# Peering Inside the Black Box: Uncovering LLM Errors in Optimization Modelling through Component-Level Evaluation

Dania Refai<sup>a,\*</sup>, Moataz Ahmed<sup>a,b</sup>

<sup>a</sup>Information and Computer Science Department, KFUPM, Dhahran 31261, Saudi Arabia

<sup>b</sup>SDAIA-KFUPM Joint Research Center for Artificial Intelligence, King Fahd University of Petroleum & Minerals, Dhahran, 31261, Saudi Arabia

## Abstract

Large language models (LLMs) are increasingly used to convert natural language descriptions into mathematical optimization formulations. Current evaluations often treat formulations as a whole, relying on coarse metrics like solution accuracy or runtime, which obscure structural or numerical errors. In this study, we present a comprehensive, component-level evaluation framework for LLM-generated formulations. Beyond the conventional optimality gap, our framework introduces metrics such as precision and recall for decision variables and constraints, constraint and objective root mean squared error (RMSE), and efficiency indicators based on token usage and latency. We evaluate GPT-5, LLaMA 3.1 Instruct, and DeepSeek Math across optimization problems of varying complexity under six prompting strategies. Results show that GPT-5 consistently outperforms other models, with chain-of-thought, self-consistency, and modular prompting proving most effective.

<sup>\*</sup>Corresponding author: Dania Refai (E-mail: Dania.Refai@hotmail.com)

Analysis indicates that solver performance depends primarily on high constraint recall and low constraint RMSE, which together ensure structural correctness and solution reliability. Constraint precision and decision variable metrics play secondary roles, while concise outputs enhance computational efficiency. These findings highlight three principles for NLP-to-optimization modeling: (i) Complete constraint coverage prevents violations, (ii) minimizing constraint RMSE ensures solver-level accuracy, and (iii) concise outputs improve computational efficiency. The proposed framework establishes a foundation for fine-grained, diagnostic evaluation of LLMs in optimization modeling.

Keywords: Large language models, optimization, optimization modeling, prompt engineering, linear programming, combinatorial optimization, fine-tuning, in-context learning.

## 1. Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across diverse domains in recent years, including natural language processing (NLP), computer vision, software engineering, biomedical research, and scientific reasoning. Their ability to understand, generate, and reason over human language has enabled state-of-the-art performance in tasks such as machine translation, summarization, question answering, and code generation. Beyond these traditional applications, LLMs have emerged as promising tools for bridging the gap between natural language instructions and complex computational tasks [1], with several recent studies highlighting their use in sentiment analysis [2, 3], retrieval-augmented reasoning [4],

scientific data modeling [5], and autonomous task benchmarking [6].

One domain where this potential is particularly valuable is mathematical optimization. Formulating optimization problems from real-world descriptions requires substantial expertise, careful reasoning, and precise encoding of decision variables, constraints, and objective functions. The process is time-consuming and error-prone, often requiring multiple iterations between domain experts and optimization engineers. Automating this translation, from natural language descriptions to solver-ready mathematical formulations, has the potential to democratize access to optimization tools and significantly reduce modeling overhead [7, 8, 9, 39].

Several recent studies have begun to explore this direction. Early approaches such as OPTIMUS [10], LM4OPT [11], and AutoFormulation [12] demonstrated that LLMs can generate mathematical formulations directly from textual descriptions. Other works, including OptiBench [13], ORMind [14], and Chain-of-Experts [15], have investigated advanced prompting strategies, fine-tuning, retrieval-augmented generation (RAG), and multiagent pipelines to improve accuracy and solver readiness. Benchmarks such as NL4Opt, ComplexOR, and StarJob have also been introduced to evaluate model performance across a range of problem domains [16, 13, 17]. Collectively, this growing literature underscores the promise of LLMs for optimization modeling, but also reveals important gaps.

Despite these advances, several challenges remain. First, most evaluations continue to rely on proprietary models such as GPT-3.5 and GPT-4, with limited exploration of the latest generation of LLMs (e.g., GPT-5). Second, prompting strategies are often investigated in narrow contexts, with little

systematic analysis of handcrafted prompts and their effectiveness across problems of varying complexity. Third, existing evaluation practices remain largely coarse-grained, focusing on global indicators such as solution accuracy, feasibility, or runtime, while overlooking structural errors in the formulation and interactions between different metrics. These limitations point to the need for a diagnostic evaluation framework that can reveal not only whether a formulation works, but also where and why errors occur.

To address these gaps, we propose a new evaluation framework for LLM-generated optimization formulations. Unlike prior work that treats formulations as a whole, our approach evaluates them component by component, introducing metrics for decision variables (Precision and recall), constraints (Precision, recall, and constraint root mean squared error (Cons-RMSE)), and objective function root mean squared error (obj-RMSE). We complement these with solution-level quality (Optimality gap) and efficiency measures (latency and token usage). Together, these metrics provide a fine-grained view of model behavior, enabling the identification of structural weaknesses, numerical inaccuracies, and efficiency bottlenecks. We validate this framework through extensive experiments on four benchmark optimization problems of increasing complexity, applying six prompting strategies to three state-of-the-art LLMs: GPT-5, LLaMA 3.1, and DeepSeek.

The main contributions of this paper can be summarized as follows:

• We design a comprehensive evaluation framework for NLP to Optimization tasks, introducing a new set of metrics that assess formulations component by component. The framework evaluates (i) Solution-level quality (Optimality gap), (ii) component-level accuracy (Precision

and recall for decision variables and constraints, obj-RMSE and const-RMSE), and (iii) efficiency (Latency and token usage). By isolating individual components, our metrics precisely identify the source of errors within generated formulations, enabling a systematic analysis of the relationships between metrics and formulation correctness.

- We design six prompting strategies: Basic, Act-As-Expert, Chain-of-Thought, Program-of-Thought, Self-Consistency, and Modular Prompting, to investigate their effect on the quality and reliability of optimization formulations.
- We perform an extensive comparative evaluation of state-of-the-art LLMs, including GPT-5, DeepSeek, and LLaMA, across optimization problems of varying complexity (Easy, medium, and hard).
- We reveal how prompts, model reasoning, and metric interactions affect
  performance, identifying error patterns and trade-offs between structural accuracy, solver quality, and efficiency, providing guidance for
  automated optimization modeling and LLM applications in decisionmaking.

The remainder of this paper is organized as follows. Section 2 reviews related work on the use of LLMs for optimization modeling. Section 3 presents the evaluation framework, covering both existing metrics from the literature and our proposed extensions. Section 4 describes the experimental setup, including the selected optimization problems, the prompting strategies employed, and the LLMs evaluated. Section 5 reports the experimental results, beginning with problem-specific performance analyses and followed by cross-problem insights and a broader discussion of findings. Finally, Section 6

concludes the paper and outlines directions for future research.

## 2. Literature Review

LLMs have recently shown strong potential in automating optimization problem formulation, translating natural language descriptions into mathematical models with explicit decision variables, constraints, and objective functions. Prior research has explored their reliability in generating solverready formulations across diverse optimization domains [7, 8, 9, 18]. Existing approaches fall into two main categories. (i) In-context learning, which guides models through carefully designed prompts, zero- or few-shot examples, and advanced prompting strategies such as chain-of-thought (CoT) and tree-of-thought (ToT) reasoning [19, 20, 21]. This paradigm emphasizes adaptability without parameter updates and is widely used for both linear and combinatorial problems. (ii) Fine-tuning, often implemented with parameter-efficient techniques such as low-rank adaptation (LoRA), specializes models on domain-specific corpora [22, 23, 24]. Some studies further integrate fine-tuning with RAG to inject domain knowledge during inference [25, 26]. Together, these approaches form the methodological backbone of current research on LLM-based optimization modeling.

Applications span multiple problem domains, covering both classical operations research tasks and emerging real-world challenges. Linear programming (LP) is frequently used as a benchmark because of its simplicity and established solver ecosystem [27, 28]. More commonly, research targets combinatorial optimization, including integer and mixed-integer linear programming (ILP, MILP), which underpin applications in scheduling, routing, and resource allocation [18, 22, 23, 29]. A smaller but growing body

of work explores hybrid formulations that combine linear and combinatorial elements [24, 30]. Overall, combinatorial optimization dominates the literature, providing a rich testbed for evaluating LLM reasoning and formulation capabilities [9, 31].

To benchmark LLM performance, researchers have introduced diverse datasets spanning synthetic and real-world optimization problems. Commonly used benchmarks include NL4Opt, LP word problems (LPWP), Mamo, IndustryOR, NLP4LP, ComplexOR, and OptiBench, as well as specialized corpora such as StarJob, ReSocratic, and Cycle Share [7, 8, 9, 22, 30, 32]. These datasets vary in scale and complexity, from simple word problems to industrial-grade tasks, supporting systematic evaluation of LLM formulation capability. Among them, NL4Opt remains the most widely adopted due to its accessibility and diversity [7, 8, 30]. For solver evaluation, generated formulations are typically tested using CPLEX, SCIP, COPT, PuLP, and OR-Tools, with Gurobi emerging as the dominant choice owing to its robustness and broad support for linear and combinatorial problems [9, 18, 29, 33].

A broad range of LLMs has been applied to optimization tasks, spanning proprietary and open-source models. The most frequently studied include GPT-4, GPT-3.5, and LLaMA, which dominate the literature due to their strong reasoning ability and accessibility [7, 9, 22, 23]. Other notable models, Mistral, DeepSeek, Qwen, PaLM, Zephyr, Phi, Mixtral, Falcon, and Claude, have been explored in specialized contexts or paired with advanced prompting strategies [24, 30, 34, 35]. Despite increasing interest in open-source alternatives, proprietary GPT models continue to dominate comparative studies [14, 33]. This trend highlights their superior performance yet

raises reproducibility and transparency concerns, prompting a growing shift toward open-source and fine-tuned variants [36, 37].

The effectiveness of LLMs in optimization formulation largely depends on prompting and adaptation strategies. Most studies employ in-context learning, using carefully designed prompts, zero- or few-shot examples, and reasoning strategies such as CoT and ToT to guide model outputs [19, 20, 21]. Others leverage self-consistency and multi-agent prompting, where multiple LLMs collaborate to refine or validate formulations [18, 37, 38]. Beyond prompting, RAG has been explored to inject domain-specific knowledge [25, 26], while fine-tuning with parameter-efficient methods such as LoRA adapts models for optimization modeling tasks [22, 23, 24].

Evaluation practices in recent literature employ diverse metrics to assess LLM-generated optimization formulations. Common measures include solution quality (e.g., optimality gap), surface-form accuracy (e.g., F1), build-ability and robustness (e.g., compilation accuracy), efficiency (e.g., solving time), mathematical fidelity (e.g., integrity gap), domain utility (e.g., utility improvement), and human-centered evaluation. Although widely adopted, these metrics have notable limitations when used in isolation, as they often fail to capture structural correctness or reveal the sources of formulation errors. Our study directly addresses these gaps; hence, a detailed discussion of existing metrics, their shortcomings, and our proposed evaluation framework is provided in Section 3.

In summary, while prior studies have demonstrated the ability of LLMs to generate optimization formulations, several important gaps remain. First, most evaluations continue to rely on proprietary models such as GPT-3.5 and

GPT-5 or systematically comparing them with strong open-source alternatives like Llama and DeepSeek. Second, prompting strategies are often tested in narrow settings, with little systematic analysis of handcrafted prompts or their behavior across optimization problems of different complexity levels. Third, while a wide range of evaluation metrics has been introduced, these metrics remain largely coarse-grained, focusing on overall solution accuracy, feasibility, or runtime, without diagnosing where errors occur within the formulation or how different metrics interact. These limitations highlight the need for a comprehensive evaluation framework that can assess optimization formulations at a finer granularity, account for both structural and numerical correctness, and evaluate models more rigorously across prompts and problems of varying complexity. Addressing this gap is the central objective of our work, and we provide a detailed discussion of evaluation metrics, their shortcomings, and our proposed framework in Section 3.

#### 3. Evaluation Metrics

The evaluation of LLM-generated optimization formulations is a challenging task because a formulation is not a monolithic object but a structured composition of **decision variables**, **constraints**, and an **objective function**. Each of these components must be generated correctly for the overall formulation to be valid and useful. Accordingly, evaluation metrics should ideally capture correctness at both the *solution level* and the *component level*. In this section, we review existing metrics, highlight their limitations, and present our proposed framework that addresses these gaps.

#### 3.1. Metrics in the Literature

Evaluation practices in the literature employ a diverse set of metrics designed to assess both solution quality and structural correctness. Solution-based metrics quantify the degree to which LLM-generated solutions match reference results or solver outputs. Symbolic-level metrics evaluate syntactic accuracy, while robustness measures examine the model's reliability under varying conditions. Additional metrics focus on efficiency (e.g., runtime, convergence), mathematical fidelity (e.g., integrity gap, semantic similarity), and domain-specific utility (e.g., hypervolume, IGD). Complementing these quantitative measures, human-centered evaluations provide expert qualitative assessments of solution clarity, interpretability, and practical usefulness [39]. A concise summary of all metrics, along with their definitions and representative studies, is presented in Table 1.

# 3.2. Limitations of Existing Metrics

Although the literature reports a broad spectrum of metrics, these measures remain limited when applied in isolation. In particular:

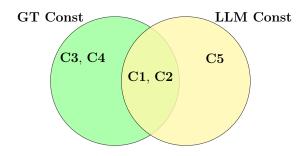
• Solution-based metrics can be misleading: Metrics such as the optimality gap may suggest high solution quality even when the generated formulation is structurally flawed. For instance, an LLM-generated model may achieve an optimality gap close to 0 while omitting critical constraints, as conceptually illustrated in Fig. 1. To demonstrate this issue with a real-world example, we evaluated an LLM on the aircraft assignment problem from the ComplexOR dataset [53]. As shown in Fig. 2, the LLM achieved perfect solution quality (optimality gap = 0) despite identifying only 50% of the ground truth constraints correctly;

Table 1: Summary of evaluation metrics for LLM-generated optimization formulations.

Category	Metric	Definition	Representative Studies
Solution quality	Solution accuracy	Deviation of obtained solution from the best-known or optimal value.	[9, 19, 7, 8, 24, 30, 34, 40, 25, 10, 18, 41, 42, 43, 21, 12, 32, 14, 35, 38, 37, 44, 24, 28, 33, 27, 45]
Solution quanty	Optimality gap	Relative difference between achieved objective and known optimum.	[22, 23, 31, 29, 25, 46, 47]
	Relative regret	Loss in performance vs. the optimal solution.	[26]
	Avg. improvement ratio	Comparison vs. heuristic baselines; $< 1 =$ better than baseline.	[31]
Surface-form accuracy	Precision, Recall, F1-score	Token-level comparison to reference formulations.	[48]
Buildability & Robustness	Compilation accuracy	Whether code parses successfully and solver accepts it without syntax or schema errors.	[18, 48]
Buildability & Robustiless	Execution rate	Proportion of compilable runs that complete successfully without runtime errors.	[14, 30, 34, 33]
	Solving / runtime	Average time for solver to produce results.	[49, 30, 18, 26, 38]
Efficience & Count	Convergence rate	Time to reach within 1% of best-known solution.	[26]
Efficiency & Search	Valid@k (time t)	Fraction where at least one of top- $k$ solutions is valid within time limit.	[20]
	Utility improvement	Gain in native units (e.g., Mbps, \$, min) vs. baseline.	[51, 26]
Domain utility	Hypervolume	Extent of objective space covered by produced Pareto set.	[52]
	Inv. gen. distance	Distance of produced Pareto set to the reference front.	[52]
	Integrity gap	Structural mismatch in variables, constraints, and associations; $0 = perfect match$ .	[50]
Mathematical fidelity	Semantic similarity	Equivalence of meaning even if expressed differently; measured via embeddings or LLM-judge.	[36]
Human-centered evaluation	Expert assessment	Qualitative evaluation of clarity, maintainability, and practical usefulness.	[38]

it missed the binary and capacity constraints while incorrectly adding a non-negativity constraint. In such cases, the optimality gap fails to reflect the unreliability of the formulation. This highlights the need for component-level metrics that evaluate not only how well a model solves, but also what it generates in terms of decision variables, constraints, and objectives.

# • Symbolic accuracy is restricted to token-level comparisons:



Precision = 2/3, Recall = 2/4, Optimality gap  $\approx 0$ 

Figure 1: LLM achieves a near-zero optimality gap but misses constraints, which shows why component-level metrics are necessary. GT Const = Ground Truth Constraints, LLM Const = LLM-generated Constraints.

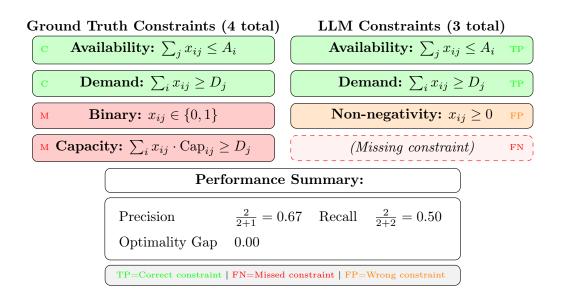


Figure 2: Real experiment showing constraint identification vs. solution quality. The LLM correctly detected availability and demand constraints (TP=2), missed binary and capacity ones (FN=2), and added a non-negativity constraint incorrectly (FP=1). Despite these errors, the solution quality remained perfect.

Metrics such as precision, recall, and F1, used in studies like Ner4Opt [48], treat the entire optimization formulation as a sequence of tokens rather than evaluating individual components like variables, constraints, or objectives. This means they measure how closely the generated tokens match the reference, without indicating whether key elements are correct or missing. As a result, a model may achieve high token-level scores while producing structurally incomplete or incorrect formulations. Moreover, token-level F1 is highly sensitive to superficial differences, such as variable naming, indexing, or formatting, potentially underestimating semantic equivalence between mathematically identical formulations.

- Numerical fidelity of objectives is overlooked: Even when an LLM generates an objective function that appears syntactically correct, small coefficient errors can alter trade-offs or decision priorities. Prior studies rarely quantify this functional closeness (e.g., via *obj-RMSE* over sampled inputs), making it difficult to detect numerically inaccurate but superficially plausible objectives.
- Constraint behavior is not assessed numerically: Feasibility is often treated as a binary outcome (feasible/infeasible) or inferred indirectly from solution quality. However, this ignores how closely generated constraints behave relative to their ground-truth counterparts under identical variable assignments. A function-level measure such as const-RMSE can reveal coefficient errors, sign mistakes, or misplaced variable terms that binary feasibility checks miss.
- Efficiency measures are incomplete: Existing works typically

focus on solver runtime while ignoring the computational cost of using LLMs themselves. Metrics such as **latency** and **token usage** are crucial for assessing the practical deployment of LLMs in optimization pipelines.

- Interdependence between metrics is unexplored: No prior study has systematically analyzed the relationships between metrics, making it difficult to understand trade-offs among structural correctness, numerical fidelity, and efficiency.
- Formulation components are not evaluated separately: Critically, no existing work assesses decision variables, constraints, and objectives as distinct components of the formulation. Without such granularity, diagnostic insights into model weaknesses remain limited.

These limitations motivate the introduction of component- and functionlevel metrics that capture both structural correctness and numerical fidelity, forming the foundation of our proposed comprehensive evaluation framework.

## 3.3. Proposed Evaluation Framework for LLM-Generated Formulations

To systematically evaluate the performance of LLMs in generating optimization formulations, we introduce a set of metrics designed to capture both symbolic correctness and functional fidelity. These metrics extend beyond traditional solution-based measures by explicitly assessing the structural, numerical, and efficiency aspects of generated formulations.

First, **constraint precision and recall** quantify how accurately the model identifies and reconstructs the symbolic structure of constraints. Second, **decision variable precision and recall** evaluate the model's ability to

correctly extract and represent the relevant decision variables. Third, the optimality gap measures the difference in objective value between the solution obtained from the LLM-generated formulation and that of the ground-truth formulation, thereby reflecting overall solution quality. Fourth, the objective function RMSE assesses numerical fidelity by computing the deviation between the generated and reference objective functions across sampled inputs. Fifth, the constraint RMSE evaluates the functional accuracy of generated constraints by measuring their numerical error relative to the ground truth. Finally, a set of efficiency metrics, including response latency and token usage, captures the computational cost and practicality of deploying LLMs in optimization modeling. In the following subsections, we provide formal definitions of each metric.

## 3.3.1. Constraints Precision and Recall

We propose the constraints precision and recall metrics to evaluate the structural accuracy of constraint generation in LLM-produced optimization formulations. Unlike prior evaluation approaches that assess token-level similarity or overall formulation overlap, this metric specifically measures how accurately an LLM reproduces the set of constraints that define the feasible region of the optimization problem. Accurately capturing these constraints is essential, as missing or incorrect constraints can lead to infeasible or suboptimal solutions, even when the objective function is correctly specified.

Let true positives ( $TP_{constraints}$ ) represent the constraints correctly generated by the LLM that also appear in the ground-truth formulation. False positives ( $FP_{constraints}$ ) are constraints generated by the LLM that do not exist in the ground truth, while false negatives ( $FN_{constraints}$ ) are constraints

present in the ground-truth formulation but missing from the LLM's output. Based on these definitions, precision and recall for constraints are computed as follows:

$$Precision_{constraints} = \frac{TP_{constraints}}{TP_{constraints} + FP_{constraints}}$$
(1)

$$Recall_{constraints} = \frac{TP_{constraints}}{TP_{constraints} + FN_{constraints}}$$
 (2)

Here, Precision<sub>constraints</sub> quantifies the proportion of generated constraints that are correct, while Recall<sub>constraints</sub> measures the proportion of ground-truth constraints that are successfully recovered by the LLM. Together, these metrics offer complementary perspectives on the structural accuracy of constraint generation.

## 3.3.2. Decision Variables Precision and Recall

We introduce the decision variables precision and recall metrics to evaluate how accurately an LLM identifies and reproduces the decision variables that define the solution space of an optimization problem. Decision variables are fundamental to the structure of any formulation, as they represent the quantities the model must determine to achieve an optimal solution. Errors in identifying these variables, such as missing, redundant, or misclassified ones, can compromise the entire optimization process, even if constraints and objectives are correctly generated.

In this context, true positives ( $TP_{variables}$ ) refer to decision variables correctly generated by the LLM that also appear in the ground-truth formulation. False positives ( $FP_{variables}$ ) are variables introduced by the LLM that are not present in the ground truth, while false negatives ( $FN_{variables}$ ) are ground-truth decision variables omitted from the LLM's output. Precision

and recall for decision variables are defined as follows:

$$Precision_{variables} = \frac{TP_{variables}}{TP_{variables} + FP_{variables}}$$
(3)

$$Recall_{\text{variables}} = \frac{TP_{\text{variables}}}{TP_{\text{variables}} + FN_{\text{variables}}}$$
(4)

Here, Precision<sub>variables</sub> reflects the proportion of generated variables that are relevant, indicating how well the model avoids introducing incorrect variables. Recall<sub>variables</sub>, on the other hand, captures the model's ability to identify all relevant decision variables from the ground truth. Together, these metrics provide a comprehensive view of the model's performance in recognizing the structural components of the decision space.

# 3.3.3. Optimality Gap (Adopted from Prior Literature)

The optimality gap is a standard metric for assessing how closely a solution approximates the true optimum. Here, it evaluates the numerical quality of LLM-generated formulations by measuring the relative deviation between the LLM's objective value and the known optimum; a smaller gap indicates higher solution quality.

To compute this metric, the LLM-generated formulation is first converted into an executable optimization model and solved using the Gurobi solver [54]. The resulting objective value is then compared against the ground-truth optimal value. The optimality gap is defined as:

Optimality gap = 
$$\frac{|\text{Optimal value} - \text{LLM objective value}|}{|\text{Optimal value}|}$$
(5)

This metric provides a direct and interpretable measure of solution quality, with values approaching zero indicating that the LLM effectively reproduces the optimal behavior of the ground-truth formulation.

## 3.3.4. RMSE for Objective Function Comparison

We are proposing the objective function root mean squared error (obj-RMSE) to evaluate the functional accuracy of the objective functions generated by the LLMs. Specifically, this metric aims to capture how numerically close the LLM-generated objective values are to the ground-truth values across a range of problem instances, regardless of the underlying symbolic structure. This allows us to assess whether the generated objectives behave similarly to the true ones when evaluated over concrete inputs. A lower obj-RMSE indicates stronger agreement with the ground truth, while a higher obj-RMSE reflects greater deviation and reduced fidelity.

Let True objective<sub>i</sub> denote the optimal objective value for the i-th instance, LLM objective<sub>i</sub> the corresponding value produced by the LLM, and n the total number of evaluated instances. The Obj-RMSE is calculated as:

Obj-RMSE = 
$$\sqrt{\frac{1}{n} \sum_{i=1}^{n} (\text{True objective}_i - \text{LLM objective}_i)^2}$$
 (6)

This metric provides a direct, quantitative measure of the extent to which the LLM's objective function approximates the numerical behavior of the true formulation.

## 3.3.5. RMSE for Constraint Behavior Comparison

The proposed constraint root mean squared error (Cons-RMSE) is used to evaluate how well an LLM's generated constraints replicate the functional behavior of the ground-truth formulation. Unlike symbolic metrics such as constraint precision and recall, which assess structural or syntactic correctness, Cons-RMSE captures the numerical alignment of matched constraints when evaluated on identical input values.

The core aim of Cons-RMSE is to answer the following question: If we plug the same decision variable values into both the LLM-generated constraints and the ground-truth constraints, do they behave the same numerically? A low Cons-RMSE indicates strong behavioral fidelity, suggesting that the LLM's constraints closely mimic the ground truth in practice. Conversely, a high Cons-RMSE implies that, even if a constraint appears structurally correct, it behaves differently, for example, due to incorrect coefficients or misapplied variable terms.

To compute this metric, we first calculate the const-RMSE individually for each matched constraint by comparing its evaluations over a fixed set of decision variable samples with those of the corresponding ground-truth constraint. We then take the average const-RMSE across all matched constraints to obtain the final Cons-RMSE score for the formulation. This averaging approach allows us to assess the overall behavioral correctness of the model's output without being skewed by any single constraint.

Importantly, Cons-RMSE is computed only over the subset of constraints that are considered matched between the LLM-generated and ground-truth formulations. Constraints that are hallucinated, omitted, or structurally incompatible are excluded from this analysis. This design ensures that Cons-RMSE focuses solely on the correctness of the constraints the LLM attempted to reproduce, complementing symbolic metrics: Precision penalizes extraneous constraints, recall penalizes missing ones, while Cons-RMSE evaluates the functional accuracy of those that are present and matched.

Let  $x^{(i)}$  denote the *i*-th decision variable sample,  $f_k^{\text{GT}}(x^{(i)})$  the evaluation of the *k*-th ground-truth constraint on that input, and  $f_k^{\text{LLM}}(x^{(i)})$  the corre-

sponding evaluation of the LLM-generated constraint. Let C be the set of matched constraints and n the number of input samples. The Cons-RMSE is computed as:

Cons-RMSE = 
$$\sqrt{\frac{1}{|\mathcal{C}| \cdot n} \sum_{k \in \mathcal{C}} \sum_{i=1}^{n} \left( f_k^{GT}(x^{(i)}) - f_k^{LLM}(x^{(i)}) \right)^2}$$
 (7)

In the experimentation, to ensure fairness and consistency across all evaluations, we use a fixed set of n=100 randomly generated input samples for each problem. These samples are drawn from valid input domains tailored to each task (e.g., continuous time intervals for scheduling problems or binary vectors for combinatorial tasks like knapsack) using a fixed random seed. This controlled sampling process eliminates randomness as a confounding factor, enabling a reliable assessment of functional correctness.

# 3.3.6. Efficiency

Efficiency evaluates the practical performance of LLMs in optimization tasks, focusing on computational resources and responsiveness. It is measured by input tokens (Total tokens given to the model, with fewer indicating efficient problem interpretation), output tokens (Tokens generated, with fewer reflecting concise formulations), and latency (Time to generate a response, with lower values indicating faster performance). These metrics capture the trade-off between computational cost and solution quality.

Fig. 3 presents a taxonomy of evaluation metrics, summarizing both those reported in the literature and the new metrics proposed in this work.

#### 4. Experiment Setup

This section presents the experimental setup and methodology used to evaluate the performance of LLMs in optimization modeling. The overall

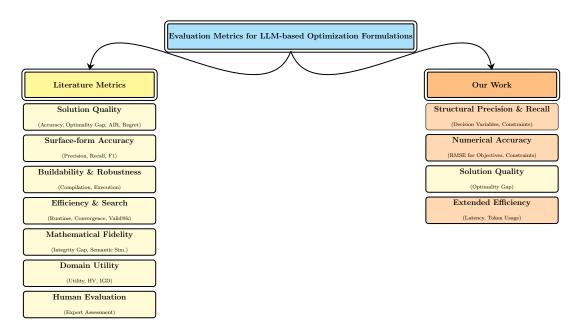


Figure 3: Taxonomy of evaluation metrics.

workflow, illustrated in Fig. 4, is organized into four main phases. First, we select a set of benchmark optimization problems with varying levels of complexity to ensure comprehensive coverage across different formulation types and difficulty tiers. Second, we employ multiple prompting strategies to guide the models in generating optimization formulations. The prompt—response loop is iteratively refined until we obtain a final set of prompt templates that yield the most accurate and consistent formulations from the LLMs. Third, we evaluate the generated formulations using the component-level metrics introduced in Section 3, encompassing structural correctness, numerical fidelity, solution quality, and efficiency. Finally, we conduct a comparative analysis across LLMs, prompting strategies, and problem complexities to uncover systematic error patterns and trade-offs in model behavior. The fol-

lowing subsections elaborate on each part of this framework in detail, covering the benchmark problems, prompting strategies, and the evaluated LLMs.

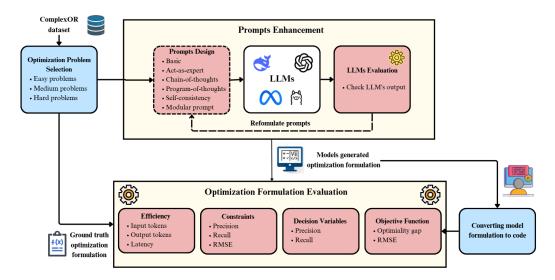


Figure 4: The main phases of the adopted methodology.

# 4.1. Optimization Problems

To evaluate the ability of LLMs to handle optimization tasks of varying difficulty, we selected four representative problems from the ComplexOR dataset [53]. These problems differ in structural and computational complexity, enabling us to assess how effectively various prompting techniques guide LLMs in translating natural language descriptions into valid mathematical formulations and solving the corresponding optimization problems. Difficulty levels: Easy, medium, and hard, are assigned based on intrinsic factors such as problem dimensionality, constraint coupling, and variable interactions. An overview of the selected problems is presented in Table 2, while detailed information about problems is provided in Appendix A.

Table 2: Representative optimization problems categorized by type and complexity.

No.	Problem	Optimization Type	Difficulty	Description
1	Knapsack problem [53]	Integer linear programming	Easy	Maximize total value of selected items with-
				out exceeding the weight limit.
2	Aircraft assignment problem [53]	Mixed-integer linear programming	Medium	Assign aircraft to routes while minimizing to-
				tal operational costs and meeting demand.
3	Diet optimization problem [53]	Linear programming	Medium	Minimize food costs while satisfying nutri-
				tional and dietary requirements.
4	Aircraft landing problem [53]	Mixed-integer linear programming	Hard	Schedule aircraft landings minimizing total
				delay penalties while respecting separation
				constraints.

## 4.2. Prompting Techniques

To evaluate how effectively LLMs can translate natural language descriptions of optimization problems into precise mathematical formulations, we employ six prompting strategies that span different levels of guidance and reasoning structure. These strategies were designed to capture a wide spectrum of reasoning depth and structural complexity, ensuring that the evaluation does not depend on a single prompting style. Each is designed to be general-purpose and applicable across diverse optimization problems, regardless of their formulation type or difficulty.

The basic prompt provides direct instructions and relies solely on the model's internal reasoning, while the act-as-expert prompt instructs the model to behave like an optimization specialist, systematically identifying objectives and constraints. The chain-of-thought prompt encourages explicit step-by-step reasoning to decompose the problem, whereas the program-of-thought prompt structures the reasoning in a code-like, algorithmic flow that enhances logical clarity for complex cases. To improve robustness, the self-consistency prompt generates multiple reasoning paths and selects the most consistent solution. Finally, the modular prompting approach divides the task into se-

quential stages, with each stage focusing on a specific component, such as decision variables, objectives, or constraints, incrementally building the complete optimization model. Fig. 5 presents the six prompting strategies used in this study.

#### **Prompting Techniques**

#### Prompt 1: Basic Prompt

Solve the above optimization problem.

- Step 1: Identify the decision variables, objective function, and constraints.
- Step 2: Write the complete mathematical formulation. Do not include any explanation, only output the formulation.

#### Prompt 2: Act-As-Expert

As an expert in optimization, solve the problem by:

- Defining the decision variables.
- Formulating the objective function.
- Listing the constraints.

Then, provide the complete formulation. Do not include any explanation, only output the formulation.

#### Prompt 3: Chain-of-Thought

Solve the problem by thinking step-by-step:

- Identify the objective.
- Define the decision variables.
- Formulate the objective function.
- List the constraints.

Finally, present the full formulation. Do not include any explanation, only output the formulation.

#### Prompt 4: Program-of-Thought

Use a logic-based reasoning process:

- Define decision variables.
- Specify the objective function.
- Add constraints.
- $\bullet$  Identify the model type. Do not include any explanation, only output the formulation.

#### Prompt 5: Self-Consistency

Generate three formulations independently for the same optimization problem:

• Each should include (decision variables, objective function, and constraints).

Internally compare and select the best one. Output only the final selected formulation.

Do not include any explanation or mention of other formulations.

#### Prompt 6: Modular Prompting Steps

- Step 1: Identify and define ONLY the decision variables. Do not solve or explain anything else.
- Step 2: Write ONLY the mathematical expression for the objective function. Do not explain or solve.
- Step 3: Write ONLY the mathematical constraints from the problem. Do not solve or interpret.
- Step 4: Combine the decision variables, objective function, and constraints into a complete optimization model. Do not explain or solve it.

Figure 5: Prompting techniques for guiding LLMs in optimization formulation.

## 4.3. Large Language Models

In this study, we employed three advanced LLMs for mathematical reasoning and optimization formulation: DeepSeek Math 7B Instruct, fine-tuned for structured mathematical problem-solving; LLaMA 3.1 8B Instruct, Meta's latest instruction-tuned model with strong reasoning performance; and GPT-5, OpenAI's newest generation offering state-of-the-art reasoning, efficiency, and robustness for complex formulations. All models were run with a low temperature (0.1) to ensure deterministic, consistent outputs. Table 3 summarizes their versions, token limits, and accessibility.

Table 3: LLM models, versions, and configurations used in the experiments.

No.	Model	Model Version	Temp.	Max Tokens	Source
1	DeepSeek Math	deepseek-math-7b-instruct	0.1	800	Open
2	LLaMA 3.1 Instruct	meta-llama/Llama-3.1-8B-Instruct	0.1	800	Open
3	GPT-5	gpt-5	0.1	800	Closed

## 5. Results and Discussion

This section evaluates the performance of LLMs in generating optimization formulations using the proposed evaluation framework. The analysis spans four benchmark problems of varying complexity: Knapsack, aircraft assignment, diet optimization, and aircraft landing, with results summarized in Tables 4-7. Each table reports the outcomes of all six prompting strategies across the three selected LLMs, assessed using the complete set of evaluation metrics: Constraint precision and recall, decision variable precision and recall, optimality gap, obj- RMSE, cons-RMSE, and efficiency (latency, input, and output tokens).

The discussion is structured in two parts. The first part presents a problem-specific analysis, examining model performance across all metrics within each problem. The second part synthesizes the results across problems, identifying systematic trends, the best-performing models and prompts, and the key relationships between metrics.

Table 4: Summary of all evaluation metrics for the knapsack problem.

Prompt	Model	Cons-P	Cons-R	DV-P	DV-R	Opt. Gap	Obj-RMSE	Cons-RMSE	Latency	Input Tokens	Output Tokens
P1	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	5.79	137	205
	LLaMA 3.1	0.66	1.00	1.00	1.00	0.00	0.00	0.00	7.97	137	251
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	2.00	137	39
P2	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	5.08	149	186
	LLaMA 3.1	0.66	1.00	1.00	1.00	0.00	0.00	0.00	8.12	149	256
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	7.00	149	60
P3	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	158	214
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	0.00	0.00	6.45	158	204
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	158	48
P4	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	4.51	142	164
	LLaMA 3.1	0.66	1.00	1.00	1.00	0.00	0.00	0.00	6.41	142	203
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	1.00	142	50
P5	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	19.66	146	714
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	0.00	0.00	6.10	146	193
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	15.00	146	55
P6	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	1.81	429	60
	LLaMA 3.1	0.66	1.00	1.00	1.00	0.00	0.00	0.00	7.32	429	230
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	8.00	429	20

## 5.1. Problem-Specific Performance Analysis

## 5.1.1. Knapsack Problem

For knapsack problem, GPT-5 and DeepSeek achieved perfect constraint precision and recall (1.00), reproducing the minimal ground-truth constraints exactly. LLaMA 3.1 maintained full recall but often added redundant conditions, such as non-negativity ( $x_i \geq 0$ ), reducing its precision to 0.66. These extra constraints did not affect feasibility but lowered structural fidelity.

All models handled decision variables flawlessly, with precision and recall of 1.00, indicating no omissions or errors. Solver-level metrics confirmed cor-

Table 5: Summary of all evaluation metrics for the aircraft assignment problem.

Prompt	Model	Cons-P	Cons-R	DV-P	DV-R	Opt. Gap	Obj-RMSE	Cons-RMSE	Latency	Input Tokens	Output Tokens
P1	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	12.34	198	439
	LLaMA 3.1	1.00	1.00	0.50	1.00	0.04	279.00	0.00	11.32	198	358
	GPT-5	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	<b>11.00</b>	<b>198</b>	<b>159</b>
P2	DeepSeek	0.66	0.66	0.50	1.00	0.00	0.00	0.07	5.91	210	210
	LLaMA 3.1	1.00	1.00	0.50	1.00	0.04	263.00	0.00	11.64	210	364
	GPT-5	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	10.00	210	329
P3	DeepSeek	0.75	1.00	0.50	1.00	0.45	525.35	0.00	9.28	219	340
	LLaMA 3.1	0.66	0.66	0.50	1.00	-	2031.00	0.43	10.44	219	331
	GPT-5	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	13.00	<b>219</b>	<b>292</b>
P4	DeepSeek	0.66	0.66	0.50	1.00	0.12	528.83	0.07	4.43	203	163
	LLaMA 3.1	0.66	0.66	0.33	1.00	-	2147.00	0.45	8.23	203	262
	GPT-5	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	5.00	203	253
P5	DeepSeek LLaMA 3.1 GPT-5	1.00 0.33 1.00	1.00 0.33 1.00	1.00 0.33 1.00	1.00 1.00 1.00	<b>0.00</b> 160.00 0.00	<b>0.00</b> 533.00 0.00	<b>0.00</b> 37.63 0.06	<b>7.73</b> 9.91 21.00	<b>207</b> 207 207	281 312 460
P6	DeepSeek	0.50	1.00	1.00	1.00	0.45	0.00	0.00	3.56	432	123
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	252.00	0.00	6.80	432	214
	GPT-5	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>1.00</b>	<b>0.00</b>	<b>0.00</b>	<b>0.00</b>	7.00	<b>432</b>	<b>31</b>

Table 6: Summary of all evaluation metrics for the diet problem.

Prompt	Model	Cons-P	Cons-R	DV-P	DV-R	Opt. Gap	Obj-RMSE	Cons-RMSE	Latency	Input Tokens	Output Tokens
P1	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	8.56	210	328
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	0.00	0.00	8.56	210	271
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	2.00	210	209
P2	DeepSeek	1.00	0.80	1.00	1.00	0.00	0.00	1.42	10.49	223	271
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	0.00	0.00	10.49	223	331
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	3.00	223	214
P3	DeepSeek	1.00	1.00	1.00	1.00	0.00	0.00	0.00	6.37	229	415
	LLaMA 3.1	1.00	1.00	1.00	1.00	0.00	0.00	0.00	6.37	229	199
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	4.00	229	180
P4	DeepSeek	1.00	0.80	1.00	1.00	0.00	0.00	1.42	7.58	216	391
	LLaMA 3.1		1.00	0.50	1.00	0.00	0.00	0.00	7.58	216	241
	GPT-5	1.00	0.40	1.00	1.00	0.00	0.00	2.90	0.60	216	68
P5	DeepSeek	1.00	0.80	1.00	1.00	0.00	0.00	1.42	19.17	219	467
	LLaMA 3.1		0.80	1.00	1.00	0.00	0.00	1.42	19.17	219	610
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	2.00	219	210
P6	DeepSeek	1.00	0.80	0.00	0.00	0.00	0.00	14.00	5.68	476	297
	LLaMA 3.1		1.00	1.00	1.00	0.00	0.00	0.00	5.68	476	178
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.60	476	34

Table 7: Summary of all evaluation metrics for the aircraft landing problem.

Prompt	Model	Cons-P	Cons-R	DV-P	DV-R	Opt. Gap	${\bf Obj\text{-}RMSE}$	Cons-RMSE	Latency	Input Tokens	Output Tokens
P1	DeepSeek	1.00	0.66	1.00	0.75	0.00	7.78	1.65	10.75	248	393
	LLaMA 3.1	1.00	0.66	0.50	0.50	0.85	86.37	1.85	13.65	248	432
	GPT-5	1.00	0.75	1.00	0.75	0.00	0.00	1.57	24.00	248	248
P2	DeepSeek	1.00	0.75	1.00	0.25	0.24	0.00	3.22	12.16	260	445
	LLaMA 3.1	0.75	0.50	0.50	0.50	0.85	86.37	2.98	16.10	260	509
	GPT-5	1.00	0.75	1.00	0.75	0.00	0.00	1.57	27.00	260	247
P3	DeepSeek	1.00	0.50	1.00	0.25	0.89	195.57	2.91	14.26	269	522
	LLaMA 3.1	0.75	0.50	0.50	0.50	0.85	86.54	2.98	15.63	269	496
	GPT-5	1.00	0.75	1.00	0.75	0.00	0.00	1.57	22.00	269	161
P4	DeepSeek	1.00	1.00	1.00	0.25	0.92	0.00	0.00	5.85	253	213
	LLaMA 3.1	1.00	1.00	0.33	0.25	0.85	102.24	0.00	10.29	253	325
	GPT-5	1.00	0.75	1.00	0.75	0.00	0.00	1.57	56.00	253	98
P5	DeepSeek	1.00	0.50	1.00	0.25	1.00	258.20	1.65	20.73	257	755
	LLaMA 3.1	1.00	0.66	0.50	0.25	1.00	85.15	1.85	13.82	257	435
	GPT-5	1.00	1.00	1.00	0.75	0.00	0.00	0.00	15.00	257	168
P6	DeepSeek	1.00	1.00	1.00	0.25	0.75	0.00	0.00	5.80	1008	199
	LLaMA 3.1	1.00	1.00	0.25	0.25		99.33	1.85	9.70	1008	298
	GPT-5	1.00	1.00	1.00	1.00	0.00	0.00	0.00	5.00	1008	65

rectness: the optimality gap and objective RMSE were 0.00 for every model and prompt, demonstrating exact reproduction of the objective. Constraint behavior, measured by Cons-RMSE, was also perfect (0.00) for GPT-5 and DeepSeek. LLaMA 3.1 matched this functional behavior despite reduced precision due to redundant constraints.

Efficiency highlighted differences: GPT-5 produced the most concise outputs, with lower latency and fewer tokens. DeepSeek was moderately efficient, while LLaMA generated longer outputs with higher latency.

Summary: All models reliably captured the knapsack problem, achieving perfect recall and zero optimality gap. GPT-5 delivered the exact minimal formulation most efficiently, DeepSeek matched its correctness but was less efficient, and LLaMA preserved feasibility while reducing precision with extra constraints.

## 5.1.2. Aircraft Assignment Problem

The aircraft assignment problem, a medium complexity case that requires binary assignment and resource allocation constraints, revealed a wider variation in model performance than the knapsack problem. GPT-5 consistently achieved perfect constraint precision and recall (1.00) across all prompts. DeepSeek performed well but dropped to 0.66 in Prompts 2 and 4 due to omitted binary restrictions. LLaMA 3.1 was the least stable, ranging from perfect performance to 0.33–0.66 when constraints appeared in auxiliary or non-standard forms.

Decision variable identification mirrored this pattern: GPT-5 maintained perfect precision and recall, while DeepSeek and LLaMA occasionally misspecified variables, especially under non-standard formulations. On the other hand, solver-level metrics confirmed these trends. GPT-5 achieved an optimality gap of 0.00 across prompts. DeepSeek showed minor deviations, while LLaMA produced large errors, with objective RMSE exceeding 2000 in some cases. Cons-RMSE followed a similar pattern: GPT-5 was nearly perfect, DeepSeek deviated slightly in Prompts 2 and 4 (Cons-RMSE 0.07), and LLaMA showed severe deviations (up to 37.63) due to non-standard formulations.

Efficiency further distinguished the models. GPT-5 generated concise outputs with low token usage and moderate latency. DeepSeek was less efficient, and LLaMA was the least efficient, producing longer outputs with variable latency.

Summary: The aircraft assignment problem exposed larger gaps between models. GPT-5 was the most robust, consistently accurate, and efficient.

DeepSeek was generally reliable but occasionally sensitive to prompt design, while LLaMA was unstable and prone to redundant or altered constraints that reduced precision and efficiency.

## 5.1.3. Diet Optimization Problem

The diet optimization problem, a medium-complexity task with continuous variables and multiple nutrient constraints, showed more variability in constraint precision and recall. GPT-5 consistently achieved perfect scores (1.00), reliably representing all food- and nutrient-related constraints. DeepSeek generally performed well but dropped recall to 0.80 when non-negativity conditions were omitted. LLaMA 3.1 was less consistent: precision fell to 0.71 in Prompt 4 due to redundant or auxiliary constraints, and recall dropped to 0.80 in other prompts where constraints were missed.

Decision variable handling was highly accurate across all models. GPT-5 maintained perfect precision and recall, while DeepSeek and LLaMA had minor lapses. Solver-level metrics confirmed robustness, with optimality gap and objective RMSE at 0.00 for all models.

Constraint behavior, measured by Cons-RMSE, highlighted differences. DeepSeek was perfect in Prompts 1 and 3 (Cons-RMSE 0.00) but showed moderate deviations (~ 1.42) in Prompts 2, 4, and 5 due to omitted non-negativity constraints, and severe errors (Cons-RMSE 14.00) in Prompt 6 from hallucinated variables. LLaMA aligned perfectly in Prompts 1, 2, 3, 4, and 6 (Cons-RMSE 0.00) but had 1.42 in Prompt 5. GPT-5 reproduced all constraints correctly except in Prompt 4 (Cons-RMSE 2.90), reflecting partial but mostly correct behavior.

Efficiency further distinguished models. GPT-5 produced the most con-

cise outputs with the lowest tokens and latency. DeepSeek was less efficient, particularly in Prompt 5, while LLaMA was the least efficient, consistently generating longer outputs with higher and variable latency.

Summary: The diet optimization problem revealed clear model differences. GPT-5 was the most reliable and efficient, with only minor omissions. DeepSeek performed well but occasionally missed non-negativity conditions or introduced hallucinations. LLaMA was the least stable, often producing redundant or structurally inconsistent formulations.

# 5.1.4. Aircraft Landing Problem

The aircraft landing problem is the most complex benchmark case, combining sequencing decisions, time-window feasibility, and pairwise separation constraints. Constraint precision and recall reflect this challenge. GPT-5 maintained high precision but had lower recall in early prompts, partially enforcing separation rules. DeepSeek had perfect precision, but recall dropped to 0.50 when separation rules were oversimplified. LLaMA 3.1 was unstable, with recall from 0.50 to 0.66 when separation rules were misapplied or reversed. Time-window constraints were generally handled well across models.

Decision variable handling varied. GPT-5 and DeepSeek had perfect precision (1.00) but weaker recall, dropping to 0.25 in some prompts. LLaMA performed worst, with both precision and recall frequently below 0.50. However, solver-level metrics emphasized these differences. GPT-5 consistently had an optimality gap of 0.00 and an objective RMSE of 0.00. DeepSeek and LLaMA showed poor alignment, with gaps up to 1.00 and RMSE exceeding 285 and 102.24, respectively.

Constraint behavior (Cons-RMSE) highlighted model robustness. DeepSeek

and LLaMA achieved perfect alignment in Prompts 4 and 6 but lower recall in other prompts. GPT-5 showed partial handling of separations in Prompts 1–4 (Cons-RMSE 1.57), but Prompts 5 and 6 achieved perfect alignment (0.00) using structured separation formulations.

Efficiency further distinguished models. GPT-5 had the lowest latency and token usage, particularly in Prompts 5 and 6. DeepSeek was moderately efficient with variable output lengths, while LLaMA was the least efficient, generating longer outputs with high latency.

Summary: The aircraft landing problem exposed major weaknesses in constraint modeling. All models handled time windows well, but pairwise separations were challenging. GPT-5 was the most effective, producing accurate, complete, and efficient formulations. DeepSeek achieved partial success, often relying on simplified rules, while LLaMA was the least consistent, frequently misapplying or oversimplifying separation constraints.

## 5.2. General Insights Across Problems

This section synthesizes insights from the four problems to provide a cross-cutting view of how LLMs perform in generating optimization formulations. We organize the discussion into three parts: (i) Identifying the best-performing model for each problem, (ii) comparing the effectiveness of different prompting strategies, and (iii) analyzing the relationships between evaluation metrics. Together, these perspectives provide a holistic understanding of strengths, weaknesses, and systematic trends across models and prompts.

## 5.2.1. Best Model per Problem

This section presents a cross-problem comparison to evaluate how the three models: GPT-5, DeepSeek, and LLaMA 3.1, perform across different levels of problem complexity. The aim is to capture systematic strengths and weaknesses by jointly considering constraint handling, solver outcomes, and efficiency.

The radar charts in Fig. 6 illustrate model performance across the four benchmark problems. In the simple knapsack problem, all models achieved high scores, but GPT-5 stood out with exact structural fidelity and efficient outputs; DeepSeek was reliable but less efficient, while LLaMA 3.1 preserved recall yet lowered precision by adding redundant constraints. In the aircraft assignment problem, differences became clearer: GPT-5 dominated across all metrics, DeepSeek was generally accurate but sometimes omitted binary constraints, and LLaMA lagged with weaker precision, solver alignment, and efficiency. In the diet optimization problem, GPT-5 again delivered near-perfect scores and the most efficient outputs. DeepSeek handled constraints well but often missed non-negativity conditions, and LLaMA was inconsistent, achieving good recall in some prompts but lower precision and solver stability overall. The gap widened in the complex aircraft landing problem, where only GPT-5 consistently produced solver-ready formulations, DeepSeek struggled with recall and solver correctness, and LLaMA performed worst with markedly lower constraint and solver scores. Overall, GPT-5 was the most robust model across all problems, followed by DeepSeek, while LLaMA showed the least consistency and efficiency.

To construct the radar charts, performance scores were first averaged

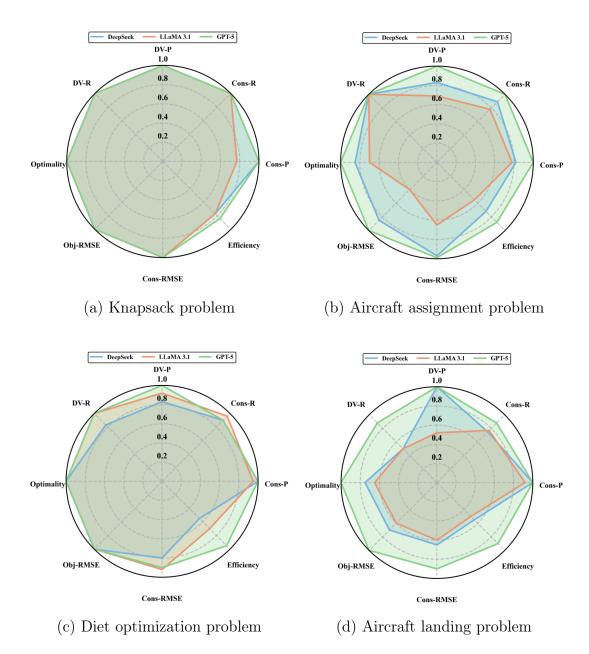


Figure 6: Radar chart comparison of model performance across four benchmark optimization problems. Each axis represents an evaluation metric: Constraint precision (Cons-P), constraint recall (Cons-R), decision variable precision (DV-P), decision variable recall (DV-R), optimality gap (Optimality), objective RMSE (Obj-RMSE), constraint RMSE (Cons-RMSE), and efficiency (Latency, input tokens, output tokens). Higher values indicate better performance.

across prompts and then normalized for fair comparison. Metrics where higher values represent better performance were scaled to the [0,1] range using min–max normalization, while error-based metrics were inverted before scaling so that higher values consistently indicated better results. Efficiency was computed from latency, input tokens, and output tokens by normalizing, inverting, and averaging them into a single score. Finally, all normalized scores were averaged across prompts to obtain one representative value per model and problem.

# 5.2.2. Best Prompting Strategy

Beyond comparing models, it is equally important to understand how different prompting strategies affect formulation quality. Because each problem type poses unique structural and semantic challenges, the design of prompts plays a crucial role in guiding models to correctly capture objectives, variables, and constraints. This section identifies the most effective prompts for each problem, discusses why they work, and explains how their performance was systematically evaluated.

As shown in Fig. 7, prompt effectiveness varied noticeably with problem complexity. In the simple Knapsack problem, Prompts P3 and P5 consistently achieved the highest scores across all models. In the aircraft assignment problem, Prompts P5, P3, P6, and P1 proved most reliable, while P2 and P4 frequently led to missing binary constraints and weaker results. For the diet optimization problem, Prompts P3, P5, P6, and P1 produced the strongest outcomes, whereas P2 and P4 sometimes introduced structural errors. In the most complex aircraft landing problem, Prompts P5 and P6 were essential for achieving full alignment, with P3 also performing strongly,

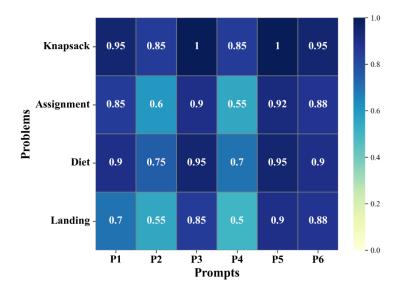


Figure 7: Heatmap of prompt effectiveness across problems. Each cell shows the aggregated performance score (0–100%) for each *problem-prompt* pair. Darker shades indicate better performance, with P3, P5, and P6 showing the highest overall scores.

# particularly for GPT-5.

Collectively, these findings highlight P3 (Chain-of-Thought), P5 (Self-Consistency), and P6 (Modular Prompting Steps) as the most effective strategies overall. P3 was effective because its structured, step-by-step reasoning reduced logical oversights when identifying objectives, variables, and constraints. P5 improved reliability by generating multiple candidate formulations and internally selecting the best, minimizing omissions and inaccuracies. P6, especially valuable in medium- and high-complexity problems, decomposed the task into sequential stages, first defining variables, then objectives, then constraints, ensuring comprehensive coverage and well-structured outputs. In particular, P5 and P6 were crucial for complex tasks such as aircraft assignment and aircraft landing, where modular decomposition and

redundancy checking were needed to generate solver-ready formulations.

To construct the heatmap and evaluate prompt effectiveness across all problems, we computed an aggregate performance score for each problem-prompt pair. Metrics where higher values indicate better performance were min-max normalized to [0,1], while error-based and efficiency metrics were inverted before normalization so that higher scores consistently reflected better outcomes. Efficiency (latency, input tokens, output tokens) was averaged into a single score. Scores were then averaged across models and metrics to yield a percentage-like (0–100%) measure of overall prompt effectiveness.

## 5.2.3. Relationships Between Metrics

To understand how different metrics interact when evaluating generated formulations, we analyze their pairwise correlations. The goal is to identify which metrics most strongly influence solver performance, which ones overlap, and how efficiency relates to correctness. The correlation heatmap in Fig. 8 provides a clear overview of these relationships.

First, constraint recall (Cons-R) showed the strongest connection to constraint RMSE (Cons-RMSE). Their correlation was strongly negative (r = -0.51), meaning that the more constraints are covered, the fewer violations are observed. When recall = 1, all constraints are present, and Cons-RMSE is almost always 0. When recall drops below 1, at least one constraint is missing: if the missing constraint is critical (e.g., demand satisfaction or aircraft separation), Cons-RMSE rises sharply, but if the missing constraint is redundant (e.g., a non-negativity bound), Cons-RMSE may remain close to 0. Precision also correlated negatively with Cons-RMSE (r = -0.42), but with a weaker effect, confirming that recall is the primary factor for reducing

violations. In short, full recall (Cons-R=1) ensures Cons-RMSE is near 0, while recall below 1 can lead to high Cons-RMSE when critical constraints are missing, though harmless omissions may have little effect.

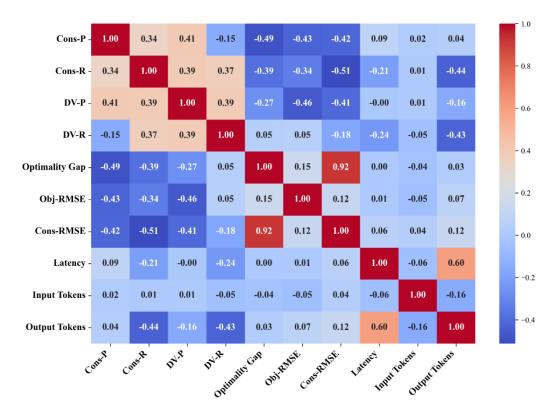


Figure 8: Correlation heatmap showing relationships among constraint metrics, solver metrics, and efficiency measures. Red indicates positive and blue negative correlations, with color intensity reflecting strength.

Second, constraint RMSE (Cons-RMSE) was the most reliable predictor of solver outcomes. Its correlation with the optimality gap was extremely strong and positive (r = 0.92). This means that when Cons-RMSE is close to 0, indicating that all constraints are satisfied, the optimality gap is also close to 0, and the solver reaches an optimal solution. By contrast, when

Cons-RMSE increases because some constraints are violated, the gap widens sharply, showing that the solution is far from optimal. The correlation with objective RMSE was weaker but still positive (r = 0.12). This shows that constraint violations can sometimes propagate into inconsistencies in the objective values, though this effect is less consistent than for the gap. In short, when Cons-RMSE is low, the solver produces optimal solutions, but as Cons-RMSE grows, the optimality gap rises and objective values become less reliable.

Third, the two solver-level metrics behaved differently from each other. The optimality gap and objective RMSE showed only a weak positive correlation (r=0.15), meaning that a large gap does not always imply a large objective error, and vice versa. In practice, the gap is near 0 when all constraints are satisfied and the solver finds an optimal solution, but it increases when critical constraints are missing. Similarly, Objective RMSE remains close to 0 when the objective values align with the optimal solution, but rises when constraint violations distort the objective function. Both metrics, however, were moderately negatively correlated with constraint recall (optimality gap: r=-0.39, Obj-RMSE: r=-0.34), showing that incomplete constraint coverage is the common driver of poor solver performance. In short, when recall is high, both the gap and objective RMSE stay near 0; when recall drops, both metrics worsen, even though they are only weakly related to each other.

Fourth, the constraint metrics were moderately related to decision variable precision but not to decision variable recall. Cons-P and Cons-R were positively correlated with decision variable precision (DV-P: r = 0.41 with

Cons-P, r = 0.39 with Cons-R), showing that when constraints are represented more precisely and more completely, decision variables also tend to be specified more accurately. By contrast, correlations with decision variable recall (DV-R) were close to 0, indicating that covering or omitting constraints has little direct influence on whether all decision variables are included. These results suggest that constraint handling mainly improves the correctness of decision variable definitions rather than their completeness. In short, higher constraint precision and recall both support higher decision variable precision, while decision variable recall remains largely unaffected.

Finally, efficiency metrics formed a distinct group. Latency correlated strongly with output tokens (r = 0.60), showing that more verbose outputs consistently increased inference time. In contrast, Input tokens had near-zero correlations with all other metrics, confirming that efficiency is mainly driven by output verbosity rather than input size. In practice, when output tokens are large, latency is high, while input length makes little difference.

Overall, the heatmap indicates that solver performance depends strongly on two factors: High constraint recall and low cons-RMSE. These metrics directly determine whether the generated formulation satisfies the problem structure and produces correct solutions. Constraint precision and decision variable metrics contribute positively but play a secondary role. Efficiency, in turn, is shaped mainly by output verbosity, with longer outputs increasing latency. In summary, three key principles emerge: Complete constraint coverage prevents violations, minimizing constraint RMSE ensures solver-level correctness, and concise outputs deliver greater efficiency.

### 6. Conclusion and Future Research

Formulating optimization problems traditionally demands deep domain knowledge, mathematical precision, and iterative collaboration between analysts and engineers, a process that is time-consuming, error-prone, and inaccessible to non-experts. Large language models (LLMs) offer a transformative opportunity to automate this process by translating natural language descriptions into structured mathematical formulations. Yet, despite their advanced reasoning and coding abilities, LLM performance in optimization modeling remains underexplored and lacks systematic, fine-grained evaluation.

We present a comprehensive framework for evaluating LLM-generated optimization formulations at the component level, assessing decision variables, constraints, objective functions, and solution quality. The framework introduces metrics for precision, recall, and root mean squared error (RMSE), complemented by efficiency indicators such as token usage and latency. This multi-dimensional approach provides a diagnostic view of structural correctness, numerical fidelity, and computational efficiency. We validated the framework on four optimization problems of varying complexity using state-of-the-art LLMs (GPT-5, LLaMA, and DeepSeek) and six prompting strategies. Our results identify top-performing models and prompts, revealing trade-offs between structural accuracy, solver quality, and efficiency, and uncover interdependencies among evaluation metrics that shed light on LLM behavior in optimization formulation.

Future work can extend this framework to broader problem sets and additional LLMs, including domain-specialized models, to assess generalizability and architectural impacts on performance. This study lays the foundation

for targeted improvements, enabling the development of more reliable and mathematically capable LLMs for optimization problem formulation.

# Acknowledgement

The authors would like to acknowledge the support received from the Saudi Data and AI Authority (SDAIA) and King Fahd University of Petroleum & Minerals (KFUPM) under the SDAIA-KFUPM Joint Research Center for Artificial Intelligence Grant JRC-AI-RFP-20.

## Appendix A. Optimization Problem Details

This appendix provides detailed descriptions, mathematical formulations, and sample data for the optimization problems used in our experiments. The selected problems include the knapsack problem, aircraft assignment problem, diet problem, and aircraft landing problem, all drawn from the ComplexOR dataset [53].

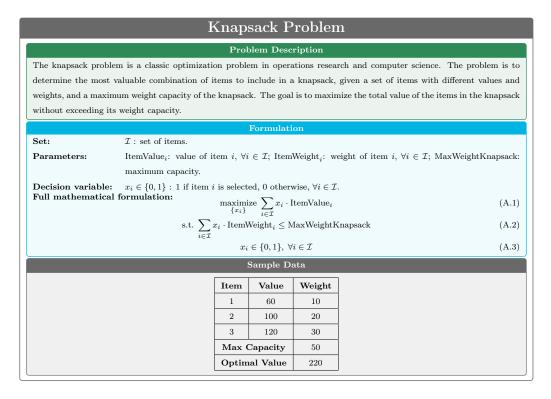


Figure A.9: Illustration of the Knapsack problem, presenting its description, ground-truth mathematical formulation, and sample dataset used in the experimental evaluation.

#### Aircraft Assignment Problem Problem Description The aircraft assignment problem aims to assign aircraft to routes in order to minimize the total cost while satisfying demand constraints with available aircraft. The problem involves a set of aircraft and a set of routes. Given the costs of assigning an aircraft to a route. The objective is to minimize the total cost of the assignment. There are limited available aircraft. It is constrained that the number of each aircraft allocated does not exceed its available number. Given the demand of each route and the capabilities (the largest number of people can be carried) of an aircraft for a route. The demand constraint ensures that the total allocation for each route satisfies the demand. The problem seeks to find the most cost-effective assignment of aircraft to routes. Sets: A: set of aircraft; R: set of routes. Parameters: $\text{Costs}_{a,r}$ : cost of assigning aircraft a to route r; Availability $_a$ : number of available aircraft of type a; Capabilities a,r: passenger capacity of aircraft a on route r; Demand r: required passengers on route r. Decision variable: $x_{a,r} \in \{0,1\}$ : 1 if aircraft a is assigned to route r, 0 otherwise. Full mathematical formulation: $\underset{\{x_{a,r}\}}{\text{minimize}} \sum_{a \in \mathcal{A}} \sum_{r \in \mathcal{R}} \text{Costs}_{a,r} \cdot x_{a,r}$ (A.4)s.t. $\sum_{r \in \mathcal{R}} x_{a,r} \leq \text{Availability}_a, \quad \forall a \in \mathcal{A}$ (A.5) $\sum_{a \in \mathcal{A}} x_{a,r} \cdot \text{Capabilities}_{a,r} \geq \text{Demand}_r, \quad \forall r \in \mathcal{R}$ (A.6) $x_{a,r} \in \{0,1\}, \quad \forall a \in \mathcal{A}, \ r \in \mathcal{R}$ (A.7)Sample Data Aircraft Availability: [2, 3, 1], Route Demand: [100, 150] Capabilities Matrix (Passengers per Aircraft per Route): Aircraft Route 1 Route 2 50 70 2 80 3 90 70 Cost Matrix (Assignment Costs): Aircraft Route 1 Route 2 100 200 150 250 3 Optimal Solution Value: 700

Figure A.10: Illustration of the aircraft assignment problem, presenting its description, ground-truth mathematical formulation, and sample dataset used in the experimental evaluation.

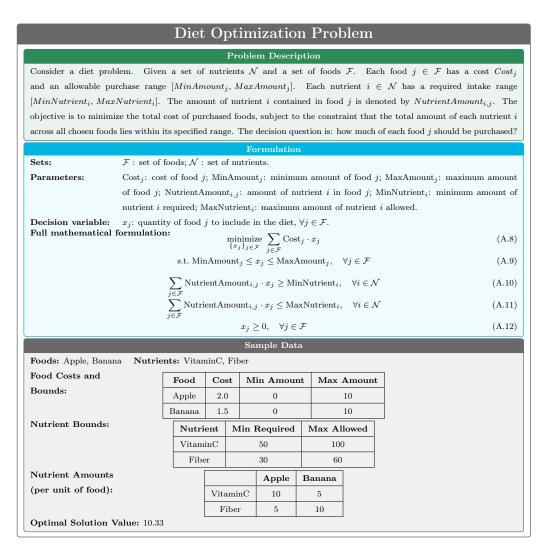


Figure A.11: Illustration of the diet problem, presenting its description, ground-truth mathematical formulation, and sample dataset used in the experimental evaluation.

#### landing time, target landing time, and penalties for landing before or after the target landing time for each aircraft. There is also a separation time that represents the minimum time required between the landing of two aircraft. The objective of the problem is to minimize the total penalties of landing before or after the target time for each aircraft. The problem includes several constraints. The order constraint ensures that the aircrafts land in a specific order. The separation constraint ensures that there is enough separation time between the landing of aircraft. The lower and upper time window constraints ensure that each aircraft lands within its respective earliest and latest time windows. **Sets and Indices:** A: set of aircraft, indexed by i, j. $E_i$ : earliest landing time for aircraft i; $L_i$ : latest landing time; $T_i$ : target landing time; $P_i^{\text{early}}$ : penalty Parameters: per unit time if aircraft i lands before $T_i$ ; $P_i^{\text{late}}$ : penalty per unit time if aircraft i lands after $T_i$ ; $S_{i,j}$ : minimum separation time if i lands before j; M: large constant. **Decision variables**: $x_i$ : landing time of aircraft i; $e_i$ : earliness; $l_i$ : lateness; $z_{i,j} \in \{0,1\}$ , 1 if i lands before j, 0 otherwise. $\underset{x_{i}, e_{i}, l_{i}, z_{i, j}}{\text{minimize}} \sum_{i \in \mathcal{A}} P_{i}^{\text{early}} e_{i} + P_{i}^{\text{late}} l_{i}$ Full mathematical formulation: (A.13) $E_i \le x_i \le L_i, \quad \forall i \in \mathcal{A}$ (A.14) $e_i \ge T_i - x_i, \quad \forall i \in \mathcal{A}$ (A.15) $l_i \ge x_i - T_i, \quad \forall i \in \mathcal{A}$ (A.16) $z_{i,j} + z_{j,i} = 1, \quad \forall i \neq j \in \mathcal{A}$ (A.17) $x_j \ge x_i + S_{i,j} - M(1 - z_{i,j}), \quad \forall i \ne j \in \mathcal{A}$ (A.18)Sample Data Earliest Landing Times: [1, 3, 5], Latest Landing Times: [10, 12, 15], Target Landing Times: [4, 8, 14] Penalties: Early Penalty Late Penalty 10 2 10 20 3 15 30 Separation Times: **A**1 $\mathbf{A2}$ $\mathbf{A3}$ 3 Α1 0 2 0 4 А3 3 4 0 Optimal Solution Value: 0

Aircraft Landing Problem (ALP)

Problem Description

The Aircraft Landing Problem (ALP) is the problem of deciding a landing time on an appropriate runway for each aircraft in a given set of aircraft such that each aircraft lands within a predetermined time window; and separation criteria between the landing of an aircraft, and the landing of all successive aircraft, are respected. We are given the earliest landing time, latest

Figure A.12: Illustration of the aircraft landing problem, presenting its description, ground-truth mathematical formulation, and sample dataset used in the experimental evaluation.

### References

- [1] P. Kumar, Large language models (llms): Survey, technical frameworks, and future challenges, Artificial Intelligence Review 57 (10) (2024) 260.
- [2] H. Zou, Y. Wang, A novel large language model enhanced joint learning framework for fine-grained sentiment analy-Neurocomputing  $\sin$ on drug reviews, 626 (2025)129589. doi:https://doi.org/10.1016/j.neucom.2025.129589. URL https://www.sciencedirect.com/science/article/pii/ S0925231225002619
- [3] H. Zou, Y. Wang, Large language model augmented syntax-aware domain adaptation method for aspect-based sentiment analysis, Neurocomputing 625 (2025) 129472. doi:https://doi.org/10.1016/j.neucom.2025.129472.

  URL https://www.sciencedirect.com/science/article/pii/S0925231225001444
- [4] T. Guo, C. Wang, Y. Liu, J. Tang, P. Li, S. Xu, Q. Yang, X. Gao, Z. Li, Y. Wen, Leveraging inter-chunk interactions for enhanced retrieval in large language model-based question answering, Neurocomputing 650 (2025) 130931. doi:https://doi.org/10.1016/j.neucom.2025.130931.
  URL https://www.sciencedirect.com/science/article/pii/
- [5] D. Bzdok, A. Thieme, O. Levkovskyy, P. Wren, T. Ray, S. Reddy,

S0925231225016030

Data science opportunities of large language models for neuroscience and biomedicine, Neuron 112 (5) (2024) 698–717. doi:https://doi.org/10.1016/j.neuron.2024.01.016.

URL https://www.sciencedirect.com/science/article/pii/ S0896627324000424

- [6] J. Gao, J. Cui, H. Wu, L. Xiang, H. Zhao, X. Li, M. Fang, Y. Yang, Z. He, Can large language models independently complete tasks? a dynamic evaluation framework for multi-turn task planning and completion, Neurocomputing 638 (2025) 130135. doi:https://doi.org/10.1016/j.neucom.2025.130135.
  - URL https://www.sciencedirect.com/science/article/pii/ S0925231225008070
- [7] Q. Li, L. Zhang, V. Mak-Hau, Synthesizing mixed-integer linear programming models from natural language descriptions, arXiv preprint arXiv:2311.15271 (Nov. 2023).
- [8] T. Ahmed, S. Choudhury, LM4OPT: Unveiling the potential of large language models in formulating mathematical optimization problems, Inf. Syst. Oper. Res. 62 (4) (2024) 559–572.
- [9] Z. Huang, Y. Yang, Y. Wang, Z. Guo, W. Shi, X. Han, F. Liang, L. Song, X. Liang, J. Tang, OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling, arXiv preprint arXiv:2407.09887 (Jul. 2024).
- [10] A. AhmadiTeshnizi, W. Gao, H. Brunborg, S. Talaei, M. Udell,

- OptiMUS-0.3: Using large language models to model and solve optimization problems at scale, arXiv preprint arXiv:2407.19633 (Jul. 2024).
- [11] T. Wang, Rare mip challenges: Leveraging large language models for mixed integer programming, arXiv preprint arXiv:2403.01131 (2024). URL https://arxiv.org/abs/2403.01131
- [12] N. Astorga, T. Liu, Y. Xiao, M. van der Schaar, Autoformulation of mathematical optimization models using LLMs, arXiv preprint arXiv:2411.01679 (Nov. 2024).
- [13] A. AhmadiTeshnizi, W. Gao, M. Udell, Optimus: Optimization modeling using mip solvers and large language models, arXiv preprint arXiv:2310.06116 (2023).
  URL https://arxiv.org/abs/2310.06116
- [14] Z. Wang, B. Chen, Y. Huang, Q. Cao, M. He, J. Fan, X. Liang, ORMind: A cognitive-inspired end-to-end reasoning framework for operations research, arXiv preprint arXiv:2506.01326 (Jun. 2025).
- [15] T. Ahmed, S. Choudhury, Lm4opt: Unveiling the potential of large language models in formulating mathematical optimization problems, arXiv preprint arXiv:2405.13144 (2024). URL https://arxiv.org/abs/2405.13144
- [16] R. Pan, S. Xing, S. Diao, W. Sun, X. Liu, K. Shum, J. Zhang, R. Pi, T. Zhang, Plum: Prompt learning using metaheuristics, in: Findings of the Association for Computational Linguistics: ACL 2024, 2024, pp.

2177 - 2197.

URL https://aclanthology.org/2024.findings-acl.129/

- [17] D. Tsouros, Holy grail 2.0: Constraint programming with large language models, in: Proceedings of the 30th International Conference on Principles and Practice of Constraint Programming, 2024, pp. 1–16. URL https://arxiv.org/abs/2401.12345
- [18] Z. Xiao, D. Zhang, Y. Wu, L. Xu, Y. J. Wang, X. Han, X. Fu, T. Zhong, J. Zeng, M. Song, et al., Chain-of-Experts: When LLMs meet complex operations research problems, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2023.
- [19] T. Wang, W.-Y. Yu, Z. He, Z. Liu, X. Han, H. Gong, H. Wu, W. Shi, R. She, F. Zhu, et al., BPP-Search: Enhancing tree of thought reasoning for mathematical modeling problem solving, arXiv preprint arXiv:2411.17404 (Nov. 2024).
- [20] J. Li, R. Wickman, S. Bhatnagar, R. K. Maity, A. Mukherjee, NL2OR: Solve complex operations research problems using natural language inputs, arXiv preprint arXiv:2408.07272 (Aug. 2024).
- [21] T. Wang, Z. He, W.-Y. Yu, X. Fu, X. Han, Large language models are good multi-lingual learners: When LLMs meet cross-lingual prompts, in: Proc. Int. Conf. Comput. Linguist. (COLING), 2025, pp. 4442–4456.
- [22] H. Abgaryan, A. Harutyunyan, T. Cazenave, LLMs can schedule, arXiv preprint arXiv:2408.06993 (Aug. 2024).

- [23] H. Abgaryan, T. Cazenave, A. Harutyunyan, Starjob: Dataset for LLM-driven job shop scheduling, arXiv preprint arXiv:2503.01877 (Mar. 2025).
- [24] Z. Tang, C. Huang, X. Zheng, S. Hu, Z. Wang, D. Ge, B. Wang, ORLM: Training large language models for optimization modeling, arXiv preprint arXiv:2405.17743 (May 2024).
- [25] Anonymous, DRoC: Elevating large language models for complex vehicle routing via decomposed retrieval of constraints, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2025.
- [26] R. Zhang, H. Du, Y. Liu, D. Niyato, J. Kang, Z. Xiong, A. Jamalipour, D. In Kim, Generative AI agents with large language model for satellite networks via a mixture of experts transmission, IEEE J. Sel. Areas Commun. 42 (12) (2024) 3581–3596.
- [27] J. Li, R. Wickman, S. Bhatnagar, R. K. Maity, A. Mukherjee, Abstract operations research modeling using natural language inputs, Inf. 16 (2) (2025) 128.
- [28] H. Deng, B. Zheng, Y. Jiang, T. H. Tran, CAFA: Coding as autoformulation can boost large language models in solving linear programming problem, in: Workshop on Mathematical Reasoning and AI at NeurIPS'24, 2024.
- [29] T. Wang, W.-Y. Yu, R. She, W. Yang, T. Chen, J. Zhang, Leveraging large language models for solving rare MIP challenges, arXiv preprint arXiv:2409.04464 (Sep. 2024).

- [30] C. Jiang, X. Shu, H. Qian, X. Lu, J. Zhou, A. Zhou, Y. Yu, LLMOPT: Learning to define and solve general optimization problems from scratch, arXiv preprint arXiv:2410.13213 (Oct. 2024).
- [31] Y. Zhou, J. Wang, Y. Kuang, X. Li, W. Luo, J. HAO, F. Wu, LLM4Solver: Large language model for efficient algorithm design of combinatorial optimization solver (2024).
- [32] Z. Jiao, M. Sha, H. Zhang, X. Jiang, W. Qi, City-LEO: Toward transparent city management using LLM with End-to-End optimization, arXiv preprint arXiv:2406.10958 (Jun. 2024).
- [33] B. Zhang, P. Luo, OR-LLM-agent: Automating modeling and solving of operations research optimization problem with reasoning large language model, arXiv preprint arXiv:2503.10009 (Mar. 2025).
- [34] Y. Chen, J. Xia, S. Shao, D. Ge, Y. Ye, Solver-informed RL: Grounding large language models for authentic optimization modeling, arXiv preprint arXiv:2505.11792 (May 2025).
- [35] X. Huang, Q. Shen, Y. Hu, A. Gao, B. Wang, LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages, in: Proc. Findings Assoc. Comput. Linguistics: NAACL, 2025, pp. 2678–2710.
- [36] C. Liu, E. Noriega-Atala, A. Pyarelal, C. T. Morrison, M. Cafarella, Variable extraction for model recovery in scientific literature, arXiv preprint arXiv:2411.14569 (Nov. 2024).

- [37] M. Mostajabdaveh, T. T. Yu, R. Ramamonjison, G. Carenini, Z. Zhou, Y. Zhang, Optimization modeling and verification from problem specifications using a multi-agent multi-stage LLM framework, Inf. Syst. Oper. Res. 62 (4) (2024) 599–617.
- [38] A. Nammouchi, C. Chaabani, A. Theocharis, A. Kassler, Towards explainable renewable energy communities operations using generative AI, in: Proc. IEEE Innov. Smart Grid Technol. Eur. (ISGT EUROPE), 2024, pp. 1–5.
- [39] M. J. Abdel-Rahman, Y. Alslman, D. Refai, A. Saleh, M. A. Abu Loha, M. Y. Hamed, Teaching llms to think mathematically: A critical study of decision-making via optimization, arXiv preprint arXiv:2508.18091 (2025).
- [40] P. T. Amarasinghe, S. Nguyen, Y. Sun, D. Alahakoon, AI-copilot for business optimisation: A framework and a case study in production scheduling, arXiv preprint arXiv:2309.13218 (Sep. 2023).
- [41] H. Chen, G. E. Constante-Flores, C. Li, Diagnosing infeasible optimization problems using large language models, Inf. Syst. Oper. Res. 0 (2024) 1–15.
- [42] Z. Wang, Z. Zhu, Y. Han, Y. Lin, Z. Lin, R. Sun, T. Ding, OptiBench: Benchmarking large language models in optimization modeling with equivalence-detection evaluation, in: Int. Conf. Learn. Represent. (ICLR), 2024.

- [43] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, X. Chen, Large language models as optimizers, in: Proc. Int. Conf. Learn. Represent. (ICLR), 2024.
- [44] B. Li, K. Mellou, B. Zhang, J. Pathuri, I. Menache, Large language models for supply chain optimization, arXiv preprint arXiv:2307.03875 (Jul. 2023).
- [45] Y. Zhang, Q. Kang, W. Y. Yu, H. Gong, X. Fu, X. Han, T. Zhong, C. Ma, Decision information meets large language models: The future of explainable operations research, arXiv preprint arXiv:2502.09994 (Feb. 2025).
- [46] G. Sun, W. Xie, D. Niyato, H. Du, J. Kang, J. Wu, S. Sun, P. Zhang, Generative AI for advanced UAV networking (Nov. 2025).
- [47] Y. Hao, Y. Zhang, C. Fan, Planning anything with rigor: General-Purpose zero-shot planning with LLM-based formalized programming, arXiv preprint arXiv:2410.12112 (Oct. 2024).
- [48] S. Kadıoğlu, P. Pravin Dakle, K. Uppuluri, R. Politi, P. Raghavan, S. Rallabandi, R. Srinivasamurthy, Ner4Opt: Named entity recognition for optimization modelling from natural language, Constraints 29 (3) (2024) 261–299.
- [49] M. Luzzi, F. Guerriero, M. Maratea, G. Greco, M. Garofalo, ChatGPT and operations research: Evaluation on the shortest path problem, Soft Comput. 29 (2025) 1–12.

- [50] S. Li, J. Kulkarni, I. Menache, C. Wu, B. Li, Towards foundation models for mixed integer linear programming, arXiv preprint arXiv:2410.08288 (Oct. 2024).
- [51] I. Gemp, R. Patel, Y. Bachrach, M. Lanctot, V. Dasagi, L. Marris, G. Piliouras, S. Liu, K. Tuyls, Steering language models with gametheoretic solvers, in: Proc. Agentic Markets Workshop at Int. Conf. Mach. Learn. (ICML), 2024.
- [52] S. Yao, F. Liu, X. Lin, Z. Lu, Z. Wang, Q. Zhang, Multi-objective evolution of heuristic using large language model, in: Proc. AAAI Conf. Artif. Intell., Vol. 39, 2025, pp. 27144–27152.
- [53] Z. Xiao, D. Zhang, Y. Wu, L. Xu, Y. J. Wang, X. Han, X. Fu, T. Zhong, J. Zeng, M. Song, G. Chen, Chain-of-experts: When LLMs meet complex operations research problems, in: The Twelfth International Conference on Learning Representations, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ
- [54] Gurobi Optimization, LLC, Gurobi Optimizer Reference Manual (Version 12.0) (May 2025).
  URL https://docs.gurobi.com/\_/downloads/optimizer/en/12.0/pdf/