaluation. Overall solver efficiency metrics are reported by averaging over all correct solutions at each scale.

Case: Scaling a Transportation Problem

Base Problem Statement

Consider a transportation problem. Given a set of `Origins` and a set of `Destinations`, each origin i has a $\texttt{Supply}_i$ and each destination j has a $\texttt{Demand}_j$. Shipping one unit of goods from origin i to destination j incurs a cost $\texttt{Rate}_{i,j}$ and is subject to an upper bound $\texttt{Limit}_{i,j}$. The goal is to minimize the total transportation cost while satisfying all supply, demand, and capacity constraints.

Variable assignments (base scale)

```
supply = [20, 30]; demand = [30, 20]; rate = [[8, 6],[5, 10]]; limit = [[15,
25],[25, 20]]
```

Algorithm 1 Transportation Problem Automated Scaling Procedure

input base instance sizes $|\mathcal{O}_0|$, $|\mathcal{D}_0|$ and scale factor s
1: Set $|\mathcal{O}| \leftarrow s \cdot |\mathcal{O}_0|$
2: Set $|\mathcal{D}| \leftarrow s \cdot |\mathcal{D}_0|$
3: Sample capacity limits for all origin–destination pairs
4: Sample transportation costs for all origin–destination pairs
5: Construct destination demands from column capacity totals
6: Construct origin supplies from row capacity totals
7: Adjust boundary rows or columns to enforce total balance
output a feasible scaled transportation instance

Scaled Problem at $\times 8$. At scale factor $s = 8$, the transportation problem expands from 2×2 to 16×16 origins and destinations, increasing the number of decision variables from 4 to 256. The problem statement and constraint structure remain identical to the base formulation.

Variable assignments (at $\times 8$ scale)

```
supply = [503, 604, 376, 434, 520, 454, ..., 458, 485] % 16 origins
demand = [420, 416, 538, 425, 434, 532, ..., 518, 496] % 16 destinations
rate   = [
           [ 6,  8,  9, 11, 13,  8, ...,  9,  7],
           [10, 15, 11, 14,  6,  4, ...,  7,  2],
           ...
           [ 5, 15, 12,  8,  5,  9, ...,  2, 10]
         ]                                        % 16 x 16 cost matrix
limit  = [
           [42, 53, 31, 48, 41, 57, ..., 57, 52],
           [31, 55, 26, 30, 35, 54, ..., 50, 59],
           ...
           [22, 29, 40, 40, 28, 44, ..., 56, 39]
         ]                                        % 16 x 16 capacity matrix
```

A.3. Hyperparameter Settings

We report the main hyperparameter configurations used in both the supervised fine-tuning and reinforcement learning stages. All training experiments were conducted on a single compute node equipped with eight NVIDIA H100 (80 GB) GPUs. The selected hyperparameters were empirically tuned to ensure stable optimization, efficient convergence, and balanced reward scaling across stages. Table 7 summarizes the complete training configurations.

Table 7. Key hyperparameters for SFT and RL training.

Stage	Type	Parameter	Value
SFT	**Data**	Optimizer	AdamW
		Learning rate	1×10^{-5}
		Max sequence length	8192
		Training epochs	2
RL	**Algorithm**	Advantage estimator	GRPO
	Data	Batch size	32
		Learning rate	1×10^{-6}
		Max prompt length	2048
		Max response length	8192
	Actor/Rollout	KL loss type	low_var_kl
		KL loss coefficient	0.001
		PPO mini batch size	8
		PPO micro batch size per GPU	4
		Rollout number	8
		Training epochs	2
	Segment Weights	strategy	2.0
		modeling	1.5
		others	1.0

B. Additional Experiment Results

B.1. Statistical Robustness

Table 8 reports the mean and standard deviation of Pass@1 accuracy over five repeated evaluations with different decoding seeds. SAGE maintains the highest average performance among the compared methods, achieving 79.0 ± 1.84 on average. Although all methods exhibit some variance on smaller and harder benchmarks, SAGE consistently retains clear advantages on ComplexOR, IndustryOR, and OptMATH, indicating that the main performance gains are robust to decoding randomness.

Table 8. **Statistical robustness of main results with Pass@1 accuracy (%).** We report mean ± standard deviation over five repeated evaluations with different decoding seeds. The best results are highlighted in **bold** and the second-best results are underlined.

Models	Easy Tasks				Complex Tasks				Avg.
	NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
DeepSeek-V3	$90.6_{\pm 0.30}$	<u>$95.6_{\pm 0.16}$</u>	$89.9_{\pm 0.36}$	$87.1_{\pm 0.35}$	$60.7_{\pm 2.65}$	$51.1_{\pm 4.16}$	<u>$67.6_{\pm 2.86}$</u>	<u>$41.1_{\pm 1.50}$</u>	<u>$73.0_{\pm 1.54}$</u>
DeepSeek-R1	$88.5_{\pm 1.67}$	$88.3_{\pm 1.05}$	$84.5_{\pm 1.61}$	$80.4_{\pm 1.75}$	$66.8_{\pm 3.80}$	<u>$52.2_{\pm 4.44}$</u>	$57.6_{\pm 4.10}$	$35.3_{\pm 2.49}$	$69.2_{\pm 2.61}$
SIRL-Q2.5-7B	<u>$94.7_{\pm 0.19}$</u>	$\mathbf{96.9}_{\pm 0.27}$	<u>$96.3_{\pm 0.76}$</u>	<u>$91.2_{\pm 0.37}$</u>	<u>$80.4_{\pm 0.36}$</u>	$46.7_{\pm 2.72}$	$50.5_{\pm 3.50}$	$25.5_{\pm 1.35}$	$72.8_{\pm 1.19}$
StepORLM-Q3-8B	$\mathbf{96.9}_{\pm 0.64}$	$95.2_{\pm 0.74}$	$95.8_{\pm 1.04}$	$87.4_{\pm 0.56}$	$79.8_{\pm 4.05}$	$48.9_{\pm 2.22}$	$52.4_{\pm 4.76}$	$13.6_{\pm 3.09}$	$71.3_{\pm 2.14}$
SAGE-DS-14B (Ours)	$92.9_{\pm 0.35}$	$93.4_{\pm 0.75}$	$\mathbf{98.3}_{\pm 1.51}$	$\mathbf{93.4}_{\pm 0.85}$	$\mathbf{81.6}_{\pm 2.59}$	$\mathbf{60.0}_{\pm 4.16}$	$\mathbf{68.1}_{\pm 2.86}$	$\mathbf{44.7}_{\pm 1.68}$	$\mathbf{79.0}_{\pm 1.84}$

B.2. Main results of Qwen3-8B

Table 9 reports the main results under the Qwen3-8B backbone, comparing our method with representative online RL baselines, including SIRL and StepORLM. All methods are evaluated under the same Pass@1 accuracy metric.

Overall, SAGE-Qwen3-8B achieves the highest average performance among Qwen3-8B-based models. While the performance on easier benchmarks is generally comparable across methods, the advantage of our approach becomes more pronounced on complex tasks. In particular, SAGE-Qwen3-8B consistently outperforms prior methods on MAMO-Complex, ComplexOR, IndustryOR, and OptMATH, indicating stronger robustness in handling structurally complex and large-scale optimization problems.

These results are consistent with the main findings reported in the full-scale experiments, further demonstrating that the proposed strategy-aware reinforcement learning framework remains effective even under a fixed and relatively compact model backbone.

Table 9. **Main results of Qwen3-8B models with Pass@1 accuracy (%).** The best results are highlighted in **bold** and the second-best results are underlined.

Types	Models	Easy Tasks				Complex Tasks				Avg.
		NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
Online-RL	SIRL-Q2.5-7B	94.8	**98.0**	96.6	89.6	81.1	44.4	50.0	27.1	72.7
	StepORLM-Q3-8B	**96.7**	95.4	**97.2**	88.6	79.3	50.0	52.4	13.9	71.7
	SAGE-Qwen3-8B (Ours)	93.9	95.0	96.6	**92.1**	**82.0**	**55.6**	**57.1**	**43.4**	**77.0**

B.3. Generalizability to Open-Source Solvers

To evaluate whether strategy-aware reasoning transfers beyond the Gurobi Python API, we conduct a controlled solver-transfer experiment. For each method, we extract only the solver-agnostic optimization model specification, including sets, parameters, variables, objective, and constraints, and use the same DeepSeek-V3 model to generate HiGHS and OR-Tools code from this specification. This setup reduces the confound from solver-specific code-generation ability. The HiGHS and OR-Tools results are reported in Table 10 and Table 11, respectively.

As shown in the results, SAGE achieves the best average transfer performance on both solver backends. Its advantage is especially clear on harder benchmarks such as ComplexOR, IndustryOR, and OptMATH, suggesting that SAGE learns more transferable optimization model specifications rather than only Gurobi-specific code patterns.

Table 10. **Transferability of strategy-aware reasoning to HiGHS with Pass@1 accuracy (%).** DeepSeek-V3 is used to generate HiGHS code from the structured optimization model specification produced by each method. The best results are highlighted in **bold** and the second-best results are underlined.

Models	Easy Tasks				Complex Tasks				Avg.
	NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
SIRL-Q2.5-7B + DS-V3	94.8	**96.3**	**98.3**	81.3	47.1	21.1	46.8	12.4	62.3
StepORLM-Q3-8B + DS-V3	**96.2**	88.0	96.8	81.4	46.4	20.4	52.4	1.4	60.4
SAGE-DS-14B + DS-V3	91.7	93.5	95.7	**82.0**	**48.3**	**40.7**	**58.7**	**25.1**	**67.0**

Table 11. **Transferability of strategy-aware reasoning to OR-Tools with Pass@1 accuracy (%).** DeepSeek-V3 is used to generate OR-Tools code from the structured optimization model specification produced by each method. The best results are highlighted in **bold** and the second-best results are underlined.

Models	Easy Tasks				Complex Tasks				Avg.
	NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
SIRL-Q2.5-7B + DS-V3	94.8	**96.6**	96.1	**81.1**	37.5	5.6	34.9	13.5	57.5
StepORLM-Q3-8B + DS-V3	**95.9**	92.4	**96.4**	79.7	31.5	7.4	45.2	2.2	56.3
SAGE-DS-14B + DS-V3	90.9	89.4	93.1	79.1	**37.8**	**31.5**	**48.4**	**28.7**	**62.4**

B.4. Segment Weight Sensitivity

We further study the sensitivity of Segment-Weighted GRPO to different segment weight settings. We fix $\alpha_{\text{check}} = 1.0$ and vary the weights of the strategy and modeling segments. The results are reported in Table 12. Overall, performance is reasonably stable across different settings, while the moderate setting $(\alpha_{\text{strategy}}, \alpha_{\text{modeling}}) = (2.0, 1.5)$ achieves the best average performance. This suggests that emphasizing strategy and modeling tokens is beneficial, but overly large weights may hurt performance on harder benchmarks.

Table 12. **Sensitivity analysis of segment weights with Pass@1 accuracy (%).** We fix $\alpha_{\text{check}} = 1.0$ and vary α_{strategy} and α_{modeling}. The best results are highlighted in **bold** and the second-best results are underlined.

$(\alpha_{\text{strategy}}, \alpha_{\text{modeling}})$	Easy Tasks				Complex Tasks				Avg.
	NL4OPT	MAMO Easy	NLP4LP	OptiB.	MAMO Cpx.	CpxOR	IndOR	OptM.	
(1.0, 1.0)	92.5	**94.7**	96.6	91.8	73.9	55.6	<u>64.3</u>	44.0	76.7
(1.5, 1.25)	<u>94.8</u>	93.0	97.2	<u>93.3</u>	**85.6**	**61.1**	<u>64.3</u>	<u>45.2</u>	<u>79.3</u>
(2.0, 1.5)	94.3	**94.7**	**98.9**	93.8	<u>84.7</u>	**61.1**	**69.0**	**45.8**	**80.3**
(2.5, 1.75)	**95.3**	92.8	**98.9**	**94.5**	81.1	55.6	57.1	44.6	77.5

B.5. Case Study

Case Study 1: Strategy Diversity with Increasing Pass@K. This case study illustrates how our model discovers increasingly diverse optimization modeling strategies as the sampling budget increases. We consider a project assignment problem drawn from the ComplexOR benchmark and examine two fundamentally different optimization formulations generated by our model at different sampling stages.

The first formulation, produced at pass@1, adopts a compact continuous transportation-style modeling strategy, where working hours are treated as divisible flow variables directly allocated between people and projects. In contrast, a second formulation emerges only at pass@16, which follows a fundamentally different modeling paradigm by discretizing working hours into unit time slots and formulating the problem as a time-indexed 0–1 assignment model. Although both formulations are mathematically valid and yield correct solutions, they differ substantially in their underlying modeling philosophy, variable granularity, constraint construction, and computational characteristics, as summarized in Table 13.

Table 13. Comparison of Two Modeling Strategies Generated at Different Pass@K

	Continuous Flow Model (pass@1)	**Unit-Time 0–1 Assignment Model** (pass@16)
Modeling Strategy	Directly model hour flow between people and projects	Discretize hours into unit time slots and assign individually
Model Type	Linear Programming (LP)	Integer Programming (IP / MILP)
Decision Variables	x_{ij}: total hours from person i to project j	x_{ijk}: whether hour k of person i is assigned to project j
Variable Type	Continuous ($x_{ij} \geq 0$)	Binary ($x_{ijk} \in \{0, 1\}$)
Supply Constraints	$\sum_j x_{ij} = \text{Supply}_i$	$\sum_j x_{ijk} = 1, \ \forall i, k$
Practical Efficiency	High, scalable	Low, variable explosion
Compactness	Highly compact	Highly disaggregated

Case 1: Diverse Optimization Modeling

Problem Statement
Consider a project assignment problem. Given a set of people `People` and a set of projects `Projects`. Each person i has a certain number of available hours Supply_i and each project j requires a certain number of hours Demand_j. The cost per hour of work for person i on project j is $\text{Cost}_{i,j}$. Each person i can contribute to project j up to a maximum limit $\text{Limit}_{i,j}$. The problem aims to minimize the total cost of assigning people to projects. It is constrained that the total number of hours assigned from each person i equals its supply and the total number of

hours assigned to each project j equals its demand.

Variable explanations
Args:

- `Supply`: available hours per person
- `Demand`: required hours per project
- `Cost`: cost per hour for each person–project pair
- `Limit`: maximum allowed contribution per pair

Returns:

- `total_cost`: minimized total assignment cost

Variable assignments
{"supply": [8, 7], "demand": [5, 10], "cost": [[10, 20], [15, 25]], "limit": [[5, 6], [4, 7]]}

Continuous Flow Model (pass@1)

```
# Decision variables: x[i,j] >= 0 (hours)
x = model.addVars(len(supply), len(demand), lb=0.0)

# Objective
model.setObjective(
    gp.quicksum(cost[i][j] * x[i,j]
                for i in range(len(supply))
                for j in range(len(demand))),
    GRB.MINIMIZE
)

# Supply constraints
for i in range(len(supply)):
    model.addConstr(sum(x[i,j] for j in range(len(demand))) == supply[i])

# Demand constraints
for j in range(len(demand)):
    model.addConstr(sum(x[i,j] for i in range(len(supply))) == demand[j])

# Limit constraints
for i in range(len(supply)):
    for j in range(len(demand)):
        model.addConstr(x[i,j] <= limit[i][j])
```

Unit-Time 0–1 Assignment Model (pass@16)

```
# Binary variables x[i,j,k]
x = {}
for i in people:
    for j in projects:
        for k in range(supply[i]):
            x[i,j,k] = model.addVar(vtype=GRB.BINARY)

# Objective
model.setObjective(
    gp.quicksum(cost[i][j] * x[i,j,k]
                for i in people
                for j in projects
                for k in range(supply[i])),
    GRB.MINIMIZE
)
```

```python
# Demand constraints
for j in projects:
    model.addConstr(sum(x[i,j,k] for i in people for k in range(supply[i]))
                    == demand[j])

# Limit constraints
for i in people:
    for j in projects:
        model.addConstr(sum(x[i,j,k] for k in range(supply[i])) <= limit[i][j])

# Assignment constraints
for i in people:
    for k in range(supply[i]):
        model.addConstr(sum(x[i,j,k] for j in projects) == 1)
```

Case Study 2: Why Strategy-Aware Modeling Leads to Higher Solver Efficiency. In this case study, we investigate why the formulations generated by SAGE consistently achieve higher solver efficiency compared to strong baselines. We focus on a project assignment problem from the ComplexOR benchmark and compare two correct formulations: one generated by SAGE and the other by DeepSeek-V3.

Table 14 shows the solver efficiency details between these two formulations. Although both models encode the same optimization problem and yield valid solutions, the formulation produced by DeepSeek-V3 introduces redundant constraints and variables, resulting in a larger and less compact constraint matrix. In contrast, SAGE generates a more concise formulation that avoids unnecessary modeling artifacts. This structural difference directly translates into improved solver performance, as reflected in reduced model size and faster convergence.

Table 14. Solver efficiency comparison between SAGE and DeepSeek-V3 formulations.

Model	#Vars	#Constr.	NNZ	Solver Time (s)	Total Iterations
Compact Model (SAGE)	384	40	768	0.0063	29
Redundant Model (DS-V3)	384	424	1152	0.0186	29

Case 2: Higher Solver Efficiency Analysis

Problem Statement
The Aircraft Assignment Problem aims to assign aircraft to routes in order to minimize the total cost while satisfying demand constraints with available aircraft. The problem involves a set of aircraft and a set of routes. Given the costs of assigning an aircraft to a route, the objective is to minimize the total cost of the assignment. There are limited available aircraft, and it is constrained that the number of each aircraft allocated does not exceed its available number. Given the demand of each route and the capabilities (the largest number of people that can be carried) of an aircraft for a route, the demand constraint ensures that the total allocation for each route satisfies the demand. The problem seeks to find the most cost-effective assignment of aircraft to routes.

Variable explanations
Args:

- `availability`: list, availability of each aircraft
- `demand`: list, demand for each route
- `capabilities`: 2D list, capabilities of each aircraft for each route
- `costs`: 2D list, costs of assigning each aircraft to each route

Returns:

- `min_total_cost`: float, the minimum total cost of the assignment

Model 1: Compact Formulation (SAGE)

```
# Define sets
num_aircraft = len(availability)
num_routes = len(demand)
I = range(num_aircraft)
J = range(num_routes)

# Decision variables
x = model.addVars(I, J, vtype=GRB.INTEGER, name="x")

# Objective
model.setObjective(
    gp.quicksum(costs[i][j] * x[i,j] for i in I for j in J),
    GRB.MINIMIZE
)

# Availability constraints
model.addConstrs(
    (gp.quicksum(x[i,j] for j in J) <= availability[i] for i in I),
    name="availability"
)

# Demand constraints
model.addConstrs(
    (gp.quicksum(capabilities[i][j] * x[i,j] for i in I) >= demand[j] for j in J),
    name="demand"
)
```

Model 2: Redundant Formulation (DeepSeek-V3)

```
# Sets
aircraft = range(len(availability))
routes = range(len(demand))

# Variables
x = model.addVars(aircraft, routes, vtype=GRB.INTEGER, name="x")

# Objective
model.setObjective(
    gp.quicksum(costs[i][j] * x[i,j] for i in aircraft for j in routes),
    GRB.MINIMIZE
)

# Availability constraints
for i in aircraft:
    model.addConstr(
        gp.quicksum(x[i,j] for j in routes) <= availability[i],
        f"availability_{i}"
    )

# Demand constraints
for j in routes:
    model.addConstr(
        gp.quicksum(capabilities[i][j] * x[i,j] for i in aircraft) >= demand[j],
        f"demand_{j}"
    )

# Non-negativity constraints
for i in aircraft:
    for j in routes:
        model.addConstr(x[i,j] >= 0, f"non_neg_{i}_{j}")
```

C. Additional Methodology Details

C.1. Efficiency Reward Design.

To capture solver efficiency consistently across different optimization problem types, we design problem-specific efficiency metrics that reflect the computational cost observed during solver execution. For each correctly solved instance, we extract a scalar efficiency measure $M(y)$ from solver feedback and transform it into a bounded reward using:

$$R_{\text{efficiency}}(y) = 1 - \tanh\left(\frac{M(y)}{\alpha_{\text{eff}}}\right), \tag{13}$$

where α_{eff} is a global scaling constant controlling the reward's sensitivity to solver efficiency. A smaller $M(y)$ implies better solver performance, yielding a higher $R_{\text{efficiency}}(y)$. The definition of $M(y)$ depends on the underlying problem class, as detailed below.

Linear Programs (LP). For linear programming problems, solver effort is primarily determined by the number of iterations required for convergence under simplex or barrier methods. Hence, we define:

$$M_{\text{LP}}(y) = \text{IterationCount}(y). \tag{14}$$

The efficiency reward then becomes:

$$R^{(\text{LP})}_{\text{efficiency}}(y) = 1 - \tanh\left(\frac{\text{IterationCount}(y)}{\alpha_{\text{iter}}}\right). \tag{15}$$

Here, α_{iter} is a scaling factor determining the solver iteration range considered "typical." This design penalizes unnecessarily long convergence sequences while ensuring smooth gradient variation even for large-scale LPs.

Mixed-Integer Linear Programs (MILP). For MILP problems, solver efficiency depends jointly on the tightness of continuous relaxations and the complexity of the branch-and-bound search process. Two solver-provided quantities are used: the *optimality gap*, measuring the relative distance between the incumbent and best-known bound, and the *node count*, measuring the number of explored search tree nodes. These are normalized and aggregated to form the unified metric:

$$M_{\text{MILP}}(y) = \beta_{\text{gap}} \cdot \text{Gap}(y) + \beta_{\text{nodes}} \cdot \text{Nodes}(y), \tag{16}$$

where β_{gap} and β_{nodes} control the relative contribution of the gap and node count. Both terms are first normalized by solver-scale statistics to balance their numerical ranges. The corresponding reward is given by:

$$R^{(\text{MILP})}_{\text{efficiency}}(y) = 1 - \tanh\left(\frac{M_{\text{MILP}}(y)}{\alpha_{\text{eff}}}\right). \tag{17}$$

This formulation encourages the model to generate formulations that improve solver relaxation tightness (lower gap) and reduce combinatorial branching (fewer nodes), both of which are well-established indicators of modeling efficiency in MILP research.

Scaling and Weighting Factors. Table 15 summarizes all scaling and weighting factors used in the efficiency reward computation. The scaling factors $\alpha_{\text{iter}}, \alpha_{\text{gap}}, \alpha_{\text{nodes}}$ control the sensitivity of the reward to solver performance metrics, while the weighting factors $\beta_{\text{gap}}, \beta_{\text{nodes}}$ determine their relative contribution in mixed-integer optimization tasks. All parameters were empirically selected to ensure smooth reward shaping and numerical stability across scales.

Table 15. Scaling and weighting factors used in the efficiency reward computation.

	$\boldsymbol{\alpha}_{\text{iter}}$	$\boldsymbol{\alpha}_{\text{gap}}$	$\boldsymbol{\alpha}_{\text{nodes}}$	$\boldsymbol{\beta}_{\text{gap}}$	$\boldsymbol{\beta}_{\text{nodes}}$
Value	15.0	1/3	5.0	0.5	0.5

C.2. Prompt Templates

Prompts for Data Generation. We use three prompt templates to generate diverse optimization modeling data in a structured pipeline. Each template serves a distinct role in producing strategy-diverse, solver-executable training samples.

The first template is used to generate multiple modeling strategies for the same optimization problem. Given a selected strategy, the second template guides the model to produce a detailed step-by-step modeling reasoning and the corresponding Gurobi Python implementation. Finally, the third template is used for strategy de-duplication. Using an LLM-as-Judge, it compares the generated strategies and their solver code to identify and remove semantically redundant formulations that differ only superficially.

To assess the reliability of the LLM-as-Judge used for strategy de-duplication, we manually audited a random subset of 50 generated strategy pairs. On this subset, the judge's decisions agreed with human judgments in approximately 87% of cases. Most disagreements arose from borderline cases involving formulations with nearly identical mathematical structure, such as differences limited to variable renaming or constraint reordering. Importantly, all candidate formulations are solver-verified prior to de-duplication, thus the LLM-as-Judge is therefore only used to control redundancy among already correct and executable models, rather than to assess formulation correctness.

Modeling strategy generation prompt

You are a mathematical modeling expert. Given the following optimization problem, propose **3 different modeling strategies** that could be used to formulate it. Make sure the strategies differ not just in wording or variable names, but in modeling structure, such as using different variable types, constraint formulations, or problem decompositions.

For each strategy, provide the following fields in the same structure:

——

Strategy X
Strategy Name: [Short name of the formulation]
Description: [The summary of the approach]

——

Do not write equations or code. Keep the output structured and machine-readable.
Here is the problem:
{{Question}}

Modeling reasoning & Gurobi code generation prompt

You are a mathematical modeling expert. Given the following modeling strategy for an optimization problem, please do the following:

1. Write a detailed **step-by-step modeling reasoning** explaining how to build the model according to this strategy:
   - What sets are defined?
   - What parameters are introduced?
   - How are the decision variables designed?
   - What is the objective function?
   - What are the key constraints?
2. Then, write the full **Gurobi Python** code that implements this formulation.

Strategy Description:
{{Strategy}}

Only output the Python code inside a code block:

```
```python
# ... your final code ...
```

Here is the problem:
{{Question}}
```

Strategy de-duplication prompt

SYSTEM: You are an expert in combinatorial optimization and mathematical modeling. Your task is to determine whether the given strategies represent fundamentally different modeling paradigms. Judge differences based on the following aspects:

- Variable definitions (binary, continuous, state-based, etc.)
- Objective formulation (value maximization, cost minimization, etc.)
- Constraints (linear, logical, recursive, etc.)
- Solution method (e.g., MIP, Dynamic Programming, Branch and Bound, Heuristic, Reinforcement Learning, etc.)

Two strategies are different if they differ in any of these aspects, even if they share similar wording or target the same problem.Return your answer in **STRICT JSON format** only.

Your response must be a single valid JSON object with the exact structure: {"verdict": "different" or "similar", "similar_ids": [list of ids if similar, otherwise empty list]}

Do not include any text, explanation, markdown, or additional formatting outside the JSON braces.

USER: Here is the Strategy and its Gurobi code: {{Strategies}} + {{Gurobi Code}}

System Prompts and LLM Response These prompts are used during both the RL training stage and the inference stage to guide the model to generate a complete optimization formulation and executable solver code in a fixed and consistent format. They enforce a standardized reasoning and output structure, which is essential for stable policy optimization, solver-based reward computation, and reliable evaluation.

In addition, we provide an example illustrating the input and output of this template on a representative optimization problem from the IndustryOR dataset. The example shows how a natural language problem description is transformed into a structured reasoning trace and corresponding solver code under the strategy-aware format enforced by the system prompt.

Strategy-aware system prompt

SYSTEM: You are an expert in mathematical optimization and operations research. Your goal is to design a globally consistent mathematical model and implement it in Gurobi Python to solve the given problem. Your reasoning must be inside <think> and </think> tags and follow the strict format:

<think>
<strategy>

1. Paradigm Selection: Identify the problem class (e.g., TSP, Scheduling, Facility Location) and choose **the most robust and efficient** modeling paradigm (e.g., Time-indexed, Flow-based, Big-M).
2. Decision Variables: Define the core variables (names, indices, categories) that serve as the foundation.
3. Constraint Logic: Identify key couplings and explain how the defined variables interact to enforce complex constraints.

</strategy>

<modeling>
Formulate the model following the design: Sets, Parameters, Variables, Objective, Constraints. Ensure every segment must match the strategy.
</modeling>

<check>

1. Consistency Check: Does the model strictly follow the selected strategy?
2. Logic Check: Are the dependencies between variables and constraints logically sound?
3. Dimension Check: Do the variable indices in the code match the constraint loops?
4. Efficiency Check: Is the formulation compact and free of redundant variables?

</check>
</think>

Output the final Python code inside a code block:

```
```python
# ... your final code ...
```

USER: Here is the problem:
{{Question}}

Example Problem

 # Question:
The Zhang family has 6 children, Harry, Hermione, Ron, Fred, George, and Ginny. The cost of taking Harry is $1200, Hermione is $1650, Ron is $750, Fred is $800, George is $800, and Ginny is $1500. Which children should the couple take to minimize the total cost of taking the children?

They can take a maximum of 4 children on the upcoming trip.

Ginny is the youngest, so the Zhang family will definitely take her.

If the couple takes Harry, they will not take Fred because Harry doesn't get along with him.

If the couple takes Harry, they will not take George because Harry doesn't get along with him.

If they take George, they must also take Fred.

If they take George, they must also take Hermione.

Although this will cost them a lot of money, the Zhang family has decided to take at least three children.

Example Response: Reasoning & Solver Code

`<think>`
`<strategy>`

1. **Paradigm Selection**: This is a knapsack problem with additional constraints. We use binary decision variables for each child (1 if taken, 0 otherwise).
2. **Decision Variables**:
   - Let `x[i]` be a binary variable for each child $i \in \{$`Harry`, `Hermione`, `Ron`, `Fred`, `George`, `Ginny`$\}$.
3. **Constraint Logic**:
   - Ginny must be taken: `x['Ginny'] = 1`.
   - Maximum of 4 children: `sum(x[i] for i in children) <= 4`.
   - Minimum of 3 children: `sum(x[i] for i in children) >= 3`.
   - If Harry is taken, then Fred and George cannot be taken:
     - `x['Harry'] + x['Fred'] <= 1`
     - `x['Harry'] + x['George'] <= 1`
   - If George is taken, then Fred and Hermione must be taken:
     - `x['George'] <= x['Fred']`
     - `x['George'] <= x['Hermione']`
   - Note: The cost of each child is given as parameters.

`</strategy>`

`<modeling>`

- **Sets**:
  - Children: $\{$`Harry`, `Hermione`, `Ron`, `Fred`, `George`, `Ginny`$\}$
- **Parameters**:
```

  - `cost`: dictionary mapping child to cost
  - `max_children` = 4
  - `min_children` = 3
- **Variables**:
  - `x[i]` for each child i: binary (1 if taken, 0 otherwise)
- **Objective**:
  - Minimize total cost: `sum(cost[i] * x[i] for i in children)`
- **Constraints**:
  - `x['Ginny'] == 1`
  - `sum(x[i] for i in children) <= max_children`
  - `sum(x[i] for i in children) >= min_children`
  - `x['Harry'] + x['Fred'] <= 1`
  - `x['Harry'] + x['George'] <= 1`
  - `x['George'] <= x['Fred']`
  - `x['George'] <= x['Hermione']`

</modeling>

<check>

1. Consistency: The model uses binary variables and all constraints are linear.
2. Logic: All constraints are logically derived from the problem description.
3. Dimensions: Variable indices match the children set.
4. Efficiency: The formulation is compact with no redundant variables.

</check>

Now, the code:

</think>

```python
import gurobipy as gp
from gurobipy import GRB

# Create model
model = gp.Model("ZhangFamilyTrip")

# Children and costs
children = ['Harry', 'Hermione', 'Ron', 'Fred', 'George', 'Ginny']
costs = {
    'Harry': 1200,
    'Hermione': 1650,
    'Ron': 750,
    'Fred': 800,
    'George': 800,
    'Ginny': 1500
}

# Decision variables: binary for each child
x = model.addVars(children, vtype=GRB.BINARY, name="take")

# Objective: minimize total cost
model.setObjective(
    gp.quicksum(costs[i] * x[i] for i in children),
    GRB.MINIMIZE
)

# Constraints
```

```
model.addConstr(x['Ginny'] == 1, "Ginny_must_be_taken")
model.addConstr(gp.quicksum(x[i] for i in children) <= 4, "max_four")
model.addConstr(gp.quicksum(x[i] for i in children) >= 3, "min_three")

# Harry conflicts
model.addConstr(x['Harry'] + x['Fred'] <= 1, "Harry_Fred_conflict")
model.addConstr(x['Harry'] + x['George'] <= 1, "Harry_George_conflict")

# George dependencies
model.addConstr(x['George'] <= x['Fred'], "George_requires_Fred")
model.addConstr(x['George'] <= x['Hermione'], "George_requires_Hermione")

# Optimize
model.optimize()
```