OR-LLM-BENCH: A PIPELINE FOR SCALABLE AND VERIFIABLE TEXT-TO-OPTIMIZATION SYNTHESIS

Zhiqi Gao[1], **Albert Ge**[1], **Alexander Berenbeim**[2], **Nathaniel D. Bastian**[2] **& Frederic Sala**[1]
[1]Department of Computer Sciences, University of Wisconsin-Madison
[2]Department of Electrical Engineering & Computer Science, United States Military Academy
{zhiqi, fredsala}@cs.wisc.edu, afge@wisc.edu
{alexander.berenbeim, nathaniel.bastian}@westpoint.edu

ABSTRACT

Operations research (OR)-style modeling poses challenges for large language models (LLMs). It requires long-context consistency, producing precise mathematical formulations, and the ability to infer implicit constraints. To study these challenges under controlled conditions, we build a verifiable synthetic pipeline that generates large-scale certified optimization problem instances. Using this pipeline, we obtain several insights: first, direct natural language translation of optimization problems runs into an *effective context limit*, beyond which frontier models abruptly fail to maintain global variable–constraint consistency—despite remaining within nominal context window length. Second, naive divide-and-conquer scaling strategies struggle due to context explosion and semantic fragmentation. Third, while frontier models can reliably infer high-level optimization structure they struggle to correctly bind large, dense numerical data to variables at scale. Taken together, these findings identify important limitations for current LLM-based optimization approaches. For example, we synthesize an OR task where GPT-5 nano has an effective reasoning context limit of only $\sim$2,000 tokens and suffers a more than 50% performance drop.

1 INTRODUCTION

Operations research (OR) is a cornerstone of modern industrial intelligence, powering critical processes in logistics, energy, and supply chain management. While large language models (LLMs) have shown remarkable progress in general-purpose coding and math applications, solving complex optimization problems is nevertheless a challenging hurdle even for frontier models. An LLM must not only comprehend a natural language narrative but also potentially infer unstated constraints, map entities to variables at scale, and generate executable, error-free optimization code.

Several works have sought to benchmark LLM capabilities on such OR optimization problems, but *building realistic problems at scale is challenging*. The NL4Opt competition (Ramamonjison et al., 2022) treated optimization modeling as a two-stage semantic entity extraction and intermediate representation generation task. This dataset is limited to small-scale linear programs with an average of only two variables and constraints. More recent evaluations, such as the ORQA benchmark (Mostajabdaveh et al., 2025), rely on expert-curated problems from 20 diverse domains. However, these benchmarks fail to address two fundamental barriers. First, they lack a mechanism for **verifiable high-volume synthesis of optimization modeling problems**; relying on manual curation prevents the creation of the massive datasets needed to train robust models, while unverified synthetic data risks mathematical infeasibility. Second, they do not address **context-scaling demands imposed by long-horizon optimization problems**. Real-world industrial optimization requires processing dense, high-dimensional numerical inputs—such as hourly energy scheduling over a full year—together with natural-language problem specifications and operational constraints. These problems can quickly exhaust standard LLM context windows. Consequently, these benchmarks fail to challenge the limits of LLMs or verify their performance in industrial-scale environments.

In this paper, we introduce OR-LLM-Bench, a verifiable pipeline designed to reliably scale synthetic operations research benchmarks. Our main idea is to **decouple the *natural language generation***

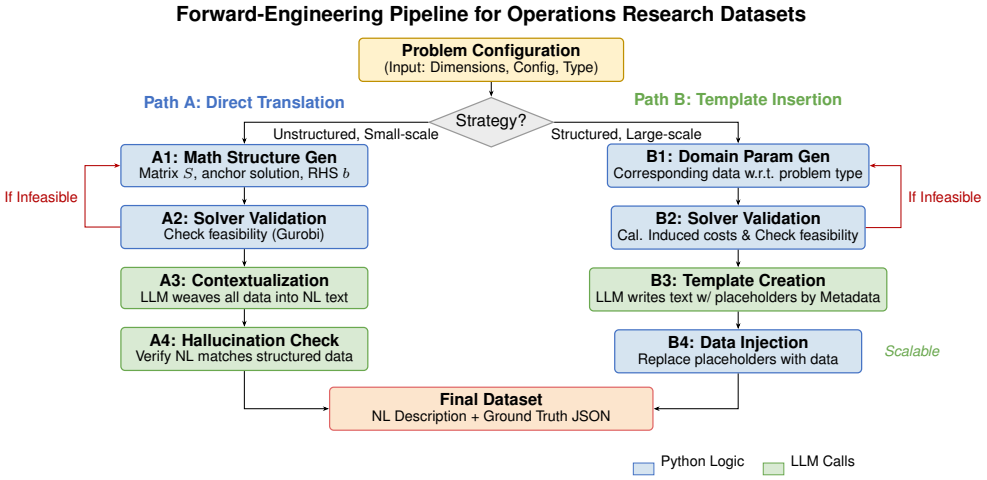


Figure 1: Synthetic OR Data Generation Pipeline

of the problem **from the** ***underlying mathematical representation of the problem*** in a way that permits scaling both aspects reliably. At a high level, we symbolically generate the mathematical structure of the optimization problem, and then use an LLM to equip the problem with a natural language contextualization. This approach guarantees: **1) mathematical validity**, where every problem instance is guaranteed to be feasible and paired with a *verified, solver-generated theoretical optimum*, and **2) domain realism**, allowing the LLM to generate problems within authentic contexts. Furthermore, this dataset enables the creation of large-scale problems—ranging from Mixed-Integer Linear Programming (MILP) to Mixed-Integer Quadratic Programming (MIQP)—each paired with a precomputed optimum.

However, as problem complexity increases to thousands of variables, naively representing the full optimization problem in text quickly exceeds the effective model context limits. To overcome this issue, we propose **templated insertion**, a scalable technique for synthetic problem generation that separates *how a problem is described* from *the numerical values it contains*. Instead of asking the LLM to explicitly enumerate all variables and coefficients, the model generates a structured natural-language template that captures the problem's objectives and constraints using placeholders. These placeholders are then filled deterministically with large-scale numerical data. This design keeps the model's context usage constant while enabling the synthesis of industrial-scale optimization benchmarks.

Our contributions are:

- **A Novel Verifiable Synthesis Pipeline**: We develop a scalable framework for generating five distinct classes of verifiable OR problems (including in popular domains like logistics and power systems) with executable optimization package gold solutions.
- **Insights into Modeling vs. Translating**: We demonstrate that "modeling" (providing structural architecture) is intrinsically easier for LLMs than "translating" (mapping instances), and use this to create high-fidelity datasets.
- **Stress-Testing Scaling**: We introduce a new scaling test to identify the exact variable/constraint limits where current frontier models fail, providing a benchmark for future context-window and reasoning improvements.

2 Related Work

The automation of translating natural language into executable optimization models lies at the intersection of operations research (OR) and neurosymbolic AI, spanning three primary tasks: entity extraction, reasoning, and synthetic generation.

Benchmarks and Reasoning Limits. Early efforts like **NL4Opt** (Ramamonjison et al., 2022) treated modeling as a two-stage process that involves entity recognition and forming intermediate representation. However, recent benchmarks such as **ORQA** (Mostajabdaveh et al., 2025) and **NLMOptimizer** (Berenbeim et al., 2025) highlight a fundamental "reasoning deficit," showing that models struggle with implicit constraints and logical hallucinations when moving beyond small-scale, standardized linear programs. Recent work by Huang et al. (2025b) further characterizes the gap between natural language descriptions and formal mathematical structures. For example, NLMOptimizer shows that even tiny OR problems can pose serious challenges to models.

Reasoning, Search, and Agentic Architectures. To handle industrial complexity, frameworks like **OptiMUS** (AhmadiTeshnizi et al., 2024) and **Chain-of-Experts** (Xiao et al., 2024) use modular decomposition. Other approaches focus on specialized training or end-to-end formulation; for instance, **ORLM** (Huang et al., 2025a) introduces a customizable training framework for automated modeling, while **LLMOPT** (Jiang et al., 2025) attempts to learn to define and solve problems from scratch, while **SolverLLM** (Li et al., 2025) employs Monte Carlo Tree Search (MCTS) to explore formulation spaces. Although these improve scalability, they remain susceptible to error propagation or high inference costs. Similarly, **OptiChat** (Chen et al., 2025) focuses on LLMs as interfaces rather than generators. Recent agentic frameworks have attempted to mitigate this; Zhang et al. (2025) introduce reasoning-enhanced agents for OR, while Ding et al. (2025) utilize test-time reinforcement learning to automate modeling. Additionally, Zhang (2025) explores enabling mathematical modeling directly through LLMs.

Synthetic Data. Scarcity of expert data has led to frameworks like **OptMATH** (Lu et al., 2025), which uses "back-translation" to reverse-engineer descriptions from formulas. In contrast, our **forward engineering** paradigm co-generates descriptions and models from a simulated "world state," ensuring constructive feasibility and naturalistic ambiguity. Finally, we adopt rigorous verification standards similar to **OptiBench** (Wang et al., 2025) to ensure semantic correctness via solver-driven ground truth. This follows a growing trend of leveraging synthetic pipelines for reasoning tasks; Liu et al. (2025) and Goldie et al. (2025) have shown that verifiable synthetic data is crucial for scaling logical reasoning. Similarly, Seegmiller et al. (2025) emphasize the need for fine-grained analysis in data synthesis pipelines to improve mathematical reasoning.

3 METHODS

Below, we describe our overall approach and describe each component in depth.

3.1 FORWARD ENGINEERING VS. BACKWARD ENGINEERING

Instead of starting from an optimal solution and constructing constraints around it (often described as a *backward* engineering approach) we use a *forward engineering* framework. Our method begins by simulating a *world state*: a coherent set of business parameters, resource limits, and logical rules. This approach offers three distinct advantages:

- **Efficient Synthesis:** By defining the solution space first, we guarantee that the generated problem is mathematically feasible and bounded by construction. This eliminates the need for computationally expensive rejection sampling of infeasible instances.
- **Semantic Realism:** Rather than describing equations directly, the LLM generates a narrative grounded in the simulated *world state*, producing problem statements that mirror how optimization tasks are specified in real industrial settings.
- **Structural Control:** By specifying the *World State* up front, we directly control the induced optimization structure and domain semantics. For example, resource allocation problems naturally produce sparse interaction matrices, whereas transportation problems induce bipartite graph structures.

3.2 SYNTHETIC DATASET GENERATION PIPELINE

In our forward engineering approach, the pipeline begins by constructing a mathematically valid optimization problem in standard form. This ensures that every generated problem is feasible, bounded, and has a known optimal solution before any natural-language description is produced.

Algorithm 1 Direct Translation Dataset Generation

Input: Dimensions n, m, Sparsity S
▷ **Phase 1: Construction (Guaranteed Feasibility)**
$A \leftarrow \text{RandomMatrix}(m, n, \text{sparsity} = S)$
$x_{\text{anchor}} \leftarrow \text{RandomVector}(n, \min = 0)$
$s \leftarrow \text{RandomVector}(m, \min = 0.5)$
$b \leftarrow Ax_{\text{anchor}} \pm s$ ▷ Constructs b s.t. x_{anchor} is feasible
$c \leftarrow \text{RandomVector}(n)$
▷ **Phase 2: Verification (Ensured Optimality)**
$x^*, z^*, \text{status} \leftarrow \text{GurobiSolve}(A, b, c)$
if status == UNBOUNDED **then**
    return Retry ▷ Reject unbounded directions
else if status $\neq$ OPTIMAL **then**
    return Retry ▷ Reject numerical instability
end if
$\mathcal{D}_{\text{struct}} \leftarrow \{A, b, c, \text{senses}, \text{types}\}$
$\mathcal{T}_{\text{text}} \leftarrow \text{LLM}(\text{SystemPrompt}, \mathcal{D}_{\text{struct}})$
Output: Pair $(\mathcal{T}_{\text{text}}, \mathcal{D}_{\text{struct}})$

To support both small- and large-scale problems, we adopt two complementary generation strategies. For general linear programs that fit within standard context limits, we use *direct translation*. For high-dimensional or highly structured problems, we instead apply *template-based insertion*, which we describe in Section 3.2.2.

3.2.1 Approach 1: Direct Translation

We use this approach to craft a set of *resource allocation* problems where the interaction matrix A is sparse and unstructured. It is suitable for small-to-medium instances where the full problem description fits within the context window of standard LLMs.

1. Mathematical Structure Generation. We first generate the parameters for a linear programming problem in standard form:

$$\begin{aligned} \text{minimize} \quad & c^T x \\ \text{subject to} \quad & Ax \gtreqless b \\ & x \geq 0 \end{aligned} \tag{1}$$

To ensure control over the problem's characteristics, we use an *anchor solution*, as follows:

1. **Matrix Construction (A):** We initialize $A \in \mathbb{R}^{m \times n}$ with random values and apply a sparsity mask to simulate real-world interactions.
2. **Anchor Solution (x_{anchor}):** We sample a feasible solution $x_{\text{anchor}} \geq 0$.
3. **RHS Derivation (b):** The vector b is derived via $b_i = (Ax_{\text{anchor}})_i + s_i$, ensuring feasibility by construction.

2. Natural Language Translation The structured representation is passed to an LLM (GPT-5 in our case) with a prompt enforcing **data embedding**: all numerical coefficients from A, b, and c must be naturally woven into the narrative. For example, a variable with objective coefficient $c_1 = 6.86$ and constraint interaction $a_{0,1} = 0.8$ is rendered as a contextual entity (e.g., "each package adds 6.86 in contribution and uses 0.8 units of emissions allowance"). A full example is provided in Appendix A.

3.2.2 Scalable Generation Strategy

Adopted Solution: Template-Based Scalable Generation. For structured problems or large-scale instances (100+ variables), *direct translation* becomes impractical due to an **effective context limit** (Section 4.1.2). We initially explored hierarchical decomposition via block-diagonal structures ($A = \text{diag}(A_1, \ldots, A_k)$), but discarded it due to context explosion (>100K tokens), semantic

Algorithm 2 Template-Based Generation Pipeline

Input: Problem Type T, Dimensions n, m
$\mathcal{D}_{\text{struct}} \leftarrow \text{GenerateParameters}(T, n, m)$
$x^*, z^* \leftarrow \text{SolverVerification}(\mathcal{D}_{\text{struct}})$
if Small Scale / Unstructured **then**
    $\mathcal{T}_{\text{text}} \leftarrow \text{LLM}(\text{Prompt}_{\text{Direct}}, \mathcal{D}_{\text{struct}})$
else ▷ Large Scale / Structured
    $\mathcal{T}_{\text{template}} \leftarrow \text{LLM}(\text{Prompt}_{\text{Template}}, \text{Meta}(\mathcal{D}_{\text{struct}}))$
    $\mathcal{T}_{\text{text}} \leftarrow \text{InsertData}(\mathcal{T}_{\text{template}}, \mathcal{D}_{\text{struct}})$
end if
Output: Pair $(\mathcal{T}_{\text{text}}, \mathcal{D}_{\text{struct}})$

fragmentation, and topological inflexibility. Instead, we developed a *template-based* pipeline that decouples data scale from linguistic complexity.

1. Structured Parameter Generation. Instead of a generic matrix A, we generate domain-specific parameters. For example, when generating facility location-themed problems:

- Locations: Coordinates for N facilities and M customers.
- Parameters: Fixed costs f_i, demands d_j, and transport rates r.
- Induced Costs: The cost matrix C_{ij} is computed deterministically via Euclidean distance, avoiding LLM arithmetic errors.

2. Template Generation via LLM. The LLM is tasked with generating e.g., a high-quality "business memo" template that describes the problem logic but *excludes* the numerical data. The system prompt explicitly instructs the model to use placeholders (e.g., {`COST_MATRIX`}, {`DEMANDS`}) instead of inventing numbers.

- **Input:** Metadata about the problem size and type (e.g., "3 factories, 50 stores").
- **Output:** A narrative explaining the objective and constraints, with a designated "Data Annex" containing the placeholders.

3. Deterministic Data Insertion. The pipeline programmatically replaces placeholders with generated data formatted as Markdown tables, decoupling linguistic complexity from numerical complexity and enabling scaling to thousands of variables with constant LLM context overhead.

3.3 Redefining the Task: From Translation to Structural Modeling

The adoption of the template-based pipeline fundamentally transforms the nature of the OR benchmark. By abstracting numerical data into placeholders, we shift the challenge from a **sentence-to-code translation** task to a **mathematical modeling** task.

3.3.1 Abstraction of Variable Logic

In standard benchmarks, the model often acts as a parser spending its capacity extracting coefficients from text. In our framework, the model must instead deduce the *logical relationship* between data structures and decision variables. For example, rather than identifying that "Constraint 1 has a limit of 50," the model must recognize that:

$$\forall i \in \mathcal{I}, \quad \sum_{j \in \mathcal{J}} x_{ij} \leq \texttt{Capacity}[i]. \tag{2}$$

This forces the LLM to understand the *schema* of the provided matrices (e.g., dimensions, sparsity patterns) and correctly instantiate variables based on these abstract definitions, decoupling reasoning from arithmetic perception.

3.3.2 INDUCED CONSTRAINTS AND PHYSICAL PRIORS

Beyond direct data mapping, our framework also introduces the complexity of **induced constraints**. In these instances, the parameters required for the optimization model are not explicitly present in the data annex but must be derived via domain knowledge or physical laws.

Prime examples are the *facility location*-style problem or *drone routing* scenarios, where the cost matrix C_{ij} is often latent:

- **Given:** A list of coordinates for Depots D and Customers C.
- **Latent Parameter:** The travel cost c_{ij}, which implies a Euclidean distance calculation: $c_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

This requires the model to identify the necessary bridge formula (e.g., Euclidean distance, gravity models, or specific physical decay rates) to transform raw attributes into actionable optimization parameters, mimicking the implicit reasoning required in real-world engineering tasks.

3.4 EVALUATION

We prompt LLMs to generate an executable **Gurobi**[1] **code snippet**, which is then executed by a Python interpreter. A critical design choice in our benchmark is the exclusion of infeasible instances from the ground truth dataset.

If the ground truth were infeasible, an LLM could generate fundamentally broken code with trivial contradictions (e.g., $x > 1$ and $x < 0$) that coincidentally results in an infeasible status. A solver-based grader would incorrectly classify this unrelated failure as a correct prediction (a false positive). By restricting our benchmark exclusively to feasible problems, we ensure that any infeasible result returned by the generated code is correctly penalized as a reasoning error.

4 EXPERIMENTS

4.1 EVALUATION RESULT

We evaluated our pipeline using a mix of proprietary frontier models (**GPT-5**, **GPT-5-Nano**) and efficient open-weight models (**Qwen2.5-7B**, **Qwen2.5-1.5B**). To test the limits of these models in multiple perspectives, we stratified the benchmark into five distinct problem categories:

- **Direct:** Baseline resource allocation problems generated via direct translation (Section 3.2.1). This category focuses on standard linear programming formulations where decision variables represent resource quantities subject to budget and capacity constraints.
- **Template (Std):** Transportation problems, a classic bipartite matching scenario generated via standard templates. The objective is to minimize shipping costs between multiple supply sources and demand destinations while satisfying inventory limits, we mainly test the ability.
- **Template (Hard):** Disaster response logistics, a large-scale multi-period mixed-integer linear programming (MILP) problem. This complex scenario requires managing vehicle fleets, route security risks, and supply shortages across a dynamic operational timeline.
- **Induced (Geo):** Facility location problems that require implicit geometric reasoning. The model must derive transportation costs by calculating Euclidean distances between candidate facility coordinates and customer locations to minimize total setup and shipping costs.
- **Induced (Phys):** Power transmission network design, the most challenging category requiring the derivation of constraints from physical formulas. The model must construct a quadratic cost function for power loss based on Ohm's Law and Joule heating (I^2R) to optimize grid topology.

[1] Gurobi is a popular solver package.

Generation Cost. All problem instances are generated using a single call to **GPT-5**. With another call for quality check. For *direct translation* problems, the full structured representation (variables, constraints, and interaction matrix) is included in the prompt, is approximately $\sim\$0.1$ per problem, and varies with problem scale. The cost for *template-based* problems is cheaper.

Main Results. Table 1 presents the Pass@1 accuracy for generating valid optimization models. Here, we treat a model response as correct if and only if it generates a runnable python code snippet using Gurobi that solves to the same objective value as the gold solution. **GPT-5** demonstrates robust capabilities, maintaining $> 90\%$ accuracy across all categories, and good at inducing constraints. In contrast, **GPT-5-Nano** shows a distinct performance cliff on complex reasoning, dropping from 96.0% on **Template (Std)** to 58.0% on **Template (Hard)**. The open-weight models struggle significantly with formulation; **Qwen2.5-7B** fails to generalize to harder templates (0.0%), while the 1.5B variant yields near-zero results globally, indicating insufficient capacity for implicit constraint derivation.

Table 1: Pass@1 Accuracy (%) across different problem types.

Model	Direct	Template Scale		Induced Constraints	
	Vanilla	**Std**	**Hard**	**Geo (Euclidean)**	**Phys (Formula)**
	n=1012	*n=50*	*n=50*	*n=50*	*n=50*
GPT-5	92.59	98.0	90.0	92.0	96.0
GPT-5-Nano	50.15	96.0	58.0	80.0	60.0
Qwen2.5-7B	15.91	22.0	0.0	8.0	2.0
Qwen2.5-1.5B	0.69	0.0	0.0	0.0	0.0

Failure Mode Analysis. Beyond aggregate accuracy, we examine *how* models fail. For **direct translation**, **GPT-5**'s rare failures (7.4%) are almost entirely wrong objectives with near-zero runtime errors, indicating robust code generation but occasional coefficient misalignment on larger instances. **GPT-5-Nano** follows a similar pattern at lower accuracy: 35.3% wrong objectives and 4.4% infeasible formulations, suggesting the model grasps problem structure but loses track of specific coefficients at scale.

The open-weight models fail qualitatively differently. **Qwen2.5-7B** produces wrong objectives in 56.2% of cases and infeasible formulations in 12.8%, while 8.0% result in runtime errors. **Qwen2.5-1.5B** has 20.4% cause runtime errors (malformed code) and 14.0% produce no valid solver status, revealing limitation in its code generation ability.

For **induced constraint** problems, failure modes shift from data-binding errors to *modeling* errors. On **Induced (Geo)**, even GPT-5 occasionally misapplies the cost model—e.g., incorrectly multiplying transport cost by customer demand within the distance calculation, yielding a subtly wrong objective. GPT-5-Nano computes Euclidean distances correctly in all failed cases but errs in how costs are aggregated into constraints. On **Induced (Phys)**, GPT-5-Nano attempts the I^2R loss derivation in 85% of failures but often miscomputes the quadratic loss coefficient (e.g., incorrect unit conversion between MW and Volts, or wrong resistance-per-km scaling); in 15% of failures, the physics formula is omitted entirely. Open-weight models near-universally fail on induced problems (0–8%), lacking the capacity to derive implicit constraints from domain knowledge. For **Template (Hard)**, the multi-period MILP structure (vehicle fleets, route security, dynamic timelines) overwhelms smaller models, with Qwen2.5-7B achieving 0% by failing to correctly formulate linking constraints across time periods.

4.1.1 Influence of Prompting Techniques

To investigate if advanced prompting improves performance, we conducted an ablation study with **GPT-5-Nano** on a held-out subset of *resource allocation* problems. Table 2 summarizes the accuracy rates. We found that increasing prompt complexity yielded negligible gains. While adding *extra reasoning* or a *second pass* provided marginal improvement (peaking at 50.0%), other techniques were detrimental. Notably, adding *explicit warnings* or *one-shot* examples degraded performance to

45.7% and 44.4% respectively, below the 48.4% baseline. We attribute this to the effective context limit (Section 4.1.2): the primary bottleneck is not instruction quality but the model's capacity to faithfully process dense numerical specifications, which prompting alone cannot address.

Table 2: Ablation study of prompting strategies on GPT-5-Nano (subset, $n = 248$)

Prompting Strategy	**Accuracy (%)**
Base Prompt (with Template)	48.4
Base + Extra Reasoning	**50.0**
Base + Explicit Warnings	45.7
Base + Additional Focus Requirements	47.6
Base + Second Pass Refinement	**50.0**
One-Shot Example	44.4

4.1.2 Context Limitation and Effective Context Length

We investigate the root cause of failure for frontier LLMs in the direct translation task. Our experiments reveal that the primary bottleneck is not necessarily a lack of reasoning capability used for mathematical derivation, but rather the ***effective context limit***—the maximum token window within which a model can maintain robustness in mapping variables and constraints.

Figure 2 illustrates the relationship between prompt token length (a proxy for problem scale) and translation accuracy across the GPT-5 and Qwen model families. To create a smooth accuracy trend, we apply a Gaussian-weighted moving average over a window of k neighbors. This gives more importance to the closest data points while reducing noise from outliers. For GPT-5, we also test some long-context data since its accuracy does not drop drastically after 5000 tokens. We observe three key findings:

- **Non-Linear Trends:** Accuracy exhibits a sharp, non-linear decay as the prompt length increases. For every model evaluated, there exists a critical threshold beyond which performance collapses, suggesting a *phase transition* where the model loses track of the global variable state.
- **Model-Specific Effective Limits:** Different models demonstrate distinct effective context windows. While **GPT-5** maintains near 100% accuracy for prompts up to approximately 2,000 tokens, smaller models fail much earlier. This confirms that the standard "context window" advertised by model providers (often 128k+) does not equate to the *effective* window for dense, logical retrieval tasks. This observation aligns with recent findings on the limits of context scaling; for instance, Shi et al. (2025) and Zhou et al. (2025) demonstrate that reasoning performance often degrades non-linearly even within the nominal window, while Yan et al. (2025) proposes mechanisms to break these length barriers.
- **Scaling in the Qwen Family:** We observe a clear correlation between parameter count and effective context within the Qwen series, from 0.5B to 70B. This validates that larger model capacity contributes to more robust attention over long sequences.

4.2 Fine-Tuning and Curriculum Learning

To validate the utility of our synthetic OR pipeline as a training substrate, we study how different training paradigms affect a model's ability to solve structured optimization problems and generalize beyond the training distribution. We focus on the resource allocation problem family, which requires translating natural-language specifications into precise mathematical constraints and executable solution logic.

Experimental Setup: SFT vs. RL We consider two training paradigms on synthetic OR data:

- **Supervised Fine-Tuning (SFT):** Models are trained with standard cross-entropy loss on fully specified solution traces generated by our pipeline.

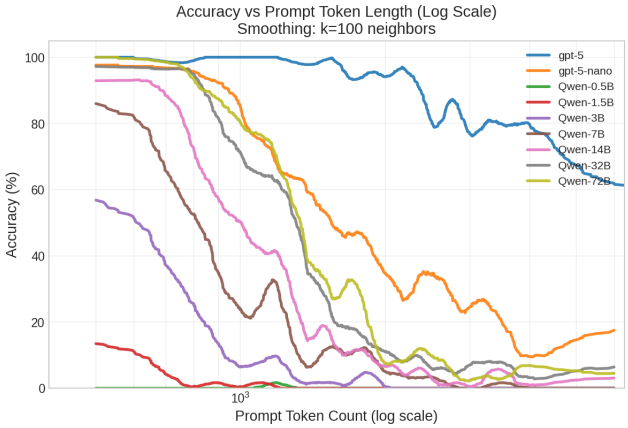


Figure 2: Translation Accuracy vs. Prompt Token Length. We observe a distinct *effective context limit* for each model, where accuracy remains at 80% or more before undergoing a rapid decay.

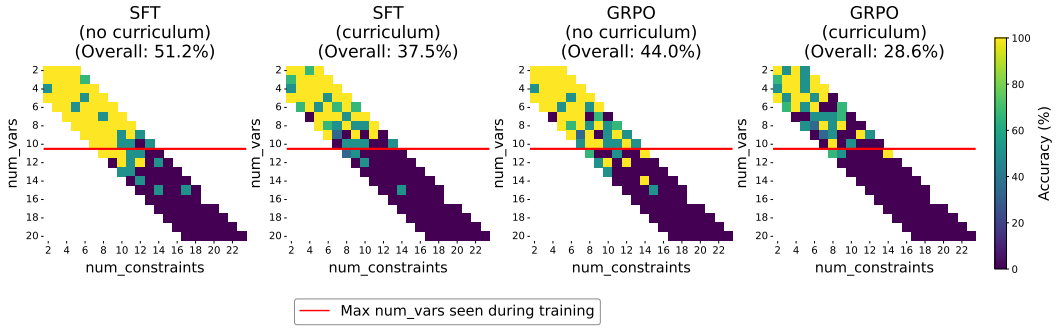


Figure 3: Accuracy heatmaps for the resource allocation task across number of variables and constraints. Models trained without an explicit curriculum achieve comparable or better generalization than curriculum-trained counterparts. The red line denotes the maximum problem difficulty in the training dataset. Across both training regimes, SFT consistently outperforms GRPO at the 7B scale.

- **Reinforcement Learning (GRPO):** Models are optimized using group relative policy optimization, where reward is assigned based on correctness of the final solution.

Unless otherwise specified, experiments are conducted on **Qwen2.5-7B**. We additionally study the effect of curriculum learning by comparing models trained on progressively increasing problem sizes (staged training) versus models trained on the full distribution from the start.

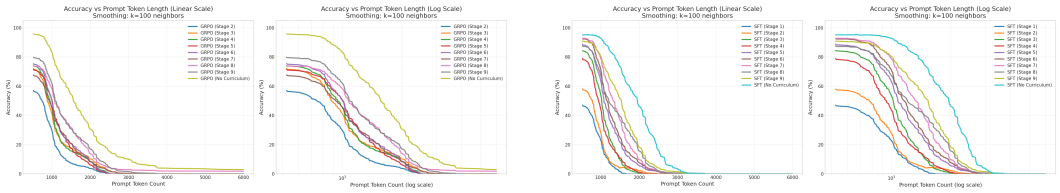
Generalization Across Problem Scale. Figure 3 shows accuracy as a function of the number of variables and constraints, evaluated on problem sizes that extend beyond the training regime.

Across all settings, we observe that **SFT consistently outperforms RL-based training at the 7B scale**. In particular:

- **SFT achieves higher overall accuracy** and maintains strong performance across a broader range of variable–constraint regimes.
- **RL-trained models exhibit brittle behavior**, with sharp drops in accuracy as problem size increases, especially in underconstrained or overconstrained regimes.

While RL occasionally succeeds on isolated larger instances, these gains are not systematic and do not translate into higher overall generalization.

Effect of Curriculum Learning. Somewhat surprisingly, we find that **curriculum learning is not necessary** for strong performance in either SFT or RL:



(a) GRPO (b) SFT

Figure 4: Token-level accuracy as a function of problem length for Resource Allocation. SFT maintains higher accuracy across longer sequences, while GRPO exhibits faster degradation as problem complexity increases.

- Models trained directly on the full distribution perform as well as, and often better than, models trained with staged curricula.
- This holds for both SFT and GRPO, suggesting that sufficiently diverse synthetic data can implicitly induce a curriculum through data coverage alone.

These results indicate that explicit curriculum design provides limited additional benefit once the synthetic data generator reliably spans the underlying problem space.

Why Does SFT Work Better Than RL? We hypothesize that the superior performance of SFT stems from the nature of OR problems themselves. Solving optimization problems typically requires:

- Faithful parsing of every constraint and coefficient,
- Correct symbolic translation into mathematical expressions,
- Precise execution of multi-step reasoning with little tolerance for error.

SFT provides dense, token-level supervision at each step of this process, whereas RL supplies only sparse, outcome-level feedback. As a result, RL may lack a sufficient signal to reliably learn the fine-grained procedural structure required for OR problem solving, even when rewards are accurate.

Takeaway: At the 7B scale, **explicit supervision remains the most effective way to train models for structured optimization tasks**. While RL is attractive for its potential to reduce annotation effort, our results suggest that, for OR-style reasoning, high-quality synthetic supervision offers a stronger and more reliable training signal.

5 Conclusion and Future Directions

In this work, we introduced OR-LLM-Bench, a pipeline for synthesizing verifiable, scalable Operations Research (OR) benchmarks using a forward-engineering approach. By decoupling the generation of mathematical structure from natural language contextualization, our method guarantees feasibility and provides solver-generated ground truth for problems ranging from linear resource allocation to complex quadratic constraints. From evaluation side, we identify a critical effective context limit in frontier models, where accuracy in translating dense numerical data collapses well before the theoretical context window is reached. Besides this, we found that supervised fine-tuning consistently outperforms reinforcement learning for these tasks, as the sparse signals in outcome-based RL are insufficient for learning the precise, multi-step reasoning required for valid constraint formulation.

Based on our findings, we propose several possibilities for future research. First, to address **context limitations**, future work can explore decomposition methods to enable reliable translation of large-scale instances beyond current context thresholds. Second, we are interested in **dense reward shaping for RL**: given that standard GRPO struggles with the precision required for OR, research could investigate hybrid RL approaches using process reward-models to outperform SFT. Finally, we aim to **expand problem types** to encompass broader scientific domains beyond the current set.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. Optimus: Scalable optimization modeling with (mi)lp solvers and large language models, 2024. URL https://arxiv.org/abs/2402.10172.

Alexander Michael Berenbeim, Ryan McNeil, Timeo Williams, and Nathaniel D. Bastian. NL-MOptimizer: A neurosymbolic framework and benchmark for operations research optimization problems from natural language, 2025. URL https://openreview.net/forum?id=skctEx59f2.

Hao Chen, Gonzalo Esteban Constante-Flores, Krishna Sri Ipsit Mantri, Sai Madhukiran Kompalli, Akshdeep Singh Ahluwalia, and Can Li. Optichat: Bridging optimization models and practitioners with large language models, 2025. URL https://arxiv.org/abs/2501.08406.

Zezhen Ding, Zhen Tan, Jiheng Zhang, and Tianlong Chen. Or-r1: Automating modeling and solving of operations research optimization problem via test-time reinforcement learning, 2025. URL https://arxiv.org/abs/2511.09092.

Anna Goldie, Azalia Mirhoseini, Hao Zhou, Irene Cai, and Christopher D. Manning. Synthetic data generation multi-step rl for reasoning tool use, 2025. URL https://arxiv.org/abs/2504.04736.

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, November 2025a. ISSN 1526-5463. doi: 10.1287/opre.2024.1233. URL http://dx.doi.org/10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages, 2025b. URL https://arxiv.org/abs/2405.13144.

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, 2025. URL https://arxiv.org/abs/2410.13213.

Dong Li, Xujiang Zhao, Linlin Yu, Yanchi Liu, Wei Cheng, Zhengzhang Chen, Zhong Chen, Feng Chen, Chen Zhao, and Haifeng Chen. Solverllm: Leveraging test-time scaling for optimization problem via llm-guided search, 2025. URL https://arxiv.org/abs/2510.16916.

Junteng Liu, Yuanxiang Fan, Zhuo Jiang, Han Ding, Yongyi Hu, Chi Zhang, Yiqi Shi, Shitong Weng, Aili Chen, Shiqi Chen, Yunan Huang, Mozhi Zhang, Pengyu Zhao, Junjie Yan, and Junxian He. Synlogic: Synthesizing verifiable reasoning data at scale for learning logical reasoning and beyond, 2025. URL https://arxiv.org/abs/2505.19641.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. Optmath: A scalable bidirectional data synthesis framework for optimization modeling, 2025. URL https://arxiv.org/abs/2502.11102.

Mahdi Mostajabdaveh, Timothy Tin Long Yu, Samarendra Chandan Bindu Dash, Rindra Ramamonjison, Jabo Serge Byusa, Giuseppe Carenini, Zirui Zhou, and Yong Zhang. Evaluating llm reasoning in the operations research domain with orqa. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23):24902–24910, Apr. 2025. doi: 10.1609/aaai.v39i23.34673. URL https://ojs.aaai.org/index.php/AAAI/article/view/34673.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In Marco Ciccone, Gustavo Stolovitzky, and Jacob Albrecht (eds.), *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 28 Nov–09 Dec 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Parker Seegmiller, Kartik Mehta, Soumya Saha, Chenyang Tao, Shereen Oraby, Arpit Gupta, Tagyoung Chung, Mohit Bansal, and Nanyun Peng. Flames: Improving llm math reasoning via a fine-grained analysis of the data synthesis pipeline, 2025. URL https://arxiv.org/abs/2508.16514.

Jingzhe Shi, Qinwei Ma, Hongyi Liu, Hang Zhao, Jeng-Neng Hwang, and Lei Li. Explaining context length scaling and bounds for language models, 2025. URL https://arxiv.org/abs/2502.01481.

Yiwei Wang et al. Optibench: Benchmarking large language models in optimization modeling with equivalence-detection evaluation. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://openreview.net/forum?id=KD9F5Ap878.

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 *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

Yuchen Yan, Yongliang Shen, Yang Liu, Jin Jiang, Mengdi Zhang, Jian Shao, and Yueting Zhuang. Inftythink: Breaking the length limits of long-context reasoning in large language models, 2025. URL https://arxiv.org/abs/2503.06692.

Bowen Zhang, Pengcheng Luo, Genke Yang, Boon-Hee Soong, and Chau Yuen. Or-llm-agent: Automating modeling and solving of operations research optimization problems with reasoning llm, 2025. URL https://arxiv.org/abs/2503.10009.

Guoyun Zhang. Large language model enabled mathematical modeling, 2025. URL https://arxiv.org/abs/2510.19895.

Yang Zhou, Hongyi Liu, Zhuoming Chen, Yuandong Tian, and Beidi Chen. Gsm-infinite: How do your llms behave over infinitely increasing context length and reasoning complexity?, 2025. URL https://arxiv.org/abs/2502.05252.

A Appendix

Example: Data Embedding Transformation

We sample a specific variable and constraint to demonstrate the mapping from structured parameters to natural language narrative.

1. Variable Embedding
Input (Structured):

- `Var_1`: Type `Integer`, Obj Coeff $c_1 = 6.86$
- `Interaction`: Consumes 0.8 of Resource C_0

Output (Narrative):
"**On-Site Retrofit Packages**: Each completed package adds **6.86** in contribution. Each package uses **0.8 units** from our environmental emissions allowance..."

---

2. Constraint Embedding
Input (Structured):

- `Constraint C0`: Sense $\leq$, RHS $b_0 = 8.25$

Output (Narrative):
"**Environmental Emissions Allowance**: Total available is **8.25** allowance units and cannot be exceeded."