OptiVerse: A Comprehensive Benchmark towards Optimization Problem Solving

Xinyu Zhang[1,2*], **Boxuan Zhang**[1,2*], **Yuchen Wan**[1,2], **Lingling Zhang**[1,2 †], **Yixing Yao**[1,2], **Bifan Wei**[1,2], **Yaqiang Wu**[4], **Jun Liu**[1,3]
[1]School of Computer Science and Technology, Xi'an Jiaotong University
[2]Ministry of Education Key Laboratory of Intelligent Networks and Network Security, China
[3]Shaanxi Province Key Laboratory of Big Data Knowledge Engineering, China
[4]Lenovo Research
zhang1393869716@stu.xjtu.edu.cn, {zhanglling,liukeen}@xjtu.edu.cn

Abstract

While Large Language Models (LLMs) demonstrate remarkable reasoning, complex optimization tasks remain challenging, requiring domain knowledge and robust implementation. However, existing benchmarks focus narrowly on Mathematical Programming and Combinatorial Optimization, hindering comprehensive evaluation. To address this, we introduce **OptiVerse**, a comprehensive benchmark of 1,000 curated problems spanning neglected domains, including Stochastic Optimization, Dynamic Optimization, Game Optimization, and Optimal Control, across three difficulty levels: Easy, Medium, and Hard. The experiments with 22 LLMs of different sizes reveal sharp performance degradation on Hard problems, where even advanced models like GPT-5.2 and Gemini-3 struggle to exceed 27% accuracy. Through error analysis, we identify that modeling & logic errors remain the primary bottleneck. Consequently, we propose a **Dual-View Auditor Agent** that improves the accuracy of the LLM modeling process without introducing significant time overhead. OptiVerse will serve as a foundational platform for advancing LLMs in solving complex optimization challenges.

1 Introduction

Large Language Models (LLMs) have revolutionized diverse domains, such as open-ended dialogue (Liu et al., 2025b), mathematical reasoning (Jansen et al., 2025; Yan et al., 2025), code generation (Jansen et al., 2025), visual understanding (Zhang et al., 2025b, 2023, 2025c), temporal analysis (Zhang et al., 2022), and science reasoning (Zhang et al., 2025d,e). However, a pivotal question remains regarding the practical utility of these advancements: *To what extent can LLMs translate their reasoning and programming capabilities into real-world application scenarios?*

Problem

Consider a single-period inventory scenario where daily demand x and order quantity Q are measured in hundreds of bundles. This demand follows the discrete probability distribution $P(x)$ provided below. Given a marginal profit of \$1 per unit ($\alpha = 1$) and a unit overage cost (loss on unsold inventory) of \$1 ($\beta = 1$), determine the optimal order quantity Q that maximizes expected profit.

x	8	9	10	11	12	13	14	15
$P(x)$	0	0.05	0.15	0.20	0.40	0.15	0.05	0

Optimization Result

- Optimal Order Quantity (Q): 12 hundred copies
- Maximum Expected Profit: 10.7 dollar
- Minimum Expected Loss: 0.9 dollar

Category: Stochastic Inventory (SO)

Difficulty: Medium

Figure 1: An illustrative example from **OptiVerse**.

Optimization, serving as a cornerstone discipline underpinning operations research, engineering design, and strategic decision-making (Huang et al., 2025a), presents a quintessential testbed for this inquiry. Far beyond basic arithmetic, optimization demands a sophisticated synthesis of skills. It requires interpreting domain semantics, formulating rigorous mathematical models, and implementing executable code with solvers, as exemplified in Figure 1. The challenge of meeting these rigorous demands, despite the general capabilities of LLMs, has ignited a growing research interest in LLM-based optimization modeling (Liu et al., 2025c).

However, existing evaluation benchmarks suffer from a **distinctly narrow disciplinary scope**, which significantly hinders a comprehensive assessment of model capabilities. Current benchmarks, such as OptiMath (Lu et al., 2025) and OptiBench (Yang et al., 2025b), focus primarily on Mathematical Programming (MP) and Combinatorial Optimization (CO). Crucially, they systematically neglect other essential domains ubiquitous in optimization problems. These overlooked areas include Stochastic Optimization (SO), Dynamic

*These authors contributed equally to this work.
†Corresponding author

Findings of the Association for Computational Linguistics: ACL 2026, pages 3059–3073
July 2-7, 2026

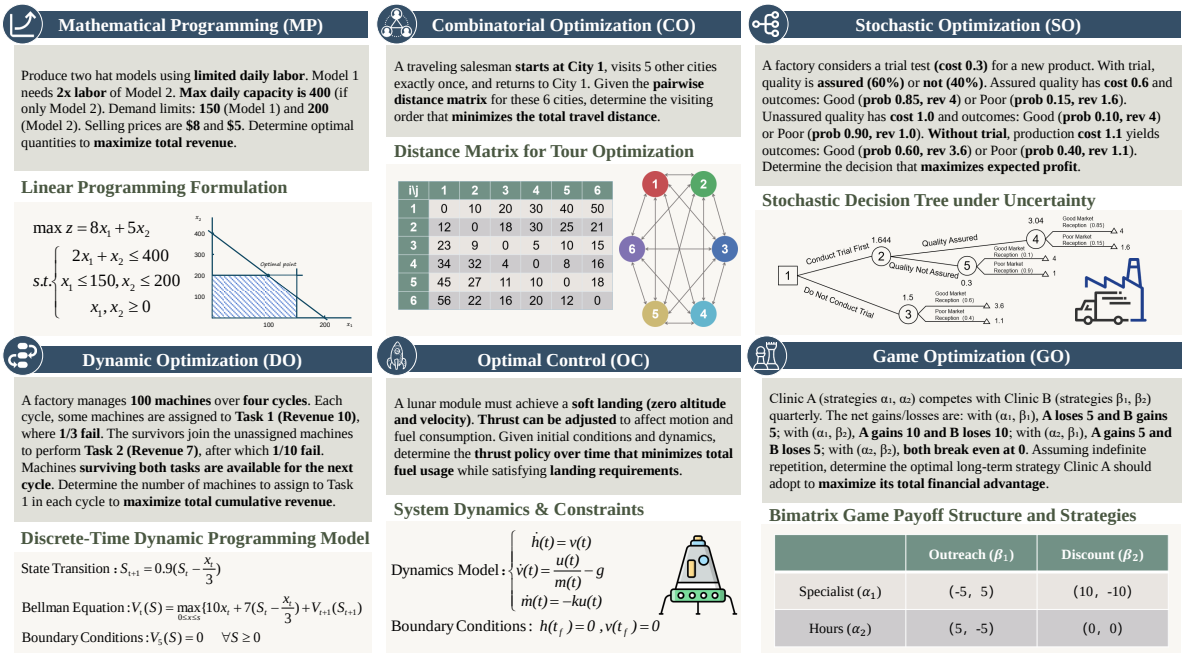

Figure 2: The hierarchical taxonomy of OptiVerse benchmark, which provides comprehensive coverage across six distinct optimization domains to comprehensively evaluate the diverse reasoning capabilities of LLMs.

Optimization (DO), Optimal Control (OC), and Game Optimization (GO) (Halperin, 2022). This omission fails to reflect the true breadth of optimization capabilities, thereby limiting our analysis of LLM generalization across diverse domains.

To bridge this, we introduce **OptiVerse**, a comprehensive benchmark meticulously designed to provide a rigorous, multi-dimensional evaluation of LLM-based optimization solving. **OptiVerse** encompasses 1,000 carefully curated problems spanning the full spectrum of six distinct optimization disciplines: MP, CO, SO, DO, OC, and GO, as show in Figure 2. This holistic coverage enables a systematic assessment of domain-specific strengths and weaknesses that single-discipline benchmarks cannot capture. Furthermore, to simulate the complexity of realistic applications, we organize problems into a hierarchical complexity stratification (Easy, Medium, and Hard), comprising 300, 400, and 300 problems, respectively. This design enables a fine-grained analysis of how model performance degrades as problem sophistication scales.

We conduct extensive and rigorous experiments to evaluate 22 Large Language Models (LLMs; evaluated on textual inputs only) across varying scales, ranging from 8B parameter models such as Qwen3-8B (Yang et al., 2025a) to current flagship frontiers such as Gemini-3-Pro (DeepMind, 2025b) and GPT-5.2 (OpenAI, 2025a). Our experiments yield critical insights: (1) significant performance disparities exist across domains, with success rates in understudied categories (e.g., Optimal Control) often trailing common MP and CO tasks; and (2) while current LLMs exhibit robust performance on Easy tasks, they struggle profoundly with Hard problems, where even top-tier LLMs like Gemini-3-Pro and GPT-5.2 can only reach a maximum of 27% and 25.33% accuracy, respectively.

To investigate the reasons for these failures, we perform a systematic error analysis, revealing that **Modeling & Logic** errors constitute the predominant bottleneck, often manifesting as silent semantic discrepancies where the modeling logic and executable code deviate from the problem intent despite successful code execution. To address this, we propose the **Dual-View Auditor Agent (DVA-Agent)**, which detects and repairs these subtle semantic alignment defects. Experimental results demonstrate that DVA-Agent significantly enhances solving capabilities; for instance, for Qwen3-235B-Instruct, it boosts success rates on Hard and Medium problems by 7.66% and 10.5% respectively. Crucially, the repair mechanism is triggered in only 32.3% of instances, thereby maintaining computational efficiency by avoiding unnecessary iterations on already correct solutions.

2 Related Work

2.1 Benchmarks for Optimization Modeling

Evaluating LLM capabilities in optimization requires benchmarks ranging from foundational prob-

Benchmark	Size	Table	Graph	Difficulty	Answer Form	Problem Category					
						MP	CO	SO	DO	OC	GO
ComplexOR	37	✗	✓	✗	Scalar	✓	✓	✗	✗	✗	✗
NLP4LP	269	✗	✗	✗	Scalar	✓	✗	✗	✗	✗	✗
MAMO	863	✓	✓	✓	Scalar	✓	✓	✗	✗	✗	✗
IndustryOR	100	✓	✗	✗	Scalar	✓	✓	✗	✗	✗	✗
NL4OPT	289	✗	✗	✗	Scalar	✓	✗	✗	✗	✗	✗
Optibench	605	✓	✗	✗	Scalar	✓	✓	✗	✗	✗	✗
OptMATH	166	✓	✓	✗	Scalar	✓	✓	✗	✗	✗	✗
CO-Bench	6483	✗	✓	✗	Scalar	✓	✓	✗	✗	✗	✗
OptiVerse-Easy	300	✓	✓	✓	Vector	✓	✓	✓	✓	✓	✓
OptiVerse-Medium	400	✓	✓	✓	Vector	✓	✓	✓	✓	✓	✓
OptiVerse-Hard	300	✓	✓	✓	Vector	✓	✓	✓	✓	✓	✓
OptiVerse	1000	✓	✓	✓	Vector	✓	✓	✓	✓	✓	✓

Table 1: Comparison with existing benchmarks. OptiVerse benchmark provides structured textual contexts (table and graph), difficulty level, and vector-based solution across six optimization domains: Mathematical Programming, Combinatorial Optimization, Stochastic Optimization, Dynamic Optimization, Optimal Control, Game Optimization.

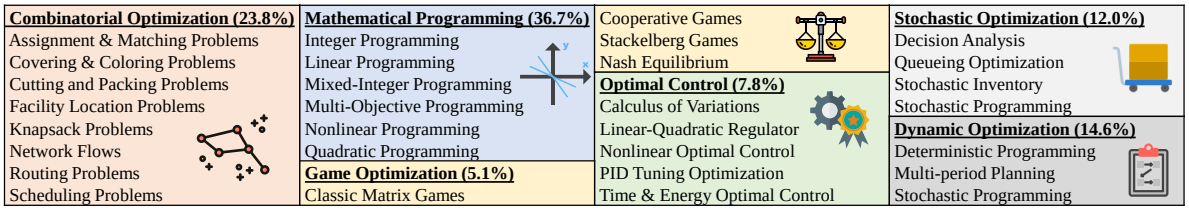

Figure 3: Domain distribution in OptiVerse benchmark, with each color representing a distinct optimization domain.

lems to complex industrial applications. Early datasets like NL4Opt (Ramamonjison et al., 2023) and NLP4LP (AhmadiTeshnizi et al., 2024) established baselines for linear and mixed-integer programming, while recent benchmarks such as OptiBench (Yang et al., 2025b), MAMO (Huang et al., 2025b), and OptMATH (Lu et al., 2025) target advanced mathematical reasoning. IndustryOR (Huang et al., 2025a) and ComplexOR (Xiao et al., 2023) focus on practical operations research constraints. However, current benchmarks predominantly cover MP and CO, leaving other domains of optimization problems unexplored.

2.2 LLM-based Optimization Modeling

While LLMs demonstrate remarkable math reasoning capabilities, they face significant limitations in optimization problem solving involving different constraints. To bridge this gap, researchers develop prompt-based frameworks that utilize multi-agent workflows (Xiao et al., 2023; AhmadiTeshnizi et al., 2024) or search algorithms (Liu et al., 2025c; Astorga et al., 2025) to improve reasoning. Complementary to these are fine-tuning methods like FOARL (Jiang et al., 2025), ORLM (Huang et al., 2025a), and SIRL (Chen et al., 2025), which train specialized models on operations research datasets to internalize complex modeling patterns. However, prior methods often overlook modeling errors, which often manifest as silent semantic discrepancies where the logic deviates from the problem intent despite successful code execution.

3 Benchmark

3.1 Data Collection and Curation

We construct OptiVerse benchmark through a comprehensive five-stage pipeline consisting of: **Acquisition**, **Standardization**, **Translation and Verification**, **Quality Filtering**, and **Classification**.

Acquisition. We source optimization problems from authoritative textbooks and academic publications such as (Li, 2023; Hillier and Lieberman, 2020; Boyd and Vandenberghe, 2004; Liu et al., 2022; Korte and Vygen, 2018; Birge and Louveaux, 2011; Bertsekas, 2017; Osborne, 2004) that are widely adopted in relevant curricula globally. These span diverse optimization domains, ranging from mathematical programming to optimal control, and are selected for their rigorous academic standards and accessibility for scholarly research.

Standardization. Using the MinerU2.5 framework (Niu et al., 2025), we extract and structure problem content from source documents. Each

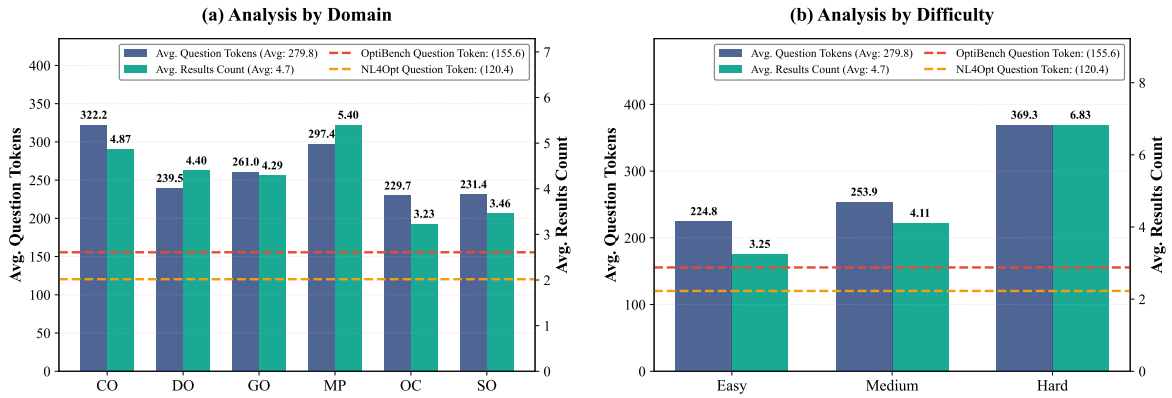

Figure 4: Statistical analysis of question tokens and result counts across optimization problem domain and difficulty, benchmarked against NL4Opt (Ramamonjison et al., 2023) and Optibench (Yang et al., 2025b).

problem undergoes deduplication and format normalization. Furthermore, we ensure that tabular and graphical data are properly structured and accurately preserved alongside textual descriptions.

Verification and Translation. Ph.D. candidates and master's students specializing in operations research and applied mathematics are recruited to review all instances to ensure completeness and correctness. Additionally, these professionals translated non-English problems to guarantee rigorous accuracy regarding domain-specific terminology, mathematical notation, and technical precision.

Quality Filtering. Following established practices (Rein et al., 2024; Zhang et al., 2025a), we exclude problems whose solutions are readily accessible through web searches (via a five-minute Google Search) to mitigate data contamination risks and ensure the validity and benchmark integrity.

Classification. The finalized benchmark is organized into a two-dimensional taxonomy: the three aforementioned **difficulty levels** cross-referenced with six **optimization domains** as shown in Table 1, designed to comprehensively evaluate the ability of different models to solve optimization problems.

The six domains are defined with precise boundaries: *Mathematical Programming (MP)* for continuous deterministic models; *Combinatorial Optimization (CO)* for discrete structures and network flows; *Stochastic Optimization (SO)* for uncertainty and probabilistic models; *Dynamic Optimization (DO)* for multi-stage decision processes; *Optimal Control (OC)* for systems governed by differential equations; and *Game Optimization (GO)* for multi-agent strategic interactions, as shown in Figure 2. This provides a comprehensive framework for evaluating the diverse reasoning capabilities of large language models, as illustrated in Figure 3.

3.2 Benchmark Characteristics

OptiVerse benchmark consists of 1,000 carefully curated optimization problems, providing comprehensive coverage across diverse mathematical disciplines. Figure 4 presents a statistical analysis of the benchmark's complexity and its distribution. Three distinctive features characterize this benchmark:

Comprehensive Domain Coverage: Unlike existing benchmarks confined to MP and CO, OptiVerse establishes a more rigorous standard by spanning six distinct optimization domains: MP, CO, SO, DO, OC, and GO. This coverage captures a full spectrum of challenges, ranging from static deterministic models to complex dynamic and stochastic systems. Consequently, this breadth ensures a holistic evaluation, testing whether LLMs possess generalized reasoning capabilities rather than merely excelling in specific sub-domains.

Enhanced Problem Complexity and Scaling: OptiVerse exhibits higher linguistic and structural complexity than existing benchmarks, with average question token and result dimensions substantially exceeding those of NL4Opt (Ramamonjison et al., 2023) and OptiBench (Yang et al., 2025b). Furthermore, the benchmark is stratified into three difficulty levels—Easy, Medium, and Hard—where complexity scales naturally. For instance, Hard problems average 369.3 tokens and require an average of 6.83 result outputs, providing a rigorous test for complex optimization problem solving.

Cross-Paradigm Tool Versatility: Given the extreme diversity of problem types—ranging from Nash Equilibrium in GO to PID tuning Optimization in OC, no single specialized solver, even a powerful industrial standard like Gurobi, can serve as a general solution for the entire benchmark. Consequently, our dataset evaluates a model's abil-

ity to transcend single-solver dependencies. The LLMs must demonstrate strategic flexibility by autonomously identifying the problem's underlying paradigm and implementing the most appropriate algorithmic approach, whether through specialized libraries or custom-coded heuristic solutions.

4 Experiments

4.1 Experimental Setup

Model Selection. We select 22 recent LLMs of different sizes and divide them into three categories: **(1) Open-Source Non-Thinking Models.** This category encompasses the Qwen3 series (covering Instruct and Coder variants across 8B, 30B, and 235B scales) (Yang et al., 2025a), Qwen2.5-72B (Yang et al., 2024), DeepSeek-V3.2-Chat (Liu et al., 2025a), Kimi-K2 (Kimi Team and Bai, Yifan and Bao, Yiping and Chen, Guanduo and Chen, Jiahao and Chen, Ningxin and Chen, Ruijue and Chen, Yanru and Chen, Yuankun and Chen, Yutian and others, 2025), Ministral3-8B (Mistral AI Team, 2025), and Internlm3-8B (Team, 2025). **(2) Open-Source Thinking Models.** To evaluate reasoning capabilities within the open-weights landscape, we select OpenAI's open-sourced GPT-120B (Agarwal et al., 2025). Additionally, we include the specialized "Thinking" variants of the Qwen3 series (8B, 30B, and 235B) (Yang et al., 2025a), and DeepSeek-V3.2 (Liu et al., 2025a), which are explicitly optimized for complex reasoning tasks. **(3) Closed-Source Thinking Models.** We assess proprietary systems, categorized into reasoning-intensive models—such as OpenAI's o3 and o4-mini (OpenAI, 2025b), and flagship general-purpose models including GPT-5.2 (OpenAI, 2025a), Claude-4.5-Sonnet (Anthropic, 2025), and the Gemini series (2.5-Flash/Pro (Deepmind, 2025a,b) and 3-Flash/Pro) (DeepMind, 2025a,b).

Inference Configuration. We adopt a chain-of-thought strategy where the model processes the problem P to generate a composite response $R = \text{LLM}(P)$, consisting of a mathematical modeling component and an executable Python code segment C. This formulation-first approach outperforms direct code generation by ensuring that the implementation is grounded in a formal logical derivation. The code C is then extracted from R and executed within a sandboxed environment to yield the final results $O = \text{Exec}(C)$. To accommodate diverse optimization problem domains, our environment integrates a comprehensive suite of scientific libraries, including *gurobi*, *casadi*, *pyomo*, *nashpy*, *scikit-opt*, *cvxpy*, *ortools*, *pulp*, and *scipy*.

LLM	Accuracy (%)
Qwen3-235B-Instruct	97.8
Qwen3-235B-Thinking	99.4
DeepSeek-V3.2-Chat	98.4
DeepSeek-V3.2-Reasoner	99.6

Table 2: Evaluation Accuracy of Selected LLMs.

Evaluation Methodology. To handle the complexity of diverse solver outputs and varying formatting styles, we employ a two-stage evaluation process powered by an LLM-as-judge framework.

Stage 1: Answer Extraction. In the first stage, we utilize an LLM as an extractor to parse the execution output O. The extraction process yields a corresponding set of values denoted as $A = \text{LLM}_{\text{extract}}(O, R) = \{a_1, a_2, \ldots, a_n\}$, where each a_j represents the extracted numerical value for the specific requirement $r_j \in R$ in problem.

Stage 2: Answer Verification. In the second stage, this judge framework performs a comparison between the extracted values A and the ground truth set A^*. To account for potential minor floating-point variations inherent to solver computations, the LLM is instructed to verify precision with a relative error tolerance of $\epsilon = 0.1\%$. The correctness of the problem is determined by the verification of all individual components:

$$\text{IsCorrect}(A) \iff \text{LLM}_{\text{judge}}(A, A^*, \epsilon) \quad (1)$$

A problem is marked as correct only when the LLM-as-judge verifies that every required variable and objective satisfies the precision threshold, thereby ensuring a rigorous assessment.

Evaluation Accuracy. Table 2 presents the performance statistics derived from an evaluation set of 500 samples. It can be observed that advanced LLMs generally satisfy the requirements. However, considering the trade-offs between cost and inference speed, we ultimately select DeepSeek-V3.2-Chat as the standard evaluation LLM.

4.2 Main Results

The comprehensive evaluation results on the benchmark are presented in Table 3. We summarize our key findings into four primary observations:

Superiority of Reasoning Chains: Models equipped with explicit reasoning capabilities con-

Large Language Model	Domain						Difficulty			Avg.
	MP	CO	SO	DO	OC	GO	Easy	Med.	Hard	
Open-Source Non-Thinking Models										
Internlm3-8B-instruct	11.44	7.56	5.00	8.22	0.00	3.92	18.67	3.75	3.00	8.00
Ministral3-8B-Instruct	26.01	18.44	19.35	12.08	2.56	5.88	39.33	14.75	5.67	18.20
Qwen3-8B-Instruct	23.98	22.69	20.83	14.38	6.41	13.73	42.67	12.25	7.67	20.00
Qwen2.5-72B	31.61	27.31	20.00	15.75	7.69	13.73	48.00	16.50	10.33	24.10
Qwen3-Coder-30B	30.79	31.09	23.33	23.29	8.97	9.80	49.67	19.25	11.67	26.10
Qwen3-30B-Instruct	38.96	36.13	31.67	34.25	19.23	15.69	70.33	21.75	14.00	34.00
Kimi-K2	40.05	44.54	38.33	39.73	24.36	25.49	71.33	31.50	16.33	38.90
Qwen3-235B-Instruct	44.41	47.06	45.83	41.10	42.31	41.18	78.33	39.75	16.67	44.40
DeepSeek-V3.2-Chat	51.23	47.48	45.00	43.15	21.79	35.29	79.67	39.25	19.00	45.30
Open-Source Thinking Models										
Qwen3-8B-Thinking	40.33	43.70	41.67	39.73	25.64	37.25	73.00	33.00	16.00	39.90
GPT-OSS-120B	49.54	55.33	34.19	38.93	32.05	35.29	78.67	39.25	18.67	44.90
Qwen3-30B-Thinking	49.59	46.64	46.67	46.58	30.77	43.14	80.67	40.00	20.33	46.30
DeepSeek-V3.2-Reasoner	53.68	52.52	45.00	49.32	28.21	49.02	84.33	44.50	21.33	49.50
Qwen3-235B-Thinking	53.68	51.26	49.17	52.74	43.59	49.02	87.33	47.00	21.33	51.40
Closed-Source Thinking Models										
Gemini-2.5-Flash	46.05	46.22	48.33	48.63	48.72	54.90	82.33	42.75	18.67	47.40
Gemini-2.5-Pro	53.68	49.16	50.00	47.95	38.46	43.14	87.00	43.75	20.00	49.60
Claude-4.5-Sonnet	53.41	52.52	46.67	47.95	34.62	45.10	83.67	45.25	21.67	49.70
o3	52.04	53.78	45.83	56.85	39.74	45.10	86.67	47.25	20.67	51.10
o4-mini	50.95	54.20	48.33	55.48	42.31	47.06	87.67	46.75	20.67	51.20
GPT-5.2	55.86	57.14	50.00	57.53	50.00	54.90	91.00	50.75	25.33	55.20
Gemini-3-Flash	54.77	59.24	54.17	55.48	48.72	56.86	88.67	53.25	25.33	55.50
Gemini-3-Pro	58.04	57.14	56.67	56.85	42.31	50.98	89.00	52.75	27.00	55.90

Table 3: Model performance comparisons on the OptiVerse benchmark. The evaluation is categorized by **Domain** (MP, CO, SO, DO, OC, and GO) and **Difficulty** (Easy, Medium, and Hard). **Avg.** denotes the mean performance.

sistently outperform their standard instruction-tuned counterparts by a significant margin. For instance, Qwen3-8B-Thinking surpasses the performance achieved by Qwen3-8B-Instruct, which demonstrates that internal reasoning chains are crucial for addressing complex optimization problems.

Significant Difficulty Sensitivity: All LLMs exhibit sharp performance degradation as task difficulty scales from Easy to Hard. While top-tier LLMs like Gemini-3-Pro maintain high accuracy on Easy problems, they also struggle significantly with Hard problems, indicating that LLMs struggle with solving complex optimization problems.

Scaling Law Effects: Within the same model family, performance scales positively with parameter count. In the Qwen3 series, performance increases steadily from 8B to the 235B model, a trend suggesting that larger models possess more robust optimization problem-solving capabilities.

Domain-Specific Fragility: There is a notable performance disparity across domains, as most LLMs perform significantly better on MP and CO tasks than on the other four categories. This indicates that current models lack cross-domain robustness and may be sensitive to the specific symbolic representations or reasoning patterns required by different sub-fields of optimization problems.

4.3 Error Kind Distribution Analysis

To identify the different failure modes impeding optimization problem solving, we conduct a fine-grained manual annotation of error types. We categorize the failure modes into five distinct classes:

- **Modeling & Logic Error:** The LLMs fail to correctly comprehend the problem semantics or translate the natural language description into a valid mathematical model (e.g., misinterpreting dynamic programming state transitions).
- **Parameter & Data Utilization Error:** While the modeling logic is correct, the LLM fails to accurately utilize the values, coefficients, conditions, and dimensions from the text (e.g., hallucinating data, confusing data across different stages, or misaligning matrix dimensions).
- **Feasibility Violation:** The derived solution violates the problem's hard constraints, rendering

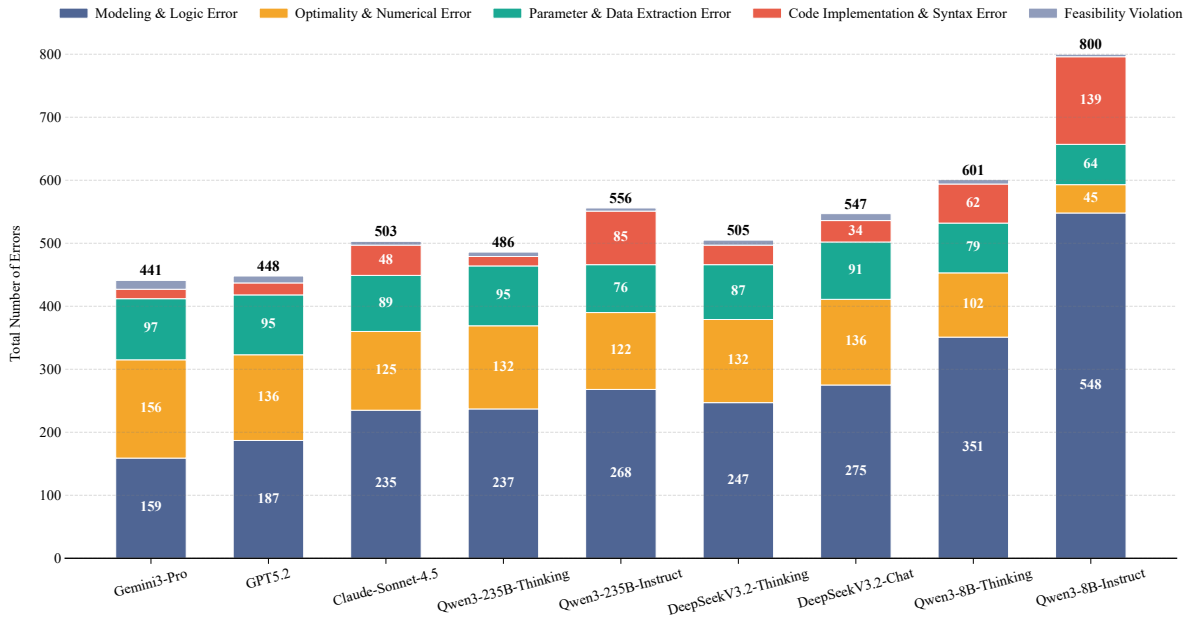

Figure 5: Distribution of error types across nine representative models. While **Code & Syntax** errors become negligible with increased capability, **Modeling & Logic** errors remain the predominant bottleneck across all LLMs.

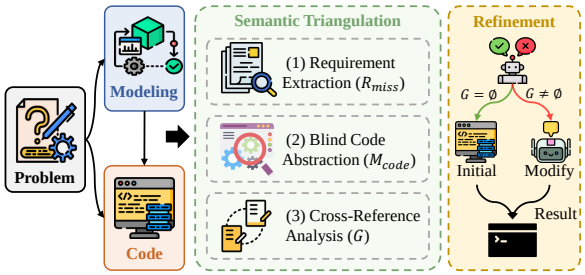

Figure 6: The workflow of the Dual-View Auditor Agent which utilizes a three-phase auditing mechanism to detect semantic discrepancies between the problem and code, determining whether to make modifications.

the result mathematically invalid or infeasible.

- **Optimality & Numerical Error:** The LLMs produce a feasible solution, but it is significantly suboptimal compared to the ground truth.
- **Code & Syntax Error:** The generated code contains syntax errors, runtime exceptions, or incomplete logic, preventing successful execution

Figure 5 visualizes the comparative distribution of these error types across a diverse set of nine representative LLMs, revealing three critical insights:

Modeling is the Primary Bottleneck. Even for state-of-the-art models, the prevalence of this error underscores that correctly abstracting natural language into mathematical formulations remains a persistent and fundamental cognitive bottleneck.

Optimality & Numerical Challenges Persist. After modeling errors, optimality and numerical issues constitute the second largest failure. This suggests that even with mathematical models, LLMs often struggle to converge on optimal solutions.

Reasoning Models Enhance Code Stability. For example, the "thinking" variant of Qwen3-235B incurs only 15 errors compared to 85 in its instruct counterpart, demonstrating that extended chain-of-thought processes effectively mitigate syntax hallucinations and implementation flaws.

5 Dual-View Auditor Agent

5.1 Method

Motivated by the error distribution analysis in Section 4.3, which identifies modeling & logic as the predominant bottleneck in LLM-based optimization, we introduce the *Dual-View Auditor Agent (DVA-Agent)*. Conventional self-correction methods often falter because models tend to hallucinate correctness when merely re-reading their own generated code against the prompt. To overcome this, DVA-Agent acts as an adversarial evaluator designed to uncover deep logical discrepancies.

Central to this framework is the **Semantic Triangulation**. Unlike simple syntax checkers, this detects "silent semantic errors" scenarios where code executes without runtime exceptions but solves a mathematical model deviating from the problem intent. This operates through a three-phase process:

Requirement Extraction (Text-to-Math). By referencing the initial modeling logic (M_{init}) and problem (P), it identifies core mathematical com-

ponents, such as specific parameter values or complex constraints, that are present in P but the modeling logic M_{init} tends to omit or misinterpret. This results in a set of missing requirements (R_{miss}) that serves as a reference for validation.

Blind Code Abstraction (Code-to-Math). To mitigate confirmation bias, the agent performs a "blind" review. Without access to the problem P, this acts as a code interpreter, reverse-engineering the underlying mathematical logic solely from the initial code. This extracts the objective function, decision variables, and constraints (M_{code}). By isolating the code from the problem, we ensure that the derived logic reflects what the code actually executes, rather than what the LLM claims.

Cross-Reference Analysis. Finally, this synthesizes information from the three sources: the original problem P, the reverse-interpreted code logic M_{code}, and the extracted requirements R_{miss}. This comparison produces a discrepancy set G, which quantifies the critical gap between the intended requirements and the implemented logic.

Refinement. This workflow culminates in a decision-making process based on the verdict, defined by the discrepancy set G. If the analysis yields an empty set ($G = \emptyset$), signifying that the implemented code is semantically aligned with the problem, the initial code is validated and executed to compute the result. Conversely, if any discrepancies ($G \neq \emptyset$) are detected, the identified gaps serve as corrective guidance, prompting the LLM to modify the modeling logic and code. This iterative refinement ensures that the final code accurately captures the complex constraints of the original problem before producing the final result.

5.2 Evaluation

To rigorously evaluate the efficacy and universality of DVA-Agent, we conduct comprehensive experiments using three advanced LLMs: Qwen3-30B, Qwen3-235B (Yang et al., 2025a), and DeepSeek-V3.2 (Liu et al., 2025a). This setup is specifically designed to validate two key advantages of DVA-Agent: (1) **Plug-and-Play Capability**, evidenced by the seamless integration with these diverse LLMs without requiring extensive adaptation; and (2) **Efficiency**, assessed by the impact of the conditional repair strategy that triggers regeneration only when specific discrepancies are detected.

Baselines. In order to evaluate the performance against existing paradigms, we compare DVA-Agent with: Chain of Experts (CoE) (Xiao et al., 2023), a multi-agent system utilizing domain-specific role-playing for refinement, and OptiMUS (AhmadiTeshnizi et al., 2024), a specialized framework designed for iterative optimization modeling.

Framework	Easy	Medium	Hard	Cost Time
Qwen3-30B-Instruct				
Baseline	70.33	21.75	14.00	6.1s
CoE	70.67	22.50	14.67	62.4s
OptiMUS	71.00	24.25	15.33	35.7s
DVA-Agent	72.33	30.50	18.67	30.2s
Qwen3-235B-Instruct				
Baseline	78.33	39.75	16.67	11.8s
CoE	80.00	42.25	18.00	121.3s
OptiMUS	81.33	44.70	20.33	69.3s
DVA-Agent	85.67	50.25	24.33	58.8s
DeepSeek-V3.2-Chat				
Baseline	79.67	39.25	19.00	15.7s
CoE	80.67	40.75	19.67	136.1s
OptiMUS	81.33	42.00	20.67	83.2s
DVA-Agent	84.00	47.50	24.00	67.3s

Table 4: Solving accuracy (%) comparison across OptiVerse benchmark using three different LLMs.

Experimental Results. Table 4 empirically validates DVA-Agent's effectiveness and superiority over the existing workflow across two dimensions:

(1) Robust Performance. DVA-Agent consistently achieves the highest accuracy across all difficulty tiers, confirming its Plug-and-Play Capability. This superiority across varying model scales demonstrates that the Semantic Triangulation mechanism is model-agnostic and scales effectively with stronger reasoning backbones.

(2) Computational Efficiency. The agent optimizes the accuracy-time trade-off via a conditional repair strategy that triggers refinement only when discrepancies arise ($G \neq \emptyset$). With modification rates of merely 23.6%, 32.3%, and 28.5% for Qwen3-30B, Qwen3-235B, and DeepSeek-V3.2-Chat, respectively, the system effectively avoids unnecessary iterations on already correct solutions.

6 Conclusion

We introduce **OptiVerse**, a benchmark comprising 1,000 curated optimization problems across six disciplines designed to rigorously evaluate the reasoning depth of LLMs. Extensive experiments with 22 state-of-the-art models reveal that despite reasoning enhancements, performance on hard tasks degrades sharply, falling below 27% accuracy. Our fine-grained error analysis identifies semantic modeling and logic as primary bottlenecks, exacerbated

by significant domain fragility when generalizing from standard mathematical programming to specialized fields like Optimal Control. To address these deficiencies, we propose the Dual-View Auditor Agent, an adversarial framework that yields consistent improvements by targeting silent semantic defects. Ultimately, OptiVerse provides a robust framework and actionable insights for advancing the next generation of optimization-capable LLMs.

7 Acknowledgements

This work was supported by Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM116), National Natural Science Foundation of China (No. 62137002, 62293550, 62293553, 62293554, 62437002, 62477036, 62477037, 62192781), the Shaanxi Provincial Social Science Foundation Project (No. 2024P041), the Youth Innovation Team of Shaanxi Universities "Multi-modal Data Mining and Fusion", and Xi'an Jiaotong University City College Research Project (No. 2024Y01).

8 Limitation

Despite the comprehensive nature of OptiVerse, which spans six optimization domains, two key limitations warrant discussion regarding benchmark construction and evaluation. First, our problems are primarily derived from authoritatives and academic publications, focusing on rigorous reasoning under idealized mathematical conditions rather than fully reflecting the messy realities of industrial scenarios. Real-world optimization often involves data noise, large-scale approximations, and ambiguous constraints that are not fully captured here. However, it is worth noting that mastering these rigorous formulations under idealized conditions serves as the fundamental textbook prerequisite for solving complex real-world operational problems. Furthermore, extensive experiments reveal that current state-of-the-art LLMs' performance on these "idealized" Hard problems remains unsatisfactory, with even GPT-5.2 and Gemini-3 failing to exceed 27% accuracy. Therefore, OptiVerse remains a highly valuable testbed for evaluating models' core ability to translate natural language into executable optimization logic. In future research, we aim to adapt these problems to simulate real-world noise and approximation requirements. Second, our evaluation framework, while ensuring high precision through an LLM-as-a-judge mechanism, incurs certain computational costs. While the two-stage verification ensures accuracy, we plan to explore more efficient evaluation pipelines in future work to optimize the time-cost trade-off.

9 Ethical Statement

In developing OptiVerse, we carefully considered and addressed potential implications and risks. Our benchmark is sourced exclusively from public authoritative textbooks and academic publications, and we have conducted rigorous data cleansing and standardization to ensure reliability. To mitigate the risk of data contamination and ensure fair evaluation, we strictly excluded problems whose solutions were readily accessible through web searches. We strictly comply with the copyright and licensing terms of all source materials and the models used in our experiments. Committed to environmental sustainability and research reproducibility, we will publicly release the complete dataset and evaluation scripts under appropriate open-source licenses. This allows the community to reuse our resources, thereby cutting down the carbon footprint associated with redundant data curation, while driving the advancement of Large Language Models in complex reasoning tasks.

References

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. 2024. Optimus: Scalable optimization modeling with (mi) lp solvers and large language models. In *Forty-first International Conference on Machine Learning*.

Anthropic. 2025. Introducing claude sonnet 4.5.

Nicolás Astorga, Tennison Liu, Yuanzhang Xiao, and Mihaela van der Schaar. 2025. Autoformulation of mathematical optimization models using llms. In *Forty-second International Conference on Machine Learning*.

Dimitri P Bertsekas. 2017. *Dynamic Programming and Optimal Control, Vol. I & II*, 4 edition. Athena Scientific.

John R Birge and François Louveaux. 2011. *Introduction to Stochastic Programming*. Springer Science & Business Media.

Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. 2025. Solver-informed rl: Grounding large language models for authentic optimization modeling. *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Google Deepmind. 2025a. Gemini 2.5 flash.

Google Deepmind. 2025b. Gemini 2.5 pro.

Google DeepMind. 2025a. Gemini 3 flash.

Google DeepMind. 2025b. Gemini 3 pro.

Igor Halperin. 2022. *Reinforcement Learning and Stochastic Optimization: A Unified Framework for Sequential Decisions: by Warren B. Powell (ed.), Wiley (2022). Hardback. ISBN 9781119815051.*, volume 22. Taylor & Francis.

Frederick S Hillier and Gerald J Lieberman. 2020. *Introduction to Operations Research*, 11 edition. McGraw-Hill Education.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. 2025a. Orlm: A customizable framework in training large models for automated optimization modeling. *Operations Research*.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. 2025b. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 2678–2710, Albuquerque, New Mexico. Association for Computational Linguistics.

Peter Jansen, Oyvind Tafjord, Marissa Radensky, Pao Siangliulue, Tom Hope, Bhavana Dalvi, Bodhisattwa Prasad Majumder, Daniel S Weld, and Peter Clark. 2025. Codescientist: End-to-end semi-automated scientific discovery with code-based experimentation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 13370–13467.

Xia Jiang, Yaoxin Wu, Minshuo Li, Zhiguang Cao, and Yingqian Zhang. 2025. Large language models as end-to-end combinatorial optimization solvers. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Kimi Team and Bai, Yifan and Bao, Yiping and Chen, Guanduo and Chen, Jiahao and Chen, Ningxin and Chen, Ruijue and Chen, Yanru and Chen, Yuankun and Chen, Yutian and others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Bernhard Korte and Jens Vygen. 2018. *Combinatorial Optimization: Theory and Algorithms*, 6 edition. Springer.

Gongnong Li. 2023. *Fundamentals of Operations Research and MATLAB Applications*, 2nd edition. Tsinghua University Press, Beijing. In Chinese.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. 2025a. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

Dongshuo Liu, Zhijing Wu, Dandan Song, and Heyan Huang. 2025b. A persona-aware LLM-enhanced framework for multi-session personalized dialogue generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 103–123, Vienna, Austria. Association for Computational Linguistics.

Haoyang Liu, Jiang Hu, Yongfeng Li, and Zaiwen Wen. 2022. *Optimization: Modeling, Algorithms, and Theory*. Higher Education Press. Available at http://faculty.bicmr.pku.edu.cn/ wenzw/optbook.html.

Haoyang Liu, Jie Wang, Yuyang Cai, Xiongwei Han, Yufei Kuang, and Jianye HAO. 2025c. Optitree: Hierarchical thoughts generation with tree search for llm optimization modeling. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. 2025. Optmath: A scalable bidirectional data synthesis framework for optimization modeling. In *Forty-second International Conference on Machine Learning*.

Mistral AI Team. 2025. Introducing mistral 3.

Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, Zirui Tang, Boyu Niu, Ziyang Miao, Hejun Dong, Siyi Qian, Junyuan Zhang, Jingzhou Chen, Fangdong Wang, Xiaomeng Zhao, Liqun Wei, Wei Li, Shasha Wang, Ruiliang Xu, Yuanyuan Cao, Lu Chen, Qianqian Wu, Huaiyu Gu, Lindong Lu, Keming Wang, Dechen Lin, Guanlin Shen, Xuanhe Zhou, Linfeng Zhang, Yuhang Zang, Xiaoyi Dong, Jiaqi Wang, Bo Zhang, Lei Bai, Pei Chu, Weijia Li, Jiang Wu, Lijun Wu, Zhenxiang Li, Guangyu Wang, Zhongying Tu, Chao Xu, Kai Chen, Yu Qiao, Bowen Zhou, Dahua Lin, Wentao Zhang, and Conghui He. 2025. Mineru2.5: A decoupled vision-language model for efficient high-resolution document parsing. *Preprint*, arXiv:2509.22186.

OpenAI. 2025a. Introducing gpt-5.2.

OpenAI. 2025b. Introducing openai o3 and o4-mini.

Martin J Osborne. 2004. *An Introduction to Game Theory*. Oxford University Press.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. 2023. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pages 189–203. PMLR.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*.

InternLM Team. 2025. Internlm3-8b-instruct.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. 2023. Chain-of-experts: When llms meet complex operations research problems. In *The twelfth international conference on learning representations*.

Yibo Yan, Jiamin Su, Jianxiang He, Fangteng Fu, Xu Zheng, Yuanhuiyi Lyu, Kun Wang, Shen Wang, Qingsong Wen, and Xuming Hu. 2025. A survey of mathematical reasoning in the era of multimodal large language model: Benchmark, method & challenges. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 11798–11827, Vienna, Austria. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv e-prints*, pages arXiv–2412.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. 2025b. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *The Thirteenth International Conference on Learning Representations*.

Lingling Zhang, Xiaojun Chang, Jun Liu, Minnan Luo, Zhihui Li, Lina Yao, and Alex Hauptmann. 2022. Tn-zstad: Transferable network for zero-shot temporal activity detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3848–3861.

Lingling Zhang, Xinyu Zhang, Qianying Wang, Wenjun Wu, Xiaojun Chang, and Jun Liu. 2023. Rpmg-fss: Robust prior mask guided few-shot semantic segmentation. *IEEE Transactions on Circuits and Systems for Video Technology*, 33(11):6609–6621.

Xinyu Zhang, Yuxuan Dong, Yanrui Wu, Jiaxing Huang, Chengyou Jia, Basura Fernando, Mike Zheng Shou, Lingling Zhang, and Jun Liu. 2025a. PhysReason: A comprehensive benchmark towards physics-based reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16593–16615, Vienna, Austria. Association for Computational Linguistics.

Xinyu Zhang, Yuxuan Dong, Lingling Zhang, Chengyou Jia, Zhuohang Dang, Basura Fernando, Jun Liu, and Mike Zheng Shou. 2025b. Cofft: Chain of foresight-focus thought for visual language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Muye Huang, and Jun Liu. 2025c. Cognitive predictive coding network: Rethinking the generalization in raven's progressive matrices. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 4097–4106.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Muye Huang, Wenjun Wu, Bo Li, Shaowei Wang, Basura Fernando, and Jun Liu. 2025d. Diagram-driven course questions generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5995–6010.

Xinyu Zhang, Lingling Zhang, Yanrui Wu, Shaowei Wang, Wenjun Wu, Muye Huang, Qianying Wang, and Jun Liu. 2025e. Memory-enriched thought-by-thought framework for complex diagram question answering. *Computer Vision and Image Understanding*, page 104608.

A Details of Human Annotators

For data annotation, translation, and result verification, we engaged Ph.D. candidates and Master's students who are good at Operations Research and Applied Mathematics, who are also co-authors of this paper. All annotators possess rigorous academic backgrounds across the six optimization domains (MP, CO, SO, DO, OC, and GO), making them well-qualified to ensure the technical precision of domain-specific terminology and mathematical notations. Since the annotators were active researchers involved in the study, no formal recruitment process or external compensation was required, and they were fully aware of the data usage protocols. The annotation process focused solely on the evaluation of optimization problems and mathematical modeling logic, without involving the collection of any personal identifying information or exposing annotators to risks. As this research involved the analysis of academic optimization content rather than external human subjects, it was determined to be exempt from formal ethics review board approval.

B Details of Ai Assistants In Research Or Writing

We used Claude-4.5-Sonnet and Gemini-3.0-Pro to help us write code and polish the paper.

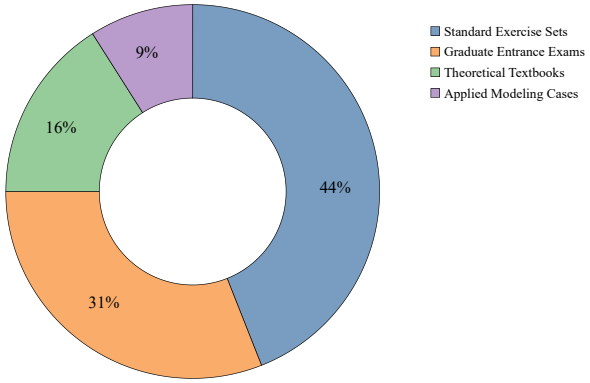

Figure 7: Distribution of Data Sources.

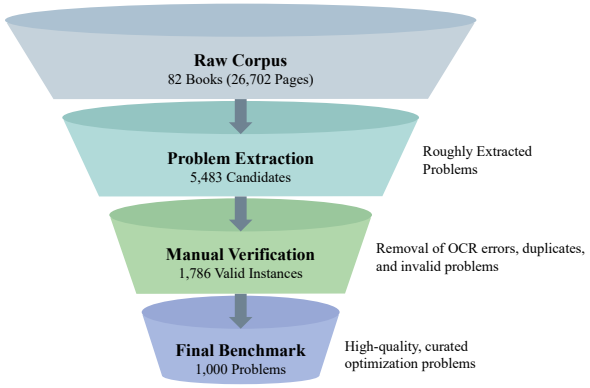

Figure 8: Data Construction Pipeline.

C Data Source

To construct a benchmark that is both comprehensive and rigorously challenging, we curated a massive dataset derived from highly authoritative academic resources. The raw corpus consists of 82 distinct textbooks, specialized problem sets, and graduate-level solution manuals, encompassing a total of 26,702 pages of source material. These documents cover a wide spectrum of optimization domains, ranging from fundamental Linear and Integer Programming to advanced topics in Optimal Control, Convex Optimization, and Game Theory.

Notably, to ensure the benchmark effectively evaluates the deep reasoning capabilities of Large Language Models, a significant portion of the questions was extracted from graduate entrance examination papers and mathematical modeling competition case studies, which require multi-step logical deduction rather than simple formula application.

Figure 7 illustrates the distribution of these data sources by category, highlighting our focus on practical problem-solving and high-difficulty exam questions.

Figure 8 details the construction pipeline, demonstrating how we distilled the massive unstructured corpus into a refined set of 1,000 high-quality problems through a multi-stage filtration process.

D Error Type Details

The detailed definitions and representative examples for each error category are presented below:

1. **Modeling & Logic Error:**
   - *Description:* Failures where the model correctly identifies the problem domain but fails to translate the natural language description into a valid mathematical model or optimization logic. This includes misinterpreting the objective function, omitting critical constraints, or selecting an inappropriate modeling paradigm.
   - *Examples:*
     - Formulating a multi-stage Stochastic Programming problem as a deterministic Linear Programming model, ignoring probability distributions.
     - In a Network Flow problem, failing to establish flow conservation constraints (i.e., inflow must equal outflow) at intermediate nodes.
     - Incorrectly defining the decision variables, such as using continuous variables for a problem that strictly requires binary or integer decisions (e.g., Knapsack Problem).
2. **Parameter & Data Extraction Error:**
   - *Description:* Cases where the overarching modeling logic is correct, but the model fails to accurately extract or align specific numerical values, coefficients, or dimensions from the problem text. This often manifests as "data hallucination" or matrix misalignment.
   - *Examples:*
     - Extracting a cost coefficient of "10" when the problem statement specifies

"100", or confusing unit costs with total costs.
  - In a Traveling Salesman Problem (TSP), misreading the distance matrix indices, treating row i column j as the distance from j to i in an asymmetric graph.
  - Dimensionality mismatch, such as defining a constraint vector of size N when the decision variable vector size is $N + 1$.

3. **Code Implementation & Syntax Error:**
   - *Description:* Errors occurring during the translation of the mathematical model into executable Python code. These include syntax errors, incorrect API usage of optimization solvers (e.g., Gurobi, PuLP, SciPy), or incomplete implementation logic that causes the runtime environment to crash.
   - *Examples:*
     - **Syntax/Runtime Errors:** Identifying variables but failing to define them before use, or causing `KeyError` or `IndexError` when accessing data structures.
     - **API Misuse:** Calling non-existent methods of a solver library (e.g., using `model.addConstrs` with incorrect syntax in Gurobi) or failing to call the `optimize()` method.
     - **Logical Implementation Gaps:** The code runs without crashing, but the loop structure logic does not match the intended mathematical summation, resulting in an empty or trivial model.

4. **Feasibility Violation:**
   - *Description:* This category encompasses two distinct types of failures where the output is mathematically invalid. First, it includes cases where the proposed solution explicitly violates the hard constraints defined in the problem (e.g., resource limits or non-negativity). Second, and critically, it includes cases where the generated mathematical model itself exhibits unreasonable mathematical properties—such as contradictory constraints or unbounded objectives—rendering the problem mathematically infeasible or impossible to be solved by standard algorithms.
   - *Examples:*
     - **Constraint Violation:** In a logistics problem, proposing a delivery route that exceeds the maximum vehicle capacity constraint, or outputting negative values for physical quantities like mass or time.
     - **Model Infeasibility:** Constructing a Linear Programming model with mutually exclusive constraints (e.g., requiring $x \geq 10$ and $x \leq 5$ simultaneously), resulting in an empty feasible region.
     - **Unreasonable Mathematical Property:** Formulating an optimization objective (e.g., "Maximize Profit") without necessary bounding constraints, causing the solver to return an "Unbounded" status or diverge during calculation.

5. **Optimality & Numerical Error:**
   - *Description:* Cases where the model finds a valid, feasible solution, but it is significantly suboptimal compared to the ground truth, or exhibits precision issues. This often arises from inappropriate algorithm selection (e.g., using a greedy heuristic instead of an exact solver) or numerical instability.
   - *Examples:*
     - Using a local search algorithm (like Hill Climbing) for a non-convex function and getting stuck in a local optimum, missing the global maximum.
     - **Numerical Precision:** Returning a result that deviates from the standard solution beyond the acceptable tolerance ($\epsilon > 0.1\%$) due to floating-point errors or aggressive rounding.
     - **Solver Configuration:** Failing to set an appropriate "MIP Gap" or time limit, causing the solver to return a premature solution that has not yet converged to optimality.

Problem

You are a project manager at a manufacturing firm tasked with upgrading a production line. The project consists of four key activities (A, B, C, D) with their dependencies, durations, and costs detailed in the table below. Your company incurs an indirect overhead cost of ¥4,500 for each day the project remains active. Using the provided data, determine the optimal duration for each activity to minimize the total project cost, which is the sum of direct, indirect, and any potential crashing costs.

Act.	Pred.	Normal Time	Crash Time	Normal Cost	Crash Cost	Slope
A	-	3	1	10	18	4,000
B	A	7	3	15	19	1,000
C	A	4	2	12	20	4,000
D	C	5	2	8	14	2,000

(Time in days, Cost in ¥1,000)

Optimization Result

- Optimal Project Duration: 7 days
- Minimum Total Cost: ¥91,500

Category
Combinatorial Optimization(CO)

Difficulty
Easy

Figure 9: An Easy Example in OptiVerse.

E Example

We have provided a representative example for each of the three difficulty levels—Easy (Figure 9), Medium (Figure 10), and Hard (Figure 11)—to serve as a guide.

The easy-level problem (Figure 9) represents a classic Combinatorial Optimization task involving project scheduling. It requires the model to perform algorithmic execution—specifically Critical Path Method (CPM) and cost-slope analysis. The solution path is linear and iterative: identify the critical path, select the activity with the lowest crash cost, and repeat until the optimal trade-off is found. This tests the model's ability to follow standard algorithmic procedures and perform precise arithmetic operations on tabular data.

The medium-level problem (Figure 10) introduces Game Optimization coupled with stochastic elements. Unlike the easy problem, it requires the integration of multiple domain concepts: Game Theory (Stackelberg Leader-Follower dynamics), Probability (Expected Value calculation), and Calculus (Profit Maximization). The model must employ backward induction—first solving the follower's reaction function under different demand scenarios, and then optimizing the leader's strategy based on expected payoffs. This demands a sophisticated understanding of multi-agent interactions and conditional logic.

Problem

Consider a market where a leader sets production q_L before uncertain demand A is realized. The inverse demand is $P = A - (q_L + q_F)$, where A is 150 or 90 with equal probability (0.5). The leader's cost is $C_L = 20q_L$, and the follower's cost is $C_F = 40q_F + 0.5q_F^2$. The game sequence is: (1) Leader fixes q_L; (2) Demand A is realized; (3) Follower observes q_L and A, then optimizes q_F. Determine the optimal q_L^* to maximize the leader's expected profit.

Optimization Result

- Leader Output q_L*: 50
- Follower Output Low Demand q_F(Low): 0
- Follower Output High Demand q_F(High): 20
- Expected Price E[P]: 60
- Leader Expected Profit E[Π_L]: 2000

Category
Game Optimization(GO)

Difficulty
Medium

Figure 10: A Medium Example in OptiVerse.

The hard-level problem (Figure 11) presents a complex Mathematical Programming scenario involving multi-period production planning. The core challenge lies not just in the scale of variables (inventory, workforce, production across 4 months), but in the translation of logical business constraints into linear mathematical forms. Specifically, it requires modeling "hiring" and "firing" costs by splitting an unrestricted variable into two non-negative variables ($S_t = S_t^+ - S_t^-$), a modeling trick that LLMs often struggle to identify. The solution demands rigorous constraint handling to balance flow conservation across time periods while minimizing a composite objective function.

These examples demonstrate the progressive complexity in OptiVerse. From executing standard algorithms in Easy tasks, to synthesizing cross-domain concepts in Medium tasks, and finally to formulating complex, coupled constraints in Hard problems, the benchmark rigorously evaluates the depth of LLMs' optimization reasoning.

F Prompt Setting

To ensure experimental reproducibility and the reliability of automated evaluation, we designed structured prompts that impose strict constraints on output format and reasoning logic. Unlike general-purpose queries, these prompts are engineered to bridge the gap between natural language reasoning and rigorous optimization solving.

Problem

A mid-sized electronics manufacturer is preparing its production plan for the upcoming quarter (March through June). The forecasted monthly demand for its key component is 520, 720, 520, and 620 units, respectively. The company currently employs 10 regular production workers. To adapt to the variable demand, management can hire or lay off workers at the start of each month. Hiring a new worker incurs a $200 onboarding cost, while a layoff involves a $400 severance package. Each regular worker has a standard output of 12 units per month. If needed, temporary workers can be brought in, but they are less experienced and have a productivity of only 10 units per month. Any production exceeding monthly demand can be stored as inventory at a cost of $50 per unit per month. The operations team needs to determine the most cost-effective hiring, layoff, and staffing strategy to satisfy all customer demand over this four-month period.

Optimization Result

- Minimum Total Cost: $19,500
- Temp Workers (March): 50 (Hired 50)
- Temp Workers (April): 50 (No Change)
- Temp Workers (May): 45 (Laid off 5)
- Temp Workers (June): 45 (No Change)
- Inventory Level (End of March): 100 units
- Inventory Level (End of May): 50 units

Category	**Difficulty**
Mathematical Programming(MP)	Hard

Figure 11: A Hard Example in OptiVerse.

F.1 Inference Prompt

The solver prompt operates under a **Code-based Chain-of-Thought** paradigm. To ensure that the generated code is executable within our sandbox environment, the prompt enforces three critical constraints:

- **Expert Persona & Tool Authorization.** The prompt establishes the model's role as an "Operations Research Expert" and explicitly permits access to domain-specific libraries (e.g., `scipy`, `pulp`, `ortools`, `cvxpy`).
- **Targeted Output Format.** To facilitate automated answer extraction, the prompt dynamically injects the required output keys (e.g., specific decision variables or objective values) and mandates that the code strictly utilizes `print()` statements to output these calculated results.
- **Model-then-Code Generation.** We enforce a structured reasoning process where the model is guided to first explicitly articulate the mathematical modeling logic (i.e., defining variables, constraints, and objectives) before proceeding to code implementation. Empirical results suggest that this pre-computation formalization significantly outperforms direct code generation.

F.2 Evaluation Prompts

We employ a two-stage prompting pipeline to ensure the robustness of the assessment, converting raw execution logs into verified verdicts.

F.2.1 Answer Extraction Prompt

Since the raw standard output from code execution often contains unstructured intermediate logs, we first utilize a **Structure Synthesis Prompt** to normalize the results. This prompt receives the raw execution output and the required schema keys from the ground truth. It acts as an extractor, parsing specific numerical values or decisions into a clean, valid JSON object, while handling missing values with standardized "N/A" markers.

F.2.2 Accuracy Verification Prompt

Subsequently, we employ an LLM-as-a-Judge framework to verify the correctness of the synthesized answer. The judge model is prompted to act as a "Strict Mathematics Teaching Assistant," evaluating the "Student's Answer" (the synthesized JSON) against the "Standard Solution" based on the following criteria:

- **Completeness & Precision.** The judge verifies that all keys present in the ground truth exist in the prediction and enforces a relative numerical error tolerance of $\leq 0.1\%$.
- **Semantic Flexibility.** The judge is instructed to intelligently handle unit variations (e.g., "0.5" vs. "50%") and assess the semantic equivalence of non-numerical strategies (e.g., game theory decisions) rather than relying on rigid string matching.
- **Reasoning-First Verification.** To ensure reliability, the judge must generate a step-by-step verification log (output as the JSON field `"reason"`) prior to delivering the final boolean verdict (`"is_correct"`).