Automated Reformulation of Robust Optimization via Memory-Augmented Large Language Models

Jinbiao Chen[1], **Shuang Jin**[2], **Guoyun Zhang**[3,4],
Junyu Zhang[1], **Guanyi Wang**[1,*], **Hanzhang Qin**[1,3,*]
[1]Department of Industrial Systems Engineering and Management,
National University of Singapore
[2]Department of Data and Systems Engineering, The University of Hong Kong
[3]Institute of Operations Research and Analytics, National University of Singapore
[4]Agency for Science, Technology and Research (A*STAR)
bill.cjb@nus.edu.sg, jinsh7@hku.hk, zhang.guoyun@u.nus.edu
junyuz@nus.edu.sg, guanyi.w@nus.edu.sg, hzqin@nus.edu.sg

Abstract

Robust optimization (RO) provides a principled framework for decision-making under uncertainty, but its practical use is often limited by the need to manually reformulate uncertain optimization models into tractable deterministic counterparts. Recent large language models (LLMs) have been shown promising for automating optimization formulation, yet RO reformulation remains challenging because it requires precise multi-step reasoning and mathematically consistent transformations. To facilitate systematic evaluation of LLM-based reformulation, for which no dedicated benchmark currently exists, we develop AutoRO-Bench, a benchmark featuring an automated data generation pipeline for the core RO reformulation task and a curated dataset for the RO application task. To address the reformulation challenge, we propose Automated Reformulation with Experience Memory (AutoREM), a tuning-free memory-augmented framework that autonomously builds a structured textual experience memory by reflecting on past failed trajectories through a tailored offline adaptation procedure. AutoREM requires neither domain-specific expert knowledge nor parameter updates, and the resulting memory readily transfers across different base LLMs. Experimental results show that AutoREM consistently improves the accuracy and efficiency of RO reformulation across in-distribution datasets, out-of-distribution datasets, and diverse base LLMs.

1 Introduction

Robust optimization (RO) has been widely studied for producing reliable solutions under uncertainty [1–4]. As illustrated in Figure 1, applying RO typically follows a three-stage pipeline of formulation, reformulation, and solving. Recently, large language models (LLMs) have demonstrated strong reasoning and coding capabilities [5, 6], which greatly advances the automation of the solving stage and shows promise for automating the formulation stage [7]. However, the reformulation stage remains a critical bottleneck: mathematically transforming uncertain models into tractable deterministic counterparts requires rigorous manual derivations grounded in convex analysis and duality theory, which severely limits the scalability of RO applications. This work, therefore, focuses on automating this reformulation stage with an LLM-based approach.

Recent methods for enhancing LLM reasoning are broadly divided into parameter-tuning and tuning-free approaches. Parameter-tuning methods, such as RL-based post-training, suffer from significant

*Corresponding Authors.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

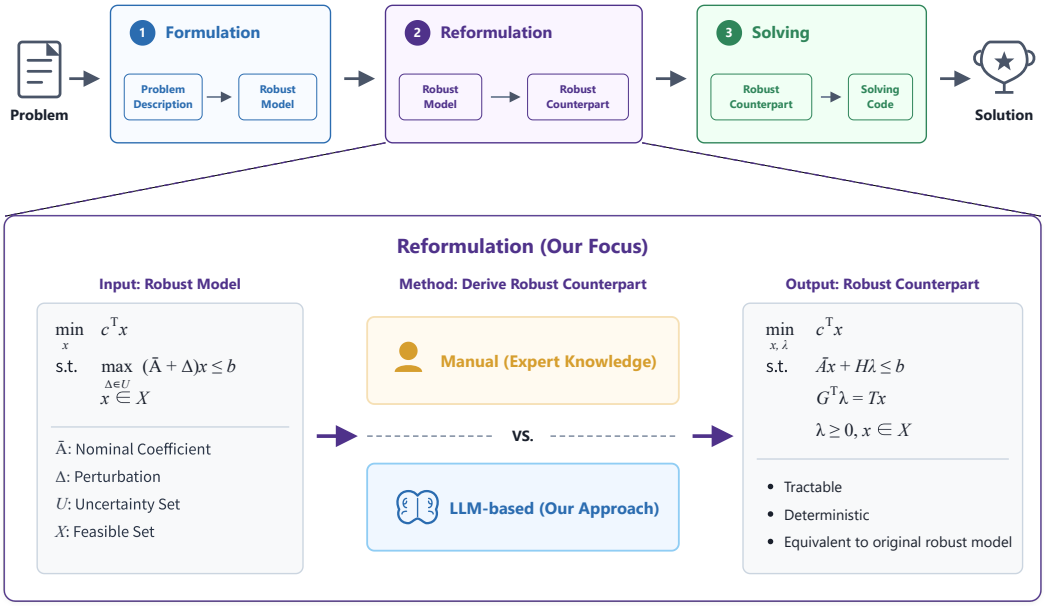


Figure 1: The robust optimization pipeline and our focus on automated reformulation.

sample and computation inefficiency. They often require thousands of expensive rollouts and substantial computation to fit new tasks. Recent studies [8–11] further suggest that RL post-training primarily induces distribution sharpening rather than enabling genuinely new capabilities. By contrast, memory-augmented tuning-free methods [12, 13] offer a highly scalable paradigm: they allow models to integrate interpretable and reusable knowledge from past trajectories at runtime, bypassing costly parameter updates and per-instance test-time search. This is uniquely suited for RO reformulation, where algebraic derivation and duality transformation patterns frequently recur across instances.

However, applying existing memory-augmented frameworks [14, 15] to RO reformulation typically suffers from two fundamental limitations: an additive-only paradigm and a lack of quality verification. As RO derivations must be mathematically exact and a single flawed memory trace can corrupt the full reasoning, memory quality can significantly outweigh quantity for RO reformulation capability. To systematically evaluate this capability, we introduce AutoRO-Bench, a dedicated benchmark with verified ground-truth solutions that comprises an automated data generation pipeline for the core reformulation task and a curated dataset for the application task. To overcome the limitations of existing memory-based methods, we propose Automated Reformulation with Experience Memory (AutoREM), a tuning-free framework with a tailored offline adaptation phase. AutoREM achieves fine-grained memory management through unit-level experiences (ULE) and structured memory operators (SMO), and ensures high-quality memory updates through dual-check commit (DCC) and validation-based acceptance (VBA).

In summary, this paper makes the following contributions. (1) We propose AutoRO-Bench, the first benchmark for systematically evaluating LLMs' RO reformulation capability. It features two complementary tasks: the RO reformulation task that takes mathematical models as input to test derivation capability of reformulation, and the RO application task that takes language-described problems as input to assess end-to-end practical problem solving, adopting benchmark settings similar to previous LLM-based formulation studies. (2) We propose AutoREM, a tuning-free, memory-augmented framework for automated RO reformulation, featuring a textual experience memory that accumulates reusable reformulation knowledge without expert instruction. (3) We design a tailored offline adaptation phase with four key components (ULE, SMO, DCC, and VBA) to automatically construct and refine high-quality reformulation memory. (4) Experiments demonstrate that AutoREM consistently improves accuracy and efficiency across both in-distribution and out-of-distribution datasets, and diverse base LLMs.

2 Related work

LLMs for optimization formulation. Recent studies have extensively explored LLMs for optimization formulation via parametric fine-tuning [16–22] and tuning-free methods [23–38]. However, automated RO reformulation remains largely unexplored, as existing methods focus on problem modeling, while lacking the capability to convert uncertain models to tractable counterparts. The closest work by Bertsimas and Margaritis [39] relies heavily on manually crafted, domain-expert prompts and evaluates on only seven case studies. In contrast, we first introduce a significantly broader benchmark for systematic evaluation, and then propose AutoREM to autonomously collect reusable reformulation experiences without expert curation. Interestingly, the auto-generated memory from AutoREM consistently outperforms the expert-written prompt in the experiments, suggesting that the LLM reflector distills more targeted guidance from the LLM reformulator's specific behaviors than static expert knowledge.

Tuning-free LLM reasoning methods. Since the introduction of Chain-of-Thought (CoT) [40, 41], a useful distinction has emerged between intra-instance test-time scaling [42–50] and cross-instance context adaptation. Unlike test-time scaling, which processes isolated instances without retaining lessons, context adaptation efficiently accumulates reusable knowledge across instances. Within this paradigm, approaches divide into prompt optimization [51–55] and memory augmentation [14, 15, 56–59]. Notably, state-of-the-art (SOTA) memory methods [14] utilizing offline adaptation have recently demonstrated superiority over leading prompt optimizers [51]. While prompt optimization suffers from a brevity bias that discards details, memory methods maintain a structured repository to accumulate domain-specific knowledge without information loss [12, 13]. However, applying existing memory methods to RO reformulation is problematic due to their monolithic additive mechanisms and lack of rigorous verification. In fact, in our experiments, we even observe slight performance degrade over the base LLM after applying the SOTA memory methods. To address these gaps, we design a tailored offline adaptation procedure to automatically construct and refine high-quality reformulation memory.

3 Preliminary

3.1 Reformulation of robust optimization

A *robust model* [3, 60, 61] requires the feasibility to every uncertainty realization, yielding infinitely many constraints that render it directly intractable. *Reformulation* derives a *robust counterpart* (RC) of the original problem, which is an equivalent finite-dimensional deterministic problem that can be handled by standard solvers. A robust model is defined by a nominal problem and an uncertainty set.

Nominal LP. We focus on linear programs (LPs) as the nominal problem class, which is classical and well-studied in robust optimization. A nominal LP takes the form:

$$\min_{\mathbf{x}\in\mathcal{X}} \ \bar{\mathbf{c}}^\top \mathbf{x} \quad \text{s.t.} \quad \bar{\mathbf{a}}_i^\top \mathbf{x} \le b_i, \ i = 1, \dots, m, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the decision vector, $\bar{\mathbf{c}} \in \mathbb{R}^n$ the cost vector, $\bar{\mathbf{a}}_i \in \mathbb{R}^n$ the i-th constraint row, and $\mathcal{X}$ any additional deterministic constraints.

Uncertainty set. An uncertainty set specifies admissible perturbations to the nominal coefficients. In general, they take the form $\tilde{\mathbf{c}} = \bar{\mathbf{c}} + \mathbf{P}^c\boldsymbol{\zeta}^c$ and $\tilde{\mathbf{a}}_i = \bar{\mathbf{a}}_i + \mathbf{P}_i\boldsymbol{\zeta}_i$, with $\boldsymbol{\zeta}^c \in \mathcal{U}_c$, $\boldsymbol{\zeta}_i \in \mathcal{U}_i$ and perturbation matrices $\mathbf{P}^c, \mathbf{P}_i$. We take $\mathbf{P}^c = \mathbf{P}_i = \mathbf{I}$, yielding the element-wise form $\tilde{c}_j = \bar{c}_j + \zeta_j^c$ and $\tilde{a}_{ij} = \bar{a}_{ij} + \zeta_{ij}$. This avoids the extraneous numerical entries in a general $\mathbf{P}_i$ and keeps the focus on the reformulation derivation itself. We consider three classical choices: box, budget, and polyhedral uncertainty set. Each *row* (i.e., the objective function or a constraint) of the model may independently be assigned any of these uncertainty types. Together with the nominal LP (1), they define the robust model:

$$\min_{\mathbf{x}\in\mathcal{X}} \ \max_{\boldsymbol{\zeta}^c\in\mathcal{U}_c} (\bar{\mathbf{c}} + \boldsymbol{\zeta}^c)^\top \mathbf{x} \quad \text{s.t.} \quad \max_{\boldsymbol{\zeta}_i\in\mathcal{U}_i} (\bar{\mathbf{a}}_i + \boldsymbol{\zeta}_i)^\top \mathbf{x} \le b_i, \ i = 1, \dots, m. \tag{2}$$

Robust counterpart (RC). Reformulation eliminates each inner maximization in (2) by LP duality, yielding a finite set of linear constraints. For all three uncertainty sets above, the RC is an LP. Table 1 summarizes their definitions and RC forms; full derivations can be found in Appendix A.

Table 1: Three uncertainty sets and their LP robust counterparts.

Uncertainty	**Definition of $\mathcal{U}_i$**	**Robust Counterpart**
Box	$\lvert\zeta_j\rvert \le \delta_j, \forall j$	$\bar{\mathbf{a}}_i^\top \mathbf{x} + \sum_j t_j \le b_i; t_j \ge \pm\delta_j x_j, t_j \ge 0$
Budget	$\lvert\zeta_j\rvert \le \delta_j, \forall j; \sum_j \lvert\zeta_j\rvert/\delta_j \le \Gamma_i$	$\bar{\mathbf{a}}_i^\top \mathbf{x} + \Gamma_i z_{i0} + \sum_j z_{ij} \le b_i; z_{i0} + z_{ij} \ge \pm\delta_j x_j, z_{i0}, z_{ij} \ge 0$
Polyhedral	$\mathbf{F}\boldsymbol{\zeta}_i \le \mathbf{g}$	$\bar{\mathbf{a}}^\top \mathbf{x} + \mathbf{g}^\top \boldsymbol{\lambda} \le b; \mathbf{F}^\top \boldsymbol{\lambda} = \mathbf{x}, \boldsymbol{\lambda} \ge \mathbf{0}$

3.2 Memory-augmented LLM methods

Let θ denote the parameters of an LLM. In standard generation, the model receives an input problem q and a task prompt p, generating a response $y \sim P_\theta(y \mid p, q)$. To enhance reasoning capabilities without updating θ, memory-augmented methods [12, 13] introduce a structured repository of reusable external memory $\mathcal{M}$ so that the model can draw on accumulated past experience at inference time.

Memory retrieval. Rather than a simple string, $\mathcal{M}$ is typically a structured textual object. It follows two retrieval paradigms. In *explicit retrieval*, an external module pre-selects a relevant subset $\mathcal{M}_{\text{sub}}$ (e.g., via semantic similarity [15]), yielding $y \sim P_\theta(y \mid p, \mathcal{M}_{\text{sub}}, q)$. Conversely, *implicit retrieval* supplies the entire $\mathcal{M}$ directly to the context window [14], relying on the LLM's self-attention to dynamically filter information, yielding $y \sim P_\theta(y \mid p, \mathcal{M}, q)$.

Memory formation. Memory is constructed by distilling generalizable insights from past trajectories into structured, editable entries. This proceeds in two modes. *Online adaptation* continuously updates $\mathcal{M}$ during inference, enabling continual learning on new tasks. Alternatively, *offline adaptation* constructs $\mathcal{M}$ on a pre-deployment dataset; given abundant data, this enables thorough reflection to build a high-quality memory without compromising inference efficiency.

4 Methodology

4.1 AutoRO-Bench: automated RO reformulation benchmark

To comprehensively evaluate LLM-based methods for robust optimization (RO) reformulation, we introduce AutoRO-Bench, comprising two complementary tasks that cover different input modalities: the *RO reformulation task*, which takes formal mathematical models expressed in LaTeX as input, and the *RO application task*, which takes natural-language problem descriptions as input.

RO reformulation task. The RO reformulation task tests the core mathematical challenge: given a formal RO model in LaTeX, derive its deterministic robust counterpart. Each instance consists of two components: a LaTeX problem description as the LLM input q, and a ground-truth optimal value f^* as the evaluation target. The problem is a robust LP model (1) with uncertain coefficients as in (2). Although the symbolic robust counterpart cannot be produced automatically, f^* is obtained by directly solving the robust model via the `RSOME` solver [62, 63].

To generate such verified instances at scale, we propose an automated generation pipeline (Algorithm 1 in Appendix B). It takes as input the number of decision variables n, a target number of valid instances N, a candidate uncertainty type set $\mathcal{U}$ (instantiated with box, budget, and polyhedral), and a template set $\mathcal{T}$ (instantiated with 8 diverse templates $T_{b_0 b_1 b_2}$ parameterized by three binary dimensions to encourage notation-invariant learning, see Appendix B.5). For each candidate instance, the pipeline proceeds through five steps: **(1)** generate a nominal LP by sampling objective coefficients $\mathbf{c}$ and optimization direction, and constructing a sparse feasible constraint polytope $(\mathbf{A}, \mathbf{b})$ with a guaranteed interior point $\mathbf{x}_0$ via the FEASIBLEPOLYTOPE algorithm (Appendix B.3). **(2)** Assign an uncertainty type from $\mathcal{U}$ to each row. **(3)** For each uncertain row, randomly select a sparse subset $\mathcal{S}_i$ of coefficients as the uncertain support, then generate the corresponding uncertainty parameters. The box and budget types use dedicated sampling procedures (Appendix B.4), while polyhedral type employs FEASIBLEPOLYTOPE to construct a sparse uncertainty polytope. **(4)** Solve via `RSOME` and discard infeasible, unbounded, or degenerate instances. **(5)** Render the accepted instance into LaTeX using the selected template(s). Each output record contains the LaTeX problem description, optimal solution $\mathbf{x}^*$, and optimal value f^*.

RO application task. The RO application task evaluates end-to-end performance on realistic RO problem solving. Its natural-language problem description setting is aligned with existing LLM-based optimization formulation benchmarks, enabling direct and comprehensive comparison with those methods. To support this task, we curate the *RO application dataset* of 32 instances. We first select 16 nominal problems from two established benchmarks with 8 from IndustryOR [18] and 8 from OptiBench [64], covering both LP and mixed-integer linear program (MILP). For each nominal problem, we append a natural-language description of the uncertain parameters and the robustness requirement to the original problem statement, constructing a complete RO specification in plain text (see Appendix D).

We generate two RO instances per nominal problem: the first uses the same parameter sampling procedure as the RO reformulation task and the second uses practically-motivated parameters following the RO literature [1, 4]. Unlike the first procedure where uncertainty parameters are given as explicit numerical values, the second leaves some parameters as unevaluated expressions involving operations such as ratios or square roots over problem quantities. For example, deviations may be given as proportions of nominal coefficients, or polyhedral bounds may involve the square root of the uncertain coefficient count. These expressions must be preserved in the reformulation output rather than evaluated to a single number. Full parameter details are given in Appendix C. Since each instance is highly tailored to its nominal problem, all 32 instances were independently verified by three operations research domain experts. Dataset statistics are summarized in Appendix E.

Evaluation protocol. The benchmark evaluates reformulation quality by solver verification: a model receives problem description q and outputs the robust counterpart; the output is translated into executable code and run by a standard LP/MILP solver to obtain $\hat{f}$, which is compared against the ground-truth f^* stored in each data point. An instance is correctly reformulated if $\hat{f}$ matches f^* within a numerical tolerance, providing an automatic, solver-grounded evaluation signal agnostic to the syntactic form of the output. Concrete examples of both tasks are provided in Appendix F.

This paper employs a two-stage pipeline with a *reformulator* and a *coder*, each an LLM with distinct roles. The reformulator, parameterized by θ, takes problem description q and task prompt p as input and generates the robust counterpart $y \sim P_\theta(y \mid p, q)$ or $y \sim P_\theta(y \mid p, \mathcal{M}, q)$ with experience memory $\mathcal{M}$. The coder, parameterized by ϕ, takes y and a coding prompt p' as input and generates executable solver code $c \sim P_\phi(c \mid p', y)$; running c yields $\hat{f}$. We use cost-efficient flagship LLMs as the reformulator and fix a substantially stronger but also more expensive model as the coder. Coding from a correct reformulation is mechanical, so fixing a strong coder isolates reformulation quality as the sole variable.

4.2 AutoREM: automated reformulation with experience memory

We propose AutoREM, a tuning-free LLM framework augmented with a reformulation experience memory (REM). As shown in Figure 2, it operates through two interconnected phases: *offline adaptation* that constructs a high-quality memory $\mathcal{M}$ on a dedicated offline dataset before deployment, and *online inference* that leverages the resulting $\mathcal{M}$ to guide the reformulator at test time.

Online inference. During inference, we employ the implicit retrieval paradigm: the entire memory $\mathcal{M}$ is provided directly in the reformulator's context, formulated as $y \sim P_\theta(y \mid p, \mathcal{M}, q)$, allowing the model to dynamically attend to relevant past derivation patterns without any external retrieval.

Offline adaptation. The quality of online inference depends on the mathematical reliability of $\mathcal{M}$. In RO reformulation where derivations must be exact, even a single erroneous memory entry can corrupt reasoning, making quality far more critical than quantity. This motivates a dedicated offline adaptation phase rather than relying on manual curation or expensive test-time search. Crucially, the AutoRO-Bench pipeline provides rich training instances at low cost, making data-driven memory construction feasible and scalable.

4.3 Offline adaptation of reformulation experience memory

The offline adaptation phase constructs and refines REM through four tightly coupled components organized around two design principles. The first principle is fine-grained memory management: unit-level experience (ULE) addresses the representation dimension by organizing knowledge at the granularity of individual uncertain constraint rows, making each entry independently reusable;

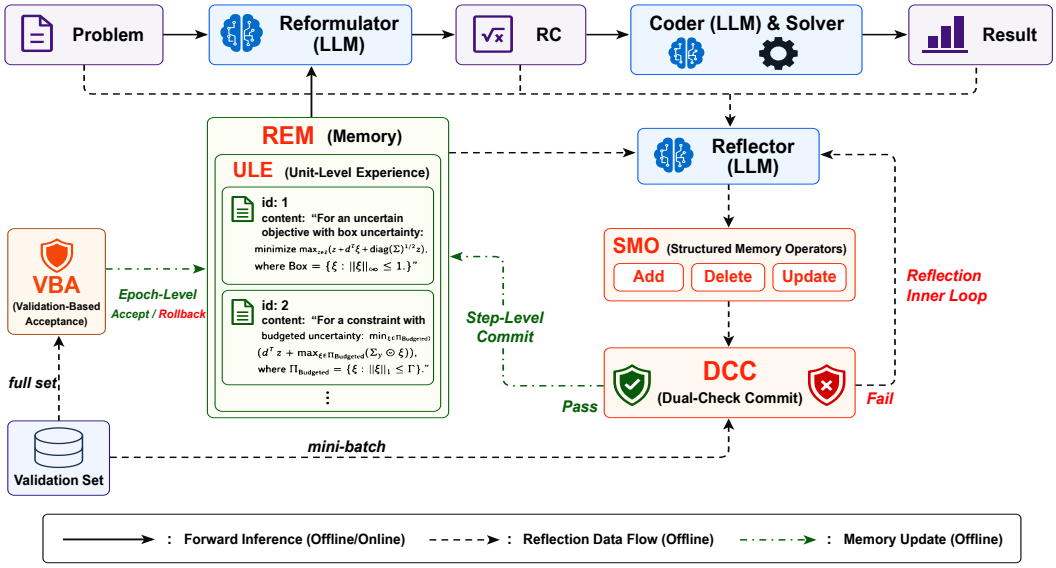


Figure 2: Overview of the AutoREM pipeline.

structured memory operators (SMO) address the editing dimension by providing atomic add, update, and delete operations for precise, interpretable memory modification. The second principle is high-quality memory verification: dual-check commit (DCC) operates at the step level, stress-testing each proposed update against a targeted validation batch before committing; validation-based acceptance (VBA) operates at the epoch level, evaluating the full validation set and rolling back any harmful accumulated changes.

Structured representations. Following prior memory-augmented methods [14, 15], $\mathcal{M}$ is organized as an ordered list of independently editable entries. Each entry has two fields: a unique integer `experience_id` for identification, and a free-form `content` field encoding reformulation knowledge for one minimal independent reformulation unit. The Reformulator and Reflector outputs adopt a unified JSON format. The Reformulator returns `reasoning`, `matched_experience_ids` (the entries used, supporting fine-grained attribution), and `final_answer` (the robust counterpart in LaTeX). The Reflector returns `reasoning` and `final_answer` (a list of memory operators). An external controller parses these outputs and applies the operations. See full prompts in Appendix Q.

Overall flow. Adaptation proceeds for E epochs of S steps each. At each step, a training instance q_s is evaluated under $\mathcal{M}$. If it is already correct, then the step is skipped. Otherwise, the system enters an *inner reflection loop* of up to K iterations, where a Reflector LLM proposes SMO edits on a temporary copy $\hat{\mathcal{M}}$ that targets ULE entries while leaving the committed $\mathcal{M}$ untouched. Once q_s becomes correct under $\hat{\mathcal{M}}$, the DCC gate validates it on a locked batch of validation samples: a full pass commits $\hat{\mathcal{M}}$ to $\mathcal{M}$, while any regression feeds the failure details back into the next Reflector call. If no commit occurs within K iterations, $\hat{\mathcal{M}}$ is discarded. After each step, the controller records a usage-success or usage-failure event on every matched experience, allowing the Reflector to perceive per-experience quality accumulated over training. At epoch end, VBA evaluates the full validation set. If the current accuracy is below the best seen so far across all epochs (tracked as $\mathcal{M}^*$), $\mathcal{M}$ is rolled back to $\mathcal{M}^*$ before next epoch. See complete procedure in Algorithm 3, Appendix I.

Unit-level experience (ULE). ULE defines the granularity of memory entries: each entry targets one minimal independent reformulation unit, making it self-contained and independently applicable to any structurally similar unit in a new problem. This fine decomposition enables more precise attribution and flexible reuse of reformulation knowledge than coarser, problem-level experiences. In practice, ULE is realized by simply adding the phrase "minimal independent reformulation unit" to the Reflector's prompt. The Reflector itself decides what counts as such a unit for each problem and distills experience at that level.

Structured memory operators (SMO). The Reflector outputs a JSON list of memory operations, each of three types: ADD (insert a new entry), UPDATE (revise an existing entry's content), or DELETE (remove an entry), each specifying the target `experience_id`. An external controller parses this JSON output and applies the corresponding operations to the working copy $\hat{\mathcal{M}}$, while leaving the committed $\mathcal{M}$ untouched. The Reflector also receives the full trace of prior operations and outcomes of the inner loop, enabling it to diagnose why earlier edits failed and propose targeted corrections.

Dual-check commit (DCC). Committing after a single training fix risks silent regression on previously correct instances. When q_s first becomes correct under $\hat{\mathcal{M}}$, a mini-batch of up to B validation instances is selected from the validation set and locked for the remainder of the step. Selection prioritizes *linked* instances in the validation set, whose prior correct answers are attributed to the modified experience entries, over the *unlinked* ones, filling the remaining budget with currently correct unlinked instances. Linking is maintained via an experience-instance correspondence map, which the external controller updates by reading the `matched_experience_ids` field in the Reformulator's JSON output. ULE's fine-grained unit-level decomposition makes this attribution more accurate. The locked batch is then evaluated under $\hat{\mathcal{M}}$. On DC pass, $\hat{\mathcal{M}}$ is committed to $\mathcal{M}$ and the correspondence map updated. On DC fail, the regression details (failing problem, reformulator output, and solver feedback) are returned to the Reflector as additional context, making DCC an active source of learning signal rather than a passive filter.

Validation-based acceptance (VBA). DCC covers only a mini-batch of B instances per step, which provides a fast but biased estimate of memory quality and can let locally beneficial yet globally harmful edits slip through. To obtain a more reliable assessment, VBA further evaluates the full fixed held-out validation set of size $|\mathcal{V}|$ at each epoch end and retains the best-ever snapshot as $\mathcal{M}^*$. If the epoch-end accuracy falls below the best recorded value, the current $\mathcal{M}$ is rolled back before the next epoch begins, preventing the accumulation of such harmful edits over time.

5 Experiments

5.1 Experimental settings

Tasks and datasets. We evaluate AutoREM on the two tasks of AutoRO-Bench, each with a distinct evaluation goal and a separate set of baselines.

For the RO reformulation task, we compare AutoREM against LLM-based reformulation methods using three datasets generated by our pipeline. The *Random* dataset (64 instances) serves as the in-distribution benchmark, following the same generation procedure as the training data with 2–5 decision variables. The *Hard* dataset (32 instances) is an out-of-distribution stress test where each instance is designed around observed failure modes of the base LLM (see Appendix B.6). The *Large* dataset (32 instances) is a second out-of-distribution split that uses the same random procedure as *Random* but scales the variable count to 6 and 7, probing generalization to larger problem dimensions. Train, validation, and test instances are independently generated without reusing underlying problems or uncertainty-parameter sets. Split-independence and input-length details are provided in Appendix B.7.

For the RO application task, we instead compare AutoREM against LLM-based formulation methods, evaluating end-to-end RO problem solving from natural-language input on the 32-instance RO *Application* dataset. Following the full RO solving pipeline (Figure 1), we first directly employ the same base LLM as a formulator to translate each natural-language problem description into a LaTeX robust model, which is then passed to AutoREM as the reformulation component.

Baselines. We compare AutoREM against two groups of baselines, all using the same base LLM. The datasets and code are publicly available[2] to support reproducibility.

LLM reformulation methods operate on the RO reformulation task, taking a LaTeX robust model as input; all use zero-shot CoT [40] prompting and share the same fixed coder LLM: (1) **Base LLM**; (2) **Max Thinking**: same base LLM with maximum reasoning effort, scaling test-time computation per instance; (3) **Expert Prompt** [39]: the RO domain expert prompt; (4) **ReasoningBank** [15]: a SOTA online adaptation memory method, adapted to our offline setting; (5) **ACE** [14]: a SOTA offline

[2] `https://github.com/Bill-CJB/AutoREM`

Table 2: Comparison of methods across the three datasets of the RO reformulation task.

Methods	Random (In-Distribution)		Hard (Out-of-Distribution)		Large (Out-of-Distribution)	
	Accuracy/%↑	Output Tokens↓	Accuracy/%↑	Output Tokens↓	Accuracy/%↑	Output Tokens↓
Base LLM	87.5 ± 5.4	7777 ± 162	70.8 ± 10	9026 ± 295	76.0 ± 3.6	11757 ± 135
Max Thinking	90.6 ± 1.6	13857 ± 331	<u>83.3 ± 1.8</u>	14902 ± 651	77.1 ± 3.6	22695 ± 1164
Expert Prompt	<u>92.7 ± 3.3</u>	8750 ± 200	<u>83.3 ± 3.6</u>	7549 ± 101	<u>79.2 ± 9.5</u>	13243 ± 1033
ReasoningBank	85.4 ± 0.9	6575 ± 131	80.2 ± 4.8	8089 ± 701	71.9 ± 13.6	11774 ± 471
ACE	87.0 ± 6.3	**4419 ± 56**	81.3 ± 9.4	**5238 ± 570**	63.5 ± 4.8	**6706 ± 531**
AutoREM	**97.4 ± 0.9**	<u>6386 ± 56</u>	**94.8 ± 3.6**	<u>6944 ± 27</u>	**85.4 ± 7.1**	<u>10929 ± 238</u>

adaptation memory framework, the most related competitor. AutoREM, ACE, and ReasoningBank each receive $10 \times 8 = 80$ training exposures. Offline adaptation protocols are detailed in Appendix H.

LLM formulation methods address the RO application task end-to-end from natural-language input: (6) **AlphaOPT** [26]: an experience-library-based formulation method; (7) **LEAN-LLM-OPT** [28]: a retrieval-augmented-generation-based agentic formulation framework; (8) **OptiTree** [30]: a tree-search-based test-time scaling formulation method.

Hyperparameter settings. All methods use DeepSeek-V4-Flash as the base LLM with default reasoning effort (high), maximum output tokens of 32K, and temperature 0 (greedy decoding). DeepSeek-V4-Pro is used as the fixed coder. For the offline adaptation phase, we run $E = 10$ epochs with $S = 8$ steps per epoch. The dual-check batch size is set to $B = 4$, the maximum inner reflection iterations to $K = 3$, and the validation set size to $|\mathcal{V}| = 64$.

Metrics. We report two metrics to jointly assess reformulation quality and efficiency. *Accuracy* (percentage of instances whose reformulation yields an optimal objective value matching the ground truth within numerical tolerance) measures reformulation quality. The objective-agreement criterion is $|\hat{f} - f^*| < 10^{-4}$. Malformed outputs and solver failures count as incorrect. *Output Tokens* (average number of tokens generated by the reformulator per instance) measures reformulation efficiency: output tokens are the primary determinant of online inference latency, and output tokens are priced significantly higher than prompt tokens in API-based deployments, making this a practically important metric. Although temperature is set to 0 (greedy decoding), LLM responses may still vary across API calls due to engineering-level non-determinism. We therefore run each method three times with the same frozen learned memory where applicable and report the mean ± sample standard deviation. In all tables, **bold** indicates the best result and <u>underline</u> indicates the second best.

5.2 Experimental results

Comparison. AutoREM achieves the highest accuracy on the in-distribution *Random* dataset (97.4%), surpassing Expert Prompt by 4.7%, Max Thinking by 6.8%, and both SOTA memory methods (ReasoningBank, ACE) by more than 10%. It also attains the best accuracy and efficiency trade-off, generating 54% fewer tokens than Max Thinking while delivering far higher accuracy. Notably, both SOTA memory methods (ReasoningBank: 85.4%, ACE: 87.0%) fail to improve over the Base LLM (87.5%), showing that on mathematically rigorous tasks, blindly accumulating experiences is counterproductive. In contrast, AutoREM's selective adaptation converges to just 8 verified, non-redundant templates, demonstrating that high-quality experiences matter far more than quantity. See Appendices M and J for the full qualitative comparison and adaptation dynamics.

Out-of-distribution generalization. AutoREM generalizes strongly to both out-of-distribution settings. On the *Hard* dataset, the advantage is most pronounced: all baselines drop substantially from their *Random* scores (e.g., Expert Prompt: -9.4%, ACE: -5.7%), whereas AutoREM declines only marginally (-2.6%), outperforming the best baseline by 11.5% and the Base LLM by 24.0%. This robustness demonstrates that the learned memory genuinely improves the model's core reformulation capability rather than memorizing training-set patterns. AutoREM also reduces output tokens by 23% compared to the Base LLM. On the *Large* dataset, all methods exhibit a consistent accuracy drop, indicating that the bottleneck is long-context processing rather than reformulation capability. AutoREM still leads at 85.4% with consistent efficiency gains.

We further evaluate generalization across uncertainty structures and problem sources using the same frozen memory without re-adaptation. The three 32-instance datasets cover non-diagonal perturbation mappings (*General* P_i), shared primitive uncertainty across rows (*Coupled Row*), and

Table 3: Comparison of methods across three base LLMs.

Methods	DeepSeek-V4-Flash		GPT-5.4		Qwen3.6-Plus	
	Accuracy/%↑	Output Tokens↓	Accuracy/%↑	Output Tokens↓	Accuracy/%↑	Output Tokens↓
Base LLM	87.5 ± 5.4	7777 ± 162	90.6 ± 3.1	2871 ± 78	81.2 ± 2.7	3448 ± 74
AutoREM	**97.4 ± 0.9**	**6386 ± 56**	**96.9 ± 0**	**2550 ± 21**	**88.5 ± 1.8**	**2697 ± 73**

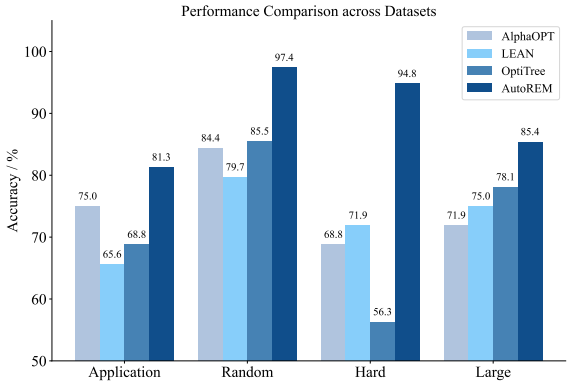


Figure 3: Comparison with LLM-based benchmarks.

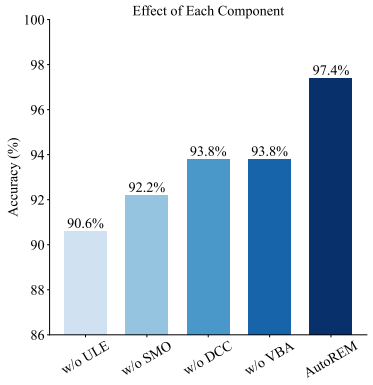


Figure 4: Ablation results.

textbook and published robust models independent of our generator (*External*). AutoREM improves accuracy from 89.6% to 90.6%, from 76.0% to 89.6%, and from 92.7% to 97.9%, respectively. These results provide additional evidence that the trained memory transfers to unseen instances sharing relevant reformulation patterns, including textbook and published RO models beyond our generator (Appendix N).

Cross-model universality. To evaluate transferability, we directly apply the memory trained on DeepSeek-V4-Flash to GPT-5.4 and Qwen3.6-Plus without re-adaptation. These three models are high cost-efficiency flagship offerings from DeepSeek, OpenAI, and Alibaba, providing broad coverage of model architectures and training paradigms. As shown in Table 3, AutoREM consistently improves both accuracy and token efficiency across all three base LLMs, with accuracy gains of 6.3% and 7.3% on GPT-5.4 and Qwen3.6-Plus respectively. This indicates that the reformulation experience memory encodes transferable knowledge that generalizes across heterogeneous base LLMs.

Application results. As shown in Figure 3, on the end-to-end *Application* dataset where all methods operate on natural-language problem descriptions, AutoREM achieves 81.3% against the best baseline of 75.0% (AlphaOPT), demonstrating strong practical capability for real-world RO problem solving. When the same three formulation methods are evaluated on the pure reformulation datasets (bypassing any formulation step), the advantage of AutoREM becomes even more pronounced: on the *Hard* dataset, the best baseline reaches only 71.9% while AutoREM maintains 94.8%, a gap exceeding 22%. This widening margin reveals that the bottleneck lies specifically in reformulation: these formulation-oriented methods lack the algebraic machinery for robust counterpart derivations, and it is precisely this capability that AutoREM's reformulation-specific memory supplies.

Ablation and hyperparameter analysis. Figure 4 shows that removing any components from ULE, SMO, DCC, and VBA can cause a non-trivial accuracy drop, demonstrating that each is indispensable. Figures 5 and 6 further show that the chosen defaults ($B = 4$, $|\mathcal{V}| = 64$) achieve the best accuracy, with performance remaining robust around these values. Implementation details and a full sensitivity analysis are provided in Appendices K and L.

6 Conclusion

We introduced AutoRO-Bench and AutoREM, respectively the first benchmark and the first tuning-free memory-augmented framework for automated RO reformulation. AutoREM consistently achieves state-of-the-art accuracy across in-distribution and out-of-distribution settings, transfers across diverse base LLMs without re-adaptation, and outperforms LLM-based formulation methods

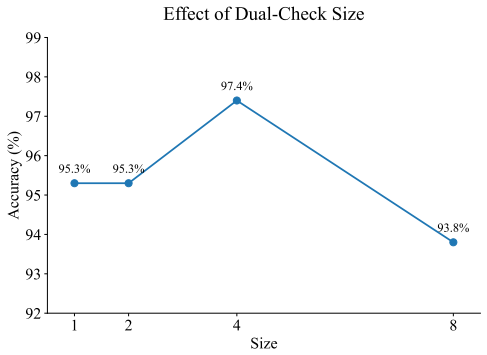


Figure 5: Effect of dual-check size B.

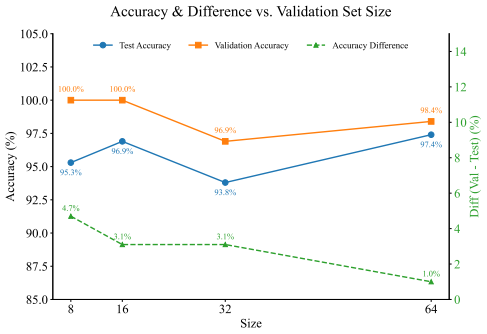


Figure 6: Effect of validation set size $|\mathcal{V}|$.

on both the pure reformulation task and the end-to-end application task. One current limitation is that offline adaptation relies on solver-verified objective values as the supervision signal, which may not be available for all problem classes; future work could explore weaker supervision such as feasibility checks. Extending the framework beyond LP-based robust counterparts to more complex RO is another important direction. Further directions include conic and adjustable-robust models, stronger equivalence verification, and safeguards for untrusted inputs.

Acknowledgments and Disclosure of Funding

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative, the National Natural Science Foundation of China, and the Ministry of Education, Singapore. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore and Infocomm Media Development Authority.

Hanzhang Qin is supported by the Trust Tech Funding Initiative under grant DTC-IGC-07 (project: Multi-Party Evaluation: Blockchain-Enabled AI Safety and Verification Framework) and by the National Natural Science Foundation of China under grants U25A20450 and 62571374. Jinbiao Chen and Guanyi Wang are supported by the Ministry of Education, Singapore, under the Academic Research Fund (AcRF) Tier-1 grant (A-8000607-00-00) 22-5539-A0001 and Tier-2 grant T2EP20125-0030. The authors declare no competing interests.

References

[1] Chaithanya Bandi and Dimitris Bertsimas. Tractable stochastic analysis in high dimensions via robust optimization. *Mathematical Programming*, 134(1):23–70, 2012.

[2] Dimitris Bertsimas and Melvyn Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.

[3] Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of uncertain linear programs. *Operations Research Letters*, 25(1):1–13, 1999.

[4] Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of Linear Programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3):411–424, 2000.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[6] Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-Reasoner-Zero: An Open Source Approach to Scaling Up Reinforcement Learning on the Base Model. In *Advances in Neural Information Processing Systems*. arXiv, July 2025.

[7] Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, et al. A survey of optimization modeling meets llms: progress and future directions. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 10742–10750, 2025.

[8] Andre He, Daniel Fried, and S. Welleck. Rewarding the unlikely: Lifting GRPO beyond distribution sharpening. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 25548–25560, 2025.

[9] Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. *arXiv preprint arXiv:2506.10947*, 2025.

[10] Yuda Song, Julia Kempe, and Remi Munos. Outcome-based exploration for LLM reasoning. *arXiv preprint arXiv:2509.06941*, 2025.

[11] Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? In *Advances in Neural Information Processing Systems*, 2025.

[12] Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the Age of AI Agents. *arXiv preprint arXiv:2512.13564*, 2025.

[13] Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model-based agents. *ACM Transactions on Information Systems*, 43(6):1–47, 2025.

[14] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026.

[15] Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026.

[16] Yitian Chen, Jingfan Xia, S. Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, 2025.

[17] 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*, pages 228–236, 2026.

[18] 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, 2025.

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

[20] Ziyang Xiao, Yuan Jessica Wang, Xiongwei Han, Shisi Guan, Jingyan Zhu, Jingrong Xie, Lilin Xu, Han Wu, Wing Yin Yu, Zehua Liu, Xiaojin Fu, Gang Chen, and Dongxiang Zhang. DeepOR: A deep reasoning foundation model for optimization modeling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 34052–34060, 2026.

[21] Wantong Xie, Yi-Xiang Hu, Jieyang Xu, Feng Wu, and Xiang-Yang Li. MURKA: Multi-Reward Reinforcement Learning with Knowledge Alignment for Optimization Tasks. In *Advances in Neural Information Processing Systems*, 2025.

[22] 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*, 2026.

[23] Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *International Conference on Machine Learning*, 2024.

[24] Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of Mathematical Optimization Models Using LLMs. In *International Conference on Machine Learning*, pages 1864–1886. PMLR, October 2025.

[25] Xia Jiang, Yaoxin Wu, Chenhao Zhang, and Yingqian Zhang. DRoC: Elevating Large Language Models for Complex Vehicle Routing via Decomposed Retrieval of Constraints. In *International Conference on Learning Representations*, volume 2025, pages 46731–46752, May 2025.

[26] Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, Shenhao Wang, Haris Koutsopoulos, Hai Wang, Cathy Wu, and Jinhua Zhao. AlphaOPT: Formulating optimization programs with self-improving LLM experience library. *arXiv preprint arXiv:2510.18428*, 2025.

[27] Dong Li, Xiaoyu Zhao, Linlin Yu, Yanchi Liu, Wei Cheng, Zhengzhang Chen, Zhong Chen, Feng Chen, Chen Zhao, and Haifeng Chen. SolverLLM: Leveraging test-time scaling for optimization problem via LLM-guided search. In *Advances in Neural Information Processing Systems*, 2025.

[28] Kuo Liang, Yuhang Lu, Jianming Mao, Shuyi Sun, Chunwei Yang, Congcong Zeng, Xiao Jin, Hanzhang Qin, Ruihao Zhu, and Chung-Piaw Teo. Large-scale optimization model auto-formulation: Harnessing LLM flexibility via structured workflow. *arXiv preprint arXiv:2601.09635*, 2026.

[29] Fan Liu, Zherui Yang, Cancheng Liu, Tianrui Song, Xiaofeng Gao, and Hao Liu. MM-agent: LLM as agents for real-world mathematical modeling problem. In *Advances in Neural Information Processing Systems*, 2025.

[30] Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *Advances in Neural Information Processing Systems*, 2025.

[31] Tennison Liu and Mihaela van der Schaar. MATHMO: Automated Mathematical Modeling Through Adaptive Search. In *International Conference on Learning Representations*, 2026.

[32] Xiaotian Pan, Junhao Fang, Feng Wu, Sijia Zhang, Yi-Xiang Hu, Shaoang Li, and Xiang-Yang Li. Guiding large language models in modeling optimization problems via question partitioning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2657–2665, 2024.

[33] Rindra Ramamonjison, Haley Li, Timothy Yu, Shiqi He, Vishnu Rengan, Amin Banitalebi-dehkordi, Zirui Zhou, and Yong Zhang. Augmenting operations research with auto-formulation of optimization models from problem descriptions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 29–62, 2022.

[34] Teng Wang, Wing Yin Yu, Zhenqi He, Zehua Liu, HaileiGong HaileiGong, Han Wu, Xiongwei Han, Wei Shi, Ruifeng She, Fangzhou Zhu, and Tao Zhong. BPP-search: Enhancing tree of thought reasoning for mathematical modeling problem solving. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 821–838, 2025.

[35] Zhiyuan Wang, Bokui Chen, Yinya Huang, Qingxing Cao, Ming He, Jianping Fan, and Xiaodan Liang. ORMind: A cognitive-inspired end-to-end reasoning framework for operations research. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 104–131, 2025.

[36] Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *International Conference on Learning Representations*, 2024.

[37] Jihai Zhang, Wei Wang, Siyan Guo, Li Wang, Fangquan Lin, Cheng Yang, and Wotao Yin. Solving general natural-language-description optimization problems with large language models. In *Proceedings of the North American Chapter of the Association for Computational Linguistics*, pages 483–490, 2024.

[38] Yansen Zhang, Qingcan Kang, Wing Yin Yu, Xiaojin Fu, Xiongwei Han, Tao Zhong, and Chen Ma. Decision Information Meets Large Language Models: The Future of Explainable Operations Research. In *International Conference on Learning Representations*, volume 2025, pages 65698–65722, May 2025.

[39] Dimitris Bertsimas and Georgios Margaritis. Robust and adaptive optimization under a large language model lens. *arXiv preprint arXiv:2501.00568*, 2024.

[40] Shixiang Shane Gu, Yusuke Iwasawa, Takeshi Kojima, Yutaka Matsuo, and Machel Reid. Large language models are zero-shot reasoners. In *Advances in Neural Information Processing Systems*, pages 22199–22213, 2022.

[41] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

[42] Uri Alon, Peter Clark, Nouha Dziri, Luyu Gao, Prakhar Gupta, Shashank Gupta, Skyler Hallinan, Katherine Hermann, Aman Madaan, Bodhisattwa Prasad Majumder, Shrimai Prabhumoye, Niket Tandon, Sean Welleck, Sarah Wiegreffe, Yiming Yang, and Amir Yazdanbakhsh. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, pages 46534–46594, 2023.

[43] Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, Noah Shinn, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8634–8652, 2023.

[44] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 17682–17690, 2024.

[45] Yuan Cao, Tom Griffiths, Karthik Narasimhan, Izhak Shafran, Shunyu Yao, Dian Yu, and Jeffrey Zhao. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, pages 11809–11822, 2023.

[46] Yichao Fu, Xuewei Wang, Yuandong Tian, and Jiawei Zhao. Deep think with confidence. In *International Conference on Learning Representations*, 2026.

[47] Zhewei Kang, Xuandong Zhao, and Dawn Song. Scalable best-of-n selection for large language models via self-certainty. In *Advances in Neural Information Processing Systems*, 2025.

[48] Aayush Karan and Yilun Du. Reasoning with sampling: Your base model is smarter than you think. In *International Conference on Learning Representations*, 2026.

[49] Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. In *International Conference on Learning Representations*, 2025.

[50] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023.

[51] Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026.

[52] Yumin Choi, Jinheon Baek, and Sung Ju Hwang. System prompt optimization with meta-learning. In *Advances in Neural Information Processing Systems*, 2025.

[53] Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P. Xing, and Zhiting Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. In *International Conference on Learning Representations*, 2024.

[54] Jianyu Wang, Zhiqiang Hu, and Lidong Bing. Evolving prompts in-context: An open-ended, self-replicating perspective. In *International Conference on Machine Learning*, 2025.

[55] Chenrui Zhang, Lin Liu, Chuyuan Wang, Xiao Sun, Hongyu Wang, Jinpeng Wang, and Mingchen Cai. PREFER: Prompt ensemble learning via feedback-reflect-refine. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19525–19532, 2024.

[56] Kyunghoon Bae, Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Honglak Lee, Lajanugen Logeswaran, and Sungryull Sohn. AutoGuide: Automated generation and selection of context-aware guidelines for large language model agents. In *Advances in Neural Information Processing Systems*, pages 119919–119948, 2024.

[57] Yitao Liu, Chenglei Si, Karthik R Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 14179–14198, 2025.

[58] Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. REMem: Reasoning with episodic memory in language agent. In *International Conference on Learning Representations*, 2026.

[59] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Y. Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 19632–19642, 2023.

[60] Aharon Ben-Tal and Arkadi Nemirovski. Robust convex optimization. *Mathematics of Operations Research*, 23(4):769–805, 1998. doi: 10.1287/moor.23.4.769.

[61] Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2009. ISBN 978-0-691-14368-2.

[62] Zhi Chen and Peng Xiong. RSOME in Python: An open-source package for robust stochastic optimization made easy. *INFORMS Journal of Computing*, 35(4):717–724, 2023.

[63] Zhi Chen, Melvyn Sim, and Peng Xiong. Robust stochastic optimization made easy with RSOME. *Management Science*, 66(8):3329–3339, 2020.

[64] Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *International Conference on Learning Representations*, 2025.

[65] Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, Zirui Zhou, and Yong Zhang. Towards human-aligned evaluation for linear programming word problems. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 16550–16556, 2024.

[66] Haotian Zhai, Connor Lawless, Ellen Vitercik, and Liu Leqi. EquivaMap: Leveraging LLMs for automatic equivalence checking of optimization formulations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 74288–74305, 2025.

[67] Zukui Li, Ran Ding, and Christodoulos A. Floudas. A comparative theoretical and computational study on robust counterpart optimization: I. robust linear optimization and robust mixed integer linear optimization. *Industrial & Engineering Chemistry Research*, 50(18):10567–10603, 2011.

[68] Zukui Li, Qiuhua Tang, and Christodoulos A. Floudas. A comparative theoretical and computational study on robust counterpart optimization: II. probabilistic guarantees on constraint satisfaction. *Industrial & Engineering Chemistry Research*, 51(19):6769–6788, 2012.

[69] Marc Goerigk and Mohammad Khosravi. Benchmarking problems for robust discrete optimization. *Computers & Operations Research*, 166:106608, 2024.

Automated Reformulation of Robust Optimization via Memory-Augmented Large Language Models (Appendix)

A Derivations of robust counterparts

This section derives the robust counterpart (RC) for each of the three polyhedral uncertainty sets introduced in Section 3.1. All three derivations follow the same two-step template: (i) reformulate the worst-case objective function or constraint as an inner maximization problem, and (ii) replace this inner maximization with its LP dual, yielding a finite set of linear rows.

Throughout, we work with the i-th uncertain constraint in the direct perturbation form

$$\sum_{j=1}^{n} \tilde{a}_{ij}\, x_j \le b_i, \qquad \tilde{a}_{ij} = \bar{a}_{ij} + \zeta_{ij},$$

where $\boldsymbol{\zeta}_i \in \mathcal{U}_i$ is the perturbation vector, ζ_{ij} is its j-th component, and $\bar{a}_{ij}$ is the nominal coefficient. Within each subsection the row index i is fixed, so we write $\boldsymbol{\zeta}$ and ζ_j for $\boldsymbol{\zeta}_i$ and ζ_{ij} for brevity. Requiring this to hold for every $\boldsymbol{\zeta}_i \in \mathcal{U}_i$ is equivalent to

$$\sum_{j=1}^{n} \bar{a}_{ij}\, x_j + \max_{\boldsymbol{\zeta}_i \in \mathcal{U}_i} \sum_{j=1}^{n} \zeta_{ij}\, x_j \le b_i. \tag{3}$$

The inner maximization in (3) is a linear program in $\boldsymbol{\zeta}_i$ (for fixed $\mathbf{x}$); replacing it by its dual gives the RC.

A.1 Box uncertainty set

For $\mathcal{U}_{\text{box}} = \{\boldsymbol{\zeta} \mid |\zeta_j| \le \delta_j, \forall j\}$, the inner problem is

$$\max_{\boldsymbol{\zeta}} \sum_j \zeta_j\, x_j \quad \text{s.t.} \quad |\zeta_j| \le \delta_j,\ \forall\, j.$$

This separates over j: for each j, $\max_{|\zeta_j| \le \delta_j} \zeta_j x_j = \delta_j |x_j|$, achieved at $\zeta_j^* = \delta_j \operatorname{sign}(x_j)$. Introducing auxiliary variables $t_j \ge 0$ to linearize $\delta_j |x_j|$ (i.e., $t_j \ge \delta_j x_j$ and $t_j \ge -\delta_j x_j$) yields the LP robust counterpart:

$$\sum_{j=1}^{n} \bar{a}_{ij}\, x_j + \sum_{j=1}^{n} t_j \le b_i, \qquad t_j \ge \delta_j x_j, \quad t_j \ge -\delta_j x_j, \quad t_j \ge 0. \tag{4}$$

A.2 Budget uncertainty set

For $\mathcal{U}_\Gamma = \left\{\boldsymbol{\zeta} \mid |\zeta_j| \le \delta_j, \forall j;\ \sum_j |\zeta_j|/\delta_j \le \Gamma_i\right\}$, the inner problem is

$$\max_{\boldsymbol{\zeta}} \sum_j \zeta_j\, x_j \quad \text{s.t.} \quad |\zeta_j| \le \delta_j,\ \forall\, j; \quad \sum_j \frac{|\zeta_j|}{\delta_j} \le \Gamma_i.$$

Substituting $u_j = \zeta_j/\delta_j$ (so $|u_j| \le 1$ and $\sum_j |u_j| \le \Gamma_i$) and dualizing the budget constraint with multiplier $z_{i0} \ge 0$ and the individual box constraints with multipliers $z_{ij} \ge 0$, strong duality gives $z_{i0} + z_{ij} \ge \delta_j |x_j|$ for each j. Splitting the absolute value into two linear constraints yields the LP robust counterpart:

$$\sum_{j=1}^{n} \bar{a}_{ij}\, x_j + \Gamma_i\, z_{i0} + \sum_{j=1}^{n} z_{ij} \le b_i, \quad z_{i0} + z_{ij} \ge \pm\delta_j\, x_j,\ \forall\, j, \quad z_{ij} \ge 0,\ z_{i0} \ge 0. \tag{5}$$

Setting $\Gamma_i = |\mathcal{S}_i|$ recovers the box RC (4) (budget constraint becomes inactive).

Algorithm 1 Data generator for the RO reformulation task

Require: n (number of variables); N (target instances); uncertainty type set $\mathcal{U}$; template set $\mathcal{T}$
Ensure: Dataset $\mathcal{D}$

$\mathcal{D} \leftarrow \emptyset$
$\text{count} \leftarrow 0$
while $\text{count} < N$ **do**
  Step 1: Nominal LP. Sample objective coefficients $\mathbf{c}$ and direction; construct sparse feasible constraint polytope $(\mathbf{A}, \mathbf{b})$ with interior point $\mathbf{x}_0$ via the FEASIBLEPOLYTOPE algorithm (Appx. B.3)
  Step 2: Uncertainty types. Assign an uncertainty type from $\mathcal{U}$ to each row
  if all rows deterministic **then continue**
  end if
  Step 3: Uncertainty parameters (see Appx. B.4)
  for each uncertain row i **do**
    Sample sparse support $\mathcal{S}_i \subseteq \{j : a_{ij} \neq 0\}$, $|\mathcal{S}_i| \geq 2$
    if Box **then** sample per-element deviations $\boldsymbol{\delta}_i$
    else if Budget **then** sample $\boldsymbol{\delta}_i$ and budget Γ_i
    else if Poly **then**
      $(\mathbf{F}, \mathbf{g}) \leftarrow \text{FEASIBLEPOLYTOPE}(|\mathcal{S}_i|, \boldsymbol{\zeta}^0, [-p_r, p_r], |\mathcal{S}_i|)$ (Appx. B.3)
    end if
  end for
  Step 4: Solve & filter. Solve via `RSOME`; discard if infeasible, unbounded, or degenerate
  Step 5: Render. Select $\mathcal{T}' \subseteq \mathcal{T}$; render LaTeX using each $T_{b_0 b_1 b_2} \in \mathcal{T}'$; add records to $\mathcal{D}$
  $\text{count} \leftarrow \text{count} + 1$
end while
return $\mathcal{D}$

A.3 General polyhedral uncertainty set

For $\mathcal{U}_{\text{poly}} = \{\boldsymbol{\zeta} \mid \mathbf{F}\boldsymbol{\zeta} \leq \mathbf{g}\}$, the uncertain row evaluates to $\bar{\mathbf{a}}^\top \mathbf{x} + \boldsymbol{\zeta}^\top \mathbf{x}$, so the inner problem is

$$\max_{\boldsymbol{\zeta}} \ \boldsymbol{\zeta}^\top \mathbf{x} \quad \text{s.t.} \quad \mathbf{F}\boldsymbol{\zeta} \leq \mathbf{g}.$$

Associating dual variables $\boldsymbol{\lambda} \geq \mathbf{0}$ with the constraints $\mathbf{F}\boldsymbol{\zeta} \leq \mathbf{g}$, the LP dual is

$$\min_{\boldsymbol{\lambda} \geq \mathbf{0}} \ \mathbf{g}^\top \boldsymbol{\lambda} \quad \text{s.t.} \quad \mathbf{F}^\top \boldsymbol{\lambda} = \mathbf{x}.$$

By strong duality, the RC becomes

$$\bar{\mathbf{a}}^\top \mathbf{x} + \mathbf{g}^\top \boldsymbol{\lambda} \leq b, \quad \mathbf{F}^\top \boldsymbol{\lambda} = \mathbf{x}, \quad \boldsymbol{\lambda} \geq \mathbf{0}. \tag{6}$$

This formulation subsumes the box and budget cases: both can be cast as instances of $\mathcal{U}_{\text{poly}}$ with appropriate choices of $\mathbf{F}$ and $\mathbf{g}$.

Extension to equality constraints. In practice (see Appendix B.4), the polyhedral uncertainty set may additionally include equality constraints $\mathbf{E}\boldsymbol{\zeta} = \mathbf{e}$, used to zero out coefficients outside the uncertain support $\mathcal{S}_i$. Introducing free dual variables $\boldsymbol{\mu}$ for these constraints, the RC generalizes to

$$\bar{\mathbf{a}}^\top \mathbf{x} + \mathbf{g}^\top \boldsymbol{\lambda} + \mathbf{e}^\top \boldsymbol{\mu} \leq b, \quad \mathbf{F}^\top \boldsymbol{\lambda} + \mathbf{E}^\top \boldsymbol{\mu} = \mathbf{x}, \quad \boldsymbol{\lambda} \geq \mathbf{0}, \quad \boldsymbol{\mu} \text{ free}. \tag{7}$$

B RO reformulation task: data generation details

This section supplements Section 4.1 with the full details of the data generation pipeline for the RO reformulation task. Algorithm 1 gives the complete generation procedure.

B.1 Problem structure and scale

Variable count and constraint count. Each RO instance is parameterized by a number of decision variables $n \in \{2, 3, 4, 5\}$. The number of inequality/equality constraints equals n, i.e., the constraint matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$.

Variable bounds. A nominal range $r \sim \mathrm{Uniform}(1, 10)$ (rounded to 1 decimal place) is sampled per instance. A variable-bound type $\tau_x \in \{a,\ b,\ c\}$ is then selected uniformly at random, determining the box $[x_l, x_u]$ that every decision variable x_i must satisfy:

$$\tau_x = a:\ x_l = -r,\ x_u = r; \qquad \tau_x = b:\ x_l = 0,\ x_u = r; \qquad \tau_x = c:\ x_l = -r,\ x_u = 0.$$

The interior point $\mathbf{x}_0 \in [x_l, x_u]^n$ used in Step 1 is sampled component-wise as $x_{0,i} \sim \mathrm{Uniform}(x_l, x_u)$, rounded to 1 decimal place.

Objective coefficients. The n objective coefficients are sampled i.i.d. as $c_j \sim \mathrm{Uniform}(-10, 10)$, rounded to 1 decimal place. The optimization direction (maximize or minimize) is chosen uniformly at random and is kept fixed for the entire instance.

Polyhedral perturbation range. The polyhedral range is set to $p_r = 0.1 \cdot r$ (1 decimal place), where r is the nominal range defined above. A perturbation-bound type $\tau_p \in \{a,\ b,\ c\}$ is selected uniformly at random *once per instance* and shared across all polyhedral uncertain rows:

$$\tau_p = a:\ p_l = -p_r,\ p_u = p_r; \qquad \tau_p = b:\ p_l = 0,\ p_u = p_r; \qquad \tau_p = c:\ p_l = -p_r,\ p_u = 0.$$

Deterministic row guarantee. The generator enforces that *exactly one* row (counting the objective as row 0 and the n constraints as rows 1 through n) is fully deterministic (uncertainty type $= 0$). If the nominal LP (Step 1) happens to produce an equality constraint at row j, that row is forced to be the deterministic row. Otherwise, the deterministic row is chosen uniformly at random from $\{0, 1, \ldots, n\}$. If the forced deterministic row is the objective (row 0), all n constraint rows may be uncertain; otherwise, the objective row is assigned an uncertainty type drawn uniformly from $\{1, 2, 3\}$ (box / budget / polyhedral). Each remaining constraint row is assigned an uncertainty type independently and uniformly from $\{1, 2, 3\}$.

B.2 Step 4: solve and filter

After the instance parameters are fully generated, the robust model is constructed and solved using `RSOME` (via `rsome.ro.Model`). An instance is *accepted* and added to the dataset only if *all three* of the following conditions hold:

1. **Not all-deterministic.** At least one row (objective or constraint) must have a positive uncertainty type; instances where every row is deterministic are discarded before solving.
2. **Feasible and bounded.** The solver must return status `optimal`; instances that are infeasible, unbounded, or produce a solver error are discarded.
3. **Non-degenerate solution.** The optimal solution $\mathbf{x}^*$ must not be identical to the variable lower or upper bound at every component, i.e., it is required that $\mathbf{x}^*$ is not coordinatewise equal to $\mathbf{x}_l$ or $\mathbf{x}_u$. Formally, the instance is rejected if $x_i^* \approx x_l$ or $x_i^* \approx x_u$ for *all* i, as such solutions are considered degenerate boundary solutions that offer little reformulation challenge.

Rejected instances are discarded entirely (all parameters are re-sampled), and the generator loops until the target number of valid instances is reached.

B.3 Shared module: feasible polytope generator

A key primitive used in two places of the generator is a procedure that constructs a random system of linear inequalities for which a prescribed point is guaranteed to be a strictly interior point. It is invoked as:

1. **Nominal LP** (Step 1): generate the constraint matrix $\mathbf{A}$ and right-hand side $\mathbf{b}$ such that $\mathbf{x}_0$ is strictly feasible.
2. **Polyhedral uncertainty set** (Step 3): generate the halfspace representation $(\mathbf{F}, \mathbf{g})$ of the perturbation polytope $\mathcal{U}_{\mathrm{poly}}$ such that the sampled interior point $\boldsymbol{\zeta}^0$ is guaranteed to lie strictly inside.

Algorithm 2 FEASIBLEPOLYTOPE: random feasible inequality system generator

Require: Dimension d; interior point $\mathbf{v}^0 \in \mathbb{R}^d$; coefficient range $[l, u]$; number of rows m
Ensure: Matrix $\mathbf{A} \in \mathbb{R}^{m \times d}$, vector $\mathbf{b} \in \mathbb{R}^m$, directions $\mathbf{s} \in \{\leq, \geq\}^m$

for $i = 1, \ldots, m$ **do**
  Sample $\mathbf{a}_i \sim \mathrm{Uniform}([l, u]^d)$; clip $|a_{ij}| \geq 0.1$; round to 1 decimal
  if $d > 2$ **then** apply random binary mask, keeping ≥ 2 non-zero entries per row
  end if
  Sample $s_i \sim \mathrm{Uniform}(\{\leq, \geq\})$
  $v_i \leftarrow \mathbf{a}_i^\top \mathbf{v}^0$
  if $s_i = \leq$ **then** $b_i \sim \mathrm{Uniform}(v_i,\ v_i + \|\mathbf{a}_i\|_1 \cdot |u|)$ ▷ slack above $\mathbf{v}^0$
  else $b_i \sim \mathrm{Uniform}(v_i - \|\mathbf{a}_i\|_1 \cdot |u|,\ v_i)$ ▷ slack below $\mathbf{v}^0$
  end if
  Round b_i to 2 decimal places
end for
return $\mathbf{A}$, $\mathbf{b}$, $\mathbf{s}$

Equality constraints in the nominal LP. When constructing the nominal constraint system (invocation 1 above), a single row may additionally be converted to an equality: for $n \geq 3$, this occurs with probability 0.5 for one randomly chosen row, setting $b_i = \mathbf{a}_i^\top \mathbf{x}_0$ exactly. Such a row is always designated as the deterministic row of the RO instance.

B.4 Step 3: uncertainty parameter generation

Both the nominal LP constraint matrix $\mathbf{A}$ and the polyhedral uncertainty polytopes are sparse by design. For $n \geq 3$, each row of $\mathbf{A}$ has at least two but generally fewer than n non-zero entries, determined by the random binary mask in FEASIBLEPOLYTOPE (Algorithm 2). Uncertainty is further sparsified at the row level: for each uncertain row i, a random subset $\mathcal{S}_i$ of $|\mathcal{S}_i| \geq 2$ indices is selected from the non-zero coefficient positions, so that only a portion of each row's coefficients is subject to perturbation. Uncertainty parameters are generated as follows.

Box uncertainty. Per-element deviations are set as $\Delta_{i,j} = \max(0.1,\ \delta_{ij} \cdot |\bar{a}_{i,j}|)$ with $\delta_{ij} \sim \mathrm{Uniform}(0.05, 0.20)$, rounded to 1 decimal. The uncertainty set is:

$$\mathcal{U}_{\mathrm{box}} = \big\{\boldsymbol{\zeta}_i \in \mathbb{R}^{|\mathcal{S}_i|} : |\zeta_{i,j}| \leq \Delta_{i,j},\ \forall j \in \mathcal{S}_i\big\}.$$

Budget uncertainty. The same per-element deviations $\Delta_{i,j}$ are used, with an additional budget $\Gamma_i \sim \mathrm{Uniform}(0, |\mathcal{S}_i|)$ (rounded to 1 decimal) limiting total perturbation:

$$\mathcal{U}_{\mathrm{bud}} = \Big\{\boldsymbol{\zeta}_i : |\zeta_{i,j}| \leq \Delta_{i,j},\ \sum_{j \in \mathcal{S}_i} \frac{|\zeta_{i,j}|}{\Delta_{i,j}} \leq \Gamma_i\Big\}.$$

Polyhedral uncertainty. A polyhedral range $p_r = 0.1r$ is set, and component-wise bounds $[p_l, p_u]$ are chosen uniformly from $\{[-p_r, p_r],\ [0, p_r],\ [-p_r, 0]\}$ (one choice shared across all polyhedral rows of the same instance). An interior point $\boldsymbol{\zeta}^0 \sim \mathrm{Uniform}([p_l, p_u]^{|\mathcal{S}_i|})$ is sampled. The halfspace representation is obtained by calling FEASIBLEPOLYTOPE$(|\mathcal{S}_i|,\ \boldsymbol{\zeta}^0,\ [-p_r, p_r],\ |\mathcal{S}_i|)$ and retaining only the first $|\mathcal{S}_i| - 1$ inequality rows to form $\mathbf{F} \in \mathbb{R}^{(|\mathcal{S}_i|-1) \times |\mathcal{S}_i|}$ and $\mathbf{g} \in \mathbb{R}^{|\mathcal{S}_i|-1}$. The resulting uncertainty set is:

$$\mathcal{U}_{\mathrm{poly}} = \big\{\boldsymbol{\zeta}_i \in \mathbb{R}^{|\mathcal{S}_i|} : \mathbf{F}\boldsymbol{\zeta}_i \leq \mathbf{g},\ p_l \leq \zeta_{i,j} \leq p_u,\ \forall j\big\}.$$

By construction, $\boldsymbol{\zeta}^0$ is strictly interior to $\mathcal{U}_{\mathrm{poly}}$, guaranteeing non-emptiness.

B.5 Template dimensions

The template set $\mathcal{T}$ comprises a family of 8 templates $T_{b_0 b_1 b_2}$, parameterized by three independent binary dimensions: explicit/implicit worst-case direction (b_0), vector/scalar notation style (b_1), and explicit/implicit uncertainty type annotation (b_2). Using diverse templates encourages AutoREM to

Table 4: The three binary dimensions parameterizing the 8 LaTeX presentation templates.

Dim.	Name	Value $= 0$ (Implicit)	Value $= 1$ (Explicit)
b_0	Worst-case direction	Universal quantifier $\forall \boldsymbol{\xi} \in \mathcal{U}$	Nested operator $\min_{\boldsymbol{\xi}\in\mathcal{U}}$ / $\max_{\boldsymbol{\xi}\in\mathcal{U}}$
b_1	Notation style	Scalar: each term written element-wise	Vector/matrix: compact inner-product form
b_2	Uncertainty type annotation	No label; set given by math definition only	Named label, e.g., "Box Uncertainty"

Table 5: Input-description token statistics for the original reformulation test splits.

Split	Instances	Mean tokens	Token range
Random	64	720.00	282–1,450
Hard	32	885.97	348–1,812
Large	32	1,579.66	882–2,419

learn notation-invariant reformulation knowledge, preventing the experience memory from simply memorizing surface-level text styles.

Each of the 8 presentation templates $T_{b_0 b_1 b_2}$ is fully determined by three independent binary dimensions, summarized in Table 4.

B.6 Hard dataset configuration

The *Hard* dataset is generated by the same pipeline as the *Random* dataset (Algorithm 1), but with the random sampling constrained so that each instance simultaneously satisfies four structural conditions identified as failure modes of the base LLM during preliminary experiments:

- all uncertain constraints use polyhedral uncertainty;
- the objective direction is maximization;
- the uncertain constraints include both $\geq$ and $\leq$ directions;
- decision variables are allowed to take both positive and negative values.

The decision-variable count remains $n \in \{2, 3, 4, 5\}$ as in the Random split, so the accuracy gap relative to Random is attributable to these structural difficulty factors rather than to dimensionality.

B.7 Split independence and input length

The original reformulation test splits contain 64 *Random*, 32 *Hard*, and 32 *Large* instances. Training, validation, and these test instances are independently generated: no underlying problem instance, uncertainty-parameter set, or rendered variant is reused across them. The generator and presentation-template family are shared, rather than the generated instances. The original 32-instance *Application* set is evaluated with the frozen memory learned from the reformulation training distribution.

Table 5 characterizes the problem descriptions using the DeepSeek tokenizer. Only each instance's description field is encoded. System prompts, memory, and generated outputs are excluded. *Large* has 2.19 times the mean description length of *Random*. The generator increases both the number of variables and the number of nominal constraints from $n \in \{2, 3, 4, 5\}$ to $n \in \{6, 7\}$, with the constraint count set to n. Thus model dimension, uncertainty descriptions, numerical matrices, and context length increase together.

C RO application dataset: literature-motivated parameter generation

Source-problem selection and coverage. The original Application set is a curated stress test rather than an exhaustive sample of industrial robust optimization. Its 16 nominal problems are drawn equally from IndustryOR and OptiBench, with eight problems from each source and two robust variants per problem. Selection prioritizes meaningful uncertainty in multiple rows: the relevant coefficients have physical or economic interpretations that can plausibly vary. Many transportation, assignment, and allocation formulations in the source benchmarks predominantly use structural coefficients such as 0,

1, or -1. Imposing continuous uncertainty on these coefficients can be physically artificial. Fifteen selected nominal problems support multi-row uncertainty, while one has only objective uncertainty. For each nominal problem, the first variant uses the reformulation benchmark's numerical parameter-generation procedure, while the second uses the literature-motivated settings detailed below. This gives diverse and interpretable tests within the supported uncertainty families, while retaining the limitations of a small curated set. The full problem-class, variable-count, and uncertain-row statistics are reported in Appendix E.

For the second RO instance of each nominal problem in the RO application dataset, uncertainty parameters are generated following practically-motivated settings from the RO literature.

Uncertain row selection. A row is made uncertain only if it contains more than two coefficients that are simultaneously non-zero and not equal to ± 1. Let $\mathcal{S}$ denote the index set of such qualifying coefficients, $m = |\mathcal{S}|$ the number of uncertain coefficients, and define the scale baseline $s = \frac{1}{m}\sum_{j\in\mathcal{S}} |\bar{a}_j|$, where $\bar{\mathbf{a}}$ are the nominal coefficients of the row. Note that m is distinct from n (the number of decision variables defined in Section 3.1).

Box uncertainty [4]. Each deviation is $\Delta_j = p \cdot |\bar{a}_j|$ for $j \in \mathcal{S}$, where p is randomly chosen from $\{0.01\%,\ 0.1\%,\ 1\%\}$:

$$\mathcal{U}_{\text{box}} = \{\boldsymbol{\xi} : |\xi_j| \le \Delta_j,\ \forall j \in \mathcal{S}\}.$$

Budget uncertainty. The same deviation Δ_j as for box is used. To reflect varying degrees of conservatism, the budget parameter Γ_b is randomly chosen to be 40%, 60%, or 80% of m. Note that Γ_b here is the budget threshold and is distinct from the robustness level parameter Γ used in the polyhedral sets below:

$$\mathcal{U}_{\text{bud}} = \Big\{\boldsymbol{\xi} : |\xi_j| \le \Delta_j,\ \sum_{j\in\mathcal{S}} \tfrac{|\xi_j|}{\Delta_j} \le \Gamma_b\Big\}.$$

Polyhedral uncertainty [1]. One of the following four statistically-motivated types is randomly selected. All use a fixed robustness level $\Gamma = 2.0$. Unlike box and budget uncertainty where all parameters are presented as explicit numerical values, these sets define their constraint bounds through symbolic expressions involving m and sampled distribution parameters (e.g., $\Gamma\sigma\sqrt{m}$, m/λ). These expressions are retained symbolically in the problem description and must be preserved accordingly in the reformulation output to avoid numerical precision loss.

CLT-based. Derived from the Central Limit Theorem for i.i.d. random variables with mean μ and standard deviation σ:

$$\mathcal{U}_{CLT} = \Big\{\boldsymbol{\xi} \;\Big|\; -\Gamma\sigma\sqrt{m} \le \textstyle\sum_{j\in\mathcal{S}} \xi_j - m\mu \le \Gamma\sigma\sqrt{m}\Big\},$$

where $\mu \sim \text{Uniform}(-0.1s,\ 0.1s)$ and $\sigma \sim \text{Uniform}(0.05s,\ 0.2s)$.

Heavy-tail. For heavy-tailed distributions with tail index $\alpha \in (1, 2]$, sums scale as $m^{1/\alpha}$ rather than $\sqrt{m}$:

$$\mathcal{U}_{HT} = \Big\{\boldsymbol{\xi} \;\Big|\; -\Gamma \le \frac{\sum_{j\in\mathcal{S}} \xi_j - m\mu}{m^{1/\alpha}} \le \Gamma\Big\},$$

where $\mu \sim \text{Uniform}(-0.1s,\ 0.1s)$ and $\alpha = 1.5$.

Typical set – exponential. Based on the typical set of i.i.d. Exponential(λ) variables, with non-negative perturbations:

$$\mathcal{U}^E = \Big\{\boldsymbol{\xi} \;\Big|\; \tfrac{m}{\lambda} - \tfrac{\sqrt{m}}{\lambda}\Gamma \le \textstyle\sum_{j\in\mathcal{S}} \xi_j \le \tfrac{m}{\lambda} + \tfrac{\sqrt{m}}{\lambda}\Gamma,\ \xi_j \ge 0\Big\},$$

where $\lambda \sim \text{Uniform}\big(\frac{1}{0.2s},\ \frac{1}{0.05s}\big)$.

Typical set – uniform. Based on the typical set of i.i.d. Uniform(a, b) variables:

$$\mathcal{U}^U = \Big\{\boldsymbol{\xi} \;\Big|\; m\cdot\tfrac{a+b}{2} - \Gamma\sqrt{m} \le \textstyle\sum_{j\in\mathcal{S}} \xi_j \le m\cdot\tfrac{a+b}{2} + \Gamma\sqrt{m},\ a \le \xi_j \le b\Big\},$$

where $a, b \sim \text{Uniform}(-0.2s,\ 0.2s)$ with $a < b$.

D RO application dataset: problem description format

Each instance in the RO application dataset consists of the original natural-language problem description from IndustryOR or OptiBench, followed by an appended **Robust Extension** paragraph. The extension begins with a preamble explaining the robust optimization setup (uncertain parameters modeled as nominal plus perturbation, worst-case feasibility requirement), lists each uncertain row with its uncertainty set description, and closes with a statement of the problem objective.

Format of the Robust Extension. The appended text follows this structure:

1. **Preamble.** A fixed paragraph explaining that certain parameters are uncertain, perturbations are confined to uncertainty sets, and the goal is a worst-case feasible decision.
2. **Uncertain parameter descriptions.** One bullet per uncertain row, identifying the physical meaning of the uncertain coefficients and specifying the uncertainty set. The set is described as:
   - *Box:* "Each coefficient may deviate by at most $\pm[\Delta_1, \ldots, \Delta_m]$ from its nominal value."
   - *Budget:* Same as box, followed by "the sum of normalized deviations ($\sum |\xi_j|/\Delta_j$) across all m uncertain coefficients must not exceed $\Gamma_b = \langle\text{value}\rangle$."
   - *Polyhedral:* "The perturbations $\boldsymbol{\xi}$ are confined to a polyhedral uncertainty set with each ξ_i bounded within $[l, u]$, and satisfying the following k linear constraint(s): $\langle$constraint list$\rangle$."
3. **Closing statement.** "The robust model seeks a decision that guarantees [maximizing/minimizing] the objective value under all worst-case realizations within the uncertainty sets above."

Example. The following is an excerpt from one instance (a bakery production problem from OptiBench), illustrating the box uncertainty format:

> *Robust Extension:* In this robust version, certain parameters in the problem are treated as uncertain ...
>
> The following parameters are subject to uncertainty:
>
> • *Objective coefficients*: The profit per cake type may vary due to ingredient cost and pricing fluctuations. Each relevant coefficient may fluctuate independently. The maximum absolute deviation for each coefficient is $\pm[1.97,\ 1.64,\ 2.46]$.
>
> • *Batter required per cake*: The batter quantity per cake may vary due to recipe variation. Each relevant coefficient may fluctuate independently. The maximum absolute deviation for each coefficient is $\pm[71.76,\ 89.70,\ 80.73]$.
>
> • *Milk required per cake*: May vary. The maximum absolute deviation for each coefficient is $\pm[26.67,\ 40.00,\ 46.67]$.
>
> The robust model seeks a decision that guarantees maximizing the objective value under all worst-case realizations within the uncertainty sets above.

E RO application dataset: statistics

This section reports detailed statistics of the RO application dataset and contrasts them with the RO reformulation dataset (Random split). Since each application instance is curated from a real-world problem rather than algorithmically generated, its problem-level characteristics differ substantively from the reformulation dataset, while the per-instance reformulation challenge—measured by the number of uncertain rows—remains broadly comparable. We index the 16 nominal problems as P1–P16 (P1–P8 from IndustryOR, P9–P16 from OptiBench), each contributing 2 instances as described in Section 4.1.

Problem class. The 16 nominal problems comprise 7 LPs and 9 MILPs, yielding 14 LP and 18 MILP instances after the two-instance expansion. The reformulation dataset is exclusively LP, since its automatic generator (Algorithm 1) does not produce integer variables. The presence of MILPs in the application dataset stress-tests AutoREM's ability to apply LP-derived reformulation memory to mixed-integer settings.

Table 6: Detailed statistics of the RO application dataset (32 instances). Each source problem produces 2 instances.

Attribute	Value	# Instances	Source Problems
Problem class	LP	14	P1, P2, P6, P8, P9, P11, P13
	MILP	18	P3, P4, P5, P7, P10, P12, P14, P15, P16
# decision variables	2	12	P4, P8, P13, P14, P15, P16
	3	8	P3, P5, P7, P10
	4	4	P9, P12
	5	4	P6, P11
	6	2	P2
	13	2	P1
# uncertain rows	1	2	P3 (objective only)
	2	6	P8, P14, P16
	3	12	P1, P7, P10, P12, P13, P15
	4	12	P2, P4, P5, P6, P9, P11

Decision variables. Application instances contain 2 to 13 decision variables (median 3), spanning a wider range than the reformulation Random split's $\{2, 3, 4, 5\}$. The 13-variable instance (P1) substantially exceeds the dimensionality of any reformulation training instance, providing an additional dimensionality stress test.

Uncertain rows. The number of uncertain rows, i.e., the count of objective and constraint rows containing uncertain coefficients, is the most direct indicator of reformulation difficulty. Each additional uncertain row requires an independent counterpart derivation, increasing both reasoning depth and output context length. Application instances contain 1 to 4 uncertain rows, with 75% (24/32) containing 3 or 4. By construction, every reformulation instance with n decision variables contains exactly n uncertain rows (one of the $n+1$ rows is forced deterministic; see Appendix B.1), giving 2–5 uncertain rows for the Random split. Thus, despite the application dataset's smaller size, its per-instance reformulation challenge is broadly comparable to that of the reformulation dataset.

F Sample dataset instances

This section presents one concrete instance from each dataset to complement the abstract descriptions in Section 4.1. We use the last instance of each dataset.

F.1 RO reformulation dataset instance (Random split)

The instance has identifier `5_16_T011`, with $n = 5$ decision variables and 5 constraints. The verified ground-truth solution is $\mathbf{x}^* = [1.8,\ 1.0231,\ 1.8,\ 0,\ 0.8]^\top$ with optimal value $f^* = 29.6985$.

Problem (LLM input).

$$
\begin{array}{lllll}
\text{Maximize} & \mathbf{c}^\top \mathbf{x} & & \forall \mathbf{c} \in \mathcal{U}_{c,Bud} & \text{(Budgeted Uncertainty Objective)} \\
\text{subject to} & \mathbf{a}_1^\top \mathbf{x} \geq b_1 & & \forall \mathbf{a}_1 \in \mathcal{U}_{a,Box} & \text{(Box Uncertainty Constraint)} \\
& \mathbf{a}_2^\top \mathbf{x} \leq b_2 & & & \text{(Deterministic Constraint)} \\
& (\mathbf{a}_3^{nom} + \boldsymbol{\delta}_{a3})^\top \mathbf{x} \geq b_3 & & \forall \boldsymbol{\delta}_{a3} \in \mathcal{U}_{a3,P} & \text{(Polyhedral Uncertainty Constraint)} \\
& (\mathbf{a}_4^{nom} + \boldsymbol{\delta}_{a4})^\top \mathbf{x} \leq b_4 & & \forall \boldsymbol{\delta}_{a4} \in \mathcal{U}_{a4,P} & \text{(Polyhedral Uncertainty Constraint)} \\
& (\mathbf{a}_5^{nom} + \boldsymbol{\delta}_{a5})^\top \mathbf{x} \geq b_5 & & \forall \boldsymbol{\delta}_{a5} \in \mathcal{U}_{a5,P} & \text{(Polyhedral Uncertainty Constraint)}
\end{array}
$$

$$
\begin{aligned}
\text{where:}\quad & \mathcal{U}_{c,Bud} = \{\mathbf{c} \in \mathbb{R}^5 : \mathbf{c} = \mathbf{c}^{nom} + \mathbf{D}_c \boldsymbol{\xi}_c, \|\boldsymbol{\xi}_c\|_\infty \leq 1, \|\boldsymbol{\xi}_c\|_1 \leq \Gamma_c\} \\
& \mathcal{U}_{a,Box} = \{\mathbf{a}_1 \in \mathbb{R}^5 : \mathbf{a}_1 = \mathbf{a}_1^{nom} + \mathbf{D}_{a1} \boldsymbol{\xi}_{a1}, \|\boldsymbol{\xi}_{a1}\|_\infty \leq 1\} \\
& \mathcal{U}_{a3,P} = \{\boldsymbol{\delta}_{a3} \in \mathbb{R}^5 : \mathbf{M}_{a3} \boldsymbol{\delta}_{a3} \leq \mathbf{q}_{a3}, \quad \mathbf{E}_{a3} \boldsymbol{\delta}_{a3} = \mathbf{e}_{a3}\} \\
& \mathcal{U}_{a4,P} = \{\boldsymbol{\delta}_{a4} \in \mathbb{R}^5 : \mathbf{M}_{a4} \boldsymbol{\delta}_{a4} \leq \mathbf{q}_{a4}, \quad \mathbf{E}_{a4} \boldsymbol{\delta}_{a4} = \mathbf{e}_{a4}\} \\
& \mathcal{U}_{a5,P} = \{\boldsymbol{\delta}_{a5} \in \mathbb{R}^5 : \mathbf{M}_{a5} \boldsymbol{\delta}_{a5} \leq \mathbf{q}_{a5}, \quad \mathbf{E}_{a5} \boldsymbol{\delta}_{a5} = \mathbf{e}_{a5}\}
\end{aligned}
$$

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]^\top$$
$$\mathbf{c}^{nom} = [7.7, -1.8, 8.8, -2.9, 2.7]^\top$$
$$\mathbf{a}_1^{nom} = [0, 0, -0.1, 0, -0.7]^\top$$
$$b_1 = -1$$
$$\mathbf{a}_2 = [-0.2, -1.3, 0, -0.7, -1.3]^\top$$
$$b_2 = -2.73$$
$$\mathbf{a}_3^{nom} = [-0.4, 0, 0, 0, -1.1]^\top$$
$$b_3 = -2.64$$
$$\mathbf{a}_4^{nom} = [0, 1.8, -1.4, -0.2, 0.9]^\top$$
$$b_4 = 6.8$$
$$\mathbf{a}_5^{nom} = [0.3, 1, -1.1, 0, 0]^\top$$
$$b_5 = -2.76$$
$$\mathbf{D}_c = \text{diag}(0, 0, 0, 0.5, 0.5)$$
$$\Gamma_c = 0.8$$
$$\mathbf{D}_{a1} = \text{diag}(0, 0, 0.1, 0, 0.1)$$
$$\mathbf{M}_{a3} = \begin{bmatrix} -0.2 & 0 & 0 & 0 & -0.2 \\ 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$
$$\mathbf{q}_{a3} = [0.06, 0, 0.2, 0, 0.2]^\top$$
$$\mathbf{E}_{a3} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$
$$\mathbf{e}_{a3} = [0, 0, 0]^\top$$
$$\mathbf{M}_{a4} = \begin{bmatrix} 0 & -0.2 & 0 & 0 & -0.1 \\ 0 & -0.1 & 0 & 0.1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$
$$\mathbf{q}_{a4} = [0.04, 0.01, 0, 0.2, 0, 0.2, 0, 0.2]^\top$$
$$\mathbf{E}_{a4} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$
$$\mathbf{e}_{a4} = [0, 0]^\top$$
$$\mathbf{M}_{a5} = \begin{bmatrix} -0.1 & -0.1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{q}_{a5} = [0.04, 0, 0.2, 0, 0.2]^\top$$
$$\mathbf{E}_{a5} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\mathbf{e}_{a5} = [0, 0, 0]^\top$$
$$0 \le x_j \le 1.8, \quad \forall j \in \{1, \ldots, 5\} \qquad \text{(Variable Bounds)}$$

F.2 RO application dataset instance

The instance has identifier 32, sourced from OptiBench [64]. The verified ground-truth optimal value is $f^* = 1761.568$.

Problem (LLM input).

There has been an oil spill in the ocean and ducks need to be taken to shore to be cleaned either by boat or by canoe. A boat can take 10 ducks per trip while a canoe can take 8 ducks per trip. Since the boats are motor powered, they take 20 minutes per trip while the canoes take 40 minutes per trip. In order to avoid further environmental damage, there can be at most 12 boat trips and at least 60% of the trips should be by canoe. If at least 300 ducks need to be taken to shore, how many of each transportation method should be used to minimize the total amount of time needed to transport the ducks?

Robust Extension:

In this robust version, certain parameters in the problem are treated as uncertain — they are not fixed at their nominal values. Instead, each uncertain parameter is modeled as its nominal value plus a perturbation, and it is this perturbation that is confined within specified uncertainty sets. The goal is to find a decision plan that remains feasible and near-optimal under all possible realizations of these uncertain parameters (the worst-case guarantee).

The following parameters are subject to uncertainty:

• Objective coefficients: The time per trip for boats and canoes may vary due to water conditions. Each coefficient may deviate by at most 0.10% of its nominal value. Additionally, the combined perturbation is limited: the sum of normalized deviations $(\sum |\xi_j|/d_j)$ must not exceed $\Gamma = 1.2$, which means on average at most 60% of the uncertain coefficients can simultaneously reach their maximum deviation. This captures the realistic situation where large simultaneous deviations across all parameters are unlikely.

• The number of ducks transported per trip may vary due to animal behavior. This uncertainty set accounts for heavy-tailed distributions (tail index $\alpha = 1.5$): the normalized sum $(\Sigma\xi_i - n\mu)/n^{1/\alpha}$ must lie within $[-\Gamma, +\Gamma]$, so the sum range is $[n\mu - \Gamma\cdot n^{1/\alpha}, n\mu + \Gamma\cdot n^{1/\alpha}]$. Compared to the CLT-based set, the normalization factor $n^{1/\alpha}$ grows more quickly than $n^{1/2}$, allowing for heavier tails. Each component is bounded within $[\mu - \Gamma\cdot n^{1/\alpha - 1}, \mu + \Gamma\cdot n^{1/\alpha - 1}]$ (parameters: $\mu = -0.8139$, $\alpha = 1.5$, $\Gamma = 2.0$, $n = 2$).

• *Variable type*: The decision variables (x_1, x_2) represent trips by each transport method. Since these quantities are inherently discrete, all decision variables are constrained to be non-negative integers ($x_i \in \mathbb{Z}_+$).

The robust model seeks a decision that guarantees minimizing the objective value under all worst-case realizations within the uncertainty sets above.

G Evaluation reliability: mathematical and coder audits

A matching objective is an automatic evaluation signal, not a certificate that the generated counterpart is mathematically equivalent to the input robust model. In particular, an incorrect model can share an optimum with the reference model on a particular instance. Comparing arbitrary formulations is also complicated by equivalent representations involving permutations, scaling, redundant constraints, auxiliary variables, and extended formulations [65, 66]. The model and coder audits provide additional evidence for the evaluation reliability of objective matching.

G.1 Expert audit of mathematical counterparts

Three operations research experts manually audited all 128 outputs from one pre-selected *Random* run: 64 Base LLM outputs and 64 AutoREM outputs. They first judged each mathematical counterpart without viewing its generated code, and then examined whether the coder faithfully implemented the counterpart. This separates errors in the reformulator's mathematical output from changes introduced during coding. Table 7 shows model-level judgments and the original objective-matching outcomes. "Model correct" refers to the reformulator's mathematical counterpart, before coding. The Base LLM produced 8 incorrect mathematical counterparts, whereas AutoREM produced 1. Three of the Base LLM's 58 objective-positive outputs were structurally incorrect, compared with none of AutoREM's 62 objective-positive outputs. AutoREM's improvement is therefore 10.9% under direct model judgment (56/64 versus 63/64), compared with 6.3% under objective matching (58/64 versus 62/64). The imperfect objective metric does not account for the observed advantage.

G.2 Coder fidelity and error attribution

Table 8 shows coder fidelity on the same 128 outputs. Among the 56 correct Base LLM counterparts, the coder introduced one structural error (1/56, or 1.8%), which changed the objective. Among the

Table 7: Structural audit of 64 outputs per method. "Objective only" means that the solved objective matches the reference despite an incorrect mathematical counterpart. "Model only" means a correct mathematical counterpart whose evaluated objective does not match.

Method	Both correct	Objective only	Model only	Both wrong	Objective accuracy	Model accuracy
Base LLM	55	3	1	5	90.6%	87.5%
AutoREM	62	0	1	1	96.9%	98.4%

Table 8: Coder audit and objective-failure attribution on the same 64 outputs per method. Structural alterations count coder changes to the mathematical counterpart, including changes that do not affect the optimum. The repair denominator is the number of incorrect mathematical counterparts.

Audit result	Base LLM	AutoREM
Incorrect mathematical counterparts	8/64 (12.5%)	1/64 (1.6%)
Objective failures from reformulation errors	5/64 (7.8%)	1/64 (1.6%)
Coder structural alterations	1/64 (1.6%)	2/64 (3.1%)
Coder-only objective failures	1/64 (1.6%)	1/64 (1.6%)
Verified coder repairs	0/8	0/1
Objective accuracy after correcting coder-only errors	59/64 (92.2%)	63/64 (98.4%)

63 correct AutoREM counterparts, it introduced two structural errors (2/63, or 3.2%). Only one changed the objective (1/63, or 1.6%), while the other was nonbinding. No incorrect mathematical counterpart was verified as repaired by the coder. The Base LLM's six objective failures comprise five reformulator-caused failures and one coder-only failure; AutoREM's two comprise one of each. Correcting only the confirmed coder-only errors changes objective accuracy from 58/64 to 59/64 for Base LLM and from 62/64 to 63/64 for AutoREM. The gap remains 4/64, or 6.25%. The main improvement on this run consequently comes from reducing incorrect reformulations from eight to one, with small and balanced outcome-affecting coder errors.

H Baseline implementation and offline adaptation protocols

AutoREM, ACE, and ReasoningBank each receive $10 \times 8 = 80$ training-sample exposures from the same RO generator and distribution. They use DeepSeek-V4-Flash for reformulation and memory agents, the same fixed DeepSeek-V4-Pro coder, and the same Gurobi solver. Memory construction occurs offline, and all memories are frozen during testing. This matches training exposure and the base models.

ACE. The playbook starts with seven empty section headings and no RO rules. Each sample passes through its Generator and the common coder/solver. The Generator–Reflector–Curator mechanism is retained, using three Reflector calls per sample, addition-only curation, and semantic deduplication. At test time, the frozen playbook is supplied to the Generator, followed by the common coder and evaluator. No test-time playbook updates are performed.

ReasoningBank. The bank starts empty. When entries are available, each training trajectory retrieves the top-1 entry by cosine similarity, runs the common reformulator–coder–solver pipeline, and uses the original successful- or failed-trajectory induction prompt to append memory. Its accumulation is moved from an online test stream to the offline training split. At test time, retrieval uses the frozen bank and performs no further memory induction.

Formulation-method baselines. Base LLM, Max Thinking, Expert Prompt, ReasoningBank, and ACE directly address pure reformulation. AlphaOPT, LEAN-LLM-OPT, and OptiTree primarily address the end-to-end Application task. In their additional pure-reformulation evaluations, the natural-language-to-nominal-formulation step is bypassed: each receives the same LaTeX robust model and is instructed to produce a deterministic robust counterpart. Each retains its original pipeline, and final objectives are evaluated with the same 10^{-4} matching tolerance. These comparisons test transfer of formulation-oriented pipelines to algebraic reformulation.

Algorithm 3 Offline adaptation of reformulation experience memory

Require: Epochs E; steps per epoch S; DC batch size B; max inner iters K; validation set $\mathcal{V}$; initial memory $\mathcal{M}$
Ensure: Adapted memory $\mathcal{M}^*$

1: $\mathcal{M}^* \leftarrow \mathcal{M}$; $\text{acc}^* \leftarrow \textsc{EvalVal}(\mathcal{M}, \mathcal{V})$
2: **for** $e = 1, \ldots, E$ **do**
3: **for** $s = 1, \ldots, S$ **do**
4: Sample training instance q_s on the fly
5: $(e_m, \text{fb}, y) \leftarrow \textsc{Eval}(q_s, \mathcal{M})$ ▷ Reformulator + Coder + Solver
6: **if** $e_m = 1$ **then continue**
7: **end if** ▷ Already correct — skip
8: $\hat{\mathcal{M}} \leftarrow \mathcal{M}$; prev_trace $\leftarrow \varnothing$; dc_fail $\leftarrow \varnothing$; committed $\leftarrow$ False
9: **Lock** DC sample set $\mathcal{V}_s \subseteq \mathcal{V}$, $|\mathcal{V}_s| \leq B$ ▷ linked-first priority; locked for this step
10: **for** $k = 1, \ldots, K$ **do** ▷ Inner reflection loop
11: **[SMO]** ops $\leftarrow \textsc{Reflect}(q_s, \hat{\mathcal{M}}, y, \text{fb}, \text{prev_trace}, \text{dc_fail})$
12: **[ULE]** Apply ops $\in \{\textsc{add}, \textsc{update}, \textsc{delete}\}$ to $\hat{\mathcal{M}}$
13: $(e'_m, \text{fb}', y') \leftarrow \textsc{Eval}(q_s, \hat{\mathcal{M}})$; prev_trace $\leftarrow (k, \text{ops}, e'_m, \text{fb}')$; dc_fail $\leftarrow \varnothing$
14: **if** $e'_m = 0$ **then** $y \leftarrow y'$; fb $\leftarrow$ fb$'$; **continue**
15: **end if**
16: **[Dual-Check Commit]** (pass, dc_fail) $\leftarrow \textsc{DualCheck}(\hat{\mathcal{M}}, \mathcal{V}_s)$
17: **if** pass **then** $\mathcal{M} \leftarrow \hat{\mathcal{M}}$; committed $\leftarrow$ True; **break**
18: **else** $y \leftarrow y'$; fb $\leftarrow$ fb$'$ ▷ DC failed — Reflector gets dc_fail next iter
19: **end if**
20: **end for**
21: **if** $\neg$ committed **then** Discard $\hat{\mathcal{M}}$
22: **end if**
23: **end for**
24: **[Validation-based Acceptance]** acc $\leftarrow \textsc{EvalVal}(\mathcal{M}, \mathcal{V})$
25: **if** acc $\geq$ acc* **then** $\mathcal{M}^* \leftarrow \mathcal{M}$; acc$^* \leftarrow$ acc
26: **else** $\mathcal{M} \leftarrow \mathcal{M}^*$ ▷ Restore best memory
27: **end if**
28: **end for**
29: **return** $\mathcal{M}^*$

I Offline adaptation algorithm

Algorithm 3 provides the complete pseudocode for the offline adaptation procedure introduced in Section 4.3. The procedure iterates over E epochs, each consisting of S training steps. At each step, a training instance q_s is sampled on the fly and evaluated against the current memory $\mathcal{M}$; if already solved correctly, the step is skipped. Otherwise, an inner reflection loop of up to K iterations is entered: the **SMO**-based Reflector proposes structured memory edits (line 11), the **ULE**-based updater applies them at the granularity of individual uncertain constraint rows (line 12), and the updated memory is re-evaluated on q_s. Once the training instance is solved correctly, the **Dual-Check Commit (DCC)** mechanism stress-tests the proposed memory against a locked batch $\mathcal{V}_s$ drawn from the validation set (line 16): only if all samples in $\mathcal{V}_s$ pass is the update committed; otherwise the failure feedback is fed back into the next reflection iteration. If no commit is achieved within K iterations, the candidate memory is discarded and $\mathcal{M}$ remains unchanged. At the end of each epoch, the **Validation-based Acceptance (VBA)** gate evaluates the full validation set $\mathcal{V}$ (line 24): an improvement or tie is accepted; if the epoch-level accuracy falls below the best recorded acc^*, the memory is rolled back to $\mathcal{M}^*$, preventing harmful accumulated changes from persisting. The best memory $\mathcal{M}^*$ is returned after all epochs.

J Offline adaptation dynamics

Figure 7 visualizes the evolution of validation accuracy throughout the offline adaptation process. Starting from an initial accuracy of 87.5%, the model improves rapidly, reaching 89.1% by epoch 1

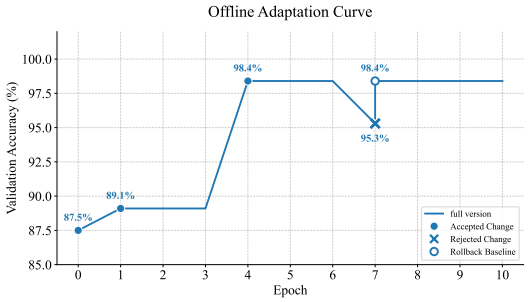


Figure 7: Validation accuracy curve of offline adaptation.

and making a decisive leap to 98.4% by epoch 4. After this point, the accuracy stabilizes at a high plateau. The figure also illustrates the protective role of VBA: at epoch 7, a detrimental memory edit briefly causes validation accuracy to drop to 95.3%; this update is correctly identified as a rejected change and the rollback baseline is triggered, swiftly restoring accuracy to 98.4%. This concretely demonstrates that the VBA gate is not merely a passive quality filter but an active safeguard that prevents harmful updates from permanently degrading hard-won improvements. These dynamic trajectories corroborate the static ablation results in Figure 4: both perspectives jointly confirm that all four components of AutoREM make distinct and significant contributions to its performance.

K Ablation study details

Each ablation variant removes exactly one component of AutoREM while keeping the rest intact. *w/o ULE* is implemented by removing unit-level decomposition from the prompts, so the model must process the full constraint matrix as a monolithic block rather than attributing experiences to individual uncertain rows. *w/o SMO* is implemented by modifying both the prompts and the internal controller to allow only a single addition operator (following Zhang et al. [14]), and replacing operator-aware deduplication with cosine-similarity-based deduplication. This prevents the reflector from updating or deleting existing unit-level entries. *w/o DCC* skips the dual-check step entirely while retaining epoch-level VBA on the fixed validation set. *w/o VBA* removes epoch-level validation and rollback. DCC remains active using a local replay buffer, so memory updates are still subject to a step-level check.

As shown in Figure 4, each removal causes a distinct accuracy drop: ULE (97.4%→90.6%), SMO (92.2%), DCC (93.8%), and VBA (93.8%). The largest drop from ULE confirms that unit-level attribution is the backbone of effective memory reuse. DCC and VBA contribute equally in magnitude but at complementary granularities: DCC catches step-level regressions before they are committed, while VBA prevents epoch-level accumulation of locally beneficial but globally harmful updates.

L Hyperparameter sensitivity analysis

We study the sensitivity of AutoREM to three key hyperparameters: the dual-check size B, the validation set size $|\mathcal{V}|$, and the inner reflection loop cap K.

L.1 Dual-check size B

Figure 5 shows the effect of varying $B \in \{1, 2, 4, 8\}$. Accuracy plateaus at 95.3% for $B \leq 2$, peaks at **97.4%** at $B = 4$, then drops to 93.8% at $B = 8$. The non-monotonic behavior reflects two opposing forces: a larger B improves regression coverage across diverse constraint types, but it also lengthens the failure feedback fed into the Reflector, making it harder to produce a repair that satisfies all sampled instances simultaneously. $B = 4$ balances these two effects.

Table 9: Per-invocation usage of the inner reflection loop during offline adaptation. Iter outcomes are: *training fail* = the training sample is still incorrect under the proposed memory (no DC performed); *DC fail* = training sample correct but the dual-check batch regressed; *DC pass* = both training sample and DC batch correct (commit).

(Epoch, Step)	Iterations Used	Per-iteration outcome
$(1, 1)$	$3\ (= K)$	Iter 1: training fail; Iter 2: DC fail; Iter 3: DC pass (committed)
$(4, 2)$	2	Iter 1: DC fail; Iter 2: DC pass (committed)
$(7, 2)$	1	Iter 1: DC pass (committed; later rolled back by VBA)

L.2 Validation set size $|\mathcal{V}|$

Figure 6 shows that accuracy rises monotonically from 95.3% to **97.4%** as $|\mathcal{V}|$ grows from 8 to 64, while the generalization gap shrinks from 4.7% to 1.0%. Smaller validation sets introduce high variance into the VBA signal; larger sets improve reliability but increase adaptation cost. We adopt $|\mathcal{V}| = 64$ as the default, where the generalization gap has already stabilized near its minimum.

L.3 Inner reflection loop cap K

We report the empirical usage of the inner reflection loop (Algorithm 3, parameter K) on the offline adaptation training run, and use these statistics to assess whether the chosen value $K = 3$ is well-calibrated.

Invocation statistics. Across the full training run ($E = 10$ epochs $\times$ $S = 8$ steps $= 80$ step attempts), the inner reflection loop was invoked exactly 3 times, once per training step that was initially incorrect under the current memory. The remaining 77 steps were already correct on entry and skipped reflection (Algorithm 3, line 6). Table 9 lists each invocation with the iterations it consumed and its per-iteration outcomes. The distribution of iterations per invocation is exactly one invocation each at 1, 2, and 3 iterations, with mean 2.0. Only the first invocation reached the cap K, and it terminated by a successful commit at iteration 3 rather than by exhausting the budget without a fix.

Is $K = 3$ a reasonable setting? The data supports $K = 3$ as a tightly calibrated choice for three reasons:

- *Large enough to enable the hardest reflection.* The first invocation, which bootstrapped the memory from the empty state, required all 3 iterations: the very first iteration could not even fix the training sample, the second produced a fix that regressed on the dual-check batch, and only the third yielded a generalizable update that passed both checks. With $K \leq 2$, this foundational update would have been discarded.
- *Small enough to control cost.* Only 1 of the 3 invocations (33%) reached the cap, and that single cap-hit invocation succeeded at the cap rather than exhausting it. Increasing K to 4 or beyond would not have changed any outcome or call count on this run, because all invocations already terminate within three iterations; it only increases the permitted worst-case budget on other trajectories.
- *Reflection difficulty decays over training.* The three invocations consumed 3, 2, and 1 iterations respectively, indicating that as the memory accumulates correct templates, later reflections converge faster. Beyond epoch 1 the cap is therefore not the binding constraint: it is needed only for the cold-start phase.

In summary, $K = 3$ provides exactly the headroom required for the hardest observed reflection (cold-start) while remaining tight enough to keep Reflector cost low once the memory has matured.

M Qualitative comparison of learned memories

This section provides a qualitative illustration of the knowledge representations learned by AutoREM, ACE, and ReasoningBank, alongside the Expert Prompt used as a baseline. These examples offer a

Table 10: Memory size overview.

Method	# Memory Entries	Entry Style
Expert Prompt	—	Manually written reformulation rules
ReasoningBank	80	One item per training step; multi-paragraph narrative
ACE	51	29 strategies + 18 formula entries + 4 code snippets
AutoREM (Ours)	**8**	Compact; one-liner reformulation templates

concrete basis for interpreting the accuracy differences. As shown in Table 10, AutoREM's memory is **6× smaller than ACE** and **10× smaller than ReasoningBank**, while achieving strictly higher accuracy. The following subsections illustrate the qualitative contrast between these representations.

M.1 AutoREM's learned memory

AutoREM produces 8 compact entries for observed failure modes, rather than exhaustively enumerating all $3 \times 2 \times 2 = 12$ combinations of uncertainty set type, direction (min/max), and objective/-constraint. Memory is learned on demand: reflection is triggered only when the reformulator fails on a training sample. The reformulator prompt requires deriving unmatched units from scratch, so the learned entries supplement observed capability gaps rather than define the full set of supported cases. Each entry is a self-contained, directly applicable template for one specific combination of uncertainty set type and problem component. There is no redundancy, no contradictory guidance, and no implementation-specific noise. This tight, minimal structure is in stark contrast with the memories learned by ACE and ReasoningBank, as shown below.

A deterministic symbolic reformulator is preferable when the grammar, uncertainty families, and complete correct rules are specified and encoded. AutoREM does not aim to outperform this closed-world compiler. It studies automatic acquisition, revision, deletion, and validation of knowledge from failures and solver feedback, starting from empty memory. Extending a compiler requires experts to derive, implement, and verify new rules. AutoREM targets incomplete or evolving coverage. Expert Prompt instead supplies fixed expert knowledge to an LLM rather than executing symbolic rules.

Listing 1: AutoREM: all 8 learned memory entries.

```
[Entry 1]
For an uncertain objective with box uncertainty:
minimize max_{xi in Box} (c_bar + diag(c_hat) xi)^T x,
where Box = {xi : ||xi||_inf <= 1}.
Reformulate as:
  minimize c_bar^T x + sum_j u_j,
  with u_j >= c_hat_j x_j, u_j >= -c_hat_j x_j
  for all j where c_hat_j > 0.
(u_j are free variables, automatically nonneg by the constraints.)

[Entry 2]
For a constraint with budgeted uncertainty:
min_{xi in Budgeted} (a_bar + diag(a_hat) xi)^T x >= b,
where Budgeted = {xi : ||xi||_inf <= 1, ||xi||_1 <= Gamma}.
Reformulate as:
  a_bar^T x - Gamma*p - sum_j q_j >= b,
  with p >= 0, q_j >= 0,
  p + q_j >= a_hat_j x_j, p + q_j >= -a_hat_j x_j, for each j.

[Entry 3]
For a constraint with polyhedral uncertainty:
min_{delta in Poly} (a_bar + delta)^T x >= b,
where Poly = {delta : D delta <= d, E delta = e}.
Reformulate using LP duality:
  a_bar^T x - d^T lambda - e^T mu >= b,
  D^T lambda + E^T mu = -x, lambda >= 0, mu free.

[Entry 4]
For a constraint with box uncertainty:
```

```
min_{xi in Box} (a_bar + diag(a_hat) xi)^T x >= b,
where Box = {xi : ||xi||_inf <= 1}.
Reformulate as:
  a_bar^T x - sum_j w_j >= b,
  with w_j >= 0, w_j >= a_hat_j x_j, w_j >= -a_hat_j x_j,
  for each j where a_hat_j > 0.

[Entry 5]
For an objective with polyhedral uncertainty:
maximize min_{delta in Poly} (c_bar + delta)^T x,
where Poly = {delta : D delta <= d, E delta = e}.
Reformulate as:
  maximize t, subject to
  t <= c_bar^T x - d^T lambda - e^T mu,
  D^T lambda + E^T mu = -x, lambda >= 0, mu free.

[Entry 6]
For a constraint with polyhedral uncertainty and max direction:
max_{delta in Poly} (a_bar + delta)^T x <= b,
where Poly = {delta : D delta <= d, E delta = e}.
Reformulate as:
  a_bar^T x + d^T lambda + e^T mu <= b,
  D^T lambda + E^T mu = x, lambda >= 0, mu free.

[Entry 7]
For a constraint with budgeted uncertainty and max direction:
max_{xi in Budgeted} (a_bar + diag(a_hat) xi)^T x <= b,
where Budgeted = {xi : ||xi||_inf <= 1, ||xi||_1 <= Gamma}.
Reformulate as:
  a_bar^T x + Gamma*p + sum_j q_j <= b,
  with p >= 0, q_j >= 0,
  p + q_j >= a_hat_j x_j, p + q_j >= -a_hat_j x_j, for each j.

[Entry 8]
For an objective with polyhedral uncertainty:
minimize max_{delta in Poly} (c_bar + delta)^T x,
where Poly = {delta : M delta <= q, E delta = e}.
Reformulate as:
  minimize c_bar^T x + q^T lambda + e^T mu,
  subject to M^T lambda + E^T mu = x, lambda >= 0, mu free.
```

M.2 ACE's learned memory (representative samples)

ACE accumulates 51 entries organized into "strategies", "formulas/calculations", and "code snippets". The strategy entries are verbose, covering overlapping reformulation scenarios with varying levels of detail and occasional sign-convention inconsistencies. We show three representative entries below.

In total, ACE contains 51 entries, many of which are redundant or mutually contradictory on sign conventions (e.g., the max-min stationarity condition is stated differently in `strat-00088`, `calc-00030`, `calc-00089`, and `calc-00106`). The cross-referencing burden and conflicting signals likely explain why the LLM fails to extract reliable guidance on challenging problem types, contributing to the accuracy drop on the *Hard* dataset.

Listing 2: ACE: three representative strategy entries (abridged).

```
[strat-00019]  helpful=169  harmful=0
For robust optimization problems with multiple uncertain components
(objectives and constraints), adopt a modular approach:
(1) Identify each uncertain component and its uncertainty set type
    (box, budget, polyhedral).
(2) For box uncertainty, check variable bounds: if all uncertain
    variables are nonnegative, replace |x_j| with x_j; if nonpositive,
    replace with -x_j; if mixed, use general absolute values and add
    w_j with -w_j <= x_j <= w_j; for budget, introduce p_j and z with
```

```
    p_j + z >= hat_j w_j; for polyhedral, dualize with lambda.
(3) Use distinct auxiliary variable names per component (e.g., prefix
    'obj_', 'con1_') to avoid accidental coupling.
(4) Include all deterministic constraints and variable bounds unchanged.

[strat-00088]  helpful=44  harmful=3
For systematic dualization of any polyhedral robust component (objective
or constraint), follow this step-by-step Lagrangian approach:
(1) Determine the inner optimization direction: for a <= constraint,
    inner is max; for a >= constraint, inner is min; ...
(2) Write the inner problem as an LP in P delta <= q (convert all
    inequalities including box bounds).
(3) Form the Lagrangian by adding lambda^T(P delta - q) (lambda >= 0)
    and mu^T(E delta - e) (mu unrestricted) ...
(4) Rearrange to isolate delta: Lagrangian = delta^T(x + P^T lambda +
    E^T mu) - lambda^T q - mu^T e ...
... [48 more lines of derivation guidance with sign mnemonic tables] ...

Note: For inner max, stationarity is P^T lambda + E^T mu = x and
penalty is added. For inner min, stationarity is P^T lambda + E^T mu
= -x and penalty is subtracted. Always re-derive the Lagrangian when
uncertain. [Contradicted by strat-00105 which revises this rule.]

[calc-00089]  helpful=50  harmful=1
Correct dualization of polyhedral uncertainty set for all robust
component types:
(1) <= constraint (inner max): a_bar^T x + lambda^T q + mu^T e <= b,
    P^T lambda + E^T mu = x, lambda >= 0, mu unrestricted.
(2) >= constraint (inner min): a_bar^T x - lambda^T q - mu^T e >= b,
    P^T lambda + E^T mu = -x, lambda >= 0, mu unrestricted.
(3) Objective min-max: min c_bar^T x + lambda^T q + mu^T e,
    s.t. P^T lambda + E^T mu = x.
(4) Objective max-min: max c_bar^T x - lambda^T q - mu^T e,
    s.t. P^T lambda + E^T mu = -x.
... [Note: conflicts with calc-00030 and is partially corrected by
calc-00106; sign convention for max-min stationarity is listed
inconsistently across entries.] ...
```

M.3 ReasoningBank's learned memory (representative samples)

ReasoningBank produces one multi-item memory record per training step (80 in total). Each record contains 2–3 narrative-style memory items grounded in the specific training instance, including concrete numerical values from that instance. We show two representative entries below.

While individual ReasoningBank entries are conceptually sound, the 80-entry corpus is heavily redundant: the same LP-duality insight is restated dozens of times with minor phrasing variations tied to specific training instances. With top-1 retrieval, the LLM reads only the retrieved entry. Redundancy may instead limit the diversity or relevance of retrieved guidance.

Listing 3: ReasoningBank: two representative memory items.

```
[epoch1_step1 -- Memory Item 1]
Title: Worst-case direction for box uncertainty in minimization problems
Description: Under box uncertainty and nonneg variables, the worst-case
for minimization over uncertainty is achieved at the extreme point where
each parameter takes the opposite sign of its perturbation coefficient.
Content: For (a_i + b_i xi_i) x_i with xi_i in [-1,1] and x_i >= 0,
the minimum over xi_i occurs at xi_i = -sign(b_i). This rule applies
directly to both robust objectives of the form max min_xi and robust
constraints of the form min_xi (.) >= 0, all with nonnegative variables.

[epoch1_step2 -- Memory Item 1]
Title: Correct dual constraint sign for inner minimization over polyhedral set
Description: When dualizing an inner minimization problem over a
```

```
polyhedral uncertainty set in a max-min objective, the dual constraint
must match the negative of the coefficient vector.
Content: In the reformulation of max_x min_{delta: D delta <= d}
(c_bar + delta)^T x, the inner minimization min delta^T x has dual
max_{pi >= 0} -d^T pi subject to D^T pi = -x. A common error is to
write D^T pi = x, which corresponds to the dual of a maximization over
delta. Always derive the dual from the primal direction.

[epoch1_step2 -- Memory Item 3]
Title: Include all indices in dual variables for budgeted uncertainty
Description: When constructing the robust counterpart for box-budget
uncertainty, explicitly include dual variables for every index to
avoid subtle omissions.
Content: Even if some deviation coefficients hat_a_i are zero, the
dual constraint r_i + s >= hat_a_i x_i is still needed. For zero
deviation, the constraint simplifies to r_i + s >= 0, which can be
satisfied by r_i = 0 and s >= 0. Omitting such indices can lead to
an incomplete set of dual variables and potential errors.
```

M.4 Expert Prompt

The Expert Prompt baseline provides a single fixed prompt containing a set of manually designed reformulation rules that cover the five standard uncertainty set types (ellipsoidal, polyhedral, general p-norm ball, 1-norm ball, ∞-norm ball). For reference, the core rules are as follows.

Listing 4: Expert Prompt: core reformulation rules (from Bertsimas and Margaritis [39]).

```
REFORMULATION RULES

Assume a robust constraint (a + P z)^T x <= b  for all z in Z.

- If Z is the Euclidean ball with radius r (||z||_2 <= r):
    a^T x + r ||P^T x||_2 <= b.

- If Z is a polyhedral set (Z = {z : D z <= d}):
    a^T x + d^T y <= b,  D^T y = P^T x,  y >= 0.

- If Z is a general p-norm ball with radius r (||z||_p <= r):
    a^T x + r ||P^T x||_q <= b,
    where 1/p + 1/q = 1  (q is the dual norm of p).

- If Z is a 1-norm ball with radius r (||z||_1 <= r):
    a^T x + r ||x||_inf <= b.

- If Z is an inf-norm ball with radius r (||z||_inf <= r):
    a^T x + r ||x||_1 <= b.
```

N Generalization across uncertainty structures and problem sources

We construct three datasets to test whether the frozen memory generalizes under changes in uncertainty structure and problem source. They cover non-diagonal perturbation mappings in *General* P_i, shared primitive uncertainty across rows in *Coupled Row*, and independently sourced textbook and published robust models in *External*. The *General* P_i, *Coupled Row*, and *External* datasets each contain 32 instances. Table 11 reports mean $\pm$ sample standard deviation over three test-time runs.

N.1 Controlled non-diagonal perturbation matrices

Beyond the *Random* data-generation setting, every uncertain row in *General* P_i uses a non-diagonal perturbation matrix while preserving nominal sparsity, so nominal zeros outside the selected support remain unperturbed. The modest accuracy gain is consistent with partial transfer. The current memory was learned from the identity/diagonal perturbation setting used in training. It captures family- and

Table 11: Generalization results on *General* P_i, *Coupled Row*, and *External* with the same frozen memory.

Dataset	Method	Accuracy (%)	Output tokens
General P_i	Base LLM	89.6 ± 4.8	8574 ± 49
	AutoREM	90.6 ± 5.4	8379 ± 290
Coupled Row	Base LLM	76.0 ± 4.8	8055 ± 183
	AutoREM	89.6 ± 4.8	6788 ± 328
External	Base LLM	92.7 ± 1.8	5195 ± 236
	AutoREM	97.9 ± 1.8	7347 ± 263

Table 12: Sources and problem families of the *External* dataset.

Source	Instances	Problem families
Ben-Tal et al. [61]	4	Drug production and single-period portfolio.
Li et al. [67]	12	Two-variable robust LP and mixed 0–1 production planning.
Li et al. [68]	4	Multiperiod production planning.
Goerigk and Khosravi [69]	6	Robust selection.
Ben-Tal and Nemirovski [4]	6	NETLIB robust LP.

direction-level dualization patterns, but does not explicitly encode the new $P_i^\top x$ mapping, which must largely be derived at inference.

N.2 Shared primitive uncertainty across two rows

Beyond the *Random* data-generation setting, two uncertain rows in *Coupled Row* share one primitive uncertainty vector and set. AutoREM improves accuracy from 76.0% to 89.6% and reduces output tokens by 15.7%. This is consistent with the familiar row-wise transformations remaining applicable. The new challenge is recognizing and consistently reusing the shared uncertainty set.

N.3 A literature-derived 32-instance challenge set

The *External* dataset contains 32 numerical robust instances from five textbook and published-benchmark sources, independent of the AutoRO-Bench generator. It spans seven problem families: drug production, single-period portfolio, two-variable robust LP, mixed 0–1 production planning, multiperiod production planning, robust selection, and NETLIB robust LP. Table 12 summarizes the sources.

The original sources provide numerical robust instances, model equations, or benchmark files. For each instance, we extract the source-defined objective, constraints, variable domains, uncertainty set, and numerical parameters and convert the original robust model into a self-contained LaTeX input. Thus, *External* is evaluated as an RO reformulation task: the explicit robust model is passed directly to the reformulator and checked by the same fixed coder and solver, without the natural-language formulation stage used for the RO application task. The set contains 20 LPs and 12 MILPs; 14 box, eight budget, seven polyhedral, and three box–polyhedral intersection instances. Decision-variable counts range from 2 to 201, original constraint counts from 1 to 56, and uncertain-row counts from 1 to 14. 16 instances have multiple uncertain rows.

With the existing memory frozen and without adaptation to these instances, AutoREM improves accuracy from 92.7% to 97.9%, a gain of 5.2%, while output tokens increase from 5,195 to 7,347.

O Additional experiments: DeepSeek-V3.2 as base LLM

Experimental setup. This section reports results on the *Random* dataset using DeepSeek-V3.2 (Chat Mode) as the base LLM, following the same evaluation protocol as the main experiments. Both the reformulator and the reflector in AutoREM use this base LLM, and offline adaptation is performed on DeepSeek-V3.2 (Chat Mode) from scratch. Note that the Max Thinking baseline here refers to

DeepSeek-V3.2 with Thinking Mode. The evaluation is conducted on the *Random* dataset, using 32 instances in a single run.

Results and analysis. Table 13 reports the results. The base LLM achieves only 46.9% accuracy, a drop of over 40 percentage points from the DeepSeek-V4-Flash result of 87.5% in Table 2, reflecting V3.2's considerably weaker capability on this algebraically demanding task. Baselines improve upon the base LLM only partially. Max Thinking reaches 59.4%, Expert Prompt 68.8%, and ACE (the strongest baseline) 71.9%, all far below their V4-Flash counterparts, indicating that a weaker base LLM propagates degradation uniformly across the compared baselines.

Table 13: Comparison of methods with DeepSeek-V3.2 as the base LLM.

Methods	Random	
	Accuracy/%↑	Output Tokens↓
Base LLM	46.9	4748
Max Thinking	59.4	13971
Expert Prompt	68.8	3734
ReasoningBank	53.1	<u>2510</u>
ACE	<u>71.9</u>	3184
AutoREM	**96.9**	**2006**

AutoREM achieves 96.9% accuracy, nearly identical to its V4-Flash result of 97.4%, and notably surpasses the V4-Flash base LLM (87.5%) despite operating on a substantially weaker model. The absolute gain over the V3.2 base LLM is 50.0%, far exceeding the ≈10% improvement observed on V4-Flash. The margin over the best baseline (ACE, 71.9%) reaches 25.0%, compared to only 4.7 percentage points on V4-Flash. These results show that the improvement brought by AutoREM is far greater when the base LLM is weaker, suggesting that the offline adaptation procedure is especially effective in filling the larger gap left by a less capable model. AutoREM also achieves the lowest output token count (2006 tokens), far below Max Thinking (13971 tokens), further showing that AutoREM is both more accurate and more efficient than inference-time computation scaling.

Note on DeepSeek-V3.2. DeepSeek-V3.2 is the previous-generation flagship model from DeepSeek. During our experiments, DeepSeek released a new model version and deprecated the V3.2 API, making it impossible to continue experiments on V3.2. We therefore switched to the latest DeepSeek-V4-Flash for our main experiments, which constitutes the primary results reported in the paper. The V3.2 results above are included for reference, reflecting the initial stage of our experimental process.

P Discussion of empirical scope and security

P.1 Empirical scope

Our empirical evaluation focuses on static robust linear models with box, budget, and polyhedral uncertainty, including application instances with natural-language problem description. It does not establish performance on ellipsoidal uncertainty, conic or semidefinite counterparts, or adjustable robust optimization. Extending AutoREM to these settings would require suitable data, reformulation units, solvers, and verifiers. Frozen memory may offer limited benefit on entirely unseen structures.

P.2 Security considerations

The experiments assume trusted generated or expert-curated inputs and do not evaluate adversarial robustness. Prompt injection, memory poisoning, malformed uncertainty sets, and unsafe generated code remain potential threats. Existing parsing and memory-update controls do not establish resistance to these threats or guarantee safe execution. Deployment with untrusted inputs would require additional semantic validation and sandboxed execution, and high-stakes applications would require expert verification of the resulting model.

Q LLM prompts

This section gives the verbatim prompts used for the three LLM agents in AutoREM. Dynamic inputs inserted at runtime are marked with «PLACEHOLDER» notation.

Q.1 Reformulator prompt

```
You are a Reformulator.

Your task is to convert the input robust optimization problem into a tractable
robust counterpart -- a fully explicit deterministic formulation in which all
effects of uncertainty are resolved, with no shorthand operators remaining
(e.g., absolute values |*|, max/min, norms, or supremum expressions).

Use the experience memory to assist: match each minimal independent reformulation
unit (e.g., a single uncertain constraint or uncertain objective term) to the
most relevant experience if one exists, and apply that experience's reformulation
approach; derive the counterpart from scratch for unmatched units.

All specific numeric values from the problem instance must be preserved and
appear explicitly in the final answer.

======================================================================
INPUT
======================================================================

=== PROBLEM ===
<<PROBLEM INPUT>>

=== REFORMULATION EXPERIENCE MEMORY ===
<<EXPERIENCE MEMORY>>

======================================================================
OUTPUT FORMAT
======================================================================

Your ENTIRE response MUST be a single valid JSON object -- nothing else.
  * Do NOT include any text, commentary, or markdown outside the JSON.
  * Do NOT wrap the JSON in code fences.
  * The FIRST character of your response MUST be { and the LAST MUST be }.

"reasoning" MUST cover:
  (a) For each reformulation unit: which experience was matched (or none) and why.
  (b) The derivation process that transforms each unit into its tractable
      robust counterpart.

"matched_experience_ids" MUST be a list of integer experience IDs matched to
reformulation units in (a). Use [] if no unit was matched.

"final_answer" MUST be a complete mathematical formulation of the tractable
robust counterpart in LaTeX within a \begin{align}...\end{align} environment,
with all numeric values from the problem instance appearing explicitly.

Return ONLY a RAW JSON object with the following structure:
{
  "reasoning": <matching analysis and derivation process for each reformulation
    unit>,
  "matched_experience_ids": [<list of matched experience IDs as integers>],
  "final_answer": <the tractable robust counterpart with numeric values in LaTeX>
}
```

Q.2 Coder prompt

```
You are a Coder.

Your task is to write Python code that uses the ‘gurobipy’ library to solve
the given optimization problem (a tractable robust counterpart).

Import gurobipy and create a Model for the given problem. Solve the model, and
then print the optimal objective value using print(model.ObjVal). If the model
is infeasible or unbounded, print "infeasible" or "unbounded" accordingly.

Note: In Gurobi, the default lower bound for variables is 0. Free variables
(unbounded) MUST be declared explicitly with a negative infinity lower bound
(e.g., lb=float(‘-inf’) or lb=-GRB.INFINITY).

================================================================
INPUT
================================================================

=== PROBLEM (ROBUST COUNTERPART) ===
<<ROBUST COUNTERPART INPUT>>

================================================================
OUTPUT FORMAT
================================================================

Your ENTIRE response MUST be a single valid JSON object -- nothing else.
  * Do NOT include any text, commentary, or markdown outside the JSON.
  * Do NOT wrap the JSON in code fences.
  * The FIRST character of your response MUST be { and the LAST MUST be }.

Return ONLY a RAW JSON object with the following structure:
{
  "reasoning": <step-by-step thinking on how to translate the formulation into
    code>,
  "final_answer": <the complete Python code>
}
```

Q.3 Reflector prompt

```
You are a Reflector.

Your task is to reflect on all training samples in the current step (both
correct and incorrect) and update the reformulation experience memory -- a
collection of experiences that each capture how a specific type of robust
optimization structure should be correctly reformulated into its tractable
robust counterpart. Use three operators: ‘add’, ‘update’, and ‘delete’.
Each experience_id MUST appear at most once across all operators.

The usage scores shown in the memory (success / failure counts) record how
often each experience has led to a correct or incorrect final answer when used.

Note: A problem may match multiple experiences (one per reformulation unit).
A failure does not imply every matched experience is wrong -- the root cause
may lie in only one of the matched experiences, while others are correct.

In your ‘reasoning’, analyze the root cause of each failure by examining the
Reformulator’s matched experience IDs, its reasoning, and the environment
feedback for all samples. Use the correct samples as reference to understand
what works, and focus your operator decisions on resolving the failures without
disrupting what is already correct. Consider all samples together and decide on
a unified set of memory operations. Then decide which operators to apply:
  - ‘add’ a new experience if no suitable one exists in memory.
  - ‘update’ an existing experience if its content is incorrect or misleading.
  - ‘delete’ an experience if it is harmful.
```

```
================================================================
INPUT
================================================================

=== EXISTING REFORMULATION EXPERIENCE MEMORY ===
<<EXPERIENCE MEMORY>>

--- SAMPLE 1 (CORRECT / INCORRECT) ---

=== PROBLEM ===
<<PROBLEM INPUT>>

=== REFORMULATOR'S OUTPUT ===
<<REFORMULATOR OUTPUT>>

=== ENVIRONMENT FEEDBACK ===
<<SOLVER FEEDBACK>>

<<... ADDITIONAL SAMPLES ...>>

================================================================
OUTPUT FORMAT
================================================================

Your ENTIRE response MUST be a single valid JSON object -- nothing else.
  * Do NOT include any text, commentary, or markdown outside the JSON.
  * Do NOT wrap the JSON in code fences.
  * The FIRST character of your response MUST be { and the LAST MUST be }.

Fields per operator:
- add:    {"operator": 'add', "experience_id": int, "content": "..."}
- update: {"operator": 'update', "experience_id": int, "content": "..."}
- delete: {"operator": 'delete', "experience_id": int}

Note on 'add' experience_id: assign max(existing experience ids) + 1 for the
first 'add', max(existing experience ids) + 2 for the second, and so on.
Each 'add' MUST use a distinct new ID.

Return ONLY a RAW JSON object with the following structure:
{
  "reasoning": <step-by-step thinking covering: (1) failure diagnosis,
                (2) correct reformulation derivation for the affected unit(s),
                and (3) operator decisions>,
  "final_answer": [
    { ... }
  ]
}
```

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the four contributions: AutoRO-Bench benchmark, AutoREM framework, the offline adaptation procedure, and the experimental validation. All claims are supported by the experimental results.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The Conclusion section explicitly discusses two limitations: (1) offline adaptation relies on solver-verified objective values, which may not be available for all problem classes; (2) the framework currently handles only LP-based robust counterparts and does not extend to more complex RO problem classes.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include formal theoretical results (theorems or lemmas). The mathematical derivations of robust counterparts are provided in the appendix as standard reference material, not as novel theoretical contributions.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The paper reports all hyperparameters (E, S, B, K, $|\mathcal{V}|$), base LLM choices, decoding settings, and evaluation protocols. The data generation pipeline is described in details. Code and datasets will be publicly released.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The paper states that datasets and code will be publicly released to support reproducibility. The automated data generation pipeline further allows independent regeneration of training and evaluation data.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies all experimental settings including dataset descriptions, baseline details, hyperparameter values with justification, evaluation metrics, and the fixed coder LLM. Additional implementation details are provided in the appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All methods are run three independent times and results are reported as mean $\pm$ standard deviation. The source of variability (engineering-level non-determinism in API calls under greedy decoding) is explicitly stated.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All LLMs are accessed via commercial APIs; no local GPU resources are required. The primary compute cost is in API token consumption, which is characterized by the output token counts reported in Section 5.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research involves automated optimization reformulation with no ethical concerns. All datasets are synthetically generated or curated from publicly available benchmarks with proper attribution, and no sensitive personal data is used.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work addresses automated mathematical reformulation for robust optimization, a foundational research topic with no foreseeable direct negative societal impacts. The positive impact is democratizing access to robust optimization by removing the manual reformulation bottleneck.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The released assets (benchmark datasets and optimization framework code) pose no risk of misuse; they are domain-specific tools for mathematical reformulation with no dual-use concerns.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: All external assets are properly cited: RSOME, IndustryOR, OptiBench, and all baseline methods. All LLMs are accessed through official commercial APIs under their respective terms of service.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: AutoRO-Bench is documented in Section 4 with full generation pipeline details. The AutoREM framework is described in Section 4 and the appendix. Both the dataset and code will be released publicly with documentation to support reproducibility.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing. The expert verification of the RO application dataset was conducted by three domain experts as a quality check, not as a formal human subjects study involving data collection from participants.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects; IRB approval is not required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of the proposed method. AutoREM uses LLMs in two roles: as the reformulator that generates robust counterparts and as the reflector that proposes memory edits during offline adaptation. All LLM choices and configurations are described in Section 5.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.