Opt-Aligner: Two-Stage Experience Transfer via Cognitive Alignment for Optimization Modeling

Anonymous Authors[1]

Abstract

Recent advancements in Large Language Models (LLMs) for optimization modeling have shown promising potential. To solve the knowledge-intensive problems, existing methods primarily rely on passive knowledge injection, such as fine-tuning or retrieval on an existing database. Inspired by the human learning paradigm that individuals actively acquire knowledge from problem-solving attempts, we explore a closed-loop two-stage framework, where a reader actively requests experiences from the writer when encountering failures. However, this active process faces a critical bottleneck: cognitive misalignment, where the writer's sophisticated guidance exceeds the reader's cognitive boundaries, leading to failure. To address this challenge, we propose Opt-Aligner, a framework that adaptively tailors the experience transfer process to align with the reader's capacity. We reframe the interaction as an Experience Reinforcement Learning (ERL) problem. Acting as an adaptive agent, the writer utilizes Q-values to estimate the effectiveness and accessibility of the generated experiences. By iteratively reflecting on the reader's success and failure trajectories, the writer optimizes a hierarchical alignment instruction to maximize the expected utility of the guidance. With only an 8B model, we find that a lightweight reader can achieve comparable performance to the high-capacity 671B writer, significantly outperforming existing passive and active methods.

1. Introduction

Optimization models are the critical part in operations research (OR), supporting a wide range of modern industrial applications, from chip design (Ma et al., 2019) to energy system scheduling (Muckstadt & Wilson, 1968). Typically, solving complex OR problems requires transforming unstructured user requirements into structured optimization models. However, this process is often time-consuming and costly, demanding extensive human expertise. In recent years, Large Language Models (LLMs) have revolutionized this process by leveraging their powerful semantic understanding capabilities to directly generate the optimization model and executable solver code.

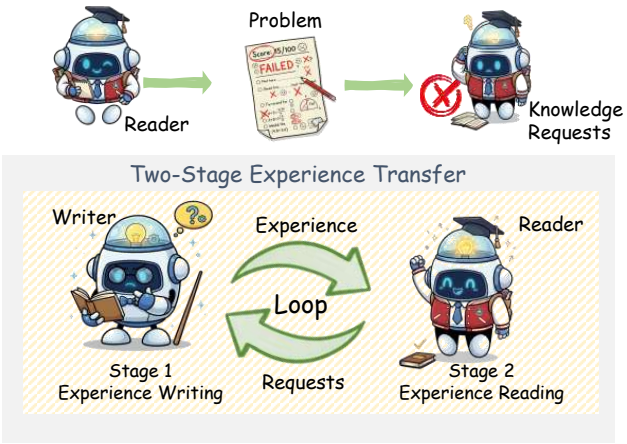

Figure 1. The two-stage knowledge acquisition paradigm.

To further improve the capability of LLMs, domain knowledge is injected into them in recent works. They attempt to fine-tune the LLM with domain-specific data (Huang et al., 2025; Jiang et al., 2025; Lu et al., 2025; Chen et al., 2025) or construct domain-specific knowledge bases for retrieval (Liu et al., 2025). While these methods have achieved notable success, the LLMs remain passive in acquiring domain knowledge. Inspired by the human learning paradigm—where individuals actively acquire relevant knowledge for problem-solving—this work embarks on an initial exploration of applying this learning paradigm to optimization modeling. We propose a closed-loop framework that enables LLMs to actively acquire the necessary knowledge to effectively tackle optimization modeling problems. Our approach comprises two key stages: 1) **experience writing**: an LLM, known as the writer, summarizes and

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

produces domain-specific experiences (implemented as hierarchical natural languages) specifically for a given task. 2) **experience reading**: another LLM referred to as the reader, which can be the same as the writer, leverages the knowledge from experiences to revise its behavior and handle new requests. The two-stage design allows to allocate jobs in each stage to the most suitable LLM. Specifically, the experience-writing process, which requires a high degree of capability, can be assigned to a powerful but resource-intensive model (e.g., GPT-4o or DeepSeek-R1). In contrast, the experience-reading process, which is comparatively simpler, can be assigned to a lightweight and cost-effective model (e.g., Qwen3-4B). This approach not only enhances the problem-solving capabilities of LLMs, but also significantly reduces the average computational cost of addressing a series of optimization modeling problems.

In the above framework, we further identify a critical bottleneck arising from misalignment when there is a significant capability gap between the writer and the reader. For example, the high-capacity writer may write an optimization modeling experience, like "use the MTZ approach to eliminate subtours to prevent closed loops", which could exceed the interpretive capacity of a less capable reader. We refer to this as the **cognitive misalignment problem**, which hinders the efficiency of the experience writing and reading process. In such cases, the writer is forced to iteratively revise the experience until the reader is able to understand it. To address this challenge, we propose Opt-Aligner, a strategy that adaptively tailors the experience writing process to align with the reader's cognitive boundaries. To this end, we frame the experience transfer process as an Experience Reinforcement Learning (ERL) problem, where the quality of the written experience is assessed based on its accessibility and effectiveness, using a Q-value (expected utility). By dynamically analyzing the reader's success and failure cases, the writer iteratively optimizes a hierarchical textual meta-policy. This process maximizes the Q-values of the generated experiences, ensuring that the guidance provided is both executable and comprehensible for the reader.

Our main contributions are summarized as follows:

- We introduce a novel two-stage framework for experience transfer in optimization modeling, consisting of a writer and a reader.
- We propose Opt-Aligner to address the cognitive misalignment in the two-stage framework, enabling the writer to act as an adaptive agent that iteratively optimizes its alignment meta-policy to align with the reader's capacity.
- Extensive experiments across multiple mathematical modeling benchmarks demonstrate that Opt-Aligner significantly outperforms direct prompting methods and achieves performance comparable to or even surpassing supervised fine-tuning (SFT) baselines.

2. Related Work on LLM Optimization Modeling

LLMs have gained more and more attention in automated mathematical modeling, significantly lowering the barrier for modeling complex real-world problems. Existing research primarily follows two trajectories: prompt-based workflows and fine-tuning-based specialized models. Prompt-based approaches leverage frontier commercial models through sophisticated frameworks. For instance, CoE (Xiao et al., 2024) and OptiMUS (Ahmaditeshnizi et al., 2024) utilize multi-agent orchestration for iterative code generation, while search-oriented methods like MCTS (Astorga et al., 2025). These works rely on static prompts, while recent research focuses on leveraging LLMs to collect and utilize knowledge from past problem-solving processes, such as retrieval-augmented systems like OptiTree (Liu et al., 2025) and experience learning like AlphaOPT (Kong et al., 2025). Conversely, fine-tuning-based methods, such as ORLM (Huang et al., 2025), LLMOPT (Jiang et al., 2025), OptMATH (Lu et al., 2025), and SIRL (Chen et al., 2025) focus on distilling modeling expertise into open-source models via supervised training on large-scale optimization datasets.

3. Two-Stage Experience Transfer Framework for Active Knowledge Acquisition

Two-Stage Experience Transfer Framework The two-stage experience learning framework is inspired by the human learning paradigm, where individuals actively acquire relevant knowledge from their successful or failed historical problem-solving trajectories. We propose a closed-loop system involving two roles: a Writer $\mathcal{T}$ and a Reader $\mathcal{S}$. The process begins when the reader fails to solve a problem and requests for relevant knowledge. Then, the writer provides external guidance to help the reader for problem-solving. More specifically,this process involves two stages:

- **Experience Writing**: After receiving the request, the Writer $\mathcal{T}$ acts as the knowledge producer. For a given problem p, it analyzes the reader's previous failure reasoning trajectory τ and the ground-truth y^*. Conditioned on a learnable generation control parameters θ, the Writer $\mathcal{T}$ generates a textual experience e via its policy $\pi_{\mathcal{T}}(e|p, \tau, y^*; \theta)$. These generated experiences are accumulated into an experience library $\mathcal{M}$.
- **Experience Reading**: The Reader $\mathcal{S}$ acts as the executor. It retrieves relevant experiences from the experience library $\mathcal{M}$ to augment its context and generates the mathematical formulation and solution $y \sim \pi_{\mathcal{S}}(\cdot|p, \mathcal{M})$. If

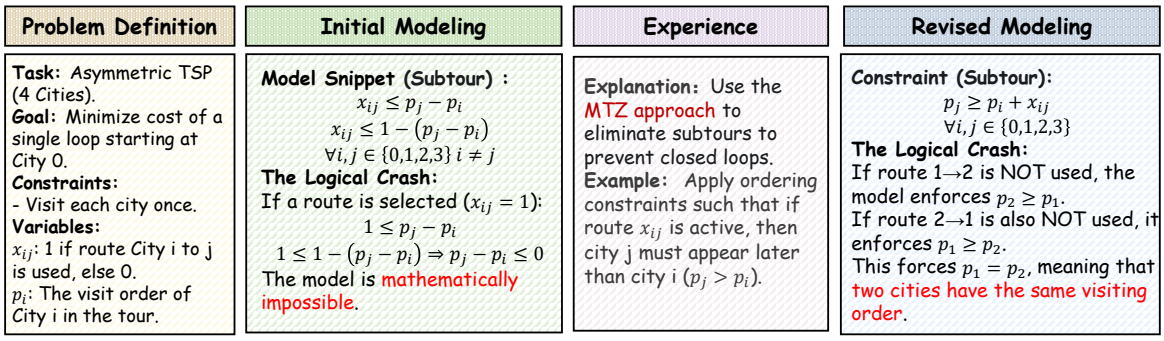

Figure 2. Case studies illustrating cognitive misalignment between the high-capacity writer and weaker reader models. The writer's high-level instruction on "MTZ constraints" leads to concept misunderstanding by the novice reader. The example highlights that unaligned writer guidance often induces modeling failures rather than corrections.

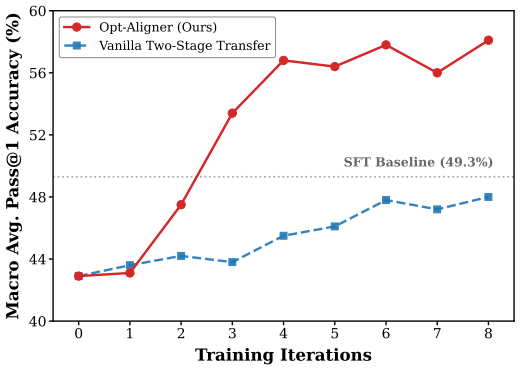

Figure 3. The evolution of the performance as the knowledge acquisition process proceeds.

the reader still fails to solve the problem, it is allowed to make a new request to acquire additional relevant knowledge until a solution is found.

The two-stage design allows us to assign the most suitable models for each stage. The experience writing stage requires analyzing complex failure patterns, which demands a high-capacity model. In contrast, experience reading stage primarily involves following existing guidance, which is simpler and can be handled by a lightweight model. This design leverages the writer's capability to build the library while ensuring the reader remains computationally efficient.

Challenge: Cognitive Misalignment Despite the potential of the two-stage framework, we identify a critical bottleneck: cognitive misalignment. This occurs when the high-capacity writer's guidance exceeds the lightweight reader's capabilities. As shown in Figure 2, the writer often relies on high-level directives (e.g., "apply MTZ constraints") that assume implicit domain knowledge. However, the reader lacks this foundation and fails to translate the abstract concepts into executable code. This suggests that simply writing experience is insufficient; the guidance must be cognitively aligned with the reader's capacity. Furthermore, we observe a vanilla writer struggles to improve the accuracy of the reader (shown in Figure 3), especially during the early iterations. This observation confirms our hypothesis and motivates us to accelerate the cognitive alignment process by the writer-reader interactions.

4. Opt-Aligner: Cognitive Alignment in Two-Stage Experience Transfer

In this part, we introduce Opt-Aligner, an approach designed to bridge the cognitive gap in the two-stage experience transfer process. The overall framework is illustrated in Figure 4. In Section 4.1, we formulate the interaction between the writer and reader as an experience reinforcement learning (ERL) problem, incorporating a cognitive alignment reward function. Furthermore, we design an alignment instruction for the writer in Section 4.2. This alignment instruction, along with aligned experiences, will be iteratively updated to maximize the alignment reward in Section 4.3.

4.1. Two-Stage Experience Transfer as ERL

We formulate the experience transfer process as a Markov decision process (MDP). In this setting, the writer acts as the *agent* optimizing its experience generation policy, while the reader, along with a solver constitute the environment.

MDP Formulation. We define the MDP with a tuple $\langle \mathcal{X}, \mathcal{A}, \mathcal{R} \rangle$. Each state $p_t \in \mathcal{X}$ in the state space is the specific optimization problem instance encountered at step t. The writer's policy $\pi_{\mathcal{T}}$ takes the current problem p_t (and the reader's request) as input to produce a natural language experience e_t, which serves as an action in the action space $\mathcal{A}$. After that, the reader receives the experience e_t, generates code, and attempts to solve the problem. The environment then returns a scalar reward $r_t \in \mathcal{R}$, which evaluates the quality of the written experience e_t.

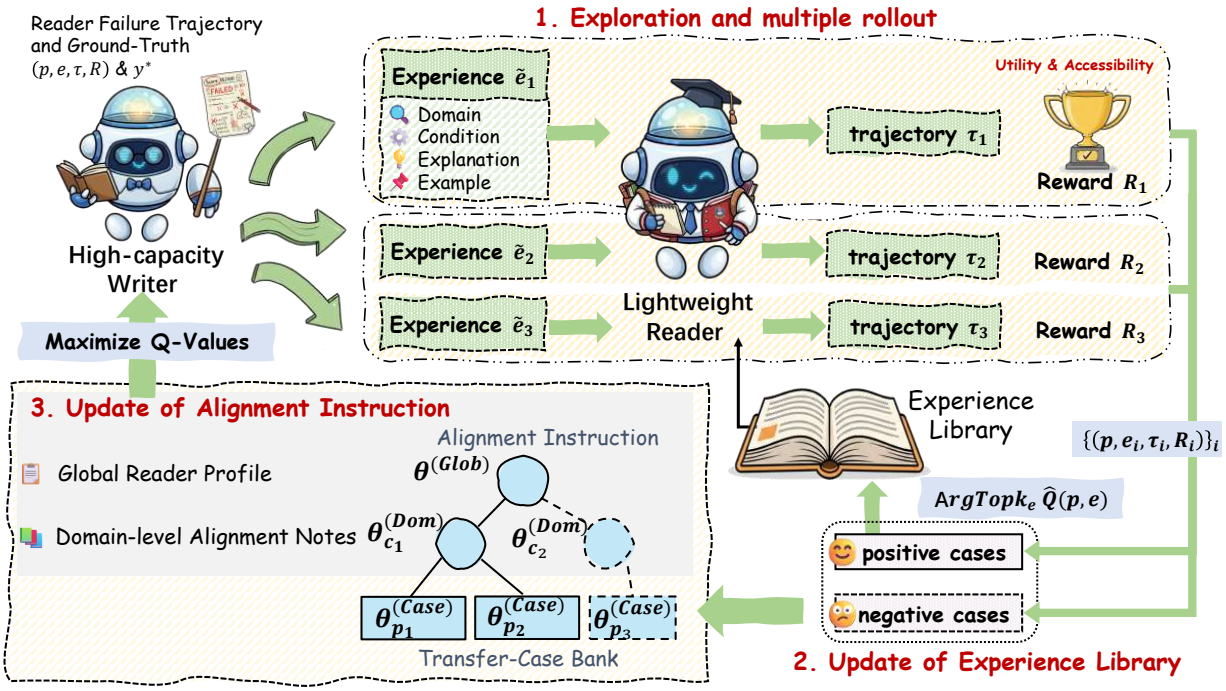

Figure 4. The Opt-Aligner framework. The cognitive alignment process is formulated as an experience reinforcement learning problem.

Cognitive Alignment Reward Function To ensure the experience is both helpful for problem-solving and easy for the reader to understand, we design a composite reward:

$$\mathcal{R}(p, e) = \mathcal{U}(p, e) - \beta \cdot \mathcal{I}(e) \tag{1}$$

where β is a non-negative coefficient balancing the two terms. The task utility term $\mathcal{U}(p, e) = \mathbb{I}(\hat{y}_e, y^*) - \mathbb{I}(\hat{y}, y^*)$ measures the actual performance improvement brought by the experience e. It is calculated as $\mathcal{U}(p, e) = \mathbb{I}(\hat{y}_e, y^*) - \mathbb{I}(\hat{y}, y^*)$, where $\mathbb{I}$ is an indicator function that equals 1 if the solution matches the ground truth y^* and 0 otherwise. Here, $\hat{y}_e$ is the solution generated with the experience e, and $\hat{y}$ is the solution generated without it. Simultaneously, the accessibility term $\mathcal{I}(e)$ measures how "readable" the experience is for the reader. We compute it using the perplexity of the experience tokens $w_{1:|e|}$ on the Reader $\mathcal{S}$: $\mathcal{I}(e) = \exp\left(-\frac{1}{|e|}\sum_{k=1}^{|e|} \log P_{\mathcal{S}}(w_k \mid w_{<k})\right)$. A lower perplexity implies that the written experience aligns well with the reader's cognitive boundaries.

Optimization Objective Building on the MDP formulation and reward function described above, a direct optimization objective is to maximize the expected cumulative reward by finding the optimal experience e^*:

$$e^* = \arg\max_{e \in \mathcal{A}} \mathbb{E}\left[\sum_{t=0}^{N} \mathcal{R}(p_t, e)\right].$$

However, the action space $\mathcal{A}$ is prohibitively large, making direct optimization challenging. To address this, we introduce an alternative optimization objective that seeks to maximize the expected cumulative reward by finding both the optimal experience e^* and the optimal experience generator parameters θ of the writer:

$$e^*, \theta^* = \arg\max_{e \in \mathcal{A}, \theta \in \Theta} \mathbb{E}\left[\sum_{t=0}^{N} \mathcal{R}(p_t, e, \theta)\right], \tag{2}$$

where the experience is governed by the writer parameters θ, as $e \sim \pi_{\mathcal{T}}(\cdot|p_t, \tau_t, y_t^*; \theta)$. By adopting this alternative objective, we can define a compact parameter space Θ for the writer, which not only reduces the optimization complexity but also facilitates the generation of aligned experiences.

4.2. The Parameter Space Design for Writer

We consider a hierarchical language instruction as the parameter space of the writer, structured across **case-level**, **domain-level**, and **global-level** perspectives:

- **Case-level Bank.** In the case-level, a transfer-case bank $\left\{\theta_p^{(Case)}\right\}$ acts as a memory buffer that archives case-level alignment notes derived from individual problems. For a specific problem p, the writer performs a contrastive analysis—comparing successful reader problem-solving trajectories against failed ones—to generate a case-level alignment note $\theta_p^{(\text{case})}$.

- **Domain-level Alignment Notes.** The domain-level notes $\left\{\theta_c^{(dom)}\right\}$ represent the writer's high-level strategy, which is iteratively refined based on the notes in the transfer-case bank. Each note $\theta_c^{(\text{dom})}$ is maintained for each problem domain $c \in \mathcal{C}$, such as the scheduling problems and routing problems.

- **Global-level Profile.** In the global level, a global reader profile $\theta^{(\text{glob})}$ captures the writer's understanding of the reader's general cognitive boundaries. It is categorized into two dimensions: (1) mastered knowledge: modeling logic that the reader can reliably implement across all domains; and (2) cognitive gaps: complex optimization concepts that remain inaccessible to the reader.

4.3. Overall Training Loop of Experience Reinforcement Learning

Exploration and Rollout. For a given problem p_t at step t, the reader first attempts to solve it using relevant experiences $\mathcal{M}^{p_t}$ retrieved from the current library $\mathcal{M}_t$. If the reader fails (yielding a low reward), the writer intervenes to explore better guidance. Conditioned on the reader's failure trajectory τ_t and the ground truth y_t^*, the writer generates a set of new candidate experiences $\tilde{\mathcal{M}}^{p_t} = \{\tilde{e}_1, \ldots, \tilde{e}_M\}$. The reader then executes these candidates to verify their effectiveness, producing an immediate reward $\tilde{R}$ for each new experience.

Library Update. Based on the rollout results, we update the experience library to reflect the latest evaluations. We maintain a Q-value estimate $\hat{Q}(p, e)$ for each experience, which is updated as follows:

$$\hat{Q}_{t+1}(p_t, e) = \begin{cases} \tilde{R}, & \text{for } e \in \tilde{\mathcal{M}}^{p_t}; \\ (1-\eta)\hat{Q}_t(p_t, e) + \eta R, & \text{for } e \in \mathcal{M}^{p_t}, \end{cases} \tag{3}$$

where $\eta \in (0, 1]$ is a coefficient balancing historical and current feedback. For existing experiences that failed to guide the reader, the reward R is set to 0, lowering their Q-values. For the new candidates, we identify the most effective ones and add them to the library:

$$\mathcal{M}_{t+1} = \mathcal{M}_t \cup \{\underset{\tilde{e} \in \tilde{\mathcal{M}}^{p_t}}{\text{argTopk}}\, \hat{Q}_{t+1}(p_t, \tilde{e})\}. \tag{4}$$

This mechanism allows the library to dynamically expand with successful new experiences while down-weighting ineffective ones. Then, all the rollout trajectories are sent into the transfer-case bank.

Update of Alignment Instruction We update the writer parameter θ by analyzing the rollout trajectories collected during the library updating. This update process follows a bottom-up hierarchical structure, refining the instructions from specific cases to the global profile.

(1) Case-Level Reflection. For each problem $p_t \in \mathcal{X}$, we partition the generated trajectories from the transfer-case bank into positive exemplars $\mathcal{C}_p^+$ (high Q-value, successful solution) and negative exemplars $\mathcal{C}_p^-$ (low Q-value, failed solution). The writer compares these contrastive sets to identify the specific causes of the reader's failure and generates a case-level alignment note:

$$\theta_p^{(\text{case})} = \text{LLM}_{\mathcal{T}}(\mathcal{C}_p^+, \mathcal{C}_p^- \mid \text{Case Reflection Prompt}). \tag{5}$$

(2) Domain-Level Summary. To generalize from individual cases, we aggregate the case-level notes into domain-specific knowledge. For each problem category c, the writer updates the domain-level alignment note $\theta_c^{(\text{dom})}$ by merging the new case notes with the existing domain knowledge:

$$\theta_{c,t+1}^{(\text{dom})} = \text{LLM}_{\mathcal{T}}\big(\theta_{c,t}^{(\text{dom})}, \{\theta_p^{(\text{case})}\}_{p \in \mathcal{X}_c} \mid \text{Summary Prompt}\big). \tag{6}$$

(3) Global Profile Evolution. Finally, the writer synthesizes the updated domain notes to refine the global reader profile $\theta^{(\text{glob})}$, which captures the reader's cognitive boundaries:

$$\theta_{t+1}^{(\text{glob})} = \text{LLM}_{\mathcal{T}}\left(\theta_t^{(\text{glob})}, \{\theta_{c,t+1}^{(\text{dom})}\}_{c \in \mathcal{C}} \mid \text{Update Prompt}\right). \tag{7}$$

This update rule ensures that the writer continuously adapts its guidance strategy to the reader's evolving capabilities.

4.4. Reader Inference with Experience Library

Once the experience transfer process is complete, the reader utilizes the experience library $\mathcal{M}$ to enhance its reasoning for a given optimization problem model p. First, the reader retrieves the top-k relevant experiences $\{e_i^*\}_{i=1}^k$ based on semantic similarity to the current problem context. The reader then integrates $\{e_i^*\}_{i=1}^k$ with the original problem description to form an augmented context. Conditioned on this combined input, the reader policy $\pi_{\mathcal{S}}$ generates the mathematical modeling code $\hat{y}$, formally denoted as $\hat{y} \sim \pi_{\mathcal{S}}(\cdot \mid p, \{e_i^*\}_{i=1}^k)$. Finally, the generated code is executed by an external solver to verify its validity and correctness.

5. Experiments

5.1. Experiment Setups

Dataset We evaluate our proposed Opt-Aligner and the baselines across a wide range of optimization modeling benchmarks. The Nl4Opt (Ramamonjison et al., 2021) contains elementary-level linear programming. The MAMO dataset (Huang et al., 2024), which includes both easy and harder splits (EasyLP and ComplexLP), is also a popular benchmark used in previous works. IndustryOR (Huang et al., 2025) featuring real-world industrial cases. Resocratic is proposed in (Yang et al., 2025). We also compare

Table 1. Comparison of our methods and the baselines on six modeling benchmarks. The numerical value represents the Pass@1 modeling accuracy (%). The star * means the results are cited from previous papers, and - are missing data due to reproduction difficulties.

Method	NL4Opt	MAMO		IndustryOR	Resocratic	OptMATH	Macro Avg
		EasyLP	ComplexLP				
Specialized Modeling Agent							
Prompt-Based Methods							
CoE* (GPT-4o)	64.2	-	40.2	-	-	-	-
OptiMUS* (GPT4)	78.8	77.0	43.6	31.0	45.8	20.2	49.4
Finetuned Methods							
ORLM (8B)	85.7	82.3	37.4	38.0	51.1	2.6	49.5
LLMOPT (14B)	80.3	89.5	44.1	29.0	53.8	12.5	51.5
OptMATH (7B)	94.7	86.5	51.2	20.0	57.9	24.4	55.8
High-Capacity LLMs							
GPT-4o	83.4	87.7	45.2	38.0	49.4	20.5	54.0
DeepSeek-V3	86.1	95.2	53.2	36.0	55.1	23.4	58.2
Knowledge Acquistion Paradigms (Using DeepSeek-V3 as Writer)							
DeepSeek-V3 Reader							
Two-Stage	92.0	95.7	51.6	36.0	56.8	24.0	59.3
Ours	**93.1**	**95.9**	**55.0**	**38.0**	**58.0**	**24.7**	**60.8**
Qwen3-4B-Instruct Reader							
Direct Output	74.1	65.3	15.2	19.0	40.8	22.9	39.6
Distillation	72.3	76.3	41.7	33.0	42.5	22.9	48.2
Two-Stage	81.3	80.3	38.3	31.0	48.8	23.4	50.5
Ours	**84.8**	**85.7**	**45.9**	**35.0**	**50.4**	**25.3**	**54.5**
Qwen3-8B Reader							
Direct Output	77.5	74.6	21.3	24.0	43.7	16.2	42.9
Distillation	73.0	77.5	44.1	35.0	44.0	20.5	49.0
Two-Stage	86.9	81.3	46.9	29.0	53.6	19.9	52.9
Ours	**91.0**	**90.2**	**50.7**	**36.0**	**57.3**	**23.4**	**58.1**

the methods on a challenging OptMATH benchmark (Lu et al., 2025). The performance on these benchmarks can reflect the modeling performance from easy to hard problems.

Baselines We compare Opt-Aligner against three categories of baselines: (1) Specialized optimization modeling methods, covering prompt-based systems (CoE (Xiao et al., 2024), OptiMUS (Ahmaditeshnizi et al., 2024)) and SFT fine-tuned models (OptMATH (Lu et al., 2025), LLMOPT (Jiang et al., 2025)) tailored for mathematical optimization tasks; (2) Strong LLMs, including GPT-4o (OpenAI, 2024) and DeepSeek-V3 (DeepSeek-AI et al., 2025); (3) Finally, to validate the efficacy of our active alignment approach, we assess alternative knowledge acquisition paradigms for the reader. This category includes passive knowledge acquisition, Knowledge Distillation (Distillation), and the active acquisition baseline vanilla two-stage method (Two-Stage).

Implementation We use 400 instances selected from the OptMATH training dataset for all the experience learning methods. The writer updates its alignment instructions every 50 steps (50 problems), the rollout number is set to 3, and the iteration number is set to 1. For the DeepSeek-V3 reader for which we cannot obtain the model output token logits, we use just the task utility in the reward function.

Metric Following prior work on LLM-based optimization modeling (Ahmaditeshnizi et al., 2024; Lu et al., 2025; Xiao et al., 2024; Astorga et al., 2025), we evaluate the modeling accuracy. A generated model is considered accurate if and only if its optimal objective value (computed by Gurobi) matches the ground-truths.

5.2. Main Results

Table 1 presents the comparative results across six benchmarks. The results demonstrate that Opt-Aligner effectively bridges the gap between the high-capacity writer and the lightweight reader. (1) Opt-Aligner consistently outperforms other knowledge acquisition methods, including Distillation and the vanilla Two-Stage framework. Notably, it achieves higher accuracy than Knowledge Distillation without requiring any parameter updates. (2) Despite being a training-free approach, our method with the Qwen3-8B reader outperforms specialized models that were fully fine-tuned on optimization datasets, such as ORLM and

Table 2. Ablation studies on Different writers and readers.

Method	NL4Opt	MAMO		IndustryOR	Resocratic	OptMATH
		EasyLP	ComplexLP			
DeepSeek-V3 Reader						
Ours (Deepseek-V3 Writer)	93.1	95.9	55.0	38.0	58.0	24.7
Qwen3-4B-Instruct Reader						
Two-Stage (GPT-4o Writer)	73.3	77.4	26.1	32.0	46.0	22.3
Ours (GPT-4o Writer)	82.7	**86.6**	42.1	**35.0**	**51.6**	24.1
Two-Stage (Deepseek-V3 Writer)	81.3	80.3	38.3	31.0	48.8	23.4
Ours (Deepseek-V3 Writer)	**84.8**	85.7	**45.9**	**35.0**	50.4	**25.3**
Qwen3-8B Reader						
Two-Stage (GPT-4o Writer)	84.1	87.0	43.6	34.0	49.1	18.7
Ours (GPT-4o Writer)	88.2	89.5	49.2	**37.0**	55.7	21.6
Two-Stage (Deepseek-V3 Writer)	86.9	81.3	46.9	29.0	53.6	19.9
Ours (Deepseek-V3 Writer)	**91.0**	**90.2**	**50.7**	36.0	**57.3**	**23.4**

Table 3. Ablation studies on components in the reward function. We use Qwen3-8B as the reader and DeepSeek-V3 as the writer.

Method	NL4Opt	MAMO		IndustryOR	Resocratic	OptMATH
		EasyLP	ComplexLP			
Opt-Aligner	91.0	90.2	50.7	36.0	57.3	23.4
Opt-Aligner w/o Task Utility	83.7	77.1	39.8	32.0	45.6	19.3
Opt-Aligner w/o Accessibility	87.1	86.5	47.9	35.0	56.8	22.9

LLMOPT. (3) Opt-Aligner enables the lightweight reader to reach performance levels comparable to its teacher. The Qwen3-8B reader even shows comparable performance to the powerful DeepSeek-V3 writer.

5.3. Impacts of Different Writers and Readers

We investigate the generalization capability of Opt-Aligner by varying the writer and reader models. Table 2 presents the results using GPT-4o and DeepSeek-V3 as writers, and Qwen3-4B and Qwen3-8B as readers. First, Opt-Aligner consistently outperforms the vanilla two-stage transfer method across all combinations. This indicates that our alignment mechanism effectively improves knowledge transfer regardless of the specific models used. Second, our framework reduces the capability requirements for the reader model. With the aligned experiences, the lightweight Qwen3-8B reader achieves performance comparable to the powerful DeepSeek-V3 writer.

5.4. Ablations Studies for Opt-Aligner

In this part, we use the DeepSeek-V3 as the writer and Qwen3-8B as the reader for ablation studies.

5.4.1. Impact of the Reward Function

We investigate the impact of the reward function. As presented in Table 3, removing the accessibility term leads to a consistent performance decline. This validates that without this constraint, the writer tends to generate experiences that exceed the reader's knowledge boundaries. Conversely, omitting the task utility term results in an even more severe degradation, confirming that the optimization must remain grounded in the primary goal of solution correctness. Furthermore, the sensitivity analysis for the coefficient β in the left of Figure 5 demonstrates that a balanced configuration yields optimal performance, while the model consistently achieves high performance around $\beta = 1$.

5.4.2. Impact of the Hierarchical Structures in Alignment Instruction

The alignment instruction comprises a global reader profile and domain-specific alignment notes, and we evaluate their individual contributions in Table 5. Removing domain-level notes (w/o Dom) results in a substantial performance drop. This indicates that optimization modeling relies on specialized knowledge patterns, such as distinguishing subtour elimination in routing from precedence constraints in scheduling. Similarly, excluding the global profile (w/o Glob) degrades performance consistently across all datasets. This suggests that certain reader limitations, especially coding errors, are universal across the problems. A global profile enables the writer to transfer these insights to unseen domains, preventing the repetition of fundamental errors.

5.4.3. Learning Curves over the ERL Updates

The middle of Figure 5 illustrates the learning curves of modeling accuracy over ERL updates. We observe a rapid

Table 4. Cost statistics on the knowledge acquisition process, measured on DeepSeek-V3 as writer model.

Methods	Marco Avg Accuracy	Reader Output Token	Writer Call	Writer Output Token	Training Time	Parameter Update
Distillation (Qwen3-8B)	49.0	-	1.0	1353.2	10h 37min	True
Two-Stage (Qwen3-8B Reader)	52.9	3092.6	3.2	2858.6	1h 05min	False
Opt-Aligner (DeepSeek Reader)	60.8	4974.3	6.3	3978.0	2h 31min	False
Opt-Aligner (Qwen3-8B Reader)	58.1	5257.9	7.6	4386.8	1h 55min	False

Table 5. Ablation studies on the effects of hierarchical structures in alignment instructions. We use Qwen3-8B as the reader agent and DeepSeek-V3 as the writer agent.

Method	**NL4Opt**	**MAMO**		**IndustryOR**	**Resocratic**	**OptMATH**
		EasyLP	**ComplexLP**			
Opt-Aligner	91.0	90.2	50.7	36.0	57.3	23.4
Opt-Aligner w/o Dom	80.3	84.5	41.7	28.0	50.2	20.5
Opt-Aligner w/o Glob	88.6	87.7	45.0	33.0	54.8	22.3

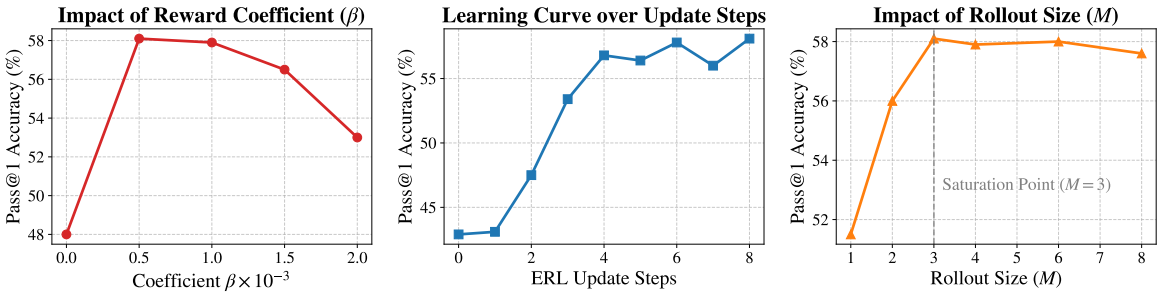

Figure 5: Analysis on the model performance (macro average of pass@1 accuracy) across the datasets. Left: Analysis on the reward coefficient β. Middle: The reader's performance evolves over the ERL process. Right: Analysis of the rollout number.

performance improvement in the early iterations, where the writer identifies the most prominent errors of the reader. Subsequently, the performance stabilizes as the alignment process proceeds. This demonstrates that Opt-Aligner efficiently constructs a robust experience generation policy within a limited number of interactions, making it computationally friendly for test-time adaptation.

5.5. Further Analysis

Cost Analysis We investigate the trade-off between performance and computational cost as detailed in Table 4. First, compared to Knowledge Distillation, which requires over 10 hours with GPU training, Opt-Aligner completes the policy adaptation in approximately 2 hours without requiring any parameter updates. This significantly saves time and eliminates the need for expensive GPU resources associated with training. Second, our framework facilitates the deployment of lightweight models in resource-constrained settings. The lightweight Qwen3-8B reader achieves a high accuracy of 58.1%, significantly outperforming the fine-tuned distillation baseline. Finally, while the iterative alignment process incurs higher token usage and writer calls compared to the vanilla two-stage method, this additional overhead is justified by the substantial performance improvement.

Rollout Analysis During the exploration and rollout process, the writer samples M candidate experiences $\tilde{\mathcal{M}}^{p_t}$ for the reader. We analyze the impact of the rollout size M on performance in the right of Figure 5. We observe that the performance improves as M increases, as a larger search space allows the writer to discover more effective experiences. However, the gain diminishes beyond $M = 3$, implying that the rollout and alignment process is efficient.

6. Conclusion

In this paper, we introduced Opt-Aligner, a two-stage experience transfer framework designed to bridge the cognitive gap between a high-capacity writer model and a weaker reader in optimization modeling. We identified cognitive misalignment as a critical bottleneck in existing experience transfer paradigms and addressed it by reframing the transfer process as an experience reinforcement learning problem. Extensive experiments across multiple benchmarks demonstrate that Opt-Aligner outperforms direct prompting and achieves competitive results against SFT methods, all while maintaining high computational efficiency. This work paves the way for the scalable deployment of lightweight models in complex industrial decision-making scenarios without the burden of model retraining.

Impact Statements

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

References

Ahmaditeshnizi, A., Gao, W., and Udell, M. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In Salakhutdinov, R., Kolter, Z., Heller, K., Weller, A., Oliver, N., Scarlett, J., and Berkenkamp, F. (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html.

Astorga, N., Liu, T., Xiao, Y., and van der Schaar, M. Autoformulation of mathematical optimization models using llms. In *International Conference on Machine Learning, ICML*, Proceedings of Machine Learning Research, 2025.

Chen, Y., Xia, J., Shao, S., Ge, D., and Ye, Y. Solver-informed rl: Grounding large language models for authentic optimization modeling, 2025. URL https://arxiv.org/abs/2505.11792.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report, 2025. URL https://arxiv.org/abs/2412.19437.

Huang, C., Tang, Z., Hu, S., Jiang, R., Zheng, X., Ge, D., Wang, B., and Wang, Z. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 2025. doi: 10.1287/opre.2024.1233. URL https://doi.org/10.1287/opre.2024.1233.

Huang, X., Shen, Q., Hu, Y., Gao, A., and Wang, B. Mamo: A mathematical modeling benchmark with solvers. *CoRR*, abs/2405.13144, 2024.

Jiang, C., Shu, X., Qian, H., Lu, X., Zhou, J., Zhou, A., and Yu, Y. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9OMvtboTJg.

Kong, M., Qu, A., Guo, X., Ouyang, W., Jiang, C., Zheng, H., Ma, Y., Zhuang, D., Tang, Y., Li, J., Wang, S., Koutsopoulos, H., Wang, H., Wu, C., and Zhao, J. Alphaopt: Formulating optimization programs with self-improving llm experience library, 2025. URL https://arxiv.org/abs/2510.18428.

Liu, H., Wang, J., Cai, Y., Han, X., Kuang, Y., and HAO, J. Optitree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=Ej20yjWMCj.

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

Ma, K., Xiao, L., Zhang, J., and Li, T. Accelerating an fpga-based sat solver by software and hardware co-design. *Chinese Journal of Electronics*, 28(5):953–961, 2019.

Muckstadt, J. A. and Wilson, R. C. An application of mixed-integer programming duality to scheduling thermal generating systems. *IEEE Transactions on Power Apparatus and Systems*, (12), 1968.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J. R., Mehrabian, A., Kumar, M. P., See, A., Chaudhuri, S., Holland, G., Davies, A., Nowozin, S., Kohli, P., and Balog, M. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Ouyang, S., Yan, J., Hsu, I.-H., Chen, Y., Jiang, K., Wang, Z., Han, R., Le, L. T., Daruki, S., Tang, X., Tirumalashetty, V., Lee, G., Rofouei, M., Lin, H., Han, J., Lee, C.-Y., and Pfister, T. Reasoningbank: Scaling agent self-evolving with reasoning memory, 2025. URL https://arxiv.org/abs/2509.25140.

Ramamonjison, R., Yu, T. T. L., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 Competition Track*, pp. 189–203, Virtual, 2021.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: language agents with verbal reinforcement learning. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2024. Curran Associates Inc.

Xiao, Z., Zhang, D., Wu, Y., Xu, L., Wang, Y. J., Han, X., Fu, X., Zhong, T., Zeng, J., Song, M., and Chen, G. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

Yang, L., Yu, Z., Zhang, T., Cao, S., Xu, M., Zhang, W., Gonzalez, J. E., and CUI, B. Buffer of thoughts: Thought-augmented reasoning with large language models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=ANO1i9JPtb.

Yang, Z., Wang, Y., Huang, Y., Guo, Z., Shi, W., Han, X., Feng, L., Song, L., Liang, X., and Tang, J. Optibench meets resocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fsDZwS49uY.

Ye, H., Wang, J., Cao, Z., Berto, F., Hua, C., Kim, H., Park, J., and Song, G. Reevo: Large language models as hyper-heuristics with reflective evolution. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=483IPG0HWL.

A. Further Related Work

Experiences for LLMs Experience learning aims to empower LLMs to accumulate and reuse historical problem-solving trajectories, thereby enhancing their reasoning robustness. Current research largely falls into two primary directions. (1) The first line of research focuses on the efficient archiving and recall of experiences. Methods such as Buffer of Thoughts (Yang et al., 2024) and ReasoningBank (Ouyang et al., 2025) store high-level thought templates or success/failure trajectories to prevent error repetition. (2) The second direction emphasizes the continuous refinement of experiences. To achieve self-improvement without parameter updates, approaches like AlphaEvolve (Novikov et al., 2025), ReEvo (Ye et al., 2024), and Reflexion (Shinn et al., 2024) introduce evolutionary algorithms. Specifically in the context of optimization modeling, AlphaOpt (Kong et al., 2025) constructs a self-improving library by reflecting on solver feedback. However, a critical limitation of these works is their reliance on strong backbone LLMs, explicitly presupposing extensive domain knowledge and advanced reasoning capabilities that are typically absent in smaller models.

B. Analysis on the Experience Library

In this section, we analyze the experience library $\mathcal{M}$ of the reader constructed by Opt-Aligner during the training phase. We focus on its statistical characteristics, the expansion of experiences over time, and its data efficiency compared to standard transfer paradigms.

B.1. Statistics of the Experiences

Table 6 presents the detailed statistics of the experience library derived from the 400 training instances. Only 563 experiences were retained from the exploration process, indicating that the framework effectively filters out biased guidance.

Table 6. Statistics of the experience library.

Metric	Value
Total Training Instances	400
Library Size	563
Avg. Token Length per Experience	687.1

B.2. Library Expansion

Figure 5 illustrates the number of accumulated experiences in the library over the training process. In the early stage, the library grows rapidly as the writer LLM actively identifies and corrects the reader's fundamental modeling errors. As training proceeds, the growth rate slows down.

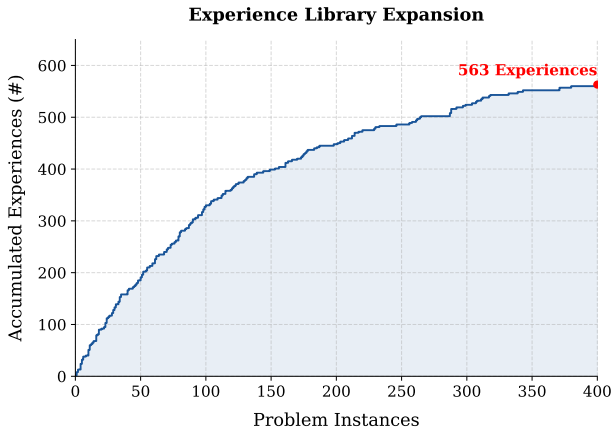

Figure 5. The number of accumulated experiences in the library over the training process

C. Analysis on the Transfer-Case Bank and Alignment Instruction

In this section, we provide a detailed statistical analysis of the transfer-case bank and the alignment instruction.

C.1. Statistics of the Transfer-Case Bank

Table 7 summarizes the statistics of the transfer-case bank. The transfer-case bank stores 251 specific interaction cases, covering diverse failure modes. These cases were classified into 53 distinct domain-level alignment notes.

Table 7. Statistics of the transfer-case bank and alignment instructions.

Component	Count	Avg. Tokens	Role
Transfer Cases	251	5462.7	Raw Experience Replay
Case-level Notes	251	915.9	Specific Error Reflection
Domain-level Notes	53	459.8	Category-Specific Syllabus
Global Reader Profile	1	513	Universal Cognitive Boundary

C.2. Evolution of Domain-Level Knowledge

Figure 6 illustrates the growth of domain-level alignment notes over training iterations. In the early stages, the number of domain notes increases rapidly as the reader explores new problem types. As training progresses, the curve grows more slowly.

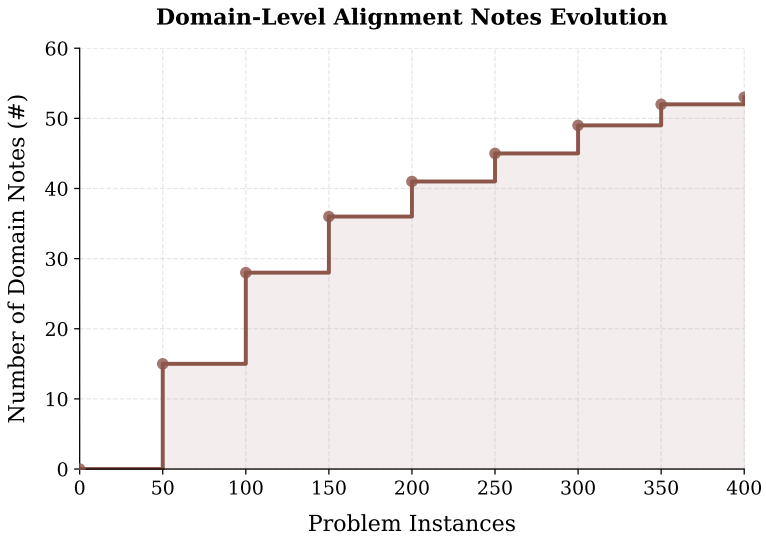

Figure 6. The evolution of the number of domain-level alignment notes during the training process.

D. Details of Implementations

D.1. Implementation of the Baselines

Knowledge Distillation This method transfers knowledge by training the LLM on trajectories generated by a writer LLM using SFT. We first prompt the writer LLM to solve the training problems and filter the outputs, retaining only those trajectories that pass the Gurobi solver check. We utilize the training split of the OptMATH dataset, which consists of approximately 5,000 high-quality optimization problems paired with ground-truth solutions. We use a learning rate of $2e-4$ and 20 epochs with a batch size of 16.

Vanilla Two-Stage In this process, the writer generates a modeling code for a given problem. If the code fails to execute or produces the correct objective value, the specific error message and the ground-truth process are fed back to the LLM, asking it to fix the error and summarize the experiences.

D.2. Implementation of Opt-Aligner

Model Configuration For the reader agent $\mathcal{S}$, we use Qwen3-4B-Instruct and Qwen3-8B-Instruct. For the writer agent $\mathcal{T}$, we utilize DeepSeek-V3 and GPT-4o. The writer operates with a temperature of 0.7 during the exploration phase to encourage diverse experience generation strategies, and reduces the temperature to 0.2 during the policy optimization phase to ensure stable summarization.

Hyperparameters We perform the policy update for $T = 50$ steps. During rollout, for each reader failure, the writer samples $M = 3$ candidate experiences. For the Q-Value update, we use an exponential moving average coefficient η of 0.1. To retrieve relevant experiences, we ask the reader to retrieve the top-5 experiences. The Gurobi Optimizer v9.5.2 was used for reward calculation.

E. Error Analysis

We provide a detailed error analysis to understand the limitations of our Opt-Aligner. Logical modeling errors are cases where the solver code can execute successfully but produces the wrong answer. Syntax errors are those with the code cannot run successfully. The experiments are conducted in the IndustryOR dataset.

Table 8. Distribution of error modes for the base model and Opt-Aligner.

Error Category	Logic Error	Syntax Error	Correct Answer
Base Model	44.0	37.0	19.0
Opt-Aligner (Ours)	39.0	26.0	35.0

F. Prompts for Opt-Aligner

An Example of Reader Profile


```
{"profile": {"summary": "The student has made progress in understanding logical
    constraints and variable definitions, but continues to struggle with translating
    verbal constraints into mathematical inequalities and correctly applying relational
     operators. Persistent misconceptions remain in Big-M formulations and inventory
    balance equations, indicating a need for targeted instruction in these areas.", "
    capabilities": {"mastered": [], "emerging": ["Logical Constraints", "Variable
    Definition"], "missing_or_misconceived": ["Correct use of relational operators in
    constraints", "Translation of verbal constraints into mathematical inequalities", "
    Understanding of non-strict inequalities for supply and demand constraints", "Big-M
     formulations", "Inventory balance equations", "Logical exclusions and implications
    "]}, "common_error_patterns": ["The student frequently misapplies relational
    operators, using < and > instead of \u2264 and \u2265 in supply and demand
    constraints.", "There is a recurring issue with translating verbal constraints into
     accurate mathematical inequalities.", "Misunderstanding of inventory balance and
    startup constraints due to unclear logical relationships between production, sales,
     and inventory.", "Struggles with correctly setting up Big-M constraints due to
    unclear guidance on selecting appropriate M values or understanding the
    relationship between binary decisions and continuous variables.", "Errors in
    logical exclusions and implications due to confusion about translating natural
    language into precise mathematical constraints."], "suggested_teaching_style": "Use
     concrete examples that clearly demonstrate the correct use of relational operators
     in constraints. Provide explicit guidance and illustrative examples to reinforce
    the translation of verbal constraints into mathematical inequalities, focusing on
    the importance of non-strict inequalities for supply constraints. Incorporate step-
    by-step walkthroughs of inventory and resource activation dynamics to clarify
    logical flows and interactions over time. Provide detailed examples of Big-M
    formulations and logical exclusions to ensure understanding of binary and
    continuous variable interactions."}}
```

G. Prompts for Opt-Aligner

Prompt of Experience Retrieval for Reader

```
You are an expert in Industrial Engineering and Operations Research.

A colleague has made a preliminary selection of potentially relevant **experiences**
    after analyzing the optimization task. Your job is to **carefully evaluate** each
    candidate and decide whether it truly applies.

You are given:
1. A problem description.
2. A collection of experiences, each with:
    - **domain**: the classification of the modeling, formulation, or code
    implementation area it belongs to.
    - **condition**: Statement of both when the experience **does apply (applicability
    condition)** and when it **does not (inapplicability condition)**, grounded in
    problem-specific context to prevent misuse.

### Problem description
{problem_description}

### Candidate experiences
{candidate_experiences}

### YOUR TASK
1. Carefully read the problem description, then:
    - Identify which problem domain(s) and modeling techniques are involved.
    - Analyze potential formulation pitfalls the problem may involve.
2. Evaluate each candidate experience one by one. **Only keep** those that directly
    apply to solving this specific problem. Be careful about the **inapplicability
    condition** that indicates exclusion scenarios where applying the experience would
    mislead; do not return this experience if the problem falls within those exclusion
    scenarios.
3. Remove redundancy: when multiple experiences overlap, keep only **the most relevant
    one(s)** based on their applicability condition.
4. Use the **exact experience_id** provided with each candidate; do not invent new IDs.

### STRICT OUTPUT FORMAT
Return only a JSON array of **the experiences you think are applicable**. Do not
    include explanations, markdown, or extra text.
Each array element must be an object with keys `"experience_id"` (integer) and `"reason
    "` (string).

Example:
```json
[
    {{"experience_id": 1, "reason": "<1-2 sentences explaining why it fits based on the
     condition>"}},
    {{"experience_id": 5, "reason": "<1-2 sentences explaining why it fits based on the
     condition>"}}
]
```

If no experience is applicable, return exactly:
[]
```

Prompt of Modeling for Reader

```
You are an expert in Industrial Engineering and Operations Research, who are proficient
     in Gurobi. Your task is to Translate the given mathematical model into a complete
    and executable Gurobipy program.

You are given:
1. A problem description for the optimization task.
2. A proposed mathematical model for solving this task.
3. A collection of insights, each with:
        - taxonomy: the classification of code-implementation area (level 1) and
    specific aspect/issue (level 2) it addresses.
        - condition: A trigger explicitly grounded in the mathematical model, when the
    insight should apply to avoid potential mistakes. It first states the general
    modeling pattern, then uses the specific model as an example.
        - explanation: A concise description that states the best practice, the common
    mistake, and its cause.
        - example: the demonstration showing wrong vs. correct version (principle,
    formula, or code snippet).


### Problem description
{problem_description}

### Mathematical model
{mathematical_model}

### Insights
(Note: The insights list may be empty. If it is empty, or none are applicable, do NOT
    fabricate any |insightsjust proceed normally without using insights.)
{insights}

### Your Task
First, review the given insights one by one, analyse whether each applies and how to
    apply it.
Second, following the guidance of applicable insights, produce the complete and
    runnable gurobipy code that **strictly adheres to the given mathematical model**.
Finally, **only output and enclose the code in a single Markdown-style Python code
    block** that starts with ```python and ends with ```, and follow the overall
    structure like this:

```python
import gurobipy as gp
from gurobipy import GRB
model = gp.Model("OptimizationProblem")
# your code from here
model.optimize()
```

Guidelines:
- Annotate your code with inline brief comments only if insights are provided and
    applicable: indicate the insight ID and how it influenced that specific code
    segment. If no insights are applicable, skip such annotations and do not invent IDs.

- Ensure model.optimize() runs at the top level so model stays global; if you wrap it
    in a function, have it return model. Avoid any if __name__ == "__main__": guard.
- Only output exactly fenced code block (delimited by the opening python and the
    closing); no text before or after.
- Ensure the objective's units are consistent with the problem statement. Apply any
    stated rounding rule in the description to the objective value.
- **DO NOT GENERATE OR MODIFY ANY CODE (e.g., `if model.Status == GRB.OPTIMAL:`) after
    `model.optimize()`**.
```

```
Now Take a deep breath and think step by step.
```

Prompt of Experience Generation for LLM

```
You are an expert in Industrial Engineering and Operations Research teaching graduate
    readers to avoid modeling-and-coding mistakes in solving optimization problems.

You are given:
1. A problem description for the optimization task
2. A mathematical model proposed by your colleague which failed to yield an optimal
    solution when solved with the Gurobi optimizer (hereafter referred to as *the
    failed mathematical model*)
3. The gold-standard program, which embodies the correct mathematical formulation of
    the optimization task
4. A reader profile and alignment strategy.


### Problem description
{problem_description}

### The failed mathematical model
(Note: the model is written in LaTeX and presented in a plain-text code block (```))
{failed_formulation}

### Raised Error
{error}

### The gold-standard program
{correct_program}

### Reader profile
{reader_profile}

### Alignment Note
{alignment_strategy}

### Your task
Step 1: Compare the failed mathematical model with the correct mathematical model
    embodied in the gold-standard program to identify issues that prevent optimality.
    Note that variable names in the proposed model may differ from those in the gold-
    standard program. Please align them carefully based on the problem specification.

Step 2: Using the insight domain dictionaries provided below, extract one or more new
    insights, which should be a distinct and concise lesson derived from a specific
    issue identified in the failed mathematical model relative to the gold-standard
    program.

Each new insight must contain exactly four fields:

1) **Domain** Problem Domain (e.g., "Network Flow")

2) **condition** | Write it as a trigger explicitly grounded in the problem description
     or in the defining features of the problem domain. First state the general
    situation, then use this problem as an example. **Use the pattern below**, and keep
     it strictly non-prescriptive: do not give any solution, advice or decision:
"This insight applies when ... For example, when the problem statement mentioned ...".

3) **explanation** | A brief and self-contained description that specifies, under this
    condition, what the best practice is, what the common mistake is and its cause.
    First, use this problem as an example to illustrate; Then, appropriately generalize
     the correct modeling strategy it reflects, if applicable.
```

```
**Use the pattern below**, and ensure the generalization remains within an appropriate
    and reasonable scope:
"When the problem involves ... . The best practice is ... . A common mistake is ... ,
    which happens because ... . More generally, this reflects that ... ."

4) **example** | A brief, self-contained demonstration showing wrong vs. correct
    version (principle, formulation, or code snippet). Clearly mark them as '# Wrong'
    and '# Correct'.

### domain Dictionaries
**Domain Modeling**
{domain_taxo}


### STRICT OUTPUT FORMAT
Return a single JSON **array** of insight objects. No text before/after. Example with
    two insights (but not must be two):

[
    {{
        "domain": {{
            <text>
        }},
        "condition": "<text>",
        "explanation": "<text>",
        "example": "<text>"
    }},

    {{
        "domain": {{
            <text>
        }},
        "condition": "<text>",
        "explanation": "<text>",
        "example": "<text>"
    }}
]

Now take a deep breath and think step by step.
```

Prompt of Case-Level Alignment Note Generation

```
You are an expert **Optimization Modeling Writer** acting as an adaptive agent. Your
    goal is to analyze the performance of a **Reader Model** to identify "cognitive
    misalignments|"situations where your high-level expert guidance exceeded the reader
    's reasoning capabilities, leading to failure.

You will be performing a **Contrastive Reflection**. You are provided with:
1.  **Problem Description**: The optimization task.
2.  **Positive Trajectories ($\mathcal{C}^+$)**: Instructions that led to correct
    solutions (or the ground truth if the reader failed).
3.  **Negative Trajectories ($\mathcal{C}^-$)**: Instructions that caused the reader to
     fail, including the generated code and the resulting error.

### Input Data
**Problem Description:**
{problem_description}

**Positive Trajectories (Successes/Ground Truth):**
{positive_trajectories}

**Negative Trajectories (Failures):**
{negative_trajectories}

### YOUR TASK
1.  **Analyze the Contrast**: Compare the logic in the Negative Trajectories against
    the Positive Trajectories.
2.  **Identify the Cognitive Gap**: Determine why the reader failed.
    *   Did the reader misunderstand a high-level abstract concept (e.g., "Use MTZ
    constraints")?
    *   Did a complex sentence structure cause a logical crash?
    *   Did the reader hallucinate constraints that don't exist?
3.  **Formulate the Alignment Note**: Summarize your findings into a structured note
    that includes:
    *   **Target Problem**: A concise summary of the specific problem context.
    *   **Critical Error Points**: The specific mathematical or logical errors the
    reader committed in the negative trajectories.
    *   **Linguistic/Logical Trigger**: The precise part of the instruction or problem
    logic that triggered the failure (e.g., "The phrase 'prevent subtours' caused the
    reader to hallucinate incorrect indices").
    *   **Instructional Adjustment**: The concrete correction needed. How must the
    instruction be simplified or explicitly detailed to ensure the reader can execute
    it? (e.g., "Replace abstract 'MTZ' terminology with explicit algebraic inequalities
    ").

### STRICT OUTPUT FORMAT
Return only a **single JSON object** containing the reflection. Do not include markdown
     formatting like ```json ... ```.

```json
{{
    "target_problem_summary": "Brief summary of the optimization task...",
    "critical_error_points": [
        "Description of error 1 (e.g., defined binary variable as continuous)",
        "Description of error 2 (e.g., logic mismatch in constraint A)"
    ],
    "failure_trigger": "The specific linguistic or logical cause (e.g., The reader
    lacks the knowledge base for 'Traveling Salesman Subtour elimination' and
    hallucinates when prompted with high-level keywords.)",
    "instructional_adjustment": "The corrective strategy (e.g., Do not use the term '
    MTZ constraint'. Instead, explicitly provide the formula: u_i - u_j + C*x_ij <= C -
     1.)"
}}
```
```

Prompt of Domain-Level Alignment Note Generation

```
You are an expert **Optimization Modeling Writer** acting as a meta-optimizer. Your
    goal is to synthesize specific problem-solving experiences into a generalized **
    Domain-Level Alignment Note**.

You are responsible for the **"{domain_name}"** domain. You will update the current
    domain knowledge based on a batch of new specific case studies.

You are given:
1.  **Current Domain-Level Note**: The existing summarized knowledge for this domain (
    if any).
2.  **New Case-Level Reflections**: A list of specific alignment notes derived from
    recent problems in this domain ($\{\theta^{(case)}_p\}$). Each note contains
    specific errors and successful correction strategies.

### Inputs
**Current Domain-Level Note:**
{current_domain_note}

**New Case-Level Reflections:**
{case_reflections}

### YOUR TASK
1.  **Synthesize and Abstract**: Analyze the *New Case-Level Reflections*. Identify
    recurring patterns in reader failures and effective corrections.
2.  **Update the Domain Knowledge**: Merge these new insights into the *Current Domain-
    Level Note*.
    - If the reader repeatedly fails at a specific concept (e.g., "Subtour Elimination"
     or "Time Window constraints") across different problems, promote this to a **
    Domain-Specific Weakness**.
    - Generalize specific "Instructional Adjustments" into a **Universal Instruction
    Strategy** for this domain (e.g., change "Fix indices for Problem A" to "Always
    explicitly define index sets for flow variables").
3.  **Maintain Structure**: Ensure the output strictly adheres to the four standardized
     components defined in the framework.

### STRICT OUTPUT FORMAT
Return only a **single JSON object**. Do not include markdown formatting, explanations,
     or extra text.

The JSON object must have exactly the following keys:
- `"domain_definition"`: A concise definition of the optimization problem class (e.g.,
    "Resource Constrained Scheduling Problems involving time slots and precedence").
- `"common_failure_modes"`: A list of recurring high-level error patterns observed in
    this domain (e.g., ["Confusing start/end time indices", "Hallucinating quadratic
    constraints for linear precedence"]).
- `"weak_knowledge_points"`: A list of specific optimization concepts the Reader Model
    consistently fails to grasp abstractly (e.g., ["Big-M formulation", "Traveling
    Salesman Subtour logic"]).
- `"summarized_instruction_strategy"`: The generalized alignment policy for this domain
    . How should the writer prompt the reader for *any* problem in this category to
    prevent the identified failures? (e.g., "Never use implicit set notation. Always
    explicitly expand logical implications into linear inequalities using binary
    indicator variables.")

Example:
```json
{{
    "domain_definition": "Vehicle Routing Problems (VRP) with capacity constraints.",
    "common_failure_modes": [
        "Incorrectly indexing flow variables across non-existent edges.",
        "Failing to enforce flow conservation at intermediate nodes."
```
```

```
    ],
    "weak_knowledge_points": [
        "MTZ Subtour Elimination constraints",
        "Vehicle capacity aggregation logic"
    ],
    "summarized_instruction_strategy": "Direct the reader to define the graph fully G=(
    V,E). Explicitly provide the algebraic form of subtour elimination. Do not rely on
    natural language descriptions like 'ensure no loops'."
}}
```

Prompt of Reader Profile Generation

```
You are an expert **Optimization Modeling Writer** acting as a pedagogical architect.
    Your goal is to evolve the **Global Reader Profile|**a universal representation of
    the Reader Model capabilities and limitations.

You will effectively "know the reader" by synthesizing specific domain observations
    into general truths.

### Inputs
**1. Current Global Reader Profile ($\theta^{(glob)}_t$):**
(The existing belief about the reader's general capacity)
{current_global_profile}

**2. Aggregated Domain-Level Alignment Notes ($\{\theta^{(dom)}_c\}$):**
(The latest summaries from specific domains like Routing, Scheduling, etc., containing
    specific failure modes and successful strategies)
{domain_level_notes}

### YOUR TASK
1.  **Cross-Domain Synthesis**: Analyze the *Domain-Level Alignment Notes* to find
    patterns that persist across different problem types.
    *   *Example:* If the reader fails at "conditional logic" in Scheduling AND "
    implication constraints" in Routing, this is a **Global Cognitive Gap**.
2.  **Verify Mastery**: Identify concepts the reader handles correctly across all
    domains (e.g., "Variable declaration syntax is always correct").
3.  **Update the Profile**: Evolve the *Current Global Reader Profile* based on new
    evidence.
    *   **Universal Cognitive Boundaries**: Separate what the reader *can* abstractly
    understand vs. what requires explicit mathematical hand-holding.
    *   **Global Constraints**: Identify limitations in context window, reasoning depth
    , or specific library syntax (e.g., "Always hallucinates non-linear constraints in
    Gurobi").
4.  **Define Meta-Instruction Policy**: Formulate a high-level style guide for *how* to
     teach this reader regardless of the problem (e.g., "Always break complex sentences
     into bullet points").

### STRICT OUTPUT FORMAT
Return only a **single JSON object**. Do not include markdown formatting or
    explanations.

The JSON object must have exactly the following keys:
- `"mastered_knowledge"`: A list of modeling logics or syntax patterns the reader
    reliably implements across ALL domains.
- `"universal_cognitive_gaps"`: A list of optimization concepts that are fundamentally
    inaccessible to the reader without explicit formulaic translation (e.g., "Logical
    implications," "Min-max linearization").
- `"parametric_limitations"`: Observations on the reader's reasoning bandwidth (e.g., "
    Cannot handle constraints with more than 3 nested logical conditions," "Prone to
    syntax errors when context exceeds 2000 tokens").
```

````
- `"global_meta_policy"`: The universal rule for interacting with this reader. (e.g., "
    Adopt a translation-heavy role. Do not describe the *logic* of a constraint;
    describe the *algebra*. Use standard inequalities rather than descriptive text.")

Example:
```json
{{
    "mastered_knowledge": [
        "Standard linear variable declarations (continuous, binary, integer)",
        "Basic summation constraints (sum(x) <= C)",
        "Correct Gurobi syntax for addVar and addConstr"
    ],
    "universal_cognitive_gaps": [
        "Conditional constraints (Big-M method) - consistently fails to calculate tight
     bounds",
        "Multi-objective scalarization - confuses weights",
        "Abstract set notation operations"
    ],
    "parametric_limitations": [
        "Fails to maintain consistency when variable names are similar (e.g., x_ij vs
    x_ji)",
        "Hallucinates 'lazy constraints' which are not supported by the simplified
    solver interface"
    ],
    "global_meta_policy": "Treat the reader as a code translator, not a problem solver.
     Provide the exact mathematical inequalities in LaTeX format for any logic
    involving 'if-then' or 'either-or'. Never use academic shorthand like 'Miller-
    Tucker-Zemlin'."
}}
````
````