ASTOR: BENCHMARKING AND IMPROVING AUTOFORMULATION FOR SOFT-CONSTRAINT OPTIMIZATION WITH A FORMAL GRAMMAR

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world decision problems combine two qualitatively different kinds of constraints: requirements that must be satisfied, called hard constraints, and preferences that can be violated, called soft constraints. Because these problems are often posed to decision-makers in natural language, large language models offer a natural tool for assisting with their solution. This has given rise to autoformulation, which focuses on using LLMs to translate natural-language descriptions of decision problems into executable mathematical optimization models. However, this research often neglects soft constraints like those proposed by consumers, managers, and planners, despite the fact that many practical problems require modeling them explicitly. We introduce **ASTOR**, an inference-time framework that maps a natural-language decision problem to a grammar-guided abstract syntax tree (AST) before deterministically compiling the resulting representation into executable optimization code. We also introduce **SoftOR**, a unified benchmark of natural language problems which focuses on soft constraint formulations. On SoftOR, ASTOR achieves the best aggregate accuracy among comparable single-request and multi-agent LLM methods while remaining competitive on standard autoformulation benchmarks that include only hard constraints. Code and data are available here.

1 INTRODUCTION

In practical settings, decision-makers often have to balance preferences from many stakeholders, some of which are not always satisfiable. These preferences do not constrain their decisions the same way that a strict requirement might. For example, a hospital manager must assign nurses to cover every shift, but may accommodate preferences to avoid night shifts or maintain consecutive days off. A travel agent must choose hotels and flights that obey the requested destination, but may not always be able to accommodate brand preferences. A shopping agent must obey the consumer's budget, but may choose to suggest a longer delivery date than the one desired. An ideal optimization model must therefore distinguish what **must** hold from what is **preferred** to hold, and when preferences conflict, determine which take priority.

Natural language is a familiar way to express these distinctions, and the growing use of large language models (LLMs) makes it an increasingly common interface for decision-making. Consumers use AI tools for shopping and planning (Associated Press, 2025; Shopify, 2026; McKinsey & Company, 2026), while firms are incorporating AI for investments or as retail assistants (The Wall Street Journal, 2026; Reuters, 2026). These settings often present decision-makers with hard and soft constraints. As a result, LLM autoformulation provides a promising approach for this translation problem. Research in this field often uses LLM prompting or fine-tuning methods to successfully translate natural-language descriptions of real decision problems into solvable mathematical optimization formulations. However, their methods often prioritize only hard formulations (Wei et al., 2022; Deng et al., 2024; Yang et al., 2026) and current benchmarks only test those formulations (Ramamonjison et al., 2022; AhmadiTeshnizi et al., 2024; Xiao et al., 2024; Huang et al., 2025a). They do not directly test whether a model distinguishes requirements from preferences or preserves the ordering among preferences. This leaves an important class of practical decision problems without a strong autoformulation method or evaluation tools.

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.

We address this gap with ASTOR and SoftOR. **ASTOR** is an inference-time method which prompts an LLM to express a decision problem as an abstract syntax tree generated by a formal grammar, then validates and compiles that representation deterministically into solver code. The constructed grammar is adequate for the class of hard and soft constraint optimization problems considered in this work; see Appendix C. **SoftOR** is our newly introduced **benchmark** for evaluating whether models can recover weighted-penalty and lexicographic soft-constraint formulations from natural language. It consists of tasks involving travel planning and scheduling nurses for shifts in a hospital.

Across the 355 SoftOR instances, ASTOR achieves 88.5% solution accuracy, compared with 84.8% for the strongest alternative. It achieves the highest accuracy on three of the four SoftOR datasets, with the nurse scheduling dataset as the only exception. On 435 conventional hard-constraint problems, ASTOR achieves 81.8%, within 0.5 percentage points of the best-performing method.

2 Related Work

Autoformulation. Autoformulation methods differ in both their output representation and their use of inference-time computation. Under a single-request budget, Direct prompting returns a final solution, Codegen produces an executable solver program, and Chain-of-Thought prompting generates a natural-language derivation before producing the program (Deng et al., 2024; Wei et al., 2022; Yang et al., 2026).

Other systems improve autoformulation by adding training supervision or iterative inference. ORLM (Huang et al., 2025a) fine-tunes models on generated optimization instances, while LLMOPT (Jiang et al., 2025) trains models to produce structured formulations before code generation. Chain-of-Experts, OptiMUS, and ORThought instead distribute interpretation, formulation, implementation, and correction across multiple agents or execution-conditioned stages (Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Yang et al., 2026). These methods demonstrate the value of additional supervision and refinement. Some methods use intermediate representations to organize formulation or code generation (Jiang et al., 2025; AhmadiTeshnizi et al., 2024), but these representations are often not tailored to the task of optimization, nor do they admit any formal coverage guarantees. ASTOR uses the inference budget of the single-request methods and a bespoke formal representation designed for optimization formulations to produce more accurate models.

Benchmarking. Autoformulation benchmarks evaluate whether a system can translate a natural-language problem into a solvable mathematical program. Early benchmarks focused on recovering standard linear programs from relatively structured descriptions, as in NL4Opt (Ramamonjison et al., 2022). Later work expanded the task to longer descriptions, mixed-integer programs, executable solver code, implicit constraints, and industrial problems (AhmadiTeshnizi et al., 2024; Huang et al., 2025b; Xiao et al., 2024; Huang et al., 2025a; Yang et al., 2026). Together, these benchmarks have moved evaluation toward more realistic and structurally diverse formulation tasks (Xiao et al., 2025).

Evaluation methods differ by benchmark, but the most common kind is called *solution accuracy* (Jiang et al., 2025; Yang et al., 2026; AhmadiTeshnizi et al., 2024; Xiao et al., 2024). In this method, each natural language instance is labeled a priori with a gold-standard optimal *value* that the authors consider correct. This optimal value can be a number if there is a single objective function, or a vector if there are multiple. To score an LLM autoformulation attempt, evaluators can take the optimization model produced by an LLM for a single natural language instance, perform the optimization using a solver like Gurobi, and compare the optimal value produced by the LLM's optimization formulation to the gold-standard value. The LLM's attempt is scored as correct if and only if the optimal value produced by its formulation exactly matches the gold-standard value. Other methods use an LLM and graph methods like embedding distance or edit distance to score the optimization formulation itself, instead of just the optimal value Liu et al. (2026); Zhang et al. (2026); Abdul Rahman et al. (2026); Li et al. (2025); Wang et al. (2025). This paper uses solution accuracy as the primary evaluation metric, as well as some measures of formulation success.

Related benchmarks such as TravelPlanner, COMPASS, and SchedulingBenchmarks Nurse Rostering contain requirements and preferences that may conflict (Xie et al., 2024; Qin et al., 2026; Curtois, 2014). However, these are not well prepared for autoformulation benchmarking, because none jointly provides complete natural language instances with gold-standard labeled optimal values. Additionally, existing autoformulation benchmarks focus primarily on feasibility constraints and scalar objectives. SoftOR is the first of its kind to directly target soft constraint formulations.

Formal Language in LLM Inference Formal languages have proven to be a useful intermediate step in other LLM translation tasks. Researchers have defined bespoke formal languages to successfully aid translation from natural language into formal logic or code, often representing the formulas produced in those languages through tree-graphs called Abstract Syntax Trees (ASTs) (Aho et al., 2007; Alur et al., 2013; Padhi et al., 2023). For example, NL2LOGIC uses a Backus–Naur Form grammar and AST representation to generate logical expressions that are translated into Boolean-solver code, reporting up to 99% syntactic faithfulness (Putra et al., 2026). AST-T5 and AST-based NL-to-SQL methods likewise use syntactic structure to support code generation (Gong et al., 2024; Liao et al., 2026), while AST-Probe finds recoverable code structure in language-model representations (Hernández López et al., 2022).

Furthermore, the formal languages themselves often admit provable guarantees which are not accessible to other inference methods, and their computational representations as ASTs are useful for downstream formal evaluation such as graph embedding and isomorphism (Li et al., 2025; Wang et al., 2025). These results motivate ASTs as an interface between language models and executable formal systems, especially NL2LOGIC, which even uses ASTs for LLM translation and deterministic solving. ASTOR applies that interface to optimization and newly extends it to represent hard constraints, weighted penalties, and lexicographic priorities.

3 Methodology

3.1 Formalizing Soft Preferences

Consider a user who specifies a collection of requirements and preferences over decisions x. We represent each requirement as a hard constraint $h_j(x) \le 0$, which every feasible solution must satisfy, and each preference i with a nonnegative violation function $v_i(x)$, where $v_i(x) = 0$ indicates satisfaction and larger values indicate greater violation. The problem may also include a base objective $f_0(x)$, such as monetary cost or total working time.

We consider two common formulations of these soft constraints:

$$\text{lexicographic:} \quad \operatorname*{lex\,min}_x \big(V_1(x), \ldots, V_K(x), f_0(x)\big), \quad V_k(x) = \sum_{i:\mathrm{Priority}(i)=k} v_i(x),$$

$$\text{weighted penalty:} \quad \min_x \; f_0(x) + \sum_i w_i v_i(x), \qquad w_i \ge 0, \qquad \text{s.t. } h_j(x) \le 0.$$

In the lexicographic formulation, each soft constraint is assigned to one of K priority tiers by $\mathrm{Priority} : \mathbb{N} \to \mathbb{N}$, with lower indices indicating higher priority. These are indices that denote priority order in the mathematical formulation, operationally our method uses larger numbers to denote higher ranks, which is equivalent. The aggregate violation $V_k(x)$ sums the violations in tier k. The objective then minimizes these tier violations in priority order, so an improvement at a lower-priority tier cannot compensate for a loss at a higher-priority tier.

In the weighted-penalty formulation, each violation v_i instead receives a nonnegative weight w_i. The weighted violations are added directly to the base objective, allowing preferences to trade off against one another and against the original objective (Golden et al., 2024; Bärmann et al., 2025). Both formulations are common in goal-programming applications such as scheduling and planning.

3.2 Formal Grammar

The formal grammar G defines the intermediate language at the center of our method. G provides compositional rules to assemble variables, domains, numbers, expressions, formulas, boolean connectives, aggregations, and explicit priority levels into formulas that represent optimization models. This grammar is given to our method as its output language, which guides it to create well-formed formulas in a language we know to be well-suited for optimization with soft constraints. Other methods, which output in natural language or solver-code, do not provide a bespoke formal representation designed for a specific optimization task. Our method will do this through the use of a formal grammar.

However, the grammar used by our method does achieve one such guarantee. In particular, we use a notion of *adequacy*, which is a standard notion in formal logic for establishing that a formal language

can represent its intended class of structures (Cheney et al., 2012; Otto, 2013). In our case, G is *adequate* for the class of optimization problems considered in this work when the correspondence holds in both directions: every well-formed formula generated by G denotes a problem in the class, and every problem in the class can be represented by a well-formed formula generated by G. Thus, the grammar admits exactly the target problems without admitting any outside formulas (Cheney et al., 2012; Otto, 2013). Appendix C gives the complete grammar, formal definition, and proof.

3.3 ASTOR Model Architecture

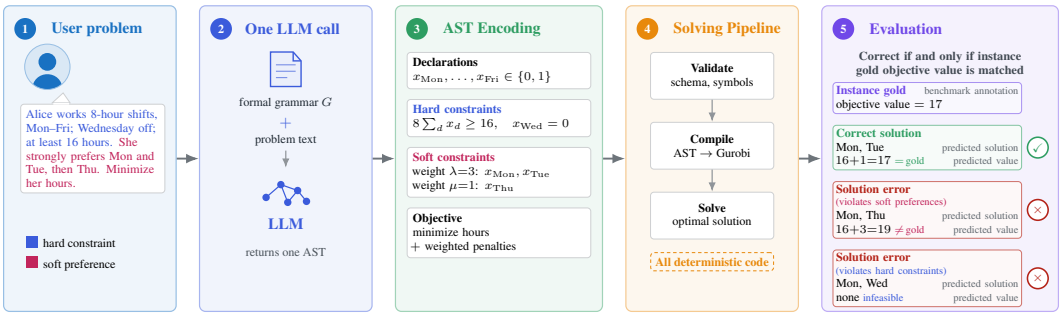


Figure 1: ASTOR: One LLM call forms an AST which is compiled and solved deterministically.

Our method prompts an LLM with a natural-language problem and the grammar G to produce a well-formed expression in the language generated by G, which in this case is an optimization problem that may have hard or soft constraints. Every such expression admits a corresponding Abstract Syntax Tree (AST), a tree-structured representation that makes the compositional relationships between optimization components explicit. This gives our method a constrained intermediate representation between natural language and executable optimization code. AST-based intermediate representations have been used in natural-to-formal language translation to reduce generation errors and enforce compositional consistency (Putra et al., 2026; Gong et al., 2024; Liao et al., 2026; Hernández López et al., 2022). We adapt this approach to mathematical optimization, and develop a typed grammar, which is more expressive than some first-order-logic grammars developed in previous AST papers. For this reason, we name our method **ASTOR**: *Abstract Syntax Trees for Operations Research.*

The specific implementation is in Figure 1. Concretely, the model is given a prompt including a brief description of the task, the formal grammar, and a JSON schema which serializes the AST and is the expected output form. The AST-JSON contains variable declarations, hard constraints, soft constraints, and priority annotations. A deterministic post-processing pipeline applies only local JSON repairs, validates and parses the resulting tree, and compiles it into executable Gurobi code. Thus, the LLM is responsible for formulating the optimization problem, while all subsequent translation from the formal representation to solver code is deterministic. The resulting solver output is evaluated against the gold optimum. Worked examples and execution traces are provided in Appendix B, with complete prompts and execution rules in Appendix D.

4 Building a New Benchmark: SoftOR

Existing LLM autoformulation benchmarks primarily evaluate feasibility and objective recovery for problems expressed through hard constraints. Existing datasets contain preferences, but do not jointly provide natural-language instances, verified gold optima, and annotations for both weighted-penalty and lexicographic objectives. We therefore introduce SoftOR (Soft Operations Research), a unified natural-language soft-constraint autoformulation benchmark covering both objective structures. To construct and annotate lexicographic instances, we draw on the theory of **modality** from linguistics, which studies expressions such as *must*, *should*, and *may* that describe necessity, obligation, and desire (Henning et al., 2022; Garson, 2023). These provide an immediate natural-language interface for soft constraints, and linguistic theory already motivates an ordering on modal words by strength that can be used to define lexicographic levels.

SoftOR comprises Nurse Rostering, COMPASS, TravelPlanner, and LexiBench. For the first three components, we use published datasets as basic source material, but construct the natural-language instances, soft-constraint semantics, priority annotations, and evaluation rules required for autoformu-

lation. LexiBench is entirely original. Table 1 summarizes the resulting components and identifies the source data used in their construction.

Table 1: SoftOR components with new natural language problems and evaluations.

Component	n	Objective	Priority Signal	Gold construction	Source Material
Nurse Rostering	15	Weighted penalty	Explicit	Published optimum	Curtois (2014); Curtois & Qu (2014)
COMPASS	85	Lexicographic	Explicit	Enumeration	Qin et al. (2026)
TravelPlanner	180	Lexicographic	Modal	Enumeration	Xie et al. (2024)
LexiBench	75	Lexicographic	Mixed	Enumeration/Solver Code	This work

Full benchmark construction details, including natural language cues, are included in Appendix F.

4.1 Constructing New SoftOR Autoformulation Instances from Source Data

The source data from Nurse Rostering, COMPASS, and TravelPlanner were designed for scheduling or planning evaluation rather than natural-language optimization autoformulation and were therefore not directly usable as SoftOR instances without our adaptations as described below.

For Nurse Rostering, we convert published tabular requirements into new natural-language instances with explicit priority markers. Each published request weight becomes a fixed phrase. In particular, the original priority markers, annotated as 1, 2, 3, and 100, become “slightly prefers," “prefers," “strongly prefers" and “far more serious." The original numbers are not kept in the final natural language instance, since we intend to test whether these ordinal priorities can be recovered from language alone. Optima are provided by the original benchmark authors (Curtois, 2014). Numerical weights are not given to any method. Each method instantiates its own weights, if it chooses to use them, and the specific numerical weights given by the benchmark are used only post-modeling for scoring and evaluation. We chose not to show weights to any route during decision-making so we can evaluate each route’s ability to infer preference strength from natural language. As an ablation, we run a case where routes can see weights before decisions in Appendix F.

The original COMPASS source material provided a natural-language query, structured hard constraints and preferences, a finite set of candidate plans, and one annotated gold plan ID (Qin et al., 2026). It did not present these elements as a single autoformulation instance or annotate the underlying optimization objective. To adapt these instances for SoftOR, we combine the query and structured fields into a complete natural-language problem using fixed templates that mark requirements as "required" and wishes as equally important soft constraints that are "preferred". We also reconstruct the gold objective. Because multiple plans may satisfy the same maximum number of preferences, we enumerate every provided plan and use that maximum as gold rather than requiring the single plan ID supplied originally by COMPASS.

The original TravelPlanner source already provided natural-language travel-planning problems and structured candidate data, but it was designed to evaluate plan generation and does not supply annotated optimization formulations or gold objective vectors (Xie et al., 2024). For our adaptation, we convert each instance into a lexicographic autoformulation task by assigning its linguistic cues to three priority tiers. These cues, called **modals**, express obligation, intention, or desire, and the tiers follow the deontic–teleological–bouletic distinction in modal linguistics (Rubinstein et al., 2013; von Fintel & Iatridou, 2008; Portner, 2009; Henning et al., 2022): deontic language states what must hold, teleological language what the speaker plans to achieve, and bouletic language what the speaker wants. We then enumerate every valid combination of transport, one hotel per city, and restaurants; compare plans lexicographically by tier violations; and break ties by total cost. The resulting optimal vector is the gold annotation, and a formulation is correct only if its solution matches that vector at every tier and in cost. The complete inventory of linguistic cues and benchmark construction details is included in Appendix F.

4.2 Original Construction of LexiBench

LexiBench is an original dataset of 75 problems that tests richer ordered preferences than the preceding sources, which contain either fixed weights or at most three tiers. Figure 2 shows its five preference structures. Each structure is instantiated for three decision types. *Atomic* problems select one element from a candidate set, as in COMPASS. *Assembly* problems build a bundle from several candidate sets, as in TravelPlanner. *Constructive* problems formulate constraints and optimize over

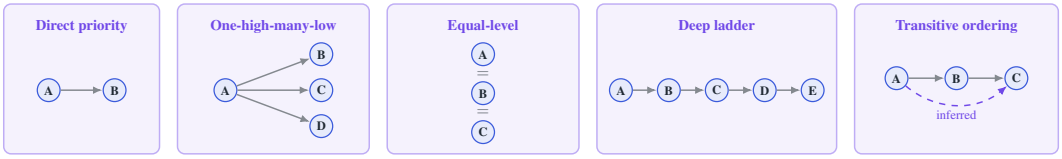


Figure 2: Priority annotation in LexiBench instances.

an implicitly defined feasible region, as in conventional hard-constraint benchmarks. Hardness is signaled with the same modal cues as TravelPlanner, and when more than three levels are needed, explicit comparatives such as "more important than," "higher priority," and "above all" fix the remaining order. Gold is computed by enumeration for atomic and assembly problems and by solving the annotated lexicographic model in Gurobi for constructive problems. A formulation is correct if it attains the gold objective vector, regardless of which optimal solution it returns.

The *priority signal* column of Table 1 records how each source communicates priority, and Appendix A disaggregates accuracy by it. *Explicit* sources state priority in words that name a level directly, such as "strongly prefer" and "slightly prefer" in Nurse Rostering or "equally important" in COMPASS. *Modal* sources encode priority only through the modal verbs above, so the tier must be inferred from the strength of the wording. *Mixed* sources, namely LexiBench, combine modal cues with explicit comparatives and pairwise statements, so recovering the ordering requires both readings. Appendix F gives the complete construction and annotation procedures for SoftOR.

5 Experiments

To evaluate ASTOR, we compare it with single-request prompting methods and previously published agentic autoformulation methods across hard and soft constraint benchmarks. Our experiments address the following research questions:

- RQ1: How accurately do single-request and multi-agent inference methods solve soft-constraint benchmarks? How does this performance differ by lexicographic and weighted penalty objectives?
- RQ2: How does accuracy vary by priority signal and problem difficulty?
- RQ3: What technical and solution errors do the methods produce?
- RQ4: How expensive is ASTOR compared to other methods?
- RQ5: How do the ASTOR components, temperature, and backbone affect performance?

5.1 Experimental Setup

For our main experiment, we evaluate on SoftOR using temperature-zero Gemini 3.7 Flash as the LLM backbone with the provider-default thinking configuration on most benchmarks. The only exception is Nurse Rostering, where all routes receive an increased thinking budget and use Gemini 3.8 Flash, due to its more complex formulations. Appendix A includes additional experiments on standard autoformulation benchmarks distributed by Yang et al. (2026), but these include almost exclusively hard constraints. A non-reasoning model, a non-Gemini reasoning model, and a larger Gemini reasoning model are included in later ablation studies on SoftOR and standard benchmarks.

Solution Accuracy, as defined in Section 2, is the primary evaluation metric, and other metrics are reported in Table 3 for more granularity on which soft constraints are most often satisfied. Since lexicographic problems have multiple objectives, gold-standard labels are vectors, and LLM formulations are considered correct if and only if the entire lexicographic objective vector produced exactly matches the gold-standard one. Weighted-penalty problems have a single objective, so the gold standard label is a number, and similarly, formulations are correct if and only if the number produced is an exact match to the gold-standard.

We compare against three single-request routes using the same backbone model. Direct is given the natural language problem and predicts the final answer with no intermediate representation. Code-generation (Codegen) is a single-request LLM prompting method which is given the natural language problem and outputs executable Gurobi code to solve the problem (Deng et al., 2024). This is the **main control** of the experiment, and tests whether a formal grammar provides any advantage over another formal intermediate representation, like solver code. Chain-of-Thought (CoT) asks the LLM to think in steps to produce a mathematical optimization model and executable Gurobi code

(Wei et al., 2022). ORThought uses agents to formulate the optimization problem as executable code and at most three repair agents to debug the formulation or code (Yang et al., 2026). Chain-of-Experts (CoE) uses a panel of many LLM agents to collaborate on formulation, coding, and natural-language extraction (Xiao et al., 2024). ORThought and CoE use published prompts.

Direct, Codegen, and CoT are single-request LLM prompting methods, whereas CoE and ORThought are multi-request pipelines that use different LLM prompts for formulation, repair, and execution. We estimate sampling uncertainty with 10,000 paired bootstrap resamples over problem instances, using the same sampled instances for every method. Tables report bootstrap standard errors in parentheses, and figures show percentile 95% confidence intervals. Appendix D gives the prompts.

5.2 RQ1: Accuracy

Overall, ASTOR is the most accurate on SoftOR, with an aggregate accuracy across lexicographic and weighted penalty problems of 88.5%. From most to least accurate, the baselines achieve 84.8% for CoE, 83.7% for Codegen, 81.4% for CoT, 80.0% for ORThought, and 52.7% for Direct. We now report results stratified by objective type (lexicographic or weighted), including more granular data on soft constraint satisfaction.

5.2.1 Lexicographic Problems

Table 2: Solution accuracy by dataset for six routes (%), bootstrap standard errors are in parentheses. The best result is bolded.

Method	COMPASS	Travel Planner	LexiBench	Overall
Direct	56.5 (5.4)	75.0 (3.2)	5.3 (2.6)	55.0 (2.7)
Codegen	**98.8 (1.2)**	76.1 (3.2)	89.3 (3.5)	84.7 (1.9)
CoT	95.3 (2.3)	77.2 (3.1)	80.0 (4.6)	82.4 (2.1)
ORThought	96.5 (2.0)	80.0 (3.0)	66.7 (5.4)	81.2 (2.1)
CoE	**98.8 (1.2)**	79.4 (3.0)	89.3 (3.5)	86.5 (1.8)
ASTOR (Ours)	**98.8 (1.2)**	**85.0 (2.7)**	**92.0 (3.2)**	**90.0 (1.6)**

In Table 2, we compare the six routes on solution accuracy over each component of SoftOR. Table 7 in Appendix A reports results for hard constraints.

Table 3: Average soft constraint satisfaction (%) over all instances.

	COMPASS	TravelPlanner			LexiBench				
Method	Preferences	Tier 2	Tier 3	All	Tier 1	Tier 2	Tiers 3–5	All	Order Kept
Direct	35.0	93.8	86.2	87.4	4.5	3.3	1.5	3.5	–
Codegen	**62.7**	93.8	83.9	85.3	92.8	52.1	65.2	65.6	96.0
CoT	60.4	81.2	85.1	85.3	90.1	51.8	60.6	63.8	97.3
ORThought	60.7	93.8	83.9	85.3	73.1	38.6	50.0	52.0	97.3
CoE	**62.7**	**100.0**	87.4	88.4	**94.1**	50.4	**68.2**	**66.3**	96.0
ASTOR (Ours)	**62.7**	**100.0**	**92.0**	**92.6**	92.0	**53.2**	63.6	63.5	**98.7**

In Table 3, all columns except Order Kept report proportion of preferences satisfied on each lexicographic level. LexiBench tiers are grouped for brevity and data sparsity. Order Kept denotes the proportion of attempts that never invert priority levels. ASTOR performs best on COMPASS, TravelPlanner, and some LexiBench metrics. Hard constraint satisfaction is reported in Appendix A.

5.2.2 Weighted Penalty Problems (Nurse Rostering Instances)

Table 4 shows the performance of each method on each instance of Nurse Rostering. Numbers indicate incurred surplus penalty by violating staffing requirements or shift preferences. Zero penalty is an optimal solution. FAIL denotes instances where the route failed to obey all hard constraints or

Table 4: Performance relative to gold-standard optimum on the 15 Nurse Rostering instances.

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Direct	+407	+897	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
Codegen	0	0	0	+1	+1	0	0	+1	+2	0	0	+1	0	+1	0
CoT	0	0	0	+1	+1	0	0	+6	+2	0	0	+1	0	+1	0
ORThought	0	0	0	+1	+3	FAIL	0	+1	+2	0	0	+1	0	FAIL	0
CoE	0	0	0	+1	+1	0	+1	+2	+2	0	0	+1	+2	+1	0
ASTOR	0	0	0	+1	+1	+1	0	+2	+2	0	0	+1	0	+4	0

failed to produce a roster at all. Codegen and CoT are the most accurate on this dataset, with 9/15 correct instances (0 penalty). ASTOR and ORThought have 8/15, CoE has 7/15, and Direct has 0/15. ASTOR is as good as all other inference methods on staffing measures. It is neither the worst nor the best on weighted shift preferences. Details on Nurse Rostering are available in Appendix F.

5.3 RQ2: Performance by Problem Characteristics

We disaggregate results by difficulty level, with token usage as a proxy. Figure 4 has all 355 SoftOR instances as its horizontal axis, ordered from least to most token usage on the Direct route. Each tally mark indicates a correct answer for that instance. The section on the right is for those instances Direct did not produce a roster. Figure 3a does the same, but aggregates token usage by quintile. ASTOR performs best, even as the difficulty level of the problem increases. More disaggregated results are in Appendix A (Figure 6).

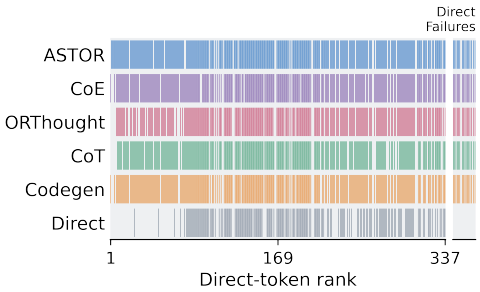


Figure 4: Correct instances ordered by Direct token usage as a proxy for difficulty

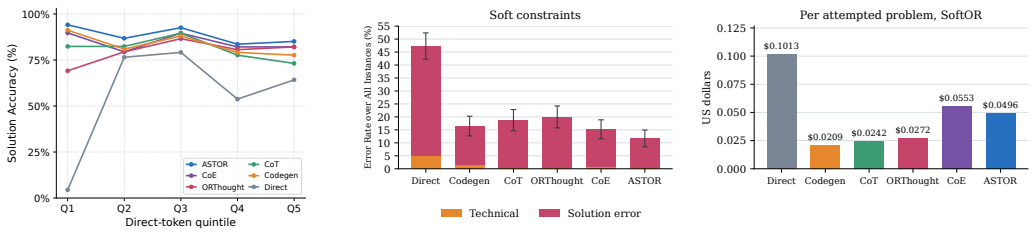
5.4 RQ3: Error Analysis

We analyze technical and solution errors as percentages of all evaluated attempts. Technical errors include malformed outputs, parsing, compilation, or execution failures. Solution errors indicate that the optimal value is different than the gold-standard. More details are in Appendix E.

ASTOR has the lowest error rate (i.e., percent of instances with either technical or solution errors) on SoftOR among the compared methods (Figure 3b). For solution errors on SoftOR, other methods fail on soft constraints because they typically let cost override the priority order, whereas ASTOR recovers the correct tier ordering in 74 of 75 LexiBench instances. Instead, it fails by over-enforcing a preference as a hard constraint or by confusing weighted-penalty for lexicographic formulations. On the hard-constraint benchmarks, error rates are similar for ASTOR, Codegen, CoT, ORThought, and CoE, for which full details are included in Appendix A.

5.5 RQ4: Cost Analysis

We report the incurred cost in USD of ASTOR and other methods at the time of evaluation. Figure 3c shows ASTOR is 10.3% less expensive than CoE, the strongest alternative, while also being the most accurate. Other cost results are in Appendix A, where ASTOR shows a similar profile.



(a) (b) (c)

Figure 3: (a) Accuracy by token quintile as a proxy for difficulty. (b) Error rates on SoftOR (%) with 95% CI. (c) Monetary cost in USD by route at time of evaluation.

5.6 RQ5: Ablation Studies

We isolate the prompt components and temperature effect of ASTOR. Following precedent for analysis on fixed benchmark subsets (Song et al., 2026; Lu et al., 2025; Yao et al., 2025), we construct a frozen, stratified 100-instance sample drawn from the eight benchmarks, including the four from SoftOR and four hard-constraint-only benchmarks in Appendix A. All main-text ablations report results on the 45 soft-constraint instances included in the sample, while Appendix A reports the corresponding results on the complementary 55 hard-constraint instances.

As seen in Appendix D, the ASTOR prompt has several components, and we remove one for each ablation. The ASTOR prompt includes the formal grammar in two parts, one as a human-readable markdown file in the style of a BNF grammar, and the other as a JSON schema that specifies the expected form of the output. It also supplies a benchmark contract, with bespoke information needed to use the data of each benchmark. Lastly, ASTOR uses a deterministic Python code repair which detects and fixes syntax mistakes in the JSON output.

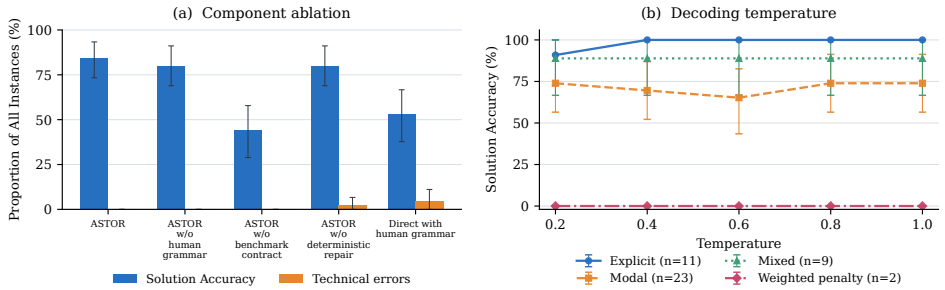


Figure 5: ASTOR ablations: (a) prompt ablations and (b) temperature ablation by priority signal. Whiskers show bootstrap 95% confidence intervals.

Removing the human-readable grammar and the deterministic repair reduced performance by only 4.4%, indicating that the serialized grammar may be enough and that syntax errors are uncommon. Removing the benchmark contract reduces performance by more. The last bar in Figure 5a gives the human grammar to the Direct method, provided as a tool for internal reasoning only. This method performs exactly the same as the Direct method on the ablation set. Across temperatures 0.0 to 1.0, ASTOR's accuracy on the 45-instance frozen SoftOR subset ranges from 75.6% to 84.4% (Figure 5b).

Table 5: Model-backbone ablation on the 45 soft-constraint instances. Solution accuracy (%); bootstrap standard errors are in parentheses.

Method	Gemini 3.7 Flash (reasoning)	Gemini 3.5 Flash Lite (no reasoning)	GPT-5 (reasoning, medium)	Gemini 3.1 Pro (reasoning)
Direct	53.3 (7.4)	6.7 (3.7)	44.4 (7.4)	33.3 (7.0)
Codegen	**84.4 (5.4)**	**44.4 (7.4)**	73.3 (6.6)	80.0 (6.0)
CoT	73.3 (6.6)	17.8 (5.7)	53.3 (7.4)	75.6 (6.4)
ORThought	77.8 (6.2)	28.9 (6.8)	73.3 (6.6)	**82.2 (5.7)**
CoE	82.2 (5.7)	42.2 (7.4)	71.1 (6.8)	80.0 (6.0)
ASTOR (Ours)	**84.4 (5.4)**	26.7 (6.6)	**75.6 (6.4)**	75.6 (6.4)

ASTOR ties Codegen for the highest accuracy on Gemini 3.7 Flash and leads on GPT-5. Codegen leads on the non-reasoning backbone, and ORThought leads on Gemini 3.1 Pro, where Codegen and CoE trail it by about two points and ASTOR by 6.6 points (Table 5).

6 Discussion and Conclusion

SoftOR makes soft-constraint autoformulation measurable through weighted penalties, lexicographic objectives, and solver-verified gold annotations, and ASTOR shows that a formal intermediate language improves accuracy on exactly those problems, particularly ordered preferences, while remaining competitive on hard constraints. Future work should define and evaluate on a metric of faithfulness to the original optimization problem or solver performance and extend the benchmark with less explicit language to test how well the method transfers to those settings. Overall, the results support formal languages and ASTs as practical foundations for optimization autoformulation.

Ethics Statement

This work does not involve human subjects, personal data, or sensitive attributes, so institutional review board approval was not required. SoftOR is built from publicly released benchmarks (COMPASS, TravelPlanner, SchedulingBenchmarks, and the ORThought releases of ComplexOR, IndustryOR, LogiOR, and NLP4LP) under their original licenses, together with LexiBench, an original dataset written and annotated by the authors. All priority annotations, gold constructions, and error labels were produced by the authors following the protocols in Appendices E and F, and no crowdsourcing or paid annotation was used. The natural-language instances describe scheduling, travel, and staffing scenarios with fictional entities and contain no real individuals. Experiments call commercial LLM APIs, and the available token usage and cost of every reported run are recorded in Appendix A at the prices listed in Appendix D so that the environmental and financial footprint of the study is visible. The methods and benchmark are intended to make optimization formulation more faithful to stated preferences and carry no foreseeable dual-use risk beyond that of general-purpose optimization tooling. The authors have no conflicts of interest or sponsorship to declare and have read and adhere to the ICLR Code of Ethics.

Reproducibility Statement

We release the complete ASTOR pipeline, the SoftOR benchmark with its gold annotations, all prompts, and the raw outputs and evaluation records of every reported run as supplementary material and at the repository linked in the abstract. The formal grammar, the definition of the problem class, and the full proof of adequacy are in Appendix C, and Appendix B works one instance through the pipeline with its literal compiler trace. Appendix D gives the exact prompts for the four single-request routes, the configuration and prompt provenance of ORThought and CoE, the benchmark contracts, the execution and evaluation rules, the frozen decoding and pricing settings, and the ablation definitions. The identifiers of the frozen 100-instance ablation subset are available through the repository. Appendix F documents every dataset adaptation, including the TravelPlanner priority rules, the LexiBench construction, the Nurse Rostering phrase mapping, the COMPASS templates, and the gold-label corrections applied to the ORThought releases. Section 5.1 states the backbone, decoding settings, and bootstrap protocol, and Appendix E gives the error-labeling decision order.

AI Use Statement

In this work, we used generative AI tools to generate the natural-language instances of SoftOR by applying a dataset-construction procedure that the authors designed without AI, to clean and reformat source datasets, and to assist in the exposition of author-developed adequacy proofs in Appendix C, but not with formulating the mathematical claims. It was also used to provide feedback on the experimental design, and to implement the ablation experiments. We have not used generative AI tools to develop the conceptual framework or the formal grammar, to propose or refine hypotheses, to design the methodology, or to interpret results. Translation and qualitative or thematic analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit software code, to create scientific figures, to brainstorm, to edit the paper for readability, and to format references. We have reviewed all AI-assisted work. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 577–596. PMLR, 2024.

Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Pearson Education, 2 edition, 2007.

Rajeev Alur, Rastislav Bodík, Garvit Juniwal, Milo M. K. Martin, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *2013 Formal Methods in Computer-Aided Design*, pp. 1–8, 2013. doi: 10.1109/FMCAD.2013.6679385.

Associated Press. AI-assisted shopping is the talk of the holiday shopping season, November 2025.

Michel L. Balinski and H. Peyton Young. *Fair Representation: Meeting the Ideal of One Man, One Vote*. Brookings Institution Press, 2 edition, 2001.

Andreas Bärmann, Alexander Müller, and Dieter Weninger. Lexicographic column generation with a tree search pricing algorithm. *Mathematical Methods of Operations Research*, 102:163–228, 2025. doi: 10.1007/s00186-025-00905-3.

James Cheney, Michael Norrish, and René Vestergaard. Formalizing adequacy: A case study for higher-order abstract syntax. *Journal of Automated Reasoning*, 49(2):209–239, 2012. doi: 10.1007/s10817-011-9221-6.

Tim Curtois. Nurse rostering benchmark instances, 2014.

Tim Curtois and Rong Qu. Computational results on new staff scheduling benchmark instances. Technical report, ASAP Research Group, School of Computer Science, University of Nottingham, 2014.

Haoxuan Deng, Bohao Zheng, Yirui Jiang, and Trung Hieu Tran. CAFA: Coding as auto-formulation can boost large language models in solving linear programming problem. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS 2024*, 2024.

James Garson. Modal logic. In Edward N. Zalta and Uri Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2023.

Bruce Golden, Linus Schrage, Douglas Shier, and Lida Anna Apergi. The unexpected power of linear programming: An updated collection of surprising applications. *Annals of Operations Research*, 343:573–605, 2024. doi: 10.1007/s10479-024-06245-5.

Linyuan Gong, Mostafa Elhoushi, and Alvin Cheung. AST-T5: Structure-aware pretraining for code generation and understanding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15839–15853. PMLR, 2024.

Sophie Henning, Nicole Macher, Stefan Grünewald, and Annemarie Friedrich. MiST: A large-scale annotated resource and neural models for functions of modal verbs in English scientific text. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1305–1324. Association for Computational Linguistics, 2022. doi: 10.18653/v1/2022.findings-emnlp.94.

José Antonio Hernández López, Martin Weyssow, Jesús Sánchez Cuadrado, and Houari Sahraoui. AST-Probe: Recovering abstract syntax trees from hidden representations of pre-trained language models. In *37th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1–11, 2022. doi: 10.1145/3551349.3556900.

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

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

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, volume 2025, pp. 101580–101606, 2025.

Sirui Li, Janardhan Kulkarni, Ishai Menache, Cathy Wu, and Beibin Li. Towards foundation models for mixed integer linear programming. In *International Conference on Learning Representations*, volume 2025, pp. 88590–88638, 2025.

Weibin Liao, Xin Gao, Tianyu Jia, Rihong Qiu, Yifan Zhu, Yang Lin, Xinyu Ma, Junfeng Zhao, and Yasha Wang. LearNAT: Learning NL2SQL with AST-guided task decomposition for large language models. In *International Conference on Learning Representations*, volume 2026, pp. 87420–87452, 2026.

Haoyang Liu, Jie Wang, Boxuan Niu, Xiongwei Han, Yian Xu, Mingxuan Ye, Zijie Geng, Fangzhou Zhu, Tao Zhong, Mingxuan Yuan, and Jianye Hao. Opt-verifier: Unleashing the power of llms for optimization modeling via dual-side verification. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 40769–40802. PMLR, 2025.

McKinsey & Company. The automation curve in agentic commerce, January 2026.

Martin Otto. Expressive completeness through logically tractable models. *Annals of Pure and Applied Logic*, 164(12):1418–1453, 2013.

Saswat Padhi, Elizabeth Polgreen, Mukund Raghothaman, Andrew Reynolds, and Abhishek Udupa. The SyGuS language standard version 2.1, 2023.

Paul Portner. *Modality*. Oxford University Press, 01 2009. ISBN 9780199292424.

Rizky Ramadhana Putra, Raihan Sultan Pasha Basuki, Yutong Cheng, and Peng Gao. NL2Logic: AST-guided translation of natural language into first-order logic with large language models. In *Findings of the Association for Computational Linguistics: EACL 2026*, pp. 6035–6051. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-eacl.317.

Tian Qin, Haoping Bai, Ting-Yao Hu, Raviteja Vemulapalli, Hema Swetha Koppula, Zhiyang Xu, Bowen Jin, Mert Cemri, Jiarui Lu, Zirui Wang, and Meng Cao. COMPASS: Benchmarking constrained optimization in LLM agents. In *Third Conference on Language Modeling*, 2026.

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, 2022.

Reuters. Retailers tap AI shopping traffic but fight to keep customer data, August 2026.

Aynat Rubinstein, Hillary Harner, Elizabeth Krawczyk, Daniel Simonson, Graham Katz, and Paul Portner. Toward fine-grained annotation of modality in text. In *Proceedings of the IWCS 2013 Workshop on Annotation of Modal Meanings in Natural Language (WAMM)*, pp. 38–46, Potsdam, Germany, March 2013. Association for Computational Linguistics.

Shopify. What is agentic shopping and what it means for your store, April 2026.

Yang Song, Anoushka Vyas, Zirui Wei, Sina Khoshfetrat Pakazad, Henrik Ohlsson, and Graham Neubig. NEMO: Execution-aware optimization modeling via autonomous coding agents. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

The Wall Street Journal. Inside BlackRock's AI transformation, April 2026.

Kai von Fintel and Sabine Iatridou. How to say ought in foreign: The composition of weak necessity modals. In Jacqueline Guéron and Jacqueline Lecarme (eds.), *Time and Modality*, volume 75 of *Studies in Natural Language and Linguistic Theory*, pp. 115–141. Springer, Dordrecht, 2008. doi: 10.1007/978-1-4020-8354-9_6.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022. doi: 10.52202/068431-1800.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *International Conference on Learning Representations*, 2024.

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

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. TravelPlanner: A benchmark for real-world planning with language agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54590–54613. PMLR, 2024.

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

Zijun Yao, Weijian Qi, Liangming Pan, Shulin Cao, Linmei Hu, Liu Weichuan, Lei Hou, and Juanzi Li. SeaKR: Self-aware knowledge retrieval for adaptive retrieval augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27022–27043. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1312.

Yansen Zhang, Qingcan Kang, Yujie Chen, Yufei Wang, Xiongwei Han, Tao Zhong, Mingxuan Yuan, and Chen Ma. SAC-Opt: Semantic anchors for iterative correction in optimization modeling. In *AIMS Workshop at ICLR 2026*, 2026.

A Additional Experimental Results

A.1 Hard-Constraint Accuracy

All results in this appendix use the backbone, request budget, and bootstrap protocol of Section 5.1.

Table 6: Exact solution accuracy (%) on hard-constraint benchmarks. Bootstrap standard errors are in parentheses.

Method	ComplexOR	IndustryOR	LogiOR	NLP4LP	Overall
Direct	**94.4 (5.4)**	75.9 (4.7)	62.0 (5.0)	78.1 (2.6)	74.9 (2.1)
Codegen	88.9 (7.5)	75.9 (4.7)	72.8 (4.7)	87.2 (2.1)	82.1 (1.8)
CoT	83.3 (8.8)	75.9 (4.7)	72.8 (4.7)	86.8 (2.2)	81.6 (1.9)
ORThought	83.3 (8.8)	78.3 (4.5)	**73.9 (4.6)**	85.5 (2.2)	81.6 (1.9)
CoE	88.9 (7.3)	**80.7 (4.3)**	71.7 (4.7)	86.4 (2.2)	**82.3 (1.8)**
ASTOR (Ours)	**94.4 (5.4)**	77.1 (4.6)	68.5 (4.9)	**87.6 (2.1)**	81.8 (1.8)

Table 7: Paired comparison of ASTOR with CoE and Codegen on conventional hard-constraint benchmarks. Δ is ASTOR accuracy minus baseline accuracy (%). Intervals are 95% confidence intervals from 10,000 bootstrap resamples.

	Instances solved correctly				ASTOR − baseline	
Baseline	Both	ASTOR only	Baseline only	Neither	Δ (%)	95% CI
CoE	345	11	13	66	-0.5	$[-2.5, 1.8]$
Codegen	346	10	11	68	-0.2	$[-2.3, 1.8]$

Table 6 reports all six methods on the conventional hard-constraint benchmarks. Aggregated over the 435 instances, ASTOR is statistically indistinguishable from CoE (Table 7), from Codegen (paired 95% CI $[-2.3, 1.8]$ pp), and from ORThought ($[-2.1, 2.5]$ pp).

Table 8 complements the soft-constraint backbone analysis in Table 5 by reporting the same comparison on the 55 hard-constraint instances in the frozen 100-instance ablation sample.

Table 8: Model-backbone ablation on the 55 hard-constraint instances. Solution accuracy (%); bootstrap standard errors are in parentheses.

Method	Gemini 3.7 Flash (reasoning)	Gemini 3.5 Flash Lite (no reasoning)	GPT-5 (reasoning, medium)	Gemini 3.1 Pro (reasoning)
Direct	78.2 (5.6)	18.2 (5.2)	78.2 (5.6)	78.2 (5.6)
Codegen	81.8 (5.2)	78.2 (5.6)	80.0 (5.4)	80.0 (5.4)
CoT	83.6 (5.0)	76.4 (5.7)	65.5 (6.4)	80.0 (5.4)
ORThought	83.6 (5.0)	**83.6 (5.0)**	**85.5 (4.8)**	80.0 (5.4)
CoE	83.6 (5.0)	80.0 (5.4)	81.8 (5.2)	**81.8 (5.2)**
ASTOR (Ours)	**85.5 (4.8)**	78.2 (5.6)	83.6 (5.0)	**81.8 (5.2)**

ASTOR leads on Gemini 3.7 Flash, ORThought leads on Gemini 3.5 Flash Lite and GPT-5, and ASTOR ties CoE for the lead on Gemini 3.1 Pro.

Table 9 lists results for four other LLM autoformulation methods as published by ORThought (Yang et al., 2026). These methods differ from ours in inference budget, number of repetitions, and backbone: CoE and ORThought use GPT-4.1-nano, ORLM uses LLaMA-3-8B, and OptiMUS is reported only on NLP4LP using author-supplied intermediate JSON.

Table 9: Published hard-constraint success rates (%) from Yang et al. (2026).

Method	ComplexOR	IndustryOR	LogiOR	NLP4LP
ORLM	77.78	32.13	15.58	67.80
OptiMUS	–	–	–	69.42
CoE	55.56	40.96	34.78	75.00
ORThought	77.78	57.83	46.01	89.02

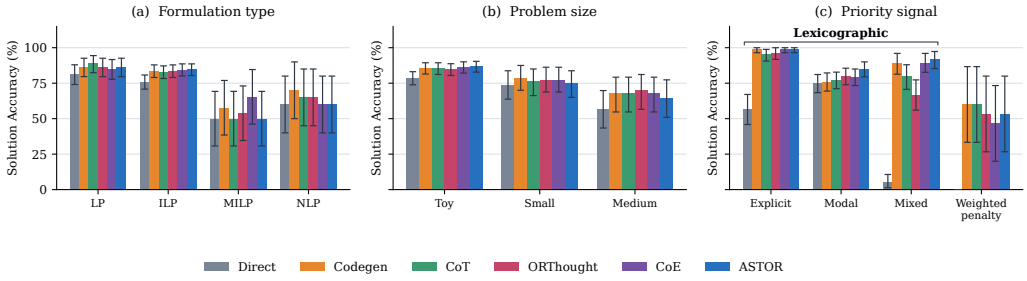


Figure 6: Hard-constraint accuracy by (a) formulation type and (b) problem size, and (c) soft-constraint accuracy by priority signal. Whiskers show bootstrap 95% confidence intervals.

Figure 6 disaggregates the hard-constraint benchmarks by formulation type and problem size for hard constraints, and priority signal for soft constraints. Problem types are Linear Program (LP), Integer Linear Program (ILP), Mixed Integer Linear Program (MILP), and Nonlinear Program (NLP). Problem sizes toy, small, and medium are defined in Yang et al. (2026). Priority is signaled by explicit, modal, or mixed markers for lexicographic formulations, and weighted penalty formulations are always explicit. ASTOR leads on ILPs and on toy problems, and a different baseline leads in each remaining category, and achieves the highest accuracy for every lexicographic signal, with its largest margins when priorities are modal or mixed. Codegen and CoT tie for the highest accuracy on weighted-penalty Nurse Rostering.

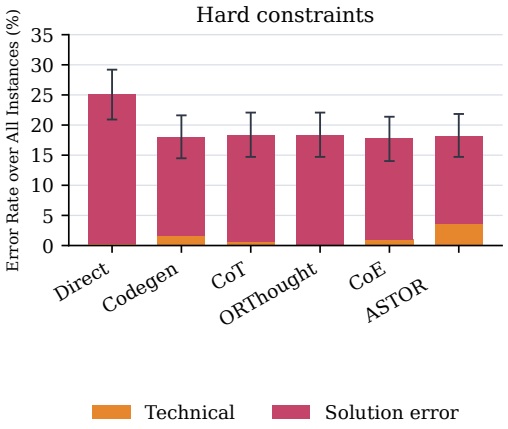


Figure 7: Error rates on the hard-constraint benchmarks as percentages of all attempts. The unplotted remainder is correct. Whiskers show bootstrap 95% confidence intervals for the total error rate.

Figure 7 reports error rates on the hard-constraint benchmarks. Every method except Direct has a similar total error rate.

A.2 Constraint Satisfaction

Table 10: Average hard constraint satisfaction (%) over all instances on SoftOR.

Method	Nurse Rostering	COMPASS	Travel Planner	LexiBench
Direct	31.8	57.6	**99.0**	5.3
Codegen	**100.0**	**100.0**	96.3	**100.0**
CoT	**100.0**	96.5	96.4	96.0
ORThought	86.7	97.6	97.4	78.7
CoE	**100.0**	**100.0**	97.5	**100.0**
ASTOR (Ours)	**100.0**	**100.0**	98.8	**100.0**

ASTOR satisfies every hard constraint on COMPASS and LexiBench and is within about a point of the lead on TravelPlanner; on Nurse Rostering it ties Codegen, CoT, and CoE at 100% hard constraint satisfaction. More details on Nurse Rostering are available in Appendix F.

Table 11: Average hard constraint satisfaction (%) over all instances on the hard-constraint benchmarks. The only oracle on these benchmarks is the gold objective, so a solution counts as satisfying all hard constraints when its objective matches gold.

Method	ComplexOR	IndustryOR	LogiOR	NLP4LP
Direct	**94.4**	75.9	62.0	78.1
Codegen	88.9	75.9	72.8	87.2
CoT	83.3	75.9	72.8	86.8
ORThought	83.3	78.3	**73.9**	85.5
CoE	88.9	**80.7**	71.7	86.4
ASTOR (Ours)	**94.4**	77.1	68.5	**87.6**

A.3 Efficiency

We report recorded monetary cost from provider token-usage telemetry and observed cache hits. Cost per correct solution divides the total recorded cost by the number of exact-correct predictions. The top row of Figure 8 aggregates over all 790 evaluated problems and the bottom row restricts to the 355 SoftOR instances. Nurse Rostering costs use the new Gemini 3.8 Flash run; other benchmarks retain their original runs. Three Nurse Rostering API failures had no usage telemetry and are excluded from the recorded costs; budget reservations are not treated as measured charges.

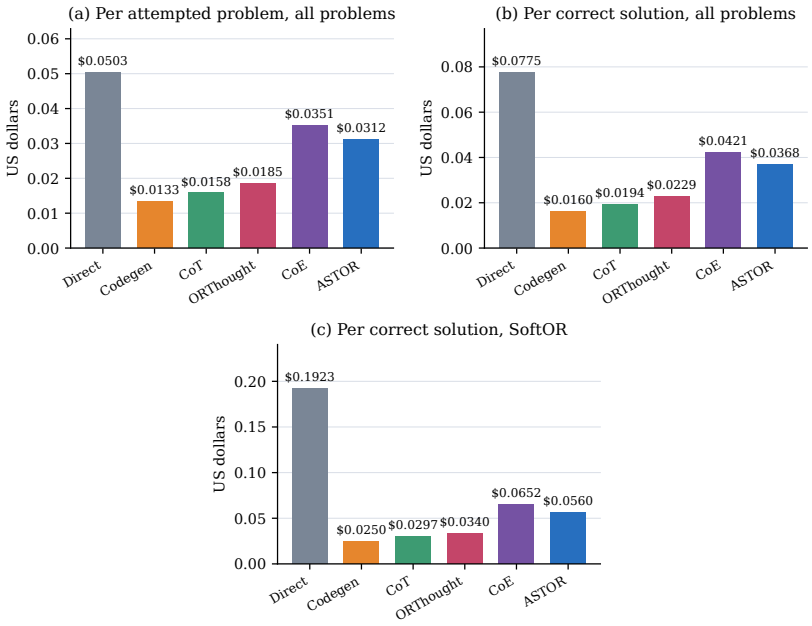


Figure 8: Recorded monetary cost per attempted problem and per correct solution over (a, b) all evaluated problems and (c) SoftOR only. ASTOR uses provider-reported cached-input pricing.

A.4 Ablation Studies

Figures 9 and 10 repeat the component and temperature ablations of Section 5.6 with the hard-constraint strata alongside the soft-constraint strata shown in the main text.

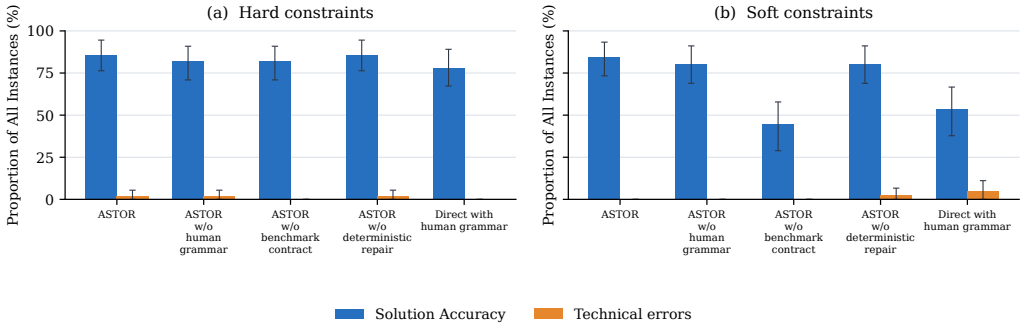


Figure 9: ASTOR component ablation on (a) hard-constraint and (b) soft-constraint benchmarks. Blue bars indicate percent of formulations that ended in a correct solution, whereas orange bars indicate the percent of formulations that terminated in technical errors. Whiskers show bootstrap 95% confidence intervals.

On the hard-constraint strata, removing the benchmark contract lowers accuracy by under four points, and removing deterministic repair changes nothing. The Direct + AST control (the Direct prompt with the ASTOR grammar prepended as a reasoning aid; see the ablation definitions in Appendix D) matches Direct exactly on both families, 78.2% on the hard-constraint instances and 53.3% on the soft-constraint instances, so the grammar helps only when its output is compiled and solved.

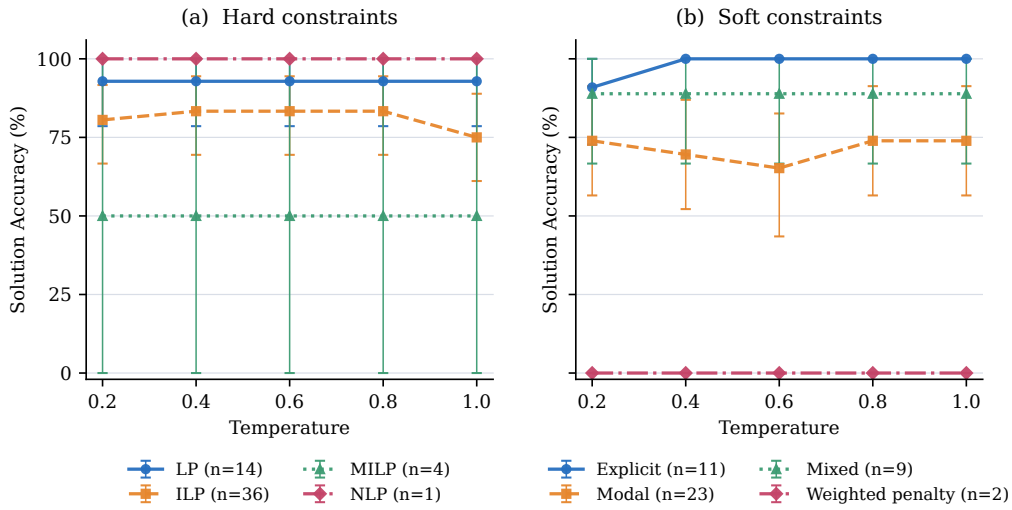


Figure 10: Impact of decoding temperature on ASTOR success rate for (a) hard-constraint and (b) soft-constraint benchmarks. Whiskers show bootstrap 95% confidence intervals.

ASTOR remains stable from temperature 0.0 to 1.0 in both families. The smallest strata have wide intervals.

B Worked Example

We use the following natural-language instance throughout. Blue marks the hard requirements and query, red marks the soft preferences.

> A ward manager is scheduling Alice for an 8-hour Day shift on any of Monday through Friday. Wednesday is already an agreed fixed day off, and her contract requires her to work at least 16 hours this week. She strongly prefers to work Monday and Tuesday, and slightly prefers to work Thursday. The manager would like to minimize the number of hours Alice works. Which days should she work?

This appendix works the Nurse Rostering example through a compact view of the grammar, from declarations to three finished models.

B.1 Simplified Grammar View

Figure 11 summarizes the compositional rules of an example grammar constructed for this problem. The complete grammar for ASTOR appears in Appendix C. A program is a block of declarations followed by a model, and a model is one objective together with a list of named constraints. Expressions are built from constants, symbols and indexed symbols, combined with arithmetic operators, a small set of functions, and aggregation over an index set. A formula either compares two expressions or joins other formulas with the usual connectives.

The objective is where the preferences are formalized, and it has three forms. `Objective(feasibility)` is a problem with nothing to optimize. `Objective(`*sense*`,`⟨`Expr`⟩`)` is a single scalar objective, which is the form a weighted-penalty model takes, since its penalty terms are folded into that one expression. `Objective(`*sense*`,`⟨`Tier`⟩`,...)` takes an ordered list of tiers, each carrying its own expression, and a lexicographic model uses that form instead. This lets an evaluator tell two formulations apart by looking at the tree, without going back to the natural language.

$$
\begin{aligned}
\langle \textit{Program} \rangle &::= \langle \textit{Declarations} \rangle \ \ \langle \textit{Model} \rangle \\
\langle \textit{Declarations} \rangle &::= \texttt{Declarations}(\langle \textit{Params} \rangle, \langle \textit{Vars} \rangle) \\
\langle \textit{Params} \rangle &::= \texttt{Parameters}(\textit{name} : \langle \textit{Const} \rangle, \ldots) \\
\langle \textit{Vars} \rangle &::= \texttt{Variables}(\langle \textit{Domain} \rangle, \textit{name}, \ldots) \\
\langle \textit{Domain} \rangle &::= \{0,1\} \mid \mathbb{Z} \mid \mathbb{R} \mid [\textit{low}, \textit{hi}] \\
\langle \textit{Model} \rangle &::= \texttt{Model}(\langle \textit{Objective} \rangle, \langle \textit{Constraints} \rangle) \\
\langle \textit{Objective} \rangle &::= \texttt{Objective(feasibility)} \\
&\quad \mid \texttt{Objective}(\textit{sense}, \langle \textit{Expr} \rangle) \\
&\quad \mid \texttt{Objective}(\textit{sense}, \langle \textit{Tier} \rangle, \ldots) \\
\langle \textit{Tier} \rangle &::= \texttt{priority}(k, \langle \textit{Expr} \rangle) \\
\langle \textit{Constraints} \rangle &::= \texttt{Constraints}(\langle \textit{Constraint} \rangle, \ldots) \\
\langle \textit{Constraint} \rangle &::= \textit{name}(\langle \textit{Formula} \rangle) \\
\langle \textit{Formula} \rangle &::= \langle \textit{Expr} \rangle \bowtie \langle \textit{Expr} \rangle \\
&\quad \mid \neg P \mid P \wedge Q \mid P \vee Q \mid P \Rightarrow Q \mid P \Leftrightarrow Q \\
\langle \textit{Expr} \rangle &::= \langle \textit{Term} \rangle \mid \langle \textit{Expr} \rangle \oplus \langle \textit{Expr} \rangle \\
&\quad \mid f(\langle \textit{Expr} \rangle, \ldots) \mid \mathrm{agg}_{v \in S} \langle \textit{Expr} \rangle \\
\langle \textit{Term} \rangle &::= \langle \textit{Const} \rangle \mid \langle \textit{Sym} \rangle \mid \langle \textit{Sym} \rangle [\langle \textit{Index} \rangle, \ldots]
\end{aligned}
$$

sense $\in \{\texttt{maximize}, \texttt{minimize}\}$; $\bowtie \in \{\leq, <, =, \neq, >, \geq\}$; $\oplus \in \{+, -, \times, \nabla\cdot, \hat{}\}$; $f \in \{\text{abs, min, max, exp, log, sqrt}\}$; $\mathrm{agg} \in \{\sum, \prod, \min, \max, \text{count}\}$; P, Q range over ⟨*Formula*⟩.

Figure 11: An example grammar, which generates optimization problems that have hard constraints, weighted-penalty objectives, or lexicographic preferences. Preferences are syntactically encoded.

B.2 Declarations

Figure 12 shows the declarations for the example. Parameters hold the numbers taken from the instance, here the shift length and the contract minimum. Variables hold the decisions, one binary per day of the planning period.

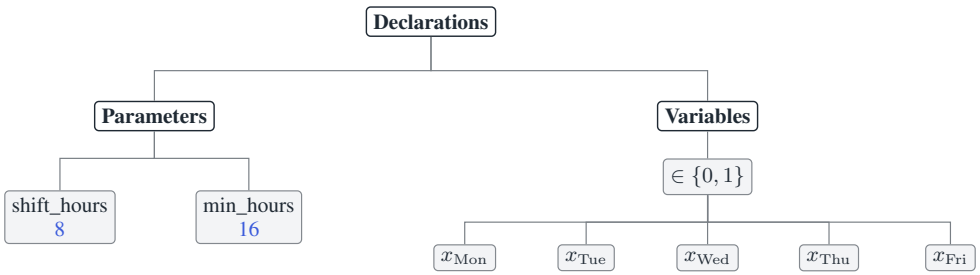


Figure 12: Declaration subtree for the nurse example, shared by all three models of Figure 13. Declares numeric data and variable names to be used in AST construction.

B.3 Three Formulations

Figure 13 sets the three formulations side by side, each AST above the model it compiles to. The declarations and the constraint subtree are the same in all three, and only the objective branch differs.

$hours = 8\,(x_{\text{Mon}} + x_{\text{Tue}} + x_{\text{Thu}} + x_{\text{Fri}})$, the total hours worked.

(a) Hard constraints

Model
- Objective:
  - minimize
    - $hours$
- Constraints:
  - min_hours
    - $hours \geq 16$
  - fixed_day_off
    - $x_{\text{Wed}} = 0$

$$\begin{aligned} \min \quad & hours \\ \text{s.t.} \quad & hours \geq 16 \\ & x_{\text{Wed}} = 0 \\ & \mathbf{x} \in \{0,1\}^5 \end{aligned}$$

optimum: any two shifts (16 h)

(b) Weighted penalty

Model
- Objective:
  - minimize
    - $hours + \lambda(1 - x_{\text{Mon}}) + \lambda(1 - x_{\text{Tue}}) + \mu(1 - x_{\text{Thu}})$
- Constraints:
  - min_hours
    - $hours \geq 16$
  - fixed_day_off
    - $x_{\text{Wed}} = 0$

$$\begin{aligned} \min \quad & hours \\ & + \lambda(1 - x_{\text{Mon}}) + \lambda(1 - x_{\text{Tue}}) \\ & + \mu(1 - x_{\text{Thu}}) \\ \text{s.t.} \quad & hours \geq 16 \\ & x_{\text{Wed}} = 0 \\ & \mathbf{x} \in \{0,1\}^5 \end{aligned}$$

optimum: Mon, Tue (16 h)

(c) Lexicographic

Model
- Objective:
  - minimize
    - priority 1 $(1 - x_{\text{Mon}}) + (1 - x_{\text{Tue}})$
    - priority 2 $(1 - x_{\text{Thu}})$
    - priority 3 $hours$
- Constraints:
  - min_hours
    - $hours \geq 16$
  - fixed_day_off
    - $x_{\text{Wed}} = 0$

$$\begin{aligned} \text{lex min} \quad & \big((1 - x_{\text{Mon}}) + (1 - x_{\text{Tue}}), \\ & (1 - x_{\text{Thu}}), \\ & hours\big) \\ \text{s.t.} \quad & hours \geq 16 \\ & x_{\text{Wed}} = 0 \\ & \mathbf{x} \in \{0,1\}^5 \end{aligned}$$

optimum: Mon, Tue, Thu (24 h)

Figure 13: Different formulations of the nurse problem. Formulation (a) contains no preferences, (b) penalizes violations by scalar terms λ, μ chosen by an author, and (c) tiers the preferences based on lexicographic strength, with the time objective as the final tie-breaker.

The optima differ as well. With no preferences stated, any two shifts that avoid Wednesday meet the 16-hour minimum, so the answer is not unique. Under the weighted objective at $\lambda = 3$ and $\mu = 1$, Alice works Monday and Tuesday: satisfying the Thursday request would cost eight more hours to relieve one point of penalty, so the solver drops it. Under the lexicographic objective she works Monday, Tuesday and Thursday, because hours only break ties among rosters that already satisfy both request levels, and no saving in hours can outrank a request at a higher level.

A weighted model will trade a preference away once the objective pays enough for it, and a lexicographic model will not.

B.4 Literal Compiler Trace

LexiBench: habitat crew-days

Problem statement. Divide the season's habitat-restoration crew-days among erosion control, wetland repair, native planting, and trail improvement. The wish we would surrender last is that at least 21 crew-days go jointly to erosion control and wetland repair; the others together do not equal it. The season allows 29 crew-days in total, and weather rules out any extension. Ignore how the days were split last season. Those remaining wishes are tied with one another and rank below it: every one of the 29 available crew-days is put to use; at least 10 crew-days go to native planting; and at least 10 crew-days go to trail improvement. Each site must receive at least one crew-day. Allocations must be nonnegative whole numbers. Determine the allocation.

Instance data.

Restoration category	Unit cost
Erosion control	13
Wetland repair	10
Native planting	12
Trail improvement	9

Available crew-days	29
Minimum per category	1

Benchmark stratum. Constructive; direct priority dominance; easy.

The model received this statement with a value-free data contract and returned an AST serialized as JSON. The reader view below is rendered directly from that JSON and preserves every declaration, hard constraint, soft constraint, priority, preference relation, objective, and unresolved item. It suppresses only serialization scaffolding such as repeated node-type labels, null defaults, expression wrappers, descriptions, and copied source text. The JSON Schema and benchmark instance are available in the anonymous repository. The instance data were bound only after formulation.

Reader view of the model-produced AST for LexiBench habitat crew-days

```
Model
|-- Declarations
|   |-- Set: Categories
|   |-- Parameters
|   |   |-- available_units : integer [crew-days]
|   |   |-- category_is_erosion_control[Categories] : integer
|   |   |-- category_is_native_planting[Categories] : integer
|   |   |-- category_is_trail_improvement[Categories] : integer
|   |   |-- category_is_wetland_repair[Categories] : integer
|   |   |-- minimum_units_per_category : integer [crew-days]
|   |   |-- unit_cost[Categories] : real
|   |-- Variable: allocated_units[Categories] : nonnegative integer [crew-days]
|-- Hard constraints
|   |-- total_available_units_limit
|   |   |-- SUM(c in Categories) allocated_units[c] <= available_units
|   |-- min_allocation_per_category
|       |-- FOR ALL c: allocated_units[c] >= minimum_units_per_category
|-- Soft constraints [aggregation: lexicographic]
|   |-- Priority 2: prefer_erosion_and_wetland_min_21
|   |   |-- erosion allocation + wetland allocation >= 21
|   |-- Priority 1 (three tied preferences)
|       |-- SUM(c in Categories) allocated_units[c] = available_units
|       |-- native planting allocation >= 10
|       |-- trail improvement allocation >= 10
|-- Other: preference_relations=[]; objective=feasibility; unresolved=[]
```

Deterministic continuation

Let x_E, x_W, x_N, x_T denote crew-days assigned to erosion control, wetland repair, native planting, and trail improvement. The compiler produced

$$x_c \in \mathbb{Z}_{\geq 0}, \qquad \sum_c x_c \leq 29, \qquad x_c \geq 1 \quad (c \in \{E, W, N, T\}),$$

with the lexicographic satisfaction vector

$$\text{lexmax} \left(\mathbb{1}[x_E + x_W \geq 21],\ \mathbb{1}\left[\sum_c x_c = 29\right] + \mathbb{1}[x_N \geq 10] + \mathbb{1}[x_T \geq 10] \right).$$

Gurobi returned $(x_E, x_W, x_N, x_T) = (1, 26, 1, 1)$. Its tier vector $(1, 1)$ matches the gold tier vector exactly.

C Formal Grammar and Its Adequacy

C.1 Complete Grammar

The complete abstract-syntax grammar used by ASTOR is given here. The suffixes `X*`, `X?`, `X{n}`, and `X{n,}` denote, respectively, zero or more occurrences, an optional occurrence, exactly n occurrences, and at least n occurrences. We read G as a typed grammar: symbol occurrences resolve to declarations, operators are applied to arguments of the displayed sorts, penalty weights and violations are nonnegative, and preference relations induce a finite consistent ordering.

Program and data declarations

```
Program
 declarations : Declaration*
 objective : Objective

Declaration ::=
 Set | Parameter | Variable |
    Piecewise
 | Constraint | SoftConstraint |
    PreferenceRelation

Set
 name : Name
 elements : SetValue*?

Parameter
 name : Name
 domain : ParameterDomain?
 indexed_by : Name*
 value_ast : Constant?
 values : ValueTable?
 units : Text?
```

Decision declarations

```
Variable
 name : Name
 domain : Domain
 indexed_by : Name*
 integrality_basis : Basis?
 units : Text?
 lower_bound : Expr?
 upper_bound : Expr?
 categories : SetValue*?

Piecewise
 name : Name
 input : Expr
 breakpoints : Number{2,}
 values : Number*?
 marginal_rates : Number*?
```

Constraints, preferences, and objectives

```
Constraint
 name : Name
 body : Formula
 source_text : Text?

SoftConstraint
 name : Name
 condition : Formula
 guard : Formula?
 aggregation : Aggregation?
 weight : Constant?
 priority : Constant?
 violation : Expr?
 source_text : Text?

PreferenceRelation
 relation : Relation
 left : Name | Expr
 right : Name | Expr
 source_text : Text?

Objective
 sense : minimize | maximize | feasibility
 expression : Expr?
```

Decision-dependent formulas and expressions

```
Formula ::=
 comparison (operator: CompareOp, left: Expr, right: Expr)
 | logical (operator: and|or, arguments: Formula{2,})
 | logical (operator: not, arguments: Formula{1})
 | logical (operator: implies|iff|xor, arguments: Formula{2})
 | quantifier (quantifier: forall|exists, variable: Name, domain: Name,
 condition: DataFormula?, body: Formula)
```

```
 | membership (element: DataExpr, collection: DataExpr, negated: Bool?)

Expr ::=
 constant (value: Value, value_type: ValueType?)
 | symbol (name: Name)
 | indexed_symbol (name: Name, indices: Expr*)
 | operation (operator: Op, arguments: Expr*,
 order: NormOrder?)
 | operation (operator: if,
 arguments: (Formula, Expr, Expr))
 | aggregation (operator: sum|product|min|max|count|average,
 index: Name, domain: Name,
 condition: DataFormula?, expression: Expr)
```

Data-only formulas and expressions

```
DataFormula ::=
 comparison (operator: CompareOp, left: DataExpr, right: DataExpr)
 | logical (operator: and|or, arguments: DataFormula{2,})
 | logical (operator: not, arguments: DataFormula{1})
 | logical (operator: implies|iff|xor, arguments: DataFormula{2})
 | membership (element: DataExpr, collection: DataExpr, negated: Bool?)

DataExpr ::=
 constant (value: Value, value_type: ValueType?)
 | symbol (name: DataName)
 | indexed_symbol (name: ParameterName, indices: DataExpr*)
 | operation (operator: Op, arguments: DataExpr*,
 order: NormOrder?)
 | operation (operator: if,
 arguments: (DataFormula, DataExpr, DataExpr))
 | aggregation (operator: sum|product|min|max|count|average,
 index: Name, domain: Name,
 condition: DataFormula?, expression: DataExpr)
```

Domains and terminal symbols

```
Domain ::= binary | integer | nonnegative_integer | real
 | nonnegative_real | bounded_integer | bounded_real
 | categorical
ParameterDomain ::= integer | real | boolean | categorical
Basis ::= explicitly_stated_lp | explicitly_stated_integer
 | indivisible_unit | divisible_quantity | convention_default
Aggregation ::= lexicographic | weighted_sum
Relation ::= prefer | higher_priority
CompareOp ::= le | ge | eq | lt | gt | neq
Op ::= add | subtract | multiply | divide | negate | power
 | sqrt | exp | log | sin | cos | tan | abs
 | min | max | norm
NormOrder ::= 0 | 1 | 2 | inf
ValueType ::= integer | real | rational | boolean | string
Value ::= Number | Text | Bool
ValueTable ::= { Text : Value }
SetValue ::= Number | Text
DataName ::= SetName | ParameterName | IndexName
SetName ::= Name
ParameterName ::= Name
IndexName ::= Name
Name ::= identifier
```

C.2 Representational Adequacy

Let $\mathcal{O}$ be the class of finite optimization problems over binary, integer, real, bounded, or finite categorical decision domains. Hard constraints and objectives are formed by finite compositions of constants, indexed quantities, arithmetic and elementary functions, norms, conditionals, piecewise-linear functions, finite aggregations, comparisons, Boolean connectives, and quantification over finite index sets. A problem may additionally contain nonnegative soft-constraint violations combined by either weighted penalties or a finite lexicographic priority ordering.

A *sort* is the type of value represented by a node, such as an integer, real number, Boolean value, string, or element of a declared finite set. An AST is *well-sorted* when every operation is applied to inputs of the appropriate types. For example, arithmetic operations require numeric inputs, a comparison of two numeric expressions produces a Boolean-valued condition, and a quantified index must range over a declared finite set. An AST is *well-formed* if its construction does not contradict the rules of G and if all references and indices are valid, expressions and aggregations are defined wherever evaluated, piecewise constructions are unambiguous, and priority annotations define a consistent ordering.

Let $\mathcal{A}$ be the set of well-formed, well-sorted ASTs generated by G. The grammar defines a map

$$[\![\cdot]\!] : \mathcal{A} \longrightarrow \mathcal{O}.$$

For an AST A, $[\![A]\!]$ is the optimization problem described by A: its decision domain, feasible set, and ordering of feasible solutions. This map is evaluated recursively, with each node applying its specified operation to the meanings of its children.

We consider two optimization problems equivalent if they have the same decision domain and feasible set and rank all feasible solutions in the same order. For this representation, adequacy means that the semantic image of G is exactly $\mathcal{O}$: the grammar represents all and only the problems in the target class (Cheney et al., 2012; Otto, 2013). The following results prove the two set inclusions.

Lemma C.1 (Representation of expressions and conditions)**.** *Every* `Expr` *generated by G is a well-sorted term, and every* `Formula` *generated by G evaluates to either true or false. Conversely, every expression and condition allowed in a problem in $\mathcal{O}$ can be represented by an* `Expr` *or* `Formula` *in G. The original and represented versions return the same value or truth value for every valid assignment of the decision variables and data.*

Proof. For the forward direction, we use induction over the nodes of a well-formed AST generated by G, proceeding from the leaves to the root.

Base case. A leaf is a `constant`, `symbol`, or `indexed_symbol`. A `constant` has its stated value and sort; a `symbol` or `indexed_symbol` inherits the sort of its declaration. Since G requires symbols and indices to agree with their declarations, every leaf is well-sorted.

Inductive step. The inductive hypothesis is that every child `Expr` or `DataExpr` is well-sorted and every child `Formula` or `DataFormula` is Boolean-valued. Proceeding to the parent, an `operation` applies a typed operator to compatible expressions; a conditional selects between compatible branches using a Boolean condition; and an `aggregation` combines well-sorted values over a declared finite domain. A `comparison`, `membership`, `logical`, or `quantifier` node returns a Boolean value when applied to children of the required sorts. By the inductive hypothesis and the typing requirements of G, the parent is therefore well-sorted if it is an expression and Boolean-valued if it is a formula. This completes the induction.

For the converse direction, we use induction over the nodes of any expression or condition allowed in a problem in $\mathcal{O}$.

Base case. A leaf is a constant, scalar quantity, or indexed quantity. It is represented by a `constant`, `symbol`, or `indexed_symbol` node, respectively, with the same value and sort.

Inductive step. The inductive hypothesis is that every child has a representation in G returning the same value or truth value on every valid assignment. Proceeding to the parent, arithmetic and elementary functions, norms, and conditionals are represented by `operation`; finite aggregations

by `aggregation`; piecewise-linear expressions by `Piecewise`; comparisons and membership conditions by their corresponding `Formula` productions; Boolean combinations by `logical`; and finite universal or existential conditions by `quantifier`. Data-only constructions use the corresponding `DataExpr` and `DataFormula` productions. By the inductive hypothesis, the represented children agree with the original children; because the parent applies the same operation, it also returns the same value or truth value. This completes the induction. □

Theorem C.1 (Well-definedness)**.** *For every $A \in \mathcal{A}$,*

$$[\![A]\!] \in \mathcal{O}.$$

Proof. Fix $A \in \mathcal{A}$. We verify each condition in the definition of $\mathcal{O}$.

Decision domain. The `Program` production contains finitely many declarations and one `Objective`. Its `Set` declarations define finite index sets, and its `Parameter` declarations define fixed data. Each `Variable` declaration selects one of the permitted domains in `Domain`. Since there are finitely many variables, these declarations determine a finitely presented decision domain

$$D_A = D_1 \times \cdots \times D_n,$$

where every D_i is binary, integer, real, bounded, or finite categorical. Each `Piecewise` declaration has finitely many breakpoints and therefore defines a finitely presented piecewise-linear function.

Feasible set. Let $C_1, \ldots, C_q$ be the `Constraint` declarations in A. Each C_j contains a `Formula`; by Lemma C.1, this formula returns a truth value for every valid $x \in D_A$. Let $h_j(x)$ denote that value. The hard constraints therefore define the feasible set

$$F_A = \{x \in D_A : h_j(x) = \text{true for every } j = 1, \ldots, q\}.$$

The same lemma ensures that each h_j is constructed only from the expressions and conditions permitted in $\mathcal{O}$.

Base objective. If the `Objective` contains an expression, Lemma C.1 implies that it defines a scalar function f_0 on D_A. To express every objective as a minimization ordering, define

$$\bar{f}_0(x) = \begin{cases} f_0(x), & \text{if the sense is } \texttt{minimize}, \\ -f_0(x), & \text{if the sense is } \texttt{maximize}, \\ 0, & \text{if the sense is } \texttt{feasibility}. \end{cases}$$

Thus, $\bar{f}_0$ induces exactly the ordering specified by the `Objective`.

Soft constraints. Let $S_1, \ldots, S_m$ be the `SoftConstraint` declarations. By Lemma C.1, each condition and guard is Boolean-valued and each explicit violation is a well-sorted expression. Under the interpretation of `SoftConstraint`, these fields define a nonnegative violation function v_i on D_A.

If the aggregation mode is `weighted_sum`, well-formedness supplies a nonnegative weight w_i for each v_i. The program orders feasible solutions by the scalar function

$$\bar{f}_0(x) + \sum_{i=1}^{m} w_i v_i(x),$$

which is a weighted-penalty objective of the form permitted in $\mathcal{O}$.

If the aggregation mode is `lexicographic`, the priority fields and `PreferenceRelation` declarations place the finitely many soft constraints into finitely many consistently ordered levels. Number these levels from most to least important and define

$$V_k(x) = \sum_{i:\mathrm{Priority}(i)=k} v_i(x).$$

The program then orders feasible solutions lexicographically by

$$\big(V_1(x), \ldots, V_K(x), \bar{f}_0(x)\big),$$

which is a finite lexicographic objective of the form permitted in $\mathcal{O}$. If A has no soft constraints, its ordering is determined by $\bar{f}_0$ alone.

Therefore, A defines a finitely presented problem with an admissible decision domain, a feasible set defined by permitted conditions, and a base, weighted-penalty, or finite lexicographic objective. Hence,

$$[\![A]\!] \in \mathcal{O}.$$

□

Theorem C.2 (Expressive completeness)**.** *For every $O \in \mathcal{O}$, there exists $A \in \mathcal{A}$ such that*

$$[\![A]\!] = O.$$

Proof. Fix $O \in \mathcal{O}$. We construct an AST $A \in \mathcal{A}$ and show that it represents the same optimization problem as O.

Decision domain. Because O is finitely presented, it contains finitely many index domains, data quantities, decision quantities, and piecewise-linear functions. Represent each finite index domain by a `Set` declaration and each data quantity by a `Parameter` declaration with the same value and domain. Represent each decision quantity by a `Variable` declaration with the same domain, indices, and bounds. Represent each piecewise-linear function by a `Piecewise` declaration with the same input, breakpoints, and values or marginal rates. These declarations reproduce the decision domain D_O of O.

Feasible set. Let $h_1, \ldots, h_q$ be the hard constraints of O. By Lemma C.1, each h_j has a `Formula` representation $\widehat{h}_j$ that returns the same truth value as h_j for every valid assignment. Place each $\widehat{h}_j$ in a `Constraint` declaration. The resulting constraints define

$$\{x \in D_O : \widehat{h}_j(x) = \text{true for every } j\} = \{x \in D_O : h_j(x) = \text{true for every } j\},$$

so they reproduce the feasible set of O.

Base objective. If O has a base objective f_0, Lemma C.1 supplies an `Expr` that returns $f_0(x)$ for every valid assignment. Place this expression in an `Objective` with the same minimization or maximization sense as O. If O is a feasibility problem, use `feasibility` with no objective expression. The resulting `Objective` therefore reproduces the base ordering of O.

Soft constraints. For each soft constraint i, use Lemma C.1 to represent its satisfaction condition by a `Formula` and its nonnegative violation function v_i by an `Expr`. Place these representations in the `condition` and `violation` fields of a `SoftConstraint` declaration.

If O uses weighted penalties, set `aggregation` to `weighted_sum` and place the corresponding nonnegative coefficient w_i in `weight`. The resulting declarations reproduce the objective

$$\bar{f}_0(x) + \sum_i w_i v_i(x).$$

If O uses lexicographic priorities, set `aggregation` to `lexicographic` and place the level of each soft constraint in `priority`. When the ordering is stated relationally, encode it with `PreferenceRelation` declarations. These declarations reproduce the ordered objective vector

$$\big(V_1(x), \ldots, V_K(x), \bar{f}_0(x)\big), \qquad V_k(x) = \sum_{i:\mathrm{Priority}(i)=k} v_i(x).$$

Assembly. Collect the constructed declarations and objective in a `Program`. All components use productions of G, and their types, domains, and aggregation fields agree with their declarations; therefore, the resulting AST A belongs to $\mathcal{A}$. Its declarations reproduce the decision domain of O, its hard constraints reproduce the feasible set of O, and its objective and soft constraints reproduce the ordering of feasible solutions in O. Under the stated notion of equivalence,

$$[\![A]\!] = O.$$

□

Corollary C.1 (Representational adequacy)**.** *The semantic image of G is exactly $\mathcal{O}$:*

$$\{[\![A]\!] : A \in \mathcal{A}\} = \mathcal{O}.$$

Proof. Theorem C.1 gives the inclusion from left to right, and Theorem C.2 gives the reverse inclusion. □

D PROMPTS AND INFERENCE PROTOCOL

The following are the exact prompt templates used in the reported comparison. Text in double braces is replaced deterministically before the request is sent.

D.1 ASTOR

AST prompt template

```
You are an expert in mathematical optimization. Your task is to formalize a
natural-language optimization problem as an optimization model.

You will be given a formal grammar that generates well-formed optimization
models. Using this grammar, construct an abstract syntax tree representing the
model the problem describes.

The problem may contain hard constraints, which
must always be satisfied, and soft constraints, which may be violated. It is
your job to determine the hardness or strength of each constraint, which may
be indicated by modal words of varied strength or by explicit comparison. For
problems with soft constraints, decide whether the optimization is
lexicographic or uses a weighted-sum penalty objective.

Here is the formal grammar:

{{grammar}}

The abstract syntax tree is serialized as JSON. You must strictly obey this
schema: it is parsed deterministically into solver code, and a response that
deviates from it is discarded unread.

{{ast_json_schema}}

TASK SEMANTICS:
{{benchmark_semantics_contract}}

BENCHMARK COMPATIBILITY CONTRACT:
{{benchmark_compatibility_contract}}

STEP 1 -- Declare every symbol. Classify each as exactly one of:
A = set: a collection the problem indexes over ("the plants", "each product")
B = parameter: fixed data the optimizer cannot change (costs, capacities, demands)
C = decision variable: a quantity the optimizer chooses (how much to ship, whether to open
    )

STEP 2 -- Express every requirement, preference, and objective in the appropriate
    prescribed form within
the serialized AST JSON, using ONLY the symbols declared in Step 1.

Do not invent or change problem information.
End your response with the complete serialized AST in a single ```json fenced
block: exactly one, containing the entire JSON object. Only that block is
parsed.
```

The request appends the instance using the exact boundary

AST request assembly

```
{{rendered_ast_prompt}}

PROBLEM:
{{problem_text_and_authoritative_data}}
```

Table 12: Deterministic substitutions in the AST prompt.

Placeholder	Source
`grammar`	The grammar reproduced in Appendix C, inserted verbatim from the artifact at runtime.
`ast_json_schema`	The strict JSON response schema, inserted in full at runtime and retained in the artifact rather than repeated in print.

Placeholder	Source
`benchmark_semantics_contract`	The benchmark-specific scoring semantics reproduced below.
`benchmark_compatibility_contract`	The benchmark-specific output interface reproduced below.

D.2 DIRECT

Direct prompt template

```
You are an expert in mathematical optimization. Your task is to formalize a
natural-language optimization problem as an optimization model.

Solve the optimization problem yourself. Use only the supplied problem
and authoritative instance data. Do not defer to a solver.

The problem may contain hard constraints, which must always be satisfied, and
soft constraints, which may be violated. It is your job to determine the
hardness or strength of each constraint, which may be indicated by modal
words of varied strength or by explicit comparison. For problems with soft
constraints, decide whether the optimization is lexicographic or uses a
weighted-sum penalty objective.

Your response must conform exactly to this JSON Schema:
{{direct_output_schema}}

When status is optimal, report a numeric objective_value and one concrete
optimal assignment in variable_values. When status is infeasible or unbounded,
use null objective_value and an empty variable_values object.

{{benchmark_semantics_contract}}

PROBLEM
{{problem_text}}

AUTHORITATIVE INSTANCE DATA
{{direct_instance_data}}
```

The inserted schema is a closed JSON object with exactly three fields: `status` in {`optimal`, `infeasible`, `unbounded`}, a numeric-or-null `objective_value`, and a `variable_values` object whose values are numeric.

D.3 CODEGEN

Codegen prompt template

```
You are an expert in mathematical optimization. Your task is to formalize a
natural-language optimization problem as an optimization model.

Write one complete Python program that formulates and solves the supplied
optimization problem with gurobipy. Do not solve in prose.

The problem may contain hard constraints, which must always be satisfied, and
soft constraints, which may be violated. It is your job to determine the
hardness or strength of each constraint, which may be indicated by modal
words of varied strength or by explicit comparison. For problems with soft
constraints, decide whether the optimization is lexicographic or uses a
weighted-sum penalty objective.

INSTANCE_DATA CONTRACT
{{instance_data_contract}}

TASK SEMANTICS
{{benchmark_semantics_contract}}

BENCHMARK COMPATIBILITY CONTRACT
{{benchmark_compatibility_contract}}

The primary Gurobi model object must be named 'model'; call model.optimize().
End by copying this reporting protocol verbatim and unchanged.
```

```
_STATUS_NAMES = {
 GRB.OPTIMAL: "optimal",
 GRB.INFEASIBLE: "infeasible",
 GRB.UNBOUNDED: "unbounded",
 GRB.INF_OR_UNBD: "infeasible_or_unbounded",
}
if model.Status == GRB.INF_OR_UNBD:
 model.Params.DualReductions = 0
 model.reset()
 model.optimize()
print(f"STATUS = {_STATUS_NAMES.get(model.Status, model.Status)}")
if model.Status == GRB.OPTIMAL:
 print(f"OBJECTIVE = {model.ObjVal}")
 for _variable in model.getVars():
 print(f"VARIABLE {_variable.VarName} = {_variable.X}")

Do not invent or change problem information. End your response with the complete program
     in a single ```python fenced
block: exactly one. Only that block is parsed.

PROBLEM
{{problem_text}}
```

D.4 CHAIN-OF-THOUGHT

CoT prompt template

```
The problem description is as follows:
{{problem_text}}

Please provide its mathematical model and Gurobipy code to solve this problem. The code
     you return should be a callable function whose parameters have default values
     extracted from the problem, and whose return value is the optimal objective function
     value of the model (if it exists), otherwise return None. Your response should
     include:

```model
[Your Model]
```

```python
[Your Code]
```

Now, Let's think step by step.

{{instance_data_contract}}
{{benchmark_semantics_contract}}
{{benchmark_compatibility_contract}}
{{reporting_protocol}}
```

The CoT route uses the same reporting adapter as Codegen. The text inserted at `reporting_protocol` begins with the following compiler adapter and is followed verbatim by the `_STATUS_NAMES` block in the Codegen prompt:

CoT compiler adapter

```
Compiler adapter: although the upstream prompt describes a callable function, this project
     executes the returned Python block as a complete standalone program. Define the
     primary Gurobi model object at top level, name it 'model', and call model.optimize().
End the program by copying this reporting protocol verbatim and unchanged.
```

D.5 ORTHOUGHT AND CHAIN-OF-EXPERTS

Both multi-agent baselines run from their official implementations, ORThought at commit `5293e248` of its public repository (Yang et al., 2026) and Chain-of-Experts at commit `afb89bf6` (Xiao et al., 2024). Their prompt prose is unchanged. Each agent prompt is followed by the same instance-data, task-semantics, and compatibility contracts that Codegen receives, and the prompt that emits the final program also carries the Codegen reporting protocol. ORThought runs its

model agent once and then up to three execution-conditioned debug rounds, which see only technical execution errors. CoE runs the conductor over the seven upstream experts for three selections, then the reducer, with the optional reflection pass disabled. Both use the backbone, decoding settings, and execution limits of Table 13, so an ORThought attempt makes one to four LLM requests and a CoE attempt makes seven. The verbatim prompt files and adaptation notes are in the released code.

D.6 Benchmark Contracts

Direct receives the task semantics only. ASTOR, Codegen, CoT, ORThought, and CoE also receive the route-specific compatibility text. These are the exact strings used for the reported benchmarks.

Hard-constraint benchmarks

```
TASK SEMANTICS:
Enforce every required choice with an equality or an activated lower bound; an upper bound
      alone is insufficient.

AST AND CODE COMPATIBILITY:
No benchmark-specific decision interface is required.
```

COMPASS

```
TASK SEMANTICS:
Select exactly one option. Report select[option_id] = 1 for the chosen option, using the
     supplied id verbatim; omitted entries count as 0.

AST COMPATIBILITY:
Declare binary decision variable select indexed by [Options] and select exactly one option
     . Bind every option-dependent soft-condition index within its formula.

CODE COMPATIBILITY:
Use Gurobi binary variables select[option] over Options and select exactly one option.
```

TravelPlanner

```
TASK SEMANTICS:
Select exactly one placement, one eligible transport option per leg, one eligible hotel
     per visited city, and one eligible restaurant per active meal slot; selected
     restaurants must be distinct. Evaluate preferences from most to least important;
     improvement on a less important preference cannot compensate for worsening a more
     important one, and consider total cost only afterward. Report the chosen plan as
     use_placement[placement_id], use_transport[transport_id], pick_hotel[placement_id,
     hotel_id], and assign_meal[meal_slot_id,restaurant_id] entries set to 1, using
     supplied ids verbatim; omitted entries count as 0.

AST COMPATIBILITY:
Candidate *_ok and *_active parameters define eligibility.

Use these binary decision variables exactly: use_placement[Placements], use_transport[
     TransportOptions], pick_hotel[Placements,Hotels], and assign_meal[MealSlots,
     Restaurants].

CODE COMPATIBILITY:
Candidate *_ok and *_active parameters define eligibility.

Use these Gurobi binary variable families exactly: use_placement[placement], use_transport
     [option], pick_hotel[placement,hotel], and assign_meal[slot,restaurant].
```

LexiBench

```
TASK SEMANTICS:
Enforce every required choice with an equality or an activated lower bound; an upper bound
      alone is insufficient. Evaluate preferences from most to least important;
     improvement on a less important preference cannot compensate for worsening a more
     important one, and apply any stated tie-break only afterward. Report exactly one
     indexed decision-variable family per supplied set, including every member with
     unselected members set to 0; report no auxiliary quantities.

AST COMPATIBILITY:
No benchmark-specific decision interface is required.
```

```
CODE COMPATIBILITY:
Create each decision-variable family with model.addVars(members, vtype=..., name="family")
      so Gurobi formats the brackets; never build a name with an f-string like f"family_{
     member}".
```

Nurse Rostering

```
TASK SEMANTICS:
Minimize the combined penalty for staffing-target deviations and unmet shift requests
     according to their stated severities. Report work[nurse,day,shift] = 1 for every
     worked shift, using supplied ids verbatim; omitted entries count as 0. Any numerical
     penalties you choose to use must be inferred from text and instantiated yourself.
     Never invent other problem data.

AST COMPATIBILITY:
Declare binary decision variable work indexed by [Nurses,Days,Shifts], where 1 means the
     nurse works that shift on that day. Auxiliary variables may use arbitrary names.

Declare every set and parameter listed in the external data contract exactly once, using
     those exact names and index domains, with values provided in EXTERNAL DATA CONTRACT
     declared null. You must instantiate any penalty weights with numbers or expressions.

CODE COMPATIBILITY:
Create work = model.addVars(nurses, days, shifts, vtype=GRB.BINARY, name="work"), where 1
     means the nurse works that shift on that day. Auxiliary variables may use arbitrary
     names.
```

For schema-bearing AST instances other than Nurse Rostering, the following sentence is appended verbatim to the AST compatibility contract:

External-data binding

```
Declare every set and parameter listed in the external data contract exactly once, using
     those exact names and index domains, with null external values.
```

The instance-data contract in Codegen and CoT states that `INSTANCE_DATA` is a flat dictionary with no `sets` or `parameters` wrapper. For schema-bearing instances, sets are lists, indexed parameters use string or comma-joined keys, and scalar parameters are bare numbers. For prose-only instances, the contract states verbatim: "`INSTANCE_DATA is an empty dict for this problem: every value you need is stated in the problem text itself.`"

D.7 Execution and Evaluation

ASTOR makes one LLM request. The output includes exactly one JSON block, repaired deterministically only when JSON syntax admits an unambiguous local correction, checked against the response schema and symbol table, and compiled deterministically to Gurobi. Direct is evaluated from its reported assignment; Codegen and CoT execute their single returned Python program, and ORThought and CoE execute the final program returned by their last agent. All routes then use the same benchmark oracle. The frozen runs contain 790 scored ASTOR attempts and no JSON-repair, error-retry, or rephrase API call artifacts; thus each scored ASTOR attempt used one LLM request.

For numeric objectives, a prediction is correct when the solver reports the gold status and the absolute error is at most $\max\{10^{-4}, 10^{-5}\max(1, |z^\star|)\}$, where $z^\star$ is the gold-standard optimal objective value for the instance. Infeasible and unbounded gold outcomes require an exact status match. Soft benchmarks additionally decode the reported assignment, verify feasibility, and recompute the weighted penalty or lexicographic tier vector; a lexicographic result must match every tier and its final numeric tie-break.

Table 13: Frozen settings for the reported runs.

Setting	Value
Provider and model	Gemini 3.7 Flash; Nurse Rostering: Gemini 3.8 Flash
Decoding	Temperature 0; maximum output 32,768 tokens (65,536 for Nurse Rostering)
Thinking	Provider default, except Nurse Rostering, where every route used a 32,768-token thinking budget
Sampling	One repeat per instance; seed 0; blind prompt mode
External tools	Disabled
Structured output	Disabled; deterministic validation applied after generation
Repair	Deterministic JSON handling enabled; no additional repair request observed
Execution	Gurobi 13+; 300-second generated-program and solver limits (3,600 seconds for Nurse Rostering)
Context cache	Enabled for reusable ASTOR context; provider-reported cache use priced as cached input
Pricing	USD per million tokens: input 0.75, cached input 0.075, output 3.75

Nurse Rostering is the one exception to the default thinking configuration. Under the provider default, all routes had substantial difficulty with Nurse Rostering, so the reported Nurse Rostering runs use a higher thinking budget and token limit for all six routes.

D.8 Ablation Definitions

The component study changes one element of the full ASTOR condition at a time. *Without benchmark contract* replaces both benchmark-contract fields by `None.`; and *without deterministic repair* disables local JSON repair while leaving the model request unchanged. The temperature and backbone studies change only their named decoding setting. *Direct + AST* is a prompt-only control rather than an ASTOR variant: the Direct prompt with the ASTOR grammar of Appendix C prepended and an instruction to construct the abstract syntax tree of the model as an internal reasoning step before answering. Its output schema, benchmark contract, instance data, decoding settings, and evaluation are those of Direct, and nothing is compiled or solved. It was run once on the frozen 100-instance sample with the Gemini 3.7 Flash configuration of the main experiment.

D.9 Frozen Ablation Sample

Following the cost-controlled analyses in NEMO, OptMATH, and SeaKR, which use restricted benchmark sets or sampled analysis subsets, we run ablations on a frozen subset of the main evaluation cohort (Song et al., 2026; Lu et al., 2025; Yao et al., 2025). We use Hamilton largest-remainder allocation to assign exactly 100 of the 790 instances proportionally across the eight reported benchmarks, yielding 55 hard-constraint and 45 soft-constraint instances (Balinski & Young, 2001). With master seed `20260825`, we sample uniformly without replacement within each stratum; the COMPASS allocation is further divided proportionally into ten hotel and one complex instance. The same IDs are used for every ablation condition, and no method output is used in their selection.

Table 14: Proportional allocation of the frozen ablation sample.

Benchmark	Cohort	Sample	Benchmark	Cohort	Sample
ComplexOR	18	2	COMPASS	85	11
IndustryOR	83	10	TravelPlanner	180	23
LogiOR	92	12	LexiBench	75	9
NLP4LP	242	31	Nurse Rostering	15	2
Hard total	435	55	Soft total	355	45

Frozen instance IDs. The identifiers of the 100 sampled instances, together with the per-stratum seeds, are provided as a machine-readable manifest in the anonymous repository.

D.10 Artifact Index

The anonymous repository contains the complete grammar and its JSON schema, the exact prompt templates for every method and for the Direct + AST control, the frozen 100-instance ablation manifest, the contract-construction code, the parser, validator, and compiler, the benchmark adapters and evaluators together with the SoftOR data, the frozen experiment configurations, the per-problem result records of the 790-instance evaluation and of the backbone ablation, the reporting inputs from which every table and figure is regenerated, and the test suite.

E Error Annotations and Discussion

For the error analysis in the main text, errors are classified as follows: a recorded parser, compiler, solver-execution, or other harness failure is *technical*. This includes a non-successful formulation-conversion status. An attempt that passes the automatic evaluator is *correct*. Every remaining attempt is a *solution error*. Thus the error rates use all evaluated attempts as their denominator, and the unplotted remainder in Figures 3b and 7 is correct.

For example, ComplexOR `prob_007` under Codegen returned no Python fence and was labeled *technical*. ComplexOR `prob_008` under CoT executed normally but returned objective 10 against gold 7 and was labeled a *solution error*.

E.1 Additional Error Analysis

E.1.1 Soft Constraints

On LexiBench, ASTOR matches gold-standard priority tiers in 74 of 75 instances. Its mistakes are usually about how strongly a requirement is enforced. On TravelPlanner it treats 17 of 124 preferences as hard constraints and 5 of 116 hard constraints as preferences. Treating a preference as hard was harmless in 11 of the 17 cases, and treating a hard constraint as a preference never caused a failure. On LexiBench it uses a weighted sum instead of a lexicographic objective on 14 instances. This is harmless when there is no cost tie-break (9 of 9 correct) and always fails when there is one (0 of 5), because the solver gives up a preference to save money.

ASTOR has 41 unsuccessful attempts on SoftOR: 27 on TravelPlanner, 6 on LexiBench, 7 on Nurse Rostering, and 1 on COMPASS. Of these, 1 is a technical failure and 40 are classified as solution errors by the evaluator. All seven unsuccessful Nurse Rostering attempts return hard-feasible rosters, with published-weight penalties between 1 and 4 points above the gold optimum. Lexicographic mistakes include wrong preference tiers or misdiagnosing a soft constraint as hard. Some examples: In the laptop instance, five equal wishes are followed by "the cheaper one wins"; ASTOR kept all five wishes but summed them with price, and satisfied three instead of four. In the service-vehicle instance, the text ranks wishes as "we would sooner run without X than without Y"; ASTOR reversed the order. In the makerspace instance, the top wish is "the one thing we will not trade"; ASTOR made it a hard constraint.

When a method gets a soft-constraint problem wrong, it is usually not because it missed a preference. It often treats one preference as too strict, or mixes the preferences together the wrong way. The baselines often end up minimizing cost instead of respecting the priority order, so a cheap plan wins even when it skips something more important. ASTOR's mistakes go the other way: it sometimes turns a preference into a hard rule, or lets cost outweigh a preference it should have kept.

E.1.2 Hard Constraints

On the hard-constraint benchmarks, a wrong answer is almost always a wrong objective value rather than a failed solve. A value better than gold means the method was able to achieve a lower value for a minimization objective or a higher value for a maximization objective than the annotated gold-standard optimum, and a value worse than gold means the opposite. Dropped constraints are the more common error for every method, by about two to one: Codegen returns 47 values better than gold and 22 worse, CoT 51 and 23, ORThought 55 and 22, CoE 46 and 24, and ASTOR 43 and 17, the fewest over-constrained formulations. Direct, which solves by hand, returns 71 better and 31 worse. Infeasible or unbounded formulations are rare, at most 3 per method.

When a method gets a hard-constraint problem wrong, it is almost always because it left out a constraint, not because it added one that was not there. That makes the solver report a better answer than is actually allowed. It is rare for a method to end up with an infeasible or over-constrained model instead. The grammar does not stop constraints from getting dropped, but when one does get dropped, it is more likely to show up as an obvious error instead of a wrong answer that looks right.

F Benchmark Construction and Dataset Adaptations

This appendix records the rules used to turn the source datasets into natural-language optimization problems and to compute their gold answers. Table 15 summarizes the four SoftOR components, and the sections below detail each construction.

Table 15: SoftOR composition. Gold annotations are provided or verified by enumeration/solver optimization.

Component	n	Objective	Priority Signal	Gold construction
Nurse Rostering	15	Weighted penalty	Explicit	Published optimum
COMPASS	85	Lexicographic	Explicit	Enumeration
TravelPlanner	180	Lexicographic	Modal	Enumeration
LexiBench	75	Lexicographic	Mixed	Enumeration/Solver Code

The specific linguistic cues used by the first three datasets are depicted in 14. These cues are further elaborated on in subsequent sections.

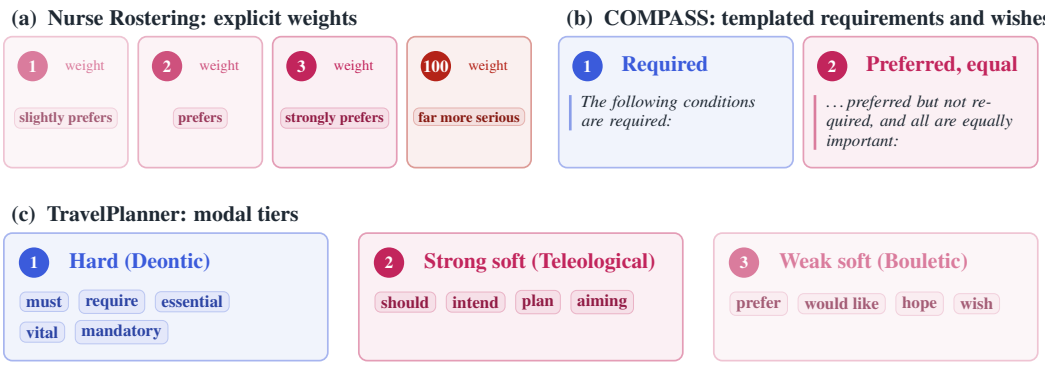


Figure 14: Priority annotation in SoftOR instances constructed from source data.

F.1 TravelPlanner Priority Rules

TravelPlanner uses modal language to distinguish requirements from preferences. We map deontic, teleological, and bouletic expressions to three optimization levels (Rubinstein et al., 2013; von Fintel & Iatridou, 2008; Portner, 2009; Henning et al., 2022).

Table 16: TravelPlanner modal priority rules.

Tier	Interpretation	Common cues
1	Hard constraint	*must, require, need, essential, mandatory, cannot, non-negotiable, ensure, will not, do not plan, avoid, not via*
2	Strong preference	*should, important, intend to, plan to, aiming for*
3	Weak preference	*prefer, would like, appreciate, hope, wish, interested in, ideally, open to, prefer not, rather not*

Budget, party size, origin, destination, and the original TravelPlanner validity rules are always hard constraints. Other local constraints are classified using the modal cue in the same clause. Tier 1 takes precedence when several cues match, except that *prefer not* and *rather not* remain weak preferences. Unrecognized expressions default to Tier 3 and are flagged for review. This procedure recognizes a cue for all 300 local constraints in the training and validation splits.

Let x range over candidate plans, let $v_i(x) \in \{0, 1\}$ indicate that plan x violates local constraint i, let $\mathrm{Priority}(i) \in \{1, 2\}$ be its strong or weak tier (tiers 2 and 3 of Table 16; tier 1 becomes a hard

constraint h_j), and let $f_0(x)$ be the plan's cost. The gold objective is

$$\begin{aligned} \operatorname*{lex\,min}_{x} \quad & \big(V_1(x),\, V_2(x),\, f_0(x)\big) \\ \text{s.t.} \quad & h_j(x) \le 0 \quad \forall j, \qquad V_k(x) = \sum_{i:\mathrm{Priority}(i)=k} v_i(x). \end{aligned}$$

A prediction is correct if it attains the same objective vector as the gold plan. It need not select the same plan when several plans are optimal.

F.2 LexiBench Construction

LexiBench contains five instances for every combination of three decision types and five preference structures, giving $3 \times 5 \times 5 = 75$ problems.

Table 17: LexiBench decision types.

Decision type	**Decision space**
Atomic	Select one of 12 supplied options.
Assembly	Select one element from each of three five-element sets, giving 125 possible combinations.
Constructive	Optimize over integer decision variables and an implicitly defined feasible region.

Table 18: LexiBench preference structures and difficulty labels.

Preference structure	**Property tested**	**Difficulty**
Direct priority dominance	One preference level dominates another.	Easy
Equal-level feature counting	All preferences have equal priority.	Easy
One-high-many-low	Several lower preferences cannot replace one higher preference.	Medium
Deep ladder	Preferences form four or five ordered levels.	Hard
Sparse transitive ordering	The full order must be inferred from pairwise comparisons.	Hard

LexiBench uses the same modal words in table 16 to signal preferences, with the same tier structure. It also uses explicit words "more important," "less important," and "above all" to get more than three tiers.

Difficulty depends only on preference structure, not decision type. The benchmark therefore contains 30 easy, 15 medium, and 30 hard problems. These labels and the gold annotations are not shown to the model.

Let $v_i(x) \in \{0, 1\}$ indicate that decision x violates preference i, let $\mathrm{Priority}(i)$ be its annotated level, ordered from highest to lowest, and let $f_0(x)$ be the stated tie-break objective, omitted when none is stated. Hard constraints h_j come from budget, capacity, or required attributes marked by deontic language in the natural language instance. The gold objective is

$$\begin{aligned} \operatorname*{lex\,min}_{x} \quad & \big(V_1(x), \ldots, V_K(x), f_0(x)\big) \\ \text{s.t.} \quad & h_j(x) \le 0 \quad \forall j, \qquad V_k(x) = \sum_{i:\mathrm{Priority}(i)=k} v_i(x). \end{aligned}$$

We certify each gold vector with Gurobi. We optimize the highest preference level, fix its optimal value, and repeat for each lower level. Any ordinary objective or tie-break is optimized last. Every stage must be optimal with zero MIP gap. A frozen manifest records the build seed, benchmark hash, solver version, and successful verification of all 75 problems.

F.3 NURSE ROSTERING

We convert the published request weights into qualitative language.

Table 19: Nurse Rostering preference phrases.

Published weight	Natural-language phrase
1	*slightly prefers*
2	*prefers*
3	*strongly prefers*
100	*far more serious*

The included instances assign penalties of 100 to understaffing and 1 to overstaffing. We express this by stating that understaffing is "far more serious" than overstaffing. The numerical penalties are not shown to the model and are only used to evaluate the final roster produced by each method. If the route chooses to use weights in creating a roster, it must create its own weights. Once the roster is produced, we only use the original numerical penalties for evaluation, but they are never used in generating a solution.

Note that the request weights 1, 2, 3, 100 are the numbers chosen by the original authors of Nurse Rostering at SchedulingBenchmarks, but those numbers are not unique in determining the preference order. For example, 3, 14, 86, and 1592 would create the same ordering. As a result, for our newly constructed natural language instances which *only* signal request strength through linguistic cues, it is precisely the task of each method to be able to infer and instantiate numerical weights which reflect the strength of the given language. In the weighted penalty experiments we never show the original weights of 1, 2, 3, 100 to the models, and allow them to choose their own numbers or whether to use weights at all. After doing this, they will produce a nurse roster for this problem, which specifies which nurses work on which days using indicator decision variables. We only use the original weights 1, 2, 3, 100 after the decision is made for the sake of evaluation: to accrue the penalty of each roster by noting when the chosen decision violates staffing and shift preferences, and multiplying by their weights in accordance with the original SchedulingBenchmarks gold-standard calculation.

Let $x_{nds} \in \{0,1\}$ indicate that nurse n works shift s on day d, and let r_{ds} be the number of nurses required on day d and shift s, as specified by the instance's cover requirement for each $(d,s) \in \{\text{days}\} \times \{\text{shifts}\}$. Each shift request i names a nurse, day, and shift (n_i, d_i, s_i) together with the phrase that carries its weight. The gold roster solves the weighted-penalty problem

$$\begin{aligned} \min_{x} \quad & V_1(x) + 2\,V_2(x) + 3\,V_3(x) + 100\,U(x) + O(x) \\ \text{s.t.} \quad & h_j(x) \le 0 \quad \forall j, \\ \text{Understaffing:} \quad & U(x) = \textstyle\sum_{(d,s)} \big(\text{required}_{ds} - \text{assigned}_{ds}(x)\big)^+, \\ \text{Overstaffing:} \quad & O(x) = \textstyle\sum_{(d,s)} \big(\text{assigned}_{ds}(x) - \text{required}_{ds}\big)^+, \end{aligned}$$

where $(\cdot)^+ = \max\{0, \cdot\}$, the sums run over days d and shifts s, $\text{required}_{ds} = r_{ds}$, $\text{assigned}_{ds}(x) = \sum_n x_{nds}$, and the hard constraints h_j encode the scheduling rules of the source benchmark. $V_k(x)$ counts the violated requests phrased *slightly prefers* ($k=1$), *prefers* ($k=2$), and *strongly prefers* ($k=3$); a request to work a shift is violated when the nurse is not assigned to it, and a request not to work a shift is violated when the nurse is. This is the weighted-penalty form of Section 3.1 with no base objective: request weights $1, 2, 3$, understaffing weight 100, and overstaffing weight 1. Each predicted roster is checked against the original hard scheduling rules described below and rescored using this objective. The published optimal objective is used as the gold value.

Table 20 reports partial-credit measures. *Hard Constraints Satisfied* is calculated separately for each roster. Each nurse is evaluated on nine hard constraint categories, which form the h_j in the formulation above: at most one shift per day, forbidden shift successions, limits on assignments to each shift type, total working-time bounds, fixed days off, maximum consecutive working days, minimum consecutive working days, minimum consecutive days off, and maximum weekends worked. A nurse receives one point for a category if and only if the rule is satisfied throughout the planning horizon. The roster score is the percentage of all nurse–category pairs receiving one point. Following

the benchmark, blocks extending beyond the start or end of the horizon are excluded from the two minimum-consecutive-length checks.

The remaining measures are also calculated separately for each roster:

$$\text{Understaffing} = \frac{\text{missing required nurse–shift positions}}{\text{total required nurse–shift positions}}.$$

$$\text{Overstaffing} = \frac{\text{assignments above staffing requirements}}{\text{total required nurse–shift positions}}.$$

$$\text{Weighted Preference Violation} = \frac{\text{total weight of violated shift requests}}{\text{total weight of all shift requests}}.$$

Understaffing and overstaffing may both be nonzero because different shifts can fall below and exceed their requirements. Table 20 reports the complements of these three measures, so that higher is better in every column: staffing coverage is one minus understaffing or overstaffing, and weighted preference satisfied is one minus weighted preference violation. Each of the 15 instances receives equal weight in its method's reported average, and an attempt with no decodable roster counts as satisfying no constraint.

Table 20: Nurse Rostering partial-credit measures (%), averaged over all 15 instances. Higher is better.

Method	Hard Constraints Satisfied	Staffing Coverage (under)	Staffing Coverage (over)	Weighted Preference Satisfied
Direct	31.8	27.9	31.1	28.1
Codegen	**100.0**	**94.8**	**97.9**	**88.0**
CoT	**100.0**	**94.8**	**97.9**	87.9
ORThought	86.7	82.3	85.2	76.2
CoE	**100.0**	**94.8**	**97.9**	87.9
ASTOR (Ours)	**100.0**	**94.8**	**97.9**	87.9

Direct returned five evaluable rosters: two satisfied all hard constraints and three violated at least one; the remaining ten attempts yielded no evaluable roster. Table 21 reports how many of the five evaluable rosters contained at least one violation from each observed rule family.

Table 21: Hard-constraint violations in Direct's five evaluable Nurse Rostering outputs.

Hard-constraint family	Rosters with a violation
Working-time bounds	2/5
Consecutive working-day bounds	2/5
Forbidden shift successions	1/5
Weekend limits	2/5
Consecutive days-off bounds	2/5
Shift-count limits	2/5

Working-time, consecutive-working-day, consecutive-days-off, weekend, and shift-count limits were each violated in two of the five evaluable rosters.

Currently, the version of Nurse Rostering we use has each route infer and instantiate its own penalty weights, and only uses author-published weights for calculating penalties for the final scoring. We chose this setup because our main question is whether methods can turn natural-language preferences into decisions. Supplying the numerical weights before optimization would provide part of the preference interpretation that we want the methods to perform. As an ablation, we ran a version of Nurse Rostering that allowed methods to see weights before decision-making. Direct sees numerical weights associated with natural language directly in the prompt, and all other routes use variable placeholders for each weight, which are deterministically replaced with the actual weights 1, 2, 3, or 100 when the code is generated, before running it to find an optimal solution. The results are presented in the table 22, which should be read in the same way as Table 4.

Table 22: Performance relative to gold-standard optimum on the 15 Nurse Rostering instances.

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Direct	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
Codegen	0	0	0	0	0	0	0	FAIL	0	0	0	0	0	0	0
CoT	0	0	0	0	0	0	FAIL	FAIL	0	0	0	0	0	0	0
ORThought	0	0	0	0	0	0	FAIL	FAIL	0	0	0	0	0	0	0
CoE	0	0	0	0	0	0	FAIL	FAIL	0	0	0	0	0	0	0
ASTOR	FAIL	0	0	0	0	0	FAIL	0	0	0	0	0	0	0	FAIL

As shown in Table 22, every feasible roster matches the gold optimum, suggesting that supplying the published weights makes it easier to optimize the corresponding preferences. These results reveal less about preference interpretation, since failures arise only from hard-constraint violations or the absence of an evaluable roster.

F.4 COMPASS

We convert each structured COMPASS record into natural language using fixed templates. Hard constraints begin with:

> The following conditions are required:

Soft constraints begin with:

> The following wishes are preferred but not required, and all are equally important:

Each requirement and wish is rendered as a numbered sentence. The generated text is parsed back into structured form to verify that no constraint was changed or omitted.

Let x range over the supplied candidates, let $v_i(x) \in \{0, 1\}$ indicate that candidate x fails wish i, and let the requirements be the hard constraints $h_j(x) \leq 0$ (marked in the instance with “require"). All wishes share one priority level, so the gold objective is

$$\begin{aligned} \operatorname*{lex\,min}_{x} \quad & \big(V_1(x)\big) \\ \text{s.t.} \quad & h_j(x) \leq 0 \quad \forall j, \qquad V_1(x) = \sum_i v_i(x). \end{aligned}$$

We recompute the optimum over all candidates and accept any feasible choice that attains it, including ties absent from the original annotation.

F.5 ORThought Gold-Label Corrections

We found 16 ORThought records whose answer files reported an optimal status but contained no objective value (Yang et al., 2026). We retained the original problem statements and reference formulations, reran the bundled solver code, and updated only the missing gold outcome. For solver results initially reported as infeasible or unbounded, we disabled dual reductions to distinguish the two statuses.

Table 23: Audited corrections to ORThought gold outcomes.

Dataset	**Problem ID**	**Correction**
ComplexOR	`prob_004`	Restored the objective value 250 from the accompanying sample and confirmed it with the reference formulation.
IndustryOR	`prob_045`	Changed optimal with a null objective to infeasible; the reference scheduling formulation produces an irreducible infeasible subsystem.

Dataset	Problem ID	Correction
NLP4LP	prob_002	Changed optimal with a null objective to infeasible; the required advertising purchases exceed the available budget.
NLP4LP	prob_027	Changed optimal with a null objective to infeasible; the vitamin D requirement is incompatible with the vitamin E upper bound.
NLP4LP	prob_050	Changed optimal with a null objective to infeasible; the heating requirement cannot be met within the electricity limit.
NLP4LP	prob_056	Changed optimal with a null objective to infeasible; the cooking-rate, oil-use, and grill-ordering requirements are incompatible.
NLP4LP	prob_057	Changed optimal with a null objective to infeasible; the minimum bottle counts and ratio require more syrup than is available.
NLP4LP	prob_067	Changed optimal with a null objective to infeasible; the crate minimum and ratio require more bananas than are available.
NLP4LP	prob_076	Changed optimal with a null objective to infeasible; the slicing-rate, grease-use, and slicer-ordering requirements are incompatible.
NLP4LP	prob_082	Changed optimal with a null objective to infeasible; the hydrogen requirement cannot be met under the pollution cap.
NLP4LP	prob_122	Changed optimal with a null objective to infeasible; the minimum nighttime-pill requirement exceeds the available painkiller medicine.
NLP4LP	prob_125	Changed optimal with a null objective to infeasible; the production requirements exceed the available labor hours.
NLP4LP	prob_134	Changed optimal with a null objective to infeasible; the pill-production requirements exceed the available material.
NLP4LP	prob_178	Changed optimal with a null objective to infeasible; the transport requirement cannot be met within the stated budget.
NLP4LP	prob_239	Changed optimal with a null objective to unbounded; the model maximizes production subject only to lower bounds and has no limiting resource constraint.
NLP4LP	prob_243	Changed optimal with a null objective to infeasible; the required bag ratio and minimum number of tiny bags exceed the energy budget.

F.6 ComplexOR Key-Name Reconciliation

Eight of ComplexOR's (Xiao et al., 2024) eighteen problems describe their data using one set of symbols in the natural-language question and another in the accompanying instance JSON. We reconciled the two representations by renaming each instance-JSON key to the symbol the question text uses for it.

Table 24: Renamed ComplexOR instance-JSON keys.

Dataset	Problem ID	Renamed key(s)
ComplexOR	prob_005	roll_width→RollWidth, widths→Width, orders→Orders, num_patterns→NumPatterns, num_rolls_width→NumRollsWidth

Dataset	Problem ID	Renamed key(s)
ComplexOR	prob_006	food_set→Foods, nutrient_set→Nutrients, food_cost→Cost, min_food_amount→MinAmount, max_food_amount→MaxAmount, min_nutrient_amount→MinNutrient, max_nutrient_amount→MaxNutrient, nutrient_amount→NutrientAmount
ComplexOR	prob_007	cost→Cost, n_min→MinRequirement, n_max→MaxRequirement, amt→NutrientAmount
ComplexOR	prob_008	jobs→Jobs, machines→Machines, proces_time→ProcesTime
ComplexOR	prob_011	supply→Supply, demand→Demand, cost→ShippingCost, limit→Limit
ComplexOR	prob_012	supply→Supply, demand→Demand, cost→Cost, limit→Limit
ComplexOR	prob_014	supply→Supply, demand→Demand, rate→Rate, limit→Limit
ComplexOR	prob_017	products→Products, stages→Stages, rate→Rate, profit→Profit, commit→Commit, market→Market, avail→Available