EVALUATING LLM-DRIVEN INTERACTIVE OPTIMIZATION MODELING: A KNOWLEDGE-GROUNDED BENCHMARK FOR CONVERSATIONAL SPECIFICATION RECOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent studies have explored the capability of Large Language Models (LLMs) for formulating and solving optimization problems. Real-world optimization modeling, however, often starts from vague and incomplete operational descriptions that must be clarified through interaction. In this work, we study whether LLMs can act as modeling experts and recover complete optimization specifications through multi-turn dialogue. We formulate this process as conversational specification recovery, where an LLM must identify missing requirements, ask questions, and reconstruct a complete problem under a fixed dialogue budget. To evaluate this capability, we introduce OptRecover, a knowledge-grounded benchmark with an integrated evaluation harness that combines reference-grounded answering from the hidden complete specification, semantic parsing of natural-language descriptions, and deterministic scoring of the resulting semantic gaps. A shared structured representation of optimization knowledge connects these components and enables controlled information removal and fine-grained evaluation. We instantiate the benchmark on matching and assignment problems, comprising 48 complete problems, 288 systematically constructed incomplete variants, and 336 structured ground-truth specifications. Our evaluation reveals substantial differences in specification recovery across LLMs and examines how performance varies across semantic dimensions, problem complexity, and missingness levels.

1 INTRODUCTION

Large language models (LLMs) are increasingly used to translate natural-language optimization problem descriptions into mathematical formulations and executable solver programs (Ahmaditeshnizi et al., 2024; Jiang et al., 2025; Huang et al., 2025). Beyond generating mathematical expressions, effective optimization modeling requires correctly understanding which decisions are controllable, which requirements are mandatory, and what makes one feasible plan preferable to another. Identifying such requirements is therefore an essential part of the modeling process.

However, the requirements underlying an optimization problem are not always fully specified in an initial problem description. For example, a manager might provide workloads and assignment costs without stating whether every available worker must be used. Similarly, a preparation fee might be mentioned without clarifying whether it is charged per job or once per dispatched worker. Such ambiguities concern policies that determine feasible plans or their costs, yet cannot be resolved from numerical data alone. A modeling assistant must therefore recover the intended requirements through conversational interaction and preserve them when assembling the final specification.

Research on language-based preference elicitation and clarification demonstrates how interaction can help establish user intent (Li et al., 2025; Zhang et al., 2025a). We study this capability in optimization as *conversational specification recovery*. Given an incomplete description of an optimization problem, the goal is to recover the underlying specification that defines its decisions, objectives, constraints, and rules. A successful recovery should restore missing requirements, preserve the information already provided, and avoid introducing unsupported assumptions. We evaluate an

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.

LLM-driven modeling assistant that acts as a virtual modeling expert: given an incomplete problem description, it interacts with a reference user to identify missing requirements and iteratively refine the specification.

The evaluation of this task introduces three connected challenges. First, it requires a reliable dataset of instances that contain complete specifications and controlled incomplete descriptions, where selected requirements are removed while the original problem remains coherent. Second, interactive evaluation requires a grounded answerer that can respond to arbitrary questions based on the complete specification while revealing the information necessary to resolve each inquiry. Third, measuring recovery quality requires semantic comparison between free-form descriptions and reference specifications beyond surface-level text matching.

We address these challenges with OptRecover, a knowledge-grounded benchmark built around a semantic representation of optimization specifications. We focus on matching and assignment problems spanning diverse resource, temporal, routing, and state-dependent mechanisms and introduce a representation organizing objects, decisions, compositional rules and associated parameters into a common semantic structure. Based on this representation, we develop a separate LLM-based evaluation harness that answers questions from the evaluated modeling assistant and parses its recovered description into structured semantics, which are then assessed by a deterministic scorer against the complete specification and the incomplete baseline.

In summary, we make the following three main contributions:

- **A dataset for conversational specification recovery.** We construct 48 complete problems across 12 operational scenarios and four complexity levels, together with 288 graded incomplete variants and 336 validated structured ground-truth specifications.
- **A knowledge-grounded representation for optimization specifications.** We introduce a representation that transforms natural-language problem descriptions into structured modeling semantics, providing a common basis for consistent specification interpretation and semantic-level comparison.
- **An integrated evaluation harness for interactive modeling.** We build an integrated LLM-based harness that combines reference-grounded answering and multistage semantic parsing with deterministic scoring for evaluation.

2 Related Work

2.1 LLMs for Optimization Modeling

LLM-based optimization modeling translates natural-language descriptions into mathematical formulations and executable solver programs. Existing work improves this process through structured representations (Ramamonjison et al., 2022; Jiang et al., 2025), agentic pipelines for modeling and programming (Xiao et al., 2024; Ahmaditeshnizi et al., 2024), and execution-aware validation or formulation search (Song et al., 2026; Astorga et al., 2025). Recent work has moved beyond prompt-only modeling toward trainable and verifiable systems. Data synthesis expands scarce description–formulation pairs (Huang et al., 2025; Yang et al., 2025; Lu et al., 2025), while instruction tuning and solver-informed reinforcement learning improve formulation generation through supervision and solver feedback (Jiang et al., 2025; Chen et al., 2025). Evaluation has similarly broadened to more diverse optimization settings (Huang et al., 2025; Yang et al., 2025; Lu et al., 2025; Mostajabdaveh et al., 2025) and more fine-grained assessment of formulation structure, equivalence, and validity (Xing et al., 2024; Zhai et al., 2025; Liu et al., 2026).

Despite these advances, existing modeling and verification methods generally assume that a problem description or formal model is already available. They evaluate whether given requirements and their inferable consequences are faithfully formalized. OptRecover instead studies settings where important operational policies are initially missing and cannot be uniquely inferred from the available description. It therefore evaluates whether interaction can recover these missing specifications before formalization and optimization.

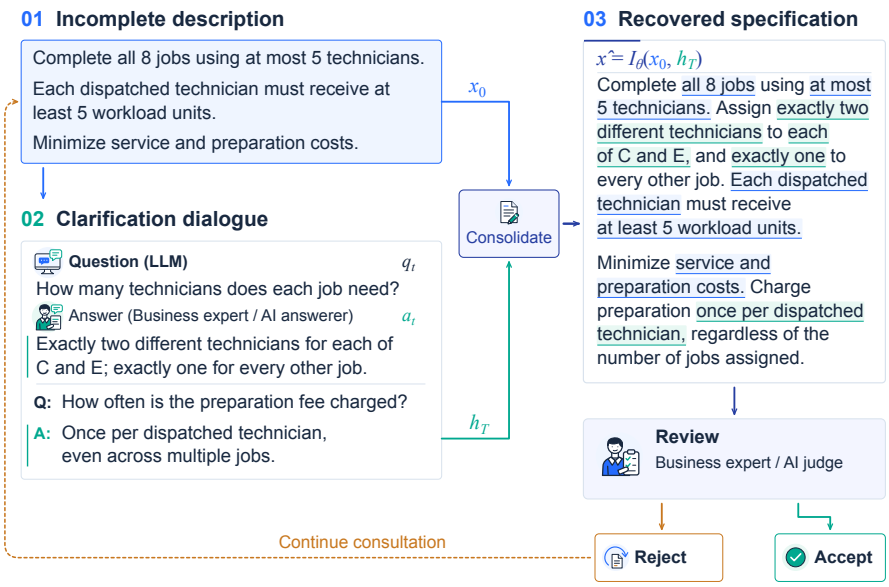

Figure 1: Conversational specification recovery in a technician-assignment example. Blue highlights mark retained information, and teal highlights mark information established through dialogue.

2.2 Interactive Clarification for Optimization

Clarification research studies how models acquire missing information through follow-up interaction, including rule-based question answering, preference elicitation, uncertainty over user intent, and anticipated future conversational outcomes (Saeidi et al., 2018; Li et al., 2025; Zhang & Choi, 2025; Zhang et al., 2025a). Related benchmarks evaluate ambiguity detection, clarification quality, and multi-turn ambiguity resolution (Zhang et al., 2024; Sarwar et al., 2026). In optimization, dialogue supports explanation, infeasibility diagnosis, and what-if analysis over existing models (Li et al., 2023; Chen et al., 2026; Zhang et al., 2025b), as well as preference elicitation and requirement clarification (Lawless et al., 2024; Xie, 2026). Simulated stakeholders further enable repeatable conversational evaluation under hidden preferences (Drossman et al., 2026).

These settings differ in whether interaction resolves ambiguity, elicits preferences, refines requirements, or revises an existing model, but they do not directly measure whether the final specification faithfully recovers the underlying problem requirements. OptRecover instead tracks what is initially provided, later disclosed, and ultimately retained, while detecting missing, altered, or unsupported requirements. We implement this through controlled incomplete descriptions, reference-grounded stakeholder answers, and semantic comparison of recovered specifications, enabling deterministic measurement of specification recovery under a fixed dialogue budget.

3 Conversational Specification Recovery

We consider an LLM-driven modeling assistant in recovering the specification of an optimization problem. The assistant starts from an incomplete optimization problem description x_0, which describes the operational setting but leaves some requirements or their details unspecified. Its task is to elicit this information through dialogue and integrate it with the initial description into a self-contained problem specification.

At round t, the assistant asks a natural-language question q_t based on x_0 and the preceding exchanges. The business expert, or an AI answerer acting in that role, responds with a_t. The accumulated dialogue is

$$h_t = ((q_1, a_1), \ldots, (q_t, a_t)). \tag{1}$$

Each answer informs subsequent questions, which may seek additional requirements or clarify the meaning, scope, or conditions of an earlier response.

Figure 1 illustrates this process for technician assignment. The initial description requires all eight jobs to be completed using at most five technicians, but leaves the staffing requirement of each job and the basis of the preparation fee unspecified. The dialogue establishes that jobs C and E each require two distinct technicians, whereas every other job requires one. It also clarifies that preparation is charged once per dispatched technician, regardless of how many jobs that technician receives.

After T rounds, the same LLM consolidates the initial description and the dialogue into a recovered specification,

$$\hat{x} = I_\theta(x_0, h_T), \tag{2}$$

where I_θ denotes the integration operation that combines the original description with the information established through dialogue.

In a real-world modeling workflow, the consolidated specification is then reviewed by the expert or an AI reviewer. Acceptance concludes the consultation, while remaining gaps or inconsistencies prompt further clarification and revision. Additional exchanges build on the information already established, and the assistant updates the specification accordingly.

4 Benchmark Construction and Evaluation

The benchmark combines complete and incomplete problem descriptions, a shared representation of domain knowledge, and an evaluation harness. All evaluated models use the same dialogue budget, answerer, integration instructions, and scoring policy.

4.1 Dataset Construction

We construct 48 complete problems spanning 12 scenarios in workforce and task allocation, service operations, review and admission, healthcare, warehousing and transportation, advertising, mentorship, team formation, and facility service. The problems share a matching-and-assignment core while incorporating eligibility and compatibility, capacities and quotas, activation and fairness rules, temporal and resource constraints, routing and split-service choices, and inter-period inventory and state transitions. Their composition varies in scope, activation conditions, and cross-rule dependencies, yielding substantially different specification structures. The 48 problems are evenly divided across four complexity levels (L1–L4), each containing 12 problems, with higher levels combining richer and more interdependent problem requirements.

Each complete problem yields six incomplete variants, with two at each of three missingness levels (Mild, Moderate, and Severe), producing 288 recovery instances. Missingness is introduced at the level of semantic facts rather than sentences. Removed content may include an entire rule, its applicability conditions, scope, counting or charging basis, or associated data. Dependent information is revised accordingly so that the remaining text does not reveal the hidden requirement. Multiple optimization experts construct and repeatedly cross-check the variants for naturalness, internal consistency, leakage of removed information, and unintended requirements. To characterize the constructed variants, Table 1 reports the average proportion of problem requirements removed at each missingness level.

We also provide a structured Golden for every complete and incomplete problem using the shared representation described below. Parser-assisted candidates are manually reviewed and revised against the corresponding descriptions by optimization experts. Review covers problem structure, rule semantics, scopes and activation conditions, and associated data. The resulting Goldens provide the reference knowledge for grounded answering and semantic evaluation.

4.2 Knowledge-Grounded Representation

We organize an optimization problem around a shared problem structure and the requirements defined over it (Figure 2A). The problem structure establishes the decision world of the problem: its objects, domains, decision relations, and decision attributes (Figure 2B). Decision relations describe which choices can be made among objects and domains, while attributes represent quantities, times, or other values determined through those choices. These shared identities provide a common founda-

Table 1: Dataset composition and construction-time omission proportions. Missingness columns report the average percentage of problem requirements removed, with variant counts in parentheses.

Complexity	Complete	Mild (%)	Moderate (%)	Severe (%)	Variants
L1	12	18.77 (24)	39.92 (24)	60.08 (24)	72
L2	12	19.19 (24)	39.71 (24)	60.29 (24)	72
L3	12	19.50 (24)	38.80 (24)	61.20 (24)	72
L4	12	20.99 (24)	40.17 (24)	59.83 (24)	72
All	48	19.61 (96)	39.65 (96)	60.35 (96)	288

tion on which objectives, constraints, conditions, and other problem requirements can be expressed consistently.

We represent recurring optimization mechanisms through 14 Blocks. Each Block defines a reusable logic together with the semantic roles needed to describe that logic. A Family instantiates a Block as one coherent policy within a problem by binding these roles to the relevant parts of the problem structure and specifying its scope and conditions (Figure 2C). A Family may therefore cover multiple closely related cases of the same policy. For example, a workload limit and a conditional higher limit for senior reviewers belong to the same Family when they jointly describe one workload policy. Policies that share a Block can have different operational meanings, while repeated applications of the same policy remain organized within a single indexed Family.

A Family links the policy to what it operates on, where it applies, what enters its computation, when particular cases take effect, and the values and evidence that support them (Figure 2D). Because these elements refer back to shared identities in the problem structure, decisions, attributes, conditions, and supporting values remain consistently connected across related requirements. The resulting representation organizes the domain knowledge that defines the optimization problem and serves as the common semantic basis for Goldens, grounded answering, and semantic comparison. Appendix A details the schema.

4.3 Evaluation Harness

Figure 3 instantiates the interaction protocol of Section 3 as an evaluation harness with three components: reference-grounded answering, recovered-description parsing, and semantic scoring. The complete reference K^* provides the private knowledge used for answering and evaluation, while the incomplete reference K_0^* represents the initial specification state. The evaluated LLM operates through the public interaction only, and the parser receives candidate text without access to either reference.

4.3.1 Reference-Grounded Answering

The answerer separates relevance judgment from evidence access. A router, the only answering LLM that reads the complete source x^*, maps the current question and public history to coarse information groups and passes forward only their identifiers and relevant public spans. A selector then works only with the structured cards and parameter menus exposed for those groups, choosing the relevant Families and any required parameter slices. A deterministic executor resolves the query against the reference, retrieves the requested facts and values, and includes the qualifiers needed to interpret them correctly. An expression model then turns the resulting evidence packet and public context into a natural-language answer. If the reference does not resolve a requested detail, the answer reports it as unknown. Final consolidation uses only x_0 and the public dialogue.

4.3.2 Recovered Description Parsing

The parser maps benchmark and recovered texts into the shared representation of Section 4.2 under the same generic semantic contracts. It first identifies the objects, domains, decision relations, and decision attributes that define the shared problem structure. It then identifies the relevant Blocks, extracts candidate Families, and reconciles repeated or disputed descriptions of the same policy. Each accepted Family is bound to the shared decisions and attributes together with its scope, computation,

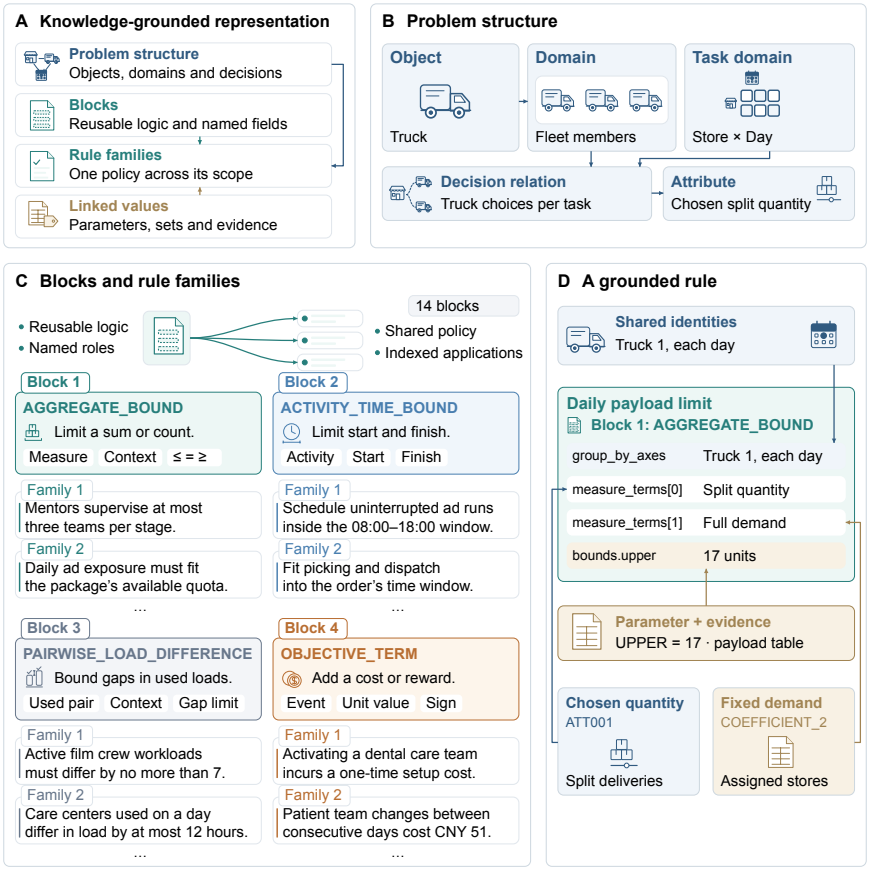

Figure 2: Knowledge-grounded representation. (A) Four linked components of the representation. (B) Shared problem structure defining the decision world. (C) Blocks and Families for organizing reusable logic and coherent policies. (D) A grounded Family linking its scope, computation, decisions, supporting values, and evidence.

conditions, and activation. A deterministic compiler checks these bindings, assembles the resulting structure, and creates the required parameter slots, which are then grounded to their semantic roles, source facts, indexing axes, units, and values. Appendix B gives the implementation details, and Section 5.5 evaluates parser reliability independently.

4.3.3 Semantic Evaluation

Evaluation is deterministic once a representation has been produced. After aligning object, decision, attribute, and parameter identities, the scorer decomposes the complete Golden into structural information groups with a total reference weight of 100. The same weights are used to compare both the incomplete state K_0^* and recovered representation $\widehat{K}$ with K^*.

Let C be correctly expressed reference weight, M missing or incorrect reference weight, and E erroneous or unsupported assertion weight, with $C + M = 100$. We define

$$S(K, K^*) = 100\frac{2C}{2C + M + E}, \qquad D(K, K^*) = 100 - S(K, K^*), \tag{3}$$

where D denotes semantic discrepancy. Missing or unresolved reference content contributes to M, explicit errors or unsupported assertions also contribute to E, and correct partial structures receive proportional credit. Recovery experiments use D to measure improvement from the incomplete baseline, while parser reliability reports S directly.

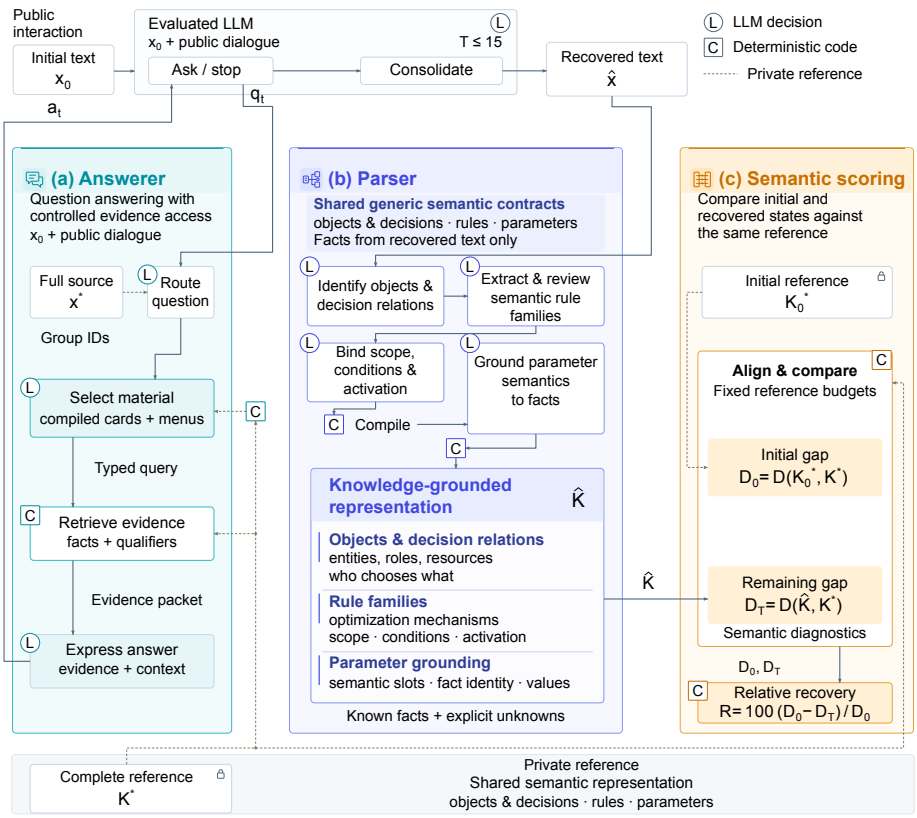

Figure 3: Evaluation harness. Controlled answering exposes selected evidence from the private reference, the parser maps text into the shared representation, and deterministic scoring measures the initial and remaining semantic gaps.

For an instance set $\mathcal{I}$, let

$$\overline{D}_0 = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} D(K^*_{0,i}, K^*_i), \qquad \overline{D}_T = \frac{1}{|\mathcal{I}|}\sum_{i\in\mathcal{I}} D(\hat{K}_i, K^*_i). \tag{4}$$

Specification Recovery Rate (SRR) is

$$\mathrm{SRR}(\mathcal{I}) = 100\frac{\overline{D}_0 - \overline{D}_T}{\overline{D}_0}, \qquad \overline{D}_0 > 0, \tag{5}$$

measuring the fraction of the initial discrepancy removed through interaction. Averaging precedes normalization, so SRR is computed from aggregate discrepancies rather than averaged per-instance ratios. Its denominator is the measured semantic gap rather than nominal missingness, and negative values indicate net degradation.

The same relative calculation yields five diagnostics: Information Coverage, Meaning Fidelity, Boundary Fidelity, Relational Fidelity, and Matching Precision. They are reported separately and are not combined into the overall score. Appendix C.3 gives the detailed semantic accounting for each diagnostic.

5 Experiments

5.1 Experimental Setup

We evaluate GPT-5.6 Sol, Claude Opus 5, DeepSeek V4.1 Flash, Qwen 3.8 Max, GLM-5.3, and Kimi K3 on the 288 incomplete variants. Each model conducts one dialogue per instance with up

Table 2: Specification recovery on the 288 incomplete instances.

Model	Overall ↑	Information Coverage ↑	Meaning Fidelity ↑	Boundary Fidelity ↑	Relational Fidelity ↑	Matching Precision ↑
Kimi K3	46.67%	48.42%	46.34%	48.28%	47.83%	27.86%
Claude Opus 5	66.14%	68.56%	64.39%	69.80%	69.00%	47.71%
GPT-5.6 Sol	47.77%	49.90%	46.50%	51.45%	49.71%	26.78%
DeepSeek V4.1 Flash	45.05%	46.22%	43.46%	46.77%	45.69%	31.07%
Qwen 3.8 Max	55.43%	57.98%	54.63%	59.45%	58.43%	32.42%
GLM-5.3	52.05%	55.22%	51.82%	55.73%	55.12%	26.43%

Table 3: Overall SRR by problem complexity.

Model	L1 ↑	L2 ↑	L3 ↑	L4 ↑
Kimi K3	61.95%	40.00%	45.13%	39.43%
GPT-5.6 Sol	72.72%	42.15%	42.38%	34.10%
Claude Opus 5	76.03%	55.23%	62.64%	69.35%
DeepSeek V4.1 Flash	62.68%	36.99%	45.72%	34.75%
Qwen 3.8 Max	67.28%	51.93%	51.56%	50.82%
GLM-5.3	72.23%	48.86%	43.91%	43.31%

to 15 clarification questions. The same model then consolidates the initial description and dialogue into a single recovered description. DeepSeek V4.1 Flash is used as the fixed backbone for both the answerer and parser throughout the recovery experiment.

Each recovered description is independently parsed three times by the fixed parser, with no reference access during parsing. We retain the parse with the highest overall agreement against the complete-problem Golden and take all diagnostics from that same parse. This best-of-three protocol reduces sensitivity to occasional parsing errors without regenerating the recovered description or selecting among multiple dialogues. We report SRR as the primary recovery measure, using the aggregation defined in Section 4.3.3.

We separately evaluate parser reliability across Kimi K3, DeepSeek V4.1 Flash, and Qwen 3.8 Max on the 48 complete descriptions. Each description is parsed once per backbone using the same parser pipeline. Overall agreement and its diagnostics are computed against the complete-problem Golden and averaged equally across descriptions.

5.2 Overall Specification Recovery

Table 2 summarizes specification recovery over all 288 incomplete instances. Claude Opus 5 achieves the highest overall SRR at 66.14%, followed by Qwen 3.8 Max at 55.43% and GLM-5.3 at 52.05%, while the other three models cluster at roughly 45–48%. The first four diagnostics show a broadly consistent pattern across models, indicating that stronger overall recovery is accompanied by improvements across multiple aspects of specification fidelity. Matching Precision remains consistently lower, revealing a common weakness in recovering exact structural correspondences.

5.3 Recovery across Problem Complexity

Table 3 reports overall SRR across the four complexity levels. Recovery is highest at L1 for every model, while performance is generally lower as specifications become richer and more interdependent. The size of this decline varies considerably across models: some retain relatively stable recovery at the higher levels, whereas others deteriorate more sharply. Overall, the results show that specification complexity poses a substantial challenge, with different models exhibiting different levels of sensitivity to it.

Table 4: Overall SRR by missingness level.

Model	Mild ↑	Moderate ↑	Severe ↑
Kimi K3	22.95%	48.88%	52.73%
GPT-5.6 Sol	11.84%	51.98%	56.35%
Claude Opus 5	37.84%	66.00%	75.29%
DeepSeek V4.1 Flash	14.77%	50.48%	50.97%
Qwen 3.8 Max	21.55%	59.15%	63.68%
GLM-5.3	28.07%	56.44%	56.67%

Table 5: Parser agreement with the complete-problem Goldens.

Model	Overall ↑	Information Coverage ↑	Meaning Fidelity ↑	Boundary Fidelity ↑	Relational Fidelity ↑	Matching Precision ↑
Kimi K3	96.67	96.42	95.71	94.88	97.13	97.29
DeepSeek V4.1 Flash	96.40	96.12	95.62	94.60	96.68	97.04
Qwen 3.8 Max	96.40	95.95	95.36	94.32	96.68	97.27

5.4 Recovery across Missingness Levels

Table 4 shows a distinct pattern across missingness levels. Every model achieves substantially lower SRR on Mild instances than on Moderate or Severe instances, while the latter two levels are much closer. This pattern follows directly from the relative definition of SRR. Mild instances begin with smaller semantic gaps, so leaving even a small number of requirements unrecovered can account for a large share of the remaining discrepancy. Higher SRR on more incomplete instances therefore reflects a larger fraction of the initial gap being recovered.

5.5 Parser Reliability across Backbones

Table 5 evaluates the parser independently on the 48 complete problems. Overall agreement is uniformly high and tightly clustered across the three backbones, ranging from 96.40 to 96.67. The individual diagnostics are similarly consistent, with Boundary Fidelity remaining the lowest across all three backbones at 94.32–94.88. The narrow variation across models indicates that the parsing pipeline behaves consistently across backbones on complete specifications, providing a stable semantic basis for the recovery evaluation.

6 Conclusion

We introduced OptRecover, a knowledge-grounded benchmark built around a semantic representation of optimization specifications. It evaluates whether LLMs can identify missing objectives, constraints, and modeling requirements through dialogue and consolidate them into a complete specification. The semantic representation provides a common basis for interpreting and comparing optimization specifications, while the integrated evaluation harness combines reference-grounded answering, recovered-description parsing, and semantic scoring. We then evaluated six LLMs on all 288 incomplete variants and separately assessed parser reliability using three different LLMs. The results show substantial variation in recovery performance across models: performance generally declines as specifications become richer and more interdependent, while the subtler omissions in mild missing instances are harder for models to identify and recover. The parser study shows broadly consistent agreement across the three tested LLMs, suggesting that the main evaluation trends are not highly sensitive to the parser model. Together, these findings highlight specification recovery as an important stage of optimization modeling: reliable formalization depends on identifying the intended requirements and preserving them throughout the interaction.

AI USE STATEMENT

Large language models (LLMs) were used to assist with language editing, rewriting, and improving the clarity and presentation of the manuscript. The core research ideas were developed by the authors, and the dataset problems were designed by experts in operations research. LLMs are also used as experimental components, as described in the paper. All AI-assisted text was reviewed by the authors, who take full responsibility for the final manuscript, methodology, data, code, and reported results.

ETHICS STATEMENT

This benchmark evaluates whether models accurately recover the stated requirements of optimization problems. It does not assess whether those requirements are fair or appropriate in practice. For example, a model receives credit for correctly recovering a rule about which employees may work a particular shift, without implying that the rule itself is fair.

REPRODUCIBILITY STATEMENT

The OptRecover dataset is available at `https://huggingface.co/datasets/OptRecover/OptRecover`, and the evaluation code is available at `https://anonymous.4open.science/r/OptRecover/README.md`. The release includes problem descriptions, structured references, prompts, and scoring implementations. The repository provides instructions and scripts for rerunning the main experiments, running a minimal example, and evaluating additional models, with a fixed dataset revision specified in the README. Appendix A , Appendix B , Appendix C describe the representation, parser, and implementation details, respectively.

REFERENCES

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 577–596, 2024.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela Van Der Schaar. Autoformulation of Mathematical Optimization Models Using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 1864–1886, 2025.

Hao Chen, Gonzalo Esteban Constante-Flores, Krishna Sri Ipsit Mantri, Sai Madhukiran Kompalli, Akshdeep Singh Ahluwalia, and Can Li. OptiChat: Bridging Optimization Models and Practitioners with Large Language Models. *INFORMS Journal on Data Science*, 5(3):199–220, 2026. doi: 10.1287/ijds.2025.0074.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-Informed RL: Grounding Large Language Models for Authentic Optimization Modeling. In *Advances in Neural Information Processing Systems*, volume 38, pp. 106027–106069, 2025. doi: 10.52202/085713-3539.

Joshua Drossman, Alexandre Jacquillat, and Sébastien Martin. Let's Have a Conversation: Designing and Evaluating LLM Agents for Interactive Optimization. *arXiv preprint arXiv:2604.02666*, 2026.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A Customizable Framework in Training Large Models for Automated Optimization Modeling. *Operations Research*, 73(6):2986–3009, 2025. doi: 10.1287/opre.2024.1233.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch. In *International Conference on Learning Representations*, volume 2025, pp. 101580–101606, 2025.

Connor Lawless, Jakob Schoeffer, Lindy Le, Kael Rowan, Shilad Sen, Cristina St. Hill, Jina Suh, and Bahareh Sarrafzadeh. "I Want It That Way": Enabling Interactive Decision Support Using Large Language Models and Constraint Programming. *ACM Transactions on Interactive Intelligent Systems*, 14(3), 2024. doi: 10.1145/3685053.

Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*, 2023.

Belinda Li, Alex Tamkin, Noah Goodman, and Jacob Andreas. Eliciting Human Preferences with Language Models. In *International Conference on Learning Representations*, volume 2025, pp. 80984–81013, 2025.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-Verifier: Unleashing the Power of LLMs for Optimization Modeling via Dual-Side Verification. *arXiv preprint arXiv:2605.29556*, 2026.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A Scalable Bidirectional Data Synthesis Framework for Optimization Modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 40769–40802, 2025.

Mahdi Mostajabdaveh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating LLM Reasoning in the Operations Research Domain with ORQA. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 24902–24910, 2025.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt Competition: Formulating Optimization Problems Based on Their Natural Language Descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220, pp. 189–203, 2022.

Marzieh Saeidi, Max Bartolo, Patrick Lewis, Sameer Singh, Tim Rocktäschel, Mike Sheldon, Guillaume Bouchard, and Sebastian Riedel. Interpretation of Natural Language Rules in Conversational Machine Reading. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2087–2097, 2018. doi: 10.18653/v1/D18-1233.

Tabinda Sarwar, Farhad Moghimifar, Cong Duy Vu Hoang, Xiaoxiao Ma, Shawn Chang Xu, Fahimeh Saleh, Poorya Zaremoodi, Avirup Sil, and Katrin Kirchhoff. CLARITY: A Framework and Benchmark for Conversational Language Ambiguity and Unanswerability in Interactive NL2SQL Systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 1218–1250, July 2026. doi: 10.18653/v1/2026.acl-industry.86.

Yang Song, Anoushka Vyas, Zirui Wei, Sina Khoshfetrat Pakazad, Henrik Ohlsson, and Graham Neubig. NEMO: Execution-Aware Optimization Modeling via Autonomous Coding Agents. *arXiv preprint arXiv:2601.21372*, 2026.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *International Conference on Learning Representations*, volume 2024, pp. 48519–48537, 2024.

Guangrui Xie. ORPilot: A Production-Oriented Agentic LLM-for-OR Tool for Optimization Modeling. *arXiv preprint arXiv:2605.02728*, 2026.

Linzi Xing, Xinglu Wang, Yuxi Feng, Zhenan Fan, Jing Xiong, Zhijiang Guo, Xiaojin Fu, Rindra Ramamonjison, Mahdi Mostajabdaveh, Xiongwei Han, Zirui Zhou, and Yong Zhang. Towards Human-aligned Evaluation for Linear Programming Word Problems. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 16550–16556, 2024.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *International Conference on Learning Representations*, volume 2025, pp. 24726–24759, 2025.

Haotian Zhai, Connor Lawless, Ellen Vitercik, and Liu Leqi. EquivaMap: Leveraging LLMs for Automatic Equivalence Checking of Optimization Formulations. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 74288–74305, 2025.

Michael Zhang, W. Bradley Knox, and Eunsol Choi. Modeling Future Conversation Turns to Teach LLMs to Ask Clarifying Questions. In *International Conference on Learning Representations*, volume 2025, pp. 60722–60742, 2025a.

Michael JQ Zhang and Eunsol Choi. Clarify When Necessary: Resolving Ambiguity Through Interaction with LMs. In *Findings of the Association for Computational Linguistics: NAACL 2025*, 2025. doi: 10.18653/v1/2025.findings-naacl.306.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. CLAMBER: A Benchmark of Identifying and Clarifying Ambiguous Information Needs in Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10746–10766, 2024. doi: 10.18653/v1/2024.acl-long.578.

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

A REPRESENTATION SPECIFICATION

The shared representation expresses the business content needed to compare specifications and answer clarification questions. It contains eight linked components, summarized below. This appendix defines their meanings, the boundaries between optimization mechanisms, and the treatment of parameters, dependencies, and incomplete knowledge.

A.1 SHARED OBJECTS, DECISIONS, AND LOCAL KNOWLEDGE

Section	Meaning
`source`	Input text, original source units, and character locations for evidence.
`R`	Objects, domains, axes, task domains, native decisions, free attributes, paths, and derived selection contracts.
`BS`	Per-Block presence status: present, absent, or unresolved.
`B`	Indexed business-rule Families organized by Block.
`definitions`	Shared state identities and links to published selection-count requirements.
`P`	Fixed numerical, set-valued, or interval-valued data with source indexing and units.
`objective`	Overall direction and aggregation; individual contributions remain in objective Families.
`unresolved`	Paths to local omissions, ambiguities, or unresolved extraction, with explanations.

Table A1: Top-level sections of a business-semantic description. Experiment metadata, model identities, and voting records are external to this representation.

Objects define populations such as jobs, technicians, or days. A base domain contains explicit members or a source-supported finite cardinality and index range. A composite domain refers to its factors instead of copying the same entities. Ordered domains retain their source order. An index such as "day 1" is not an invented calendar date, and a numbered domain is not necessarily ordered.

A decision records its carrier, target, and shared context axes. A domain carrier means that a choice is made for a domain element. A selected-relation carrier means that a subordinate choice exists on a selected parent relation. Attributes attach freely chosen values to their native carriers. Paths express already established connections; they are not additional independent decisions. Thus a quantity defined on selected deliveries is not interchangeable with an unconstrained quantity on every possible delivery.

Native selection contracts describe a set of selected target tuples for each context. They do not impose a required count. Count bounds reside in `B`; derived status fields in `definitions` point to these rules without copying their values. The distinction is important for incomplete text: knowing that a job has a technician choice does not establish whether it requires exactly one, at least one, or several technicians. Integer type likewise does not imply nonnegativity; a numerical domain restriction must be explicitly represented.

A.2 RULE FAMILIES AND THREE KINDS OF SCOPE

A Family is an indexed policy, not a single matrix cell, sentence, or emitted inequality. A table of per-technician capacities is one indexed capacity policy. Its member-dependent values are parameters. A default workload limit and a conditional higher limit for senior reviewers can belong to the same Family when they jointly express one policy. Lower and upper bounds on the same computation are paired when they belong to that policy; independently stated policies remain distinct. Different charges remain separate even when they share an event or rate.

Rule fields distinguish *application scope*, *contribution scope*, and *data scope*. Application scope specifies which outer instances are governed. Contribution scope determines which selections or quantities enter the computation. Data scope identifies where fixed source values are defined. For an aggregate grouped by technician, filtering the summed jobs changes the workload computation, not the set of technicians to which the resulting bound applies.

Activation is also separate from fixed applicability. A fixed subset can restrict the policy to named technicians, while a used-member event activates it according to the actual selections. Event identity

includes the relevant member axes and any reset axes. A whole-horizon used-technician event is different from a daily event even when both refer to the same technician.

A bound may have a default and explicit cases. Cases replace the default on their specified, mutually exclusive regions; array order does not establish priority. A Family's activation and the applicable case condition both have to hold. A null default means that no default bound is present; it is not an unknown numerical value. A case with a null bound removes that endpoint on its matching region without removing the underlying mechanism. Closed and open endpoints are represented explicitly. An unresolved lower side can coexist with a known upper side.

A.3 The Fourteen Mechanisms

Table A2 gives the full mechanism inventory. These descriptions concern supported computations and distinctions; they are not a catalogue of application-specific rules.

Table A2: The 14 mechanism templates, their computations, and the semantic distinctions they preserve.

Block	Core content	Distinction preserved
`AGGREGATE_BOUND`	Bounds on sums of selected memberships, fixed contributions, or free quantities, with grouping, filters, and activation.	`NATIVE_TARGET_COUNT` counts native selected targets; `BUSINESS_SUM` expresses other measures. Counting selections is not distinct-object coverage.
`DISTINCT_OBJECT_COUNT`	Count objects for which a specified selection event occurs at least once.	Multiple qualifying assignments to the same counted object contribute once, with the stated reset context.
`SELECTION_FEASIBILITY`	Legality of one selection relation, expressed by an explicit allowed set or a fixed property–requirement comparison.	A permitted choice need not be required, and pairwise eligibility does not prove global feasibility.
`JOINT_SELECTION_COMPATIBILITY`	Compatibility of jointly selected choices from two distinct existing decisions.	Preserve the participating decisions and context; do not collapse a joint condition into independent allowed lists.
`NUMERIC_BOUND`	Bounds on a free numeric attribute or a shared state at specified instances.	Type, sign, scope, and terminal requirements are explicit; a recurrence alone does not impose them.
`PAIRWISE_LOAD_DIFFERENCE`	Bounds on the absolute load difference between two distinct resources used in the same partition.	Preserve the load construction, resource-use event, and partition; use does not imply positive load.
`ACTIVITY_TIME_BOUND`	Start and finish limits for an identified activity, including policy cases.	A clock window differs from elapsed duration; global and activity-specific policies remain separate.
`RESOURCE_SEPARATION`	Non-overlap of activities sharing a resource, with any extra separation gap.	Activity identity prevents self-conflicts from subordinate assignments. The ordering is not assumed source-fixed.
`PRECEDENCE_GAP`	Source-defined directed predecessor–successor relations and elapsed gaps.	Preserve endpoint roles, completion basis, and aligned contexts; do not invent an optimizer-chosen order.
`START_AVAILABILITY`	External interval calendars for start only, finish only, or whole activity.	Whole-activity containment requires one continuous interval component. Endpoint closure matters, and start permission does not imply containment.
`START_BUCKET_COUNT`	Counts of actual activity starts satisfying a time-window predicate.	Separate the counted activity, grouping context, count bounds, and start-time predicate.

Block	Core content	Distinction preserved
ADJACENT_CHANGE_COUNT	Counts of selected-set changes for the same subject between consecutive periods.	One change event is not the number of additions or removals; no pre-horizon choice is invented.
STATE_BALANCE	A shared end-of-period state with initial value, inflow, and decision-dependent outflow.	Initial stock is used before the first period, not added every period. Timing and all consumption components remain explicit.
OBJECTIVE_TERM	A signed cost or benefit contribution, its event or quantity, and coefficient.	Charging event, repetition, reset, and rate indexing are separate. Independent fees are not merged by equal values.

A.4 Parameter Identity, Coordinates, and Units

A Family binds parameter roles such as LOWER, UPPER, EQUAL, COEFFICIENT, or INITIAL_STATE to entries in P. Three axis lists serve different purposes:

$$\underbrace{\texttt{application_axes}}_{\text{where used}},\quad \underbrace{\texttt{data_axes}}_{\text{source lookup}},\quad \underbrace{\texttt{value_member_axes}}_{\text{identity inside a set}}. \tag{A.1}$$

For example, a workload coefficient can be applied to job–technician pairs while being looked up only by job. A global qualification set has no data lookup axes, yet each member may be a multi-axis tuple. Such a parameter has one scalar-addressed cell at the empty tuple, not an empty data domain.

A source-defined full product and an explicitly listed key domain are different declarations. Excluding a table row from one computation does not remove that row from the source fact. Conversely, a genuinely conditional source table does not supply values outside its stated domain. Constant observed values do not justify dropping a source-defined dimension; repeated use across contexts does not justify inventing one.

Set values distinguish complete and partial membership. A complete set declares unlisted legal members false. A partial set confirms its listed members but leaves the rest unknown. An explicitly empty complete set therefore differs from an unknown or partial set. Tuple positions are meaningful: predecessor and successor positions can use the same axis while retaining distinct role labels. Row order in an unordered set is immaterial, but tuple columns cannot be independently rearranged.

Numeric units are attached to the source fact. Supported local clock times are represented in minutes from local midnight, whereas elapsed durations have no clock origin. Source dates, offsets, or units are not supplied from the Golden when absent in a candidate. A null unit records no stated unit where allowed by the schema; an unknown unit records unresolved metadata. Parameter values may remain known even when another metadata field is unresolved.

A.5 Dependencies, States, and Objective Events

A state definition identifies its owner, period axis, and end-of-period interpretation. A balance Family connects the previous state, exogenous inflow, and all decision-dependent outflow terms, with the source-defined inflow timing. Initial stock is a separate fixed fact. Nonnegativity, storage capacity, and final reserves are numerical policies rather than implicit properties of the recurrence.

A supported state-dependent bound can reference an existing state through STATE_AVAILABLE at BEFORE_OUTFLOW. This is a dynamic endpoint rather than a fixed parameter in P; its meaning follows the source-supported inflow timing.

An objective Family identifies both what is charged and how often it occurs. A selected-relation event can charge each assignment; a used-member event projects out assignment axes and charges once for each used resource. Reset axes determine whether that event repeats across periods. The overall objective direction is stored separately from term signs. Copying a positive rate into a subtractive term should not negate it again.

Problem 4374 illustrates shared dependencies. Its minimum workload of 5 and preparation charge both depend on technician use, yet have different computational roles. The capacity upper bound

applies to every technician; only the lower endpoint is conditional. The preparation fee projects out the job axis and has no daily reset. These details are preserved even though a short informal description might call all three requirements "dispatch rules."

A.6 Partial Specifications and Reference Roles

The three unknown objects are exactly `{"unknown":"NOT_STATED"}`, `{"unknown":"AMBIGUOUS"}`, and `{"unknown":"UNRESOLVED"}`. They appear only at schema-supported locations; arbitrary identity keys or coordinate positions cannot be removed while claiming that the remaining structure is well defined. An unresolved record points to the affected location and does not override contradictory formal content with an explanatory sentence.

The complete Golden is the fixed business reference. The incomplete Golden is a faithful annotation of the initial public text. The candidate is the parser's output from the recovered text. All three use the same semantics, but their permissible knowledge gaps differ. The absolute scorer interprets K_0^* in the candidate position against K^*; it does not require K_0^* to be complete. Only K^* determines scored obligations and budgets. A complete Golden with unsupported or unresolved scored content is not silently reduced to its interpretable subset.

B Parser Pipeline and Algorithms

The parser constructs a fresh representation from each input description. The same pipeline can process a complete problem or a recovered description; the source supplied to that run is its only factual authority. In particular, references to "original source" inside a parser prompt mean the input of that parser run, not the hidden complete problem in a recovery experiment.

B.1 Stage Dependencies

Figure B1 and Table B1 show the active sequence. Outputs from another repetition or model are not substituted for a failed prerequisite. Model judgments supply business interpretation, while deterministic code validates references, freezes accepted structures, constructs slots, and records partial results.

Stage	Inputs beyond the source	Output / responsibility
R1	Object-registry contract	Basic object populations and roles.
R2	Accepted objects, domain contracts	Base extents, required task tuples, and source-defined composite domains.
R3	Accepted objects and domains	Decisions, carriers, targets, attributes, and connections.
BS	Accepted registry, grouped Block cards	Per-Block presence labels and evidence.
RF A/B	Registry and present Block profiles	Independent source-order and decision-oriented rule proposals.
RF C/D	Both proposals; disputed cases if any	Source-checked Families and explicit remaining uncertainty.
BI	Confirmed rule and binding candidates	Typed business structure and source roles.
Compiler	Bound structures and role contracts	Canonical structures, references, and parameter slots.
P_BIND	Compiled Family and its slots	Source-fact identity, data axes, domain, unit, and evidence.
P_FILL	Bound slots and requested coordinates	Fixed values, sets, or intervals, with local unknowns.
Output	Accepted structures and objective reading	Final schema-checked business representation.

Table B1: Inputs and responsibilities of active parser stages. All factual readings remain grounded in the current source text.

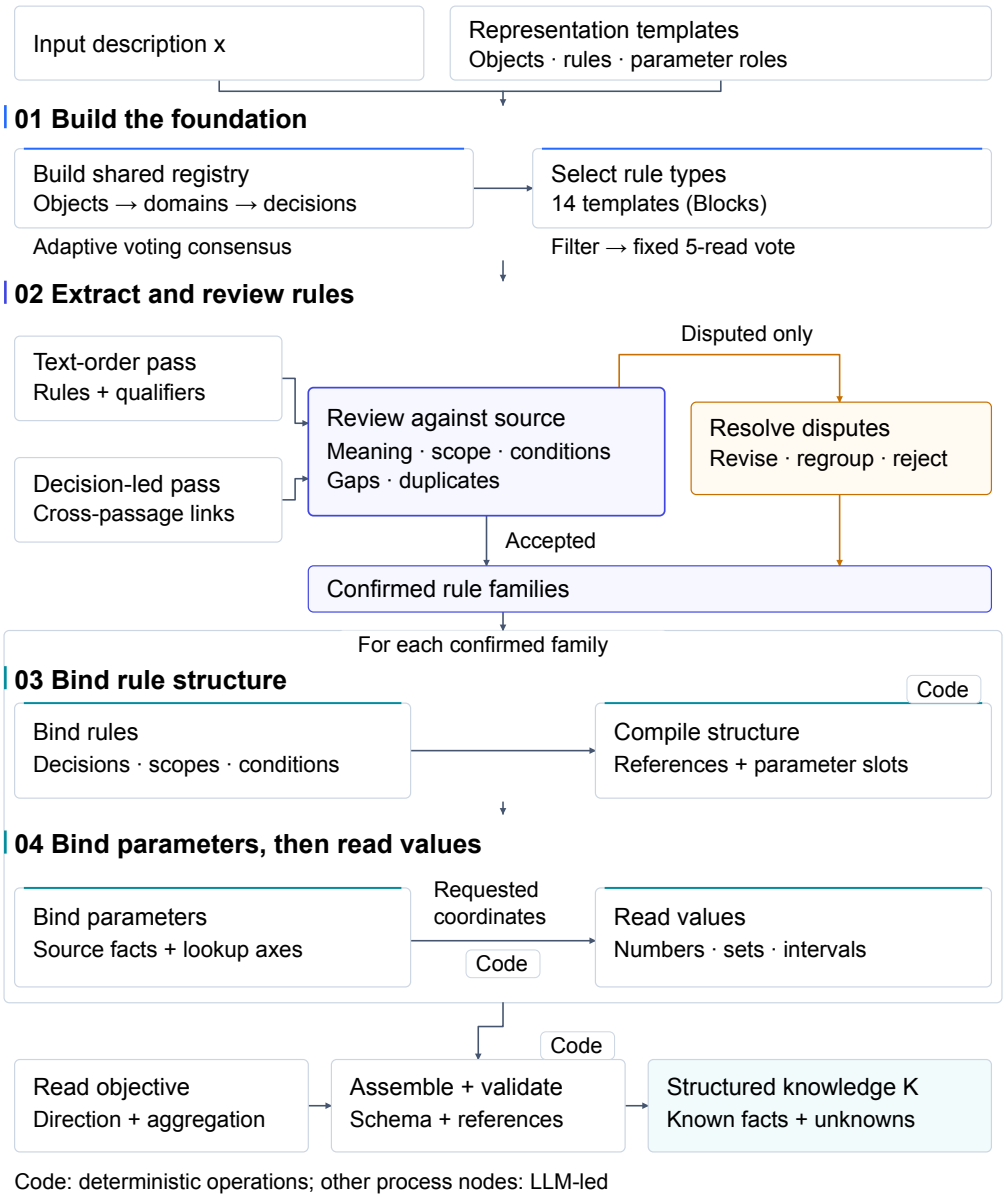

Figure B1: Detailed parser pipeline

B.2 Adaptive Registry Consensus

Each registry voting unit starts with five independent observations. A voting unit can concern R1, one base-domain realization, a required task-domain determination, composite-domain discovery, or R3. All five observations are awaited before tallying semantic groups; an empty set of valid groups has maximum support zero. A group supported by at least three observations is selected, subject to clarity, domain, and downstream representation checks.

Valid observations are grouped by deterministic, stage-specific canonical signatures. Consensus is over the resulting structures rather than independent fieldwise majorities or judgments from an additional LLM evaluator.

Only when no group reaches three votes are observations six and seven requested. They receive the same initial prompt, source, and frozen context, without earlier responses. After all seven observa-

tions arrive, a group with at least four votes is preferred. Otherwise a three-vote group is accepted; if two such groups exist, a saved unit-specific seed determines a uniform random choice. A maximum below three terminates that prerequisite without acceptance. A selected but ambiguous or unsupported interpretation does not trigger further sampling merely to obtain a more convenient answer.

Algorithm 1 Adaptive consensus for one registry unit

Require: Source x, fixed context z, stage contract p, saved seed s
$O \leftarrow$ request five independent observations with (x, z, p)
Await all observations, retaining completed invalid responses
if technical failure leaves an observation incomplete **then**
    return explicit incomplete-observation status
end if
$\mathcal{G} \leftarrow$ semantic groups among valid observations in O
$m \leftarrow \max(\{0\} \cup \{|g| : g \in \mathcal{G}\})$
if $m < 3$ **then**
    Request two more independent observations with the same (x, z, p)
    Await both; return an incomplete-observation status if necessary
    Update $\mathcal{G}$ using all seven observations
end if
if a group has at least four votes **then**
    $g^\star \leftarrow$ that group
else if at least one group has three votes **then**
    $g^\star \leftarrow$ the unique group, or seeded uniform choice among two ties
else
    return unresolved consensus
end if
Apply stage clarity, domain-support, and output-capability checks to $g^\star$
return the accepted result or the explicit failed prerequisite

A repairable protocol or internal-consistency defect can receive one same-lane correction. That request includes the original task, source, fixed context, initial response, and concrete validation error. It is not a new independent vote. Legal semantic disagreements are not repaired into agreement. A completed invalid response is an observation without a valid vote; a transport failure is not a semantic observation. The denominator is not reduced when a provider fails to return.

B.3 Grouped Mechanism Selection

An initial deterministic filter rules out mechanisms that are structurally impossible under the accepted registry. Remaining Blocks are queried in four groups: selection; quantities; time; and evolution plus objective. Each nonempty group receives five independent model requests. A response gives a separate status for every requested Block; the groups are not mutually exclusive classes.

After all observations are available, a Block requires at least three equal status labels to resolve. Otherwise it remains unresolved. An invalid entry affects that Block's vote, while an unreadable whole response affects every requested Block in that observation. Registry sampling may expand to seven; Block selection remains fixed at five. Presence selection determines which mechanisms enter rule extraction, not how many Families each contains.

B.4 Two-View Rule Extraction and Source Review

Proposal A follows source order, while proposal B follows the accepted decision structure. This gives two opportunities to identify requirements whose conditions may occur away from their main statements. A subsequent correspondence stage C reconstructs source requirements before approving matches. It checks calculations, scopes, activation, endpoints, activity identities, selection-count obligations, and parameter meanings. Agreement between proposals is not itself proof of correctness.

Stage C groups fragments globally before approving pairs. A directly accepted pair requires a complete, source-correct proposal from each view and identifies the payload retained. A group needing replacement or regrouping is sent to stage D with a specific neutral question. Independently missing policies can be added when supported by the source; an existing but defective policy is reviewed rather than counted again as an addition. Stage D can select, replace, split, merge, or reject the affected proposals according to the applicable contract, while preserving material uncertainty.

This stage is not a five-way majority vote and does not use a separate evidence-mapping ensemble. It freezes business policy identity for binding. For example, an activity's source-defined fixed-duration completion may already be captured by the activity definition; it should not become a redundant clock policy solely because both proposers omit a separate completion inequality.

B.5 Binding, Compilation, and Data Filling

BI binds each accepted Family to the typed structure needed by its Block. The model identifies the formal decisions, measures, scope, activation, and source roles. The compiler then creates the corresponding references and parameter slots deterministically. It does not infer a replacement policy from numerical values. Shared state identities are established before dependent bindings.

Application axes and tuple-value member axes are fixed by the compiled slot contract. P_BIND selects the source lookup axes from the allowed coordinates and identifies the exact fixed source fact, source-defined domain, unit, and evidence. P_FILL receives explicit coordinates and reads their values. Known values on completed cells are retained if another slot or cell is unresolved. Equal scalar values in an independent fee and a count bound do not collapse their identities. A parameter whose source domain is indexed remains indexed even when its current entries are equal.

Algorithm 2 Description-to-knowledge parsing

Require: Description x; frozen generic contracts; one parser configuration
Build source units from x without changing its factual text
$R \leftarrow$ object, domain, and decision construction using Algorithm 1
if an essential registry prerequisite is not accepted **then**
    return recorded partial representation and prerequisite status
end if
$\mathcal{B} \leftarrow$ grouped five-observation Block selection
In parallel, propose F_A by source order and F_B by decision structure
$(F_{\text{accept}}, \mathcal{U}) \leftarrow$ source-grounded correspondence and coverage review
Resolve disputed cases $\mathcal{U}$ using the source and rule contract
for each confirmed Family f **do**
    Bind f to the shared registry and typed computation
    Compile structure, connections, and fixed-data slots
    Bind each slot's source identity, data coordinates, domain, and unit
    Read fixed data at requested coordinates; preserve local unknowns
end for
Read overall objective direction and assemble K
Validate references and schema; record any remaining local uncertainty
return K and external run-status metadata

B.6 Operational Boundaries

Different instances can advance asynchronously, but a stage waits for its own prerequisites. In particular, source review waits for both rule proposals, and voting waits for all planned observations. Retries retain the logical request identity rather than increasing its voting weight. The parser retains completed outputs, including partial outputs, and does not access a Golden to choose among its internal observations. Separately, the recovery evaluation in Section 5.1 selects the highest-overall-agreement output from three independent parses of the same recovered description and takes every diagnostic from that output. The independent complete-problem parser experiment uses one parse per backbone and description.

Unsupported native ordering relations and projected relation carriers are reported as capability limits rather than rewritten into other decisions. Such limits are distinct from semantic errors in a completed parse. Likewise, finite parameter materialization limits create unresolved output where appropriate rather than invented empty tables. These conditions are recorded separately from semantic scores; their incidence cannot be inferred from mean agreement scores alone.

C Interaction and Evaluation Protocols

This appendix specifies the information available to each component, the interaction prompts, the deterministic comparison policy, and the recovery aggregation used in Section 5.1.

C.1 Experimental Configuration and Information Access

The questioner and integrator use the same evaluated model: GPT-5.6 Sol, Claude Opus 5, DeepSeek V4.1 Flash, Qwen 3.8 Max, GLM-5.3, or Kimi K3. Each model conducts one dialogue per incomplete instance. DeepSeek V4.1 Flash is the fixed backbone for both the answerer and the recovery parser. Only the separate complete-problem parser experiment varies the parser backbone, using Kimi K3, DeepSeek V4.1 Flash, and Qwen 3.8 Max.

Field	Recovery experiment	Complete-problem parser experiment
Inputs	288 incomplete variants	48 complete descriptions.
Dialogue	One per model and instance; at most 15 questions; early stop allowed	No dialogue.
Integration	Same model as questioner; first valid final text	Not applicable.
Answerer	Fixed DeepSeek V4.1 Flash; at most one targeted group expansion	Not applicable.
Parser	Fixed DeepSeek V4.1 Flash; three independent parses of each recovered text	Three backbones; one parse per description and backbone.
Parse selection	Highest overall agreement; all diagnostics from that same parse	The single parse for each description.
References	Complete Golden K^* and initial Golden K_0^*	Complete Golden K^*.
Reported measure	SRR and diagnostic recovery, with averaging before normalization	Absolute agreement and diagnostics, averaged over descriptions.

Table C1: Experimental protocol reported in Section 5.1. Repeated parsing does not create additional dialogues or recovered descriptions.

Omitting temperature or reasoning controls uses the provider's behavior; it does not establish equal effective inference budgets across providers. Transport and protocol retries concern completion of a logical call; they are separate from independent parser observations and from the three complete parses used in the recovery experiment. Parser observation and consensus handling are specified in Appendix B.

For each recovered description $\hat{x}_i$, the parser produces three candidates $\widehat{K}_i^{(1)}, \widehat{K}_i^{(2)}, \widehat{K}_i^{(3)}$. Among scorable candidates with all six scores available, evaluation selects the highest unrounded $S(\widehat{K}_i^{(r)}, K_i^*)$ and uses that candidate for every reported diagnostic. Scorable partial outputs remain eligible when all six scores are defined. Exact ties follow a fixed ordering of candidate records, with the lower repetition index taking precedence; no additional diagnostic is used to break a tie. If no candidate is eligible, the instance is recorded as lacking a complete score rather than silently assigned zero. The reference is consulted only after the parses have been produced; it is not provided to the parser and is not used to revise $\hat{x}_i$. This reference-based best-of-three selection is part of the measurement protocol, so recovery results should not be interpreted as performance under a single unselected parse. The complete-problem parser experiment uses one parse per backbone and description.

Component	Factual information supplied	Output boundary
Questioner	x_0, delivered question–answer history, remaining budget	ASK with one question, or STOP.
Integrator	x_0 and delivered dialogue	One recovered English description.
Router	Complete source, public context, generic group menu	Group IDs and exact public-text spans only.
Selector	Cards in selected groups, parameter menus, public context	Displayed card IDs, requested roles and filters, or expansion.
Executor	Frozen structured reference and selected requests	Selected rules, definitions, parameter slices, safe excerpts.
Expression	Evidence packet and public context	Natural answer and private support IDs.
Parser	Current description, generic contracts, current-run outputs	Structured candidate, including local unknowns.
Scorer	Candidate and complete Golden; initial Golden for recovery layer	Semantic ledger, scores, support/status diagnostics.

Table C2: Information boundaries throughout the evaluation. Public history consists of delivered question and answer strings, not the model's private reasoning or hidden reference artifacts.

C.2 Prompt Content and Information Boundaries

All prompts are English. The initial and follow-up questioner prompts are rendered into one complete user message. The renderer fills the initial text and remaining-question count, and adds a JSON list of delivered question–answer pairs only for follow-up turns. It substitutes placeholders in the template, not text that happens to resemble a placeholder inside the problem.

The integration payload contains the incomplete problem and a numbered list of delivered question–answer pairs. The router, selector, and expression stages receive their stage-specific instruction and allowed payload. The selector cannot see the complete source. The executor validates displayed identifiers and filters; expression support identifiers must belong to the supplied packet. These checks constrain data access but do not constitute an independent semantic audit of the answer's natural language.

The initial-question and final-integration prompts are reproduced in full at the end of this appendix. Follow-up questions additionally receive the delivered dialogue and remaining question budget. Routing, selection, and answer expression use the stage-specific information boundaries in the table above. Parser instructions combine the current description with the accepted object and decision structure, the relevant mechanism definitions, and the rule proposals, binding candidates, or parameter coordinates required by each stage. Appendix B specifies those dependencies. Integration ends with the first valid final text; there is no subsequent reference-based revision of that text.

C.3 Deterministic Semantic Comparison

The same fixed comparison policy interprets each Golden and candidate. Reference groups are derived from their formal structures rather than manually specified for individual problems. Neither the problem identifier nor the model label affects the matching or weighting rule.

Identity and finite semantic interpretation. Objects and members are matched using formal names and explicitly supplied aliases. Decisions, attributes, axes, and states then share one consistent identity correspondence across all Families. Local IDs and unordered row order have no independent score. The engine does not use descriptive prose or evidence snippets to fill missing candidate content. Nor does it exchange object identities to make numerical values match.

Comparison supports a fixed set of structural transformations, including consistent ID and axis renaming, equivalent indexed expansion, scalar versus repeated constant values where semantically valid, paired closed bounds equivalent to equality, supported placements of used-member activation, and role-preserving directed-pair representations. These transformations must preserve the representation semantics defined in Appendix A. Arbitrary mathematical equivalence, identical optimal values, and solver-generated feasibility tests are not used as substitutes for semantic correspondence.

Reference-derived groups and budgets. Objects, choices, attributes, and objective direction form groups by formal identity. Rule groups use operator class, participating references, axes, and structural roles while abstracting away numerical values and concrete member coordinates. Cost groups follow the event's relation basis and fixed-context structure. This is uniform structural grouping: it need not reproduce every distinction an annotator might make between two source sentences.

If there are G groups, group g has budget $w_g = 100/G$. Within a group, applicable Meaning, Boundary, and Relation dimensions share the budget equally. Within a dimension, distinct semantic functions share its budget, and a function's grounded instances share its allocation. Thus a long parameter table does not receive a full policy budget for each cell. Parameters are scored within their actual use context; unreferenced data entries do not earn independent credit. Identical duplicated constraints do not create additional reference obligations, while duplicate cost contributions preserve their effect on the objective.

Correspondence. Exact canonical fingerprints are aligned first, followed by unique same-category structural anchors. Remaining candidates are matched one-to-one with explicit unmatched options. Exact anchors receive similarity 0.96, with an additional 0.04 for matching relation direction. Otherwise an edge must share an already aligned formal reference, and the global policy uses

$$q = 0.50q_{\text{references}} + 0.25q_{\text{members}} + 0.25q_{\text{roles}}, \qquad q \geq 0.60. \tag{C.1}$$

Reference and member terms are set-overlap measures; the role term is the registered category/anchor-shape compatibility. Numerical thresholds, fee values, descriptive text, and the eventual score are not similarity features. Tied alternatives with different scored content are marked as alignment ambiguity rather than resolved by taking the higher score. One candidate atom cannot satisfy several independent reference atoms.

Per-slot ledger. Let a reference slot carry weight w. Its contribution to correct weight C, missing-or-incorrect weight M, erroneous assertion weight E, and asserted weight A is shown in Table C3. Partial structures are evaluated through their fixed sub-budgets.

Candidate content	C	M	E	A
Known and correct	w	0	0	w
Known and incorrect	0	w	w	w
Unknown, absent, or unbound	0	w	0	0

Table C3: The reference-slot ledger. Independently unmatched known assertions contribute additional E under the uniform extra-group rule.

Numbers are equal only within $10^{-9} + 10^{-9}\max(|v^*|, |v|)$ after registered unit normalization. A value in the wrong parameter role or coordinate does not fill the intended slot. Boolean values are not numeric surrogates. For a finite set slot, the reference positive set and its complement each receive half the budget when both are nonempty; if only one class exists, it receives the full budget. Unknown candidate membership loses reference credit without becoming a false assertion. This prevents a large negative class from overwhelming a small required positive set.

Unmatched known assertions are grouped by typed category, formal references, relation direction, and sign, not Family ID. A genuinely new group receives a one-group-budget penalty. Extra instances of a related group receive a proportional penalty capped at that budget. Content already charged as a wrong matched slot or an extra boundary instance is not penalized again as an independent policy. Unreadable or explicitly unknown placeholders are recorded as limitations rather than fabricated known assertions.

Absolute diagnostics. With $C + M = 100$, Equation 3 is the primary score. Let W_d and C_d be the reference budget and correctly expressed weight of dimension $d \in \{\text{meaning}, \text{boundary}, \text{relation}\}$. The five diagnostics are

$$s_{\text{coverage}} = A, \qquad s_d = 100C_d/W_d, \quad W_d > 0, \tag{C.2}$$

$$s_{\text{precision}} = 100C/(C + E), \qquad C + E > 0. \tag{C.3}$$

A truly inapplicable fidelity dimension is undefined. The implementation assigns Matching Precision zero when $C + E = 0$, so silence does not receive a perfect precision score. The three fidelity dimensions are reported as Meaning Fidelity, Boundary Fidelity, and Relational Fidelity. Coverage can be high when incorrect assertions are numerous; it is not a substitute for the primary score. Wrong matched claims and independently unsupported claims both enter E and therefore both affect Matching Precision.

C.4 Relative Recovery, Zero Gaps, and Aggregation

Semantic discrepancy is the absolute-score shortfall, $D(K, K^*) = 100 - S(K, K^*)$, rather than the raw $M + E$ ledger total. Both the initial Golden and the selected recovered parse are compared with the same complete Golden. For a common instance set $\mathcal{I}$, define the mean absolute scores

$$\overline{S}_0 = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} S(K^*_{0,i}, K^*_i), \qquad \overline{S}_T = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} S(\widehat{K}_i, K^*_i). \tag{C.4}$$

The reported recovery is

$$\mathrm{SRR}(\mathcal{I}) = 100 \frac{\overline{S}_T - \overline{S}_0}{100 - \overline{S}_0} = 100 \frac{\sum_{i \in \mathcal{I}} (D_{0,i} - D_{T,i})}{\sum_{i \in \mathcal{I}} D_{0,i}}, \qquad \overline{S}_0 < 100. \tag{C.5}$$

Each instance contributes equally to the mean absolute scores. Normalization occurs after averaging, as in Equation 5; this is not an arithmetic mean of per-instance recovery ratios. The same calculation is applied independently to the overall set, each complexity level, and each missingness level. An overall recovery rate is therefore not generally the unweighted mean of the subgroup recovery rates.

For each diagnostic d, let $\overline{s}_{0d}$ and $\overline{s}_{Td}$ be its initial and final mean absolute values on the same comparison set. The reported diagnostic recovery is

$$R_d(\mathcal{I}) = 100 \frac{\overline{s}_{Td} - \overline{s}_{0d}}{100 - \overline{s}_{0d}}, \qquad \overline{s}_{0d} < 100. \tag{C.6}$$

This definition applies to Information Coverage, Meaning Fidelity, Boundary Fidelity, Relational Fidelity, and Matching Precision. All diagnostics come from the parse selected by overall agreement; there is no separate best-parse selection for individual dimensions. The diagnostics are not averaged to obtain SRR.

A zero initial gap for one instance does not exclude that instance from aggregate recovery. It contributes zero to the initial-gap sum, and any subsequent degradation contributes negatively to the numerator. Only a zero aggregate initial gap makes the corresponding recovery rate undefined; that group is reported as N/A. Absolute final scores can still describe preservation in such a group. A fidelity dimension that has no applicable reference budget is undefined for a different reason and must be distinguished from a zero gap. Any exclusions or changes to a diagnostic's comparison set must be recorded consistently across models.

Negative recovery values are retained without clipping. The denominator is the measured initial semantic discrepancy, not the nominal missingness level or the recorded deletion ratio. Parser reliability on the 48 complete descriptions instead reports equally averaged absolute scores from one parse per backbone and description; the three parses in the recovery experiment are neither three dialogues nor three entries in the recovery average. Run records should retain the planned, completed, and scorable counts and the selected parse identity. If uncertainty intervals are added, the resampling unit and treatment of related variants from the same complete problem must be stated.

Arithmetic illustrations, not experimental results. For a single instance, suppose $C_0 = 60$, $M_0 = 40$, and $E_0 = 0$, giving $S_0 = 75$. A recovered candidate with $C_T = 80$, $M_T = 20$, and $E_T = 12$ has $S_T = 83.33$, so its shortfall reduction is 33.33%. Using raw $M + E$ instead would give 20%.

To illustrate aggregation, consider two instances with initial gaps $(10, 50)$ and final gaps $(5, 40)$. Their reported aggregate recovery is $100(60 - 45)/60 = 25\%$. Averaging their individual recovery rates, 50% and 20%, would give 35% and is not the protocol used here.

C.5 SCORABILITY, LIMITS, AND REPRODUCIBLE RECORDS

Evaluation distinguishes scorable outputs, scorable outputs with limitations, invalid inputs, unavailable references, empty-business outputs, and comparison failures. A candidate with both `B` and `P` empty is excluded, not automatically assigned zero. Partial unknowns in an otherwise scorable output remain eligible and lose the corresponding reference credit. The reported scores characterize semantic agreement conditional on obtaining a scorable output.

Reference interpretation uses uniform limits of 10,000 domain members, 100,000 grounded instances, and 1,000,000 candidate alignment pairs. Exceeding a limit or failing to determine a required role is reported explicitly; taking an arbitrary prefix is not a complete comparison. Each score contribution is associated with the corresponding representation element and reference-derived weight. The policy contains no per-problem or per-model overrides. Source-text descriptions, evidence wording, and numeric similarity are not alternative ways to repair a candidate during scoring.

As described in Section 4, the benchmark uses expert-reviewed Goldens for 48 complete and 288 incomplete descriptions. An initial Golden records only the content supported by its own incomplete text; the complete Golden must not be used to restore omitted requirements into that baseline. Both the initial and recovered representations are compared against the same complete Golden, with identical semantic interpretation and reference weights.

C.6 EXACT INITIAL-QUESTION PROMPT

The initial-question prompt is given below. Its placeholders receive the incomplete description and the remaining question budget.

```
You are communicating with a business expert to gain a clear and
↪  complete understanding of a business optimization problem. You have
↪  been given a description of the problem. IMPORTANT: Some business
↪  information has been removed from this description. The description
↪  is therefore incomplete, even if the remaining text appears clear
↪  and coherent. You may choose to ask questions based on that
↪  description to fill in any business information you think has not
↪  been fully provided and clarify anything that is unclear.

The current business problem description is provided below:

<problem>
{{lossy_problem}}
</problem>

You may ask up to {{questions_remaining}} more questions, including the
↪  one question you may ask this round.

Based on the description above, make one of the following two choices
↪  this round:

1) ASK: Ask one question

If you choose ASK, you may ask one question this round that you
↪  consider most valuable for filling gaps in the business information
↪  and clarifying its meaning.

Use everyday English. Your question must be a single, independent
↪  question. Do not combine multiple questions or embed subquestions
↪  within it. Do not directly ask for the original problem statement.

2) STOP: Ask no question and end immediately

If you consider the current description sufficiently complete and clear
↪  and do not need to ask the business expert a question, you may
↪  choose STOP.
```

```
Finally, depending on whether you choose to ask a question or stop
↪  asking, you must reply in exactly one of the following two forms,
↪  returning only one JSON object:

1) If you choose to ask a question:

{"decision":"ASK","question":"Enter the one question you are asking
↪  this round.","reason":"Briefly explain why this question is
↪  valuable."}

Requirements:
- decision must be exactly the string "ASK".
- Fill in question with the one English question you are asking this
↪  round.
- Fill in reason with a brief explanation of why you chose this
↪  question.

2) If you choose to stop asking:

{"decision":"STOP","question":null,"reason":"Briefly explain why no
↪  question is needed."}

Requirements:
- decision must be exactly the string "STOP".
- question must be null. Do not enter a question or use the string
↪  "null".
- Fill in reason with a brief explanation of why you are stopping.

Do not add any other fields. Return only one of the two JSON objects
↪  specified above, with no text outside that JSON object.
```

C.7 Exact Final-Integration Prompt

The final-integration prompt is given below. Its factual input is restricted to the incomplete text and delivered answers, with questions used only to interpret those answers.

```
You are preparing a clean business problem statement after a
↪  clarification interview.

We started with the incomplete planning problem B below. A business
↪  colleague then answered the questions in dialogue. Your task is to
↪  integrate the information ACTUALLY PROVIDED IN THOSE ANSWERS into
↪  B, so a reader can understand the specified planning problem
↪  without reading the interview. You do not have the original
↪  complete problem. You are not asked to solve the problem, invent
↪  missing requirements, generate a mathematical model, or make it
↪  resemble a reference answer.

YOUR ONLY SOURCES
1. The supplied lossy_problem B.
2. The delivered answer text in each numbered dialogue turn.
The questions help identify what an answer refers to, but assumptions
↪  and suggested alternatives inside a question are NOT facts. Only
↪  use them if the answer explicitly confirms them. Do not use outside
↪  business practice, remembered problems, plausible capacities,
↪  inferred fees, or an imagined list of missing information. Do not
↪  treat "we could not verify this" as "there is no such requirement".

WRITE ONE COHERENT PROBLEM
```

```
Preserve B's setting, entities, task structure, objective, fixed data,
↪  units, and all remaining business requirements. Put a clarified
↪  rule where it naturally belongs. Combine duplicate statements and
↪  rewrite overlapping sentences so the text reads as one problem, not
↪  a Q/A transcript, a list of answers, or an appended correction
↪  memo. Use ordinary business language and consistent names. Do not
↪  mention the interview, canonical JSON, evidence IDs, or the
↪  recovery method in the restored problem.

Copy B's numeric and eligibility tables exactly, including every
↪  original row and column, unless a delivered answer explicitly
↪  corrects that table. Do not reorder, summarize away, extrapolate,
↪  or "fix" values merely because a model might otherwise be
↪  infeasible. New tables are allowed only for values explicitly given
↪  in the public answers. Do not invent a full grid from a few
↪  entries.

Preserve important distinctions: each task vs the group together;
↪  assignments vs different covered objects; one time vs daily; only
↪  when used vs every available resource; at least vs exactly vs at
↪  most; whole quantity vs a shared portion; an eligible choice vs a
↪  verified globally feasible plan. Do not convert a local absence of
↪  restriction into unlimited freedom.

HANDLING INCOMPLETE OR CONFLICTING INFORMATION
If an answer says the original does not state a detail, do not fill it
↪  with a default. When relevant, say the detail is unspecified,
↪  rather than asserting a permissive rule. If an answerer says it
↪  could not verify a claim, do not add that claim or its opposite.
↪  Preserve useful verified parts of other answers.
If B and an answer appear to conflict, apply an explicit correction
↪  only if the answer clearly corrects that exact B statement.
↪  Otherwise keep the uncontested B material, do not silently choose
↪  an interpretation, and state the conflict in the problem text. If
↪  two answers conflict and neither clearly supersedes the other,
↪  record that conflict. Do not consult an unseen original to decide
↪  which one is true. A later answer clarifying an earlier imprecise
↪  answer is not necessarily a contradiction; compare the actual
↪  content.
Do not advertise this output as a fully recovered or mathematically
↪  validated original. Unasked rules remain unknown to you. Include
↪  every relevant supported business addition from the answers, but do
↪  not add service failure messages, requests to rephrase, or process
↪  commentary as business requirements.

OUTPUT
Return exactly one JSON object containing only the final English
↪  problem text:
{"restored_problem":"A complete, clean English problem statement with
↪  normal paragraphs and Markdown tables."}

Return the complete problem statement. Do not include a change ledger,
↪  source-quotation ledger, review, or separate unresolved list. Keep
↪  any relevant uncertainty or conflict in the problem text itself. No
↪  introduction, reasoning narrative, or text outside JSON.

EXPERIMENT BOUNDARY
This is an archive of the information actually obtained, not an
↪  algorithm required to recover every omission. Do not search for
↪  extra requirements to improve recovery, and do not judge the
↪  answerer by how much the questioner found. Different questioners
↪  may obtain different amounts. Preserve B and integrate only public,
↪  supported additions.
```

```
The fixed service reply "The problem does not specify this business
↪  detail." reports source-level non-specification or ambiguity for
↪  that question; it supplies no default and no alternative
↪  interpretation. Preserve the uncertainty in the problem text where
↪  relevant. A service failure or an unverified answer is different
↪  and must not become source absence. Do not add model-internal
↪  reasoning or use A/the hidden reference.

TABLE FIDELITY
Preserve all existing header meanings, row/column identities and data
↪  values. Markdown whitespace and alignment separators are
↪  formatting, not new business data. A genuinely corrected
↪  header/value requires explicit support from a delivered answer. Do
↪  not change numerical data to solve a suspected infeasibility.
↪  Formatting tolerance is not permission to change content.
```

D SOURCE AND REPRESENTATION EXAMPLES

This appendix presents a complete–incomplete description pair for problem 4338 and a structured example from problem 4374. The descriptions, policy mappings, numerical values, and event structure are shown directly below. Figure 1 illustrates clarification using policies from problem 4374; the 4338 description pair is a separate example. These examples explain specification content and representation semantics rather than document a measured dialogue trajectory.

D.1 COMPLETE DESCRIPTION: DATA-CENTER MAINTENANCE (4338)

The following is the complete description. Table headings abbreviate "Technician" to "Tech." in the cost matrix solely for typesetting. A data-center operator must assign six power, cooling, and network maintenance jobs to four technicians. The plan must respect technician qualifications and workload limits while providing rapid-response coverage for critical jobs.

Every job must be assigned to exactly one technician. A technician may handle several jobs, subject to the workload limit.

The workload for each maintenance job is:

Job	Workload
Job A	5
Job B	9
Job C	8
Job D	6
Job E	5
Job F	6

Equipment certifications and travel availability leave the following feasible technician choices:

Job	Available technicians
Job A	Technician 1, Technician 2, Technician 3
Job B	Technician 2, Technician 3, Technician 4
Job C	Technician 3, Technician 4, Technician 1
Job D	Technician 4, Technician 1, Technician 2
Job E	Technician 1, Technician 2, Technician 3
Job F	Technician 2, Technician 3, Technician 4

A technician's total workload is the sum of the workloads of all jobs assigned to that technician and may not exceed the limit below.

Technician	Workload limit
Technician 1	23
Technician 2	22
Technician 3	20
Technician 4	23

The estimated labor, travel, and delay cost for assigning each technician to each job is (CNY):

Job / Technician	Tech. 1	Tech. 2	Tech. 3	Tech. 4
Job A	298	381	178	308
Job B	179	247	471	397
Job C	162	501	360	202
Job D	455	265	502	209
Job E	257	334	487	433
Job F	185	362	316	210

Jobs A, C, and E involve critical equipment that remains in service. Technicians 1 and 3 carry rapid-diagnostic tools and common spare parts; at least two of these three jobs must be assigned to Technician 1 or Technician 3 to reduce the risk of prolonged downtime.

Formulate an integer optimization model that assigns one technician to every job and minimizes total labor, travel, and delay cost.

D.2 Incomplete Description: MR_4338_SEVERE_02

The following is the full English incomplete variant for problem 4338. Its severity is private experiment metadata, not part of the evaluated model's prompt. A data-center operator must allocate six maintenance jobs involving power, cooling and network equipment to four technicians. All listed work orders must be completed. A technician may handle several work orders. Every job must be assigned to exactly one technician.

Job	Workload
Job A	5
Job B	9
Job C	8
Job D	6
Job E	5
Job F	6

Technicians may be assigned only to work orders for which they appear in the following eligibility table.

Job	Available technicians
Job A	Technician 1, Technician 2, Technician 3
Job B	Technician 2, Technician 3, Technician 4
Job C	Technician 3, Technician 4, Technician 1
Job D	Technician 4, Technician 1, Technician 2
Job E	Technician 1, Technician 2, Technician 3
Job F	Technician 2, Technician 3, Technician 4

The service quotes in CNY include labor, travel and delay-related effects.

Job / Technician	Tech. 1	Tech. 2	Tech. 3	Tech. 4
Job A	298	381	178	308
Job B	179	247	471	397
Job C	162	501	360	202
Job D	455	265	502	209
Job E	257	334	487	433
Job F	185	362	316	210

Each job–technician assignment incurs its quoted cost. Use the service costs to develop a cost-effective allocation.

Formulate an integer optimization model for the work-order assignments.

D.3 Complete Reference and Initial Baseline

In the complete Golden, `AX001` indexes jobs, `AX002` indexes technicians, and `DEC001` selects technicians for jobs. The source policies are represented as follows:

Representation element	Business content
`SELECTION_FEASIBILITY.F001`	Fixed eligibility relation from the job–technician list.
`AGGREGATE_BOUND.F001`	Exactly one selected technician for every job.
`AGGREGATE_BOUND.F002`	Full job workloads summed per technician and bounded by the four workload limits.
`DISTINCT_OBJECT_COUNT.F001`	At least two of Jobs A, C, E covered by Technician 1 or 3.
`OBJECTIVE_TERM.F001`	The stated cost for each selected job–technician pair.
`objective`	Minimize the sum of cost contributions.

Table D1: Relevant complete-Golden content for problem 4338. Local Family IDs are addresses within this example, not global scoring identities.

The initial variant retains exact-one assignment, workloads, eligibility, and cost values. It does not supply the workload-limit table or the critical-job coverage rule, and its objective language is less explicit than the original minimization instruction. Its initial Golden must reflect those exact sentences, including the meaning of "cost-effective," rather than inherit an objective interpretation from the complete Golden. This example illustrates the distinction between retained and omitted requirements; it does not report an individual recovery score.

The recorded loss ratio attached to this variant describes the dataset's deletion metadata. It does not determine its initial score, which is computed by comparing the reviewed K_0^* with K^* under the uniform semantic policy.

D.4 Shared Dependencies in Refrigeration Service (4374)

Problem 4374 is an eight-job, six-technician instance and supplies the business policies illustrated in Figure 1. Most jobs require one technician, while Jobs C and E require two different technicians. Their full workloads count against each assigned technician's limit. The source further states that at most five technicians may be dispatched and that every dispatched technician receives at least 5 units of work. A preparation cost is charged once per dispatched technician, even if that technician handles several jobs. Figure 2 instead uses several examples to explain the representation, including a truck-payload rule; it is not a diagram of this entire technician instance.

The workload data are $(8, 7, 5, 5, 11, 5, 10, 10)$ for Jobs A–H; technician capacities are $(19, 17, 19, 19, 20, 23)$; preparation rates are $(271, 322, 233, 191, 283, 409)$ CNY for Technicians 1–6. These values occur in separate fixed facts, with different data axes and uses.

The relevant workload Family uses `BUSINESS_SUM`, groups by `AX002`, and sums selected `DEC001` pairs using job workloads. Its Family activation is `ALWAYS`. The upper bound is unconditional; the lower bound has a used-technician case with scalar value 5. The preparation Family instead uses a `USED_MEMBER` event indexed by technician, projects out the job axis, and has no reset axes. Its event structure is:

```
{
  "trigger_kind": "USED_MEMBER",
  "decision_refs": ["DEC001"],
  "index_axes": ["AX002"],
  "projected_out_axes": ["AX001"],
  "reset_by_axes": [],
  "member_axis_ref": "AX002",
```

```
  "ordered_axis": null
}
```

The Family's coefficient role retrieves the technician's preparation rate. It does not determine the event frequency. Charging the same rate for every job would therefore be a semantic error even if every number were copied correctly. Applying the minimum workload to unused technicians would likewise change the conditional boundary. These distinctions expose changes in charging frequency and activation even when the stated numerical values are unchanged.