OPTCLIMBER: CURRICULUM EXPERIENCE LEARNING FOR LLM OPTIMIZATION MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models have shown strong potential for automated optimization modeling, yet existing methods mainly focus on how modeling knowledge is provided through prompting or fine-tuning, while paying less attention to how such knowledge should be accumulated throughout learning. We introduce OptClimber, a curriculum experience learning framework that organizes optimization problems by domain and difficulty and progressively builds reusable modeling knowledge from solver-verified problem-solving trajectories. OptClimber couples curriculum scheduling with experience accumulation, allowing reliable modeling principles to be acquired and refined as problem complexity increases. The accumulated experiences are further used to construct solver-verified trajectories for supervised fine-tuning, while the same curriculum structure organizes subsequent reinforcement learning with solver-grounded feedback. In this way, OptClimber connects data organization, experience accumulation, supervised learning, and reinforcement learning within a unified framework. Experiments on six optimization modeling benchmarks show that the prompt-based OptClimber improves its backbone by 18.3% on average, with larger gains on challenging tasks. After fine-tuning and curriculum-guided reinforcement learning, OptClimber-14B achieves 72.7% average accuracy and surpasses the current state-of-the-art fine-tuned model.

1 INTRODUCTION

Optimization modeling translates real world decision problems into mathematical formulations and is central to applications such as resource allocation, scheduling, and transportation. Constructing reliable models requires domain expertise, mathematical abstraction, and careful reasoning. The diversity of optimization problems can make it difficult for learned methods to generalize across problem distributions (Liu et al., 2026b; Pu et al., 2026a), while their complex combinatorial structures also pose challenges to accurately capturing the relationships among variables and constraints (Chen et al., 2023; 2024; Pu et al., 2026b). Recent advances in large language models have driven rapid progress in automated optimization modeling, enabling models to generate mathematical formulations and executable solver code directly from natural language descriptions (Xiu et al., 2026; Li et al., 2026; Zhao et al., 2026). Existing methods mainly follow two paradigms. Prompt-based methods provide modeling knowledge through search, reflection, retrieval, or agent collaboration (Xiu et al., 2026; Yuan et al., 2026), while fine-tuning methods incorporate such knowledge into model parameters through supervised or reinforcement learning (He et al., 2026; Zhao et al., 2026). Despite their different forms, both paradigms ultimately rely on knowledge acquired from optimization problems, solution trajectories, and feedback. However, existing studies largely focus on how such knowledge is provided to the model, while often overlooking how useful modeling knowledge should be accumulated from problem solving experience, including what data to learn from and how the learning process should be organized.

This problem becomes particularly important because not all problem solving experiences are equally useful for learning. Easier problems often expose localized and interpretable errors, making it easier to identify reliable modeling principles, although the resulting knowledge may be limited in scope. As problem difficulty increases, the underlying modeling structures become richer, but the corresponding reasoning trajectories also become longer and more entangled, often involving multiple coupled errors that are difficult to diagnose reliably (Chen et al., 2026a; Li et al., 2026; Jiang

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.

et al., 2026). Experience quality therefore depends not only on the amount of available data, but also on the difficulty of the problems from which the experience is acquired. This suggests that effective knowledge accumulation requires a structured learning process that carefully organizes problems of different difficulty and progressively builds reusable modeling experience (Parashar et al., 2026; Jiang et al., 2026).

Motivated by this observation, we introduce OptClimber, a curriculum experience learning framework for LLM-based optimization modeling. Rather than treating optimization problems as independent examples, OptClimber organizes them into a structured curriculum according to problem domain and modeling difficulty, and uses this curriculum to coordinate both problem selection and knowledge accumulation. During problem-solving, model outputs are verified by an optimization solver, and transferable knowledge from failed trajectories is distilled into reusable modeling experiences that are stored in an experience structure aligned with the curriculum. As learning progresses, OptClimber gradually adjusts the exposure to problems of different difficulty and refines the curriculum when persistent failures indicate that the current difficulty partition is too coarse. In this way, OptClimber transforms unstructured problem-solving trajectories into reliable modeling experience that is accumulated progressively along the curriculum.

The accumulated experience further supports model fine-tuning. OptClimber first uses these experiences to guide the generation and selection of solver-verified modeling trajectories, and then applies supervised fine-tuning to incorporate the underlying modeling knowledge into the model. The same difficulty structure is further carried into reinforcement learning, where training examples progress from more fundamental modeling problems toward increasingly complex ones, while solver feedback provides verifiable learning signals for solutions of different quality. The curriculum therefore connects experience construction, supervised fine-tuning, and reinforcement learning within a unified learning process, allowing the knowledge accumulated earlier to be progressively converted into stronger modeling capability. Experiments on six optimization modeling benchmarks show that the prompt-based OptClimber improves its backbone model by 18.3% on average, with larger gains on more challenging tasks. We further train OptClimber at multiple model scales, including 4B, 8B, and 14B, and observe consistent gains as model capacity increases. In particular, OptClimber-14B achieves 72.7% average accuracy across six benchmarks, surpassing the current state-of-the-art fine-tuned model.

Our main contributions are as follows:

- We propose OptClimber, a curriculum experience learning framework that jointly organizes optimization problems and reusable modeling knowledge by domain and difficulty.
- We develop a unified learning process that connects solver-verified experience construction, experience-guided supervised fine-tuning, and curriculum-guided reinforcement learning.
- Extensive experiments on six benchmarks show that prompt-based OptClimber improves its backbone by 18.3% on average, while further training consistently improves performance across 4B, 8B, and 14B models. In particular, OptClimber-14B achieves 72.7% average accuracy and surpasses the current state-of-the-art fine-tuned model.

2 Related Work

LLMs for Optimization Modeling Large language models have increasingly been used to translate natural-language problem descriptions into mathematical formulations and executable solver code. Prompt-based methods improve modeling through decomposition, multi-agent collaboration, search, and retrieval. CoE (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024) iteratively construct and debug optimization models, while MCTS-based methods (Astorga et al., 2025) explore alternative formulations and OptiTree (Liu et al., 2026a) retrieves structured modeling thoughts. More recent approaches further exploit solver feedback and reusable modeling knowledge. AlphaOPT (Kong et al., 2026) maintains a solver-verified insight library, EOM (Xie et al., 2026) distills correction histories into reusable mappings, OptSkills (Yang et al., 2026) learns workflow-level skills from problem archetypes, and AdmitOR (Lian et al., 2026) studies reliable experience admission without answer labels. In parallel, fine-tuning methods such as ORLM (Huang et al., 2025a), LLMOPT (Shu et al., 2025), OptMATH (Lu et al., 2025), and SIRL (Chen et al., 2026b) improve modeling capability through domain-specific training and solver-grounded super-

vision. OR-R1 (Ding et al., 2026) further combines supervised fine-tuning with group-relative reinforcement learning for data-efficient optimization modeling. Unlike these methods, OptClimber focuses on how optimization problems and the knowledge acquired from them can be jointly organized by domain and difficulty throughout both experience accumulation and model training.

Experience Accumulation and Curriculum Learning Experience learning allows LLMs to improve by extracting reusable knowledge from previous problem-solving trajectories. Buffer of Thoughts (Yang et al., 2024), ReasoningBank (Ouyang et al., 2026), and MemGen (Zhang et al., 2026a) represent and retrieve historical reasoning experience, while SkillWeaver (Zheng et al., 2025) and FLEX (Cai et al., 2025) further support continual skill refinement. Recent work moves toward more adaptive experience management. MemSkill (Zhang et al., 2026b) learns reusable memory operations from interaction traces, while EvolveMem (Liu et al., 2026c) jointly evolves stored knowledge and retrieval strategies. A complementary line of research studies how training data should be organized according to difficulty and model competence. CAMPUS (Li et al., 2025) adapts instruction curricula to evolving model capability, while recent reinforcement-learning methods such as E2H Reasoner (Parashar et al., 2026), CurES (Zeng et al., 2026), and Actor-Curator (Gu et al., 2026) organize or select training problems to improve learning efficiency. OptClimber connects these directions by using a shared domain-and-difficulty curriculum to organize both the problems from which modeling experience is distilled and the subsequent supervised and reinforcement learning processes, allowing accumulated experience to progressively support stronger optimization modeling capability.

3 Motivation: The Role of Difficulty in Knowledge Accumulation

To understand how problem difficulty affects knowledge accumulation, we conduct controlled studies under both prompt-based and fine-tuned settings. We compare three strategies for organizing problems during learning: Curriculum, which progressively introduces problems according to difficulty, Random, which removes difficulty-aware organization, and Hard-first, which prioritizes difficult problems at the early stage. In the prompt-based setting, we use DeepSeek-V3 to accumulate modeling experiences from 200 OR-Instruct problems under each strategy and evaluate the resulting experiences on IndustryOR, OptiMATH, and Resocratic. In the fine-tuned setting, we train a 4B model using the three corresponding strategies and evaluate the resulting models on the full test sets of the same benchmarks.

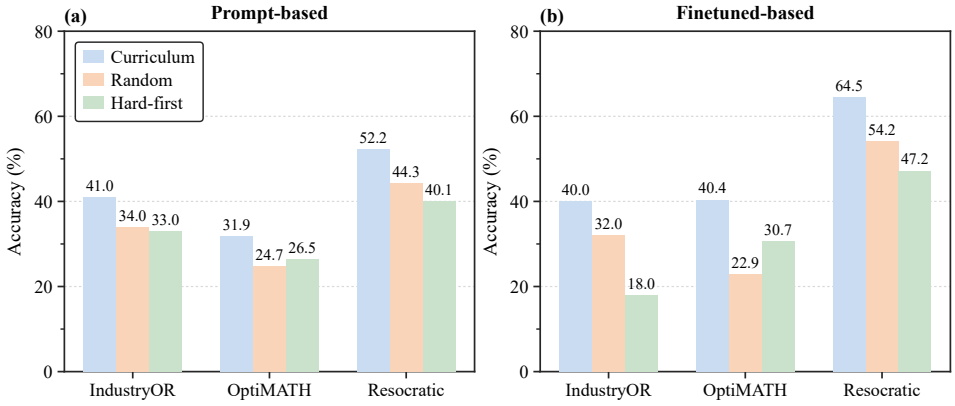

Figure 1: Comparison of different problem organization strategies under prompt-based and fine-tuned settings. In the prompt-based setting, modeling experiences are constructed from 200 OR-Instruct problems using DeepSeek-V3 and evaluated on IndustryOR, OptiMATH, and Resocratic. In the fine-tuned setting, a 4B model is trained under the corresponding strategies and evaluated on the full test sets of the same benchmarks. Curriculum consistently outperforms Random and Hard-first in both settings.

As shown in Figure 1, Curriculum consistently outperforms Random and Hard-first under both settings. In the prompt-based setting, easier problems tend to expose more localized and interpretable

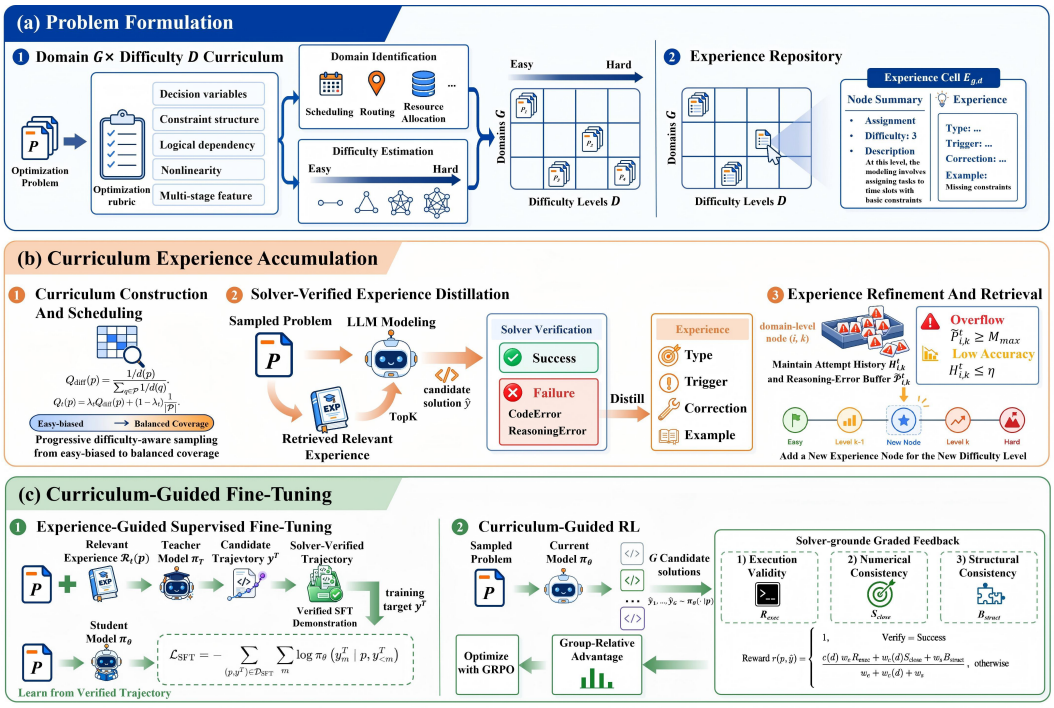

Figure 2: Overview of OptClimber. Optimization problems are first organized into a domain-and-difficulty curriculum. Following this curriculum, OptClimber progressively solves problems, verifies generated formulations with an optimization solver, and distills reusable modeling experience from failed trajectories while adaptively refining the curriculum. The accumulated experiences can be retrieved directly for prompt-based modeling or used to construct solver-verified demonstrations for supervised fine-tuning. The same difficulty-aware curriculum is further carried into reinforcement learning, where solver-grounded graded feedback progressively improves the trained policy.

failures, allowing reliable modeling knowledge to be accumulated before more complex problems are introduced. Starting directly from difficult problems, in contrast, often produces longer reasoning trajectories with multiple coupled errors, making useful knowledge harder to identify. A similar pattern is observed in the fine-tuned setting. Curriculum achieves an average accuracy of 48.3% across the three benchmarks, compared with 36.4% for Random and 32.0% for Hard-first. These results indicate that problem difficulty affects not only the reliability of experience accumulation, but also the effectiveness of subsequent model training. This motivates OptClimber to organize both modeling experience and the learning process through a shared difficulty-aware curriculum.

4 OptClimber: Curriculum Experience Learning

In this section, we introduce OptClimber, a unified curriculum experience learning framework for LLM-based optimization modeling. OptClimber organizes, accumulates, and utilizes modeling knowledge throughout optimization problem solving and model training. As illustrated in Figure 2, the framework consists of three stages. Initially, optimization problems are organized into a domain-and-difficulty curriculum. OptClimber then follows this curriculum to solve problems, verify the generated formulations with an optimization solver, and distill reusable modeling experiences from failed trajectories. These experiences can either be retrieved to enhance prompt-based modeling or used to construct solver-verified training trajectories, while the same curriculum further guides reinforcement learning.

4.1 Problem Formulation

Let $\mathcal{P} = \{(p_j, y_j)\}_{j=1}^{N}$ denote a collection of optimization modeling problems, where p_j is a natural-language problem description and y_j denotes its corresponding ground-truth target. Given a problem p, an LLM-based modeling policy π_θ generates a solution

$$\hat{y} \sim \pi_\theta(\cdot \mid p). \tag{1}$$

For each problem, we identify its modeling domain $g(p) \in \mathcal{G}$ and difficulty $d(p) \in \mathcal{D}$, and organize the problem collection into a domain-and-difficulty curriculum

$$\mathcal{C} = \mathcal{G} \times \mathcal{D}. \tag{2}$$

The domain and difficulty are determined using a structured optimization-modeling rubric that considers factors such as decision-variable complexity, constraint structure, logical dependencies, nonlinear transformations, and multi-stage characteristics. Domain captures shared modeling patterns across related problems, while difficulty characterizes the reasoning and formulation complexity required to construct a valid model. The detailed construction procedure and expert validation are provided in Appendix A.4.

Aligned with the curriculum, OptClimber maintains an experience repository

$$\mathcal{E} = \{E_{g,d}\}, \tag{3}$$

where each $E_{g,d}$ stores reusable modeling experiences associated with domain g and difficulty d. The experience accumulation and retrieval process is described in Section 4.2, while its use for supervised fine-tuning and curriculum-guided reinforcement learning is presented in Section 4.3.

4.2 Curriculum Experience Accumulation

In this section, we describe how OptClimber accumulates reusable modeling experience through curriculum-based problem scheduling (Section 4.2.1), solver-verified experience distillation (Section 4.2.2), and adaptive experience refinement and retrieval (Section 4.2.3).

4.2.1 Curriculum Construction and Scheduling

Based on the domain and difficulty identified in Section 4.1, OptClimber organizes problems into a domain-specific curriculum. Let

$$\mathcal{C}_i^t = \left(C_{i,1}^t, \ldots, C_{i,L_i^t}^t\right) \tag{4}$$

denote the curriculum hierarchy of domain g_i at step t, where each node contains problems with similar modeling difficulty.

To progressively expose the model to problems of different difficulty, we define an easy-biased distribution

$$Q_{\text{diff}}(p) = \frac{1/d(p)}{\sum_{q \in \mathcal{P}} 1/d(q)}, \tag{5}$$

and combine it with the uniform distribution:

$$Q_t(p) = \lambda_t Q_{\text{diff}}(p) + (1 - \lambda_t)\frac{1}{|\mathcal{P}|}. \tag{6}$$

The coefficient λ_t gradually decreases during accumulation, allowing the sampling distribution to evolve from easy-biased toward more balanced coverage of complex problems. Easier problems are sampled more frequently at the beginning to facilitate stable experience distillation, while the sampling probability of harder problems gradually increases as the easy bias weakens. This curriculum allows previously accumulated modeling experience to be reused when solving more complex problems, while failures on these harder cases further enrich the experience repository.

4.2.2 SOLVER-VERIFIED EXPERIENCE DISTILLATION

At step t, a problem $p \sim Q_t$ is matched to domain g_i and its current difficulty level

$$k = \ell_t(p). \tag{7}$$

The model generates a candidate solution $\hat{y}$, which is verified against the ground-truth target y using an optimization solver. We categorize the verification result as

$$R(p) \in \{\text{Success}, \text{CodeError}, \text{ReasoningError}\}. \tag{8}$$

A successful solution introduces no new experience. For a failed trajectory, OptClimber distills a reusable modeling experience

$$\kappa = \text{Distill}(p, \hat{y}, y, R(p)), \tag{9}$$

which summarizes the applicable condition, modeling error, correction principle, and a compact example. The resulting experience is associated with the corresponding domain-and-difficulty node $E^t_{i,k}$.

4.2.3 EXPERIENCE REFINEMENT AND RETRIEVAL

The initial difficulty partition may be too coarse for problems with complex reasoning requirements. OptClimber therefore uses solving outcomes to adaptively refine the curriculum and its aligned experience repository. For each domain-level node (i, k), we maintain a recent attempt history $H^t_{i,k}$ and a reasoning-error buffer $\widetilde{\mathcal{P}}^t_{i,k}$. When

$$|\widetilde{\mathcal{P}}^t_{i,k}| \geq M_{\max} \quad \text{or} \quad \text{Acc}(H^t_{i,k}) \leq \eta, \tag{10}$$

OptClimber inserts an intermediate difficulty node and updates the corresponding curriculum and experience structures. This allows the difficulty hierarchy to adapt to recurring failure patterns rather than relying on a fixed initial partition.

For a target problem p with $g(p) = g_i$ and $k = \ell_t(p)$, OptClimber retrieves the most relevant experiences from the matched and easier levels:

$$\mathcal{R}_t(p) = \text{TopK}\left(p, \bigcup_{r \leq k} E^t_{i,r}\right). \tag{11}$$

The retrieved experiences can then be incorporated into the modeling context,

$$\hat{y} \sim \pi_\theta\left(\cdot \mid p, \mathcal{R}_t(p)\right). \tag{12}$$

By restricting retrieval to the matched domain and the same or easier difficulty levels, OptClimber provides targeted guidance for optimization modeling while reducing the chance of introducing irrelevant or overly complex modeling patterns. Within this candidate set, similarity-based TopK selection further identifies the most relevant experiences.

4.3 CURRICULUM-GUIDED FINE-TUNING

In this section, we further convert the accumulated modeling experience into model capability through experience-guided supervised fine-tuning (Section 4.3.1) and curriculum-guided reinforcement learning (Section 4.3.2). The key idea is to preserve the knowledge structure established during experience accumulation. SFT transfers reusable experiences into verified modeling behaviors, while reinforcement learning continues to organize optimization problems according to the same difficulty-aware curriculum and progressively strengthens the resulting policy.

4.3.1 EXPERIENCE-GUIDED SUPERVISED FINE-TUNING

Although the accumulated experience can directly enhance prompt-based modeling through retrieval, such improvement still depends on providing external experience at inference time. We therefore use the experience repository to construct high-quality demonstrations that transfer the accumulated modeling knowledge into the model.

For each problem p, we retrieve the relevant experiences $\mathcal{R}_t(p)$ from the repository constructed in Section 4.2.3. A teacher model uses both the problem and the retrieved experience to generate a candidate trajectory:

$$y^T \sim \pi_T\left(\cdot \mid p, \mathcal{R}_t(p)\right). \tag{13}$$

The generated trajectory is retained only when it passes solver verification. This filtering step prevents the student model from learning errors that may appear in teacher-generated trajectories. It also ensures that accumulated experience is converted into executable and solver-consistent supervision. The resulting training set is

$$\mathcal{D}_{\mathrm{SFT}} = \left\{(p, y^T) \mid \mathrm{Verify}(p, y^T) = \mathrm{Success}\right\}. \tag{14}$$

The student policy π_θ is then optimized using the original problem as input:

$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{(p,y^T)\in\mathcal{D}_{\mathrm{SFT}}} \sum_m \log \pi_\theta\left(y_m^T \mid p, y_{<m}^T\right). \tag{15}$$

This design creates an asymmetric transfer process. The teacher has access to the accumulated experience when constructing the demonstration, whereas the student only observes the original problem. Consequently, the model cannot simply copy retrieved instructions during training. Instead, it must learn the modeling behavior reflected in solver-verified trajectories. Experience therefore serves as a structured supervision source that converts previously accumulated corrections and modeling principles into capabilities that can be applied without explicit retrieval.

4.3.2 Curriculum-Guided Reinforcement Learning

SFT provides a strong initialization by learning from verified demonstrations, but it primarily imitates successful trajectories and does not explicitly optimize how the model improves through its own problem-solving outcomes. We therefore further train the SFT policy π_{SFT} with curriculum-guided reinforcement learning.

A central design of OptClimber is that the curriculum used for experience accumulation is retained during reinforcement learning. Rather than randomly shuffling training problems, we organize them according to the curriculum distribution Q_t introduced in Section 4.2.1:

$$p_t \sim Q_t. \tag{16}$$

As λ_t decreases, the training distribution gradually expands from relatively fundamental problems toward more complex ones. This maintains continuity between experience accumulation and model optimization: the same difficulty structure that determines where reliable experience is acquired also determines how the model is subsequently trained.

For each problem p, the current policy samples a group of candidate solutions

$$\hat{y}_1, \ldots, \hat{y}_G \sim \pi_\theta(\cdot \mid p). \tag{17}$$

These solutions are evaluated using solver-grounded feedback. A binary reward is sufficient when a generated model is fully correct, but provides little guidance when a difficult formulation is only partially correct. We therefore use graded feedback that additionally measures execution validity, numerical closeness to the ground-truth target, and structural consistency. Let $\bar{d}(p) \in [0,1]$ denote the normalized difficulty of p. The reward is defined as

$$r(p,\hat{y}) = \begin{cases} 1, & \mathrm{Verify}(p,\hat{y}) = \mathrm{Success}, \\ c(\bar{d})\dfrac{w_{\mathrm{e}} R_{\mathrm{exec}} + w_{\mathrm{c}}(\bar{d}) S_{\mathrm{close}} + w_{\mathrm{s}} B_{\mathrm{struct}}}{w_{\mathrm{e}} + w_{\mathrm{c}}(\bar{d}) + w_{\mathrm{s}}}, & \text{otherwise}, \end{cases} \tag{18}$$

where R_{exec} measures whether the generated solver program can be executed, S_{close} measures its numerical proximity to the target, and B_{struct} measures formulation consistency when structural information is available. The functions $w_{\mathrm{c}}(\bar{d})$ and $c(\bar{d})$ increase with problem difficulty, allowing harder problems to provide more informative intermediate feedback while ensuring that an incorrect solution never receives the reward of a correct one.

Given the rewards of the G candidate solutions, we compute the group-relative advantage

$$A_i = r_i - \frac{1}{G}\sum_{j=1}^{G} r_j, \tag{19}$$

and optimize the policy using GRPO. The curriculum and reward play complementary roles. The curriculum controls which modeling capabilities are challenged at different stages of learning, while solver-grounded graded feedback indicates how close each attempt is to a valid formulation. Together, they allow OptClimber to move beyond imitation of verified demonstrations and progressively improve on complex optimization modeling problems.

5 Experiments

5.1 Experimental Setup

Benchmarks. We evaluate OptClimber on six optimization modeling benchmarks: NL4Opt (Ramamonjison et al., 2023), MAMO EasyLP (Huang et al., 2025b), MAMO ComplexLP (Huang et al., 2025b), IndustryOR (Huang et al., 2025a), OptiBench (Yang et al., 2025b), and OptMATH (Lu et al., 2025). These benchmarks span diverse optimization modeling tasks and vary substantially in problem scale, structural complexity, and domain context, ranging from relatively standard linear programs to industrial and combinatorial optimization problems. Detailed dataset statistics, preprocessing procedures, and evaluation-set construction are provided in Appendix A.

Baselines. We compare OptClimber with three groups of methods: general LLMs, prompt-based methods, and fine-tuned models. General LLMs include GPT-4o (OpenAI, 2024), OpenAI-o3 (OpenAI, 2025), DeepSeek-V3 (DeepSeek-AI, 2024), and DeepSeek-R1 (DeepSeek-AI, 2025). Prompt-based baselines include CoT Wei et al. (2022), CoE Xiao et al. (2024), OptiMUS AhmadiTeshnizi et al. (2024), MCTS Astorga et al. (2025), AlphaOPT Kong et al. (2026), and OptiTree Liu et al. (2026a). Fine-tuned baselines include ORLM Huang et al. (2025a), Step-Opt Zhou et al. (2026), LLMOPT Shu et al. (2025), OptMATH Lu et al. (2025), and SIRL Chen et al. (2026b), for which the corresponding backbone models are explicitly reported. Further baseline descriptions and configurations are provided in Appendix A.

Implementation and Evaluation. We report solving accuracy on each benchmark. Generated solutions are evaluated through solver execution and objective matching under a unified evaluation protocol. For prompt-based methods, DeepSeek-V3 is used as the backbone whenever applicable. Prompt-based OptClimber constructs its curriculum and experience repository from 200 OR-Instruct problems (Huang et al., 2025a), and OptiTree is evaluated under the same construction budget for a controlled comparison. For fine-tuned OptClimber, we train Qwen3-4B, Qwen3-8B, and Qwen3-14B (Yang et al., 2025a) using experience-guided supervised fine-tuning followed by curriculum-guided reinforcement learning. Complete training configurations, evaluation protocols, and implementation details are provided in Appendix A. The complete algorithms and prompts used by OptClimber are provided in Appendices C and E, respectively.

5.2 Main Results

Table 1 compares OptClimber with general LLMs, prompt-based methods, and fine-tuned models across the six benchmarks.

Prompt-based modeling. OptClimber achieves the strongest overall performance among prompt-based methods and consistently improves its underlying backbone across the evaluated benchmarks. More importantly, its advantage becomes clearer on problems that require more involved modeling decisions, suggesting that the accumulated experiences provide useful guidance beyond standard optimization patterns. OptClimber also maintains a clear advantage over OptiTree under the same experience-construction budget. This controlled comparison highlights an important distinction between merely collecting previous solutions and organizing them into a difficulty-aware learning process. By progressively accumulating and retrieving experiences along the curriculum, OptClimber

Table 1: Accuracy (%) on six optimization modeling benchmarks. * denotes results reported for the corresponding fine-tuned models. Best results within the prompt-based and fine-tuned groups are highlighted in bold.

Method	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH
General LLMs						
GPT-4o	79.2	81.3	41.2	38.0	48.4	17.5
OpenAI-o3	**86.8**	79.2	53.5	43.0	62.8	43.3
DeepSeek-V3	79.8	**88.3**	50.2	37.0	55.1	**44.0**
DeepSeek-R1	82.4	87.2	**67.9**	**45.0**	**67.2**	34.9
Prompt-based Methods						
CoT	74.0	82.9	40.7	35.0	53.1	21.1
CoE	79.2	85.9	43.1	33.0	55.2	24.1
OptiMUS	80.6	87.1	45.2	36.0	58.8	32.5
MCTS	89.6	88.0	51.6	46.0	67.9	38.6
AlphaOPT	93.0	92.5	61.6	49.0	69.9	39.2
OptiTree	97.1	92.4	75.9	47.0	69.4	49.9
OptClimber	**97.5**	**94.1**	**83.7**	**61.0**	**72.2**	**55.4**
Fine-tuned Models						
ORLM-Llama3-8B*	85.7	82.3	37.4	38.0	51.1	2.6
Step-Opt-Llama3-8B*	84.5	85.3	61.6	36.4	–	–
LLMOPT-Qwen2.5-14B*	80.3	89.5	44.1	29.0	53.8	12.5
OptMATH-Qwen2.5-7B*	94.7	86.5	51.2	20.0	57.9	24.4
OptMATH-Qwen2.5-32B*	95.9	89.9	54.1	31.0	66.1	34.7
SIRL-Qwen2.5-7B*	96.3	91.7	51.7	33.0	58.0	30.5
SIRL-Qwen2.5-32B*	**98.0**	94.6	61.1	42.0	**67.4**	45.8
OptClimber-Qwen3-4B	97.1	93.6	57.8	41.0	64.5	40.4
OptClimber-Qwen3-8B	**98.0**	94.6	68.7	49.0	65.6	45.6
OptClimber-Qwen3-14B	**98.0**	**95.0**	**75.9**	**52.0**	66.8	**48.7**

enables the model to reuse modeling knowledge in a more structured and transferable manner. Further analyses of experience quality, cross-difficulty transfer, matched-budget comparisons, and robustness are provided in Appendix B.

Fine-tuned modeling. The advantage of OptClimber extends beyond prompt-based experience retrieval. Across different model scales, curriculum-guided fine-tuning consistently yields strong overall performance and establishes a new state of the art among the evaluated fine-tuned models. In particular, the 8B model already surpasses substantially larger fine-tuned baselines, while further scaling the backbone continues to improve performance. This suggests that the proposed framework is not tied to a particular model capacity and that the accumulated modeling knowledge can be effectively absorbed during training.

The gains are especially evident on benchmarks with more complex modeling structures, whereas performance on relatively saturated tasks remains competitive. This behavior supports our central hypothesis that difficulty-aware knowledge accumulation is most valuable when successful modeling requires combining multiple constraints, dependencies, and modeling decisions. Together with the prompt-based results, these findings show that the same curriculum structure can support both external experience reuse and subsequent capability improvement through fine-tuning. Additional scaling results and reinforcement-learning analyses are provided in Appendix B and Appendix B.4.

6 Conclusion

We present OptClimber, a curriculum experience learning framework for LLM-based optimization modeling. By organizing problems by difficulty, accumulating solver-verified experience, and extending the same curriculum into fine-tuning, OptClimber improves both prompt-based modeling and trained model performance across diverse optimization benchmarks.

AI USE STATEMENT

In this work, we used generative AI tools for translation and language polishing to improve the grammar, clarity, and readability of the manuscript. We did not use generative AI for other tasks requiring disclosure, such as developing research methods, formulating scientific claims, designing experiments, implementing the proposed methods, or interpreting results. All AI-assisted content was reviewed by the authors, who take full responsibility for the final manuscript.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 577–596, 2024.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of mathematical optimization models using llms. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 1864–1886, 2025.

Zhicheng Cai, Xinyuan Guo, Yu Pei, Jiangtao Feng, Jinsong Su, Jiangjie Chen, Ya-Qin Zhang, Wei-Ying Ma, Mingxuan Wang, and Hao Zhou. Flex: Continuous agent evolution via forward learning from experience. *arXiv preprint arXiv:2511.06449*, 2025.

Yitian Chen, Cheng Cheng, Yinan Sun, Zi Ling, and Dongdong Ge. Opt-engine: Benchmarking the limits of llms in optimization modeling via complexity scaling. In *Forty-third International Conference on Machine Learning*, 2026a.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed rl: Grounding large language models for authentic optimization modeling. *Advances in Neural Information Processing Systems*, 38:106027–106069, 2026b.

Ziang Chen, Jialin Liu, Xinshang Wang, and Wotao Yin. On representing mixed-integer linear programs by graph neural networks. In *The Eleventh International Conference on Learning Representations*, 2023.

Ziang Chen, Jialin Liu, Xiaohan Chen, Xinshang Wang, and Wotao Yin. Rethinking the capacity of graph neural networks for branching strategy. *Advances in neural information processing systems*, 37:123991–124024, 2024.

DeepSeek-AI. DeepSeek-V3 technical report, 2024.

DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

Zezhen Ding, Zhen Tan, Jiheng Zhang, and Tianlong Chen. 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*, volume 40, pp. 228–236, 2026.

Zhengyao Gu, Jonathan Light, Raul Astudillo, Ziyu Ye, Langzhou He, Henry Peng Zou, Wei Cheng, Santiago Paternain, Philip S Yu, and Yisong Yue. Actor-curator: Co-adaptive curriculum learning via policy-improvement bandits for rl post-training. *arXiv preprint arXiv:2602.20532*, 2026.

Yiliu He, Tianle Li, Binghao Ji, Zhiyuan Liu, and Di Huang. Evoopt-llm: Evolving industrial optimization models with large language models. *arXiv preprint arXiv:2602.01082*, 2026.

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, 2025a.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710, 2025b.

Xitai Jiang, Zihan Tang, Wenze Lin, Yang Yue, Shenzhi Wang, and Gao Huang. From reasoning chains to verifiable subproblems: Curriculum reinforcement learning enables credit assignment for llm reasoning. *arXiv preprint arXiv:2605.22074*, 2026.

Maurice G Kendall and B Babington Smith. The problem of m rankings. *The annals of mathematical statistics*, 10(3):275–287, 1939.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, et al. Alphaopt: Formulating optimization programs with self-improving llm experience library. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2389–2400, 2026.

Yangning Li, Tingwei Lu, Yinghui Li, Yankai Chen, Wei-Chieh Huang, Wenhao Jiang, Hui Wang, Hai-Tao Zheng, and S Yu Philip. Teaching according to talents! instruction tuning llms with competence-aware curriculum learning. In *EMNLP (Findings)*, pp. 11724–11741, 2025.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Forty-third International Conference on Machine Learning*, 2026.

Junbo Jacob Lian, Huiling Chen, Hanzhang Qin, and Chung-Piaw Teo. Admission without answers: Label-free certification and experience learning for llm-based optimization modeling. *arXiv preprint arXiv:2608.15565*, 2026.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. Optitree: Hierarchical thoughts generation with tree search for llm optimization modeling. *Advances in Neural Information Processing Systems*, 38:120713–120781, 2026a.

Haoyang Liu, Jie Wang, Yufei Kuang, Xijun Li, Yongdong Zhang, and Feng Wu. Promoting generalization for exact combinatorial solvers via adversarial instance augmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026b.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. Evolvemem: Self-evolving memory architecture via autoresearch for llm agents. *arXiv preprint arXiv:2605.13941*, 2026c.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 40769–40802, 2025.

OpenAI. GPT-4o system card, 2024.

OpenAI. OpenAI o3 and o4-mini system card, 2025.

Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pp. 94327–94354, 2026.

Shubham Parashar, Shurui Gui, Xiner Li, Hongyi Ling, Sushil Vemuri, Blake Olson, Eric Li, Yu Zhang, James Caverlee, Dileep Kalathil, et al. Curriculum reinforcement learning from easy to hard tasks improves llm reasoning. In *International Conference on Learning Representations*, volume 2026, pp. 156944–156972, 2026.

Tianle Pu, Zijie Geng, Haoyang Liu, Shixuan Liu, Jie Wang, Li Zeng, Chao Chen, and Changjun Fan. Rome: Domain-robust mixture-of-experts for milp solution prediction across domains. *Advances in neural information processing systems*, 38:33633–33661, 2026a.

Tianle Pu, Jianing Li, Yingying Gao, Shixuan Liu, Zijie Geng, Haoyang Liu, Chao Chen, and Changjun Fan. Coco-milp: Inter-variable contrastive and intra-constraint competitive milp solution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 24882–24890, 2026b.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pp. 189–203. PMLR, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Xiang Shu, Hong Qian, Xingyu Lu, JUN ZHOU, Aimin Zhou, Yang Yu, et al. Llmopt: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, volume 2025, pp. 101580–101606, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *International Conference on Learning Representations*, volume 2024, pp. 48519–48537, 2024.

Wantong Xie, Yinghao Chen, Yi-Xiang Hu, Feng Wu, Jieyang Xu, Sijia Zhang, and Xiangyang Li. Escaping the sisyphus dilemma: Experience replay for robust text-to-optimization modeling. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 14100–14116, 2026.

Xianchao Xiu, Jianhao Li, Jun Fan, and Wanquan Liu. Large language models for operations research: A comprehensive survey. *arXiv preprint arXiv:2605.20849*, 2026.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025a.

Haochen Yang, Ke Zhao, Mengyuan Ma, Xingyu Lu, Xiangfeng Wang, and Hong Qian. Optskills: Learning generalizable optimization skills from problem archetypes via cluster-based distillation. *arXiv preprint arXiv:2605.29829*, 2026.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544, 2024.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Shi Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *International Conference on Learning Representations*, volume 2025, pp. 24726–24759, 2025b.

Haofeng Yuan, Jianing Peng, Jieyi Bi, Ni Zhang, Shiji Song, and Zhiguang Cao. Formuevo: Llm-guided evolution for discovering solver-efficient mixed-integer programming formulations. *arXiv preprint arXiv:2608.23353*, 2026.

Yongcheng Zeng, Zexu Sun, Bokai Ji, Erxue Min, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Haifeng Zhang, Xu Chen, and Jun Wang. Cures: From gradient analysis to efficient curriculum learning for reasoning llms. In *International Conference on Learning Representations*, volume 2026, pp. 1531–1556, 2026.

Guibin Zhang, Muxin Fu, and Shuicheng Yan. Memgen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, volume 2026, pp. 22555–22588, 2026a.

Haozhen Zhang, Quanyu Long, Jianzhu Bao, Tao Feng, Weizhi Zhang, Haodong Yue, and Wenya Wang. Memskill: Learning and evolving memory skills for self-evolving agents. *arXiv preprint arXiv:2602.02474*, 2026b.

Ruiqing Zhao, Fengzhi Li, Yuan Zuo, Rui Liu, Yansong Liu, Yunfei Ma, Fanyu Meng, and Junlan Feng. Strategy-aware optimization modeling with reasoning llms. In *Forty-third International Conference on Machine Learning*, 2026.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 3: system demonstrations)*, pp. 400–410, 2024.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. Steporlm: A self-evolving framework with generative process supervision for operations research language models. In *International Conference on Learning Representations*, volume 2026, pp. 6914–6940, 2026.

A Experimental Details

A.1 Datasets and Evaluation Protocol

Benchmark Statistics and Preprocessing. We evaluate OptClimber on six established optimization modeling benchmarks: NL4Opt (Ramamonjison et al., 2023), MAMO EasyLP and MAMO ComplexLP (Huang et al., 2025b), IndustryOR (Huang et al., 2025a), OptiBench (Yang et al., 2025b), and OptMATH (Lu et al., 2025). Together, these benchmarks cover a broad spectrum of optimization modeling difficulty, ranging from relatively standard linear programs to problems involving more complex constraint interactions, logical dependencies, integer decisions, and application-specific modeling structures.

NL4Opt originates from the NeurIPS optimization modeling competition and primarily contains relatively compact linear programming problems. MAMO provides two evaluation sets with different levels of modeling complexity: EasyLP contains comparatively direct linear programming instances, whereas ComplexLP contains problems with more involved constraint structures and reasoning requirements. IndustryOR consists of optimization problems derived from industrial and application-oriented scenarios. OptiBench evaluates optimization modeling across a broader collection of problem structures and was designed to provide a more challenging assessment of model formulation ability. OptMATH contains mathematically structured optimization problems and places greater emphasis on complex formulation and combinatorial reasoning. Table 2 summarizes the evaluation sets used in our experiments.

Table 2: Statistics of the optimization modeling benchmarks used for evaluation. The number of instances corresponds to the evaluation files used in our experiments.

Benchmark	# Instances	Characteristics
NL4Opt	245	Standard linear programming
MAMO EasyLP	652	Relatively simple LP modeling
MAMO ComplexLP	211	Complex and coupled LP modeling
IndustryOR	100	Application-oriented industrial OR
OptiBench	605	Diverse optimization modeling problems
OptMATH	193	Mathematical and combinatorial modeling
Total	2,006	

We use the released English problem descriptions and reference targets and normalize the benchmark records into a common evaluation format. For the benchmark instances for which corrected reference targets are available, we use the corrected versions throughout our in-house evaluation. These corrections affect only the reference targets and leave the natural-language problem descriptions unchanged. Specifically, the corrected evaluation files revise 2 instances in NL4Opt, 14 instances in MAMO EasyLP, and 24 instances in MAMO ComplexLP, while IndustryOR, OptiBench, and OptMATH remain unchanged. The corrections mainly address inconsistent or inaccurate released objective values, including suboptimal reference solutions, numerical precision issues, and incorrectly labeled solver outcomes. Unless explicitly stated otherwise, all results produced by our evaluation pipeline use the same benchmark versions.

To reduce potential train–test contamination, evaluation problems are excluded from the data used for experience construction and model training. We perform text normalization before overlap detection and remove duplicated training instances that match evaluation problems. For the fine-tuning data construction pipeline, we additionally apply near-duplicate screening before solver verification. The evaluation sets themselves are never used to construct the experience repository, supervised fine-tuning trajectories, or reinforcement-learning training data.

Evaluation Protocol. Following prior work on LLM-based optimization modeling (Huang et al., 2025a; Shu et al., 2025; Lu et al., 2025; Chen et al., 2026b), we evaluate a generated solution by executing its solver program and comparing the resulting objective value with the reference target. Let v denote the objective value obtained from the generated model and $v^\star$ denote the reference objective value. For instances with a finite reference solution, a prediction is considered correct

when

$$\frac{|v - v^\star|}{|v^\star|} \leq 10^{-6}, \tag{20}$$

with an absolute tolerance of 10^{-6} used when the reference objective is sufficiently close to zero. A generated formulation must be executable by the optimization solver before its objective value is considered for scoring. For instances whose reference status is infeasible or unbounded, a prediction is counted as correct only when the solver returns the corresponding infeasibility or unboundedness status.

Following the evaluation convention adopted by recent optimization-modeling benchmarks (Lu et al., 2025; Chen et al., 2026b), we also account for benchmark instances in which variable-type specification is ambiguous. For an executable program that does not initially match the reference objective, we perform one variable-type consistency check by switching the relevant continuous and integer declarations and re-executing the formulation. The prediction is counted as correct if the resulting model matches the reference target under the same numerical tolerance. The same procedure is used for all OptClimber variants evaluated under this protocol.

For a benchmark $\mathcal{B}$, solving accuracy is computed as

$$\mathrm{Acc}(\mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{p \in \mathcal{B}} \mathbb{I}\big[\mathrm{Correct}(p)\big]. \tag{21}$$

The overall score reported in the paper is the unweighted macro-average over the six benchmarks,

$$\mathrm{Avg} = \frac{1}{6} \sum_{j=1}^{6} \mathrm{Acc}(\mathcal{B}_j), \tag{22}$$

so that benchmarks with different numbers of test instances contribute equally to the final average. Unless otherwise specified, the same solver-based correctness criterion is used throughout the prompt-based, supervised fine-tuning, and reinforcement learning experiments.

A.2 Baseline Details

We compare OptClimber with three groups of methods that represent different ways of providing or acquiring optimization modeling knowledge: general-purpose LLMs, prompt-based modeling methods, and models specifically fine-tuned for optimization modeling. The corresponding backbone model is reported explicitly for fine-tuned methods in the main results.

General LLMs We first evaluate representative general-purpose and reasoning-oriented LLMs without additional optimization-specific experience construction or parameter updates.

GPT-4o (OpenAI, 2024) is a general-purpose multimodal model with strong instruction-following and reasoning capabilities. We use it as a representative commercial LLM to evaluate the extent to which general language and reasoning capabilities transfer to optimization modeling.

OpenAI-o3 (OpenAI, 2025) is a reasoning-oriented model designed for complex multi-step problem solving. Its inclusion provides a strong reasoning-model baseline for assessing whether general reasoning improvements alone are sufficient for reliable optimization formulation.

DeepSeek-V3 (DeepSeek-AI, 2024) is a large general-purpose open-weight model with strong coding and reasoning capabilities. It serves both as a standalone baseline and as the common backbone for prompt-based methods whenever applicable, allowing us to isolate the contribution of different inference-time modeling strategies.

DeepSeek-R1 (DeepSeek-AI, 2025) is a reasoning model trained to perform extended multi-step reasoning through reinforcement learning. We include it to examine the effectiveness of reasoning-specialized models on optimization problems that require jointly constructing variables, objectives, constraints, and executable solver programs.

Prompt-Based Methods Prompt-based methods improve optimization modeling without updating the underlying model parameters. They differ primarily in how they organize reasoning, search over candidate formulations, interact with solvers, or reuse previous modeling knowledge.

Chain-of-Thought (CoT) (Wei et al., 2022) elicits explicit intermediate reasoning before producing the final solution. We use CoT as a basic reasoning-enhanced prompting baseline without additional search, retrieval, or external experience.

Chain-of-Experts (CoE) (Xiao et al., 2024) decomposes optimization modeling into multiple specialized roles that collaboratively perform problem understanding, mathematical formulation, code generation, debugging, and verification. This multi-agent decomposition is intended to reduce the burden of solving a complex modeling problem in a single generation.

OptiMUS (AhmadiTeshnizi et al., 2024) uses a modular LLM-agent workflow to construct mathematical models and solver programs, and iteratively revises them using execution and debugging feedback. Compared with direct prompting, it places greater emphasis on closing the loop between formulation, solver execution, and model correction.

MCTS (Astorga et al., 2025) formulates automated optimization modeling as a search problem and applies Monte Carlo Tree Search to explore alternative modeling decisions. Candidate variables, objectives, and constraints are progressively generated and evaluated rather than being committed to in a single generation.

AlphaOPT (Kong et al., 2026) improves optimization modeling through a self-improving experience library. It extracts reusable modeling insights from solver-verified problem-solving trajectories and iteratively refines the stored knowledge to guide subsequent problems.

OptiTree (Liu et al., 2026a) organizes optimization modeling knowledge into a hierarchical tree of problem structures and modeling thoughts. For a new problem, related modeling knowledge is retrieved from the tree and used to guide hierarchical decomposition and formulation.

For controlled prompt-based comparisons, we use DeepSeek-V3 as the underlying model whenever the method supports backbone replacement. In particular, OptClimber and OptiTree are evaluated with the same 200-problem OR-Instruct experience-construction budget in the main comparison, preventing differences in the amount of historical modeling data from confounding the comparison. Additional matched-budget analysis is provided in Appendix B.5. For AlphaOPT, we follow its released evaluation setting and provide an additional comparison on its released test sets in Appendix B.5.

Fine-Tuned Models We further compare against optimization-specific models that incorporate modeling knowledge into model parameters through supervised training, process supervision, or reinforcement learning.

ORLM-Llama3-8B (Huang et al., 2025a) trains a specialized operations research language model using OR-Instruct, a large-scale instruction dataset for optimization modeling. The model learns to translate natural-language problem descriptions into mathematical formulations and executable solver code through domain-specific supervised fine-tuning.

Step-Opt-Llama3-8B (Zhou et al., 2026) introduces process-level supervision for optimization modeling. Rather than relying only on final solver outcomes, it uses feedback on intermediate modeling steps to improve the reliability of the formulation process and reduce errors that may be hidden by a superficially correct final objective value.

LLMOPT-Qwen2.5-14B (Shu et al., 2025) learns optimization modeling through a unified representation of problem definitions and large-scale instruction tuning. It aims to improve generalization across heterogeneous optimization problems by jointly learning structured problem understanding, mathematical formulation, and solver-code generation.

OptMATH-Qwen2.5-7B/32B (Lu et al., 2025) uses a scalable bidirectional data-synthesis framework to construct optimization modeling training examples. Its synthetic data pipeline connects optimization models with corresponding natural-language descriptions, enabling specialized models to learn from substantially enlarged solver-verifiable training corpora. We report both the 7B and 32B variants to characterize performance across model scales.

SIRL-Qwen2.5-7B/32B (Chen et al., 2026b) introduces solver-informed reinforcement learning for optimization modeling. Generated solver programs are executed and evaluated using solver-derived signals, allowing the policy to learn from verifiable outcomes rather than relying exclusively on static supervised demonstrations. We include both its 7B and 32B variants.

Finally, we train **OptClimber-Qwen3-4B**, **OptClimber-Qwen3-8B**, and **OptClimber-Qwen3-14B** using the experience-guided supervised fine-tuning and curriculum-guided reinforcement learning procedure described in Section 4. These three model sizes are evaluated under the same OptClimber training framework to study whether curriculum experience learning remains effective as model capacity increases.

For previously released fine-tuned baselines, we use the results associated with the corresponding backbone models and official evaluation settings, as indicated by the superscript in the main results table. We explicitly retain the backbone name for each model because both the training method and underlying model capacity can substantially affect optimization modeling performance.

A.3 Implementation Details

Prompt-Based OptClimber For the prompt-based setting, we use DeepSeek-V3 as the default backbone. Unless otherwise specified, the curriculum and experience repository are constructed from 200 randomly sampled problems from OR-Instruct (Huang et al., 2025a). We first identify the domain and initial difficulty of each problem and organize the resulting problems into the curriculum described in Section 4.2. During construction, problems are sampled according to the time-varying curriculum distribution Q_t. Each sampled problem is solved with the currently retrieved experiences, and the generated solver program is executed and compared with the reference target. Failed trajectories are then distilled and used to update the corresponding experience node.

We construct the experience repository in batches of 16 problems and run up to 30 construction iterations. Intermediate repositories are saved during this process, and the main prompt-based results use the checkpoint after 20 iterations. The initial maximum difficulty level is set to $L_{\max} = 10$. For inference, a target problem is first matched to its domain and current difficulty level, after which relevant experiences are retrieved from the matched and easier difficulty levels. We use a retrieval budget of $K_{\text{ret}} = 1$ in the default setting. Sensitivity to the construction schedule, curriculum granularity, retrieval budget, and refinement threshold is studied in Appendix B.3 and Appendix B.6.

Experience-Guided Supervised Fine-Tuning We use the accumulated experience repository to construct solver-verified supervised trajectories for Qwen3-4B, Qwen3-8B, and Qwen3-14B (Yang et al., 2025a). For each training problem, the teacher is provided with the problem together with the retrieved modeling experiences and generates a complete modeling trajectory containing the formulation and executable Gurobi code. Multiple candidates can be generated when necessary, and only trajectories that satisfy the solver-based verification criterion are retained. This rejection procedure prevents incorrect experience-conditioned generations from being directly used as supervision.

Importantly, the retrieved experience is used only by the teacher during data construction. The student receives the original problem without the retrieved experience and is trained to reproduce the verified trajectory. This asymmetric design forces the modeling principles contained in the experience repository to be absorbed into the model rather than copied from additional context at inference time.

We perform supervised fine-tuning with LLaMA-Factory (Zheng et al., 2024) using LoRA. Across model sizes, we use rank 64, LoRA scaling factor 128, a learning rate of 1×10^{-4}, cosine learning rate decay, and two training epochs. The effective global batch size is kept fixed at 64. After training, the LoRA adapters are merged into the backbone weights, and the resulting model is used to initialize reinforcement learning. Further statistics of the constructed SFT data are reported in Appendix B.2.

Curriculum-Guided Reinforcement Learning Reinforcement learning starts from the corresponding SFT model and is implemented with GRPO (Shao et al., 2024). We retain the curriculum used during experience accumulation rather than randomly shuffling the training problems. Specifically, the training sequence is sampled according to Q_t, with the curriculum coefficient linearly decreasing from $\lambda_0 = 0.9$ to $\lambda_{\min} = 0.1$. The resulting sequence is consumed without further shuffling, so earlier updates are biased toward more fundamental problems while increasingly difficult problems receive greater exposure as training progresses.

For each problem, we sample a group of $G = 16$ candidate solutions and execute their solver programs using Gurobi. Rewards follow the solver-grounded graded formulation in Section 4.3.2.

A fully correct solution receives the maximum reward, while unsuccessful solutions receive intermediate feedback according to execution validity, numerical proximity to the reference objective, and, when available, structural consistency with the reference formulation. Difficulty-dependent weighting provides denser feedback for harder problems while keeping the reward of every incorrect solution below that of a verified correct solution.

We optimize the policy with a learning rate of 3×10^{-6} and use a small KL regularization coefficient of 0.005 to limit excessive deviation from the SFT initialization. Group-relative advantages are computed within the candidate set for each problem. More detailed reward ablations and the effect of alternative curriculum orderings are provided in Appendix B.4.

Training and Inference Configuration All fine-tuned OptClimber models use the Qwen3 family as their backbone, with 4B, 8B, and 14B parameter scales. SFT is implemented with LLaMA-Factory, while reinforcement learning uses `verl` together with vLLM for batched rollout generation. Gurobi is used throughout data verification, reinforcement learning, and evaluation to provide a common executable verification environment.

The model is instructed to separate its mathematical formulation from the executable solver program using structured `<model>` and `<python>` fields. We disable the backbone's native thinking mode so that the same output contract is maintained across SFT, reinforcement learning, and evaluation. During evaluation, generated solver programs are executed in an isolated solver process, and correctness is determined using the unified protocol described in Appendix A.1. Unless otherwise specified, decoding is deterministic for the initial generation. For the in-house evaluation of trained OptClimber models, non-executable programs can be revised using solver feedback for up to three repair rounds, after which the final executable solution is scored according to the same solver-based correctness criterion.

A.4 Difficulty Construction and Validation

Optimization Modeling Difficulty Rubric OptClimber estimates modeling difficulty from the structural requirements of an optimization problem rather than from superficial properties such as description length. Given a problem p, an LLM-based assessor first identifies its optimization domain $g(p)$ and assigns an initial raw difficulty score $\widetilde{d}(p)$ using a structured rubric. The rubric evaluates five aspects that commonly determine the difficulty of translating a natural-language description into a correct mathematical formulation.

- **Decision-variable complexity.** This dimension measures the complexity of the decision representation, including the number and types of variables and whether indexed, integer, binary, or auxiliary variables are required.
- **Constraint structure and coupling.** This dimension reflects the complexity of the constraint system and the extent to which decisions are coupled across entities, resources, periods, or stages. Independent capacity or balance constraints are considered simpler than strongly coupled constraints involving multiple decisions.
- **Logical dependency.** This dimension measures whether feasibility depends on conditional or combinatorial logic. Problems involving implications, mutual exclusion, precedence, activation decisions, or other conditional relationships are assigned higher difficulty than problems containing only unconditional algebraic constraints.
- **Nonlinearity and transformation.** This dimension captures whether the original problem can be modeled directly using linear expressions or requires additional transformations, such as linearization, piecewise modeling, min/max reformulation, or auxiliary-variable constructions.
- **Uncertainty and staging.** This dimension measures whether decisions involve multiple stages, scenarios, uncertain parameters, or dependencies across different decision periods. Deterministic single-stage problems receive lower scores than problems requiring scenario-based or sequential reasoning.

The assessor considers these dimensions jointly and returns an integer difficulty score between 1 and 10. The score is used as an initial curriculum prior rather than an absolute measure of problem

hardness. In particular, OptClimber does not assume that a problem remains permanently assigned to its initial level. During experience accumulation, problems are dynamically matched to the evolving domain-specific hierarchy, and intermediate levels can be introduced when repeated failures indicate that the current partition is too coarse. The complete difficulty-assessment prompt is provided in Appendix E.1.

Domain-Wise Difficulty Normalization Raw difficulty scores are not necessarily directly comparable across optimization domains. For example, a moderately difficult scheduling problem may require substantially different modeling operations from a problem with a similar raw score in resource allocation. We therefore normalize difficulty within each domain before constructing the initial curriculum.

Let

$$\mathcal{P}_i = \{p \in \mathcal{P} \mid g(p) = g_i\} \tag{23}$$

denote the subset of problems belonging to domain g_i. For each $p \in \mathcal{P}_i$, we transform the raw score $\widetilde{d}(p)$ into

$$d(p) = 1 + (L_{\max} - 1)\frac{\widetilde{d}(p) - \min_{q\in\mathcal{P}_i} \widetilde{d}(q)}{\max_{q\in\mathcal{P}_i} \widetilde{d}(q) - \min_{q\in\mathcal{P}_i} \widetilde{d}(q) + \epsilon}, \tag{24}$$

where ϵ is a small constant for numerical stability and $L_{\max} = 10$ in our implementation. The initial discrete curriculum level is then obtained as

$$\ell_0(p) = \min\left\{L_{\max}, \max\left\{1, \lfloor d(p) \rceil\right\}\right\}, \tag{25}$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

This domain-wise normalization preserves relative difficulty within each modeling family while preventing domains with systematically different raw score ranges from dominating the curriculum. The resulting $\ell_0(p)$ is used to initialize the domain-specific hierarchy and the easy-biased sampling distribution. Subsequent storage, retrieval, and curriculum refinement use the dynamic level $\ell_t(p)$ determined by the current hierarchy rather than treating $\ell_0(p)$ as a fixed label. This distinction also makes the method less sensitive to moderate errors in the initial difficulty estimate, as further examined in Appendix B.6.

Expert Annotation and Reliability We additionally validate the difficulty construction with human annotations. Five PhD-level researchers with expertise in operations research independently assess the relative modeling difficulty of problems within each benchmark. Annotators follow the same five criteria used by the automatic difficulty rubric: decision-variable complexity, constraint coupling, logical dependency, nonlinearity or required transformation, and uncertainty or staging. Problems are compared within their corresponding modeling domains to avoid conflating domain differences with modeling difficulty.

We measure inter-annotator agreement using Kendall's coefficient of concordance W (Kendall & Smith, 1939), where larger values indicate stronger agreement among annotators. Table 3 reports the results. Agreement is consistently high across all six benchmarks, with an average W of approximately 0.91. This indicates that the proposed difficulty criteria capture modeling characteristics that can be assessed consistently by independent OR experts rather than relying on arbitrary instance-level rankings. When annotators disagree on the final difficulty assignment, we use majority voting to determine the reference annotation.

Table 3: Inter-annotator agreement for optimization modeling difficulty. Kendall's W is computed from annotations by five PhD-level researchers in operations research.

	NL4Opt	**MAMO-E**	**MAMO-C**	**IndustryOR**	**OptiBench**	**OptMATH**
Kendall's W	0.95	0.93	0.91	0.92	0.87	0.89

The resulting expert annotations are used only to validate the difficulty construction and are not used to train the difficulty assessor or construct the curriculum. Additional analyses of difficulty-wise experience accumulation and model performance are provided in Appendix B.7.

B Additional Experimental Results

B.1 Backbone and Model-Scale Analysis

Prompt-Based Backbone Generalization We first examine whether prompt-based OptClimber depends on a particular underlying LLM. In addition to the default DeepSeek-V3 setting, we construct the curriculum and experience repository using Qwen2.5-7B, Qwen2.5-32B (Qwen Team, 2024), GPT-4o (OpenAI, 2024), and DeepSeek-R1 (DeepSeek-AI, 2025). For each variant, the same backbone is used throughout domain identification, difficulty assessment, experience construction and retrieval, and final problem solving. Thus, weaker backbones do not rely on a stronger model for intermediate components.

Table 4: Prompt-based backbone generalization. Accuracy (%) is reported on six optimization modeling benchmarks. "OptClimber +" denotes the corresponding backbone equipped with curriculum experience learning.

Method	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH	Avg.
Qwen2.5-7B	53.5	35.6	4.3	13.0	35.5	4.8	24.4
OptClimber + Qwen2.5-7B	**85.7**	**81.6**	**39.1**	**21.0**	**51.6**	**13.2**	**48.7**
Qwen2.5-32B	79.3	79.6	44.8	31.0	40.7	16.8	48.7
OptClimber + Qwen2.5-32B	**94.6**	**90.9**	**72.9**	**47.0**	**69.4**	**37.9**	**68.8**
GPT-4o	79.2	81.3	41.2	38.0	48.4	17.5	50.9
OptClimber + GPT-4o	**95.5**	**91.2**	**81.2**	**56.0**	**69.2**	**52.4**	**74.2**
DeepSeek-R1	82.4	87.2	67.9	45.0	67.2	34.9	64.1
OptClimber + DeepSeek-R1	**98.0**	**95.4**	**85.7**	**64.0**	**71.4**	**56.6**	**78.5**

As shown in Table 4, OptClimber consistently improves the corresponding backbone across models with substantially different base capabilities. The improvement is not restricted to strong reasoning models: even the 7B backbone benefits considerably from structured experience reuse. Stronger backbones achieve higher absolute performance, while the persistent gains across model families suggest that curriculum experience learning is not tied to a specific LLM.

Fine-Tuned Model Scaling We next study how the trained OptClimber models scale with model capacity. We apply the same experience-guided SFT and curriculum-guided RL framework to Qwen3-4B, Qwen3-8B, and Qwen3-14B (Yang et al., 2025a). The results are summarized in Table 5.

Table 5: Scaling behavior of fine-tuned OptClimber models. Accuracy (%) is reported under the same evaluation protocol for all model sizes.

Model	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH	Avg.
OptClimber-Qwen3-4B	97.1	93.6	57.8	41.0	64.5	40.4	65.7
OptClimber-Qwen3-8B	98.0	94.6	68.7	49.0	65.6	45.6	70.2
OptClimber-Qwen3-14B	**98.0**	**95.0**	**75.9**	**52.0**	**66.8**	**48.7**	**72.7**

Performance improves consistently as model capacity increases. The scaling benefit is relatively limited on NL4Opt and MAMO EasyLP, where all three models are already close to saturation. In contrast, the gains are much larger on MAMO ComplexLP, IndustryOR, and OptMATH. This suggests that additional model capacity is particularly useful for absorbing and applying accumulated modeling knowledge when solving problems with more coupled constraints and longer reasoning dependencies. Notably, the 8B model already exceeds the 32B fine-tuned baselines reported in the main comparison, indicating that the improvement cannot be explained by parameter count alone.

Performance Across Difficulty Levels To examine how the benefit of curriculum experience learning changes with problem difficulty, we further analyze resource allocation problems aggregated across the evaluation benchmarks. Problems are grouped according to their final dynamic difficulty level $\ell_T(p)$. We use the prompt-based DeepSeek-V3 setting for this analysis, which isolates the effect of retrieved curriculum experience without parameter updates.

Table 6: Accuracy (%) on resource allocation problems grouped by final dynamic difficulty level. Δ denotes the improvement of prompt-based OptClimber over the DeepSeek-V3 backbone.

Difficulty Level	**1**	**2**	**3**	**4**	**5**	**6**
DeepSeek-V3	88.47	85.62	93.21	56.73	42.17	32.96
OptClimber	88.52	91.67	95.48	86.58	90.32	77.45
Δ	+0.05	+6.05	+2.27	+29.85	+48.15	+44.49

Table 6 reveals a clear difficulty-dependent pattern. On the first three levels, the backbone already solves most problems correctly, leaving relatively limited room for improvement. The gap becomes substantially larger from Level 4 onward, where additional logical dependencies, coupled constraints, and modeling transformations become more prominent. This pattern supports the central motivation of OptClimber: curriculum-organized experience is most useful when solving a problem requires combining modeling principles that are difficult to recover reliably from a single generation. A more detailed analysis of the experiences accumulated at different difficulty levels is provided in Appendix B.7.

B.2 Training Data Analysis

Experience Construction Data For prompt-based OptClimber, we construct the curriculum and experience repository from a relatively small collection of optimization modeling problems. Unless otherwise specified, we randomly sample 200 problems from OR-Instruct (Huang et al., 2025a) after removing instances that overlap with the evaluation sets. These problems are organized according to their domain and difficulty and progressively sampled following the curriculum distribution Q_t. Solver-verified failures are distilled into reusable modeling experiences, while successfully solved problems introduce no additional failure experience.

The 200-problem setting is used throughout the main prompt-based experiments. This relatively small construction budget allows us to evaluate whether the performance gain comes from structured experience accumulation rather than from building an increasingly large retrieval corpus. We further study the effect of the construction data source and size in Appendix B.2.

Supervised Fine-Tuning Data We construct supervised fine-tuning data using the experience-guided trajectory generation procedure described in Appendix A.3. For each problem, a teacher model generates candidate modeling trajectories with access to the retrieved experience, and only trajectories that satisfy the verification criterion are retained.

The resulting SFT dataset contains **15,170 verified modeling trajectories**. Each training example contains the original natural-language problem and a verified target trajectory consisting of the mathematical modeling process and executable solver code. The retrieved experience used during teacher generation is removed from the student input. Consequently, the SFT stage transfers the modeling behavior induced by accumulated experience into the model rather than making the trained policy dependent on explicit retrieval. The same data construction protocol is used for all three OptClimber model scales.

Reinforcement Learning Data For curriculum-guided reinforcement learning, we construct a separate pool of **6,658 solver-verifiable optimization problems**. The pool covers multiple optimization domains and a broad range of modeling difficulty. Each problem is associated with its domain, curriculum level, normalized difficulty, and the reference information required for solver-grounded reward computation.

Unlike conventional training that uniformly shuffles the problem pool, we organize the RL examples according to the curriculum distribution Q_t. The resulting training sequence gradually shifts from relatively fundamental modeling problems toward more challenging ones. This allows the same difficulty structure used for experience accumulation to continue organizing model optimization during reinforcement learning.

Data Efficiency We further examine whether prompt-based OptClimber is sensitive to the amount and source of data used for experience construction. We first replace the default construction set with 200 training problems from MAMO (Huang et al., 2025b). We then vary the number of OR-Instruct problems from 100 to 400 while keeping the remaining configuration unchanged. The 200-problem OR-Instruct setting corresponds to the configuration used in the main experiments. All evaluation instances remain held out under every setting.

Table 7: Effect of the source and size of the data used for prompt-based experience construction. Accuracy (%) is reported on representative benchmarks.

Construction Data	NL4Opt	MAMO-C	IndustryOR	OptMATH
200 MAMO	96.3	**85.2**	51.0	51.8
100 OR-Instruct	95.1	80.8	52.0	51.8
200 OR-Instruct	**97.5**	83.7	**61.0**	**55.4**
300 OR-Instruct	**97.5**	81.3	58.0	54.8
400 OR-Instruct	97.1	78.8	56.0	53.6

The results indicate that OptClimber does not require a large construction corpus to obtain useful modeling experience. Increasing the number of construction problems beyond 200 does not produce monotonic improvements, suggesting that the quality and organization of accumulated experience are more important than simply increasing data volume. The results obtained from an alternative construction source further show that the framework is not restricted to a particular training collection, although closer distributional alignment can benefit individual benchmarks.

B.3 Curriculum and Experience Ablations

We conduct a series of ablations to examine how curriculum organization and experience management contribute to prompt-based OptClimber. Unless otherwise specified, all experiments use DeepSeek-V3 as the backbone and change only the component under investigation while keeping the remaining configuration fixed.

Curriculum Scheduling Strategies We first examine the role of curriculum-based problem scheduling. The *w/o Curriculum Scheduling* variant replaces the difficulty-aware distribution Q_t with uniform sampling during experience accumulation, while retaining the same experience construction and retrieval procedure.

As shown in Figure 3(a), removing curriculum scheduling consistently reduces performance, with substantially larger degradation on the more challenging benchmarks. This supports the motivation that the order in which experiences are acquired matters: beginning with more accessible modeling structures provides a more reliable basis for accumulating knowledge that can later support harder problems. The complementary comparison among Curriculum, Random, and Hard-first scheduling is already provided in the motivation study in the main paper and is therefore not repeated here.

Difficulty Organization We next examine whether the structured optimization-modeling difficulty rubric is necessary for organizing the curriculum. We replace the rubric-based difficulty estimate with three simple proxies based on problem description length, number of constraints, and number of variables. The resulting curricula are then used with the same downstream experience construction procedure.

As shown in Figure 3(b), simple surface statistics provide substantially weaker curriculum signals. In particular, the number of variables or constraints does not capture logical dependencies, coupled decisions, or transformations required to obtain a valid optimization formulation. The structured rubric therefore provides a more meaningful ordering of modeling difficulty for experience accumulation. Robustness to perturbations of these initial difficulty estimates and the effect of curriculum granularity are studied separately in Appendix B.6.

Experience Retrieval We analyze how many experiences are required at inference time by varying the retrieval budget K_{ret}. All other components, including the constructed experience repository and dynamic difficulty matching, remain unchanged.

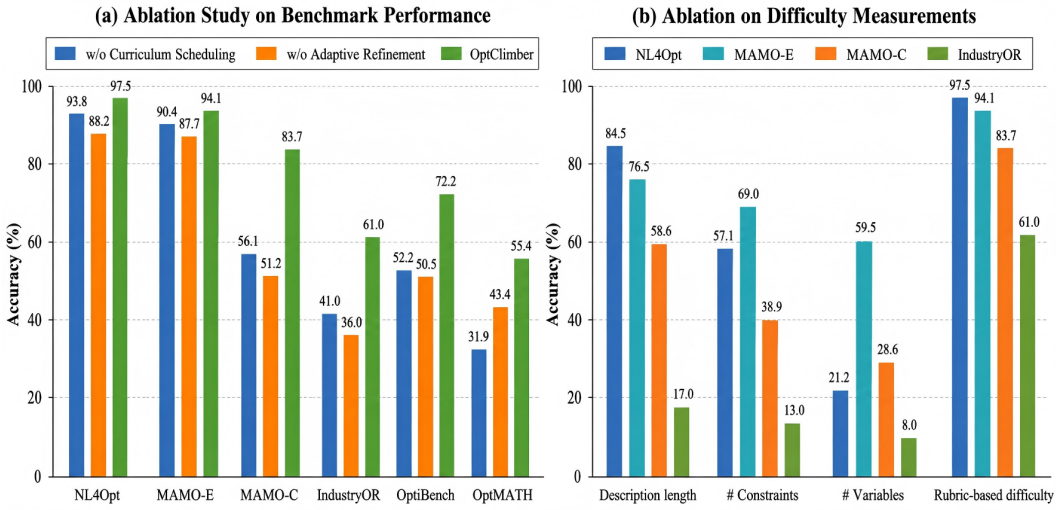

Figure 3: Ablation studies of curriculum organization. (a) Benchmark performance of OptClimber and variants without curriculum scheduling or adaptive experience refinement across six benchmarks. (b) Performance under different difficulty measurements used to organize curriculum experience accumulation. Accuracy (%) is reported.

Table 8: Effect of the number of retrieved experiences.

K_{ret}	NL4Opt	MAMO-E	MAMO-C	IndustryOR
1	97.5	94.1	**83.7**	61.0
2	**98.0**	**94.4**	83.3	**62.0**
3	96.8	93.3	82.3	61.0

Retrieving only one or two experiences is already sufficient to obtain strong performance, while increasing the retrieval budget provides no consistent benefit. This suggests that targeted experience selection is more important than exposing the model to a large amount of retrieved context. We therefore use $K_{\text{ret}} = 1$ as the default setting, which also keeps the inference context compact.

Experience Refinement The *w/o Adaptive Refinement* variant in Figure 3(a) removes adaptive refinement of the experience hierarchy. In this variant, experiences are still organized by domain, but recurring reasoning failures no longer trigger the insertion of intermediate difficulty nodes. The resulting performance degradation is particularly pronounced on MAMO ComplexLP, IndustryOR, and OptiBench.

These results indicate that a fixed initial partition is insufficient for capturing the difficulty structure encountered during problem solving. Adaptive refinement allows the curriculum and experience repository to respond to persistent generalization gaps and provides more appropriate locations for storing and retrieving newly distilled knowledge. Sensitivity to the refinement threshold η is reported in Appendix B.6.

Experience Components Finally, we examine the role of example-grounded information during experience distillation. In the default setting, the distillation process can use a verified reference solution to help identify the modeling error and construct a concrete corrective example. We remove the correct example code while retaining the failed trajectory, solver feedback, and the remaining experience distillation procedure.

Removing verified example code leads to a consistent decrease, particularly on the more difficult benchmarks, showing that concrete examples help ground abstract correction principles in executable modeling patterns. Nevertheless, the remaining performance is still substantially stronger

Table 9: Ablation of verified example code during experience distillation.

Setting	NL4Opt	MAMO-E	MAMO-C	IndustryOR
Without verified example code	95.2	93.6	79.8	56.0
Full experience	**97.5**	**94.1**	**83.7**	**61.0**

than the underlying backbone on difficult tasks, indicating that the benefit of OptClimber does not come solely from copying reference code. Instead, applicable conditions, identified modeling errors, and correction principles also provide reusable guidance across problems.

B.4 Ablations and Additional Analysis of Curriculum-Guided Reinforcement Learning

We further analyze the main design choices of curriculum-guided reinforcement learning. Unless otherwise specified, the ablations use the 4B model initialized from the same experience-guided SFT checkpoint and vary only the component under investigation. We focus on problem ordering, solver-grounded feedback, difficulty-adaptive reward shaping, and the performance progression from SFT to reinforcement learning.

Curriculum Ordering We first study whether the ordering of optimization problems influences reinforcement learning. We compare the proposed curriculum with *Random*, which removes difficulty-aware ordering, and *Hard-first*, which prioritizes difficult problems at the beginning of training. All settings use the same SFT initialization, training problems, reward function, and optimization configuration.

Table 10: Effect of problem ordering during reinforcement learning with the 4B model. Accuracy (%) is reported on three challenging benchmarks.

Ordering	IndustryOR	OptMATH	ReSocratic	Avg.
Curriculum	**40.0**	**40.4**	**64.5**	**48.3**
Random	32.0	22.9	54.2	36.4
Hard-first	18.0	30.7	47.2	32.0

Curriculum ordering clearly outperforms both alternatives. Random training removes the progression of modeling complexity, while Hard-first exposes the policy to difficult formulations before sufficiently reliable modeling behavior has been established. Together with the prompt-based motivation experiment, this result shows that difficulty-aware organization benefits both experience accumulation and subsequent policy optimization.

Solver-Grounded Reward We next examine the contribution of solver-grounded intermediate feedback. Binary correctness provides a reliable final signal but does not distinguish between different types of failed solutions. OptClimber therefore additionally uses execution validity, numerical proximity to the reference objective, and structural consistency to characterize partially correct formulations.

For an executable but incorrect solution, numerical proximity is measured by

$$S_{\text{close}} = \frac{1}{1 + 8e_{\text{rel}}}, \qquad e_{\text{rel}} = \frac{|\hat{v} - v^\star|}{|v^\star| + 1}, \tag{26}$$

where $\hat{v}$ and $v^\star$ denote the generated and reference objective values, respectively.

The results show that solver feedback provides useful information beyond binary correctness, although the benefit of individual signals varies across benchmarks. Execution validity is particularly useful on IndustryOR, numerical closeness provides the strongest average performance among the static variants, and structural consistency further improves OptMATH. This suggests that different failure signals capture complementary aspects of optimization modeling quality rather than providing a single universally dominant signal.

Table 11: Ablation of solver-grounded reward signals using the 4B model. Difficulty-dependent reward adaptation is disabled in this experiment.

Reward Signal	MAMO-C	IndustryOR	OptMATH	Avg.
Binary correctness	52.2	33.0	36.2	40.5
+ Execution validity	53.1	**40.0**	33.7	42.3
+ Numerical closeness	**55.7**	38.0	37.1	**43.6**
+ Structural consistency	54.1	36.0	**38.2**	42.8

Difficulty-Adaptive Reward OptClimber further adapts partial feedback according to problem difficulty. Let $\bar{d}(p) \in [0, 1]$ denote normalized difficulty. The weight assigned to numerical closeness is

$$w_c(\bar{d}) = 0.30 + 0.50\bar{d}, \tag{27}$$

and the scale of the partial reward is

$$c(\bar{d}) = 0.30 + 0.40\bar{d}. \tag{28}$$

Thus, more difficult problems retain a richer range of intermediate feedback, while a verified correct solution always receives the maximum reward.

Table 12: Ablation of the difficulty-adaptive reward design using the 4B model.

Reward Adaptation	MAMO-C	IndustryOR	OptMATH	Avg.
Static reward	48.2	32.0	34.2	38.1
Adaptive closeness only	52.2	36.0	36.1	41.4
Adaptive scaling only	50.7	38.0	37.8	42.2
Full difficulty-adaptive reward	**57.8**	**41.0**	**40.4**	**46.4**

Both forms of difficulty adaptation improve over the static reward, and their combination performs best on all three benchmarks. The gain is particularly clear on MAMO ComplexLP and IndustryOR, supporting the motivation that harder optimization problems benefit from more informative intermediate supervision rather than a uniform treatment of all incorrect formulations.

SFT-to-RL Learning Progression We next compare the models immediately after experience-guided SFT with their corresponding policies after curriculum-guided reinforcement learning. This comparison isolates the additional contribution of RL after accumulated experience has already been transferred into the model through supervised training.

Table 13: Performance progression from experience-guided SFT to curriculum-guided RL. Accuracy (%) is reported under the same evaluation protocol.

Scale	Stage	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH	Avg.
4B	SFT	92.6	81.6	35.1	40.0	17.1	59.7	54.4
4B	SFT + RL	**97.1**	**93.6**	**57.8**	**41.0**	**64.5**	40.4	**65.7**
8B	SFT	96.7	84.2	43.8	43.0	21.3	62.6	58.6
8B	SFT + RL	**98.0**	**94.6**	**68.7**	**49.0**	**65.6**	45.6	**70.2**
14B	SFT	96.3	90.5	63.0	45.0	65.8	40.9	66.9
14B	SFT + RL	**98.0**	**95.0**	**75.9**	**52.0**	**66.8**	**48.7**	**72.7**

RL provides substantial additional gains after SFT, and the improvement becomes more pronounced as model capacity increases. The effect is especially clear on MAMO ComplexLP and IndustryOR, where successful modeling requires more coupled constraints and structural decisions. The 14B model also improves strongly on OptMATH, suggesting that larger models are better able to exploit the curriculum and graded solver feedback on difficult formulations.

The comparison also shows that SFT and RL play complementary roles rather than serving as interchangeable training stages. SFT transfers solver-verified experience into the initial policy, while RL further improves the policy through feedback from its own generated solutions.

Additional Reinforcement Learning Results We finally examine intermediate RL checkpoints for the 14B model. This provides a more detailed view of how curriculum-guided optimization evolves beyond the SFT initialization.

Table 14: Checkpoint-wise reinforcement-learning results for OptClimber-Qwen3-14B. SFT denotes the initialization before RL.

Stage	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH	Avg.
SFT	96.3	90.5	63.0	45.0	65.8	40.9	66.9
RL Step 25	96.3	89.4	66.4	45.0	65.8	39.4	67.0
RL Step 50	95.1	90.2	64.0	48.0	67.3	48.2	68.8
RL Step 75	97.1	**96.3**	72.7	51.0	65.8	46.7	71.6
RL Step 100	**98.0**	95.0	**75.9**	**52.0**	66.8	**48.7**	**72.7**

The overall accuracy increases from 66.9 after SFT to 72.7 at Step 100. Although individual benchmarks fluctuate during optimization, the later checkpoints show increasingly strong performance on MAMO ComplexLP, IndustryOR, and OptMATH. This progression is consistent with the curriculum design: as training proceeds, the policy receives greater exposure to more difficult modeling problems, and the resulting improvements increasingly concentrate on benchmarks that require more involved formulation decisions.

B.5 Additional Baseline and Matched-Budget Comparisons

We provide two additional comparisons to clarify the relationship between OptClimber and recent experience- or retrieval-based optimization modeling methods. The first controls the amount of data used to construct the external knowledge structure, while the second evaluates OptClimber under the released AlphaOPT evaluation setting.

Matched-Budget Comparison The main comparison uses the same 200-problem OR-Instruct construction budget for OptClimber and OptiTree (Liu et al., 2026a). To further examine the stability of this comparison, we repeat both methods five times under the same construction budget and report their best, worst, and average performance.

Table 15: Matched-budget comparison with OptiTree. Both methods construct their retrieval or experience structures from the same 200 randomly sampled OR-Instruct problems. Results summarize five runs.

Setting	Method	NL4Opt	MAMO-E	MAMO-C	IndustryOR	OptiBench	OptMATH	Avg.
Best	OptiTree	98.0	**94.7**	78.3	51.0	69.6	50.6	73.7
	OptClimber	**98.0**	93.5	**83.7**	**64.0**	**73.2**	**56.6**	**78.2**
Worst	OptiTree	96.3	91.6	75.4	44.0	69.1	48.8	70.9
	OptClimber	**97.2**	**93.8**	**81.8**	**58.0**	**71.2**	**53.0**	**75.8**
Average	OptiTree	97.1	92.4	75.9	46.8	69.4	49.9	71.9
	OptClimber	**97.7**	**94.3**	**83.3**	**61.0**	**72.4**	**55.2**	**77.3**

Under the same construction budget, OptClimber improves the six-benchmark average from 71.9 to 77.3, corresponding to a gain of 5.4 percentage points. The improvement is concentrated on the more challenging benchmarks, especially MAMO ComplexLP and IndustryOR. OptClimber also achieves a substantially higher worst-case average, suggesting that curriculum-aligned experience construction provides more stable transfer than increasing retrieval structure alone. These results support the main-paper conclusion that the gain does not simply arise from access to more historical modeling examples.

Additional Evaluation with AlphaOPT AlphaOPT (Kong et al., 2026) follows a released insight-library and test-set configuration that differs from the 200-problem construction protocol used in our main experiments. We therefore additionally evaluate OptClimber directly on the released AlphaOPT test sets while retaining the corresponding evaluation setting.

Table 16: Additional comparison on the released AlphaOPT test sets. Numbers in parentheses indicate the number of evaluation instances.

Method	NLP4LP (73)	NL4Opt (64)	IndustryOR (25)	MAMO-C (34)	Avg.
AlphaOPT	86.3	79.7	60.0	85.3	77.8
OptClimber	**94.5**	**87.5**	**76.0**	**88.2**	**86.6**

OptClimber achieves higher accuracy on all four released test sets, improving the average by 8.8 percentage points. The largest gain appears on IndustryOR, where more involved application-specific modeling decisions place greater demands on reusable modeling knowledge. Because this experiment follows the released AlphaOPT evaluation setting rather than the matched 200-problem construction setting, we treat it as a complementary comparison rather than a strictly matched-data experiment.

B.6 Cross-Difficulty Generalization and Robustness

Cross-Difficulty Generalization We first examine whether experience accumulated from relatively simple problems can transfer to more challenging optimization tasks. We construct the experience repository using 200 problems sampled from NL4Opt and MAMO EasyLP, and evaluate the resulting prompt-based OptClimber on IndustryOR and MAMO ComplexLP. DeepSeek-V3 is used as the backbone throughout this experiment.

Table 17: Cross-difficulty generalization from easier construction problems to more challenging benchmarks. Accuracy (%) is reported.

Method	IndustryOR	MAMO-C
DeepSeek-V3	37.0	50.2
OptClimber	**56.0**	**76.8**

Although the experience repository is constructed only from easier problems, OptClimber substantially improves performance on both harder benchmarks. This result suggests that the accumulated experience captures transferable modeling principles rather than merely memorizing problem-specific solutions. Knowledge acquired from simpler formulations can therefore remain useful when the same modeling structures appear as components of more complex problems.

Difficulty Perturbation We further test whether OptClimber is sensitive to errors in the initial difficulty estimates. We perturb the initial difficulty scores before constructing the curriculum while keeping the remaining pipeline unchanged.

Table 18: Robustness to perturbations of the initial difficulty estimates. Accuracy (%) is reported.

Perturbation	NL4Opt	MAMO-E	MAMO-C
10%	97.5	94.0	83.7
20%	97.5	94.0	83.3
30%	97.1	94.1	83.3

Performance remains stable under moderate perturbations, indicating that OptClimber does not require perfectly calibrated initial difficulty estimates. This robustness is consistent with the role of difficulty as an initialization prior, since dynamic matching and subsequent refinement can adjust the curriculum as experience accumulates.

Curriculum Granularity We vary the initial maximum difficulty level $L_{\max}$ to study the effect of curriculum granularity. Smaller values produce a coarser initial partition, whereas larger values distinguish a wider range of modeling difficulty.

Table 19: Sensitivity to the initial curriculum granularity $L_{\max}$. Accuracy (%) is reported.

$L_{\max}$	NL4Opt	MAMO-C	IndustryOR	OptMATH
5	96.7	81.7	56.0	53.6
8	97.5	81.7	58.0	55.4
10	**97.5**	**83.7**	**61.0**	**55.4**

A finer initial curriculum is particularly beneficial on the more challenging benchmarks, while the difference is limited on relatively saturated tasks. We therefore use $L_{\max} = 10$ in the default configuration.

Retrieval Budget Sensitivity We vary the maximum number of retrieved experiences K_{ret} to study the sensitivity to retrieval context size.

Table 20: Sensitivity to the retrieval budget K_{ret}. Accuracy (%) is reported.

K_{ret}	NL4Opt	MAMO-E	MAMO-C	IndustryOR
1	97.5	94.1	**83.7**	61.0
2	**98.0**	**94.4**	83.3	**62.0**
3	96.8	93.3	82.3	61.0

One or two retrieved experiences are already sufficient for strong performance, while increasing the retrieval budget provides no consistent improvement. This suggests that targeted retrieval is more important than simply increasing the amount of additional context.

Refinement Threshold Sensitivity We finally vary the refinement threshold η, which controls when persistent failures trigger the insertion of an intermediate difficulty level.

Table 21: Sensitivity to the refinement threshold η. Accuracy (%) is reported.

η	NL4Opt	MAMO-E	MAMO-C	IndustryOR
0.25	97.5	94.1	**84.2**	**61.0**
0.30	97.5	94.1	83.7	**61.0**
0.35	97.5	**94.2**	82.8	59.0
0.40	**98.0**	93.9	80.8	54.0

The method remains stable across moderate threshold values, while overly conservative refinement reduces performance on difficult benchmarks. We use $\eta = 0.30$ in the default configuration.

B.7 Experience and Difficulty Analysis

Experience Coverage We measure experience coverage as the proportion of evaluation problems for which the final repository retrieves at least one matched experience. Figure 4 summarizes coverage together with difficulty matching accuracy and retrieved experience quality.

The experience repository covers 79.1% of evaluation problems on average, showing that many modeling patterns recur across independently constructed benchmarks. At the same time, coverage is not forced to 100%, allowing the model to avoid conditioning on unrelated experience when no suitable match is available.

Retrieved Experience Quality Coverage alone does not guarantee that the retrieved experience is relevant. We therefore additionally evaluate the reliability of difficulty matching and experience retrieval. As shown in Figure 4, the final dynamic hierarchy achieves 91.1% agreement with expert difficulty annotations, while 93.5% of retrieved experiences are judged useful for the corresponding problems.

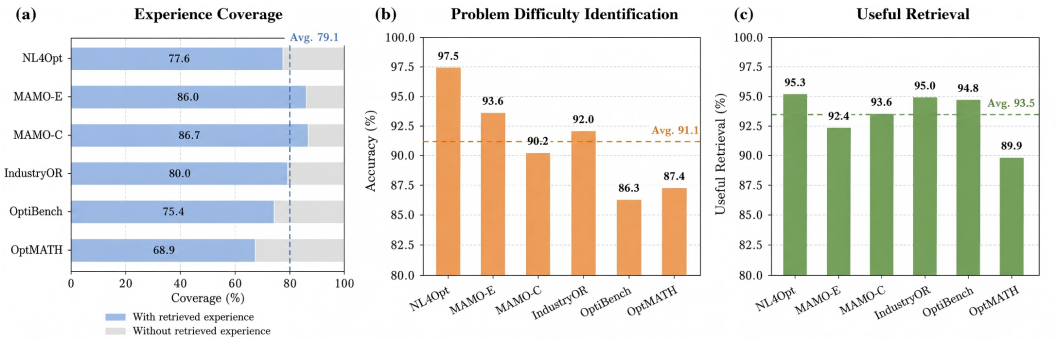

Figure 4: Experience coverage and retrieval reliability across the six benchmarks. Panel (a) shows experience coverage, panel (b) shows difficulty identification accuracy, and panel (c) shows useful retrieval. All values are percentages.

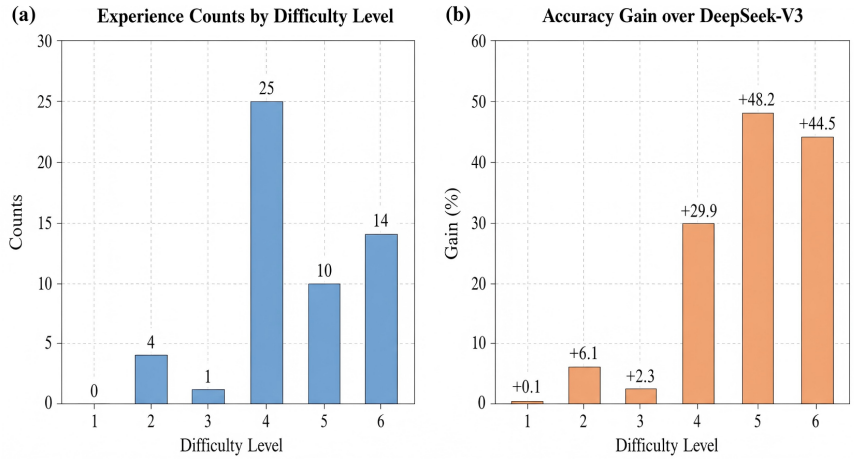

Figure 5: Difficulty-wise experience accumulation for Resource Allocation problems. Panel (a) shows the number of accumulated experiences at each difficulty level, and panel (b) shows the corresponding accuracy gain over DeepSeek-V3.

These results indicate that organizing experience jointly by domain and difficulty provides an effective mechanism for locating relevant modeling knowledge. The high retrieval quality also helps explain why relatively small retrieval budgets remain sufficient in Appendix B.6.

Difficulty-Wise Experience Accumulation We further analyze Resource Allocation problems according to their final dynamic difficulty level. Figure 5 shows the number of accumulated experiences and the corresponding accuracy improvement over the DeepSeek-V3 backbone.

Few corrective experiences are accumulated at the easiest levels, where the backbone already solves most problems successfully. In contrast, harder levels produce more distinctive failure patterns and substantially larger performance gains. This behavior is consistent with the design of OptClimber, where experiences are introduced when failures expose unresolved modeling knowledge.

Modeling Difficulty Analysis The difficulty ablation in Figure 3(b) shows that simple surface statistics, including description length and the numbers of variables and constraints, provide substantially weaker curriculum signals than the optimization-specific difficulty rubric. Modeling difficulty therefore cannot be reliably characterized by problem size alone. Instead, logical dependencies, constraint coupling, variable construction, and required transformations play a central role in determining the difficulty of optimization modeling.

B.8 COMPUTATIONAL COST

Experience Construction Cost For prompt-based OptClimber, the curriculum and experience repository are constructed once and subsequently reused across evaluation problems. The construction process requires no gradient-based parameter updates and mainly consists of difficulty assessment, problem solving, solver verification, experience distillation, and repository refinement.

Table 22 reports the total token consumption during experience construction for several representative backbones.

Table 22: Token consumption during prompt-based experience construction.

Backbone	Construction Tokens
DeepSeek-V3	8.7M
GPT-4o	10.4M
Qwen2.5-32B	11.9M

In the default DeepSeek-V3 setting, constructing the experience repository requires approximately 8.7M tokens. This cost is incurred only once, after which the same repository can be reused for subsequent optimization modeling problems.

Fine-Tuning Cost The fine-tuned OptClimber models are trained on a server equipped with **eight NVIDIA H20 GPUs**. Both experience-guided supervised fine-tuning and curriculum-guided reinforcement learning use the same eight-GPU training environment for Qwen3-4B, Qwen3-8B, and Qwen3-14B. These training costs are incurred offline, and the resulting models can be used without accessing the external experience repository at inference time.

Inference Cost We compare the inference efficiency of prompt-based OptClimber with representative prompt-based optimization modeling methods, including CoE (Xiao et al., 2024), OptiMUS (AhmadiTeshnizi et al., 2024), and MCTS (Astorga et al., 2025). For OptClimber, the total runtime consists of curriculum matching, experience retrieval, and final model inference.

Table 23: Average inference time per problem (seconds) for representative prompt-based optimization modeling methods.

Method	NL4Opt	MAMO-E	MAMO-C	IndustryOR
CoE	37.2	42.6	76.7	81.8
OptiMUS	26.4	22.5	53.2	57.8
MCTS	103.2	110.8	111.4	124.6
OptClimber: Curriculum Matching	12.9	9.2	13.6	16.4
OptClimber: Experience Retrieval	3.2	4.8	4.4	3.5
OptClimber: Model Inference	9.8	8.5	12.2	13.8
OptClimber: Total	**25.9**	**22.5**	**30.2**	**33.7**

Despite introducing experience retrieval, OptClimber remains computationally competitive with existing prompt-based methods. Its total inference time is substantially lower than CoE and MCTS across the evaluated benchmarks and is comparable to or lower than OptiMUS. Experience retrieval itself accounts for only a small fraction of the total runtime, indicating that structured experience reuse does not introduce substantial inference overhead.

We additionally report the average token consumption of the complete prompt-based pipeline in Table 24.

For the default DeepSeek-V3 backbone, the complete prompt-based pipeline uses approximately 2.7K tokens per problem on average. Together with the wall-clock comparison, these results show that the improvements of prompt-based OptClimber do not rely on substantially higher inference cost.

Table 24: Average token consumption per problem for prompt-based OptClimber with different backbones.

Backbone	Curriculum Matching	Experience Retrieval	Inference	Total
DeepSeek-V3	528	1,169	976	2,673
Qwen2.5-32B	736	1,440	1,189	3,365
Qwen2.5-7B	812	1,543	1,204	3,559

For the fine-tuned setting, no curriculum matching or external experience retrieval is required during inference. All evaluations of OptClimber-Qwen3-4B, OptClimber-Qwen3-8B, and OptClimber-Qwen3-14B are conducted using **eight NVIDIA H20 GPUs** under the same evaluation configuration and solver-based correctness protocol.

C Algorithm Details

This section provides the complete algorithmic procedures underlying OptClimber. The algorithms follow the notation introduced in the main text. The first procedure describes how the domain-and-difficulty curriculum is used to progressively accumulate solver-verified modeling experience and adapt the difficulty hierarchy. The second procedure describes how the resulting experience repository is converted into model capability through experience-guided supervised fine-tuning and curriculum-guided reinforcement learning. Together, these procedures instantiate the unified learning process described in Sections 4.2 and 4.3.

C.1 Curriculum Experience Accumulation

Curriculum experience accumulation maintains a domain-specific difficulty hierarchy together with an aligned experience repository. At each construction step, the curriculum determines which problems are visited, while solving outcomes determine how the experience repository and difficulty hierarchy are updated. Problems are therefore not treated as independent experience sources: the current curriculum affects both the order in which knowledge is acquired and the location in which newly distilled experience is stored. Algorithm 1 summarizes the complete procedure.

Curriculum Construction and Scheduling Given the problem collection $\mathcal{P} = \{(p_j, y_j)\}_{j=1}^{N}$, OptClimber first identifies the modeling domain $g(p) \in \mathcal{G}$ and initial difficulty $d(p) \in \mathcal{D}$ and constructs the domain-and-difficulty curriculum $\mathcal{C} = \mathcal{G} \times \mathcal{D}$. For domain g_i, the curriculum at step t is represented by

$$\mathcal{C}_i^t = \left(C_{i,1}^t, \ldots, C_{i,L_i^t}^t\right).$$

Problem sampling follows the same easy-biased distribution defined in the main text,

$$Q_{\text{diff}}(p) = \frac{1/d(p)}{\sum_{q \in \mathcal{P}} 1/d(q)},$$

and

$$Q_t(p) = \lambda_t Q_{\text{diff}}(p) + (1 - \lambda_t)\frac{1}{|\mathcal{P}|}.$$

As λ_t decreases, the sampling distribution gradually shifts from an easy-biased curriculum toward broader coverage of more difficult problems.

Solver-Verified Experience Distillation For a sampled problem p, OptClimber identifies its domain $g(p) = g_i$ and dynamically matches it to the current difficulty level

$$k = \ell_t(p).$$

The model then solves the problem using the retrieved experience $\mathcal{R}_t(p)$. The generated solution $\hat{y}$ is verified against the ground-truth target y using the optimization solver, producing

$$R(p) \in \{\text{Success}, \text{CodeError}, \text{ReasoningError}\}.$$

A successful trajectory introduces no new failure experience. For an unsuccessful trajectory, OptClimber distills

$$\kappa = \mathrm{Distill}\left(p, \hat{y}, y, R(p)\right),$$

where κ summarizes the applicable condition, the identified modeling error, the corresponding correction principle, and a compact example. The experience is then associated with the matched node $E^t_{i,k}$.

Experience Refinement and Retrieval The initial difficulty partition may not provide sufficient resolution for all problems. OptClimber therefore maintains a recent attempt history $H^t_{i,k}$ and a reasoning-error buffer $\widetilde{\mathcal{P}}^t_{i,k}$ for each domain-level pair. When

$$\left|\widetilde{\mathcal{P}}^t_{i,k}\right| \geq M_{\max} \qquad \text{or} \qquad \mathrm{Acc}(H^t_{i,k}) \leq \eta,$$

an intermediate difficulty node is inserted and the aligned curriculum and experience structures are updated accordingly.

For a problem assigned to level k, retrieval is restricted to the matched and easier levels,

$$\mathcal{R}_t(p) = \mathrm{TopK}\left(p, \bigcup_{r \leq k} E^t_{i,r}\right).$$

This restriction allows foundational modeling knowledge to transfer upward through the curriculum while avoiding experiences from difficulty levels that may require substantially more advanced reasoning.

Algorithm 1 Curriculum Experience Accumulation

Require: Problem collection $\mathcal{P} = \{(p_j, y_j)\}_{j=1}^{N}$; initial domain-specific curricula $\{\mathcal{C}^0_i\}$; aligned experience repository $\mathcal{E}$; construction steps T; batch size B; retrieval budget K_{ret}; refinement parameters $M_{\max}$ and η

Ensure: Updated curriculum and experience repository

for $t = 1, \ldots, T$ **do**
  Construct the sampling distribution $Q_t(p) = \lambda_t Q_{\mathrm{diff}}(p) + (1 - \lambda_t)/|\mathcal{P}|$
  Sample a mini-batch $\mathcal{B}_t$ of size B from Q_t
  for all $(p, y) \in \mathcal{B}_t$ **do**
    Identify $g(p) = g_i$ and $k \leftarrow \ell_t(p)$ under $\mathcal{C}^t_i$
    $\mathcal{R}_t(p) \leftarrow \textsc{TopK}(p, \bigcup_{r \leq k} E^t_{i,r}, K_{\mathrm{ret}})$
    $\hat{y} \sim \pi_\theta(\cdot \mid p, \mathcal{R}_t(p))$
    $R(p) \leftarrow \mathrm{Verify}(p, \hat{y}, y)$
    Update the recent attempt history $H^t_{i,k}$
    if $R(p) = \mathrm{Success}$ **then**
      continue
    end if
    $\kappa \leftarrow \mathrm{Distill}(p, \hat{y}, y, R(p))$
    Associate κ with $E^t_{i,k}$
    if $R(p) = \mathrm{ReasoningError}$ **then**
      $\widetilde{\mathcal{P}}^t_{i,k} \leftarrow \widetilde{\mathcal{P}}^t_{i,k} \cup \{p\}$
      if $|\widetilde{\mathcal{P}}^t_{i,k}| \geq M_{\max}$ **or** $\mathrm{Acc}(H^t_{i,k}) \leq \eta$ **then**
        Insert an intermediate difficulty node into $\mathcal{C}^t_i$
        Update the aligned experience nodes $E^t_{i,\cdot}$ and re-index the hierarchy
        Clear $\widetilde{\mathcal{P}}^t_{i,k}$
      end if
    end if
  end for
end for
return updated curriculum and experience repository

C.2 CURRICULUM-GUIDED FINE-TUNING

The accumulated experience can be used directly through prompt-based retrieval, but OptClimber further converts this external knowledge into model capability. The training procedure contains two successive stages. First, experience-guided SFT uses the repository to construct high-quality solver-verified demonstrations. Second, curriculum-guided RL starts from the resulting SFT policy and continues to organize training problems according to the same difficulty-aware curriculum. Algorithm 2 summarizes this transition from accumulated experience to the final trained policy.

Experience-Guided Supervised Fine-Tuning For each training problem p, the relevant experiences $\mathcal{R}_t(p)$ are first retrieved from the accumulated repository. A teacher policy π_T observes both the original problem and the retrieved experience and generates

$$y^T \sim \pi_T\left(\cdot \mid p, \mathcal{R}_t(p)\right).$$

Only solver-verified trajectories are retained:

$$\mathcal{D}_{\mathrm{SFT}} = \left\{(p, y^T) \mid \mathrm{Verify}(p, y^T) = \mathrm{Success}\right\}.$$

The student does not receive $\mathcal{R}_t(p)$ as input. Instead, it is trained only on the original problem and the verified teacher trajectory using

$$\mathcal{L}_{\mathrm{SFT}} = -\sum_{(p,y^T)\in\mathcal{D}_{\mathrm{SFT}}} \sum_m \log \pi_\theta\left(y_m^T \mid p, y_{<m}^T\right).$$

The resulting policy is denoted by π_{SFT}. This asymmetric teacher–student construction is important because it converts the behavior induced by retrieved experience into a capability that can subsequently be used without external retrieval.

Curriculum-Guided Reinforcement Learning Starting from π_{SFT}, OptClimber continues training with the same curriculum distribution used during experience accumulation. At RL step t,

$$p_t \sim Q_t,$$

and the current policy generates a group of candidate solutions

$$\hat{y}_1, \ldots, \hat{y}_G \sim \pi_\theta(\cdot \mid p_t).$$

Each candidate is evaluated using the solver-grounded graded reward from the main text:

$$r(p, \hat{y}) = \begin{cases} 1, & \mathrm{Verify}(p, \hat{y}) = \mathrm{Success}, \\ c(\bar{d})\dfrac{w_e R_{\mathrm{exec}} + w_c(\bar{d}) S_{\mathrm{close}} + w_s B_{\mathrm{struct}}}{w_e + w_c(\bar{d}) + w_s}, & \text{otherwise}, \end{cases}$$

where $\bar{d}(p) \in [0,1]$ denotes normalized problem difficulty. R_{exec} measures execution validity, S_{close} measures numerical closeness to the reference target, and B_{struct} measures structural consistency when such information is available. The difficulty-dependent functions $w_c(\bar{d})$ and $c(\bar{d})$ provide more informative intermediate feedback for difficult problems while keeping the reward of an incorrect solution below that of a verified correct solution.

For the G candidate solutions, OptClimber computes the group-relative advantage

$$A_i = r_i - \frac{1}{G}\sum_{j=1}^{G} r_j,$$

and updates the policy using GRPO (Shao et al., 2024). In this way, the curriculum determines which modeling problems are encountered at each stage, while solver-grounded feedback determines how the policy improves on those problems.

D CASE STUDIES

We provide representative cases to illustrate how curriculum experience learning affects optimization modeling at both inference and training time. The prompt-based cases focus on how retrieved experience can correct, or occasionally fail to correct, a concrete modeling error. The fine-tuned cases then illustrate how the same modeling knowledge is transferred into the model through SFT and further strengthened by solver-grounded reinforcement learning.

Algorithm 2 Experience-Guided SFT and Curriculum-Guided Reinforcement Learning

Require: Accumulated experience repository $\mathcal{E}$; teacher policy π_T; student policy π_θ; curriculum distribution Q_t; group size G; RL steps T_{RL}
Ensure: Trained modeling policy π_θ

Stage 1: Experience-Guided Supervised Fine-Tuning
$\mathcal{D}_{\mathrm{SFT}} \leftarrow \emptyset$
for all SFT problem p **do**
  Identify $g(p) = g_i$ and $k \leftarrow \ell_t(p)$
  $\mathcal{R}_t(p) \leftarrow \textsc{TopK}(p, \bigcup_{r \le k} E^t_{i,r})$
  $y^T \sim \pi_T(\cdot \mid p, \mathcal{R}_t(p))$
  if $\mathrm{Verify}(p, y^T) = \mathrm{Success}$ **then**
    $\mathcal{D}_{\mathrm{SFT}} \leftarrow \mathcal{D}_{\mathrm{SFT}} \cup \{(p, y^T)\}$
  end if
end for
Optimize π_θ using $\mathcal{L}_{\mathrm{SFT}}$ on $\mathcal{D}_{\mathrm{SFT}}$
$\pi_{\mathrm{SFT}} \leftarrow \pi_\theta$

Stage 2: Curriculum-Guided Reinforcement Learning
Initialize $\pi_\theta \leftarrow \pi_{\mathrm{SFT}}$
for $t = 1, \ldots, T_{\mathrm{RL}}$ **do**
  Sample $p_t \sim Q_t$
  for $i = 1, \ldots, G$ **do**
    $\hat{y}_i \sim \pi_\theta(\cdot \mid p_t)$
    $r_i \leftarrow r(p_t, \hat{y}_i)$
  end for
  $A_i \leftarrow r_i - \dfrac{1}{G}\sum_{j=1}^{G} r_j, \quad i = 1, \ldots, G$
  Update π_θ with GRPO using $\{(\hat{y}_i, A_i)\}_{i=1}^{G}$
end for
return π_θ

D.1 Prompt-Based Modeling Cases

Successful Experience Transfer We first consider an IndustryOR problem involving a two-stage blending process (Huang et al., 2025a). Four raw materials with different sulfur contents are used to produce two final products. Three materials must first be combined into a shared intermediate mixture, which is subsequently allocated and blended with the remaining material. The final products must satisfy product-specific sulfur requirements and market-demand limits.

Problem

Raw materials A, B, and D are first blended into a shared intermediate mixture. The mixture is then allocated between two products and blended with raw material C. Product 1 and Product 2 have maximum sulfur contents of 2.5% and 1.5%, respectively. Raw material D is limited to 50 tons, and market demands for the two products are at most 100 and 200 tons. The objective is to maximize profit.

DeepSeek-V3 makes two related errors. First, it interprets market demand as an equality and therefore forces the model to produce the full demand. Second, it creates a separate intermediate mixture for each final product, thereby collapsing the shared two-stage production process.

Modeling Error

The generated model contains constraints of the form

$$\mathrm{Mix}_1 + C_1 = 100, \qquad \mathrm{Mix}_2 + C_2 = 200,$$

and independently defines

$$\text{Mix}_1 = A_1 + B_1 + D_1, \qquad \text{Mix}_2 = A_2 + B_2 + D_2.$$

The first pair incorrectly turns demand limits into mandatory production. The second pair loses the semantics of a *shared* intermediate stream.

OptClimber retrieves experience concerning incorrect constraint aggregation and the preservation of multi-stage process structure. The retrieved knowledge encourages the model to distinguish an upper bound from an equality and to represent the intermediate mixture before allocating it to downstream products.

Experience-Guided Correction

A corrected formulation introduces one shared intermediate amount M, followed by downstream allocation:

$$M = x_A + x_B + x_D, \qquad y_1 + y_2 = M.$$

If z_j denotes the amount of material C assigned to product j, the final production amounts satisfy

$$P_j = y_j + z_j, \qquad P_1 \le 100, \qquad P_2 \le 200.$$

Product-specific sulfur constraints are then imposed after the intermediate mixture has been allocated. The formulation therefore preserves the intended

$$\text{mix} \rightarrow \text{allocate} \rightarrow \text{blend}$$

process rather than constructing two independent intermediate mixtures.

Failure Analysis Experience retrieval does not guarantee that every subtle modeling requirement will be resolved. We illustrate this limitation with another IndustryOR problem involving discrete vegetable purchases.

Problem

A customer chooses protein sources and vegetables to maximize total protein under a $20 budget and an 800g weight limit. Protein sources may be purchased continuously, while vegetables are sold only in **100g packs**. At least three different vegetable types must be selected.

OptClimber correctly retrieves experience concerning binary selection and integer decision variables. However, the retrieved principle is applied only partially. The generated formulation introduces binary selection variables but keeps vegetable quantities continuous.

Modeling Error

A representative erroneous formulation is

```
veg_select = model.addVars(
    veg_names, vtype=GRB.BINARY)

veg_quantity = model.addVars(
    veg_names, lb=0,
    vtype=GRB.CONTINUOUS)

model.addConstr(
    veg_quantity[v] <= M * veg_select[v])
```

The binary variable determines whether a vegetable is selected, but `veg_quantity` still permits fractional packs. The solver can therefore purchase, for example, 1.37 packs, which violates the 100g purchase unit.

The required distinction is between *selection integrality* and *quantity integrality*. A faithful formulation should introduce an integer pack-count variable n_v in addition to the binary selection variable z_v:

Experience-Guided Correction

$$n_v \in \mathbb{Z}_{\geq 0}, \qquad z_v \in \{0,1\},$$

with linking constraints such as

$$n_v \leq M z_v, \qquad \sum_v z_v \geq 3.$$

The purchased vegetable weight is then represented as

$$100 \sum_v n_v,$$

which explicitly preserves the 100g pack structure.

D.2 Fine-Tuned Modeling Cases

Complex Modeling Example The role of experience changes after fine-tuning. In prompt-based OptClimber, a relevant principle is explicitly retrieved at inference time. In the fine-tuned setting, experience-guided SFT instead exposes the teacher to such principles when constructing verified trajectories, while the student receives only the original problem. We use a machine no-overlap constraint to illustrate the type of modeling knowledge that this process is designed to internalize.

Problem

Consider two operations processed by the same machine. Their starting times are s_1 and s_2, and their processing times are 8 and 6 time units, respectively. The two operations cannot overlap, but either operation may be processed first.

A common modeling error is to multiply the ordering variable directly by a time-difference expression:

Modeling Error

$$(s_2 - s_1 - 8)z \geq 0, \qquad (s_1 - s_2 - 6)(1 - z) \geq 0,$$

where $z \in \{0,1\}$.
Although the intended sequencing logic is recognizable, the formulation multiplies decision variables and therefore introduces an unintended nonlinear constraint.

The reusable modeling principle is to express the conditional sequencing relationship through a Big-M reformulation:

Experience-Guided Correction

Let $z = 1$ indicate that operation 1 precedes operation 2. Then

$$s_1 + 8 \leq s_2 + M(1 - z),$$
$$s_2 + 6 \leq s_1 + Mz,$$
$$z \in \{0,1\}.$$

When $z = 1$, the first inequality becomes active and enforces operation 1 before operation 2. When $z = 0$, the second inequality becomes active and enforces the reverse ordering.

In the prompt-based setting, this correction can be provided explicitly through a retrieved "nonlinear constraint linearization" experience. Under experience-guided SFT, however, the teacher uses this experience to construct a solver-verified trajectory, while the student is trained only from the original problem. The intended outcome is therefore not memorization of the retrieved text, but internalization of the correspondence

$$\text{conditional sequencing} \quad \Longrightarrow \quad \text{binary ordering + Big-}M.$$

SFT and RL Behavior Comparison SFT and RL play complementary roles in OptClimber. SFT learns from solver-verified teacher demonstrations and therefore provides a strong initialization. However, it does not explicitly optimize how the policy should respond to its own partially correct generations. Curriculum-guided RL adds this missing signal by evaluating multiple candidate solutions with solver-grounded feedback.

To illustrate this distinction, we revisit the discrete-purchase pattern from the vegetable example above. The comparison focuses on the learning signals available to the two stages rather than on a checkpoint-specific generation.

Table 25: Illustration of the different learning signals provided by SFT and curriculum-guided RL for a discrete-purchase modeling problem.

Stage	Modeling Behavior	Learning Signal
SFT	Learns a verified trajectory containing integer pack-count variables and selection constraints.	Sequence-level imitation encourages the student to reproduce the verified modeling pattern from the problem alone.
RL: partial	Generates an executable model but treats pack quantities as continuous.	Execution succeeds, but the solution is incorrect. Numerical and structural feedback can assign partial credit while distinguishing this near-miss from a non-executable program.
RL: correct	Uses integer pack counts together with binary selection and the required linking constraints.	Solver verification succeeds and the candidate receives the maximum reward, providing a clear preference over the partially correct formulation.

The distinction is important for difficult optimization problems, where errors are rarely binary in nature. An SFT model may already learn the general structure of a formulation but still produce a locally incorrect variable domain, coefficient, or coupling constraint. Under a purely binary reward, such an executable near-miss would receive the same signal as an unrelated invalid solution. OptClimber instead uses solver-grounded graded feedback to distinguish these cases.

For the discrete-purchase example, a model with continuous pack quantities is structurally close to the correct formulation and can be executed by the solver, but it remains invalid because the integrality condition is missing. The RL reward therefore provides an intermediate signal through execution validity, numerical closeness, and structural consistency. A candidate that correctly introduces integer pack counts and satisfies the reference target receives the full reward.

E PROMPTS

We provide the core prompts used by OptClimber for curriculum construction, experience accumulation and retrieval, and SFT data construction. For readability, we report only the task-defining instructions and omit auxiliary implementation details such as repeated formatting constraints, parser-specific instructions, and retry messages. The abbreviated prompts preserve the information available to the model and the output semantics used by the method.

E.1 DIFFICULTY ASSESSMENT PROMPTS

Difficulty assessment provides the initial organization of the curriculum. For each problem p, the assessor jointly identifies its modeling domain $g(p)$ and estimates its initial difficulty according to the structural rubric described in Appendix A.4. The implementation combines domain identification and difficulty assessment into a single call. This initial estimate is subsequently normalized within each domain and is used only to initialize the curriculum rather than as a fixed difficulty label.

Domain and Difficulty Assessment

```
You are an expert in operations research modeling.

Problem:
{problem}
```

```
1. Identify the modeling domain from the predefined domain set.

2. Assess the difficulty of formulating the problem using:
   - decision-variable complexity,
   - constraint structure and coupling,
   - logical dependencies,
   - nonlinearity or required transformations,
   - uncertainty or multi-stage structure.

Assign an integer difficulty score from 1 (simple)
to 10 (very difficult).

Return:
{
  "domain": "<domain>",
  "difficulty": <1-10>,
  "reason": "<brief justification>"
}
```

The prompt focuses on modeling operations rather than superficial properties such as problem length. In particular, logical coupling, auxiliary-variable construction, linearization, and multi-stage dependencies can increase difficulty even when the natural-language description is short. The resulting raw score is converted into the domain-normalized difficulty $d(p)$ as described in Appendix A.4.

E.2 Experience Distillation Prompts

When solver verification identifies a failed trajectory, OptClimber distills the failure into a reusable experience $\kappa = \text{Distill}(p, \hat{y}, y, R(p))$. Although code errors and reasoning errors provide different failure signals, both are converted into the same compact experience representation consisting of the modeling error, its applicable condition, a correction principle, and a concrete example.

Solver-Verified Experience Distillation

```
You are given an optimization modeling problem and a failed
modeling trajectory.

Problem:
{problem}

Generated solution:
{generated_solution}

Reference information:
{ground_truth}

Verification result:
{verification_result}

Identify the reusable modeling error exposed by this failure.
Summarize only knowledge that can transfer to other problems.

Return:
{
  "error_index": "<concise error name>",
  "trigger": "<when this error occurs>",
  "correction": "<general correction principle>",
  "example": "<compact wrong-vs-correct example>"
}
```

The prompt intentionally asks for a transferable correction rather than a summary of the specific instance. For example, an erroneous multiplication between a binary variable and a continuous expression is distilled as a general linearization principle rather than as a correction tied to the particular coefficients of the original problem. This compact representation also facilitates subsequent storage and retrieval from the aligned experience repository.

E.3 Experience Refinement Prompts

The initial difficulty hierarchy may be too coarse for recurring failure patterns. When the refinement criterion in Section 4.2.3 is triggered, OptClimber summarizes the buffered reasoning-error problems $\widetilde{\mathcal{P}}^t_{i,k}$ and their associated experiences to characterize an intermediate difficulty region. The resulting summary is used to create a new curriculum node and its aligned experience node.

Intermediate Difficulty Refinement

```
You are given a group of optimization problems from the same
domain that repeatedly fail at the current difficulty level,
together with the modeling experiences distilled from them.

Failed problems:
{buffered_problems}

Distilled experiences:
{buffered_experiences}

Identify the common modeling requirements that make these
problems harder than the preceding level but more specific
than the current coarse level.

Produce a concise difficulty-node summary describing:
- the shared modeling structures,
- the additional reasoning requirements,
- the representative failure patterns.
```

The refinement prompt does not redefine the entire curriculum. It summarizes only the recurring gap revealed by recent failures. The new node is then inserted into the existing easy-to-hard hierarchy according to its modeling requirements, allowing $\mathcal{C}^t_i$ and the aligned experience repository to evolve together.

E.4 Experience Retrieval Prompts

For a target problem with $g(p) = g_i$ and $k = \ell_t(p)$, candidate experiences are restricted to the matched and easier levels, $\bigcup_{r \le k} E^t_{i,r}$. Retrieval then selects the experiences whose applicable conditions best match the target problem. This prevents unrelated or unnecessarily advanced experiences from being introduced into the modeling context.

Experience Matching

```
You are given an optimization modeling problem and a set of
candidate modeling experiences.

Problem:
{problem}

Candidate experiences:
{available_experiences}

Select up to K experiences that are directly applicable to the
modeling requirements of this problem.

Prefer experiences that address the same variable structure,
constraint logic, modeling transformation, or implementation
pitfall.

Return the selected experience IDs. If none is relevant,
return no experience.
```

The selected experiences form $\mathcal{R}_t(p)$. For prompt-based OptClimber, they are subsequently supplied to the backbone together with the original problem:

Experience-Conditioned Modeling

```
You are an expert in optimization modeling.

Problem:
{problem}

Relevant modeling experiences:
{retrieved_experiences}

Solve the problem using the relevant principles above.
Adapt the experiences to the current problem rather than
copying problem-specific variables or parameters.

Provide:
1. the mathematical formulation;
2. the corresponding executable Gurobi code.
```

The separation between matching and application is important. Retrieval determines *which* accumulated knowledge is relevant, while the modeling prompt requires the backbone to instantiate that knowledge for the current problem. As a result, retrieved experiences function as reusable modeling principles rather than stored solutions.

E.5 Supervised Fine-Tuning Data Construction Prompts

Experience-guided SFT reuses the same experience-conditioned modeling principle, but changes how the resulting trajectory is consumed. During data construction, the teacher observes both p and $\mathcal{R}_t(p)$ and generates a candidate trajectory y^T. The trajectory is retained only after solver verification. The retrieved experience is subsequently removed from the student input, so the student learns from (p, y^T) rather than from $(p, \mathcal{R}_t(p), y^T)$.

Experience-Guided Teacher Generation

```
You are an expert in operations research and Gurobi.

Problem:
{problem}

Relevant modeling experiences:
{retrieved_experiences}

Use the experiences as modeling guidance and solve the problem.
Construct a complete mathematical formulation and executable
Gurobi implementation.

The response should contain:

<model>
Decision variables and their types;
objective function;
all required constraints;
important modeling decisions.
</model>

<python>
A complete executable Gurobi implementation.
</python>
```

Candidate teacher generations are verified before entering $\mathcal{D}_{\mathrm{SFT}}$. The final supervised example therefore has the asymmetric form

$$\underbrace{\big(p, \mathcal{R}_t(p)\big) \xrightarrow{\pi_T} y^T}_{\text{teacher generation}} \quad \Longrightarrow \quad \underbrace{(p, y^T) \in \mathcal{D}_{\mathrm{SFT}}}_{\text{student supervision}},$$

where only solver-verified y^T is retained. This construction is central to OptClimber because the teacher can exploit accumulated experience while the student must recover the corresponding modeling behavior from the original problem alone. Consequently, accumulated experience is transferred from external context into model capability rather than becoming a required inference-time input.