SEMANTIC EXPLORATION FOR UNDERSPECIFICATION IN LLM-BASED OPTIMIZATION FORMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) can translate natural-language optimization problems into mathematical formulations. However, natural-language descriptions are often underspecified or ambiguous, and successful solver execution can leave these issues undetected: a solvable formulation need not capture the intended problem. We introduce DIVERSEFORM, a formulation assistant that explores alternative mathematical interpretations through history-conditioned diverse sampling. Each sampling step includes previous formulations and requests a different defensible interpretation. We design a declarative intermediate representation that simplifies modeling by directly expressing mathematical variables, constraints, and objectives. This representation makes differences between candidate formulations explicit, facilitating comparison during the search. Experiments show that DIVERSEFORM outperforms existing prompting-based approaches using frontier models and surpasses a state-of-the-art open model fine-tuned specifically for optimization modeling, without requiring additional training. Further analysis shows that semantic exploration helps identify distinct interpretations and uncover underspecification that calls for clarification.

1 INTRODUCTION

Large language models (LLMs) can translate natural-language descriptions of optimization problems into mathematical formulations, a task we call *auto-formulation*. Retrieval (Jiang et al., 2025b), agentic modeling and debugging (AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Thind et al., 2025), and specialized training (Huang et al., 2025a; Jiang et al., 2025a; Zhang et al., 2025) have advanced this capability. Nevertheless, errors remain a fundamental obstacle to practical use: identifying and repairing a wrong formulation can require more effort than formulating the problem manually. For LLM-based optimization formulation to deliver on its promise and become practically usable, we must meet two demanding requirements: push formulation correctness toward near perfection, and understand the sources of the remaining errors well enough to focus manual effort on selecting the intended formulation and clarifying the problem description.

We address these challenges with DIVERSEFORM, a framework that casts the LLM as a *formulation assistant* and uses history-conditioned diverse sampling to make it a reliable provider of candidate formulations. Each subsequent query includes earlier encodings and requests a different defensible interpretation, preserving explicit requirements and recording the modeling decision that changes (Figure 1). Through this exploration, DIVERSEFORM recovers reference-matching formulations missed by frontier models within five independent attempts. With Sonnet 4.6, it achieves 97.4% overall Pass@5, outperforming every evaluated baseline on all benchmarks . In the paired comparison with independent sampling, it recovers 29 previously missed problems and achieves a net gain of 24 successful problems at the same five-proposal budget.

Our study extends beyond just improving benchmark scores: diverse sampling gives us a lens for understanding where a problem description is underspecified. The differences among DIVERSEFORM's candidates expose choices about variable domains, constraint requirements, objective meaning, and modeling scope. These choices can remain hidden when independent samples repeatedly commit to the same interpretation. They also escape execution-based debugging: feasible but wrong-value outcomes dominate the baseline failures in Figure 2. By making alternative assump-

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.

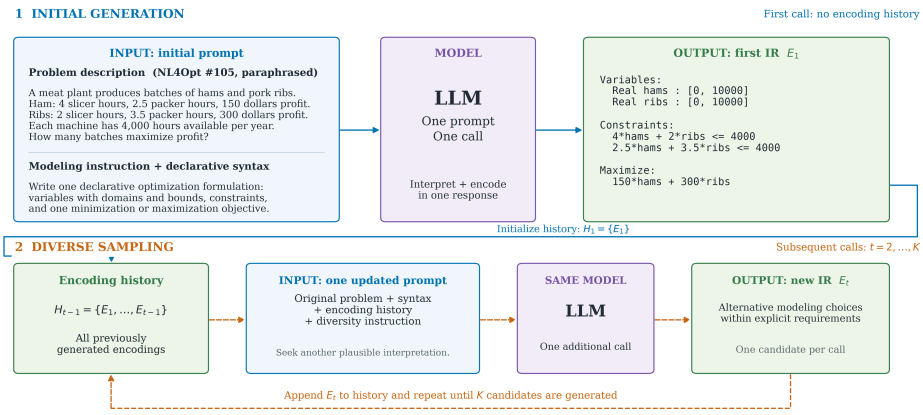

Figure 1: DIVERSEFORM conditions each subsequent query on earlier encodings and requests a different defensible interpretation of the original description.

tions explicit, DIVERSEFORM helps locate the parts of a description that require clarification and the modeling decisions that warrant review.

We investigate this diagnostic role through three complementary studies. First, we categorize the interpretation changes in recovered problems and examine their mathematical consequences. Case studies show that benchmark references impose a stated preference in one problem but omit it in another, and that alternative readings of fixed costs lead to different formulations (Section 4.7). Second, we introduce localized wording changes along identified interpretation categories and examine whether exploration recovers the original reference interpretation when independent sampling misses it (Section 4.5). Third, we compare lexical variation with diversity of optimal objective values (Section 4.6). Diverse sampling produces more distinct optima despite smaller textual differences, showing that the breadth of mathematical outcomes is not captured by surface variation alone.

We also examine generation strategy, representation, and model capability (Section 4.3). History-conditioned exploration outperforms independent sampling across three frontier models and exceeds unguided one-shot enumeration; guided enumeration matches its overall coverage under a different candidate budget. Exploration remains effective with executable code, while a declarative intermediate representation (IR) further improves coverage, particularly for the smaller model. Additional proposals benefit all seven configurations in the sampling study, and Qwen3-14B with thinking enabled reaches 87.3% Pass@5 without task-specific fine-tuning.

Our contributions are threefold. First, we introduce DIVERSEFORM, a formulation assistant that uses history-conditioned diverse sampling to explore alternative mathematical interpretations and make their modeling differences explicit (Section 3). Second, we demonstrate recovery of reference-matching formulations missed by independent sampling and higher numerical-answer coverage than every evaluated baseline across four benchmarks, with benefits extending to smaller open models (Section 4). Third, we use interpretation categories, wording interventions, case studies, and diversity measurements to identify sources of underspecification and distinguish mathematical variation from superficial output differences (Sections 4.5–4.7).

2 RELATED WORK

LLM-bsed optimization formulation. Research on auto-formulation addresses both mathematical modeling and the production of executable solver programs. NL4Opt provides structured meaning representations for optimization word problems (Ramamonjison et al., 2023); Mamo, ComplexOR, and OptiBench broaden evaluation to more demanding modeling tasks (Huang et al., 2025b; Xiao et al., 2024; Yang et al., 2025). OptiMUS, Chain-of-Experts, and OptimAI organize formulation, coding, and debugging into agentic workflows (AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Thind et al., 2025). DRoC retrieves relevant constraint knowledge, while OptiTree uses tree search over hierarchical modeling thoughts (Jiang et al., 2025b; Liu et al., 2025). ORLM, LLMOPT, and

OptiMind develop specialized modeling capabilities through training (Huang et al., 2025a; Jiang et al., 2025a; Zhang et al., 2025). These approaches provide complementary ways to improve modeling competence and inference. Our focus is the interpretation choices that remain unresolved even when a generated model executes successfully. We study whether explicitly exploring those choices recovers reference-matching formulations and reveals where the description requires clarification.

Diversity and repeated generation. Repeated generation supports several distinct objectives. Self-consistency samples reasoning paths and aggregates their answers (Wang et al., 2023), while Self-Refine uses generated feedback to iteratively improve an answer (Madaan et al., 2023). Integrative Decoding extends self-consistency to open-ended generation by conditioning on previously sampled responses and aggregating next-token predictions (Cheng et al., 2025). G2 directly targets diversity: two guides intervene in decoding to promote varied outputs while preserving generation quality (Ruan et al., 2025). These studies establish the importance of how previous outputs influence subsequent generation. DIVERSEFORM applies this idea to mathematical interpretations, asking each new candidate to change a defensible modeling assumption. It operates through standard text-generation calls and retains alternatives for review. Our evaluation connects this exploration to both candidate coverage and the specific domains, constraints, and objectives that distinguish the generated formulations.

Structured representations and formulation diversity. Structured representations expose modeling decisions before solver execution. Beyond the meaning representations in NL4Opt, Canonical Intermediate Representation encodes operational rules through constraint archetypes and candidate modeling paradigms, separating rule logic from mathematical instantiation (Lyu et al., 2026). Our declarative IR supports inspection of variables, constraints, and objectives and complements diverse generation. The relevant distinctions extend beyond syntax: equivalent rewrites can look different, and different feasible sets can share an optimum. We therefore compare lexical and numerical diversity alongside concrete interpretation changes, using the representation as a supporting tool for studying semantic exploration.

3 SEMANTIC EXPLORATION OF FORMULATIONS

DIVERSEFORM explores alternative formulations by asking the generator to reconsider modeling choices in light of its previous outputs. We first define the interpretation differences this procedure targets, then describe the generation loop and the representation used to express its candidates.

3.1 UNDERSPECIFICATION AND SEMANTIC DIVERSITY

A natural-language description is *underspecified* when it leaves a modeling choice unresolved and different defensible resolutions yield substantively different mathematical formulations. This can arise because a requirement is omitted or because a phrase admits multiple readings. For example, a stated preference for one product may be a binding requirement or a non-binding consideration; a fixed equipment cost may be incurred every period or only when the equipment is used. A defensible interpretation must respect explicit requirements and use the supplied data. Changing an unambiguous requirement or inventing a missing parameter is a modeling error, not a resolution of underspecification.

We define *semantic diversity* as variation in the substantive modeling choices expressed by a set of formulations for the same description. These choices include variable domains, the strength and scope of constraints, the meaning of parameters, and the quantity optimized. Different names, reordered constraints, and algebraically equivalent expressions do not by themselves constitute semantic diversity. Conversely, changing a preference from optional to mandatory can alter the feasible set with only a small textual edit. The target is therefore a set of distinct, defensible mathematical interpretations, each making its assumptions explicit.

Semantic diversity must also be distinguished from numerical variation. Two formulations can have different feasible sets while sharing an optimal objective value. A coefficient perturbation can change that value without representing a defensible alternative reading. Thus, neither textual distance nor distinct solver outputs alone establishes semantic diversity. Our analyses examine these measurements alongside the specific interpretation changes in generated candidates (Sections 4.5–4.7).

3.2 HISTORY-CONDITIONED EXPLORATION

Let q denote the problem description, E_t the formulation generated at step t, and z_t its solver-status description. The history is $H_{t-1} = ((E_1, z_1), \ldots, (E_{t-1}, z_{t-1}))$. At each step, the generator samples

$$E_t \sim G(\cdot \mid q, H_{t-1}, p_t), \tag{1}$$

where p_t specifies the output format and generation instruction. The first query uses an empty history and requests a formulation of q. Each subsequent query presents the original description and earlier encodings, together with their solver statuses, and asks for a different defensible interpretation. Neither their optimal objective values nor the benchmark reference answer is included in the displayed history.

The exploration instruction asks the model to change one substantive decision, preserve explicit requirements, and identify the words that justify the change. Each encoding begins with a short comment recording that decision. The prompt discourages combining changes from separate earlier candidates, keeping attention on an identifiable interpretation choice. If the model finds no further defensible reading, it records that judgment and re-emits the formulation it considers appropriate. This permits repetition when the description does not support another alternative.

Each generated encoding is compiled and executed, and its encoding and solver status are added to the history. Earlier candidates remain available as alternatives throughout the process. The reported iterative method uses a budget of five proposals. History conditioning directs later queries toward choices already made, while the diversity instruction asks the model to explore a different resolution. This is a prompting procedure: adherence to the requested interpretation changes is assessed through the empirical analyses. The exact prompts are provided in Appendix B.

3.3 DECLARATIVE REPRESENTATION

As a supporting feature, DIVERSEFORM expresses each candidate in a declarative intermediate representation (IR). An encoding $E = (V, C, O)$ specifies variable declarations V, constraints C, and an objective $O = (d, f)$, with direction $d \in \{\min, \max\}$ and expression f. Variable domains and bounds, coefficients, and the relationships among variables are written explicitly. A fixed compiler translates the encoding into solver input. The generator can therefore express an alternative through the relevant mathematical change, such as adding a preference constraint or introducing a binary activation variable, without rewriting solver-interface code.

The IR also makes candidate comparison more direct. For formulations over aligned decision variables, exact equivalence requires the same feasible set, objective values on that set, and optimization direction. Consistent renaming is handled by a correspondence between variables; formulations with auxiliary variables require comparison after projection to the original decision space. Declarative syntax exposes the objects involved in these comparisons, although equivalence still requires mathematical reasoning. We evaluate the contribution of this representation separately by applying diverse sampling to both declarative encodings and executable code (Section 4.3).

4 EXPERIMENTAL EVALUATION

4.1 EXPERIMENTAL SETUP

Benchmarks We evaluate on four public optimization-modeling benchmarks spanning elementary word problems to industrial applications: NL4Opt Ramamonjison et al. (2023), which contains elementary linear programming problems; MAMO ComplexLP Huang et al. (2025b), which requires larger, more structured formulations; IndustryOR Huang et al. (2025a), which covers real-world problems from eight industries; and ComplexOR Xiao et al. (2024), which includes challenging problems from academic papers, textbooks, and industry scenarios. We apply the same adjudicated exclusion set across methods to remove instances with defective reference answers or incomplete descriptions. Exclusion criteria and problem identifiers are provided in Appendix A.

Baseline We compare DIVERSEFORM with four prompt-based baselines. (1) Standard directly prompts a pretrained LLM to generate executable solver code. (2) Chain-of-Experts (CoE) (Xiao

Method	NL4Opt	ComplexLP	IndustryOR	ComplexOR	Total	Calls
Standard	93.3	91.7	80.6	82.4	89.9	1.7
OptiMUS	95.1	87.8	78.8	76.5	88.9	31.4
Chain-of-Experts	97.3	91.0	85.9	76.5	92.3	11.5
OptiTree	97.3	89.1	80.8	82.4	90.9	3.3
DiverseForm w/o diverse sampling (ours)	92.4	89.1	84.2	76.5	89.2	1.0
DiverseForm w/ diverse sampling (ours)	**98.7**	**98.7**	**93.8**	**88.2**	**97.4**	5.0

Table 1: Performance of DIVERSEFORM and the baselines using Sonnet

et al., 2024) organizes problem interpretation, mathematical formulation, coding, and debugging into a multi-agent workflow. (3) OptiMUS (AhmadiTeshnizi et al., 2024) uses an agentic workflow with a structured description of parameters, objectives, and constraints. (4) OptiTree (Liu et al., 2025) searches a modeling tree to retrieve a problem archetype before formulating. CoE, OptiMUS, and OptiTree include internal repair loops driven by execution feedback. These repairs are treated as part of a single run of each method and are included in its reported performance.

Implementation details The main comparison uses Sonnet 4.6 as the generator and Gurobi as the solver, with a sampling temperature of 0.7 and a budget of five attempts per problem. For DIVERSEFORM, these attempts are history-conditioned proposals; for the baselines, they are independent runs, including any internal repairs. Following prior work (Liu et al., 2025), an attempt is counted as successful when the generated formulation yields an optimal objective value within 5% relative tolerance of the benchmark reference answer. We report Pass@5, the fraction of problems for which at least one of the five attempts succeeds.

4.2 MAIN RESULTS

Table 1 compares DIVERSEFORM with the baselines using Sonnet. Without diverse sampling, DIVERSEFORM achieves 89.2% overall accuracy with only one model call per problem. This is comparable to Standard (89.9%) and OptiMUS (88.9%), which require 1.7 and 31.4 calls on average, respectively. These results suggest that a declarative representation can support competitive formulation accuracy with a smaller inference budget.

Adding diverse sampling increases overall accuracy from 89.2% to 97.4%, an improvement of 8.2 percentage points. DIVERSEFORM achieves the highest accuracy on all four benchmarks, outperforming the strongest baseline, Chain-of-Experts, by 5.1 percentage points overall while requiring fewer than half as many model calls (5.0 versus 11.5). The largest improvement over the best baseline on each benchmark occurs on IndustryOR, where accuracy increases from 85.9% to 93.7%. These results support exploring alternative mathematical interpretations as an effective way to improve formulation coverage.

Figure 2 breaks down the baseline failures. Successfully solved models with incorrect objective values form the largest category for every baseline. This pattern shows the limits of using successful execution as evidence of a correct formulation and motivates examining the modeling choices behind numerical disagreement.

4.3 ABLATION STUDY

Smaller LLMs: generality beyond frontier models. We evaluate DIVERSEFORM with smaller open models from the Qwen3 and GLM families in thinking and non-thinking configurations. Table 2 also reports our evaluation of the task-specifically fine-tuned LLMOPT-Qwen2.5-14B (Jiang et al., 2025a).

For Qwen3, increasing model size from 8B to 14B improves overall Pass@5 from 65.1% to 71.9% in the non-thinking configuration and from 82.6% to 87.3% in the thinking configuration. The differences between thinking and non-thinking configurations are larger: 17.5 percentage points at 8B and 15.4 points at 14B. The GLM models show a similar pattern: GLM-Z1-9B achieves 79.8% overall Pass@5, compared with 48.1% for GLM-4-9B. Our error inspection finds frequent

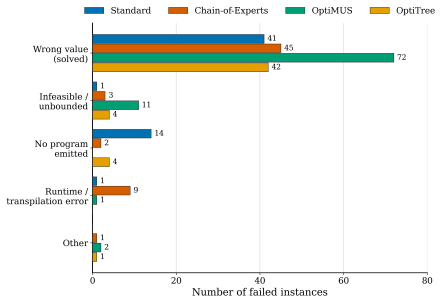

Figure 2: Baseline failures are dominated by executable formulations that return incorrect objective values.

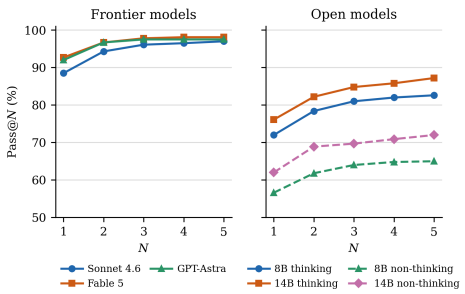

Figure 3: Numerical-answer Pass@N (%) over five proposals for frontier models and small models

Configuration	Model	NL4Opt	ComplexLP	IndustryOR	ComplexOR	Total
Fine-tuned	LLMOPT-Qwen2.5-14B	**99.6**	**84.6**	51.5	64.7	84.0
Non-thinking	GLM-4-9B-0414	79.8	26.3	16.2	17.6	48.1
	Qwen3-8B	99.1	37.2	39.4	23.5	65.1
	Qwen3-14B-FP8	98.2	53.2	47.5	41.2	71.9
Thinking	GLM-Z1-9B-0414	96.0	66.7	62.6	**88.2**	79.8
	Qwen3-8B	98.7	73.7	63.6	64.7	82.6
	Qwen3-14B-FP8	99.1	80.1	**74.7**	70.6	**87.3**

Table 2: Open-model numerical-answer Pass@5 (%). Qwen3 and GLM results use DIVERSEFORM without task-specific fine-tuning. LLMOPT-Qwen2.5-14B (Jiang et al., 2025a) is a task-specifically fine-tuned baseline evaluated in our experiments. All rows use the same benchmark and evaluation metric. Bold indicates the highest score in each column.

syntax errors in non-thinking outputs, suggesting that adherence to the declarative format remains an obstacle to effective exploration.

LLMOPT achieves 84.0% overall Pass@5. Without task-specific fine-tuning, Qwen3-14B-FP8 with thinking enabled reaches 87.3%, exceeding LLMOPT by 3.3 percentage points overall. GLM-Z1-9B achieves the highest score on ComplexOR. These results extend the evidence for DIVERSEFORM beyond frontier models, while showing that coverage varies substantially with model family and thinking configuration.

The impact of different representation on diverse sampling. Table 3 compares executable code and declarative IR under the same history-conditioned diverse-sampling instruction and five-proposal budget. Direct code generation achieves 96.5% overall Pass@5 for Sonnet 4.6 and 78.2% for Qwen3-8B with thinking enabled, showing that exploration remains effective when candidates are expressed directly as solver programs. Declarative IR improves coverage for both models, raising Pass@5 to 97.4% and 82.6%, respectively. The gain is larger for Qwen3-8B: 4.4 percentage points, compared with 0.9 points for Sonnet 4.6. These results support declarative IR as a complementary feature of DIVERSEFORM, with a larger overall benefit for the smaller model.

The impact of history-conditioned exploration We compare independent sampling with history-conditioned diverse sampling under the same initial prompt and five-proposal budget on the three frontier models. Subsequent queries in the diverse condition include earlier formulations and request a different defensible interpretation. Figure 4 shows overall Pass@5 gains of 2.8, 4.9, and 4.3 percentage points for Fable 5, Sonnet 4.6, and gpt-6-astra, respectively. Reported coverage improves or remains unchanged on every benchmark. These results support history-conditioned exploration as a way to improve numerical-answer coverage within a fixed proposal budget.

Model	Target	NL4Opt	ComplexLP	IndustryOR	ComplexOR	Total
Sonnet 4.6	Code	96.4	**98.7**	**95.7**	82.4	96.5
	Declarative	**98.7**	**98.7**	93.8	**88.2**	**97.4**
Qwen3-8B-Thinking	Code	97.3	62.8	62.6	58.8	78.2
	Declarative	**98.7**	**73.7**	**63.6**	**64.7**	**82.6**

Table 3: Numerical-answer Pass@5 (%) comparison between two target representations, executable code and declarative IR, under history-conditioned diverse sampling. Exploration remains effective with code, while declarative IR improves overall coverage for both models.

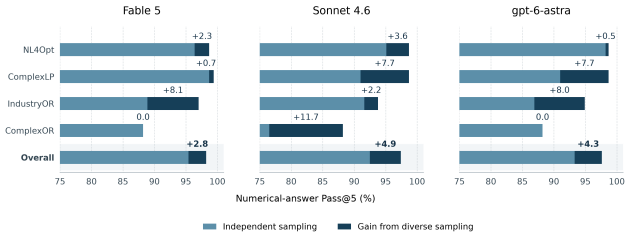

Figure 4: Numerical-answer Pass@5 gains from independent to history-conditioned diverse sampling. History-conditioned diverse sampling improves numerical-answer Pass@5 across three frontier models.

One-shot enumeration versus history-conditioned exploration. Using Sonnet 4.6 for all conditions, we compare generating multiple formulations in a single response with history-conditioned diverse sampling across iterative calls. One-shot enumeration has no explicit candidate cap and is evaluated using Pass@any over the generated candidates; iterative diverse sampling uses a five-proposal budget and is evaluated using Pass@5. We test enumeration both without guidance and with an interpretation guide derived from recovered cases (Table 7), which describes categories of underspecification and conventions for resolving them.

Table 4 shows that iterative diverse sampling achieves 97.4% overall coverage, compared with 95.2% for unguided enumeration. Both strategies require the model to identify unresolved modeling choices and explore alternative interpretations. Its higher coverage suggests that successive queries can support this exploration more effectively than unguided enumeration within a single response. Providing the interpretation guide raises enumeration coverage to 97.4%, matching iterative diverse sampling overall. These results show that explicit guidance can also support exploration. Iterative diverse sampling reaches the same overall coverage without a predefined interpretation guide.

4.4 Effects of the proposal budget

Figure 3 reports Pass@N for three frontier models and four Qwen3 configurations, each with up to five proposals. Coverage improves in all seven configurations, generally most between the first and second proposals. Fable 5 and gpt-6-astra approach saturation after three proposals. From Pass@1 to Pass@5, thinking-enabled Qwen3-8B and Qwen3-14B improve from 71.9% to 82.6% and from 76.2% to 87.3%, respectively; their non-thinking counterparts improve from 56.6% to 65.1% and from 62.0% to 71.9%. Thinking configurations maintain higher coverage throughout.

4.5 Does exploration recover interpretations under ambiguity?

The preceding experiments show that history-conditioned diverse sampling improves numerical-answer coverage over independent sampling. We next investigate whether this improvement reflects the exploration of unresolved modeling choices. We inspect the interpretation changes in recovered problems and organize them into six categories: variable domains, constraint boundaries, requirement strength, objective meaning, parameter meaning, and model scope. Table 7 defines these categories and Appendix D provides examples for each category.

Method	Guide	NL4Opt	ComplexLP	IndustryOR	ComplexOR	Total
Enumeration (1 call)	No	98.7	96.8	86.9	82.4	95.2
	Yes	**100.0**	**98.7**	90.9	**88.2**	**97.4**
Diverse sampling (5 calls)	—	98.7	**98.7**	**93.8**	**88.2**	**97.4**

Table 4: Numerical-answer coverage (%) for one-shot enumeration and history-conditioned diverse sampling using Sonnet 4.6. Diverse sampling outperforms unguided enumeration by 2.2 percentage points and matches guided enumeration overall without a predefined interpretation guide.

Category	Var-iants	Indep. misses	Diverse recov.
A Integer or fractional	7	0	-
B Equality or inequality	11	5	5
C Required or optional	11	0	-
D What the objective measures	9	4	4
E What a number means	10	4	3
F What the model includes	11	3	3
Total	59	16	**15**
Controls (unedited)	56	0	–

Figure 5: Recovery under injected ambiguity. Dashes indicate no misses to recover. Independent sampling misses 16 of 59 edited variants; diverse sampling recovers 15 of these misses.

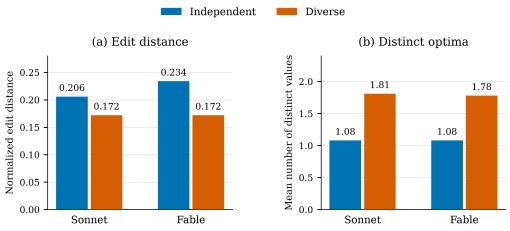

Figure 6: Lexical and numerical diversity. Diverse sampling produces smaller lexical distances (left) but more distinct optimal values (right).

To examine sensitivity to these choices, we construct 59 ambiguous variants from 56 NL4Opt problems. Each variant introduces one ambiguity through a localized wording change, without adding numerical data, and retains the original reference answer as the evaluation target. We compare independent and history-conditioned diverse sampling using Sonnet 4.6 and Gurobi, with five proposals per problem.

Table 5 shows that independent sampling solves all 56 unedited controls but misses 16 of the 59 ambiguous variants. In these missed variants, all five independent candidates commit to the same reading. Diverse sampling recovers 15 of the 16 misses (93.8%). These results show that localized wording changes can prevent independent sampling from recovering the original reference answer. Conditioning on earlier formulations and requesting alternative interpretations recovers most of these missed answers. This supports the role of exploration in broadening coverage under underspecification. The varaints are listed in Appendix E.

4.6 Evaluating diversity in generated samples

Textual and numerical variation capture different aspects of formulation diversity. Variable renaming and equivalent rewrites can increase lexical distance without changing mathematical meaning. Conversely, different feasible sets can share an optimal value: minimizing x over $[0, 1]$ or $\{0, 1\}$ yields zero in both cases. When objective definitions, directions, and units agree and both formulations are solved correctly, different optimal values establish mathematical non-equivalence, but do not establish that both interpretations are defensible.

We compare independent and diverse sampling using mean normalized pairwise edit distance and the number of distinct finite optimal values:

$$D_{\text{lex}} = \frac{2}{K(K-1)} \sum_{i<j} \frac{\text{EditDistance}(E_i, E_j)}{\max(|E_i|, |E_j|)}, \qquad D_{\text{opt}} = |\{v^\star(E_i) : i \in \mathcal{V}\}|,$$

where K is the number of candidates, $|E_i|$ is the encoding length, and $\mathcal{V}$ contains candidates with successfully computed finite optima. Optimal values within the numerical comparison tolerance are counted together, and both metrics are averaged across problems.

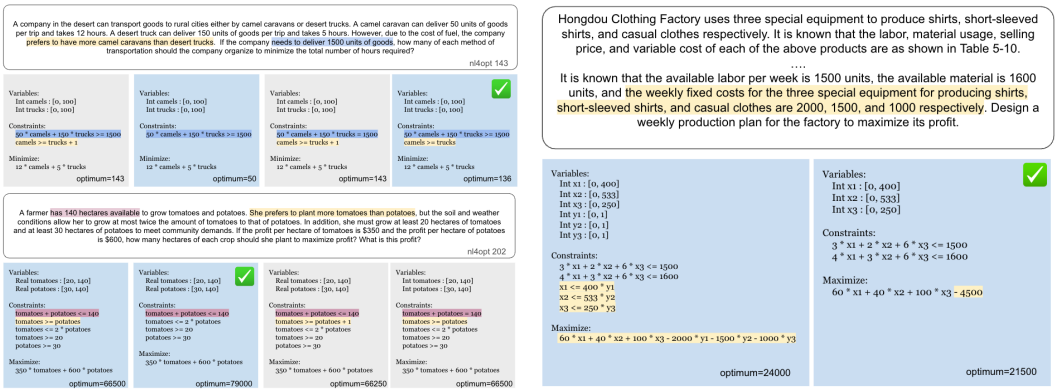

Figure 7: Semantic exploration of preference constraints in NL4Opt 143 and 202 (left) and fixed costs in IndustryOR 29 (right). Blue backgrounds indicate plausible interpretations; gray indicates inadmissible ones. Checkmarks denote reference-answer agreement.

With three candidates per problem, diverse sampling produces smaller lexical distances but more distinct optima than independent sampling for both Sonnet 4.6 and Fable 5 (Figure 6). History-conditioned exploration therefore broadens numerical outcomes despite smaller textual changes. These measurements complement the interpretation analyses: neither lexical distance nor distinct optima alone establishes semantic diversity.

4.7 Case studies of semantic exploration

Figure 7 illustrates ambiguity in requirement strength (Category C) and model scope (Category F). The reference-matching formulation imposes a preference constraint in NL4Opt 143 but omits it in NL4Opt 202, so consistently enforcing or omitting preferences cannot recover both answers. In IndustryOR 29, equipment costs may apply unconditionally or only when the corresponding product is made. Both readings are plausible, but only the unconditional reading matches the reference. These cases show how exploration exposes consequential assumptions and recovers reference answers, while numerical agreement alone does not establish the intended interpretation.

5 Discussion and conclusion

DiverseForm uses history-conditioned exploration to improve formulation coverage and expose unresolved modeling choices. It achieves 97.4% Pass@5 with Sonnet 4.6 and 87.3% with Qwen3-14B without task-specific fine-tuning. Recovering 15 of 16 ambiguous variants missed by independent sampling further supports exploring alternative interpretations. Declarative IR complements this exploration by improving coverage and making assumptions explicit.

These results support using DiverseForm as a formulation assistant, while leaving candidate selection to further review. Numerical agreement does not establish formulation correctness, and defensible interpretations may disagree with benchmark references. Future work should connect candidate differences to targeted clarification questions and reliable selection of the intended formulation.

AI use statement

Generative AI was used as part of the evaluated auto-formulation methods to produce candidate optimization formulations, as described in the paper. During manuscript preparation, we also used generative AI to assist with interpreting experimental results, drafting and revising text, improving organization and readability, formatting LaTeX, and creating or modifying scientific figures and plotting code.

The authors reviewed the AI-assisted text, interpretations, and artifacts, checking numerical claims against experimental results and figures against their underlying data. We take responsibility for the final content of this work, including all text, claims, code, and artifacts produced with generative AI assistance.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable Optimization Modeling with (MI)LP Solvers and Large Language Models. *arXiv preprint arXiv:2402.10172*, 2024. doi: 10.48550/arXiv.2402.10172. URL https://arxiv.org/abs/2402.10172.

Yi Cheng, Xiao Liang, Yeyun Gong, Wen Xiao, Song Wang, Yuji Zhang, Wenjun Hou, Kaishuai Xu, Wenge Liu, Wenjie Li, Jian Jiao, Qi Chen, Peng Cheng, and Wayne Xiong. Integrative Decoding: Improve Factuality via Implicit Self-consistency. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.01556.

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*, 2025a. doi: 10.1287/opre.2024.1233. URL https://arxiv.org/abs/2405.17743. Published online ahead of print.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for Mathematical Modeling: Towards Bridging the Gap between Natural and Mathematical Languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146. URL https://aclanthology.org/2025.findings-naacl.146/.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to Define and Solve General Optimization Problems from Scratch. In *International Conference on Learning Representations*, 2025a. URL https://arxiv.org/abs/2410.13213.

Xia Jiang, Yaoxin Wu, Chenhao Zhang, and Yingqian Zhang. DRoC: Elevating Large Language Models for Complex Vehicle Routing via Decomposed Retrieval of Constraints. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=s9zoyICZ4k.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye Hao. OptiTree: Hierarchical Thoughts Generation with Tree Search for LLM Optimization Modeling. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2510.22192.

Zhongyuan Lyu, Shuoyu Hu, Lujie Liu, Hongxia Yang, and Ming LI. Canonical Intermediate Representation for LLM-based optimization problem formulation and code generation. *arXiv preprint arXiv:2602.02029*, 2026. doi: 10.48550/arXiv.2602.02029. URL https://arxiv.org/abs/2602.02029.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. *arXiv preprint arXiv:2303.17651*, 2023. doi: 10.48550/arXiv.2303.17651. URL https://arxiv.org/abs/2303.17651.

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 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2023. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Zhiwen Ruan, Yixia Li, Yefeng Liu, Yun Chen, Weihua Luo, Peng Li, Yang Liu, and Guanhua Chen. G2: Guided Generation for Enhanced Output Diversity in LLMs. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 14116–14134. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.713. URL https://aclanthology.org/2025.emnlp-main.713/.

Raghav Thind, Youran Sun, Ling Liang, and Haizhao Yang. OptimAI: Optimization from Natural Language Using LLM-Powered AI Agents. *arXiv preprint arXiv:2504.16918*, 2025. doi: 10.48550/arXiv.2504.16918. URL https://arxiv.org/abs/2504.16918.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *International Conference on Learning Representations*, 2023. doi: 10.48550/arXiv.2203.11171. URL https://openreview.net/forum?id=1PL1NIMMrw.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-Experts: When LLMs Meet Complex Operations Research Problems. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench Meets ReSocratic: Measure and Improve LLMs for Optimization Modeling. In *International Conference on Learning Representations*, 2025. doi: 10.48550/arXiv.2407.09887. URL https://openreview.net/forum?id=fsDZwS49uY.

Xinzhi Zhang, Zeyi Chen, Humishka Zope, Hugo Barbalho, Konstantina Mellou, Marco Molinaro, Janardhan Kulkarni, Ishai Menache, and Sirui Li. OptiMind: Teaching LLMs to Think Like Optimization Experts. *arXiv preprint arXiv:2509.22979*, 2025. doi: 10.48550/arXiv.2509.22979. URL https://arxiv.org/abs/2509.22979.

A Dataset Exclusions

This section reports adjudicated dataset exclusions. The repository's adjudication protocol compares the published answer with a human-verified formulation of the description. If the formulation differs from the one under review, the human formulation is solved separately. An instance is excluded only after a defect is adjudicated; agreement among generated answers alone is insufficient. Ambiguity does not automatically warrant exclusion.

Table 5 records the exclusion-file snapshot used for this appendix. Exclusions are keyed by the dataset's `problemid`. The same exclusion set must be applied to all compared methods.

Dataset	Original	Excluded	Retained
NL4Opt	245	22	223
MAMO ComplexLP	211	55	156
IndustryOR	100	1	99
ComplexOR	18	1	17
Total	574	79	495

Table 5: Dataset sizes under the current adjudicated exclusion snapshot. Flagged cases are included in the retained count.

Exclusion criteria. *Impossible reference*: the reference is better than the verified optimum, or reports infeasibility despite a feasible verified formulation. *Non-optimal reference*: the reference is worse than the verified optimum. *Incomplete problem*: required data are absent from the description. These are the adjudication record's categories, not conclusions inferred from an LLM's failure to match a reference.

Category	Dataset	Problem identifiers
impossible label	NL4Opt	28, 32, 36, 57, 71, 73, 79, 123, 125, 163, 174, 178, 197, 210, 214
impossible label	MAMO ComplexLP	78, 131, 184, 185, 186, 208
impossible label	ComplexOR	7
not optimal label	NL4Opt	131
not optimal label	MAMO ComplexLP	198
incomplete problem	MAMO ComplexLP	59, 60, 61, 62, 64, 65, 66, 68, 69, 70, 71, 72, 74, 75, 76, 77, 79, 80, 81, 82, 84, 85, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 116, 133, 145, 146, 148, 149, 150, 154, 159, 163, 169, 170, 172

Table 6: Complete exclusion identifiers and unresolved flags.

B PROMPT TEMPLATES

The following templates are transcribed from the repository snapshot, with language names normalized to "declarative syntax" for presentation. Unicode punctuation is converted to ASCII and long lines are wrapped for display. The literal code-fence tag `rn` is retained for parser compatibility. Braced names such as `{n}` and `{k}` are runtime placeholders. These are implementation artifacts, not universal rules for interpreting optimization descriptions.

B.1 SHARED SYSTEM PROMPT AND FIRST PROPOSAL

The generator's system prompt contains the following preamble and full declarative syntax.

```
You are an operations research modeler.

Your task is to translate a natural-language optimization problem into a
formal model in the declarative syntax below, which a solver will then optimize. You
are not asked to compute the answer yourself -- you are asked to state the model
correctly.

Syntax:
Declarative syntax for optimization problems

* Variable Declarations

    Real x : [lo, hi]        continuous decision variable
    Int  n : [lo, hi]        integer decision variable
    Int  b : [0, 1]          binary decision variable

    Every variable needs finite, explicitly written bounds. Use a generous
    but
    finite upper bound (e.g. [0, 10000]) when the problem states none;
    -inf and
    inf are accepted literals but make the model much harder to solve.

* Arithmetic Expressions

    e ::= n | r | x
        | e + e | e - e | e * e | e / e | e ^ e | -e
        | sin(e) | cos(e) | tan(e) | exp(e) | log(e) | ln(e)
        | sqrt(e) | abs(e) | min(e,e) | max(e,e)

    Optimization models are almost always linear: keep the objective and
    every
    constraint linear in the decision variables whenever the problem
    allows it.
    A linear model is dispatched to an exact LP/MILP solver; a nonlinear
    one
    falls back to a slower interval engine.

* Constraints

    f ::= e = e | e <= e | e >= e | e < e | e > e
        | f And f | f Or f | Not f | f Implies f | ( f )

    An optional tolerance is appended with '~':  e - e = 0 ~ 1e-6

* Objective

    Minimize:
        <linear or nonlinear expression>

    Maximize:
        <linear or nonlinear expression>
```

```
    Exactly one objective section, and it is the last section in the file.

    The objective must be written in plain scalar variables. It may not
    contain
    an indexed component (`x[0]`) or a Vector/Matrix name -- both are build
    errors, and the failure is reported as an infeasible model.

* Declarative formulation

    Variables:
        Real x : [0, 1000]
        Int  y : [0, 100]

    Constraints:
        x + 2 * y <= 14
        3 * x - y >= 0

    Maximize:
        3 * x + 4 * y
```

B.2 ITERATIVE DIVERSE SAMPLING

The first proposal uses the shared first-proposal prompt. Later prompts include the original problem, earlier declarative encodings, their solver-status descriptions, the following instruction, and the output contract. Optimal objective values and reference answers are not included in the prior-model display.

```
You have already written {n} model(s) for this problem, shown above.

A natural-language optimization problem does not determine one model. The
same
sentences can be formalized several ways, and the differences are real
modelling
decisions, not wording:

  1. A stated preference -- "prefers", "ideally", "would like", "if
  possible" --
     can be a hard constraint, or can be left out of the model entirely.
  2. A stated restriction can be mandatory, or optional and therefore not
     binding on every solution.
  3. When the text names more than one thing to optimize, one becomes the
     objective and the others become constraints (or the objective weights
     them). Which is which is a choice.
  4. A quantity can be encoded Real or Int (or binary) -- items shipped,
  shifts
     staffed, machines bought.
  5. A limit can be per-item or over the total, per-period or over the
  horizon.
  6. A bound can include or exclude its endpoint, and a count can include or
     exclude the last unit.

Write a NEW model that takes at least one of these decisions DIFFERENTLY
from
every model above. Two rules bound that:

  * The new model must still be a defensible reading of the problem as
  written.
    Do not invent data the text does not give, do not drop a restriction the
    text states plainly, and do not change a decision the text settles. A
    model
    you would not defend is not an alternative reading.
  * Change ONE decision. Do not stack a decision from one earlier model onto
  a
    decision from another: two models whose difference you have already
```

```
    drawn
    separately is not a third reading of the text, it is their combination,
    and
    it is wrong wherever either one was.
  * If the text genuinely admits only one reading, say so rather than
    manufacturing a difference, and re-emit the model you believe is right.

Open the RN file with exactly one comment line recording the decision:

    # READING: <the decision you took differently, and the words that
    license it>

or, when you are declining:

    # READING: none -- <why the text admits only the reading above>
```

B.3 One-shot enumeration

This instruction is appended to the shared first-proposal user prompt, with `{k}` set to five. It permits fewer than five encodings when the model identifies fewer defensible readings. Its multiple-fence instruction extends the shared single-file output contract.

```
A natural-language optimization problem does not always determine one model.

First decide which of these decisions THIS text leaves open. Then write one
declarative model for each distinct defensible reading of the problem, the
reading you find most plausible first. Every model must be a reading you
would
defend: do not invent data, do not drop a restriction the text states
plainly, and do not manufacture a difference the text does not license. If the text
admits only one reading, write one model.

Output format: every model in its own '''fence, and each fence's first
line a comment recording the reading:

    # READING: <the decisions this model takes, and the words that license
    them>

You may list the open decisions in two or three lines before the first
fence. Write nothing after the last fence.
```

B.4 Interpretation guideline

The guideline is appended to the shared first-proposal prompt. The recorded implementation generates one final encoding; it does not perform a separate verification-and-edit call. The conventions were derived from recovered cases, so evaluation on those cases is in-sample. Some conventions, including treating strict comparisons as non-strict, reflect the experimental guide rather than standard mathematical semantics. Performance on held-out cases is necessary to assess whether these rules generalize.

```
The problem text may leave some modelling decisions open. Resolve them with
the conventions below, which are how problems of this kind are meant to be read.
Apply a convention only where the text is genuinely open; where the text
says otherwise, follow the text.

1. Whole or fractional. Things that are counted -- pills, bags, servings,
meals,
   bottles, vehicles, trips, machines, people, containers -- are Int, even
   when
   the text does not say so. Divisible amounts -- kilograms, litres, hours,
   acres, tons, dollars -- are Real. If the text asks for a linear program
   or
   says the decision is continuous, use Real.
```

```
2. Equality or bound.
   - "More than", "fewer than", "less than" and "exceed" between two counts
   are
     non-strict (>=, <=).
   - "A total of N", "exactly N", and a fixed order or demand quantity are
     equalities.
   - "N times as many A as B" stated as a requirement is an equality; with
   "at
     least" or "at most" it is the corresponding bound.
   - "At least" / "at most" binds only the quantity it is attached to; it
   does
     not carry over to a second quantity joined by "and".

3. Required or optional. A person's stated taste -- "prefers", "would like",
   "ideally", "likes" -- with no word of obligation is not a constraint:
   leave it
   out. Anything stated with "must", "has to", "can only", "at most" or "at
   least" is a constraint. In a problem listing business goals with numeric
   targets, the targets are constraints and the objective is the quantity
   the
   question asks for (for example total profit), not a sum of deviations.

4. What the objective measures. Optimize, and report, exactly the quantity
the
   question asks for, in its units. "Earnings by the end of year N" is the
   money
   held at that time. If the question asks for a derived quantity -- an
   area, a
   profit -- the objective is that quantity, not an intermediate one.

5. What a number means.
   - A "production rate" is output per unit time; time used = amount / rate.
   - "X% of the transportation / deliveries / trips / vehicles" is a share
   of the
     number of vehicles or trips, not of the amount they carry.
   - Something that "takes q periods to return" after period j is available
     again in period j + q + 1.
   - A decimal such as 9.15 is one number.
   - A parameter written as a nested list is indexed in the order its
   subscripts
     are named: X_{i,j} has i as the outer index.

6. What the model includes.
   - Fixed costs stated for equipment the firm has are paid whatever is
     produced; do not model them as optional unless the text says they apply
     only when the equipment is used.
   - Flows "between i and j" are between distinct i and j.
   - A network edge exists only if the source's own description lists it.

7. Infeasibility is an answer. If the constraints as stated cannot all be
met,
   the model is infeasible: do not relax, reverse or reinterpret a
   constraint to
   make the problem solvable.
```

C AMBIGUITY CATEGORIES

We classify changes by the mathematical choice they alter. The six categories below also index the injection dataset. An annotation should distinguish a plausible interpretation from a violation of explicit text, an equivalent rewrite, or an unresolved case. A label-matching optimum alone does not prove that a reading is intended. Suspected label defects are recorded separately and are not themselves a category of linguistic ambiguity.

Table 7: Categories of interpretation changes.

Category	Decision and examples
A: Integer or fractional	Whether quantities are indivisible counts or divisible amounts; for example, whole bags versus kilograms taken from bags.
B: Equality or inequality	Whether a total, ratio, or target is exact, a lower bound, or an upper bound; whether an endpoint is included.
C: Required or optional	Whether a preference or recommendation creates a binding constraint or a non-binding consideration.
D: Objective meaning	Which quantity is optimized and reported, such as a particular resource, net profit, or final wealth.
E: Parameter meaning	What a number measures: the denominator of a share, a per-member versus per-team quantity, gross versus net amounts, or rate units.
F: Model scope	Which entities or terms a restriction includes, such as per-item versus aggregate bounds or unconditional versus activated costs.

D Case Studies of Interpretation Exploration

The following examples illustrate Categories A–F using recorded diverse sampling traces from Sonnet 4.6 and Fable 5. We include two problems per category, except Category D, which contains one problem in the audited union. Each example shows the first encoding and the first later encoding that passes the benchmark check. Draw numbers identify their positions in the original sequence; intervening draws are omitted.

Yellow highlights identify changed encoding rows and the relevant wording. Blue denotes provisionally plausible readings; gray denotes a reading that softens explicit requirements. Checkmarks indicate agreement under the benchmark's 5% numerical criterion, or agreement with a no-solution label. The solver outcomes are those recorded for the full Gurobi formulations, not new solver runs. Comments are removed for readability, and omissions from descriptions or encodings are explicitly marked. Numerical agreement does not establish semantic equivalence; plausibility judgments remain subject to human adjudication.

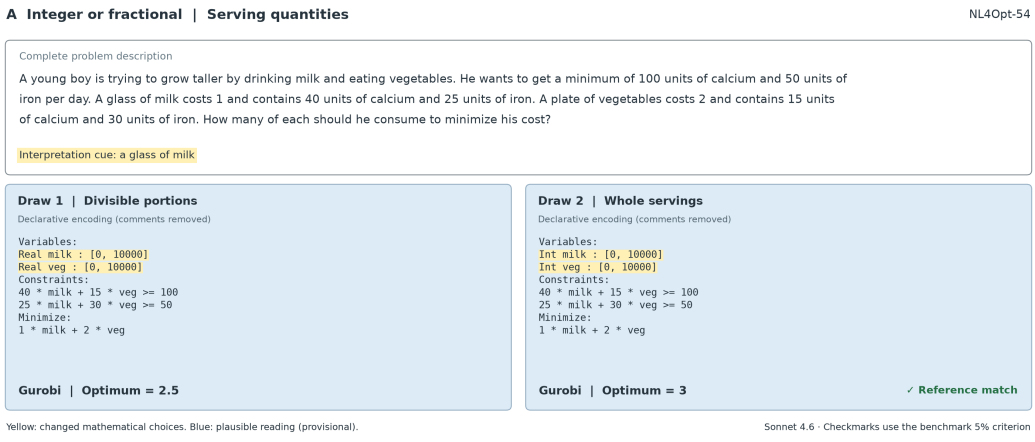

Figure 8: **Category A.** NL4Opt 54: fractional portions versus whole glasses and plates. Integer servings change the optimum from 2.5 to the reference value of 3.

E Ambiguity-Injection Problems

E.1 Construction and evaluation protocol

The injection dataset contains 26 edited descriptions and their 26 unedited controls. A base problem is operationally stable when the default executor matches its reference answer in two independent samples; this does not certify that its description has no ambiguity. Each variant changes wording to

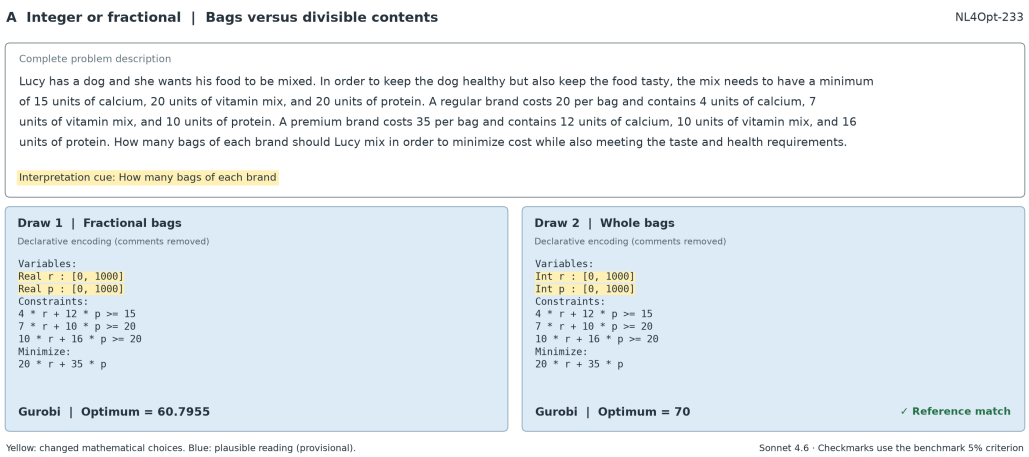

Figure 9: **Category A.** NL4Opt 233: fractional bag contents versus whole bags. The reference-matching encoding imposes integer quantities.

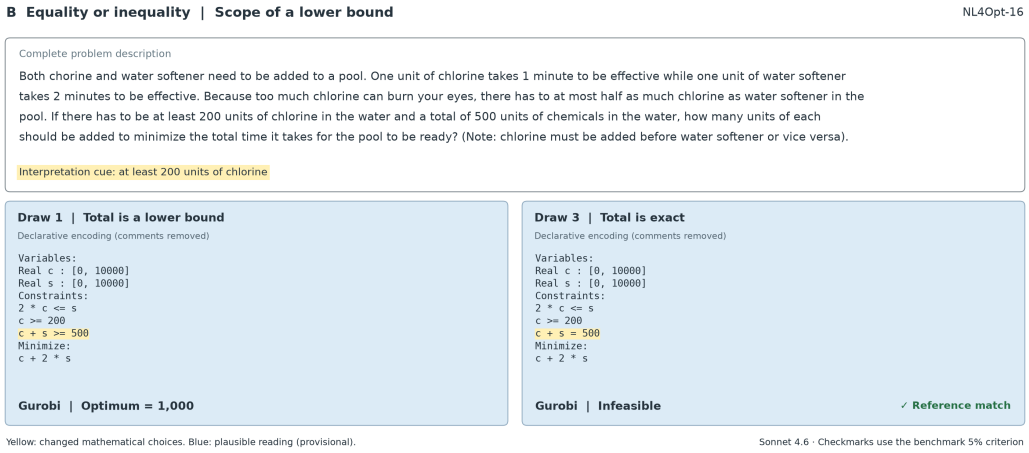

Figure 10: **Category B.** NL4Opt 16: whether "at least" also qualifies the total chemical quantity. The exact-total encoding is infeasible, matching the reference no-solution label; no finite optimum is reported for that encoding.

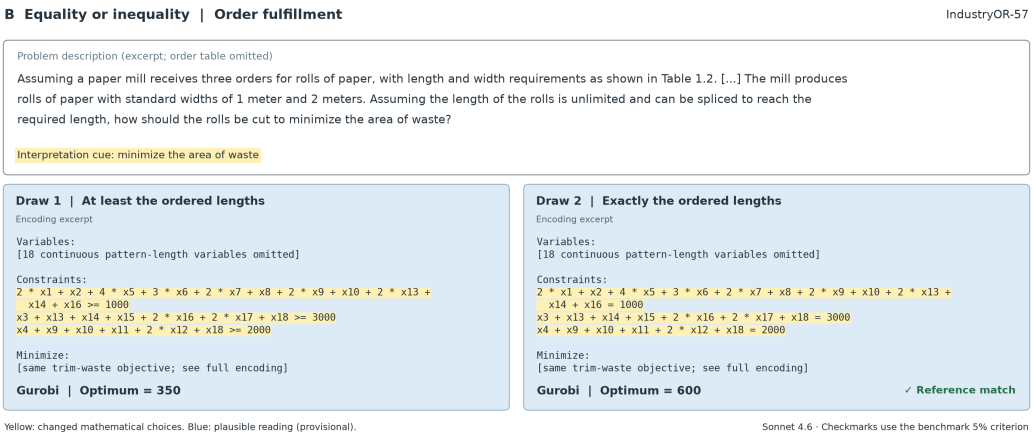

Figure 11: **Category B.** IndustryOR 57: minimum versus exact order fulfillment. Encoding excerpts expose the changed demand relations; recorded optima of 350 and 600 come from the full formulations.

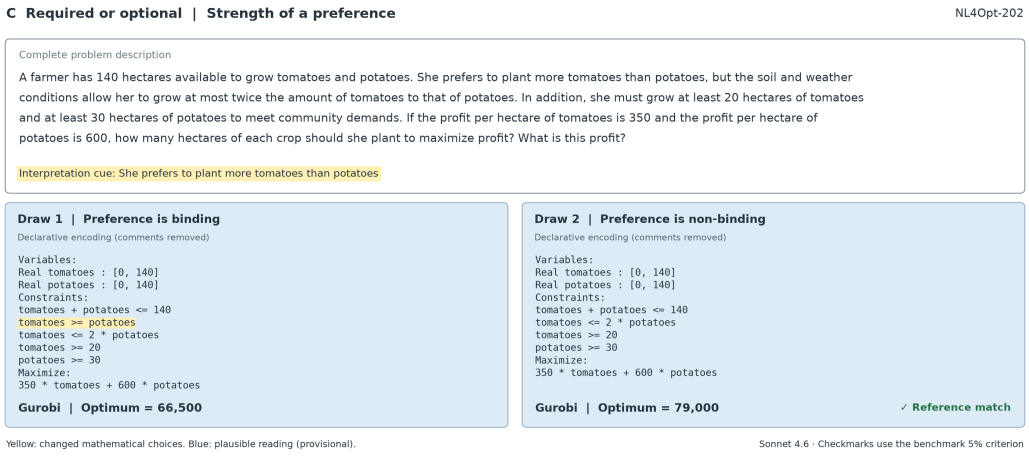

Figure 12: **Category C.** NL4Opt 202: enforcing versus omitting the preference for tomatoes. Omitting the preference yields the reference value of 79,000.

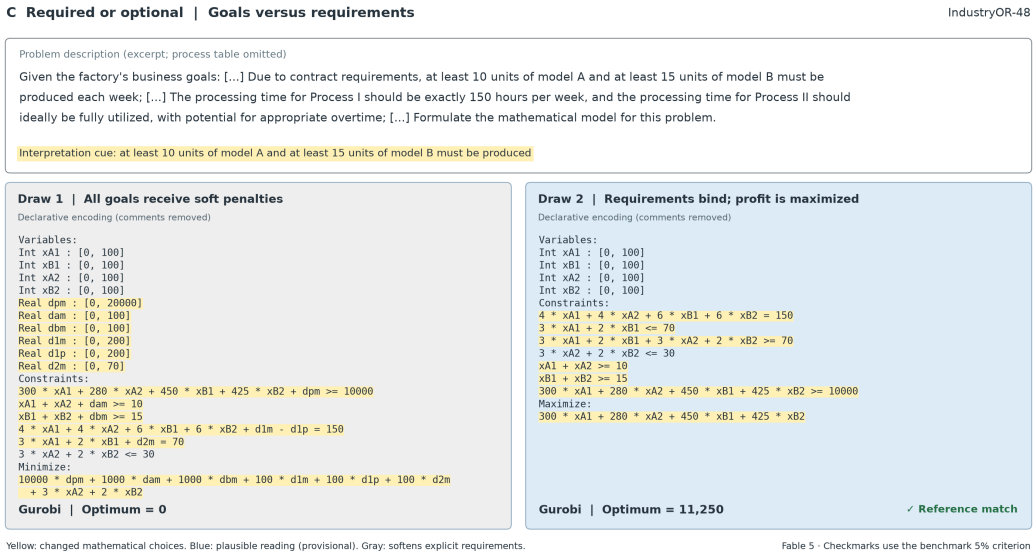

Figure 13: **Category C.** IndustryOR 48: soft goal penalties versus binding requirements and a profit objective. The gray encoding softens explicit requirements; the change also affects objective meaning (Category D).

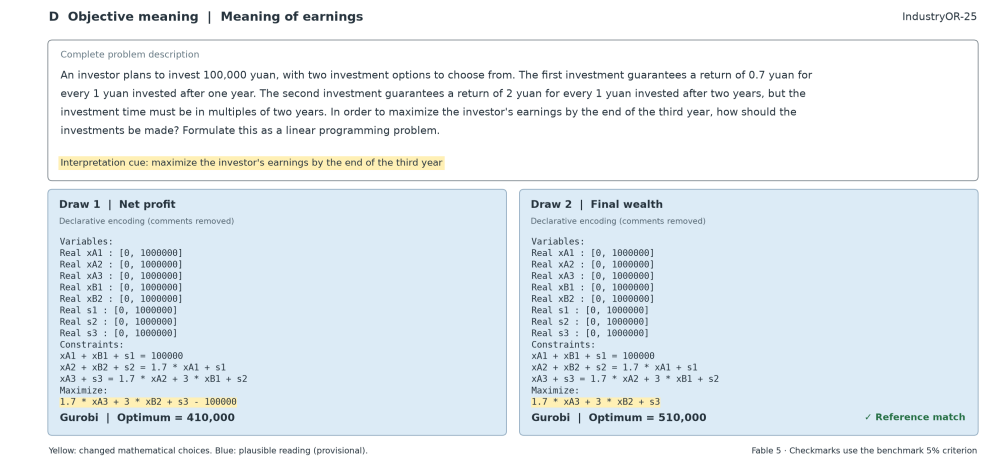

Figure 14: **Category D.** IndustryOR 25: net profit versus final wealth. Adding the initial capital changes the reported value from 410,000 to 510,000 without changing the maximizing investment plan.

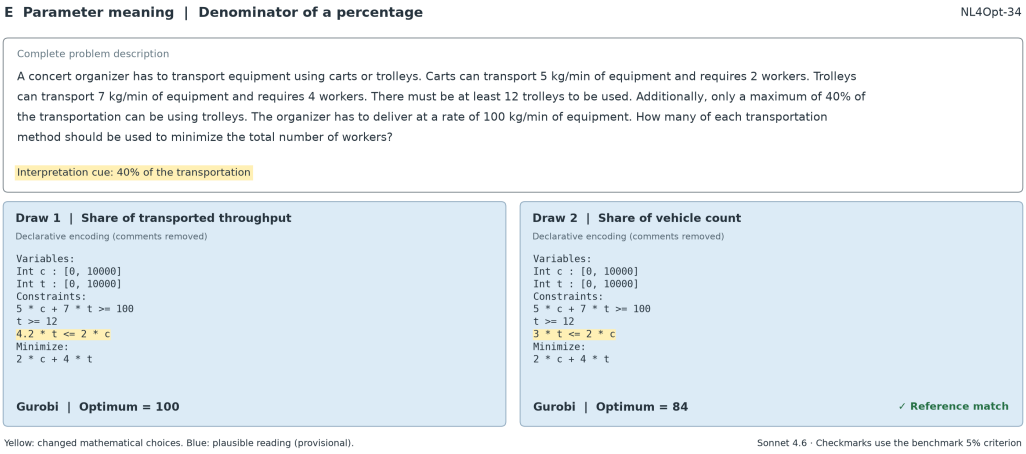

Figure 15: **Category E.** NL4Opt 34: interpreting the trolley share as a fraction of transported throughput or of vehicle count. The latter yields the reference value of 84.

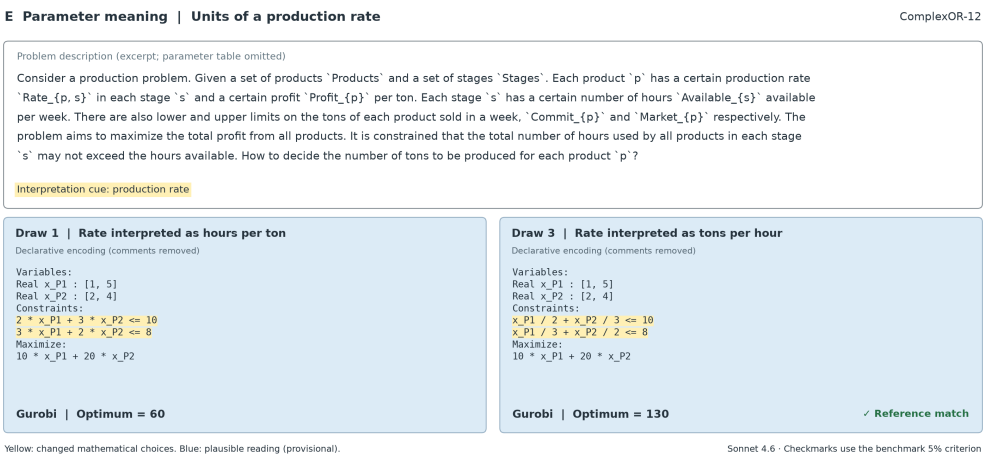

Figure 16: **Category E.** ComplexOR 12: interpreting production rates as hours per ton or tons per hour. Dividing production by the rate yields the reference value of 130.

F Model scope | Scope of fixed equipment costs IndustryOR-23

Problem description (excerpt; product table omitted)

Hongdou Clothing Factory uses three special equipment to produce shirts, short-sleeved shirts, and casual clothes respectively. [...] It is known that the available labor per week is 1500 units, the available material is 1600 units, and the weekly fixed costs for the three special equipment for producing shirts, short-sleeved shirts, and casual clothes are 2000, 1500, and 1000 respectively. Design a weekly production plan for the factory to maximize its profit.

Interpretation cue: weekly fixed costs

Draw 1 | Costs incurred only when activated

Declarative encoding (comments removed)

```
Variables:
Int x1 : [0, 500]
Int x2 : [0, 750]
Int x3 : [0, 250]
Int b1 : [0, 1]
Int b2 : [0, 1]
Int b3 : [0, 1]
Constraints:
3 * x1 + 2 * x2 + 6 * x3 <= 1500
4 * x1 + 3 * x2 + 6 * x3 <= 1600
x1 <= 500 * b1
x2 <= 750 * b2
x3 <= 250 * b3
Maximize:
60 * x1 + 40 * x2 + 100 * x3 - 2000 * b1 - 1500 * b2 - 1000 * b3
```

Gurobi | Optimum = 24,000

Draw 3 | Costs incurred every week

Declarative encoding (comments removed)

```
Variables:
Int x1 : [0, 500]
Int x2 : [0, 750]
Int x3 : [0, 250]
Constraints:
3 * x1 + 2 * x2 + 6 * x3 <= 1500
4 * x1 + 3 * x2 + 6 * x3 <= 1600
Maximize:
60 * x1 + 40 * x2 + 100 * x3 - 2000 - 1500 - 1000
```

Gurobi | Optimum = 21,500 ✓ Reference match

Yellow: changed mathematical choices. Blue: plausible reading (provisional). Sonnet 4.6 · Checkmarks use the benchmark 5% criterion

Figure 17: **Category F.** IndustryOR 23: conditional equipment activation costs versus unconditional weekly fixed costs. The latter yields the reference value of 21,500. The identifier is the benchmark problemid.

F Model scope | Permission to import commodities IndustryOR-40

Problem description (excerpt; input coefficients omitted)

The independent country of Carelland mainly exports four commodities: steel, engines, electronic components, and plastic. Carelland's Minister of Finance (i.e., Minister of Economy) wants to maximize exports and minimize imports. [...] Write a mathematical program to maximize domestic GDP and solve the problem using AMPL.

Interpretation cue: maximize exports and minimize imports

Draw 1 | Each commodity must be self-sufficient

Declarative encoding (comments removed)

```
Variables:
Real xs : [0, 200000]
Real xm : [0, 650000]
Real xe : [0, 200000]
Real xp : [0, 60000]
Real es : [0, 200000]
Real em : [0, 650000]
Real ee : [0, 200000]
Real ep : [0, 60000]
Constraints:
es = xs - 0.8 * xm - 0.01 * xe - 0.2 * xp
em = xm - 0.02 * xs - 0.01 * xe - 0.03 * xp
ee = xe - 0.15 * xm - 0.05 * xp
ep = xp - 0.01 * xs - 0.11 * xm - 0.05 * xe
6 * xs + 12 * xm + 6 * xe + 24 * xp <= 830000
xm <= 650000
xp <= 60000
Maximize:
500 * es + 1500 * em + 300 * ee + 1200 * ep - 250 * xs - 300 * xm - 50 *
  xe - 300 * xp
```

Gurobi | Optimum = 3.62886e+07

Draw 2 | Imports may cover production inputs

Declarative encoding (comments removed)

```
Variables:
Real xs : [0, 200000]
Real xm : [0, 650000]
Real xe : [0, 200000]
Real xp : [0, 60000]
Real es : [-1000000, 1000000]
Real em : [-1000000, 1000000]
Real ee : [-1000000, 1000000]
Real ep : [-1000000, 1000000]
Constraints:
es = xs - 0.8 * xm - 0.01 * xe - 0.2 * xp
em = xm - 0.02 * xs - 0.01 * xe - 0.03 * xp
ee = xe - 0.15 * xm - 0.05 * xp
ep = xp - 0.01 * xs - 0.11 * xm - 0.05 * xe
6 * xs + 12 * xm + 6 * xe + 24 * xp <= 830000
xm <= 650000
xp <= 60000
Maximize:
500 * es + 1500 * em + 300 * ee + 1200 * ep - 250 * xs - 300 * xm - 50 *
  xe - 300 * xp
```

Gurobi | Optimum = 4.30908e+07 ✓ Reference match

Yellow: changed mathematical choices. Blue: plausible reading (provisional). Fable 5 · Checkmarks use the benchmark 5% criterion

Figure 18: **Category F.** IndustryOR 40: nonnegative net exports versus allowing imports to cover production inputs. The second encoding permits negative net exports and matches the reference under the numerical tolerance.

expose one interpretation axis without introducing numerical data absent from the original problem. The reference answer remains fixed.

The construction code checks that the original reading reproduces the reference and that the specified alternative fails the 5%-tolerance answer check. This selection makes interpretation changes numerically observable and excludes ambiguities whose readings share an optimum. The results therefore characterize this constructed diagnostic set, not the frequency of ambiguity in NL4Opt.

The recorded evaluation uses Sonnet 4.6, Gurobi, at most five proposals, and reference-based early stopping. The saved result summary records eight variants missed by independent sampling and recovery of all eight in each of two diverse runs. Raw traces were not available in the local snapshot used to assemble this appendix; the inventory below is extracted from the variant-definition file.

E.2 B. Equality or inequality

Variant 1: NL4Opt-17 (bare required amount: exact vs minimum)

Category. B, equality or inequality. **Round.** v3. **Label.** 1,400. **I.i.d., all five draws.** 1,600. **Diverse.** draw 2 / 2.

Original problem. A cleaning company uses a cleansing chemical and odor-removing chemical to clean a house. Each unit of the cleansing chemical takes 4 units to be effective while each unit of the odor-removing chemical takes 6 minutes to be effective. **The company must use at least 100 units of the cleansing chemical.** In total, at least 300 units of chemicals can be used per house. Further, because the cleansing chemical is strong, there can be at most twice the amount of cleansing chemical as odor-removing chemical. How many units of each should be used to minimize the total time it takes for a house to be cleaned?

Edited problem. A cleaning company uses a cleansing chemical and odor-removing chemical to clean a house. Each unit of the cleansing chemical takes 4 units to be effective while each unit of the odor-removing chemical takes 6 minutes to be effective. **The company must use 100 units of the cleansing chemical.** In total, at least 300 units of chemicals can be used per house. Further, because the cleansing chemical is strong, there can be at most twice the amount of cleansing chemical as odor-removing chemical. How many units of each should be used to minimize the total time it takes for a house to be cleaned?

Why it is ambiguous. 'must use 100 units' names a required amount that may be exact or a minimum.

Readings. Label's reading: cleansing >= 100 (optimum 1,400). Alternative: cleansing = 100 (optimum 1,600).

Wrong encoding (i.i.d. draw 0):

```
# c = units of cleansing chemical, o = units of odor-removing chemical
# Each unit of cleansing chemical takes 4 minutes; each unit of odor-
# removing takes 6 minutes

Variables:
    Real c : [0, 10000]
    Real o : [0, 10000]

Constraints:
    # Must use exactly 100 units of cleansing chemical
    c = 100
    # Total chemicals used must be at least 300 units
    c + o >= 300
    # Cleansing chemical is at most twice the amount of odor-removing
    # chemical
    c <= 2 * o
```

```
Minimize:
    4 * c + 6 * o
```

Alternative reading, as written:
`c = 100`
Label's reading:
`c >= 100`

VARIANT 2: NL4OPT-30 (STATED RATIO: EXACT VS CAP)

Category. B, equality or inequality. **Round.** v3. **Label.** 29. **I.i.d., all five draws.** 36. **Diverse.** draw 2 / 2.

Original problem. A construction company in the tropics uses cows and elephants to carry bricks. A cow can carry 20 bricks on its back while an elephant can carry 50 bricks on its back. To avoid having elephants create too much traffic, the number of elephant cannot exceed the number of cows. **In addition, there can be at most twice the number of cows as elephants.** If the company needs to transport at least 1000 bricks, find the minimum number of animals, cows and elephants, that can be used..

Edited problem. A construction company in the tropics uses cows and elephants to carry bricks. A cow can carry 20 bricks on its back while an elephant can carry 50 bricks on its back. To avoid having elephants create too much traffic, the number of elephant cannot exceed the number of cows. **In addition, there should be twice the number of cows as elephants.** If the company needs to transport at least 1000 bricks, find the minimum number of animals, cows and elephants, that can be used..

Why it is ambiguous. 'there should be twice the number of cows as elephants' reads as a target herd mix or as a limit on cows.

Readings. Label's reading: cows <= 2 elephants (optimum 29). Alternative: cows = 2 elephants (optimum 36).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int cows : [0, 1000]
    Int elephants : [0, 1000]

Constraints:
    # Brick capacity must meet the demand
    20 * cows + 50 * elephants >= 1000
    # Elephants cannot exceed cows
    elephants <= cows
    # Twice as many cows as elephants
    cows = 2 * elephants

Minimize:
    cows + elephants
```

Alternative reading, as written:
`cows = 2 * elephants`
Label's reading:
`cows <= 2 * elephants`

VARIANT 3: NL4OPT-66 (STATED RATIO: EXACT VS MINIMUM)

Category. B, equality or inequality. **Round.** v2. **Label.** 19. **I.i.d., all five draws.** 21. **Diverse.** draw 1 / 1.

Original problem. A corn farmer sends his corn to the city by either tractor or car. A tractor can carry 40 kg of corn while a car can carry 20 kg of corn. Since tractors are very slow, **the number of**

cars used has to be at least twice the number of tractors used. If at least 500 kg of corn need to be sent to the city, minimize the total number of tractors and cars needed.

Edited problem. A corn farmer sends his corn to the city by either tractor or car. A tractor can carry 40 kg of corn while a car can carry 20 kg of corn. Since tractors are very slow, **twice as many cars as tractors have to be used.** If at least 500 kg of corn need to be sent to the city, minimize the total number of tractors and cars needed.

Why it is ambiguous. 'twice as many cars as tractors have to be used' states a ratio that may be exact or a floor.

Readings. Label's reading: cars >= 2 tractors (optimum 19). Alternative: cars = 2 tractors (optimum 21).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int t : [0, 1000]   # number of tractors
    Int c : [0, 1000]   # number of cars

Constraints:
    # each tractor carries 40 kg, each car carries 20 kg, at least 500
    # kg total
    40 * t + 20 * c >= 500
    # twice as many cars as tractors
    c = 2 * t

Minimize:
    t + c
```

Alternative reading, as written:

```
c = 2 * t
```

Label's reading:

```
c >= 2 * t
```

VARIANT 4: NL4OPT-203 (STATED SHARE: EXACT VS CAP)

Category. B, equality or inequality. **Round.** v2. **Label.** 2,500. **I.i.d., all five draws.** 2,255. **Diverse.** draw 3 / 3.

Original problem. A dog school trains labradors and golden retrievers to deliver newspaper. A labrador can deliver 7 newspapers at a time and requires 5 small bone treats for service. A golden retriever can deliver 10 newspapers at a time and requires 6 small bone treats per service. The school only has 1500 small bone treats available. In addition, at least 50 golden retrievers **must be used and at most 60% of the dogs can be labradors.** How many of each dog should be used to maximize the number of newspapers that can be delivered?

Edited problem. A dog school trains labradors and golden retrievers to deliver newspaper. A labrador can deliver 7 newspapers at a time and requires 5 small bone treats for service. A golden retriever can deliver 10 newspapers at a time and requires 6 small bone treats per service. The school only has 1500 small bone treats available. In addition, at least 50 golden retrievers **must be used. Also, 60% of the dogs should be labradors.** How many of each dog should be used to maximize the number of newspapers that can be delivered?

Why it is ambiguous. '60% of the dogs should be labradors' reads as a target mix or as a limit.

Readings. Label's reading: labradors <= 60% (optimum 2,500). Alternative: labradors = 60% (optimum 2,255).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int L : [0, 10000]   # number of labradors
```

```
    Int G : [0, 10000]   # number of golden retrievers

Constraints:
    5 * L + 6 * G <= 1500
    G >= 50
    # 60% of all dogs must be labradors: L = 0.6 * (L + G) => 0.4*L =
    # 0.6*G => 2*L = 3*G
    2 * L = 3 * G

Maximize:
    7 * L + 10 * G
```

Alternative reading, as written:
`2 * L = 3 * G`
Label's reading:
`L <= 0.6 * (L + G)`

VARIANT 5: NL4OPT-228 (INVESTMENT RATIO: EXACT VS MINIMUM)

Category. B, equality or inequality. **Round.** v3. **Label.** 225. **I.i.d., all five draws.** 190. **Diverse.** draw 1 / 1.

Original problem. Jacob has $3000 to invest. He has decided to invest in his favorite clothing company and his favorite tech company. **He has decided that the money invested in his favorite clothing company must be at least four times as much as the amount invested in his favorite tech company.** He has also limited himself to invest at most $500 in his favorite tech company. If the money invested in his favorite clothing company earns 7% and the money invested in his favorite tech company earns 10%, how much should he invest in each to maximize his profit?

Edited problem. Jacob has $3000 to invest. He has decided to invest in his favorite clothing company and his favorite tech company. **He has decided to invest four times as much in his favorite clothing company as in his favorite tech company.** He has also limited himself to invest at most $500 in his favorite tech company. If the money invested in his favorite clothing company earns 7% and the money invested in his favorite tech company earns 10%, how much should he invest in each to maximize his profit?

Why it is ambiguous. 'invest four times as much in A as in B' reads as a fixed split or as a lower limit on A.

Readings. Label's reading: clothing >= 4 tech (optimum 225). Alternative: clothing = 4 tech (optimum 190).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Real clothing : [0, 3000]
    Real tech : [0, 500]

Constraints:
    clothing + tech <= 3000
    clothing = 4 * tech
    tech <= 500

Maximize:
    0.07 * clothing + 0.10 * tech
```

Alternative reading, as written:
`clothing = 4 * tech`
Label's reading:
`clothing >= 4 * tech`

E.3 D. What the objective measures

Variant 6: NL4Opt-1 (which resource the usage objective counts)

Category. D, what the objective measures. **Round.** v2. **Label.** 350. **I.i.d., all five draws.** 950. **Diverse.** draw 4 / 3.

Original problem. A lab has 1000 units of medicinal ingredients to make two pills, a large pill and a small pill. A large pill requires 3 units of medicinal ingredients and 2 units of filler. A small pill requires 2 units of medicinal ingredients and 1 unit of filler. The lab has to make at least 100 large pills. However, since small pills are more popular at least 60% of the total number of pills must be small. How many of each should be made **to minimize the total number of filler material needed?**

Edited problem. A lab has 1000 units of medicinal ingredients to make two pills, a large pill and a small pill. A large pill requires 3 units of medicinal ingredients and 2 units of filler. A small pill requires 2 units of medicinal ingredients and 1 unit of filler. The lab has to make at least 100 large pills. However, since small pills are more popular at least 60% of the total number of pills must be small. How many of each should be made **to minimize the total amount of material needed?**

Why it is ambiguous. Both inputs are material measured in units; 'total' invites summing them.

Readings. Label's reading: minimize filler (optimum 350). Alternative: minimize filler + medicinal ingredients (optimum 950).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int L : [0, 500]   # number of large pills
    Int S : [0, 1000]  # number of small pills

Constraints:
    # Medicinal ingredient supply
    3 * L + 2 * S <= 1000

    # Must make at least 100 large pills
    L >= 100

    # At least 60% of total pills must be small: S >= 0.6*(L+S) => 2*S
    # >= 3*L
    2 * S >= 3 * L

Minimize:
    # Total material = medicinal + filler
    # Large pill: 3 medicinal + 2 filler = 5 total
    # Small pill: 2 medicinal + 1 filler = 3 total
    5 * L + 3 * S
```

Alternative reading, as written:
`Minimize: 5 * L + 3 * S`
Label's reading:
`Minimize: 2 * L + 1 * S`

Variant 7: NL4Opt-13 (which site's dose the objective counts)

Category. D, what the objective measures. **Round.** v3. **Label.** 1.5. **I.i.d., all five draws.** 2.25. **Diverse.** draw 3 / 2.

Original problem. A patient is undergoing radiation treatment involving two beams, Beam 1 and Beam 2. Beam 1 delivers a dose of 0.3 units of medicine per minute to the benign area of the pancreas and 0.2 units of medicine per minute to the benign area of the skin. Beam 2 delivers 0.2 units of medicine per minute to the benign area of the pancreas and 0.1 units of medicine per minute

to the benign area of the skin. In addition, beam 1 delivers 0.6 units of medicine per minute to the tumor and beam 2 delivers 0.4 units of medicine per minute to the tumor. At most 4 units of medicine should be received by the skin and at least 3 units of medicine should be delivered to the tumor. How many minutes of each beam should be used **to minimize the total radiation received by the pancreas?**

Edited problem. A patient is undergoing radiation treatment involving two beams, Beam 1 and Beam 2. Beam 1 delivers a dose of 0.3 units of medicine per minute to the benign area of the pancreas and 0.2 units of medicine per minute to the benign area of the skin. Beam 2 delivers 0.2 units of medicine per minute to the benign area of the pancreas and 0.1 units of medicine per minute to the benign area of the skin. In addition, beam 1 delivers 0.6 units of medicine per minute to the tumor and beam 2 delivers 0.4 units of medicine per minute to the tumor. At most 4 units of medicine should be received by the skin and at least 3 units of medicine should be delivered to the tumor. How many minutes of each beam should be used **to minimize the total radiation received by the benign area?**

Why it is ambiguous. The text names a benign area of the pancreas and a benign area of the skin; 'the benign area' may be the treated organ's, while 'total' invites summing both.

Readings. Label's reading: minimize pancreas dose (optimum 1.5). Alternative: minimize pancreas + skin dose (optimum 2.25).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Real b1 : [0, 10000]
    Real b2 : [0, 10000]

Constraints:
    # Skin constraint: at most 4 units
    0.2 * b1 + 0.1 * b2 <= 4
    # Tumor constraint: at least 3 units
    0.6 * b1 + 0.4 * b2 >= 3
    # Non-negativity (already enforced by bounds)

Minimize:
    0.3 * b1 + 0.2 * b2 + 0.2 * b1 + 0.1 * b2
```

Alternative reading, as written:
`Minimize: 0.3 * b1 + 0.2 * b2 + 0.2 * b1 + 0.1 * b2`
Label's reading:
`Minimize: 0.3 * b1 + 0.2 * b2`

VARIANT 8: NL4OPT-95 (STOCKED VS PROCURED INPUT (WHICH INGREDIENTS COUNT))

Category. D, what the objective measures. **Round.** v3. **Label.** 810. **I.i.d., all five draws.** 1,560. **Diverse.** draw 4 / 3.

Original problem. An dog hospital has 2000 units of tooth medication to make both small and large bones. A small bone requires 10 units of tooth medication and 12 units of meat. A large bone requires 15 units of tooth medication and 15 units of meat. Since most dogs prefer the small bones, at least 50% of the bones made must be small. In addition, the hospital must make at least 30 large bones. How many of each bone should be made **to minimize the amount of meat needed?**

Edited problem. An dog hospital has 2000 units of tooth medication to make both small and large bones. A small bone requires 10 units of tooth medication and 12 units of meat. A large bone requires 15 units of tooth medication and 15 units of meat. Since most dogs prefer the small bones, at least 50% of the bones made must be small. In addition, the hospital must make at least 30 large bones. How many of each bone should be made **to minimize the total amount of ingredients needed?**

Why it is ambiguous. Tooth medication is also an ingredient of every bone, though it comes from a stock already on hand while meat must be supplied.

Readings. Label's reading: minimize meat (optimum 810). Alternative: minimize meat + tooth medication (optimum 1,560).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int s : [0, 200]   # number of small bones
    Int l : [30, 133]  # number of large bones

Constraints:
    # tooth medication limit
    10 * s + 15 * l <= 2000

    # at least 50% of bones must be small: s >= 0.5*(s+l) => s >= l
    s >= l

    # at least 30 large bones
    l >= 30

Minimize:
    # total ingredients = tooth medication + meat per bone type
    22 * s + 30 * l
```

Alternative reading, as written:
`Minimize: 22 * s + 30 * l`
Label's reading:
`Minimize: 12 * s + 15 * l`

VARIANT 9: NL4OPT-189 (NAMED NUTRIENT VS ALL TRACKED NUTRIENTS)

Category. D, what the objective measures. **Round.** v3. **Label.** 582. **I.i.d., all five draws.** 882. **Diverse.** draw 2 / 2.

Original problem. A scientist in the arctic needs to get enough vitamin D and vitamin C in his diet by drinking orange and apple juice. Each box of orange juice contains 10 units of vitamin D and 8 units of vitamin C. Each box of apple juice contains 12 units of vitamin D and 6 units of vitamin C. Since the scientist prefers apple juice, he must drink at least 3 times as many apple juice boxes and orange juice boxes. However, he must also drink at least 3 orange juice boxes. To avoid a vitamin C overdose, the scientist can consume at most 300 units of vitamin C. How many of each juice box should he drink **to maximize his total vitamin D intake?**

Edited problem. A scientist in the arctic needs to get enough vitamin D and vitamin C in his diet by drinking orange and apple juice. Each box of orange juice contains 10 units of vitamin D and 8 units of vitamin C. Each box of apple juice contains 12 units of vitamin D and 6 units of vitamin C. Since the scientist prefers apple juice, he must drink at least 3 times as many apple juice boxes and orange juice boxes. However, he must also drink at least 3 orange juice boxes. To avoid a vitamin C overdose, the scientist can consume at most 300 units of vitamin C. How many of each juice box should he drink **to maximize his total vitamin intake?**

Why it is ambiguous. He needs both vitamins; vitamin C is capped against overdose, so the one to push up may be vitamin D only, yet 'total vitamin intake' reads as the sum.

Readings. Label's reading: maximize vitamin D (optimum 582). Alternative: maximize vitamin D + C (optimum 882).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int orange : [3, 37]
```

```
    Int apple  : [0, 50]

Constraints:
    # at least 3 times as many apple boxes as orange boxes
    apple >= 3 * orange
    # at least 3 orange juice boxes
    orange >= 3
    # vitamin C limit: at most 300 units
    8 * orange + 6 * apple <= 300

Maximize:
    18 * orange + 18 * apple
```

Alternative reading, as written:

```
Maximize:  18 * orange + 18 * apple
```

Label's reading:

```
Maximize:  10 * orange + 12 * apple
```

E.4 E. WHAT A NUMBER MEANS

VARIANT 10: NL4OPT-99 (PER MEMBER VS PER GROUP (SCOPE OF 'EACH'))

Category. E, what a number means. **Round.** v2. **Label.** 2,480. **I.i.d., all five draws.** 8,950. **Diverse.** draw 1 / 1.

Original problem. A lawn mowing service provides neighborhood services using small teams and large teams. **A small team requires 3 employees and can mow 50 sq ft of lawn. A large team requires 5 employees and can mow 80 sq ft of lawn.** The company has 150 employees available. Because most people have smaller lawns in the city, the number of small teams must be at least 3 times as much as the number of large teams. In addition, to make sure the company can meet all demands, there has to be at least 6 large teams and at least 10 small teams. How many of each team type should be used to maximize the amount of lawn that can be mowed?

Edited problem. A lawn mowing service provides neighborhood services using small teams and large teams. **Small teams have 3 employees that can mow 50 sq ft of lawn each. Large teams have 5 employees that can mow 80 sq ft of lawn each.** The company has 150 employees available. Because most people have smaller lawns in the city, the number of small teams must be at least 3 times as much as the number of large teams. In addition, to make sure the company can meet all demands, there has to be at least 6 large teams and at least 10 small teams. How many of each team type should be used to maximize the amount of lawn that can be mowed?

Why it is ambiguous. 'each' can distribute over the teams or over the employees it follows.

Readings. Label's reading: area per team (optimum 2,480). Alternative: area per employee (optimum 8,950).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int small_teams : [0, 50]
    Int large_teams : [0, 50]

Constraints:
    # Employee constraint: small teams use 3 employees, large teams use
    # 5
    3 * small_teams + 5 * large_teams <= 150
    # Small teams must be at least 3 times the number of large teams
    small_teams >= 3 * large_teams
    # Minimum team requirements
    large_teams >= 6
    small_teams >= 10
```

```
Maximize:
    # Small teams mow 3*50=150 sq ft total, large teams mow 5*80=400 sq
    # ft total
    150 * small_teams + 400 * large_teams
```

Alternative reading, as written:
`Maximize: 150 * small_teams + 400 * large_teams`
Label's reading:
`Maximize: 50 * small_teams + 80 * large_teams`

VARIANT 11: NL4OPT-121 (MARGIN VS SALE PRICE (NET OF STATED UNIT COST))

Category. E, what a number means. **Round.** v3. **Label.** 507.8. **I.i.d., all five draws.** 274.8. **Diverse.** draw — / —.

Original problem. Each month a store owner can spend at most $500 on carrots and cucumbers. A carrot costs the store owner $0.30 and a cucumber costs the store owner $0.50. **Each carrot is sold for a profit of $0.75 and each cucumber is sold for a profit of $0.80.** The owner estimates that the number of cucumbers sold is at most a third of the number of carrots sold. He also estimates that at least 300 carrots but at most 500 carrots are sold each month. How many of each, carrots and cucumbers, should be sold in order to maximize the profit?

Edited problem. Each month a store owner can spend at most $500 on carrots and cucumbers. A carrot costs the store owner $0.30 and a cucumber costs the store owner $0.50. **Each carrot brings in $0.75 and each cucumber brings in $0.80.** The owner estimates that the number of cucumbers sold is at most a third of the number of carrots sold. He also estimates that at least 300 carrots but at most 500 carrots are sold each month. How many of each, carrots and cucumbers, should be sold in order to maximize the profit?

Why it is ambiguous. Right after the unit costs, 'brings in' reads as the sale price the cost must come out of.

Readings. Label's reading: $0.75/$0.80 is profit (optimum 507.8). Alternative: $0.75/$0.80 is sale price, minus cost (optimum 274.8).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int carrots : [300, 500]
    Int cucumbers : [0, 166]

Constraints:
    0.30 * carrots + 0.50 * cucumbers <= 500
    cucumbers <= carrots / 3
    carrots >= 300
    carrots <= 500

Maximize:
    (0.75 - 0.30) * carrots + (0.80 - 0.50) * cucumbers
```

Alternative reading, as written:
`Maximize: (0.75 - 0.30) * carrots + (0.80 - 0.50) * cucumbers`
Label's reading:
`Maximize: 0.75 * carrots + 0.80 * cucumbers`

VARIANT 12: NL4OPT-169 (SHARE OF WHAT (VEHICLES VS PASSENGERS))

Category. E, what a number means. **Round.** v3. **Label.** 29. **I.i.d., all five draws.** 44. **Diverse.** draw 0 / 1.

Original problem. A golf course is hosting an event and can transport guests using either golf carts or pull carts. A golf cart can take 4 guests while a pull cart can take 1 guest. **Since golf carts take**

up a lot of space, at most 60% of carts can be golf carts. If the golf course needs to transport at least 80 guests, how many of each cart should be used to minimize the total number of carts needed?

Edited problem. A golf course is hosting an event and can transport guests using either golf carts or pull carts. A golf cart can take 4 guests while a pull cart can take 1 guest. **Since golf carts take up a lot of space, at most 60% of the transport can be by golf cart.** If the golf course needs to transport at least 80 guests, how many of each cart should be used to minimize the total number of carts needed?

Why it is ambiguous. 'the transport' can be counted in carts dispatched or in guests carried.

Readings. Label's reading: 60% of the carts (optimum 29). Alternative: 60% of the guests (optimum 44).

Wrong encoding (i.i.d. draw 0):

```
# g = number of golf carts (capacity 4 each)
# p = number of pull carts (capacity 1 each)
# "at most 60% of transport by golf cart" means guests via golf cart <=
# 60% of total guests

Variables:
    Int g : [0, 80]
    Int p : [0, 80]

Constraints:
    # Transport at least 80 guests
    4 * g + p >= 80
    # Golf cart guests <= 60% of total guests transported
    # 4*g <= 0.6*(4*g + p)  =>  1.6*g <= 0.6*p  =>  8*g <= 3*p
    8 * g <= 3 * p

Minimize:
    g + p
```

Alternative reading, as written:

```
8 * g <= 3 * p
```

Label's reading:

```
g <= 0.6 * (g + p)
```

VARIANT 13: NL4OPT-223 (GROSS VS NET (REVENUE OR PROFIT))

Category. E, what a number means. **Round.** v2. **Label.** 890. **I.i.d., all five draws.** 190. **Diverse.** draw 3 / 2.

Original problem. A toy store sells plush toys and dolls. Each plush toy costs the store $3 and each doll costs the store $2. The store owner can spend at most $700 on inventory. **Each plush toy is then sold for a profit of $4 while each doll is sold for a profit of $2.** The owner estimates that at least 90 plush toys but at most 190 plush toys are sold each month. He also estimates that the number of dolls sold is at most twice the amount of plush toys sold. How many of each should be bought and sold to maximize profit?

Edited problem. A toy store sells plush toys and dolls. Each plush toy costs the store $3 and each doll costs the store $2. The store owner can spend at most $700 on inventory. **Each plush toy then brings in $4 while each doll brings in $2.** The owner estimates that at least 90 plush toys but at most 190 plush toys are sold each month. He also estimates that the number of dolls sold is at most twice the amount of plush toys sold. How many of each should be bought and sold to maximize profit?

Why it is ambiguous. 'brings in' names an amount without saying whether the $3/$2 cost is already taken out.

Readings. Label's reading: \$4/\$2 is profit (optimum 890). Alternative: \$4/\$2 is sale price, minus cost (optimum 190).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int plush : [90, 190]
    Int dolls : [0, 380]

Constraints:
    3 * plush + 2 * dolls <= 700
    dolls <= 2 * plush

Maximize:
    (4 - 3) * plush + (2 - 2) * dolls
```

Alternative reading, as written:
`Maximize: (4 - 3) * plush + (2 - 2) * dolls`
Label's reading:
`Maximize: 4 * plush + 2 * dolls`

E.5 F. What the model includes

Variant 14: NL4Opt-52 (cap attached to the last-mentioned item vs the whole list)

Category. F, what the model includes. **Round.** v3. **Label.** 4,190. **I.i.d., all five draws.** 6,300. **Diverse.** draw 1 / 2.

Original problem. A snack exporter sends snacks to his customer in small and large suitcases. A small suitcase can hold 50 snacks while a large suitcase can hold 80 snacks. Most customer prefer small suitcases, and so at least twice as many small suitcases must be used as large suitcases. The exporter has available at most 70 small suitcases and 50 large suitcases. If he must send at least 15 large suitcases and **can send at most 70 suitcases in total**, how many of each should he send to maximize the total number of snacks that can be delivered?

Edited problem. A snack exporter sends snacks to his customer in small and large suitcases. A small suitcase can hold 50 snacks while a large suitcase can hold 80 snacks. Most customer prefer small suitcases, and so at least twice as many small suitcases must be used as large suitcases. The exporter has available at most 70 small suitcases and 50 large suitcases. If he must send at least 15 large suitcases and **can send at most 70 of them**, how many of each should he send to maximize the total number of snacks that can be delivered?

Why it is ambiguous. 'them' can refer back to the large suitcases just mentioned rather than all suitcases.

Readings. Label's reading: at most 70 suitcases in total (optimum 4,190). Alternative: at most 70 large suitcases (optimum 6,300).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int s : [0, 70]    # number of small suitcases
    Int l : [0, 70]    # number of large suitcases

Constraints:
    s >= 2 * l         # at least twice as many small as large
    s <= 70            # at most 70 small suitcases available
    l <= 50            # at most 50 large suitcases available
    l >= 15            # must send at least 15 large suitcases
    l <= 70            # can send at most 70 large suitcases
```

```
Maximize:
    50 * s + 80 * l
```

Alternative reading, as written:
`l <= 70`
Label's reading:
`s + l <= 70`

VARIANT 15: NL4OPT-142 (LOWER BOUND ON EACH TYPE VS ON THE COMBINATION)

Category. F, what the model includes. **Round.** v2. **Label.** 480. **I.i.d., all five draws.** 525. **Diverse.** draw 1 / 1.

Original problem. An art store makes large and small art pieces. The store has available 100 units of paint, 50 units of glitter, and 70 units of glue. To make a large art piece requires 4 units of paint, 3 units of glitter, and 5 units of glue. To make a small art piece requires 2 units of paint, 1 unit of glitter, and 2 units of glue. The store must make **at least 5 units of each large and small art pieces**. If the profit per large art piece is $30 and the profit per small art piece is $15, how many of each should be made to maximize profit?

Edited problem. An art store makes large and small art pieces. The store has available 100 units of paint, 50 units of glitter, and 70 units of glue. To make a large art piece requires 4 units of paint, 3 units of glitter, and 5 units of glue. To make a small art piece requires 2 units of paint, 1 unit of glitter, and 2 units of glue. The store must make **at least 5 units of large and small art pieces**. If the profit per large art piece is $30 and the profit per small art piece is $15, how many of each should be made to maximize profit?

Why it is ambiguous. Without 'each', 'at least 5 units of large and small art pieces' reads as a joint count.

Readings. Label's reading: at least 5 of each type (optimum 480). Alternative: at least 5 in total (optimum 525).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int large : [0, 1000]
    Int small : [0, 1000]

Constraints:
    4 * large + 2 * small <= 100
    3 * large + 1 * small <= 50
    5 * large + 2 * small <= 70
    large + small >= 5

Maximize:
    30 * large + 15 * small
```

Alternative reading, as written:
`large + small >= 5`
Label's reading:
`large >= 5`
`small >= 5`

VARIANT 16: NL4OPT-231 (UPPER CAP OVER THE TOTAL VS PER ITEM)

Category. F, what the model includes. **Round.** v2. **Label.** 256,500. **I.i.d., all five draws.** 275,700. **Diverse.** draw 1 / 1.

Original problem. Platinum Database sells two types of subscription software packages: a personal license and a commercial license which will cost $550 and $2000 to generate respectively. The

marketing department estimates that they can sell **at most 300 licenses for both versions combined a month**. The profit per personal license is $450 and the profit per commercial version is $1200. If the company does not want to spend more than $400000, how many of each software package should they produce to maximize the profits.

Edited problem. Platinum Database sells two types of subscription software packages: a personal license and a commercial license which will cost $550 and $2000 to generate respectively. The marketing department estimates that they can sell **at most 300 licenses for either version a month**. The profit per personal license is $450 and the profit per commercial version is $1200. If the company does not want to spend more than $400000, how many of each software package should they produce to maximize the profits.

Why it is ambiguous. 'for either version' can mean licenses of any version together or a cap on whichever version.

Readings. Label's reading: at most 300 licenses in total (optimum 256,500). Alternative: at most 300 licenses per version (optimum 275,700).

Wrong encoding (i.i.d. draw 0):

```
Variables:
    Int personal : [0, 300]
    Int commercial : [0, 300]

Constraints:
    # Total licenses sold constraint (at most 300 for either version)
    personal <= 300
    commercial <= 300
    # Budget constraint: cost to generate must not exceed $400,000
    550 * personal + 2000 * commercial <= 400000

Maximize:
    450 * personal + 1200 * commercial
```

Alternative reading, as written:

```
personal <= 300
commercial <= 300
```

Label's reading:

```
personal + commercial <= 300
```