CHANGES IN PROBLEM PRESENTATION CAN BREAK LLM-BASED OPTIMIZATION MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimization modeling is the process of translating natural language descriptions of optimization problems into mathematical formulations and solver code. Large language models (LLMs) are increasingly used to automate this process. In practice, users may describe the same problem in different ways. LLM-based modeling systems must handle these different but equivalent problem descriptions correctly, especially in high-stakes applications such as healthcare and energy. Yet we empirically found that LLMs can generate correct mathematical formulations and solver code for an optimization problem, but fail when the same problem is described differently. This raises a central question: how robust is LLM-based optimization modeling across different natural-language descriptions of the same problem? However, existing optimization modeling benchmarks focus primarily on accuracy but provide limited evidence of robustness to such changes in natural-language descriptions. To address this gap, we introduce RobustOpt, an extensible framework with two complementary benchmarks for requirement understanding through question answering (QA) and end-to-end optimization modeling (Coding) to quantify this robustness. Specifically, we systematically construct and validate alternative problem descriptions that preserve the underlying optimization problem, then compare LLM performance across these descriptions. On each benchmark, we evaluate the robustness of 14 LLM-based systems: seven based on general-purpose LLMs and seven on models specifically fine-tuned for optimization modeling. The results show that even fine-tuned models can fail on 34.19% of new descriptions of problems they initially handle correctly; failures reach 67.40% among general-purpose models. Further error analysis reveals mistakes in specifying decision variables, objectives, and constraints. These findings highlight robustness to problem presentation as a key requirement for trustworthy LLM-based optimization modeling, motivating its explicit integration into model evaluation and training.

1 INTRODUCTION

Optimization modeling translates natural language descriptions of optimization problems into mathematical formulations and solver code (Xiao et al., 2025). Practitioners need domain knowledge to interpret operational requirements, mathematical expertise to choose decision variables and translate rules such as capacity limits and assignment restrictions into constraints (Williams, 2013), and programming skills to implement the model through a solver's interface (Astorga et al., 2025). Large language models (LLMs) are increasingly used to lower these barriers by automating the modeling process. Recent work supports this goal in two complementary ways. One improves how LLMs work through a problem: agents with different roles collaborate (Xiao et al., 2024), solver feedback guides revisions to generated formulations and code (Ahmaditeshnizi et al., 2024), and search explores alternative formulations (Astorga et al., 2025). The other improves LLMs through additional training, or post-training. ORLM (Huang et al., 2025a) learns from worked modeling examples, while LLMOPT (Jiang et al., 2025) combines such examples with feedback on correctness. StepORLM (Zhou et al., 2026) learns from assessments of modeling steps and solver outcomes, while SIRL (Chen et al., 2025) trains LLMs with solver-based rewards to generate correct models.

These advances make the trustworthiness of LLM-based optimization modeling systems a practical concern. In applications such as nurse scheduling (Miller et al., 1976), staffing and rostering (Sandow & Bowie, 2024), and electricity-market operations (Carlson et al., 2012), the resulting de-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

cisions depend on correctly capturing operational requirements. In these settings, accuracy is only one aspect of trustworthiness; systems must also operate reliably under their intended conditions of use (Tabassi, 2023). For language models, robustness concerns maintaining performance under input variations (Liang et al., 2023) and is a recognized dimension of trustworthiness (Huang et al., 2024). For LLM-based optimization modeling, we require mathematical formulations and solver code to remain correct across different descriptions of the same optimization problem.

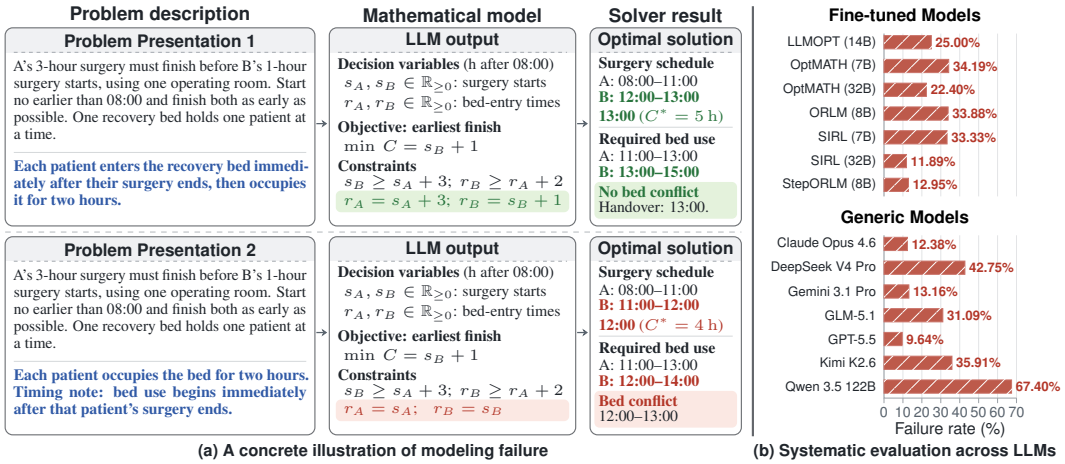

Figure 1: How modeling can fail, and how often it fails after description changes. **(a)** An illustrative surgery-scheduling example shows how misreading an unchanged timing requirement (blue) creates a recovery-bed conflict from 12:00 to 13:00. **(b) A change in description can turn success into failure.** For each LLM, bars show the percentage of equivalent descriptions that fail the numerical check on RobustOpt-Coding, considering only problems it originally passed.

In practice, users may describe the same problem in different ways. For example, resource limits may be listed in a table or described in sentences. A two-hour duration may be written as 120 minutes, and a timing condition may appear in a separate note. Descriptions may also include background information that does not affect the plan. We study such changes in *problem presentation*—how requirements are expressed and organized—while keeping the underlying optimization problem fixed. Prior studies show that LLM reasoning can be disrupted by adding unrelated information (Shi et al., 2023) or changing the order in which the same facts are presented (Chen et al., 2024). Rewording a mathematical problem can also change how often an LLM answers it correctly (Zhou et al., 2024). Our experiments reveal a similar vulnerability: LLMs can generate correct formulations and solver code under one description, yet fail under an equivalent description. Figure 1(a) illustrates how a modeling error can affect surgery scheduling, where operating-room plans must account for postoperative recovery-bed availability (Zhan et al., 2026). In this example, two patients each need the same recovery bed for two hours immediately after surgery. Both descriptions state this requirement, but the second puts the timing detail in a separate note. The mathematical model in the second row mistakenly counts bed use from the start of surgery instead of its end. It therefore schedules B's surgery to finish at 12:00, while A still needs the bed until 13:00. B would have no available recovery bed for the first hour after surgery.

This raises a central question: *how robust is LLM-based optimization modeling across different natural-language descriptions of the same problem?* However, existing benchmarks such as MAMO (Huang et al., 2025b), OptiBench (Yang et al., 2025), OptMATH (Lu et al., 2025), and MIPLIB-NL (Li et al., 2026b) measure accuracy by solving generated formulations or running solver code. They compare numerical outputs, such as objective values, with reference answers, but provide limited evidence on whether the same problems remain correctly solved when their descriptions change. OPT-Engine (Chen et al., 2026) reports stable average accuracy after wording changes, but improvements on some problems can hide failures on previously solved ones. SCHEDBench (Sharma and Sharma, 2026) tests whether schedules remain feasible after rewording or reordering, but its evaluation focuses on combinatorial scheduling. To address these gaps, we introduce RobustOpt, an extensible framework that constructs and validates alternative descriptions of the same problem. We build two

complementary benchmarks within this framework to assess requirement understanding and end-to-end optimization modeling. Our question answering (QA) benchmark, RobustOpt-QA, contains 1,200 multiple-choice question–context presentations constructed from ORQA (Mostajabdaveh et al., 2025) to assess understanding of optimization requirements through questions about objectives, model sets and decision activities, constraints, and model classification. RobustOpt-Coding contains 2,061 input instances for end-to-end optimization modeling, spanning diverse optimization problem classes, including linear, integer, quadratic, conic, and nonlinear programming. It requires LLMs to formulate a complete optimization model and express it as executable solver code; numerical outputs are compared with a reference answer. We compare correctness across descriptions and supplement Coding scores with mathematical checks of decision variables, objectives, and constraints in selected outputs. RobustOpt is extensible: both benchmarks can incorporate new optimization problems and new types of description changes through the same construction and validation protocol.

On each of RobustOpt-QA and RobustOpt-Coding, we evaluate 14 LLM-based systems: seven based on general-purpose LLMs and seven on models fine-tuned for optimization modeling. Our experiments highlight three findings. **First, stable average accuracy can hide failures on previously solved problems.** After rewriting, new successes on some problems can offset failures on problems the model previously solved, leaving average accuracy almost unchanged. We therefore track whether a model continues to solve the same problems correctly across different descriptions (Fig. 1b; Sec. 5.1). **Second, failures depend on the edit and modeling requirement.** Irrelevant background, for example, can make it harder to identify which type of optimization model a problem requires. Which ordering of the same information produces more errors also differs across LLMs (Sec. 5.2). **Third, the failures include errors in mathematical formulation.** Checks of selected generated models reveal mistakes in objectives, the values variables may take, and constraints. These errors change what the model optimizes or which decisions it allows, even though the problem requirements have not changed (Sec. 5.3).

In summary, our contributions are threefold. First, we formalize *semantic robustness* for LLM-based optimization modeling as correctness across different descriptions of the same problem. We specify which description changes preserve the problem and define correctness for QA and Coding (Sec. 3). Second, we introduce RobustOpt, an extensible framework with RobustOpt-QA and RobustOpt-Coding for requirement understanding and end-to-end optimization modeling across linear and nonlinear problems. It constructs and validates alternative descriptions, with paired metrics measuring how often correctness is retained after individual changes and across all tested descriptions (Sec. 4). Third, we combine QA evaluation, solver-based numerical checks, and checks of selected formulations to reveal failures hidden by average accuracy and errors in decision variables, objectives, and constraints (Sec. 5).

2 Related Work

Optimization modeling methods. Existing methods improve optimization modeling through multi-agent collaboration, search, and post-training. Chain-of-Experts (Xiao et al., 2024) and OptiMUS (Ahmaditeshnizi et al., 2024) use multiple LLM agents to formulate problems, generate solver code, and revise outputs using execution feedback. Autoformulation (Astorga et al., 2025) uses Monte Carlo tree search to construct and evaluate candidate formulations. ORLM (Huang et al., 2025a) and OptMATH (Lu et al., 2025) use supervised fine-tuning on modeling examples. SIRL (Chen et al., 2025) uses solver-informed reinforcement learning, while StepORLM (Zhou et al., 2026) guides preference optimization with solver feedback and a learned verifier of the modeling process. EquivaMap (Zhai et al., 2025) and ORGEval (Wang et al., 2025) check formulation equivalence. Our evaluation complements these advances by testing whether a system that succeeds on one description remains correct under equivalent descriptions.

Optimization modeling benchmarks. MAMO (Huang et al., 2025b), OptiBench (Yang et al., 2025), OptMATH (Lu et al., 2025), and MIPLIB-NL (Li et al., 2026b) emphasize solver-based accuracy. ORQA (Mostajabdaveh et al., 2025) instead tests understanding of model components. Our RobustOpt-QA builds on its questions to measure correctness after description changes. OPT-Engine (Chen et al., 2026) reports stable average accuracy under wording changes in linear programming (LP) and mixed-integer programming (MIP) problems, but finds accuracy drops after adding new constraints. We focus on failures under unchanged requirements, testing changes such as separating

related information across notes or tables and tracking whether original successes survive individual changes and all tested descriptions. SCHEDBench (Sharma and Sharma, 2026) uses paired comparisons to test constraint satisfaction in combinatorial scheduling and also reports objective quality. Our RobustOpt-QA and RobustOpt-Coding assess requirement understanding and end-to-end optimization modeling, with Coding covering linear, integer, quadratic, conic, and nonlinear problems.

LLM robustness. HELM (Liang et al., 2023) evaluates language models across tasks such as question answering and summarization, measuring task performance together with robustness to input changes such as typos. TrustLLM (Huang et al., 2024) evaluates LLM trustworthiness across dimensions including safety, fairness, privacy, and robustness, with tests of how models handle noisy inputs. CheckList (Ribeiro et al., 2020) tests whether predictions remain consistent when input changes preserve the meaning. For mathematical reasoning, Stolfo et al. (2023) study how rewording and changes in mathematical information affect model answers. Our focus is whether LLMs still solve the same optimization problem correctly when its description changes. Different formulations are acceptable as long as they represent the same problem. App. A provides extended discussion.

3 Problem Definition and Evaluation Tasks

In this section, we ask whether LLM-based systems remain correct when an optimization problem is reworded. We define equivalent descriptions, introduce requirement-understanding and end-to-end modeling tasks, and define semantic robustness from task correctness.

Natural-language descriptions and optimization problems. Users describe optimization problems by stating what they want to achieve, what resources are available, and what restrictions apply. A modeling system must translate this information into a mathematical model (Ramamonjison et al., 2023). Studies of arithmetic word problems show that different wording can express the same mathematical relationships (Thevenot, 2010). Decision theory expresses a similar principle: describing the same choice differently should not change which option a person prefers (Tversky and Kahneman, 1986). We distinguish a *problem presentation*—a natural-language description and accompanying data—from the intended optimization problem:

$$P_i: \quad \min_{z \in X_i} f_i(z), \qquad X_i = \{z \in D_i : g_i(z) \leq 0,\ h_i(z) = 0\}.$$

Here the decision vector z has domain D_i; f_i is the objective, and g_i, h_i encode inequality and equality constraints componentwise, respectively. Maximizing f_i means minimizing $-f_i$; we record this sign change. We now define when two presentations specify the same problem.

Definition 1 (Equivalent problem presentations)**.** *Let X_i, f_i and X_i', f_i' denote the feasible sets and objectives specified by two complete descriptions and their data, after recorded mappings to common decision variables, units, and objective convention. The presentations are equivalent if and only if $X_i' = X_i$ and $f_i'(z) = f_i(z)$ for every $z \in X_i$.*

Equal optimal values do not establish equivalence: an extra constraint can remove feasible decisions without changing the optimum.

Task 1: Requirement Understanding (QA). When an end-to-end modeling attempt fails, the final result alone does not reveal whether the system misunderstood a requirement or made a formulation or implementation error (Mostajabdaveh et al., 2025). QA directly tests modeling judgments through questions about variables, objectives, constraints, model types, and their relationships. Answering these questions may require connecting information across a description, even though no complete model is requested. Asking the same question under equivalent descriptions tests whether the judgment remains correct without introducing implementation or solver-execution failures. This provides focused evidence about which requirement judgments are sensitive to presentation changes.

Task 2: End-to-End Optimization Modeling (Coding). QA asks focused questions about individual modeling requirements; it does not require the system to assemble them into a complete optimization model. In a complete model, all relevant requirements must be represented together: decision variables must be used consistently in the objective and constraints, and the solver implementation must preserve the formulation. We therefore evaluate end-to-end optimization modeling by asking systems to produce a complete solver program from the description and data.

What counts as a correct output in the two tasks. 1) *QA*. The system must answer the modeling question correctly—for example, by identifying the capacity limit stated in the problem. Its answer

must agree with the reference in meaning. Equivalent descriptions must preserve both the question's meaning and its correct answer. 2) *Coding.* The program must encode the full optimization problem and execute to produce the requested valid result. Different formulations are acceptable (Zhai et al., 2025). After recorded conversion to common variables, units, and objective conventions, they must permit exactly the same feasible decisions as the reference problem and give each the same objective value. App. E.2 covers equivalent formulations, including auxiliary variables. Sec. 4.2 explains the checks, their limitations, and the metrics for tracking correctness across descriptions.

The criteria above assess correctness for one description. Semantic robustness requires correctness across equivalent descriptions of the same problem. We formalize this requirement below.

Definition 2 (Semantic robustness)**.** *Let $\mathcal{P}$ be a specified family of equivalent presentations of problem P_i, including the original presentation. At fixed inference settings, a deterministic system $\mathcal{M}$ is semantically robust on P_i with respect to $\mathcal{P}$ if $\mathcal{M}(p)$ is correct for every $p \in \mathcal{P}$, under the chosen task's correctness criterion.*

4 RobustOpt: Framework and Benchmarks

In this section, we first introduce the RobustOpt framework, then describe the two benchmarks RobustOpt-QA and RobustOpt-Coding, and finally define the corresponding robustness metrics.

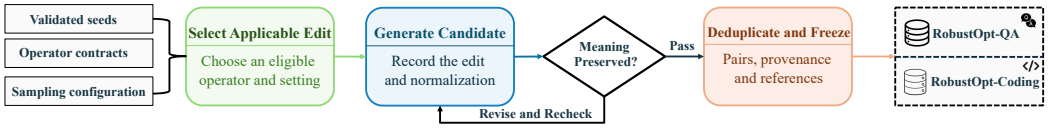

Figure 2: RobustOpt's shared construction pipeline for RobustOpt-QA and RobustOpt-Coding. The diagram highlights the common selection, generation, validation, revision, and admission stages.

RobustOpt: an extensible framework. RobustOpt creates alternative descriptions while preserving the underlying optimization problem (Figure 2). Each run uses three construction inputs: (i) a *validated seed*—an original problem description and data, checked for missing information, ambiguous requirements, and consistency with a reference answer or optimization model; (ii) an *operator contract*, defining an edit's prerequisites, permitted changes, and preservation requirements; and (iii) a *sampling configuration*, specifying the problems, edit settings, and construction order.

An *applicable edit* is a change whose prerequisites hold for the selected problem, such as reordering statements, consistently renaming entities, or converting units of linked quantities. For example, unit conversion requires identifiable quantities whose values and relationships can be converted consistently. The edit produces a *candidate input*; its normalization record maps changed names, units, and objective conventions back to the reference. Automated checks and semantic review assess whether the edited description and attached data preserve the intended requirements. Failed or unresolved candidates are revised and rechecked, or excluded. Accepted variants are deduplicated and fixed before evaluation with their sources, edit histories, and validation evidence, independently of the evaluated systems' outputs. New problems, operators, settings, and combinations follow the same procedure, with combined edits checked together. App. B.1 gives step-by-step construction and validation details. App. C details the QA implementation. App. D details the Coding implementation.

4.1 Benchmark Construction and Statistics

Using RobustOpt, we construct two complementary benchmarks from independently sourced problems to test robustness to equivalent descriptions: RobustOpt-QA for requirement understanding and RobustOpt-Coding for end-to-end optimization modeling (see Sec. 3 for task definitions).

RobustOpt-QA. We extend 200 ORQA questions (Mostajabdaveh et al., 2025), covering 11 question types and 109 contexts, into 1,200 original and perturbed question–context presentations. *(1) Targeted perturbations.* Each question receives five variants: indirect wording of key relations (Indirect), reordered statements that separate related facts (Scattered), entity aliases with separated definitions and uses (Aliased), equivalent objective or constraint expressions using remaining or cumulative

quantities (Recast), and plausible but irrelevant background (Irrelevant). The modeling requirements and correct answer remain unchanged (see App. C for perturbation definitions and construction details). *(2) Matched comparisons.* We use two settings on RobustOpt-QA: the *answer addition setting* appends an incorrect option to the original four, yielding five; the *incorrect-option replacement setting* replaces one incorrect option, retaining four. Both use the same new distractor. Each variant is paired with a matched control: the original context with that variant's question, edited options, and option order (see App. F for paired evaluation protocols). *(3) Quality checks.* Checks verify source–variant pairing, answer keys, and matching perturbed contexts and distractors across protocols (see App. C for construction and validation details).

RobustOpt-Coding. We select 200 problems from MAMO (Huang et al., 2025b), NL4Opt (Ramamonjison et al., 2023), IndustryOR (Huang et al., 2025a), OptiBench (Yang et al., 2025), OptMATH (Lu et al., 2025), and MIPLIB-NL (Li et al., 2026b), spanning linear, integer, quadratic, conic, and nonlinear optimization. *(1) Equivalent descriptions.* We reword or reorder statements, rename entities, convert units, change data formats, or add irrelevant background while keeping the optimization problem unchanged (see App. D.2 for perturbation definitions and implementation rules). *(2) Linking scattered information.* Further edits split related facts across clauses, tables, or reports, so a system must combine them to build the model. For example, output per worker-hour can be inferred from total output, worker count, and time worked (see App. D.3 for worked examples). Compatible edits can also be combined. *(3) Quality checks.* We check the edited text and linked files against the original quantities and requirements, accounting for renamed entities and converted units. Combined edits are rechecked, and accepted inputs undergo human review (see App. D.2 for validation tests and review records).

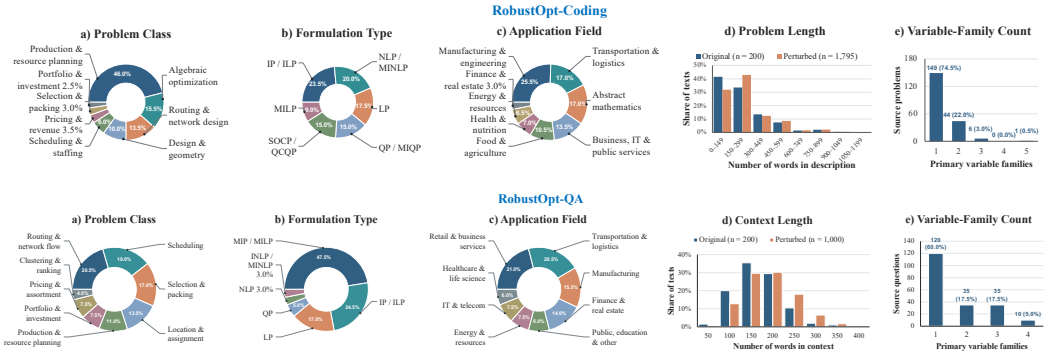

Figure 3: Benchmark profiles for RobustOpt-Coding (top) and RobustOpt-QA (bottom). *(a)* Problem classes, such as routing, scheduling, and resource planning. *(b)* Mathematical formulation types, from linear and integer to quadratic, conic, and nonlinear models. *(c)* Application fields of the optimization tasks. *(d)* Word-count distributions of original and perturbed problem descriptions (Coding) or contexts (QA). *(e)* Number of primary decision-variable families per source problem or question; a family may contain multiple indexed variables.

Benchmark statistics. Figure 3 highlights complementary coverage. Integer and mixed-integer linear formulations account for 72.0% of QA, while quadratic, conic, and nonlinear formulations together make up 50.0% of Coding. Both span routing, scheduling, and resource planning across business, manufacturing, and transportation settings. QA perturbations shift the text-length distribution toward longer contexts. Most items involve one primary decision-variable family: 60.0% of QA questions and 74.5% of Coding problems.

4.2 Robustness Metrics

Average accuracy can hide failures on previously solved problems when new successes offset them. We therefore measure both overall performance and whether earlier successes survive individual or multiple description changes. QA and Coding use separate output checks and conditional populations, following the task targets in Sec. 3.

RobustOpt-QA metrics. We compute four metrics separately for each LLM in the *answer addition setting* (Table 1) and the *incorrect-option replacement setting* (Table 6, Panel B). Each variant is paired with a *matched control*: the original context with the variant's question and edited options (Sec. 4.1). The answer addition setting compares two five-option inputs; the incorrect-option replacement setting compares two inputs with the same modified four-option set. The control differs from the untouched original input. Let n_{pair} count pairs where a selected option can be extracted from each response; correctness is checked separately against the reference answer. *(1) Correctness retention.* n_{control} counts these pairs answered correctly on the matched control, and n_{both} those also correct on the perturbed input. Their ratio $R_{\text{QA}} = n_{\text{both}}/n_{\text{control}}$ measures retention among correct control answers. *(2) Joint correctness.* $J_{\text{QA}} = n_{\text{both}}/n_{\text{pair}}$ is the fraction of all counted pairs answered correctly on both inputs. *(3) All-five retention.* Each question contributes five pairs per protocol, one per context perturbation. n_{group} counts questions with ten extractable answers and five correct matched-control answers; n_{survive} counts those also correct on all five perturbed inputs. Their ratio $S_{5,\text{QA}} = n_{\text{survive}}/n_{\text{group}}$ measures retention across all five changes. *(4) Semantic answer-change rate.* $\text{ACR}_{\text{QA}} = n_{\text{change}}/n_{\text{pair}}$, where n_{change} counts pairs with different selected answer meanings after resolving option labels. Changes include correct-to-incorrect, incorrect-to-correct, and switches between incorrect answers (App. F).

RobustOpt-Coding metrics. We define six measures under a fixed pass criterion; reported numerical passage requires execution and normalized reference-result agreement. For N eligible problems, let $K_i \geq 1$ count distinct alternatives. Indicators c_{i0}, c_{ik} equal 1 for passage under the original and kth alternative, respectively, and 0 otherwise. The first three are *(1) original accuracy* A_0, *(2) alternative accuracy* A_T, and *(3) accuracy drop* Δ:

$$A_0 = \frac{1}{N}\sum_i c_{i0}, \qquad A_T = \frac{1}{N}\sum_i \frac{1}{K_i}\sum_{k=1}^{K_i} c_{ik}, \qquad \Delta = A_0 - A_T. \tag{1}$$

A positive Δ indicates degradation; A_T averages within each problem before averaging across problems, giving every problem equal weight. *(4) Correctness retention.* $R = \frac{\sum_i c_{i0} K_i^{-1} \sum_k c_{ik}}{\sum_i c_{i0}}$ averages the per-problem passing fraction over originally passing problems. *(5) Failure rate.* $F = n_{\text{fail}}/n_{\text{alt}}$ counts failed alternatives (n_{fail}) among all eligible alternatives of originally passing problems (n_{alt}), excluding originals and recoveries. It weights alternatives equally, so F need not equal $1 - R$; higher R and lower F indicate better retention. With numerical scoring, $F_{\text{num}} = F$; Table 2 also gives Pass $= 1 - F_{\text{num}}$ on the same alternatives. *(6) Joint survival.* $S = \frac{1}{N}\sum_i c_{i0} \prod_{k=1}^{K_i} c_{ik}$ is the fraction passing the original and every tested alternative; S/A_0 is its conditional form among originally passing problems ($A_0 > 0$). Survival comparisons use a common test panel. Checks of selected formulations are supplementary diagnostics; the all-check criterion requires execution, numerical agreement, and all supported mathematical checks. Neither certifies full equivalence. Failed checks require review, and unsupported representations remain unresolved (App. G.2).

5 Experimental Results and Analysis

Following the semantic-robustness definition in Sec. 3, we ask three empirical questions: **RQ1.** How reliably is correctness retained across equivalent descriptions, and what failures does average accuracy hide? **RQ2.** Which description changes lead to the largest performance drops, and which aspects of requirement understanding are most affected? **RQ3.** What modeling errors occur after rewriting, and how effective are the tested defenses?

Experimental setup. *(1) Datasets.* We use RobustOpt-QA and RobustOpt-Coding (Sec. 4.1). *(2) Models.* Each task evaluates seven general-purpose and seven optimization-fine-tuned LLMs (see Tables 1 and 2 for model names). *(3) Metrics.* The main tables report correctness retention R_{QA}, joint correctness J_{QA}, all-five retention $S_{5,\text{QA}}$, and semantic answer-change rate ACR_{QA} for QA; for Coding, they report original numerical accuracy A_0 and conditional variant pass and failure rates (Pass and F_{num}). See Sec. 4.2 for definitions. Protocols, prompts, and settings appear in Apps. F–H.

5.1 RQ1: Correctness retention across equivalent descriptions

(1) Correct QA answers can become wrong after rewriting. In the answer addition setting of RobustOpt-QA (adding a fifth, incorrect option), we test whether correct answers survive all five

equivalent rewrites. Each rewrite is paired with the original description, with identical questions and answer options. For example, SIRL (32B) answers 156 of the 200 questions correctly in all five tests using the original description. After rewriting, 139 of these 156 questions still have five correct answers, but 17 of the 156 (10.90%) have at least one wrong answer (Table 1). For ORLM (8B), 24 of the 75 questions answered correctly in all five original-description tests have at least one wrong answer after rewriting (32.00%). We count only questions with identifiable answers in all ten tests (Sec. 4.2). All fourteen LLMs have such failures in both RobustOpt-QA settings (Table 6).

(2) The vulnerability extends to end-to-end optimization modeling. On RobustOpt-Coding, all fourteen LLMs produce outputs that cannot be executed successfully or do not match the reference result under some changed descriptions. Conditional failure rates (failures among eligible variants of problems that passed originally) range from 11.89% to 34.19% for fine-tuned LLMs and from 9.64% to 67.40% for general-purpose LLMs (Table 2). GPT-5.5 has the highest accuracy on original descriptions (70.50%), yet fails these checks on 112 of 1,162 such variants. Success on one description therefore does not ensure success on another. App. G.5 compares these outcomes with repeated unchanged inputs; Finding (4) examines the corresponding QA comparison.

Table 1: RobustOpt-QA robustness in the answer addition setting (adding a fifth, incorrect option). Cells show percentages (numerator/denominator). See Sec. 4.2 for metric definitions and Table 6, Panel B, for results in the incorrect-option replacement setting (swapping one incorrect option, retaining four options). All cells use Round 1 with natural option order; the three-round analysis is reported separately in Table 10.

	Model	$R_{\mathrm{QA}}\uparrow$ (Correctness retention)	$J_{\mathrm{QA}}\uparrow$ (Joint correctness)	$S_{5,\mathrm{QA}}\uparrow$ (All-five retention)	$\mathrm{ACR}_{\mathrm{QA}}$ (Answer-change rate)
Fine-tuned Models	LLMOPT (14B)	93.64 (707/755)	70.70 (707/1000)	83.94 (115/137)	10.40 (104/1000)
	OptMATH (7B)	92.29 (407/441)	40.70 (407/1000)	86.67 (65/75)	9.80 (98/1000)
	OptMATH (32B)	96.85 (739/763)	73.90 (739/1000)	88.89 (128/144)	4.90 (49/1000)
	ORLM (8B)	85.66 (442/516)	45.24 (442/977)	68.00 (51/75)	15.86 (155/977)
	SIRL (7B)	94.68 (569/601)	63.58 (569/895)	91.75 (89/97)	9.39 (84/895)
	SIRL (32B)	96.91 (783/808)	78.30 (783/1000)	89.10 (139/156)	4.40 (44/1000)
	StepORLM (8B)	95.39 (662/694)	66.20 (662/1000)	83.85 (109/130)	6.80 (68/1000)
Generic Models	Claude Opus 4.6	97.25 (883/908)	88.30 (883/1000)	90.34 (159/176)	4.70 (47/1000)
	DeepSeek V4 Pro	96.63 (859/889)	85.90 (859/1000)	94.51 (155/164)	6.70 (67/1000)
	Gemini 3.1 Pro	98.73 (935/947)	93.50 (935/1000)	97.84 (181/185)	2.80 (28/1000)
	GLM-5.1	96.62 (858/888)	85.80 (858/1000)	92.17 (153/166)	6.40 (64/1000)
	GPT-5.5	96.52 (831/861)	83.10 (831/1000)	93.71 (149/159)	7.20 (72/1000)
	Kimi K2.6	97.20 (867/892)	86.70 (867/1000)	97.58 (161/165)	5.70 (57/1000)
	Qwen 3.5 122B	97.11 (841/866)	84.10 (841/1000)	92.59 (150/162)	5.30 (53/1000)

(3) Stable average accuracy can hide failures on previously passing problems. On RobustOpt-Coding, SIRL (32B) illustrates this pattern: its accuracy is 42.00% on original descriptions and 42.25% on variants, with each problem weighted equally. Some variants of previously passing problems fail, subtracting 4.653 percentage points; some variants of previously failing problems pass, adding 4.906 points. Among the 84 problems that pass under their original descriptions, only 50 pass every tested variant. The 673 eligible variants of these 84 problems include 80 failures (Table 14). New successes elsewhere offset these failures in the average.

(4) Comparing description changes against a no-change baseline. We test whether our rewrites produce more answer changes than repeated identical inputs. The RobustOpt-QA repetition experiment contains five answers for each of 200 questions, with the description, question, and four options unchanged. For comparison with rewrites, we use repeated unchanged-description controls with the same questions and options as the rewrites: 29 questions in the answer addition setting and 21 in the incorrect-option replacement setting.

Table 2: RobustOpt-Coding numerical results in generation round 1 (R1). Cells show percentages (numerator/denominator). A_0 uses observed originals; Pass and F_{num} use eligible alternatives of numerically passing originals.

	Model	$A_0\uparrow$ (Original accuracy)	Pass $\uparrow$ (Variant pass rate)	$F_{\mathrm{num}}\downarrow$ (Variant failure rate)
Fine-tuned Models	LLMOPT (14B)	26.50 (53/200)	75.00 (318/424)	25.00 (106/424)
	OptMATH (7B)	14.00 (28/200)	65.81 (154/234)	34.19 (80/234)
	OptMATH (32B)	47.00 (94/200)	77.60 (589/759)	22.40 (170/759)
	ORLM (8B)	36.00 (72/200)	66.12 (404/611)	33.88 (207/611)
	SIRL (7B)	20.50 (41/200)	66.67 (220/330)	33.33 (110/330)
	SIRL (32B)	42.00 (84/200)	88.11 (593/673)	11.89 (80/673)
	StepORLM (8B)	41.50 (83/200)	87.05 (578/664)	12.95 (86/664)
Generic Models	Claude Opus 4.6	48.50 (97/200)	87.63 (701/800)	12.38 (99/800)
	DeepSeek V4 Pro	42.00 (84/200)	57.25 (395/690)	42.75 (295/690)
	Gemini 3.1 Pro	61.81 (123/199)	86.84 (878/1011)	13.16 (133/1011)
	GLM-5.1	42.50 (85/200)	68.91 (472/685)	31.09 (213/685)
	GPT-5.5	70.50 (141/200)	90.36 (1050/1162)	9.64 (112/1162)
	Kimi K2.6	45.00 (90/200)	64.09 (473/738)	35.91 (265/738)
	Qwen 3.5 122B	19.50 (39/200)	32.60 (104/319)	67.40 (215/319)

In the answer addition setting of RobustOpt-QA, rewriting produces more answer changes for six of the seven LLMs in this comparison. For DeepSeek V4 Pro, the rates are 6.55% for identical inputs and 9.08% after rewriting, a difference of 2.53 percentage points with questions

weighted equally. In the incorrect-option replacement setting, three LLMs show more changes, one the same rate, and three fewer (Table 9). The increase occurs for most models on these questions in the answer addition setting, but not uniformly across models or both QA settings. See Apps. F.4 (QA) and G.5 (Coding).

5.2 RQ2: Description changes and affected requirements

(1) Irrelevant details can obscure the required optimization model. Some RobustOpt-QA questions ask which formulation fits a problem, such as linear or mixed-integer programming. With answer options fixed, irrelevant background lowers accuracy across the seven optimization-fine-tuned LLMs by 25.40 percentage points in the answer addition setting and 33.60 in the incorrect-option replacement setting. The seven general-purpose LLMs lose 11.11 and 15.08 points, respectively. Losses vary across edits and questions: scattering facts lowers fine-tuned accuracy by only 1.68 points in the answer addition setting for questions about sets of entities, such as products (App. F.2).

(2) Which rewrite yields more failures depends on the LLM. On RobustOpt-Coding, we compare partial reordering (O2) and complete reversal (O4) of movable description blocks (App. D.2). Both edits use the same originally passing problems for each LLM, where both apply. With complete reversal rather than partial reordering, eight LLMs fail on more problems, two on the same number, and four on fewer. Irrelevant background sharing the problem's topic and roles yields more failures than unrelated background for ten LLMs and fewer for four (Table 15).

5.3 RQ3: Modeling errors and defenses

(1) Errors affect objectives, allowed variable values, and constraints. On RobustOpt-Coding, we examine errors in selected generated mathematical models. For example, SIRL (7B) produces incorrect models for 22 rewritten inputs, although it produces correct models from their original descriptions. At least ten outputs use an incorrect objective, at least two incorrectly specify which values variables can take, and at least two contain incorrect constraints (Table 17). The requirements are unchanged, but these mistakes change what the model optimizes or which decisions it allows. App. G.4 explains which outputs are checked and how the errors are counted.

(2) QA: Check requirements before answering. On RobustOpt-QA, we prompt OptMATH (7B) to review entities, quantities, objectives, and constraints before choosing an answer. We compare this with direct answering on 200 questions and five rewrites each. Accuracy on rewrites increases by 13.80 percentage points in the answer addition setting and 8.37 in the incorrect-option replacement setting, using answers readable under both procedures. Yet some correct answers still become wrong after rewriting. Future work should compare checks with the same limit on response length and test which ones keep correct answers correct after rewriting (App. F.3).

(3) Coding: Use a second candidate when the first cannot run. On RobustOpt-Coding, we generate two candidate models with DeepSeek V4 Pro using the same prompt. If the first cannot execute, we select the second when it passes execution checks, without using reference answers. For problems solved by the first candidate under their original descriptions, failures on 197 rewrites fall from 29 to 14. This recovers 15 lost successes, but successful execution does not show that the model matches the problem. Future work should check variables, objectives, and constraints when selecting candidates, using the same number of candidates on previously unseen problems (App. G.6).

6 Conclusion and limitations

We introduce the RobustOpt framework for constructing and validating equivalent problem descriptions, together with two benchmarks: RobustOpt-QA tests answers to modeling questions, and RobustOpt-Coding tests complete executable optimization models. They test whether LLMs remain correct when a problem's description changes without changing its requirements. Average accuracy can stay almost unchanged as some answers turn wrong and others turn right. Comparisons with repeated inputs help distinguish description-related changes from ordinary response variation. Checks of generated formulations reveal errors in variables, objectives, and constraints. Further evaluation should cover other languages and real application data. A good benchmark score still does not replace checking a generated model against the requirements of its intended application (Autio et al., 2024).

Reproducibility Statement

Benchmark construction and validation are detailed in Apps. B–D; scoring and statistical procedures appear in App. E. Apps. F–H provide evaluation protocols, prompts, inference and solver settings, and the input sets and denominators used for each reported result. The analyses use generated responses and solver outputs; the corresponding appendices specify sampling coverage and the scope of formulation checks. The Coding dataset, evaluation tools, and reproduction instructions are available in our anonymous repository at `https://anonymous.4open.science/r/robustopt-coding-117F/README.md`.

Ethics Statement

This work studies optimization modeling using existing research benchmarks and constructed variants. The surgery-scheduling example in Fig. 1 is synthetic and contains no patient data. We credit the source benchmarks; third-party materials remain subject to their original licenses and redistribution terms. A key risk is overreliance on a correct numerical result despite an incorrect formulation. Use in consequential applications therefore requires domain-expert review and application-specific validation. The scope and limitations of input validation and formulation checks are documented in Apps. C, D.2, and G.4.

AI Use Statement

LLM-based tools, including Codex, were used to polish the manuscript. They also assisted with generating and screening benchmark perturbations and distractors, descriptive dataset annotation, and partial reference-based formulation checks. Validation procedures, the scope of these checks, and unresolved cases are documented in Apps. C–G. LLM judgments alone do not establish semantic or mathematical equivalence. The authors take full responsibility for the manuscript and its conclusions.

References

Sumaya Abdul Rahman, Seckhen Ariel Andrade Cuellar, Ghani Raissov, and Mohammad Raza. VeriSimpl: Robust optimization modeling from natural language using simplification-based verification. *arXiv preprint arXiv:2607.20474*, 2026.

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*, PMLR 235, pp. 577–596, 2024.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using LLMs. In *International Conference on Machine Learning*, PMLR 267, pp. 1864–1886, 2025. `https://proceedings.mlr.press/v267/astorga25a.html`.

Chloe Autio, Reva Schwartz, Jesse Dunietz, Shomik Jain, Martin Stanley, Elham Tabassi, Patrick Hall, and Kamie Roberts. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. *NIST Trustworthy and Responsible AI - 600-1*, National Institute of Standards and Technology, 2024. doi: `10.6028/NIST.AI.600-1`.

Andrew M. Bean, Ryan Othniel Kearns, Angelika Romanou, Franziska Sofia Hafner, Harry Mayne, et al. Measuring what matters: Construct validity in large language model benchmarks. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025.

Dimitris Bertsimas, David B. Brown, and Constantine Caramanis. Theory and applications of robust optimization. *SIAM Review*, 53(3):464–501, 2011. doi: `10.1137/080734510`.

Brian Carlson, Yonghong Chen, Mingguo Hong, Roy Jones, Kevin Larson, Xingwang Ma, Peter Nieuwesteeg, Haili Song, Kimberly Sperry, Matthew Tackett, Doug Taylor, Jie Wan, and Eugene Zak. MISO unlocks billions in savings through the application of operations research for energy and ancillary services markets. *Interfaces*, 42(1):58–73, 2012. doi: `10.1287/inte.1110.0601`.

Xinyun Chen, Ryan Andrew Chi, Xuezhi Wang, and Denny Zhou. Premise order matters in reasoning with large language models. In *International Conference on Machine Learning*, PMLR 235, pp. 6596–6620, 2024.

Yitian Chen, Jingfan Xia, Siyu 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.

Yitian Chen, Cheng Cheng, Yinan Sun, Zi Ling, and Dongdong Ge. OPT-Engine: Benchmarking the limits of LLMs in optimization modeling via complexity scaling. In *International Conference on Machine Learning*, 2026. https://icml.cc/virtual/2026/poster/61027.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, 36, 2023. doi: 10.52202/075280-3081.

Yue Huang, Lichao Sun, Haoran Wang, Siyuan Wu, Qihui Zhang, et al. Position: TrustLLM: Trustworthiness in large language models. In *International Conference on Machine Learning*, PMLR 235, pp. 20166–20270, 2024. https://proceedings.mlr.press/v235/huang24x.html.

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

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL*, pp. 2678–2710, 2025b. doi: 10.18653/v1/2025.findings-naacl.146.

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.

Zhong Li, Qi Huang, Yuxuan Zhu, Mohammad Mohammadi Amiri, Niki van Stein, Thomas Bäck, Matthijs van Leeuwen, Zaiwen Wen, and Lincen Yang. MM-OptBench: A solver-grounded benchmark for multimodal optimization modeling. *arXiv preprint arXiv:2605.12154*, 2026a. https://arxiv.org/abs/2605.12154.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *International Conference on Machine Learning*, 2026b. https://icml.cc/virtual/2026/poster/66779.

Zhong Li, Zihan Guo, Xiaohan Lu, Juntao Wang, Jie Song, Chao Shen, Jiageng Wu, and Mingyang Sun. OptArgus: A multi-agent system to detect hallucinations in LLM-based optimization modeling. *arXiv preprint arXiv:2605.11738*, 2026c. https://arxiv.org/abs/2605.11738.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. https://openreview.net/forum?id=iO4LZibEqW.

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*, 38, pp. 133786–133854, 2025. doi: 10.52202/085713-4027.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-Verifier: Unleashing the power of LLMs for optimization modeling via dual-side verification. In *International Conference on Machine Learning*, 2026. https://icml.cc/virtual/2026/poster/64663.

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

Hongliang Lu, Zhong Li, Yuxuan Chen, Yuan Lan, Fan Zhang, and Zaiwen Wen. PEARL: Solver-in-the-loop interactive optimization modeling from natural language. *arXiv preprint arXiv:2607.18256*, 2026. https://arxiv.org/abs/2607.18256.

Holmes E. Miller, William P. Pierskalla, and Gustave J. Rath. Nurse scheduling using mathematical programming. *Operations Research*, 24(5):857–870, 1976. doi: 10.1287/opre.24.5.857.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. GSM-Symbolic: Understanding the limitations of mathematical reasoning in large language models. In *International Conference on Learning Representations*, 2025.

Mahdi Mostajabdaveh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating LLM reasoning in the operations research domain with ORQA. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 39, pp. 24902–24910, 2025. doi: 10.1609/aaai.v39i23.34673.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 5687–5711, 2023. doi: 10.18653/v1/2023.findings-emnlp.378.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, PMLR 220, pp. 189–203, 2023.

Paulius Rauba, Qiyao Wei, and Mihaela van der Schaar. Statistical hypothesis testing for auditing robustness in language models. In *International Conference on Machine Learning*, PMLR 267, pp. 51297–51313, 2025.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Annual Meeting of the Association for Computational Linguistics*, pp. 4902–4912, 2020.

Brienne M. Sandow and Danielle Bowie. A dynamic staffing & scheduling solution: The build and implementation of a logistics engine to optimize nurse schedules and rosters. *Nurse Leader*, 22(1):21–27, 2024. doi: 10.1016/j.mnl.2023.10.002.

Shrenil Shaun Sharma and Avi Sharma. SCHEDBench: A benchmark for evaluating LLM constraint faithfulness in natural-language combinatorial scheduling. *arXiv preprint arXiv:2608.00991*, version 2, 2026.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, PMLR 202, pp. 31210–31227, 2023.

Alessandro Stolfo, Zhijing Jin, Kumar Shridhar, Bernhard Schölkopf, and Mrinmaya Sachan. A causal framework to quantify the robustness of mathematical reasoning with language models. In *Annual Meeting of the Association for Computational Linguistics(Volume 1: Long Papers)*, 2023.

Elham Tabassi. Artificial Intelligence Risk Management Framework (AI RMF 1.0). *NIST AI 100-1*, National Institute of Standards and Technology, 2023. doi: 10.6028/NIST.AI.100-1.

Catherine Thevenot. Arithmetic word problem solving: Evidence for the construction of a mental model. *Acta Psychologica*, 133(1):90–95, 2010. doi: 10.1016/j.actpsy.2009.10.004.

Amos Tversky and Daniel Kahneman. Rational choice and the framing of decisions. *The Journal of Business*, 59(4, Part 2):S251–S278, 1986. doi: 10.1086/296365.

Boxin Wang, Chejian Xu, Shuohang Wang, Zhe Gan, Yu Cheng, Jianfeng Gao, Ahmed Awadallah, and Bo Li. Adversarial GLUE: A multi-task benchmark for robustness evaluation of language models. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1, 2021.

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Yizhou Han, Yufeng Lin, Zhihang Lin, Angyang Gu, Xinglin Hu, Ruoyu Sun, and Tian Ding. ORGEval: Graph-theoretic evaluation of LLMs in optimization modeling. *arXiv preprint arXiv:2510.27610*, 2025.

Yilin Wang, Heng Zhou, Dongxing Mao, Linjie Li, Jingru Tan, Haochen Han, Zhengyuan Yang, Alex Jinpeng Wang, and Min Li. OR-PRM: A process reward model for algorithmic problem in operations research. In *International Conference on Learning Representations*, 2026.

H. Paul Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 5th edition, 2013.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan 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. https://proceedings.iclr.cc/paper_files/paper/2024/hash/d45ee77826332c100a1e15f7765b99ff-Abstract-Conference.html.

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, Qingcan Kang, Jiahui Duan, Tao Zhong, Mingxuan Yuan, Jia Zeng, Yuan Wang, Gang Chen, and Dongxiang Zhang. A survey of optimization modeling meets LLMs: Progress and future directions. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, Survey Track, pp. 10742–10750, 2025. doi: 10.24963/ijcai.2025/1192.

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*, 40(40), pp. 34052–34060, 2026. doi: 10.1609/aaai.v40i40.40699.

Wantong Xie, Yi-Xiang Hu, Jieyang Xu, Feng Wu, and Xiangyang Li. MURKA: Multi-reward reinforcement learning with knowledge alignment for optimization tasks. In *Advances in Neural Information Processing Systems*, 38, pp. 39196–39223, 2025. doi: 10.52202/085713-1171.

Beinuo Yang, Qishen Zhou, Junyi Li, Chenxing Su, Panagiotis Angeloudis, and Simon Hu. OR-Thought: Benchmarking and automating logistics optimization modeling via structured LLM reasoning. *Artificial Intelligence for Transportation*, 6:100059, 2026. doi: 10.1016/j.ait.2026.100059.

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.

Yang Zhan, Siqian Shen, Guohua Wan, and Zheng Zhang. Optimizing surgery scheduling under post-anesthesia care unit capacity constraints and random services durations. *Computers & Operations Research*, 188:107368, 2026. doi: 10.1016/j.cor.2025.107368.

Haotian Zhai, Connor Lawless, Ellen Vitercik, and Leqi Liu. EquivaMap: Leveraging LLMs for automatic equivalence checking of optimization formulations. In *International Conference on Machine Learning*, PMLR 267, pp. 74288–74305, 2025.

Yue Zhou, Yada Zhu, Diego Antognini, Yoon Kim, and Yang Zhang. Paraphrase and solve: Exploring and exploiting the impact of surface form on mathematical reasoning in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 2793–2804, 2024. doi: 10.18653/v1/2024.naacl-long.153.

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.

Kaijie Zhu, Qinlin Zhao, Hao Chen, Jindong Wang, and Xing Xie. PromptBench: A unified library for evaluation of large language models. *Journal of Machine Learning Research*, 25(254):1–22, 2024.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. DynaMath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. In *International Conference on Learning Representations*, 2025.

Appendix

Table of Contents

A Extended Related Work
- A.1 Optimization modeling methods
- A.2 Optimization modeling benchmarks
- A.3 LLM robustness

B The RobustOpt Framework
- B.1 Construction and validation workflow
- B.2 Source records and duplicate handling
- B.3 Validation scope and benchmark extensions

C RobustOpt-QA: Construction and Validation
- C.1 Question sources and types
- C.2 Context perturbations and input validation
- C.3 A worked example of context perturbation

D RobustOpt-Coding: Construction and Validation
- D.1 Problem sources and formulation types
- D.2 Presentation changes and input validation
- D.3 Worked examples of input validation
- D.4 Construction records and scoring eligibility

E Shared Evaluation Conventions
- E.1 Record counts and benchmark profiles
- E.2 Equivalent optimization formulations
- E.3 Stochastic robustness and repeated evaluation
- E.4 Paired comparisons and uncertainty

F RobustOpt-QA: Experimental Protocol and Results
- F.1 RobustOpt-QA settings and robustness results
- F.2 QA accuracy by perturbation and question type
- F.3 Evaluating structured self-checking as a defense
- F.4 Repeated inputs as a QA comparison baseline
- F.5 QA prompt templates and response formats

G RobustOpt-Coding: Experimental Protocol and Results
- G.1 Coding cohorts and complete results
- G.2 Output checks and recorded execution settings
- G.3 Accuracy across three recorded rounds
- G.4 Checking variables, objectives, and constraints
- G.5 Variability without description changes
- G.6 Selecting a second model after execution failure
- G.7 Representative Coding prompts

H Shared Reproducibility and Controls
- H.1 System identification and reproducible execution
- H.2 Controls and ablations

I Evaluation Coverage and Development Exposure

A Extended Related Work

This appendix expands the three directions in Sec. 2, with particular attention to how existing methods and benchmarks relate to correctness across descriptions of a fixed optimization problem.

A.1 Optimization modeling methods

Existing approaches use multi-agent collaboration, search, and post-training to improve optimization modeling. Chain-of-Experts (Xiao et al., 2024) and OptiMUS (Ahmaditeshnizi et al., 2024) coordinate LLM agents for formulation, programming, and revision using execution feedback. Autoformulation (Astorga et al., 2025) uses Monte Carlo tree search over formulation components, while OptiTree (Liu et al., 2025) searches a hierarchy of simpler subproblems and combines their modeling guidance. ORThought (Yang et al., 2026) uses a structured two-agent reasoning workflow.

ReSocratic (Yang et al., 2025) and OptMATH (Lu et al., 2025) synthesize training examples for optimization modeling. ORLM (Huang et al., 2025a) uses supervised fine-tuning on modeling demonstrations. LLMOPT (Jiang et al., 2025) also begins with supervised fine-tuning, then applies Kahneman–Tversky Optimization (KTO) using expert-labeled correctness feedback. StepORLM (Zhou et al., 2026) jointly improves a policy and a generative process reward model after supervised fine-tuning; solver outcomes and assessments of intermediate steps determine the weights of preference pairs used to train the policy. SIRL (Chen et al., 2025) trains with rewards derived from solver outcomes and statistics of the generated mathematical model. MURKA (Xie et al., 2025) trains its extraction agent with reinforcement learning, distills its solver agent from a stronger teacher, and checks the resulting solutions through a separate agent. OR-PRM (Wang et al., 2026) studies process reward modeling for operations research. PEARL (Lu et al., 2026) learns to alternate model construction with Python execution and solver feedback, revising formulations and code over multiple turns. DeepOR (Xiao et al., 2026) combines synthesized reasoning demonstrations, supervised fine-tuning, and solver-informed reinforcement learning.

Output verification addresses a related challenge: different formulations can represent the same problem. EquivaMap (Zhai et al., 2025) uses LLM-proposed mappings between decision variables to check formulation equivalence. ORGEval (Wang et al., 2025) uses graph representations to assess structural equivalence of linear and mixed-integer linear programs. VeriSimpl (Abdul Rahman et al., 2026) checks candidate formulations through solver-generated diagnostic queries, and Opt-Verifier (Liu et al., 2026) combines structural and solution-based verification. OptArgus (Li et al., 2026c) detects and localizes inconsistencies across descriptions, formulations, and solver implementations through specialist agents. We similarly allow equivalent formulations, including renamed or auxiliary variables, and distinguish confirmed errors, verified equivalence, and unresolved cases.

A.2 Optimization modeling benchmarks

Existing benchmarks broaden the problems and capabilities covered by evaluation. NL4Opt (Ramamonjison et al., 2023) studies conversion from natural-language descriptions to formulations. NLP4LP (Ahmaditeshnizi et al., 2024), ComplexOR (Xiao et al., 2024), and IndustryOR (Huang et al., 2025a) provide additional text-to-modeling collections. MAMO (Huang et al., 2025b) evaluates mathematical modeling across several tasks, and OptiBench (Yang et al., 2025) evaluates end-to-end optimization modeling through solver-based answers. OptMATH (Lu et al., 2025) provides a scalable framework for synthesizing optimization-modeling data. MIPLIB-NL (Li et al., 2026b) constructs natural-language specifications from industrial mixed-integer programs while preserving their mathematical structure. Bench4Opt (Wang et al., 2025) pairs descriptions with formulations for graph-based evaluation, while LogiOR (Yang et al., 2026) focuses on logistics. MM-OptBench (Li et al., 2026a) evaluates modeling from textual and visual specifications, deriving inputs, canonical formulations, and reference solutions from a common solver-verified instance. ORQA (Mostajabdaveh et al., 2025) tests reasoning about model components, their properties, and their relationships through expert-authored multiple-choice questions. It motivates a complementary evaluation interface: testing understanding of problem requirements without requiring a complete executable model. RobustOpt-QA builds on ORQA contexts and questions, whereas RobustOpt-Coding evaluates end-to-end optimization modeling through generated solver programs.

OPT-Engine (Chen et al., 2026, Secs. 3.1 and 5.1) spans ten LP/MIP problem classes and reports stable average solution accuracy across three levels of lexical and syntactic complexity, preserving numerical instances and constraints. Its larger diagnostic accuracy drops follow the addition of new constraints, which can change the feasible set and optimal solution (Chen et al., 2026, Sec. 5.1). This tests the integration of new requirements; our invariance tests ask whether failures arise when requirements remain unchanged.

SCHEDBench (Sharma and Sharma, 2026, Secs. 3.2, 4.3, and 4.4) studies combinatorial scheduling under paraphrasing, thematic renaming, and reordering, with paired comparisons and rendering-seed controls. It reports objective quality for feasible schedules, while its robustness analysis centers on feasibility, constraint violations, and assignment coverage. Exploratory solver-code experiments retain the scheduling scope (Sharma and Sharma, 2026, Table 12). Our evaluation complements these two studies in three ways.

First, our Coding operators expand the construction coverage to rules separated from their scope or timing, parameters linked across tables, coefficients recovered from measurement or accounting records, and irrelevant operational information (Table 5). These changes test the recovery of unchanged requirements from distributed information.

Second, RobustOpt-Coding spans linear, integer, quadratic, conic, and nonlinear problems (Table 4), extending beyond OPT-Engine's LP/MIP inventory and SCHEDBench's scheduling scope. RobustOpt-QA tests understanding of variables, objectives, and constraints using a question set separate from the Coding problems. Checks of selected formulations supplement numerical scores without providing comprehensive verification (App. G.4).

Third, our *conditional failure and all-variant retention* metrics track whether originally correct outputs remain correct after individual changes and across all tested descriptions (Sec. 4.2). OPT-Engine reports average accuracy across language levels (Chen et al., 2026, Sec. 5.1). SCHEDBench already avoids cancellation using absolute per-instance changes (Sharma and Sharma, 2026, Sec. 4.4); our metrics complement this analysis by conditioning on prior correctness.

A.3 LLM ROBUSTNESS

NIST's AI Risk Management Framework (Tabassi, 2023) places validity and reliability at the foundation of trustworthy AI. HELM (Liang et al., 2023) evaluates robustness alongside other language-model qualities, while TrustLLM (Huang et al., 2024) includes it within a broader account of LLM trustworthiness. For optimization-based decision support, we study a specific reliability requirement: the generated formulation and implementation should preserve the decisions, objective, and constraints intended by the user across equivalent descriptions. This target addresses one part of trustworthiness; it does not establish other properties such as fairness or security.

Behavioral testing makes this requirement concrete. CheckList (Ribeiro et al., 2020) constructs tests with explicit expectations about which input changes should preserve behavior and which should change it. Adversarial evaluation is also studied in AdvGLUE (Wang et al., 2021) and supported by PromptBench (Zhu et al., 2024). Adding unrelated information can disrupt LLM reasoning (Shi et al., 2023), as can changing the order in which the same facts are stated (Chen et al., 2024). Rewording mathematical questions can also alter answer correctness (Zhou et al., 2024). GSM-Symbolic (Mirzadeh et al., 2025) varies names and numerical values within symbolic question templates, while DynaMath (Zou et al., 2025) generates visual and textual variants from programmed seed questions. Such related-instance families may change the correct answer. Our Coding invariance tests instead preserve the complete decision problem after declared normalization, such as converting all linked quantities to a common unit. The causal robustness framework of Stolfo et al. (2023) likewise distinguishes invariance from appropriate sensitivity to changed information. For optimization, matching a scalar answer is insufficient to establish equivalence: different feasible sets can share the same optimal value.

The interpretation of robustness scores also depends on evaluation design. Studies of compositional reasoning report failures on multi-hop questions despite correct answers to their subproblems (Press et al., 2023) and errors propagated from intermediate computations (Dziri et al., 2023). These findings concern other task settings; our QA and Coding benchmarks use independent problem sets and do not measure a within-problem compositionality gap. Bean et al. (2025) examine whether benchmark tasks and metrics measure their intended constructs, while Rauba et al. (2025) develop hypothesis tests for perturbation effects on stochastic LLM outputs. Finally, classical robust optimization (Bertsimas et al., 2011) chooses decisions under uncertainty in problem parameters. Our problem is different: the underlying optimization problem stays fixed while its description changes. We assess the reliability of the modeling system, rather than the performance of a decision under uncertain demand, cost, or capacity.

B THE ROBUSTOPT FRAMEWORK

This appendix specifies the workflow for constructing and validating alternative descriptions (Fig. 2), tracking their sources, and extending the benchmarks. Apps. C and D document the checks and validation coverage for the two benchmarks.

B.1 CONSTRUCTION AND VALIDATION WORKFLOW

Prepare the source inputs and construction plan. A construction seed contains the original problem description and data, plus the question and options for QA. Record the reference answer or formulation and supporting evidence separately from the model-visible input. Check completeness and ambiguity, verify domains, index sets, units, and entity references, and resolve inconsistencies with the task reference. Each operator contract specifies prerequisites, permitted edits, and preservation requirements. The sampling configuration fixes the seed pool, edit settings, and construction order before evaluation.

(1) Select an applicable edit. Choose an edit whose prerequisites hold: movable statements for reordering, identifiable entities for renaming, or quantities with an exact unit mapping. Specify the ordering, replacement names, new units, or insertion position. Applicability permits an attempt; preservation must be checked on the resulting input.

(2) Generate a candidate and record the mappings. Edit the description and any affected attachments, recording how changed names, units, or objective conventions map back to the reference. For example, expressing time in minutes changes a two-hour capacity to 120 minutes; linked durations and coefficients are converted according to their units. Renaming similarly records which original entity each new name denotes.

(3) Check preservation before admission. Apply the operator-specific checks, such as recovering quantities or verifying entity bindings. Review the complete input for preservation of feasible decisions and the objective value at every feasible decision after normalization (Definition 1). QA must also preserve the question and correct answer in meaning; added distractors must remain incorrect. Revise and recheck failed candidates, or exclude them with recorded reasons. Unresolved candidates remain outside invariance evaluation. Admission must be independent of evaluated models' outputs.

(4) Deduplicate and fix the evaluation inputs. Deduplicate complete inputs within each seed, retaining all operator memberships. Fix the originals and accepted variants together with their source links, edit settings, normalization records, and validation evidence. Construction records may include duplicates and controls; scoring eligibility is specified separately (Apps. F and G). Keep reference answers, formulations, and supporting code, solutions, and certificates separate from the inputs given to evaluated systems.

B.2 SOURCE RECORDS AND DUPLICATE HANDLING

To make each variant traceable, record its source dataset and version, split where applicable, original identifier, text and file hashes, task reference, and any corrections to the source input. For Coding, link the reference formulation to supporting solver records. The transformation record should specify the operator and version, settings, edited spans, normalization mappings, applicability decision, and review outcome. These are the framework's record requirements; Apps. C and D document the benchmark-specific implementations.

Identify duplicates within each seed using the complete model-visible input, including attachments and, for QA, the question and ordered options. Matching prose alone is insufficient when files or task fields differ. Use input hashes to group identical candidates; count each distinct alternative once in pooled invariance metrics while retaining all operator labels. Different QA questions sharing a context remain distinct source instances.

Copies of the original are not additional perturbations. Keep original baselines, matched controls, and repeated generations in their protocol-defined roles. Changed-target controls require revised references and are excluded from invariance metrics. Construction records may include duplicate

positions and controls, so their record or text-based counts (App. E.1) need not equal the eligible input counts used for evaluation (Apps. F and G).

B.3 Validation scope and benchmark extensions

Input validation checks whether an edit preserves the problem. Under Definition 1, the normalized feasible decisions and objective value at every feasible decision must remain unchanged; QA also preserves the question and correct-answer meaning. The checks in Apps. C and D provide evidence within their documented scope. Matching optima, LLM judgments alone, or observed accuracy drops cannot establish preservation. Generated outputs require separate checks (Sec. 4.2).

Scoring requires its own reference evidence. For Coding, record the reference formulation, supporting solver results, termination status, feasibility checks, and any optimality certificates. Specify the solver and scoring configurations before evaluation. A local nonlinear optimum alone does not establish global optimality. Human approval of inputs does not supply missing reference evidence; record unresolved cases and exclude them from comparisons their references cannot support. App. G documents evaluated panels and reference coverage; App. G.2 gives the output checks.

Extend the framework through new source problems, transformation settings, or operators. Apply the same admission checks to each addition and recheck combined edits together, even when their constituents have passed separately. Keep the source inputs and transformation records for each benchmark version, and report attempted candidates, accepted distinct variants, and coverage by source problem and transformation type. App. I summarizes evaluation coverage and development exposure.

C RobustOpt-QA: Construction and Validation

This appendix describes the source questions, context perturbations, and input checks used to construct RobustOpt-QA. Experimental protocols and results appear in App. F.

C.1 Question sources and types

RobustOpt-QA contains 200 questions from ORQA (Mostajabdaveh et al., 2025), drawn from 109 contexts. Each source item provides a problem description, a modeling question, four candidate answers, and a reference answer. The 11 question types cover objectives, decision activities and sets, constraints, and problem or model types. Different questions about the same context remain separate benchmark items.

C.2 Context perturbations and input validation

For each source question, we vary the context in five ways (Table 3), producing 1,000 variants and 1,200 original and perturbed question–context presentations. Each edit must preserve the problem under Definition 1, as well as the question and the meaning of its correct answer.

Constructing paired inputs. The same RobustOpt-QA context variants support two answer-option settings (Fig. 4). The answer addition setting appends a new distractor to the original four options; the incorrect-option replacement setting substitutes it for one original incorrect option. The new distractor must be incorrect under both contexts. Each variant is paired with a control using the original context and exactly the same question and options. The pair therefore holds the answer choices fixed while changing the context.

Input consistency checks. The input consistency checks confirm that every source question has all five variants and that both protocols use the same source questions in the same order. Matching variants by identifier verifies their source pairing, contexts, questions, correct-option text, and new distractors. The check for the incorrect-option replacement setting also confirms that exactly one original incorrect option changes. These checks establish consistency of the inputs and their pairing; preserving an answer key alone does not establish semantic equivalence of the contexts.

Table 3: Five context transformations used to construct RobustOpt-QA. The last column states what each edit must preserve, in addition to the shared question and correct-answer meaning.

Type	Context change	Preservation requirement
Indirect	Express key relations indirectly, using references or splitting statements within the original narrative.	Preserve the facts, quantities, and relations conveyed by the original statements.
Scattered	Reorder statements and separate related definitions, conditions, and objective information.	Retain all relevant facts and their dependencies; do not omit or summarize away requirements.
Aliased	Introduce alternative names for entities, sets, or quantities, and separate their definitions from later uses.	Maintain a consistent mapping to the original entities or quantities and their roles.
Recast	Express a local objective or constraint through remainders, cumulative quantities, slack, or complementary expressions.	Preserve the mathematical relation when the new expression is interpreted in the original variables and units.
Irrelevant	Add plausible background, such as retrospective statistics or ancillary activities.	Leave the optimization requirements unchanged; the added information must impose no new requirement on the task.

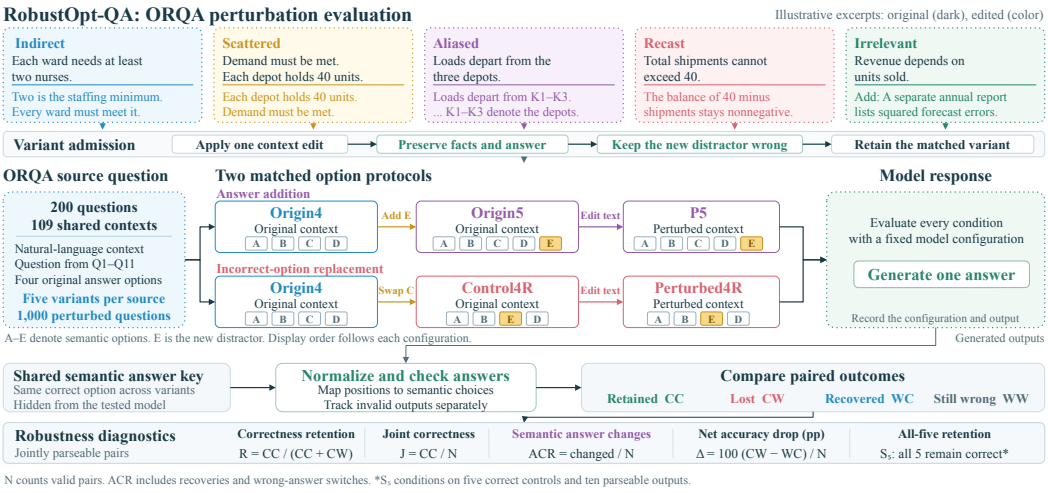

Figure 4: Constructing paired inputs for RobustOpt-QA. The top row illustrates the five context transformations. In the middle row, Origin5 and P5 share five options in the answer addition setting; Control4R and Perturbed4R share four options in the incorrect-option replacement setting. Within each pair, the question and options are fixed and only the context changes. App. F details the subsequent evaluation and response comparisons.

C.3 A WORKED EXAMPLE OF CONTEXT PERTURBATION

In the roommate-matching example (Fig. 5), the question asks which entities form a set in the model. The answer refers to the students being matched, regardless of how their preferences and matching requirements are described. In the Indirect variant, one distractor is replaced by the numerical count of students: this count is a parameter, not the set itself. The full inputs show this distinction alongside the other four context edits.

D ROBUSTOPT-CODING: CONSTRUCTION AND VALIDATION

This appendix describes the source problems, presentation changes, and input-validation procedures used to construct RobustOpt-Coding. Figures 6 and 7 illustrate original and modified inputs. Evaluation protocols and output checks appear in App. G.

Roommate matching

ORQA context 121 | source 260 | Q5

Shared question. Which of the following defines set(s) in the optimization model of this problem?

Origin4

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. The ranking is showcased in a matrix, with each row corresponding to an individual student, and each column indicating their preference for others. On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if Anna and Bob are roommates, Anna shouldn't prefer Charlie over Bob if Charlie himself prefers Anna over his current roommate, Dana. If you were merely to consider personal requests or random allocations, you might end up with disgruntled students, leading to multiple room change requests, conflicts, and an unsettling atmosphere in the dormitories.

Options Reference: B
A. Ranking
B. Roomates
C. Dormitories
D. Rooms

Indirect: Semantic paraphrasing

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. The ranking is showcased in a **matrix. Its rows correspond to individual students; a** row **belongs** to **the student whose preferences it reports.** On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. **The columns in such a row concern the other students for whom the row's owner supplied preferences.** Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if Anna and Bob are roommates, Anna shouldn't prefer Charlie over Bob if Charlie himself prefers Anna over his current roommate, Dana. **A column's entry in that row gives that owner's preference for the other student concerned.** If you were merely to consider personal requests or random allocations, you might end up with disgruntled students, leading to multiple room change requests, conflicts, and an unsettling atmosphere in the dormitories.

Options Reference: B
A. **The numerical count of freshmen in the hall**
B. Roomates
C. Dormitories
D. Rooms

Scattered: Information reordering

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. **You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if Anna and Bob are roommates, Anna shouldn't prefer Charlie over Bob if Charlie himself prefers Anna over his current roommate, Dana.** One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. The ranking is showcased in a matrix, with each row corresponding to an individual student, and each column indicating their preference for others. On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. If you were merely to consider personal requests or random allocations, you might end up with disgruntled students, leading to multiple room change requests, conflicts, and an unsettling atmosphere in the dormitories.

Options Reference: B
A. Ranking
B. Roomates
C. **The numerical count of freshmen in the hall**
D. Rooms

Aliased: Entity renaming

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. The ranking is showcased in a matrix, with each row corresponding to an individual student, and each column indicating their preference for others. On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if **R8** and Bob are roommates, **R8** shouldn't prefer **K6** over Bob if **K6** himself prefers **R8** over his current roommate, Dana. If you were merely to consider personal requests or random allocations, you might end up with disgruntled **students. R8 denotes Anna, and K6 denotes Charlie. The resulting dissatisfaction could lead** to multiple room change **requests** and **conflicts. It could also create** an unsettling atmosphere in the dormitories.

Options Reference: B
A. **The numerical count of freshmen in the hall**
B. Roomates
C. Dormitories
D. Rooms

Recast: Equivalent reformulation

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. The ranking is showcased in a matrix, with each row corresponding to an individual student, and each column indicating their preference for others. On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if Anna and Bob are **roommates and Charlie's current roommate is Dana, the count of the following preferences that hold must be at most one:** Anna **preferring** Charlie over **Bob, and** Charlie **preferring** Anna over Dana. If you were merely to consider personal requests or random allocations, you might end up with disgruntled students, leading to multiple room change requests, conflicts, and an unsettling atmosphere in the dormitories.

Options Reference: B
A. Ranking
B. Roomates
C. **Allowing one mutually preferred blocking pair**
D. Rooms

Irrelevant: Background information injection

Description
As the housing manager of a university, the start of every academic year brings with it a rush of excitement and anticipation. The campus is a buzz with energy as incoming freshmen arrive, all eager to begin their new chapter. One of your primary responsibilities is ensuring these students find a compatible roommate, setting the foundation for their first year experience. You are faced with a hall filled with 120 fresh-faced freshmen, all of whom have ranked their preference for potential roommates. The ranking is showcased in a matrix, with each row corresponding to an individual student, and each column indicating their preference for others. On paper, it seems a simple task - just match according to preferences, but the complexity of the situation is much greater. Your challenge is not only to pair them up but to do so in a way that ensures stability. This means, if Anna and Bob are roommates, Anna shouldn't prefer Charlie over Bob if Charlie himself prefers Anna over his current roommate, Dana. If you were merely to consider personal requests or random allocations, you might end up with disgruntled students, leading to multiple room change requests, conflicts, and an unsettling atmosphere in the dormitories. **At semester's end, the housing office creates another student-by-roommate matrix from completed occupancy records. Each ordered pair receives a count of maintenance contacts and a room-change review status, while complaints are coded by category. This retrospective matrix is stored with residence-life reports for comparing prior cohorts.**

Options Reference: B
A. Ranking
B. Roomates
C. Dormitories
D. **Complaint categories and room-change review statuses**

Figure 5: A roommate-matching case from ORQA (Q5; source row 260): the original input (Origin4) and five context variants in the incorrect-option replacement setting of RobustOpt-QA. Full contexts and options are shown; the question is shared. Blue highlights mark context edits and replacement options. The reference answer remains B (the source spelling "Roomates" is retained).

D.1 Problem sources and formulation types

RobustOpt-Coding contains 200 original problems, called seeds, from six datasets (Table 4). MAMO contributes optimization tasks from its Complex LP collection; OptiBench denotes the collection introduced with ReSocratic (Yang et al., 2025). The construction protocol balances formulation types and application areas while limiting repeated templates.

Table 4: Sources and formulation types of the 200 Coding problems. Counts are summarized separately; entries on the same row do not imply a source–type correspondence.

Source	Seeds	Formulation type	Seeds
MAMO	30	LP	35
NL4Opt	30	Pure IP	47
IndustryOR	30	MILP	18
OptiBench / ReSocratic	70	QP / MIQP	30
OptMATH	30	SOCP / QCQP	30
MIPLIB-NL	10	NLP / MINLP	40
Total	200	Total	200

The registry assigns each seed one formulation type. Linear models are grouped as continuous LP, pure integer programming (IP), or mixed-integer linear programming (MILP). The remaining groups distinguish quadratic objectives with linear constraints (QP/MIQP), conic or quadratic constraints (SOCP/QCQP), and more general nonlinear expressions (NLP/MINLP). Input validation and evaluated-panel coverage are documented separately in Apps. D.2 and G.

The seeds comprise 190 text-only problems and 10 with accompanying data files. The registry records 21 application labels; Figure 3 presents eight broader application fields. These are descriptive annotations of benchmark coverage.

D.2 Presentation changes and input validation

Table 5 summarizes ten basic methods, six additional methods, and their combinations. Each admitted perturbation must preserve the feasible decisions and their objective values after the recorded normalization.

Construction checks. The validation panel for the six additional methods contains 199 single-method variants, 36 combinations, 200 originals, and 36 original-identical CLEAR controls. All 471 records pass the applicable construction checks. For edits, these compare the constructed text and files with source fields or approved replacement text. Each of the 235 edited inputs also has one deliberately corrupted field rejected by its checker. These targeted tests support the implemented checks without establishing exhaustive error detection.

Human input review. All 2,061 inputs, including controls, were reviewed and approved by a team of six optimization researchers comprising PhD students, postdoctoral researchers, and professors. This review concerns the benchmark inputs. Reference coverage and checks of generated models are reported separately in App. G; input approval does not certify those outputs.

Preserving requirements in the basic methods. Reordering moves premise blocks, such as paragraphs, list items, or table blocks, while keeping the final task instruction in place. O2 changes which block comes first for approximately half of all pairs of blocks; the constructor chooses the closest attainable number. O4 reverses the entire sequence, changing the order of every pair. As an illustration, the sequence A–B–C–D can become C–A–D–B under O2 (three of six pairs reversed), or D–C–B–A under O4 (all six). This measures changes in order, not text length or problem difficulty. Both edits preserve meaning, negation, and temporal relationships. Fragmentation spreads one requirement across sentences while retaining its subject, index scope, comparison, bound, and quantifier. Added background must be independent of the current decision: for example, a past inventory quantity is not irrelevant if the problem includes carryover.

Unit conversion must consistently update quantities, coefficients, bounds, output units, and any discrete step sizes; rounding must not alter feasibility. Cross-sentence binding must retain an unambiguous entity–parameter association. Logical reformulation must preserve truth conditions over the stated domains, including the assumptions required for any big-M encoding. Conversion between prose and structured data must retain all requirement-bearing headers, qualifiers, units, captions, and file links.

Selected Coding inputs

Original and transformed descriptions

Original

Problem description 1 | nl4opt_015

A theme park transports its visitors around the park either by scooter or rickshaw. A scooter can carry 2 people while a rickshaw can carry 3 people. To avoid excessive pollution, at most 40% of the vehicles used can be rickshaws. If the park needs to transport at least 300 visitors, minimize the total number of scooters used.

LLMOPT (14B)	✓
SIRL (32B)	✓
DeepSeek V4 Pro	✓

O2: Statement order

Problem description 2 | nl4opt_008

Each week, 17 gallons of stain and 11 lengths of oak wood are available. A chair produced by Elm Furniture yields a profit of $43, while every dresser yields a $52 profit. Each chair requires 1.4 gallons of stain and 2 lengths of oak wood, while each dresser requires 1.1 gallons of stain and 3 lengths of oak wood. Determine the maximum profit.

LLMOPT (14B)	×
SIRL (32B)	✓
DeepSeek V4 Pro	×

O4: Reversed statements

Problem description 3 | nl4opt_138

The farmer needs to transport 80 cows and he only has enough money for at most 8 truck trips. **A truck can take 7 cows per trip and creates 10 units of pollution. A helicopter can take 3 cows per trip and creates 5 units of pollution. A farmer decides to move his cows to a nearby farm using helicopters and trucks.** How many of each type of trip should be taken to minimize the total amount of pollution produced?

LLMOPT (14B)	✓
SIRL (32B)	×
DeepSeek V4 Pro	✓

CB: Cross-sentence binding

Problem description 4 | optibench_434

There is a closed cylinder. The volume of the cylinder is 16π. Find the dimensions of **that same** cylinder that has the least amount of surface area.

LLMOPT (14B)	×
SIRL (32B)	×
DeepSeek V4 Pro	×

IC: Topic / distant scale

Problem description 5 | nl4opt_015

A theme park transports its visitors around the park either by scooter or rickshaw. A scooter can carry 2 people while a rickshaw can carry 3 people. To avoid excessive pollution, at most 40% of the vehicles used can be rickshaws. **During an independently funded, completed transport-planning rehearsal from an earlier accounting period, the theme park catalogued 30000 archival sample units.** If the park needs to transport at least 300 visitors, minimize the total number of scooters used.

LLMOPT (14B)	×
SIRL (32B)	✓
DeepSeek V4 Pro	✓

IC: Topic / plausible scale

Problem description 6 | nl4opt_015

A theme park transports its visitors around the park either by scooter or rickshaw. A scooter can carry 2 people while a rickshaw can carry 3 people. To avoid excessive pollution, at most 40% of the vehicles used can be rickshaws. **During an independently funded, completed transport-planning rehearsal from an earlier accounting period, the theme park catalogued 44 archival sample units.** If the park needs to transport at least 300 visitors, minimize the total number of scooters used.

LLMOPT (14B)	×
SIRL (32B)	×
DeepSeek V4 Pro	✓

IC: Topic / plausible scale

Problem description 7 | industryor_075

There are A and B two products, both requiring two successive chemical reaction processes. Each unit of product A needs 2 hours for the first process and 3 hours for the second process. Each unit of product B needs 3 hours for the first process and 4 hours for the second process. Available time for the first process is 16 hours, and available time for the second process is 24 hours. For each unit of product B produced, 2 units of by-product C are generated simultaneously, requiring no additional cost. By-product C can be sold up to 5 units, and the rest must be disposed of at a cost of 2 yuan per unit. Each unit of product A sold yields a profit of 4 yuan, each unit of product B yields a profit of 10 yuan, and each unit of by-product C sold yields a profit of 3 yuan. **During an independently funded, completed production-planning rehearsal from an earlier accounting period, the product team catalogued 6 archival sample units.** In order to maximize total profit, establish the linear programming model for this problem.

LLMOPT (14B)	✓
SIRL (32B)	×
DeepSeek V4 Pro	×

LE: Logical reformulation

Problem description 8 | nl4opt_170

A golf course is hosting an event and can transport guests using either golf carts or pull carts. A golf cart can take 4 guests while a pull cart can take 1 guest. Since golf carts take up a lot of space, **the number of golf carts must be no greater than 3/5 of the total number of** carts. If the golf course needs to transport at least 80 guests, how many of each cart should be used to minimize the total number of carts needed?

LLMOPT (14B)	✓
SIRL (32B)	×
DeepSeek V4 Pro	✓

PS: Structured format

Structured input 9 | nl4opt_015

```json
{"format": "ordered_problem_statements",
"segments": [
{"segment_id": "S001", "text": "A theme park transports its visitors around the park either by scooter or rickshaw."},
{"segment_id": "S002", "text": "A scooter can carry 2 people while a rickshaw can carry 3 people."},
{"segment_id": "S003", "text": "To avoid excessive pollution, at most 40% of the vehicles used can be rickshaws."},
{"segment_id": "S004", "text": "If the park needs to transport at least 300 visitors, minimize the total number of scooters used."}
]}
```

LLMOPT (14B)	✓
SIRL (32B)	✓
DeepSeek V4 Pro	×

Figure 6: Selected RobustOpt-Coding inputs illustrating original and transformed descriptions. Several cards illustrate different transformations of the same source problem; blue highlights mark edits relative to each input's own original. ✓/×: pass/fail under the numerical criterion, including execution failures.

Table 5: Coding perturbations and their preservation requirements. Counts describe 1,795 variant records, including repeated positions, and exclude 200 originals, 30 changed-target controls, and 36 CLEAR controls. These are construction counts; numerical-panel eligibility is reported in App. G.

Operator	Count	Preservation requirement and intended challenge
Moderate reorder	200	Same premise units; sensitivity to information position.
Complete reversal	200	Same movable premises; integration across distant requirements.
Unrelated background	200	Independent fact; generic distraction.
Topic / distant scale	200	Independent, topically related fact; spurious data inclusion.
Topic / plausible scale	200	Independent fact with plausible quantity; numerical leakage.
Constraint fragmentation	93	Same quantified constraint; cross-sentence integration.
Linked unit conversion	84	Same physical quantities after normalization; unit binding.
Cross-sentence binding	118	Same unique entity–parameter map; reference resolution.
Logical reformulation	65	Same admissible decisions; logical equivalence.
Prose–structured conversion	200	Same complete record; format dependence.
Scope and conditions (SC)	55	Separate a rule from its scope; preserve subjects, quantifiers, and conditions.
Cross-table binding (BT)	58	Link tables or a code directory; recover the same entities, values, and units.
Effective versions (RV)	55	Combine opening values with approved adjustments; exclude future plans.
Temporal scope (TM)	6	Move existing timing clauses to an annex; preserve order and boundaries.
Measurement roles (MR)	15	Recover unit rates from sample totals and trial conditions; preserve parameter roles.
Accounting conventions (AC)	10	Use payroll or settlement descriptions; preserve counting and aggregation rules.
Two-method combinations	36	Combine compatible edits; recheck the complete input against the same reference.
Total variant records	1,795	1,560 basic records + 199 single-method + 36 combined variants.

Linking information across operational materials. The six additional methods distribute information across clauses, tables, and reports. SC separates a rule from its scope and conditions while leaving their connection unique. BT requires a unique join between tables or a code directory to recover the original entity–value associations. RV expresses an original value through an opening amount and approved adjustments, excluding future plans. TM relocates existing timing clauses without changing periods or boundaries. MR recovers a coefficient from measurement totals and calibration conditions; those conditions must not become operating constraints. AC changes how charges are described while preserving whether they apply per person, per period, or once, and whether ratios are computed before or after aggregation. A method may involve several model components, so these categories do not imply independent error causes.

Combining compatible edits. The 36 combinations are SC+BT (10), RV+BT (13), TM+BT (2), SC+RV (3), MR+SC (1), AC+BT (2), and AC+RV (5). Construction considers these pairs in the listed order and retains at most one combination per seed, together with both constituent single edits. The edits must affect nonoverlapping source spans, and the complete combined input must pass both preservation checks and semantic review. Passing each single-edit check does not establish that their composition preserves the problem.

D.3 Worked examples of input validation

Recovering production rates (MR). In `optibench_325`, production line 1 produces 20, 30, and 40 units of the three products per worker-hour. The MR variant presents these rates through calibration totals of 80, 120, and 160 units, together with a log specifying two workers and two hours at the same constant rates used in production. Dividing each total by the four worker-hours recovers the original rate; for example, $80/(2 \times 2) = 20$. Production staffing and duration remain optimization decisions, with the unchanged limit of 20 workers per line.

Recovering ingredient identities (BT). In `industryor_014`, the original description associates a 100 g broccoli pack with 2.8 g of protein and a cost of \$1.20. The BT variant separates this information: the coded record gives the protein content and cost for R04, while a directory maps R04

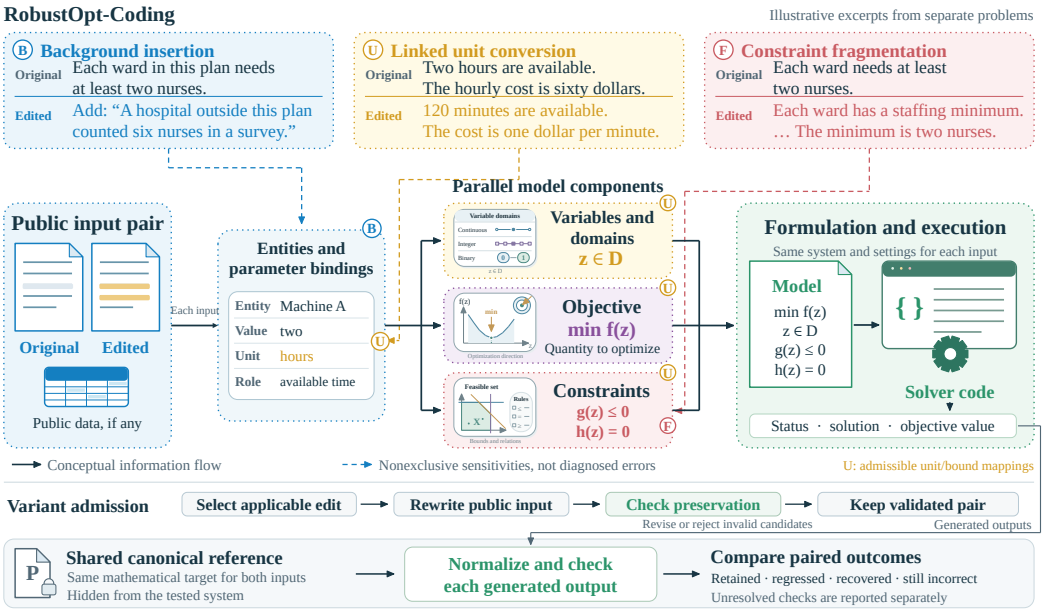

Figure 7: From presentation changes to paired evaluation in RobustOpt-Coding. The three excerpts illustrate separate transformations. Bindings support parallel domain, objective, and constraint components, not an assumed internal reasoning sequence. Each generated output is checked against the same hidden reference before paired outcomes are compared. Colored links indicate possible sensitivities, not observed errors.

to "Broccoli (100g pack)." Matching the code reconnects the ingredient, pack size, and numerical values. Vegetable purchases remain integer numbers of packs, and the meal must still contain at least three distinct vegetable types.

D.4 Construction records and scoring eligibility

The Coding construction dataset contains 200 originals, 1,795 variant records, 36 CLEAR controls, and 30 changed-target controls, totaling 2,061 records. The six additional methods and their combinations contribute 235 variants covering 128 source problems; the other 72 have only basic variants.

Moderate reordering and complete reversal produce identical inputs for six problems, leaving six redundant record positions. Both operator labels are kept for each duplicate input, but each distinct input per problem counts once in pooled retention and joint-survival metrics. The 36 CLEAR controls copy original inputs, and 30 changed-target controls alter the intended requirements; both are excluded from these metrics. Further scoring eligibility rules and the resulting numerical panel are given in App. G.

E Shared Evaluation Conventions

This appendix defines the record counts, formulation equivalence, and statistical conventions needed to interpret the reported evaluations. Benchmark-specific procedures and results appear in Apps. F and G.

E.1 Record counts and benchmark profiles

QA contains 200 source questions sharing 109 contexts. Each question has an original presentation and five variants, giving 1,200 question–context presentations; the 1,000 variants are counted by question, not by distinct context text. Coding contains 200 originals, 1,795 variant records, 30

changed-target controls, and 36 CLEAR copies, totaling 2,061 records. Collapsing the CLEAR copies and six repeated variants leaves 2,019 distinct pairs of source problem and recorded input text, including the 30 changed-target controls. These construction inventories are distinct from model-generation counts and scored evaluation panels: protocol-specific QA controls and Coding eligibility rules are given in Apps. F and G.

Figure 3 counts each original question or problem once in its composition and variable-family panels. Problem classes, formulation types, and application fields are separate marginal distributions. The length panel also includes the 1,000 QA and 1,795 Coding variant records before deduplication; lengths refer to QA contexts and Coding problem descriptions. A primary variable family groups decisions with a shared role, such as production quantities indexed by product, rather than counting individual scalar variables. Application-field annotations are descriptive and partly LLM-assisted or heuristic; they are not independently validated domain labels or estimates of deployment frequencies.

E.2 Equivalent optimization formulations

Different formulations can represent the same optimization problem. Following Sec. 3, we compare them after recorded mappings to common decision variables, units, and objective conventions: they must allow exactly the same decisions and assign each the same objective value. For an invertible linear change of variables, substitute the inverse map into the objective and constraints, and transform all bounds and variable domains, including integrality restrictions. Any declared positive constant scaling of the objective or known decision-independent offset must be undone when comparing values and reporting the requested quantity.

Auxiliary variables are allowed if eliminating them preserves this criterion. To check both feasibility and objective values, we use the objective epigraph $E(P_i) = \{(z, t) : z \in X_i,\ f_i(z) \leq t\}$, where t is an upper bound on the objective at decision z. After the same normalization, projecting the extended formulation's full epigraph onto (z, t) must give exactly $E(P_i)$. This requires the same feasible decisions; for each one, some feasible auxiliary assignment must attain $f_i(z)$, and none may yield a smaller normalized objective. Projecting only the feasible set would miss objective errors.

Figure 1 gives a concrete example of a formulation error. Let s_A and s_B be surgery start times in hours after 08:00. With $s_A \geq 0$, uninterrupted surgeries, and A before B, the shared operating room requires $s_B \geq s_A + 3$. Immediate transfer to the single recovery bed gives occupancy intervals $[s_A + 3, s_A + 5)$ and $[s_B + 1, s_B + 3)$, so avoiding overlap requires $s_B \geq s_A + 4$. Minimizing the surgery completion time $s_B + 1$ gives the unique optimum $(s_A, s_B) = (0, 4)$. The erroneous model instead counts recovery from surgery start, using $[s_i, s_i + 2)$, and returns $(0, 3)$. The resulting schedule violates the actual bed requirement from 12:00 to 13:00. This synthetic example assumes zero transfer and bed-turnover time; it does not specify clinical standards.

An incorrect formulation can also return the correct optimum. Adding the unwarranted constraint $s_A = 0$ retains $(0, 4)$ but excludes the valid schedule $(1, 5)$. In contrast, renaming variables or adding the redundant constraint $s_B \geq 4$ preserves the full problem. These examples show why equivalence concerns all feasible decisions and their objective values. The numerical checks and checks of selected formulations reported in this paper do not establish this full criterion (Sec. 4.2).

E.3 Stochastic robustness and repeated evaluation

For a fixed problem and task, a stochastic version of Definition 2 requires correctness with probability at least $1 - \epsilon$ on each presentation in the specified admissible family, including the original. Here $0 \leq \epsilon < 1$ is the tolerated failure probability; the task criterion, model, prompts, tools, and decoding settings are fixed. This bound applies separately to each presentation, not to passing all presentations jointly. Finite tests provide evidence only for the tested inputs.

Repeated calls estimate an input's success probability only under a stable output distribution. The survival metrics S and $S_{5,\mathrm{QA}}$ instead count observed groups passing all required inputs under their respective eligibility rules (Sec. 4.2); this calculation needs no independence assumption. Predicting joint success by multiplying individual success probabilities would require independence. Repeated-call summaries must specify how outputs are grouped and whether success means passing at least once or on every attempt.

Same-input controls require new generations; copying an existing response does not create another trial. Fresh client sessions and deterministic decoding do not guarantee independent outputs or an unchanged API service. Rounds with different inference settings cannot estimate one fixed success probability. The QA repetitions and Coding round comparisons are described in Apps. F.4, G.3, and G.5.

E.4 Paired comparisons and uncertainty

Pairing. Rates are percentages and accuracy differences are percentage points. For a fixed system, protocol, and operator, the paired accuracy drop is the difference between the pass-to-fail and fail-to-pass counts, divided by the matched-pair count and multiplied by 100. Coding pairs an original with an alternative under numerical scoring. QA pairs a matched control with a perturbed context using identical options and ordering, and includes only jointly parseable outputs. Its control is not necessarily the untouched source input, and its reported rates do not cover all attempted outputs.

Weighting and eligibility. Equation (1) weights problems equally. Among originally passing Coding problems, $1 - R$ averages within-problem failure rates equally, whereas F pools eligible alternatives. Their interpretation requires the original accuracy and eligible counts; empty conditional denominators are undefined. Problems with no eligible variant in the underlying panel are excluded. A sampled problem with no eligible variant drawn is a different case, handled by the sampling design in App. G.5. Survival comparisons require a common variant panel; QA retains its separate matched-control populations. Distinct source–input pairs count once in pooled invariance metrics, with operator memberships retained for separate analyses. CLEAR and changed-target controls are excluded; changed-target controls require revised references.

Uncertainty and diagnostic bounds. The QA defense experiment reports exploratory paired bootstrap intervals over shared contexts (App. F.3). Resampling preserves the compared conditions within each context; variants sharing a source are not independent replicates. These intervals are marginal, without multiple-comparison correction, and do not quantify variation in new model generations. Three-round dispersion and missing-outcome bounds describe different sources of variation and are not confidence intervals (Apps. G.3 and G.5). Likewise, formulation-component bounds describe the errors confirmed in the examined outputs, not the overall frequency of errors or statistical uncertainty. One output can have errors in several components, so adding the component counts can count the same output more than once (App. G.4). The reported analyses do not establish confirmatory significance across systems or perturbations.

F RobustOpt-QA: Experimental Protocol and Results

This appendix reports matched QA evaluation, perturbation diagnostics, self-checking defense results, and repeated API evaluations. Input construction and validation appear in App. C.

F.1 RobustOpt-QA settings and robustness results

Evaluation design. We evaluate fourteen systems on 200 source questions sharing 109 contexts. Each question has five variants—Indirect, Scattered, Aliased, Recast, and Irrelevant—giving 1,000 control–variant pairs per system under each protocol (Tables 1 and 6).

Origin4 is the unchanged four-option baseline. In the *answer addition setting* of RobustOpt-QA, Origin5 and P5 share the original options plus a constructed distractor, using the original and perturbed contexts, respectively. In the *incorrect-option replacement setting*, Control4R and Perturbed4R share four options after one incorrect option is replaced by that distractor. Within each matched pair, the question, option content, and option order are identical, so the comparison evaluates a context change with fixed options. These protocols do not form a complete factorial experiment: the edited context with the original four options is absent. The Origin4 contrasts in App. F.4 instead combine context and answer-set changes.

Metric populations. We use the definitions in Sec. 4.2. Correctness retention R_{QA} conditions on correct matched-control answers; joint correctness J_{QA} and semantic answer-change rate $\mathrm{ACR}_{\mathrm{QA}}$ use all jointly parseable pairs. Answer changes include losses, recoveries, and switches between

Table 6: RobustOpt-QA robustness in the answer addition and incorrect-option replacement settings. Each cell reports a percentage (in %) and its numerator/denominator. Both panels report the same fourteen systems. All cells use Round 1 with natural option order; the three-round analysis is reported separately in Table 10.

Model	Correctness retention R_{QA}	Joint correctness J_{QA}	All-five retention $S_{5,\mathrm{QA}}$	Answer change $\mathrm{ACR}_{\mathrm{QA}}$
A. Answer addition setting: Origin5 → P5				
LLMOPT (14B)	93.64 (707/755)	70.70 (707/1000)	83.94 (115/137)	10.40 (104/1000)
OptMATH (7B)	92.29 (407/441)	40.70 (407/1000)	86.67 (65/75)	9.80 (98/1000)
OptMATH (32B)	96.85 (739/763)	73.90 (739/1000)	88.89 (128/144)	4.90 (49/1000)
ORLM (8B)	85.66 (442/516)	45.24 (442/977)	68.00 (51/75)	15.86 (155/977)
SIRL (7B)	94.68 (569/601)	63.58 (569/895)	91.75 (89/97)	9.39 (84/895)
SIRL (32B)	96.91 (783/808)	78.30 (783/1000)	89.10 (139/156)	4.40 (44/1000)
StepORLM (8B)	95.39 (662/694)	66.20 (662/1000)	83.85 (109/130)	6.80 (68/1000)
Claude Opus 4.6	97.25 (883/908)	88.30 (883/1000)	90.34 (159/176)	4.70 (47/1000)
DeepSeek V4 Pro	96.63 (859/889)	85.90 (859/1000)	94.51 (155/164)	6.70 (67/1000)
Gemini 3.1 Pro	98.73 (935/947)	93.50 (935/1000)	97.84 (181/185)	2.80 (28/1000)
GLM-5.1	96.62 (858/888)	85.80 (858/1000)	92.17 (153/166)	6.40 (64/1000)
GPT-5.5	96.52 (831/861)	83.10 (831/1000)	93.71 (149/159)	7.20 (72/1000)
Kimi K2.6	97.20 (867/892)	86.70 (867/1000)	97.58 (161/165)	5.70 (57/1000)
Qwen 3.5 122B	97.11 (841/866)	84.10 (841/1000)	92.59 (150/162)	5.30 (53/1000)
B. Incorrect-option replacement setting: Control4R → Perturbed4R				
LLMOPT (14B)	93.19 (698/749)	69.80 (698/1000)	77.12 (91/118)	9.80 (98/1000)
OptMATH (7B)	90.28 (520/576)	52.00 (520/1000)	74.03 (57/77)	11.40 (114/1000)
OptMATH (32B)	96.13 (771/802)	77.10 (771/1000)	87.97 (117/133)	4.80 (48/1000)
ORLM (8B)	91.10 (522/573)	52.52 (522/994)	80.00 (56/70)	13.48 (134/994)
SIRL (7B)	95.00 (627/660)	68.30 (627/918)	88.51 (77/87)	9.26 (85/918)
SIRL (32B)	96.74 (801/828)	80.10 (801/1000)	90.07 (127/141)	4.90 (49/1000)
StepORLM (8B)	93.86 (719/766)	71.90 (719/1000)	80.65 (100/124)	7.60 (76/1000)
Claude Opus 4.6	98.16 (907/924)	90.70 (907/1000)	94.67 (160/169)	3.80 (38/1000)
DeepSeek V4 Pro	97.67 (882/903)	88.20 (882/1000)	95.12 (156/164)	4.40 (44/1000)
Gemini 3.1 Pro	98.64 (945/958)	94.50 (945/1000)	96.74 (178/184)	2.00 (20/1000)
GLM-5.1	96.83 (885/914)	88.50 (885/1000)	94.55 (156/165)	4.20 (42/1000)
GPT-5.5	95.92 (870/907)	87.00 (870/1000)	90.18 (147/163)	6.80 (68/1000)
Kimi K2.6	97.37 (888/912)	88.80 (888/1000)	94.01 (157/167)	5.20 (52/1000)
Qwen 3.5 122B	97.21 (871/896)	87.10 (871/1000)	93.21 (151/162)	5.30 (53/1000)

Coverage. Each row has 1,000 parseable pairs and 200 fully parseable question groups, except ORLM (8B): 977/185 in Panel A and 994/194 in Panel B; and SIRL (7B): 895/148 and 918/153. Across all three input conditions in both protocols, ORLM and SIRL (7B) have 32 and 316 unparseable outputs, respectively.

Configuration and references. The panel is agent-v3-20260911; SIRL (32B) denotes the local SIRL-COPT32B checkpoint. The 14-system panel excludes the archive's historical Sonnet runs. For Opus 4.6, R1 denotes the designated full rerun (`r1r`), collected after R2/R3; round labels do not imply execution order. Both model groups use natural option order. Local runs use greedy decoding; API requests enable reasoning or request high reasoning effort.

incorrect answers. All-five retention $S_{5,\mathrm{QA}}$ uses question groups with all ten outputs parseable and all five controls correct, and counts groups whose five variants also pass. Each variant has its own matched control because its answer set can differ; this is not survival relative to one original response. Rates are percentages, with zero denominators undefined.

F.2 QA ACCURACY BY PERTURBATION AND QUESTION TYPE

We examine which description changes affect answers to different modeling questions. Figure 8 reports results separately for the seven optimization-fine-tuned LLMs and the seven general-purpose LLMs, combining responses within each group. It compares each variant's matched control with its perturbed input, holding the question and the complete answer options fixed, so that only the description changes. The same variants also carry an original four-option input, which expresses the combined effect of the description and the option change, as explained below.

The question types cover objectives (Q1: criterion; Q8: parameters; Q9: participating decision activities), sets and decisions (Q5: model sets; Q6: decision activities), constraints (Q2: a named constraint; Q7: required constraints; Q10: a constraint's left-hand side; Q11: its scope), and

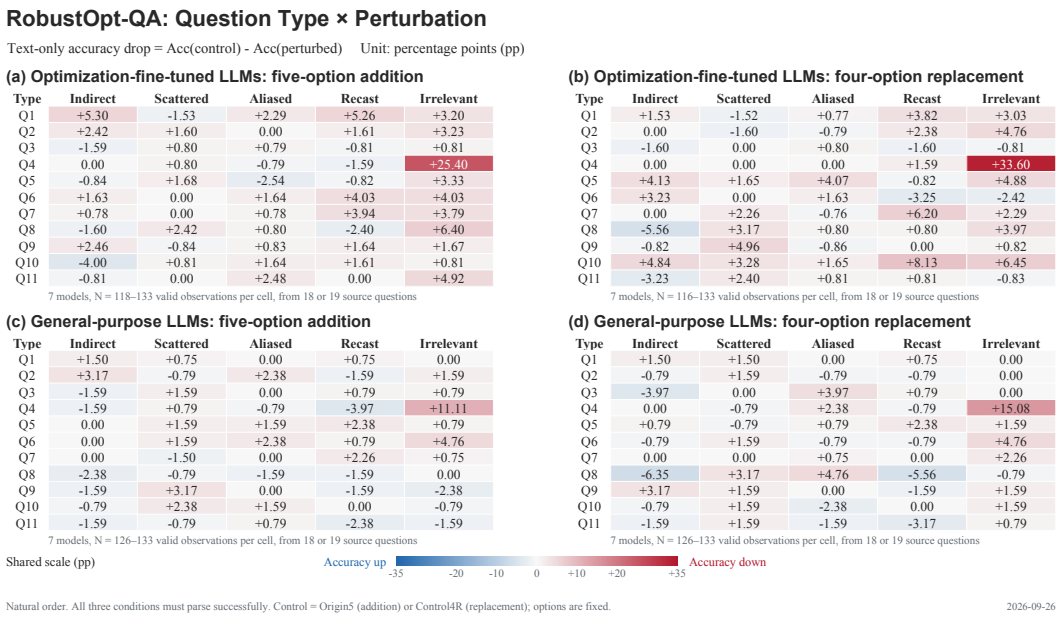

Figure 8: RobustOpt-QA accuracy changes by question and perturbation type. The top panels combine responses from seven optimization-fine-tuned LLMs; the bottom panels combine responses from seven general-purpose LLMs. The left and right panels show the answer addition and incorrect-option replacement settings, respectively. Each cell includes only observations with parseable answers to the original input, its matched control, and its perturbed version. Values are accuracy drops from the matched control to the perturbed input, in percentage points. Within a pair the question and the complete answer options are identical, so a cell isolates the description change. Red denotes a decrease; blue denotes an increase. All cells use Round 1 with natural option order; the three-round analysis is reported separately in Table 10.

classification (Q3: application category; Q4: mathematical model type, such as LP, MILP, or QP). Negated or exclusion questions retain their question-specific reference answers.

Separating context changes from answer-set changes. Figure 8 uses natural option order and compares each variant's matched control with its perturbed input; within a pair the question and the complete answer options are identical, so only the description changes. The same valid triples also carry an original four-option input, which lets the total difference be split into two steps: Origin4 to Origin5 or Control4R changes the options, and the matched control to the perturbed input changes the description with the options fixed. The figure plots the second step. These are descriptive accuracy differences, rather than isolated causal effects.

Adding irrelevant background lowers accuracy on Q4, which asks which mathematical optimization model represents the problem. With answer options held fixed, the drops in the answer addition and incorrect-option replacement settings are 25.40 and 33.60 percentage points for the optimization-fine-tuned group, and 11.11 and 15.08 for the general-purpose group; these are the values plotted in Figure 8. Including the option change as well gives total drops of 30.16 and 29.60 points for the fine-tuned group, and 12.70 and 17.46 for the general-purpose group.

A different combination of edit and question has a much smaller description-only drop. Q5 asks which sets of entities, such as products or plants, are needed in the model. In the answer addition setting, scattering related facts lowers the fine-tuned group's accuracy by 1.68 points, which is what Figure 8 shows for this cell. The larger 20.17-point total drop for the same case comes mainly from the option comparison, which accounts for another 18.49 points. Thus, the large loss in that cell is not driven by scattering the facts.

The figure also contains increases in accuracy, though they are small. In the incorrect-option replacement setting, irrelevant background coincides with a 0.83-point gain for the fine-tuned group

on Q11, which asks which objects a constraint applies to, while the general-purpose group shows a 0.79-point loss on the same cell. The gains on the same cell under the combined description and option change are much larger, 13.22 points for the fine-tuned group and 8.73 for the general-purpose group; these come mainly from the option change rather than from the description edit. They do not show that irrelevant background alone improves answers.

Interpretation and scope. The results vary by question, perturbation, and protocol; the constructed variants do not uniformly increase difficulty. Q1 and Q7 contain 19 source questions each, and the remaining question types contain 18 each. Shared contexts and repeated observations across systems make the pooled counts dependent. The figure and its decomposition are descriptive diagnostics of the evaluated responses, subject to the uncertainty limits in App. E. They do not establish input equivalence, addressed in App. C, or replace the separate Coding formulation checks.

F.3 Evaluating structured self-checking as a defense

Strategy: check the requirements before answering. On RobustOpt-QA, we test a prompt that asks OptMATH (7B), using the OptMATH-Qwen2.5-7B checkpoint, to review the requirements before choosing an answer. The model checks what the question asks, what each name refers to, which quantities are decisions or given data and which are derived, what is optimized, which constraints apply, and which sentences support its answer. It writes these checks and the final answer in one response. We apply the procedure to both original and rewritten descriptions, without additional training, solver execution, reference answers, or access to the other description in the pair.

Experimental setup. We compare self-checking with direct answering on the same 200 questions and their five rewrites per question, drawn from 109 contexts. Answer options retain their original order. The answer addition setting adds a fifth, incorrect option; the incorrect-option replacement setting substitutes one incorrect option while retaining four. For each rewrite, we evaluate three inputs: the original context and original options, the original context with the modified options, and the rewritten context with those same modified options. These are Origin4, Origin5, and P5 for the answer addition setting, and Origin4, Control4R, and Perturbed4R for the incorrect-option replacement setting. Comparing the last two inputs tests the description change while holding options fixed.

Each protocol contains 1,000 such triples and 3,000 responses; 5,968/6,000 self-checking responses contain an identifiable answer (99.47%). We use agent-v3-20260911 with greedy decoding, BF16, eager attention, seed 0, batch size 2, and an 8,192-token requested completion budget limited by available context. We compare one self-checking round with baseline round 1; the three identical baseline rounds add no independent observations.

Measuring defense effectiveness. Let B and S denote baseline and defense, P the perturbed input, and C its fixed-option control (Origin5 for the answer addition setting; Control4R for the incorrect-option replacement setting). Perturbed-answer gain $G = A_S(P) - A_B(P)$ uses the N_P perturbed records parseable in both configurations. The signed gap $D_r = A_r(C) - A_r(P)$, for $r \in \{B, S\}$, measures net accuracy loss; $M = D_B - D_S$ measures its reduction. All four accuracies in M use the same N_Q control–perturbed pairs parseable in both configurations. Thus, M compares perturbed and control gains on common observations; a positive value need not mean a smaller absolute gap or better retention. Differences are in percentage points (pp). Exploratory 95% intervals use 5,000 paired bootstrap resamples of the 109 shared contexts, without multiplicity correction or repeated-generation uncertainty.

The defense improves perturbed-answer accuracy in both RobustOpt-QA settings. Self-checking raises accuracy from 43.40% to 57.20% in the answer addition setting ($+13.80$ pp) and from 54.74% to 63.10% in the incorrect-option replacement setting ($+8.37$ pp), with positive exploratory intervals for both gains (Table 7). On 984 quadruples in the incorrect-option replacement setting, the matched gap falls from 3.05 pp to zero, with M's interval $[0.19, 5.84]$ pp. The answer addition setting gives less clear evidence of gap reduction: the signed gap changes from $+0.40$ to -1.21 pp, and M's interval spans zero. Origin5 accuracy also improves by 12.26 pp on its own baseline/defense intersection; this separate estimate cannot be used to decompose G.

Table 7: OptMATH-Qwen2.5-7B with and without self-checking. (a) Perturbed accuracy and gain G on jointly parseable outputs. (b) Signed control–perturbed gaps and their reduction M on parseable quadruples. Add and Replace denote the answer addition and incorrect-option replacement settings of RobustOpt-QA, respectively. Panels use different valid intersections; differences are computed before rounding.

(a) Accuracy on perturbed inputs

Protocol	N_P	$A_B(P)$ (%)	$A_S(P)$ (%)	G (pp)	95% interval for G
Add	993	43.40	57.20	+13.80	[9.75, 18.07]
Replace	992	54.74	63.10	+8.37	[3.71, 13.06]

(b) Signed control–perturbed accuracy gaps

Protocol	N_Q	D_B (pp)	D_S (pp)	M (pp)	95% interval for M
Add	988	+0.40	−1.21	+1.62	[−1.45, 4.54]
Replace	984	+3.05	0.00	+3.05	[0.19, 5.84]

Table 8: Signed-gap reduction M by perturbation on RobustOpt-QA. Add and Replace denote the answer addition and incorrect-option replacement settings, respectively. Each protocol–category cell uses its own parseable quadruples. Positive M means a smaller signed control–perturbed gap; these estimates are descriptive.

Perturbation	Add N_Q	Add M (pp)	Replace N_Q	Replace M (pp)
Indirect	197	+0.51	196	−0.51
Scattered	197	+1.52	195	−1.03
Aliased	200	+4.50	197	+5.58
Recast	198	−2.53	198	+4.04
Irrelevant	196	+4.08	198	+7.07

Where does the defense help? In the incorrect-option replacement setting, the largest gap reductions occur for Irrelevant (7.07 pp), Aliased (5.58 pp), and Recast (4.04 pp; Table 8). Irrelevant and Aliased improve by this criterion under both protocols; other categories vary in direction. These descriptive results show selective benefits without isolating the contribution of individual checks.

Irrelevant–Q4 illustrates recovery on mathematical model classification. Across its 18 questions, accuracy in the incorrect-option replacement setting rises from 4/18 (22.22%) to 10/18 (55.56%), and the matched gap halves from 33.33 to 16.67 pp. Accuracy in the answer addition setting rises from 16.67% to 55.56%, with a 5.56-pp gap reduction. In a packet-scheduling example, irrelevant background describes a quadratic diagnostic computed after transmission. The baseline selects non-convex quadratic programming; the defense selects the reference integer-linear-programming answer. This correction illustrates the intended benefit of checking task-relevant requirements, without establishing an internal reasoning mechanism.

Main finding: higher accuracy does not eliminate failures after rewriting. Self-checking improves average accuracy, but does not keep every correct answer correct after a rewrite. In the incorrect-option replacement setting, 984 question–variant pairs have readable answers in all four conditions: without and with self-checking, each before and after the context change. With self-checking, 69 pairs change from a correct answer to an incorrect one and 69 change in the reverse direction. These losses and recoveries cancel, leaving the two accuracies equal. Semantic answer changes increase from 11.59% to 20.02%, while retention decreases from 90.19% to 88.92%. Each retention score uses the answers correct before rewriting in its own configuration (Sec. 4.2); the two percentages therefore do not describe exactly the same set of answers. In the answer addition setting, the absolute accuracy gap grows from 0.40 to 1.21 pp despite positive M.

The defense also changes prefill, output format, parsing, and stopping rules; no matched generic-checking control isolates these changes. Mean output length rises from about 2 tokens to 193 in the answer addition setting and 201 in the incorrect-option replacement setting. Identical Origin4 prompts yield different predictions in 117/998 parseable cross-protocol pairs, all associated with different inferred batch-padding widths; this association does not identify the cause. Results therefore concern the complete defense configuration in one round, with natural option order and parseable-output conditioning.

Future work. Future experiments should compare checking strategies with equal limits on response length and the same rules for extracting final answers. They should test which checks keep previously correct answers correct across rewrites, in addition to measuring average accuracy, and repeat the comparison with more LLMs. This would show whether the benefit extends beyond the OptMATH (7B) configuration tested here.

F.4 Repeated inputs as a QA comparison baseline

Evaluation inputs and models. We ask whether rewriting a problem produces more answer changes than repeating the same input. This comparison uses responses from seven LLMs in the answer addition and incorrect-option replacement settings of RobustOpt-QA. Each of 200 questions has five responses to exactly the same original context, question, and four answer options. The evaluation also includes one unchanged-description control and one rewritten input for each of the five variants, with identical questions and options within each pair. Table 9 lists the evaluated model versions. These are the seven API models of the fourteen-system set in Table 6.

Measuring answer changes. For the original inputs, we compare all ten pairs among the five responses to each question, giving 2,000 pairs per model and protocol. We count a difference when the selected answers have different meanings. The rewrite comparison uses the 1,000 control–variant pairs, with semantic option mappings retained. All 42,000 responses can be parsed, so no records are excluded for missing answers. These rates measure changes in selected answers, including switches between wrong answers; they are not failure rates. For DeepSeek V4 Pro, repeated original four-option inputs yield nonidentical answers on 19/200 questions in runs for the answer addition setting and 21/200 in runs for the incorrect-option replacement setting. These question counts correspond to pairwise rates of 4.40% and 5.10%.

Panel A reports all 200 questions, but the comparisons use different options: repetition uses the original four, while the control–variant pairs use the protocol's added or replaced option. For example, DeepSeek's rewrite-pair rates are 6.70% in the answer addition setting and 4.40% in the incorrect-option replacement setting. The former exceeds its repetition rate, while the latter does not. The option difference prevents treating these differences as the effect of rewriting alone.

Comparing repetition and rewriting with the same options. We also identify unchanged inputs that were presented more than once with exactly the same options. Without using model outcomes to select cases, we group controls by their context, question, and complete option list, retaining groups with at least two responses. We compare the repeated responses within each group, then compare those same controls with their own rewritten versions. Thus, both comparisons use the same questions and options. We first calculate each question's two rates, then average across questions equally; questions with more repeated pairs do not receive greater weight. Option-group weights also coincide within each question in these records.

The answer addition setting provides 29 questions from 26 contexts, with 121 repetition pairs and 92 control–variant pairs. The incorrect-option replacement setting provides 21 questions from 20 contexts, with 33 and 66 pairs, respectively. Panel B reports the averages across questions, so its percentages are not obtained by dividing the pooled disagreement counts by these pair totals.

What the comparison shows. With the same options in both comparisons, rewriting yields more answer changes for six of the seven LLMs in the answer addition setting, with increases of 1.44–6.03 percentage points; the remaining model does not. In the incorrect-option replacement setting, three LLMs have higher rates, one equal rate, and three lower rates. Thus, these data support additional variation in the answer addition setting in the matched subset, but not a consistent increase across both protocols. The subset is small and is determined by repeated option sets, not random sampling. Pairs share responses, and some questions share contexts; the displayed differences are descriptive, not significance tests. The response records include final answers and reconstructed prompts but lack complete request and retry logs, so these records do not establish independent trials or an unchanged service backend. A broader comparison should collect repetitions for every option set on the full benchmark.

Table 9: Answer-change rates on RobustOpt-QA with repeated inputs and rewritten descriptions. Repeat compares identical inputs; Rewrite compares unchanged and rewritten descriptions with fixed questions and options. Panel A includes all 200 questions but uses different option sets for repetition and rewriting; cells show percent (count/pairs). Panel B uses identical option sets in both comparisons and weights questions equally; cells show percentages. Rows identify the evaluated model versions.

	Answer addition setting		Incorrect-option replacement setting	
Model identifier	Repeat	Rewrite	Repeat	Rewrite
A. All 200 questions; option sets differ between comparisons				
claude-opus-4-6	1.00 (20/2000)	4.70 (47/1000)	0.60 (12/2000)	3.80 (38/1000)
deepseek-v4-pro	4.40 (88/2000)	6.70 (67/1000)	5.10 (102/2000)	4.40 (44/1000)
gemini-3.1-pro-preview	2.50 (50/2000)	2.80 (28/1000)	2.00 (40/2000)	2.00 (20/1000)
glm-5.1	5.10 (102/2000)	6.40 (64/1000)	4.30 (86/2000)	4.20 (42/1000)
gpt-5.5	3.50 (70/2000)	7.20 (72/1000)	4.00 (80/2000)	6.80 (68/1000)
kimi-k2.6	6.50 (130/2000)	5.70 (57/1000)	4.20 (84/2000)	5.20 (52/1000)
qwen3.5-122b-a10b	4.90 (98/2000)	5.30 (53/1000)	5.65 (113/2000)	5.30 (53/1000)
B. Same options in both comparisons				
29 questions in the answer addition setting; 21 in the incorrect-option replacement setting				
claude-opus-4-6	0.00	6.03	0.00	4.76
deepseek-v4-pro	6.55	9.08	11.90	10.71
gemini-3.1-pro-preview	6.90	10.34	4.76	4.76
glm-5.1	7.82	13.74	9.52	3.57
gpt-5.5	4.02	9.66	9.52	10.71
kimi-k2.6	2.30	3.74	2.38	4.76
qwen3.5-122b-a10b	10.34	8.79	14.29	10.71

Three-round accuracy and answer agreement. We summarize three evaluation rounds for seven LLMs in the answer addition and incorrect-option replacement settings of RobustOpt-QA: fourteen model–protocol series, with 3,000 generation records reported per run. The identifiers do not verify unchanged underlying checkpoints or service configurations across rounds. Table 10 gives aggregate accuracy and three-round answer agreement. Every one of the 3,000 records per run is parseable. The reported accuracy is the mean over the three input conditions of a run; agreement is the fraction of the 3,000 (variant, condition) cells whose selected answer meaning is identical in all three rounds, using the same resolved option meanings as $\mathrm{ACR}_{\mathrm{QA}}$. Agreement therefore measures repeatability on unchanged inputs and is not retention on perturbed inputs; it cannot be equated with semantic retention or $1 - \mathrm{ACR}_{\mathrm{QA}}$.

Table 10: Three-round accuracy and answer agreement for seven LLMs. Accuracy and agreement are percentages; ranges are in pp. Add and Replace denote the answer addition and incorrect-option replacement settings of RobustOpt-QA, respectively. Accuracy is the mean over a run's three input conditions; agreement is the share of the 3,000 (variant, condition) cells per run with the same selected answer meaning in all three rounds. Ranges can differ by 0.01 pp from differences between the displayed rounded accuracies.

API identifier	Protocol	Round 1	Round 2	Round 3	Range (pp)	Agreement (%)
gemini-3.1-pro-preview	Add	95.17	94.80	95.07	0.37	97.00
	Replace	95.67	95.43	95.67	0.23	97.17
kimi-k2.6	Add	89.10	88.70	89.00	0.40	91.53
	Replace	90.57	90.17	90.47	0.40	93.63
deepseek-v4-pro	Add	89.10	88.83	89.20	0.37	91.50
	Replace	89.70	90.30	90.33	0.63	92.97
glm-5.1	Add	88.40	87.53	87.20	1.20	91.03
	Replace	90.13	89.23	89.20	0.93	92.77
claude-opus-4-6	Add	90.77	90.90	90.77	0.13	98.83
	Replace	92.10	92.27	92.23	0.17	99.27
gpt-5.5	Add	87.17	87.07	90.90	3.83	89.53
	Replace	89.53	89.37	92.67	3.30	90.70
qwen3.5-122b-a10b	Add	86.90	86.67	86.97	0.30	91.90
	Replace	88.97	88.50	88.93	0.47	92.80

Accuracy and answer agreement. Twelve series have reported accuracy ranges of 0.13–1.20 pp; gpt-5.5 has larger ranges of 3.83 pp in the answer addition setting and 3.30 pp in the incorrect-option replacement setting. Small aggregate changes can coexist with differences in individual responses: deepseek-v4-pro's ranges are 0.37 and 0.63 pp, while reported agreement is 91.50% and 92.97%. Neither aggregate stability nor agreement alone establishes correctness under changed descriptions.

Joint context-and-option comparisons. The reported contrast is $\Delta_{\text{joint}} = A(P) - A(\text{Origin4})$, where P is P5 in the answer addition setting and Perturbed4R in the incorrect-option replacement setting. Negative values indicate lower perturbed accuracy, opposite to the positive-drop convention in Sec. 4.2. These contrasts change both context and options, unlike the fixed-option comparisons Origin5 versus P5 and Control4R versus Perturbed4R.

Table 11: Joint accuracy contrasts $A(P) - A(\text{Origin4})$ on RobustOpt-QA, in pp. Negative values indicate a decrease. Both context and answer-set changes are included. No model–protocol series changes sign between positive and negative values; deepseek-v4-pro has one reported zero in the answer addition setting.

	Answer addition setting Origin4 to P5			Incorrect-option replacement setting Origin4 to Perturbed4R		
API identifier	Round 1	Round 2	Round 3	Round 1	Round 2	Round 3
gemini-3.1-pro-preview	−1.60	−1.20	−1.50	−0.80	−0.30	−0.90
kimi-k2.6	−0.50	−1.60	−0.70	+1.90	+2.70	+3.10
deepseek-v4-pro	−1.60	0.00	−0.70	+1.60	+1.40	+1.10
glm-5.1	−0.40	−2.40	−1.50	+0.20	+1.20	+2.20
claude-opus-4-6	−1.30	−1.30	−1.00	+1.10	+0.90	+0.90
gpt-5.5	−2.20	−1.40	−0.20	+1.30	+1.60	+3.30
qwen3.5-122b-a10b	−2.10	−2.50	−1.50	+1.90	+1.70	+1.90

Table 11 has 20 negative contrasts and one zero in the answer addition setting. In the incorrect-option replacement setting, six identifiers have positive contrasts in every round, while gemini-3.1-pro-preview has negative contrasts throughout. This observed sign consistency does not establish a fixed effect or statistical equivalence. Positive contrasts in the incorrect-option replacement setting also do not show that context edits help, because the options change as well. These rounded summaries are descriptive; paired inference requires aligned individual records, verified denominators, and an analysis accounting for shared contexts (App. E).

F.5 QA PROMPT TEMPLATES AND RESPONSE FORMATS

The templates below document the baseline prompting procedure for the answer addition and incorrect-option replacement settings of RobustOpt-QA. Each input supplies one context, one question, and complete numbered answer options. Labels start at zero, so `{max_index}` is 3 for four options and 4 for five. Reference answers and review annotations are excluded from the model-visible input. The self-checking configuration uses a different response procedure, described in App. F.3.

For local models using chat-formatted inputs, the system message is `You are a helpful assistant.` The model continues an assistant message initialized with `FINAL_ANSWER:`. ORLM uses plain-text completion: the same prefix follows the user template, with no separate system message.

Shared QA User Template

```
Below is an operations research multiple-choice question.
Use the scenario context to identify the optimization problem and select
  ↪ the
single complete candidate option that best answers the question.

Each numbered option is one indivisible candidate answer. If an option
  ↪ itself
contains several activities, variables, parameters, or system elements,
  ↪ they
```

```
still form one option. Never return several labels and never split one
  ↪ option
into a list of labels. This is a classification task only: do not
  ↪ formulate a
complete mathematical model, write solver code, or run a solver.

Return exactly one label from 0 to {max_index}. Put the label
  ↪ immediately after
FINAL_ANSWER:. Do not put reasoning before the label.

## Scenario context
{context}

## Question
{question}

## Complete candidate options
{options}

The labels are zero-based.
```

The API adapter uses the system message below and appends the JSON suffix to the shared user template, without an assistant prefill. It also requests a JSON response object through the API. Scoring uses the selected label in the `answer` field; the `reasoning` field is not graded.

API System Message and User Suffix

System message.

```
You answer operations research multiple-choice questions. Return only
  ↪ one valid JSON object and no markdown.
```

User-message suffix, appended to the shared template.

```
Return only one JSON object in exactly this form:
{"answer": 0, "reasoning": "one short sentence"}

The JSON field "answer" must be exactly one legal zero-based option
  ↪ label.
Do not return an option list, markdown, solver code, or any text outside
  ↪ JSON.
```

The API template retains conflicting `FINAL_ANSWER:` and JSON-only instructions. These baseline templates are reproduced verbatim; exact requests and settings for each run require the corresponding execution records (App. H.1).

G RobustOpt-Coding: Experimental Protocol and Results

This section reports numerical performance, formulation-component differences, repeated-generation comparisons, and execution-guided mitigation on the stated Coding panels.

G.1 Coding cohorts and complete results

The frozen inventory contains 2,061 records, of which 200 originals and 1,705 unique invariance variants form the primary numerical panel. The 156 exclusions comprise 84 variants without inverse-unit mappings, 30 changed-target controls without revised references, 36 CLEAR controls, and six duplicate invariance positions. Exclusions are not failures. Supplementary formulation checks use their own references and do not alter the numerical panel. Each problem has 5–13 eligible variants.

All fourteen systems have three recorded rounds. The seven local systems cover all 2,061 inputs in every round. The seven API systems use the full panel in R1 and the same frozen 600-input sample in R2/R3: every original and two non-original records sampled without replacement per problem. The sample contains 571 primary-panel positions (200 originals and 371 variants). Tables 2 and 17 report R1 numerical results and formulation diagnostics, respectively; Tables 12 and 13 give both for R2/R3. Sample fractions describe the observed sample and are not full-population execution counts. Each round is completed without stopping after an earlier success.

Numerical endpoint and system identities. Table 2 reports R1 numerical results; the tables below report R2 and R3. Original accuracy uses observed originals; Pass and F_{num} use the same eligible variants of numerically passing originals. Technical absence remains unknown. Gemini R1 has 199 observed originals and 1,696 valid paired variants owing to seven response gaps. GPT-5.5 R1 has 1,704 paired variants; its missing variant belongs to a non-passing original, so the 1,162 conditional denominator is unchanged. No-code terminal responses and observed execution failures remain operational non-passes, distinct from technical absence and from proved mathematical differences.

The Coding Claude row is Opus 4.6, and the QA Claude row now uses the same Opus 4.6 identity. GPT-5.5 R1 uses independent tool-free Codex sessions at high reasoning; R2/R3 use the recorded LLM Melon API route. The local R1 tracks use greedy decoding, while their R2/R3 tracks use sampled decoding. Consequently, the three rounds are not treated as identically configured independent trials, and no iid pass@3 claim is made. ORLM, StepORLM, and SIRL use COPT; OptMATH uses Gurobi, and LLMOPT uses its native Pyomo/GLPK implementation. Qwen's R1 numerical row uses the recorded COPT API V2 compatibility layer and size-limited execution policy, with the generated source preserved. Shared inputs do not imply identical execution systems.

Table 12: Coding numerical results and formulation diagnostics in R2. Numerical cells give percentages and observed counts; * marks API results on the frozen 600-input sample, while local systems use the complete panel. Pass and F_{num} condition on this round's passing originals. Var./Obj./Cons. use the separate mathematical cohort D defined in App. G.4; $\geq$ denotes a supported lower bound, rounded down, and N/A an empty cohort. One output may have errors in several components.

	Model	$A_0 \uparrow$	Pass $\uparrow$	$F_{\mathrm{num}} \downarrow$	Var.	Obj.	Cons.
Fine-tuned Models	LLMOPT (14B)	25.00 (50/200)	62.22 (252/405)	37.78 (153/405)	$\geq$0.00 (0/1)	$\geq$0.00 (0/1)	100.00 (1/1)
	OptMATH (7B)	6.00 (12/200)	20.39 (21/103)	79.61 (82/103)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)
	OptMATH (32B)	39.50 (79/200)	73.78 (467/633)	26.22 (166/633)	$\geq$12.96 (7/54)	$\geq$3.70 (2/54)	$\geq$0.00 (0/54)
	ORLM (8B)	32.00 (64/200)	58.00 (308/531)	42.00 (223/531)	$\geq$4.49 (4/89)	$\geq$3.37 (3/89)	$\geq$0.00 (0/89)
	SIRL (7B)	19.50 (39/200)	55.81 (173/310)	44.19 (137/310)	$\geq$0.00 (0/25)	$\geq$28.00 (7/25)	$\geq$0.00 (0/25)
	SIRL (32B)	41.00 (82/200)	84.59 (549/649)	15.41 (100/649)	$\geq$0.00 (0/31)	$\geq$12.90 (4/31)	$\geq$16.12 (5/31)
	StepORLM (8B)	35.50 (71/200)	81.02 (461/569)	18.98 (108/569)	$\geq$0.00 (0/24)	$\geq$0.00 (0/24)	$\geq$0.00 (0/24)
Generic Models	Claude Opus 4.6*	51.50 (103/200)	83.51 (162/194)	16.49 (32/194)	$\geq$0.00 (0/5)	$\geq$0.00 (0/5)	$\geq$0.00 (0/5)
	DeepSeek V4 Pro*	42.50 (85/200)	61.39 (97/158)	38.61 (61/158)	$\geq$0.00 (0/8)	$\geq$0.00 (0/8)	$\geq$0.00 (0/8)
	Gemini 3.1 Pro*	62.31 (124/199)	90.22 (203/225)	9.78 (22/225)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)
	GLM-5.1*	47.00 (94/200)	70.76 (121/171)	29.24 (50/171)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)
	GPT-5.5*	69.00 (138/200)	89.76 (228/254)	10.24 (26/254)	$\geq$0.00 (0/1)	$\geq$0.00 (0/1)	$\geq$0.00 (0/1)
	Kimi K2.6*	41.50 (83/200)	57.79 (89/154)	42.21 (65/154)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)
	Qwen 3.5 122B*	17.00 (34/200)	25.42 (15/59)	74.58 (44/59)	33.33 (3/9)	$\geq$44.44 (4/9)	33.33 (3/9)

Accuracy decomposition and joint survival across all systems. Table 14 decomposes problem-balanced accuracy on the fixed panel. Write $x_i = c_{i0}$ and $y_i = K_i^{-1} \sum_k c_{ik}$. Loss $L = N^{-1} \sum_i x_i(1 - y_i)$ and recovery $G = N^{-1} \sum_i (1 - x_i) y_i$ are accuracy contributions satisfying $A_T - A_0 = G - L$. This problem-equal weighting follows Sec. 4.2; Table 2's Pass instead pools alternatives.

For Gemini and GPT-5.5, entries affected by missing outcomes show the lower endpoint of each reported completion range. These endpoints need not correspond to the same assignment of missing labels. Gemini's fixed-panel A_0 uses 200 original positions and differs from its observed-original $123/199$ in Table 2; its joint-survival count is fixed at 57, or $57/123$ conditional on observed original success. All observed original-success cohorts have complete alternative labels.

All fourteen systems exhibit losses and recoveries under every feasible assignment of missing labels. For SIRL (32B), losses of 4.653 points and recoveries of 4.906 points leave only a 0.253-point net

Table 13: Coding numerical results and formulation diagnostics in R3. Numerical cells give percentages and observed counts; * marks API results on the frozen 600-input sample, while local systems use the complete panel. Pass and F_{num} condition on this round's passing originals. Var./Obj./Cons. use the separate mathematical cohort D defined in App. G.4; $\geq$ denotes a supported lower bound, rounded down, while unmarked component cells have complete attribution within D. One output may have errors in several components.

	Model	$A_0 \uparrow$	Pass $\uparrow$	$F_{\text{num}} \downarrow$	Var.	Obj.	Cons.
Fine-tuned Models	LLMOPT (14B)	22.00 (44/200)	57.63 (204/354)	42.37 (150/354)	0.00 (0/5)	0.00 (0/5)	100.00 (5/5)
	OptMATH (7B)	7.50 (15/200)	16.00 (20/125)	84.00 (105/125)	$\geq$0.00 (0/13)	$\geq$30.76 (4/13)	$\geq$0.00 (0/13)
	OptMATH (32B)	43.00 (86/200)	72.15 (487/675)	27.85 (188/675)	$\geq$0.00 (0/65)	$\geq$3.07 (2/65)	$\geq$4.61 (3/65)
	ORLM (8B)	31.50 (63/200)	57.83 (299/517)	42.17 (218/517)	$\geq$0.00 (0/100)	$\geq$4.00 (4/100)	$\geq$0.00 (0/100)
	SIRL (7B)	17.50 (35/200)	67.16 (182/271)	32.84 (89/271)	$\geq$0.00 (0/10)	$\geq$60.00 (6/10)	$\geq$0.00 (0/10)
	SIRL (32B)	42.50 (85/200)	82.32 (568/690)	17.68 (122/690)	$\geq$0.00 (0/34)	$\geq$5.88 (2/34)	$\geq$23.52 (8/34)
	StepORLM (8B)	39.50 (79/200)	78.23 (503/643)	21.77 (140/643)	$\geq$0.00 (0/35)	$\geq$5.71 (2/35)	$\geq$0.00 (0/35)
Generic Models	Claude Opus 4.6*	52.50 (105/200)	86.22 (169/196)	13.78 (27/196)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)	$\geq$0.00 (0/2)
	DeepSeek V4 Pro*	42.50 (85/200)	62.66 (99/158)	37.34 (59/158)	50.00 (1/2)	0.00 (0/2)	50.00 (1/2)
	Gemini 3.1 Pro*	62.44 (123/197)	81.33 (183/225)	18.67 (42/225)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)
	GLM-5.1*	38.50 (77/200)	64.96 (89/137)	35.04 (48/137)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)
	GPT-5.5*	71.50 (143/200)	94.68 (249/263)	5.32 (14/263)	$\geq$66.66 (2/3)	$\geq$0.00 (0/3)	$\geq$0.00 (0/3)
	Kimi K2.6*	37.00 (74/200)	58.96 (79/134)	41.04 (55/134)	$\geq$0.00 (0/1)	$\geq$0.00 (0/1)	$\geq$0.00 (0/1)
	Qwen 3.5 122B*	20.60 (41/199)	41.10 (30/73)	58.90 (43/73)	14.29 (1/7)	28.57 (2/7)	57.14 (4/7)

Table 14: Full-panel R1 numerical accuracy, loss/recovery contributions, and joint survival. A_0, A_T, and R are percentages; L and G are percentage-point contributions. $n_{\text{all}}/200$ counts originals and all their eligible alternatives passing. Entries affected by missing outcomes show the reported lower endpoints; see text.

Model	A_0	A_T	L (pp)	G (pp)	R	$n_{\text{all}}/200$
LLMOPT (14B)	26.50	22.66	6.570	2.726	75.21	14/200
OptMATH (7B)	14.00	14.16	4.847	5.004	65.38	3/200
OptMATH (32B)	47.00	40.18	9.516	2.691	79.75	41/200
ORLM (8B)	36.00	30.19	11.549	5.740	67.92	15/200
SIRL (7B)	20.50	17.59	6.366	3.460	68.95	10/200
SIRL (32B)	42.00	42.25	4.653	4.906	88.92	50/200
StepORLM (8B)	41.50	39.39	5.196	3.088	87.48	38/200
Claude Opus 4.6	48.50	51.55	5.548	8.598	88.56	68/200
DeepSeek V4 Pro	42.00	38.86	17.899	14.756	57.38	3/200
Gemini 3.1 Pro	61.50	61.52	7.690	7.715	86.79	57/200
GLM-5.1	42.50	41.86	12.989	12.350	69.44	14/200
GPT-5.5	70.50	67.92	6.647	4.067	90.57	97/200
Kimi K2.6	45.00	41.13	15.776	11.904	64.94	8/200
Qwen 3.5 122B	19.50	19.77	12.831	13.100	34.20	0/200

increase. Yet 34/84 originally passing problems fail on at least one tested alternative, compared with an alternative-pooled failure rate of 80/673. Aggregate accuracy and alternative-pooled retention therefore conceal different aspects of instability. All-alternative survival refers to the tested panel, and model-specific original-success sets do not support a common-cohort ranking.

Perturbation types on matched applicable problems. Each comparison in Table 15 uses the same originally passing problems for both edits within each LLM. Compared with partial reordering (O2), complete reversal (O4) has more failed problems for eight LLMs, the same number for two, and fewer for four. The ordering rules are defined in App. D.2. Equal counts need not indicate the same failures: Opus has 13/95 failures under each edit, with five problems changing in each direction.

The background comparisons change how closely an irrelevant fact resembles the problem. Unrelated background uses a different topic and roles. Topic-matched background shares the problem's topic and roles but uses numbers far from its numerical scale; this condition has more failures than unrelated background for ten LLMs and fewer for four. The third condition also brings the background numbers close to the problem's scale. Compared with topic-matched background with distant numbers, it has

more failures for six LLMs, the same number for one, and fewer for seven. All three add information that is independent of the current decisions; none adds a new constraint or changes the objective.

Table 15: R1 comparisons on the same originally passing problems for each LLM, where both edits apply. Higher / Equal / Lower count LLMs with more / the same / fewer failures under B than under A. The last column gives the range of problem counts across LLMs. Background topic and roles refer to resemblance to the problem; distant or similar numbers refer to its numerical scale. The 14 LLMs are not independent experimental replicates.

Comparison $A \to B$	Higher	Equal	Lower	n range
O2 → O4	8	2	4	28–135
Unrelated → same topic/roles	10	0	4	28–141
Distant → similar numbers	6	1	7	28–141
O2 → prose/structure	4	4	6	28–141
O2 → scope/condition	5	3	6	3–35
O2 → cross-table binding	7	2	5	3–33

The analysis covers 15 scoreable single types and seven combinations. Six O4 duplicates are excluded, leaving 194 applicable problems before conditioning on original success. Sparse types remain descriptive: temporal-scope (TM) has at most two original-success problems per system, and some combinations have at most one; empty denominators are N/A. No technical unknown falls in the observed-original-success cohorts used here. Numerical failures include code and execution failures.

G.2 Output checks and recorded execution settings

The scorer keeps code extraction, execution, solver status, numerical agreement, and mathematical evidence separate. For a finite-optimum reference, the numerical scoring rule requires successful execution of one model, a finite normalized objective, and OPTIMAL or LOCAL_OPTIMAL status, with

$$|\widehat{v}_i - v_i^\star| \le \max\{10^{-4}, 10^{-4}\max(1, |v_i^\star|)\}.$$

Known-infeasible references instead require matching INFEASIBLE execution and model statuses. Unit conversions must be supported before comparison. LOCAL_OPTIMAL is a diagnostic acceptance rule, not a global-optimality certificate. Strict OPTIMAL scoring for SIRL (32B) gives 71/200 passing originals and 75/590 conditional failures. Its recorded tighter numerical-tolerance sensitivity does not change the primary pass labels.

For SIRL (32B) in R1, the 80 failed variants of numerically passing originals comprise 51 reference-objective mismatches despite an optimal solver status, 18 runtime errors, and 11 infeasibility reports.

For completeness, the general additive tolerance template is

$$|\widehat{v}_i - v_i^\star| \le \varepsilon_{\text{abs}} + \varepsilon_{\text{rel}}\max\{1, |v_i^\star|\}, \tag{2}$$

This additive template differs from the maximum rule used for the reported numerical results. A supported all-check passage additionally requires the relevant mathematical checks. Variable renaming, redundant constraints, exact extensions, and auxiliary-variable elimination require valid correspondences; raw variable/constraint counts alone cannot determine equivalence. Uncertified nonlinear search cannot prove equivalence by failing to find a counterexample. Unknowns remain explicit in the present evidence tables.

The GPT-5.5 R1 and recorded Gemini COPT runs use COPT 8.0.6, a 120-second solver limit, 180-second wall limit, 185-second CPU limit, 4,096 MiB memory, and one CPU. Generated code is not repaired. SIRL (32B)'s checkpoint is `chenyitian-shanshu/SIRL-COPT32B`, revision `1bccf5d7bf054942317c600567c8f5f4303aa861`; its R1 configuration uses BF16, greedy decoding, eager attention, a 32,768-token context, and at most 4,096 new tokens on the recorded H100 environment. SIRL (7B)'s FP32 and LLMOPT's FP16 native Pyomo/GLPK settings remain as executed. Execution settings differ across systems; hardware specifications are taken from run metadata.

Formulation comparisons use explicit references and mathematical correspondences. A certificate may establish formulation equivalence or correctness of a complete optimization algorithm under its

stated assumptions, even when the recorded implementation fails to execute. Conversely, successful execution or objective agreement alone does not establish preservation of the reference decisions, objective, and constraints.

G.3 Accuracy across three recorded rounds

Table 16 summarizes aggregate scores across three recorded rounds. For a metric expressed in percent, let x_r denote its value in round r. We report the empirical expectation (arithmetic mean), sample variance, and standard deviation:

$$\widehat{\mathbb{E}}[x] = \bar{x} = \frac{1}{3}\sum_{r=1}^{3} x_r, \qquad s^2 = \frac{1}{2}\sum_{r=1}^{3}(x_r - \bar{x})^2, \qquad s = \sqrt{s^2}.$$

The mean is in percent, variance in squared percentage points, and standard deviation in percentage points. Calculations use exact fractions before rounding. These are descriptive moments of the three recorded rounds, not an iid variance estimate, standard error, or sampling-design variance.

Each model uses the same eligible, observed input records in all three rounds. Local systems retain the complete primary panel (200 originals and 1,705 variants); API systems restrict R1 as well as R2/R3 to the frozen sample (200 originals and 371 variants). Records with an unavailable outcome in any round, and variants whose original lacks a common outcome, are excluded from all three rounds of this comparison. Gemini therefore uses 197 originals and 364 variants, and Qwen uses 199 originals and 368 variants; no missing outcome is replaced by failure. Within this fixed input panel, Pass and F_{num} still condition on each round's passing originals, so their success cohorts may differ. API R1 values are recomputed on the common sample and can differ from Table 2's full-panel values.

Table 16: Three-round empirical expectation and dispersion on each model's common observed input panel. Mean denotes $\widehat{\mathbb{E}}[x]$, Var. denotes s^2 (pp^2), and SD denotes s (pp). * marks frozen-sample API summaries. Formulation-component cells use separate certified cohorts and are not included in these moment calculations.

	A_0			Pass			F_{num}		
Model	Mean	Var.	SD	Mean	Var.	SD	Mean	Var.	SD
LLMOPT (14B)	24.50	5.25	2.29	64.95	81.03	9.00	35.05	81.03	9.00
OptMATH (7B)	9.17	18.08	4.25	34.07	760.63	27.58	65.93	760.63	27.58
OptMATH (32B)	43.17	14.08	3.75	74.51	7.84	2.80	25.49	7.84	2.80
ORLM (8B)	33.17	6.08	2.47	60.65	22.43	4.74	39.35	22.43	4.74
SIRL (7B)	19.17	2.33	1.53	63.21	41.18	6.42	36.79	41.18	6.42
SIRL (32B)	41.83	0.58	0.76	85.01	8.52	2.92	14.99	8.52	2.92
StepORLM (8B)	38.83	9.33	3.06	82.10	20.33	4.51	17.90	20.33	4.51
Claude Opus 4.6*	50.83	4.33	2.08	85.96	5.41	2.33	14.04	5.41	2.33
DeepSeek V4 Pro*	42.33	0.08	0.29	58.54	36.82	6.07	41.46	36.82	6.07
Gemini 3.1 Pro*	62.27	0.09	0.29	86.84	24.34	4.93	13.16	24.34	4.93
GLM-5.1*	42.67	18.08	4.25	67.89	8.40	2.90	32.11	8.40	2.90
GPT-5.5*	70.33	1.58	1.26	91.99	6.19	2.49	8.01	6.19	2.49
Kimi K2.6*	41.17	16.08	4.01	61.74	34.42	5.87	38.26	34.42	5.87
Qwen 3.5 122B*	19.10	3.28	1.81	33.91	62.67	7.92	66.09	62.67	7.92

The three-round spread reflects both output variability and differences in settings: local R1 uses greedy decoding, R2/R3 use sampled decoding, and GPT-5.5 changes access route. App. G.5 instead compares unchanged inputs under the same recorded generation settings in R2/R3. Since $F_{\text{num}} = 100 - \text{Pass}$ on the observed cohort, their means sum to 100 and their variances and standard deviations coincide.

G.4 CHECKING VARIABLES, OBJECTIVES, AND CONSTRAINTS

We check whether the models generated from rewritten descriptions still represent the original problem. Tables 17, 12, and 13 summarize cases where the original output represents the reference problem correctly but the rewritten input produces a model with an established mathematical error. For each LLM and round, D counts these currently certified rewritten inputs after deduplication. It does not count all outputs checked or all numerically passing outputs. Unresolved original or variant checks are excluded; these cohorts do not constitute a complete mathematical audit.

The checks distinguish errors in the domains of the mapped decision variables, including integrality (Var.); in the objective (Obj.); and in constraints and feasibility bounds (Cons.). Bounds on auxiliary variables are classified by the constraints they induce on the reference decisions. They do not always identify every component responsible for a confirmed error. We therefore report the minimum confirmed count L_c for each component c. A cell marked $\geq$ gives $100L_c/D$, rounded down; an unmarked cell is exact within these D cases. One output may have errors in several components. A zero lower bound means that no error in that component has been confirmed, not that the component is correct. Rates are undefined when $D = 0$.

For SIRL (7B), the 22 cases are rewritten inputs with confirmed formulation errors. The checks identify an incorrect objective in at least ten outputs, incorrect allowed values for variables in at least two, and incorrect constraints in at least two. One output can contain several of these errors. Also, confirming that a model is wrong does not always identify every component responsible. The component counts therefore need not add up to 22. The denominator includes only these confirmed error cases, rather than all generated models, so the percentages do not measure how often the LLM generates an incorrect model overall.

Table 17: R1 formulation checks. Cells show component-error percentages (count/D) within the currently certified cohort. $\geq$ marks a supported lower bound, rounded down; unmarked cells have complete attribution within D. One output may have errors in several components.

	Model	Var.	Obj.	Cons.
Fine-tuned Models	LLMOPT (14B)	0.00 (0/4)	0.00 (0/4)	100.00 (4/4)
	OptMATH (7B)	$\geq$5.88 (1/17)	$\geq$0.00 (0/17)	$\geq$23.52 (4/17)
	OptMATH (32B)	$\geq$0.00 (0/51)	$\geq$9.80 (5/51)	$\geq$3.92 (2/51)
	ORLM (8B)	$\geq$4.54 (3/66)	$\geq$6.06 (4/66)	$\geq$10.60 (7/66)
	SIRL (7B)	$\geq$9.09 (2/22)	$\geq$45.45 (10/22)	$\geq$9.09 (2/22)
	SIRL (32B)	$\geq$0.00 (0/30)	$\geq$3.33 (1/30)	$\geq$10.00 (3/30)
	StepORLM (8B)	$\geq$4.16 (1/24)	$\geq$12.50 (3/24)	$\geq$4.16 (1/24)
Generic Models	Claude Opus 4.6	$\geq$22.72 (5/22)	$\geq$0.00 (0/22)	$\geq$4.54 (1/22)
	DeepSeek V4 Pro	$\geq$5.00 (1/20)	$\geq$5.00 (1/20)	$\geq$0.00 (0/20)
	Gemini 3.1 Pro	$\geq$0.00 (0/8)	$\geq$0.00 (0/8)	$\geq$12.50 (1/8)
	GLM-5.1	$\geq$13.63 (3/22)	$\geq$4.54 (1/22)	$\geq$4.54 (1/22)
	GPT-5.5	$\geq$11.11 (1/9)	$\geq$0.00 (0/9)	$\geq$0.00 (0/9)
	Kimi K2.6	$\geq$15.38 (4/26)	$\geq$0.00 (0/26)	$\geq$3.84 (1/26)
	Qwen 3.5 122B	60.00 (6/10)	30.00 (3/10)	10.00 (1/10)

Comparisons require explicit variable correspondences and the selected reference's domain and interpretation assumptions. A certificate can concern a complete optimization algorithm, including its intermediate stages, rather than only the last solver model. Proofs using recovered model blocks or idealized subproblem solves retain those conditions and do not establish successful execution. LLMOPT's R2 constraint difference uses the stated reference interpretation for `optmath_138`. Its four R1 and five R3 cases instead impose independent archival counts as current operating constraints, while preserving the reference variable domains and objectives. DeepSeek's R3 diagnoses use the frozen continuous-LP reference for `industryor_066` and an auxiliary-variable projection for `industryor_040`. The former variant has a recorded API execution error; its formulation diagnosis comes from the complete generated source. A feasible-set difference need not change the optimum or identify every responsible component. Unit conversions must preserve decision domains;

changed-target controls use revised references and remain outside invariance results. These checks do not change eligibility for the numerical results.

G.5 VARIABILITY WITHOUT DESCRIPTION CHANGES

Purpose and repeated inputs. We measure variation without perturbations by comparing fresh outputs for the 200 original RobustOpt-Coding inputs in generation rounds two and three (R2 and R3). Each problem's description, data, prompt, and recorded generation settings are unchanged. We use these two rounds because local R1 uses different decoding settings and GPT-5.5 uses a different access route in R1.

Table 18: No-perturbation baseline: numerical outcomes for unchanged originals in R2 and R3. Arrows run from R2 to R3. Changed outcome counts switches in either direction, shown as percentage (count/N). N excludes missing pairs.

Model	N	Pass in both runs	Pass→fail	Fail→pass	Fail in both runs	Changed outcome (%)
LLMOPT (14B)	200	30	20	14	136	17.00 (34/200)
OptMATH (7B)	200	4	8	11	177	9.50 (19/200)
OptMATH (32B)	200	69	10	17	104	13.50 (27/200)
ORLM (8B)	200	44	20	19	117	19.50 (39/200)
SIRL (7B)	200	23	16	12	149	14.00 (28/200)
SIRL (32B)	200	72	10	13	105	11.50 (23/200)
StepORLM (8B)	200	61	10	18	111	14.00 (28/200)
Claude Opus 4.6	200	93	10	12	85	11.00 (22/200)
DeepSeek V4 Pro	200	53	32	32	83	32.00 (64/200)
Gemini 3.1 Pro	197	104	19	19	55	19.29 (38/197)
GLM-5.1	200	60	34	17	89	25.50 (51/200)
GPT-5.5	200	133	5	10	52	7.50 (15/200)
Kimi K2.6	200	39	44	35	82	39.50 (79/200)
Qwen 3.5 122B	199	15	19	26	139	22.61 (45/199)

Counting changes. Table 18 counts pass→fail and fail→pass transitions among all originals observed in both runs, without selecting earlier successes. Their sum divided by N gives the outcome-change rate. Twelve LLMs have all 200 paired outcomes; Gemini has 197 and Qwen 199. Missing outcomes are excluded, not failures. All originals were requested in both rounds, so no variant-sampling weights are needed. "Pass" uses the numerical checks in App. G.2, not a certificate of formulation equivalence.

Observed change rates range from 7.50% to 39.50%. For SIRL (32B), 23/200 outcomes change (11.50%), although accuracy moves only from 82/200 to 85/200. This is a baseline for numerical outcome variation in the repeated generation-and-execution process, not code-text differences or isolated LLM randomness. Two runs do not precisely estimate long-run variability, and API logs cannot verify independent samples or an unchanged backend.

Comparison with rewritten inputs. We separately select the problems that passed in R2, denoted S_2, and compare their unchanged and rewritten inputs in R3. For SIRL (32B), ten of these 82 problems fail on the unchanged input: a no-perturbation failure rate of 12.20%. Rewritten inputs have a problem-averaged failure rate of 14.71%, a difference of 2.51 percentage points (Table 19). The 11.50% baseline above includes both directions of change across all 200 originals; it cannot be subtracted from this conditional variant failure rate.

Averaging variants. Local systems first average eligible variants within each problem, then average over S_2, weighting problems equally. API systems sample two non-original records per problem. To account for sampling, let J_i count all non-original records eligible for sampling, K_i count primary-panel variants in the full inventory, and s_i contain the two sampled records. With $c_{ik,3}$ indicating a numerical pass in R3, the variant pass-rate estimate is

$$\widehat{R}_{T,3|2} = \frac{1}{|S_2|} \sum_{i \in S_2} \frac{J_i}{2K_i} \sum_{k \in s_i} \mathbf{1}\{k \text{ is eligible}\}\, c_{ik,3}. \tag{3}$$

A problem with no eligible variant sampled remains in S_2 with zero observed contribution. Unchanged originals need no sampling weights. Table 19 gives pass rates; its difference also equals rewritten-minus-unchanged failure rates. In contrast, Table 2 weights variants equally.

Table 19: R3 pass rates on problems that passed in R2. Difference is unchanged minus rewritten; variant rates weight problems equally. * marks a stratified-sample estimate. Ranges complete missing outcomes, not confidence intervals.

Model	R2 passes	Unchanged (%)	Rewritten (%)	Difference (pp)
LLMOPT (14B)	50	60.00	61.09	-1.09
OptMATH (7B)	12	33.33	21.35	11.98
OptMATH (32B)	79	87.34	76.03	11.31
ORLM (8B)	64	68.75	57.50	11.25
SIRL (7B)	39	58.97	62.62	-3.65
SIRL (32B)	82	87.80	85.29	2.51
StepORLM (8B)	71	85.92	81.87	4.05
Claude Opus 4.6*	103	90.29	86.57	3.72
DeepSeek V4 Pro*	85	62.35	59.83	2.52
Gemini 3.1 Pro*	124	83.87–84.68	80.47	3.40–4.21
GLM-5.1*	94	63.83	59.97	3.86
GPT-5.5*	138	96.38	93.86	2.52
Kimi K2.6*	83	46.99	54.08	-7.09
Qwen 3.5 122B*	34	44.12	42.67	1.45

Unchanged inputs have higher pass rates for eleven LLMs and lower rates for LLMOPT, SIRL (7B), and Kimi K2.6. OptMATH (7B)'s comparison has only twelve previously passing problems. Gemini's 124 R2 passes yield 104 repeat passes, 19 failures, and one unknown; its range completes that unknown. Table 18 instead excludes missing pairs, retaining 123 R2 passes among 197 problems. The range excludes sampling uncertainty. These comparisons do not establish a statistically significant or isolated causal effect of rewriting.

G.6 Selecting a second model after execution failure

Strategy: select between two candidates using execution outcomes. For each input, we obtain two candidates, A and B, using Silra's `deepseek-v4-pro-0813` endpoint with the same prompt, high reasoning, and thinking enabled. Retain A unless it has a runtime error, syntax error, process failure, or no code. Select B only if it exits normally, produces one model without an exception, and returns OPTIMAL, LOCAL_OPTIMAL, INFEASIBLE, or UNBOUNDED. Selection uses execution outcomes only, never reference values or numerical pass labels. The rule was fixed before processing the 1,128 new candidate positions. Both candidates were obtained in advance, so the comparison uses two candidates instead of one and does not measure on-demand retry cost or latency. COPT 8.0.6 uses the limits in App. G.2, without code repair.

Sample and comparison. We evaluate DeepSeek V4 Pro on 600 frozen inputs: all 200 originals and two non-original records sampled without replacement per problem. The eligibility rules in App. G yield 571 scored inputs. The primary comparison fixes the 107 problems whose original passes under candidate A; 106 contribute 197 eligible sampled variants. Newly passing originals do not enlarge this denominator. Candidate A is generated for this experiment; it is not the R1 output used in Table 2.

Paired outcomes. Table 20 shows passage increasing from 168/197 (85.28%) to 183/197 (92.89%): 15 of 29 failures recover across 14 problems, 14 remain, and all 107 original successes are retained. Every recovery replaces an A runtime error; the remaining failures comprise eight runtime errors, five OPTIMAL outputs with objective mismatches, and one local-infeasibility outcome. B alone passes 324/571, below A's 330/571, so the gain comes from selecting complementary outputs. Zero regressions also follow from replacing only A outcomes that already fail numerical scoring. Under strict OPTIMAL scoring on the same 197 variants, passage increases from 155/197 to 162/197 (+3.55 pp); tighter objective tolerance leaves the main 168-to-183 result unchanged.

Table 20: Execution-error fallback: recovered numerical passes and remaining failures. Pass columns show counts and percentages within each row. The primary robustness comparison is separated from the broader descriptive results.

Evaluation set	A passes	Fallback passes	Recovered failures	Remaining failures
Primary: variants of problems whose original passes under A				
Perturbed inputs ($n = 197$)	168/197 (85.28%)	183/197 (92.89%)	**15**	**14**
Context: original inputs and the full scoreable sample				
Original inputs ($n = 200$)	107/200 (53.50%)	124/200 (62.00%)	17	76
All scored inputs ($n = 571$)	330/571 (57.79%)	372/571 (65.15%)	42	199

The primary row uses a fixed original-success cohort. The 197 variants and 200 originals are subsets of the 571-input panel; rows must not be added. Passes denote numerical outcomes.

Scope and development exposure. Of 1,200 candidate positions, 72 reuse outputs from a pilot experiment covering 36 inputs across 12 problems; all positions retain their first terminal outcome. Excluding those problem groups gives 158/186 versus 172/186. Outside the prior intervention list, the comparison is 92/104 versus 96/104, but every problem had prior baseline exposure, so this is not an untouched holdout. These observed-sample gains concern execution-related numerical robustness with additional candidate budget. They do not establish semantic or mathematical correctness: for example, B's reference-matching LOCAL_OPTIMAL output on `optmath_148__00` has an unbounded captured model; this case lies outside the primary 197-variant cohort.

Main finding and future work. Selecting a second candidate recovers 15 of the 29 failures in the primary comparison. This helps retain some successes after rewriting, but the selection rule checks whether a model runs, not whether it represents the requirements correctly. Future strategies should also check which values the variables allow, what the objective optimizes, and whether all required constraints are represented. Comparisons should use the same number of candidates for each strategy, include more LLMs, and use problems not seen during development. The present experiment does not evaluate these additional checks.

G.7 Representative Coding prompts

We show two Coding prompt templates: the shared COPT system instruction for general-purpose LLMs and LLMOPT's Pyomo user template. The former takes the public input as its user message; the latter uses `You are a helpful assistant.` as its system message, with `{Question}` replaced by the public input. The protocol materials include the complete templates and execution settings.

Shared Coding Prompt (System, COPT)

```
You are an expert in mathematical optimization modeling and Python
  ↪ programming.
Formulate the optimization problem faithfully and provide one complete,
  ↪ executable Python program using coptpy. Include your mathematical
  ↪ formulation as Python comments. Return exactly one Python code block
  ↪ in the final answer.
Preserve all data, objective terms, constraints, and variable domains
  ↪ stated in the question. If the question supplies files, write code
  ↪ that reads the complete files from the given relative paths; the
  ↪ previews are not the complete dataset.
Create the COPT environment with cp.Envr(), store the optimization model
  ↪ in a variable named model, and call model.solve(). Print the solver
  ↪ status and, when available, the objective value and decision
  ↪ variable values.
```

```
Do not install packages, access the network, or substitute a different
  ↪ solver. The program will be executed separately in an environment
  ↪ with coptpy and the public data files.
```

LLMOPT Coding Prompt (User, Pyomo)

````
The following is an optimization problem. You need to write the
  ↪ corresponding Pyomo code based on the problem description and
  ↪ information provided.
The problem description is as follows:

```
{Question}
```
Please write the corresponding Pyomo code. Please add `from
  ↪ pyomo.environ import *` at the beginning of your code (You can add
  ↪ other `import` as well). Please print the optimal solution and the
  ↪ value of the objective function. Please do not output the running
  ↪ log. You need to write it in the form of a class and add a main
  ↪ function:

```python
[write your code here]
```
````

H Shared Reproducibility and Controls

This section records protocol details and controls that apply across QA and Coding.

H.1 System identification and reproducible execution

A system identifier consists of the checkpoint or API model version, prompts, solver interface, tool permissions, and inference budget. Reproduced models additionally require the base checkpoint, training data, training procedure, and checkpoint hash. Coding reports all fourteen systems in three rounds, with explicitly different full-panel and frozen-sample coverage. Claude Opus 4.6 is the Coding identity; the QA Claude row now uses the same Opus 4.6 identity. The QA comparison includes fourteen LLMs. Differences in version names and settings are not silently treated as the same evaluated system. The results do not include a complete evaluation combining numerical and formulation checks.

Run records should include the benchmark version, input and reference hashes, chat template, prompts, sampling settings, output limit, timeout, solver limits, retry budget, relevant hardware, and raw outputs. Within-run input comparisons use the same recorded settings and fresh sessions. QA rates condition on jointly parseable responses; Coding counts observed no-code and execution failures as non-passes, while technical absence remains unknown (Apps. F and G). Unsupported public-file formats require explicit interface-coverage reporting rather than dropping tasks from only one condition.

H.2 Controls and ablations

The experiments do not isolate the effect of output budget, compare background additions with exactly matched lengths and insertion positions, or evaluate every combined edit alongside both constituent edits on the same problems. CLEAR records repeat the original input and do not control for added text length. Repetition comparisons require fresh generations rather than copied responses.

Changed-target evaluation requires independently validated revised references; these controls remain outside the invariance results (App. G). Such an evaluation must report correctness on both the

Table 21: Additional analyses and their purposes.

Analysis	Purpose	Reference
Repeated unchanged inputs	Compare repetition with description changes	Apps. F.4, G.5
Formulation-component analysis	Distinguish objective agreement from formulation fidelity	App. G.4
Preliminary defenses	Evaluate self-checking and execution fallback	Apps. F.3, G.6

original and revised problems: an unchanged optimum can be correct for both, while a changed output alone does not demonstrate correct adaptation.

The reported group comparisons do not isolate the effect of fine-tuning; that comparison requires matched backbones, interfaces, and inference budgets. Defense comparisons must account for the complete call and token cost. Improved execution alone does not establish improved formulation fidelity.

I Evaluation Coverage and Development Exposure

Construction counts, input validation, and evaluation results describe different aspects of the benchmarks. The Coding construction records document human review of all 2,061 inputs. Numerical scoring uses the subset defined in App. G, and formulation checks use the references and mathematical assumptions specified in App. G.4. Reviewing an input description does not establish that a generated model represents it correctly.

Development exposure also limits claims about unseen problems. Twenty-four Coding seeds were withheld from one operator-development configuration, but all 200 seeds had previously been evaluated with ORLM. Those 24 seeds therefore do not form a test set untouched by all development activities. The defense experiment's reuse of pilot outputs and its sensitivity analyses are described in App. G.6.

The reported evidence comprises matched-control QA results, numerical Coding scores, checks of selected generated formulations, repeated-generation comparisons, and defense experiments (Apps. F and G). Each analysis uses its stated inputs and denominator. We do not report complete results combining numerical and formulation checks across all outputs, or confirmatory statistical tests. The findings also do not imply that every description change lowers accuracy.