OR-Space: A WORKSPACE-BASED BENCHMARK FOR INDUSTRIAL OPTIMIZATION AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language model (LLM) agents are increasingly used for operations research (OR) modeling, but existing OR benchmarks mostly test model construction from a self-contained problem statement. Industrial optimization also requires connecting requirements to data and code, revising models, and explaining solver results. We introduce **OR-Space**, a benchmark of executable workspaces containing business documents, structured data, code, solver outputs, and task-specific evaluators. OR-Space evaluates three tasks separately: BUILD, where agents create solver-ready models from heterogeneous artifacts; REVISE, where they update models under changed requirements while preserving valid prior logic; and EXPLAIN, where they ground answers about solutions and constraints in workspace evidence. OR-Space evaluates 16 language models on 100 base problems instantiated as 300 task instances, revealing substantial failures in schema mapping, constraint grounding, legacy-code use, and solver-backed explanation. Controlled studies show that workspace interface, revision context, and solver backend affect performance. The dataset and code are available in the anonymous artifact at https://anonymous.4open.science/r/OR-Space-C324/.

1 INTRODUCTION

Large language models (LLMs) can help automate optimization modeling in operations research (OR) (AhmadiTeshnizi et al., 2024; Huang et al., 2025a). In a common setting, an LLM is given a natural-language description of a decision problem. It must identify the decision variables, objective, and constraints, then translate them into a mathematical model and executable solver code. An optimization solver computes a solution from the generated model. This approach aims to reduce the expertise and effort needed to formulate and implement optimization models.

However, industrial OR projects rarely start with a complete problem statement. Practitioners must turn business requirements into an optimization model, implement it with the available data, and check its solver results. Their work continues after the first solution. They must update the model when requirements change and explain the results to users. Evaluating industrial optimization agents therefore requires testing how they **BUILD, REVISE, and EXPLAIN optimization models** in practice.

Existing benchmarks for LLM-based optimization modeling test whether LLMs can formulate and solve optimization problems described in text (Huang et al., 2025a; Li et al., 2026). Many such evaluations use a self-contained problem statement and focus on initial model construction. This design leaves three evaluation gaps. **Consolidated context** reduces the need to find and connect information across separate files. **Static specifications** do not test how agents update an existing model when requirements change. **Outcome-focused assessment** checks the computed result, but leaves the ability to explain it with evidence untested. Although general agent benchmarks evaluate agents in richer, stateful environments (Jimenez et al., 2024; Zhou et al., 2024; Liu et al., 2025b; Xie et al., 2024; Drouin et al., 2024; Trivedi et al., 2024; Xu et al., 2024), they do not specifically assess how agents implement and explain optimization models.

To address these gaps, we introduce **OR-Space**, a workspace-grounded benchmark for industrial optimization agents. Each instance derives from an industrial optimization case and is organized as an executable workspace. Business documents, parameter files, and code remain separate, and generated programs read data directly from these files. **BUILD** asks agents to construct a solver-ready model from this workspace. **REVISE** asks them to modify an existing implementation for new

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.

requirements while preserving valid prior logic. **EXPLAIN** asks them to justify answers about the model and its solutions using evidence from the workspace and solver runs. The three tasks are evaluated independently.

Using OR-Space, we evaluate 16 LLMs and two open-source agent harnesses on 300 task instances derived from 100 base problems. The three tasks reveal different model strengths in industrial optimization. Different models lead on BUILD and EXPLAIN, while controlled comparisons show that existing code provides useful context for REVISE. Working conditions also affect performance. For example, with the underlying problem information held fixed, flattening the workspace into a single prompt raises Pass@1 by up to 19 percentage points (Section 4.3). Across solver backends, average Pass@1 over code-generation tasks differs by up to 13.4 percentage points within the same model group (Section 4.5). These findings support evaluating industrial optimization agents across construction, revision, and explanation, with workspace organization and execution conditions included in the evaluation design.

We make three contributions.

- We introduce and release **OR-Space**, to our knowledge the **first workspace-based benchmark** for evaluating industrial optimization agents across model construction, revision, and explanation. It organizes 300 task instances from 100 base problems as executable workspaces that preserve file boundaries and relative paths. We release the data, scoring code, and selected construction prompts to support reproducible evaluation.
- We define **BUILD, REVISE, and EXPLAIN** around the **practical demands of industrial OR**. The tasks test whether agents can turn business requirements into executable models, adapt existing implementations when requirements change, and explain solutions with evidence. Dedicated evaluation protocols make these capabilities separately measurable.
- We systematically evaluate 16 LLMs, including frontier proprietary and open-source models, and two leading open-source agent harnesses, OpenCode and Pi, across all three tasks. Controlled studies and failure analyses characterize task capabilities and the effects of workspace organization, revision context, solver backend, and agent harness.

2 RELATED WORK

LLMs for optimization modeling. Recent OR benchmarks increasingly evaluate LLMs through solver-grounded outcomes rather than textual similarity, including NL4Opt (Ramamonjison et al., 2023), Mamo (Huang et al., 2025b), ORLM/IndustryOR (Huang et al., 2025a), OptiMUS (AhmadiTeshnizi et al., 2024), and Chain-of-Experts/ComplexOR (Xiao et al., 2024). Subsequent work has expanded this line toward larger-scale data construction, more realistic industrial settings, and increasingly specialized modeling systems (Lu et al., 2025; Li et al., 2026; Mostajabdaveh et al., 2025; Zhang et al., 2025a; He et al., 2026; Huang et al., 2026). Most OR modeling benchmarks focus on constructing a model from a self-contained problem statement. OR-Space evaluates construction from separate documents and data files, revision using an existing model, and explanation using workspace and solver evidence.

Agent evaluation in multi-artifact environments. General agent benchmarks have increasingly moved evaluation from isolated prompts into interactive environments. SWE-bench (Jimenez et al., 2024), WebArena (Zhou et al., 2024), AgentBench (Liu et al., 2025b), GAIA (Mialon et al., 2023), MLE-bench (Chan et al., 2024), and τ-bench (Yao et al., 2024), among others, test agents through interaction with external environments and their resulting state. Related studies further show that what a benchmark measures depends not only on the underlying model, but also on how the surrounding environment and agent harness are designed (Zhu et al., 2025; Liu et al., 2025a). This environment-centric view now spans a broad range of agent settings (Xie et al., 2024; Drouin et al., 2024; Trivedi et al., 2024; Xu et al., 2024), while recent work on agent externalization similarly argues that persistent system components should be treated as part of the agent itself (Zhou et al., 2026; Yang et al., 2025). OR-Space examines how workspace artifacts and solver interfaces affect model construction, revision, and explanation in industrial optimization.

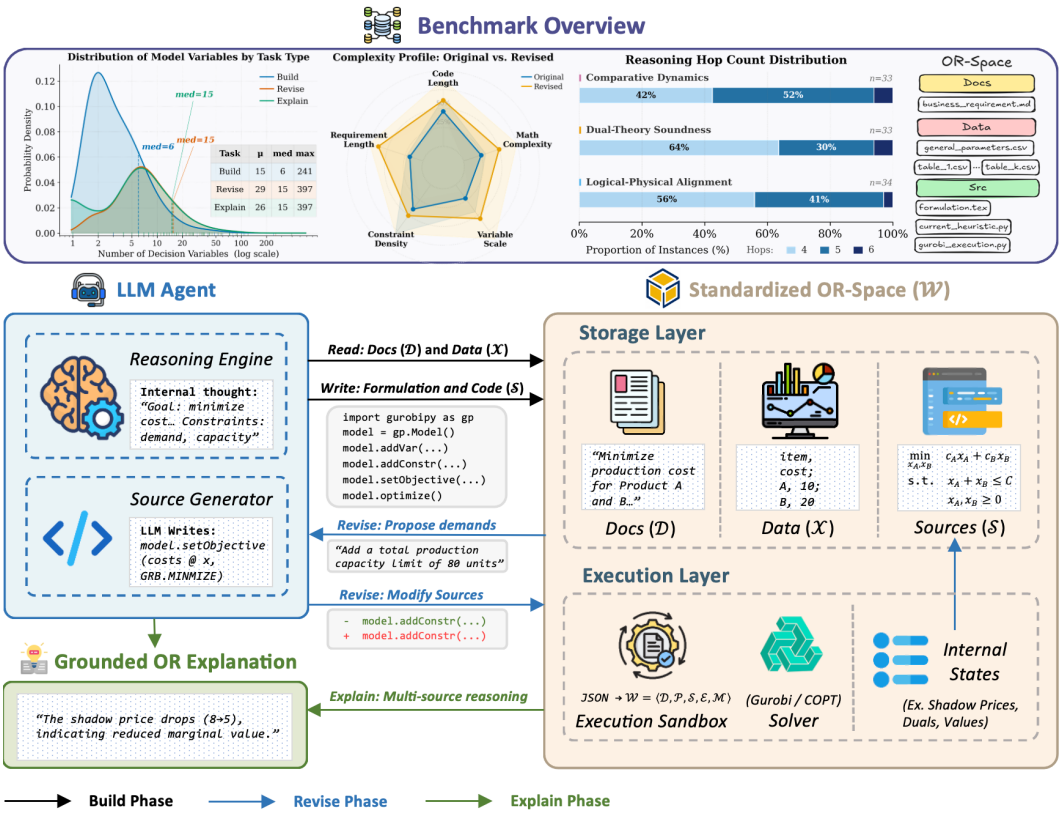

Figure 1: `OR-Space` overview: file-structured workspaces, task interfaces, and dataset statistics. The three tasks are evaluated independently; arrows mark their inputs and outputs.

3 OR-Space Benchmark

Figure 1 summarizes the workspaces, task interfaces, and dataset statistics of `OR-Space`.

3.1 OR Workspace Formalization

Definition 1 (OR Workspace). *An OR workspace and its attached evaluator are represented as*

$$\mathcal{W} = \langle \mathcal{D},\ \mathcal{P},\ \mathcal{S},\ \mathcal{E},\ \mathcal{M} \rangle, \tag{1}$$

where $\mathcal{D}$, $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{E}$ denote documents, parameters, code, and the runtime environment. The evaluator $\mathcal{M}$ is attached by the benchmark harness and is not exposed to the agent as a workspace artifact.

Documents specify the business context, objective, and requirements. Parameter files, typically CSV or JSON, are the canonical source of numerical inputs even when documents repeat quantities for readability. Values may span several tables, requiring agents to map fields to business concepts and join data across files. Code implements the model; the runtime executes it against workspace data, records outputs, and enforces a time limit. Artifacts retain their file boundaries and relative paths.

This model–data separation follows AMPL, Pyomo, and JuMP (Fourer et al., 2002; Hart et al., 2011; Dunning et al., 2017).

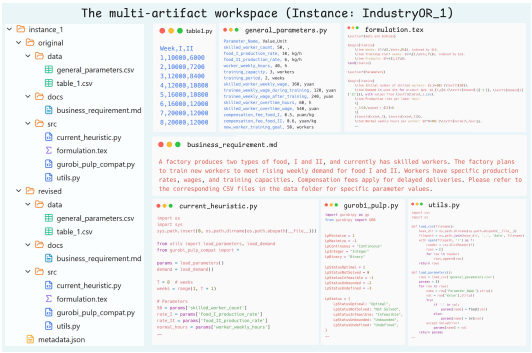

Figure 2: The `IndustryOR_1` workspace (enlarged in Appendix Figure 8).

Table 1: Workspace coverage. Columns follow Eq. 1; ✓: evaluated, ∼: partial, ×: absent.

Benchmark	Docs $\mathcal{D}$	Params $\mathcal{P}$	Code $\mathcal{S}$	Env. $\mathcal{E}$	Metric $\mathcal{M}$
OR modeling benchmarks					
NL4Opt (Ramamonjison et al., 2023)	∼	×	×	×	∼
Mamo (Huang et al., 2025b)	∼	×	×	✓	✓
IndustryOR (Huang et al., 2025a)	∼	×	×	✓	✓
ComplexOR (Xiao et al., 2024)	∼	×	×	✓	✓
NLP4LP (AhmadiTeshnizi et al., 2024)	∼	×	×	✓	✓
General agent benchmarks					
SWE-bench (Jimenez et al., 2024)	∼	×	✓	✓	✓
WebArena (Zhou et al., 2024)	✓	∼	×	✓	∼
AgentBench (Liu et al., 2025b)	∼	∼	∼	✓	∼
OR-Space	✓	✓	✓	✓	✓

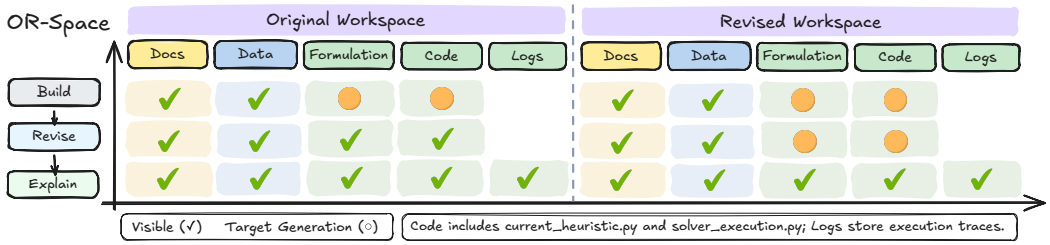

Figure 3: Artifact visibility by task. Checks denote inputs, circles outputs, blanks unavailable artifacts. REVISE combines all context variants; default REVISE-CODE includes correct original code but no revised reference code.

Table 1 compares how existing benchmarks cover these workspace components; Section 4.1 specifies our runtime and evaluation metrics.

3.2 BENCHMARK TASKS

Each base problem yields three independently evaluated task instances, with inputs and outputs shown in Figure 3.

BUILD ($\mathcal{T}_{\text{BUILD}}$). The agent receives the workspace artifacts: business documents, parameter files, and a code scaffold. It must formulate and implement a complete optimization model using the configured solver backend, connecting the requirements to the supplied data.

REVISE ($\mathcal{T}_{\text{REVISE}}$). The agent receives revised requirements and data together with context from the original model. It must implement the changes while preserving valid prior logic. REVISE-CODE supplies correct original code, REVISE-MODEL the mathematical formulation, and REVISE-ALL both. The revised reference implementation is hidden from the agent.

EXPLAIN ($\mathcal{T}_{\text{EXPLAIN}}$). The agent receives verified workspaces for both the original and revised problems, including documents, data, reference code, execution logs, and solver records. It answers questions about the solutions, model behavior, and effects of the revision, supporting its claims with workspace and solver evidence.

3.3 BENCHMARK CONSTRUCTION PIPELINE

We construct 300 task instances from 100 IndustryOR problems (Huang et al., 2025a), preserving a common mathematical foundation across tasks. For each problem, we recover a clean specification consistent with the ground-truth model, then distribute its requirements and numerical inputs across a business-facing workspace.

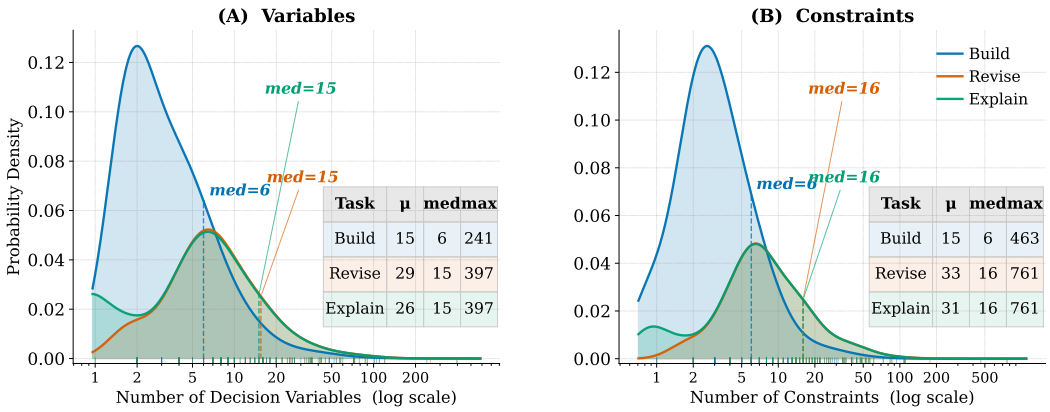

Figure 4: Distributions of decision-variable and constraint counts across tasks (log-scaled axes).

For BUILD, the verified formulation and executable implementation are retained as hidden references. To derive REVISE, we introduce structural changes to the business requirements that require updates across the existing formulation. For EXPLAIN, we generate analytical questions from verified executions, linking observed solution behavior to modeling decisions or business implications. Each question requires evidence from both the workspace context and the solved model.

Quality control. We independently rebuild and execute the original and revised models to verify the requested changes and preservation of unaffected logic. Every reference implementation must reproduce the ground-truth objective within a construction-time relative tolerance of 10^{-3}. Revised instances are also reconstructed from the visible workspace to detect inconsistencies between the requirements and the model. Structural and linguistic checks are followed by manual review by researchers with OR experience. Appendix E provides the construction details and prompts.

Dataset scale and extensibility. Figure 4 compares decision-variable and constraint counts in the current 100-problem release. Median BUILD instances have six variables and six constraints, rising to 15 and 16 in REVISE and EXPLAIN. Their longer right tails include substantially larger formulations. Although REVISE and EXPLAIN have similar scale distributions, they evaluate model modification and grounded interpretation of solutions, respectively. Our framework and benchmark paradigm are extensible: additional source data and problem instances can be incorporated through the same workspace construction, validation, and evaluation pipeline.

4 EXPERIMENTS

4.1 EXPERIMENT SETUP

Agent pipeline and runtime. Unless otherwise stated, we evaluate 16 open- and closed-source models on 100 instances per task with the filesystem setting, REVISE-CODE, and Gurobi 13.0.1 (Table 2), including available reasoning variants. Our fixed `OR-Space` pipeline exposes the visible documents, data, and code with their original paths; BUILD and REVISE programs read these workspace files directly. Each program runs in a separate Python process with a 180-second timeout. Output budgets depend on task and provider; truncated EXPLAIN answers are regenerated with larger budgets before scoring. Section 4.2 compares two open-source harnesses on all three tasks.

Evaluation metrics. For BUILD and REVISE, a submission passes if it executes successfully with the required solver and its objective value v matches the Gurobi reference optimum $v^\star$ within tolerance ϵ:

$$\mathcal{M}_{\text{obj}}(v) = \mathbf{1}\left[\frac{|v - v^\star|}{\max\{1, |v^\star|\}} \leq \epsilon\right]. \tag{2}$$

Table 2: Main results (100 instances/task). BUILD/REVISE-CODE: Pass@1 (%); EXPLAIN: Rubric (0–100); $\Delta_{\mathrm{R-B}}$: percentage points. Bold/underline: best/second-best in each group.

Model / system	BUILD	REVISE (CODE)	EXPLAIN	$\Delta_{\mathrm{R-B}}$
Closed-source Models				
`gemini-3.1-pro`	**72.0**	**87.0**	69.13	+15
`gemini-3-flash`	<u>68.0</u>	82.0	64.21	+14
`claude-opus-4.6`	62.0	**87.0**	78.70	+25
`gpt-5.5`	59.0	85.0	**85.09**	+26
`claude-sonnet-4.5`	53.0	82.0	<u>81.11</u>	+29
`gemini-2.5-flash`	53.0	71.0	65.46	+18
`gemini-2.5-pro`	47.0	81.0	60.22	+34
`gpt-4o`	25.0	36.0	56.17	+11
Open-source Models				
`deepseek-v4-pro`	**67.0**	**80.0**	<u>73.62</u>	+13
`deepseek-r1`	<u>55.0</u>	65.0	61.59	+10
`deepseek-v4-flash`	54.0	<u>70.0</u>	73.43	+16
`qwen3-max`	49.0	68.0	72.95	+19
`qwen3-32b`	32.0	32.0	58.38	0
`qwen3-32b-thinking`	28.0	32.0	59.02	+4
`qwen3.5-27b`	38.0	48.0	**77.79**	+10
`qwen3-14b`	20.0	19.0	53.74	−1
Harness system with `gemini-3.1-pro`				
`OpenCode` (OpenCode Team, 2025)	**81.0**	<u>93.0</u>	<u>65.93</u>	+12
`Pi` (Zechner, 2025)	<u>78.0</u>	**95.0**	**69.51**	+17

We report Pass@1 and retain execution outcomes for failure analysis. EXPLAIN uses a reference checklist for each question: programmatic checks verify exact facts, and an independent LLM judge assesses content coverage, reasoning, supporting evidence, and answer quality. Unsupported claims incur a penalty of up to 20 points and are retained for audit. We report the mean score over 100 instances on a 0–100 scale. Appendix C details the rubric, Appendix E.2 provides the prompt templates used in construction and scoring, and Appendix F compares automated scores with expert assessments of 90 responses.

4.2 MAIN RESULTS

Table 2 reports results for 16 models with the fixed `OR-Space` pipeline and two additional agent harnesses. Under the fixed pipeline, the highest BUILD Pass@1 is 72%, achieved by `gemini-3.1-pro`; even the strongest model fails on more than a quarter of the instances. Section 4.6 traces these failures to errors in data mapping and formulation.

In REVISE-CODE, 14 of the 16 models outperform their BUILD scores. The original implementation provides a starting point, although agents must still identify which parts to retain or change. Section 4.4 isolates the effect of this context by varying the code and formulation provided.

The rankings differ for EXPLAIN: `qwen3.5-27b` leads the open-source models under the rubric despite weaker BUILD and REVISE scores than several peers. Across all models, BUILD and REVISE are strongly correlated ($r = 0.93$), while EXPLAIN is less closely related to BUILD ($r = 0.58$) and REVISE ($r = 0.66$). Grounded explanation therefore reveals differences that construction and revision scores alone do not capture.

Harness system. The final group in Table 2 compares OpenCode (OpenCode Team, 2025) and Pi (Zechner, 2025) using `gemini-3.1-pro` with the same model under the `OR-Space` pipeline and default task settings. Both harnesses improve BUILD and REVISE Pass@1 by 6 to 9 percentage points. The EXPLAIN score falls by 3.20 points with OpenCode and rises by 0.38 with Pi. Harness choice thus affects the tasks differently.

Reference evidence chains. Figure 5 summarizes all 100 EXPLAIN questions. Reference paths usually contain four or five evidence stages: depth five accounts for 52% of comparative-dynamics questions, while depth four accounts for 64% of dual-theory and 56% of logical–physical questions. Depth counts stages in GPT-5.4-generated reference paths, not observed model reasoning. Appendix B.2 gives the counting rule.

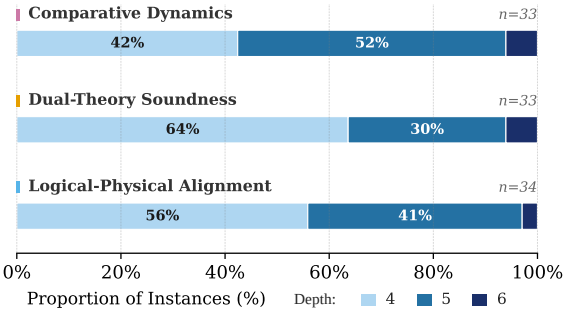

Figure 5: Reference evidence-chain depth by question type ($n = 100$).

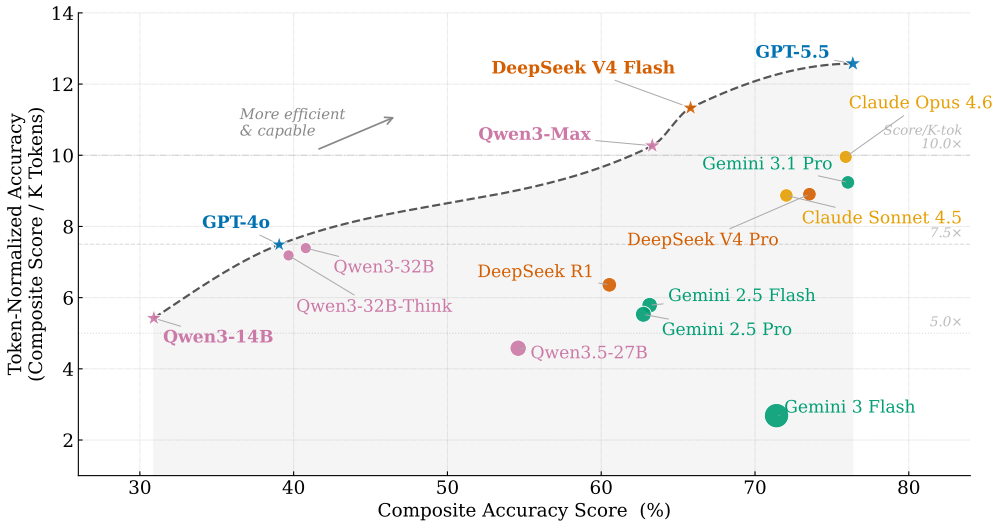

Figure 6: Accuracy and token efficiency across 16 models. Marker area denotes mean token use; the dashed curve connects highlighted configurations.

Token efficiency. Figure 6 compares accuracy with token use. For this descriptive comparison, composite accuracy averages BUILD Pass@1, REVISE-CODE Pass@1, and the EXPLAIN Rubric score; efficiency divides this score by mean total tokens in thousands. `gpt-5.5` combines a high composite score with high token efficiency, whereas `gemini-3-flash` reaches a relatively high score with substantially lower efficiency. Strong task performance can therefore entail different token requirements. Appendix A.5 gives the definitions and accounting details; the main evaluation continues to report each task separately.

4.3 FILESYSTEM VS. FLAT PROMPT

We compare **FILESYSTEM**, which preserves file paths and boundaries and requires programs to read workspace data, with a **FLAT** prompt containing the same requirements and data inline. Table 3 reports paired results for six models on identical instances, with solver, scoring, and REVISE-CODE context fixed.

Filesystem lowers average Pass@1 by 8.2 percentage points on BUILD and 6.7 on REVISE compared with Flat. BUILD falls for five of six models and is unchanged for one; REVISE falls for all six. These gaps reflect the difficulty of connecting separate artifacts into executable programs that match workspace schemas and file paths.

EXPLAIN changes little: Filesystem scores 0.2 Rubric points higher on average. Flattening makes evidence easier to access, but both interfaces require connecting solver results to code, data, and business requirements to explain solution properties and revision effects.

Table 3: Filesystem versus Flat prompt on six models with matched inputs.

Model	BUILD FS (↑)	BUILD FLAT (↑)	REVISE-CODE FS (↑)	REVISE-CODE FLAT (↑)	EXPLAIN FS (↑)	EXPLAIN FLAT (↑)
`gemini-3.1-pro`	**72**	**72**	**87**	**94**	69.13	70.55
`deepseek-v4-pro`	67	70	80	85	73.62	73.76
`gpt-5.5`	59	66	85	89	**85.09**	**84.17**
`claude-sonnet-4.5`	53	62	82	85	81.11	79.68
`qwen3-max`	49	60	68	75	72.95	74.19
`gpt-4o`	25	44	36	50	56.17	54.57
Average $\Delta_{\mathrm{Fs-Flat}}$	-8.2		-6.7		$+0.2$	

Table 4: REVISE context variants across solvers, macro-averaged Pass@1 (%); deltas are relative to REVISE-CODE.

Solver	REVISE-CODE	REVISE-ALL	REVISE-MODEL	All − Code	Model − Code
High-capability models					
Gurobi	**79.5**	**84.4**	**76.6**	$+4.9$	-2.9
COPT	72.3	77.4	73.3	$+5.1$	$+1.0$
CBC	72.9	76.4	69.3	$+3.5$	-3.6
HiGHS	68.4	63.8	68.3	-4.6	-0.1
Lightweight models					
Gurobi	**48.6**	**49.4**	**40.6**	$+0.8$	-8.0
COPT	33.5	34.6	29.9	$+1.1$	-3.6
CBC	44.1	47.6	38.4	$+3.5$	-5.8
HiGHS	37.3	36.0	33.6	-1.3	-3.6

4.4 REVISE CONTEXT VARIANTS

To isolate revision context, we vary the original code and formulation while holding revised requirements, data, models, scoring, and reference objectives fixed. Table 4 reports mean Pass@1 for high-capability and lightweight models across four backends, relative to REVISE-CODE. Group membership is fixed and listed in Appendix A.

Adding the formulation to code improves Pass@1 with Gurobi, COPT, and CBC in both groups, with gains at least as large for high-capability models. HiGHS is the exception: providing both lowers the average score in each group.

The formulation alone underperforms code in seven of eight group–backend comparisons. Code supplies a concrete implementation to revise, while the formulation helps interpret its mathematical structure. Models that abstract reusable structure from legacy code tend to improve; those copying stale details can decline. Appendices A.3 and A.5 give model-level results and token analysis.

4.5 CROSS-SOLVER ROBUSTNESS

Table 5 compares Gurobi, COPT, CBC, and HiGHS for the same model groups. Problems and scoring remain fixed; solver APIs, code, and solver records vary. Appendix A.1 reports each model's results.

Solver choice changes absolute performance and interacts with model capability. Gurobi gives the highest average score on the code-generation tasks in both groups. COPT ranks second in the high-capability group, while CBC ranks second in the lightweight group. COPT is more competitive in the high-capability group: its gap from Gurobi is 4.7 percentage points, compared with 13.4 points in the lightweight group. Thus, the backend effect varies with model capability. EXPLAIN scores also vary across backends, with Gurobi yielding the highest average score in both model groups and HiGHS the lowest.

Gurobi and COPT give broadly similar rankings: Spearman $\rho = 0.90$ when models are ranked by their mean BUILD, REVISE-CODE, and EXPLAIN score. Individual models can still change substantially across backends (Appendix A.4), motivating evaluation with multiple solver interfaces.

Table 5: Cross-solver performance by capability group, macro-averaged from Appendix Table 6. OBJ. AVG averages BUILD and the three REVISE variants.

Solver	BUILD	R-MODEL	R-CODE	R-ALL	EXPLAIN	OBJ. AVG	Range
High-capability models							
Gurobi	**60.6**	**76.6**	**79.5**	**84.4**	**73.3**	**75.3**	23.8
COPT	59.5	73.3	72.3	77.4	70.3	70.6	17.9
CBC	57.9	69.3	72.9	76.4	66.2	69.1	18.5
HiGHS	55.9	68.3	68.4	63.8	65.4	64.1	12.5
Lightweight models							
Gurobi	**37.1**	**40.6**	**48.6**	**49.4**	**63.0**	**43.9**	12.3
COPT	24.0	29.9	33.5	34.6	58.6	30.5	10.6
CBC	37.0	38.4	44.1	47.6	58.4	41.8	10.6
HiGHS	35.3	33.6	37.3	36.0	54.2	35.5	3.6

Range: max–min of the group means for BUILD and the three REVISE variants (pp).

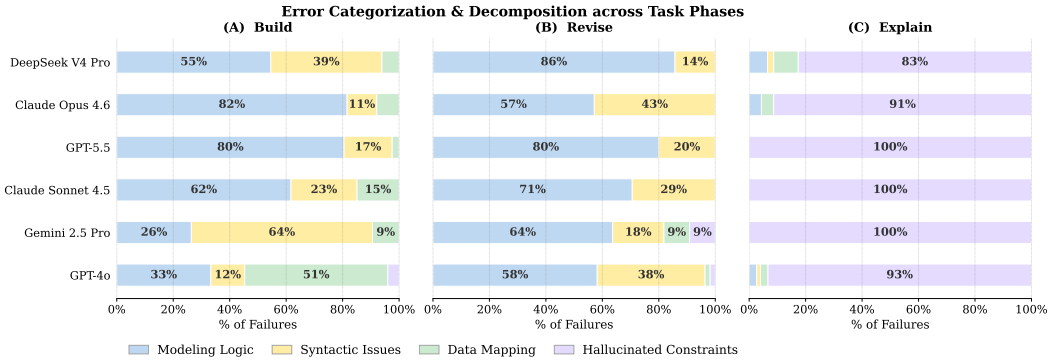

Figure 7: Composition of manually audited failures in BUILD, REVISE-ALL, and EXPLAIN. Each model–task bar is normalized over its audited failures, with one mutually exclusive primary category per failure; proportions are not model failure rates.

4.6 FAILURE ANALYSIS

The three tasks expose different failure patterns (Figure 7). BUILD errors often arise from misinterpreting requirements or data. REVISE adds errors from reusing code that no longer fits the updated workspace, while EXPLAIN can produce fluent but unsupported claims.

Many failed BUILD and REVISE submissions execute successfully but return the wrong objective value. We distinguish these from execution failures, which in REVISE can arise from inherited imports or variable names without the required modules or definitions. Appendix A.6 provides the error labels and breakdowns.

A closer audit of `gemini-3.1-pro`'s 28 BUILD failures uses execution traces, objective deviations, and stated requirements to identify data-mapping errors (39%) and unsupported constraints (29%); the rest involve modeling or syntax errors. For example, one program changes parameter-name case and reads nonexistent keys; another replaces the stated objective with profit maximization. Appendix Table 10 traces these errors to workspace evidence.

5 CONCLUSION

`OR-Space` independently evaluates construction, revision, and explanation on 100 base problems, using executable workspaces of business documents, data, code, and solver outputs. Across 16 LLMs, it exposes failures in data mapping, constraint grounding, legacy-code use, and solver-backed explanation. Controlled comparisons show that workspace interface, revision context, and solver backend affect performance.

AI USE STATEMENT

Generative AI is used in the benchmark construction and evaluation pipeline described in this paper. These uses include generating synthetic workspaces and requirement revisions, rewriting business documents, generating EXPLAIN questions and scoring criteria, performing quality-control checks, and judging semantic criteria in EXPLAIN responses. Appendix E.2 documents the corresponding prompt roles and validation procedures. AI assistance was also used to check the manuscript's LaTeX source.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models, 2024. URL https://arxiv.org/abs/2402.10172.

Mubashara Akhtar, Omar Benjelloun, Costanza Conforti, Pieter Gijsbers, Joan Giner-Miguelez, Nitisha Jain, Michael Kuchnik, Quentin Lhoest, Pierre Marcenac, Manil Maskey, Peter Mattson, Luis Oala, Pierre Ruyssen, Rajat Shinde, Elena Simperl, Goeffry Thomas, Slava Tykhonov, Joaquin Vanschoren, Jos van der Velde, Steffen Vogler, and Carole-Jean Wu. Croissant: A metadata format for ML-ready datasets. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning (DEEM '24), co-located with SIGMOD/PODS '24*, pp. 1–6, 2024. doi: 10.1145/3650203.3663326. URL https://doi.org/10.1145/3650203.3663326.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering, 2024. URL https://arxiv.org/abs/2410.07095. OpenAI; accepted at ICLR 2025.

Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Banghua Zhu, Hao Zhang, Michael I. Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot Arena: An open platform for evaluating LLMs by human preference. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, volume 235 of *PMLR*, pp. 8359–8388, July 2024. URL https://proceedings.mlr.press/v235/chiang24b.html.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

Iain Dunning, Joey Huchette, and Miles Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017. doi: 10.1137/15M1020575. URL https://doi.org/10.1137/15M1020575.

Robert Fourer, David M. Gay, and Brian W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2 edition, 2002. URL https://ampl.com/resources/the-ampl-book/.

Dongdong Ge, Qi Huangfu, Zizhuo Wang, Jian Wu, and Yinyu Ye. Cardinal optimizer (COPT) user guide, 2022. URL https://arxiv.org/abs/2208.14314.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, December 2021. doi: 10.1145/3458723. URL https://doi.org/10.1145/3458723.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2024. URL https://www.gurobi.com.

William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and solving mathematical programs in Python. *Mathematical Programming Computation*, 3(3):219–260, 2011. doi: 10.1007/s12532-011-0026-8. URL https://doi.org/10.1007/s12532-011-0026-8.

Yiliu He, Tianle Li, Binghao Ji, Zhiyuan Liu, and Di Huang. EvoOpt-LLM: Evolving industrial optimization models with large language models, 2026. URL https://arxiv.org/abs/2602.01082.

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, November 2025a. ISSN 0030-364X. doi: 10.1287/opre.2024.1233. URL https://pubsonline.informs.org/doi/10.1287/opre.2024.1233.

Chenyu Huang, Jianghao Lin, Zhengyang Tang, Bo Jiang, Ruoqing Jiang, Benyou Wang, and Lai Wei. InvEvolve: Evolving white-box inventory policies via large language models with performance guarantees, 2026. URL https://arxiv.org/abs/2605.00369.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages, 2025b. URL https://arxiv.org/abs/2405.13144. Findings of NAACL 2025.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?, November 2024. URL http://arxiv.org/abs/2310.06770. arXiv:2310.06770 [cs].

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. Prometheus: Inducing fine-grained evaluation capability in language models, 2023. URL https://arxiv.org/abs/2310.08491. ICLR 2024.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization, 2023. URL https://arxiv.org/abs/2307.03875.

Zhong Li, Hongliang Lu, Tao Wei, Wenyu Liu, Yuxuan Chen, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark, 2026. URL https://arxiv.org/abs/2602.10450.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research (TMLR)*, 2023. URL https://openreview.net/forum?id=iO4LZibEqW. Featured Certification. arXiv:2211.09110.

Jianghao Lin, Zi Ling, Chenyu Zhou, Tianyi Xu, Ruoqing Jiang, Zizhuo Wang, and Dongdong Ge. From soliloquy to agora: Memory-enhanced LLM agents with decentralized debate for optimization modeling, 2026. URL https://arxiv.org/abs/2604.25847. Working paper.

Weiwen Liu, Jiarui Qin, Xu Huang, Xingshan Zeng, Yunjia Xi, Jianghao Lin, Chuhan Wu, Yasheng Wang, Lifeng Shang, Ruiming Tang, Defu Lian, Yong Yu, and Weinan Zhang. Position: The real barrier to LLM agent usability is agentic ROI, 2025a. URL https://arxiv.org/abs/2505.17767.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as Agents, October 2025b. URL http://arxiv.org/abs/2308.03688. arXiv:2308.03688 [cs].

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-Eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2511–2522, December 2023. doi: 10.18653/v1/2023.emnlp-main.153. URL https://aclanthology.org/2023.emnlp-main.153/.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling, 2025. URL https://arxiv.org/abs/2502.11102.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: a benchmark for General AI Assistants, 2023. URL https://arxiv.org/abs/2311.12983. arXiv:2311.12983; accepted at ICLR 2024.

Mahdi Mostajabdaveh, Timothy T. Yu, Samarendra Chandan Bindu Dash, Rindranirina Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating LLM reasoning in the operations research domain with ORQA, 2025. URL https://arxiv.org/abs/2412.17874. AAAI 2025.

OpenCode Team. OpenCode: The open source ai coding agent, 2025. URL https://github.com/anomalyco/opencode. Open-source software. Accessed September 26, 2026.

Mahima Pushkarna, Andrew Zaldivar, and Oddur Kjartansson. Data Cards: Purposeful and transparent dataset documentation for responsible AI. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency (FAccT '22)*, pp. 1776–1826, 2022. doi: 10.1145/3531146.3533231. URL https://doi.org/10.1145/3531146.3533231.

Rindranirina Ramamonjison, Timothy T. 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, 2023. URL https://arxiv.org/abs/2303.08233.

Rickard Stureborg, Dimitris Alikaniotis, and Yoshi Suhara. Large language models are inconsistent and biased evaluators, 2024. URL https://arxiv.org/abs/2405.01724.

Aman Singh Thakur, Kartik Choudhary, Venkat Srinik Ramayapally, Sankaran Vaidyanathan, and Dieuwke Hupkes. Judging the judges: Evaluating alignment and vulnerabilities in LLMs-as-judges, 2024. URL https://arxiv.org/abs/2406.12624.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents, 2024. URL https://arxiv.org/abs/2407.18901. ACL 2024.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators, 2023. URL https://arxiv.org/abs/2305.17926.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-Experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ. Introduces a multi-agent LLM framework for OR modeling and the ComplexOR benchmark.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024. URL https://arxiv.org/abs/2404.07972.

Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. TheAgentCompany: Benchmarking LLM agents on consequential real world tasks, 2024. URL https://arxiv.org/abs/2412.14161.

Yingxuan Yang, Huacan Chai, Yuanyi Song, Siyuan Qi, Muning Wen, Ning Li, Junwei Liao, Haoyi Hu, Jianghao Lin, Gaowei Chang, Weiwen Liu, Ying Wen, Yong Yu, and Weinan Zhang. A survey of AI agent protocols, 2025. URL https://arxiv.org/abs/2504.16736.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045. arXiv:2406.12045.

Mario Zechner. Pi: Agent harness and coding agent, 2025. URL https://github.com/earendil-works/pi. Open-source software. Accessed September 26, 2026.

Bowen Zhang, Pengcheng Luo, Genke Yang, Boon-Hee Soong, and Chau Yuen. OR-LLM-Agent: Automating modeling and solving of operations research optimization problems with reasoning LLM, 2025a. URL https://arxiv.org/abs/2503.10009.

Xinzhi Zhang, Zeyi Chen, Humishka Zope, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Janardhan Kulkarni, Ishai Menache, and Sirui Li. OptiMind: Teaching LLMs to think like optimization experts, 2025b. URL https://arxiv.org/abs/2509.22979.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023. URL https://arxiv.org/abs/2306.05685.

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models, 2025a. URL https://arxiv.org/abs/2509.22558.

Chenyu Zhou, Jingyuan Yang, Linwei Xin, Yitian Chen, Ziyan He, and Dongdong Ge. Auto-formulating dynamic programming problems with large language models, 2025b. URL https://arxiv.org/abs/2507.11737.

Chenyu Zhou, Huacan Chai, Wenteng Chen, Zihan Guo, Rong Shan, Yuanyi Song, Tianyi Xu, Yingxuan Yang, Aofan Yu, Weiming Zhang, Congming Zheng, Jiachen Zhu, Zeyu Zheng, Zhuosheng Zhang, Xingyu Lou, Changwang Zhang, Zhihui Fu, Jun Wang, Weiwen Liu, Jianghao Lin, and Weinan Zhang. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering, 2026. URL https://arxiv.org/abs/2604.08224.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A Realistic Web Environment for Building Autonomous Agents, April 2024. URL http://arxiv.org/abs/2307.13854. arXiv:2307.13854 [cs].

Jiachen Zhu, Menghui Zhu, Renting Rui, Rong Shan, Congmin Zheng, Bo Chen, Yunjia Xi, Jianghao Lin, Weiwen Liu, Ruiming Tang, Yong Yu, and Weinan Zhang. Evolutionary perspectives on the evaluation of LLM-based AI agents: A comprehensive survey, 2025. URL https://arxiv.org/abs/2506.11102.

APPENDIX

A Additional Experimental Results 14
A.1 Full Solver-Backend Results 14
A.2 Filesystem Interface Delta Details 16
A.3 REVISE Context Model-Level Details 17
A.4 Solver-Backend Sensitivity Details 17
A.5 Token Efficiency and Model-Level Context Effects 18
A.6 Failure Details and Case Studies 18
B Supplementary Workspace and Dataset Visualizations 19
B.1 Workspace Anatomy 19
B.2 Dataset Statistics 21
C EXPLAIN Evaluation Rubric 21
D Additional Related Work 23
E Benchmark Construction and Reproducibility 23
E.1 Release, Reproducibility, and Responsible Use 23
E.2 Dataset Construction and Prompting Templates 24
F Expert Assessment: Calibration and Representativeness 29

A ADDITIONAL EXPERIMENTAL RESULTS

This appendix expands the main-text results along the dimensions that are compressed in the main tables: solver backend, workspace interface, revision context, model capability, token use, and failure mechanism. Unless explicitly stated otherwise, the task definitions, visible artifacts, scoring rules, and execution limits are identical to those described in Section 4.

The additional analyses expose variation that group averages hide: two models with similar aggregate performance may respond differently to a filesystem interface, an alternative solver API, or additional revision context. They also separate controlled effects from general model differences: each delta table changes one experimental factor while holding the evaluated model subset, task instances, and scoring procedure fixed. These comparisons complement the main results.

Capability groups. The capability-stratified analyses use two fixed eight-model groups defined before examining `OR-Space` results. The high-capability group contains `gemini-3.1-pro`, `claude-opus-4.6`, `gpt-5.5`, `gemini-3-flash`, `deepseek-r1`, `claude-sonnet-4.5`, `deepseek-v4-pro`, and `qwen3-max`; the lightweight group contains `gemini-2.5-pro`, `gemini-2.5-flash`, `gpt-4o`, `deepseek-v4-flash`, `qwen3-32b-thinking`, `qwen3-32b`, `qwen3.5-27b`, and `qwen3-14b`. The split follows model generation and provider product positioning available before evaluation, not open- versus closed-source status or performance on `OR-Space`, and remains fixed across solver backends and REVISE context variants.

A.1 FULL SOLVER-BACKEND RESULTS

This subsection reports the complete per-model results across the four solver backends used in the robustness analysis. Rows follow the model order and task definitions of Table 2. BUILD and the three REVISE variants report Pass@1 under the objective oracle $\mathcal{M}_{\text{obj}}$, while EXPLAIN reports the penalty-adjusted rubric score $\mathcal{M}_{\text{exp}}$. The Gurobi BUILD, REVISE-CODE, and EXPLAIN entries are identical to the default results in Table 2; the two additional Gurobi revision variants and the other solver tracks retain their controlled solver-robustness results. Because these metrics evaluate different

outputs, their absolute values should be interpreted within each task, not as directly interchangeable measures.

The table can be read in two complementary directions. Comparisons within a solver track show how models differ under the same task interface and backend requirement. Comparisons of the same model across solver tracks show whether performance remains stable when the required solver API changes. Aggregate backend averages can otherwise conceal model-specific gains, regressions, or ranking changes.

Each task contains 100 instances, so one additional successful instance changes Pass@1 by one percentage point. Differences of one or two points should be read with caution, especially when they occur for a single model–backend pair. The more reliable patterns are those that recur across models, task variants, or solver tracks. The controlled delta tables that follow make these recurring effects easier to inspect without repeating the complete result matrix.

All four backend tracks use the same 100 underlying problems, requirements, data, task definitions, execution limits, and scoring rules. The required solver API and the corresponding backend-specific code, execution logs, and solver evidence vary across tracks. The resulting gaps should not be interpreted as pure differences between solver algorithms: they also reflect whether a model can generate valid code for each solver API. The table measures end-to-end robustness to backend requirements rather than solver quality in isolation.

Table 6: Full model-wise results under four solver backends on `OR-Space` (100 instances per task). BUILD and the three revise variants report Pass@1 ($\mathcal{M}_{\text{obj}}$); EXPLAIN reports the penalty-adjusted rubric mean score in $[0, 100]$. The Gurobi BUILD, REVISE-CODE, and EXPLAIN entries match Table 2. Higher is better for all metric columns; within each solver and model category, best values are bolded and second-best values are underlined. Model order and grouping follow Table 2.

Solver	Model	BUILD (↑)	REVISE-all (↑)	REVISE-model (↑)	REVISE-code (↑)	EXPLAIN (↑)
Gurobi	*Closed-source Models*					
	gemini-3.1-pro	**72**	91	84	**87**	69.13
	gemini-3-flash	68	89	70	82	64.21
	claude-opus-4.6	62	**93**	81	**87**	78.70
	gpt-5.5	59	90	**85**	85	**85.09**
	claude-sonnet-4.5	53	83	70	82	81.11
	gemini-2.5-flash	53	67	59	71	65.46
	gemini-2.5-pro	47	89	71	81	60.22
	gpt-4o	25	45	22	36	56.17
	Open-source Models					
	deepseek-v4-pro	**67**	**86**	**82**	**80**	73.62
	deepseek-r1	55	72	70	65	61.59
	deepseek-v4-flash	54	73	64	70	73.43
	qwen3-max	49	71	71	68	72.95
	qwen3-32b	32	32	24	32	58.38
	qwen3-32b-thinking	28	31	28	32	59.02
	qwen3.5-27b	38	40	41	48	**77.79**
	qwen3-14b	20	18	16	19	53.74
COPT	*Closed-source Models*					
	gemini-3.1-pro	**73**	85	82	83	74.89
	gemini-3-flash	57	80	71	73	58.22
	claude-opus-4.6	70	**89**	**88**	**84**	73.76
	gpt-5.5	66	83	86	80	**81.43**
	claude-sonnet-4.5	52	83	62	78	75.69
	gemini-2.5-flash	24	39	18	33	53.06
	gemini-2.5-pro	45	75	63	77	55.29
	gpt-4o	21	22	16	31	51.83
	Open-source Models					
	deepseek-v4-pro	49	**74**	**69**	**65**	63.56
	deepseek-r1	**58**	68	63	57	66.40
	deepseek-v4-flash	43	48	59	58	63.85
	qwen3-max	51	57	65	58	68.76
	qwen3-32b	21	35	25	22	53.28
	qwen3-32b-thinking	21	31	25	26	56.85
	qwen3.5-27b	13	24	30	20	**76.14**
	qwen3-14b	4	3	3	1	58.48
CBC	*Closed-source Models*					
	gemini-3.1-pro	**72**	**88**	**81**	**80**	68.06
	gemini-3-flash	59	80	73	72	60.47

Solver	Model	BUILD (↑)	REVISE-all (↑)	REVISE-model (↑)	REVISE-code (↑)	EXPLAIN (↑)
HiGHS	claude-opus-4.6	64	84	**81**	78	70.59
	gpt-5.5	60	85	75	**80**	**82.33**
	claude-sonnet-4.5	51	76	69	73	71.71
	gemini-2.5-flash	49	65	53	62	51.37
	gemini-2.5-pro	48	75	51	62	63.34
	gpt-4o	23	40	22	43	52.66
	Open-source Models					
	deepseek-v4-pro	**54**	**75**	**66**	**72**	58.55
	deepseek-r1	**54**	60	52	67	47.24
	deepseek-v4-flash	49	70	56	63	52.95
	qwen3-max	49	63	57	61	70.45
	qwen3-32b	28	32	23	29	57.09
	qwen3-32b-thinking	30	28	26	33	55.30
	qwen3.5-27b	41	47	**66**	38	**74.26**
	qwen3-14b	28	24	10	23	60.21
	Closed-source Models					
	gemini-3.1-pro	66	76	76	77	57.65
	gemini-3-flash	49	73	58	70	50.10
	claude-opus-4.6	**67**	84	**87**	**83**	75.25
	gpt-5.5	59	**85**	85	80	**82.20**
	claude-sonnet-4.5	60	68	66	73	75.57
	gemini-2.5-flash	43	32	35	37	42.58
	gemini-2.5-pro	34	66	53	67	43.29
	gpt-4o	37	37	34	24	54.42
	Open-source Models					
	deepseek-v4-pro	**57**	35	**75**	63	56.02
	deepseek-r1	37	34	43	37	57.36
	deepseek-v4-flash	54	**61**	58	61	52.15
	qwen3-max	52	55	56	**64**	68.92
	qwen3-32b	34	24	25	19	53.68
	qwen3-32b-thinking	33	22	19	27	52.33
	qwen3.5-27b	33	31	36	47	**76.41**
	qwen3-14b	14	15	9	16	59.13

COPT extraction correction. The COPT entries for deepseek-v4-pro in BUILD, REVISE-MODEL, and REVISE-CODE use a deterministic re-execution of the originally stored responses after applying the same provider-reasoning removal used by the Gurobi extractor: text through the final </think> marker is discarded before executing the Python program. This recovers 13, 23, and 16 submissions, respectively, yielding scores of 49, 69, and 65. No model was called again; the post-marker code, workspace, COPT backend, timeout, Gurobi oracle, and relative-error tolerance were unchanged. The REVISE-ALL score is unaffected.

A.2 FILESYSTEM INTERFACE DELTA DETAILS

The main text reports the full filesystem-vs-flat table. To make the interface effect easier to inspect in the appendix, Table 7 reports only the per-model deltas, defined as filesystem score minus flat-prompt score under the same task and model. Negative values therefore indicate a filesystem penalty.

Filesystem lowers average Pass@1 by 8.2 percentage points on BUILD and 6.7 points on REVISE-CODE. All six models decline on REVISE-CODE; five decline on BUILD and one is unchanged. These gaps reflect the difficulty of connecting information across separately organized artifacts and generating code that matches the data schemas and file paths in the workspace. For EXPLAIN, three models improve and three decline, with an average difference of $+0.2$ rubric points. Flattening makes the evidence easier to access, but models still need to connect solver results with the relevant code, data, and business requirements to explain solution properties and the effects of a revision.

Table 7: Per-model filesystem effect on the controlled six-model subset. Entries are Filesystem minus Flat scores. BUILD and REVISE-CODE use Pass@1 percentage-point differences, while EXPLAIN uses rubric-score differences. Summary rows report the distribution and the numbers of models that improve, remain unchanged, or decline.

Model	BUILD Δ	REVISE-CODE Δ	EXPLAIN Δ
`gemini-3.1-pro`	0	−7	−1.42
`deepseek-v4-pro`	−3	−5	−0.14
`gpt-5.5`	−7	−4	+0.92
`claude-sonnet-4.5`	−9	−3	+1.43
`qwen3-max`	−11	−7	−1.24
`gpt-4o`	−19	−14	+1.60
Average	−8.2	−6.7	+0.2
Min–max	[−19, 0]	[−14, −3]	[−1.42, +1.60]
Improved / unchanged / worse	0/1/5	0/0/6	3/0/3

A.3 REVISE CONTEXT MODEL-LEVEL DETAILS

Table 8 adds a model-level Gurobi view to the capability-stratified solver results in the main text, separating models that benefit from REVISE-ALL from those that do not.

The model-level deltas refine the aggregate conclusion: adding a formulation to code helps several strong agents but not all, while formulation-only context is usually less reliable than executable context.

Table 8: Representative model-level Gurobi REVISE context deltas. Models are grouped by whether adding the formulation improves over REVISE-CODE. Code, All, and Model are Pass@1 (%); deltas are relative to REVISE-CODE.

Model	Code	All	Model	All − Code	Model − Code
REVISE-ALL *improves over* REVISE-CODE					
`gemini-3.1-pro`	87	91	84	+4	−3
`gemini-3-flash`	82	89	70	+7	−12
`claude-opus-4.6`	87	93	81	+6	−6
`gpt-5.5`	85	90	85	+5	0
`gemini-2.5-pro`	81	89	71	+8	−10
`deepseek-v4-pro`	80	86	82	+6	+2
`qwen3-max`	68	71	71	+3	+3
REVISE-ALL *does not improve over* REVISE-CODE					
`qwen3.5-27b`	48	40	41	−8	−7
`qwen3-32b-thinking`	32	31	28	−1	−4

A.4 SOLVER-BACKEND SENSITIVITY DETAILS

Table 6 preserves every model row behind the grouped solver averages in the main text. Solver effects are heterogeneous, and a model can be stable under one backend while failing under another. Table 9 contrasts representative backend-sensitive and backend-stable BUILD results.

Backend robustness is not monotone in model scale, and stable models are not always the highest-scoring models. The main text therefore reports both the default Gurobi track and grouped results across all four backends.

Table 9: Representative model-level BUILD sensitivity across solver backends, contrasting backend-sensitive and backend-stable models. Entries are Pass@1 (%); Range is the max–min spread across the four backends.

Model	Gurobi	COPT	CBC	HiGHS	Range
Backend-sensitive models					
`deepseek-v4-pro`	67	49	54	57	18
Backend-stable models					
`gemini-3.1-pro`	72	73	72	66	7
`claude-opus-4.6`	62	70	64	67	8

A.5 TOKEN EFFICIENCY AND MODEL-LEVEL CONTEXT EFFECTS

The composite accuracy of model m in Figure 6 is the equal-weight mean of the three default-track scores:

$$S_m = \frac{1}{3}\big(\text{Pass@1}_{m,\text{BUILD}} + \text{Pass@1}_{m,\text{REVISE-CODE}} + \text{Rubric}_{m,\text{EXPLAIN}}\big).$$

All three terms are expressed on a 0–100 scale. Let $\mathcal{K}_m$ denote the evaluated instances with available or verified token usage. We compute

$$T_{mi} = T_{mi}^{\text{prompt}} + T_{mi}^{\text{completion}}, \quad \bar{T}_m = \frac{1}{|\mathcal{K}_m|}\sum_{i\in\mathcal{K}_m} T_{mi}, \quad E_m = \frac{S_m}{\bar{T}_m/1000}.$$

Thus, E_m measures composite accuracy per thousand mean tokens. Figure 6 includes all 16 evaluated models, with marker area encoding $\bar{T}_m$. The composite is used only for descriptive visualization, while the main evaluation reports each task separately.

Token efficiency need not rank models in the same order as accuracy: the frontier contains models with different balances between accuracy and token consumption, and higher token use does not uniformly translate into better workspace-level performance.

Models also differ in how they use legacy-code context. Some agents recover useful modeling structure, data joins, and indexing information from the existing implementation, whereas others inherit irrelevant imports or stale control flow.

A.6 FAILURE DETAILS AND CASE STUDIES

Figure 7 summarizes semantic failure categories across BUILD, REVISE-ALL, and EXPLAIN. Each audited failure is manually assigned to one mutually exclusive primary category based on the generated program or answer, execution trace, objective deviation, and workspace requirements. Each bar is normalized over the audited failures and therefore shows failure composition, not the model's failure rate. These categories complement automatic labels such as WRONGVALUE, RUNTIMEERROR, `module_not_found`, and NAMEERROR. The latter two often arise from inherited non-solver imports or heuristic identifiers reused without corresponding solver variables.

Table 10 traces representative BUILD failures from observed outcomes to workspace evidence and root causes, covering objective, parsing, data-mapping, and constraint errors from the same audit. Among `gemini-3.1-pro`'s 28 audited BUILD failures, the rounded shares are 21% modeling mistakes, 39% data-mapping errors, 29% hallucinated constraints, and 11% syntactic issues.

Table 10: Representative workspace-specific BUILD failures for `gemini-3.1-pro`.

Instance	Observed result	Workspace evidence	Root cause
`IndustryOR_2`	Wrong objective value: 25 vs. reference 125	`a1=10`, `a2=15`, and `training_capacity_per_jet=5` define trained-pilot capacity.	The agent introduced combat-jet variables and optimized `C1 + C2`, mapping training capacity to the wrong physical quantity.
`IndustryOR_9`	Runtime parser error before model construction	`general_parameters.csv` includes free-text business descriptions with commas in the context column.	The agent loaded the full CSV with `pd.read_csv`; the descriptive column triggered a Pandas C-parser error instead of being ignored or parsed robustly.
`IndustryOR_27`	Runtime KEYERROR: `unit_price_product_i`	The CSV keys use Roman numerals with case-sensitive suffixes, e.g., `unit_price_product_I`.	The agent normalized product names with `p.lower()`, creating keys that do not exist in the parameter file.
`IndustryOR_18`	Wrong objective value: 255000 vs. reference 5	The document asks the factory to fully utilize time, minimize overtime, and meet fabric sales targets.	The agent converted the task into profit maximization over fabric output, changing the business objective instead of only encoding the stated requirements.

The figure and case table separate three failure mechanisms: optimizing a plausible but unsupported business quantity, parsing semi-structured data too rigidly, and altering schema keys during normalization. All three can occur before or alongside otherwise valid mathematical modeling.

B SUPPLEMENTARY WORKSPACE AND DATASET VISUALIZATIONS

B.1 WORKSPACE ANATOMY

`OR-Space` contains three workspace families, each with 100 instances: `build_workspaces/`, `revise_workspaces/`, and `explain_workspaces/`. The tree below shows the completed workspace state after evaluation, not the agent-visible state at task start. Completed workspaces retain task inputs, generated outputs, and evaluator-side reference artifacts; the initial views follow Figure 3. Thus, BUILD withholds its source model, REVISE exposes only its designated context variant and withholds the revised reference implementation, and EXPLAIN provides the validated artifacts for both versions.

Figure 2 uses `IndustryOR_1` to illustrate this organization. It shows how a single task distributes evidence across business requirements, structured data, and legacy code, which an agent must connect in all three tasks.

The accompanying tree abstracts this instance into the completed evaluator-side workspace layout instead of enumerating all 300 task directories. File names may vary slightly across domains, but their roles remain fixed: `docs/` contains the natural-language requirements, `data/` stores numerical parameters, `src/` contains modeling code or reference implementations, `logs/` captures execution traces, and `solver_records/` stores the solver evidence used by EXPLAIN questions.

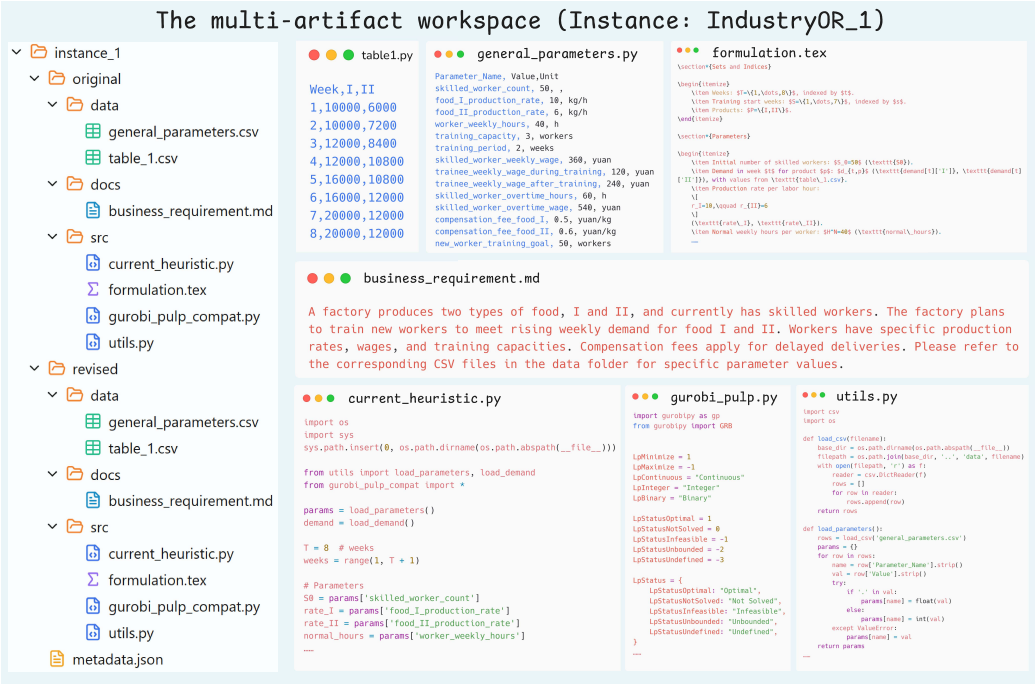

Figure 8: Enlarged view of the workspace in Figure 2. The left panel shows the file hierarchy of IndustryOR_1; the right panels show its requirement document, data tables, and legacy code.

Representative post-evaluation workspace layout

```
Workspace_OR/
|-- build_workspaces/
|   |-- instance_k/
|   |   |-- metadata.json
|   |   |-- _agent_solution.py
|   |   |-- data/    (general_parameters.csv, table_1.csv, ...)
|   |   |-- docs/    (business_requirement.md)
|   |   `-- src/     (current_heuristic.py, formulation.tex,
|   |                 gurobi_pulp_compat.py, utils.py)
|   `-- instance_1 ... instance_100
|
|-- revise_workspaces/
|   |-- instance_k/
|   |   |-- metadata.json
|   |   |-- original/ (data/, docs/, src/)
|   |   `-- revised/  (data/, docs/, src/)
|   `-- instance_1 ... instance_100
|
`-- explain_workspaces/
    |-- index.json
    |-- instance_k/
    |   |-- metadata.json
    |   |-- original/
    |   |   |-- data/  docs/  src/
    |   |   |-- logs/           (execution_record.json, solver_output.log)
    |   |   `-- solver_records/ (Gurobi log, LP relaxation, solution JSON)
    |   `-- revised/
    |       |-- data/  docs/  src/
    |       |-- logs/
    |       `-- solver_records/
    `-- instance_1 ... instance_100
```

B.2 Dataset Statistics

OR-Space contains 100 base industrial optimization topologies, each instantiated in Build, Revise, and Explain, yielding 300 task instances in total. The three task families are paired by topology, not sampled independently. Cross-task statistics compare original and revised formulations for the same underlying problems, reducing confounding from differences in the problem mixture.

The base topologies span manufacturing, logistics, energy, and finance and include structures such as multi-stage coupling and linearized relations. This diversity is not fully captured by raw model size: workspaces with similar numbers of variables and constraints may still differ in requirement structure, cross-file schema dependencies, and the extent to which a change propagates through existing code.

Figure 4 in the main text reports the distributions of decision-variable and constraint counts across tasks.

Revision difficulty also arises from structural change, not model size alone. Figure 9 compares each original problem with its revised counterpart along requirement length, code length, mathematical complexity, variable scale, and constraint density. The increases are uneven: requirement length and variable scale grow most, mathematical complexity roughly doubles, and code length and constraint density increase more moderately. A revision thus requires coordinated changes to the optimization model, not a local textual edit.

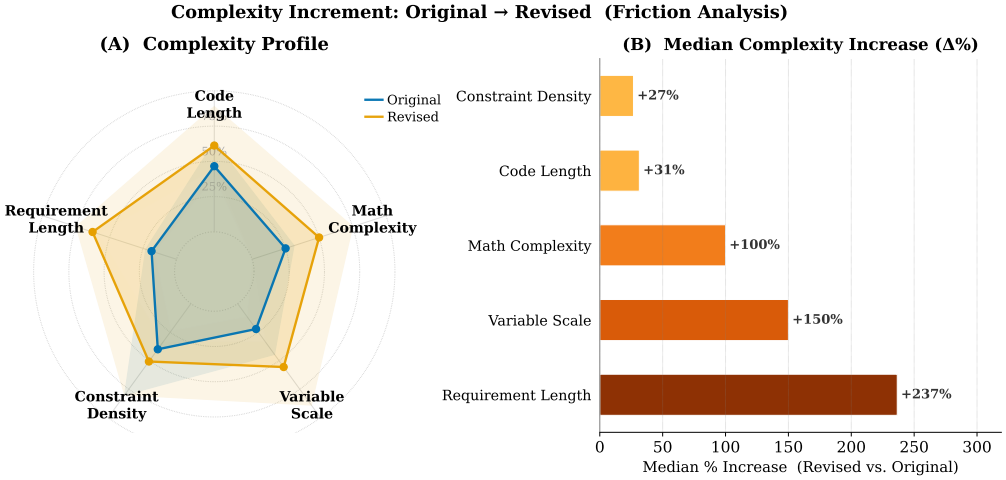

Figure 9: Original-versus-revised complexity profiles across five dimensions: requirement length, code length, mathematical complexity, variable scale, and constraint density. The left panel compares normalized original and revised profiles, while the right panel reports the median percentage increase from the original to the revised workspace.

Reference evidence paths. Figure 5 in the main text summarizes the reference paths generated by GPT-5.4 during question construction. Depth counts arrow-delimited evidence stages: `A -> B -> C -> D` has depth four.

C Explain Evaluation Rubric

We assess open-ended, short-answer responses in the **Explain** task with four positive components and a hallucination penalty, backed by a ground-truth checklist and an independent `gpt-5.1` judge. Each checklist criterion is typed as either `exact_match` or `llm_boolean_judgment`. Exact-match criteria—including required numeric values, variable and constraint names, and CSV columns—are scored programmatically after Unicode, whitespace, numeric-separator, and identifier-boundary normalization. Criteria requiring semantic interpretation are assessed by the independent judge against criterion-specific instructions and verified workspace evidence.

Exact Coverage gives equal weight to every required target entity in the `exact_match` criteria. If N is the number of target entities and $h_i \in \{0, 1\}$ is the deterministic match result for entity i, then $S_{\text{coverage}} = 35 \sum_{i=1}^{N} h_i/N$. For every `llm_boolean_judgment` item, the judge assigns coverage, causality, and grounding subscores in $\{0, 1\}$ and a depth subscore in $\{0, 1, 2\}$. The item score is $s_j = (c_j + a_j + g_j + d_j/2)/4$, and $S_{\text{reasoning}} = 35 \sum_j s_j/M$ over the M semantic items. Grounding and Answer Quality each contain four 0–3 anchored subdimensions, whose sums are linearly rescaled to 20 and 10 points. The judge separately classifies unsupported claims as *none*, *minor*, *material*, or *severe*, corresponding to penalties of 0, 5, 12, or 20 points. An empty unsupported-claim list forces the *none* level. The final score is

$$S_{\text{final}} = \text{clip}\left(S_{\text{coverage}} + S_{\text{reasoning}} + S_{\text{grounding}} + S_{\text{quality}} - P_{\text{hallucination}},\ 0,\ 100\right).$$

The quoted unsupported claims and assigned penalty level are retained in the per-instance audit record.

Table 11: Scoring rubric for the EXPLAIN task. The four positive components sum to 100 points; unsupported claims can subtract up to 20 points before clipping.

Dimension	Evaluation Criteria	Points
Exact Coverage	Includes the required atomic facts, including variable names, constraint names, CSV columns, units, and numeric values.	35
Reasoning	Correctly connects the facts through the relevant optimization logic, such as binding constraints, slack, objective changes, or revision effects.	35
Grounding	Grounds the explanation in the provided workspace artifacts, not generic OR knowledge or unstated assumptions.	20
Answer Quality	Gives a concise, coherent answer that directly addresses the question and uses clear terminology.	10
Hallucination Penalty	Subtracts credit for unsupported constraints, parameters, solver facts, numerical values, or causal claims.	up to -20

C.1 SCORING GUIDELINES

The judge applies the discrete hallucination levels above in addition to the four positive dimensions. For the final score $s = S_{\text{final}}$ after subtraction and clipping, the score bands are:

- $90 \leq s \leq 100$ **(Excellent):** The explanation covers the required facts, gives the correct causal or mathematical reasoning, grounds the answer in the relevant workspace artifacts, and avoids unsupported claims.
- $60 \leq s < 90$ **(Good):** The core answer is correct, but it may omit a supporting fact, give an incomplete reasoning chain, or provide weaker artifact grounding.
- $30 \leq s < 60$ **(Fair):** The response identifies part of the answer but gives vague, incomplete, or partially incorrect mathematical justification.
- $0 \leq s < 30$ **(Poor):** Irrelevant, contradictory to solver output, or ungrounded across files.

C.2 EVALUATION PROCEDURE

For each instance, the `OR-Space` agent pipeline exposes the visible original and revised documents, data, reference code, execution logs, and solver records together with the instance question. Answer generation and subsequent judging both use temperature 0.0. Responses terminated only by the output-length limit are regenerated with a larger completion budget before scoring.

Scoring combines deterministic and judge-assisted checks. Exact-match items are normalized for Unicode, whitespace, and numeric separators and matched with identifier boundaries. Semantic checklist items are evaluated by a blind `gpt-5.1` judge against their criterion-specific instructions, the complete candidate answer, and compact verified workspace evidence. The same judge and rubric are used for every evaluated model. Per-item decisions, judge rationales, unsupported-claim quotations, penalty levels, and all score components are stored for audit. The reported EXPLAIN score is the arithmetic mean of the clipped, penalty-adjusted totals over the 100 instances.

D Additional Related Work

Optimization modeling benchmarks. NL4Opt frames optimization modeling as semantic parsing from natural language to solver-ready linear programming representations (Ramamonjison et al., 2023). Mamo uses solvers to evaluate whether LLM-generated mathematical models produce correct numerical results across ordinary differential equations, linear programming, and mixed-integer linear programming (Huang et al., 2025b). ORLM studies data synthesis and open-source model training for optimization modeling through OR-Instruct and IndustryOR (Huang et al., 2025a). OptiMUS decomposes modeling into formulation, solver-code generation, debugging, solution evaluation, and refinement (AhmadiTeshnizi et al., 2024), while Chain-of-Experts coordinates multiple LLM experts and releases ComplexOR (Xiao et al., 2024). Recent datasets and training frameworks extend this direction: OptMATH scales bidirectional data synthesis for optimization modeling (Lu et al., 2025); DP-Bench/DPLM studies auto-formulation for dynamic programming (Zhou et al., 2025b); ORQA evaluates domain-specific OR reasoning and OptiGuide studies supply-chain optimization explanation (Mostajabdaveh et al., 2025; Li et al., 2023); OptiMind and StepORLM inject optimization expertise, solver verification, and process supervision into model training (Zhang et al., 2025b; Zhou et al., 2025a); OR-LLM-Agent, EvoOpt-LLM, InvEvolve, and Agora-Opt study agentic decomposition, model or policy evolution, memory, and debate for OR modeling and deployment (Zhang et al., 2025a; He et al., 2026; Huang et al., 2026; Lin et al., 2026); and MIPLIB-NL stresses industrial scale and model–data separation from real MIP instances (Li et al., 2026). `OR-Space` retains solver-grounded evaluation and provides workspaces with documents, data, code, and execution state for construction, revision, and explanation tasks.

Interactive agent benchmarks. SWE-bench evaluates repository editing against tests (Jimenez et al., 2024); WebArena evaluates browser agents in realistic websites (Zhou et al., 2024); AgentBench spans multiple interactive environments (Liu et al., 2025b); GAIA uses real files and tools for general-assistant tasks (Mialon et al., 2023); MLE-bench evaluates Kaggle-style machine-learning engineering (Chan et al., 2024); and τ-bench studies multi-turn tool-agent-user interaction under domain policies, with broader survey and ROI-centered perspectives emphasizing environment, metrics, and usability costs (Yao et al., 2024; Zhu et al., 2025; Liu et al., 2025a). OSWorld evaluates multimodal agents in real computer environments with file I/O and execution-based checks (Xie et al., 2024); WorkArena targets enterprise knowledge-work tasks in web software (Drouin et al., 2024); AppWorld evaluates interactive coding agents over realistic app APIs with state-based unit tests (Trivedi et al., 2024); and TheAgentCompany simulates a workplace where agents browse, code, run programs, and communicate (Xu et al., 2024). Work on agent externalization, protocols, and memory selection provides a systems perspective on how memory, skills, protocols, and harness design can change what an LLM agent can do in practice (Zhou et al., 2026; Yang et al., 2025). `OR-Space` differs in its oracle and state: correctness is mathematical consistency among requirements, data, the model, and returned solutions.

Benchmark validity and judge design. HELM argues for multi-metric reporting, not a single proxy for general capability (Liang et al., 2023). Dataset documentation work such as Datasheets for Datasets (Gebru et al., 2021), Data Cards (Pushkarna et al., 2022), and Croissant (Akhtar et al., 2024) motivates explicit reporting of motivation, composition, collection, and intended use. `OR-Space` follows this practice by reporting Build, Revise, and Explain separately and tying each to its oracle. For Explain, we use an LLM-as-judge only for short workspace-grounded responses; the rubric is explicit to mitigate sensitivity to judge prompts, positional effects, anchoring, leniency, and other known evaluator biases (Zheng et al., 2023; Chiang et al., 2024; Liu et al., 2023; Kim et al., 2023; Wang et al., 2023; Stureborg et al., 2024; Thakur et al., 2024).

E Benchmark Construction and Reproducibility

E.1 Release, Reproducibility, and Responsible Use

Public artifacts. The benchmark data, generation code, and evaluation code are available in the anonymous artifact linked on the first page. The release contains the Build, Revise, and Explain workspaces, split metadata, objective references, Explain rubrics, public scoring code, and selected construction and evaluation prompts.

Compute and uncertainty. BUILD and REVISE submissions run in fresh Python subprocesses with a 180 s wall-clock limit per submission. Gurobi 13.0.1 (Gurobi Optimization, LLC, 2024) is used for oracle generation and the main backend, while solver-robustness tracks rerun selected settings with COPT (Ge et al., 2022), CBC through PuLP, and HiGHS. LLM inference for hosted models is performed through provider APIs, so provider-side hardware is not controlled by the authors. The main Pass@1 numbers are fixed-split descriptive rates over $n = 100$ instances; one solved instance corresponds to one percentage point, and a binomial standard error can be read as $\sqrt{p(1-p)/100}$ for a rate $p \in [0, 1]$. We therefore interpret small differences cautiously and emphasize large cross-task gaps and qualitative failure modes.

Licensing and responsible use. The seed IndustryOR topologies are cited in the paper and are used under their published non-commercial research terms; the `OR-Space` release uses a compatible research license and does not redistribute proprietary solver binaries or API credentials. The benchmark contains synthetic workspaces and is not intended to include personal or sensitive information. The benchmark is intended to support reproducible evaluation of optimization agents and to expose failures before deployment; high benchmark scores should not be treated as certification for production optimization decisions. Generated models and solver outputs therefore require independent inspection before operational use.

E.2 DATASET CONSTRUCTION AND PROMPTING TEMPLATES

This section summarizes the prompt templates used by the LLM calls in `OR-Space`. The templates are condensed for presentation. The public package includes selected prompts used by its released construction and evaluation utilities, while the templates below document the broader construction and scoring protocol. The prompts cover four groups: (i) **dataset generation** (BUILD workspace generation, L5 REVISE generation, and business-style rewriting), (ii) **quality control** (revision difficulty and business-style quality), (iii) **benchmark evaluation and scoring** (EXPLAIN answer scoring), and (iv) **EXPLAIN question generation**. Placeholder `{...}` fields are filled in at runtime from the workspace instance.

All calls use low-temperature decoding (temperature $\leq$ 0.3; 0.0 for judges) and `response_format=json_object` whenever a strict JSON return schema is declared.

E.2.1 DATASET GENERATION PROMPTS

P1. BUILD workspace generation. Given a raw IndustryOR text instance, a generator LLM emits a complete workspace $(\mathcal{D}, \mathcal{P}, \mathcal{S})$ as strict JSON. The key constraint is that $\mathcal{S}$ is split into a two-file solver (`current_heuristic.py` + `utils.py`) that reads CSV data from `../data/` and prints a single line `OBJECTIVE_VALUE: <v>`.

P1: BUILD workspace generator

```
You generate OR benchmark workspaces. Convert a clean OR problem statement into an
executable workspace with three folders: docs/, data/, src/.

Strict requirements:
1. docs/business_requirement.md states the business rules and may restate necessary
quantitative requirements, but it does NOT duplicate raw parameter tables.
2.    data/*.csv  are  the  canonical  source  of  numeric  parameters.    Use
general_parameters.csv for scalars and table_{k}.csv for indexed data.
3. src/current_heuristic.py reads CSVs from ../data/, builds a PuLP model, solves
with commercial solver backends (e.g., via pulp.GUROBI_CMD or pulp.COPT), and
prints the single final line OBJECTIVE_VALUE: <value>.
4. Helper math (e.g. big-M derivation, index construction) lives in src/utils.py
and is imported by current_heuristic.py.
5. Running cd src && python current_heuristic.py must reproduce the ground-truth
objective within 10^-3 relative tolerance.

Return strict JSON:
{
```

```
"docs": {...},
"data": {...},
"src": {...},
"run": {
"run.sh": "cd src && python current_heuristic.py"
},
"evaluation": {
"ground_truth": <float>
}
}
```

The 10^{-3} condition in P1 is a generation-time quality gate used to accept workspace drafts; benchmark submissions are scored by the objective oracle with the tolerance ϵ defined in Equation 2.

P2. L5 REVISE generation. L5 REVISE generation uses two calls: a generator and an independent verifier. The generation prompt embeds the full L5 specification: a quantitative operator profile (counts of $+var$, $-var$, $+c$, $-c$, mod c, modobj, mod data), an "L5 iff" criterion, and three worked templates (multi-stage decisions with shared resources, linked if-then constraints, piecewise-linear objective transform). A condensed form of the generation prompt is shown below; the full text is released as `02_revise_modeling/prompts/regenerate_revise_prompt.md`.

P2a: L5 REVISE generator (condensed)

```
You generate one L5 revise of a given OR instance.

Design principles (all hard):
• Business need must be unambiguous: every new parameter has a numeric value in a
CSV and a verbal definition in business_requirement.md; no "reasonable values", no
unnamed penalties.
• Difficulty arises from structure, not prose opacity. Trigger at least one of
{coupling, cascading, implicit_dependency}.
• L5 iff distinct_ops ≥ 4 OR (coupling ∧ cascading ∧ op_count ≥ 5).
• Revised GT must differ from the original; the problem stays feasible and bounded;
CBC solves within 60 s.
• Start revise_description with a business motivation sentence; write ≤ 250 chars;
reference every new parameter by name.

Return a strict JSON with the following structure:
{
"instance_id": ...,
"revise_type_name": ...,
"revise_description": ...,
"metadata": {
"diff": { "operator_counts": ..., "booleans": ..., "complexity_level": ... }
},
"original_workspace": { "docs": ..., "data": ..., "src": ... },
"revised_workspace": { "docs": ..., "data": ..., "src": ... },
"evaluation": {
"original_ground_truth": ...,
"revised_ground_truth": ...
}
}

Follow templates A (multi-stage + shared resource), B (activation + if-then), or C
(piecewise objective + variable collapse); do not output commentary.
```

P2b: Independent revision verifier

```
You are an independent OR modeller. Solve only from docs/business_requirement.md
and data/*.csv. You may NOT read the generator's src/. Produce your own PuLP script,
```

print OBJECTIVE_VALUE: <v>, and stop. Your objective must agree with the generated revised ground truth within 10^{-3} relative tolerance; otherwise the instance is rejected.

As in P1, the 10^{-3} verifier threshold is used during dataset construction to reject inconsistent generated workspaces; benchmark submissions are scored with the objective tolerance ϵ used throughout the main evaluation.

P3. Business-style requirement rewrite. The *mechanical* revised description (which explicitly names constraints, penalties, and variables) is rewritten into an authentic business-style version, producing the $W_{\text{rev-b}}$ workspace. The rewrite removes modeling meta-language while preserving the quantitative requirements and parameter identifiers needed to recover the same model; CSV files remain the canonical parameter source.

P3: Business-style requirement rewriter

You are a business analyst documenting an operational change for an OR team. Rewrite the revised business_requirement.md so it reads like its build-time sibling (same voice, same paragraph style).

Forbidden meta-language: "add a constraint", "modify the objective", "introduce a variable", "penalize . . . ", "this revision", "the model must", "subject to", "in addition to the original problem", explicit MILP / big-M / decision-variable / objective-function vocabulary.

Motivation first: introduce each new restriction or objective change through a credible, domain-appropriate business reason, then state the rule in business language while preserving its mathematical meaning.

Preserve every concrete number, parameter name, unit, threshold, time window, and penalty coefficient from the mechanical revision. When it names a parameter key, retain that key in backticks; the corresponding CSV remains the canonical source used by the program.

Length: 2–5 paragraphs and at most $5\times$ the original build description; avoid padding or boilerplate.

E.2.2 Quality-Control Prompts

P4. Revise difficulty assessment. Each $(W_{\text{orig}}, W_{\text{rev}})$ pair is scored by an LLM judge on six archetypes (A–F) with score caps and three 0–5 axes: coupling, cascading, implicit dependency.

P4: Revise difficulty judge

You are a rubric difficulty judge for OR revise tasks. Read the original workspace, the revised workspace, and the computed diff. Return STRICT JSON:
{"archetype":"A|B|C|D|E|F",
"coupling":0..5, "cascading":0..5, "implicit_dependency":0..5,
"composite":<sum>, "tier":"L1|L2|L3|L4|L5", "justification":"..."}

Archetype caps (hard):
A (Parameter tweak) ≤ 2 on each axis; B (Local bound) ≤ 3; C (Business rule) ≤ 4; D (Penalty/fixed cost) ≤ 4; E (Objective transform) ≤ 5; F (Scenario restructuring) ≤ 5.
Tier: composite $\leq 4 \Rightarrow$ L1; 5–7 L2; 8–10 L3; 11–13 L4; ≥ 14 L5.
Axes definitions: coupling = a new/removed variable forces existing constraints to change; cascading = one edit propagates through the existing model topology; implicit_dependency = the change is stated via business semantics and must be inferred into a structural edit.

P5. Business-style quality assessment. Each rewritten instance is scored 1–5 on five axes (D1 Style, D2 No-Meta-Guidance, D3 Numerical Specificity, D4 Motivation, D5 Inference Difficulty). Instances below 4 on D1–D4 go through the rescue loop (P3 re-run).

P5: Business-style quality judge

```
You evaluate a REVISED business description against the ORIGINAL build-style
description of the same problem. For each of the five dimensions below, return an
integer score 1–5 and a one-sentence justification.

Evaluation Dimensions:
D1 Style: does it sound like the build-time sibling document (tone, paragraph
shape, level of formality)?
D2 No-Meta-Guidance: does it avoid instructing the modeller (no "add a constraint",
no "penalize ...", and no MILP vocabulary)?
D3 Numerical Specificity: are quantitative limits, ratios, thresholds, time
windows, and penalties stated with concrete values or named parameters rather than
vague wording?
D4 Motivation: is there a credible operational reason for the change (supplier
renegotiation, rota policy, etc.)?
D5 Inference Difficulty: after reading, does the modeller still have to infer the
constraint/cost? (Score low if the text paraphrases the equation; score high if
authentic inference is required.)

Return STRICT JSON:
{
"D1_business_style": n,
"D2_no_meta_guidance": n,
"D3_numerical_specificity": n,
"D4_motivation_plausibility": n,
"D5_inference_difficulty": n,
"rationale": "..."
}
```

E.2.3 Benchmark Evaluation and Scoring Prompts

P6. Explain answer scoring. Each open-ended Explain item has a ground-truth checklist of atomic criteria. Criteria labeled `exact_match` (e.g., numeric values, variable or constraint names, and CSV columns) are scored programmatically after text normalization. Separately, the judge receives all `llm_boolean_judgment` criteria, compact verified workspace and solver evidence, and the complete candidate response. It supplies the Reasoning, Grounding, Answer Quality, and Hallucination Penalty components in Appendix C.

P6: Explain answer judge

```
You are a STRICT evaluator for an OR-Space Explain answer. Exact Coverage is scored
deterministically outside this call.

For each Boolean-checklist item, return four subgrades with a short verbatim
evidence quote from the candidate answer: sub_coverage (0 or 1) for required
entities, names, or numbers; sub_causality (0 or 1) for the required causal or
logical link; sub_grounding (0 or 1) for supporting code symbols, CSV fields, or
solver-record fields; and sub_depth (0, 1, or 2), where 0 is shallow, generic, or
wrong, 1 is correct but terse, and 2 is correct and explains the mechanism. The
scorer computes (sub_coverage + sub_causality + sub_grounding + sub_depth/2)/4 for
each item.

Score four Grounding subdimensions from 0 to 3:
code_or_model_structure, data_or_numeric_evidence, solver_record_evidence, and
business_or_formulation_link. Score four Answer Quality subdimensions from 0 to
3: directness, causal_explanation, organization, and internal_consistency. Use 0
for absent, 1 for weak or generic, 2 for concrete and correct, and 3 for concrete,
correct, and well integrated. Each Grounding subdimension must include a verbatim
evidence quote; if no quote is available, reduce the score by one level.
```

```
Also return an unsupported-claim assessment with level none|minor|material|severe
and quoted claims. The scorer maps these levels to penalties of 0, 5, 12, and
20 points, respectively. Use minor for unsupported names or numbers that do not
change the conclusion, material for a specific unsupported factual claim that
changes the interpretation, and severe when the explanation is largely unsupported
or contradicts the workspace; an empty claim list forces none. Return strict JSON
containing reasoning_items, grounding, answer_quality, hallucination, and notes.
Base every judgment only on the supplied question, checklist, complete candidate
answer, and verified evidence excerpts.
```

E.2.4 EXPLAIN QUESTION-GENERATION PROMPTS

P7. EXPLAIN seed-question generation. EXPLAIN seed MCQs are generated by three parallel prompts, each targeting a different reasoning dimension. Each generator uses the same four option roles: correct, plausible but incomplete, irrelevant technical terminology, and logical inverse. It also emits a `reasoning_explanation` under which only the *correct* option survives a multi-hop trace $\mathcal{D} \to \mathcal{S} \to \mathcal{P} \to \mathcal{M}$.

P7a: Cross-file alignment question generator

```
Design a 4-option question asking what a CODE expression physically represents.
The correct answer must RECONCILE business_requirement.md (business meaning) +
code (mathematical definition) + CSV (units).

Option roles:
A. Correct: exact physical quantity with correct unit, reconciling md + py + csv.
B. Plausible but incomplete: looks only at code OR only at doc; misses the CSV
unit subtlety.
C. Irrelevant technical terminology: invokes an OR term (network flow, KKT
stationarity, dual simplex) that does NOT apply.
D. Logical inverse: swaps two entities (I vs II, capacity vs demand) or flips
direction.

Return strict JSON:
{
"question": "...",
"options": {
"A": "...",
"B": "...",
"C": "...",
"D": "..."
},
"reasoning_explanation": "..."
}
```

P7b: Solver-evidence question generator

```
Based on the SOLVER FACTS (constraint name, RHS, LP dual, slack, binding flag,
MIP vs LP objectives, integer gap, problem sense), design a 4-option question on
shadow-price meaning and its implication for a ±1 change in RHS.
Option roles:
A Correct: integrates slack=0 + dual sign + LP→MIP caveat correctly.
B Plausible but Incomplete: correct for LP but IGNORES the integer gap or the
shadow-price validity range.
C Irrelevant Technical Terminology: misuses KKT / complementary slackness / Benders
terminology.
D Logical Inverse: flips the sign of the dual OR the direction of RHS change.
```

P7c: Revision-effect question generator

```
Given (orig workspace, revise diff, revised workspace, revised MIP state), design
a 4-option question asking what structural effect the revision has on the feasible
region or on a chosen big-M.
Correct answer must trace a causal chain revise_description → added/removed
variable → existing constraint that tightens → binding/slack change. Incorrect
options use the same roles as P7a.
```

P8. Open-ended question conversion. Each MCQ is converted into a free-form question with a typed ground-truth checklist. The conversion prompt below declares the checklist schema explicitly so that deterministic exact matching and P6 semantic judging remain separate.

P8: Open-ended question and checklist generator

```
Convert the MCQ (question, correct option A) into a free-form question that elicits
the same reasoning without any options.

Also emit FOUR checklist criteria that a correct answer must contain:
1-2. two exact: required values, units, variables, constraint names, CSV columns,
or business terms;
3-4. two semantic: the required cross-artifact causal chain and its optimization
interpretation or qualification.
Set checklist_type to exact_match for the first two criteria and
llm_boolean_judgment for the last two. Assign each criterion a stable ID.

Return strict JSON:
{
"question": "...",
"expected_short_answer": "...",
"checklist": [
{ "criterion_id": "c1", "checklist_type": "exact_match", "criterion": "...",
"target_entities": ["..."] },
{ "criterion_id": "c2", "checklist_type": "exact_match", "criterion": "...",
"target_entities": ["..."] },
{ "criterion_id": "c3", "checklist_type": "llm_boolean_judgment", "criterion":
"...", "instruction_for_judge": "..." },
{ "criterion_id": "c4", "checklist_type": "llm_boolean_judgment", "criterion":
"...", "instruction_for_judge": "..." }
]
}
```

After validation, 97 released rubrics retain all four criteria. In three instances, one unsupported or redundant semantic criterion was removed, so the public release contains 200 exact-match and 197 semantic criteria.

Reproducibility note. Dataset-construction calls record the model id, temperature, rendered user message, and raw reply in stage-specific logs. Evaluation runners additionally store the rendered prompt, generated code or answer, execution output, and per-instance score under task- and model-specific result directories. For BUILD and REVISE, the OR-Space agent pipeline presents the visible workspace with its original paths and the solver-specific interface requirements. The third-party agent frameworks in Section 4.2 are pointed at the same workspace and are scored by the same oracle.

F EXPERT ASSESSMENT: CALIBRATION AND REPRESENTATIVENESS

We compared automated EXPLAIN scores with independent assessments by two OR experts on 90 responses to 30 questions. The study measured score agreement and the judge's ability to detect substantive mathematical errors. Table 12 reports the results.

F.1 Sampling and Blinding

We sampled 10 questions uniformly without replacement from each of the three question types: comparative dynamics, dual-theory soundness, and logical–physical alignment. For each question, we included responses from `gpt-5.5`, `deepseek-v4-pro`, and `qwen3-14b`, giving 90 responses in total. We selected these models before inspecting disagreements between the judge and experts, to cover closed- and open-source families at three capability levels. We retained all selected responses, including empty, incorrect, or difficult ones.

The sampling seed was `20260925`. The fixed manifest `calib_explain_v1` records the seed, question and response identifiers, and dataset revision. Calibration used the same dataset revision and released `gpt-5.1` judge version as the main experiments.

Each annotation packet contained the question, original and revised requirements, data, code, solver records, scoring criteria, and an anonymized response. We hid model identities, automated scores, and judge rationales, and randomized response order separately for each expert.

F.2 Independent Expert Assessment

Both experts hold advanced degrees in operations research and have at least five years of applied LP/MILP modeling experience, including interpreting commercial solver output. Neither expert authored `OR-Space` instances; one reviewed the rubric wording during construction but did not participate in automated scoring. Before the rubric was frozen, both experts completed a familiarization set of five held-out questions: two on comparative dynamics, two on dual theory, and one on logical–physical alignment. These questions were excluded from the calibration sample.

The experts then rated each response independently. They assessed every atomic checklist criterion, including whether exact-match entities were used correctly in context, and assigned Reasoning, Grounding, Answer Quality, and Hallucination Penalty scores. We calculated Exact Coverage and the clipped total using the weights in Appendix C. Each expert also flagged substantive mathematical errors, including invalid sensitivity claims, contradictions of solver evidence, and unsupported causal explanations that changed the answer. Purely stylistic differences did not count as errors. We retained the independent ratings to measure agreement between experts, then adjudicated disagreements to obtain the reference for comparison with the automated judge.

The annotation form recorded defects in questions, evidence, or reference criteria separately from errors in model responses. Under the protocol, unresolved items would remain in the audit record but be excluded from judge–expert accuracy estimates, with counts and reasons reported. Any rubric correction would require rescoring all compared responses. No such correction was needed in this sample.

F.3 Agreement and Error Detection

We report mean absolute error (MAE) for total and dimension scores, rescaling each dimension to $[0, 100]$ for comparison. Signed total-score bias is the mean automated score minus the adjudicated expert score. For atomic-decision agreement, we calculated the fraction of matching checklist decisions within each response, then averaged across responses so that each response received equal weight.

We counted a response as flagged by the judge only if the saved record explicitly identified a substantive error. A low score alone did not count. Two independent codings, blind to model identities and expert labels, determined these flags. Sensitivity is the fraction of responses with expert-identified errors that the judge flagged; specificity is the fraction without expert-identified errors that it left unflagged. We report the counts behind both rates, treating a rate as undefined if its denominator is zero. We also audited whether the judge identified the same error as the experts, since matching flags can rest on different rationales.

We estimated 95% confidence intervals from the 2.5th and 97.5th percentiles of 10,000 bootstrap resamples. We resampled base problems within each question type, keeping all responses and expert ratings for a problem together. We also examined agreement by question type and model, and paired model-score differences on the same questions.

Table 12: Expert calibration of the EXPLAIN evaluator. All score errors are reported on a 0–100 scale; agreement and detection rates are percentages. N counts paired responses, except that sensitivity and specificity use expert-positive and expert-negative responses, respectively. Confidence intervals resample base problems.

Measure	N	**Estimate**	**95% CI**
Expert–expert total-score MAE ↓	90	4.6	$[3.7, 5.6]$
Expert–expert atomic agreement ↑	90	88.2	$[85.1, 91.0]$
Judge–expert total-score MAE ↓	90	6.9	$[5.6, 8.4]$
Judge–expert total-score bias	90	-0.9	$[-2.3, 0.5]$
Judge–expert Reasoning MAE ↓	90	8.3	$[6.6, 10.1]$
Judge–expert Grounding MAE ↓	90	9.2	$[7.3, 11.2]$
Judge–expert atomic agreement ↑	90	83.7	$[80.6, 86.5]$
Substantive-error sensitivity ↑	24	75.0	$[53.3, 90.2]$
Substantive-error specificity ↑	66	92.4	$[82.9, 97.5]$

F.4 CALIBRATION FINDINGS

No question-level or reference-criteria defects were reported, so all 90 responses were included in Table 12.

The two experts' total scores differed by an MAE of 4.6 points, with 88.2% atomic-decision agreement. Against the adjudicated reference, the automated judge had a total-score MAE of 6.9 points and atomic agreement of 83.7%. Its signed bias was -0.9 points; the 95% confidence interval $[-2.3, 0.5]$ included zero. Reasoning and Grounding MAEs were 8.3 and 9.2 points, respectively, after rescaling to 0–100.

The experts found substantive errors in 24 responses. The judge flagged 23 responses: 18 with expert-identified errors and 5 without them. It therefore flagged $18/24$ responses with errors (sensitivity 75.0%) and left $61/66$ responses without expert-identified errors unflagged (specificity 92.4%). The false-positive rate was 7.6%. False positives were concentrated in `qwen3-14b` responses, where the judge sometimes treated a vague argument as unsupported even when it contained no identified error. The audit found no incorrect rationale among the 18 true positives.

Total-score MAE also varied by question type: 5.8 points for logical–physical alignment, 6.7 for comparative dynamics, and 8.1 for dual-theory soundness. These results quantify the judge's disagreement with experts on the sampled responses. The judge missed six of the 24 responses with substantive errors, and the sensitivity estimate has a wide 95% confidence interval ($[53.3, 90.2]$). Automated EXPLAIN scores should therefore be interpreted alongside the qualitative failure analysis; calibration on this sample is insufficient to validate rankings across all 16 models.

F.5 PRACTITIONER ASSESSMENT OF TASK REPRESENTATIVENESS

`OR-Space` workspaces are synthesized from existing optimization problem structures (Section 3.3). To assess how well the resulting tasks reflect industrial OR work, we asked eight practitioners to evaluate 30 workspaces, with 10 from each task and stratification by formulation scale. All practitioners had at least three years of industrial LP/MILP modeling experience, and none had participated in benchmark construction. They rated five aspects of task representativeness on a 5-point Likert scale, completing their ratings independently before any discussion.

Table 13: Practitioner ratings of task representativeness ($K = 8$ raters, $M = 30$ workspaces, 5-point Likert scale). We report means, standard deviations (SD), and the percentage of ratings of 4 or 5.

Item	**Mean (SD)**	≥ 4
Requirement documents resemble real business specifications	4.1 (0.7)	78%
Workspace organization resembles a real project layout	3.9 (0.8)	70%
Revisions resemble realistic change requests	4.0 (0.8)	73%
Tasks are likely to arise in industrial practice	3.9 (0.7)	71%
Solving them requires skills used in practice	4.3 (0.6)	85%
Overall	*4.0 (0.8)*	*75%*

Table 13 shows that the skills required to solve the tasks received the highest mean rating (4.3), while workspace organization and the likelihood of encountering the tasks in industrial practice tied for the lowest (3.9). Agreement among raters was moderate (Krippendorff's $\alpha = 0.64$).