OPTGRAPH: LLM-ENHANCED AUTOMATED OPTIMIZATION MODELING VIA GRAPHRAG

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have emerged as a powerful tool for automated optimization modeling, but existing methods remain limited in pattern reuse, error-aware refinement, and retrieval robustness across diverse tasks. To address these limitations, we propose OptGraph, the first optimization agentic workflow that integrates graph retrieval-augmented generation (GraphRAG). Specifically, OptGraph first constructs reusable experience as a typed graph, capturing the relationships among modeling patterns, problem formalization, implementation details, and error corrections. In the inference stage, OptGraph leverages graph neighborhood information to enrich retrieved knowledge, providing structured context to improve modeling, verification, and iterative refinement. Moreover, OptGraph supports adaptive knowledge updates, enabling the distillation of execution traces and verification feedback into reusable graph knowledge without undertaking LLM parameter tuning. Experiments on benchmark datasets show that our OptGraph improves average exact accuracy by 8.9% over state-of-the-art prompt-based automated optimization modeling frameworks.

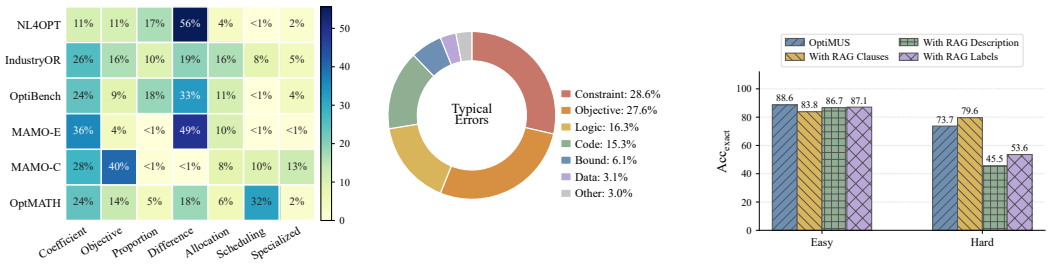
1 INTRODUCTION

Optimization problems are fundamental to modern decision-making systems, with broad applications spanning transportation, finance, and manufacturing, see the recent surveys Archetti et al. (2022); Gunjan & Bhattacharyya (2023); Weckenborg et al. (2024). In practice, the bottleneck often lies not only in solving the optimization model but also in formulating it. Practitioners must translate ambiguous natural-language descriptions into decision variables, objectives, and constraints, and subsequently implement the resulting model as executable code compatible with optimization solvers such as Gurobi and CPLEX. Clearly, this process needs extensive expertise in operations research (OR), which makes optimization modeling both time-consuming and difficult to directly apply to diverse real-world scenarios Guo et al. (2025a); Fan et al. (2026).

Recently, LLMs have shown strong potential for automated optimization modeling Vaswani et al. (2017); Ramamonjison et al. (2023); Chen et al. (2024a; 2026). Leveraging advances in mathematical reasoning, program synthesis, and algorithm design Ahn et al. (2024); Ma et al. (2026); Liu et al. (2026), LLMs can interpret natural-language problem descriptions, formulate optimization models, and generate solver-ready implementations. Existing approaches can be broadly categorized as learning-based or prompt-based Zhang et al. (2025); Xiu et al. (2026). Learning-based methods adapt LLMs using optimization-specific data, execution feedback, or reinforcement learning, whereas prompt-based methods enhance pretrained LLMs through reasoning workflows, retrieval, agent collaboration, and validation mechanisms Huang et al. (2025a); Lu et al. (2025); Jiang et al. (2025); Ding et al. (2026); Xiao et al. (2024); AhmadiTeshnizi et al. (2024b); Liu et al. (2025); Liang et al. (2026). Together, these methods reduce the human effort required to formulate and implement optimization problems.

Despite recent progress, existing LLM-driven optimization frameworks face three limitations. First, Fig. 1a reveals heterogeneous yet reusable modeling patterns across benchmark datasets, suggesting that modeling knowledge should be organized by problem types and variants rather than as isolated examples or static templates Kong et al. (2025). Second, Fig. 1b shows that failures extend beyond code execution to recurring formulation errors, including incorrect objectives, missing or inconsistent constraints, and data-interpretation errors. This motivates validation-guided, error-aware repair

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



(a) (b) (c)

Figure 1: Observations for existing frameworks: (a) diverse reusable patterns across datasets, (b) recurring formulation errors, and (c) unstable effects of naive RAG for OptiMUS.

Lian et al. (2026). Third, Fig. 1c shows that naive retrieval-augmented generation (RAG) Lewis et al. (2020) can help on hard problems but may harm performance when retrieved examples are logically irrelevant. Textual similarity alone therefore does not reliably capture modeling logic Zhu et al. (2026). These observations motivate structured knowledge reuse, validation-based correction, and robust retrieval for automated optimization modeling.

In fact, the aforementioned issues are largely attributable to hallucination prevalent in LLMs when dealing with mathematical reasoning, domain knowledge, and executable code Huang et al. (2025b). Popular methods such as Chain-of-Thought Wei et al. (2022), Tree-of-Thought Yao et al. (2023), and Graph-of-Thought Besta et al. (2024) improve reliability through structured reasoning. In addition, RAG incorporates external knowledge, but naive RAG mainly retrieves isolated text chunks based on semantic similarity Chen et al. (2024b). Graph-augmented retrieval instead organizes knowledge and its relationships in graph structures. Graph RAG (GraphRAG) Edge et al. (2024); Peng et al. (2025), together with HippoRAG Gutiérrez et al. (2024) and LightRAG Guo et al. (2025b), validates the potential of this paradigm. *This raises a natural question: can graph-augmented retrieval boost automated optimization modeling by connecting problem patterns, formulations, code snippets, typical errors, and repair hints?*

To fill this gap, we develop an adaptive GraphRAG-based agentic workflow for automated optimization, dubbed OptGraph. To the best of our knowledge, it is the first framework inspired by graph-augmented retrieval designed for optimization modeling and solving. Compared to existing prompt-based frameworks such as OptiTree Liu et al. (2025) and Lean-LLM-OPT Liang et al. (2026), it achieves superior accuracy and robustness. The main contributions are as follows:

- We characterize reusable optimization knowledge as a heterogeneous graph and retrieve structurally connected knowledge to facilitate formulation and code generation.
- We introduce a validation-guided repair mechanism that evaluates objective consistency, constraint completeness, and program correctness, and leverages error-aware graph knowledge to correct failed formulations and implementations through iterative refinement.
- We design an adaptive graph update that distills execution traces and validation feedback into reusable knowledge, enabling non-parametric accumulation across tasks.

2 Related Work

Automated Optimization Modeling LLM-based automated optimization methods can be broadly divided into learning-based and prompt-based approaches. Learning-based methods adapt LLMs using optimization-specific data or execution feedback. For example, ORLM trains LLMs on optimization instructions, while OptMATH constructs modeling data through bidirectional synthesis and filtering Huang et al. (2025a); Lu et al. (2025). LLMOPT further combines instruction tuning with self-correction, and OR-R1 incorporates execution-based reinforcement learning Jiang et al. (2025); Ding et al. (2026). In contrast, prompt-based methods improve pretrained LLMs through inference-time workflows. CoE uses forward construction and backward reflection, OptiMUS combines problem decomposition with retrieval, OptiTree organizes knowledge into hierarchical cat-

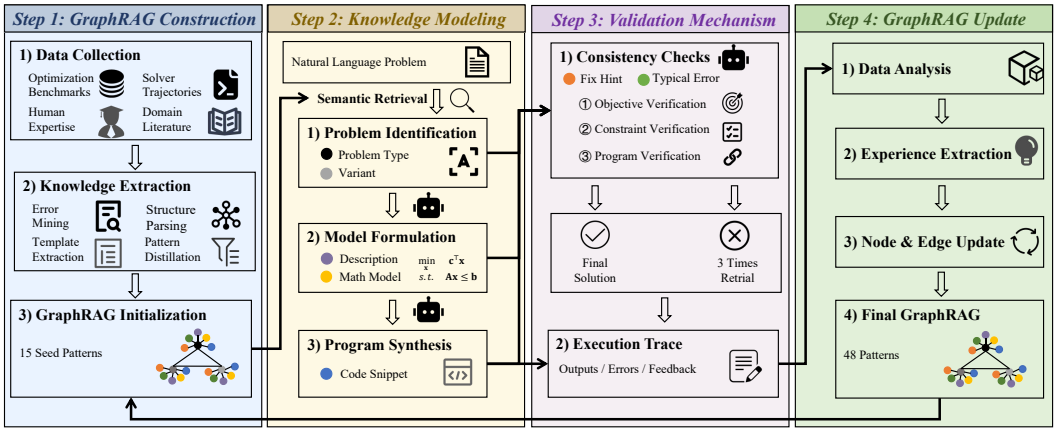


Figure 2: Overall workflow of the proposed OptGraph. Given a natural-language optimization problem, OptGraph retrieves structured knowledge from a heterogeneous modeling graph and guides LLM agents to generate a mathematical formulation and executable code. The generated code is then executed and checked by the validation mechanism. If the result is not accepted, OptGraph retrieves error-aware knowledge and performs repair.

egories, and Lean-LLM-OPT integrates prompting, retrieval, and agent collaboration Xiao et al. (2024); AhmadiTeshnizi et al. (2024a); Liu et al. (2025); Liang et al. (2026). Unlike learning-based methods, OptGraph operates in a training-free setting, improving a fixed backbone LLM through retrieval, validation, and adaptive updates of external modeling knowledge.

GraphRAG GraphRAG augments LLMs with graph-structured external knowledge, enabling retrieval that considers both semantic relevance and relationships among knowledge units. An influential early formulation constructs an entity graph from a document corpus and aggregates community-level summaries to support both local and global query-focused reasoning Edge et al. (2024). The GraphRAG literature further formalizes this paradigm as graph-based indexing, graph-guided retrieval, and graph-enhanced generation Peng et al. (2025). Representative methods include HippoRAG, which combines knowledge graphs with Personalized PageRank to support multi-hop retrieval and long-term memory Gutiérrez et al. (2024), and LightRAG, which uses dual-level retrieval and incremental updates to improve retrieval efficiency and contextual awareness Guo et al. (2025b). However, existing GraphRAG methods mainly organize entities and relations extracted from text corpora for question answering or summarization. In contrast, OptGraph represents optimization-specific knowledge, including problem types, formulations, code snippets, typical errors, and repair hints, and updates this knowledge using execution traces and validation feedback.

3 Methodology

OptGraph contains four components: GraphRAG construction, knowledge modeling, validation-guided repair, and GraphRAG update. Fig. 2 presents the overall workflow.

3.1 GraphRAG Construction

Data Collection OptGraph collects reusable optimization resources from benchmark instances, solver trajectories, human expertise, and domain literature. These resources provide problem descriptions, formulations, executable code, execution outputs, common errors, and repair experience.

Knowledge Extraction Error mining, structure parsing, template extraction, and pattern distillation transform the collected resources into reusable modeling knowledge. Specifically, they identify recurrent errors, construct typed knowledge units, summarize formulation and code patterns, and merge similar cases. The extracted knowledge is represented as a directed heterogeneous graph:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}, \tau, \rho), \tag{1}$$

An Example of the Modeling Graph

Problem Type: Vehicle routing problem (VRP)
Variant: Capacitated vehicle routing problem (CVRP) **Description:** Design vehicle delivery routes to serve all customers with minimal travel cost under vehicle capacity constraints.
Math Model:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \sum_{k\in K}\sum_{i\in V}\sum_{j\in V} c_{ij}x_{ijk} \\ \text{s.t.} \quad & \sum_{k\in K}\sum_{j\in V} x_{ijk} = 1, && \forall i \in C \\ & \sum_{i\in C} d_i \sum_{j\in V} x_{ijk} \le Q, && \forall k \in K \\ & x_{ijk} \in \{0,1\}, && \forall i,j \in V,\ k \in K \end{aligned}$$

Code Snippet:

```
x = addVars(V, V, K, vtype=GRB.BINARY)
min sum(c[i,j] * x[i,j,k] for i in V
  for j in V for k in K)
addConstrs(each customer visited once)
addConstrs(vehicle capacity limits)
```

Typical Error: Missing customer coverage, ignoring vehicle capacity, disconnected subtours.
Fix Hint: Visit each customer once, add capacity constraints, and eliminate subtours.

Figure 3: A simple example of the modeling graph in GraphRAG initialization, where all displayed entries are treated as nodes connected by semantic and functional edges.

where $\mathcal{V}$ and $\mathcal{E}$ denote the node and edge sets, while τ and ρ map nodes and edges to their types. The node types include

$$\mathcal{T}_v = \left\{ \begin{array}{l} \texttt{Problem Type}, \texttt{Variant}, \texttt{Description}, \texttt{Math Model}, \\ \texttt{Code Snippet}, \texttt{Typical Error}, \texttt{Fix Hint} \end{array} \right\}. \tag{2}$$

GraphRAG Initialization The resulting graph initializes GraphRAG with seed modeling patterns that connect problem semantics, formulations, implementations, errors, and repair knowledge. An illustrative example is provided in Fig. 3. This structure provides structured context for subsequent modeling, validation, and graph updates.

3.2 Knowledge Modeling

In Step 2, OptGraph retrieves relevant graph knowledge to guide formulation and program synthesis. Let $\mathcal{V}_{\text{var}}$ denote the set of `Variant` nodes, each representing a reusable modeling pattern. Given a natural-language optimization problem $\mathcal{P}$, OptGraph scores each variant node by

$$S(\mathcal{P}, v), \quad v \in \mathcal{V}_{\text{var}}, \tag{3}$$

where $S(\mathcal{P}, v)$ measures semantic and modeling-pattern relevance. The top-k variant nodes are selected via

$$\mathcal{R}_k(\mathcal{P}) = \text{TopK}_{v\in\mathcal{V}_{\text{var}}} S(\mathcal{P}, v). \tag{4}$$

OptGraph then expands their graph neighborhoods:

$$\mathcal{K}(\mathcal{P}) = \bigcup_{v\in\mathcal{R}_k(\mathcal{P})} (\{v\} \cup \mathcal{N}(v)), \tag{5}$$

where $\mathcal{N}(v)$ denotes the adjacent nodes of v. Thus, $\mathcal{K}(\mathcal{P})$ jointly provides related problem types, descriptions, formulations, code snippets, typical errors, and repair hints, rather than isolated text examples.

Problem Identification Based on $\mathcal{K}(\mathcal{P})$, the problem-identification agent determines the likely `Problem Type` and `Variant`, selecting relevant modeling patterns before formulation.

Model Formulation The model-formulation agent A_{model} generates a mathematical formulation from the input problem and retrieved context:

$$\hat{\mathcal{M}} = A_{\text{model}}(\mathcal{P}, \mathcal{K}(\mathcal{P})). \tag{6}$$

The retrieved `Description` and `Math Model` nodes guide the construction of decision variables, objectives, and constraints.

Program Synthesis The program-synthesis agent A_{code} generates executable code:

$$\hat{\mathcal{C}} = A_{\text{code}}(\mathcal{P}, \hat{\mathcal{M}}, \mathcal{K}(\mathcal{P})). \tag{7}$$

Retrieved `Code Snippet`, `Typical Error`, and `Fix Hint` nodes provide implementation guidance and reduce formulation–code mismatches. The formulation and code are then passed to the validation mechanism.

3.3 Validation Mechanism

In Step 3, OptGraph executes the generated code, validates the formulation and result, and records the resulting execution trace for graph updates.

Consistency Checks The generated code $\hat{\mathcal{C}}$ is executed as

$$(o, \hat{y}, s) = \text{Exec}(\hat{\mathcal{C}}), \tag{8}$$

where o, $\hat{y}$, and s denote the raw execution output, extracted numerical answer, and execution status, respectively. OptGraph validates objective consistency, constraint completeness, and program correctness:

$$z = V(\mathcal{P}, \hat{\mathcal{M}}, \hat{\mathcal{C}}, o, \hat{y}, s), \tag{9}$$

where $z \in \{0, 1\}$ indicates whether the result is accepted. At repair round t, the corresponding decision is denoted by $z^{(t)}$. For compact notation, the current modeling and execution state is defined as

$$\eta^{(t)} = (\hat{\mathcal{M}}^{(t)}, \hat{\mathcal{C}}^{(t)}, o^{(t)}, \hat{y}^{(t)}, s^{(t)}). \tag{10}$$

Error-Aware Repair When $z^{(t)} = 0$, OptGraph retrieves relevant `Typical Error` and `Fix Hint` nodes as error-aware context, and the repair agent updates the formulation and code by

$$(\hat{\mathcal{M}}^{(t+1)}, \hat{\mathcal{C}}^{(t+1)}) = A_{\text{repair}}(\mathcal{P}, \eta^{(t)}, \mathcal{K}_{\text{err}}(\mathcal{P})). \tag{11}$$

This process terminates when the result is accepted or the maximum repair budget is reached; we set the budget to 3 rounds.

Execution Trace For each problem, OptGraph records the intermediate states, validation decisions, and feedback:

$$\xi = \{(\eta^{(t)}, z^{(t)}, f^{(t)})\}_{t=0}^{\tau}, \tag{12}$$

where $f^{(t)}$ is the validation feedback and τ is the terminal round. If $z^{(\tau)} = 1$, OptGraph returns the accepted answer; otherwise, it records a failed trace. These traces are used for experience extraction and graph updates.

3.4 GraphRAG Update

In Step 4, OptGraph distills execution traces into reusable modeling knowledge and incorporates it into the graph.

Data Analysis For each problem $\mathcal{P}_q$, $q = 1, \ldots, Q$, OptGraph analyzes the execution trace ξ_q in Eq. 12, including retrieved context, generated formulation and code, execution results, and validation feedback. Successful, failed, and repaired cases provide evidence of reusable patterns and recurrent errors.

Experience Extraction The candidate update extracted from ξ_q is defined as

$$\Delta\mathcal{G}_q = U(\xi_q) = (\Delta\mathcal{V}_q, \Delta\mathcal{E}_q), \tag{13}$$

where $\Delta\mathcal{V}_q$ and $\Delta\mathcal{E}_q$ are candidate nodes and edges. They may encode new `Variant`, `Math Model`, `Code Snippet`, `Typical Error`, and `Fix Hint` knowledge, together with their semantic or functional relations.

Node & Edge Update For each processed problem, the graph is updated by

$$\mathcal{G}^{(q)} = \mathcal{G}^{(q-1)} \oplus \Delta\mathcal{G}_q, \tag{14}$$

where $\oplus$ inserts new knowledge or merges it with semantically and functionally similar nodes. The corresponding edges are added or updated to preserve relations among problem types, formulations, implementations, errors, and repair knowledge.

Final GraphRAG After processing all problems, the updated graph $\mathcal{G}^{(Q)}$ preserves newly acquired modeling experience as external knowledge. Subsequent tasks can therefore reuse prior execution and validation feedback without modifying the underlying LLM parameters.

Overall, the procedure of OptGraph is summarized in Alg. 1 in the appendix. Different from conventional retrieval methods that return isolated examples or text snippets, OptGraph maintains a unified graph-structured modeling memory and updates it with execution traces and validation feedback. It further updates this structured knowledge using execution traces and validation feedback, enabling non-parametric knowledge accumulation across tasks without modifying the parameters of the underlying LLMs.

4 Experiments

This section evaluates the superiority of our OptGraph by comparing with the state-of-the-art methods such as CoE Xiao et al. (2024), OptiMUS AhmadiTeshnizi et al. (2024a), OptiTree Liu et al. (2025), and Lean-LLM-OPT Liang et al. (2026). Moreover, four LLMs including GPT-5.1, Gemini-3.1-Pro, DeepSeek-V3.2, and Qwen3.6-Plus are added as baselines.

4.1 Experimental Setups

Datasets In the experiments, six representative optimization datasets with different sources, scales, and modeling difficulties are chosen. These datasets cover reusable OR modeling patterns such as objective interpretation, proportion and difference constraints, allocation, scheduling, routing, flow, blending, and other specialized structures. The details of these selected datasets are described as follows: (1) NL4Opt Ramamonjison et al. (2023) contains 214 linear programming problems from the NeurIPS 2022 NL4Opt competition. (2) IndustryOR Huang et al. (2025a) has 100 real-world industrial OR problems, covering five categories and three difficulty levels.(3) OptiBench Wang et al. (2024) includes 605 problems with diverse mathematical structures, such as linear, integer, nonlinear, and tabular optimization. (4) MAMO Easy Huang et al. (2025c) (MAMO-E) collects 642 relatively simple linear programming problems with short descriptions and basic constraint structures. (5) MAMO Complex Huang et al. (2025c) (MAMO-C) collects 203 complex problems with longer descriptions and richer constraints. (6) OptMATH Lu et al. (2025) contains 166 challenging modeling problems from mathematical programming instances.

Evaluation Metrics In this paper, two popular metrics are considered, i.e., exact accuracy and 5% tolerance accuracy. Specifically, exact accuracy follows the definition used in CoE Xiao et al. (2024), where a prediction is counted as correct only when the extracted answer exactly matches the ground-truth answer. For each problem, the generated code is executed, and the extracted answer $\hat{y}$ is compared with the ground-truth answer y. It is defined as

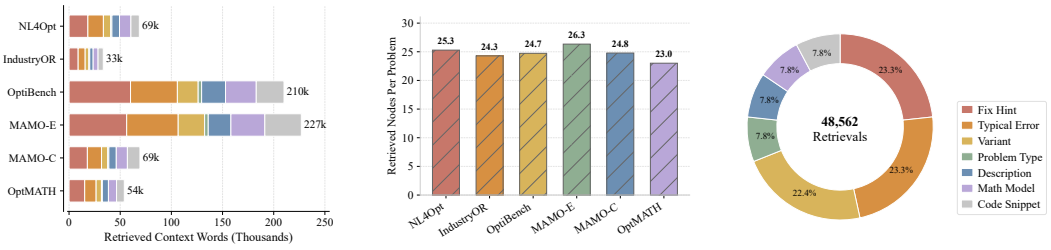
$$\mathrm{Acc}_{\mathrm{exact}} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(\hat{y}_i = y_i), \tag{15}$$

where N is the number of problems Besides, 5% tolerance accuracy is adopted from OptiTree Liu et al. (2025), which is defined as

$$\mathrm{Acc}_{5\%} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{I}\left(\frac{|\hat{y}_i - y_i|}{|y_i| + \eta} \le 0.05\right), \tag{16}$$

Table 1: Accuracy comparisons (exact accuracy / 5%-tolerance accuracy) of representative methods on selected datasets. The best and second-best results are marked in red and blue, respectively.

Methods	NL4Opt	IndustryOR	OptiBench	MAMO-E	MAMO-C	OptMATH	Average
GPT-5.1 (2025)	72.9%/78.0%	43.0%/51.0%	58.0%/65.8%	81.6%/85.8%	47.8%/58.1%	27.7%/30.7%	55.2%/61.6%
Gemini-3.1-Pro (2026)	73.4%/78.5%	42.0%/50.0%	58.2%/66.0%	82.1%/86.1%	46.8%/57.1%	27.1%/30.1%	54.9%/61.3%
DeepSeek-V3.2 (2025)	72.4%/77.6%	44.0%/50.0%	57.4%/65.1%	81.0%/85.0%	48.3%/57.6%	26.5%/30.1%	54.9%/60.9%
Qwen3.6-Plus (2026)	69.6%/74.8%	42.0%/49.0%	55.4%/62.6%	78.0%/81.8%	46.3%/55.7%	27.1%/28.9%	53.1%/58.8%
CoE (2024)	77.1%/83.6%	51.0%/61.0%	65.5%/73.2%	93.5%/93.8%	67.0%/70.3%	40.4%/43.4%	65.8%/70.9%
OptiMUS (2025)	86.9%/93.9%	54.0%/60.0%	63.0%/72.1%	87.8%/89.0%	52.2%/70.4%	38.5%/41.3%	63.7%/71.1%
OptiTree (2025)	84.1%/92.5%	64.0%/71.0%	62.5%/71.9%	90.7%/91.2%	65.4%/72.8%	31.3%/33.7%	66.3%/72.2%
Lean-LLM-OPT (2026)	86.5%/94.9%	55.0%/63.0%	61.8%/70.3%	93.8%/94.2%	57.6%/70.0%	44.5%/45.2%	66.5%/72.9%
OptGraph (Ours)	86.9%/95.3%	71.0%/81.0%	66.0%/78.2%	93.9%/94.4%	76.4%/89.2%	58.4%/60.8%	75.4%/83.2%



(a) (b) (c)

Figure 4: GraphRAG retrieval analysis across six datasets: (a) retrieved context volume, (b) retrieved nodes per problem, and (c) node-type distribution.

where η avoids division by zero. Obviously, higher precision accuracy and 5% tolerance accuracy indicate better performance of automated optimization.

Implementation Details For compared methods, their released implementations are used directly, with only dataset interfaces, execution scripts, and answer extraction modules adapted when necessary. For OptGraph, the graph-structured knowledge base is initialized from reusable optimization resources and then used throughout the inference workflow. During inference, connected graph knowledge is retrieved to guide modeling, validation, repair, and graph updating.

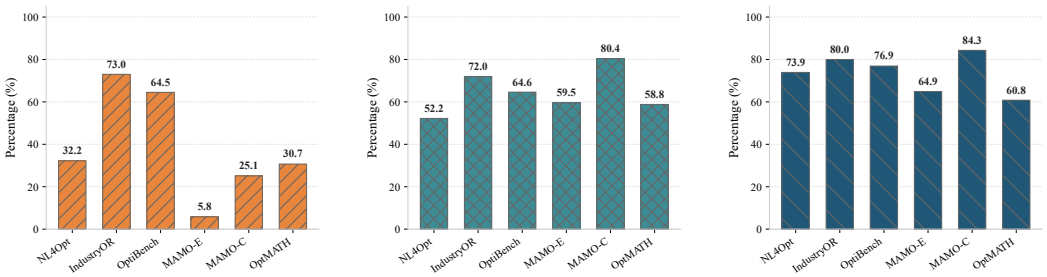
We would like to point out that GPT-5.1 is utilized for Step 1, Step 2, and Step 4, covering GraphRAG construction, knowledge modeling, and GraphRAG update. Moreover, Gemini-3.1-Pro is employed for Step 3, which serves as the verification mechanism. This setting leverages a powerful model for graph construction, formulation generation, code generation, and knowledge updating, while adopting an independent model for validation to reduce same-model self-verification bias.

4.2 Main Results

Tab. 1 reports the overall accuracy comparison on six datasets, with the best and second-best results highlighted in red and blue, respectively. In addition, the average results for each method are provided. It is evident that OptGraph achieves the highest exact accuracy and 5% tolerance accuracy on all datasets, outperforming CoE, OptiMUS, OptiTree, and Lean-LLM-OPT. Notably, OptGraph demonstrates a more pronounced advantage on complex datasets such as IndustryOR, MAMO-C, and OptMATH, where problems involve richer domain semantics, more complex constraints, and more challenging mathematical structures. For example, on OptMATH, the exact accuracy is improved by 13.9% and 5% tolerance accuracy by 15.6% compared to the second-best method. This indicates that OptGraph can effectively handle complex optimization tasks through the introduction of GraphRAG and verification mechanisms. It is also found that, while standalone LLMs like GPT-5.1 and Gemini-3.1-Pro exhibit competitive optimization ability, but their performance remains clearly lower than agentic and graph-guided methods on average. This suggests that general-purpose LLMs alone are insufficient without structured modeling knowledge and validation support.

Table 2: Ablation study of OptGraph on selected datasets. Values are reported as exact accuracy / 5% tolerance accuracy, and the best results are highlighted in bold.

GraphRAG	Validation	NL4Opt	IndustryOR	OptiBench	MAMO-E	MAMO-C	OptMATH	Avg.
×	×	67.8/92.5	47.0/52.0	60.8/67.6	85.8/89.1	52.2/63.5	36.1/41.0	58.3/67.6
×	✓	83.6/93.9	52.0/57.0	63.0/69.3	91.7/92.8	54.2/67.9	42.2/44.6	64.5/70.9
✓	×	84.6/94.4	66.0/74.0	64.5/74.5	91.9/93.0	68.0/79.8	51.8/54.8	71.1/78.4
✓	✓	**86.9/95.3**	**71.0/81.0**	**66.0/78.2**	**93.9/94.4**	**76.4/89.2**	**58.4/60.8**	**75.4/83.2**



(a) (b) (c)

Figure 5: Effects of the validation mechanism across six datasets: (a) validation trigger rate, (b) exact repair success rate, and (c) 5% repair success rate.

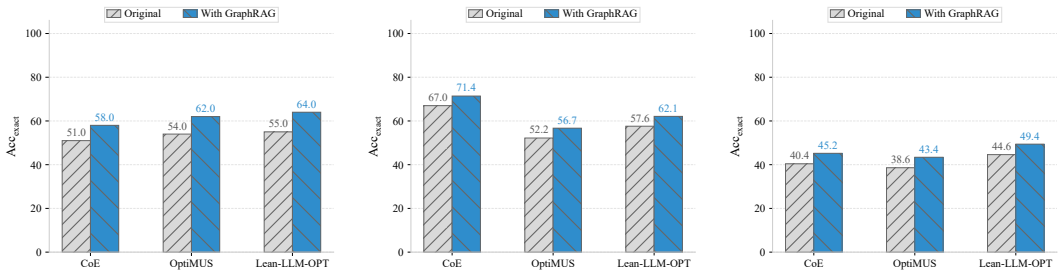
Furthermore, Fig. 4 illustrates the retrieval behavior behind these improvements. As demonstrated in Fig. 4a, the retrieved context volume varies across datasets and node types. Larger and more diverse datasets, such as OptiBench and MAMO-E, require more retrieved context, while IndustryOR and OptMATH use relatively compact but still structured retrieval. Fig. 4b shows that the average number of retrieved nodes per problem remains stable across datasets, suggesting that OptGraph does not simply increase retrieval size for harder datasets, but retrieves a compact graph neighborhood for each problem. Fig. 4c reveals that the retrieved knowledge is not dominated by a single node type. Instead, OptGraph retrieves a balanced mixture of semantic, formulation, implementation, and repair-related knowledge.

Overall, OptGraph achieves consistently strong performance across the six datasets, with particularly clear gains on complex problems. These results indicate that GraphRAG provides compact and structurally relevant modeling knowledge, while validation-guided repair further improves the reliability of formulation and code generation.

4.3 Ablation Studies

Tab. 2 conducts the ablation experiments of OptGraph. It can be concluded that the full model achieves the best performance, confirming that graph-guided retrieval and validation-based repair work together to improve reliability. GraphRAG provides structured optimization knowledge for formulation and program generation, while the validation mechanism checks generated results and uses error-aware graph context to repair modeling, execution, and answer-extraction errors. Validation further checks the generated results and uses error-aware graph knowledge to repair formulation errors, execution failures, and answer-extraction mistakes. Removing either component leads to clear performance degradation, while removing both produces the weakest results across all datasets. Notably, on MAMO-C and OptMATH, removing both components results in a performance decline of over 20% in exact accuracy and 5% tolerance accuracy.

In addition, Fig. 5 examines the effect of the validation mechanism. Fig. 5a displays the proportion of cases that trigger the validation mechanism. Obviously, IndustryOR and OptiBench have relatively high trigger rates, indicating that real-world and structurally diverse problems require more frequent checking and correction. In contrast, MAMO-E has a much lower trigger rate, which is



(a) (b) (c)

Figure 6: Plug-and-play analysis of GraphRAG on other LLM-based optimization frameworks across (a) IndustryOR, (b) MAMO-C, and (c) OptMATH.

consistent with its relatively simple and regular problem structure. Moreover, Fig. 5b and 5c report the post-trigger repair success rates under exact and 5% tolerance evaluation. The repair module achieves strong recovery on most datasets, especially under the 5% tolerance criterion. These results confirm that validation is not merely an execution filter, but an active repair mechanism that works together with GraphRAG to correct formulation-level and code-level errors.

4.4 Plug-and-play Analysis

This part evaluates whether GraphRAG can serve as a plug-and-play module for other automated optimization frameworks. As shown in Fig. 6, the same graph-augmented retrieval mechanism is integrated into CoE, OptiMUS, and Lean-LLM-OPT. It can be found that GraphRAG consistently improves all three frameworks on these representative datasets. For example, on IndustryOR, the exact accuracy of CoE, OptiMUS, and Lean-LLM-OPT increases from 51.0%, 54.0%, and 55.0% to 58.0%, 62.0%, and 64.0%, respectively. Similar improvements are also observed on MAMO-C and OptMATH. These results convince us to believe that graph-structured modeling knowledge can be incorporated into different prompt-based or agentic optimization modeling systems. Further experimental results are provided in App. B.

5 Conclusion

This paper proposed a novel graph-augmented agentic workflow for automated optimization modeling. Instead of relying on fixed prompting strategies or isolated retrieved examples, OptGraph leverages reusable optimization knowledge as a structured graph that connects `Problem Type`, `Variant`, `Description`, `Math Model`, `Code Snippet`, `Typical Error`, and `Fix Hint`. Moreover, the graph is adaptively updated based on execution trajectories and validation feedback. Numerical studies show that OptGraph achieves competitive or even superior accuracy. These findings suggest that structured and reusable modeling knowledge is a promising direction for building more robust LLM-based automated optimization modeling systems.

In the future, more efficient graph-augmented frameworks can be developed to reduce token consumption and shorten inference time while preserving modeling accuracy Xiao et al. (2025). Furthermore, in industrial-scale scenarios involving vast numbers of variables and multi-stage stochastic structures, existing LLM-based methods fall short of expectations regarding accuracy and stability Da Ros et al. (2026). Finally, the generated graph-structured trajectories facilitate the analysis of how retrieved knowledge influences problem formulation decisions and can be further integrated with symbolic verification techniques Wang et al. (2025).

AI USE STATEMENT

In this work, we used generative AI tools to implement and evaluate the proposed method. Specifically, GPT-5.1 was used within OptGraph for graph construction, optimization model formulation, code generation, graph knowledge updating, and iterative repair. Gemini-3.1-Pro was used as an independent verifier to assess generated formulations and implementations. We also used generative AI tools to assist with language polishing and translation during manuscript preparation.

We have not used generative AI tools to generate synthetic datasets or to formulate or prove mathematical claims. All AI-assisted text, experimental outputs, and scientific claims were reviewed and verified by the authors. We take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

The experimental setting is described in Sec. 4, and additional experimental results are provided in App. B. An anonymized code package will be uploaded as supplementary material to facilitate reproducibility. We promise to publicly release the source code upon acceptance.

REFERENCES

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. OptiMUS-0.3: Using large language models to model and solve optimization problems at scale. *arXiv preprint arXiv:2407.19633*, 2024a.

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *International Conference on Machine Learning*, 2024b.

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. Large language models for mathematical reasoning: Progresses and challenges. In *Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pp. 225–237, 2024.

Claudia Archetti, Lorenzo Peirano, and M Grazia Speranza. Optimization in multimodal freight transportation problems: A survey. *European Journal of Operational Research*, 299(1):1–20, 2022.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of Thoughts: Solving elaborate problems with large language models. In *the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Hao Chen, Gonzalo E Constante-Flores, and Can Li. Diagnosing infeasible optimization problems using large language models. *INFOR: Information Systems and Operational Research*, 62(4): 573–587, 2024a.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. Benchmarking large language models in retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17754–17762, 2024b.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. *Advances in Neural Information Processing Systems*, 38:106027–106069, 2026.

Francesca Da Ros, Michael Soprano, Luca Di Gaspero, and Kevin Roitero. Large language models for combinatorial optimization: A systematic review. *ACM Computing Surveys*, 58(11):1–53, 2026.

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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 228–236, 2026.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

Zhenan Fan, Bissan Ghaddar, Xinglu Wang, Linzi Xing, Yong Zhang, and Zirui Zhou. Artificial intelligence for optimization: Unleashing the potential of parameter generation, model formulation, and solution methods. *European Journal of Operational Research*, 332(1):1–30, 2026.

Abhishek Gunjan and Siddhartha Bhattacharyya. A brief review of portfolio optimization techniques. *Artificial Intelligence Review*, 56(5):3847–3886, 2023.

Tiande Guo, Anqi Li, and Congying Han. Machine learning method for combinatorial optimization problems. *SCIENTIA SINICA Mathematica*, 55(2):451, 2025a.

Zirui Guo, Lianghao Xia, Yanhua Yu, Tian Ao, and Chao Huang. LightRAG: Simple and fast retrieval-augmented generation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, 2025b.

Bernal J Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024.

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

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ACM Transactions on Information Systems*, 43(2):1–55, 2025b.

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

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

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, et al. AlphaOPT: Formulating optimization programs with self-improving LLM experience library. In *NeurIPS Workshop on GPU-Accelerated and Scalable Optimization*, 2025.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020.

Junbo Jacob Lian, Yujun Sun, Huiling Chen, Chaoyu Zhang, Hanzhang Qin, and Chung-Piaw Teo. ReLoop: Structured modeling and behavioral verification for reliable LLM-based optimization. *arXiv preprint arXiv:2602.15983*, 2026.

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

Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. A systematic survey on large language models for algorithm design. *ACM Computing Surveys*, 58(8):1–32, 2026.

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

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

Zeyuan Ma, Yue-Jiao Gong, Hongshu Guo, Jiacheng Chen, Yining Ma, Zhiguang Cao, and Jun Zhang. LLaMoCo: Instruction tuning of large language models for optimization code generation. *IEEE Transactions on Evolutionary Computation*, pp. 1–1, 2026. doi: 10.1109/TEVC.2026.3656374.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 44(2):1–52, 2025.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Advances in Neural Information Processing Systems*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Zhuohan Wang, Ziwei Zhu, Yizhou Han, Yufeng Lin, Zhihang Lin, Ruoyu Sun, and Tian Ding. OptiBench: Benchmarking large language models in optimization modeling with equivalence-detection evaluation. 2024.

Ziyu Wang, Bowen Yang, Shihao Zhou, Chenyi Li, Yuan Zhang, Bin Dong, and Zaiwen Wen. Translating informal proofs into formal proofs using a chain of states. *arXiv preprint arXiv:2512.10317*, 2025.

Christian Weckenborg, Patrick Schumacher, Christian Thies, and Thomas S Spengler. Flexibility in manufacturing system design: A review of recent approaches from operations research. *European Journal of Operational Research*, 315(2):413–441, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-Thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

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

Ziyang Xiao, Jingrong Xie, Lilin Xu, Shisi Guan, Jingyan Zhu, Xiongwei Han, Xiaojin Fu, WingYin Yu, Han Wu, Wei Shi, et al. A survey of optimization modeling meets LLMs: Progress and future directions. In *International Joint Conference on Artificial Intelligence*, pp. 10742–10750, 2025.

Xianchao Xiu, Jianhao Li, Jun Fan, and Wanquan Liu. Large language models for operations research: A comprehensive survey. *arXiv preprint arXiv:2605.20849*, 2026.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, 2023.

Yisong Zhang, Ran Cheng, Guoxing Yi, and Kay Chen Tan. A systematic survey on large language models for evolutionary optimization: From modeling to solving. *arXiv preprint arXiv:2509.08269*, 2025.

Qingling Zhu, Jiahao Fu, Junchuang Cai, Ka-Chun Wong, Qiuzhen Lin, Jianqiang Li, and Kay Chen Tan. LLM-Enhanced accumulated learning prediction for evolutionary dynamic multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 2026.

Algorithm 1: Workflow of OptGraph

Input: Optimization problem set $\{\mathcal{P}_q\}_{q=1}^{Q}$, initial graph $\mathcal{G}^{(0)}$, maximum repair rounds T, retrieval size k
Output: Final answers $\{\hat{y}_q\}_{q=1}^{Q}$, execution traces $\{\xi_q\}_{q=1}^{Q}$, updated graph $\mathcal{G}^{(Q)}$

```
// Step 1: GraphRAG Construction
Initialize GraphRAG with the heterogeneous graph G^(0) defined in Eq. 1
for q ← 1 to Q do
    // Step 2: Knowledge Modeling
    Retrieve the top-k variant nodes from G^(q−1) according to Eq. 4
    Expand their graph neighborhoods and obtain the retrieved context K(P_q) according to Eq. 5
    Identify the problem type and relevant modeling variants from K(P_q)
    Generate the initial formulation M̂_q^(0) and executable code Ĉ_q^(0) according to Eqs. 6 and 7
    // Step 3: Validation Mechanism
    Set r ← 0, ŷ_q ← ∅, and ξ_q ← ∅
    while r ≤ T do
        Execute Ĉ_q^(r) and obtain (o_q^(r), ŷ_q^(r), s_q^(r)) according to Eq. 8
        Define the current modeling and execution state η_q^(r) according to Eq. 10
        Validate the formulation, code, and answer to obtain z_q^(r) and validation feedback f_q^(r) according
          to Eq. 9
        Append (η_q^(r), z_q^(r), f_q^(r)) to ξ_q according to Eq. 12
        if z_q^(r) = 1 then
            Set ŷ_q ← ŷ_q^(r)
            break
        if r = T then
            break
        Retrieve error-aware context K_err(P_q) from G^(q−1)
        Repair the formulation and code according to Eq. 11
        Set r ← r + 1
    if ŷ_q = ∅ then
        Mark ξ_q as a failed execution trace
    // Step 4: GraphRAG Update
    Analyze the execution trace ξ_q
    Extract the candidate graph update ΔG_q according to Eq. 13
    Update the graph as G^(q) ← G^(q−1) ⊕ ΔG_q according to Eq. 14
return {ŷ_q}_{q=1}^{Q}, {ξ_q}_{q=1}^{Q}, and G^(Q)
```

A ALGORITHM

This section provides the complete pseudocode of OptGraph. Given an initial modeling graph and a set of optimization problems, OptGraph retrieves graph-structured knowledge, generates and validates formulations and code, performs error-aware repair when needed, and updates the graph using execution traces and validation feedback. The complete procedure is summarized in Alg. 1.

B ADDITIONAL EXPERIMENTAL ANALYSIS

This section discusses the limitations of naive RAG, effects of validation rounds, and computational cost on three complex datasets, i.e., IndustryOR, MAMO-C, and OptMATH.

Limitations of Naive RAG Fig. 7a compares the original workflows with naive RAG and GraphRAG on the same datasets. The results show that simply adding retrieved examples does not necessarily improve optimization performance. On all three datasets, naive RAG performs worse than the original workflows, decreasing exact accuracy from 52.0% to 50.0% on IndustryOR, from 54.2% to 52.2% on MAMO-C, and from 42.2% to 40.4% on OptMATH. In contrast, our OptGraph with GraphRAG achieves 71.0%, 76.4%, and 58.4% on these datasets, respectively. This suggests

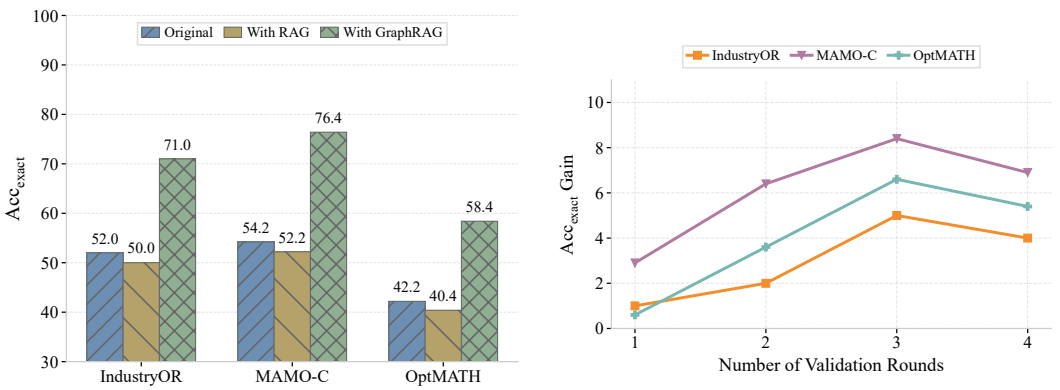


Figure 7: Analysis of retrieval and validation mechanisms: (a) effects of naive RAG and GraphRAG on the original workflows across IndustryOR, MAMO-C, and OptMATH; (b) accuracy gains over the zero-round setting with different numbers of validation rounds.

Table 3: Efficiency and average accuracy comparisons, where token usage is estimated from visible logs, and the best
results are marked in bold.

Method	Tokens (k)	Time (s)	$\text{Acc}_{\text{exact}}$	$\text{Acc}_{5\%}$
CoE	**1.3**	79.1	65.8	70.9
OptiMUS	15.9	111.1	63.7	71.1
OptiTree	4.6	**14.3**	66.3	72.2
Lean-LLM-OPT	1.5	17.1	66.5	72.9
OptGraph	9.7	93.9	**75.4**	**83.2**

that semantic similarity alone may retrieve examples that are textually similar but not aligned with the underlying logic. Therefore, retrieval for automated optimization should not only return similar text snippets, but also preserve structured relations.

Effects of Validation Rounds Fig. 7b analyzes the effect of the maximum number of validation rounds. The reported values denote the exact-accuracy gain over the zero-round setting. Increasing the validation budget generally improves performance on all three datasets, showing that iterative validation and repair are useful for complex optimization modeling tasks. The gains peak at 3 validation rounds, where IndustryOR, MAMO-C, and OptMATH obtain improvements of 5.0%, 8.4%, and 6.6%, respectively. When the budget is increased to 4 rounds, the gains decrease on all three datasets, indicating that excessive repair may introduce unnecessary changes or over-correction. Therefore, the maximum number of validation rounds is set to 3 by default, which provides a good balance between accuracy improvement and computational cost.

Computational Cost Tab. 3 lists the computational costs and average accuracy of the different frameworks. It can be seen that our proposed OptGraph consumes more tokens than lightweight methods such as CoE and Lean-LLM-OPT because it retrieves graph-structured modeling knowledge and performs validation-based repair. Meanwhile, the average runtime of OptGraph is also higher than that of OptiTree and Lean-LLM-OPT. However, among all the compared methods, OptGraph achieves the highest average exact accuracy and 5% tolerance accuracy. It is noted that compared with OptiMUS, our OptGraph requires fewer average seconds per problem while achieving substantially higher accuracy. These results demonstrate that OptGraph trades off additional retrieval and verification costs for greater formulation robustness and solution reliability, making it more suitable for accuracy-oriented automated optimization.