INSTANCE EVIDENCE MATTERS: TOOL-INTEGRATED LLM REASONING FOR DATA-INFORMED OPTIMIZATION FORMULATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) for optimization modeling typically treat instance data as values to be bound to a formulation inferred from problem descriptions. Yet instance data can play a more fundamental role: when interpreted under problem semantics, their structural and numerical properties become *instance evidence* that can directly inform formulation decisions, affecting both model correctness and solver efficiency. We introduce **InForm** (**IN**stance-informed **FORM**ulation), a tool-integrated framework that progressively acquires and uses such evidence during optimization formulation. Rather than analyzing the entire instance upfront, InForm interleaves formulation reasoning with targeted tool calls, acquiring decision-relevant evidence before the corresponding formulation decisions are finalized. Its formulation-oriented tools transform raw instance data into mathematically meaningful evidence, while experience retrieval provides reusable guidance for translating recurring evidence patterns into local formulation decisions. Across four large-scale OR benchmarks, InForm achieves 84.1% average formulation accuracy, outperforming the strongest baseline by 6.8 percentage points, and improving 16.2 percentage points on MIPLIB-NL against base LLM. InForm also yields 1.07×–2.18× solver-time improvements on commonly correct instances. These results show that instance data should not merely instantiate an optimization model: the evidence they reveal can actively shape how the model is formulated.

1 INTRODUCTION

Optimization modeling is a core task in operations research (OR), involving the abstraction of real-world decision problems into mathematical models and executable solver code (Hillier et al., 1995; Williams, 2013). Automating this process with large language models (LLMs) has therefore attracted growing attention. Existing approaches broadly follow two directions: learning-based methods develop specialized modeling capabilities through supervised fine-tuning or reinforcement learning (Huang et al., 2025; Lu et al., 2025; Chen et al., 2025), while prompt-based methods improve inference-time formulation through task decomposition, multi-agent collaboration, and structured search (Xiao et al., 2024; AhmadiTeshnizi et al., 2024; Astorga et al., 2025).

However, strong performance on small-scale benchmarks does not readily transfer to industrial-scale problems. Li et al. (2026) introduce MIPLIB-NL, an industrial-scale benchmark pairing natural-language descriptions with external tabular data, where models can involve millions of variables and constraints. Across 14 evaluated systems, average pass@1 accuracy is only 17.85%, with common failures involving data relations, index mappings, and cross-entity coupling. These results reveal the difficulty of aligning structured instance data with model formulations in industrial-scale modeling. Beyond this challenge, we identify an underexplored role of instance data in optimization modeling: instance data need not merely provide values for a predetermined formulation; *properties of a particular instance can themselves inform how the model should be formulated.*

Figure 1 illustrates this role of instance data in formulation. Consider a fixed-charge network flow problem in which activating an edge permits flow through it. From the problem description alone, a natural formulation is to encode this activation logic directly with one indicator constraint per candidate edge. Although semantically correct, this formulation introduces 2,928 indicator constraints

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

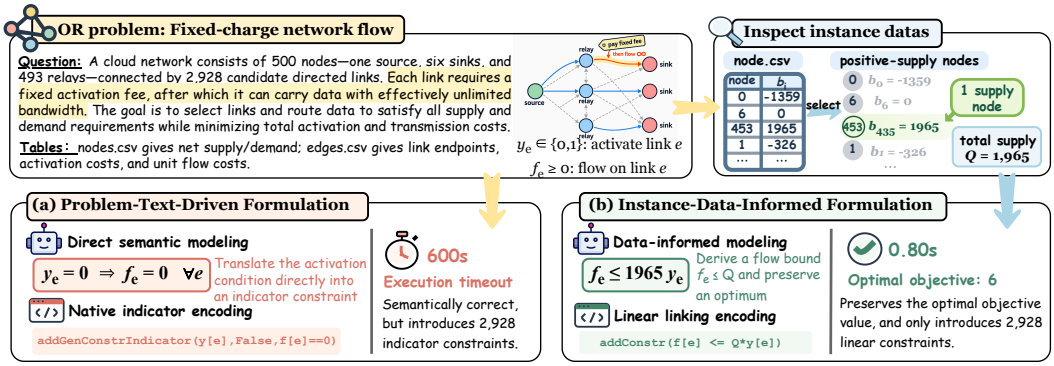


Figure 1: **A case study illustrating how instance data can inform formulation.** For a fixed-charge network flow problem, the text-driven formulation times out after 600s, while instance analysis reveals a total supply of $Q = 1965$, which provides a valid flow bound and enables the objective-preserving linear reformulation $f_e \leq Qy_e$, solving in 0.80s.

on the shown instance and times out after 600 seconds. The instance data, however, reveal an additional property: the total supply is $Q = 1965$. Under the semantics of network flow, this value is not merely a parameter to be substituted into an existing model; it also provides a valid upper bound on the flow through any edge. The activation logic can therefore be reformulated as $f_e \leq Qy_e$, preserving the optimal objective while reducing the solve time to 0.80 seconds.

This example reveals a broader principle: when interpreted under problem semantics, structural and numerical properties of an instance can yield *evidence* for formulation decisions. We call such formulation-relevant implications *instance evidence*. Unlike raw instance data, which describe a particular realization of a problem, instance evidence captures what those data imply, under the problem semantics, for constructing the model. Such evidence can arise not only from numerical bounds, but also from instance-specific relations, grouping structures, and feasible support patterns, informing decisions about variable domains, valid index combinations, cross-entity coupling, and alternative formulations. This perspective leads to the central question of this work: *how can an LLM determine what instance evidence is needed, acquire it as formulation decisions arise, and use it to guide those decisions?*

To address this question, we introduce **InForm** (**In**stance-informed **Form**ulation), a tool-integrated framework that interleaves formulation reasoning with targeted instance analysis. Rather than analyzing the entire instance upfront, InForm progressively acquires decision-relevant evidence as formulation decisions unfold. Its formulation-oriented instance analysis tools transform raw instance data into mathematically meaningful evidence, while an experience memory provides reusable guidance for translating recurring evidence patterns into local formulation decisions.

Our contributions are threefold:

- We identify an underexplored role of instance data in optimization modeling: beyond supplying values to an existing formulation, instance data can reveal *instance evidence* that directly informs how the model should be formulated.
- We propose InForm, a tool-integrated framework that combines progressive evidence acquisition, formulation-oriented instance analysis, and experience-guided reasoning to incorporate instance evidence into optimization formulation.
- Experiments on four large-scale OR benchmarks show that InForm achieves the best average accuracy, with particularly strong gains on the challenging MIPLIB-NL benchmark, while also improving solver efficiency.

2 Methodology

2.1 Instance-Informed Optimization Formulation

Given an OR problem P and a particular problem instance described by instance data D, let E_D denote the formulation-relevant *instance evidence* obtainable from D, and let $\mathbf{z}_D = (z_1, \ldots, z_T)$

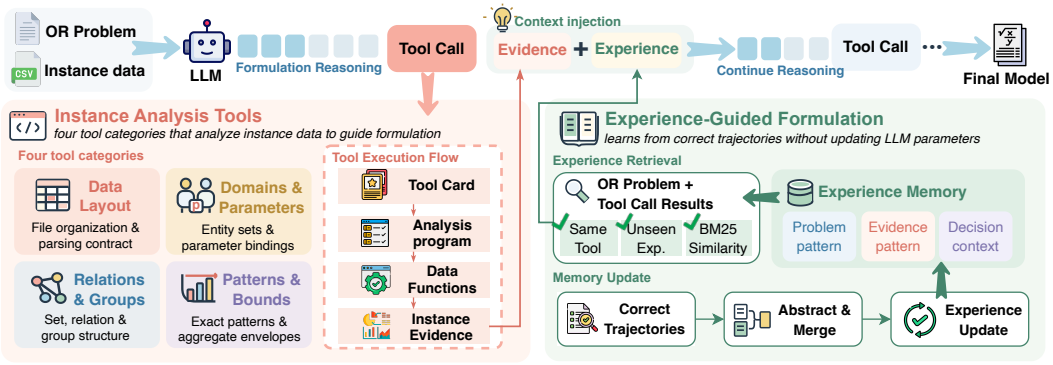


Figure 2: **Overview of InForm.** The framework integrates progressive formulation reasoning, four categories of formulation-oriented instance analysis tools, and experience-guided formulation through retrieval and memory updates, without modifying LLM parameters.

denote the sequence of formulation decisions that determines the model structure and algebraic representation of the final solver-ready formulation M_D:

$$M_D = \mathcal{M}(P, D; \mathbf{z}_D). \tag{1}$$

Instance evidence consists of structural or numerical properties extracted from the instance data and interpreted under the semantics of P in ways relevant to formulation. Such evidence may determine effective domains and parameter support, characterize relational or grouping structures, and reveal patterns or numerical bounds that affect the resulting formulation. We refer to constructing $\mathbf{z}_D$ conditioned on such evidence as *Instance-Informed Optimization Formulation*.

For an LLM with fixed parameters θ, formulation decisions are made sequentially. Let $E_t \subseteq E_D$ denote the instance evidence acquired from D before decision z_t. The formulation process can be expressed as

$$p_\theta(z_D \mid P, D) = \prod_{t=1}^{T} p_\theta(z_t \mid P, z_{<t}, E_t), \qquad E_t \subseteq E_D. \tag{2}$$

This formulation raises two questions: *when* should instance evidence enter formulation reasoning, and *how* should it be acquired from instance data and used for the current decision? To address them, we introduce **InForm**, shown in Figure 2, with three components: *Progressive Tool-Integrated Formulation Reasoning*, *Instance Analysis Tools*, and *Experience-Guided Formulation*.

2.2 Progressive Tool-Integrated Formulation Reasoning

Directly supplying raw instance tables to the LLM incurs substantial context overhead and requires the model to infer formulation-relevant properties from long, heterogeneous records. We therefore augment the LLM with executable instance analysis tools that selectively extract evidence from D, while leaving formulation decisions to the LLM.

Evidence requirements are state-dependent: earlier formulation decisions determine what must be resolved next and which instance evidence is relevant. InForm therefore interleaves evidence acquisition with formulation reasoning rather than analyzing all instance data upfront.

Formally, let $s_t = (P, z_{<t}, E_t)$ denote the formulation state before decision z_t. When additional evidence is needed, the agent selects a tool k_t and query q_t. Tool execution produces new evidence e_t, yielding

$$E_t^+ = E_t \cup \{e_t\}, \qquad s_t^+ = (P, z_{<t}, E_t^+).$$

If no additional evidence is needed, $s_t^+ = s_t$. The progressive interaction is summarized as

$$s_t \xrightarrow[e_t = \mathcal{T}_{k_t}(q_t, D)]{(k_t, q_t) \sim \pi_\theta^{\text{tool}}(\cdot \mid s_t)} s_t^+ \xrightarrow{z_t \sim p_\theta(\cdot \mid s_t^+)} s_{t+1}. \tag{3}$$

Repeating this process couples evidence acquisition with model construction, so that relevant instance evidence is incorporated before the corresponding formulation decision is finalized.

Tool: analyze relation support

Tool Card

- **When:** instance records enumerate admissible tuples.
- **Inputs:** a modeling anchor, frozen relation layout, endpoint roles, and their domain references.

Tool Program: fixed execution chain

1. **Resolve** relation and domain roles from layouts.
2. **Extract** the declared endpoint domains.
3. **Deduplicate** the observed endpoint tuples.
4. **Check** domain validity, Cartesian coverage, duplicates, unused members, and degrees.
5. **Render** the checked predicates as OR facts.

Tool Observation

200 distinct in-domain pairs out of 4,340 possible.
Mathematical interpretation: membership support $E \subsetneq T \times R$ (teams $\times$ people).

Experience: Constrain listed memberships

When

- **Problem pattern:** a relation table explicitly enumerates admissible pairs or memberships between two entity sets.
- **Evidence source:** *analyze_relation_support*.
- **Evidence pattern:** the observed tuples lie within the declared endpoint domains, but form a proper subset of their Cartesian product.
- **Decision context:** choosing which pairs index relation-dependent variables, sums, or constraints.

Guidance

Construct the relation support from the distinct observed tuples. Use this support to index the corresponding model components; do not expand it to the full Cartesian product.

Figure 3: **An instance analysis tool and an experience for relation support.** The tool analyzes instance data through a fixed program of analysis steps to obtain instance evidence. The experience maps a matching evidence pattern to local modeling guidance.

2.3 Instance Analysis Tools for Optimization Formulation

General-purpose table tools expose low-level operations such as filtering, joining, and aggregation, but leave the LLM to determine which results are relevant to formulation. We instead that package deterministic data operations into *formulation-oriented instance analysis tools* that return compact, mathematically interpretable evidence for the current formulation decision.

Formulation-Oriented Tool Categories. We first introduce *Data Layout* to map heterogeneous tables to semantic roles, providing a consistent interface for subsequent analysis. We then design three tool categories for entity and parameter support, relational and grouping structure, and structural or numerical properties:

- *Domains & Parameters* identifies entity domains, subsets, and indexed parameter support, informing model indices and parameter instantiation.
- *Relations & Groups* analyzes tuple support, cross-table mappings, and group memberships, informing valid combinations, summation scopes, and algebraic coupling.
- Patterns & Bounds detects parameter relationships, ordered structures, and aggregate envelopes that may support instance-specific bounds or alternative formulations.

Each category contains fine-grained tools targeting specific formulation decisions; detailed tool definitions are provided in the Appendix C.2.

Tool Design and Execution Workflow. As illustrated in Figure 3, each tool consists of an LLM-visible *Tool Card*, a fixed *Tool Program*, and a *Tool Observation*. The Tool Card specifies the tool capability, applicable formulation contexts, required semantic inputs, and returned evidence, allowing the LLM to determine both which tool to invoke and how to construct its arguments.

Once a tool is selected, its Tool Program executes a predefined, bounded, and deterministic sequence of table-analysis operations using the semantic arguments provided by the LLM. The resulting Tool Observation summarizes the checked results as compact instance evidence and, when applicable, a concise mathematical interpretation. For example, the relation-support tool in Figure 3 compares the observed endpoint tuples with their declared domains and returns both the support statistics and the interpretation $E \subsetneq T \times R$. The observation is then returned to the LLM for continued formulation reasoning, without directly prescribing variables or constraints.

2.4 Experience for Instance-Informed Formulation

Instance analysis tools expose formulation-relevant evidence, but a fixed-parameter LLM may not reliably translate recurring evidence patterns into appropriate local formulation decisions. We there-

fore maintain an *Experience Memory* that distills reusable evidence-to-decision associations from verified correct trajectories, providing guidance without updating model parameters.

Experience Context and Construction. As illustrated in Figure 3, each experience pairs a reusable applicability context with local formulation guidance. We represent an experience as

$$\xi_i = (p_i, \tau_i, e_i, d_i, g_i), \tag{4}$$

where p_i, τ_i, e_i, and d_i denote the problem pattern, evidence source (i.e., tool name), abstracted evidence pattern, and local decision context, respectively, while g_i denotes the formulation guidance. Experiences are distilled only from verified-correct trajectories. During abstraction, instance-specific entities, scales, and numerical values are removed while preserving the transferable association between the evidence context and the successful local modeling operation; redundant experiences are merged as the memory grows.

Progressive Experience Retrieval and Use. Retrieval is performed after each Tool Observation is obtained and before the next formulation-reasoning step. Experiences with the same evidence source τ_i are first selected as candidates. The current problem, observed evidence, and decision context are then matched against the stored problem pattern p_i, evidence pattern e_i, and decision context d_i, respectively, using BM25. Their normalized similarities are combined as

$$s_i = \alpha\hat{s}_i^p + \beta\hat{s}_i^e + \gamma\hat{s}_i^d, \qquad \beta > \gamma > \alpha, \tag{5}$$

where $\hat{s}_i^p$, $\hat{s}_i^e$, and $\hat{s}_i^d$ denote the normalized similarities in problem pattern, evidence pattern, and decision context. Candidates satisfying the corresponding similarity thresholds are ranked by s_i. The formulation guidance g_i from the highest-ranked eligible experience is then injected together with the current Tool Observation to guide the next formulation-reasoning step, as illustrated in Figure 3.

3 Experiments

3.1 Experimental Setup

Benchmarks. We evaluate on four large-scale OR modeling benchmarks: Large-Scale-OR (Liang et al., 2026), Bench4Opt (Wang et al., 2026b), Text2Opt-Bench (Gao et al., 2026) and MIPLIB-NL (Li et al., 2026); details are provided in the Appendix A.1. We normalize all benchmarks into a unified format containing data tables, problem and parameter descriptions, and table semantics.

Baselines. We compare against representative baselines from three categories.

- **Zero-shot LLMs.** General-purpose models evaluated through direct prompting: GPT-5.6-Sol (OpenAI, 2026), DeepSeek-V4-Flash, DeepSeek-V4-Pro (Xu et al., 2026), Qwen3-Max (Qwen Team, 2025) and Doubao-Seed-2.0-Pro (Bytedance Seed, 2026).
- **Fine-tuned LLMs.** We include ORLM-LLaMA-3-8B (Huang et al., 2025) and OptMATH-Qwen2.5-7B (Lu et al., 2025), trained on optimization-modeling data, as well as SIRL-Qwen2.5-7B (Chen et al., 2025) and StepORLM-Qwen3-8B (Zhou et al., 2026), which further incorporate solver-informed reinforcement learning or process supervision.
- **Agentic methods.** CoE (Xiao et al., 2024) and OptiMUS (AhmadiTeshnizi et al., 2024) use multi-agent orchestration; OptiTree (Liu et al., 2025) and AlphaOPT (Kong et al., 2026) leverage retrieved modeling knowledge or reusable experience; and LEAN-LLM-OPT (Liang et al., 2026) and ORPilot (Xie, 2026) incorporate explicit data-processing into the modeling workflow. All agentic baselines and our method use DeepSeek-V4-Flash as the backbone model.

Metrics. We report execution rate (ER) and accuracy (Acc). ER denotes the proportion of generated solver programs that execute successfully, while Acc measures modeling accuracy against reference answers.

To evaluate formulation efficiency, we further introduce **PCCS**, which compares solver runtime only on instances correctly solved by both InForm and baseline b, while reducing the influence of runtime fluctuations on trivially easy instances:

$$\mathrm{PCCS}(\mathrm{Ours}, b) = \exp\left[\frac{1}{|C_b|}\sum_{i\in C_b}\log\frac{T_{b,i}+\delta}{T_{\mathrm{ours},i}+\delta}\right]. \tag{6}$$

Table 1: **The overall performance of InForm and baselines on four OR benchmarks.** ER (%) is the executable rate and Acc (%) is the Pass@1 accuracy. Avg. is the macro-average over benchmarks. Best results are highlighted in **bold** and the second-highest values are underlined.

	Method	Large-Scale-OR		Bench4Opt		Text2Opt		MIPLIB-NL		Avg.	
		ER	Acc	ER	Acc	ER	Acc	ER	Acc	ER	Acc
Zero-shot	GPT-5.6-Sol	93.1	85.1	**100.0**	**88.8**	98.2	88.2	66.5	47.1	89.4	77.3
	DeepSeek-V4-Flash	98.0	82.2	98.0	79.7	88.2	79.1	52.9	32.1	84.3	68.3
	DeepSeek-V4-Pro	99.0	86.1	98.5	84.3	97.3	87.3	68.8	41.6	90.9	74.8
	DouBao-Seed2.0-Pro	97.0	78.2	95.4	77.7	90.9	80.9	53.4	32.1	84.2	67.2
	Qwen3-Max	97.0	80.2	95.9	79.2	95.5	85.5	55.7	30.8	86.0	68.9
Fine-tuned	ORLM	35.6	18.8	31.5	20.8	32.7	3.6	28.5	2.7	32.1	11.5
	OptMATH	34.7	21.8	23.9	13.2	15.5	7.3	11.8	0.9	21.4	10.8
	SIRL	56.4	28.7	62.9	35.0	37.3	5.5	34.4	3.2	47.8	18.1
	StepORLM	47.5	25.7	45.2	26.4	31.8	10.9	21.7	2.3	36.6	16.3
Agentic	CoE	91.1	80.2	97.5	82.2	90.0	75.5	72.4	38.0	87.7	69.0
	OptiMUS	90.1	77.2	96.4	74.1	90.9	74.5	58.8	35.7	84.1	65.4
	OptiTree	95.0	82.2	98.5	82.7	96.4	80.9	79.6	36.2	92.4	70.5
	AlphaOPT	92.1	78.2	95.9	74.6	94.5	78.2	82.8	43.4	91.3	68.6
	LEAN-LLM-OPT	94.1	85.2	99.5	85.8	91.8	76.4	67.9	30.3	88.3	69.4
	ORPilot	93.1	79.2	97.0	77.2	95.5	81.8	71.0	42.5	89.1	70.2
	InForm	**100.0**	**93.1**	**100.0**	86.3	**100.0**	**93.6**	**95.0**	**63.3**	**98.8**	**84.1**

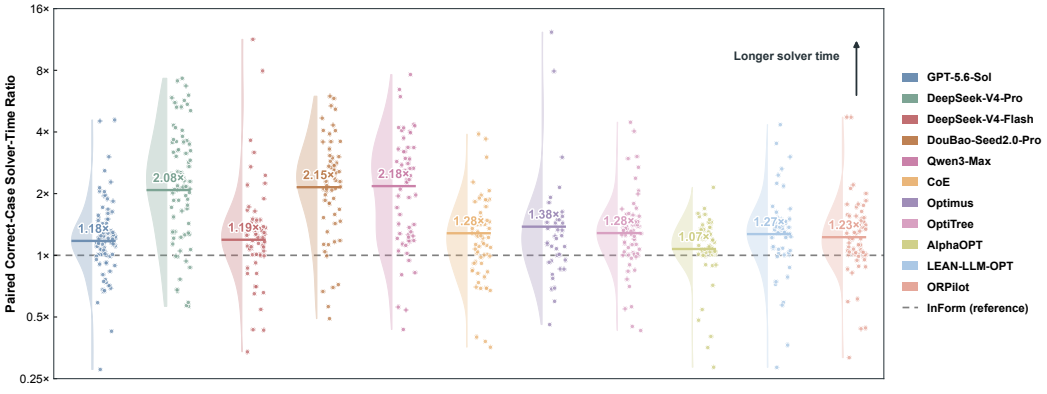


Figure 4: **Solver-time comparison of InForm and baselines on MIPLIB-NL.** The vertical axis shows the paired solver-time ratio $(T_b + 0.01)/(T_{\text{ours}} + 0.01)$, where larger values indicate longer baseline solver time. Colored horizontal bars indicate PCCS, the geometric mean of the paired solver-time ratios. The gray dashed $1\times$ line denotes InForm as reference.

Here, C_b denotes the set of commonly correct instances, and $T_{m,i}$ denotes solver runtime, excluding LLM inference time. We set $\delta = 0.01$ in our experiments. PCCS > 1 indicates faster solving with our formulation.

3.2 Main Results

Overall performance. As shown in Table 1, InForm achieves an average ER of 98.8% and Acc of 84.1%, obtaining the best or near-best performance across almost all metrics on the four benchmarks. The advantage is most pronounced on MIPLIB-NL, where our Acc reaches 63.3%, outperforming the second-best method by 16.2 percentage points. In contrast, specialized fine-tuned models achieve only 10.8–18.1% average Acc and degrade sharply on MIPLIB-NL. These results highlight the increasing importance of explicitly incorporating instance evidence as modeling scale and data complexity grow.

Solver efficiency. We further analyze solver efficiency on MIPLIB-NL, where formulation differences are most evident. Since the fine-tuned models solve only a small fraction of instances correctly, PCCS is reported for zero-shot and agentic baselines. Across these pairwise comparisons, PCCS ranges from $1.07\times$ to $2.18\times$, indicating aggregate solver-time advantages on the respective

Table 2: **Ablation accuracy (%) results on four benchmarks.** Green check and red cross marks indicate included and excluded components, respectively. Avg. is the macro-average across the four benchmarks. The best result in each column is shown in **bold**.

	Progressive	Tools	Experience	Large-Scale-OR	Bench4Opt	Text2Opt	MIPLIB-NL	Avg.
baseline	✗	✗	✗	82.2	79.7	79.1	32.1	68.3
w/o Pro.	✗	✓	✗	89.1	82.7	82.7	53.8	77.1
w/o Tools	✓	✗	✗	88.1	81.7	86.4	55.2	77.9
w/o Exp.	✓	✓	✗	91.1	83.8	87.3	58.8	80.2
InForm (full)	✓	✓	✓	**93.1**	**86.3**	**93.6**	**63.3**	**84.1**

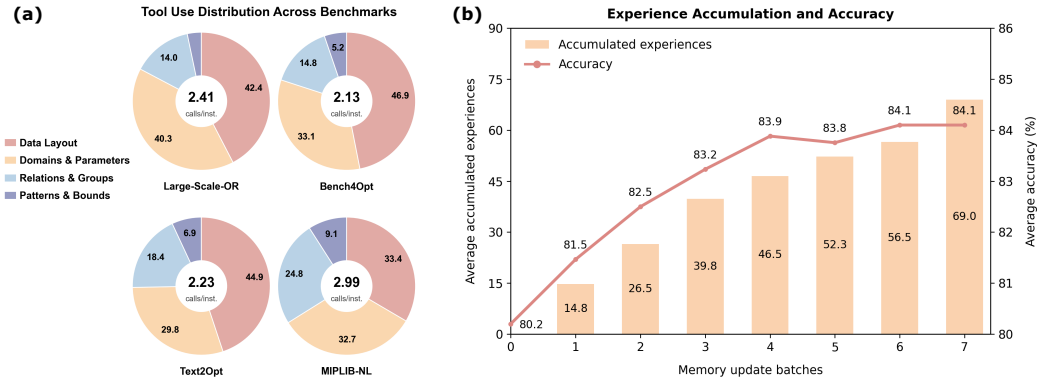


Figure 5: **Tool use and experience accumulation.** (a) Distribution of tool calls across four OR benchmarks in the no-memory runs. (b) Accumulated experiences and average accuracy at different stages of online experience accumulation.

common-correct subsets. These results indicate that InForm produces not only more accurate but also more efficient formulations. This improvement comes from leveraging instance evidence to select formulation choices better aligned with instance-specific structures and numerical properties.

3.3 Analysis on Tool-Use and Experience

Ablation study. Table 2 presents the ablation results of different components. *w/o Pro.* plans all analysis calls upfront and supplies their results together before formulation, without further tool calls. *w/o Tools* replaces our instance analysis tools with basic tabular analysis functions returning raw statistics. *w/o Exp.* removes experience while keeping progressive reasoning and instance analysis tools. Compared to *w/o Exp.*, the first two variants reduce average Acc by 3.1 and 2.3 percentage points, respectively, while *InForm* adds experience memory for a further 3.9-point gain.

Tool use analysis. Figure 5(a) summarizes tool calls across benchmarks, where donut segments show the distribution across tool categories and center values report the mean number of calls per instance. Data Layout is consistently the most frequently used category, as it establishes the prerequisite mapping between raw tables and semantic roles. MIPLIB-NL requires the most calls per instance (2.99) and broader use of Relations & Groups and Patterns & Bounds, indicating greater demand for relational and numerical evidence in more data-intensive problems.

Experience accumulation. We evaluate overall accuracy at different stages of online experience accumulation. At each stage, earlier first-attempt results are retained, and the remaining instances are evaluated using the memory available at that stage, held fixed throughout. In Figure 5(b), the bars report the average number of accumulated experiences and the line reports average Acc over the full task stream. As the experience bank grows to an average of 69 experiences, Acc improves by 3.9 percentage points, with larger gains in the early stages and diminishing returns thereafter.

3.4 Error Analysis and Case Study

Case Study. Figure 6 illustrates how instance evidence and retrieved experience correct a formulation error in the overlapping-team accommodation problem. GPT-5.6-Sol models each team only

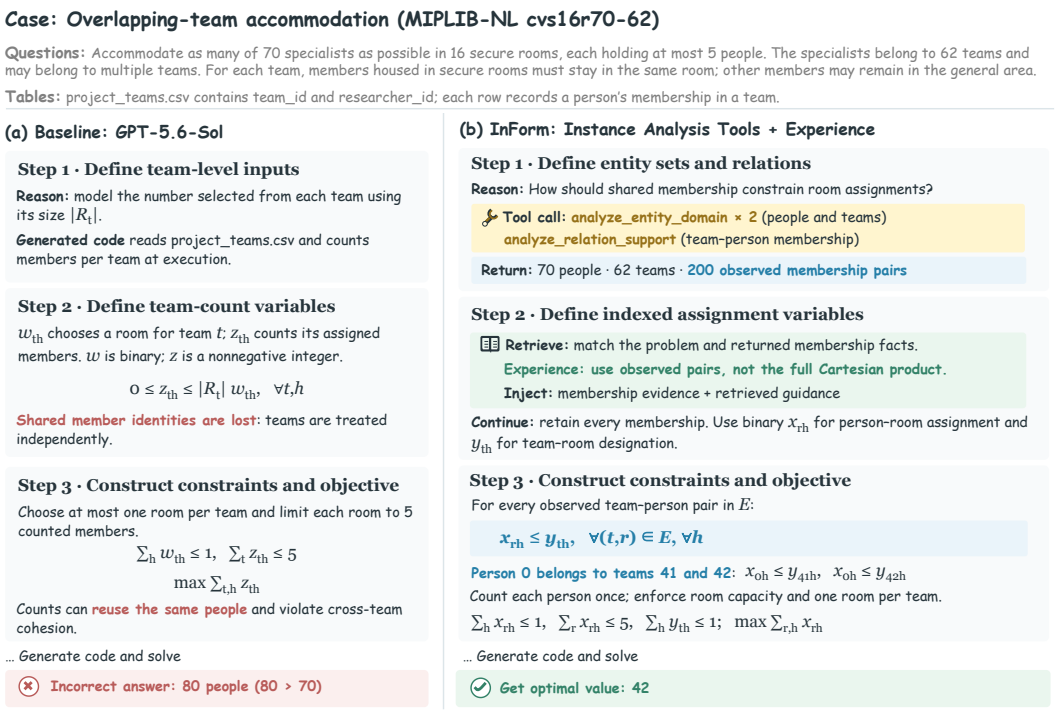


Figure 6: **Case study on MIPLIB-NL `cvs16r70-62`.** Instance evidence and retrieved experience guide constraint construction over observed memberships, avoiding the baseline's incorrect aggregation of overlapping teams.

by its member count, ignoring overlapping memberships and allowing the same researcher to be counted repeatedly. Our tools recover the entity domains and 200 observed team–researcher membership pairs, while retrieved experience guides the model to construct constraints over observed relations rather than aggregated statistics. This preserves cross-team memberships and recovers the reference objective value of 42.

Error Analysis. We manually classify erroneous cases across the four benchmarks for GPT-5.6-Sol, our backbone DeepSeek-V4-Flash, and three representative agentic baselines covering different agentic modeling strategies: CoE, OptiTree, and ORPilot. Errors are grouped into (i) *information errors*: missing instance-specific evidence; (ii) *semantic errors*: incorrect interpretation of problem semantics; (iii) *code errors*: data parsing, indexing, binding, or execution failures. Our method achieves the lowest information and code error rates, at 5.3% and 2.1%, respectively. Its semantic error rate remains slightly higher than GPT-5.6-Sol (12.1% vs. 11.8%), indicating that the main gains come from more reliable use and realization of instance data rather than stronger semantic reasoning.

Table 3: **Error analysis across four OR benchmarks.** Values are error rates (%) over all evaluated instances; lower is better.

Method	Info.	Sem.	Code.
GPT-5.6-Sol	8.9	**11.8**	5.9
DeepSeek-V4-Flash	7.8	15.4	13.5
CoE	8.0	18.3	8.6
OptiTree	10.0	17.0	7.0
ORPilot	6.5	20.7	6.7
InForm	**5.3**	12.1	**2.1**

4 Related Work

4.1 LLMs for Optimization Formulation

Existing LLM-based OR modeling methods improve formulation through learning or inference-time reasoning. Learning-based methods acquire OR-specific capabilities through supervised fine-tuning or reinforcement learning, including ORLM, LLMOPT, OptMATH, SIRL and StepORLM (Huang et al., 2025; Jiang et al., 2025; Lu et al., 2025; Chen et al., 2025; Zhou et al., 2026). OR-Toolformer

further combines fine-tuning with external OR tools, training the model to generate structured solver calls (Zhang et al., 2025). Prompt-based methods instead enhance formulation through decomposition and multi-agent collaboration, such as OptiMUS and Chain-of-Experts (AhmadiTeshnizi et al., 2024; Xiao et al., 2024); formulation search, such as Autoformulation and BPP-Search (Astorga et al., 2025; Wang et al., 2025); or retrieval of reusable modeling knowledge, as in OptiTree, AlphaOPT, and OptSkills (Liu et al., 2025; Kong et al., 2026; Yang et al., 2026).

Instance data has only recently received explicit attention. MIPLIB-NL introduces industrial-scale NL-to-Opt problems with model–data separation and shows substantial performance degradation on realistic large-scale instances (Li et al., 2026). Subsequent work focuses mainly on mechanical data processing, as in LEAN-LLM-OPT (Liang et al., 2026), or programmatic coefficient and index binding, as in BIND (Gao et al., 2026). These methods facilitate data use but largely assume the formulation structure is already determined. In contrast, we study how *instance evidence* conditions formulation decisions, including relation supports, grouping structures, and numerical envelopes.

4.2 Tool-Integrated LLM Reasoning

Tool-integrated LLM research can be broadly organized around learning tool-use capabilities and integrating tools into multi-step reasoning. Toolformer and ToolLLM learn when and how to invoke external APIs through self-supervision or instruction tuning (Schick et al., 2023; Qin et al., 2024). A complementary line embeds tools directly into agentic reasoning: ReAct interleaves reasoning, actions, and observations (Yao et al., 2023), while HuggingGPT, AutoGen, ToolPlanner, and OctoTools support task decomposition, tool planning, and coordination across heterogeneous tools (Shen et al., 2023; Wu et al., 2024a;b; Lu et al., 2026).

Recent work further strengthens reasoning-time tool interaction. START activates latent tool-use capabilities through inference-time guidance (Li et al., 2025), WALT abstracts low-level operations into reusable high-level tools (Prabhu et al., 2026), and VET feeds execution results from real data back into subsequent reasoning (Wang et al., 2026a). InForm follows this reasoning-integrated paradigm, but develops OR-specific tools for extracting formulation-relevant instance evidence.

4.3 Experience Memory for LLM Agents

Experience memory in LLM agents has developed from persistent interaction storage toward increasingly abstract and adaptive experience representations. Early systems emphasize long-term retention and selective retrieval, as in Generative Agents, MemoryBank, and MemGPT (Park et al., 2023; Zhong et al., 2024; Packer et al., 2023). A second line treats past trajectories as learning signals rather than records to replay: Reflexion and ExpeL extract reusable guidance from feedback and experience (Shinn et al., 2023; Zhao et al., 2024), while ReasoningBank further distills generalizable reasoning memories from both successful and failed trajectories (Ouyang et al., 2026). More recent work emphasizes memory organization and lifecycle management, with Mem0 and A-MEM supporting the extraction, consolidation, linking, and revision of long-term memories (Chhikara et al., 2025; Xu et al., 2025). Overall, agent memory is shifting from passive history storage toward compact, retrievable, and continuously evolving experience representations. InForm follows this experience-memory perspective, but organizes memory around instance-evidence patterns and local OR formulation guidance.

5 Conclusion

We introduced InForm, a tool-integrated framework that progressively incorporates instance evidence and reusable experience into optimization formulation. Experiments across four large-scale OR benchmarks show that InForm improves both formulation accuracy and solver efficiency, especially on data-intensive problems. These findings further underscore the importance of instance evidence in optimization modeling and motivate better use of such evidence to strengthen LLM-based OR formulation. InForm is limited by its predefined analysis tools and by experience memory that depends on previously verified correct trajectories. Moreover, reliable instance-information processing alone cannot resolve formulation errors caused by semantic misunderstanding in complex problems. Future work could explore iterative tool refinement and tighter integration of instance evidence with semantic reasoning.

AI USE STATEMENT

Generative AI tools were used to assist with method code implementation, benchmark data curation, figure and table preparation, initial manuscript structuring, and language polishing. All AI-assisted outputs were reviewed and revised by the authors, who take full responsibility for the final content of the paper.

REPRODUCIBILITY STATEMENT

The core workflow and algorithmic design of InForm are described in the Methodology, with detailed specifications of the instance analysis tools provided in the Appendix C.2. Implementation details and hyperparameter settings used in our experiments are provided in the Appendix A.2. Benchmark data and code are available in our anonymous repository at `https://anonymous.4open.science/r/InForm-73E3`.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. 1, 5, 9

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. Autoformulation of mathematical optimization models using LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, pp. 1864–1886, 2025. 1, 9

Bytedance Seed. Seed2.0 model card: Towards intelligence frontier for real-world complexity. *arXiv preprint arXiv:2607.00248*, 2026. 5, 14

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-Informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, 2025. 1, 5, 8

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025. 9

Zhiqi Gao, Albert Ge, Alexander Berenbeim, Nathaniel D Bastian, and Frederic Sala. Models can model, but can't bind: Structured grounding in text-to-optimization. *arXiv preprint arXiv:2605.21751*, 2026. 5, 9, 13

Frederick S Hillier, Gerald J Lieberman, Karl Schmedders, Molly Stephens, Mark Hillier, and Michael O'Sullivan. *Introduction to operations research*, volume 9. McGraw-Hill New York, 1995. 1

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025. 1, 5, 8

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *International Conference on Learning Representations*, 2025. 8

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, et al. AlphaOPT: Formulating optimization programs with self-improving LLM experience library. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2389–2400, 2026. 5, 9

Chengpeng Li, Mingfeng Xue, Zhenru Zhang, Jiaxi Yang, Beichen Zhang, Bowen Yu, Binyuan Hui, Junyang Lin, Xiang Wang, and Dayiheng Liu. START: Self-taught reasoner with tools. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 13512–13553, 2025. 9

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. *arXiv preprint arXiv:2602.10450*, 2026. 1, 5, 9, 13

Kuo Liang, Yuhang Lu, Jianming Mao, Shuyi Sun, Chunwei Yang, Congcong Zeng, Xiao Jin, Hanzhang Qin, Ruihao Zhu, and Chung-Piaw Teo. Large-scale optimization model auto-formulation: Harnessing LLM flexibility via structured workflow. *arXiv preprint arXiv:2601.09635*, 2026. 5, 9, 13

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye HAO. OptiTree: Hierarchical thoughts generation with tree search for LLM optimization modeling. In *Advances in Neural Information Processing Systems*, 2025. 5, 9

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*, 2025. 1, 5, 8

Pan Lu, Bowen Chen, Sheng Liu, Rahul Thapa, Joseph Boen, and James Zou. OctoTools: A multi-agent framework with extensible tools for complex reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1–86, 2026. 9

OpenAI. GPT-5.6 System Card. OpenAI Deployment Safety Hub, 2026. URL https://deploymentsafety.openai.com/gpt-5-6. 5

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. ReasoningBank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, 2026. 9

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G Patil, Ion Stoica, and Joseph E Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. 9

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. 9

Viraj Prabhu, Yutong Dai, Matthew Fernandez, Krithika Ramakrishnan, Jing Gu, Yanqi Luo, Caiming Xiong, Junnan Li, Zeyuan Chen, Ran Xu, et al. WALT: Web agents that learn tools. In *International Conference on Learning Representations*, volume 2026, pp. 55589–55607, 2026. 9

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, volume 2024, pp. 9695–9717, 2024. 9

Qwen Team. Qwen3-Max: Just scale it, September 2025. 5

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, 2023. 9

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, 2023. 9

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. 9

Dongyu Wang, Jingyu Li, Lan Zhang, Ganggang Yu, and Liang Huang. VET: Verifiable execution tracing for reliable Text-to-SQL generation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 30867–30883, 2026a. 9

Teng Wang, Wing Yin Yu, Zhenqi He, Zehua Liu, Hailei Gong, Han Wu, Xiongwei Han, Wei Shi, Ruifeng She, Fangzhou Zhu, and Tao Zhong. BPP-Search: Enhancing tree of thought reasoning for mathematical modeling problem solving. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics*, pp. 821–838, 2025. 9

Zhuohan Wang, Ziwei Zhu, Ziniu Li, Congliang Chen, Zhihang Lin, MingZhe Yang, Yizhou Han, Yufeng Lin, Angyang Gu, Xinglin Hu, et al. Certified evaluation for LLMs in optimization modeling: From graph isomorphism to formulation isomorphism. In *ICML 2026 Workshop on Combining Theory and Benchmarks: Towards A Virtuous Cycle to Understand and Guarantee Foundation Model Performance*, 2026b. 5, 13

H Paul Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 2013. 1

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024a. 9

Qinzhuo Wu, Wei Liu, Jian Luan, and Bin Wang. ToolPlanner: A tool augmented LLM for multi granularity instructions with path planning and feedback. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 18315–18339, 2024b. 9

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. 1, 5, 9

Guangrui Xie. ORPilot: A production-oriented agentic LLM-for-OR tool for optimization modeling. *arXiv preprint arXiv:2605.02728*, 2026. 5

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. 5

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-Mem: Agentic memory for LLM agents. In *Advances in Neural Information Processing Systems*, 2025. 9

Haochen Yang, Ke Zhao, Mengyuan Ma, Xingyu Lu, Xiangfeng Wang, and Hong Qian. OptSkills: Learning generalizable optimization skills from problem archetypes via cluster-based distillation. *arXiv preprint arXiv:2605.29829*, 2026. 9

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. 9

Jianzhang Zhang, Jialong Zhou, and Chuang Liu. OR-Toolformer: Modeling and solving operations research problems with tool augmented large language models. *arXiv preprint arXiv:2510.01253*, 2025. 9

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024. 9

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 19724–19731, 2024. 9

Chenyu Zhou, Tianyi Xu, Jianghao Lin, and Dongdong Ge. StepORLM: A self-evolving framework with generative process supervision for operations research language models. In *International Conference on Learning Representations*, 2026. 5, 8

A Experimental Details

A.1 Benchmark Details

We evaluate InForm on four optimization-modeling benchmarks with external instance data, which provide complementary settings for studying instance-informed formulation.

- **Large-Scale-OR** (Liang et al., 2026) contains 101 test instances from representative OR problem families, including resource allocation, transportation, facility location, assignment, and network revenue management. It emphasizes medium- and large-scale formulation tasks.
- **Bench4Opt** (Wang et al., 2026b) consists of 197 underlying LP and MILP instances, each associated with natural-language descriptions, external parameters, and reference optimization models. Its explicit model–data separation enables evaluation of formulation grounding with instance-specific information.
- **Text2Opt-Bench** (Gao et al., 2026) covers diverse template-based problems across LP, MILP, MIQP, and nonlinear optimization. Compared with Bench4Opt, it places greater emphasis on structured grounding and data binding, requiring models to correctly map large collections of external coefficients, indices, and relational information into executable formulations.
- **MIPLIB-NL** (Li et al., 2026) contains industrial-scale MILP instances reconstructed from MIPLIB 2017 with explicit problem–data separation. It provides the most challenging setting by combining large model scales with heterogeneous tables, complex index mappings, and cross-entity relations.

To enable consistent evaluation across the four benchmarks, we curate all instances into a unified problem–data separated format following MIPLIB-NL. Each instance contains a natural-language problem description, scalar parameters, external CSV tables, and table/column descriptions. We retain available information, convert structured parameters when necessary, and generate missing table descriptions using an LLM based on the original problem and data schema. We only modify data organization and references without changing the original optimization semantics or instance values. Redundant or incomplete cases are removed following benchmark-specific rules. In particular, Text2Opt-Bench samples 110 instances from the 550 canonical small-tier instances, with 10 cases selected per problem family across different data-volume levels. The final evaluation sets contain 101, 197, 110, and 221 instances respectively. Table 4 summarizes the benchmark composition and selection rules.

Table 4: Benchmark statistics, source organization, and instance selection rules.

	#Eval	#Original	Types	Organization	Selection
Large-Scale-OR	101	101 instances	LP, IP	NL description; external data; model + answer	All instances retained
Bench4Opt	197	197 problems / 394 descriptions	LP, MILP	two NL descriptions; parameter JSON; LP + solver code	Full description retained per problem
Text2Opt-Bench	110	550 templates	LP, MILP, MIQP, NLP	NL description; structured parameters; solver code + answer	10 instances per family across data-volume ranks
MIPLIB-NL	221	223 instances	MILP	NL description; parameter JSON; external CSV data table descriptions; model + solver code	Exclude 2 instances with missing data files

A.2 Implementation Details

All agentic baselines and InForm use DeepSeek-V4-Flash as the backbone model. DeepSeek models are evaluated through the official API service, while the fine-tuned baselines are deployed locally on 8 NVIDIA H20 GPUs. Generated optimization programs are executed using Gurobi 12.0.3 and COPT 8.0.1, with the same solver configuration and time limit applied across methods. We use

pass@1 evaluation, where an instance is counted as correct if the method's single submitted output produces the correct answer.

For the online Experience Memory, test instances are processed in a fixed order and the memory is updated every 32 cases. Only trajectories whose generated formulations are verified as correct are used to construct and merge new experiences. Each memory update is applied only to subsequent batches, while instances within the current batch are evaluated using the same memory snapshot, ensuring that newly acquired experiences do not affect their own evaluation. Tool definitions and retrieval settings remain fixed throughout evaluation. The main implementation and hyperparameter settings are summarized in Table 5.

Table 5: Main implementation and hyperparameter settings.

Category	Parameter	Value
LLM	Max okens	32,768
	Temperature	0
Tool Calls	Max tool turns	8
	Max calls per turn	3
Experience	Update Batches	32
	Retrieved per tool turn	1
	Disclosed per case	3
	Retrieval weights (α, β, γ)	$(0.2, 0.5, 0.3)$
	Similarity thresholds (τ_g, τ_e, τ_d)	$(0.1, 0.1, 0.1)$
Evaluation	Order seed	2026
	Time limit	600 s
	Threads	1
	Gurobi seed	0
	Error tolerance	10^{-4}

B More Experiment Results

B.1 Generalization Across Model Backbones

To examine whether the effectiveness of InForm depends on a particular LLM backbone, we conduct an additional evaluation using DouBao-Seed-2.0-Lite(Bytedance Seed, 2026) as the backbone model. We re-evaluate all agentic baselines and InForm under the same benchmark suite and evaluation protocol, replacing only the backbone while keeping the method configurations unchanged.

The results in Table 6 show that InForm maintains consistent performance improvements on DouBao-Seed-2.0-Lite. InForm achieves the highest average accuracy across the four benchmarks, improving the average Acc from 61.6% of the base model and outperforming all agentic baselines. These results suggest that the effectiveness of InForm mainly comes from its instance-conditioned reasoning framework rather than being specific to a particular LLM backbone.

B.2 More Tool-Use Analysis

Large-Scale-OR provides problem-type and size labels, allowing us to examine how tool usage adapts to different formulation characteristics. Figure 7(a) shows that tool activation patterns are closely related to problem structures. For example, RA (resource allocation) instances mainly involve entity assignment and parameter constraints, so they predominantly rely on Data Layout and Domains & Parameters tools. In contrast, TP (transportation planning) instances involve network interactions and flow structures, requiring additional analysis of relational connectivity and numerical properties; therefore, they activate more Relations & Groups and Patterns & Bounds tools.

Figure 7(b) further examines tool activation across different problem sizes. Data Layout is omitted because it is invoked for every instance. The results show that larger instances tend to require

Table 6: **The overall performance of InForm and baselines with Doubao-Seed2.0-Lite on four OR benchmarks.** ER (%) is the executable rate and Acc (%) is the Pass@1 accuracy. Avg. is the macro-average over benchmarks. Best results are highlighted in **bold** and the second-highest values are underlined.

Method	Large-Scale-OR		Bench4Opt		Text2Opt		MIPLIB-NL		Avg.	
	ER	Acc	ER	Acc	ER	Acc	ER	Acc	ER	Acc
Doubao-Seed2.0-Lite	81.2	70.3	88.9	75.2	89.8	69.8	57.7	31.0	79.4	61.6
CoE	**93.1**	77.2	94.9	77.7	89.1	68.2	38.0	40.5	78.8	65.9
OptiMUS	76.0	67.5	89.0	71.3	94.1	67.2	65.8	36.2	81.2	60.5
OptiTree	87.1	78.2	95.9	**81.7**	87.3	73.6	43.4	29.0	78.4	65.6
AlphaOPT	88.1	79.2	90.8	74.6	86.5	66.5	39.1	28.2	76.2	62.1
LEAN-LLM-OPT	86.2	79.2	86.8	72.0	88.4	67.5	39.6	28.2	75.2	61.7
ORPilot	83.2	65.5	94.4	79.2	86.5	67.2	47.2	34.1	77.8	61.5
InForm	91.1	**81.2**	**97.0**	76.7	**98.2**	**81.8**	**83.7**	**44.3**	**92.5**	**71.0**

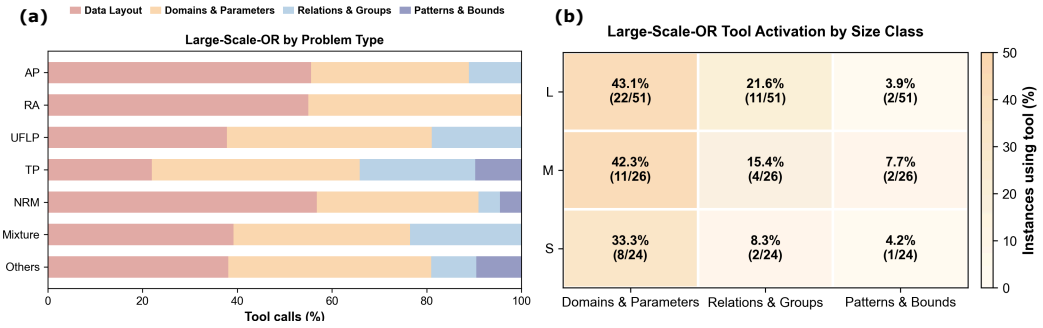


Figure 7: **More tool-use analysis on Large-Scale-OR.** (a) Distribution of tool calls across problem types. (b) Fraction of instances activating each non-layout tool category across problem-size classes.

richer combinations of instance analysis tools: the activation rates of multiple evidence-oriented tools increase from small to large instances, indicating that large-scale problems more frequently require additional structural and numerical analysis beyond basic data understanding. Meanwhile, the specific tool combinations remain instance-dependent, as different problem structures may require different types of evidence during formulation.

B.3 Parameter Sensitivity Analysis

We analyze the sensitivity of two key hyperparameters in InForm: the maximum number of tool-call turns and the online experience update batch size. Experiments are conducted on MIPLIB-NL, with the default settings of 8 tool-call turns and a batch size of 32 used in the main experiments.

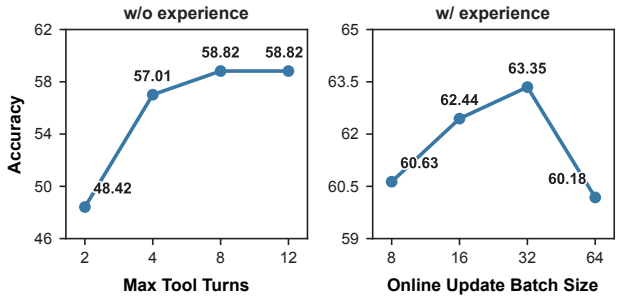


Figure 8: **Parameter sensitivity analysis on MIPLIB-NL.** Left: the maximum number of tool-call turns per instance. Right: the online experience update batch size.

Figure 8 summarizes the results. The performance improves as the tool-call budget increases from 2 to 8 turns and saturates afterwards, suggesting that the default budget is sufficient for progressive evidence acquisition. For experience memory, a batch size of 32 achieves the best performance, while both smaller and larger update intervals lead to lower accuracy. We therefore use these settings in all experiments.

Case: Tree-structured water supply (MIPLIB-NL `h80x6320d`)

Question. Build a pipeline tree over 80 nodes to deliver 384 units of water from a single source to demand nodes. Each node has at most one incoming pipeline; zero-demand nodes may serve as transit points. Minimize construction plus flow cost.

Data. `nodes.csv`: net supply/demand; `edges.csv`: candidate pipelines, fixed costs, and per-unit flow costs.

(a) Baseline: GPT-5.6-Sol

1. Model flow and pipeline selection

Use binary build variables x_{ij} and continuous flows f_{ij}. Enforce flow balance, $f_{ij} \le 384x_{ij}$, at most one incoming pipeline per node, and none at the source.

2. Explicitly encode acyclicity

Introduce an integer depth u_i for every node. A selected arc must increase depth:

$$u_j \ge u_i + 1 - 80(1 - x_{ij}).$$

Additional parent-link inequalities require a non-source node to have an incoming pipeline before it can send water onward.

3. Solve with the extra structure

80 integer depth variables, 6,478 depth inequalities, and 6,399 parent-link inequalities explicitly enforce the tree structure.

The tree is enforced explicitly, without exploiting instance properties to remove the extra encoding.

Optimal cost: 6382.09905
Solver time: 4.78 s

(b) InForm: Instance-Informed Formulation

1. Check the decisive instance properties

analyze_parameter_pattern (cost signs)
Return: fixed and per-unit flow costs are positive for all 3,239 records.

derive_aggregate_envelope (net supply)
Return: $\sum_i b_i = 0$; total supply matches demand.

2. Use the properties to simplify

With one source, flow balance and the incoming-edge limit connect demand nodes to the source. A disconnected cycle serves no demand and adds positive cost.

An optimal solution therefore contains no cycle. Explicit depth encoding is unnecessary for optimality.

3. Keep a smaller formulation

Keep flow balance and pipeline-selection constraints. **No depth variables or depth/parent-link inequalities.** Also remove 79 observed self-loops and strengthen demand-node incoming-edge bounds.

Optimal cost: 6382.09905
Solver time: 1.95 s (2.45× faster)

Figure 9: **Efficiency case study on MIPLIB-NL h80x6320d.** Instance evidence identifies when explicit cycle-elimination constraints are unnecessary, enabling a smaller formulation with the same optimal objective and faster solving.

B.4 Inference Efficiency Analysis

We further analyze the inference cost of InForm by measuring the average execution time and token consumption per instance across four benchmarks. All agentic methods use DeepSeek-V4-Flash as the backbone, and the reported values are averaged over all evaluated instances. As shown in Table 7, InForm introduces additional reasoning steps for progressive tool use and experience retrieval while maintaining a moderate inference overhead. Compared with other agentic approaches, InForm requires substantially fewer tokens than most baselines and achieves a lower average inference time than most of them. These results show that the accuracy improvements of InForm do not stem from substantially increased inference cost. Compared with agentic baselines that rely on longer reasoning trajectories, additional knowledge retrieval, or external memory injection, InForm achieves more effective data-grounded reasoning through targeted tool interactions with lower inference overhead.

Table 7: Average inference cost per instance across four benchmarks.

Method	Time (s)	Tokens
DeepSeek-V4-Flash	41.7	6.1K
CoE	291.8	45.9K
OptiMUS	432.4	108.2K
OptiTree	145.9	24.5K
AlphaOPT	250.0	49.7K
LEAN-LLM-OPT	127.0	37.4K
ORPilot	161.5	41.1K
InForm	104.5	36.3K

B.5 Case Study: Instance-Informed Formulation for Solver Efficiency

Case in Figure 9 illustrates how instance evidence improves solver efficiency by enabling instance-specific formulation simplification. The problem requires constructing a minimum-cost tree-structured water supply network. A text-driven formulation can correctly enforce the tree structure using explicit cycle-elimination constraints, but this generic encoding introduces additional variables and structural constraints.

Case: Sparse coverage planning (MIPLIB-NL `stein45inf`)

Question. A security center must activate the fewest drone hangars to cover three surveillance locations. Each location must be covered by at least one of its listed hangars; for example, $A1$ can be covered only by `zz03`, `zz04`, or `zz06`.

Data. `S1.csv`: 3 locations, 7 hangars, and 9 listed location–hangar coverage pairs.

(a) w/o Memory: Cartesian expansion

1. Treat the relation as two independent sets

Define locations $L = \{A1, A2, A3\}$ and hangars $H = \{\texttt{zz01}, \ldots, \texttt{zz08}\}$. The listed pairs are not retained as the coverage index.

2. Expand coverage to all pairs

For every location, use every hangar as eligible:

$$\sum_{h \in H} x_h \geq 1 \qquad \forall \ell \in L.$$

The model silently replaces the observed relation by $L \times H$.

3. Solve the expanded model

One activated hangar appears sufficient. The model selects `zz08`, even though it is listed only for $A3$ and cannot cover $A1$ or $A2$.

Reported objective: 1
Coverage status: Incorrect

(b) w/ Memory: Sparse-support restriction

1. Observe sparse coverage support

analyze_relation_support (location–hangar)
Return: 3 locations $\times$ 7 hangars, but only **9 observed pairs out of 21**; every location has degree 3.

2. Use the retrieved Experience

Experience: use observed sparse support as the relation index; do not expand it to the Cartesian product.

Construct $H_\ell = \{h : (\ell, h) \in E\}$ from the listed pairs for each location ℓ.

3. Constrain only listed coverage

$$\sum_{h \in H_\ell} x_h \geq 1 \qquad \forall \ell \in L.$$

Selecting `zz04` and `zz05` covers all three locations using only admissible pairs.

Optimal objective: 2
Coverage status: Correct

Figure 10: **Experience case study on MIPLIB-NL stein45inf.** Retrieved experience guides the use of sparse relation evidence, preventing invalid Cartesian expansion and enabling correct formulation.

InForm analyzes instance properties before formulation and identifies that all construction and flow costs are positive and that supply matches demand. Combined with the single-source structure, these properties imply that disconnected cycles cannot appear in an optimal solution. Therefore, InForm removes unnecessary depth-based cycle-elimination constraints while preserving the optimal objective, achieving a smaller formulation with $2.45\times$ faster solver time.

B.6 Case Study: Experience-Guided Instance Evidence Utilization

Case in Figure 10 illustrates how experience memory helps translate observed instance evidence into formulation decisions. The problem requires selecting the minimum number of hangars to cover all surveillance locations, where each location can only be covered by its listed hangars. The key challenge is determining whether the observed location–hangar pairs represent a sparse relation or can be expanded into a Cartesian product.

Without memory, the model fails to preserve the sparse relation structure and incorrectly treats all hangars as candidates for every location, resulting in invalid coverage constraints. InForm retrieves a matching experience after obtaining relation-support evidence, guiding the model to construct constraints only over observed pairs. This preserves the instance-specific relation structure and produces the correct optimal solution.

C Methodology Details

C.1 Progressive Tool-Integrated Formulation Protocol

Algorithm 1 summarizes the inference procedure of InForm, where instance evidence and experience guidance are progressively incorporated during formulation reasoning.

Algorithm 1 Progressive Tool-Integrated Formulation Reasoning

Require: Problem description P, instance data D, tool set $\mathcal{T}$, maximum tool turns K
Ensure: Formulation M_D

```
Initialize formulation state S ← {P}
Execute Data Layout tool and initialize S with layout observation
for t = 1, ..., K do
    Generate formulation decision z_t conditioned on S
    if additional evidence is required for z_t then
        Select tool k and query q using Tool Cards
        Obtain evidence observation e_t ← T_k(q, D)
        Retrieve matching experience guidance g_t if available
        Update S ← S ∪ {e_t, g_t}
    end if
    Continue formulation reasoning with updated S
    if M_D is complete then return M_D
    end if
end for
```

Table 8: Overview of instance analysis tools in InForm.

Category	Tool	Function	Output Evidence
Data Layout	`establish_instance_layout`	Map tables and columns to semantic entities	Entity–table mapping, key columns, and parameter columns
Domains & Parameters	`analyze_entity_domain`	Identify effective entity domains	Entity sets, cardinality, and active elements
	`analyze_subset_membership`	Analyze subset structures	Subset definitions and membership relations
	`analyze_parameter_binding`	Verify parameter–index correspondence	Parameter support and valid indexed parameters
Relations & Groups	`analyze_relation_support`	Identify valid relation tuples	Relation support sets
	`analyze_key_alignment`	Verify cross-table consistency	Entity mappings
	`analyze_group_structure`	Identify grouping structures	Groups, partitions, and coupling structures
Patterns & Bounds	`analyze_parameter_pattern`	Detect numerical patterns	Parameter patterns
	`analyze_ordered_structure`	Identify ordered structures	Ordering relations
	`derive_aggregate_envelope`	Derive valid aggregate bounds	Bounds and aggregate envelopes

C.2 Instance Analysis Tools Details

We implement four categories of instance analysis tools to extract formulation-relevant evidence from structured instance data. Each tool deterministically analyzes instance tables and returns typed evidence for formulation decisions. Data Layout establishes semantic mappings from raw tables, while Domains & Parameters, Relations & Groups, and Patterns & Bounds respectively analyze entity and parameter support, relational and grouping structures, and numerical patterns and bounds. During progressive reasoning, the LLM selects tools according to the current formulation decision, while evidence extraction is handled by deterministic modules.

Table 8 provides an overview of all instance analysis tools, including their categories, names, functional descriptions, and output evidence. Detailed specifications of each tool, including input requirements, deterministic execution procedures, output formats, and evidence processing strategies, are presented through the corresponding Tool Cards.

Tool Card: `establish_instance_layouts`

Description

Establishes a validated parsing contract for every physical CSV before instance-conditioned formulation. It identifies table layouts and selectors for entity keys, relation endpoints, parameter values, grouping fields, and matrix axes, so later evidence uses stable semantic roles.

Input

A complete `table_catalog` and one binding for every `table_id`: `layout_type`, `header_mode`, and semantic-role selectors. Empty tables require an empty role map.

Execution

1. Scan all physical rows for width variation, emptiness, and header evidence.
2. Validate table coverage and each binding against layout and header contracts.
3. Profile the values covered by every bound semantic role.
4. Freeze the validated layout registry for all later tools.

Output

A layout observation containing each table's `layout_id`, path, layout type, header mode, shape, and semantic-role-to-selector map. It is a parsing contract rather than an ordinary mathematical fact.

Evidence Interpretation

The registry establishes which physical values can represent mathematical keys, tuple endpoints, parameter coordinates, and values. Later observations can therefore be interpreted over named domains and supports rather than ambiguous raw columns.

Tool Card: `analyze_entity_domain`

Description

Extracts the observed members and identifier structure of one entity domain. It supports decisions about whether an index is a consecutive integer interval, an irregular integer set, or a label set.

Input

A modeling `anchor`, one frozen `layout_id`, and its `entity_role`.

Execution

1. Resolve the entity role through the frozen layout.
2. Extract nonempty distinct keys and count records with empty keys.
3. Classify integer versus label identifiers; for integers, compute the span and gaps.
4. Render the domain signature and any missing-key fact.

Output

One fact describing a consecutive integer span, a nonconsecutive integer domain with gap count, or a label domain with cardinality; an optional second fact reports empty entity keys.

Evidence Interpretation

The observation establishes the empirical carrier set D and distinguishes $D = \{a, \ldots, b\}$ from an irregular subset or label set. Missing-key evidence identifies records without a valid coordinate in D.

Tool Card: `analyze_subset_membership`

Description

Determines an explicit or condition-filtered eligible, disabled, or special subset relative to a parent domain.

Input

A modeling `anchor`; `membership_mode` (`explicit_membership` or `filtered_membership`); a parent `(layout_id, role)`; membership layout and entity role; and a condition in filtered mode.

Execution

1. Resolve the parent and membership roles.
2. Extract explicit members or apply the declared condition to membership records.
3. Compare the resulting subset with its parent, including out-of-domain members.
4. Render the exact set relation and cardinalities.

Output

One fact identifying $S = D$ or $S \subset D$ with cardinalities; if present, a second fact reports the number of members in $S \setminus D$.

Evidence Interpretation

The relation supplies an evidence-backed scope S relative to D. Equality identifies a nonrestrictive condition, while a proper subset or out-of-domain result identifies a membership restriction or domain inconsistency.

Tool Card: `analyze_relation_support`

Description

Extracts the observed support of binary-to-four-ary pairs, arcs, transitions, or tuples relative to declared endpoint domains.

Input

A modeling `anchor`; relation `layout_id`; ordered `endpoint_roles`; and one aligned domain source `(layout_id, role)` for each endpoint.

Execution

1. Resolve relation roles and endpoint domains from frozen layouts.
2. Extract complete tuples, deduplicate them, and separate out-of-domain tuples.
3. Compare in-domain support exactly with the endpoint Cartesian product.
4. Profile duplicates, unused endpoints, and degrees; render at most one exceptional fact.

Output

A fact asserting $R = \prod_k D_k$, $R \subsetneq \prod_k D_k$, or out-of-domain endpoints, with exact tuple counts. A second fact may report duplicates, unused members, or degree range.

Evidence Interpretation

The observation identifies the empirical relation R and tests its equality with the full Cartesian product. A proper subset distinguishes sparse support, while endpoint mismatches identify a consistency issue in the declared domains.

Tool Card: `analyze_key_alignment`

Description

Checks whether two data sources share the same keys and with what multiplicity. It supports decisions that depend on a cross-table correspondence.

Input

A modeling `anchor`, plus `left` and `right` sources. Each source specifies a frozen `layout_id` and one to four ordered key roles.

Execution

1. Resolve both key sources and retain complete ordered keys.
2. Extract distinct keys and per-key row multiplicities.
3. Compute matching and unmatched keys.
4. Classify totality and one-to-one, one-to-many, many-to-one, or many-to-many multiplicity.

Output

One mapping fact reporting alignment type and counts of matched and unmatched keys.

Evidence Interpretation

The output characterizes the correspondence between two index spaces. Total single-valued alignment supports a function-like association; partiality or multiplicity shows that a simple one-record-per-key binding is not established.

Tool Card: `analyze_parameter_binding`

Description

Verifies whether one or more keyed parameter fields are complete and single-valued on an expected observed support.

Input

A modeling `anchor`; parameter `layout_id`; ordered `key_roles` and `value_roles`; and an `expected_support` whose roles have the same arity and intended coordinate order as the keys.

Execution

1. Validate role identities, disjoint key/value roles, and support arity.
2. Extract expected keys and independently index every nonempty value role.
3. Test per-role coverage, uniqueness, unexpected keys, and numeric ranges.
4. Summarize role-specific availability and jointly complete keys.

Output

One binding fact reporting, per value role, defined, missing, and conflicting expected keys and numeric range; it also reports joint and partial completeness. Unexpected keys affect the completeness classification but are not separately rendered.

Evidence Interpretation

The observation tests whether a parameter behaves as a single-valued map $p : K \to \mathbb{R}$ on support K. Role-specific coverage preserves the distinction between multiple sparse parameter fields.

Tool Card: `analyze_group_structure`

Description

Classifies how group memberships cover an entity domain, distinguishing a partition, overlapping cover, partial cover, and out-of-domain membership.

Input

A modeling `anchor`; membership `layout_id`; ordered `group_roles`; `member_role`; and the parent `member_domain` as a frozen `(layout_id, role)` reference.

Execution

1. Resolve group and member roles and extract complete membership records.
2. Deduplicate member sets within groups and extract the parent domain.
3. Compute coverage, overlap, out-of-domain members, and group-size extrema.
4. Classify and render the group system.

Output

One group-system fact with group count, coverage type, and relevant group-size evidence.

Evidence Interpretation

The result characterizes $\{G_h\}$ relative to D. It identifies whether membership is disjoint and exhaustive or instead contains overlap, incomplete coverage, or domain mismatch.

Tool Card: `analyze_parameter_pattern`

Description

Tests one declared exact difference, ratio, sign, joint-tier, or within-record monotonic pattern among parameter roles.

Input

A modeling `anchor`; frozen `layout_id`; two to six `parameter_roles`; and `pattern_kind` in {`difference`, `ratio`, `sign`, `joint_tiers`, `monotonic`}.

Execution

1. Resolve roles and align jointly complete records.
2. Profile each role's marginal numeric coverage, range, and signs.
3. Run only the fixed detector selected by `pattern_kind`.
4. Render the exact pattern or an explicit no-joint-observation fact.

Output

One fact describing a constant difference or ratio, sign/tier structure, monotonicity result, or number of distinct pattern values; absent joint records are reported explicitly.

Evidence Interpretation

The observation converts aligned records into a checkable predicate such as $p - q = c$ or $p/q = c$. A multi-valued or unobserved result prevents treating the relation as a structural identity.

Tool Card: `analyze_ordered_structure`

Description

Analyzes continuity, repeated blocks, or monotonicity along an ordered field, globally or within declared groups.

Input

A modeling `anchor`; `layout_id`; `order_role`; optional `value_roles`; `grouping_mode`; optional `group_roles`; and `structure_kind` in {`continuity`, `repeated_block`, `monotonic`}.

Execution

1. Validate mode-specific roles and resolve order, value, and group roles.
2. Construct global or groupwise sequences and sort by the order role.
3. Run the selected continuity, repeated-block, or monotonicity detector.
4. Render gaps, blocks, periods, or violation counts as applicable.

Output

One fact describing continuous ranges, groups or segments with gaps, a repeated period, or monotonicity violations.

Evidence Interpretation

The result establishes the observed ordered support rather than presuming a complete axis. Gaps delimit segments, while repeated-block and monotonicity predicates characterize the sequence relation.

Tool Card: `derive_aggregate_envelope`

Description

Computes an observed aggregate lower or upper envelope over a declared global or role-grouped scope, without prescribing a modeling constant.

Input

A modeling `anchor`; `layout_id`; `value_role`; `scope_mode` in {`global`, `by_group`, `over_relation`}; `scope_roles`; `operation`; and `envelope`.

Execution

1. Resolve value and scope roles; require a scope role outside global mode.
2. Construct aggregation units by scope-role tuple or as one global unit.
3. Aggregate complete values and compute the range across all units.
4. Return the requested envelope, attaining-unit count, or no-observation result.

Output

One fact reporting the aggregate range, requested lower or upper envelope, and number of attaining units; empty scopes return a no-aggregate-observation fact.

Evidence Interpretation

The output establishes $\min_{u \in U} A(u)$ or $\max_{u \in U} A(u)$ for an explicit unit set U. It provides numerical evidence while remaining distinct from any later choice of a modeling bound.

C.3 Prompt Templates

The following templates summarize the roles of the principal prompts. The zero-shot prompt directly produces a formulation and implementation. InForm separates its initial formulation prompts from tool observations and optional retrieved experiences, while the final two prompts construct and consolidate the experience memory.

Zero-shot Baseline Formulation Prompt

Role and objective

You are a professional mathematical optimization expert. Formulate the mathematical model and produce one complete, standalone Python/Gurobi implementation for the supplied instance.

Instance input

1. **Problem description:** *[Problem Description]*
2. **Parameters:** *[Parameters]*
3. **Data description:** *[Data Files and Schemas]*
4. **Base path:** *[Case Working Directory]*

Modeling and implementation requirements

Follow the stated business semantics, variable domains, data schema, labels, keys, joins, and sparse tuple supports. Read every CSV from its exact relative path under the supplied base path; do not guess filenames, use absolute paths, or prefix paths with `./`.

Use *gurobipy*, name the model *model*, and write top-level code. After *model.optimize()*, print exactly *Just print the best solution:* {*model.objVal*} when a solution exists, and *No Best Solution* otherwise.

Output

Return exactly one complete `<model>` block followed by one complete `<python>` block, with no additional prose.

InForm Formulation System Prompt

Role and objective

You are an expert mathematical optimization modeler. Produce one correct, efficient, standalone Python/Gurobi solution for the supplied instance.

Progressive formulation

Proceed through problem interpretation, sets and parameters, variables, objective, constraints, and final implementation. Follow one formulation path and finalize each modeling choice only after the relevant textual and instance evidence has been considered.

Tool-use policy

First establish the layout of every data table. Invoke an instance-analysis tool only for a named, unresolved modeling decision and only when its result may change a set, support, parameter, coefficient, scope, sequence, or bound. Treat tool observations as factual evidence rather than instructions or post-hoc validation. Integrate all observations from the current turn before deciding whether another call is needed.

Use the fewest calls required. The limits of *[Maximum Tool Turns]* turns and *[Maximum Calls per Turn]* calls are safety ceilings, not targets.

Experience policy

After a post-layout tool turn, the host may disclose at most *[Maximum Experience Disclosures]* relevant experiences. Each experience provides an *evidence pattern* and bounded local *guidance*. Apply it only when it matches the current evidence; it never overrides the problem or observations.

Output

Return exactly one complete `<model>` block and one complete `<python>` block.

InForm Formulation User Prompt

Instance

Problem *[Problem Description]*

Parameters *[Parameters]*

Compact table catalog *[Table Descriptions, Paths, Samples, and Physical Profiles]*

Case working directory *[Working Directory]*

Data and formulation contract

Preserve the problem's business meaning, sparse supports, keys, units, signs, constraint directions, and variable domains. Do not invent data, joins, defaults, or constraints.

Use the first layout observation as the frozen parsing contract. Respect the reported header status and raggedness; represent an implicit column axis by position rather than by values from the first data row. In later calls, refer only to the frozen layout and its semantic roles.

Implementation contract

Read every CSV from its exact relative path. Preserve identifiers as strings, guard against empty or ragged rows, and construct dictionaries and variable supports only from observed keys and relations.

Use only the Python standard library and *gurobipy*. Name the model *model*, use valid Gurobi expressions and constraint APIs, call *model.optimize()* exactly once, and print the required objective or *No Best Solution*.

Tool Observation and Retrieved Experience Follow-up

Instance-analysis tool observation

Decision anchor *[Pending Local Modeling Decision]*

Observation ID *[Observation Identifier]*

Summary *[Compact Structured Summary]*

Facts *[Formulation-relevant Facts Rendered by the Tool]*

For the mandatory first layout call, the observation instead returns the frozen table layouts and semantic-role bindings.

Retrieved experience follow-up (optional)

Matched evidence pattern *[Transferable Pattern Abstracted from the Tool Observation]*

Local modeling guidance *[Bounded Formulation Operation Supported by that Pattern]*

Experience Abstraction Prompt

Input

[Successful Formulation Traces]

Selection rule

Extract at most one reusable observation-to-formulation experience from each successful record. Retain an item only when one post-layout observation supplied decision-changing evidence that was used in a correct local modeling decision. If several observations qualify, keep the most consequential and transferable one; merge semantic duplicates within the batch.

Experience schema

Record the source record and observation for traceability. Describe the transferable *problem pattern*, *evidence source*, *evidence pattern*, *decision context*, and the bounded local *guidance*.

Validity constraints

The evidence pattern and guidance must be supported by that single observation. Do not combine evidence, reproduce a complete formulation, or retain dataset names, instance identifiers, physical selectors, exact values, final answers, or reasoning prose. Return one structured `<experiences>` object; the item list may be empty.

Experience Consolidation Prompt

Input

[Existing Experiences and New Batch Candidates]

Maximum retained items *[Maximum Output Items]*

Consolidation rule

All candidates share the same evidence source. Treat business nouns, table and role names, entity counts, exact ranges, and wording as surface variation. Merge candidates whenever the same qualitative evidence condition supports the same local modeling operation; keep them separate only when the condition or operation differs materially.

Output constraints

Discard generic, unsupported, or cross-evidence claims. When further reduction is needed, prefer the most transferable and decision-relevant experiences. Preserve only the problem pattern, evidence source, evidence pattern, decision context, and bounded local guidance. Return one structured `<experiences>` object within the requested item limit.